EDB Postgres AI Hybrid Manager

Release 1.4.0

EnterpriseDB Corporation

Jul 02, 2026

Contents

1 What you can do **3**
1.1 Start here . . . 3

2 Hybrid Manager Innovation Release Notes **5**
2.1 Hybrid Manager dual release strategy . . . 5
2.1.1 Innovation Release release notes . . . 6
2.1.1.1 Hybrid Manager 1.4.0 release notes . . . 6
2.1.1.1.1 Release details . . . 7
2.1.1.2 Contributing Innovation Releases . . . 17
2.1.1.2.1 Hybrid Manager 2025.11.0 release notes . . . 17
2.1.1.2.2 Hybrid Manager 2025.12.0 release notes . . . 19
2.1.1.2.3 Hybrid Manager 2026.1.0 release notes . . . 22
2.1.1.2.4 Hybrid Manager 2026.2.0 release notes . . . 23
2.1.1.2.5 Hybrid Manager 2026.3.0 release notes . . . 25
2.1.1.2.6 Hybrid Manager 2026.4.1 release notes . . . 28
2.1.1.2.7 Hybrid Manager 2026.5.0 release notes . . . 37
2.1.1.2.8 Hybrid Manager 2026.5.1 release notes . . . 40

3 EDB Postgres AI Hybrid Manager **43**
3.1 Why Hybrid Manager? . . . 43
3.2 Key capabilities . . . 43
3.3 Supported environments . . . 44
3.4 Integrated solutions . . . 44
3.5 Learn more . . . 44
3.6 Get started . . . 45
3.6.1 Hybrid Manager architecture . . . 45
3.6.1.1 Scalability . . . 45
3.6.1.2 Backup architecture . . . 46
3.6.1.3 High availability and resilience . . . 46
3.6.2 Supported database types . . . 46
3.6.2.1 Distributed high availability . . . 46
3.6.2.2 Advanced high availability . . . 46
3.6.2.3 High availability . . . 47
3.6.2.4 Single node . . . 47
3.6.2.5 Cluster type architectures . . . 47
3.6.2.5.1 Distributed high availability . . . 47
3.6.2.5.2 Advanced high availability . . . 50

3.6.2.5.3 High availability . . . 51
3.6.2.5.4 Single-node cluster . . . 52
3.6.3 Known issues . . . 52
3.6.3.1 Multi-DC . . . 52
3.6.3.1.1 Multi-DC configuration loss after upgrade . . . 52
3.6.3.1.2 Location dropdown list is empty for multi-DC setups . . . 53
3.6.3.1.3 Dedicated object storage for project isolation . . . 54
3.6.3.2 Core platform and resources . . . 54
3.6.3.2.1 Cross-stream upgrade issues for OpenShift users . . . 54
3.6.3.2.2 `upm-beacon-agent` memory limits are insufficient in complex environments . . . 54
3.6.3.2.3 IdP configuration fails after upgrade to version 2026.2 . . . 54
3.6.3.2.4 `accm-server` fails to start due to missing Langflow secret . . . 55
3.6.3.2.5 No `kapp-controller` support in Hybrid Manager app to pull images . . . 55
3.6.3.2.6 Beacon server enters a crashloop when spire is broken . . . 55
3.6.3.2.7 SPIRE agent fails to start on RKE2 with SELinux enabled . . . 55
3.6.3.2.8 Image library discovery fails when duplicate operand images share the same tags and SHA . . . 56
3.6.3.3 Database cluster engine . . . 56
3.6.3.3.1 Failure to create 3-node PGD cluster when `max_connections` is non-default . . . 56
3.6.3.3.2 AHA witness node resources are over-provisioned . . . 56
3.6.3.3.3 HA clusters use `verify-ca` instead of `verify-full` for streaming replication certificate authentication . . . 56
3.6.3.3.4 Second node is too slow to join large HA clusters . . . 56
3.6.3.3.5 EDB Postgres Distributed (PGD) cluster with 2 data groups and 1 witness group not healthy . . . 56
3.6.3.3.6 AKS deployments limited to one PGD data group with private access per region . . . 57
3.6.3.3.7 Network access type defaults to `Private` when not explicitly selected on RKE . . . 57
3.6.3.3.8 CNPG operator enters an infinite reconcile loop when bootstrap `Job` creation is interrupted . . . 57
3.6.3.4 Backup and recovery . . . 57
3.6.3.4.1 Replica cluster creation fails when using volume snapshot recovery across regions . . . 57
3.6.3.4.2 Volume snapshot restoration is limited to the same region . . . 58
3.6.3.4.3 WAL archiving is slow due to default parallel configuration . . . 58
3.6.3.4.4 `transporter-db` disaster recovery (DR) process may fail due to WAL gaps . . . 58
3.6.3.4.5 Full cluster recovery fails with `pg_ctl: server does not shut down` error . . . 58
3.6.3.5 Agent Factory and model management . . . 58
3.6.3.5.1 Failure to deploy `nim-nvidia-nvclip` model with profile cache . . . 58
3.6.3.5.2 AI Model cluster deployment stalls if object storage path for model profiles is empty . . . 60
3.6.3.5.3 Model configuration settings reset after Innovation Release upgrade . . . 60
3.6.3.5.4 Error page appears when editing knowledge base credentials . . . 60
3.6.3.5.5 Missing aidb_users role on PGD witness node prevents AI functionality . . . 60
3.6.3.5.6 GenAI Builder structures and tools execution failure . . . 61
3.6.3.5.7 Pipeline Designer-created knowledge bases not visible to GenAI Builder . . . 61
3.6.3.5.8 Editing external inference service parameters returns a name change error . . . 62
3.6.3.5.9 Missing validation message for invalid external model names . . . 62
3.6.3.5.10 Pipeline creation fails on PGD clusters with AIDB v6.1 . . . 62

3.6.3.5.11 External cloud-hosted OCR models fail to register via the HM console . . 63
3.6.3.6 Analytics and tiered tables . . . 63
3.6.3.6.1 Updating tiered, partitioned tables fails with PGAA error . . . 63
3.6.3.6.2 PGAA causes a server crash (SIGSEGV) when executing JOIN queries with constant result relations in PostgreSQL 17 . . . 63
3.6.3.7 HM console and observability . . . 64
3.6.3.7.1 False positive backup alerts for Advanced High Availability and Distributed High Availability (PGD) clusters . . . 64
3.6.3.7.2 Event markers overlapping when displaying two different events . . . 64
3.6.3.7.3 HTTP 431 “Request Header Fields Too Large” error when accessing the Estate page . . . 64
3.6.3.7.4 Missing notifications for task and cluster lifecycle events . . . 64
3.6.3.7.5 WAL storage percent metric displays incorrect values on the monitoring page . . . 65
3.6.3.7.6 GUC settings aren’t applied until after a restart . . . 65
3.6.3.7.7 Alert threshold updates intermittently fail to take effect . . . 65
3.6.3.7.8 Asset Library apps require Project Owner or Project Editor role . . . 65
3.6.3.7.9 Asset Library apps sorted by package name instead of display name . . . 65
3.6.3.7.10 Metabase fails to deploy when using an EPAS cluster as its database backend 66
3.6.3.7.11 Asset Library apps pgAdmin4 and Apache Superset fail to deploy on RHOS 66
3.6.3.7.12 Editing cryptographic keys for Airflow or Superset causes the app to enter a failed state . . . 66
3.6.3.7.13 Superset deployment fails with `DuplicateTable` error . . . 66
3.6.3.7.14 Asset Library launch app always uses port 443 regardless of custom portal port . . . 66
3.6.3.7.15 Asset Library app CPU and memory metrics include failed and completed pod states . . . 67
3.6.3.7.16 Deleting an app or flow gets stuck due to missing service account . . . 67
3.6.3.7.17 Asset Library apps may get stuck pending on RHOS when storage classes differ . . . 67
3.6.3.8 Clusters and upgrades . . . 68
3.6.3.8.1 Upgrade to version 2026.4 fails due to immutable `fluent-bit-k8s-events` StatefulSet . . . 68
3.6.3.8.2 Major version upgrade fails for Advanced High Availability and Distributed High Availability (PGD) clusters with witness groups . . . 68
3.6.3.8.3 Upgrade to version `2026.3` fails due to logging namespace termination . 68
3.6.3.9 Agent Factory . . . 68
3.6.3.9.1 Knowledge base data isn’t replicated to non-primary nodes on PGD clusters 68
3.6.3.9.2 Index analysis actions fail due to missing `hypopg` extension . . . 69
3.6.3.9.3 Deleted clusters still appear in AI components . . . 69
3.6.3.9.4 Missing validation for required MCP tool parameters . . . 69
3.6.3.9.5 Some EDB Component tools are non-functional due to incompatible API request formatting . . . 69
3.6.3.9.6 Advanced high Availability and Distributed High Availability (PGD) witness groups appear in database selection lists . . . 70
3.6.3.9.7 Text search may match on non-visible fields . . . 70
3.6.3.9.8 Langflow MCP server components not supported in HM Flow Hosting . . 70
3.6.3.9.9 EDB Knowledge Base component returns a 403 error when dragged into Langflow . . . 70
3.6.3.9.10 EDB Knowledge Base component fails to connect when using a global variable for the database user . . . 70
3.6.3.9.11 Running a test query against an empty knowledge base returns a 500 error 71
3.6.3.9.12 Flow Hosting deployment fails intermittently with `FlowValidationError` 71

3.6.3.9.13 EDB Model Server component causes agent failures when used with the EDB Airman MCP component . . . 71
3.6.3.9.14 `upm-ai-model-server-beaco` OOMKills due to insufficient memory limits . . . 71
3.6.3.9.15 Updated Langflow flow changes aren't immediately reflected in HM after republishing . . . 71
3.6.3.10 Migrations . . . 72
3.6.4 Supported Postgres distributions . . . 72
3.6.4.1 EDB Postgres Advanced Server . . . 72
3.6.4.2 EDB Postgres Extended Server . . . 72
3.6.4.3 PostgreSQL . . . 72
3.6.4.4 Distribution comparison . . . 73
3.6.4.5 Documentation by distribution . . . 73

4 Installing Hybrid Manager **75**
4.1 Installing Hybrid Manager . . . 75
4.1.1 step-by-step guide . . . 75
4.1.1.1 Overview . . . 75
4.1.1.2 Phase 1: Planning your architecture . . . 75
4.1.1.3 Phase 2: Gathering your system requirements . . . 76
4.1.1.4 Phase 3: Deploying your Kubernetes cluster . . . 76
4.1.1.5 Phase 4: Preparing your environment . . . 77
4.1.1.6 Phase 5: Installing Hybrid Manager . . . 77
4.1.1.7 Phase 6: Exploring your post-installation options . . . 77
4.1.1.8 Configuring HM . . . 78
4.1.1.8.1 Planning your architecture . . . 78
4.1.1.8.2 Gathering your system requirements . . . 92
4.1.1.8.3 Deploying your Kubernetes cluster . . . 112
4.1.1.8.4 Preparing your environment . . . 118
4.1.1.8.5 Install Hybrid Manager . . . 123
4.1.1.8.6 Exploring your post-installation options . . . 131
4.1.2 Configuring SOPs . . . 133
4.1.2.1 Syncing images to a local private registry . . . 133
4.1.2.1.1 Prerequisites . . . 133
4.1.2.1.2 Configure sync variables . . . 133
4.1.2.1.3 Update configuration . . . 134
4.1.2.1.4 Next steps . . . 135
4.1.2.2 Using the in-cluster container registry . . . 135
4.1.2.2.1 Prerequisites . . . 135
4.1.2.2.2 Security considerations . . . 135
4.1.2.2.3 Loading images into HMRegistry . . . 136
4.1.2.2.4 Configure the HybridControlPlane to use the HMRegistry . . . 137
4.1.2.2.5 Switching back to a registry external to HM . . . 137
4.1.2.3 Configuring image discovery . . . 138
4.1.2.3.1 Prerequisites . . . 138
4.1.2.3.2 imageDiscovery . . . 138
4.1.2.3.3 imagesetDiscoveryContainerRegistryURL . . . 139
4.1.2.3.4 imagesetDiscoveryAllowInsecureRegistry . . . 139
4.1.2.3.5 image_discovery_credentials_secret_name . . . 140
4.1.2.3.6 Generic / on-Premises (RKE2/OpenShift) . . . 140
4.1.2.3.7 Azure Container Registry (ACR) . . . 140
4.1.2.3.8 AWS Elastic Container Registry (ECR) . . . 140
4.1.2.3.9 Additional configuration for package installation . . . 141
4.1.2.3.10 Google Artifact Registry (GAR) . . . 142

4.1.2.3.11 GitLab Container Registry . . . 142
4.1.2.4 Customizing secrets . . . 143
4.1.2.4.1 Customizing GenAI Builder secrets . . . 143
4.1.2.4.2 Customizing Catalog secrets . . . 145
4.1.2.4.3 Customizing Migration Portal secrets for secure internal communication . . . 146
4.1.2.5 Implementing TLS certificates . . . 151
4.1.2.5.1 Set up a custom x.509 certificate for the Hybrid Manager Portal . . . 151
4.1.2.5.2 Set up a custom cert-manager issuer for the HM Portal . . . 153
4.1.2.5.3 Bring your own private certificate authority . . . 153
4.1.2.6 Configuring object storage . . . 154
4.1.2.6.1 Provider configurations . . . 155
4.1.2.6.2 Providing a CA bundle . . . 159
4.1.2.6.3 Configuring per-project object storage . . . 160
4.1.2.7 Deploying Hybrid Manager across multiple data centers . . . 160
4.1.2.7.1 Multi-DC for high availability . . . 160
4.1.2.7.2 Configuring a Hybrid Manager deployment across multiple data centers . . . 163
4.1.2.8 Enabling Key Management Systems (KMS) for TDE . . . 168
4.1.2.8.1 Adding KMS support . . . 169
4.1.2.8.2 HashiCorp Vault . . . 170
4.1.2.8.3 AWS Key Management Service (KMS) . . . 173
4.1.2.8.4 Azure Key Vault (Key Management Service) . . . 175
4.1.2.8.5 Google Cloud Key Management Service (Cloud KMS) . . . 177
4.1.2.8.6 Deleting a key . . . 178
4.1.2.9 Optimizing service mesh memory . . . 178
4.1.2.9.1 Understanding Istio memory requirements . . . 178
4.1.2.9.2 Recommended memory limits . . . 178
4.1.2.9.3 Configuring Istio memory limits . . . 179
4.1.2.9.4 Monitoring Istio memory usage . . . 179
4.1.2.10 API audit logging . . . 180
4.1.2.10.1 What gets audited . . . 180
4.1.2.10.2 Disabling API audit logging . . . 180
4.1.2.10.3 Re-enabling API audit logging . . . 181
4.1.2.10.4 Querying audit logs . . . 181
4.1.2.10.5 Log format . . . 182
4.1.2.10.6 Log fields . . . 182
4.1.3 Migrating from bootstrap to operator installation . . . 182
4.1.3.1 Overview . . . 182
4.1.3.2 Understanding the migration challenge . . . 183
4.1.3.2.1 The Helm ownership conflict . . . 183
4.1.3.2.2 Why this happens . . . 183
4.1.3.2.3 The CRD deletion problem . . . 183
4.1.3.3 Prepare the migration with edbctl . . . 183
4.1.3.3.1 Run the conversion . . . 183
4.1.3.4 Manual fallback for edbctl less than 1.7.0 . . . 185
4.1.3.4.1 Configuration mapping reference . . . 185
4.1.3.4.2 Convert using the migration script . . . 185
4.1.3.4.3 Helm ownership transfer (manual) . . . 188
4.1.3.4.4 How it works . . . 189
4.1.3.4.5 Prerequisites . . . 189
4.1.3.4.6 Step 1: Delete bootstrap deployments . . . 191
4.1.3.4.7 Step 2: Transfer Helm ownership - ServiceAccount . . . 191
4.1.3.4.8 Step 3: Transfer Helm ownership - CRDs . . . 192
4.1.3.4.9 Step 4: Transfer Helm ownership - ClusterRoles . . . 192
4.1.3.4.10 Step 5: Transfer Helm ownership - ClusterRoleBindings . . . 193

4.1.3.4.11 Step 6: Transfer Helm ownership - Roles . . . 193
4.1.3.4.12 Step 7: Transfer Helm ownership - RoleBindings . . . 194
4.1.3.4.13 Step 8: Transfer Helm ownership - Services . . . 194
4.1.3.4.14 Step 9: Transfer Helm ownership - Webhook Configurations . . . 194
4.1.3.4.15 Step 10: Verify ownership transfer . . . 195
4.1.3.5 Support . . . 195
4.1.3.6 Next steps . . . 195
4.1.4 Troubleshooting Hybrid Manager installations . . . 195
4.1.4.1 Understanding Preflight checks . . . 195
4.1.4.1.1 Prerequisites . . . 196
4.1.4.1.2 Outcomes . . . 196
4.1.4.1.3 Checking the preflight status . . . 196
4.1.4.1.4 Understanding the states . . . 197
4.1.4.1.5 Troubleshooting preflight failures . . . 197
4.1.4.2 Monitoring with Postflight checks . . . 198
4.1.4.2.1 Prerequisites . . . 198
4.1.4.2.2 Outcomes . . . 198
4.1.4.2.3 What postflight checks do . . . 198
4.1.4.2.4 Checking the postflight status . . . 199
4.1.4.2.5 Troubleshooting unhealthy checks . . . 200
4.1.4.2.6 Configuring the check interval . . . 201

5 Using Hybrid Manager 203
5.1 Where to start . . . 203
5.2 For users . . . 203
5.3 For Kubernetes administrators . . . 204
5.3.1 Hybrid Manager Terminology . . . 204
5.3.1.1 Hybrid Manager . . . 204
5.3.1.1.1 Private Database-as-a-Service (DBaaS) . . . 204
5.3.1.1.2 Engineered System . . . 205
5.3.1.2 PG AI Console . . . 205
5.3.1.2.1 Projects . . . 205
5.3.1.2.2 Estate . . . 205
5.3.1.2.3 Launchpad . . . 205
5.3.1.3 Clusters (disambiguation) . . . 205
5.3.1.3.1 Kubernetes cluster . . . 205
5.3.1.3.2 Kubernetes stretch cluster . . . 206
5.3.1.3.3 Postgres cluster . . . 206
5.3.1.3.4 Database cluster . . . 206
5.3.1.4 Database cluster types in Hybrid Manager . . . 206
5.3.1.4.1 HM-managed clusters . . . 207
5.3.1.4.2 External clusters . . . 207
5.3.1.4.3 Self-managed cluster . . . 207
5.3.1.4.4 Cloud Service Provider-managed cluster . . . 208
5.3.1.4.5 External CNPG cluster . . . 208
5.3.1.5 Other terms . . . 208
5.3.1.5.1 Data node . . . 208
5.3.1.5.2 Postgres instance . . . 208
5.3.2 Hybrid Manager console . . . 208
5.3.2.1 What is the Hybrid Manager console? . . . 208
5.3.2.2 What can you do in the HM console? . . . 209
5.3.2.3 HM console organization . . . 209
5.3.2.4 Launching third-party services . . . 209
5.3.2.4.1 Exploring the Hybrid Manager console . . . 209

5.3.2.4.2 Postgres clusters . . . 212
5.3.2.4.3 OSS Library . . . 214
5.3.3 For Users of Hybrid Manager . . . 228
5.3.3.1 Projects . . . 228
5.3.3.1.1 Creating projects . . . 228
5.3.3.1.2 Project details . . . 228
5.3.3.1.3 Managing projects . . . 229
5.3.3.1.4 Configure projected object storage . . . 229
5.3.3.2 Working with clusters . . . 236
5.3.3.2.1 What can you do? . . . 237
5.3.3.3 Upgrading database clusters in Hybrid Manager . . . 294
5.3.3.3.1 Postgres minor version upgrades in Hybrid Manager . . . 294
5.3.3.3.2 Postgres major version upgrades . . . 295
5.3.3.4 Migrating databases with Hybrid Manager . . . 300
5.3.3.4.1 How do I use migration capabilities in HM? . . . 301
5.3.3.4.2 What kind of migrations does the EDB Data Migration Service support? . . . 301
5.3.3.4.3 What kind of technologies does the EDB Data Migration Service use? . . . 301
5.3.3.5 Asset management . . . 473
5.3.3.5.1 Managing HM Postgres and model assets . . . 473
5.3.3.6 Backups and restores . . . 478
5.3.3.6.1 Backups . . . 478
5.3.3.6.2 Restores . . . 484
5.3.3.7 EDB Hybrid Manager Chat Agent . . . 486
5.3.3.7.1 Supported operations . . . 486
5.3.3.7.2 Model-dependent capabilities . . . 487
5.3.3.7.3 Documentation scope . . . 487
5.3.3.7.4 Next steps . . . 487
5.3.3.8 Governance in Hybrid Manager . . . 502
5.3.3.8.1 What HM provides . . . 502
5.3.3.8.2 Prerequisites . . . 502
5.3.3.8.3 Connecting to HM . . . 502
5.3.3.8.4 Learn more . . . 502
5.3.3.9 Hybrid Manager CLI - edbctl . . . 503
5.3.3.10 Interacting programmatically with HM . . . 503
5.3.3.10.1 Access Key . . . 504
5.3.3.11 Managing identity providers . . . 508
5.3.3.11.1 Configuring your IdP in the HM console . . . 508
5.3.3.11.2 Rotating IdP connector encryption keys . . . 513
5.3.4 For Kubernetes Administrators . . . 516
5.3.4.1 Managing users in Hybrid Manager . . . 516
5.3.4.1.1 Managing native users for testing . . . 516
5.3.4.1.2 Managing authentication policy . . . 519
5.3.4.1.3 User roles and authorization . . . 520
5.3.4.2 Monitoring with Hybrid Manager . . . 526
5.3.4.2.1 Understanding the monitoring architecture . . . 526
5.3.4.2.2 Connecting data into external tools . . . 528
5.3.4.2.3 Related topics . . . 528
5.3.4.3 Upgrading Hybrid Manager . . . 607
5.3.4.3.1 LTS upgrade paths . . . 608
5.3.4.3.2 Cross-stream upgrade paths . . . 608
5.3.4.4 Hybrid Manager backup and disaster recovery (DR) . . . 626
5.3.4.4.1 Disaster recovery overview . . . 626
5.3.4.4.2 Configuring backups to support disaster recovery (DR) . . . 628
5.3.4.4.3 Hybrid Manager disaster recovery . . . 630

5.3.4.5 Enabling the Migration Portal AI Copilot . . . 640
5.3.4.5.1 Third-party model (cloud) . . . 640
5.3.4.5.2 Self-hosted model (on-premises) . . . 640
5.3.4.6 Troubleshooting Hybrid Manager . . . 646
5.3.4.6.1 Error codes . . . 646
5.3.4.6.2 Errors in detail . . . 646
5.3.4.7 Using query recommendations in Hybrid Manager . . . 660
5.3.4.8 Using query recommendations in Hybrid Manager . . . 660
5.3.4.8.1 Who uses query recommendations? . . . 661
5.3.4.8.2 What are query recommendations? . . . 661
5.3.4.8.3 Why use query recommendations? . . . 661
5.3.4.8.4 When to use query recommendations . . . 661
5.3.4.8.5 How query recommendations work . . . 661
5.3.4.8.6 Integrating query recommendations into your tuning process . . . 662
5.3.4.8.7 Workflow: From query to optimization . . . 662
5.3.4.8.8 Requirements for external clusters . . . 662
5.3.4.8.9 Example scenarios . . . 663
5.3.4.8.10 Supported recommendation types . . . 663
5.3.4.8.11 Related information . . . 663

6 Agent Factory in Hybrid Manager 665
6.1 Example AI solutions you can build . . . 665
6.2 Agent Factory in Hybrid Manager workloads . . . 666
6.2.1 Build AI flows . . . 666
6.2.2 Model management and serving . . . 666
6.3 How Agent Factory fits within Hybrid Manager . . . 666
6.4 Sovereign AI in Hybrid Manager . . . 666
6.5 Get started . . . 667
6.5.1 Getting Started with AI Factory . . . 667
6.5.1.1 Getting started with Agent Factory on Hybrid Manager . . . 667
6.5.1.1.1 What you'll build . . . 667
6.5.1.1.2 Before you start . . . 667
6.5.1.1.3 Option A — Build an AI flow with Langflow . . . 668
6.5.1.1.4 Option B — Create a knowledge base pipeline . . . 668
6.5.1.1.5 Access model endpoints from your applications . . . 668
6.5.1.1.6 Monitor and iterate . . . 668
6.5.1.2 Prerequisites for Agent Factory on Hybrid Manager . . . 668
6.5.1.2.1 Enabling Agent Factory . . . 668
6.5.1.2.2 Infrastructure Requirements . . . 668
6.5.1.2.3 Registry Configuration . . . 669
6.5.1.2.4 Next Steps . . . 669
6.5.1.3 GPU Recommendations for Default NIM Models . . . 670
6.5.1.3.1 Overview . . . 670
6.5.1.3.2 Default NIM Models . . . 670
6.5.1.3.3 Minimum GPU Requirement . . . 670
6.5.1.3.4 Cloud Mappings . . . 670
6.5.2 Understand the AI Factory Architecture . . . 670
6.5.2.1 Agent Factory Architecture on Hybrid Manager . . . 670
6.5.2.1.1 Architectural Overview . . . 670
6.5.2.1.2 Core Components . . . 671
6.5.2.1.3 Infrastructure . . . 672
6.5.2.2 Use Cases and Personas for Agent Factory on Hybrid Manager . . . 672
6.5.2.2.1 Overview . . . 672
6.5.2.2.2 Key Use Cases . . . 672

6.5.2.2.3 User Personas . . . 674
6.5.2.2.4 Industry-Specific Implementations . . . 675
6.5.2.2.5 Implementation Considerations . . . 676
6.5.2.2.6 Getting Started by Use Case . . . 676
6.5.2.2.7 Additional Resources . . . 677
6.5.2.3 Frequently Asked Questions - Agent Factory on Hybrid Manager . . . 677
6.5.2.3.1 Table of Contents . . . 677
6.5.2.3.2 Platform Capabilities . . . 677
6.5.2.3.3 Installation and Setup . . . 678
6.5.2.3.4 Model Management . . . 679
6.5.2.3.5 Langflow . . . 679
6.5.2.3.6 Operations and Maintenance . . . 679
6.5.2.3.7 Troubleshooting . . . 680
6.5.2.3.8 Security and Compliance . . . 681
6.5.2.3.9 Performance and Scaling . . . 681
6.5.2.3.10 Additional Resources . . . 682
6.5.3 Model Management . . . 682
6.5.3.1 Model Management on Hybrid Manager . . . 682
6.5.3.1.1 Purpose and Benefits . . . 682
6.5.3.1.2 Core Concepts . . . 682
6.5.3.1.3 Implementation Workflow . . . 683
6.5.3.1.4 Using deployed models . . . 684
6.5.3.2 Deploy Agent Factory with Hybrid Manager UI . . . 685
6.5.3.2.1 Overview . . . 685
6.5.3.2.2 Prerequisites . . . 685
6.5.3.2.3 Step 1: Configure Registry Authentication . . . 685
6.5.3.2.4 Step 2: Access Model Library . . . 686
6.5.3.2.5 Step 3: Deploy Model Server Cluster . . . 686
6.5.3.2.6 Step 4: Monitor Deployment Status . . . 687
6.5.3.2.7 Step 5: Configure API Access . . . 688
6.5.3.2.8 Step 6: Build Gen AI Applications . . . 688
6.5.3.2.9 Step 7: Update and Manage Models . . . 688
6.5.3.2.10 High availability and isolation for Langflow flows . . . 689
6.5.3.2.11 Troubleshooting . . . 689
6.5.3.2.12 Additional Resources . . . 689
6.5.3.3 External inference services . . . 690
6.5.3.3.1 Context . . . 690
6.5.3.3.2 Inference Services list . . . 690
6.5.3.3.3 Register an external inference service . . . 691
6.5.3.3.4 Retrieve inference service details . . . 693
6.5.3.3.5 Update inference service parameters . . . 695
6.5.3.3.6 De-register an external inference service . . . 696
6.5.4 Build with AI . . . 696
6.5.4.1 Langflow . . . 696
6.5.4.1.1 Prerequisites . . . 696
6.5.4.1.2 What's next . . . 697
6.5.4.2 Pipeline Designer . . . 737
6.5.4.2.1 How Pipeline Designer relates to AIDB . . . 737
6.5.4.2.2 Prerequisites . . . 737
6.5.4.2.3 Supported cluster types . . . 737
6.5.5 Operate & Observe AI Factory . . . 767
6.5.5.1 Observability (Agent Factory on HM) . . . 767
6.5.5.1.1 Observability in Hybrid Manager . . . 767
6.5.5.1.2 Accessing metrics and logs . . . 767

6.5.5.1.3 Typical commands . . . 767
6.5.5.1.4 Grafana dashboard setup for Agent Factory metrics . . . 768
6.5.5.1.5 Key takeaway . . . 768
6.5.5.2 Troubleshooting Agent Factory on Hybrid Manager . . . 769
6.5.5.2.1 Diagnostic Overview . . . 769
6.5.5.2.2 Quick Reference Links . . . 769
6.5.5.2.3 Infrastructure Issues . . . 769
6.5.5.2.4 Model Serving Failures . . . 770
6.5.5.2.5 Langflow and Gen AI application issues . . . 772
6.5.5.2.6 Log Analysis . . . 772
6.5.5.2.7 Alert Configuration . . . 772
6.5.5.2.8 Escalation Procedures . . . 773
6.5.5.2.9 Preventive Measures . . . 773

7 Analytics in Hybrid Manager 775
7.1 Configuring tiered storage . . . 775
7.1.1 Managing analytics for your cluster . . . 775
7.1.2 Viewing tables . . . 776
7.1.3 Converting to a tiered table . . . 776
7.1.4 Replicating to analytics . . . 777
7.1.5 Offloading to analytics . . . 777
7.1.6 Triggering compaction . . . 778
7.2 Managing catalogs . . . 778
7.2.1 Managing your cluster's catalog . . . 778
7.2.2 Querying analytical data across clusters . . . 779

8 Sovereign Data and Agent Factory 781
8.1 Why choose a hardware-based system? . . . 781
8.2 What's inside . . . 781
8.3 Supported workloads . . . 782
8.4 Hardware configurations . . . 782
8.5 Explore capabilities . . . 782
8.5.1 Concepts and architecture . . . 782
8.5.1.1 Node types and roles . . . 783
8.5.1.1.1 Administration node . . . 783
8.5.1.1.2 Hybrid Manager nodes . . . 783
8.5.1.1.3 Compute nodes . . . 783
8.5.1.1.4 AI nodes (optional) . . . 783
8.5.1.2 Orchestration and lifecycle . . . 783
8.5.1.3 What is "sovereign"? . . . 784
8.5.1.4 Supported workload patterns . . . 784
8.5.2 Frequently asked questions . . . 784
8.5.2.1 Why use this system instead of a cloud service? . . . 784
8.5.2.2 Who installs and maintains the hardware? . . . 784
8.5.2.3 What software does it run? . . . 785
8.5.2.4 Can I connect this system to cloud services? . . . 785
8.5.2.5 What models are supported for AI use? . . . 785
8.5.2.6 Can I monitor external databases? . . . 785
8.5.2.7 How do I scale the system? . . . 785
8.5.2.8 What's included in support? . . . 785
8.5.3 Glossary and key terms . . . 785
8.5.3.1 Administration node . . . 786
8.5.3.2 AI node . . . 786
8.5.3.3 Agentic workflow . . . 786

8.5.3.4 Control plane . . . 786
8.5.3.5 Compute node . . . 786
8.5.3.6 Embedding . . . 786
8.5.3.7 EDB Hybrid Manager . . . 786
8.5.3.8 Inference . . . 786
8.5.3.9 KServe . . . 786
8.5.3.10 Lifecycle services . . . 786
8.5.3.11 EDB Postgres Distributed (PGD) . . . 786
8.5.3.12 Retrieval-augmented generation (RAG) . . . 787
8.5.3.13 Sovereign AI . . . 787
8.5.3.14 Vector store . . . 787
8.5.4 System specifications . . . 787
8.5.4.1 Configuration options . . . 787
8.5.4.2 Configuration #1: Core Compute . . . 787
8.5.4.2.1 Administration node . . . 787
8.5.4.2.2 Core Compute – Hyper 2U Server (per server x6) . . . 788
8.5.4.3 Configuration #2: Advanced AI . . . 788
8.5.4.3.1 2-GPU Hyper 2U Server (per server x2) . . . 789
8.5.4.3.2 8-GPU B200 10U Server (per server x2) . . . 789
8.5.5 Common workloads and use cases . . . 790
8.5.5.1 Why use the engineered system over cloud? . . . 790
8.5.5.1.1 Data sovereignty and compliance . . . 790
8.5.5.1.2 Regulated environments . . . 790
8.5.5.1.3 Unified platform for Postgres + AI . . . 790
8.5.5.1.4 Predictable performance and cost . . . 790
8.5.5.1.5 Lifecycle-managed infrastructure . . . 791
8.5.5.2 Supported workload patterns . . . 791
8.5.5.2.1 High-availability Postgres . . . 791
8.5.5.2.2 Hybrid DBaaS . . . 791
8.5.5.2.3 Embedding and vector pipelines . . . 791
8.5.5.2.4 Retrieval-augmented generation (RAG) . . . 791
8.5.5.2.5 On-premises inference . . . 791
8.5.5.2.6 Agentic AI workflows . . . 791
8.5.6 What to expect . . . 791
8.5.6.1 Days 0–21: Site readiness and installation . . . 792
8.5.6.1.1 What happens . . . 792
8.5.6.2 Days 21–28: Deploying your first workloads . . . 792
8.5.6.2.1 What you can do . . . 792
8.5.6.3 Days 28–90: Lifecycle management and scaling . . . 792
8.5.6.3.1 What's included: . . . 792

EDB Postgres AI Hybrid Manager (HM) is the control plane of the EDB Postgres AI Platform—a **sovereign AI, analytics, and Postgres management solution** that runs anywhere you run Kubernetes. It offers a unified experience for deploying databases, running GenAI workloads, and scaling Postgres-native analytics across **cloud**, **on-premises**, or **hybrid environments**.

It is currently supported on the follow Kubernetes distributions for on-premises: **Red Hat OpenShift (RHOS)**, **Rancher RKE2**, and for in-cloud: **Amazon EKS**, and **Google GKE**.

CHAPTER 1

What you can do

Area	Capabilities
Database provisioning	Deploy highly available PostgreSQL or Distributed Postgres clusters using the console, templates, or declarative manifests.
AI integration	Run sovereign GenAI pipelines , serve models with KServe, and build AI knowledge bases governed by your infrastructure.
Analytics	Offload cold data to Delta Lake or Iceberg , query with Postgres syntax, and enable hybrid transactional/analytical processing.
Monitoring and observability	Use the integrated dashboards that leverage Prometheus, Grafana, and Loki to obtain deep visibility into query health, resource usage, and system alerts.
Security and access management	Integrate with identity providers, define RBAC for projects and clusters, and manage secure access to your databases and services.
Database migration	Migrate schemas and data from external databases (Oracle, self-managed Postgres) to HM-managed clusters. Leverage integrated services for migration and schema assessment, data transfer (snapshot), and continuous replication (streaming) to achieve minimal downtime migrations.

1.1 Start here

- *Install Hybrid Manager* → A detailed guide to walk you through a full HMinstallation.
- *Sovereign Data and Agent Factory* → Sovereign Data and Agent Factory hardware
- *Using Hybrid Manager* → backup/restore, monitoring, migrations, cluster creation, multi-DC deployment and more
- *Getting started with Agent Factory on Hybrid Manager* → sovereign GenAI pipelines, model serving, and retrieval systems
- *Analytics in Hybrid Manager* → Delta Lake, Iceberg, tiered storage, and lakehouse patterns
- *Hybrid Manager 2025.11.0 release notes*

CHAPTER 2

Hybrid Manager Innovation Release Notes

!!!info Release cadence update HM now follows a dual release strategy: **LTS** and **Innovation Releases**. For details on this new model, see *Hybrid Manager dual release strategy* .

Version	Release date
Hybrid Manager 1.4.0 release notes	24 June 2026

2.1 Hybrid Manager dual release strategy

With Hybrid Manager (HM) 2025.11, we are adopting a new release strategy designed to deliver new capabilities and stability improvements to you faster and more predictably.

Notice that in addition to the traditional release scheme (with semantic versioning such as 1.3.0), we added a date-based release scheme (with version numbering such as 2025.11), which we are calling our Innovation Releases. This change supports our commitment to continuous delivery and clearer release cadences:

- **Innovation releases (monthly):** These releases, such as 2025.11, are delivered monthly. They focus on continuous platform hardening, workflow improvements, stability, reliability and the introduction of smaller, iterative features. They're great for customers and teams that want to be on the leading edge of development, and see where we're taking the product.
- **Long-term support (LTS) releases (bi-annual):** These releases, such as 1.3.0, are delivered twice a year. They consolidate the contents of previous Innovation Releases into a stable foundation for deployments requiring an extended support commitment. These LTS releases are supported for 15 months from their release date, with monthly patch releases, such as 1.3.1, that include bug and security fixes.

For up-to-date platform compatibility of Hybrid Manager, please refer to our Platform Compatibility page .

2.1.1 Innovation Release release notes

These notes cover the Innovation Releases that culminated in the 1.4 LTS release of Hybrid Manager:

Version	Release date
Hybrid Manager 2026.5.1 release notes	2 June 2026
Hybrid Manager 2026.5.0 release notes	19 May 2026
Hybrid Manager 2026.4.1 release notes	23 Apr 2026
Hybrid Manager 2026.3.0 release notes	12 Mar 2026
Hybrid Manager 2026.2.0 release notes	12 Feb 2026
Hybrid Manager 2026.1.0 release notes	15 Jan 2026
Hybrid Manager 2025.12.0 release notes	11 Dec 2025
Hybrid Manager 2025.11.0 release notes	07 Nov 2025

2.1.1.1 Hybrid Manager 1.4.0 release notes

Hybrid Manager (HM) 1.4.0 (LTS) expands deployment flexibility with Microsoft AKS support, simplifies day-to-day operations with a new automations engine, and deepens AI capabilities across pipelines, migration, and the built-in chatbot. This long-term support release consolidates all features from Innovation Releases 2025.11 through 2026.5.1. It also includes numerous bug fixes and quality-of-life improvements.

Release Date: June 24, 2026

!!!note Postgres refers to all distributions supported by EDB

- PostgreSQL
- EDB Enterprise Postgres - formerly, EDB Postgres Extended Server (PGE)
- EDB Enterprise Postgres (Oracle Compatible) - formerly, EDB Postgres Advanced Server (EPAS)
- ** *Azure Kubernetes Service (AKS) support* **: HM can now be deployed on Microsoft AKS, with full support for AI Pipelines, Analytics Accelerator, and the Data Migration Service.
- ** *Selective installations* **: Customize your HM deployment by enabling only the `scenarios` you need — `core` , `dbaas` , `migration` , `ai` , and `analytics` — at install time or later, reducing resource usage and simplifying component management.
- ** *Automations engine* **: A new automation framework handles routine database operations including disk auto-scaling, CPU and memory scaling, index application, and minor and patch database version updates — all with configurable maintenance windows and optional human-in-the-loop approvals.
- ** *OSS Library* **: Deploy community database tools — Apache Airflow, Apache Superset, Metabase, pgAdmin 4, pgBadger, and HammerDB — directly to HM projects from the** Asset Library**, with configurable parameters and approval workflows.
- ** *What the Chat Agent can do* **: A conversational assistant built into the HM console for Postgres cluster management, estate browsing, monitoring and alerts, health report retrieval, Oracle-to-Postgres SQL translation, and documentation Q&A with citation-backed answers. Supports models from OpenAI, Anthropic, and Google Gemini, as well as self-hosted models deployed through HM.
- ** *Custom load balancer configuration* **: HM supports custom load balancer annotations for use cases such as MetalLB address pool assignment and disabling Azure Private Link on Azure private clusters.
- ** *Three-location cluster deployments* **: Deploy Postgres clusters across three geographically separated locations for zero-data-loss configurations via synchronous replication. HM also supports Distributed High Availability (DHA) Witness Groups in a third location for quorum and split-brain prevention.

- ** *Unified schema migration* **: Migrate database schemas from Oracle and Postgres sources to Postgres destinations directly through HM, with Oracle SQL compatibility assessment and API-driven automation via AI agents.
- ** *Model Context Protocol (MCP) integration* **: HM now exposes its API capabilities through MCP, enabling AI agents and developer tooling to consume HM APIs for agentic workflows.
- ** *Agent Factory enhancements* **: Visual AI Pipeline Designer, multistep composable pipelines, hybrid search, the `summarize_text_aggregate()` SQL function, and expanded Langflow components including EDB Airman MCP Server, EDB Embedded Models, and EDB Vector Index.
- ** *Langflow integration* **: Langflow is natively integrated into the HM Launchpad. You can build AI-powered agents and workflows within HM, deploy flows as managed workloads, and use a Langflow template for end-to-end database migrations from Oracle and Postgres sources to Postgres destinations.
- ** *Enhanced migration risk assessment* **: Granular assessments for source databases covering replication slot health, WAL generation rates, and risky table configurations (primary keys, `REPLICA IDENTITY`, `TOAST`, and unsupported data types for the Data Migration Service).
- ** *Advanced observability* **: Daily cluster health reports, schema change detection, new query detection, a metrics comparison tool, and enhanced monitoring dashboards.
- ** *ARM architecture support* **: HM components now run on ARM-based architectures, enabling deployments on modern ARM platforms and Kubernetes nodes.

2.1.1.1.1 Release details

Hybrid Manager (HM) platform

Type	Description
Platform support	Added support for Kubernetes 1.34 and OpenShift 4.21.
Platform support	Added ARM64 architecture support for HM, extending deployment options to ARM64-based Kubernetes environments. See the *Troubleshooting* for the supported configurations.
Feature	Added support for HM deployments on Microsoft Azure Kubernetes Service (AKS), with full support for AI Pipelines and the Data Migration Service (DMS).
Feature	Added support for bringing your own identity provider (IdP), configurable from the HM console (**Settings > Identity Providers**). HM can now automatically assign user roles based on IdP group attributes, declaratively mapping user groups to roles.
Feature	Added the Chat Agent, a conversational assistant in the HM console for informational queries and Postgres cluster management.
Feature	HM now exposes its API capabilities through the Model Context Protocol (MCP), enabling developers and AI agents to consume the HM API for agentic use cases, including integration with Langflow for streamlined database agent configuration.
Feature	The HM console now includes Langflow in the Launchpad for building and testing AI-powered agents and workflows directly within the platform.
Feature	HM now supports selective installations. Customize your deployment by enabling only the *scenarios* you need (*core*, *dbaas*, *migration*, *ai*, and *analytics*) at install time or later, reducing resource usage and simplifying component management.
Feature	Added support for deploying Postgres clusters across three geographically separated locations, enabling zero-data-loss architectures through synchronous replication. HM also supports deploying Distributed High Availability (DHA) Witness Groups in a third location to maintain quorum and prevent split-brain scenarios during automated failovers.

continues on next page

Table 3 – continued from previous page

Type	Description
Feature	Introduced the **Automations** engine for automating routine database operations. Available automations include database storage auto-scaling, CPU and memory auto-scaling, automatic index application, and automatic minor and patch image upgrades. The framework features project-level maintenance windows, granular execution controls, optional human-in-the-loop approvals, and an integrated **Task Manager** for centralized monitoring of all automated activities.
Feature	Introduced OSS Library, a catalog of community database tools deployable directly to HM projects. The initial catalog includes Apache Airflow, Apache Superset, Metabase, pgAdmin 4, pgBadger, and HammerDB. Apps are deployed from the **Asset Library > Apps** section, with configurable parameters, optional deployment scheduling, and human-in-the-loop approval workflows.
Feature	Added the ability to initiate a write-leader switchover for Distributed High Availability clusters directly from the cluster detailed view in the HM console.
Feature	Added API audit logging. HM now emits structured JSON audit log entries for all authenticated API requests through the ingress gateway, capturing user, session, API key, and response details for compliance and security auditing. Logs are queryable via Grafana or *kubectl*.
Feature	Added dynamic PGD previews, enabling real-time preview capabilities for PGD configurations during cluster provisioning and management workflows.
Feature	Added *nodeSelector* and *tolerations* support in edbctl for Primary/Standby Replication (PSR) cluster create and edit operations, enabling dedicated and isolated node pools for Postgres workloads.
Enhancement	Added support for *nodeSelector* and *tolerations* in the Postgres cluster provisioning and patch API for advanced workload isolation.
Enhancement	The Chat Agent now uses a selection list interaction method and a separate password transmission path to ensure passwords aren't uploaded to the large language model.
Enhancement	Added support for publishing Langflow flows as bundles and deploying them within HM.
Enhancement	Simplified the multi-location (multi-dc) deployment procedure. Multi-dc topology is now primarily controlled via *primary.yaml* and *secondary.yaml* configuration files, supporting standard Helm practices.
Enhancement	The cluster detailed view now lists all group nodes of Distributed High Availability clusters, providing visibility into current topology and status before and after a switchover operation.
Enhancement	You can now change network configuration for Witness groups through the HM console.
Enhancement	Added global configuration parameters for HM ingress load balancers, providing more granular control over HM platform networking and traffic management.
Enhancement	Added support for Custom Loadbalancer annotations in EDB Postgres AI agent (*beacon-agent*) provisioning across supported platforms.
Enhancement	Enhanced load balancer configuration: *private* mode now allows custom annotations alongside HM default private annotations; *public* mode supports custom annotations, HM default private, and HM default public annotations.
Enhancement	Added a Custom Loadbalancer Configuration option for network access when creating or editing clusters, allowing custom load balancer service annotations (key-value pairs) for read-write and read-only connections.
Enhancement	Added support for custom Kubernetes Service annotations on Kafka load balancers, including the Data Migration Service's Kafka services, enabling BYO-network and on-premises deployment scenarios.

continues on next page

Table 3 – continued from previous page

Type	Description
Enhancement	HM now supports configuring dedicated object storage per HM project, including via the CLI using *edbctl project create-object-store*.
Enhancement	HM cluster APIs now provide more visibility when provisioning fails, enabling self-service diagnostics and faster time to resolution.
Enhancement	Added optional read-only connection strings for HA and DHA clusters to route read operations to standby replicas or non-write-leader nodes, reducing load on the primary.
Enhancement	Added advanced compression to HM backup capabilities through Barman Cloud integration. WAL compression is now enabled by default, using Snappy compression for basebackups and zstd for WAL files.
Enhancement	Introduced the consolidated system database (*upm-system-db*).
Enhancement	Enhanced the database notification system with expanded event sources, including agentic automated tasks generated by the platform. All resource types in the audit log table are now clickable.
Enhancement	Added support for GitLab container registry as a registry provider for image discovery.
Enhancement	Enhanced the image library **Clusters Using This Image** view to only show clusters you have permission to access, based on your user role and project permissions.
Enhancement	Added support for configuring *kapp-controller* TLS settings in the HM operator Helm chart. You can now set a custom CA certificate trust or disable TLS verification for private container registries.
Enhancement	Added a machine-readable API index following the llms.txt convention, providing a streamlined reference of available endpoints for LLM-powered coding assistants.
Change	Changed the default value of the *edb_stat_monitor.edbsm_normalized_query* GUC from *false* to *true*. Values used in predicates are no longer saved in HM by default, improving data privacy.
Change	Decoupled the HM operator from HM releases to enable independent versioning and deployment cycles. The operator now enforces the use of the remote method for operations, overriding any locally specified methods to ensure consistent cluster management behavior.
Change	Automated the intermediate steps required when upgrading HM to a new version. Previously, upgrades required manually executing a series of Kubernetes operations such as patching deployments, annotating resources, and applying CRDs. These steps are now handled automatically once the user initiates an upgrade by specifying the target version.
Change	Embedded the HM plugin into the edbctl CLI. The edbctl CLI now includes *hm status*, *hm component suspend*, *hm component resume*, and *hm component retry* commands.
Change	Reduced the minimum deployment footprint by consolidating multiple databases into a single database, decreasing operational overhead.
Change	Added support for an optional in-cluster container registry, enabled through the operator Helm chart.
Change	The prometheus-operator component can now be disabled when it's already installed on the target platform, rather than only annotating CRDs to avoid conflicts.
Change	Simplified load balancer and NodePort configuration. You can now configure both Istio and database services to use load balancers or NodePorts through a single parameter in the custom resource.
Change	Enhanced preflight checks to validate object storage connectivity before deployment begins, preventing failures caused by incorrect object storage secrets.
Change	Added support for configuring all PGD GUCs through HM, including previously unavailable settings like *bdr.ddl_replication*.

continues on next page

Table 3 – continued from previous page

Type	Description
Change	Database cluster management capabilities — including provisioning, backups, storage locations, and cluster templates — are now controlled by a dedicated *dbaas* scenario, separated from the *core* scenario. New installations include *dbaas* by default. **Existing installations upgrading from 2026.3 must explicitly add `dbaas` to their scenarios list** to retain Postgres cluster management functionality. Without *dbaas*, cluster management APIs return 404 and the **Clusters** section is hidden in the HM console.
Deprecation	Deprecated the AHA – Advanced High Availability (PGD-S) cluster type. The cluster type is disabled in the HM console, preventing new AHA cluster provisioning. AHA is also deprecated in the MCP and chatbot components, and deprecation notices have been added to the API documentation. Existing AHA clusters continue to function normally. Plan to migrate existing AHA clusters to PGD single distributed group configurations.
Deprecation	Removed the Apache Spark component bundled with HM. Connecting to external Spark clusters remains supported. No action required. This component isn't replaced.
Deprecation	Deprecated the *beaconServer.additionalTrustDomains* field in the HM CRD. No action required — the field is still accepted by the CRD for backward compatibility but is no longer used. Any value set has no effect. This field has no replacement.
Deprecation	Deprecated the *bootstrap* and *edbpgai-bootstrap* Helm chart installation methods. **Action required:** migrate to the unified HM Helm chart. See *Troubleshooting* .
Deprecation	Removed the Lakehouse Analytics Cluster feature from the HM console. No action required — existing clusters are unaffected and no migration to another cluster type is needed. Analytics functionality is now available via PGAA in Distributed High Availability (DHA) clusters and in single-node Postgres clusters with the PGAA extension enabled.
Deprecation	Removed the *upm-beacon-analytics-cnpg-plugin* component, which was tied to the Lakehouse Analytics Cluster feature. No action required. Use PGAA in DHA or single-node Postgres clusters instead.

Agent Factory

Type	Description
Feature	Added the ability to create, manage, and view all AIDB knowledge bases directly within HM.
Feature	Added the ability to review available AI models and add new ones directly from the HM interface, with essential model metadata including context window limitations and optimal use cases.
Feature	Agent Factory now supports Postgres 18.
Feature	AI Pipelines now includes Visual Pipeline Designer — a web-based graphical design tool for developing, deploying, and managing AI data preparation pipelines, integrated into HM.
Feature	Added support for accessing external services for model serving, allowing you to connect and use your own remote AI models within HM, data pipelines, and the HM chatbot.
Feature	Extended the AIDB pipeline framework to support multistep, composable pipelines. Chain preparer functions (for example, PDF to text, HTML to text) so that the output of one step feeds into the next, enabling flexible pipeline composition through the visual pipeline designer.
Feature	Added the *summarize_text_aggregate()* function.

continues on next page

Table 4 – continued from previous page

Type	Description
Feature	Added a PDF-to-image conversion pipeline step to AIDB, supporting complex table layouts through a pipeline that converts pages to images and applies OCR (PaddleOCR) to extract text for downstream embedding and retrieval.
Feature	Expanded AIDB coverage in Langflow with hybrid search support for the EDB Knowledge Base component, and three new components: EDB Airman MCP Server; EDB Embedded Models, for CPU-based text and image inference; and EDB Vector Index, for creating vector indexes on EDB knowledge bases.
Feature	AI Pipelines now supports multiple compatible pipelines writing into a single shared knowledge base, enabling hybrid knowledge bases from multiple sources and formats. The vector table includes a new *pipeline_id* column to track source pipelines per embedding.
Feature	Added support for deploying Langflow flows to Kubernetes environments managed by HM, with configurable QoS constraints including memory, CPU allocation, autoscaling, and restart policies. Deployed flows include full observability and alerting integration.
Feature	Added hybrid search capabilities to AI Pipelines, combining full-text search, vector/semantic search, and predicate filtering. Includes convenience functions and documentation examples for common use cases.
Enhancement	You can build AI workflows using Langflow and publish your flows as assets on the HM platform, then run them as standalone jobs.
Enhancement	Added an error log and dead letters queue to improve AI pipeline error handling. Track which files or records succeed and which encounter errors during pipeline execution.
Enhancement	Added non-blocking error handling to AI pipelines. Pipelines now continue processing when errors affect only individual records, while still halting on pipeline-level errors. Use *aidb.get_error_logs()* and *aidb.clear_error_logs()* to query and manage errors.
Enhancement	Added support for creating AI data pipelines for on-premises databases and external CloudNativePG clusters.
Enhancement	Added AIDB pipeline support for external Postgres clusters managed by HM, including clusters running on secondary locations and outside the HM platform.
Enhancement	The AI Pipeline Designer now retrieves available models dynamically instead of using static defaults, enabling model selection based on specific project, resource, and database combinations.
Enhancement	Added support for configuring the PDF to Image pipeline step directly in the visual AI Pipeline Designer, including format, DPI, page ranges, and annotation rendering.
Enhancement	Added support for AIDB 7 in the AI Pipeline Designer, including improved knowledge base validation and fixes for pipelines with existing knowledge base tables and PGD clusters.
Enhancement	Updated the AI Pipeline Designer to support AIDB 7's multi-pipeline Knowledge Base model. Knowledge bases can now be associated with multiple pipelines. The Pipeline Designer displays KB table references using full table format (*vector_schema.vector_table*); pipeline creation supports selecting existing KB tables; and the KB list view shows associated source pipelines with hyperlinks.
Enhancement	Added support for registering external inference services with insecure connections. The system requires explicit user acknowledgment of the security risk.
Enhancement	Added support for Hugging Face as a new model provider, configurable via Hugging Face Model Name, Object Storage path, or both.
Enhancement	Extended the invalid model name error message — previously only available on the Internal Model Cluster page — to the external inference service configuration page.
Enhancement	Enhanced error handling in the AI model creation form to display detailed error messages, improving user feedback when model creation requirements aren't met.

continues on next page

Table 4 – continued from previous page

Type	Description
Enhancement	Added automatic retry for transient LLM errors in AI workflows, improving reliability when inference providers return temporary failures.
Enhancement	Restricted Langflow access to users with the AI Model Manager role for tighter control over AI workflow authoring.
Enhancement	You can now search AI models and services by tag name in the HM console.
Enhancement	Enhanced the chatbot to render data from metrics APIs using Mermaid charts, providing improved visualization of database metrics and monitoring data.
Enhancement	Added support for additional AI model families in the chatbot, including GPT-5 variants (mini, nano, codex), all Gemini models, and Anthropic models.
Enhancement	The conversation interface now displays the model name, providing greater transparency about which AI model is being used for each conversation.
Enhancement	Added health check validation when registering inference services. HM now detects invalid model endpoints during registration and provides detailed error messages. You can test registered inference services using a test button, and a new metrics endpoint exposes credential check results for Prometheus monitoring.
Enhancement	Added knowledge cutoff awareness to the HM documentation skill. Documentation snapshots now include compilation timestamps, and the AI assistant displays appropriate disclaimers when answering questions about topics that might not be covered in the current documentation snapshot.
Change	Removed the resolver component from the EDB Postgres AI agent (*beacon-agent*) and updated the agent to use AIDB 7 views directly, as part of supporting multi-pipeline knowledge base capabilities.
Change	Removed the AI Pipelines **Default Settings** page from the **Sovereign AI** section, as it was incompatible with the new instance-specific models list approach.

Observability and monitoring

Type	Description
Feature	Added proactive recommendation categories for **Configuration** and **Security** . Configuration recommendations suggest optimizing Postgres system parameters such as *shared_buffers*. Security recommendations flag potential security issues and provide actionable guidance for continuous monitoring of database environment integrity.
Feature	Added new critical alerts to monitor pods terminated by Out of Memory (OOM) events, backups exceeding 12 hours, and disk/WAL usage thresholds.
Feature	Users can now use event annotations on time-series charts as delimiters for selecting time periods, enabling easier isolation and analysis of data ranges associated with specific cluster events.
Feature	Added cluster health reports, automatically generated once a day for each HM-managed Postgres cluster. Reports are retained longer than raw metrics and can be used to track long-term health trends, spot anomalies, and share performance snapshots. The Chat Agent can retrieve and compare reports.
Feature	Added schema design recommendations via the chat agent. The chat agent assesses schema suitability to the running workload and provides actionable recommendations to improve performance.
Feature	Added support for monitoring multiple external PGD groups across different Kubernetes clusters as a single logical cluster, enabling unified visibility and topology management.

continues on next page

Table 5 – continued from previous page

Type	Description
Feature	Added detection of queries that haven't been executed on a monitored database before. HM identifies new queries within one day, with mechanisms to disable detection and suppress excessive notifications.
Feature	Added schema change detection that automatically identifies changes in monitored database schemas and emits activity events to the cluster activity feed, enabling you to spot unexpected modifications, receive notifications, and compare database states before and after changes.
Feature	Added a metrics comparison tool to compare database metrics and query statistics between two user-defined time periods, with side-by-side or overlaid chart views, filters, zoom controls, and highlighted differences.
Enhancement	Self-managed PGD clusters now have full feature parity for replication matrices and lag monitoring.
Enhancement	Added API capabilities to annotate host metrics charts with activity logs as vertical dotted lines, enabling correlation of specific cluster events with host performance at exact points in time.
Enhancement	The GUC *query_advisor.exclude_schema_list* now includes predefined default values to automatically exclude internal database schemas from Query Advisor recommendations.
Enhancement	Added support for monitoring of self-managed EFM and PGD clusters.
Enhancement	The **Disk Throughput** chart now displays read and write throughput separately for more accurate monitoring.
Enhancement	HM now provides deeper deployment insights via pod-level status conditions in the cluster APIs.
Enhancement	Extended observability support for external PGD clusters to include CNPG-GC clusters.
Enhancement	Added a new *EnableEDBUsageReporting* configuration option for the beacon server with retry logic that attempts reconnection to IT links every 24 hours when unreachable.
Enhancement	When a cluster spec is updated — by a user change, autoscaling, or agentic automation — an annotation is automatically generated and displayed in the observability view.
Enhancement	Added configurable limits for Prometheus query requests to prevent resource exhaustion from overly broad queries. Store components now support *request-samples* and *request-series* limits.
Enhancement	Added an Annotations API to enable marking selected activity events on the monitoring dashboard, providing a foundation for automated operations.
Enhancement	Added WAL metrics collection for on-premises Postgres instances monitored by the EDB Postgres AI agent.
Enhancement	Added a *validate* command to the EDB Postgres AI agent (*beacon-agent*) for verifying database connectivity before ingestion.
Enhancement	HM now collects and displays cluster status metrics from self-managed EDB Failover Manager clusters.
Change	Updated the default alert evaluation interval to 2 minutes.
Change	Added tracing support to Thanos, Prometheus, and Grafana deployments, enabling detailed performance analysis of PromQL queries and better evaluation of monitoring system performance configurations.
Change	Removed CPU limits from EDB Postgres AI agent deployments to prevent throttling and improve monitoring performance.
Change	Enabled auto downsampling by default in Thanos Query, improving query performance for large time ranges by automatically using lower-resolution data when appropriate.

continues on next page

Table 5 – continued from previous page

Type	Description
Change	Configured Istio to automatically expire old metric series for vanished peers on sidecars, preventing unbounded metric growth over time as peers come and go.
Change	Updated the beacon readiness probe to report ready status only after the EDB Postgres AI agent is fully booted with all connections and services active.
Change	Modified backend logic to ensure time data sent to charts includes appropriate timezone information or is converted to the user's local time for accurate chart rendering.

HM console

Type	Description
Feature	Added AI-powered query generation to the Query Editor. Select **Generate with AI** to create SQL queries using natural language prompts targeting data across your estate.
Enhancement	Added the ability to view database health reports directly in the HM console without downloading the PDF.
Enhancement	Added a configuration interface for HM authentication policies in the **Settings** section, including idle timeout for web login sessions.
Enhancement	Added an **Invalidate User Sessions** action to the Identity Providers management interface, allowing administrators to invalidate all active sessions for a specific identity provider.
Enhancement	Added filters in the role assignment interface to display only manually assigned roles, excluding roles assigned via IdP group mappings.
Enhancement	Added the *beacon.enterprisedb.com/exclude-storage-from-dbaas=true* label to exclude specific *StorageClass* or *VolumeSnapshotClass* resources from the **Create Cluster** dropdowns in shared Kubernetes environments.
Enhancement	You can now filter clusters by cluster or group ID in the HM console.
Enhancement	Added a searchable model-service picker in the HM console.
Enhancement	You can now display dates in a custom timezone in the HM console.
Enhancement	Improved keyboard accessibility in the HM console by adding keyboard event handlers to interactive elements that previously only supported mouse interactions.

Analytics

Type	Description
Feature	Added the ability to configure tiered tables directly from the HM console. Users can select and convert an existing table into a tiered table structure within the console.
Feature	Postgres Analytics Accelerator now supports Postgres 18.
Enhancement	Added insights and visual metrics for tiered tables, providing visual proof of tiered table configurations and moving diagnostics from the command line to the HM console.
Enhancement	Postgres Analytics Accelerator now supports reading and writing Delta and Iceberg tables in Azure Data Lake Storage and Azure Blob Storage.
Enhancement	Updated the PGAA version shipped with HM to 1.9.1 .
Enhancement	Enhanced Lakekeeper warehouse creation to support object storage environments that don't support AWS Security Token Service (STS). The system now automatically retries warehouse creation with STS disabled if the initial attempt fails on S3-compatible storage.

Migrations

Type	Description
Platform support	HM migration features are now fully supported in air-gapped on-premise environments (RKE and RHOS deployments).
Feature	Added a Langflow template for AI-powered, end-to-end database migrations from Oracle, and Postgres sources to Postgres destinations, including provisioning HM-managed clusters, migrating schemas, and performing snapshot plus streaming data migration. Oracle sources require schema conversion via Migration Portal before running the migration flow.
Feature	The HM console **Sources** and **Destinations** migration pages now display the *RW_SERVICE_HOST* environment variable needed by the Data Migration Service (DMS) agent (*cdcagent*) to connect to the HM instance, enabling self-service retrieval of this value.
Feature	HM now automatically creates Migration Portal projects when source database schemas are ingested by the EDB Postgres AI agent (*beacon-agent*), eliminating the need to manually create a Migration Portal project in most cases.
Feature	Added a replication slot health monitor that tracks the state of logical replication slots on the source Postgres database, reporting health status, replay lag, and disk retention for each slot in the downloadable metrics report.
Feature	Added a dedicated metrics fetcher that runs independently of schema change detection, enabling continuous monitoring of database metrics such as WAL generation rate at regular poll intervals.
Feature	Added a per-table tuple mutation rate assessment that estimates each table's contribution to WAL generation, enabling more granular migration planning for source Postgres databases.
Feature	Added a global WAL generation rate assessment that monitors WAL accumulation on the source Postgres database, helping estimate storage capacity requirements during migration snapshots.
Feature	Added a migration assessment that checks for risky combinations of primary key, *REPLICA IDENTITY*, and *TOAST* configurations in Postgres tables, identifying scenarios where the Data Migration Service (DMS) might silently miss changes.
Feature	Added a migration assessment that identifies Postgres data types unsupported or risky for DMS-based CDC, providing early warnings about potential compatibility issues before starting a migration.
Feature	Added a public API endpoint for downloading Migration Portal assessment reports in JSON format.
Feature	Added the ability to enter and display metadata about applications associated with databases and schemas in many-to-many relationships, providing better context for migration planning and prioritization.
Feature	Added the ability for the EDB Postgres AI agent (*beacon-agent*) to automatically configure Oracle and Postgres database connections from Oracle *TNSNAMES.ORA* files, CSV files, or AWS RDS auto-discovery.
Feature	Added unified schema migration capabilities to migrate database schemas from Oracle or Postgres sources to Postgres destinations directly through HM, with support for HM-managed and externally managed destinations, and API access for AI agent automation.
Feature	Added the ability to assess Oracle application SQL statements for compatibility with EDB Postgres Advanced Server, supporting SQL extracted from Oracle trace files and system views, as well as statements provided via the HM console or API.

continues on next page

Table 8 – continued from previous page

Type	Description
Feature	Added a **Metrics** tab to the Migration details page in the HM console. While a migration is active, the tab links to a Grafana dashboard pre-filtered to that migration, showing throughput (rows/s, bytes/s), batch latency, commit latency, and batch sizing for both the read and write sides of the DMS agent.
Enhancement	HM migrations now include a migration **Error history** tab that provides detailed error messages including error source, activity status, timestamp, and full log outputs to help with troubleshooting.
Enhancement	The Oracle database details page now includes a schema compatibility summary describing database compatibility as reported by the Migration Portal project.
Enhancement	Migration Portal now includes information on dependent objects and schemas to prevent related incompatibility issues such as missing relation references.
Enhancement	The Postgres database details page **Assessment** tab now includes new sections listing configuration items that could impact migration complexity: extensions installed, foreign servers, foreign tables, tablespaces, and installed procedural languages.
Enhancement	The **Migrate > Migrations** list view now includes **Migration Type** and **Migration Progress** columns. The Migration Progress column shows live metrics per migration: total tables included, snapshot tables completed, and cumulative rows migrated. The **Tables** tab in the migration details view shows per-table snapshot status, mapped columns, rows migrated, and migration rate (rows/s).
Enhancement	Enhanced the EDB Postgres AI agent and related services to better support ingestion, assessment, and concurrent migration of large numbers of external databases.
Enhancement	DMS has been tested and certified to support migrations to HM-managed Postgres 18 databases, and from or to externally managed Postgres 18 databases.
Enhancement	The migration assessment in the detailed view now provides clearer diagnostics regarding Replica Identity for Postgres sources.
Enhancement	Improved the search for migration statuses in the **Estate** > **Migrations** tab for faster results.
Enhancement	Error logs for ongoing migrations now provide minute-level data for easier troubleshooting.
Enhancement	Added new DMS agent environment variables for finer control over the schemas and tables in a source database that are available for data migrations.
Enhancement	Added a section to the Postgres database details page to display tablespace usage, listing the count of object types by schema for each tablespace.
Enhancement	Improved DMS snapshot performance by deferring primary key application until after the snapshot phase. Composite primary keys are collected and applied in order post-snapshot, with support for deduplication between snapshot and streaming phases.
Enhancement	Extended the Migration Portal assessment reports API to support bulk downloads. The new endpoint accepts multiple database resource IDs and returns assessment reports in a single JSONL file.
Enhancement	When a Postgres database is registered for migration, its schemas are now automatically ingested, parsed, and stored in the Migration Portal backend database and made available for migration via new API behaviors.
Enhancement	The migration agent now honors upstream cancellation signals when extracting Oracle schemas, enabling cleaner shutdown behavior.
Enhancement	Added support for replicating PostgreSQL *DOMAIN* types based on *DATE*, *TIME*, *TIMESTAMP*, *INTERVAL*, *UUID*, *JSON*, *XML*, and *POINT* types in both snapshot and CDC replication modes.

continues on next page

Table 8 – continued from previous page

Type	Description
Enhancement	Added concurrent schema finalization for *schema_and_data* migrations. After the snapshot stage completes, you can select **Finalize Schema Concurrently** to apply indexes, foreign keys, and other schema objects in the background using non-blocking DDL variants, significantly reducing cutover time.
Enhancement	Extended Postgres-to-Postgres schema migration in the DMS to handle Postgres-specific database objects — including custom domain types — during schema migration, ensuring correct dependency ordering for complete end-to-end schema migration between Postgres databases.
Enhancement	Enhanced the Simple Migration Agent Langflow template with an improved architecture leveraging unified schema migration, streamlined agent components, automatic password generation, database setup, and support for Llama 3 Nemotron 49B and Nano 4B models.
Enhancement	The Simple Migration Agent Langflow template is passwordless, using global secrets management to handle credentials without exposing them to the large language model (LLM).
Enhancement	Optimized the migration creation form to skip table loading and validation when the migration scope is set to schema-only.
Change	Updated the Migration Copilot with improved AI models: GPT 5.2 is now the default model, with support for validated Nemotron 49B Llama3 as an alternative.
Change	Added support for per-database binary path configuration in the EDB Postgres AI agent (*beacon-agent*). You can now specify a custom *pg_bin_path* for each database entry in *beacon_agent.yaml*, enabling a single *beacon-agent* to connect to multiple PostgreSQL flavors that require different *pg_dump* and *pg_dumpall* binaries.
Change	Added pre-flight validation for Postgres table *REPLICA IDENTITY* before enrolling tables in DMS *publications*. Tables without primary keys or proper *REPLICA IDENTITY* settings are rejected at setup time with clear remediation steps, rather than causing *UPDATE*/*DELETE* operations to fail after migration begins.
Change	Added support for PostgreSQL 18 *pg_dump* and *pg_dumpall* output parsing, including new psql meta-commands (*restrict* and *unrestrict*) and improved ROLE object classification.
Change	Updated the Migration Copilot to use hardened Echo base container images, improving security posture and reducing vulnerability exposure.
Deprecation	The *domain_name* parameter moved from *componentsParameters["transporter-rw-service"]* to *globalParameters.dms_domain_name* in an earlier Innovation Release. If you're upgrading directly from the latest patch version of HM 1.3.x, this migration is handled automatically by the *hmupgrader* tooling. Users who followed the Innovation Release track will have already applied this change.

2.1.1.2 Contributing Innovation Releases

2.1.1.2.1 Hybrid Manager 2025.11.0 release notes

This innovation release includes features and enhancements.

Release Date: November 07, 2025

!!!info Release cadence update

HM now follows a dual release strategy: **LTS** and **Innovation Releases**. For details on this new model, see *Hybrid Manager dual release strategy* .

- **GenAI Builder interface**: Introduced visual tools for creating and managing knowledge bases and enabled

model review and configuration directly within HM.

- **Tiered tables observability and configuration**: Enabled tiered tables configuration directly from the HM console and provided enhanced observability for Lakehouse queries and configurations.
- **Switchover for Advanced High Availability and Distributed High Availability clusters**: Added the ability to initiate write-leader switchovers directly from the HM console.
- **Migration transparency and debugging**: Added detailed migration progress visibility and error history directly in the HM console, including table status, real-time migrated row counts, error sources and log outputs.
- **Database configuration and security recommendations**: Expanded the database **Recommendations** tab to include proactive checks for database configuration and security.

Model and knowledge base management in the GenAI Builder UI

Knowledge base management: Users can now create, manage, and view all AIDB knowledge bases directly within HM. All existing knowledge bases created in the database are now visible from the GenAI Builder interface.

Model review and configuration: You can now review available models and add new ones directly from the interface. The interface provides essential model metadata, including context window limitations and optimal use cases, simplifying model selection and configuration for your RAG pipelines.

Analytics and Lakehouse integration

Tiered tables configurable in the HM console: The complex task of setting up tiered tables is now a simple HM console action. Users can now select and convert an existing table into a tiered table structure directly within the HM console.

Improved observability for tiered tables: We've added insights and visual metrics for tiered tables and Lakehouse queries. This provides visual proof of the setup and expected functionality of your tiered table configurations, moving diagnostics from the command-line to the HM console.

Unified Lakehouse integration: You can now view connected external Apache Iceberg tables on the dedicated Lakehouse or Catalog page. This unified view allow you to see and manage all your data in one central location.

Streamlined switchover

Switchover from the HM console: You can now initiate a write-leader switchover for Advanced High Availability and Distributed High Availability clusters directly from the cluster detailed view. This turns an expert-only task into a low-risk operation.

Enhanced node visibility: The cluster detailed view now lists all group nodes of Advanced High Availability and Distributed High Availability clusters, providing crucial visibility into the current topology and status before and after a switchover operation.

Migration progress visibility

Detailed progress in the HM console: HM now displays comprehensive, real-time migration progress information directly in the HM console.

Table status and row counts: You gain visibility into the status of every table being migrated, including the real-time count of rows migrated and the migration rate. This allows you to track data transfer progress without needing to check logs or external tools.

Migration error history

Detailed error messaging in the HM console: HM migrations now include a migration **Error history** tab. This provides detailed error messages, including the error source, activity status, timestamp, and full log outputs to help you with troubleshooting.

Migration assessment details for Postgres

Enhanced Postgres database details page: When an external Postgres database is registered and assessed for migration, the **Estate** view now provides expanded details, including information about extensions, foreign servers, foreign tables, procedural languages and tablespaces.

Schema compatibility summary

Schema compatibility summary in the HM console: The Oracle database details page now includes a summary that describes database compatibility as reported by the Migration Portal project.

Enhancements to the Migration Portal integration

JSON downloads: Added the ability to download a JSON-formatted version of Migration Portal schema assessment reports.

Improved reporting of dependent objects: Migration Portal now includes information on dependant objects and schemas to prevent related incompatibility issues. (for example, "relation x does not exit").

Proactive recommendations

New recommendation categories: The system now generates recommendations for two new critical categories: **Configuration** and **Security**.

Configuration tuning: Recommendations now include suggestions for optimizing Postgres system parameters, such as tuning `shared_buffers` , to help you prevent bottlenecks and improve performance on managed databases.

Proactive security checks: The tab now flags potential security issues and provides actionable guidance to address them, ensuring continuous monitoring of your database environment's integrity.

2.1.1.2.2 Hybrid Manager 2025.12.0 release notes

This innovation release includes features and enhancements.

Release Date: December 11, 2025

!!!info Release cadence update

HM now follows a dual release strategy: **LTS** and **Innovation Releases**. For details on this new model, see *Hybrid Manager dual release strategy* .

- **Platform stability & flexibility:** You can now deploy Hybrid Manager (HM) on Microsoft Azure Kubernetes Service (AKS). This support extends to AI Pipelines and enables Postgres Analytics Accelerator to query tables in Azure Data Lake Storage and Blob Storage. However, support for GenAI Builder and migration capabilities on AKS will be added in a future release, as well as support for Azure Key Vault usage with TDE-enabled databases.
- **Security and identity control improvements:** You can now configure and integrate your own Identity Providers (IdP) directly from the HM console, streamlining user access management.
- **Migration visibility and assessment:** The Postgres database assessment now includes additional configuration items that provide visibility into migration complexity. Additional HM console information allows you to better track both active and completed data migrations for Oracle and Postgres sources.

- **Support for large-scale database migrations:** Enhanced the EDB Postgres AI agent and DMS configuration to better and more efficiently support schema ingestion, database assessment, and concurrent migration of large numbers of external databases.

Features and enhancements

Hybrid Manager (HM) platform

Type	Description
Feature	Added support for HM deployments on Microsoft Azure Kubernetes Service (AKS).
Feature	Added the capability to configure your own identity provider or IdP from the HM console > **Settings** > **Identity Providers** .
Enhancement	Simplified the multi-location (multi-dc) deployment procedure. Now, multi-dc topology is mainly controlled via *primary.yaml* and *secondary.yaml* configuration files, supporting standard helm practices.
Enhancement	HM now collects and displays cluster status metrics from self-managed EDB Failover Manager clusters.
Enhancement	HM now supports configuring dedicated object storage per HM project.

Migrations

Type	Description
Feature	The HM console **Sources** and **Destinations** migration pages have been updated to display the *RW_SERVICE_HOST* environment variable setting needed by the DMS agent (packaged as *cdcagent*) to successfully connect to the HM instance. Displaying this variable enables users to retrieve this information in a self-service manner instead of needing to contact the HM administrator for it.
Feature	HM now automatically creates Migration Portal projects when Oracle database schemas are ingested into the Hybrid Manager by the EDB Postgres AI agent (packaged as *beacon-agent*). This feature eliminates the need for users to manually create a Migration Portal project in most cases.
Enhancement	The Postgres database details page **Assessment** tab has been updated to include new sections that list configuration items in the selected database that could impact the complexity of migrating to HM-managed databases. The following new sections have been added: - Extensions installed - Foreign servers - Foreign tables - Tablespaces - Installed procedural languages
Enhancement	The **Migrate > Migrations** list view has been enhanced with the addition of the **Migration Type** and **Migration Progress** columns to facilitate monitoring of migration progress for all active and completed data migrations. The **Migration Type** column identifies whether a listed migration is a **Snapshot** only or **Snapshot + Streaming** type data migration. The **Migration Progress** column identifies the number of tables selected for the migration, the number of tables that have completed the snapshot phase, and the number of rows that have been migrated. For active data migrations, values for the number of snapshotted tables and migrated rows are refreshed automatically as the migration progresses.
Enhancement	Enhanced the EDB Postgres AI agent and related services to better and more efficiently support ingestion and assessment of large numbers of external databases. Improvements were also made to the Data Migration Service (DMS) to better support large numbers of concurrent data migrations.

continues on next page

Table 10 – continued from previous page

Type	Description
Enhancement	The Data Migration Service (DMS) has been tested and certified to support migrations to HM-managed Postgres 18*cluster databases, and from or to externally-managed Postgres 18* databases. The HM Migration Portal has also been updated to support assessment and migration of Oracle databases to EDB Enterprise Postgres (Oracle Compatible) 18 databases.

AI Factory

Type	Description
Feature	Added support for HM deployments on Azure for AI pipelines. Support for GenAI Builder will be added in a future release.
Feature	AI Factory now supports Postgres 18*.
Feature	AI Pipelines now includes Visual Pipeline Designer - a web-based graphical design tool for developing, deploying and managing AI data preparation pipelines, integrated into HM.
Feature	EDB has added packages, security maintenance and support services for VectorChord, an advanced, open source vector search extension for Postgres. VectorChord packages are immediately available to EDB customers via the EDB repositories
Feature	GenAI Builder now includes Resource Transfer, a capability to enable users to transfer their applications between environments and projects.
Feature	GenAI Builder now integrates with HM projects, enabling users to privately collaborate on agentic application development projects. The previous GenAI Builder data lake, data source, structures, tools and other information are now at the default project in HM.
Feature	GenAI Builder now supports per-project AI Model configurations for assistants and other platform capabilities.
Enhancement	Added instructions for a recommended deployment path for applications built with the GenAI Builder SDK.
Enhancement	GenAI Builder now benefits from improved, user-defined Tools and Structures sandboxing for better resource isolation.
Enhancement	GenAI Builder's Threads API now supports bulk operations.
Enhancement	GenAI Builder Knowledge Bases now have better support for international character encoding sets.

Analytics Accelerator

Type	Description
Feature	Postgres Analytics Accelerator now supports Postgres 18*.
Enhancement	Postgres Analytics Accelerator now supports reading and writing Delta and Iceberg tables in Azure Data Lake Storage and Azure Blob Storage.

!!!note * Postgres refers to all distributions supported by EDB

- PostgreSQL
- EDB Enterprise Postgres - formerly, EDB Postgres Extended Server (PGE)
- EDB Enterprise Postgres (Oracle Compatible) - formerly, EDB Postgres Advanced Server (EPAS) !!!

2.1.1.2.3 Hybrid Manager 2026.1.0 release notes

This innovation release includes features and enhancements.

Release Date: January 15, 2026

!!!info Release cadence update

HM now follows a dual release strategy: **LTS** and **Innovation Releases**. For details on this new model, see *Hybrid Manager dual release strategy* .

- **Enhanced load balancer configuration**: Introduced new global configuration parameters for load balancers, providing more granular control over networking and traffic management.
- **Expanded support for Microsoft AKS**: The Data Migration Service (DMS) now fully supports Microsoft AKS, allowing you to run migration workloads when Hybrid Manager (HM) is deployed on Azure's managed Kubernetes environment.
- **Streamlined migration diagnostics**: The detailed view of a Postgres database registered in the **Estate** > **Migrations** tab now provides clearer diagnostics regarding Replica Identity. The **Migrations** tab also features a high-performance search for migration statuses to help you track progress across large environments.
- **Upgraded database observability**: The **Monitoring** tab now features improved data visualizations, including stacked charts for database sizes.
- **Comprehensive event and recommendations visibility**: Added deeper visibility into custom and model cluster events within the global **Activity Logs** section accesible through the user's menu. We also updated Query Advisor to automatically exclude internal system objects for cleaner, more relevant cluster performance recommendations.

Features and enhancements

Hybrid Manager (HM) platform

Type	Description
Feature	Hybrid Manager (HM) now exposes its API capabilities through the Model Context Protocol (MCP). This integration enables developers and AI agents to consume the Hybrid Manager API for agentic use cases. For example, you can use these capabilities alongside Langflow to streamline database agent configuration.
Feature	The HM console now includes Langflow in the Launchpad. You can use this integration to build and test AI-powered agents and workflows directly within the HM platform.
Enhancement	Added new global configuration parameters for load balancers.
Enhancement	HM cluster APIs now provide more visibility when provisioning fails, enabling self-service diagnostics and quicker time to resolution.
Enhancement	Improved the handling of fields in the **Settings** > **Identity Providers** management page.
Deprecation	The global parameter *TRANSPORTER_RW_SERVICE_DOMAIN_NAME* is renamed to *GLOBAL.DMS_DOMAIN_NAME*. Ensure that *GLOBAL.DMS_DOMAIN_NAME* is explicitly set during the upgrade process (if you followed the sequential upgrade strategy, this was already covered during the *Troubleshooting*). Additionally, ensure all Transporter components are updated to the current version of HM as stated in *Troubleshooting* .

Observability and monitoring

Type	Description
Enhancement	The HM observability functionality has been upgraded with stacked charts for database sizes, historical event markers, and the ability to sync query tables with chart selections.
Enhancement	Added API capabilities to annotate host metrics charts. You can now display activity logs as vertical dotted lines within the host historical charts in a cluster's **Monitoring** tab. This allows you to correlate specific cluster events with host performance fluctuations at exact points in time.
Enhancement	The Grand Unified Configuration (GUC) *query_advisor.exclude_schema_list* now includes predefined default values. These defaults automatically exclude internal database schemas, ensuring that Query Advisor recommendations (index and stats metrics displayed in the **Recommendations** tab of a cluster) remain focused on user-defined data and are not cluttered by internal system objects.
Enhancement	The EDB Postgres AI agent uninstalls on Debian are now cleaner and warning free.
Enhancement	Added visibility into additional events (custom events and model cluster events) in the global **Activity Log** section accesible through the user's menu.
Enhancement	Added new critical alerts to monitor pods terminated by Out Of Memory (OOM) events, backups exceeding 12 hours, and disk/WAL usage thresholds. Self-managed PGD clusters now have full feature parity for replication matrices and lag monitoring.
Enhancement	HM now provides deeper deployment insights via pod-level status conditions in the cluster APIs.

Migrations

Type	Description
Feature	The Data Migration Service (DMS) now fully supports deployments on Microsoft AKS.
Enhancement	The assessment available in a migration's *Troubleshooting* (**Estate** > **Migrations**) now provides clearer diagnostics regarding Replica Identity for Postgres sources.
Enhancement	Improved the search for migration statuses in the **Estate** > **Migrations** tab for faster results.
Enhancement	Error logs available for a project's ongoing migration in the **Migrate** > **Migrations** section now provide minute-level data for easier troubleshooting, enabling faster issue resolution.
Enhancement	Added new DMS agent environment variables that provide finer control over the schemas and tables in a source database that are available for data migrations.

2.1.1.2.4 Hybrid Manager 2026.2.0 release notes

This Innovation Release includes features and enhancements.

Release Date: February 12, 2026

!!!info Release cadence update

HM now follows a dual release strategy: **LTS** and **Innovation Releases**. For details on this new model, see *Hybrid Manager dual release strategy* .

- **Selective installation** — You can now customize your HM deployment by enabling only the specific capabilities you require. This significantly reduces resource usage and simplifies the management of your HM deployment by deploying only the features you need.
- **Integrated native chatbot** — This release introduces a native chat interface within the HM console, allowing you to answer questions and obtain guidance on operations directly through chat.
- **Three-location cluster deployments** — Enhanced architecture support now allows for deploying Postgres database clusters across three geographically separated locations, enabling zero-data-loss configurations.
- **Support for external AI models** — The AI Factory has been expanded to support external model serving, allowing you to connect and use your own remote AI models within HM.
- **Unified monitoring for self-managed clusters** — Observability is now extended to self-managed EFM and PGD clusters, allowing you to monitor a wider array of self-managed databases through the HM console.

Features and enhancements

Hybrid Manager (HM) platform

Type	Description
Feature	HM now supports selective installations. Customize your deployment by enabling only the capabilities you need (Observability & DBaaS, Migrations, Analytics, AI) at install time or later. This reduces resource usage and simplifies component management.
Feature	Added a new native chatbot interface that supports basic informational queries, direct SQL execution, and database management.
Enhancement	Added support for deploying Postgres database clusters across three geographically separated locations, enabling advanced architectures where replica clusters or data groups can be deployed to other locations. This update allows for a zero-data-loss architecture through synchronous replication. Additionally, HM now supports deploying PGD-X Witness Groups in a third location to maintain quorum and prevent split-brain scenarios during automated failovers.

Observability and monitoring

Type	Description
Feature	Users can now use event annotations on time-series charts as delimiters for selecting time periods. This allows for easier isolation and analysis of data ranges associated with specific cluster events.
Enhancement	Added support for database monitoring of self-managed EFM and PGD clusters within HM. The EDB Postgres AI agent funnels metrics, logs, and events from these external clusters into the central observability platform, providing a unified view of HA cluster health.

AI Factory

Type	Description
Feature	Added support for accessing external services for model serving, allowing you to connect and use your own remote AI models. Use your existing model infrastructure to simplify integrating multiple models into your data pipelines, GenAI Builder applications, and HM chatbot.

Migrations

Type	Description
Enhancement	A new section has been added to the Postgres database details page to display tablespace usage. This view lists the count of object types by schema for each tablespace, helping users assess the complexity of migrating externally managed Postgres databases.
Platform support	HM migration features are now fully supported in air-gapped on-premise environments. This ensures that all migration capabilities function correctly in RKE and RHOS deployments without requiring external network access.
Platform support	HM migration features are now fully supported on Azure Kubernetes Service (AKS), ensuring consistent functionality and performance for deployments in Azure environments.
Bug fix	Fixed a Migration Portal issue introduced in the HM 2025.12 release which prevented existing databases to be used as the destination for online schema migrations.
Bug fix	Fixed the Migration Portal analysis feature so it now identifies the use of the *PIVOT* operator in DDL and reports it as an incompatible feature.
Bug fix	Fixed the Migration Portal analysis feature so that it no longer inaccurately reports the *IS JSON CHECK CONSTRAINT* as an incompatible feature.

2.1.1.2.5 Hybrid Manager 2026.3.0 release notes

This Innovation Release includes features and enhancements.

Release Date: March 12, 2026

!!!info Release cadence update

HM now follows a dual release strategy: **LTS** and **Innovation Releases**. For details on this new model, see *Hybrid Manager dual release strategy* .

- **Introduction of HM automations:** This release debuts the `Automations` engine, a major step toward autonomous database management. The first available task is `Disk Auto-Scaling` , which allows the system to monitor disk usage and dynamically expand the primary storage based on project-level maintenance windows and human-in-the-loop approvals.
- **New "Summarize" SQL aggregate function in AI Factory:** You can now use the `summarize_text_aggregate()` function in standard SQL `GROUP BY` queries. This allows you to summarize text from multiple rows into a single LLM-generated response. The engine automatically handles large text volumes via a map/reduce approach to ensure you never hit model token limits.
- **Automated daily cluster health reports:** Hybrid Manager now automatically generates comprehensive health reports once a day for every cluster in your estate. These reports are retained longer than raw metrics, allowing you to track long-term health trends, spot anomalies, and share performance snapshots. You can even ask the chat agent to retrieve or compare reports for you.
- **Interactive AI insights via chatbot:** The Hybrid Manager chatbot is now more deeply integrated into the operational workflow. It can now provide automated schema design recommendations based on running workloads

and retrieve and compare daily-generated cluster health reports via simple natural language queries.

- **Enhanced migration risk assessment:** To ensure higher success rates for data migrations, this release adds granular assessments for source databases. This includes monitoring replication slot health and identifying risky table configurations (Primary Keys, TOAST, and data types) that could impact Debezium-based CDC.
- **Advanced AI pipeline capabilities:** AI Factory now supports multi-step, composable pipelines. Users can now chain complex tasks together, such as converting structured PDFs to images, extracting text via OCR, and summarizing the results via LLM—directly within a visual pipeline designer.

Features and enhancements

Hybrid Manager (HM) platform

Type	Description
Feature	Introducing *Automations*, a powerful new engine that allows users to automate routine database operations. The initial release includes *Disk auto-scaling*, the first in a series of planned automated tasks. The new framework features project-level *Maintenance Windows* to schedule tasks, granular execution controls, optional human-in-the-loop approvals and an integrated *Task Manager* to monitor all automated activities from a single, centralized interface.
Enhancement	Added the option to configure dedicated object storage per project via the CLI using the new *edbctl project create-object-store* command. When creating a new project, you can now choose between using the default shared object storage or configuring a dedicated object storage, ensuring project-level isolation for Postgres backups. Existing projects are protected and will not be affected unless explicitly overwritten using a force flag.
Enhancement	Updated the PGAA version shipped with HM to *Manual fallback for edbctl less than 1.7.0* .
Platform support	Added support for Kubernetes 1.34 and OpenShift 4.21.
Platform support	Added support for running Hybrid Manager components on ARM-based architectures, enabling deployments on modern ARM platforms (such as ARM-based Kubernetes nodes and cloud instances).
Bug fix	Updated the Azure ObjectStore authentication mechanism from Managed Identity-only to DefaultAzureCredential. This enables support for multiple authentication methods, automatically attempting various credentials until a successful connection is established.
Bug fix	Fixed an issue whereby Hybrid Manager attempted to create public load balancers when running in a private VPC, causing cluster provisioning failures. The system now correctly creates private load balancers when the environment is in a pure private network.
Bug fix	Fixed an issue in GUC (Grand Unified Configuration) validation where the system used unevaluated values instead of the final evaluated values applied by Postgres, ensuring more accurate configuration validation.
Bug fix	Fixed an issue whereby adding a replica cluster to a primary cluster failed following a major version upgrade due to database file incompatibilities.
Bug fix	Fixed an issue whereby replica cluster provisioning failed due to an error in the replication configuration enrichment logic. The reconciliation process was updated to ensure the replica lifecycle completes successfully.
Bug fix	Fixed an issue whereby the chat agent UI lost content when receiving rapid streaming of small chunks.
Change	The chat agent now supports *nvidia/llama-3.3-nemotron-super-49b-v1.5*.

AI Factory

Type	Description
Feature	Extended the AIDB pipeline framework to support multistep, composable pipelines. You can now chain preparer functions (e.g., PDF to text, HTML to text) so that the output of one step feeds into the input of the next, enabling flexible pipeline composition through the visual pipeline designer.
Feature	Added the *summarize_text_aggregate()* function to AIDB, enabling you to summarize text from multiple rows into a single LLM-generated response within a standard SQL GROUP BY query. The function supports configurable token limits, custom prompts, and automatically employs a map-reduce approach for large text volumes that exceed the model's token limit.
Feature	Added a PDF-to-image conversion pipeline step to AIDB. You can now process PDF documents — including those with complex table layouts — through a pipeline that converts pages to images and applies OCR (PaddleOCR) to extract text suitable for downstream embedding and retrieval.
Change	Starting with this version, GenAI Builder uses LangFlow as its default orchestration engine.
Enhancement	Extended the invalid model name error message — previously only available on the Internal Model Cluster page — to the external inference service configuration page.
Bug fix	Fixed an issue whereby AIDB volumes with a dash in the name couldn't be deleted due to double-quoting of identifiers.

Observability and monitoring

Type	Description
Feature	Added cluster health reports. These are automatically generated once a day for each cluster in your estate. You can examine health reports to spot anomalies or share a snapshot of your database performance with others. Since they are retained for longer than raw metrics and query statistics, they can also be used to track health over longer periods. The chat agent can also retrieve and compare reports - just ask it!
Feature	Added schema design recommendations via the chat agent. You can now ask the chat agent to give you schema design recommendations and it will assess how well-suited your schema is to your running workload and provide actionable recommendations to improve performance.
Feature	Added support for managing multiple external PGD groups across different Kubernetes clusters as a single logical cluster, enabling unified visibility and topology management in the **Estate** section.
Enhancement	The Disk Throughput chart now displays read and write throughput separately for more accurate monitoring.
Enhancement	EDB Postgres AI agent container images and Helm charts are now published to the community360 and standard repositories in addition to the enterprise repository, simplifying access for broader deployment scenarios.
Bug fix	Fixed an issue whereby the database filter was incorrectly required in the recommendation service API.
Bug fix	Fixed an issue whereby the Nodes Status total in monitoring included *initdb* and *join* pods, resulting in incorrect counts.

continues on next page

Table 22 – continued from previous page

Type	Description
Bug fix	Fixed an issue whereby the breadcrumb path for External CNP Clusters was incorrectly displayed as */Projects/<project-name>/Detail/Details* instead of the expected */Projects/<project-name>/Postgres Clusters/<cluster-name>*.
Bug fix	Fixed an issue whereby API requests to the **Estate** endpoint failed with a 431 error due to excessively large request headers when processing estates with numerous projects.

Migrations

Type	Description
Feature	Added a replication slot health monitor that tracks the state of logical replication slots on the source Postgres database, reporting health status, replay lag, and disk retention for each slot in the downloadable metrics report.
Feature	Added a dedicated metrics fetcher that runs independently of schema change detection, enabling continuous monitoring of database metrics such as WAL generation rate at regular poll intervals.
Feature	Added a per-table tuple mutation rate assessment that estimates each table's contribution to WAL generation, enabling more granular migration planning for source Postgres databases.
Feature	Added a global WAL generation rate assessment that monitors the rate of WAL accumulation on the source Postgres database, helping estimate storage capacity requirements during migration snapshots.
Feature	Added a migration assessment that checks for risky combinations of primary key, REPLICA IDENTITY, and TOAST configurations in Postgres tables, identifying scenarios where Debezium CDC might silently miss changes.
Feature	Added a migration assessment that identifies Postgres data types unsupported or risky for Debezium-based CDC, providing early warnings about potential compatibility issues before starting a migration.
Feature	Added a public API endpoint for downloading Migration Portal assessment reports in JSON format, removing the need to use the GUI.
Enhancement	Improved Data Migration Service (DMS) snapshot performance by deferring primary key application until after the snapshot phase. Composite primary keys are now collected and applied in order post-snapshot, achieving performance on par with pg_dump and pg_restore. Also added support for deduplication between snapshot and streaming phases.
Bug fix	Fixed an issue whereby dismissing a Migration Portal project creation failure notification didn't remove the failed project entry from the backend, preventing you from creating a new Migration Portal project for the same schema.
Bug fix	Fixed an issue whereby schemas weren't listed for AWS RDS PostgreSQL and Aurora PostgreSQL databases in the **Database Details** page. The EDB Postgres AI agent was attempting to connect to the protected *rdsadmin* system database, causing schema collection to fail for all databases on the instance.

2.1.1.2.6 Hybrid Manager 2026.4.1 release notes

This Innovation Release includes features and enhancements.

Release Date: April 23, 2026

!!!info Release cadence update

HM now follows a dual release strategy: **LTS** and **Innovation Releases**. For details on this new model, see *Hybrid Manager dual release strategy* .

- **Automations engine expansion:** New **Automations** tasks let you auto-apply recommended indexes, auto-increase CPU and memory, and automate minor and patch database updates — all with configurable maintenance windows and human-in-the-loop approvals.
- **Expanded AIDB coverage in Langflow:** New Langflow components expand the AIDB integration, including hybrid search for the EDB Knowledge Base component and new EDB Airman MCP Server, EDB Embedded Models, and EDB Vector Index components. You can also now deploy Langflow flows as managed workloads on HM with configurable resource constraints and full observability integration.
- **Langflow migration template:** A new AI-powered Langflow template automates the end-to-end migration process from Oracle to Postgres, including provisioning HM-managed clusters, migrating schemas, and performing snapshot plus streaming data migration.
- **Unified schema migration:** You can now migrate database schemas from Oracle or Postgres sources to Postgres destinations directly through Hybrid Manager, with support for Oracle SQL compatibility assessment and API-driven automation via AI agents.
- **Proactive query and schema change detection:** HM now automatically detects queries running for the first time on a monitored database and identifies schema changes — both within one day — helping you spot unexpected modifications and investigate potential regressions early.
- **``dbaas`` scenario separation:** Database cluster management capabilities are now controlled by a dedicated `dbaas` scenario, separated from the `core` scenario. Existing installations upgrading from 2026.3 *must* explicitly add `dbaas` to their scenarios list to retain Postgres cluster management functionality.
- **OSS Library:** Deploy community database tools — including Apache Airflow, Apache Superset, Metabase, pgAdmin4, pgBadger, and HammerDB — directly to your HM projects from the **Asset Library**, with configurable parameters, scheduling, and approval workflows.

Release details

Hybrid Manager (HM) platform

Type	Description
Feature	Introduced OSS Library, a catalog of community database tools that can be deployed directly to your HM projects. The initial catalog includes Apache Airflow, Apache Superset, Metabase, pgAdmin 4, pgBadger, and HammerDB. Apps are deployed from the **Asset Library > Apps** section, with configurable parameters, optional deployment scheduling, and human-in-the-loop approval workflows. Deployed apps can be monitored, upgraded, and deleted from **Estate > Apps** or your project's **Apps** view.
Feature	Added the ability to automatically apply recommended database indexes via the **Automations** engine. Filter suggestions by databases, schemas or tables and choose to apply them with explicit human approval and/or during your **Maintenance Window** .
Feature	You can now configure HM-managed clusters in the **Automations** engine to automatically scale up CPU and/or memory resources when usage thresholds are exceeded.
Feature	You can now automate minor and patch database updates via the **Automations** engine. Set your preferred schedule and maintenance rules to ensure your HM-managed clusters stay current with zero manual effort.

continues on next page

Table 24 – continued from previous page

Type	Description
Enhancement	Added advanced compression to HM backup capabilities through integration with Barman Cloud compression algorithms. WAL compression is now enabled by default to improve backup performance, reduce storage consumption, and prevent WAL storage full incidents. Defaults to Snappy compression for basebackup and zstd compression for WAL files.
Enhancement	Added optional read-only connection strings for HA, AHA, and DHA clusters. Enable a dedicated read-only endpoint to route read operations to standby replicas (HA) or nodes that aren't currently designated as the write leader (AHA and DHA), reducing load on the primary. Configure in *Cluster setup* (HA) or per data group in *EDB Postgres Distributed (PGD) cluster with 2 data groups and 1 witness group not healthy* (AHA and DHA). Once enabled, the connection string is available on the cluster's *Troubleshooting* .
Enhancement	Added support for node selectors and tolerations in the cluster provisioning and patch API. You can now specify additional pod scheduling constraints to control where PostgreSQL pods are deployed within Kubernetes clusters, including the ability to target specific nodes and tolerate node taints for advanced workload isolation scenarios.
Enhancement	Added support for Custom Loadbalancer annotations in EDB Postgres AI agent (*beacon-agent*) provisioning, allowing you to configure load balancer annotations for database deployments across supported platforms. This functionality can be enabled or disabled at the location capabilities level.
Enhancement	Added the ability to provision clusters via API with custom load balancer annotations (metal-lb or any custom annotation), when this capability is supported by the location via EDB Postgres AI agent (*beacon-agent*) config. This is an API-only feature for this release; UI and CLI support will be added in the upcoming LTS.
Feature	Added the ability to automatically assign HM user roles based on IdP group attributes. You can now map LDAP user attributes or group names to roles in a declarative way, eliminating the need to manually assign roles for users in your LDAP directory.
Enhancement	The Chat Agent now becomes available automatically after you create an inference service. Previously, you had to refresh your browser to make the Chat Agent available.
Enhancement	Enhanced the database notification system with expanded event sources. The notification system can now notify you of events logged by the HM system, including agentic automated tasks generated by the platform. Additionally, all resource types in the audit log table are now clickable, extending beyond the previously available task-only functionality.
Enhancement	Enhanced the image library **Clusters Using This Image** view to only show clusters you have permission to access, based on your user role and project permissions.
Enhancement	Added support for GitLab container registry as a registry provider for image discovery.
Enhancement	You can now create HM-managed clusters through the Chat Agent, including High Availability, Advanced High Availability, and Distributed High Availability distributions.
Enhancement	Enhanced the activity log to include API key identifiers in audit log entries, improving identity metadata tracking at the gateway for better security monitoring and correlation.
Enhancement	The Chat Agent's functionality and security have been enhanced. The Chat Agent now uses a selection list interaction method which increases the certainty of interactive information, and uses a separate password transmission path which ensures passwords aren't uploaded to the large language model.

continues on next page

Table 24 – continued from previous page

Type	Description
Enhancement	Added support for deploying published flow bundles via the *upm-flow-proxy* service. The HM platform can now fetch flows directly from the internal flow proxy service as an alternative to cloud storage (S3, Azure, GCP), enabling streamlined deployment of published flow configurations.
Enhancement	Introduced the consolidated system database (*upm-system-db*). The operator preflight check now validates that the required database credential secrets (*db-upm*, *db-dex*, *db-langflow*, *db-lakekeeper*, *db-beacon-app*, *db-transporter*, *db-superuser*) exist before installation or upgrade. These secrets are auto-generated by *edbctl setup create-install-secrets* when targeting IR 2026.4 and later versions.
Enhancement	Database cluster management capabilities — including provisioning, backups, storage locations, and cluster templates — are now controlled by a dedicated *dbaas* scenario, separated from the *core* scenario. New installations include *dbaas* by default when no scenarios are specified. Existing installations upgrading from 2026.3 must explicitly add *dbaas* to their scenarios list after upgrading the operator to retain Postgres cluster management functionality. Without *dbaas*, cluster management APIs return 404 and the **Clusters** section is hidden in the HM console.
Change	Changed the default value of the *edb_stat_monitor.edbsm_normalized_query* GUC from *false* to *true*. This change ensures that values used in predicates aren't saved in HM by default, providing better data privacy for customers who don't want sensitive query data to leave their databases.
Deprecation	Removed the Apache Spark component bundled with Hybrid Manager. Note that connecting to external Spark clusters remains supported.
Deprecation	Deprecated the *beaconServer.additionalTrustDomains* field in the HM CRD. This field is no longer used by the *upm-beacon* Helm chart and has been replaced by dedicated Helm values configuration.
Security	Enhanced network security policies for the Langflow component to restrict access to AI model server APIs. Langflow can now access only read-only discovery APIs for model server clusters and inference services, while blocking dangerous mutating operations such as create, update, and delete.
Security	Fixed a security issue whereby SPIRE server audit logging was disabled, which prevented logging of security-relevant events such as SVID issuance, entry modifications, and federation operations. The SPIRE server now has audit logging enabled to meet SOC2 and PCI DSS compliance requirements.
Security	Enhanced project-scoped authorization for published flows. The *upm-flow-proxy* now enforces project-level permissions, ensuring users can only access and modify published flows within their authorized projects. When creating flows, users must have access to all specified projects. When updating existing flows, users can only add or remove projects they have access to, while projects outside their scope remain unchanged.
Bug fix	Fixed an issue whereby ClusterWrapper could become stuck in the soft deleting phase due to a race condition.
Bug fix	Fixed an issue whereby component defaults weren't applied in the *upm-beaco-ff-base* component, which could cause the *accm-server* to crash loop due to missing configuration values for *usage_generator_interval*, *AES_KEY_ROTATION_INTERVAL*, and *SESSION_DURATION_SECONDS* parameters.
Bug fix	Fixed an issue whereby read-only connection tests failed on Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS), and Red Hat OpenShift (RHOS) platforms due to unprotected null reference errors when no PostgreSQL status store existed.

continues on next page

Table 24 – continued from previous page

Type	Description
Bug fix	Fixed an issue whereby *beaconServer.logLevel* defaulted to *info*, overriding user-configured values. The setting now correctly respects the user-configured value.
Bug fix	Fixed an issue whereby a pod was crash looping on RHOS environments due to DNS resolution failures when the SPIRE agent attempted to connect to the kubelet via node hostname, which was not resolvable in cluster DNS.
Bug fix	Fixed an issue whereby the EDB Postgres AI agent (*beacon-agent*) produced confusing error messages when re-enrolling servers with newer agent versions against older HM servers. The agent now handles version compatibility more gracefully and provides clearer error reporting when server APIs are unavailable.
Bug fix	Fixed a bug where AI Pipelines didn't correctly detect deleted source records on volume sources. This bug would cause orphaned destination records to be left behind, and is now fixed.
Bug fix	The *upm-beacon-server* enters a crashloop when the spire component is broken or unavailable. Because spire is required for all HM deployments — which are configured to support secondary locations — a broken spire component prevents the deployment of Postgres clusters.
Bug fix	Fixed an issue whereby the credentials creation process in the *edbctl setup create-install-secrets* command had non-deterministic ordering, causing namespaces and secrets to be created in different orders across runs.

AI Factory

Type	Description
Feature	Expanded AIDB coverage in Langflow with hybrid search support added to the EDB Knowledge Base component, combining text matching with vector similarity search. Three new components were added: EDB Airman MCP Server; EDB Embedded Models, for CPU-based text and image inference using AIDB's embedded models; and EDB Vector Index, for creating vector indexes on EDB knowledge bases.
Feature	Added a Langflow template for AI-powered, end-to-end database migrations. The template automates the complete migration process from Oracle to Postgres, including provisioning HM-managed clusters, migrating schemas, and performing a snapshot plus streaming data migration. You can customize the migration flows to integrate your own solutions as needed. This initial version supports single database migrations with fully compatible source schemas and requires GPT-4o or better reasoning-class models.
Feature	The AI Pipelines component of the AIDB extension for Postgres now supports multiple compatible pipelines writing into a single shared knowledge base, enabling hybrid knowledge bases from multiple sources and formats — including multi-model (image and text) — each with independent processing steps. The vector table includes a new *pipeline_id* column to track source pipelines per embedding, and retrieval functions return the originating pipeline name.
Feature	Added support for deploying Langflow flows to Kubernetes environments managed by Hybrid Manager. You can now deploy your Langflow flows with configurable quality-of-service constraints including memory, CPU allocation, autoscaling, and restart policies. Deployed flows include full observability and alerting integration with Hybrid Manager and can be managed through the HM interface with deploy, undeploy, and status monitoring capabilities.

continues on next page

Table 25 – continued from previous page

Type	Description
Feature	Added hybrid search capabilities to the AI Pipelines component of the AIDB extension, combining full-text search, vector/semantic search, and predicate filtering. Previously, only vector/semantic search was supported. You can now perform advanced retrieval queries from AI Pipelines Knowledge Bases using custom SQL queries. These queries may include arbitrary relational filtering or sub-queries and can also perform full text search using a bm25 index. AI Pipelines includes convenience functions to help build those queries, and documentation contains examples for common use cases.
Enhancement	Added an error log and dead letters queue to improve AI pipeline error handling. You can now track which files or records are being processed successfully and which files encounter errors during pipeline execution.
Enhancement	Added non-blocking error handling to AI pipelines. Pipelines now continue processing when errors affect only individual records (such as malformed PDF files or invalid data), while still halting on pipeline-level errors like missing models or configuration issues. You can query and manage pipeline errors using the new *aidb.get_error_logs()* and *aidb.clear_error_logs()* functions.
Enhancement	Added support for creating AI data pipelines for on-premises databases and external CloudNativePG clusters. You can now create control plane pipelines for unmanaged estates, expanding pipeline capabilities beyond HM-managed database instances.
Enhancement	Added AIDB pipeline support for external Postgres clusters managed by the Hybrid Manager (HM) platform. You can now build AIDB pipelines on these external clusters, including clusters running on secondary locations of HM and clusters running outside of the HM platform, with access to model and inference services from the primary location.
Enhancement	The AI Pipeline Designer now retrieves available models dynamically instead of using static default settings. This change enables model selection based on specific project, resource, and database combinations while maintaining the same user experience.
Enhancement	Added support for configuring the PDF to Image pipeline step directly in the visual AI Pipeline Designer. You can now set conversion options such as format, DPI, page ranges, and annotation rendering through the designer interface.
Enhancement	Added support for EDB AIDB pipeline extension v6.1.0 in Hybrid Manager, bringing the latest pipeline capabilities and enhancements to your AIDB workflows.
Enhancement	You can build your AI workflow using LangFlow, and publish your flow as assets on the Hybrid Manager platform. You can then run these flows as standalone jobs on selected locations within your projects.
Enhancement	Added support for Hugging Face as a new model provider. You can now configure AI models using Hugging Face Model Name, Object Storage path, or both when adding new models to the platform.
Enhancement	Enhanced error handling in the AI model creation form to display detailed error messages, improving user feedback when model creation requirements aren't met.
Security	Fixed a remote code execution vulnerability in the Langflow component whereby an attacker could execute arbitrary commands through the data parameter in the *build_public_tmp* endpoint.
Bug fix	Fixed an issue whereby the AI model service couldn't be listed from the Visual pipeline builder on RKE, GKE, and AKS platforms due to network policy restrictions preventing Postgres pods from accessing the model server API.
Bug fix	Fixed an issue whereby the AIDB Pipeline Designer provisioning page displayed clusters from all projects instead of filtering clusters to show only those from the specific project.

Observability and monitoring

Type	Description
Feature	Added detection of queries that haven't been executed on a monitored database before. HM now identifies new queries within one day, helping you verify they aren't causing issues or easily find the root cause if they are. The feature includes mechanisms to disable detection and suppress excessive notifications, such as when first adding a database where all queries would initially appear as new.
Feature	Added schema change detection that automatically identifies changes in monitored database schemas within one day. You can now spot unexpected schema modifications and compare database states before and after changes to identify potential regressions.
Feature	Added a new comparison tool in the HM console that lets you compare database metrics and query statistics between two user-defined time periods. You can now easily identify performance differences with side-by-side or overlaid chart views, apply filters and zoom controls, and see highlighted differences to understand the impact of changes over time.
Enhancement	Extended the monitoring dashboard annotation capability to support user-inserted custom events. You can now manually mark interesting platform and database activities alongside system-generated events, making it easier to correlate changes with performance trends.
Enhancement	Extended observability support for external PGD clusters to include CNPG-GC clusters. Building on the monitoring capabilities introduced in previous releases, you can now monitor metrics from both self-managed PGD and CNPG-GC clusters through HM observability features.
Enhancement	Added a new *EnableEDBUsageReporting* configuration option for the beacon server to provide more explicit control over usage reporting. The beacon server now includes retry logic that attempts to reconnect to IT links every 24 hours when they're unreachable, improving operation reliability in various environments including air-gapped setups.
Enhancement	When a cluster spec is updated — whether by a user change, autoscaling, or an agentic automation — an annotation is automatically generated and displayed in the observability view.
Bug fix	Fixed an issue whereby the EDB Postgres AI agent (*beacon-agent*) failed to start due to an invalid Postgres DSN error when the port was specified as 0, which is outside the valid port range.
Bug fix	Fixed an issue whereby metric time and values showed discrepancies between cluster-level and node-level views in the HM console. The backend now returns consistent data across both views and aligns timestamps correctly for node-level metrics.
Bug fix	Enhanced the robustness and performance of the internal metrics statistics database, optimizing its storage and memory usage.
Bug fix	Fixed an issue whereby the WAL storage percent metric displayed incorrect values above 100% in monitoring charts. The metric now accurately aggregates WAL usage by primary nodes only, providing proper percentage calculations.

HM console

Type	Description
Feature	Added AI-powered query generation to the Query Editor. You can now select **Generate with AI** to create SQL queries using natural language prompts. The feature generates valid SQL targeting data across your estate that you're authorized to query, making it easier to explore and analyze your data without writing complex SQL syntax.
Enhancement	Added the ability to view database health reports directly in the UI without downloading the PDF. You can now preview report contents inline before deciding to download.
Enhancement	Added a configuration interface in the HM console for HM authentication policies in the **Settings** section. Organization owners can now configure authentication policy settings, including idle timeout for web login sessions.
Enhancement	Added an **Invalidate User Sessions** action to the Identity Providers management interface. You can now invalidate all active user sessions for a specific identity provider through the HM console, providing administrators with better control over user access management.
Enhancement	Added filters in the role assignment interface to display only manually assigned roles when showing current user permissions. Roles assigned via IdP group mappings are now excluded from this view to reduce noise.
Bug fix	Fixed an issue whereby machine users couldn't be filtered correctly in the HM console. The filtering now uses the proper identity provider parameter instead of a general query parameter.

Analytics

Type	Description
Enhancement	Updated the PGAA version shipped with HM to 1.8 .
Enhancement	Enhanced Lakekeeper warehouse creation to support object storage environments that don't support AWS Security Token Service (STS). The system now automatically retries warehouse creation with STS disabled if the initial attempt with STS enabled fails on S3-compatible storage, improving compatibility with providers like Vultr's Ceph Object Gateway.

Migrations

Type	Description
Feature	Added the ability to enter and display metadata about applications that use each database and schema. You can now associate applications with databases and schemas in many-to-many relationships, providing better context for database migration planning and prioritization. Application metadata can be entered manually through the Hybrid Manager interface or programmatically via the Hybrid Manager APIs.
Feature	Added the ability for the EDB Postgres AI agent (*beacon-agent*) to automatically configure Oracle and Postgres database connections using established sources of connection information, including Oracle *TNSNAMES.ORA* files and CSV files. The option to auto-discover RDS Postgres databases and configure the agent connections to them is also available.

continues on next page

Table 29 – continued from previous page

Type	Description
Feature	Added unified schema migration capabilities to migrate database schemas from Oracle or Postgres sources to Postgres destinations directly through Hybrid Manager, eliminating the need for external tools. The service supports both HM-managed and externally managed Postgres destinations and integrates with data migration workflows for schema-only, data-only, or combined migrations, with API access for AI agent automation. When migrating from Oracle to EDB Postgres Advanced Server, compatible schemas are retrieved from the Migration Portal.
Feature	Added the ability to assess Oracle application SQL statements for compatibility with EDB Postgres Advanced Server. You can now assess SQL statements extracted from Oracle trace files and system views, as well as those provided directly through user interfaces or programmatic APIs. This capability includes support for MCP tools and introduces a new application information object within the Hybrid Manager migration architecture to serve as a container for SQL statements and assessment reports.
Enhancement	Added schema parsing, storage, and migration capabilities to the Migration Portal. When a Postgres database is registered with the Hybrid Manager for migration purposes, Postgres database schemas are now automatically ingested, parsed, and stored in the Migration Portal backend database. Those schemas are then available for migration with new API behaviors.
Enhancement	Extended the Migration Portal assessment reports API to support bulk downloads. The new endpoint accepts multiple database resource IDs and returns assessment reports for all specified resources in a single JSONL file, making it easier to download reports at scale.
Enhancement	The migration agent now honors upstream cancellation signals when extracting Oracle schemas. Previously, when the EDB Postgres AI agent (*beacon-agent*) received a termination signal during Oracle schema extraction, the *sqlplus* process would continue running until completion, wasting resources on work that couldn't be delivered. The agent now properly terminates the *sqlplus* subprocess when receiving cancellation signals, enabling cleaner shutdown behavior.
Enhancement	Added support for replicating PostgreSQL *DOMAIN* types based on *DATE*, *TIME*, *TIMESTAMP*, *INTERVAL*, *UUID*, *JSON*, *XML*, and *POINT* types. *DOMAIN* types are user-defined types that wrap existing PostgreSQL types with additional constraints. Both snapshot and CDC replication modes now correctly handle these *DOMAIN* types, whether the target uses the same *DOMAIN* or just the underlying base type.
Bug fix	Fixed an issue whereby Oracle schema collection reported incorrect table counts when the EDB Postgres AI agent (*beacon-agent*) connected as a non-privileged user.
Bug fix	Fixed an issue whereby a null string exception occurred in the *GetMigrationDatabaseAssessmentData* function when DDL extraction for PostgreSQL databases failed during assessment, causing API requests to return a 500 error.
Bug fix	Fixed an issue whereby database features weren't being collected during Oracle assessments in certain environments.
Bug fix	Fixed an issue whereby the gRPC UI encountered errors when schemas had dependencies.
Bug fix	Fixed an issue whereby nested button elements in the **Migration Databases** empty state caused invalid DOM structure warnings.
Bug fix	Fixed an issue whereby the Create OLTP Migration form didn't show error messages when source table loading failed, leaving users without feedback on what went wrong.
Bug fix	Fixed an issue whereby CDC replication failed when PostgreSQL table or column names contained multi-byte UTF-8 characters such as Japanese, Chinese, or Korean characters.
Bug fix	Fixed an issue which caused the migration of wide tables to fail with no recovery.

continues on next page

Table 29 – continued from previous page

Type	Description
Bug fix	Fixed an issue whereby the reader would throw an error when uploading objects to S3 after bumping the Netty dependency. The error occurred due to connection pool timeout issues with the async S3 client, which has been resolved by switching to a synchronous S3 client.

2.1.1.2.7 Hybrid Manager 2026.5.0 release notes

This Innovation Release includes features and enhancements.

Release Date: May 19, 2026

!!!info Release cadence update

HM now follows a dual release strategy: **LTS** and **Innovation Releases**. For details on this new model, see *Hybrid Manager (HM) dual release strategy* .

- **Expanded Postgres-to-Postgres schema migration:** The Data Migration Service (DMS) now handles Postgres-specific database objects during schema migration, including custom domain types and other Postgres-native type definitions, enabling complete end-to-end schema migration between Postgres databases.
- **Enhanced agentic migration:** The Migration Agent Langflow template now uses an improved architecture leveraging HM's unified schema migration feature, with added support for Llama 3 Nemotron 49B and Nano 4B models.
- **HM chatbot enhancements:** Added Anthropic Claude as a chatbot provider and bundled HM and community PostgreSQL documentation for citation-backed answers.
- **Role mapping improvements:** You can now assign multiple roles in a single role mapping rule, streamlining identity provider group-to-role assignments.

Release details

AI Factory

Type	Description
Enhancement	Added support for AIDB 7 in the AI Pipeline Designer, including improved knowledge base validation and fixes for issues affecting pipelines with existing knowledge base tables and EDB Postgres Distributed (PGD) clusters. For full details, see the AIDB 7 release notes .
Enhancement	Added support for registering external inference services with insecure connections. You can now enable the **Allow insecure connection** option when registering external model services that don't have signed certificates, such as those in test environments. The system requires explicit user acknowledgment of the security risk before allowing the registration.
Enhancement	Enhanced error handling when publishing AI flows with invalid parameters. The UI now displays detailed error messages from the server instead of a generic **Failed to publish flow** message, helping you identify and fix parameter naming issues.
Change	Removed the AI Pipelines **Default Settings** page from the **Sovereign AI** section. The default settings feature was incompatible with the new instance-specific models list approach.

continues on next page

Table 30 – continued from previous page

Type	Description
Bug fix	Resolved a startup failure in the EDBAirman component where uvx dependency resolution errors prevented the Model Context Protocol (MCP) server from initializing. The runtime now includes all required dependencies.

HM console

Type	Description
Enhancement	Added the *beacon.enterprisedb.com/exclude-storage-from-dbaas=true* label to exclude specific *StorageClass* or *VolumeSnapshotClass* resources from the HM console. In shared Kubernetes environments, you can apply this label to storage classes that aren't suitable for Postgres workloads—such as GPU- or HM-only classes—to prevent them from appearing in the **Create Cluster** dropdowns. The label takes effect on the next HM polling cycle and doesn't require a restart.
Enhancement	Enhanced the role mapping rules interface to support selecting multiple roles for a single rule. You can now assign multiple roles (such as both **Project Viewer** and **Catalog Data Writer**) to the same group and identity provider combination in a single rule, rather than creating separate rules for each role.
Enhancement	Added Anthropic Claude as a supported provider for the HM chatbot. The chatbot now renders Mermaid diagrams inline in responses. Bundled HM and community PostgreSQL (16/17/18) documentation with the chatbot so it can answer product questions with citations.

Observability and monitoring

Type	Description
Bug fix	Thanos Ruler used the default Kubernetes service account. It now uses a dedicated service account.

Analytics

Type	Description
Enhancement	Updated the PGAA version shipped with HM to 1.9 .

Migrations

Type	Description
Enhancement	Added concurrent schema finalization for *schema_and_data* migrations. After the snapshot stage completes, you can select **Finalize Schema Concurrently** on the migration detail page to apply indexes, foreign keys, and other schema objects in the background using non-blocking data definition language (DDL) variants. When you later complete the migration, only lightweight validation steps remain, significantly reducing cutover time.

continues on next page

Table 34 – continued from previous page

Type	Description
Enhancement	Extended Postgres-to-Postgres schema migration in the DMS to handle Postgres-specific database objects—including custom domain types and other Postgres-native type definitions—during schema migration. The DMS now ensures it applies these schema objects to the target before other schema objects that depend on them, enabling complete, end-to-end schema migration between Postgres databases.
Enhancement	Enhanced the Simple Migration Agent Langflow template with an improved architecture that leverages the unified schema migration feature, replacing the legacy Migration Portal approach. The updated workflow includes streamlined agent components, automatic password generation, and database setup, with redundant components removed.
Enhancement	Added support for Llama 3 Nemotron 49B and Llama 3 Nemotron Nano 4B models in the Simple Migration Agent Langflow template, providing additional AI model options for migration workflows.
Enhancement	Enhanced database password security in the Simple Migration Agent Langflow template by moving cluster creation operations outside of large language model (LLM) processing and leveraging global secrets management to prevent credential exposure.
Enhancement	Optimized the migration creation form to skip unnecessary table loading and validation when you set the migration scope to schema-only. The form now only fetches table data for data-inclusive migration scopes, improving performance and user experience.
Bug fix	Fixed an issue in the Simple Migration Agent Langflow template whereby the list of tools in an MCP Tools component wouldn't update after selecting an MCP server until you refreshed the page.
Bug fix	Added support for migrating EDB Postgres Advanced Server (EPAS) *PACKAGE* and *PACKAGE BODY* schema objects from EPAS source databases to EPAS destination databases. The DMS now properly handles package and package body statements during schema migration, including correct dependency ordering to ensure the DMS applies package bodies after the tables they reference.
Bug fix	Added support for migrating EPAS non-public *SYNONYM* schema objects from EPAS source projects. The DMS now properly handles non-public synonym statements during schema migration. The DMS extracts and uploads *PUBLIC SYNONYM* objects into the Migration Portal project but doesn't yet support migrating them to the destination database.
Bug fix	Fixed an issue whereby the schema migration capability couldn't migrate database objects for schemas that didn't contain any tables. The migration process now handles all schemas regardless of whether they include tables.
Bug fix	Fixed an issue whereby schema-only migrations dropped *PRIMARY KEY* constraints and their dependent foreign key and named *CHECK* constraints when you disabled data migration for specific schemas but enabled it for others. The transporter now correctly applies *PRIMARY KEY* constraints during the **Finalize Schema** phase for tables excluded from data migration.
Bug fix	Fixed an issue whereby the DMS validation incorrectly required all schemas in the source database to have a healthy Migration Portal project, even when the migration didn't include those schemas. The validation now only checks schemas that the migration's table list references.
Bug fix	Fixed an issue whereby the transformed SQL file had missing line endings when the source SQL file contained Windows-style CR LF line endings.

2.1.1.2.8 Hybrid Manager 2026.5.1 release notes

This Innovation Release includes enhancements and bug fixes.

Release Date: June 2, 2026

!!!info Release cadence update

HM now follows a dual release strategy: **LTS** and **Innovation Releases**. For details on this new model, see *Hybrid Manager (HM) dual release strategy* .

Hybrid Manager (HM) platform

Type	Description
Enhancement	Added support for custom Kubernetes Service annotations on Kafka load balancers, including the Transporter's Kafka services. When using custom-only load balancer mode, you can now specify annotations such as *metallb.universe.tf/address-pool* for MetalLB address pool selection or *networking.gke.io/internal-load-balancer-subnet* for GCP Shared VPC internal load balancer subnet configuration, as well as node-pool classes, enabling BYO-network and on-premises deployment scenarios.
Platform support	Certified HM 2026.5.1 for deployment using the Red Hat OpenShift (RHOS) operator.
Bug fix	Fixed an issue whereby WAL archiving to Azure Blob Storage returned HTTP 403 errors on AKS, EKS, and GKE clusters. The system now correctly handles backup credentials for Azure Storage.

AI Factory

Type	Description
Enhancement	Restricted Langflow access to users with the AI Model Manager role. Users without this role can no longer access Langflow, providing tighter control over AI workflow authoring.
Enhancement	Added automatic retry for transient LLM errors in AI workflows, improving reliability when inference providers return temporary failures.
Enhancement	You can now search AI models and services by tag name in the HM console.
Change	Moved the Langflow flow cache to ephemeral storage. The cache no longer persists across pod restarts and is rebuilt automatically on startup.
Bug fix	Fixed an issue whereby soft-deleted AI model services reappeared in the HM console.

HM console

Type	Description
Enhancement	Extended HM chatbot agent support to all external inference providers, including Gemini 2.x models.
Enhancement	You can now filter clusters by cluster or group ID in the HM console.
Enhancement	Added a searchable model-service picker in the HM console.
Bug fix	Fixed an issue whereby marketplace package synchronization failed on installations using an in-cluster registry. Packages now sync correctly when using an in-cluster registry.

continues on next page

Table 37 – continued from previous page

Type	Description
Bug fix	Fixed an issue whereby cluster creation with an unsupported or empty storage class would hang indefinitely. The operation now fails immediately with an informative error.

Observability and monitoring

Type	Description
Change	Updated the default alert evaluation interval to 2 minutes.
Bug fix	Fixed an incorrect calculation of the top database size metric in the observability dashboard.

Migrations

Type	Description
Enhancement	Hardened the Migration Portal API with a deny-by-default security model. Unregistered endpoints now return HTTP 403, and the API now performs tenant-ownership checks to prevent unauthorized cross-tenant access.
Bug fix	Fixed an issue whereby schemas with special characters or mixed-case names failed to migrate correctly through the Migration Portal.
Bug fix	Fixed an issue whereby Oracle-to-EDB Postgres Advanced Server (EPAS) schema migrations failed to complete correctly.
Bug fix	Fixed an issue whereby phantom or duplicate schemas appeared in Migration Portal projects.
Bug fix	Fixed an issue whereby Migration Portal projects whose first schema was empty failed to complete.

CHAPTER 3

EDB Postgres AI Hybrid Manager

EDB Postgres AI Hybrid Manager (HM) is the control plane for Postgres, sovereign AI, and analytics workloads across Kubernetes-based infrastructures. It provides a consistent and unified way to provision, operate, and monitor PostgreSQL clusters—whether you're deploying on-prem, in the cloud, or across hybrid environments.

Hybrid Manager enables end-to-end data workflows from database deployment to AI analytics, all governed within your infrastructure.

3.1 Why Hybrid Manager?

Modern data platforms require more than just database provisioning—they need to support:

- Kubernetes-native scalability and orchestration
- Resilient, high-availability deployments
- AI and GenAI integration
- Postgres-native analytics with open formats
- Governance, observability, and access control

Hybrid Manager was created to meet these demands. It extends Kubernetes capabilities with built-in Postgres expertise, making it easier to operate stateful workloads without sacrificing the flexibility of containerized infrastructure.

3.2 Key capabilities

Area	Description
Database provisioning	Deploy highly available PostgreSQL or Distributed Postgres clusters using the console, templates, or declarative manifests.
AI integration	Run *Troubleshooting* , serve models with KServe, and build AI knowledge bases governed by your infrastructure.

continues on next page

Table 1 – continued from previous page

Area	Description
Analytics	Offload cold data to Delta Lake or Iceberg , query with Postgres syntax, and enable hybrid transactional/analytical processing.
Monitoring and observability	Use the integrated dashboards that leverage Prometheus, Grafana, and Loki to obtain deep visibility into query health, resource usage, and system alerts.
Security and access management	Integrate with identity providers, define RBAC for projects and clusters, and manage secure access to your databases and services.
Database migration	Migrate schemas and data from external databases (Oracle, self-managed Postgres) to HM-managed clusters. Leverage integrated services for migration and schema assessment, data transfer (snapshot), and continuous replication (streaming) to achieve minimal downtime migrations.

3.3 Supported environments

Hybrid Manager runs on Kubernetes and supports these common managed Kubernetes cloud services:

- Microsoft Azure Kubernetes Service (AKS)
- Amazon Elastic Kubernetes Service (EKS)
- Red Hat OpenShift Service on AWS (ROSA)
- Google Kubernetes Engine (GKE)

HM also supports on-premises Kubernetes platforms:

- Red Hat OpenShift (RHOS)
- SUSE Rancher RKE2

It supports both cloud-native and on-premises architectures, allowing you to build sovereign, compliant, and flexible data infrastructures.

3.4 Integrated solutions

Hybrid Manager unlocks core functionality across the EDB Postgres AI ecosystem. It consists of:

- *Getting Started with AI Factory* – Build and run GenAI workloads using private data.
- *Analytics in Hybrid Manager* – Enable open-format analytics with Iceberg, Delta Lake, and tiered offloading.
- *Simple Migration Agent* – Accelerate adoption of Postgres with in-platform migration tools.
- *Monitoring execution* – Monitor operations using built-in observability stack with Kubernetes and database insights.
- *Missing notifications for task and cluster lifecycle events* – Create templates, perform upgrades, configure HA settings, and restore data.

3.5 Learn more

- *Architecture overview*
- *Supported Postgres distributions*
- *Supported database types*

3.6 Get started

- *Install Hybrid Manager*
- Use Hybrid Manager

3.6.1 Hybrid Manager architecture

Hybrid Manager runs on a Kubernetes cluster deployed on a cloud service provider's infrastructure, such as AKS on Azure , EKS on AWS or GKE on GCP , or on-premises infrastructure. "On-premises" in this context includes deployments on physical servers (bare metal), virtual machines, or private clouds hosted within an organization's infrastructure.

The three main logical groupings within the Kubernetes cluster are:

1. **Kubernetes control plane**: The Kubernetes control plane is comprised of the core Kubernetes components and implemented by two or three Kubernetes control nodes. Examples of core components are `kube-apiserver` , `etcd` , `kube-scheduler` , `kube-controller-manager` , and, if on a cloud setup, `cloud-controller-manager` .
2. **Control compute grouping**: The control compute grouping of HM is a logical grouping of 70+ pods running on 2 or 3 Kubernetes worker nodes. It implements HM's core management components, such as Grafana, Loki, Thanos, Prometheus for observability, Cert Manager for certification management, Istio for networking, a trust manager for securing software releases, and many others.

The 70+ pods for the components are distributed across the worker nodes according to how the `kube-scheduler` places them.

If needed, you can add more worker nodes to support more database-as-a-service resources.

3. **Data compute grouping**: The data compute grouping of HM is a logical grouping of Postgres clusters organized by namespaces. It runs on a number of pods distributed across at least 3 Kubernetes worker nodes as the `kube-scheduler` places them.

You can increase the number of worker nodes from three as more databases are added using HM. However, to do this automatically may require either adding the worker nodes manually or configuring auto-scaling alongside your infrastructure.

Both the control compute grouping's worker nodes and the data compute grouping's worker nodes are backed up nightly to either snapshots (on-premises using a SAN or using EBS with EKS) or S3-compatible object store using Barman for the cloud .

3.6.1.1 Scalability

To support scalability, HM integrates with Kubernetes auto-scaling capabilities. This means that on platforms like AWS, worker nodes in the data compute grouping can be automatically added to the cluster as needed using AWS's managed service Auto Scaling groups . However, you can disable HM's Kubernetes auto-scaling capability to control costs. In addition, it's important to know that enabling auto-scaling functionality for on-premises deployments requires additional infrastructure configuration. That's because there's no out-of-the-box auto-scaling feature for on-premises deployments as there is for AWS's Auto Scaling groups.

You can monitor the distribution and resource usage of pods and worker nodes implementing the database clusters using the *included Grafana dashboards* or by using Kubernetes `kubectl commands <Tracing clusters back to their Kubernetes resources>` .

3.6.1.2 Backup architecture

HM provides robust backup options to ensure data integrity and recoverability. Each HM worker node in the data compute grouping performs nightly backups by default, stored using Barman (for the cloud) or using snapshot backups.

Snapshot backups are available for faster recovery times but require a compatible CSI driver in the storage layer (for both cloud-based snapshots and local disk snapshots). Given the constraints of logical volume manager (LVM) setups, snapshots are optimal for local recovery but may face limitations when used in multi-site or global recovery scenarios setups due to the lack of shared access across sites.

For more robust, multi-availability zone recovery, Barman cloud backups can be stored in object stores like AWS S3, which natively supports multi-zone replication. Alternatively, local object stores might require additional configuration to ensure backups are replicated to remote sites. Stretch clusters mitigate these limitations by enabling data replication and redundancy across geographically dispersed locations.

3.6.1.3 High availability and resilience

HM leverages Kubernetes' proven technologies to ensure high availability through automated failover mechanisms. These mechanisms function within stretch clusters, across availability zones, or between zones in the same region. The architecture supports faster backups, via snapshot backups, as well as recovery for large datasets, though multi-site recovery capabilities depend on the underlying storage infrastructure. For customers with advanced storage configurations, such as storage area networks (SANs) spanning multiple racks or data centers, HM can leverage the infrastructure to enhance resilience and ensure rapid failover. However, these configurations may not be common among all users.

3.6.2 Supported database types

Selecting the right Postgres database cluster type when creating new clusters in Hybrid Manager (HM) is crucial for optimizing your database environment. EDB offers a wide range of solutions, each tailored for different needs, from basic single-node deployments to highly resilient distributed solutions.

3.6.2.1 Distributed high availability

Distributed high-availability (DHA) clusters are powered by EDB Postgres Distributed (PGD) and leverage multi-master logical replication. This architecture enables true active-active solutions across multiple regions, providing the highest levels of availability and resilience.

Select this option if your:

- Organization requires ultra-high availability and near-zero downtime.
- Applications demand distributed deployments with low latency access for users.
- Business has mission-critical applications where continuous operation and data consistency across multiple active nodes are paramount.
- Scenarios require uninterrupted online database maintenance and in-place major version upgrades without impacting availability.

3.6.2.2 Advanced high availability

Advanced high availability (AHA) focuses on delivering robust HA solutions beyond basic primary/standby setups, aiming for higher uptime guarantees and more sophisticated recovery mechanisms. This can encompass various enhanced HA configurations, often using EDB's specialized tools and expertise to provide superior resilience. Unlike DHA clusters, AHA clusters use a single region and are designed to provide high availability within that region.

Select this option if your:

- Teams prefer a simpler setup than DHA that still leverages advanced EDB Postgres Distributed features and allows for future migration to full DHA.

- Business requires a balance between complexity and high uptime, offering stronger resilience without the full multi-master distributed architecture.
- Enterprise needs enhanced high availability that surpasses standard primary/standby configurations.
- Organization seeks to minimize downtime for planned maintenance and unplanned outages with more flexible deployment options.

3.6.2.3 High availability

High-availability (HA) primary/standby configurations typically involve a primary database server and one or more standby (replica) servers. The standby servers continuously receive data changes from the primary, ready to take over if the primary fails. This setup uses physical streaming replication.

Select this option if your:

- Applications require data durability and continuous service with minimal downtime, and strict active-active multi-master isn't a requirement.
- Use cases have read-heavy workloads that can benefit from offloading queries to hot standby replicas via a dedicated *read-only connection endpoint* .

3.6.2.4 Single node

A single-node deployment consists of a single PostgreSQL database instance running without any standby or replica servers. This is the simplest configuration and is suitable for non-critical workloads or development environments.

Select this option for:

- Development and testing environments where high availability isn't a concern.
- Non-critical applications that can tolerate downtime for maintenance or upgrading, or in case of failure.
- Use cases that prioritize simplicity and minimal infrastructure cost above all else.
- Small-scale applications with low traffic and data volume.

3.6.2.5 Cluster type architectures

- *Network latency in Distributed High Availability (DHA) clusters*
- *Advanced high availability*
- *High availability and isolation for Langflow flows*
- *Single node*

3.6.2.5.1 Distributed high availability

Distributed high-availability clusters are powered by EDB Postgres Distributed . They use multi-master logical replication to deliver more advanced cluster management than a physical replication-based system. Distributed high-availability clusters allow you to deploy a cluster across multiple regions or a single region. For use cases where high availability across regions is a requirement, a cluster deployment with distributed high availability can provide up to two data groups in (2) distinct regions, along with a witness group in a third separate region.

This configuration provides a true active-active solution as each data group is configured to accept write operations.

Distributed high-availability clusters support both EDB Postgres Advanced Server and EDB Postgres Extended Server database distributions.

Distributed high-availability clusters contain one or two data groups. Your data groups can contain either three data nodes or two data nodes and one witness node. At any given time, one of these data nodes in each group is the leader

and accepts writes, while the rest are referred to as shadow nodes . We recommend that you don't use two data nodes and one witness node in production unless you use asynchronous commit scopes .

PGD Connection Manager can route all application traffic to the write leader node. Each group has its own write leader, reducing distributed data conflicts. PGD leverages a distributed consensus model to determine availability of the data nodes in the cluster. On failure or unavailability of the leader, PGD elects a new write leader and redirects application traffic. Together with the core capabilities of EDB Postgres Distributed, this mechanism of routing application traffic to the leader node enables fast failover and switchover.

The witness node/witness group doesn't host data but exists for management purposes. It supports operations that require a consensus, for example, in case of a region failure.

!!!note Operations against a distributed high-availability cluster leverage the EDB Postgres Distributed set-leader feature, which provides subsecond interruptions during planned lifecycle operations.

Single data location

A configuration with a single data location has one data group and either:

- Two data nodes with one lead, one shadow, and a witness node, each in separate availability zones

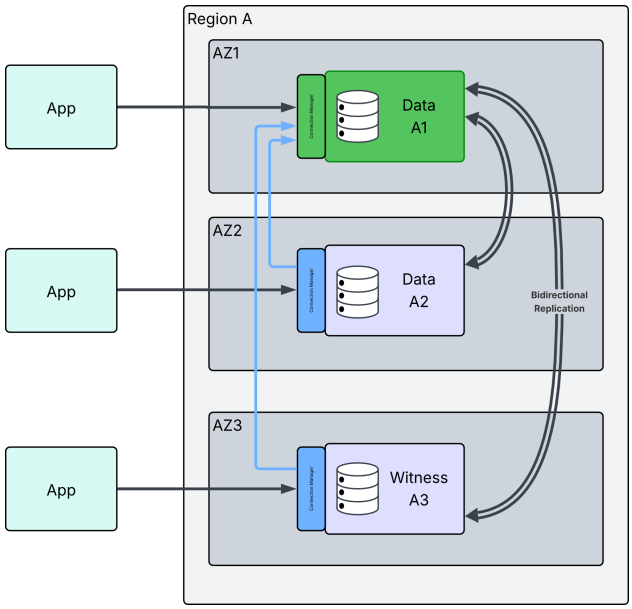


Fig. 1: region(2 data + 1 witness)

- Three data nodes with one lead and two shadow nodes, each in separate availability zones

Multiple groups with a witness node

A configuration with multiple data locations has two data groups that contain either three data nodes in each group:

- A data node and two shadow nodes in one group
- The same configuration in another group
- A witness node in a third group

Groups will currently be in the same region (the region the cluster resides in), but this may change in the future.

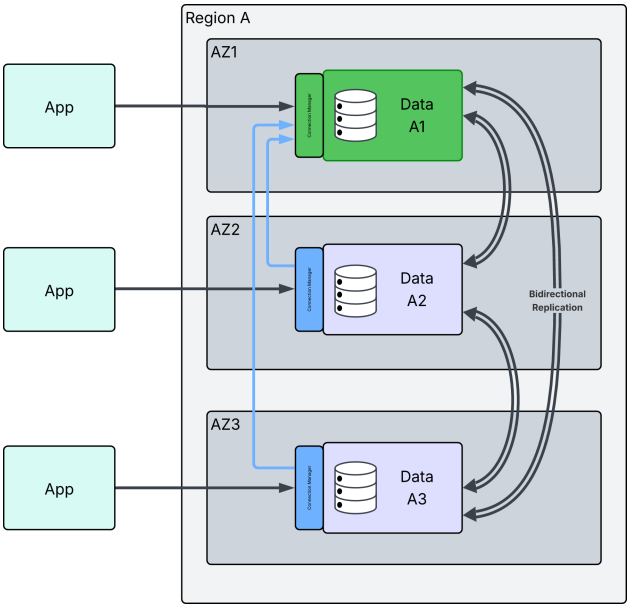


Fig. 2: region(3 data)

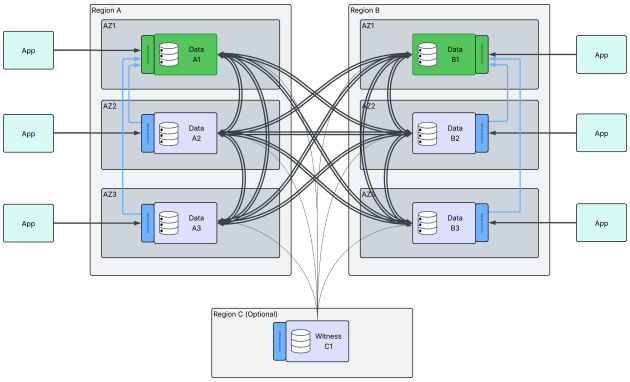


Fig. 3: groups(2 x 3 data + 1 witness)

Advisory locks

Advisory locks aren't replicated between Postgres nodes, so advisory locks taken on a shadow node don't conflict with advisory locks taken on the leader. We recommend that applications that rely on advisory locking avoid using read-only workloads for those transactions.

DHA clusters also support read-only connections per data group, allowing read-only operations to be routed to nodes that aren't currently designated as the write leader, improving performance and workload isolation. To enable this, see *EDB Postgres Distributed (PGD) cluster with 2 data groups and 1 witness group not healthy* .

For more information

For instructions on creating, retrieving information from, and managing a distributed high-availability cluster using the HM CLI, see *Using the HM CLI* .

3.6.2.5.2 Advanced high availability

Advanced high-availability (AHA) clusters are powered by EDB Postgres Distributed and are based on the Essential edition of PGD. Unlike *Tracing back distributed high-availability clusters* , advanced high-availability clusters use a single region and are designed to provide high availability in that region.

These clusters are designed for users who want a simpler setup and fewer configuration options than distributed high-availability clusters. At the same time, these users benefit from the advanced features of EDB Postgres Distributed and the ability to easily migrate to distributed high-availability clusters in the future.

AHA clusters are ideal for applications that require robust failover capabilities without the complexity of multi-region deployments. The underlying architecture of advanced high-availability clusters is similar to that of distributed high-availability clusters but requires the users to connect via PGD Connection Manager (embedded in every node) to ensure connections go to the current write leader node. This eliminates the need to manage conflicts across the cluster, making application development easier.

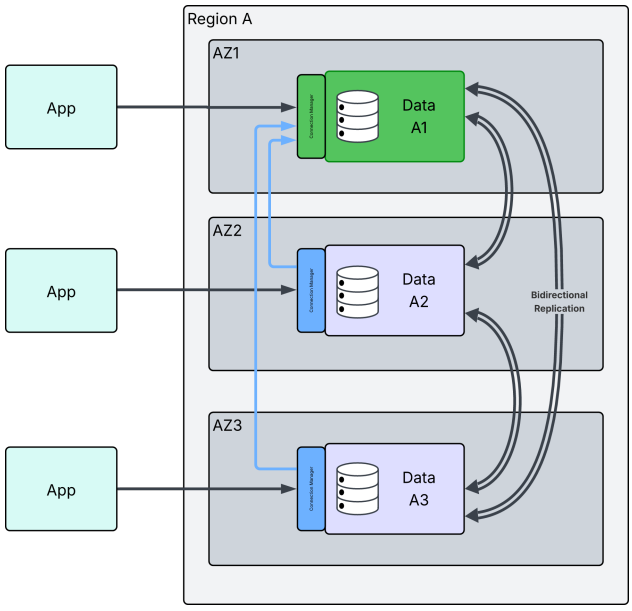


Fig. 4: Advanced high-availability cluster architecture

The diagram shows the applications connecting to the PGD cluster through Connection Manager ports. Connection Manager is responsible for routing the read and write transactions to the appropriate nodes in the group. The write leader is responsible for handling all write transactions and is shown at the top in AZ1 in green.

The other nodes in the group are read-only nodes that replicate data from the write leader. Applications connecting to the read-only nodes' Connection Manager read/write ports will have their queries and changes routed to the write leader. All the time, the nodes are communicating with each other, replicating data to ensure they're in sync.

AHA clusters support both EDB Postgres Advanced Server and EDB Postgres Extended Server database distributions. They don't support community PostgreSQL distributions.

AHA clusters also support read-only connections per data group, allowing read-only operations to be routed to nodes that aren't currently designated as the write leader, improving performance and workload isolation. To enable this, see *EDB Postgres Distributed (PGD) cluster with 2 data groups and 1 witness group not healthy* .

3.6.2.5.3 High availability

Primary/standby high availability minimizes downtime in cases of failures. Primary/standby high-availability clusters—one *primary* and one or two *standby replicas*—are configured automatically. Standby replicas stay up to date through physical streaming replication.

If read-only workloads are enabled, standby replicas serve the read-only workloads. In a two-node cluster, the single standby replica serves read-only workloads. In a three-node cluster, both standby replicas serve read-only workloads. The connections are made to the two standby replicas randomly and on a per-connection basis. You can enable this functionality by configuring a *Read-only connection information* for your cluster, which provides a dedicated read-only endpoint alongside the standard read/write connection string.

In cloud regions with availability zones, clusters are provisioned across zones to provide fault tolerance in the face of a data center failure.

In case of temporary or permanent unavailability of the primary, a standby replica becomes the primary.

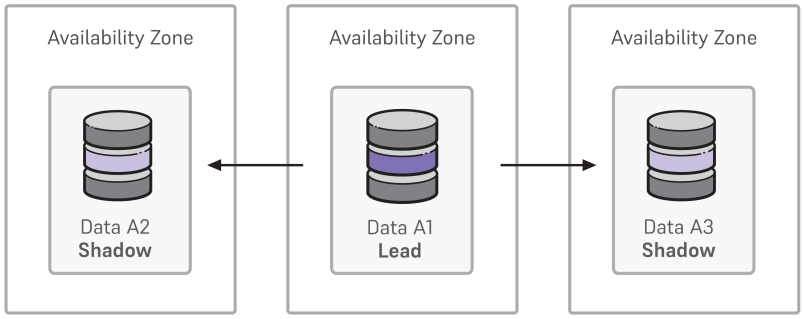


Fig. 5: Primary/standby high availability

Incoming client connections are always routed to the current primary. In case of failure of the primary, a standby replica is promoted to primary, and new connections are routed to the new primary. When the old primary recovers, it rejoins the cluster as a standby replica.

Standby replicas

By default, replication is synchronous to one standby replica and asynchronous to the other. That is, one standby replica must confirm that a transaction record was written to disk before the client receives acknowledgment of a successful commit.

In a cluster with one primary and one replica (a two-node primary/standby high-availability cluster), you run the risk of the cluster being unavailable for writes because it doesn't have the same level of reliability as a three-node cluster. Hybrid Manager disables synchronous replication during maintenance operations of a two-node cluster to ensure write

availability. You can also change from the default synchronous replication for a two-node cluster to asynchronous replication on a per-session or per-transaction basis.

In PostgreSQL terms, `synchronous_commit` is set to `on` , and `synchronous_standby_names` is set to `ANY 1 (replica-1, replica-2)` . You can modify this behavior on a per-transaction, per-session, per-user, or per-database basis using `SET` or `ALTER` commands.

Since Hybrid Manager replicates to only one node synchronously, some standby replicas in three-node clusters might experience replication lag. Also, if you override the Hybrid Manager synchronous replication configuration, then the standby replicas are inconsistent.

3.6.2.5.4 Single-node cluster

For nonproduction use cases where high availability isn't a primary concern, a cluster deployment without high availability enabled provides one primary with no standby replicas for failover or read-only workloads.

In case of unrecoverable failure of the primary, a restore from a backup is required.

Fig. 6: Single node cluster

3.6.3 Known issues

These are the currently known issues and limitations identified in the Hybrid Manager (HM) Innovation Releases. Workarounds can sometimes help you mitigate the impact of these issues. These issues are actively tracked and will be resolved in a future release.

3.6.3.1 Multi-DC

3.6.3.1.1 Multi-DC configuration loss after upgrade

!!!tip

Resolved in HM versions 2025.12 and later.

Workaround: After every HM upgrade or component reversion, run the multi-DC setup scripts again to reapply the necessary configurations.

3.6.3.1.2 Location dropdown list is empty for multi-DC setups

!!!tip

Resolved in HM versions 2025.12 and later.

Workaround: This advanced workaround requires cluster administrator privileges to limit the amount of image set information being returned by the API. It involves modifying the image discovery tag rules in the `upm-image-library` and `upm-beacon` ConfigMaps, followed by restarting the related pods.

Workaround details

The workaround modifies the regular expressions (tag rules) used by the image library and EDB Postgres AI agent components to temporarily limit the number of image tags being indexed. This reduces the API response size, allowing the locations to load.

1. Find the `upm-image-library` ConfigMap:

```
kubectl get configmaps -n upm-image-library | grep upm-image-library
# Example Output: upm-image-library-ttkt29fmf7 1 5d3h
```

2. Edit the ConfigMap and modify the `tags` rule under each image discovery rule (`edb-postgres-advanced` , `edb-postgres-extended` , `postgresql`). Replace the existing regex with the limiting regex:

```
# Snippet of the YAML you will edit in the ConfigMap
"imageDiscovery": {
  "rules": {
    "(^|.*/)edb-postgres-advanced$": {
      "readme": "EDB postgres advanced server",
      "tags": [
        "^(?P<major>\d+)\.(?P<minor>\d+)-2509(?P<day>\d{2})(?P<hour>\d{2})(?P<minute>
↪\d{2})     (?:-(?P<pgdFlavor>pgdx|pgds))?(?:-(?P<suffix>full))?$"
      ]
    },
    # ... repeat for edb-postgres-extended and postgresql ...
  }
}
```

> **Note**
>
> If you're running a multi-DC setup, perform this step on the primary HM cluster.

3. Restart the Image Library pod:

```
kubectl rollout restart deployment upm-image-library -n upm-image-library
```

1. Get the`upm-beacon` ConfigMap to modify the EDB Postgres AI agent configuration:

```
kubectl get configmaps -n upm-beacon beacon-agent-k8s-config
```

5. Edit the ConfigMap (`beacon-agent-k8s-config`) and modify the `tag_regex` rule under each `postgres_repositories` entry (`edb-postgres-advanced` , `edb-postgres-extended` , `postgresql`).

```
# Snippet of the YAML you will edit in the ConfigMap
postgres_repositories:
  - name: edb-postgres-advanced
```

(continues on next page)

(continued from previous page)

```
    description: EDB postgres advanced server
    tag_regex: "^(?P<major>\d+)\.(?P<minor>\d+)-2509(?P<day>\d{2})(?P<hour>\d{2})(?
↪ P<minute>\d{2})(?:-(?P<pgdFlavor>pgdx|pgds))?(?:-(?P<suffix>full))?$"
  # ... repeat for edb-postgres-extended and postgresql ...
```

1. Restart the EDB Postgres AI agent pod:

```
kubectl rollout restart deployment -n upm-beacon upm-beacon-agent-k8s
```

After completing these steps, the reduced image data size allows the location API call to succeed and the locations to appear correctly in the HM console.

3.6.3.1.3 Dedicated object storage for project isolation

!!!tip

Resolved in HM versions 2025.12 and later.

Workaround: Create dedicated object storage for new projects and enable project isolation.

Workaround details: If you need to isolate project resources, you can configure *Dedicated object storage for project isolation* for each project.

3.6.3.2 Core platform and resources

3.6.3.2.1 Cross-stream upgrade issues for OpenShift users

!!!tip Resolved in HM versions 1.4.0 and later.

Workaround: There is no workaround available for this issue. We are currently decoupling the Operator versions from the HM application versions to resolve this.

3.6.3.2.2 `upm-beacon-agent` memory limits are insufficient in complex environments

!!!tip Resolved in HM versions 1.3.1 and later.

Workaround: Manually patch the Kubernetes deployment to increase the memory resource limits for the `upm-beacon-agent` pod.

3.6.3.2.3 IdP configuration fails after upgrade to version 2026.2

Description: In some cases, upgrading from version 2026.1 to 2026.2 can cause your Identity Provider (IdP) configuration to display the **Expiring Soon** status in the HM console. This is caused by a race condition during the migration process. While this issue doesn't occur in every environment, it prevents users from successfully authenticating via their IdP provider if triggered.

Workaround: To resolve this state and allow the configuration to restore correctly, a cluster administrator must manually update the internal application database.

Execute the following SQL statement on the internal `app-db` database **before** attempting to re-setup the IdP integration in the HM console:

```
UPDATE upm_api_admin.idp SET deleted_at = NOW() WHERE deleted_at IS NULL;
```

3.6.3.2.4 `accm-server` fails to start due to missing Langflow secret

!!!tip

Resolved in HM versions 2026.3 and later.

Workaround: To allow the `accm-server` to start in environments where AI features aren't installed, you must manually create a placeholder secret in the `upm-beaco-ff-base` namespace.

You can create an empty secret using the following kubectl command:

```
kubectl create secret generic langflow-secret -n upm-beaco-ff-base --from-
↪literal=LANGFLOW_NOT_INSTALLED=true
```

Once the secret exists, the `accm-server` pod will be able to mount the volume and transition to a **Running** state.

3.6.3.2.5 No `kapp-controller` support in Hybrid Manager app to pull images

Description: The `kapp-controller` doesn't support Managed Identity/IAM Roles for pulling images.

Workaround: Users must manually configure the `PackageRepository` using a `SecretRef` (Kubernetes Secret) for registry authentication, as cloud-native managed identities are currently not supported.

3.6.3.2.6 Beacon server enters a crashloop when spire is broken

!!!tip Resolved in HM versions 2026.5 and later.

Description: The `upm-beacon-server` enters a crashloop when the spire component is broken or unavailable. Because spire is required for all HM deployments — which are configured to support secondary locations — a broken spire component prevents the deployment of Postgres clusters.

Workaround: As an administrator with `kubectl` access, edit the beacon server configmap to disable spire integration. Run:

```
kubectl edit cm -n upm-beacon beacon-server-config
```

Then set the following two spiffe-related config items:

```
general:
  spiffe:
    enabled: false
  spiffeid_controller:
    disabled: true
```

Revert these settings before connecting a secondary location.

3.6.3.2.7 SPIRE agent fails to start on RKE2 with SELinux enabled

Description: On RKE2 clusters running on RHEL (or other Linux distributions with SELinux in enforcing mode), the SPIRE agent pod enters a `CrashLoopBackOff` state with the error `create UDS listener: listen unix /tmp/spire-agent/public/spire-agent.sock: bind: permission denied`. This occurs because SELinux blocks the container process from creating a Unix domain socket on the `HostPath` volume `/run/spire/agent-sockets`. Environments without SELinux (such as Ubuntu-based RKE2 clusters) aren't affected.

Workaround: Disable SELinux on all RKE2 cluster nodes. For this, set `SELINUX=disabled` in `/etc/selinux/config` and reboot the nodes. After reboot, verify that `getenforce` shows `Disabled`. The SPIRE agent pods will recover automatically, though pods already in `CrashLoopBackOff` state may take several minutes due to the backoff timer.

3.6.3.2.8 Image library discovery fails when duplicate operand images share the same tags and SHA

!!!tip Resolved in HM versions 2026.4 and later.

Description: If two sets of EDB official PostgreSQL operand images are uploaded with the same tags and SHA digest but different repository names, the image library discovery process fails and no images are available for cluster provisioning.

Workaround: Delete one set of the duplicated operand images. The image library will discover the remaining set successfully.

3.6.3.3 Database cluster engine

3.6.3.3.1 Failure to create 3-node PGD cluster when `max_connections` is non-default

Description: Creating a 3-data-node PGD cluster fails if the configuration parameter `max_connections` is set to a non-default value during initial cluster provisioning.

Workaround: Create the PGD 3-data-node cluster using the default `max_connections` value. Update the value after the cluster is successfully provisioned.

3.6.3.3.2 AHA witness node resources are over-provisioned

Description: For advanced high-availability (AHA) clusters with witness nodes, the witness node incorrectly inherits the CPU, memory, and disk configuration of the larger data nodes, leading to unnecessary resource over-provisioning.

Workaround: Manually update the pgdgroup YAML configuration to specify and configure the minimal resources needed by the witness node.

3.6.3.3.3 HA clusters use `verify-ca` instead of `verify-full` for streaming replication certificate authentication

Description: Replica clusters use the less strict `verify-ca` setting for streaming replication authentication instead of the recommended, most secure `verify-full` . This is currently necessary because the underlying CloudNativePG (CNP) clusters don't support IP subject alternative names (IP SANs), which are required for `verify-full` in certain environments (like GKE load balancers).

Workaround: None. A fix depends on the underlying CNP component supporting IP SANs.

3.6.3.3.4 Second node is too slow to join large HA clusters

!!!tip

Resolved in HM versions 2025.12 and later.

Workaround: Avoid the best practice of loading data into a single node and then scaling to HA. Instead, load data directly into an HA cluster from the start. There's no workaround for restoring a large cluster into an HA configuration.

3.6.3.3.5 EDB Postgres Distributed (PGD) cluster with 2 data groups and 1 witness group not healthy

!!!tip Resolved in HM versions 1.4.0 and later.

3.6.3.3.6 AKS deployments limited to one PGD data group with private access per region

!!!tip Resolved in HM versions 2026.5 and later.

Description: Azure enforces a hard limit of eight private link services per Standard Load Balancer (`PrivateLinkServicesPerLoadBalancerLimitReached`). Because each PGD data group with private access requires five load balancer services (one per node, plus the group and proxy services), an AKS-based HM deployment can only host one PGD data group with private access per region. Enabling a read-only connection adds one additional load balancer service per data group, further reducing available quota.

A full two-data-group and witness cluster with private access consumes 12 private link service slots across the standard load balancer limit of 8, which prevents multi-node PGD bootstrap on AKS in a single location.

Workaround: Deploy data groups across multiple locations to distribute load balancer quota. For example, deploy one data group and the witness in one location, and the second data group in a separate location. Don't create more than one PGD data group with private access in the same AKS region within a single HM instance.

3.6.3.3.7 Network access type defaults to `Private` when not explicitly selected on RKE

!!!tip Resolved in HM versions 2026.5 and later.

Description: On RKE environments, two related network access type issues exist:

- When a location has both **Public** and **Private** access enabled, the network access type field doesn't have a default selection. If you don't explicitly choose an option, the backend defaults to `Private` .
- When a location has **NodePort** enabled, the backend correctly configures NodePort for the cluster, but the HM console displays the access type as `Private` instead.

Workaround: Always explicitly select the intended network access type when creating a cluster on RKE. A fix is being tracked for a future release.

3.6.3.3.8 CNPG operator enters an infinite reconcile loop when bootstrap `Job` creation is interrupted

Description: If a cluster reconciliation is interrupted after the PVC is created but before the bootstrap `Job` is created — for example, due to an optimistic locking conflict — subsequent reconciles enter an infinite loop. The operator waits for the PVC to reach a `Ready` status, but this can never occur without the `Job` , causing the loop to continue indefinitely.

Workaround: Delete the affected PVC to allow the operator to recreate it and break the loop. Because this issue only affects newly created PVCs, the PV won't contain any data yet and it's safe to delete.

```
kubectl delete pvc <pvc-name> -n <cluster-namespace>
```

3.6.3.4 Backup and recovery

3.6.3.4.1 Replica cluster creation fails when using volume snapshot recovery across regions

!!!tip

Resolved in HM versions 2025.12 and later.

Workaround: Manually trigger a Barman backup from the primary cluster first. Then use that Barman backup (instead of the volume snapshot) to provision the cross-region replica cluster.

3.6.3.4.2 Volume snapshot restoration is limited to the same region

Description: While HM automatically handles cross-region data synchronization for replica clusters using the internal backup mechanism, manual restoration of a cluster from a volume snapshot doesn't support cross-region operations. If you attempt to restore a cluster to a location in a different region using a volume snapshot, the operation will fail because snapshots are geographically restricted to their source region.

Workaround: To restore data to a location in a different region, use a Barman backup instead of a volume snapshot. Barman backups are accessible across regions.

3.6.3.4.3 WAL archiving is slow due to default parallel configuration

Description: The default setting for `wal.maxParallel` is too restrictive, which slows down WAL archiving during heavy data loads. This can cause a backlog of ready-to-archive WAL files, potentially leading to disk-full conditions. This parameter isn't yet configurable on the HM console.

Workaround: Manually edit the `objectstores.barmancloud.cnpg.io` Kubernetes resource for the specific backup object store and increase the `wal.maxParallel` value (for example, to 20) to accelerate archiving.

3.6.3.4.4 `transporter-db` disaster recovery (DR) process may fail due to WAL gaps

!!!tip Resolved in HM versions 1.4.0 and later.

Workaround: To ensure a successful restore, select an older backup to restore that has at least one archived WAL file immediately following it. This makes the needed transactional logs available for the recovery process.

3.6.3.4.5 Full cluster recovery fails with `pg_ctl: server does not shut down` error

Description: Under race conditions, restoring into a cluster with HA architecture that is created and managed by HM may fail. The error logs on the full-recovery pods will contain entries similar to the following:

```
pg_ctl: server does not shut down
pg_ctl: waiting for server to shut down.....................................................
↪.............. failed
controller: Error while deactivating instance err=error stopping PostgreSQL instance:␣
↪exit status 1
```

Workaround: Restore into a single-node cluster first. After the restore completes, expand the cluster to HA architecture.

3.6.3.5 Agent Factory and model management

3.6.3.5.1 Failure to deploy `nim-nvidia-nvclip` model with profile cache

Description: Model creation for the `nim-nvidia-nvclip` model fails in the Agent Factory when the profile cache is used during the deployment process.

Workaround: An administrator must manually download the necessary model profile from the NVIDIA registry to a local machine. They must then upload the profile files directly to HM's object storage path. Then they deploy the model by patching the Kubernetes InferenceService YAML with a specific environment variable to force it to use the pre-cached files instead of attempting a failed network download.

Workaround details

1. Log in to the NVIDIA Container Registry (nvcr.io) using your NGC API key:

```
docker login nvcr.io -u $oauthtoken -p $NGC_API_KEY
```

2. Pull the Docker image to your local machine:

```
docker pull nvcr.io/nim/nvidia/nvclip:latest
```

3. Prepare a local directory for the downloaded profiles:

```
mkdir -p ./model-cache
chmod -R a+w ./model-cache
```

4. Select the profile for your target GPU.

For example, A100 GPU profile: `9367a7048d21c405768203724f863e116d9aeb71d4847fca004930b9b9584bb6`

5. Run the container to download the profile. The container is run in CPU-only mode (`NIM\_CPU\_ONLY=1`) to prevent GPU-specific initialization issues on the download machine.

```
export NIM_MANIFEST_
↪PROFILE=9367a7048d21c405768203724f863e116d9aeb71d4847fca004930b9b9584bb6
export NIM_CPU_ONLY=1

docker run -v ./model-cache:/opt/nim/.cache -u $(id -u) -e NGC_API_KEY -e NIM_CPU_
↪ONLY -e NIM_MANIFEST_PROFILE --rm nvcr.io/nim/nvidia/nvclip:latest
```

This container doesn't exit. You must manually stop the run (Ctrl+C) after you see the line `Health method called` in the logs, which confirms the profile download is complete.

6. Upload the profiles from your local machine to the object storage bucket used by your HM deployment:

```
gcloud storage cp -r ./model-cache gs://uat-gke-edb-object-storage/model-cache/nim-
↪nvidia-nvclip
```

!!!note

Adjust the `gs://` path to match your deployment's configured object storage location.

7. Create the model `nim-nvidia-nvclip` using the HM console, specifying the **Model Profiles Path** field as the previous location (for example, `/model-cache/nim-nvidia-nvclip`). The deployment will initially fail or become stuck.

8. Export the InferenceService YAML from the HM Kubernetes cluster.

9. Add the necessary environment variable, `NIM_IGNORE_MODEL_DOWNLOAD_FAIL` , to the env section of the `spec.predictor.model` block in the exported YAML. This flag tells the NIM container to use the locally available cache (the files you uploaded) and ignore the network download failure.

```
# --- Snippet of the modified InferenceService YAML ---
spec:
  predictor:
    minReplicas: 1
    model:
      modelFormat:
        name: nim-nvidia-nvclip
      name: ""
      env:
      - name: NIM_IGNORE_MODEL_DOWNLOAD_FAIL  # <-- ADD THIS LINE
        value: "1"                            # <-- ADD THIS LINE
      resources:
        # ... resource requests/limits ...
```

(continues on next page)

(continued from previous page)

```
      runtime: nim-nvidia-nvclip
      storageUri: gs://uat-gke-edb-object-storage/model-cache/nim-nvidia-nvclip
# ------------------------------------------------
```

10. Apply the modified YAML using kubectl to force the deployment to use the pre-downloaded profiles:

```
kubectl apply -f <modified-inference-service-file.yaml> -n <model-cluster-namespace>
```

The pods now start successfully, using the model profiles you manually provided using object storage.

3.6.3.5.2 AI Model cluster deployment stalls if object storage path for model profiles is empty

!!!tip

Resolved in HM versions 2025.11 and later.

Workaround: Ensure that the object storage path specified in the **Model Profiles Path** field contains a correct, valid profile before initiating the model cluster deployment.

3.6.3.5.3 Model configuration settings reset after Innovation Release upgrade

!!!tip HM now uses Langflow instead of Griptape, so this issue is no longer applicable.

Description: After upgrading HM from the `2025.11` to the `2025.12` Innovation Release, the existing model configuration settings are unintentionally reset to empty. This results in a `model_not_found` error, preventing access to AI services and causing issues like knowledge bases (KBs) failing to display in the HM console.

Workaround: In the HM console, manually reenter or set your required model configurations to restore functionality.

3.6.3.5.4 Error page appears when editing knowledge base credentials

!!!tip HM now uses Langflow instead of Griptape, so this issue is no longer applicable.

Description: When editing a knowledge base (KB) that was created from a pipeline in the HM console, skipping the username field and immediately navigating to the password field triggers a client-side JavaScript error (Cannot read properties of null), which results in an **Unexpected Application Error!** page.

Workaround: To prevent the error page from appearing, fill in the **Username** field immediately after selecting **Edit KB** and before attempting to enter the password.

3.6.3.5.5 Missing aidb_users role on PGD witness node prevents AI functionality

Description: The `aidb_users` role (necessary for Agent Factory functionality) and its related extension aren't being successfully replicated to the witness node of a PGD cluster during cluster initialization. This issue is specific to PGD clusters that use a single witness node (as opposed to a witness group) and results in the Agent Factory encountering errors due to the missing required role.

Workaround: To manually install the necessary role and allow the Agent Factory to function, execute the following SQL commands directly on the PGD witness node:

```
begin;
set local bdr.ddl_replication = off;
SET LOCAL bdr.commit_scope = local;
create user aidb_users;
commit;
```

3.6.3.5.6 GenAI Builder structures and tools execution failure

!!!tip

Resolved in HM versions 2025.12 and later.

```
[Errno -2] (Name or service not known)
```

Workaround: When creating a structure in GenAI Builder, set the following environment variables in the **Create Structure** panel:

- **``GT_CLOUD_BASE_URL``** : Must be set to the full project path: `https://<PORTAL URL>/apps/genai-builder/projects/<PROJECT ID>`
- **``REQUESTS_CA_BUNDLE``** : Must be set to the following certificate path: `/etc/ssl/certs/hcp.crt`

Screenshot of variables

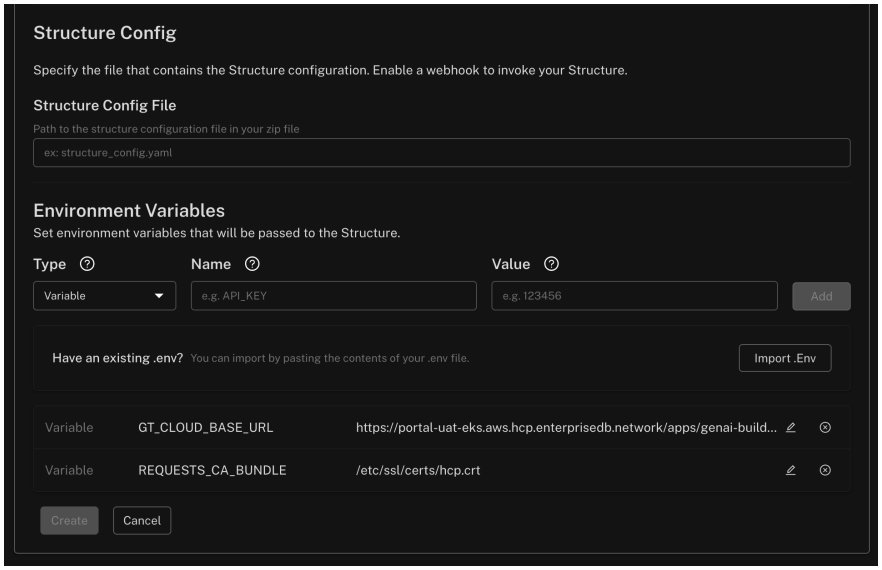


Fig. 7: Screenshot of environment variables to be set in the UI

3.6.3.5.7 Pipeline Designer-created knowledge bases not visible to GenAI Builder

Description: In HM 2025.12, knowledge bases (KBs) created with Pipeline Designer (PD) are assigned to the `visual_pipeline_user` role. This is done to enforce strict isolation, ensuring HM users can't access SQL-created KBs (and vice versa) by default. However, this isolation prevents GenAI Builder from querying these KBs out of the box.

Workaround: Explicitly share PD-created KBs with your specific PostgreSQL user account to make them queryable in GenAI Builder.

Workaround details

Example scenario

Alice connects to HM as `alice@acme.org` and to PostgreSQL as the database user `alice` . She must share the PD KBs with the `alice` database user to enable GenAI Builder agents to query them using her credentials.

1. Prerequisite: Ensure user existence

Ensure the target PostgreSQL user (`alice` in this example) exists and is assigned the `aidb_users` role. If the user doesn't exist, execute the following (assuming EDB documentation was followed for AIDB installation):

```
CREATE USER alice WITH PASSWORD ********;
GRANT CONNECT ON DATABASE <some_db> TO alice;
GRANT CREATE ON SCHEMA <some_schema> TO alice;
GRANT aidb_users TO alice;
```

2. Grant role access (the workaround)

To allow the user to view and query PD-created knowledge bases, grant them membership in the `visual_pipeline_user` role:

```
GRANT visual_pipeline_user TO alice;
```

3. Configure

Update the GenAI Builder agent configuration to use the `alice` credentials (username and password) you set.

3.6.3.5.8 Editing external inference service parameters returns a name change error

!!!tip

Resolved in HM versions 2026.3 and later.

Workaround: To update the parameters of an external inference service, you must delete the existing service and create a new one with the desired configurations. This issue specifically affects external model services and doesn't impact internal inference services.

3.6.3.5.9 Missing validation message for invalid external model names

!!!tip

Resolved in HM versions 2026.3 and later.

To be valid, both names must:

- Consist only of lowercase alphanumeric characters, hyphens (`-`), or dots (`.`).
- Start and end with an alphanumeric character.

Workaround: Ensure that all names provided during registration strictly follow the character requirements. If the page appears to hang after register an external service, verify that your service and model names don't contain uppercase letters or unsupported special characters.

3.6.3.5.10 Pipeline creation fails on PGD clusters with AIDB v6.1

!!!tip Resolved in HM versions 1.4.0 and later, which ships with AIDB 7.3.2.

Description: In HM version 2026.3, when using Agent Factory with EDB Postgres Distributed (PGD) and AIDB v6.1, creating a pipeline on a replicated table may fail. You may encounter the following error: `ERROR: replicated relation <table_name> cannot have triggers that are owned by roles other than the relation owner (SQLSTATE 42501)`.

This occurs because PGD requires that both the trigger and its associated trigger handler functions be owned by the same role that owns the table (typically `visual_pipeline_user`). In version 2026.3, these functions may be assigned to a different owner by default.

Workaround: To resolve this ownership conflict, a superuser must manually transfer ownership of the AIDB trigger handler functions to the `visual_pipeline_user` role.

Execute the following SQL commands on the affected database:

```
ALTER FUNCTION aidb.pipeline_background_trigger_handler OWNER TO visual_pipeline_user;
ALTER FUNCTION aidb.pipeline_live_trigger_handler OWNER TO visual_pipeline_user;
```

3.6.3.5.11 External cloud-hosted OCR models fail to register via the HM console

Description: The `aidb.sync_hcp_models` function automatically appends a `/v1/infer` suffix to all model URLs during synchronization. While this suffix is required for local containerized NIM models, it breaks connectivity for external cloud-hosted models such as NVIDIA's PaddleOCR (which uses a URL path like `.../cv/baidu/paddleocr`). As a result, external OCR models cannot be successfully configured or used via the HM console.

Workaround: Bypass the HM console registration flow and register the model directly in the database using SQL via the `aidb.create_model` function, providing the correct URL explicitly.

Option 1 — use the built-in default URL for remote NVIDIA models:

```
SELECT aidb.create_model(
    my_ocr_model,
    nim_paddle_ocr,
    credentials => jsonb_build_object(api_key, current_setting(aidb.nvidia_nim_api_
↪key, true)),
    replace_credentials => true
);
```

Option 2 — explicitly set a custom full URL:

```
SELECT aidb.create_model(
    my_ocr_model,
    nim_paddle_ocr,
    config => {"url":"https://ai.api.nvidia.com/v1/cv/baidu/paddleocr"}::JSONB,
    credentials => jsonb_build_object(api_key, current_setting(aidb.nvidia_nim_api_
↪key, true)),
    replace_credentials => true
);
```

3.6.3.6 Analytics and tiered tables

3.6.3.6.1 Updating tiered, partitioned tables fails with PGAA error

!!!tip

Resolved in HM versions 2026.2 and later.

3.6.3.6.2 PGAA causes a server crash (SIGSEGV) when executing JOIN queries with constant result relations in PostgreSQL 17

!!!tip Resolved in HM versions 1.4.0 and later.

Description: When the Postgres Analytics Accelerator (PGAA) extension is loaded, executing a `JOIN` query where one side is a constant result relation — for example, `SELECT aaa FROM v_dual LEFT JOIN bbb` or `SELECT aaa FROM (SELECT 1) d LEFT JOIN bbb` — causes a SIGSEGV crash. This issue is due to a type-confusion bug in PostgreSQL 17, where three different path subtypes share the same `T_Result` pathtype.

Workaround: Disable join and aggregate pushdown in PGAA:

```
ALTER DATABASE <your_database> SET pgaa.enable_join_pushdown    = off;
ALTER DATABASE <your_database> SET pgaa.enable_groupby_pushdown = off;
```

If you're using a Advanced High Availability or Distributed High Availability (PGD) cluster, replicate the settings to all nodes:

```
SELECT bdr.run_on_all_nodes($$
ALTER DATABASE <your_database> SET pgaa.enable_join_pushdown    = off;
ALTER DATABASE <your_database> SET pgaa.enable_groupby_pushdown = off;
$$);
```

3.6.3.7 HM console and observability

3.6.3.7.1 False positive backup alerts for Advanced High Availability and Distributed High Availability (PGD) clusters

!!!tip

Resolved in HM versions 2026.2 and later.

Workaround: To avoid unnecessary notifications in the HM console, you can silence these specific alerts for your Advanced High Availability and Distributed High Availability (PGD) clusters using one of the following methods:

- HM console: In your project, navigate to **Settings** > **Alerts** and silence `pem_time_since_last_backup_available_seconds` and `pem_backup_longer_than_expected` .
- Alertmanager: Create a silence rule for these two alerts targeting your Advanced High Availability and Distributed High Availability (PGD) environment.

3.6.3.7.2 Event markers overlapping when displaying two different events

!!!tip

Resolved in HM versions 2026.3 and later.

Workaround: Use the Event markers filters to display specific events and avoid overlapping Event markers.

3.6.3.7.3 HTTP 431 "Request Header Fields Too Large" error when accessing the Estate page

!!!tip

Resolved in HM versions 2026.3 and later.

Workaround: For users impacted by this error, an organization owner must reduce the size of the request headers by performing one of the following actions:

- Revoke project access: Remove the affected user's access to projects that are no longer actively required.
- Remove unused projects: Delete projects that are no longer in use to reduce the total count associated with the user's profile.

3.6.3.7.4 Missing notifications for task and cluster lifecycle events

!!!tip Resolved in HM versions 2026.4 and later. However, the workaround still applies to upgrades towards 2026.3.

Description: After upgrading to version 2026.3, users may stop receiving in-app and webhook notifications for cluster lifecycle events and automation tasks (such as triggered scaling or maintenance). This issue is caused by a failed database migration (`005_activity_processor_setting.sql`) where orphaned tables from previous versions conflict with the new schema, preventing the activity processor from initializing. Fresh installations are not affected.

Workaround: An HM administrator must manually create the missing settings table and initialize the timestamp in the internal application database.

Execute the following commands within the `app-db` pod located in the `upm-beaco-ff-base` namespace:

```
CREATE TABLE upm_notification_system.activity_processor_settings (
    last_processed_at TIMESTAMPTZ NOT NULL
);

INSERT INTO upm_notification_system.activity_processor_settings (last_processed_at)
VALUES (NOW());
```

3.6.3.7.5 WAL storage percent metric displays incorrect values on the monitoring page

!!!tip Resolved in HM versions 2026.4 and later.

Workaround: n/a

3.6.3.7.6 GUC settings aren't applied until after a restart

!!!tip Resolved in HM versions 1.4.0 and later.

Description: GUC settings configured in the HM console, such as `shared_preload_libraries` , aren't applied immediately. The settings remain in their previous state until at least one restart has been performed.

Workaround: After applying GUC changes in the HM console, trigger a restart to ensure the new settings take effect.

3.6.3.7.7 Alert threshold updates intermittently fail to take effect

!!!tip Resolved in HM versions 2026.5 and later.

Description: After saving a new alert threshold value in the HM console — for example, reducing the memory usage alert threshold from 90% to 85% — the change is intermittently not applied and the threshold reverts to the previous value.

Workaround: In the alert settings, verify that the **Notify After** field is set to a value greater than 0. If it's currently set to 0, update it to a non-zero value and save.

3.6.3.7.8 Asset Library apps require Project Owner or Project Editor role

Description: All Asset Library apps require users to have at least the **Project Editor** role to access, deploy, or manage them. Users with lower-privilege roles, such as **Project Viewer**, can't access Asset Library apps. This role requirement applies to all Asset Library apps and isn't currently configurable on a per-app basis.

Workaround: Ensure that any user who needs to access or manage Asset Library apps is assigned at least the **Project Editor** role for the relevant project. Role assignments can be managed by a Project Owner or organization owner in the project settings.

3.6.3.7.9 Asset Library apps sorted by package name instead of display name

!!!tip Resolved in HM versions 2026.5 and later.

Description: In the **Asset Library**, apps are sorted by their internal package name rather than their display name. This causes apps to appear in unexpected positions when browsing alphabetically — for example, Apache Airflow and Apache Superset may not appear together as expected.

Workaround: None. Manually browse the full app list to locate apps until this is resolved in a future release.

3.6.3.7.10 Metabase fails to deploy when using an EPAS cluster as its database backend

Description: Deploying Metabase from the **Asset Library** fails when the selected database cluster is an EDB Postgres Advanced Server (EPAS) instance. Metabase enters a `CrashLoopBackOff` state with a Liquibase migration error: `ERROR: syntax error at or near "$"`. This occurs because Liquibase identifies EDB Postgres Advanced Server as a different database type than community PostgreSQL. As a result, property definitions scoped to `dbms="postgresql"` (such as `${TEXT.TYPE}` and `${TIMESTAMP_TYPE}`) are skipped, while dependent changesets still execute with unresolved placeholders, causing the SQL syntax error.

Workaround: Use a community PostgreSQL cluster as Metabase's database backend instead of EDB Postgres Advanced Server. EDB Postgres Advanced Server clusters can still be connected to Metabase as data sources for analytics after deployment.

3.6.3.7.11 Asset Library apps pgAdmin4 and Apache Superset fail to deploy on RHOS

!!!tip This issue was resolved for Apache Superset in HM versions 1.4 and later. pgAdmin4 continues to have this issue.

Description: In Red Hat OpenShift (RHOS) environments, deploying the pgAdmin4 and Apache Superset Asset Library applications fails due to OpenShift Security Context Constraint (SCC) violations. The upstream images for these apps run as specific user IDs (for example, `5050` for pgAdmin4 and `0` for Apache Superset's init container) that fall outside the allowed UID ranges enforced by OpenShift's restricted SCC policies. This is an upstream image compatibility issue.

Workaround: Don't deploy pgAdmin4 or Apache Superset in RHOS environments until this issue is resolved upstream.

3.6.3.7.12 Editing cryptographic keys for Airflow or Superset causes the app to enter a failed state

!!!tip Resolved in HM versions 2026.5 and later.

Description: Editing the `fernetKey` or `apiSecretKey` for Airflow, or the `secretKey` for Superset, through the Asset Library UI after initial deployment causes the app to enter a failed state. These keys are used for cryptographic operations — including encrypting connections, variables, sessions, and CSRF tokens — and the underlying Helm charts don't support key rotation. Changing these values after initial creation creates a mismatch between the new keys and data encrypted with the original keys, breaking the application.

Workaround: Don't modify `fernetKey` , `apiSecretKey` (Airflow), or `secretKey` (Superset) after initial app creation. If the app has entered a failed state due to a key change, redeploy it with the original key values to restore functionality.

3.6.3.7.13 Superset deployment fails with `DuplicateTable` error

Description: Deploying Superset from the Asset Library may fail with a `DuplicateTable` error: `psycopg2.errors.DuplicateTable: relation "idx_user_id" already exists` . This occurs when the selected backend database already contains tables from a previous Superset deployment.

Workaround: Use a new, empty database when deploying Superset.

3.6.3.7.14 Asset Library launch app always uses port 443 regardless of custom portal port

!!!tip Resolved in HM versions 2026.5 and later.

Description: When HM is configured with a custom `portal_port` value other than 443, app or flow URLs in the **Estate** use port 443 instead of the configured port. This causes the launched app or flow to be unreachable in environments where the HM portal isn't exposed on the standard HTTPS port.

Workaround: Manually replace port 443 with your configured `portal_port` in the app or flow URL to access it.

3.6.3.7.15 Asset Library app CPU and memory metrics include failed and completed pod states

!!!tip Resolved in HM versions 1.4.0 and later.

Description: The CPU and memory usage metrics displayed for Asset Library apps include pods in `Failed` or `Completed` states, such as pods spawned by jobs. These metrics are only available during the app's startup phase, so the displayed figures may not accurately reflect the app's current resource usage until the metrics stabilize.

Workaround: Wait 30 minutes after app startup for the metrics to stabilize and reflect the correct state.

3.6.3.7.16 Deleting an app or flow gets stuck due to missing service account

!!!tip Resolved in HM versions 2026.5 and later.

Description: When deleting an app or flow, the deletion sometimes gets stuck with the error `Delete failed: Preparing kapp: Getting service account: serviceaccounts "beacon-app-installer" not found`. This occurs because the `PackageInstall` resource's finalizer attempts to use a service account that no longer exists, preventing the app namespace from being deleted.

Workaround: Identify the stuck namespace and manually remove the finalizer from the `PackageInstall` and its associated `App` resource.

1. Find namespaces stuck in `Terminating`:

```
kubectl get ns --field-selector status.phase=Terminating
```

The app namespace matches the application ID, for example, `app-abc123xyz`.

1. Identify the `PackageInstall` name:

```
kubectl get packageinstall -n <app-namespace>
```

The name follows the format `<app-namespace>-pkginstall`.

1. Remove the finalizer from the `PackageInstall` and the associated `App` (the `App` name matches the `PackageInstall` name):

```
kubectl patch packageinstall <name> -n <app-namespace> --type=json -p=[{"op": "remove
↪", "path": "/metadata/finalizers"}]
kubectl patch app <name> -n <app-namespace> --type=json -p=[{"op": "remove", "path":
↪"/metadata/finalizers"}]
```

1. Verify the namespace deletion completes:

```
kubectl get ns <app-namespace>
```

3.6.3.7.17 Asset Library apps may get stuck pending on RHOS when storage classes differ

Description: On Red Hat OpenShift (RHOS) environments where the cluster default storage class differs from the `spec.globalParameters.storage_class` value — for example, where the default is `lvms-vm` (backed by control-plane nodes) but the configured storage class is `lvms-bm` (backed by worker nodes) — Asset Library apps that require persistent volumes, such as pgAdmin4, may get stuck in a `Pending` state indefinitely. This occurs because the app's PVC inherits the cluster default storage class, which may not have capacity available on the nodes where app pods can be scheduled.

Workaround: On environments where an app deployment gets stuck, delete the pending PVC in the app namespace and recreate it with `storageClassName` set to the value from `hcp.spec.globalParameters.storage_class`. Alternatively, change the cluster default StorageClass to match `hcp.spec.globalParameters.storage_class` so all new PVCs automatically use the correct storage.

3.6.3.8 Clusters and upgrades

3.6.3.8.1 Upgrade to version 2026.4 fails due to immutable `fluent-bit-k8s-events` StatefulSet

Description: When upgrading to HM 2026.4, the upgrade process may fail with a `Failed to install component` error for `upm-fluent-bit`. This occurs because the upgrade attempts to modify an immutable field in the `fluent-bit-k8s-events` StatefulSet spec, which the Kubernetes API server rejects. The error message is: `StatefulSet.apps "fluent-bit-k8s-events" is invalid: spec: Forbidden: updates to statefulset spec for fields other than 'replicas', 'ordinals', 'template', 'updateStrategy', 'persistentVolumeClaimRetentionPolicy' and 'minReadySeconds' are forbidden`.

Workaround: Before upgrading to 2026.4, manually delete the `fluent-bit-k8s-events` StatefulSet in the `edb-observability` namespace. The operator will recreate it with the correct configuration during the upgrade:

```
kubectl delete statefulset fluent-bit-k8s-events -n edb-observability
```

3.6.3.8.2 Major version upgrade fails for Advanced High Availability and Distributed High Availability (PGD) clusters with witness groups

!!!tip

Resolved in HM versions 2026.2 and later.

Workaround: Avoid performing an in-place major version upgrade for this specific cluster configuration. Instead, provision a new Advanced high Availability or Distributed High Availability (PGD) cluster using the target major version and use the Data Migration Service (DMS) to migrate your schema and data from the old cluster to the new one.

3.6.3.8.3 Upgrade to version `2026.3` fails due to logging namespace termination

!!!tip Resolved in HM versions 2026.4 and later. However, the workaround still applies to upgrades towards 2026.3.

Workaround option 1 (recommended):

Before initiating the upgrade to version 2026.3, run the following command to allow the namespace to be orphaned during the cleanup process:

```
kubectl annotate namespace logging kapp.k14s.io/delete-strategy=orphan --
↪overwrite=true
```

Workaround option 2 (ff upgrade is already stuck):

If the upgrade has already started and the `logging` namespace is stuck in a `Terminating` state, run the following command to manually remove the finalizers and unblock the upgrade:

```
kubectl patch fluentbit/fluent-bit -n logging --type=merge -p {"metadata":{"finalizers
↪":[]}}
```

3.6.3.9 Agent Factory

3.6.3.9.1 Knowledge base data isn't replicated to non-primary nodes on PGD clusters

!!!tip Resolved in HM versions 1.4.0 and later.

Description: When using Agent Factory with AIDB 7.3.0 on a PGD cluster, knowledge base data (visible via the `aidb.knowledge_bases_v6` view) is only present on the primary node and isn't automatically replicated to other

nodes. This occurs because the `aidb.knowledge_base_registry` and `aidb.knowledge_base_pipeline` tables, introduced in AIDB 7, weren't included in the PGD setup script.

Workaround: Before creating any knowledge bases, manually run the following SQL commands on the PGD cluster:

```
SELECT bdr.alter_sequence_set_kind(aidb.knowledge_base_registry_id_seq::regclass,␣
↪galloc, 1);
SELECT bdr.replication_set_add_table(aidb.knowledge_base_registry);
SELECT bdr.replication_set_add_table(aidb.knowledge_base_pipeline);
```

3.6.3.9.2 Index analysis actions fail due to missing `hypopg` extension

Description: The Agent Factory actions `analyze_workload_indexes` and `analyze_query_indexes` are currently unavailable because the required `hypopg` extension is missing from the standard HM database images. Attempting to run these actions will result in an error indicating the extension is not installed.

Workaround: There is currently no manual workaround to install this extension on managed clusters. Avoid using these two specific actions until the extension is included in a future image update.

3.6.3.9.3 Deleted clusters still appear in AI components

!!!tip

Resolved in HM versions 2026.2 and later.

Workaround: Before selecting a cluster in the AI components, verify its status in the HM console or via the clusters API. Ensure you only select clusters that are currently in a "Healthy" or "Active" state.

3.6.3.9.4 Missing validation for required MCP tool parameters

Description: Several actions within the MCP tools allow fields to be left empty during configuration, which results in execution errors rather than validation warnings. If these parameters are not populated, the workflow may fail to build or return an "Unsupported object type" error.

Affected actions and parameters:

- `list_objects` and `get_object_details` : The Object Type field must not be empty.
- `explain_query` : The Analyze and `hypothetical_indexes` fields must be populated with specific data types.

Workaround: Ensure the following parameters are manually filled before executing the flow:

- `list_objects` (Object Type): Enter `table` , `view` , `sequence` , `function` , `stored procedure` , or `extension` .
- `get_object_details` (Object Type): Enter `table` , `view` , `sequence` , or `extension` .
- `explain_query` (Analyze): Enter `True` or `False` .
- `explain_query` (`hypothetical_indexes`): Use tool mode to provide a valid list.

3.6.3.9.5 Some EDB Component tools are non-functional due to incompatible API request formatting

Description: Some tools within the EDB Component are currently non-functional due to incompatible API request formatting. By default, the component exposes too many tools, which can exceed API limits and trigger errors.

Workaround: Enable tool mode and apply specific filters to limit the number of tools exposed by the component.

3.6.3.9.6 Advanced high Availability and Distributed High Availability (PGD) witness groups appear in database selection lists

!!!tip

Resolved in HM versions 2026.2 and later.

Workaround: When configuring these components, manually ignore any entries labeled as a witness group. Only select data groups to ensure a valid connection and successful data retrieval.

3.6.3.9.7 Text search may match on non-visible fields

Description: When using the search functionality in the HM console, for example in the **Activity Log** section, the system performs a full-text search across all underlying data fields. As a result, search results may include resources or activity log entries where the visible **Activity Name** or other shown attributes don't appear to match the search term.

Workaround: There is currently no workaround for this behavior as it is a result of the underlying search design. Users should be aware that search results are inclusive of internal action details not displayed in the primary log view.

3.6.3.9.8 Langflow MCP server components not supported in HM Flow Hosting

Description: Flows that include a Langflow MCP (Model Context Protocol) server component fail when executed via the Hybrid Manager Flow Hosting API with the following error: `Error building Component my_mcp: Error updating tool list: Langflow MCP server functionality is not available. This feature requires the full Langflow installation.` The HM Flow Hosting runtime (`lfx-runtime`) is a lightweight environment that does not include the full Langflow installation. Flows using MCP server components will appear to deploy successfully — the pod reports as healthy — but will fail at runtime when invoked via the REST API.

Workaround: Avoid including Langflow MCP server components in flows intended for deployment via the HM Flow Hosting API. If you need to run the Airman or HM MCP servers, use the EDB Airman or EDB Platform components instead — these components are compatible with the lfx-runtime environment. For any other MCP server functionality, use the full Langflow interactive environment instead.

3.6.3.9.9 EDB Knowledge Base component returns a 403 error when dragged into Langflow

!!!tip Resolved in HM versions 2026.5 and later.

Description: When dragging the EDB Knowledge Base component into Langflow, an error appears in the bottom left corner: `API call to .../projects/None/clusters returned 403 .` This occurs because the project ID is missing from the API call at the time the component is loaded.

Workaround: Select the **Hybrid Manager Project** dropdown on the component and select **Refresh list** to load the available projects. Once the project ID is populated, the error clears.

3.6.3.9.10 EDB Knowledge Base component fails to connect when using a global variable for the database user

!!!tip Resolved in HM versions 2026.5 and later.

Description: When configuring the EDB Knowledge Base component in Langflow with a global variable for the database user field, the connection attempt fails. Global variable values aren't resolved correctly for this field during the initial connection setup.

Workaround: Enter the database username as a raw text value instead of a global variable.

3.6.3.9.11 Running a test query against an empty knowledge base returns a 500 error

!!!tip Resolved in HM versions 1.4.0 and later.

Description: When running a test query from the **Knowledge Bases** list view against a knowledge base whose underlying embeddings table is empty — for example, while the KB pipeline is still in a `Processing` state — the API returns an HTTP 500 error: `ERROR: Query returned no data. Hint: The "<pipeline_name>" table is likely empty.` The API should return an empty result set instead.

Workaround: Wait until the knowledge base pipeline finishes processing before running test queries. The pipeline status changes from `Processing` to an active state once data has been indexed and the embeddings table is populated.

3.6.3.9.12 Flow Hosting deployment fails intermittently with `FlowValidationError`

!!!tip

Resolved in HM versions 2026.5 and later.

Description: When creating a flow via the Langflow UI, the secret or credential field of any components must use a global variable. Pasting a password directly will cause errors when deploying the flow via HM Flow Hosting. This is a default behavior across all components from Langflow v0.19.

Workaround: Store the credential in a global variable and reference it in the flow configuration.

3.6.3.9.13 EDB Model Server component causes agent failures when used with the EDB Airman MCP component

!!!tip Resolved in HM versions 1.4.0 and later.

Description: When building an agent flow in Langflow that uses the EDB Model Server component together with the EDB Airman MCP component, the agent may fail with an error similar to: `messages with role 'tool' must be a response to a preceding message with 'tool_calls'`. This issue is intermittent and doesn't occur when using other model provider components.

Workaround: Use the OpenAI component or the generic model component in place of the EDB Model Server component. Flows using these alternative model providers aren't affected by this issue.

3.6.3.9.14 `upm-ai-model-server-beaco` OOMKills due to insufficient memory limits

!!!tip Resolved in HM versions 2026.5 and later.

Description: The `upm-ai-model-server-beaco` pod may enter a crashloop due to OOMKill in environments with significant AI workloads. The default memory allocation for this component is insufficient and isn't currently configurable via the standard Helm values or the `HybridControlPlane` CR.

Workaround: Manually patch the Kubernetes deployment to increase the memory resource limits for the `upm-ai-model-server-beaco` pod from the default 256MB to 512MB, then monitor the pod to confirm stability.

3.6.3.9.15 Updated Langflow flow changes aren't immediately reflected in HM after republishing

Description: When you edit a previously published Langflow flow in the HM Langflow editor and republish it to the Asset Library, the updated version doesn't immediately appear in HM. There is a delay of up to 5 minutes before the changes propagate and become visible.

Workaround: After republishing a flow, wait approximately 5 minutes before verifying that the changes are reflected in HM.

3.6.3.10 Migrations

Known issues pertaining to the HM Migration Portal, data migration workflows, and schema ingestion workflows are maintained on a dedicated page in the *Migrating databases with Hybrid Manager* documentation. See *Known issues, limitations, and notes* for a complete list.

3.6.4 Supported Postgres distributions

Selecting the right Postgres distribution when creating a new database cluster in Hybrid Manager (HM) is crucial for optimizing your database environment. EDB offers a range of options, each tailored to different needs, from open-source flexibility to enterprise-grade features and Oracle compatibility.

3.6.4.1 EDB Postgres Advanced Server

EDB Postgres Advanced Server is EDB's most comprehensive PostgreSQL distribution. It supports features such as *advanced security* , *advanced SQL* , high-availability and multi-master replication in combination with *EDB Postgres Distributed (PGD) cluster with 2 data groups and 1 witness group not healthy* and, optionally, Oracle compatibility . It's ideal for demanding enterprise workloads and seamless migration from cost-intensive Oracle databases.

This option is suited for:

- Enterprises migrating from Oracle databases seeking high compatibility to reduce migration costs and complexity.
- Organizations with mission-critical applications requiring advanced security, performance, and management tools.
- Businesses that need comprehensive enterprise-grade features, including enhanced auditing, SQL injection protection, and robust high availability.

3.6.4.2 EDB Postgres Extended Server

EDB Postgres Extended Server provides specific enhancements designed for enterprise requirements, particularly in data security, for example, through transparent data encryption . It also supports advanced replication with EDB's powerful high-availability and multi-master replication solution, *EDB Postgres Distributed (PGD) cluster with 2 data groups and 1 witness group not healthy* .

This option is suited for:

- Enterprises requiring enhanced data-security features.
- Organizations looking to leverage EDB's high-availability and multi-master replication solutions.
- Users who want a PostgreSQL experience with additional enterprise capabilities while maintaining strong compatibility with community PostgreSQL.

3.6.4.3 PostgreSQL

Open-source PostgreSQL is a powerful, reliable, and feature-rich relational database system. It's known for its strong adherence to SQL standards, robust architecture, data integrity, and a global community.

This option is suited for:

- Organizations prioritizing cost effectiveness and open-source freedom.
- Projects that don't require advanced security features, multi-master replication solutions, or Oracle compatibility.
- Teams with in-house PostgreSQL expertise and the resources to manage their database environment.

3.6.4.4 Distribution comparison

For a comparison of the available distributions by functionality and feature, see:

- *Feature support per distribution*
- *Advanced replication*
- *Advanced security*
- *Advanced SQL*
- *Advanced performance*
- *Oracle support*
- *Tools and extensions*
- *Deployment options*

3.6.4.5 Documentation by distribution

To get the most out of any of the Postgres distributions, refer to its documentation:

- PostgreSQL
- *EDB Postgres Extended Server*
- *EDB Postgres Advanced Server*

CHAPTER 4

Installing Hybrid Manager

4.1 Installing Hybrid Manager

This section contains everything you need to install and configure Hybrid Manager (HM).

- ** Installation guide ** — The six-phase guide covering architecture planning through post-installation configuration. Start here for new installations.
- ** *Step 3: Task Configuration* ** — Detailed SOPs for advanced configuration topics: TLS, IdPs, KMS for TDE, multi-DC, image sync, and more.
- ** *Migrate from bootstrap to operator* ** — For existing bootstrap installations migrating to the operator method.
- ** *Troubleshooting Langflow in Hybrid Manager* ** — Guidance for diagnosing and resolving installation issues.

4.1.1 step-by-step guide

4.1.1.1 Overview

This guide walks you through planning, configuring, and installing Hybrid Manager (HM) onto your infrastructure.

The process is divided into **6 chronological phases** to ensure dependencies are met before deployment proceeds.

!!!Note

Please follow these phases in order. Skipping steps, most importantly *Phase 2: Gathering your system requirements* or *Preparing your environment* , are the most common cause of installation failure.

4.1.1.2 Phase 1: Planning your architecture

> **Role focus:** CTO / Architect / Lead Engineer

This initial phase is about **designing** your deployment architecture.

It concludes with documenting that architecture (Architectural Decision Record (ADR)), which then serves as the blueprint for all subsequent steps.

- **Goal:** Define the foundational architecture for HM.
- **Key activities:**
- Review and select the appropriate **Kubernetes flavor**, topology, and deployment locations.
- Document architectural decisions (ADR) as inputs for infrastructure requirements and Kubernetes installation (Phases 2 and 3).
- **Documentation:** *Phase 1: Planning your architecture*

!!!Note

You, as the customer, must ultimately own your deployment architecture. While EDB's Sales Engineering, Professional Services, Support Team, or documentation can be consulted, the final architectural decisions rest with your team.

4.1.1.3 Phase 2: Gathering your system requirements

Role focus: Infrastructure engineer / CSP Administrator / Platform Engineer

This phase focuses on **identifying, preparing, and verifying** the infrastructure needed to support the architecture defined in Phase 1. This is where you begin preparing your configuration.

- **Goal:** Fulfill all prerequisites and prepare the inputs for the core configuration.
- **Key Activities:**
- Identify **hardware, networking, storage, DNS requirements, and other system requirements** for your chosen architecture.
- Ensure all prerequisites are met for Kubernetes deployment.
- Prepare to populate critical configuration entries for Hybrid Manager in the `HybridControlPlane` CR.
- **Documentation:** *Phase 2: Gathering your system requirements*

!!!Note

EDB's Sales Engineering and Professional Services are the primary resources for communicating and clarifying detailed deployment requirements during the sales cycle or via a Statement of Work (SoW). The official documentation also provides comprehensive checklists.

4.1.1.4 Phase 3: Deploying your Kubernetes cluster

Role focus: Kubernetes engineer / CSP Administrator

This phase executes the deployment of your chosen Kubernetes environment based on the previous planning.

- **Goal:** Deploy and provision the operational Kubernetes cluster.
- **Key activities:**
- Deploy Kubernetes according to the defined architecture and infrastructure requirements.
- Implement necessary abstractions (node pools, storage classes, networking).
- Validate that infrastructure requirements are met and prepare critical configuration entries.
- **Documentation:** *Phase 3: Deploying your Kubernetes cluster*

!!!Note

EDB support context: Customers own the installation and lifecycle operation of their Kubernetes cluster. Professional Services can be engaged via a Statement of Work (SoW), and Support can offer assistance through knowledge base articles.

4.1.1.5 Phase 4: Preparing your environment

Role focus: Site reliability engineer (SRE) / Infrastructure Engineer

Final preparations are made here to ensure the cluster is optimally configured to receive and run HM.

- **Goal:** Provision necessary security and configuration resources.
- **Key activities:**
- Provision essential **secrets and configuration resources**
- Perform further validation of cluster readiness for Hybrid Manager installation.
- **Documentation:** *Preparing your environment*

!!!Note

EDB Support Context: Final preparation for Hybrid Manager is primarily the **customer's responsibility** (except for Engineered Systems), as is the cluster's lifecycle operation. **Professional Services** can be engaged via a Statement of Work (SoW), and **Support** can offer assistance through knowledge base articles.

4.1.1.6 Phase 5: Installing Hybrid Manager

Role focus: Site reliability engineer (SRE) / Infrastructure Engineer

This is the final deployment step, where Hybrid Manager is installed into your prepared environment.

- **Goal:** Deploy Hybrid Manager and confirm successful operation.
- **Key activities:**
- Final validation of the configuration files, and environment and deployment settings.
- Deploy Hybrid Manager
- Confirm successful installation and operational readiness.
- **Documentation:** *Installing Hybrid Manager*

4.1.1.7 Phase 6: Exploring your post-installation options

Role focus: Infrastructure Engineer/Platform Administrator

Once the system is running, this phase covers the "Day 2" configurations required to unlock advanced features like AI workloads, external monitoring, and migration assistance.

- **Goal:** Integrate the platform with your wider ecosystem and enable optional modules.
- **Key activities:**
- **AI Capabilities:** Connect S3 buckets and Fernet keys to enable the GenAI Builder.
- **Observability:** Configure notification channels (PagerDuty, Email) and install agents for external database monitoring.
- **Migration:** Enable Migration Co-pilot to assist with schema conversion.
- **Documentation:** *See Post-installation options*

4.1.1.8 Configuring HM

Several detailed configuration SOPS are referenced throughout the installation procedure and are either useful or necessary to consult for specific instructions:

1. *Syncing images to a local private registry*
2. *Using the in-cluster container registry*
3. *Configuring image discovery*
4. *Customizing secrets*
5. *Implementing TLS certificates*
6. Deploying HM across multiple data centers
7. *Configuring IdPs*
8. Enabling KMS for TDE
9. *Optimizing service mesh memory*
10. *Migrating from the bootstrap to operator method*

4.1.1.8.1 Planning your architecture

Overview

Role: CTO / Architect / Lead Engineer

Prerequisites

- *A list of the business goals that your architecture plan should enable (examples: bugdet constraints, desired uptime, and desired latency)*

Outcomes

- *An Architectural Decision Record (ADR) defining the topology, locality, and redundancy model of your Hybrid Manager (HM) architecture. (at minimum: architecture diagrams with notes)*
- *Initial inputs for the HM configuration — a ``HybridControlPlane`` Custom Resource manifest.*

!!!Note

You, as the customer, ultimately own your deployment architecture. While EDB's Sales Engineering, Professional Services, Support Team, or documentation can be consulted, the final architectural decisions rest with your team.

Architectural discovery

The goal of architectural discovery is to navigate and then document the necessary decisions to successfully deploy Hybrid Manager (HM). These decisions form the blueprint for ** *Phase 2: Gathering your system requirements* ** (Phase 2) and ** *Preparing your environment* ** (Phase 3).

The accompanying questions cover a broad set of considerations extending beyond just the database layer. This guide should be viewed from two perspectives:

- **Current state:** Where are your existing database and application workloads today?
- **Target state:** Where do you intend to deploy HM immediately, and where do you plan to expand over the next 1-2 years?

Recommendation: Acquiring and reviewing diagrams of your current and target state is the most efficient way to complete this phase.

Locality: Where will HM live?

Understanding the physical or logical locations of your database and dependent applications is crucial for determining the necessary architecture.

- **Questions to answer:**
- Where is the current database solution located in terms of **cloud regions (CSP)** or **physical data centers (on-premises)**?
- Where are the dependent application workloads for these databases located?
- Are there **upstream layers of dependency**, and where are those located?
- **Analysis:**
- Locality determines the initial scope of the deployment (for example, single cloud region vs. multi-region).
- If you plan to span multiple regions, clouds, or hybrid cloud environments, **Postgres Distributed** is likely the appropriate database service recommendation.
- The locality of upstream applications is key to minimizing network latency.

Disaster recovery (hot/cold)

Disaster recovery (DR) ensures business continuity across different locations.

- **Questions to answer:**
- How is disaster recovery—as a subset of business continuity—accomplished across these locations, or is there an **additional location assigned specifically as disaster recovery**?
- Is there an **additional location assigned specifically as DR**?
- How is DR capability **validated**, and how often?
- **Analysis:**
- Having a dedicated secondary location indicates a strong architectural requirement.
- If no formal DR practice exists, the HM DBaaS far-away replica solution may provide new capabilities.

Activeness (Active/Passive vs. Active/Active)

Activeness describes how your distributed locations are utilized for critical workloads.

- **Questions to answer:**
- If you have multiple locations, how does the critical dependent workload utilize these systems?
- Is one location active and the other passive for transaction processing (OLTP)?
- Is one location active for OLTP, and the other active for analytical processing (OLAP/BI)?
- **Analysis:**
- If your target state requires simultaneous writes to multiple database instances (that is, true **active/active** across locations), **Postgres Distributed** is the required solution due to its multi-writer capability..
- Understanding whether a location is passively waiting (cold standby) or actively running (hot standby) helps define resource requirements and recovery time objectives (RTO).

- **Business continuity:** The architectural choices around active/passive, active/active, and standby models must balance the organization's tolerance for downtime/data loss against the cost of maintaining redundant systems.

These topics naturally follow the discussion of Activeness and help complete the picture of your application ecosystem.

- Ingress traffic routing in terms of the consuming application.
- Replication at various application layers.
- Caching layers (and their location relative to the database).
- Session demands (for example, is session replication handled at the application layer?).

Lifecycle operations

Understanding your operations practices helps determine the complexity of the Kubernetes environment required to manage the database service.

- **Questions to answer:**
- Do you utilize life cycle operations patterns such as **Blue/Green** or **Canary**?
- How do you handle **DML/DDL updates** (data and schema) vs. **engine upgrades** (major versions)?
- What **pre-production environments** (staging, development, testing) are required?
- **Analysis:**
- Practices like Blue/Green deployment align well with the zero-downtime features offered by EDB's database solutions.
- The number of pre-production environments directly influences the total cluster count and resource sizing defined in ** *Phase 2: Gathering your system requirements* **.

Supported platforms

HM and Kubernetes have a 1:1 relationship—each HM deployment requires its own dedicated Kubernetes cluster. The Kubernetes cluster must be dedicated to HM in its current version; sharing with other workloads is not supported.

- **Amazon EKS** (Elastic Kubernetes Service)
- **Microsoft Azure AKS** (Azure Kubernetes Service)
- **Google GKE** (Google Kubernetes Engine)
- **ROSA** (Red Hat OpenShift Service on AWS)
- **RHOS** (Red Hat OpenShift Container Platform)
- **Rancher RKE** (Rancher Kubernetes Engine / RKE2)

!!!Note

The customer is responsible for the full life cycle management of the Kubernetes cluster: provisioning, deploying, upgrading, and scaling.

The HM distributed reference architecture represents the ultimate goal for achieving the highest levels of scale and fastest SLAs. It typically spans multiple data centers.

Diagram legend reference

The legend defines the colors and logical groupings used in the architecture diagram:

- **Locality:** The highest-level physical or logical grouping, such as a physical data center or a geographical region (for example, "City 1" and "Data Center 1").

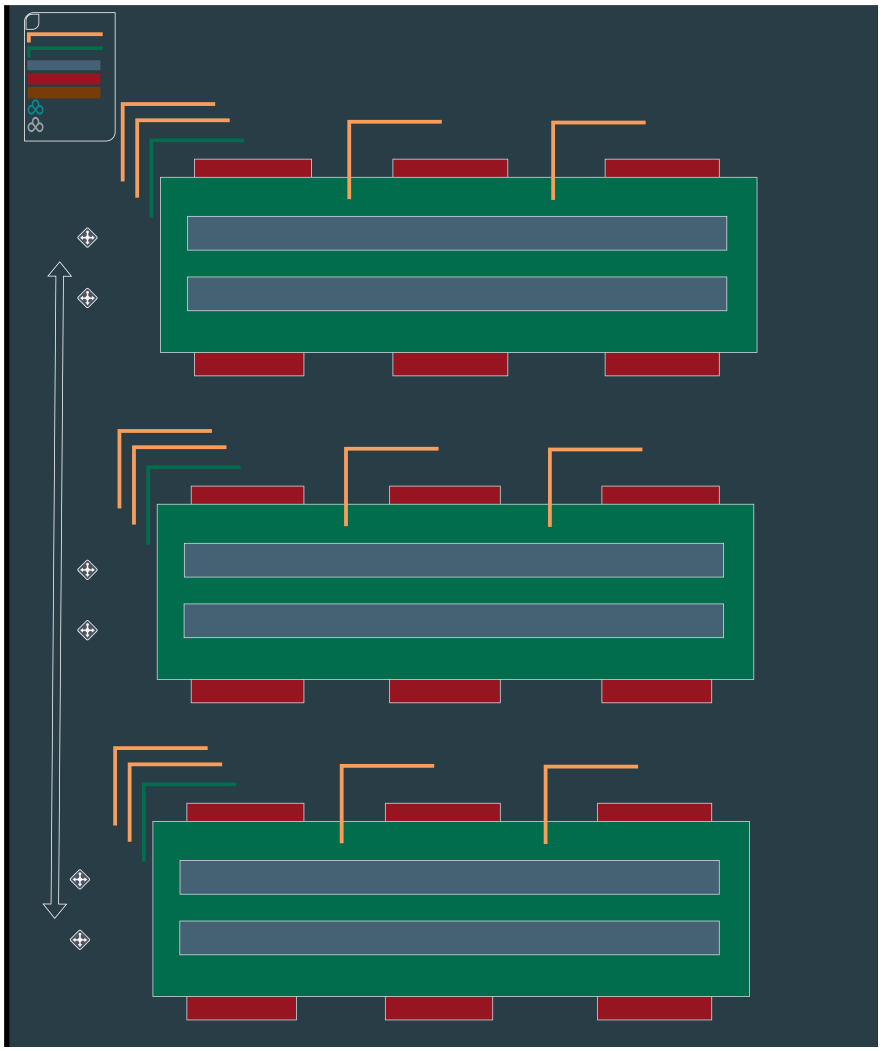


Fig. 1: HM reference architecture

- **Kubernetes Cluster:** The complete Kubernetes environment—including all CP and worker nodes—hosting the entire platform.
- **EDB HM:** The logical boundary for the core HM components. This is typically implemented as a dedicated Kubernetes namespace (for example, control-plane).
- **Compute Machine:** The virtual machines (for example, vm01, vm02, vm03) that serve as the Kubernetes worker nodes, providing the CPU, memory, and storage for the cluster.
- **Infrastructure Abstraction:** This critical layer represents Kubernetes-native resources that abstract underlying physical or virtual infrastructure. These resources must be provided by the Kubernetes cluster's environment.
- **Example 1: type: LoadBalancer**: This is a Kubernetes Service type that requests an external load balancer. In public cloud environments (like AWS, GCP, Azure), this is automatically provisioned as a managed service. In on-premises or bare-metal deployments, you must provide a solution (like MetalLB) to fulfill these LoadBalancer requests.
- **Example 2: StorageClass**: This resource abstracts the "Block Storage" and "Object Storage" requirements. It maps Kubernetes storage requests (Persistent Volume Claims) to actual, provisioned storage hardware or software (like local-pv, Ceph, vSphere, or cloud-based disks).

HM Control Plane vs. Data Plane

Attribute	Control Plane (CP)	Data Plane (DP)
Role	Orchestration, configuration, and observability	Postgres cluster hosting and replication
Protocol	HTTP (REST / gRPC) via Istio service mesh	Postgres wire protocol (TCP) per PG cluster
Multi-location topology & traffic	Star — Primary is mandatory hub across locations; Primary pushes Control outbound to each Secondary, Secondaries return Logs to Primary	Optional mesh — locations may be fully isolated or directly peered for Postgres replication; no hub required
Lifecycle isolation	All CPs share management context; CP upgrades affect the full estate	Each DP is an independent failure, security, and upgrade domain
DR role	Secondary CPs provide management continuity	DPs are the replication targets; hot/cold standby model determined per deployment
Scenarios	core, migration, ai	dbaas, analytics, ai

Protocol and ingress

The simplest way to distinguish the two planes is by protocol.

The **Control Plane is entirely EDB product**, and every component in it speaks HTTP. That HTTP traffic is served and secured by the **Istio service mesh**, which provides a single, centralized ingress surface across all CP services. Authentication, TLS, and traffic policy are all enforced at the Istio layer before a request reaches any CP component.

The **Data Plane is customer-provisioned Postgres**, and every cluster in it speaks the **Postgres wire protocol (PSQL)**. There's no middleware in the path. Each Postgres cluster listens on its own dedicated TCP connection — there's no shared proxy or mesh aggregating these connections. This is a deliberate design choice: dedicated ingress per cluster maximizes both availability and the security isolation boundary between customer workloads.

In short: **HTTP CP ingress is centralized via Istio; PSQL DP ingress is dedicated per Postgres cluster.**

Lifecycle isolation

Lifecycle isolation is the most operationally significant distinction between the two planes.

The CP and every DP it manages are on **independent upgrade lifecycles**. We publish patches on a monthly cadence for the Long Term Support (LTS) release. Customers are strongly encouraged to apply these promptly, and the architecture makes this practical: **upgrading the CP does not restart customer Postgres on the DP.** Applications continue serving connections through the upgrade, and there's no maintenance window required for Postgres availability when the CP is patched. This separation holds because the embedded CloudNativePG (CNPG) operator that manages Postgres on the DP is configured for **in-place instance-manager updates**. When the operator is upgraded, the instance manager binary inside each Postgres pod is swapped while the running postmaster continues serving traffic. No pod restart, no switchover, no application-visible event.

These monthly CP patches carry two categories of content:

1. **CP component security patches** — hardening the orchestration layer itself.
2. **New Postgres container images** — updated images for Postgres engine versions, extensions, and the container host OS, all with current security patches applied.

Once new images are available in the CP, it's **the customer's decision** when to use the CP's orchestration capabilities to roll those images out to their DP Postgres deployments. This preserves full customer control over Postgres upgrade timing while ensuring that hardened images are always available on demand.

Exception to the rule

A small fraction of CNPG operator releases — historically well under 10% — change the Postgres **Pod specification itself**, for example modifications to anti-affinity rules, container probes, or init containers. In these releases, Kubernetes must recreate the pods to match the new spec, and a rolling restart of HA Postgres clusters is unavoidable as part of the CP upgrade, even with in-place instance-manager updates enabled. When an HM release embeds one of these CNPG versions, **EDB calls this out explicitly in the HM upgrade documentation and in pre-upgrade guidance to affected customers.** This means customers know in advance whether a given HM upgrade is fully non-disruptive or whether a rolling Postgres restart will occur as part of it — and can plan a maintenance window only when one is actually required.

Availability and disaster recovery

CP availability and DP availability are independent concerns, and the two planes warrant different recovery strategies.

- Control Plane*

The CP doesn't sit in the data path of customer Postgres traffic. A CP outage doesn't interrupt connections to provisioned Postgres clusters — running workloads continue unaffected. The CP has a robust DR capability: with disciplined operational practice (regular backups, documented runbooks), a CP can be restored in **under one hour**.

- Data Plane*

Because the DP carries live Postgres traffic, higher availability targets require active replication across **failure domains** — optimally across regions. Postgres replication across DP locations (where network peering is provisioned) allows the DP to survive a regional failure without any impact to application connectivity.

- Availability zones*

Both planes can take full advantage of **availability zones within a location**. Every HM deployment location may contain CP elements, DP elements, or both. Any location can be recovered from disaster using backups. The difference is ceiling: the CP's recovery is a restore operation measured in minutes to an hour, while the DP's Postgres clusters

have the architectural capacity to remain continuously available through a disaster by maintaining live replicas in other failure domains.

Deployment architectures

!!!Note

This legend above also applies to deployment architectures below.

The minimum install colocates the HM Control Plane (CP) on the Kubernetes control nodes.

This is fully functional for:

- Centralizing a view of your Postgres/Oracle Estate.
- Database migration capabilities.
- GenAI (limited capabilities due to lack of managed Postgres instances).

Internal architecture: HM Control Plane

HM is composed of several core microservices running within the Kubernetes cluster. Understanding these components is helpful for planning resource allocation and security boundaries.

- **GenAI:** Provides the AI/ML capabilities. If enabled, this component dictates the need for GPU-enabled worker nodes in your ** *Phase 2: Gathering your system requirements* **. - *See: GenAI in HM*
- **Postgres life cycle operations:** The orchestration engine that manages deployment, scaling, and updates of the databases. - *See: Cluster Management*
- **Telemetry:** Collects metrics and logs. This service requires outbound network access to report health status. - *See: Monitoring with HM*
- **Database Migration Assistant:** Facilitates the movement of data from external sources into the platform. - *See: Migrating databases with HM*
- **Estate:** Manages the inventory of resources creating using the HM DBAAS internal system as well as external databases. - *See: Enable monitoring>On external database clusters*
- **Federation:** Manages secure communication and authorization across multiple HM instances in a Multi-Location topology. - *See: Configuring multiple data centers for HM*

Architectural dependencies

The architecture diagrams above reference several external components. While you verify the specific hardware/software requirements for these in ** *Phase 2: Gathering your system requirements* **, you must account for their connectivity in your architectural design.

- **Identity provider (IdP):** Required for user authentication. The architecture relies on OIDC (LDAP/SAML) for all human access.
- **Key Management Service (KMS):** (Optional) Required only if your security policy demands Transparent Data Encryption (TDE).
- **Object Storage:** Required for system resilience. It hosts backups, logs, and facilitates data replication for Multi-Location topologies.
- **Block Storage:** Required for database performance. Your storage architecture must provide persistent volumes (PVCs) for the Postgres data layer.
- **Local network:** The fabric connecting the CP to Data Plane. Latency here drives your Locality decisions.

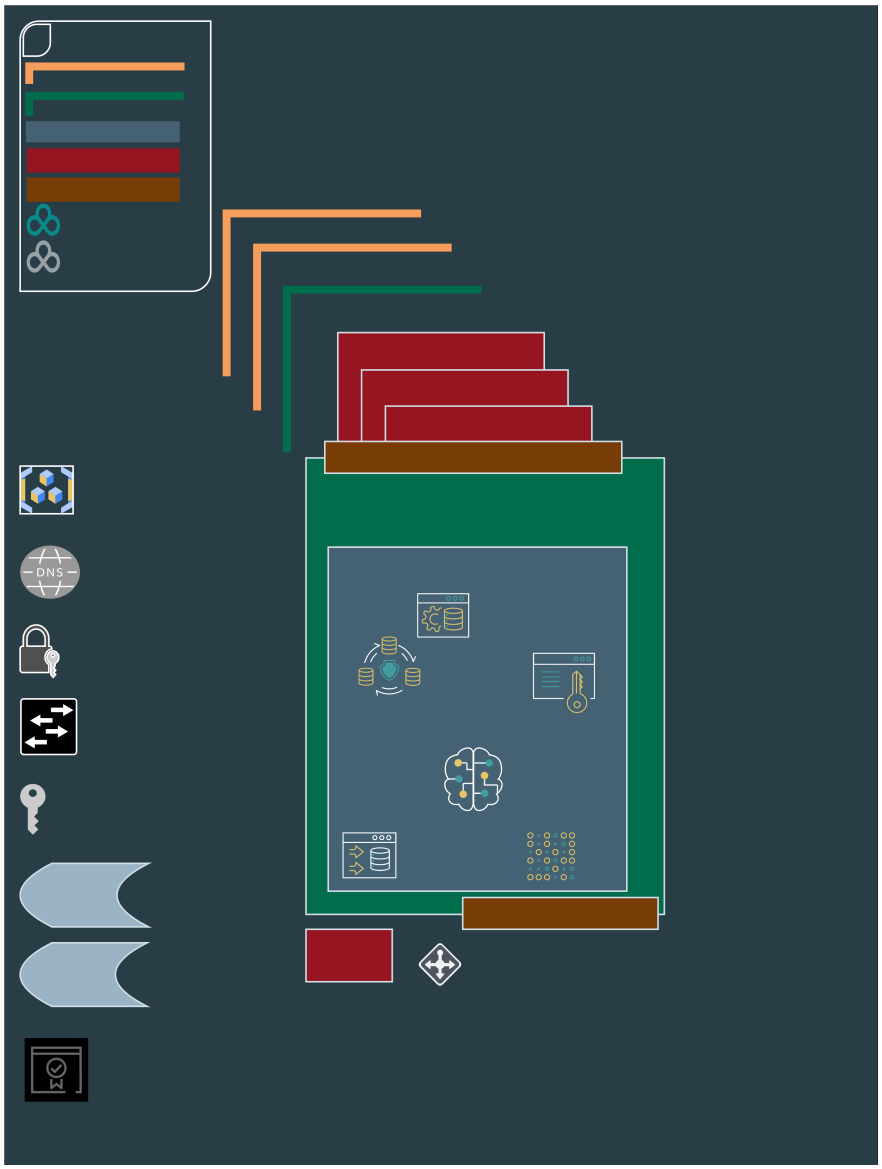


Fig. 2: HM minimum

- **Container Registry:** The source of truth for application images. For air-gapped designs, this represents your local synchronized registry.

HM Control Plane + Data Plane

Sitting alongside the HM CP is the HM Data Plane (DP). This is where your actual database workloads reside.

- Postgres clusters: The actual database instances (Primary and Standbys).
- Extensions: PostGIS, PGVector, and other database extensions.
- Backup agents: Local tools (like Barman) managing WAL archiving to your Object Storage.

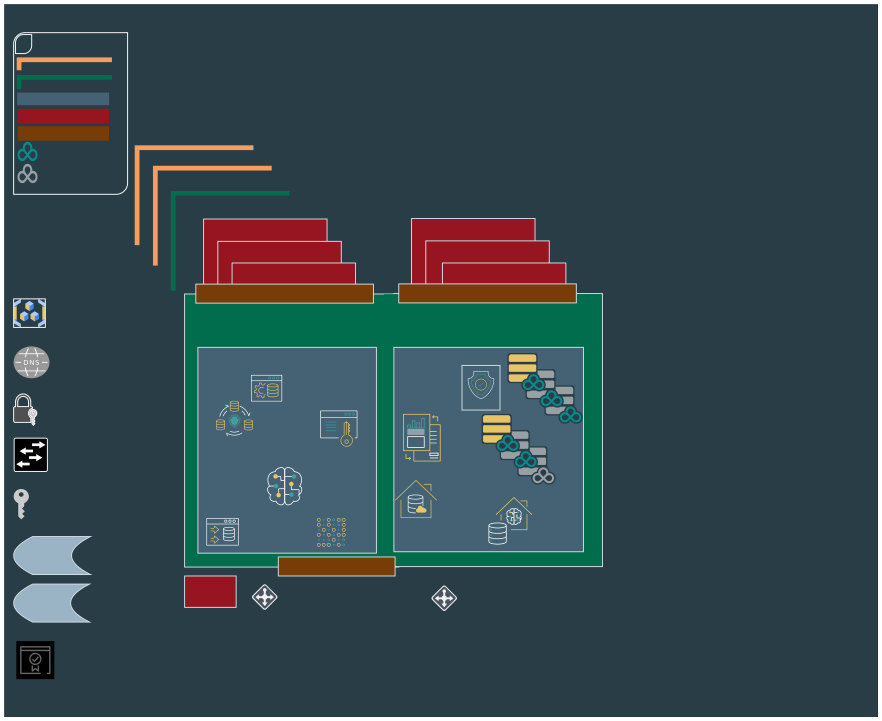


Fig. 3: HM Data Plane

HM fully featured deployment

This view shows a fully capable HM deployment, including resources like GPU acceleration for AI workloads.

Multi-location topologies in EDB Hybrid Manager

EDB Hybrid Manager supports multi-location deployments through two distinct but complementary network topologies — one for the Control Plane and one for the Data Plane. Each topology reflects the role and protocol of its plane.

The multi-location capability is a DBaaS offering following a hub and spoke model.

1. As a DBaaS offering, secondary HMs have a reduced capability set compared to the primary.
2. The primary HM controls the Secondary.
3. Connectivity is established using load-balanced endpoints; not a network mesh service like Submariner.

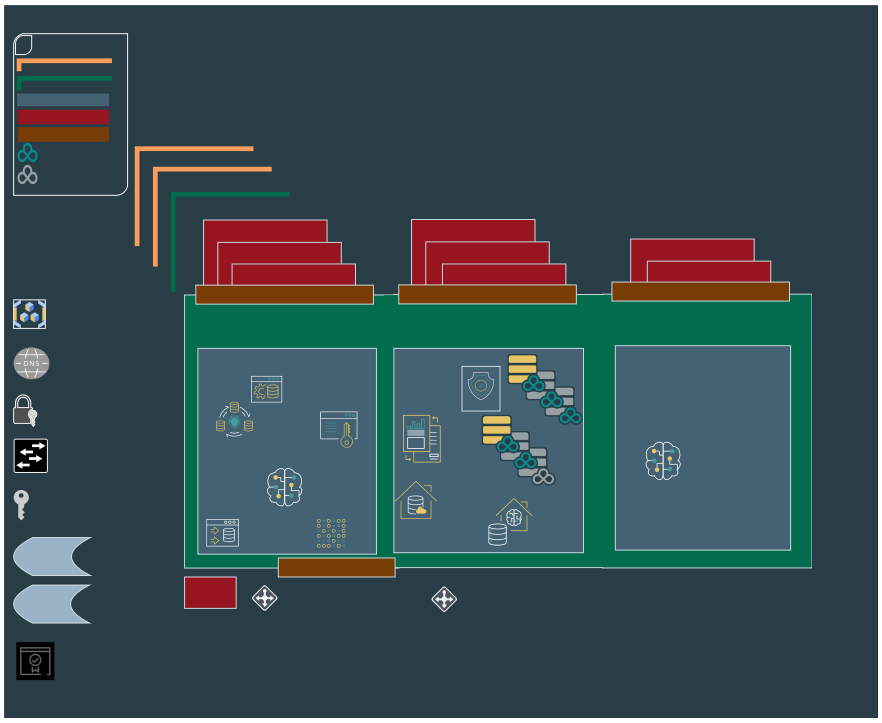


Fig. 4: HM fully featured

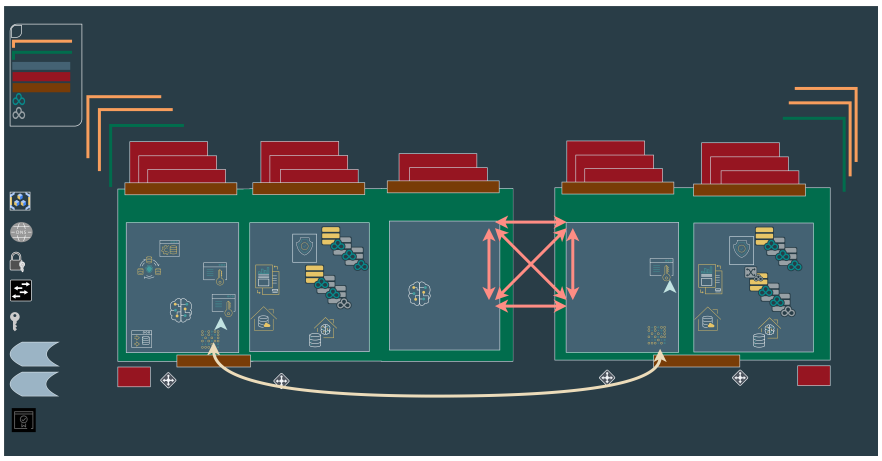


Fig. 5: HM multi-location

Control Plane: star topology

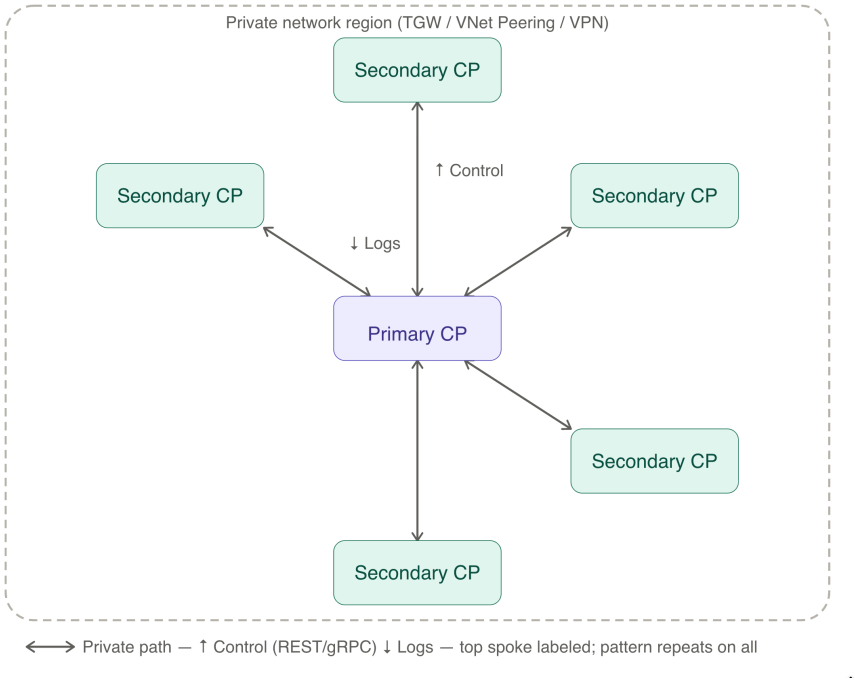


Fig. 6: CP star topology

The Control Plane is arranged in a **star topology**: one designated Primary CP sits at the center and up to five Secondary CPs radiate outward from it.

Traffic follows a strict directional pattern. **Control flows outward** from the Primary to each Secondary — configuration, orchestration commands, and lifecycle operations all originate at the hub. **Logs flow inward**, returning from each Secondary back to the Primary. Secondaries have no lateral visibility: they can't control one another, and they can't self-govern. All authority is centralized in the Primary.

This hub-and-spoke authority model has an important consequence. Because each Secondary CP is managed by the Primary but is otherwise autonomous in terms of what workloads it hosts, any Secondary can serve an entirely different engagement. Common examples include:

- **Dev/test vs. production** — separate Secondaries enforce environment isolation at the infrastructure level
- **One customer vs. another** — multi-tenant or MSP deployments where workload separation is a contractual or security requirement
- **On-premises vs. cloud** — a single Primary can govern CPs deployed in a private data center alongside CPs deployed in one or more CSPs
- **Geo-distributed regions** — each Secondary can reside in a different geographic region, managed from a single operational center of gravity The star is connected exclusively over **private network paths** — internal Load-Balancer endpoints accessed via VPC/VNet peering, Transit Gateway, VPN, or equivalent. No Secondary is reachable over a public endpoint.

Data Plane: optional mesh topology

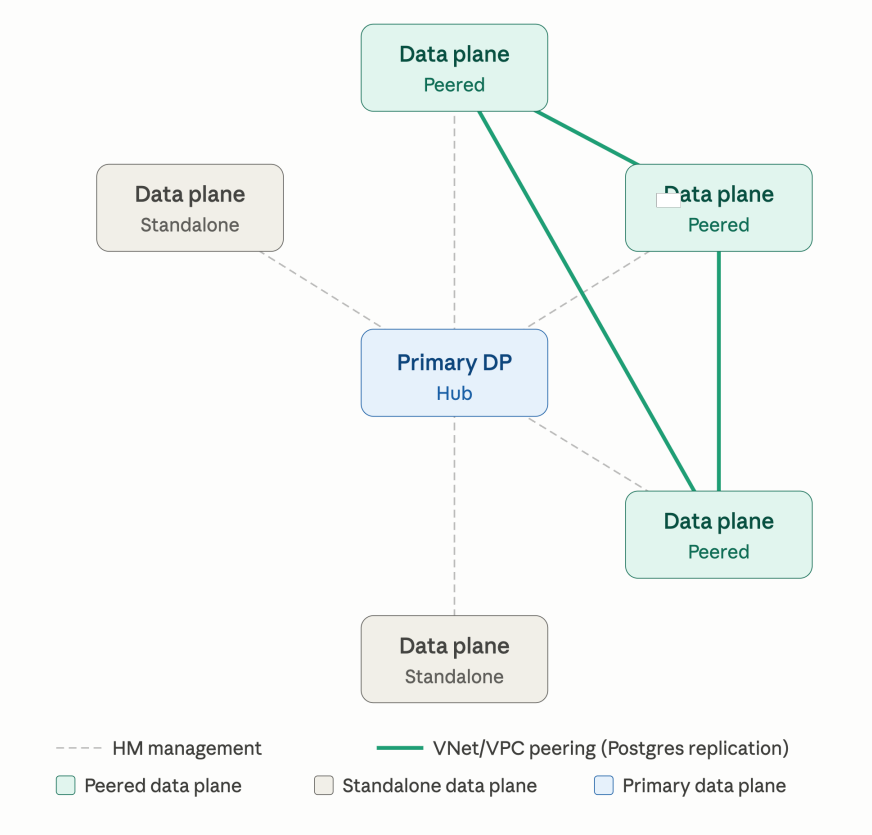


Fig. 7: DP optional mesh topology

The Data Plane topology is **optional mesh**: any two Data Plane locations may be left entirely isolated, or they may be directly interconnected via VPC/VNet peering for Postgres-level replication. There's no hub, no mandatory center, and no required connectivity between locations.

When two Data Planes are peered, the network path enables **direct Postgres replication** between clusters hosted in each location. This is the foundation for two distributed Postgres topologies supported by EDB:

- **EDB Postgres Distributed (PGD)** — active/active multi-writer replication across locations, suitable for applications requiring zero RPO and near-zero RTO across regions. See the PGD documentation .
- **CloudNativePG distributed** — streaming replication across Kubernetes clusters and failure domains using CloudNativePG's distributed topology. See the CloudNativePG distributed documentation . Data Planes that aren't peered remain fully independent — separate security domains, separate failure domains, separate upgrade timelines. Peering is additive: it introduces a replication path without changing the isolation characteristics of either plane for any other purpose.

The optional nature of the mesh means the Data Plane topology is shaped entirely by the customer's **replication and availability requirements**, not by any architectural mandate from HM itself.

Relationship between the two topologies

The CP star and DP mesh are independent overlays on the same set of physical or cloud locations. A location can host a Secondary CP, a Data Plane, or both. The CP star governs what is deployed and how it is managed; the DP mesh governs how Postgres data moves between locations. Neither topology constrains the other.

	Control Plane	Data Plane
Shape	Star — mandatory hub	Optional mesh — peer-to-peer
Hub required	Yes — Primary CP	No
Cross-node traffic	Control out, Logs in	Postgres replication (when peered)
Isolation model	Secondaries are independent engagement boundaries	Each DP is an independent failure and security domain
Network requirement	Private path to Primary required for all Secondaries	Private path only between peered DPs

Choosing an installation scenario

HM provides a comprehensive suite of capabilities by default. To meet strict security standards and organizational governance, HM supports a modular installation through "scenarios". This allows you to deploy a curated subset of features, effectively reducing the software footprint for security audits and streamlining the UI by removing unauthorized or unlicensed components.

!!!note

Scenarios are intended for advanced production deployment planning. For pilots, proof of concepts (PoCs), and initial evaluations, We recommend a full installation to maintain maximum optionality and ensure all integrated capabilities are available for testing. If this is the case, omit the `scenarios` configuration parameter, as this enables all available scenarios by default.

The four available scenarios are:

Module	Description	Included capabilities
core	**Required** . The foundational layer for all deployments.	Estate management, Observability, DBaaS provisioning, and system services (Cert-manager, Istio, etc.).
migration	Enables schema and data migration tools.	Migration Portal integration, Data Migration Service (DMS), and other migration services.
analytics	Large-scale data processing and cataloging.	Data Catalog.
ai	Tools for building and serving generative AI.	Sovereign AI, model serving (kserve), Langflow, and GenAI builders.
dbaas	Database clusters management.	Postgres cluster management, including provisioning, scaling, and updates.

!!!Note

From 2026.4.0 onwards, the `dbaas` , database cluster management capabilities are now controlled by a dedicated `dbaas` scenario, separated from the `core` scenario. Existing installations upgrading from 2026.3 *must* explicitly add `dbaas` to their scenarios list to retain Postgrescluster management functionality.

When choosing which scenarios to install, consider the following architectural and operational factors:

Functional footprint: While the baseline resource usage is similar across scenarios, each scenario introduces specific services and endpoints. Limiting your installation to required scenarios simplifies security audits and reduces the "attack surface" of your production environment.

UI and feature governance: The HM console dynamically hides navigation links and tools (such as GenAI Builder or Data Catalog) for disabled scenarios. This ensures users only interact with authorized and licensed capabilities.

Default behavior: If the `scenarios` parameter is omitted from your configuration, the system installs all scenarios by default to maintain backward compatibility.

Decide whether your environment requires a full installation or a targeted subset.

!!!Important

Currently, modifying installation scenarios after deployment isn't supported. Ensure you install all required capabilities, as you can't add or remove them once HM is deployed.

The decisions made during this discovery process directly determine the some of the root parameters of your installation configuration.

While you do not need to create the file yet, your **Architecture Decision Record** should specify the values for these keys. The **SRE/Admin** builds on these inputs, by either recording them or beginning/continuing the configuration file in *Phase 2: Gathering your system requirements* and/or uses these values to build the configuration file in ** *Phase 4: Preparing your environment* **.

Values needed for the configuration

The information and field values you gather in Phases 1–2 are recorded in a `HybridControlPlane` Custom Resource manifest, used with the `edb-hcp-operator` helm chart.

Configuration details

Architecture decision	HybridControlPlane CR field	Example value
Kubernetes Platform	*spec.flavour*	*aks*, *eks*, *gke*, *rhos*
Target location	*spec.componentsParameters.upm-beacon.beacon_location_id*	*aws-us-east-1*
Provisioning mode	*spec.beaconAgent.provisioning.provider*	*azure*, *aws*, or *gcp*
Installation scenarios	*spec.scenarios* (YAML list)	*core*, *migration*, *ai*, *analytics*, *dbaas*

Impact on configuration file

The following YAML snippet shows how your architectural decisions above map to the final `HybridControlPlane` CR structure.

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
```

(continues on next page)

(continued from previous page)

```
  flavour: <Kubernetes_Flavor> # for example, rhos, rke2, aks, eks, gke
  imageRegistry: <Container_Registry_Domain>/pgai-platform
  version: <Version>
  scenarios: # Omit to install all scenarios by default
    - core
    - migration # remove if not needed
    - ai        # remove if not needed
    - analytics # remove if not needed
    - dbaas     # remove if not needed
  componentsParameters:
    upm-beacon:
      beacon_location_id: <Deployment_Location_Name>
  beaconAgent:
    provisioning:
      provider: <Provider_Name> # Azure, AWS, GCP
      openshift: <Boolean_Value> # Defaults to false, set to true if deploying on RHOS
```

Next phase

Your architecture is defined and ideally recorded in an ADR for reference.

Proceed to :ref:`Phase 2: Gathering your system requirements → <Gathering your system requirements>` to verify that your infrastructure can match your design in your ADR.

4.1.1.8.2 Gathering your system requirements

Overview

Role focus: Infrastructure engineer / CSP Administrator / Platform Engineer

Prerequisites

- *Phase 1: Planning your architecture* (Completed)

Outcomes

- *A validated high-level inventory of compute, network, and storage infrastructure to be used in Phase 3: *Phase 3: Deploying your Kubernetes cluster* *
- *A number of requirements validated as ready for use in Phase 4 :ref:`Preparing your environment <Preparing your environment>` and initial inputs for the HM configuration — a ``HybridControlPlane`` Custom Resource manifest.*

> **Note**
>
> EDB's Sales Engineering and Professional Services are the primary resources for communicating and clarifying detailed deployment requirements during the sales cycle or via a Statement of Work (SoW). The official documentation also provides comprehensive checklists. !!!

Next phase: Phase 3: *Phase 3: Deploying your Kubernetes cluster*

Connection to architecture

The requirements listed below for Phase 2 are not all generic; many are direct consequences of the decisions made in ** *Phase 1: Planning your architecture* **:

- **Locality decisions** determine your latency requirements between the Control Plane and Data Plane.
- **Disaster recovery goals** determine the need for specific Object Storage configurations (for replication).
- **Activeness (Active/Active)** determines if you need the specialized networking required for Postgres Distributed.

Important

While this document serves as your comprehensive technical checklist, **EDB Professional Services** is the primary resource for validating complex deployment architectures. If your "Target state" from Phase 1 involves multi-region redundancy or high-scale requirements, we recommend clarifying these detailed requirements via a Statement of Work (SoW) before procuring infrastructure. !!!

Deployment readiness checklist

Use this checklist to verify at a high level that you have all necessary infrastructure components available.

- 1. :ref:` <#1-management-workstation-bastion-host>`
- 2. :ref:` <EDB Platform>`
- 3. :ref:` <#3-compute-node-requirements>`
- 3.1 *3.1 Control plane nodes*
- 3.2 *3.2 Data plane nodes*
- 3.3 *3.3 AI model nodes*
- 4. :ref:` <#4-local-networking>`
- 5. :ref:` <#5-ingress>`
- 5.1 *5.1 Load balancer controller*
- 5.2 Domain Name Service
- 5.3 Portal certificate management/TLS
- 6. :ref:` <EDB Database>`
- 6.1 *6.1 Volume snapshots*
- 6.2 *6.2 Excluding storage from Hybrid Manager discovery*
- 7. :ref:` <#7-object-storage>`
- 8. :ref:` <#8-container-registry>`
- 9. :ref:` <#9-identity-provider-idp>`
- 10. :ref:` <#10-key-management-service-kms>`

1. Management workstation (Bastion host)

Optionality: Strict requirement.

You require a designated machine to orchestrate the deployment. This can be a laptop (for public clusters) or a cloud-based Bastion host (for private clusters).

> **Note**
>
> A **Bastion host** (or jump host/box) is strongly recommended for secure, private access to the Kubernetes API, Portal, and PostgreSQL endpoints. This host serves as your dedicated operational workstation for installation and validation activities. For EDB Remote DBA (RDBA) or Managed Services contracts, a dedicated Bastion is **mandatory**. !!!

General requirements

- **Operating System:** Linux (AMD64/ARM64) or macOS. * *Note:* Windows users must use **WSL2** (Windows Subsystem for Linux).
- **Network Access: * Kubernetes API:** Must have network connectivity to the Cluster API Server (typically port 6443). * **Internet/Registry:** Must be able to reach the Container Registry to pull Helm charts and images.
- **Storage:** Minimal (enough to hold configuration files and certificates).

Required tooling inventory

You do not need to install these tools in this phase. However, you must ensure your workstation environment allows for the installation and execution of the following binaries in the next phase (Phase 3):

Core installation tools

- `edbctl` (EDB Hybrid Manager CLI)
- `kubectl` (Kubernetes CLI) + `cnpg` plugin
- `helm` (Kubernetes package manager)
- `yq` (Command-line YAML processor)

Utilities

- `curl` (Download utility)
- `openssl` (Certificate management)
- `htpasswd` (Required only if using Static Users)

Platform CLIs (Environment dependent)

- `aws` (AWS CLI)
- `gcloud` (Google Cloud CLI)
- `oc` (OpenShift CLI)

Operational tools (Post-install)

- `pgdcli` (Postgres Distributed management)
- `sentinel` (Monitoring and failover management)

2. Kubernetes platform verification

Ensure the cluster you provision matches the platform selected in the ** *Phase 1: Planning your architecture* ** phase.

Supported distributions: * Cloud service provider platforms: * Microsoft Azure AKS (IR release stream only) * Amazon EKS * Google GKE * ROSA RedHat OpenShift Service on AWS (IR release stream only)

- On premises platforms: * SUSE Rancher RKE2 * OCP RedHat OpenShift (RHOS)

For a more detailed overview of HM compatibility per LTS/IR version, see

Hybrid Manager platform compatibility .

Provisioning constraints:

- **Dedicated cluster:** The cluster must be dedicated to Hybrid Manager. Multi-tenanting with other workloads is **not supported**.
- **Relationship:** 1:1 (One HM deployment per one Cluster).
- **Lifecycle:** You are responsible for the provisioning, upgrading, and scaling of the Kubernetes layer.

Configuration dependencies

Your choice of platform dictates specific configuration values.

Record these now for the next phase:

Parameter	Custom Resource key	Required Value
System flavor	*spec.flavour*	*eks*, *gke*, *aks*, *rke2*, or *rhos*
OpenShift	*spec.beaconAgent.provisioning.openshift*	Set to *true* **only** if using Red Hat OpenShift.
OpenShift ingress domain	*spec.globalParameters.default_ingress_domain*	Required only for *rhos* (query via *oc get ingresses.config/cluster -o jsonpath={.spec.domain}*).
OpenShift console domain	*spec.componentsParameters.upm-istio-gateway.openshift_console_domain_nan*	Required only for OpenShift; used for console linking and UX.

3. Compute (Node requirements)

Ensure your worker nodes meet the minimum sizing for your intended deployment topology (Minimum vs. Fully featured) as defined in the ** *Planning Architecture* ** phase.

Hybrid Manager components require **AMD64/x86-64 nodes**. Mixed-architecture clusters (ARM64 + AMD64) are not supported for HM, even though PostgreSQL itself can run on ARM64.

Node roles summary

Node type	Purpose	Count	Kubernetes nodes	Required label
Control plane	Runs HM control plane & telemetry.	3+	Control or Worker	*edbaiplatform.io/control-plane: "true"*

continues on next page

Table 6 – continued from previous page

Node type	Purpose	Count	Kubernetes nodes	Required label
Data plane	Runs Postgres databases.	0 or 3+	Worker	*edbaiplatform.io/postgres: "true"*
AI model (GPU)	*Optional:* Runs AI/ML workloads.	0 or 2+	Worker	*nvidia.com/gpu: "true"*

Use Node Pools (e.g., AWS NodeGroups, RHOS Machine Sets) to manage resources and apply required labels/taints.

Note

The HM system databases — the core platform database and the databases backing the Beacon Server, Migration Portal, and Transporter (the DMS service) — run on control-plane nodes, not the data plane. The data-plane pool hosts only the user-provisioned Postgres clusters. !!!

In most cases, HM runs on Kubernetes worker nodes in cloud environments (since you do not have access to managed control nodes—see EKS control plane access), and can run on control nodes (see Kubernetes control plane documentation) or worker nodes for on-premises clusters.

3.1 Control plane nodes

Optionality: Strict requirement.

Resource requirements for the control plane increase with the number of databases monitored. (Note: The Telemetry stack scales with the monitoring load).

Sizing guidelines:

Resource	Minimum (Up to 10 DBs)	Standard (10–50 DBs)	Large (>50 DBs)
CPU	8 vCPUs	16 vCPUs	16+ vCPUs
Memory	32 GB RAM	64 GB RAM	64+ GB RAM
Disk	100 GB SSD	200 GB SSD	>200 GB SSD
Quantity	3 nodes	3 nodes	3+ nodes
Max pods/node	≥ 60	≥ 60	≥ 60

Pod density (``maxPods``) — easy to miss, and not a CPU/memory problem. The control-plane stack is pod-dense: in addition to telemetry and the service mesh, the **HM system databases run on control-plane nodes** (see *Node roles summary*). An all-scenario install places **~150 pods** on the control-plane pool, which across the minimum 3 nodes is ~50 pods/node — so each control-plane node must allow **at least 60 pods**.

This is independent of CPU and memory: a node can have spare CPU/RAM and still reject pods once it hits its pod cap. On **AKS the Azure CNI default is ``maxPods=30``** , which is too low — control-plane-pinned pods (for example `upm-beacon-server`) stay `Pending` with *"Too many pods"* and the install stalls. Set the limit **at node-pool creation time** (`az aks nodepool add … --max-pods 110` ; you can't change it afterward) or use Azure CNI Overlay (default 250). GKE (default 110) and EKS (per-ENI IP limits) generally have enough headroom, but verify against the ~60-pods/node figure above.

3.2 Data plane nodes

Optionality: Strict requirement.

The data plane nodes must be sized to host the **PostgreSQL** database clusters provisioned by users.

- **Sizing:** Depends entirely on expected database workloads, storage requirements, and performance SLAs as defined in *Phase 1 - Pipeline options* .

3.3 AI model nodes

Optionality: Required only if utilizing GenAI/Agent Factory capabilities (Installation scenario `ai` is enabled).

- **Sizing:** Current recommendations require **Nvidia B200 GPUs** (or equivalent supported hardware).

Node abstractions

To ensure the HM Control Plane runs on dedicated resources and Postgres workloads are on independent resources, or to partition dedicated resources for your AI workloads, you must apply specific configurations to each of these node pools before *Phase 3: Deploying your Kubernetes cluster* .

Node allocatable resources requirements

Hybrid Manager components have specific resource requirements that the nodes they're scheduled on must satisfy. When defining your node pools, ensure the allocatable resources meet the minimum requirements for the Hybrid Manager components and Postgres clusters you plan to run. Each scenario corresponds to a set of Hybrid Manager components and has its own resource requirements:

Scenario	Total CPU	Total Memory	Total Storage	Total Ephemeral Storage
ai	3366m	4749Mi	4Gi	250Mi
analytics	200m	456Mi	0	0
core	10020m	30445Mi	454706Mi	3Gi
dbaas	0	0	0	0
migration	2880m	6432Mi	54Gi	0

In addition, the Hybrid Manager operator itself requires `260m CPU` and `576Mi memory` .

4. Local networking

Local network requirements

HM is not strictly opinionated about local networking logic. The standard networking capabilities provided by major Cloud Service Providers (AWS, Azure, GCP) are sufficient, provided they offer **low latency and high bandwidth** between availability zones.

- **On-premises:** Networking should follow best practices regarding switching, link bonding, and link redundancy to ensure high availability.
- **Protocol: IPv4 only**. Kubernetes must be configured for IPv4. IPv6 has not been thoroughly productized for Hybrid Manager; if IPv6 is required, it must be masked behind a Load Balancer so that internal communication remains IPv4.

- **Defined Network Address Spaces (CIDRs)**: Your cluster must be configured with distinct, non-overlapping IP ranges for: * Pod Network (clusterCIDR): The IP range from which all pods are assigned their IPs. * Service Network (serviceCIDR): The IP range from which all internal services (ClusterIPs) are assigned their virtual IPs. * Functional DNS and NTP: The cluster's internal DNS service (typically CoreDNS) must be running and able to resolve both internal cluster services (e.g. kubernetes.default) and external addresses.

Container Network Interface (CNI)

A functional CNI is a strict requirement for pod-to-pod networking.

- **Examples**: Calico, Cilium, Flannel
- **Context:** For the **data plane**, database I/O — not pod density — is the bottleneck, so CNI pod-density tuning rarely matters there. The **control plane** is different: it is pod-dense, so your CNI and per-node `maxPods` setting must allow at least 60 pods per control-plane node. See *3.1 Control plane nodes* for platform-specific guidance on setting `maxPods` . Cilium is more efficient at very high pod counts if you need it.
- **Validation:** Verify CNI setup and CIDR assignment:

```
kubectl get nodes -o custom-columns=NAME:.metadata.name,PODCIDR:.spec.podCIDR
kubectl get svc
```

5. Ingress

You must choose an ingress strategy that matches your infrastructure capabilities. HM can deploy either a load balancer or NodePort service for its own Portal and services, as well as individual load balancers and NodePorts for each Postgres database you create. Choose an ingress strategy that matches your infrastructure capabilities. HM can deploy either a load balancer or NodePort service for its own portal and services. It can also deploy individual load balancers and NodePorts for each Postgres database you create. The table below defines the supported combinations of DNS Management, Endpoint Types, and SSL Termination.

Scenario	DNS Management	Endpoint Type	Portal SSL/TLS Termination
CSP LB (ELB, ALB)	Manual portal, dynamic postgres	Dynamic via Load Balancer Controller	Istio Ingress Gateway
F5 + F5 K8s Controller	Manual portal, dynamic postgres	Dynamic via Load Balancer Controller	Istio Ingress Gateway
MetalLB	Manual	Dynamic via Load Balancer Controller	Istio Ingress Gateway
NodePort (Conventional LB)	Manual (A record against LB)	Static (manual)	Istio Ingress Gateway
NodePort (No LB)	Manual (Round Robin against Nodes)	Static (manual)	Istio Ingress Gateway

5.1 Load balancer controller

Optionality: Required option for an optimal experience.

Examples of load balancer controllers range from the AWS Load Balancer controller to MetalLB or a BigIP F5 combined with DNS capability and their F5 Load Balancer Controller.

General requirements

- **Firewall Annotations:** If using a specific provider scheme (e.g., AWS Internet Facing), you know the correct `resourceAnnotations` to provision the load balancer correctly:

- AWS example annotation*:

```
globalParameters:
  load_balancer_mode: public
  load_balancer_provider: aws
resourceAnnotations:
  - name: istio-ingressgateway
    kind: Service
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
```

- MetalLB example annotation*:

```
globalParameters:
  load_balancer_mode: custom-only
resourceAnnotations:
  - name: istio-ingressgateway
    kind: Service
    annotations:
      metallb.io/address-pool: production-public-ips
```

Configuration dependencies

Parameter	Custom Resource key	Value / Format
Enable LB	*spec.globalParameters.load_balancer_mode*	*"public"*, *"private"*, or *"disabled"*
Provider	*spec.globalParameters.load_balancer_provider*	*aws*, *azure*, or *gcp*
Annotations	*spec.resourceAnnotations[]*	Provider-specific list (see below).

The `load_balancer_mode` parameter lets you instruct HM to create a load balancer service for both the Portal and the Postgres databases. If set to `disabled` , both the portal and the Postgres databases use NodePorts; otherwise, they use a load balancer. The `public` and `private` values only affect the Postgres databases and have no impact on the portal load balancer. Similarly, the `load_balancer_provider` parameter applies only to Postgres clusters. The `resourceAnnotations` parameter is relevant only for the HM Portal.

Spot validation

To verify that your controller is active, use the following command:

```
kubectl get pods -A | grep -i load\|lb\|router\|metallb\|gateway
```

HM control plane ports

The following ports must be open on your load balancer or firewall for Hybrid Manager to function:

Port	Protocol	Service	Description
80	TCP	*http2*	HM Portal (HTTP — redirects to HTTPS)
443	TCP	*https*	HM Portal (HTTPS ingress)
9443	TCP	*grpc-tls*	Beacon gRPC API
8444	TCP	*spire-tls*	Spire TLS

continues on next page

Table 11 – continued from previous page

Port	Protocol	Service	Description
9445	TCP	*beacon-spire*	Beacon Spire TLS
9198	TCP	*kafka*	Kafka (required for the migrations scenario)
24224	TCP	*fluent-bit*	Inter-cluster log forwarding (multi-DC only)
10901	TCP	*thanos-query*	Inter-cluster metrics (multi-DC only)

Postgres ports (Data Plane)

With a proper load balancer setup, the following are Postgres ports:

Port	Protocol	Description
5432	TCP	Default PSQL
6432	TCP	PGD Connection Manager PSQL

SSL Termination

HM is exposed through the Istio Gateway by using TLS. If you are using a load balancer in front of Istio to access HM, you have two options:

1. **SSL passthrough**: Configure the load balancer to forward TCP traffic without modification.
2. **TLS termination**: Terminate the TLS session from the client at the load balancer, then have the load balancer establish a new TLS connection with Istio.

5.1.2 NodePort alternative

Optionality: Alternative to *5.1 Load balancer controller* .

If a Load Balancer is not available, you must use a **NodePort** strategy.

- **Configuration:** You must configure `spec.globalParameters.load_balancer_mode: "disabled"` and define a `spec.globalParameters.node_port_domain` in your HM configuration.
- **Context:** By default, HM components expose services on specific NodePort values. You may use NodePort directly, or front these ports with **MetalLB** (software load balancer) or a **hardware load balancer** for a friendlier DNS name and better failover.

Default NodePort assignments

If using NodePort for ingress, the following specific ports must use **NodePort**.

- **Constraint:** You must define a `node_port_domain` (Base DNS name).
- **Context:** HM components expose services on specific NodePort values. You may use NodePort directly, or front these ports with a hardware load balancer.

Configuration dependencies

Parameter	Custom Resource key	Value / Format
Disable LB	*spec.globalParameters.load_balancer_mode*	*"disabled"*
Domain	*spec.globalParameters.node_port_domain*	Postgres DNS domain (e.g., *pg.myorg.com*).

Default NodePort assignments

NodePort variable	Default port	LB port	Description
spec.componentsParameters.upm-istio-gateway.ingre	**32542**	80	HM Portal (HTTP)
spec.componentsParameters.upm-istio-gateway.ingre	**30288**	443	HM Portal (HTTPS)
spec.componentsParameters.upm-istio-gateway.ingre	**30290**	9443	Beacon gRPC TLS
spec.componentsParameters.upm-istio-gateway.ingre	**30292**	8444	Spire TLS
spec.componentsParameters.upm-istio-gateway.ingre	**30294**	9445	Beacon Spire TLS
spec.componentsParameters.upm-istio-gateway.ingre	**30296**	10901	Inter-cluster metrics (multi-DC only)
spec.componentsParameters.upm-istio-gateway.ingre	**30298**	24224	Inter-cluster log forwarding (multi-DC only)
spec.componentsParameters.upm-istio-gateway.enab	n/a	n/a	Enable server stored session (*"true"* or *"false"*)

Postgres NodePorts

Postgres clusters use the `node_port_domain` and increment on the port, provisioned in the port range `30000+` .

5.2 DNS

Optionality: DNS configuration is a **required option for an optimal experience**.

You must ensure proper DNS resolution both inside the cluster and for external access.

Internal cluster DNS & NTP

- **CoreDNS:** The cluster's internal DNS service (typically CoreDNS) must be running and able to resolve both internal cluster services (e.g., `kubernetes.default`) and external internet addresses.
- **NTP:** Functional NTP is required to ensure time synchronization across nodes.
- **Container Registry:** Your DNS service must be able to resolve the DNS entries for the container registry (whether local or EDB's).

Configuration dependencies

You must provision and control the following domains. These are inputs into your HM configuration in the next phase:

Domain Purpose	Custom Resource key	Description
Portal domain	*spec.globalParameters.portal_domain_name*	The host name for the HM Portal UI.
Agent domain	*spec.componentsParameters.upm-beacon.server_host*	The host name through which the Beacon Server API is reachable.

continues on next page

Table 15 – continued from previous page

Domain Purpose	Custom Resource key	Description
Migration domain	*spec.globalParameters.dms_domain_name*	The internal domain for the Transporter migration service. Required unless the *migration* installation scenario isn't enabled.
NodePort base	*spec.globalParameters.node_port_domain*	**(Scenario B Only)** The base domain for Postgres access if using NodePort.

In case of NodePort approach, the above DNS entries should be round robin on the compute IP where the control-plane label is applied on the Kubernetes Nodes.

DNS configuration strategy

Your DNS configuration depends on your Ingress selection.

Scenario A: Load balancer (recommended)

- **Dependency:** Ensure `spec.globalParameters.load_balancer_mode:` is set to `"public"` or `"private"` in your HM configuration.
- **Portal:** In the case of an optimal load balancer controller like AWS ELB or an F5 with DNS capability, manually configure DNS to point to the resulting Load Balancer IP for `istio-ingress` . Domain names must be resolvable to a **locally routable IP** so that the ingress service on Kubernetes can properly route traffic according to the hostname in the HTTPS request.
- **Postgres:** You do *not* need to manually manage DNS for every database.

Postgres DNS records are populated automatically as a function of the load balancer controller updating the status of the Service in the namespace dedicated to that Postgres cluster.

Scenario B: NodePort alternative

- **Prerequisite:** You disabled load balancers and defined a `node_port_domain` in the Ingress section.
- **Portal:** DNS entries should be a **Round Robin A Record** pointing to the compute IPs of the **Control Plane nodes** (nodes labeled `edbaiplatform.io/control-plane`).
- **Postgres:** The `node_port_domain` value is used as the base URL for all Postgres instances. It should be a **Round Robin DNS record** pointing to the IP addresses of the **Worker nodes** where the Postgres clusters are running.

5.3 Certificate management

Optionality: Strictly required.

Secure communication requires TLS certificates for the Portal and API endpoints. You must decide on a certificate management strategy before installation.

Decision matrix

Option	Description	Typical use case
A. Custom cert-manager issuer	Use an existing *cert-manager* Issuer.	Production (Recommended).
B. Customer CA	Bring your own CA to sign internal certs.	Enterprises with internal PKI.

continues on next page

Table 16 – continued from previous page

Option	Description	Typical use case
C. Customer Certificate	Provide your own pre-generated x.509 cert and private key as a Kubernetes secret.	Specific organization-issued cert.
D. Self-signed	Installer generates self-signed certs.	**Test/Non-Production only.**

Configuration dependencies by strategy

Strategy	Custom Resource key	Description
A. Cert Manager	*spec.globalParameters.portal_certificate_issuer_kind*	**Required.** The Resource Kind (*Issuer* or *ClusterIssuer*).
A. Cert Manager	*spec.globalParameters.portal_certificate_issuer_name*	**Required.** The Resource Name (e.g., *letsencrypt-prod*).
B. BYO CA	*spec.globalParameters.ca_secret_name*	**Required.** The name of your CA Secret.
C. BYO Cert	*spec.globalParameters.portal_certificate_secret*	**Required.** The name of your Certificate Secret.
D. Self-signed	N/A	**Default behavior.** (No configuration required).

6. Block storage (Database & logs)

Optionality: Strictly required.

You must define a Kubernetes `StorageClass` that presents persistent storage volumes (PVCs).

The default storage class used for the **HM Control Plane** is set in your HM configuration. The same storage class is an option for all Postgres instances' deployments as well. However, for the **Data Plane** (your Postgres deployments), you may establish any number of additional storage classes in the cluster. All of them are available options for Postgres deployments, providing the flexibility needed for critical or intensive workloads.

Control plane requirements

Block storage demands for the HM Control Plane are relatively light. A storage class roughly equivalent to the capability of **AWS gp2** is sufficient.

Data plane requirements (Postgres)

Postgres workloads may require specialized configurations depending on their specific I/O patterns.

- **Latency Sensitivity:** Postgres is generally more sensitive to latency—and specifically the *consistency* of latency—than raw throughput.
- **Example: Analytics:** Analytic capabilities perform as expected with approximately **16,000 IOPS** capacity per PVC.
- **Example: Azure:** In Azure, IOPS-capable storage (Premium SSD) is highly preferred—even when provisioning low IOPS capacity—due to its highly consistent latency compared to Standard tiers.

Notable considerations

Hybrid Manager and containerized Postgres are particularly well suited to leverage **Local Storage**. Unlike traditional applications, these solutions do not strictly depend on the PVC being replicated across the Kubernetes cluster by the

storage provider. If a Postgres instance loses its underlying node (and local disk), the new pod does not need to remount the old PVC. Instead, the Hybrid Manager High Availability (HA) logic rebuilds the new node from its peers or backup.

Configuration dependencies

Parameter	Custom Resource key	Value / Format
Storage Class	*spec.globalParameters.storage_class*	The exact name of your K8s StorageClass.

Validation

Verify available storage classes:

```
kubectl get sc
```

6.1 Volume snapshots

- **Optionality:** Required for an optimal experience (backups will be limited without it).

The underlying storage provider **must support Volume Snapshots** to enable the backup and recovery features of the platform.

- **Mechanism:** Hybrid Manager automatically utilizes volume snapshots if the capability is exposed via the Kubernetes API.
- **Implementation note:** * **Cloud:** Typically handled by the CSP's CSI driver (e.g., AWS EBS CSI), but often requires installing a "Snapshot Controller" add-on. * **On-Premises:** Your storage backend must support the CSI Snapshotter specification, and you must define a `VolumeSnapshotClass` that maps to it.

Validation

Verify that a snapshot class is configured and available:

```
kubectl get volumesnapshotclass
```

6.2 Excluding storage from Hybrid Manager discovery

In shared Kubernetes environments, some `StorageClasses` or `VolumeSnapshotClasses` may not be suitable for Postgres workloads — for example, GPU- or HM-only classes. By default, every discovered class appears in the Hybrid Manager **Create Cluster** dropdowns, which can lead operators to pick a class on which a Postgres cluster will never successfully deploy.

To hide a class from the Hybrid Manager provisioning UI, apply the `beacon.enterprisedb.com/exclude-storage-from-dbaas` label with value `true` :

```
kubectl label storageclass <name> beacon.enterprisedb.com/exclude-storage-from-
↪dbaas=true
kubectl label volumesnapshotclass <name> beacon.enterprisedb.com/exclude-storage-from-
↪dbaas=true
```

Behavior:

- **Opt-in:** Only the exact value `"true"` excludes a class. An absent label, or any other value, preserves current behavior — the feature is a no-op for existing deployments.

- **Default-safe:** An excluded `StorageClass` is never promoted to Hybrid Manager's discovered default, even if it carries the `storageclass.kubernetes.io/is-default-class` annotation.
- **Takes effect on the next agent heartbeat:** No agent restart is required.
- **UI-only:** The label hides the class from Hybrid Manager's provisioning UI. Workloads that reference the class by name — for example, a custom resource or a direct `kubectl apply` — can still use it.

To restore a class to the provisioning UI, remove the label:

```
kubectl label storageclass <name> beacon.enterprisedb.com/exclude-storage-from-dbaas-
kubectl label volumesnapshotclass <name> beacon.enterprisedb.com/exclude-storage-from-
↪dbaas-
```

Validation

Audit which classes are currently excluded:

```
kubectl get sc,volumesnapshotclass -l beacon.enterprisedb.com/exclude-storage-from-
↪dbaas=true
```

7. Object storage

Optionality: Strictly required.

Object storage is the backbone of disaster recovery and data mobility. It is used for:

- Velero backups (Cluster DR)
- Postgres WAL archiving (Point-in-Time Recovery)
- Unstructured data for GenAI/AIDB
- Parquet files for Lake Keeper
- Log files

General requirements

- **Protocol:** Must be fully **S3 Compatible**.
- **Consistency:** In multi-location deployments, the object storage configuration must be available equally across all locations. (See *multi-dc guidance* for details.)

Note

You must provision a Kubernetes Secret named *edb-object-storage* in the *default* namespace. The contents of this secret must be identical in every cluster. You are not expected to run *kubectl create secret* at this point in the process. Here, you are gathering the raw credentials (API keys, certificates, passwords) so you have them ready to inject into the cluster in *Phase 4: Preparing your environment* . !!!

Important

Your object storage must be dedicated entirely to the Hybrid Manager. Any prior data existing in your object storage interferes with the successful operation of the HM. Similarly, if HM is removed and reinstalled, then the object storage must be emptied prior to beginning the new installation. !!!

Authentication strategies

Choose the authentication method supported by your provider:

Provider	Auth Method	Example Use Case
AWS (EKS/ROSA)	Workload Identity (IAM)	Native AWS integration (Recommended).
AWS (Other K8s)	Static Keys	Generic Kubernetes connecting to AWS S3.
Azure Blob	Static Keys	Connect via Connection String/Key.
GCP Storage	Static Keys	Base64-encoded Service Account JSON.
S3 Compatible	Static Keys	On-Premises (MinIO, Ceph, etc.).

Configuration dependencies

Configuration file

These are the configuration keys for your HM installation.

8. Container registry

Optionality: A local container registry is a **required option for an optimal experience**.

You require a container registry that is accessible from your Kubernetes Cluster to host HM and Postgres images.

- **Purpose:** To serve the specific application images required for the Control Plane and Data Plane.
- **Access Requirement:** The cluster must be able to pull images from this registry.
- **Permissions:** The authentication credential used must have permissions to **List Repositories**, **List Tags**, and **Read Tag Manifests**.

If your environment cannot access the public EDB repository, you must pull the artifacts and push them to your local private registry using `edbctl` .

Choosing a registry approach

You have two registry options:

- **External private registry (recommended).** Use your organization's existing registry — for example, AWS ECR, Google Artifact Registry, Azure Container Registry, or Harbor. Best for environments with established registry tooling, scanning, and access control. See *Syncing images to a local private registry* .
- **In-cluster container registry.** A registry embedded in the EDB HCP Operator itself. Suitable only for fully air-gapped clusters whose nodes have no public IPs. See *Using the in-cluster container registry* .

If your nodes have public IPs, don't use the in-cluster registry. Use an external private registry instead.

Registry provider	Supported Auth	Recommended Auth
Azure (ACR)	Token, Basic	Token
Amazon (ECR)	EKS Managed Identity	EKS Managed Identity
Google (GAR)	Token, Basic	Basic
EDB Repo	Token	Token (PoC/Pilot only!)

Preparation

- **Sync images to your local container registry:** You must use `edbctl` to synchronize images if using a private registry. * ***SOP: Sync images to local registry***
  - *Guidance: Using ``edbctl`` to mirror Platform and Operator images.*
- **Secret:** You must provision a Kubernetes Secret with pull credentials. (See *Phase 4: Preparing your environment* for implementation details.)

> **Note**
>
> **Configure image discovery:** Once images are synced to your local registry, you may need to configure the installer to locate them. This ensures the cluster pulls artifacts from the correct location (local registry vs. public registry).

- ***SOP: Image discovery configuration***
  - *Guidance: Configuring registry mirrors and pull policies.*

Collect these values for your HM configuration in the next phase (Phase 3).

Parameter	Custom Resource key	Value / Format
Registry URL	*spec.imageRegistry*	*<Registry Domain>/pgai-platform*
Image version	*spec.version*	HM version tag (e.g., *"v2026.5.1"*).
Pull Secret	*spec.imagePullSecrets[]*	Kubernetes image pull secret references (for example, *[{name: edb-cred, namespace: edbpgai-bootstrap}]*).
Discovery Registry	*spec.beaconAgent.provisioning.imagesetDiscoveryContainerRegistryURL*	*<Registry Domain>/pgai-platform*
Discovery Auth	*spec.beaconAgent.provisioning.imagesetDiscoveryAuthenticationType*	*<Auth Type>* (e.g., *token*, *basic*, *eks_managed_identity*)
TLS Validation	*spec.beaconAgent.provisioning.imagesetDiscoveryAllowInsecureRegistry*	Set to *true* only if intentionally allowing insecure (no/invalid TLS) registry discovery.

9. Identity provider (IdP)

Hybrid Manager is **not intended** to provide user authentication directly. It relies on an external Identity Provider (IdP) to manage user access securely.

Optionality: Required for optimal experience and security.

- **Mechanism:** Connects to your existing directory service via OIDC.
- **Supported Backends:** LDAP or SAML.

You must prepare the following values and artifacts for *Preparing your environment* (Phase 4):

10. Key management service (KMS)

Integrating an external Key Management Service (KMS) is required to enable **Transparent Data Encryption (TDE)** for your Postgres databases.

- **Optionality:** Required for optimal security and compliance.
- **Mechanism:** Hybrid Manager integrates with cloud providers or external vaults to manage encryption keys securely.

Supported Authentication

- **Workload Identity:** Native cloud authentication (Recommended for EKS/GKE/AKS).
- **Credentials:** Static keys stored in Kubernetes secrets.

Configuration dependencies

Configuration file

You must prepare the following values for your HM configuration in *Phase 4 guide* :

Parameter	Custom Resource key	Value / Format
TDE Providers	*spec.beaconAgent.transparentDataEncryptionMethods*	List of enabled methods (e.g., *passphrase*, *aws_kms*, *azure_kms*, *gcp_kms*, *hashicorp_vault*).

- **Auth Secrets:** If using `credentials` mode, specific Kubernetes Secrets must be provisioned with API keys.

Impact on configuration

The infrastructure specifications detailed in this document, combined with your architecture decisions from Phase 1, dictate the values you use in the next phase.

Requirements inventory guide

Use this inventory guide to ensure you have every required value determined before proceeding to Phase 3.

The `HybridControlPlane` CR key is specified for each requirement.

General & platform

Source	Requirement	Custom Resource key	Condition
Phase 1	**K8s Flavor**	*spec.flavour*	**Required** (e.g., *eks*, *gke*, *aks*, *rhos*)
Phase 1	**Provider**	*spec.beaconAgent.provisioning.provider*	**Required** (e.g., *AWS*, *GCP*)
Phase 1	**OpenShift**	*spec.beaconAgent.provisioning.openshift*	**Required** if using RHOS (*true*)

continues on next page

Table 23 – continued from previous page

Source	Requirement	Custom Resource key	Condition
Phase 1	**Location**	*spec.componentsParameters.upm-beacon.beac*	**Required** (String ID)
Phase 1	**Installation scenarios**	*spec.scenarios[]*	**Optional** . All scenarios enabled by default. Customize with *core* plus any of *migration*, *ai*, *dbaas* or *analytics*.

Networking & ingress

Source	Requirement	Custom Resource key	Condition
Phase 2	**Portal Domain**	*spec.globalParameters.portal_domain_name*	**Required**
Phase 2	**Agent Domain**	*spec.componentsParameters.upm-beacon.serve*	**Required**
Phase 2	**Migration Domain**	*spec.componentsParameters.transporter-rw-se*	**Required**
Phase 2	**Migration URL**	*spec.componentsParameters.transporter-dp-ag*	**Required**
Phase 2	**Load Balancer Mode**	*spec.globalParameters.load_balancer_mode*	**Required** (*"public"*, *"private"*, or *"disabled"*)
Phase 2	**LB Annotations**	*spec.resourceAnnotations[]*	**Required** if using CSP LB (e.g., AWS)
Phase 2	**NodePort Domain**	*spec.globalParameters.node_port_domain*	Required if load balancer mode is *"disabled"*

Storage & registry

Source	Requirement	Custom Resource key	Condition
Phase 2	**Storage Class**	*spec.globalParameters.storage_class*	**Required**
Phase 2	**Registry URL**	*spec.imageRegistry*	**Required**
Phase 2	**Image Version**	*spec.version*	**Required**

continues on next page

Table 25 – continued from previous page

Source	Requirement	Custom Resource key	Condition
Phase 2	**Discovery Reg.**	*spec.beaconAgent.provisioning.imagesetDiscoveryContainerRegistryURL*	**Required**
Phase 2	**Registry Auth**	*spec.beaconAgent.provisioning.imagesetDiscoveryAuthenticationType*	**Required**
Phase 2	**Insecure TLS**	*spec.beaconAgent.provisioning.imagesetDiscoveryAllowInsecureRegistry*	Optional (if using self-signed registry)

Security & identity

Source	Requirement	Custom Resource key	Condition
Phase 2	**IDP Secret**	*spec.componentsParameters.pgai.portal.authentication.clientSecret*	**Required** (if using OIDC)
Phase 2	**IDP Config**	*spec.componentsParameters.pgai.portal.authentication.idpConnectors*	**Required** (if using OIDC)
Phase 2	**Static User**	*spec.componentsParameters.pgai.portal.authentication.staticPasswords.**	**PoC Only** (if no IdP)
Phase 2	**TDE Keys**	*spec.beaconAgent.transparentDataEncryptionMethods*	Optional (if using TDE)

Certificates (Choose one strategy)

Select **one** strategy (A, B, C, or D) and record the corresponding value(s).

Strategy	Custom Resource key	Description
A. Cert Manager	*spec.globalParameters.portal_certificate_issuer_kind*	**Required.** The Resource Kind (e.g., *ClusterIssuer*).

continues on next page

Table 27 – continued from previous page

Strategy	Custom Resource key	Description
A. Cert Manager	*spec.globalParameters.portal_certificate_issuer_name*	**Required.** The Resource Name (e.g., *letsencrypt-prod*).
B. BYO CA	*spec.globalParameters.ca_secret_name*	**Required.** The name of your CA Secret.
C. BYO Cert	*spec.globalParameters.portal_certificate_secret*	**Required.** The name of your Certificate Secret.
D. Self-signed	N/A	**Default behavior.** (No configuration required).

Example configuration

According to the decisions made in previous steps, your configuration file should reflect those choices.

Here is the configuration for a `HybridControlPlane` CR. For a complete field reference, see the *Configuration mapping reference* .

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  flavour: <Kubernetes> # e.g., eks, gke, rhos, rke2
  imageRegistry: <Container Registry Domain>/pgai-platform
  version: <Version>

  imagePullSecrets:
    - name: edb-cred
      namespace: edbpgai-bootstrap

  globalParameters:
    CONTAINER_REGISTRY_URL: <Container Registry Domain>/pgai-platform
    portal_domain_name: <Portal Domain>
    storage_class: <Block Storage>
    portal_certificate_issuer_kind: <ClusterIssuer>
    portal_certificate_issuer_name: <my-issuer>
    trust_domain: <Portal Domain>

  componentsParameters:
    upm-beacon:
      beacon_location_id: <Location>
      server_host: <Agent Domain>
    transporter-rw-service:
      domain_name: <Migration Domain>
    transporter-dp-agent:
      rw_service_url: https://<Migration Domain>/transporter

  beaconAgent:
    provisioning:
      imagesetDiscoveryAuthenticationType: <Authentication Type for the Container␣
↪Registry>
      imagesetDiscoveryContainerRegistryURL: "<Container Registry Domain>/pgai-
↪platform"
```

(continues on next page)

(continued from previous page)

```
  transparentDataEncryptionMethods:
    - <available_encryption_method>

scenarios:
  - core
  - <if required, migration>
  - <if required, ai>
  - <if required, analytics>
  - <if required, dbaas>

resourceAnnotations:
  - name: istio-ingressgateway
    kind: Service
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-scheme: internal
      service.beta.kubernetes.io/load-balancer-source-ranges: 10.0.0.0/8
```

Next phase

You now have a complete inventory and deployment strategy, as well as initial inputs for your HM configuration — a `HybridControlPlane` CR. Continue building your configuration as you go, or use these inputs to assemble the file at the end of *Phase 4: Preparing your environment* .

Proceed to :ref:`Phase 3: Deploying your Kubernetes cluster → <Deploying your Kubernetes cluster>` to provision and deploy your Kubernetes clsuter.

4.1.1.8.3 Deploying your Kubernetes cluster

Overview

Role focus: Kubernetes engineer / CSP Administrator

Prerequisites:

- *Phase 1: Planning your architecture* (Completed)
- *Phase 2: Gathering your system requirements* (Completed)

Outcomes

- *A running, reachable Kubernetes cluster that meets all compute, storage, and networking requirements for Hybrid Manager (HM).*

> **Note**
>
> **EDB support context:** Customers own the installation and lifecycle operation of their Kubernetes cluster. Professional Services can be engaged via a Statement of Work (SoW), and Support can offer assistance through knowledge base articles. !!!

Next phase: Phase 4: *Preparing your environment*

Start the provisioning process

With your system requirements defined, you are ready to provision the infrastructure. This phase covers deploying the base Kubernetes cluster that hosts the Hybrid Manager (HM) platform.

HM is largely platform-agnostic, supporting major cloud providers and on-premises distributions. For platform-specific variabilities,

Set up your management workstation

Provision and configure your management workstation (Bastion or other) as planned in the previous phase: *Phase 2: Gathering your system requirements* .

Install core tooling

You must install the following CLI tools to orchestrate the deployment.

1. **Install Kubernetes Tools (``kubectl`` & ``helm``):** Follow the official guides to install kubectl and Helm .
2. **Install ``edbctl`` (EDB Hybrid Manager CLI):** Run the installation script to download the binary:

```
curl -sfL https://get.enterprisedb.com/edbctl/install.sh | sh -
```

3. **Install utilities:**

Ensure `yq` (v4+), `curl` , and `openssl` are installed via your package manager.

```
# Example for Ubuntu/Debian
sudo apt-get update && sudo apt-get install -y curl openssl

# Example for MacOS
brew install yq curl openssl
```

4. **Install platform CLIs (Environment dependent):**

If you are deploying to a public cloud, install the relevant CLI for authentication. * **AKS:** Install Azure

```
* **AWS:**
```

Install AWS CLI

- **Google Cloud:** Install gcloud SDK
- **OpenShift:**

Install oc CLI

Verify connectivity

Before proceeding to the configuration phase, validate that your workstation can reach the necessary endpoints.

1. **Verify Internet/Registry access**

```
curl -I https://docker.enterprisedb.com/v2/
```

1. **Verify Cloud Identity (if using AWS/GCP)**

for AWS:

```
    aws sts get-caller-identity
```  or for GCP:
```
gcloud auth list
```
```
1. **Verify local tools**
```
```
edbctl version
kubectl version --client
```
```
 .. Note::
 Ensure your workstation meets the CPU and RAM requirements to host the temporary␣
↪bootstrap cluster during installation. !!!

 ## Deploy your Kubernetes cluster

 With your management workstation configured, you are ready to deploy your cluster␣
↪according to the relevant specifications defined in the previous Phase.

 ### Select your deployment path

 Select the guide below that matches the **Kubernetes flavor** you selected in Phase␣
↪1 or 2 and proceed by deploying according to your system requirements defined in ␣
↪:ref:`Phase 2: Gathering your system requirements <Phase 2: Gathering your system␣
↪requirements>` .

 ### Public cloud providers

 .. csv-table::
:header: Provider,Official documentation,EDB Knowledge Base,Key considerations
:widths: 15,12,12,20
:align: left
:class: longtable

Azure (AKS) , ` **Tutorial - Create an Azure Kubernetes Service (AKS) cluster**
↪<https://learn.microsoft.com/en-us/azure/aks/tutorial-kubernetes-deploy-cluster>`__␣
↪, ` **HM on Azure Guide** <https://knowledge.enterprisedb.com/hc/en-us/articles/
↪24863144589980-Guidance-on-AKS-for-Hybrid-Manager>`__ ,""
AWS (EKS) , ` **Creating an EKS cluster** <https://docs.aws.amazon.com/eks/
↪latest/userguide/create-cluster.html>`__ , ` **HM on EKS Guide** <https://
↪knowledge.enterprisedb.com/hc/en-us/articles/24212491501468-Guidance-on-AWS-for-
↪Hybrid-Manager>`__ ,"Requires IAM OIDC provider, EBS CSI driver, and LoadBalancer␣
↪Controller."
Google Cloud (GKE) , ` **Creating a regional cluster** <https://cloud.google.com/
↪kubernetes-engine/docs/how-to/creating-a-regional-cluster>`__ , ` **HM on GKE␣
↪Guide** <https://knowledge.enterprisedb.com/hc/en-us/articles/24212699192732-
↪Guidance-on-GCP-for-Hybrid-Manager>`__ ,Requires Workload Identity enabled and␣
↪specific VPC firewall rules.
**Red Hat OpenShift Service on AWS (ROSA),:ref:`Troubleshooting <Getting started with␣
↪Pipeline Designer>` ,n/a,""

 ### Private cloud and on-premises
```

(continues on next page)

(continued from previous page)

```
 .. csv-table::
:header: Distribution,Official documentation,EDB Knowledge Base,Key considerations
:widths: 15,12,12,20
:align: left
:class: longtable

**Rancher RKE2** ,:ref:`Troubleshooting <Langflow quickstart>` , ` **HM on RKE2␣
↪Guide**  <https://knowledge.enterprisedb.com/hc/en-us/articles/24212371387420-
↪Guidance-on-Rancher-for-Hybrid-Manager>`__ ,Requires manual setup of Longhorn (or␣
↪similar) storage and MetalLB for ingress.
**Red Hat OpenShift** ,:ref:`Troubleshooting <Build with AI>` , ` **HM on OpenShift␣
↪Guide**  <https://knowledge.enterprisedb.com/hc/en-us/articles/24212799018780-
↪Guidance-on-RHOS-for-Hybrid-Manager>`__ ,Requires specific SCC (Security Context␣
↪Constraints) and Route configuration.

 ###  Set your node abstractions

 ####  Control Plane

 **Required labels & taints:**

 *  **Label:** `edbaiplatform.io/control-plane: "true"`

 *  **Taint:** * Key: `edbaiplatform.io/control-plane`  * Value: `"true"`  * Effect:␣
↪`NoSchedule`

 ```yaml
 spec:
     replicas: 3
     template:
     spec:
         metadata:
         labels:
             edbaiplatform.io/control-plane: "true"
         taints:
         - key: edbaiplatform.io/control-plane
             value: "true"
             effect: NoSchedule
```

!!!important Set the control-plane pool's **per-node pod limit to at least 60** (`maxPods`). The control-plane stack — including the HM system databases — places ~150 pods on this pool, which exceeds the default on some platforms (**AKS Azure CNI defaults to ``maxPods=30``**) and leaves control-plane pods unschedulable even when CPU and memory are sufficient. On AKS this is set at node-pool creation only (`az aks nodepool add … --max-pods 110`). See *3.1 Control plane nodes* .

Required labels & taints:

- **Label:** `edbaiplatform.io/postgres: "true"`
- **Taint:** * Key: `edbaiplatform.io/postgres` * Value: `"true"` * Effect: `NoSchedule`

```
#  Example Node Pool Specification
spec:
```

(continues on next page)
```

(continued from previous page)

```
    replicas: 3 # Minimum recommended for High Availability
    template:
      metadata:
        labels:
          edbaiplatform.io/postgres: "true"
      spec:
        taints:
        - key: edbaiplatform.io/postgres
          value: "true"
          effect: NoSchedule
```

```
```  #### AI model nodes

!!!note Required only for deployments where the `ai` installation scenario is enabled. This scenario is included by default unless it is manually excluded via the `scenarios` configuration parameter.

.. ::
* *Sizing:** Current recommendations require **Nvidia B200 GPUs** (or equivalent supported hardware).

Required labels & taints:

* **Label:** `nvidia.com/gpu: "true"`

* **Taint:** * Key: `nvidia.com/gpu` * Value: `"true"` * Effect: `NoSchedule` ## Validate cluster specifications

Now that you have deployed your cluster, before proceeding:

1. Confirm that your :ref:`Set up your management workstation <Set up your management workstation>` can reach all relavent endpoints for configuration in the next phase: :ref:`Phase 4: Preparing your environment <Phase 4: Preparing your environment>` .

1. Confirm that your deployed cluster matches the relevant specifications you defined in :ref:`Phase 2: Gathering your system requirements <Phase 2: Gathering your system requirements>` :

Compute resources

Source: :ref:`Gathering your system requirements>Compute (Node requirements) <Gathering your system requirements>`

* **Node count:** Ensure your node pools match what was decided in the previous phases.

* **Architecture:** Verify that all nodes use the **AMD64 (x86_64)** architecture. *ARM-based nodes are not currently supported for the Control Plane.*

* **Pods per node:** Verify the control-plane nodes allow **≥ 60 pods** each (`kubectl get nodes -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.allocatable.pods}{"\n"}{end}'`). On AKS Azure CNI the default of 30 is too low and
```

(continues on next page)
```

(continued from previous page)

```
↪must be raised at node-pool creation.

####  Validation of node abstractions

After provisioning your infrastructure, run these commands to verify the nodes are␣
↪correctly labeled and tainted. This ensures the Hybrid Manager installer can␣
↪schedule pods correctly.

1. **Validate Control Plane nodes:**

```bash
    kubectl get nodes -l edbaiplatform.io/control-plane="true"
```

1. **Verify taints**

```
kubectl get nodes -o json | jq .items[] | select(.metadata.labels["edbaiplatform.io/
↪control-plane"]=="true") | {name: .metadata.name, taints: (.spec.taints // [] |␣
↪map(select(.key=="edbaiplatform.io/control-plane")))}
```

1. **Validate Data Plane nodes:**

```
kubectl get nodes -l edbaiplatform.io/postgres="true"
```

1. **Verify taints**

```
kubectl get nodes -o json | jq .items[] | {name: .metadata.name, labels: .metadata.
↪labels["edbaiplatform.io/postgres"], taints: (.spec.taints // [] | map(select(.key==
↪"edbaiplatform.io/postgres")))}
```

Block storage configuration

Source: *Gathering your system requirements>6. Block storage (Database & logs)*

- **Storage class:** Ensure the cluster has the ***Storage Class** default* you identified in Phase 2 for the Control Plane (e.g., `gp2`).
- **Capability:** If desired, you may establish any number of storage classes for your Data Plane (Postgres workloads), and they are then options to use when provisioning data base clusters using HM.

Ingress

Source: *Gathering your system requirements>5. Ingress*

- **Ingress controller:** Verify that your chosen Ingress mechanism (LoadBalancer or NodePort) is active and exposing the required ports.
- **DNS:** Ensure CoreDNS (or equivalent) is healthy and resolving internal service names.

Post-deployment validation

Once your cluster is `Active` or `Ready` , perform these quick checks to ensure it is ready for the **Preparing your environment** phase.
```

Verify API Access

From your *Set up your management workstation* , ensure you can reach the API server.

```
kubectl cluster-info
```

- **Success:** Returns the Kubernetes control plane URL.

Verify Storage Capability

Confirm a valid storage class is present and set to (default).

```
kubectl get sc
```

- **Success:** Output lists a class (e.g., `gp3`) with `(default)` in the name.

Verify Identity (Cloud Only)

If you are on EKS/GKE/AKS, verify that your current IAM identity has administrative rights.

```
kubectl auth can-i create secrets --all-namespaces
```

- **Success:** Output is `yes` .

Next phase

Your infrastructure is provisioned. You can now connect to this cluster.

Next sync images to your local registry, stage your secrets, configure TLS, set up any relevant advanced features, and finalize your HM configuration.

** *Proceed to Phase 4: Preparing your environment* →**

4.1.1.8.4 Preparing your environment

Overview

Role focus: Site reliability engineer (SRE) / Infrastructure Engineer

Prerequisites

- *Phase 1: Planning your architecture* (Completed)
- *Phase 2: Gathering your system requirements* (Completed)
- *Phase 3: Deploying your Kubernetes cluster* (Completed & Running)

Outcomes

- *A Kubernetes cluster configured with necessary secrets (TLS, Auth, Storage, other necessary secrets)*
- *Any chosen advanced features configured (IDP, multi-location, KMS for TDE)*
- *A finalized and validated configuration ready for Hybrid Manager (HM) installation—a ``HybridControlPlane`` Custom Resource manifest*

!!!Note

EDB Support Context: Final preparation for Hybrid Manager is primarily the **customer's responsibility** (except for *Sovereign Systems*), as is the cluster's lifecycle operation. **Professional Services** can be engaged via a Statement of Work (SoW), and **Support** can offer assistance through knowledge base articles.

Phase 4 guide

Now that your Kubernetes cluster is running, you must prepare it for the HM platform.

This involves:

1. establishing administrative access through a remote management workstation.
2. staging required artifacts and secrets into the cluster,
3. translating your architectural decisions into the final configuration file.

Creating secrets

Create image pull secret

Use `edbctl` to create the `ImagePullSecret` . Doing this gives your Kubernetes cluster access to the EDB registry to pull images required for the installation.

```
edbctl image-pull-secret create
```

For more CLI options, see *edbctl image-pull-secret* .

Create component secrets

Use `edbctl` to create secrets required by all HM components (`core` , `ai` , `migration` , `analytics`).

```
edbctl hybrid-manager create-install-secrets
```

For more CLI options, see *edbctl hybrid-manager* . To customize your component's secrets, see *Customizing secrets* .

Create object storage secret

To implement the *object storage requirement* , you must create the `edb-object-storage` Kubernetes secret in the `default` namespace before installation. For provider-specific configuration and a full parameter reference, see *Configuring object storage* .

Implementing TLS

You must implement the TLS strategy chosen during the *Phase 2: Gathering your system requirements* phase. Some TLS strategoes require secrets to be created.

SOP: Implementing TLS certificates describes the implementation options.

Configuring advanced features

Configuring multi-location architecture

Deploying HM across multiple locations (multi-DC) requires significant preparation to ensure the primary and secondary clusters can communicate securely.

You must establish trust domains, sync storage secrets, and configure the HM-internal beacon agent to register the secondary location.

Preparation checklist:

1. **Network:** Ensure connectivity on ports `8444` (SPIRE) and `9445` (Beacon) between clusters.
2. **Storage:** Synchronize the `edb-object-storage` secret across all clusters.
3. **Identity:** Configure SPIRE federation to allow cross-cluster trust.
4. **Configuration:** Define unique `beacon_location_id` values and trust domains in your HM configuration.

- ***SOP: Configuring multiple data centers for Hybrid Manager***
  - *Guidance: Detailed steps for setting up SPIRE federation and cross-DC wiring.*

Implementing KMS for TDE

If you require encryption at rest for your databases (TDE), you should enable the Key Management Service (KMS) provider and create your keys now.

- ***SOP: Adding KMS support***

Depending on the KMS you are using, refer to the guides below to create the corresponding key.

- *HashiCorp Vault*
- *AWS Key Management Service (KMS)*
- *Google Cloud Key Management Service (Cloud KMS)*

Completing environmental preparation

Name your backup folder

1. Create a new folder to store the HM backups for disaster recovery.
2. Assign a unique folder name (e.g., using a UUID generator) and provide it in your HM configuration under `spec.globalParameters.internal_backup_folder`.
3. The string must be 12 characters comprised of letters and numbers i.e. `internal_backup_folder: "a7462dbc7106"`.

Add required component configuration

The following component-specific parameters should be added to your HM configuration under `spec.componentsParameters`:

```
spec:
  componentsParameters:
    edb-migration-portal:
      enable_pdb: "false"
    upm-api-pnxt-metric:
      log_level: "info"
      tsdb_host: "thanos-query.monitoring.svc.cluster.local"
      tsdb_port: "9090"
    upm-api-pnxt-pd:
      log_level: "info"
    upm-beaco-ff-base:
```

(continues on next page)

(continued from previous page)

```
    aes_key_rotation_interval: 2160h
    usage_generator_interval: 24h
  upm-beaco-usages:
    usage_generator_interval: "24h"
  upm-beacon:
    beacon_location_id: <your-location-id>
    server_host: <your-beacon-server-host>
  upm-istio-gateway:
    enable_server_session: "true"
    openshift_console_domain_name: <your-openshift-console-domain>
  upm-prometheus:
    memory: "4Gi"
  upm-thanos:
    memcached_cpu: "500m"
    memcached_memory: "2Gi"
    query_frontend_cpu: "200m"
    query_frontend_memory: "120Mi"
  upm-trust-manager:
    cert_manager_namespace: "cert-manager"
```

Finalizing the cluster configuration

You now construct or update the final configuration for your HM installation.

This configuration references your **system requirements** from Phase 2 and the environment settings and feature configurations you prepared here in Phase 4.

Create the HybridControlPlane Custom Resource

Configure your Hybrid Manager installation as a `HybridControlPlane` Custom Resource, used with the `edb-hcp-operator` chart. It provides version management and Kubernetes-native resource handling.

- ***SOP: Install with the Operator*** guide for a complete HybridControlPlane manifest template and field mapping reference.

Validating the configuration file, secrets, LoadBalancer, and cluster readiness

Before proceeding to deployment, validate both your configuration file and the runtime environment.

Validate configuration syntax

Validate your configuration file before proceeding.

Use `kubectl` dry-run to validate your `HybridControlPlane` manifest:

```
kubectl apply --dry-run=client -f hybridmanager.yaml
```

If this command completes without error, your manifest syntax is valid.

!!!Note This dry-run resolves the `HybridControlPlane` kind through the cluster's API discovery, so it requires the operator's Custom Resource Definitions (CRDs) to already be installed. The operator installs those CRDs in *Phase 5: Installing Hybrid Manager* ; until then the command fails with `error: unable to recognize ... no matches for kind "HybridControlPlane"` . Run this validation **after** installing the operator (Phase 5) and before applying

the manifest for real. To check only YAML syntax at this phase, use a YAML linter such as `yq . hybridmanager.yaml`.

First, run `add` and `update` the Helm repository:

```
helm repo add enterprisedb-edbpgai "https://downloads.enterprisedb.com/${EDB_
↪SUBSCRIPTION_TOKEN}/pgai-platform/helm/charts/"
helm repo update
```

Then validate your `values.yaml` against the Helm chart:

```
helm template edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap \
  --values values.yaml \
  --version <target-helm-version> > /dev/null
```

If this command completes without error, your YAML syntax is valid.

Validate secrets

Ensure the secrets referenced in your configuration exist in the cluster. To list the image pull secrets:

```
edbctl image-pull-secret list
```

To list the component secrets:

```
edbctl hybrid-manager list-install-secrets
```

Validate LoadBalancer

Verify that your cluster can provision an external load balancer service (required for ingress).

```
kubectl create service loadbalancer test-lb --tcp=80:80
kubectl get svc test-lb
```

- **Success:** If the load balancer controller is functioning, the test-lb service will be assigned an external IP or hostname.
- **Cleanup:** `kubectl delete svc test-lb`

Run diagnostic tooling

For a comprehensive check of the cluster's readiness, use the diagnostic plugin.

- **SOP: Use kubectl-edb-diagnostic** .

Next phase

Now that your management workstation is ready, images are synced, secrets are created, advanced features are configured, your HM configuration is finalized, and your configuration file is validated, you are ready to install HM.

** *Proceed to Phase 5: Installing Hybrid Manager →* **

4.1.1.8.5 Install Hybrid Manager

Overview

Role focus: Site reliability engineer (SRE) / Infrastructure Engineer

Prerequisites

- *Phase 1: Planning your architecture* (Completed)
- *Phase 2: Gathering your system requirements* (Completed)
- *Phase 3: Deploying your Kubernetes cluster* (Running and validated)
- *Phase 4: Preparing your environment* (Completed and validated)

Outcomes

- *A fully deployed Hybrid Manager (HM) platform, accessible via the web portal.*

> **Note**
>
> **Deployment responsibility:** Installation of Hybrid Manager (HM) is optimally executed by the **customer's SRE team**, and the customer owns its life cycle operation. If you have a *Sovereign System* , the EDB CX team executes the HM install. **Professional Services** and **Sales Engineering** can also be engaged for assistance, and **Support** has knowledge base articles available. !!!

Next phase: Phase 6: *Exploring post-installation options*

Install Hybrid Manager

With your environment prepared and your configuration validated, you are ready to deploy HM.

This phase involves applying your configuration artifact to your Kubernetes cluster and monitoring the deployment.

Hybrid Manager is installed using the **EDB Postgres AI Operator** helm chart and a **HybridControlPlane Custom Resource**. This approach provides version flexibility, Kubernetes-native resource management, and GitOps support.

Understanding the HybridControlPlane Custom Resource

The `HybridControlPlane` is a Kubernetes Custom Resource that defines the desired state of your Hybrid Manager installation.

Key benefits of the operator approach

1. **Kubernetes-native:** Follows standard Kubernetes patterns with Custom Resources
2. **Version flexibility:** Can install different HM versions independently from the operator version
3. **GitOps friendly:** CR manifests work seamlessly with GitOps tools (ArgoCD, Flux)
4. **Declarative:** The operator reconciles the actual state to match your desired state

Verify final configuration

Before installing, prepare your `HybridControlPlane` manifest using the example below as a template.

Example HybridControlPlane manifest

Here's a complete example showing the structure:

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
  annotations:
    biganimal.enterprisedb.io/deletion-protect: enabled
spec:
  # Basic configuration
  flavour: eks  # or aks, gke, rhos, rke
  imageRegistry: docker.enterprisedb.com/pgai-platform
  version: "v2026.5.1"

  # Source configuration
  source:
    remote: true  # Use remote bootstrap image
    # OR
    # useLocalKustomizations: true  # Use operator embedded manifests

  # Image pull secrets
  imagePullSecrets:
    - name: edb-cred
      namespace: edbpgai-bootstrap

  # Global parameters
  globalParameters:
    CONTAINER_REGISTRY_URL: docker.enterprisedb.com/pgai-platform
    internal_backup_folder: "abc123xyz789"
    portal_domain_name: "portal.my-company.com"
    trust_domain: "portal.my-company.com"
    dms_domain_name: "migration.my-company.com"
    storage_class: "gp3"
    portal_certificate_issuer_kind: "ClusterIssuer"
    portal_certificate_issuer_name: "letsencrypt-prod"
    load_balancer_mode: "public"  # or "private" or "disabled"
    load_balancer_provider: "aws"  # or "gcp", "azure". Leave empty for self-managed␣
↪LB
    # node_port_domain: "pg.my-company.com"  # Required only if load_balancer_mode is␣
↪"disabled"

  # Component-specific parameters
  componentsParameters:
    upm-beacon:
      beacon_location_id: "us-east-1"
      server_host: "beacon.my-company.com"
    transporter-rw-service:
      domain_name: "migration.my-company.com"
```

(continues on next page)

(continued from previous page)

```
    transporter-dp-agent:
      rw_service_url: "https://migration.my-company.com/transporter"
    upm-istio-gateway:
      cookie_aeskey: "your-16-or-32-byte-key"
    transporter-data-operator:
      fips_enabled: "false"

  # Beacon server configuration
  beaconServer:
    logLevel: info
    hybridManagerInstallation: "my-company-prod"
    additionalTrustDomains:
      - "secondary-location.my-company.com"

  # Beacon agent configuration
  beaconAgent:
    accessKey: "your-access-key"
    logLevel: info
    providers:
      - appliance
    provisioning:
      imageDiscovery: true
      imagesetDiscoveryAuthenticationType: "basic"
      imagesetDiscoveryContainerRegistryURL: "docker.enterprisedb.com/pgai-platform"
      openshift: false
      operandSetVersions:
        - "1\.1\.[0-9]+"
      nodeSelector:
        kubernetes.io/os: linux
      topologySpreadConstraints:
        - maxSkew: 1
          topologyKey: "topology.kubernetes.io/zone"
          whenUnsatisfiable: "DoNotSchedule"
          podSelector: "MatchDatabasePods"
    transparentDataEncryptionMethods:
      - passphrase
      - hashicorp_vault

  # Cluster groups (for multi-DC deployments)
  clusterGroups:
    role: standalone  # or primary/secondary
    # For multi-DC primary:
    # primary:
    #   domainName: "primary.my-company.com"
    # secondaries:
    #   - domainName: "secondary.my-company.com"

  # Installation scenarios
  scenarios:
    - core
    - migration
    - ai
```

(continues on next page)

(continued from previous page)

```
    - analytics
    - dbaas

  # Optional: Disable specific components
  # disabledComponents:
  #   - component-name

  # Optional: provision an internal container registry (not for use except on fully-
↪air-gapped clusters!)
  # hmRegistryRef:
  #   name: hm-registry

  # Optional: Resource annotations
  resourceAnnotations:
    - name: istio-ingressgateway
      kind: Service
      annotations:
        service.beta.kubernetes.io/aws-load-balancer-scheme: internal
        service.beta.kubernetes.io/load-balancer-source-ranges: "10.0.0.0/8"
```

Install the EDB Postgres AI Operator

Add the EDB Helm repository

Register the EDB Helm chart repository and update your local cache. Replace `<EDB_SUBSCRIPTION_TOKEN>` with your EDB Repos 2.0 token.

```
helm repo add enterprisedb-edbpgai \
  "https://downloads.enterprisedb.com/${EDB_SUBSCRIPTION_TOKEN}/pgai-platform/helm/
↪charts/"
helm repo update
```

Create namespace

Create a dedicated namespace for the operator:

```
kubectl create namespace edbpgai-bootstrap --dry-run=client -o yaml | kubectl apply -
↪f -
__OUTPUT__
namespace/edbpgai-bootstrap created
```

Install the operator helm chart

Install the operator with `edbctl hm upgrade-operator`. The command lists the operator image tags available in the target registry, selects the latest one, and installs (or upgrades) the chart. No operator version needs to be picked by hand — starting with HM 1.4, the operator is decoupled from the HM release cadence and a single operator version supports multiple HM lines.

The recommended path is to mirror the HM artifacts (operator images + chart included) to your private registry first — see *Syncing images to a local private registry* — and then install the operator from that private registry. Re-using the same variables defined in the sync step (`LOCAL_REGISTRY_URI`, `LOCAL_REGISTRY_USER`, `LOCAL_REGISTRY_PWD`):

```
edbctl hm upgrade-operator \
  --registry-uri "${LOCAL_REGISTRY_URI}" \
  --registry-username "${LOCAL_REGISTRY_USER}" \
  --registry-password "${LOCAL_REGISTRY_PWD}"
__OUTPUT__
Latest HM Operator version found in "${LOCAL_REGISTRY_URI}": 2.0.0
Namespace "edb-hcp-operator-system" does not exist, creating it...
namespaces/edb-hcp-operator-system created
Creating image pull secret "edb-hcp-operator-pull-secret" in namespace "edb-hcp-
↪operator-system"...
secrets/edb-hcp-operator-pull-secret created
Installing HM Operator chart "edb-hcp-operator" (version 2.0.0) into namespace "edb-
↪hcp-operator-system"...
HM Operator chart "edb-hcp-operator" is now at version 2.0.0 in namespace "edb-hcp-
↪operator-system".
```

If your private registry uses a self-signed certificate, add `--registry-insecure`. To pass extra Helm values (for example to enable the in-cluster HM Registry component), use `--set`:

```
edbctl hm upgrade-operator \
  --registry-uri "${LOCAL_REGISTRY_URI}" \
  --registry-username "${LOCAL_REGISTRY_USER}" \
  --registry-password "${LOCAL_REGISTRY_PWD}" \
  --set hmRegistry.enabled=true
```

If you don't operate a private registry, you can also pull directly from EDB Repos 2.0 by pointing `--registry-uri` at `docker.enterprisedb.com/pgai-platform` and passing your EDB Repos 2.0 token as `--registry-password` (with `--registry-username pgai-platform`).

Verify operator installation

`edbctl hm upgrade-operator` waits for the operator pod to be ready before returning. If you want to double-check:

```
kubectl get pods -n edb-hcp-operator-system
```

You should see output similar to:

```
NAME                                  READY   STATUS    RESTARTS   AGE
edb-hcp-operator-xxxxxxxxxx-xxxxx   1/1     Running   0          30s
```

Manual Helm fallback

For environments still on edbctl `< 1.7.0`, the previous manual Helm flow still works. Re-using the same private-registry variables defined in *Syncing images to a local private registry* (`LOCAL_REGISTRY_URI`, `LOCAL_REGISTRY_USER`, `LOCAL_REGISTRY_PWD`):

```
kubectl create namespace edbpgai-bootstrap --dry-run=client -o yaml | kubectl apply -
↪f -

#  Create a pull secret for your private registry:

kubectl create secret docker-registry edb-cred \
  --namespace edbpgai-bootstrap \
```

(continues on next page)

(continued from previous page)

```
  --docker-server="${LOCAL_REGISTRY_URI}" \
  --docker-username="${LOCAL_REGISTRY_USER}" \
  --docker-password="${LOCAL_REGISTRY_PWD}" \
  --dry-run=client -o yaml | kubectl apply -f -

#  Install the operator using Helm:

helm template \
  --namespace edbpgai-bootstrap \
  --version <OPERATOR_VERSION> \
  --include-crds \
  edb-hcp-operator enterprisedb-edbpgai/edb-hcp-operator \
  --set "controllerManager.manager.image.repository=${LOCAL_REGISTRY_URI}/edb-hcp-
↪operator/manager" \
  --set "controllerManager.manager.image.tag=<OPERATOR_VERSION>" \
  --set "imagePullSecrets[0].name=edb-cred" \
  | kubectl apply -n edbpgai-bootstrap -f -
```

`<OPERATOR_VERSION>` for HM 1.4 is `2.0.0` or later from the `2.x` lineage. Check the *Hybrid Manager 2025.11.0 release notes* for the version supported on the HM release you are installing.

If you don't operate a private registry, swap `${LOCAL_REGISTRY_URI}` for `docker.enterprisedb.com/pgai-platform` and use your EDB Repos 2.0 token (with username `pgai-platform`).

Create the HybridControlPlane resource

Prepare your HM manifest

Create a file named `hybridmanager.yaml` with your configuration. Use the example above as a template, customizing the values based on your environment preparation.

Critical fields to verify:

1. ``**spec.flavour**`` : Your Kubernetes distribution
2. ``**spec.imageRegistry**`` : Your container registry URL
3. ``**spec.version**`` : The HM version you want to install
4. ``**spec.globalParameters.portal_domain_name**`` : Your portal FQDN
5. ``**spec.beaconAgent.provisioning**`` : Your provisioning configuration

Apply the HM resource

Deploy the HybridControlPlane:

```
kubectl apply -f hybridmanager.yaml
__OUTPUT__
hybridcontrolplane.edbpgai.edb.com/edbpgai created
```

Monitor installation progress

Once the HM resource is created, the operator begins reconciling and deploying HM components.

Check HM status

Monitor the HybridControlPlane resource status:

```
kubectl get hybridcontrolplane edbpgai -o yaml
```

Look for the `status` section:

```
status:
  phase: deployed  # Can be: "", "deploying", "deployed", "failed", "deleting"
  components:
    - name: component-name
      state: success  # Can be: pending, success, failed, deleted
      version: "2026.1.0"
      timestamp: "2026-03-09T10:30:00Z"
  preflightChecks:
    phase: passed
    totalChecks: 10
    failedChecks: 0
  postflightChecks:
    phase: passed
    totalChecks: 8
    failedChecks: 0
```

Use kubectl to monitor

Get a quick status view:

```
kubectl get hcp
```

Expected output:

```
NAME      PHASE      PREFLIGHT   POSTFLIGHT
edbpgai   deployed   passed      passed
```

Check component deployment

View all components and their status:

```
kubectl get hcp edbpgai -o jsonpath={.status.components[*].name} | tr   \n
__OUTPUT__
upm-beacon
upm-istio-gateway
transporter-rw-service
transporter-dp-agent
transporter-data-operator
```

Check for failed components:

```
kubectl get hcp edbpgai -o jsonpath={.status.components[?(@.state=="failed")]}
```

If no output is returned, all components deployed successfully.

Check postflight

Verify postflight status of the HM:

```
kubectl get postflights.edbpgai.edb.com edbpgai -o yaml
__OUTPUT__
...
status:
  checkResults:
  - description: Check whether all pods belonging to the Hybrid Manager are ready.
    name: PodsChecker
    state: Passed
  - description: Check whether all internal databases belonging to the Hybrid Manager
      are ready.
    name: DBsChecker
    state: Passed
  - description: Check whether the latest Velero Backup has been completed.
    name: VeleroChecker
    state: Passed
  - description: Check whether all nodes are ready.
    name: NodesChecker
    state: Passed
  - description: Check whether all certificates belonging to the Hybrid Manager are
      valid.
    name: CertificatesChecker
    state: Passed
  lastTransitionTime: "2026-01-27T08:50:25Z"
  phase: Healthy
```

What to look for:

- Postflight should transition from `→ Healthy`
- All `.status.checkResults` should have a `Passed` state

> **Note**
>
> If the HM phase shows *failed* , check component status for error messages. If postflight checks fail, review the *checkResults* for specific failure reasons. !!!

Managing the HybridControlPlane

Updating configuration

To update your HM configuration:

1. Edit your `hybridmanager.yaml` manifest
2. Apply the changes:

```
kubectl apply -f hybridmanager.yaml
__OUTPUT__
hybridcontrolplane.edbpgai.edb.com/edbpgai configured
```

1. The operator will reconcile and apply the changes

Checking operator logs

If you encounter issues, check the operator logs:

```
kubectl logs -n edb-hcp-operator-system deployment/edb-hcp-operator-controller-
↪manager -f
```

Upgrading HM version

Run `edbctl hm upgrade-operator` (it picks up the latest operator version available in your registry — no version needs to be set by hand), then update the `spec.version` field in your `HybridControlPlane` manifest to the target HM version and `kubectl apply` it. The operator is decoupled from the HM release cadence, so the same `2.x` operator covers a range of HM versions — see the *Hybrid Manager 2025.11.0 release notes* for the operator version supported on the HM release you are upgrading to.

Verify portal access

Once your installation is healthy, verify that you can access the HM console.

1. **Open your browser** and navigate to the portal domain you defined during *Phase 4: Preparing your environment* (e.g., `https://portal.my-company.com`).
2. **Accept the certificate:** If you used a self-signed certificate (or haven't trusted your custom CA yet), you may see a browser warning. Proceed past it for now.
3. **Log In:** Use the **User-0** credentials you configured during environment preparation.

Success: You should see the HM console landing page.

Next phase

The platform is installed and running. You are now ready to perform the initial post-installation configuration, such as connecting your first self-managed cluster or adding team members.

** *Proceed to Phase 6: Exploring your post-installation options* →**

4.1.1.8.6 Exploring your post-installation options

Overview

Role: Infrastructure Engineer / Platform Administrator

Prerequisites

- *Phase 1: Planning your architecture* (Completed)
- *Phase 2: Gathering your system requirements* (Completed)
- *Phase 3: Deploying your Kubernetes cluster* (Running and validated)
- *Phase 4: Preparing your environment* (Completed and validated)

- *Phase 5: Installing Hybrid Manager* (Completed & Verified)
- Administrator access to the Hybrid Manager (HM) Portal.

Outcomes

- *A fully configured HM environment with security integrations, AI capabilities, and monitoring enabled.*

Next Phase: *Using Hybrid Manager*

Now that the HM Control Plane is running, you can enable optional but critical integrations. While the platform is functional without these, configuring them now ensures your environment is production-ready for security, AI workloads, and observability.

Configuring identity providers (IdP)

Configure your IdP integration via the HM console to enable SSO for users. This is a critical step for production environments to ensure secure access management.

Unlock AI capabilities: Complete GenAI Builder setup

- ***Guide: Gen AI Builder** — Requires the `ai` installation scenario (enabled by default).

Set up notifications (Observability)

Ensure your team is alerted to critical events (such as cluster failovers, backup failures, or capacity issues). HM allows you to configure notification channels globally or per project.

Supported channels:

- Email
- Webhooks (Slack, Microsoft Teams, generic JSON)
- PagerDuty
- ***Guide: Notifications**

Monitor external databases (Observability)

HM can monitor Postgres databases that it does not manage (examples: RDS, self-managed EC2, or on-prem instances).

To enable this, you must install the **EDB Postgres AI agent** on the external database host.

The agent collects metrics and logs, shipping them back to the HM control plane for centralized visualization.

- ***SOP: Monitor External Databases**

Enable Migration Co-pilot

Requires the `migration` installation scenario (enabled by default).

To use the AI-assisted **Migration Co-pilot**, you must apply specific secrets and configurations to the control plane.

This feature assists in analyzing and converting schemas from legacy databases to EDB Postgres.

- ***SOP: Enable Migration Co-pilot**

Next steps

Your HM platform is now fully deployed and integrated. You are ready to onboard users and begin provisioning clusters.

Proceed to :ref:`Using Hybrid Manager <Using Hybrid Manager>` →

4.1.2 Configuring SOPs

4.1.2.1 Syncing images to a local private registry

Related installation phase: *Phase 4: Preparing the Environment*

4.1.2.1.1 Prerequisites

- **`edbctl CLI tool** <Hybrid Manager CLI - edbctl>` installed and configured.
- A valid **EDB Repos 2.0 Token** .
- Write access to your target local private registry (examples: AWS ECR, Google Artifact Registry, Harbor).

Hosting images internally ensures:

- **Security & compliance:** You control the scanning and approval of artifacts.
- **Reliability:** Your deployment is not dependent on external internet connectivity or third-party uptime.
- **Performance:** Lower latency for cluster nodes pulling images.

This process uses the `edbctl` tool to copy artifacts from EDB Repos 2.0 to your private registry while preserving SHA256 digests.

The software stack of HM is pushed into EDB Repos 2.0 registry to provide artifacts for you to use in your local private registry: customer-managed internal registry for RHOS or Rancher RKE2 on-premises scenarios, or a self-managed registry on your cloud service provider (CSP) registry AWS Elastic Container Registry (ECR), Google Cloud Artifact Registry (GAR), or Azure Container Registry (ACR) for HM on CSP scenarios.

Before you begin, ensure you have a secure, approved private registry ready to host the container images.

You need the registry's URI, along with a username and password (or token) that has write access. Additionally, confirm the specific version of EDB Postgres AI you intend to install.

Using these credentials, the sync process copies all necessary artifacts from EDB Repos 2.0 directly into your local private registry, ensuring they are available for your Helm chart installation or upgrade.

4.1.2.1.2 Configure sync variables

Required Information:

- **Release Version:** The tag of Hybrid Manager (HM) you intend to install (example:`v2026.5.1`).
- **EDB Token:** Your access token for `docker.enterprisedb.com` .
- **Local Registry:** The URI and credentials for your organization's registry.

1. Define the EDB release version

```
export EDBPGAI_RELEASE=<EDB-pgai-release-version>
```

2. Define EDB credentials (Source)

```
export CS_EDB_TOKEN=<your-edb-repos-token>
export EDB_SOURCE_REGISTRY=pgai-platform
```

3. Define private registry credentials (Destination)

```
export LOCAL_REGISTRY_URI=<your_local_container_registry_uri>
export LOCAL_REGISTRY_USER=<your_local_registry_user>
export LOCAL_REGISTRY_PWD=<your_local_registry_password>
```

!!!Note

Cloud Registries: If you are using AWS ECR, Google Artifact Registry, or Azure ACR, ensure your local environment is authenticated (e.g., via `aws ecr get-login-password`) and that your `LOCAL_REGISTRY_PWD` reflects a valid token.

Execute the following command to sync the platform images and the operator artifacts in a single step:

```
edbctl image sync-to-local-registry \
    --destination-registry-uri "${LOCAL_REGISTRY_URI}" \
    --version "${EDBPGAI_RELEASE}" \
    --source-registry-username "${EDB_SOURCE_REGISTRY}" \
    --source-registry-password "${CS_EDB_TOKEN}" \
    --destination-registry-username "${LOCAL_REGISTRY_USER}" \
    --destination-registry-password "${LOCAL_REGISTRY_PWD}"
```

The command mirrors the following artifacts to your private registry:

- **Platform images** — Portal, Beacon, Transporter, and other HM components.
- **Operator manager image** — `edb-hcp-operator` controller image.
- **Operator bundle image** — OCP-certified bundle, used to build self-hosted OperatorHub catalogs in air-gapped OpenShift clusters.
- **Operator helm chart** — `edb-hcp-operator` chart, mirrored as an OCI artifact alongside the images.

4.1.2.1.3 Update configuration

Once the sync is complete, you must configure HM to pull images from your private registry.

Edit your `HybridControlPlane` CR to set `spec.imageRegistry` to your private registry URI:

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  imageRegistry: "<your_local_container_registry_uri>"
  # ... your other spec fields
```

Apply the updated CR:

```
kubectl apply -f hybridmanager.yaml
```

4.1.2.1.4 Next steps

With your images synced and your configuration updated, you are ready to proceed with the installation.

Return to :ref:`Phase 4: Preparing your environment <Phase 4: Preparing your environment>` or :ref:`Phase 5: Installing Hybrid Manager <Phase 5: Installing Hybrid Manager>` .

4.1.2.2 Using the in-cluster container registry

Related installation phase: *Phase 4: Preparing your environment*

4.1.2.2.1 Prerequisites

- A running Kubernetes cluster meeting the *Phase 2: Gathering your system requirements* .
- All cluster nodes deployed on a private subnet with no public IPs (see *Security considerations for Langflow*).
- The EDB Postgres AI Operator already installed on the cluster (see *Install the EDB Postgres AI Operator*).
- Creating Migration Portal secrets using `edbctl <Creating Migration Portal secrets using *edbctl*>` installed and configured on your management workstation.
- `kubectl` installed and authenticated against the cluster.
- `yq` (Command-line YAML processor).

The EDB Postgres AI Operator embeds a container registry, defined by the `HMRegistry` custom resource definition, that can run alongside the operator itself. Enabling the registry doesn't require syncing any additional artifacts beyond the operator image.

Use this registry for **air-gapped or fully internal deployments** where you either don't have or don't want to maintain a private registry external to HM. After installing the operator, you sync EDB images into the in-cluster registry once, and the rest of the platform pulls from it.

If you have a private registry external to HM that meets your security and operational requirements, prefer it over the in-cluster registry. See *Syncing images to a local private registry* for that flow.

4.1.2.2.2 Security considerations

> **Warning**
>
> The in-cluster registry serves images over **plain HTTP without authentication** and exposes itself to cluster nodes through a *NodePort* .

This setup is only safe when **all cluster nodes sit on a private subnet with no public IPs**. If any node has a public IP, the registry, along with every image it holds, is reachable from the internet without credentials.

If your nodes have public IPs, don't use the in-cluster registry. Use a private registry that's external to HM, as documented in *Syncing images to a local private registry* .

`HMRegistry` is a custom resource that deploys an in-cluster Open Container Initiative (OCI) image registry intended for Hybrid Manager (HM) workloads. The operator provisions a ConfigMap, Deployment, PVC, and Service for the registry, and on OpenShift it also patches `image.config.openshift.io/cluster` so the cluster's container runtime trusts the registry's NodePort/ClusterIP endpoint as an insecure source.

Create a local file called `hm-registry.yaml` and add the following content:

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HMRegistry
metadata:
  name: hm-registry
  annotations:
    biganimal.enterprisedb.io/deletion-protect: "enabled"
spec:
  mode: Insecure
  resources:
    requests:
      cpu: 250m
      memory: 512Mi
    limits:
      cpu: 1000m
      memory: 4Gi
  storage:
    pvc:
      size: 120Gi
```

Now run:

```
kubectl apply -f hm-registry.yaml
```

You can now follow the creation of the HMRegistry via:

```
kubectl get hmregistries.edbpgai.edb.com
```

When finished, you will see output that looks similar to this:

```
$ kubectl get hmregistries.edbpgai.edb.com hm-registry
NAME          PHASE   MESSAGE               CRI URI           KUBE URI
hm-registry   Ready   HMRegistry is ready   localhost:30183   hm-registry-service.
↪edbpgai-bootstrap.svc.cluster.local:5000
```

And the Operator will be managing a corresponding ConfigMap, Deployment, PVC, and Service, in the `edbpgai-bootstrap` namespace. You can see these via:

```
kubectl get cm,deployment,pods,pvc,svc -n edbpgai-bootstrap
```

4.1.2.2.3 Loading images into HMRegistry

Required Information:

- **Release Version:** The tag of Hybrid Manager (HM) you intend to install (example:`v1.4.0`).
- **EDB Token:** Your access token for `docker.enterprisedb.com` .

1. Define the EDB release version

```
export EDBPGAI_RELEASE=<EDB-pgai-release-version>
```

2. In a separate shell, run

```
kubectl port-forward svc/hm-registry-service 5000:5000 -n $EDB_OPERATOR_NAMESPACE
```

3. Define EDB credentials (Source)

```
export CS_EDB_TOKEN=<your-edb-repos-token>
export EDB_SOURCE_REGISTRY=pgai-platform
```

4. Sync images into the HMRegistry

```
edbctl image sync-to-local-registry \
    --version "${EDBPGAI_RELEASE}" \
    --source-registry-username "${EDB_SOURCE_REGISTRY}" \
    --source-registry-password "${CS_EDB_TOKEN}" \
    --destination-registry-uri "localhost:5000" \
    --insecure-destination-registry
```

5. Terminate the port-forward running in the separate shell with `ctrl-c` .

4.1.2.2.4 Configure the HybridControlPlane to use the HMRegistry

Update the HybridControlPlane custom resource like so:

```
kubectl patch hybridcontrolplanes edbpgai --type=merge --patch-file=/dev/stdin <<EOF
  spec:
    hmRegistryRef:
      name: hm-registry
EOF
```

The Operator then automatically inspects the `hm-registry` custom resource to find the `CRI URI` of the registry, and updates the `spec.imageRegistry` key of the `edpbgai` HybridControlPlane resource accordingly. No further configuration is necessary.

4.1.2.2.5 Switching back to a registry external to HM

If you no-longer wish to use the in-cluster registry, you will need to do the following:

1. Remove the reference to `hm-registry` from the HybridControlPlane resource

```
kubectl patch hybridcontrolplanes edbpgai --type=json -p [{"op":"remove","path":"/
↪spec/hmRegistryRef"}]
```

2. Update HybridControlPlane resource with your container registry URI

```
kubectl patch hybridcontrolplanes edbpgai --type=merge --patch-file=/dev/stdin <<EOF
  spec:
    imageRegistry: <your_container_registry_uri>
EOF
```

You may also need to consult *Configuring image discovery* .

3. Disable the deletion-protection on the in-cluster registry

```
kubectl patch hmregistries.edbpgai.edb.com hm-registry --type=merge --patch-file=/dev/
↪stdin <<EOF
  metadata:
    annotations:
      biganimal.enterprisedb.io/deletion-protect: disabled
EOF
```

4. Delete the `hm-registry` HMRegistry instance

`bash kubectl delete hmregistries.edbpgai.edb.com hm-registry` This will destroy everything associated with the in-cluster HMRegistry in the `edbpgai-bootstrap` namespace: ConfigMap, Deployment, PVC, and Service.

4.1.2.3 Configuring image discovery

Related installation phase: *Phase 4: Preparing the Environment*

4.1.2.3.1 Prerequisites

- ***Synced images to your local container registry*** (Completed) - You must have a private registry populated with EDB images.
- **Registry Compliance:** Your registry must be OCI compliant .
- **Permissions:** Access to configure IAM roles (AWS/GCP) or Registry Access Control (On-Prem).

Overview

Image discovery is a background process running within the **HM-internal beacon agent**. It connects to your private container registry, scans for available Postgres versions (tags), and registers them in the Hybrid Manager (HM).

Image discovery ensures that when you sync new database versions to your registry, they automatically become available for your users to provision without manual intervention.

Configure image discovery

To enable this feature, you must configure the Beacon agent with your registry details.

Edit your `HybridControlPlane` CR to add the image discovery settings under `spec.beaconAgent` :

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  beaconAgent:
    provisioning:
      imageDiscovery: true
      imagesetDiscoveryContainerRegistryURL: "<your-private-registry-uri>"
      # Set to true only for self-signed/internal CA scenarios
      imagesetDiscoveryAllowInsecureRegistry: false
    # ... your other beaconAgent settings
```

4.1.2.3.2 imageDiscovery

This boolean option enables or disables the image discovery feature. It defaults to `false` .

4.1.2.3.3 imagesetDiscoveryContainerRegistryURL

This option specifies the URL of the container registry from which to discover PostgreSQL container images.

Registry provider	Registry URL	Example
EDB Repo 2.0	*docker.enterprisedb.com/pgai-platform*	*docker.enterprisedb.com/pgai-platform*
Azure Container Registry (ACR)	*<registry-name>.azurecr.io*	*myregistry.azurecr.io*
Amazon Elastic Container Registry (ECR)	*<aws-account-id>.dkr.ecr.<aws*	*123456079902.dkr.ecr.us-east-1.amazonaws.com*
Google Artifact Registry (GAR)	*<gcp-region-id>-docker.pkg.dev*	*us-east1-docker.pkg.dev*
GitLab Container Registry	*registry.gitlab.com*	*registry.gitlab.com*
MicroK8s Built-in Registry	*localhost:32000*	*localhost:32000*

imagesetDiscoveryContainerRegistryProvider

This option defines the provider for the registry located at `imagesetDiscoveryContainerRegistryURL` . It defaults to `oci` .

Provider	Description
oci	Set this if your container registry provider is fully OCI compliant , including the */v2/_catalog* endpoint.
gitlab	Set this if your container registry provider is *GitLab Container Registry* .

The recommended provider value per registry provider is as follows:

Registry provider	Provider value
EDB Repo 2.0	*oci*
Azure Container Registry (ACR)	*oci*
Amazon Elastic Container Registry (ECR)	*oci*
Google Artifact Registry (GAR)	*oci*
GitLab Container Registry	*gitlab*
Harbor Registry	*oci*
MicroK8s Built-in Registry	*oci*

4.1.2.3.4 imagesetDiscoveryAllowInsecureRegistry

If this option is enabled, it allows establishing a TLS connection without certificate validation. It defaults to `false` .

imagesetDiscoveryAuthenticationType

The supported authentication types by registry provider are as follows:

Registry provider	Supported	Recommended
EDB Repo 2.0	*token*	*token*
Azure Container Registry (ACR)	*token*, *basic*	*token*
Amazon Elastic Container Registry (ECR)	*eks_managed_identity*	*eks_managed_identity*
Google Artifact Registry (GAR)	*token*, *basic*	*basic*
GitLab Container Registry	*token*	*token*
MicroK8s Built-in Registry	*none*	*none*

4.1.2.3.5 image_discovery_credentials_secret_name

This value is the name of the Kubernetes secret containing the credentials of registry for image discovery. Defaults to "edb-cred".

Configure registry permissions

The HM-internal beacon agent requires specific permissions to scan your registry.

It must be able to:

- List repositories
- List tags
- Read tag manifests

Select your platform below to configure the necessary permissions.

4.1.2.3.6 Generic / on-Premises (RKE2/OpenShift)

If you are using a standard OCI registry (like Harbor, Quay, or Artifactory), ensure the username/password you provided when creating your **Image Pull Secret** * has `read` and `list` permissions on the target repositories.

Refer to your registry's documentation (examples: Quay.io Permissions) to configure the service account.

4.1.2.3.7 Azure Container Registry (ACR)

The ACR token used as a Kubernetes image pull secret must have the `_repositories_pull_metadata_read` scope-map.

```
az acr token create --name <token-name> --registry <registry-name> \
    --scope-map "_repositories_pull_metadata_read"
```

4.1.2.3.8 AWS Elastic Container Registry (ECR)

When running on EKS with ECR, **EKS Managed Identity** is the required authentication method.

You must create an IAM role with the `AmazonEC2ContainerRegistryReadOnly` policy and associate it with the Beacon's service account.

Run this script to configure the IAM Role and Pod Identity:

```
#  Configuration Variables

EKS_CLUSTER_NAME="<eks_cluster_name>"
EKS_CLUSTER_REGION="<eks_cluster_region>"
IMAGE_DISCOVERY_IAM_ROLE_NAME="<iam_role_name>"

#  1. Create Trust Policy

cat <<EOF > ./image-discovery-trust.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
EOF

#  2. Create Role & Attach Policy

aws iam create-role --role-name "${IMAGE_DISCOVERY_IAM_ROLE_NAME}" \
    --assume-role-policy-document file://image-discovery-trust.json

aws iam attach-role-policy --role-name "${IMAGE_DISCOVERY_IAM_ROLE_NAME}" \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly

#  3. Associate with EKS Service Account

IMAGE_DISCOVERY_IAM_ROLE_ARN=$(aws iam get-role --role-name ${IMAGE_DISCOVERY_IAM_
↪ROLE_NAME} | jq -r .Role.Arn)

aws eks create-pod-identity-association --cluster-name "${EKS_CLUSTER_NAME}" \
    --namespace upm-beacon \
    --service-account upm-beacon-agent-k8s \
    --role-arn "${IMAGE_DISCOVERY_IAM_ROLE_ARN}" \
    --region "${EKS_CLUSTER_REGION}"
```

4.1.2.3.9 Additional configuration for package installation

To allow **package installation** for downloading and deploying packages, you must also grant ECR access to the **kapp-controller** component.

The kapp-controller is responsible for pulling Helm charts and package manifests from ECR during application deployment. Without this configuration, package installations will fail with credential errors.

Add Pod Identity association for kapp-controller:

```
#  Use the same IAM role ARN from above

aws eks create-pod-identity-association --cluster-name "${EKS_CLUSTER_NAME}" \
    --namespace kapp-controller \
    --service-account kapp-controller-sa \
    --role-arn "${IMAGE_DISCOVERY_IAM_ROLE_ARN}" \
    --region "${EKS_CLUSTER_REGION}"
```

4.1.2.3.10 Google Artifact Registry (GAR)

When running on GKE, the Service Account used by the Beacon must have permissions to list projects and read artifacts.

Required Roles:

- `roles/artifactregistry.reader` (to read images)
- `roles/browser` (specifically `resourcemanager.projects.list` to find repositories)

Run this script to apply the IAM bindings:

```
#  Replace with your Project ID and Service Account Name

PROJECT_ID="<your-project-id>"
SERVICE_ACCOUNT_EMAIL="<service-account-name>@${PROJECT_ID}.iam.gserviceaccount.com"

#  Grant Reader Role

gcloud projects add-iam-policy-binding ${PROJECT_ID} \
    --member="serviceAccount:${SERVICE_ACCOUNT_EMAIL}" \
    --role="roles/artifactregistry.reader"

#  Grant Browser Role (Required for discovery)

gcloud projects add-iam-policy-binding ${PROJECT_ID} \
    --member="serviceAccount:${SERVICE_ACCOUNT_EMAIL}" \
    --role="roles/browser"
```

4.1.2.3.11 GitLab Container Registry

The Personal access token used as a Kubernetes image pull secret must have the following scopes:

- `read_registry` — allows read-only access to container registry images on private projects.
- `read_api` — allows read access to projects and container repositories.

For more details, see the GitLab Personal access token scopes .

> **Note**
>
> Only *legacy* tokens are supported. !!!

Configure Kubernetes Secret

The final step is ensuring HM knows which Kubernetes Secret contains the registry credentials (if not using Cloud Identity).

Confirm or update the credentials secret name in your `HybridControlPlane` CR under `spec.componentsParameters.upm-beacon`:

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  componentsParameters:
    upm-beacon:
      image_discovery_credentials_secret_name: "edb-cred"
    # ... your other componentsParameters
```

Apply the updated CR:

```
kubectl apply -f hybridmanager.yaml
```

Verification:

After applying these changes, check the logs of the **HM-internal beacon agent** pod. You should see messages indicating that it is scanning the registry and discovering image tags.

4.1.2.4 Customizing secrets

By default, Hybrid Manager (HM) includes pre-configured secrets to facilitate a quick, out-of-the-box installation. However, these default secrets are not intended for production use. For any live environment, you must create custom secrets to ensure your installation remains secure and unique.

HM secrets are categorized by installation scenarios. Depending on which installation scenarios you have enabled, you must configure the corresponding secrets below.

Scenario	Feature	Key secrets	Docs
ai	GenAI Builder	Fernet key, NVIDIA NGC credentials	*Customizing GenAI Builder secrets*
analytics	Catalog	Confounding key	*Customizing Catalog secrets*
migration	Migration Portal	DB accounts, Copilot HTTP/Metrics auth	*Customizing Migration Portal secrets for secure internal communication*

4.1.2.4.1 Customizing GenAI Builder secrets

Required for deployments with the `ai` installation scenario enabled. These secrets handle model pulls and data encryption for AI agents.

NGC API Key: Required for pulling NVIDIA NIM models.

Fernet key: Used for symmetric encryption of sensitive data within the GenAI stack.

Object storage: Requires S3-compatible buckets (AWS, GCS, etc.) for the DataLake.

Creating GenAI Builder secrets

!!!note Required for deployments with the `ai` installation scenario enabled. This scenario is included by default unless it is manually excluded via the scenarios configuration (`spec.scenarios` in the `HybridControlPlane` CR).

NVIDIA NGC documentation to enable model image pulls.

1. You can create a Fernet key secret and NGC key secrets using the `edbctl` CLI.
    - For manual installations, run this command and follow the interactive prompts:

```
edbctl hybrid-manager create-install-secrets --version <version> --scenario ai
```

```
- If you are running the installation via a CI/CD pipeline, you must suppress␣
↪interactive prompts. The method for achieving this depends on your `edbctl` ␣
↪version:
```

1. Ensure the NGC API key you created above is available as a variable.
2. Configure `edbctl` for non-interactive behavior:

```
edbctl config set interactive_mode off
edbctl config set confirm_mode off
```

1. Run the setup command:

```
edbctl hybrid-manager create-install-secrets --version <version> --scenario ai
```

This creates the Fernet key secret, as well as `nvidia-nim-secrets` and `ngc-cred` secrets in the `default` namespace with the appropriate replication annotations.

!!!note Fernet is a cryptographic library used by Python. It provides symmetric encryption/decryption and is required to store secret data. .. ::

1. The HM administrator must keep the Fernet key safe and back it up. The secret name and namespace depend on the version of Hybrid Manager you are running.

For versions 2026.2 and earlier, the solution is based on Griptape. Use the following command to retrieve the secret:

```
kubectl get secret -n upm-griptape fernet-secret -o yaml
```

For versions 2026.3 and later, the solution shifted to LangFlow. The secret is now stored in the default namespace:

```
kubectl get secret langflow-secret -n default -o yaml
```

Store the key safely.

1. Configure DataLake object storage for GenAI builder by creating a DataLake bucket in the object storage you're using for your Hybrid Manager deployment. GenAI Builder uses it to store structures, tools, and indexed data.

```
aws s3 mb s3://<your-datalake-bucket-name> -region     <your-region>
```

```
gsutil mb -l <your-region> gs://<your-datalake-bucket-name>
```

Use your provider's management console or CLI to create a bucket with a unique name for your DataLake bucket.

1. Capture the following information for your bucket. You will need it later when you first use the GenAI launchpad application. The console will prompt you for your DataLake bucket configuration, which requires:

- `DATA_LAKE_ROOT_BUCKET` : The name of the bucket for use with DataLake.

- `DATA_LAKE_S3_ACCESS_KEY` : The access_key used to connect to the DataLake bucket.
- `DATA_LAKE_S3_SECRET_ACCESS_KEY` : The secret_access_key used to connect to the DataLake bucket.
- `DATA_LAKE_S3_ENDPOINT_UR` : The endpoint URL used to connect to the DataLake bucket.

1. Update the bucket's settings with the CORS configuration:

Update the bucket's settings to have the following CORS configuration :

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "PUT",
            "POST",
            "DELETE",
            "GET",
            "HEAD"
        ],
        "AllowedOrigins": [
            "https://${PORTAL_DOMAIN_NAME}"
        ],
        "ExposeHeaders": []
    }
]
```

Where `https://${PORTAL_DOMAIN_NAME}` is the domain configured for your Hybrid Manager.

The S3 interoperability layer in GCS allows GenAI Builder to use GCS as an S3-compatible object store. 1. In the GCS console, under **Settings**, turn on the s3 interoperability. 2. Update or create a service account with the Storage Admin and Service Account Token Creator roles. 3. Create an

HMAC key pair for the service account. 4. Create a config file with a CORS configuration that points at the Hybrid Manager endpoint:

edb_notranlate_8 Where `https://${PORTAL_DOMAIN_NAME}` is the domain configured for your Hybrid Manager. 5. Apply the CORS configuration to the previously created bucket:

edb_notranlate_9 Use your provider's management console or CLI to configure cross-origin resource sharing (CORS) with Hybrid Manager.

4.1.2.4.2 Customizing Catalog secrets

Required for deployments with the `analytics` installation scenario enabled.

Confounding key: A critical randomized string used to encrypt storage credentials.

Warning: Loss of this key prevents access to managed Lakehouse data. Always back up the `pg-confounding-key` secret.

Creating Catalog secret

!!!note Required for deployments with the `analytics` installation scenario enabled. This scenario is included by default unless it is manually excluded via the scenarios configuration (`spec.scenarios` in the `HybridControlPlane` CR).

```
- For manual installations, run this command and follow the interactive prompts:
```

```
      edbctl hybrid-manager create-install-secrets --version <version> --scenario␣
↪analytics
```  - If you are running the installation via a CI/CD pipeline, you must suppress␣
↪interactive prompts. The method for achieving this depends on your `edbctl` ␣
↪version:

1. Configure `edbctl` for non-interactive behavior:

```shell
          edbctl config set interactive_mode off
          edbctl config set confirm_mode off
```

1. Run the setup command:

edb_notranlate_2 !!!note - A confounding key is a randomized string that's at least 32 bytes long. - Create a confounding key for each Hybrid Manager deployment. .. ::

1. The Hybrid Manager administrator must keep the confounding key safe and back it up.

> **Warning**
>
> The loss of the confounding key in a disaster scenario leads to a situation in which there's no mechanism for accessing the Lakehouse data managed by the Hybrid Manager data catalog. Instead, the administrator would have to create and store the new key, restart the *upm-lakekeeper* /*lakekeeper* workload, and rebuild all of the existing data catalogs carefully without deleting them. That procedure is very risky and would require support from EDB PG AI Professional Services team. Fetch the key:

`shell kubectl get secrets -n upm-lakekeeper pg-confounding-key -o yaml` Store the key safely.

4.1.2.4.3 Customizing Migration Portal secrets for secure internal communication

Required for deployments with the `migration` installation scenario enabled. These secrets manage internal communication between the Migration Portal, the AI Copilot, and the underlying databases.

While `edbctl` can generate these automatically, production environments often require manual overrides for specific service accounts:

Database accounts: Internal service users and superuser credentials.

Copilot authentication: HTTP and Metrics authentication for secure component inter-communication.

While our standard *installation guide* already includes instructions on using `edbctl` to generate these custom secrets, this page provides a deeper dive, which includes the manual creation of secrets as an alternative and how to override existing secrets to update or rotate your credentials.

Creating Migration Portal secrets using `edbctl`

!!!note Required for deployments with the `migration` installation scenario enabled. This scenario is included by default unless it is manually excluded via the scenarios configuration (`spec.scenarios` in the `HybridControlPlane` CR).

- For manual installations, run this command and follow the interactive prompts:

```
edbctl hybrid-manager create-install-secrets --version <version> --scenario migration
```

- If you are running the installation via a CI/CD pipeline, you must suppress interactive prompts. The method for achieving this depends on your `edbctl` version:

1. Configure `edbctl` for non-interactive behavior:

```
edbctl config set interactive_mode off
edbctl config set confirm_mode off
```

1. Run the setup command:

```
edbctl hybrid-manager create-install-secrets --version <version> --scenario migration
```

This creates the namespaces, custom secrets, and generates secure random passwords for all five service accounts:

- `edb-migration-portal.db_secrets` - `edb-migration-portal.db_superuser_secrets` - `edb-migration-portal.copilot_secrets` - `edb-migration-copilot.db_secrets` - `edb-migration-copilot.metrics_auth_secrets`

For more information, see *Understanding Migration Portal secrets* .

Overriding existing secrets manually

If you need specific usernames or passwords, you can create the secrets manually. For production environments, we recommend using appropriate secure processes that don't expose the data to your terminal.

Creating secrets

1. Create a function that generates secure passwords for each of the service user accounts:

```
function generate_pw {
 dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64 | tr '+/' '-_' | tr     -d '='
}
```

1. Set environment variables for all the users and passwords required by your secrets:

```
HTTP_USERNAME=test_user_001
HTTP_PASSWORD=$(generate_pw)
DB_SUPERUSER_USERNAME=postgres
DB_SUPERUSER_PASSWORD=$(generate_pw)
PORTAL_DB_USERNAME=test_user_002
PORTAL_DB_PASSWORD=$(generate_pw)
COPILOT_DB_USERNAME=test_user_003
COPILOT_DB_PASSWORD=$(generate_pw)
METRICS_USERNAME=test_user_004
METRICS_PASSWORD=$(generate_pw)
```

1. Create the namespaces where you will store the custom secrets:

```
kubectl create namespace edb-migration-copilot
kubectl create namespace edb-migration-portal
```

1. Create the secret that determines the *database user credentials* :

```
kubectl create secret generic custom-db-secrets \
  --namespace=edb-migration-portal \
  --from-literal=username="${PORTAL_DB_USERNAME}" \
  --from-literal=password="${PORTAL_DB_PASSWORD}" \
  --type=kubernetes.io/basic-auth
```

1. Create the secret that determines the *database superuser credentials* :

```
kubectl create secret generic custom-db-superuser-secrets \
  --namespace=edb-migration-portal \
  --from-literal=username="${DB_SUPERUSER_USERNAME}" \
  --from-literal=password="${DB_SUPERUSER_PASSWORD}" \
  --type=kubernetes.io/basic-auth
```

1. Create the secret that *authenticates HTTP communication* between the copilot and Migration Portal:

```
kubectl create secret generic custom-edb-migration-copilot-auth \
  --namespace=edb-migration-portal \
  --from-literal=username=${HTTP_USERNAME} \
  --from-literal=password=${HTTP_PASSWORD} \
  --type=kubernetes.io/basic-auth
```

1. Create the secret that provides the *copilot with access to the Migration Portal database* :

```
kubectl create secret generic custom-ragchew-db-secrets \
  --namespace=edb-migration-copilot \
  --from-literal=username="${COPILOT_DB_USERNAME}" \
  --from-literal=password="${COPILOT_DB_PASSWORD}" \
  --type=kubernetes.io/basic-auth
```

1. Create the secret that enables monitoring of the copilot instance :

```
kubectl create secret generic custom-edb-migration-copilot-metrics-auth \
  --namespace=edb-migration-copilot \
  --from-literal=username=${METRICS_USERNAME} \
  --from-literal=password=${METRICS_PASSWORD} \
  --type=kubernetes.io/basic-auth
```

Apply the secrets to your installation

To override the default secrets, reference the new secret names in your configuration.

Edit your `HybridControlPlane` CR to reference the custom secrets under `spec.componentsParameters` :

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  componentsParameters:
    edb-migration-copilot:
      db_secrets: custom-ragchew-db-secrets
      metrics_auth_secrets: custom-edb-migration-copilot-metrics-auth
    edb-migration-portal:
```

(continues on next page)

(continued from previous page)

```
    copilot_secrets: custom-edb-migration-copilot-auth
    db_owner: ${PORTAL_DB_USERNAME}
    db_secrets: custom-db-secrets
    db_superuser_secrets: custom-db-superuser-secrets
  # ... your other componentsParameters
```

Apply the updated CR:

```
kubectl apply -f hybridmanager.yaml
```

The operator reconciles the change and configures the installation to use the secret overrides.

If you're overriding the secrets after installation, reapply the CR to make the modifications effective.

Back up the secrets

You can fetch the secrets like this:

```
kubectl get secret -n <namespace> <secret_name> -o yaml
```

This command prints the secret's contents in YAML format, which you can then copy and store safely and securely. Repeat this with each of the created secrets.

Understanding Migration Portal secrets

For each secret, **Description** explains why you need it. **Configuration key** shows where to reference it in the `HybridControlPlane` CR. **Parameters** shows how to set non-default values.

See *Apply the secrets to your installation* for an example of how to configure custom secrets and override the default ones.

Migration Portal database account

Description: Configures a database service account used by Hybrid Manager to store Migration Portal system data and schema assessment results into a dedicated database.

Configuration key: `spec.componentsParameters.edb-migration-portal.db_secrets`

```
apiVersion: v1
kind: Secret
metadata:
  name: <db-secret-name>
  namespace: edb-migration-portal
stringData:
  username: "<db username>"
  password: "<db password>"
type: kubernetes.io/basic-auth
```

Parameters:

- `metadata.name` : Name of the secret you will later reference in the configuration key.
- `stringData.username` : User that connects to the database. Also enter this user in the `db_owner` key for `edb-migration-portal` (see *Apply the secrets to your installation*).
- `stringData.password` : Assign a generated password for this user.

Migration Portal database superuser account

Description: Configures a database superuser account used by Hybrid Manager to provision Migration Portal and the copilot databases.

Configuration key: `spec.componentsParameters.edb-migration-portal.db_superuser_secrets`

```
apiVersion: v1
kind: Secret
metadata:
  name: <db-superuser-secret-name>
  namespace: edb-migration-portal
stringData:
  username: "postgres"
  password: "<db superuser password>"
type: kubernetes.io/basic-auth
```

Parameters:

- `metadata.name` : Name of the secret you will later reference in the configuration key.
- `stringData.username` : Must always be `postgres` .
- `stringData.password` : Assign a generated password for this user.

Migration Copilot HTTP access from Migration Portal

Description: Defines the credentials used to authenticate HTTP communication between the copilot and Migration Portal.

Configuration key: `spec.componentsParameters.edb-migration-portal.copilot_secrets`

```
apiVersion: v1
kind: Secret
metadata:
  name: <copilot-auth-secret>
  namespace: edb-migration-portal
stringData:
  username: "<http username>"
  password: "<http password>"
type: kubernetes.io/basic-auth
```

Parameters:

- `metadata.name` : Name of the secret used to authenticate HTTP requests that you will later reference in the configuration key.
- `stringData.username` : Choose a username for Migration Portal to use when authenticating with the copilot.
- `stringData.password` : Assign a generated password for this user.

Migration Copilot and Migration Portal database account

Description: Allows the copilot to communicate with the Migration Portal database. On initialization, the `edb-migration-copilot` component ensures this user is created and that correct ownership and permissions are set in the copilot database.

Configuration key: `spec.componentsParameters.edb-migration-copilot.db_secrets`

```
apiVersion: v1
kind: Secret
metadata:
  name: <copilot_mp_secret_name>
  namespace: edb-migration-copilot
stringData:
  username: "<db username>"
  password: "<db password>"
type: kubernetes.io/basic-auth
```

Parameters:

- `metadata.name` : Name of the secret used to provide the copilot with access to the Migration Portal database.
- `stringData.username` : Choose a username for the copilot to use when accessing the Migration Portal database.
- `stringData.password` : Assign a generated password for this user.

Monitoring and Migration Copilot

Description: Allows Hybrid Manager to monitor the state of the copilot deployment via Grafana.

Configuration key: `spec.componentsParameters.edb-migration-copilot.metrics_auth_secrets`

```
apiVersion: v1
kind: Secret
metadata:
  name: <copilot-metrics-secret-name>
  namespace: edb-migration-copilot
stringData:
  username: <metrics-http-username>
  password: <metrics http password>
type: kubernetes.io/basic-auth
```

Parameters:

- `metadata.name` : Name of the secret used to allow monitoring of the copilot.
- `stringData.username` : Set this to a username different from the one configured for Migration Copilot HTTP access from Migration Portal.
- `stringData.password` : Assign a generated password for this user.

4.1.2.5 Implementing TLS certificates

4.1.2.5.1 Set up a custom x.509 certificate for the Hybrid Manager Portal

You can change the self-signed x.509 certificate used by the Hybrid Manager (HM) Portal with your own.

To import your own certificate you need to create a Kubernetes secret containing the server certificate and the private key.

Generate the certificate

Generate the base64 encoded string of your certificate (example: `my-certificate.crt`) and private key (example: `my-certificate.key`).

```
cat my-certificate.crt | base64
cat my-certificate.key | base64
```

Create the yaml file

Create a file (example: `my-secret.yaml`) with the following content and replace `<base64 encoded string>` with the values generated in the previous step.

```
apiVersion: v1
data:
  tls.crt: <base64 encoded string>
  tls.key: <base64 encoded string>
kind: Secret
metadata:
  name: my-portal-certificate
  namespace: default
  annotations:
    replicator.v1.mittwald.de/replicate-to: istio-system
type: kubernetes.io/tls
```

Apply the secret in your Kubernetes cluster

```
kubectl apply -n default -f ./my-secret.yaml
```

Configure HM to use the new certificate

Edit your `HybridControlPlane` CR to add the `portal_certificate_secret` parameter under `spec.globalParameters` :

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  globalParameters:
    portal_certificate_secret: "my-portal-certificate"
    # ... your other globalParameters
```

Apply the updated CR:

```
kubectl apply -f hybridmanager.yaml
```

The operator reconciles the change and updates the portal certificate configuration.

For more information on how the secret can be formatted, consult Istio documentation .

4.1.2.5.2 Set up a custom cert-manager issuer for the HM Portal

The HM Portal's certificate can also be generated and managed using one of the x.509 issuers supported by HM's internal cert-manager, e.g. The ACME Issuer for Let's Encrypt certificates.

You can follow the documentation of the issuer of your choice directly from the cert-manager website to set it up.

> **Note**
>
> We suggest to set up a ClusterIssuer rather than an Issuer. If you prefer to set up an Issuer, you need to create it in the istio-system namespace. !!!

Once the ClusterIssuer is configured, pass its name to HM.

> **Note**
>
> Using a cert-manager issuer and providing your own certificate secret are mutually exclusive. If you configure an issuer, don't also set *spec.globalParameters.portal_certificate_secret* in your CR. !!!

Edit your `HybridControlPlane` CR to add the issuer parameters under `spec.globalParameters` :

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  globalParameters:
    portal_certificate_issuer_kind: "ClusterIssuer"  # Valid values are Issuer and␣
↪ClusterIssuer
    portal_certificate_issuer_name: "my-issuer"       # Your Issuer name
    # ... your other globalParameters
```

Apply the updated CR:

```
kubectl apply -f hybridmanager.yaml
```

The operator reconciles the change and configures the portal to use the specified issuer.

4.1.2.5.3 Bring your own private certificate authority

By default, all the certificates used by HM are signed by an internal certificate authority (CA) powered by cert-manager. The CA is created at install time.

If you prefer to use your own Private CA, follow the steps in this section. Otherwise, you can skip this section.

> **Note**
>
> To successfully use this method you must have access to the CA private key. !!!

Create one CA secret if it doesn't exist.

```
apiVersion: v1
data:
  ca.crt: <base64 encoded string> # real ca crt provided by customers
  tls.crt: <base64 encoded string> # real tls crt provided by customers
  tls.key: <base64 encoded string> # real tls key provided by customers
kind: Secret
metadata:
  name: my-custom-ca
  namespace: default
  annotations:
    replicator.v1.mittwald.de/replicate-to: cert-manager
type: kubernetes.io/tls
```

Warning

You can change my-custom-ca to a name of your choosing, except *global-ca-secret* . Remember to be consistent with the name you have chosen when running the following commands. !!!

Edit your `HybridControlPlane` CR to add the `ca_secret_name` parameter under `spec.globalParameters` :

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  globalParameters:
    ca_secret_name: "my-custom-ca"
    # ... your other globalParameters
```

Apply the updated CR:

```
kubectl apply -f hybridmanager.yaml
```

The operator reconciles the change and configures HM to use your private CA.

4.1.2.6 Configuring object storage

Before installing Hybrid Manager (HM), you must create a Kubernetes secret named `edb-object-storage` in the `default` namespace. HM uses this secret to access object storage for:

- Postgres database backups and WAL archiving
- Restore operations
- Data replication in multi-location deployments
- Logs (Postgres and internal services)
- Metrics (Postgres and internal services)

This secret is referenced automatically during installation. For backup and recovery planning, see *Backup readiness* .

For multi-location deployments, the `edb-object-storage` secret must be synchronized across all locations. See *Configuring multiple data centers* for details.

!!!note Your object storage must be dedicated entirely to Hybrid Manager. Any data already present in the bucket interferes with the successful operation of HM. If HM is removed and reinstalled, empty the bucket before starting the new installation.

To create the secret, complete the following steps before installing HM:

1. Select the configuration for your provider from *Provider configurations* below and fill in the required values.
2. Save the YAML to a file, for example `object-storage-secret.yaml` .
3. Apply it to your cluster:

```
kubectl apply -f object-storage-secret.yaml
```

4.1.2.6.1 Provider configurations

AWS IAM (EKS and ROSA)

Use workload identity to authenticate when running on EKS or ROSA. No static credentials are required.

```
apiVersion: v1
kind: Secret
metadata:
  name: edb-object-storage # cannot be changed
  namespace: default # cannot be changed
stringData:
  auth_type: workloadIdentity
  aws_region: <AWS_BUCKET_REGION>
  aws_role_arn: <PRIMARY_IDENTITY_ROLE_ARN>
  bucket_name: <AWS_BUCKET_NAME>
  secondary_role_arn: <SECONDARY_IDENTITY_ROLE_ARN>
  secondary_role_external_id: <SECONDARY_IDENTITY_EXTERNAL_ID>
```

Parameters:

Parameter	Required	Description
auth_type	Yes	Must be *workloadIdentity*.
aws_region	Yes	AWS region where the bucket is located. For example, *us-east-1*.
aws_role_arn	Yes	ARN of the primary IAM role that HM assumes to access the bucket.
bucket_name	Yes	Name of the S3 bucket.
secondary_role_arn	No	ARN of a secondary role for cross-account bucket access.
secondary_role_external_id	No	External ID required when assuming the secondary role.

AWS static credentials (generic Kubernetes)

Use static access keys when running on a generic Kubernetes cluster that connects to AWS S3 without workload identity.

```
apiVersion: v1
kind: Secret
```

(continues on next page)

(continued from previous page)

```
metadata:
  name: edb-object-storage
  namespace: default
stringData:
  auth_type: credentials
  aws_region: <AWS_BUCKET_REGION>
  bucket_name: <AWS_BUCKET_NAME>
  aws_access_key_id: <AWS_ACCESS_KEY_ID>
  aws_secret_access_key: <AWS_SECRET_ACCESS_KEY>
  secondary_role_arn: <SECONDARY_IDENTITY_ROLE_ARN>
  secondary_role_external_id: <SECONDARY_IDENTITY_EXTERNAL_ID>
```

Parameters:

Parameter	Required	Description
auth_type	Yes	Must be *credentials*.
aws_region	Yes	AWS region where the bucket is located.
bucket_name	Yes	Name of the S3 bucket.
aws_access_key_id	Yes	AWS access key ID for the IAM user or role.
aws_secret_access_key	Yes	AWS secret access key.
secondary_role_arn	No	ARN of a secondary role for cross-account bucket access.
secondary_role_external_id	No	External ID required when assuming the secondary role.

Azure Blob Storage

```
apiVersion: v1
kind: Secret
metadata:
  name: edb-object-storage
  namespace: default
stringData:
  provider: azure
  subscription_id: <AZURE_SUBSCRIPTION_ID>
  resource_group_name: <AZURE_RESOURCE_GROUP>
  storage_account_name: <AZURE_STORAGE_ACCOUNT>
  storage_account_container_name: <AZURE_STORAGE_CONTAINER>
  storage_account_key: <AZURE_STORAGE_KEY>
  region: <AZURE_REGION>
  client_id: <AZURE_CLIENT_ID>
  client_secret: <AZURE_CLIENT_SECRET>
  tenant_id: <AZURE_TENANT_ID>
```

Parameters:

Parameter	Required	Description
provider	Yes	Must be *azure*.
subscription_id	Yes	Azure subscription ID.

continues on next page

Table 35 – continued from previous page

Parameter	Required	Description
resource_group_name	Yes	Azure resource group containing the storage account.
storage_account_name	Yes	Name of the Azure storage account.
storage_account_container_name	Yes	Name of the blob container within the storage account.
storage_account_key	Yes	Storage account access key.
region	Yes	Azure region of the storage account. For example, *eastus*.
client_id	Yes	Client ID of the Azure service principal used for authentication.
client_secret	Yes	Client secret of the Azure service principal.
tenant_id	Yes	Azure Active Directory tenant ID.

GCP object storage

```
apiVersion: v1
kind: Secret
metadata:
  name: edb-object-storage
  namespace: default
stringData:
  provider: gcp
  location_id: <GCP_BUCKET_REGION>
  project_id: <GCP_PROJECT_ID>
  bucket_name: <GCP_BUCKET_NAME>
  credentials_json_base64: <GCP_CREDENTIAL_BASE64>
```

Parameters:

Parameter	Required	Description
provider	Yes	Must be *gcp*.
location_id	Yes	GCP region where the bucket is located. For example, *us-central1*.
project_id	Yes	GCP project ID.
bucket_name	Yes	Name of the GCS bucket.
credentials_json_base64	Yes	Base64-encoded contents of the GCP service account JSON key file. See *Troubleshooting* for instructions.

Other S3-compatible storage

Use this configuration for on-premises or third-party S3-compatible storage such as MinIO or Ceph. Only AWS static credentials (`auth_type: credentials`) are supported for S3-compatible storage — workload identity isn't available for non-AWS providers.

```
apiVersion: v1
kind: Secret
metadata:
  name: edb-object-storage
  namespace: default
```

(continues on next page)

(continued from previous page)

```
stringData:
  auth_type: credentials
  aws_endpoint_url_s3: <S3_ENDPOINT_URL>
  aws_access_key_id: <AWS_ACCESS_KEY_ID>
  aws_secret_access_key: <AWS_SECRET_ACCESS_KEY>
  bucket_name: <S3_BUCKET_NAME>
  aws_region: <S3_REGION>
  # Optional: use ONE of the following if your endpoint uses a private or self-signed␣
↪CA
  aws_ca_bundle: <CA_BUNDLE_PEM>          # raw PEM format
  # aws_ca_bundle_base64: <CA_BUNDLE_BASE64>  # base64-encoded PEM (alternative)
  # Set to true if server-side encryption is disabled on the bucket
  server_side_encryption_disabled: <true|false>
  # Required for on-premises storage that does not support checksum headers; added in␣
↪HM 2025.12
  aws_request_checksum_calculation: when_required
  aws_response_checksum_validation: when_required
```

Parameters:

Parameter	Required	Description
auth_type	Yes	Must be *credentials*.
aws_endpoint_url_s3	Yes	Full URL of the S3-compatible endpoint, including protocol. For example, *https://minio.example.com*.
aws_access_key_id	Yes	Access key ID for the S3-compatible storage.
aws_secret_access_key	Yes	Secret access key.
bucket_name	Yes	Name of the bucket.
aws_region	Yes	Region identifier. For storage that doesn't use regions, use a placeholder such as *us-east-1*.
aws_ca_bundle	No	CA bundle in raw PEM format. Use this or *aws_ca_bundle_base64* — not both. Required if your S3 endpoint uses a certificate signed by a private or self-signed CA.
aws_ca_bundle_base64	No	CA bundle as a base64-encoded PEM string. Use this or *aws_ca_bundle* — not both. See *Providing a CA bundle* for instructions.
server_side_encryption_disabled	No	Set to *true* if server-side encryption is disabled on the bucket. Defaults to *false*.
aws_request_checksum_calculation	No	Controls when the AWS SDK calculates checksums for requests. Set to *when_required* for on-premises storage that doesn't support checksum headers. This resolves the *x-amz-content-sha256* error seen with some S3-compatible providers. Added in HM 2025.12.
aws_response_checksum_validation	No	Controls when the AWS SDK validates response checksums. Set to *when_required* alongside *aws_request_checksum_calculation*. Added in HM 2025.12.

4.1.2.6.2 Providing a CA bundle

If you are using *Other S3-compatible storage* and your endpoint uses a certificate signed by a private or self-signed certificate authority (CA), you must provide the CA bundle. Add it to the `aws_ca_bundle` or `aws_ca_bundle_base64` field in your secret. Two fields are supported — use one or the other, not both:

Field	Format	When to use
aws_ca_bundle	Raw PEM text	You have the *.pem* or *.crt* file and can paste its contents directly.
aws_ca_bundle_base64	Base64-encoded PEM	You need to embed the bundle as a single-line string.

Using `aws_ca_bundle` (raw PEM)

Paste the contents of your CA bundle file directly as the value. A PEM file starts with `-----BEGIN CERTIFICATE-----` .

```
stringData:
  aws_ca_bundle: |
    -----BEGIN CERTIFICATE-----
    MIIDazCCAlOgAwIBAgI...
    -----END CERTIFICATE-----
```

Using `aws_ca_bundle_base64` (base64-encoded PEM)

Encode your CA bundle file before using it in the secret.

On Linux:

```
base64 -w 0 /path/to/ca-bundle.pem
```

On macOS:

```
base64 -i /path/to/ca-bundle.pem
```

Copy the output and set it as the value for `aws_ca_bundle_base64` .

!!!note If your CA bundle contains a certificate chain (intermediate + root CAs), include all certificates in the PEM file before encoding, concatenated in order from the issuing CA to the root.

If you are using a GCP configuration , you must encode your service account JSON key file as a base64 string before adding it to the secret. Use the following commands to generate that value.

To encode your credentials file:

```
base64 -w 0 /path/to/service-account-key.json
```

On macOS, use:

```
base64 -i /path/to/service-account-key.json
```

4.1.2.6.3 Configuring per-project object storage

By default, all projects share the object storage configured during installation. You can configure dedicated object storage for individual projects at any time after installation, before creating any Postgres workloads in that project. This isolates each project's backup and WAL data.

For step-by-step instructions — including how to retrieve the project ID, normalize it, and apply the secret — see *Configure dedicated object storage for a project* .

4.1.2.7 Deploying Hybrid Manager across multiple data centers

You can configure your HM installation across multiple data centers (multi-DC topology) to meet your high availability requirements. This topology includes a primary HM and one or more data-only HMs (secondaries) across multiple Kubernetes clusters.

- *Overview* — Reviews multi-DC concepts, explores the available multi-DC topologies, and provides an architecture review of a multi-DC HM installation.
- *Configuring a multi-DC deployment* — Provides a step-by-step guide of how to configure your HM installation across multiple data centers, and then provision a Postgres cluster across locations.

4.1.2.7.1 Multi-DC for high availability

Deploying Hybrid Manager (HM) across multiple data centers (multi-DC) provides high availability and disaster recovery for your Postgres database clusters. This setup allows you to manage databases across different geographical locations from a single interface.

!!!important This excludes the recovery of the HM installation itself, which you must *back up and restore separately* .

Deploying HM across multiple data centers gives you high availability and disaster recovery for Postgres database clusters. It allows you to:

- Survive a database loss (DR) — Keep one or two warm secondary Postgres databases ready. If the primary data center is unavailable, promote replicas in the secondary location and restore services.
- Minimize downtime (RTO) — Leverage multiple, geographically distributed, write endpoints (with DHA Postgres clusters) or failover quickly (with single-node, HA, and AHA Postgres clusters) to rapidly restore services during maintenance or unplanned site outages.
- Protect data (RPO) — Continuous replication to a second data center reduces potential data loss compared to single-site backups only.
- Reduce blast radius — Faults, misconfigurations, or noisy neighbors in one data center don't take down the other.
- Meet compliance/sovereignty — Keep copies in a specific region or facility while still centralizing control.
- Operate at scale — Split read traffic, stage upgrades, or run blue/green cutovers across data centers.
- Centralized management — A single primary HM manages databases across all remote Kubernetes clusters, simplifying administration while providing geo-redundancy.

Architecture and roles: Hub and Spoke

HM uses a Hub and Spoke model to manage multiple data centers. It is important to distinguish between the control plane (the management software) and the data plane (your actual Postgres databases).

The primary (The Hub)

The primary location is the "source of truth" and the only site where the full HM control plane is active. The following processes are owned by the primary or hub:

- **HM console (UI):** Your "single pane of glass." You log into the primary's URL to manage databases across all regions.
- **Central scheduler:** The primary decides where database nodes should live and sends provisioning instructions to the secondaries.
- **Identity provider:** Acts as the root of trust for SPIRE, issuing secure identities to all secondary locations.
- **Observability hub:** Uses Thanos and Loki to aggregate metrics and logs from every location into a single view.

The secondaries (The Spokes)

Secondaries are "lean" installations designed for execution rather than management. They don't run their own independent consoles, but they do run components that facilitate the communication and management from the primary or hub:

- **HM-internal Beacon agent:** Instead of a UI, secondaries run the HM-internal Beacon agent. This agent heartbeats back to the primary, making local Kubernetes resources (nodes, storage) available to the central scheduler.
- **Local execution:** When you deploy a cluster to a secondary location, the primary instructs the secondary's local operators to provision the Postgres pods.

Data plane & storage

While the control plane is Hub-and-Spoke, your data stays synchronized across the environment:

- **Postgres clusters:** You can deploy a primary Postgres cluster in DC-A and asynchronous Replicas in DC-B and DC-C. For Distributed High Availability (PGD), you can spread Data Groups and Witness nodes across all three locations.
- **Unified storage:** All locations must share a consistent object store configuration. This ensures that backups and WAL files generated in one data center are immediately available for recovery in another.

Topologies

You can deploy the multi-DC setup with the following topologies:

- **Primary and data-only HM:** This is the basic setup with one primary HM and at least one data-only HM. It allows for Postgres cluster deployments across two locations.
- **Consolidated management with multiple data-only HMs:** Manage databases across multiple remote Kubernetes clusters from a single primary HM. This architecture provides a unified control plane while maintaining a lightweight, data-only footprint in each target location.
- **Three HMs for PGD:** For optimal PGD deployments (Distributed and Advanced High Availability Postgres clusters), especially with two data groups and a witness group, we recommend a three-location topology. This configuration ensures quorum for proper distributed consensus.

Resilience and failover

Because you can mix and match different database topologies across your data centers, the impact of a site failure varies. Failover is performed per database cluster, not for the entire Hybrid Manager environment at once. Here are some examples:

Database topology	Setup	Recovery action	RTO (Time)	RPO (Data)
Single-Node	No replicas, located in DC-A.	**Full restore:** Must recreate the cluster from object storage/backup.	**High:** depends on DB size and restore speed.	**Medium:** Up to the last backup/WAL archive.
Single-Node + Replica	Single-node in DC-A, replica in DC-B.	**Manual promotion:** Promote the replica in DC-B to a standalone primary.	**Low:** is driven by runbook and DNS updates.	**Very low:** Seconds (based on async replication lag).
HA cluster	Primary/Replicas in DC-A, replica in DC-B.	**Manual promotion:** Promote the cross-DC replica in DC-B to primary.	**Low:** is driven by runbook and DNS updates.	**Very low:** Seconds (based on async replication lag).
DHA cluster	Data Groups in DC-A and DC-B, witness in DC-C.	**Automatic:** PGD maintains quorum; surviving nodes stay writable.	**Near zero:** Instant, limited only by DNS/LB TTL.	**Zero:** If using sync commit, otherwise near-zero.

The differences in the recovery depend on both the multi-DC setup as well as the topology you have chosen for each specific database cluster.

Local HA vs. Multi-DC Failover: From a disaster recovery perspective, both single-instance and HA clusters require a promotion to a secondary location if the primary site fails. However, an HA cluster (with local replicas) provides internal resilience; it can automatically recover from a primary instance failure within the same data center. A single-instance deployment, lacking a local standby, requires the database to completely restart or undergo a full cross-location failover to recover from any local interruption.

RTO (Recovery Time Objective): This is your downtime. For any manual promotion (Single-node + Replica or HA cluster), your RTO is limited by how fast you can run your failover playbook.

RPO (Recovery Point Objective): This is your data loss. For async replication, which is standard for multi-DC, this typically results in an RPO of a few seconds. In the case of sync replication, you can achieve zero RPO, but this adds latency to every write as it waits for the remote data center to acknowledge the data.

Network latency in Distributed High Availability (DHA) clusters

There's an inherent trade-off between how quickly Distributed High Availability clusters (powered by PGD) can react to consensus issues and the amount of network latency present. Higher latency requires more relaxed consensus settings, potentially increasing the time to detect and respond to failures. Some Raft settings may not be directly configurable through the HM console.

Latency is less of a concern in on-premises, tightly coupled environments compared to cloud environments that span regions. You can configure more relaxed Raft settings to accommodate high-latency scenarios.

You can use the `pg` CLI to interact with the cluster and adjust configuration settings that aren't exposed in the console, particularly Raft settings. Use the `bdr run on nodes` function to execute commands on other nodes.

HM backups and DR for multi-DC environments

HM uses two methods to protect your data. In a multi-DC setup, these roles are specific:

Barman (Cross-DC): Backups are stored in a global object store. This is your primary tool for disaster recovery across locations.

Volume snapshots (Local): Disk-based backups local to a specific data center. Use these for fast "point-in-time" recovery within a healthy site.

To ensure you can recover both your Hybrid Manager control plane and your Postgres databases during a site outage, verify the following:

- Ensure your primary HM metadata (configurations, users, and cluster definitions) is backed up and stored outside the primary data center.
- All HM instances (primary and secondaries) must have an identical `edb-object-storage` secret to access the shared backups live.
- You must configure *automated backup schedules* for each location individually, settings don't sync automatically between sites.
- Your organization knows how to *restore a database using the CLI* if the HM console is down due to a primary site's outage.

Current limitations

These are the current limitations for multi-DC deployments:

- Up to three locations (primary and one secondary, or a primary and two secondaries).
- Manual failover: Promote replicas in one of the secondary locations if the primary is down.
- Same cloud/on-prem family: Cross-CSP multi-DC is not supported.

4.1.2.7.2 Configuring a Hybrid Manager deployment across multiple data centers

To be able to deploy Postgres databases into multiple data centers, you first must deploy HM installations into two or three different geographical locations or regions, supporting the *Hub & Spoke or Primary & Secondary architecture* . You then link these installations to each other, so you can manage them from a single HM console. This guides help you:

- Create all HM installation configuration files, and deploy them.
- Connect two or more HM installations (HM Kubernetes clusters) on the same provider/on-prem family. These can be on different geographical regions. For example, you can have a primary and one secondary, or a primary and two secondaries.
- Align object storage (identical `edb-object-storage` secret), so backups/artifacts are usable in all data centers.
- Wire the HM-internal agent (Beacon, or `upm-beacon-agent`) so the secondary can register to the primary as managed locations and provision there (9445/TCP).
- Prepare a Postgres topology with a primary Postgres cluster in one data center and replica Postgres cluster(s) in the other(s); perform manual failover by promoting replicas.

Before you start

Before starting the multi-DC procedure, ensure you have made yourself familiar with:

- *Overview*

Prerequisites

Architecture prerequisites

This multi-DC setup follows a Hub and Spoke model, where a single Primary manages several lean Secondaries.

- Hub cluster: One Kubernetes cluster to host the Primary HM. This cluster serves as the central management plane and UI.
- Spoke clusters: One or two Kubernetes clusters to host Secondary HMs. These act as execution points for your database workloads.
- Network connectivity:
  - 8444/TCP open between clusters (SPIRE bundle endpoint).
  - 9445/TCP from secondaries → primary (Beacon gRPC).
  - Same provider/on-prem family (no cross-cloud).
- Shared object storage: The standard HM installation requires each individual Kubernetes cluster (HM installation) to have dedicated object storage. In the case of a multi-DC deployment, this object storage is shared between all clusters.

Collect the required information

1. Prepare two (or three) copies of your HM installation configuration file. Name them `primary.yaml` , `secondary.yaml` if you are deploying in two locations, or `primary.yaml` , `secondaryA.yaml` and `secondaryB.yaml` if you are deploying in three locations. These files are `HybridControlPlane` CR manifests.
2. Domain names:

Each HM installation must be configured with a dedicated domain name, set as `portal_domain_name` in the configuration. This parameter is used by the primary and secondaries.

Create these domain names for the each of the HM installations (two or three installations, depending on your configuration), and record this information.

Object storage across locations

HM uses an object store for backups, artifacts, WAL, and internal bundles. **In multi-DC, all HM installations must use the same object store configuration**.

Key requirement

All HM installations must have an identical Kubernetes secret named `edb-object-storage` in the default namespace. Store this secret in the primary and secondary if you are running on two locations. Store this secret in the primary, and secondary locations, if you have three data centers.

Parameter uniqueness

Because the two or three clusters involved in this multi-data-center deployment are using the same object store, we must ensure that specific parameters are different between all clusters.

Parameter	CR path	Requirement	Why it matters
beacon_location_id	*spec.componentsParameters.upm-beacon.beac*	Must be unique per location.	Human-readable location identifier. Identifies the specific location in the HM console and API.

continues on next page

Table 40 – continued from previous page

Parameter	CR path	Requirement	Why it matters
internal_backup_folder	*spec.globalParameters.internal_backup_folder*	Must be unique per location AND match the format: *^[0-9a-z]{12}$*.	Separates database backups so that a restore doesn't pull the wrong data.
metrics_storage_prefix	*spec.globalParameters.metrics_storage_prefix*	Must be unique per location.	Ensures observability data from each site is stored in its own directory.
logs_storage_prefix	*spec.componentsParameters.upm-loki.logs_sto*	Must be unique per location.	Prevents logs from different locations from overwriting each other in shared storage.

Configure the multi-DC topology

In this step, you define the Hub-and-Spoke relationship. The Hub (primary) must know about all its Spokes (secondaries), and each Spoke must know how to reach the Hub.

Add the following stanza to each of your configuration files (`primary.yaml` and `secondary.yaml`). Fill in the `domainName` values using the `portal_domain_name` you previously set for each HM installation.

In `primary.yaml` : Set role: `primary` . This cluster will act as the Hub.

In `secondary.yaml` : Set role: `secondary` . This cluster will act as a Spoke and use the `primary.domainName` to find its manager.

Add the `clusterGroups` stanza under `spec` in each `HybridControlPlane` CR:

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  clusterGroups:
    # Set to primary for your Hub, secondary for your Spokes
    role: (secondary|primary|standalone)
    primary:
      domainName: <primary portal domain>
    secondaries:
    - domainName: <secondary portal domain>
      # Add an additional entry here for additional locations
  # ... your other spec fields
```

Reduced set of components (Spoke clusters only)

The HM consists of a number of different components, some of which are not necessary on the secondary locations. While the full set can be installed successfully on secondary locations, we recommend to reduce that list by setting the parameter `scenarios` to be just `core` , and disabling the HM console (UI).

Add the following parameters to your secondary location configuration files:

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
```

(continues on next page)

(continued from previous page)

```
  name: edbpgai
spec:
  scenarios:
    - core
  disabledComponents:
    - upm-ui
  # ... your other spec fields
```

1. Validation checklist:

- The `edb-object-storage` secret must be identical across all locations (compare .data only).
- All locations can list/write the bucket (quick Pod/Job test).
- `beacon_location_id` , `internal_backup_folder` , `metrics_storage_prefix` , and `logs_storage_prefix` must be unique per location.
- `scenarios` has been set to be `core` for secondary locations.
- `disabledComponents` has the `upm-ui` component listed for secondary locations.

Hybrid Manager installation

Using your configuration files (`primary.yaml` and `secondary.yaml` , or `primary.yaml` , `secondaryA.yaml` and `secondaryB.yaml` for a three-location setup), install the Hybrid Manager on each cluster. Follow *Phase 5: Installing Hybrid Manager* for the full installation procedure.

Validate wiring

Once the installations are complete, verify the Hub and Spoke link.

1. On the primary, list the managed secondary locations:

```
kubectl get location
```

Each managed location appears as a row named `managed-<LOCATION_ID>` (for example, `managed-primary` and `managed-secondary`), with the location ID under `LOCATIONID` and a recent `LASTHEARTBEAT` :

```
NAME                SCOPE     LOCATIONID   LASTHEARTBEAT
managed-primary     managed   primary      2026-01-15T10:30:00Z
managed-secondary   managed   secondary    2026-01-15T10:30:00Z
```

1. Validate that SPIRE federation is present on all locations:

edb_notranlate_4 You should see a `federation list` showing the relationships (the peer trust domain) with `bundle endpoint profile: https_spiffe` and the peer's `:8444` URL.

On the primary, you should see one or two entries in the federation list (one for each secondary).

On each secondary instances, you will only see one entry (the relationship back to the primary).

Create a Postgres database cluster across different data centers

HM can now provision into the secondary locations, but you must still choose and create the actual database topology. In the HM console, create a database cluster and make sure to select different locations for each database node.

- For single-node and high availability database clusters, during *Replica cluster creation fails when using volume snapshot recovery across regions* , use the ***Replica Clusters** tab* to add replica database clusters in other locations.
- For advanced high availability and distributed high availability database clusters, during *Replica cluster creation fails when using volume snapshot recovery across regions* , use the ***Data Groups** tab > **Node Settings** > **Deployment Location*** option to distribute the cluster data groups across locations.

!!!note Witness nodes in PGD require a minimum of 10GB of disk space. .. ::
Ensure backups are writing to the shared object store from both data centers.

Operational notes

- DB TLS is separate from SPIRE/Beacon (platform identity). Configure PG TLS per your policy.
- Verify StorageClasses in each DC meet PG IOPS/latency.
- Open replication ports between sites.

Validation (end-to-end)

On the primary location, perform the following checks to validate the multi-DC setup:

1. Validate primary/secondary cluster relationships:

```
kubectl -n spire-system exec svc/spire-server -c spire-server -- \
/opt/spire/bin/spire-server federation list
```

1. Validate that the secondary locations are registered:

```
kubectl get location
```

1. Validate provisioning to secondary works:

- From the primary, deploy a small test workload to a secondary location. - Telemetry (optional) Thanos stores show federated peer; Loki queries return logs tagged from secondary. - Object storage Both clusters can read/write the bucket; secrets are identical.

Manual failover

Manual failover procedure for databases from the primary location to a secondary location

1. Suspend writes to the primary location (maintenance mode/LB cutover).
2. Promote a replicas in a secondary location to primary (using the HM console or your scripts).
3. Redirect clients (DNS/LB) to the secondary location.
4. Observe: confirm that writes succeed; and the replication role is updated.
5. When original the primary location returns: re-seed it as a replica of the new primary; optionally plan a later cutback.

!!!note While you promote the Postgres database in the secondary DC, the HM console itself still runs on the primary. If the primary DC (where HM lives) is completely gone, follow the *HM disaster recovery guide* to restore the management console elsewhere.

- Keep DNS TTL low enough for cutovers.
- Track downtime to measure RTO.

- Validate backups post-promotion.

Troubleshooting

- Problem: **No federation relationships**
  - Re-generate and cross-apply `ClusterFederatedTrustDomain` CRs.
  - Confirm 8444/TCP reachability.
- Problem: **secondary not listed in ``kubectl get location``**
  - Recheck Beacon values on both sides; restart Beacon server/agent.
  - Confirm 9445/TCP reachability to primary portal; trust domains correct.
- Problem: **Object store access fails on secondary**
  - Re-sync `edb-object-storage` .
  - For EKS/IRSA: ensure secondary OIDC is in the role's trust policy.
- Problem: **Telemetry federation missing**
  - Reinstall with the correct -l primary|secondary flags and unique prefixes.
  - Check Thanos `/api/v1/stores` and Loki read API.
- Problem: **Replica lag / connectivity** - Verify network ACLs/SGs, TLS certs, and storage performance.

Appendix B — Quick daily checks

- `kubectl get location` on primary shows secondary Ready.
- Thanos/Loki federation healthy (if enabled).
- Object store writes succeed from both DCs.
- Replication lag within SLOs.

4.1.2.8 Enabling Key Management Systems (KMS) for TDE

When creating new clusters, you can enable **Transparent Data Encryption (TDE)** to encrypt the data stored in your databases.

To secure the data encryption key , you can either enter a passphrase of your choice, or bring your own key from a key management provider such as HashiCorp Vault or AWS Key Management Service.

!!!note TDE is available for EDB Postgres Advanced Server and EDB Postgres Extended Server for version 15 and later.

- *Adding KMS support*

Then, configure your Key Management System provider:

- *HashiCorp Vault*
- *AWS KMS*
- *Azure KMS*
- *Creating the Google Cloud KMS key*

4.1.2.8.1 Adding KMS support

When you deploy the Hybrid Manager (HM), the `HybridControlPlane` Custom Resource includes the capability of using passphrases or HashiCorp Vault keys for TDE implementation by default:

```
# - transparentDataEncryptionMethods: List of TDE Encryption Methods for the agent.
#     Supported options: passphrase, aws_kms, azure_kms, gcp_kms, hashicorp_vault.
#     Default values: passphrase, hashicorp_vault.
transparentDataEncryptionMethods:
  - passphrase
  - hashicorp_vault
```

If you want to use a different Key Management System for TDE, you can edit this section during installation. Alternatively, you can add the key provider after installing the Hybrid Manager. In that case, you'll have to reapply the `HybridControlPlane` CR with the updated configuration to populate the system with the new settings. See the installation guides for infrastructure-specific instructions on how to apply the CR.

HashiCorp Vault

The Hybrid Manager configuration includes support for HashiCorp out of the box.

AWS KMS

To use Amazon's KMS, Amazon Key Management Service, add `aws_kms` to the list:

```
# - transparentDataEncryptionMethods: List of TDE Encryption Methods for the agent.
#     Supported options: passphrase, aws_kms, azure_kms, gcp_kms, hashicorp_vault.
#     Default values: passphrase, hashicorp_vault.
transparentDataEncryptionMethods:
  - passphrase
  - hashicorp_vault
  - aws_kms
```

Azure KMS

To use Azure's KMS, Azure Key Vault, add `azure_kms` to the list:

```
# - transparentDataEncryptionMethods: List of TDE Encryption Methods for the agent.
#     Supported options: passphrase, aws_kms, azure_kms, gcp_kms, hashicorp_vault.
#     Default values: passphrase, hashicorp_vault.
transparentDataEncryptionMethods:
  - passphrase
  - hashicorp_vault
  - azure_kms
```

Google KMS

If you want to use Google's KMS, Cloud Key Management Service, add `gcp_kms` to the list:

```
# - transparentDataEncryptionMethods: List of TDE Encryption Methods for the agent.
#     Supported options: passphrase, aws_kms, azure_kms, gcp_kms, hashicorp_vault.
#     Default values: passphrase, hashicorp_vault.
transparentDataEncryptionMethods:
```

(continues on next page)

(continued from previous page)

```
  - passphrase
  - hashicorp_vault
  - gcp_kms
```

Next steps

- *Configure HashiCorp Vault as the KMS for TDE*
- *Configure AWS Key Management Service as the KMS for TDE*
- *Configure Azure Key Vault as the KMS for TDE*
- *Configure Google Cloud Key Management Service as the KMS for TDE*

4.1.2.8.2 HashiCorp Vault

Before you can create TDE-enabled clusters with a HashiCorp Vault key, an administrator must install and configure Vault for the Hybrid Manager instance.

This example installs a customized version of Vault on the same Kubernetes environment as the Hybrid Manager, configures the Vault provider, and then uses the key to enable TDE for provisioning a new database cluster with encryption enabled.

If you're aiming for a vendor-neutral key management solution, it can be practical to use HashiCorp Vault, as it provides consistent, centralized key management across diverse cloud providers and on-premises environments, preventing vendor lock-in.

Installing Vault

Install HashiCorp Vault on the same Kubernetes environment where you have deployed the Hybrid Manager. Ensure you install it on its own `vault` namespace.

1. Set up the Helm repository:

```
helm repo add hashicorp https://helm.releases.hashicorp.com
```

2. Create a customization override for the installation to ensure Vault is installed on the Hybrid Manager's control plane, port 8200:

```
cat > helm-vault-raft-values.yml <<EOF
injector:
 tolerations: |
   - key: "edbaiplatform.io/control-plane"
     effect: NoSchedule
     operator: Equal
     value: "true"
```

```
 affinity: |
   nodeAffinity:
     preferredDuringSchedulingIgnoredDuringExecution:
     - preference:
         matchExpressions:
         - key: edbaiplatform.io/control-plane
           operator: Exists
```

(continues on next page)

(continued from previous page)

```
      weight: 1
server:
 tolerations: |
   - key: "edbaiplatform.io/control-plane"
     effect: NoSchedule
     operator: Equal
     value: "true"

 affinity: |
   nodeAffinity:
     preferredDuringSchedulingIgnoredDuringExecution:
     - preference:
         matchExpressions:
         - key: edbaiplatform.io/control-plane
           operator: Exists
       weight: 1
 ha:
     enabled: true
     raft:
        enabled: true
        setNodeId: true
        config: |
           cluster_name = "vault-integrated-storage"
           storage "raft" {
              path    = "/vault/data/"
           }

           listener "tcp" {
              address = "[::]:8200"
              cluster_address = "[::]:8201"
              tls_disable = "true"
           }
           service_registration "kubernetes" {}
EOF
```

3. Install Vault using the override:

```
helm install -n vault vault hashicorp/vault --values helm-vault-raft-values.yml  --
↪create-namespace
```

4. Wait until all pods reach the "Healthy" status before proceeding:

```
kubectl -n vault get pods
```

Now you have to initialize your Vault instance using your base64-encoded public PGP key or Keybase username and unseal it with the master key. See Vault's documentation to Initialize and unseal Vault .

Configuring Vault and creating a key

Enable communication between the Vault service and the Hybrid Manager:

1. Access the pod where Vault is installed:

```
kubectl -n vault exec --stdin=true --tty=true vault-0 -- /bin/sh
```

2. Enable Kubernetes communication for the vault instance:

```
vault auth enable kubernetes
```

3. Update Vault to authenticate with the Kubernetes cluster and allow the Hybrid Manager to use Vault:

```
vault write auth/kubernetes/config \
  token_reviewer_jwt="$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" \
  kubernetes_host=https://${KUBERNETES_PORT_443_TCP_ADDR}:443 \
  kubernetes_ca_cert=@/var/run/secrets/kubernetes.io/serviceaccount/ca.crt
```

4. Enable the *Transit* secrets engine to activate key management capabilities:

```
vault secrets enable transit
```

5. Create a new file that contains the Vault policy definition for the keys:

In this example the file with the policy is called `tde.txt` .

```
cat > /tmp/tde.txt << EOL
path "transit/*" {
  capabilities = ["create", "read", "update", "patch", "delete", "list"]
}
EOL
```

6. Use the created file to update HashiCorp Vault's security policy:

```
vault policy write tde /tmp/tde.txt
```

7. Review the default parameter values for the Transit engine (e.g. around key rotation, encryption method, etc.) and

adjust according to your requirements
. Alternatively, can use the

vault write command during key
creation in the following step to override the default values for a specific key.

8. Create a new key that uses the transit engine:

Replace `<my-tde-key>` with a key name of your choice.

```
vault write -f transit/keys/<my-tde-key>
```

9. Create a role called `tde` that maps Vault policies to Kubernetes service accounts:

```
vault write auth/kubernetes/role/tde \
  bound_service_account_names="p-*" \
  bound_service_account_namespaces="p-*" \
  policies=tde \
  ttl=72h
```

> **Note**
>
> You can configure a different *ttl* value to specify how often an application will have to re-authenticate. See Vault's documentation on

token metadata for more information.

10. Exit the pod:

```
exit
```

Now that you have configured Vault and created a key, add it to a project in the PG AI Console.

Assigning the Vault key to a project

Add the created key to a project in Hybrid Manager using the console:

1. On the Console, go to the project under which you'll create TDE-enabled clusters. You can use a key for several clusters in a project, but we recommend setting up a new key per additional Hybrid Manager project.
2. On the left-side navigation, select **Settings ► Security**, and then **+ Add a key**.
3. Select a location to configure the key.
4. Select **HashiCorp Vault** under **Select key management system provider**.
5. In **Vault Address**, enter the HashiCorp Vault server endpoint.

If Vault was installed in the `vault` namespace, and in the same Kubernetes environment as the Hybrid Manager, the default Kubernetes endpoint for Vault will be `http://vault.vault.svc.cluster.local:8200/` . If your setup is different, infer the endpoint from your environment like this: `<pod_name>.<namespace>.svc.cluster.<host>:<port_configured_for_vault>` .

6. In **Vault Role**, enter the name of the role you you created earlier, namely `tde` .
7. In **Vault Key**, enter the name of the key you created earlier (`<my-tde-key>`).
8. If desired, you can enter an alternative key name that is easy to remember in **(Optional) Enter a friendly name for your key**.
9. Select **Add Key**.

You can now use this key if you want to enable encryption when you create clusters. The added key will appear as an option when you *enable TDE during cluster creation* .

4.1.2.8.3 AWS Key Management Service (KMS)

Before you can create TDE-enabled clusters with an AWS Key Management Service key, an administrator must install and configure AWS KMS for the Hybrid Manager instance.

This example creates a symmetric customer-managed key in AWS, configures the KMS provider in the Hybrid Manager, and then uses the key to enable TDE for provisioning a new database cluster with encryption enabled.

If you are running the Hybrid Manager on Amazon Elastic Kubernetes Service (EKS), it can be practical to use AWS KMS for key management, as it provides a native integration and alignment with the AWS ecosystem's best practices.

Prerequisites

Ensure you have set the parameter for AWS KMS in the `HybridControlPlane` CR of the Hybrid Manager. See *Adding KMS support* to enable it.

Creating the AWS KMS key

Create a symmetric customer-managed key using AWS Console, AWS CLI or AWS API in the same AWS account and region where the Hybrid Manager is installed.

Use the following configuration for the new key:

- **Key type**: Symmetric
- **Key usage**: Encrypt and decrypt
- **Alias**: Assign a user-friendly alias for the key (do not enter sensitive information)
- **Key policy**: Ensure the main IAM user or role configured for the Hybrid Manager in the Kubernetes cluster has permissions to encrypt and decrypt keys.

```
{
    "Sid": "Allow HM identity to use this KMS key",
    "Effect": "Allow",
    "Principal": {
        "AWS": "<IAM-user-or-role-arn>"
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt"
    ],
    "Resource": "*"
}
```

Now that you have created a customer-managed KMS key, add it to a project in the Console.

Assigning the AWS KMS key to a project

Add the created key to a project:

1. On the Console, go to the project under which you'll create TDE-enabled clusters. You can use a key for several clusters in a project, but we recommend setting up a new key per additional Hybrid Manager project.
2. On the left-side navigation, select **Settings ► Security**, and then **+ Add a key**.
3. Select a location to configure the key.
4. Select **AWS** under **Select key management system provider**.
5. In **Enter AWS Key Management System ARN**, enter the AWS KMS key ARN. See Find the key ID and key ARN in AWS KMS documentation.
6. If desired, you can enter an alternative key name that is easy to remember in **(Optional) Enter a friendly name for your key**.
7. Select **Add Key**.

You can now use this key if you want to enable encryption when you create clusters. The added key will appear as an option when you *enable TDE during cluster creation* .

4.1.2.8.4 Azure Key Vault (Key Management Service)

Before you can create TDE-enabled clusters with an Azure Key Vault key, an administrator must install and configure Azure Key Vault for the Hybrid Manager instance.

This example creates a customer-managed key in Azure Key Vault, configures the KMS provider in the Hybrid Manager, and then uses the key to enable TDE for provisioning a new database cluster with encryption enabled.

If you are running the Hybrid Manager on Azure Kubernetes Service (AKS), it can be practical to use Azure Key Vault for key management, as it provides a native integration and alignment with the Azure ecosystem's best practices.

Prerequisites

Ensure you have set the parameter for Azure KMS in the `HybridControlPlane` CR of the Hybrid Manager. See *Adding KMS support* to enable it. Specifically, `azure_kms` must be present in the agent's `transparentDataEncryptionMethods` list, otherwise **Azure** won't appear as an option under **Select key management system provider** when adding a key in the Console.

Creating the Azure Key Vault key

Create a key in Azure Key Vault using the Azure portal, Azure CLI, or Azure API in the same Azure region where the Hybrid Manager is installed.

Use the following configuration for the new key:

- **Key type**: RSA or EC (as supported by your vault)
- **Key operations**: Ensure at minimum the following permitted operations are enabled:

Action	Purpose
encrypt / *decrypt*	Wrap and unwrap the data encryption key
wrapKey / *unwrapKey*	Key wrap operations
get / *read*	Read key metadata

Granting workload identity access to the key

The Postgres cluster uses TDE backed by your Azure Key Vault key. The workload identity associated with the cluster's pods must be authorized to perform cryptographic operations on that key.

Assign the built-in role **Key Vault Crypto User** to the workload identity. This role grants only the operations TDE needs:

Action	Purpose
Microsoft.KeyVault/vaults/keys/encrypt/action	Wrap the data encryption key
Microsoft.KeyVault/vaults/keys/decrypt/action	Unwrap the data encryption key
Microsoft.KeyVault/vaults/keys/wrapKey/action	Key wrap operations

continues on next page

Table 42 – continued from previous page

Action	Purpose
Microsoft.KeyVault/vaults/keys/unwrapKey/action	Key unwrap operations
Microsoft.KeyVault/vaults/keys/read	Read key metadata

Assign the role using Azure CLI:

```
az role assignment create \
    --role "Key Vault Crypto User" \
    --assignee-object-id <WORKLOAD_IDENTITY_OBJECT_ID> \
    --assignee-principal-type ServicePrincipal \
    --scope "/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/
↪providers/Microsoft.KeyVault/vaults/<VAULT_NAME>/keys/<KEY_NAME>"
```

Verify the assignment:

```
az role assignment list \
    --assignee <WORKLOAD_IDENTITY_OBJECT_ID> \
    --scope "/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/
↪providers/Microsoft.KeyVault/vaults/<VAULT_NAME>" \
    -o table
```

You should see a row with **Role: Key Vault Crypto User**.

Once you have created an Azure Key Vault key, add it to a project in the Hybrid Manager Console.

Assigning the Azure Key Vault key to a project

Add the created key to a project:

1. On the Console, go to the project under which you'll create TDE-enabled clusters. You can use a key for several clusters in a project, but we recommend setting up a new key per additional Hybrid Manager project.
2. On the left-side navigation, select **Settings** -> **Security**, and then + **Add a key**.
3. Select a location to configure the key.
4. Select **Azure** under **Select key management system provider**.
5. In **Key Vault Name**, enter the name of the Azure Key Vault that holds the key.
6. In **Key Name**, enter the key name in your Microsoft Azure account. This name must match the name of the key in Azure Key Vault.
7. In **Key Version**, enter the version of the key that will be used for encryption and decryption.
8. If desired, you can enter an alternative key name that is easy to remember in **(Optional) Enter a friendly name for your key**.
9. Select **Add Key**.

You can now use this key if you want to enable encryption when you create clusters. The added key will appear as an option when you *enable TDE during cluster creation* .

4.1.2.8.5 Google Cloud Key Management Service (Cloud KMS)

Before you can create TDE-enabled clusters with a Google Cloud Key Management Service key, an administrator must install and configure Google Cloud Key Management for the Hybrid Manager instance.

This example creates a customer-managed key in Cloud KMS, configures the key provider in the Hybrid Manager, and then uses the key to enable TDE for provisioning a new database cluster with encryption enabled.

If you are running the Hybrid Manager on Google Kubernetes Engine (GKE), it can be practical to use Cloud KMS for key management, as it provides a native integration and alignment with the Google ecosystem's best practices.

Prerequisites

Ensure you have set the parameter for Google Cloud KMS in the `HybridControlPlane` CR of the Hybrid Manager. See *Adding KMS support* to enable it.

Creating the Google Cloud KMS key

Create a customer-managed key using Google Cloud Console, CLI or API in the same project and region where the Hybrid Manager is installed. A key exists on exactly one key ring tied to a specific location.

Use the following configuration for the new key ring:

- **Key ring name**: choose a specific name for the key ring
- **Location type**: Region
- **Region**: Select the same region where Hybrid Manager is deployed

Create the key associated with the new key ring:

- **Key name**: choose a specific name for the key
- **Protection level**: Software
- **Purpose**: symmetric encrypt/decrypt

Review the default settings for the key (e.g. around key rotation, protection level, key material, versions, etc.) and adjust according to your requirements .

Configure the key permissions:

- Edit the key permissions and grant access for the Google service account that is used by Hybrid Manager. This is the account configured during the Hybrid Manager installation.
- Ensure encrypt and decrypt permissions are granted (e.g. use the Google built-in Cloud KMS CryptoKey Encrypter and Decrypter roles).

Now that you have created a Google Cloud KMS key, add it to a project in the Hybrid Manager Console.

Assigning the Google Cloud KMS key to a project

Add the created key to a project:

1. On the Console, go to the project under which you'll create TDE-enabled clusters. You can use a key for several clusters in a project, but we recommend setting up a new key per additional Hybrid Manager project.
2. On the left-side navigation, select **Settings ► Security**, and then **+ Add a key**.
3. Select a location to configure the key.
4. Select **Google Cloud** under **Select key management system provider**.
5. In **Region**, enter the region where Hybrid Manager is deployed, e.g. us-west-2.

6. In **Project ID**, enter the ID of the project containing the key ring.
7. In **Key ring name**, enter the exact name of the key ring as defined in your Google Cloud account.
8. In **Key name**, enter the exact key name as defined in your Google Cloud account.
9. If desired, you can enter an alternative key name that is easy to remember in **(Optional) Enter a friendly name for your key**.
10. Select **Add Key**.

You can now use this key if you want to enable encryption when you create clusters. The added key will appear as an option when you *enable TDE during cluster creation* .

4.1.2.8.6 Deleting a key

If you no longer need a key, you can delete it.

Go to the project where you previously added the key. On the left-side navigation, select **Settings ► Security**, and delete the key.

!!!note

You can't delete keys that are being used for encryption on existing clusters.

4.1.2.9 Optimizing service mesh memory

Hybrid Manager uses Istio as its service mesh to manage network traffic between Postgres clusters and other components. As the number of managed Postgres clusters grows, Istio's memory requirements increase. You can adjust the memory limits for Istio to optimize performance based on your expected workload.

4.1.2.9.1 Understanding Istio memory requirements

The memory requirements for Istio depend on several factors:

- **Number of Postgres clusters**: More clusters require more memory for service discovery and routing
- **Cluster architecture**: Complex architectures (such as PGD clusters with multiple data groups) require more memory than simple standalone clusters
- **Network traffic**: Higher network activity increases memory usage

By default, Hybrid Manager configures Istio with conservative memory limits. As your environment grows, you may need to adjust these limits to prevent performance degradation or out-of-memory errors.

4.1.2.9.2 Recommended memory limits

The following table shows the recommended memory limits based on the number of Postgres clusters you plan to manage:

Number of Postgres Clusters	Recommended Memory Limit
up to 10-20 clusters	512Mi
up to 30-40 clusters	1Gi
up to 60-70 clusters	1.5Gi
scale more if you need	> 1.5Gi

> **Note**
>
> These recommendations are guidelines. Your actual requirements may vary depending on your cluster architectures and network traffic patterns. Monitor Istio's memory usage and adjust accordingly. !!!

4.1.2.9.3 Configuring Istio memory limits

You can configure the Istio memory limits by setting the `memory_limit` and `cpu_limit` values for the `upm-istio` component.

> **Tip**
>
> Start with the recommended values for your expected cluster count. If you plan to scale beyond 70 clusters, increase the memory limit proportionally (for example, add approximately 512Mi per additional 30 clusters). !!!

Edit your `HybridControlPlane` CR to add or update the Istio memory settings under `spec.componentsParameters.upm-istio`:

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  componentsParameters:
    upm-istio:
      # Memory limit recommendations:
      # - 512Mi for 10-20 Postgres clusters
      # - 1Gi for 30-40 Postgres clusters
      # - 1.5Gi for 60-70 Postgres clusters
      # Adjust based on your expected load and architecture
      memory_limit: "512Mi"
      cpu_limit: "500m"
    # ... your other componentsParameters
```

Apply the updated CR:

```
kubectl apply -f hybridmanager.yaml
```

The operator reconciles the change and updates the Istio resource limits.

4.1.2.9.4 Monitoring Istio memory usage

After adjusting the memory limits, monitor Istio's performance to ensure the configuration is appropriate for your workload.

Using Grafana dashboards

Monitor Istio memory usage through the Grafana dashboards available in Hybrid Manager's Launchpad:

1. From the Hybrid Manager portal, open the user menu and select **Launchpad**.
2. On the **Grafana** card, select **View service**.
3. Navigate to **Dashboards** > **Kubernetes**.

4. Select the **Kubernetes / Compute Resources / Workload** dashboard.
5. Filter by:

- **namespace**: `istio-system`
- **workload**: `istiod` (for the Istio control plane)

Monitor the **Memory Usage** panel to see if usage is consistently approaching your configured limits. If it is, consider increasing the `memory_limit` value.

Check for memory-related issues

If Istio is experiencing memory pressure, you might see the following symptoms:

- Pods restarting frequently
- OOMKilled (Out of Memory Killed) events in pod descriptions
- Slow response times for Postgres cluster operations

To check pod status and events:

```
kubectl get pods -n istio-system
kubectl describe pod <pod-name> -n istio-system
```

Look for events mentioning `OOMKilled` or memory-related errors in the output.

4.1.2.10 API audit logging

EDB Postgres® AI Hybrid Manager (HM) can emit structured audit logs for authenticated API requests passing through the ingress gateway. When enabled, every API call that requires authentication produces a JSON log entry providing a detailed record of who accessed what, when, and from where. This functionality is useful for compliance reporting, security incident investigation, and access auditing.

API audit logging is available from HM 2026.4.

4.1.2.10.1 What gets audited

HM emits an audit log entry for a request when both these conditions are met:

- The request requires authentication. Anonymous or exempt endpoints are not audited.
- The request targets an API path, such as REST APIs (`/api/…`), Iceberg catalog APIs, launchpad application APIs, or gRPC service endpoints.

These request types aren't audited:

- Static resources (for example, JavaScript, CSS, images)
- Exempt paths, including `/api/ui` and `/api-docs`

4.1.2.10.2 Disabling API audit logging

API audit logging is enabled by default. To disable it, set the `enable_api_audit` setting for the `upm-istio-gateway` component to `"false"` during installation.

> **Note**
>
> The accepted values are the strings *"true"* and *"false"* only. Boolean types are not permitted. !!!

Edit your HybridControlPlane custom resource to add or update the `enable_api_audit` setting under `spec.componentsParameters.upm-istio-gateway`:

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  componentsParameters:
    upm-istio-gateway:
      enable_api_audit: "false"
```

Apply the updated custom resource:

```
kubectl apply -f hybridmanager.yaml
```

4.1.2.10.3 Re-enabling API audit logging

To re-enable API audit logging after disabling it, set `enable_api_audit` to `"true"` and reapply using the same steps.

4.1.2.10.4 Querying audit logs

Audit log entries are written to the ingress gateway's standard log output. You can query them using your cluster's log aggregation tool by filtering for the `EDB-HM-AUDIT` key.

Viewing audit logs in Grafana

Grafana is included in HM's observability stack and queries logs aggregated by Loki. You can access it as a service from the *Launchpad* in the HM console.

> **Note**
>
> To access Launchpad and Grafana, you must have the Platform Admin role. See *User role permissions by use case* for details. !!!

To view audit logs in Grafana:

1. From your user menu in the HM console, select **Launchpad**.
2. Select **Grafana**.
3. In Grafana, select **Explore** from the left side pane.
4. Select the **Loki** data source.
5. Run a LogQL query to filter audit entries. For example:

```
{namespace="istio-system", container="istio-proxy"} |= "EDB-HM-AUDIT" | json
```

Using kubectl

If you don't have a log aggregation tool available, use this command to stream logs directly from the ingress gateway pod for quick inspection:

```
kubectl logs -n istio-system -l istio=ingressgateway -c istio-proxy | grep EDB-HM-
↪AUDIT
```

4.1.2.10.5 Log format

Each audit log entry is a JSON object nested under the `EDB-HM-AUDIT` key. You can use this key to filter audit entries in your log aggregation system, for example, with Loki.

Example log entry:

```
{
  "EDB-HM-AUDIT": {
    "method": "GET",
    "uri": "/api/v1/projects?page=1",
    "user_id": "usr_abc123",
    "user_name": "alice",
    "key_id": "ak_def456",
    "session_id": "aksid_a3f8c2e1b9d04f67",
    "request_id": "a1b2c3d4-e5f6-7890",
    "user_agent": "curl/7.68.0",
    "ip": "10.0.0.1",
    "status_code": 200
  }
}
```

4.1.2.10.6 Log fields

Each audit log entry contains these fields. Fields with empty or unavailable values are omitted to reduce noise.

Field | Description ——|———— `method` | HTTP method (`GET` , `POST` , `PUT` , `DELETE` , etcetera). `uri` | Original request URI, including any query string. `user_id` | Authenticated user ID. Empty if authentication failed (for example, 401 response). `user_name` | Authenticated user display name. `key_id` | Access key ID. Present only for access-key-based authentication. Omitted for browser or OIDC sessions. `session_id` | Session identifier. For access key requests, this field is a deterministic hash derived from the key and client context, enabling correlation across requests from the same automated client. For browser sessions, this field is the OIDC session ID. `request_id` | Unique request trace ID (`x-request-id`). Useful for correlating audit entries with upstream service logs. `user_agent` | Client User-Agent header value. `ip` | Client IP address, extracted from the `x-forwarded-for` header. `status_code` | HTTP response status code (for example, 200, 401, 403, 500). `grpc_status` | gRPC status code. Present only for gRPC requests. The HTTP transport always returns 200. The actual outcome is reflected here (for example, 16 = Unauthenticated).

4.1.3 Migrating from bootstrap to operator installation

4.1.3.1 Overview

If you previously installed Hybrid Manager using the **bootstrap method** (`edbpgai-bootstrap` helm chart with `values.yaml`) and want to migrate to the **operator method** (`edb-hcp-operator` helm chart with `HybridControlPlane` CR), this guide provides the necessary steps.

Simply uninstalling the bootstrap **deletes all CRDs (Custom Resource Definitions)**, which will **cascade to delete all your database clusters and destroy your data**. Follow the CRD-preserving migration steps in this guide to avoid data loss.

4.1.3.2 Understanding the migration challenge

4.1.3.2.1 The Helm ownership conflict

If you try to install the operator on top of an existing bootstrap installation, you encounter an error like this:

```
Release "edb-hcp-operator" does not exist. Installing it now.
Error: Unable to continue with install: ServiceAccount "edb-hcp-operator-controller-
↪manager"
in namespace "edbpgai-bootstrap" exists and cannot be imported into the current␣
↪release:
invalid ownership metadata; annotation validation error: key "meta.helm.sh/release-
↪name"
must equal "edb-hcp-operator": current value is "edbpgai-bootstrap"
```

4.1.3.2.2 Why this happens

- The bootstrap installation creates Kubernetes resources owned by the `edbpgai-bootstrap` Helm release
- The operator installation tries to create resources owned by the `edb-hcp-operator` Helm release
- Helm prevents one release from taking ownership of another release's resources
- Both installations use overlapping namespaces and resource names

4.1.3.2.3 The CRD deletion problem

When you uninstall the bootstrap helm chart, it **deletes all Custom Resource Definitions (CRDs)**, which **cascade to delete all your database clusters and custom resources**. This must be avoided so that data is not lost.

- HybridControlPlane
- Preflight
- Postflight

If these CRDs are deleted, Kubernetes automatically deletes **all instances** of these custom resources, rendering your HM platform non-functional.

4.1.3.3 Prepare the migration with edbctl

The recommended way to migrate from the bootstrap installation to the operator installation is `edbctl hm prepare-bootstrap-operator-conversion`. The command does both halves of the migration in a single invocation:

1. **Exports** the current `HybridControlPlane` CR to a local YAML file (defaults to `hybridmanager.yaml`) so you have a starting point for the operator-managed manifest.
2. **Transfers Helm ownership** of the bootstrap-installed resources (CRDs, ClusterRoles, ClusterRoleBindings, Roles, RoleBindings, ServiceAccounts, Services, webhook configurations) from the `edbpgai-bootstrap` release to the `edb-hcp-operator` release, so a subsequent `helm install` /`edbctl hm upgrade-operator` adopts the existing resources cleanly without ownership errors.

CRDs and custom resources are preserved — your database clusters keep running throughout the migration.

4.1.3.3.1 Run the conversion

With your kubeconfig pointed at the cluster where the bootstrap-installed HM lives, run:

```
edbctl hm prepare-bootstrap-operator-conversion
__OUTPUT__
Migration prepared. Next steps:
  1. Review the generated CR: hybridmanager.yaml
  2. Install the operator chart for release "edb-hcp-operator" into namespace
↪"edbpgai-bootstrap" — the transfer
     stamped Helm ownership for that namespace, so install there (not elsewhere):
       helm install edb-hcp-operator <chart> -n edbpgai-bootstrap
     or install the latest operator with edbctl:
       edbctl hm upgrade-operator --release-name edb-hcp-operator --namespace edbpgai-
↪bootstrap --registry-uri <uri> --registry-username <user> --registry-password
↪<password>
  3. Apply the reviewed CR: kubectl apply -f hybridmanager.yaml
```

Defaults:

- `--namespace edbpgai-bootstrap` — namespace where the bootstrap release lives.
- `--operator-namespace <same as --namespace>` — namespace the operator chart will be installed into. The ownership transfer stamps this namespace, so the subsequent `helm install` /`edbctl hm upgrade-operator` **must** target it.
- `--source-release edbpgai-bootstrap` — Helm release currently owning the resources.
- `--target-release edb-hcp-operator` — Helm release to transfer ownership to.
- `--output hybridmanager.yaml` — file where the exported CR is written.

Override any of them if your installation uses non-default names. Use `--dry-run` to preview deletions and annotations without changing the cluster.

Important

Review the generated *hybridmanager.yaml* before applying it. In particular:

- ``**spec.version**`` : derived from the live CR. The operator uses its own versioning scheme — `spec.version` is the HM bootstrap version (e.g., `v1.4.0`), not the operator's own version (`2.0.0` or later from the `2.x` lineage).
- ``**spec.imagePullSecrets**`` : each entry needs both `name` and `namespace` ; verify they reference the right namespace if you store secrets outside `edbpgai-bootstrap` .

1. Review the generated `hybridmanager.yaml` .
2. Install (or upgrade to) the operator chart in the namespace the conversion stamped — `edbctl hm upgrade-operator --namespace edbpgai-bootstrap --registry-uri <REGISTRY_URI> --registry-username <USER> --registry-password <PASSWORD>` (auto-discovers the latest operator version from the registry). On edbctl `< 1.7.0` , fall back to `helm install edb-hcp-operator <chart> -n edbpgai-bootstrap` .
3. Apply the reviewed CR: `kubectl apply -f hybridmanager.yaml` . The exported file deliberately omits the bootstrap release's `meta.helm.sh/release-name=edbpgai-bootstrap` annotation that the live CR still carries; applying the file overwrites that stale annotation, so a later `helm uninstall edbpgai-bootstrap` (cleaning up the bootstrap release) does not cascade-delete the HybridControlPlane CR (and your database clusters with it).

4.1.3.4 Manual fallback for edbctl less than 1.7.0

For environments still on edbctl `< 1.7.0`, the previous manual flow still works. The rest of this guide preserves it for reference — but the `edbctl hm prepare-bootstrap-operator-conversion` path above is the recommended one.

4.1.3.4.1 Configuration mapping reference

The following table shows how `values.yaml` fields map to `HybridControlPlane` CR fields, if you need to hand-write the CR from a bootstrap `values.yaml`:

values.yaml field	HybridControlPlane field	Description
system	*spec.flavour*	Kubernetes distribution (eks, aks, gke, rhos, etc.)
containerRegistryURL	*spec.imageRegistry*	Container registry URL
bootstrapImageTag	*spec.version*	HM version to install
imagePullSecrets	*spec.imagePullSecrets[]*	Image pull secret references. In *values.yaml* this is a list of secret names (strings). In the CR, each entry requires *name* and *namespace* fields.
disabledComponents	*spec.disabledComponents[]*	Components to disable
*parameters.global.**	*spec.globalParameters*	Global configuration parameters
*parameters.<component>.**	*spec.componentsParameters.<component>*	Component-specific parameters
beaconServer	*spec.beaconServer*	Beacon server configuration
beaconAgent	*spec.beaconAgent*	Beacon agent configuration
clusterGroups	*spec.clusterGroups*	Multi-DC cluster topology
scenarios	*spec.scenarios[]*	Installation scenarios (core, migration, ai, analytics). In *values.yaml* this is a comma-separated string (e.g., *"core,migration,ai"*). In the CR, it is a YAML list.
resourceAnnotations	*spec.resourceAnnotations[]*	Custom resource annotations
source.remote	*spec.source.remote*	Use remote bootstrap image for manifests

4.1.3.4.2 Convert using the migration script

Use the following script to automatically generate a `HybridControlPlane` CR from your existing `values.yaml`.

Prerequisites: `yq` (version 4+) must be installed.

```
# !/bin/bash
#  convert-values-to-hcp.sh
#  Converts a bootstrap values.yaml to a HybridControlPlane Custom Resource manifest.
#
#  Usage: ./convert-values-to-hcp.sh <values.yaml> [output-file]
#    If output-file is omitted, prints to stdout.

set -euo pipefail

VALUES_FILE="${1:?Usage: $0 <values.yaml> [output-file]}"
OUTPUT_FILE="${2:-}"

if [ ! -f "$VALUES_FILE" ]; then
  echo "Error: File $VALUES_FILE not found." >&2
  exit 1
fi
```

(continues on next page)

(continued from previous page)

```
if ! command -v yq &> /dev/null; then
  echo "Error: yq (v4+) is required. Install from https://github.com/mikefarah/yq" >&2
  exit 1
fi

#  Build the HybridControlPlane CR using yq

HCP=$(yq -n
  .apiVersion = "edbpgai.edb.com/v1alpha1" |
  .kind = "HybridControlPlane" |
  .metadata.name = "edbpgai" |
  .metadata.annotations."biganimal.enterprisedb.io/deletion-protect" = "enabled"
)

#  Basic configuration mapping

SYSTEM=$(yq .system // "" "$VALUES_FILE")
[ -n "$SYSTEM" ] && HCP=$(echo "$HCP" | yq ".spec.flavour = \"$SYSTEM\"")

REGISTRY=$(yq .containerRegistryURL // "" "$VALUES_FILE")
[ -n "$REGISTRY" ] && HCP=$(echo "$HCP" | yq ".spec.imageRegistry = \"$REGISTRY\"")

VERSION=$(yq .bootstrapImageTag // "" "$VALUES_FILE")
[ -n "$VERSION" ] && HCP=$(echo "$HCP" | yq ".spec.version = \"$VERSION\"")

#  Source configuration

SOURCE_REMOTE=$(yq .source.remote // "" "$VALUES_FILE")
if [ -n "$SOURCE_REMOTE" ] && [ "$SOURCE_REMOTE" = "true" ]; then
  HCP=$(echo "$HCP" | yq ".spec.source.remote = true")
else
  HCP=$(echo "$HCP" | yq ".spec.source.useLocalKustomizations = true")
fi

#  Image pull secrets

PULL_SECRETS_COUNT=$(yq .imagePullSecrets | length // 0 "$VALUES_FILE" 2>/dev/null ||␣
↪echo "0")
if [ "$PULL_SECRETS_COUNT" -gt 0 ]; then
  SECRETS_JSON=$(yq -o=json .imagePullSecrets | [.[] | select(. != null and . != "")␣
↪| {"name": ., "namespace": "edbpgai-bootstrap"}] "$VALUES_FILE")
  HCP=$(echo "$HCP" | yq ".spec.imagePullSecrets = $SECRETS_JSON")
fi

#  Global parameters (parameters.global.* -> spec.globalParameters)

GLOBAL_PARAMS=$(yq .parameters.global // "" "$VALUES_FILE")
if [ -n "$GLOBAL_PARAMS" ] && [ "$GLOBAL_PARAMS" != "null" ]; then
  HCP=$(echo "$HCP" | yq ".spec.globalParameters = $(yq -o=json .parameters.global "
↪$VALUES_FILE")")
  [ -n "$REGISTRY" ] && HCP=$(echo "$HCP" | yq ".spec.globalParameters.CONTAINER_
```

(continues on next page)

(continued from previous page)

```
↪REGISTRY_URL = \"$REGISTRY\"")
  [ -n "$VERSION" ] && HCP=$(echo "$HCP" | yq ".spec.globalParameters.HCP_VERSION = \"
↪$VERSION\"")
fi

#  Component-specific parameters (parameters.<component>.* -> spec.
↪componentsParameters.<component>)

COMPONENTS=$(yq .parameters | keys | .[] | select(. != "global") "$VALUES_FILE" 2>/
↪dev/null || true)
if [ -n "$COMPONENTS" ]; then
  while IFS= read -r component; do
    HCP=$(echo "$HCP" | yq ".spec.componentsParameters.[\"$component\"] = $(yq -
↪o=json ".parameters.[\"$component\"]" "$VALUES_FILE")")
  done <<< "$COMPONENTS"
fi

#  Beacon server configuration

BEACON_SERVER=$(yq .beaconServer // "" "$VALUES_FILE")
if [ -n "$BEACON_SERVER" ] && [ "$BEACON_SERVER" != "null" ]; then
  HCP=$(echo "$HCP" | yq ".spec.beaconServer = $(yq -o=json .beaconServer "$VALUES_
↪FILE")")
fi

#  Beacon agent configuration

BEACON_AGENT=$(yq .beaconAgent // "" "$VALUES_FILE")
if [ -n "$BEACON_AGENT" ] && [ "$BEACON_AGENT" != "null" ]; then
  HCP=$(echo "$HCP" | yq ".spec.beaconAgent = $(yq -o=json .beaconAgent "$VALUES_FILE
↪")")
fi

#  Cluster groups

CLUSTER_GROUPS=$(yq .clusterGroups // "" "$VALUES_FILE")
if [ -n "$CLUSTER_GROUPS" ] && [ "$CLUSTER_GROUPS" != "null" ]; then
  HCP=$(echo "$HCP" | yq ".spec.clusterGroups = $(yq -o=json .clusterGroups "$VALUES_
↪FILE")")
fi

#  Installation scenarios

SCENARIOS=$(yq .scenarios // "" "$VALUES_FILE")
if [ -n "$SCENARIOS" ] && [ "$SCENARIOS" != "null" ]; then
  SCENARIOS_JSON=$(echo "$SCENARIOS" | tr , \n | sed s/^[[:space:]]*//;s/[[:space:]]*
↪$// | yq -o=json [.[] | select(. != "")] -)
  HCP=$(echo "$HCP" | yq ".spec.scenarios = $SCENARIOS_JSON")
fi

#  Disabled components
```

(continues on next page)

(continued from previous page)

```
DISABLED=$(yq .disabledComponents // "" "$VALUES_FILE")
if [ -n "$DISABLED" ] && [ "$DISABLED" != "null" ]; then
  HCP=$(echo "$HCP" | yq ".spec.disabledComponents = $(yq -o=json .disabledComponents
↪"$VALUES_FILE")")
fi

#  Resource annotations

ANNOTATIONS=$(yq .resourceAnnotations // "" "$VALUES_FILE")
if [ -n "$ANNOTATIONS" ] && [ "$ANNOTATIONS" != "null" ]; then
  HCP=$(echo "$HCP" | yq ".spec.resourceAnnotations = $(yq -o=json .
↪resourceAnnotations "$VALUES_FILE")")
fi

#  Output

if [ -n "$OUTPUT_FILE" ]; then
  echo "$HCP" > "$OUTPUT_FILE"
  echo "HybridControlPlane CR written to: $OUTPUT_FILE"
else
  echo "$HCP"
fi
```

Usage:

```
#  Print the converted CR to stdout:

chmod +x convert-values-to-hcp.sh
./convert-values-to-hcp.sh values.yaml

#  Write to a file:

./convert-values-to-hcp.sh values.yaml hybridmanager.yaml
```

Important

Review the generated manifest carefully before applying it:

- ``**spec.version**`` : The bootstrap `bootstrapImageTag` may use a different versioning scheme than the operator `spec.version` .
- ``**spec.imagePullSecrets**`` : The script assumes all secrets are in the `edbpgai-bootstrap` namespace. Adjust if your secrets are in a different namespace.

4.1.3.4.3 Helm ownership transfer (manual)

This approach transfers Helm ownership of existing resources from the `edbpgai-bootstrap` release to the `edb-hcp-operator` release. This allows the operator to adopt and manage the existing infrastructure without recreating anything.

> **Note**
>
> This method preserves **all resources** (CRDs, RBAC, webhooks, database clusters) and simply changes the Helm ownership annotations. Your database clusters will remain running throughout the migration, and the operator will seamlessly take over management. !!!

4.1.3.4.4 How it works

The migration works by changing the `meta.helm.sh/release-name` annotation on all resources from `edbp-gai-bootstrap` to `edb-hcp-operator` . This tells Helm that these resources now belong to the operator release instead of the bootstrap release.

Key benefits of this approach:

1. **No resource deletion** - CRDs, hybridControlPlane, preflight and postflight resources remain intact
2. **No Helm conflicts** - The operator can adopt existing resources without ownership errors
3. **Minimal downtime** - Only the HM operator pods are affected during the transition
4. **Simple rollback** - Can transfer annotations back if needed
5. **Clean migration** - After verification, the bootstrap namespace can be safely removed

What gets transferred:

- Custom Resource Definitions (CRDs)
- ClusterRoles and ClusterRoleBindings
- Roles and RoleBindings in the bootstrap namespace
- ServiceAccounts
- Services
- Webhook configurations

4.1.3.4.5 Prerequisites

- Access to kubectl and helm CLI tools
- `jq` command-line JSON processor installed

The following script performs all migration steps in sequence. You can run it as a single script or execute each step manually.

```
# !/bin/bash
set -e

echo "Starting migration from bootstrap to operator installation..."

echo "Step 1: Deleting bootstrap deployments..."
kubectl delete deployments -n edbpgai-bootstrap --all

echo "Step 2: Transferring Helm ownership of ServiceAccount..."
kubectl annotate sa edb-hcp-operator-controller-manager \
  meta.helm.sh/release-name=edb-hcp-operator \
  --overwrite \
  -n edbpgai-bootstrap
```

(continues on next page)

(continued from previous page)

```
echo "Step 3: Transferring Helm ownership of CRDs..."
for line in $(kubectl get crd | grep edbpgai.edb.com | awk {print $1}); do
  kubectl annotate crd $line \
    meta.helm.sh/release-name=edb-hcp-operator \
    --overwrite
done

echo "Step 4: Transferring Helm ownership of ClusterRoles..."
for line in $(kubectl get clusterroles -o json | \
  jq -r .items[] | select(.metadata.annotations["meta.helm.sh/release-name"] ==
↪"edbpgai-bootstrap") | .metadata.name); do
  kubectl annotate clusterrole $line \
    meta.helm.sh/release-name=edb-hcp-operator \
    --overwrite
done

echo "Step 5: Transferring Helm ownership of ClusterRoleBindings..."
for line in $(kubectl get clusterrolebindings -o json | \
  jq -r .items[] | select(.metadata.annotations["meta.helm.sh/release-name"] ==
↪"edbpgai-bootstrap") | .metadata.name); do
  kubectl annotate clusterrolebinding $line \
    meta.helm.sh/release-name=edb-hcp-operator \
    --overwrite
done

echo "Step 6: Transferring Helm ownership of Roles..."
for line in $(kubectl get roles -n edbpgai-bootstrap -o json | \
  jq -r .items[] | select(.metadata.annotations["meta.helm.sh/release-name"] ==
↪"edbpgai-bootstrap") | .metadata.name); do
  kubectl annotate role $line \
    meta.helm.sh/release-name=edb-hcp-operator \
    --overwrite \
    -n edbpgai-bootstrap
done

echo "Step 7: Transferring Helm ownership of RoleBindings..."
for line in $(kubectl get rolebindings -n edbpgai-bootstrap -o json | \
  jq -r .items[] | select(.metadata.annotations["meta.helm.sh/release-name"] ==
↪"edbpgai-bootstrap") | .metadata.name); do
  kubectl annotate rolebinding $line \
    meta.helm.sh/release-name=edb-hcp-operator \
    --overwrite \
    -n edbpgai-bootstrap
done

echo "Step 8: Transferring Helm ownership of Services..."
for line in $(kubectl get svc -n edbpgai-bootstrap -o json | \
  jq -r .items[] | select(.metadata.annotations["meta.helm.sh/release-name"] ==
↪"edbpgai-bootstrap") | .metadata.name); do
  kubectl annotate svc $line \
    meta.helm.sh/release-name=edb-hcp-operator \
```

(continues on next page)

(continued from previous page)

```
    --overwrite \
    -n edbpgai-bootstrap
done

echo "Step 9: Transferring Helm ownership of Webhook Configurations..."
kubectl annotate mutatingwebhookconfiguration \
  edb-hcp-operator-mutating-webhook-configuration \
  meta.helm.sh/release-name=edb-hcp-operator \
  --overwrite

kubectl annotate validatingwebhookconfiguration \
  edb-hcp-operator-validating-webhook-configuration \
  meta.helm.sh/release-name=edb-hcp-operator \
  --overwrite

echo "migration completed successfully! Now the operator can be installed without␣
↪conflicts."
```

4.1.3.4.6 Step 1: Delete bootstrap deployments

Remove all deployments in the bootstrap namespace. This stops the bootstrap controllers:

```
#  Delete all deployments in the bootstrap namespace

kubectl delete deployments -n edbpgai-bootstrap --all
__OUTPUT__
deployment.apps "edb-hcp-operator-controller-manager" deleted
deployment.apps "file-server" deleted
```

```
#  Verify deployments are removed

kubectl get deployments -n edbpgai-bootstrap
__OUTPUT__
No resources found in edbpgai-bootstrap namespace.
```

4.1.3.4.7 Step 2: Transfer Helm ownership - ServiceAccount

Transfer the service account ownership annotation:

```
#  Re-annotate the service account to change Helm ownership

kubectl annotate sa edb-hcp-operator-controller-manager \
  meta.helm.sh/release-name=edb-hcp-operator \
  --overwrite \
  -n edbpgai-bootstrap
__OUTPUT__
serviceaccount/edb-hcp-operator-controller-manager annotated
```

4.1.3.4.8 Step 3: Transfer Helm ownership - CRDs

Transfer ownership of all CRDs under `edbpgai.edb.com`:

```
#  Update Helm release annotation for all edbpgai.edb.com CRDs

for line in $(kubectl get crd | grep edbpgai.edb.com | awk {print $1}); do
  kubectl annotate crd $line \
    meta.helm.sh/release-name=edb-hcp-operator \
    --overwrite
done
__OUTPUT__
customresourcedefinition.apiextensions.k8s.io/hybridcontrolplanes.edbpgai.edb.com␣
↪annotated
customresourcedefinition.apiextensions.k8s.io/postflights.edbpgai.edb.com annotated
customresourcedefinition.apiextensions.k8s.io/preflights.edbpgai.edb.com annotated
```

```
#  Verify the annotations were updated

kubectl get crd | grep edbpgai.edb.com | head -5 | while read crd rest; do
  echo "CRD: $crd"
  kubectl get crd $crd -o jsonpath={.metadata.annotations.meta\.helm\.sh/release-name}
  echo ""
done
__OUTPUT__
CRD: hybridcontrolplanes.edbpgai.edb.com
edb-hcp-operator
CRD: postflights.edbpgai.edb.com
edb-hcp-operator
CRD: preflights.edbpgai.edb.com
edb-hcp-operator
```

4.1.3.4.9 Step 4: Transfer Helm ownership - ClusterRoles

Transfer ownership of ClusterRoles:

```
#  Update Helm release annotation for ClusterRoles

for line in $(kubectl get clusterroles -o json | \
  jq -r .items[] | select(.metadata.annotations["meta.helm.sh/release-name"] ==
↪"edbpgai-bootstrap") | .metadata.name); do
  kubectl annotate clusterrole $line \
    meta.helm.sh/release-name=edb-hcp-operator \
    --overwrite
done
__OUTPUT__
clusterrole.rbac.authorization.k8s.io/create-hcp annotated
clusterrole.rbac.authorization.k8s.io/edb-hcp-operator-hybridcontrolplane-editor-role␣
↪annotated
clusterrole.rbac.authorization.k8s.io/edb-hcp-operator-hybridcontrolplane-viewer-role␣
↪annotated
clusterrole.rbac.authorization.k8s.io/edb-hcp-operator-manager-role annotated
clusterrole.rbac.authorization.k8s.io/edb-hcp-operator-metrics-auth-role annotated
```

(continues on next page)

(continued from previous page)

```
clusterrole.rbac.authorization.k8s.io/edb-hcp-operator-metrics-reader annotated
clusterrole.rbac.authorization.k8s.io/edb-hcp-operator-postflight-admin-role annotated
clusterrole.rbac.authorization.k8s.io/edb-hcp-operator-postflight-editor-role␣
↪annotated
clusterrole.rbac.authorization.k8s.io/edb-hcp-operator-postflight-viewer-role␣
↪annotated
clusterrole.rbac.authorization.k8s.io/edb-hcp-operator-preflight-admin-role annotated
clusterrole.rbac.authorization.k8s.io/edb-hcp-operator-preflight-editor-role annotated
clusterrole.rbac.authorization.k8s.io/edb-hcp-operator-preflight-viewer-role annotated
```

4.1.3.4.10 Step 5: Transfer Helm ownership - ClusterRoleBindings

Transfer ownership of ClusterRoleBindings:

```
#  Update Helm release annotation for ClusterRoleBindings

for line in $(kubectl get clusterrolebindings -o json | \
  jq -r .items[] | select(.metadata.annotations["meta.helm.sh/release-name"] ==
↪"edbpgai-bootstrap") | .metadata.name); do
  kubectl annotate clusterrolebinding $line \
    meta.helm.sh/release-name=edb-hcp-operator \
    --overwrite
done
__OUTPUT__
clusterrolebinding.rbac.authorization.k8s.io/create-hcp annotated
clusterrolebinding.rbac.authorization.k8s.io/edb-hcp-operator-manager-rolebinding␣
↪annotated
clusterrolebinding.rbac.authorization.k8s.io/edb-hcp-operator-metrics-auth-
↪rolebinding annotated
```

4.1.3.4.11 Step 6: Transfer Helm ownership - Roles

Transfer ownership of Roles in the bootstrap namespace:

```
#  Update Helm release annotation for Roles

for line in $(kubectl get roles -n edbpgai-bootstrap -o json | \
  jq -r .items[] | select(.metadata.annotations["meta.helm.sh/release-name"] ==
↪"edbpgai-bootstrap") | .metadata.name); do
  kubectl annotate role $line \
    meta.helm.sh/release-name=edb-hcp-operator \
    --overwrite \
    -n edbpgai-bootstrap
done
__OUTPUT__
role.rbac.authorization.k8s.io/edb-hcp-operator-leader-election-role annotated
```

4.1.3.4.12 Step 7: Transfer Helm ownership - RoleBindings

Transfer ownership of RoleBindings in the bootstrap namespace:

```
#  Update Helm release annotation for RoleBindings

for line in $(kubectl get rolebindings -n edbpgai-bootstrap -o json | \
  jq -r .items[] | select(.metadata.annotations["meta.helm.sh/release-name"] ==
↪"edbpgai-bootstrap") | .metadata.name); do
  kubectl annotate rolebinding $line \
    meta.helm.sh/release-name=edb-hcp-operator \
    --overwrite \
    -n edbpgai-bootstrap
done
__OUTPUT__
rolebinding.rbac.authorization.k8s.io/edb-hcp-operator-leader-election-rolebinding␣
↪annotated
```

4.1.3.4.13 Step 8: Transfer Helm ownership - Services

Transfer ownership of Services in the bootstrap namespace:

```
#  Update Helm release annotation for Services

for line in $(kubectl get svc -n edbpgai-bootstrap -o json | \
  jq -r .items[] | select(.metadata.annotations["meta.helm.sh/release-name"] ==
↪"edbpgai-bootstrap") | .metadata.name); do
  kubectl annotate svc $line \
    meta.helm.sh/release-name=edb-hcp-operator \
    --overwrite \
    -n edbpgai-bootstrap
done
__OUTPUT__
service/edb-hcp-operator-controller-manager-metrics-service annotated
service/edb-hcp-operator-webhook-service annotated
```

4.1.3.4.14 Step 9: Transfer Helm ownership - Webhook Configurations

Transfer ownership of webhook configurations:

```
#  Update MutatingWebhookConfiguration

kubectl annotate mutatingwebhookconfiguration \
  edb-hcp-operator-mutating-webhook-configuration \
  meta.helm.sh/release-name=edb-hcp-operator \
  --overwrite
__OUTPUT__
mutatingwebhookconfiguration.admissionregistration.k8s.io/edb-hcp-operator-mutating-
↪webhook-configuration annotated
```

```
#  Update ValidatingWebhookConfiguration

kubectl annotate validatingwebhookconfiguration \
```

(continues on next page)

(continued from previous page)

```
  edb-hcp-operator-validating-webhook-configuration \
  meta.helm.sh/release-name=edb-hcp-operator \
  --overwrite
__OUTPUT__
validatingwebhookconfiguration.admissionregistration.k8s.io/edb-hcp-operator-
↪validating-webhook-configuration annotated
```

4.1.3.4.15 Step 10: Verify ownership transfer

Verify that resources now have the correct Helm ownership:

```
#  Check a sample of resources

echo "Checking CRD ownership:"
kubectl get crd | grep edbpgai.edb.com | head -1 | awk {print $1} | xargs -I {} \
  kubectl get crd {} -o jsonpath={.metadata.annotations.meta\.helm\.sh/release-name}
echo ""

echo "Checking ClusterRole ownership:"
kubectl get clusterroles -o json | \
  jq -r .items[] | select(.metadata.annotations["meta.helm.sh/release-name"] == "edb-
↪hcp-operator") | .metadata.name | head -3
```

4.1.3.5 Support

If you encounter issues during migration:

- Check the *troubleshooting guide*
- Contact EDB Support with migration logs and error messages
- Consult EDB Professional Services for complex migration scenarios

4.1.3.6 Next steps

After successful migration:

- Review the *operator installation guide* for ongoing management
- Learn about *updating HCP configuration*
- See *Upgrading HM version* using the operator

4.1.4 Troubleshooting Hybrid Manager installations

The Hybrid Manager (HM) installation comes with preflight and postflight checks that you can use to validate your installation, as well as monitor running deployments. These tools also provide detailed information to help you troubleshoot any failed checks.

4.1.4.1 Understanding Preflight checks

Check the results of the `Preflight` check to ensure all installation-related secrets are valid.

Role focus: Site reliability engineer (SRE) / Infrastructure Engineer

4.1.4.1.1 Prerequisites

- *Phase 5: Installing Hybrid Manager* — You have run the installation command and the process seems stuck.

4.1.4.1.2 Outcomes

- *Ability to diagnose and resolve secret-related blockers during Hybrid Manager (HM) installation.*

When you start the HM installation, the operator automatically creates a `Preflight` custom resource. This resource validates that all required Kubernetes secrets exist and are correctly structured **before** the operator proceeds to deploy the platform services.

!!!Note

You do not create the Preflight resource yourself. The operator creates it automatically after the `HybridControlPlane` resource is created.

After you apply your `HybridControlPlane` manifest, the operator:

1. Reconciles the `HybridControlPlane` resource.
2. Creates a `Preflight` resource with the secret requirements for the deployment.
3. Validates each secret requirement: verifies the secret exists, then evaluates validation rules against the secret content.
4. Reports the result in the preflight resource status.

If all requirements pass, the operator proceeds with deploying the platform services. If any requirement fails, **deployment is blocked** and the operator retries every 30 seconds until the issues are resolved.

4.1.4.1.3 Checking the preflight status

If the installation appears stuck after running the *installation Helm command* , inspect the preflight status:

```
kubectl get preflight -o yaml
```

The `status.secretRequirementsStatus` section shows the result for each checked secret:

```
status:
  observedGeneration: 1
  secretRequirementsStatus:
    - secret:
        name: hm-portal-bootstrap
        namespace: default
      state: Valid
    - secret:
        name: edb-object-storage
        namespace: default
      state: Invalid
      messages:
        - "failed to pass the rule: aws-configurations"
    - secret:
        name: langflow-secret
        namespace: default
      state: SecretNotFound
```

4.1.4.1.4 Understanding the states

State	Meaning	Action required
Valid	Secret exists and passes all validation rules.	None.
Invalid	Secret exists but one or more validation rules failed.	Review the *messages* field to identify which rules failed, then fix the secret content.
SecretNotFound	The referenced secret does not exist.	Create the secret in the correct namespace with the required keys.

!!!Warning

Deployment remains blocked until **every** secret requirement reaches the `Valid` state. The operator retries automatically every 30 seconds. Once all requirements pass, deployment proceeds without manual intervention.

Each secret requirement is associated with an **installation scenario**: `core` , `migration` , `ai` , or `analytics` . The operator uses the scenarios configured in `spec.scenarios` on the `HybridControlPlane` to filter which requirements are checked.

- If your configuration defines specific scenarios (for example, `core, ai`), only requirements matching those scenarios are validated. Requirements for other scenarios (such as `migration`) are skipped.
- If your configuration does **not** define `spec.scenarios` , all secret requirements are validated regardless of scenario.

This means you only need to provide the secrets for the scenarios you are actually deploying. For example, if you are not using the `analytics` scenario, you do not need to provide the Lakekeeper confounding key secret.

4.1.4.1.5 Troubleshooting preflight failures

Installation appears stuck

If the Helm command completes but pods are not starting, the preflight checks may be blocking the deployment. Check the preflight status first:

```
kubectl get preflight -o yaml
```

Look at `status.secretRequirementsStatus` for any entries that are not `Valid` .

Secret not found

If a requirement shows `SecretNotFound` , verify the secret exists:

```
kubectl get secret <secret-name> -n <namespace>
```

If the secret does not exist, create it following the documentation for *Preparing your environment* . Ensure the name and namespace match exactly what the preflight expects.

Validation rule failures

If a requirement shows `Invalid` , the `messages` field identifies which rules failed. For example:

```
failed to pass the rule: aws-configurations
```

This means the secret exists but does not meet the expected structure. Inspect the secret:

```
kubectl get secret <secret-name> -n <namespace> -o yaml
```

Compare the secret's data keys and values against the rule message. Common causes include:

- **Missing keys** — The secret is missing a required data field (for example, `aws_region` or `bucket_name` for AWS object storage).
- **Missing annotations** — Some secrets require specific replicator annotations to be set on the secret metadata.
- **Wrong secret type** — For example, a TLS secret requires type `kubernetes.io/tls` .
- **Empty values** — A required field exists but contains no data.

After fixing the secret, the operator automatically re-evaluates within 30 seconds.

Requirements not being checked

If you expect a requirement to be validated but it does not appear in the status, check the scenario configuration:

1. Verify which scenarios are defined for your HM installation:

```
kubectl get hybridcontrolplane <name> -o jsonpath={.spec.scenarios}
```

2. Confirm the established scenario matches the scenarios established by the CTO / Architect / Lead Engineer during the *Planning architecture* phase. If the output of the command doesn't match the expected scenarios, it means that the installation was not carried out as planned, and one or several of the scenarios were skipped during the installation.

4.1.4.2 Monitoring with Postflight checks

Check the results of the `Postflight` check to monitor your HM deployment is ready.

Role focus: Infrastructure Engineer / Platform Administrator

4.1.4.2.1 Prerequisites

- *Phase 5: Installing Hybrid Manager* — You have completed and verified the installation of HM. The deployment is running and you want to either check its state or troubleshoot failed checks.

4.1.4.2.2 Outcomes

- *Confirmation that the deployed Hybrid Manager (HM) platform is healthy and ready for day-2 configuration.*

The **Postflight** custom resource provides automated health monitoring for your deployed HM stack. It periodically runs a set of built-in checks and reports an overall health status.

> **Note**
>
> Unlike *Understanding Preflight checks* , postflight provides monitoring capabilities. It does **not** block any operations or trigger automated remediation. It provides visibility for troubleshooting. !!!

4.1.4.2.3 What postflight checks do

The operator uses a `Postflight` resource to run 5 built-in health checks at a configurable interval. Checks only begin after the `HybridControlPlane` reaches the `deployed` phase. Each check verifies a different aspect of the running platform:

Check	What it verifies
Pods	All HM pods are running and ready.
Databases	All internal EDB Postgres for Kubernetes clusters are ready and their latest backups completed.
Velero backups	The latest Velero backup completed successfully.
Nodes	All Kubernetes cluster nodes are ready.
Certificates	All cert-manager certificates are valid and ready.

If **all** checks pass, the overall phase is `Healthy` . If **any** check fails, the phase is `Unhealthy` .

4.1.4.2.4 Checking the postflight status

Quick overview

Check whether your installation is healthy:

```
kubectl get postflight
```

The output should look like this:

```
NAME                   PHASE     LAST CHECK
edbpgai-postflight     Healthy   2025-01-15T10:30:00Z
```

Detailed results

If the phase is `Unhealthy` , obtain more detailed results:

```
kubectl get postflight <name> -o yaml
```

The `status.checkResults` section shows individual results:

```
status:
  phase: Unhealthy
  lastTransitionTime: "2025-01-15T10:30:00Z"
  checkResults:
    - name: PodsChecker
      description: "Check whether all pods belonging to the Hybrid Manager are ready."
      state: Passed
    - name: DBsChecker
      description: "Check whether all internal databases belonging to the Hybrid␣
↪Manager are ready."
      state: Passed
    - name: VeleroChecker
      description: "Check whether the latest Velero Backup has been completed."
      state: Failed
      messages:
        - "the latest velero backup is in a unhealthy phase: PartiallyFailed"
    - name: NodesChecker
      description: "Check whether all nodes are ready."
      state: Passed
    - name: CertificatesChecker
      description: "Check whether all certificates belonging to the Hybrid Manager␣
```

(continues on next page)

(continued from previous page)

```
↪are valid."
      state: Passed
```

4.1.4.2.5 Troubleshooting unhealthy checks

When the postflight phase is `Unhealthy` , inspect the `status.checkResults` section to identify which checks are failing and their associated messages.

Pods check is failing

One or more HM pods are not in a ready state. Investigate the failing pods:

```
kubectl get pods -A | grep -v Running
kubectl describe pod <pod-name> -n <namespace>
kubectl logs <pod-name> -n <namespace>
```

Common causes include image pull failures, resource limits, and crashlooping containers.

Databases check is failing

An EDB Postgres for Kubernetes database cluster is not ready, or its latest backup failed. Check the cluster and backup status:

```
kubectl get clusters.postgresql.k8s.enterprisedb.io -A
kubectl get backups.postgresql.k8s.enterprisedb.io -A --sort-by=.metadata.
↪creationTimestamp
```

Common causes include storage issues, connection failures, and WAL archiving problems.

Velero backup check is failing

The latest Velero backup is in a failed or partially failed state:

```
kubectl get backups.velero.io -n velero --sort-by=.metadata.creationTimestamp
kubectl describe backup <backup-name> -n velero
```

Nodes check is failing

One or more cluster nodes are not ready. This affects the entire cluster, not just HM:

```
kubectl get nodes
kubectl describe node <node-name>
```

Common causes include disk pressure, memory pressure, network unreachable conditions, and kubelet issues.

Certificates check is failing

A cert-manager certificate has not been issued or has expired:

```
kubectl get certificates -A
kubectl describe certificate <cert-name> -n <namespace>
kubectl get certificaterequests -A
```

Common causes include issuer configuration problems, rate limiting (for ACME/Let's Encrypt), and DNS validation failures.

4.1.4.2.6 Configuring the check interval

You can also change the interval in which the postflight check runs.

The `spec.interval` field controls how often checks run. The operator respects this interval when the phase is `Healthy` , requeuing for the remaining duration after each successful run.

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: Postflight
metadata:
  name: edbpgai-postflight
spec:
  hm:
    name: edbpgai
  interval: 30m
```

Note

When the phase transitions to *Unhealthy* , the status is updated immediately. The interval only affects the cadence of re-checks when the system is healthy. !!!

CHAPTER 5

Using Hybrid Manager

This section helps you understand how to operate EDB Postgres AI Hybrid Manager (HM) across different use cases, from routine cluster management to high availability and disaster recovery.

The topics are structured by audience—starting with general platform users (application teams, database engineers, platform owners), and continuing to Kubernetes administrators responsible for cluster security, observability, and integrations.

5.1 Where to start

Begin with the foundational sections:

- *Terminology* introduces key HM concepts.
- *Hybrid Manager overview* provides essential context for Postgres, Lakehouse, and AI workloads.

From there, explore topics based on your role and responsibilities.

5.2 For users

These sections cover day-to-day tasks in Hybrid Manager:

- *Migration Portal Projects Owner* — create, manage, and configure project-level settings including alerting and usage reporting.
- Cluster Management — create, monitor, and manage Postgres clusters, including backups, query diagnostics, and templates.
- Backup and Restore — manage backup schedules, restore points, and disaster recovery configurations.
- Creating Migration Portal secrets using `edbctl <Creating Migration Portal secrets using *edbctl*>` — use the CLI to automate deployment, configuration, and data operations.

- *Step 5: Configure API Access* — automate platform functions with the HM API.
- Postgres database upgrades - upgrade your HM-managed Postgres databases to the next minor or major version.
- *Simple Migration Agent* — run schema and data migrations from Oracle, RDS, or other Postgres installations.
- Image Management — manage container images, repositories, and AI knowledge base assets.
- Enable Migration Copilot — unlock guided migration workflows with Copilot.
- *What the Chat Agent can do* — operate Hybrid Manager in plain language: browse your estate, provision clusters, troubleshoot performance, and consult the docs.
- *Governance in Hybrid Manager* — how EDB Agent Governance integrates with HM to audit AI agents operating on your Postgres clusters.

5.3 For Kubernetes administrators

For those administering the infrastructure beneath Hybrid Manager:

- *Managing users in Hybrid Manager* — configure native users.
- IdP management — configure identity providers and rotate IdP connector encryption keys.
- *Monitoring execution* — set up agents, explore built-in dashboards, and extend observability with Prometheus and Grafana.
- *Configure the multi-DC topology* - set up multiple data centers, a primary and secondary locations.
- Hybrid Manager upgrades - upgrade the HM platform to the next minor/major/innovation release version.
- HA/DR — configure high availability and disaster recovery for production clusters.
- *Troubleshooting Langflow in Hybrid Manager* — investigate and resolve issues across all platform areas.

5.3.1 Hybrid Manager Terminology

Review these terms to understand and leverage Hybrid Manager (HM) capabilities.

5.3.1.1 Hybrid Manager

Hybrid Manager is the name given to EDB's solution for deployment, management, monitoring and automation of Postgres and Lakehouse databases and sovereign AI systems.

It is a Kubernetes-based software stack that simplifies the lifecycle of deployed database clusters by making best use of modern private cloud, public cloud, multi-cloud or on-premise infrastructures, allowing developers to self-manage databases for their applications through a Private or Sovereign Database-as-a-Service (DBaaS) model.

5.3.1.1.1 Private Database-as-a-Service (DBaaS)

A key feature of the Hybrid Manager is the ability to provision and manage databases on Kubernetes. This capability is referred to as a Private Database-as-a-Service (DBaaS).

Through this capability, the Hybrid Manager becomes a self-managed, internal database service within your infrastructure, which gives you the agility of a public cloud DBaaS with the control and security of a private environment.

This approach enables developers to perform self-service provisioning, observability and troubleshooting.

5.3.1.1.2 Engineered System

The Engineered System is Hybrid Manager in a box, a ready-to-go physical server that you can slot into a rack and deploy as a containerized appliance. This deployment method offers a flexible and portable solution for organizations seeking to streamline their database operations in a hybrid or multi-cloud world.

5.3.1.2 PG AI Console

The PG AI Console is a web-based graphical interface that provides users with a centralized point of interaction for the Hybrid Manager. From this dashboard, you will be able to deploy and manage databases, connect and monitor external databases, perform data migrations, manage user permissions, deploy AI workloads and more.

For an overview of the PG AI Console's functionality, see *About the PG AI Console* .

5.3.1.2.1 Projects

In the Hybrid Manager, **Projects** serve as fundamental organizational units that provide isolation for managing and creating resources. You can create and manage your clusters within these individual projects. Use projects to manage resource per team (engineering, operations, data science), per environment (development, staging, production), per product, per customer, etc.

See *Migration Portal Projects Owner* for more information.

5.3.1.2.2 Estate

In the Hybrid Manager, **Estate** serves as a fundamental overview of all database resources it has access to. Whether you created a database or Analytics cluster with the Hybrid Manager, or connected an external database for monitoring, it becomes part of your Estate.

See HTTP 431 "Request Header Fields Too Large" error when accessing the Estate page for more information.

5.3.1.2.3 Launchpad

The Launchpad gives access to a number of third-party services which can be used to enhance the capabilities of the Hybrid Manager.

See *Launchpad* for more information.

5.3.1.3 Clusters (disambiguation)

The term **cluster** in computing refers to a group of interconnected components that work together as a unified system to achieve a common goal. As many technologies adopt this approach, the term **cluster** becomes widely spread to refer to systems where multiple individual components are managed in a unified way.

5.3.1.3.1 Kubernetes cluster

Within the scope of the Hybrid Manager, a Kubernetes cluster represents the foundational infrastructure upon which the Hybrid Manager software itself is deployed as a self-contained instance. Hybrid Manager leverages the capabilities of this underlying Kubernetes cluster to manage Postgres database deployments, with CloudNativePG as its tool for orchestrating Postgres on Kubernetes.

Not to be confused with **Postgres* cluster* , or **Database* cluster* .

5.3.1.3.2 Kubernetes stretch cluster

This is a subtype of a Kubernetes cluster that you can also use to deploy the Hybrid Manager.

!!!note

This functionality will be added in a later version.

Typically, all nodes of a Kubernetes cluster reside in a single Availability Zone. For Hybrid Manager deployments with high availability (HA), advanced high availability (AHA) or distributed high availability (DHA) enabled, the platform automatically handles primary node failures by initiating a seamless takeover by a standby node.

However, this built-in failover mechanism is insufficient when an entire Availability Zone experiences an outage. This is where **Kubernetes stretch clusters** become crucial. They extend or "strecth" a Kubernetes cluster across multiple Availability Zones or geographically separated locations, ensuring that if one zone fails, application workloads can continue to be served by healthy nodes in other locations. This setup requires the use of AZ-independent, shared storage to maintain data accessibility during an AZ failure.

5.3.1.3.3 Postgres cluster

In Postgres, a cluster is a single *instance of Postgres* , also known as a Postgres server. This single occurrence of a Postgres instance can contain one or multiple different databases. For example, a server for MYCOMPANY with databases *HR*, *FINANCE* and *LEGAL*.

This term is not used in Hybrid Manager documentation, as the usage of "cluster" in this context could be ambiguous. Here, we'll refer to it as Postgres instance.

Not to be confused with **Kubernetes* cluster* , or **Database* cluster* .

5.3.1.3.4 Database cluster

In Hybrid Manager documentation, **database cluster** refers to multiple instances of Postgres using replication capabilities to support database resiliency and improve performance. Since such cluster-based systems are essential features of modern enterprise Postgres operations, the Hybrid Manager is built around database clusters. For this reason, even single-node Postgres instances are represented as clusters.

For example, if a postgres server for MYCOMPANY contains the databases *HR*, *FINANCE* and *LEGAL*, and then the same server gets replicated for redundancy and failover, we refer to the whole system as a database cluster. In a scenario with two instances, you would have a database server A with *HR*, *FINANCE*, and *LEGAL* and a replica server B with *HR*, *FINANCE*, and *LEGAL*, where server A and B have the same data. Each server in this replication setup lives on its own data node.

Database clusters can either be created within the Hybrid Manager or created externally but then connected to the Hybrid Manager for observability. See *Database cluster types in the Hybrid Manager* for an overview of all cluster types you can create or connect to the Hybrid Manager.

Not to be confused with **Kubernetes* cluster* or **Postgres* cluster* .

5.3.1.4 Database cluster types in Hybrid Manager

The Hybrid Manager instance is a centralized pane of glass that can host, manage and monitor several database cluster categories. Some database clusters are created and completely maintained within the Hybrid Manager, while others are created externally but connected to Hybrid Manager to provide monitoring and migration capabilities.

Currently, you can:

- Create *HM-managed clusters and external CNPG clusters*

or,

- Connect *Requirements for external clusters* .

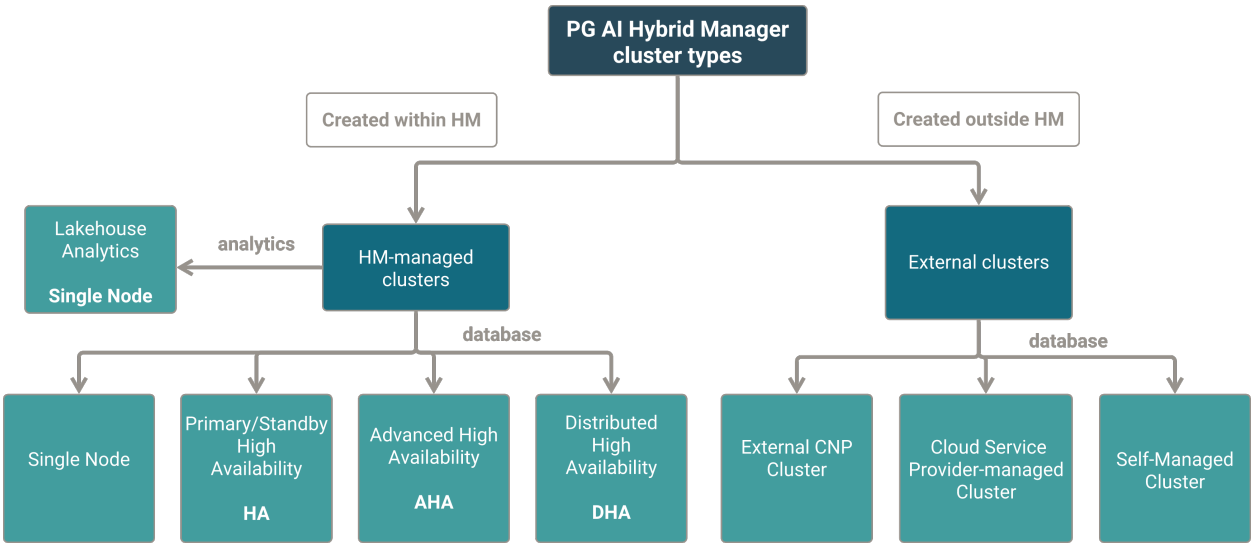


Fig. 1: Cluster Types in Hybrid Manager

5.3.1.4.1 HM-managed clusters

HM-managed clusters are database servers you created with Hybrid Manager. They are deployed in the same Kubernetes cluster as the Hybrid Manager, using its provisioning capabilities. In the **Estate** page, HM-managed clusters have the **Hybrid Manager** label under the **Management** column.

Examples: Single Node, Primary/Standby High Availability (HA), Advanced High Availability (AHA), or Distributed High Availability (PGD) database instances with PostgreSQL, EDB Postgres Advanced Server, or EDB Postgres Extended Server that were deployed using Hybrid Manager.

EDB offers the following database topology options.

Single Node: This option creates a single Postgres instance. See *Single-node cluster* for more information.

Primary/Standby High Availability: Also called HA, this topology option hosts a Postgres instance and replicates it on at least one additional node. See *Primary/standby high availability cluster* for more information.

Advanced High Availability: Also called AHA, this topology hosts at least two groups of a Postgres instance and its replicas across availability zones in a single region. See *Advanced high availability cluster* for more information.

Distributed High Availability: Also called DHA, this topology hosts at least two groups of a Postgres instance and its replicas across availability zones in multiple regions. See *Distributed high availability cluster* for more information.

5.3.1.4.2 External clusters

You can connect clusters that were provisioned externally.

5.3.1.4.3 Self-managed cluster

Self-managed clusters are external Postgres, or Oracle clusters (created outside of Hybrid Manager). They are deployed using other provisioning capabilities than the Hybrid Manager (e.g. manually), and are often deployed on a different machine than Hybrid Manager.

In the **Estate** page, Self-managed clusters have the **Self-Managed** label under the **Management** column.

You can connect self-managed clusters to Hybrid Manager for observability with the *Simple Migration Agent* and perform migrations with the *Data Migration Service (read-only)* .

Examples: PostgreSQL, EDB Postgres Advanced Server, EDB Postgres Extended Server, or Oracle database instances deployed by yourself on a physical server, virtual machine or similar. In the case of virtual machines, they can run on cloud infrastructures, but the provisioning of the database cluster is managed by you, and not by the Cloud Service Provider.

!!!note

Self-managed Oracle databases that you connect to Hybrid Manager don't appear under **Estate** > **Postgres**. Switch to **Estate** > **Migrations** to see information.

Self-managed Postgres clusters that you connect to the Hybrid Manager appear in both **Estate** > **Postgres** and **Estate** > **Migrations**.

5.3.1.4.4 Cloud Service Provider-managed cluster

Cloud Service Provider-managed clusters or CSP-managed clusters are external clusters (created outside of Hybrid Manager) that host one or more occurrences of a Postgres instance. They are deployed using Cloud Service Provider (CSP) provisioning capabilities and their major operational tasks are handled by the CSP (backups, upgrades, failover, etc). However, architecture, developer, and application-specific activities are handled by you.

In the **Estate** page, Cloud Hosted Database clusters have the **Cloud Hosted Databases** label under the **Management** column.

You can connect CSP-managed clusters to Hybrid Manager for observability and perform migrations with the *Data Migration Service (read-only)* .

Examples: Postgres on AWS RDS.

5.3.1.4.5 External CNPG cluster

External CNPG database clusters are external database clusters created on a Kubernetes environment outside of Hybrid Manager with CloudNativePG. They are deployed using the provisioning capabilities of open-source CloudNativePG, EDB CloudNativePG Cluster or EDB CloudNativePG Global Cluster.

In the **Estate** page, External CNPG clusters have the **External CNP Cluster** label under the **Management** column.

Examples: PostgreSQL, EDB Postgres Advanced Server, or EDB Postgres Extended Server database instances deployed with CloudNativePG, EDB CloudNativePG Cluster or EDB CloudNativePG Global Cluster.

5.3.1.5 Other terms

5.3.1.5.1 Data node

The term "data node" is used to refer to a Kubernetes worker node specifically designated to host Postgres database instances managed by Hybrid Manager.

5.3.1.5.2 Postgres instance

A Postgres server that can host multiple databases. It starts on a specific data directory containing all data and files required to run that instance. In high-availability deployments, at least one additional instance is replicated on a different Availability Zone.

5.3.2 Hybrid Manager console

5.3.2.1 What is the Hybrid Manager console?

The Hybrid Manager (HM) console is a user interface for provisioning and managing Postgres clusters. It's a centralized location for viewing, monitoring, and managing the lifecycle of databases.

In this console, you organize your resources in *projects*. A project associates organizationally related resources together, which enables simpler administration and access.

5.3.2.2 What can you do in the HM console?

Using the console, you can:

- Create, view, and manage Postgres clusters
- Create, view, and manage projects
- Create Migration Portal projects and perform migrations
- Monitor database migrations
- View and manage catalogs
- View the status of model clusters
- Manage your asset library
- Monitor HM-managed and external clusters
- Deploy and manage open-source apps, such as Apache Airflow, Apache Superset, Metabase, pgAdmin 4, and pgBadger, directly within HM.

5.3.2.3 HM console organization

The console is organized around three main pages that you access by selecting the page names that appear on every page in HM:

- ***Projects*** , where you create and manage projects, create and manage clusters, and perform tasks to manage your cluster and each project.
- ***Estate*** , where you monitor your resources. You can also use it as a starting point for creating clusters.
- ***Asset Library*** , where you can view container images and knowledge bases and manage repositories.

5.3.2.4 Launching third-party services

You can use the HM *Launchpad* to launch related third-party services without leaving HM.

5.3.2.4.1 Exploring the Hybrid Manager console

Accessing the console

Once you have your account, to access Hybrid Manager (HM) console, use the URL provided by your administrator, which is in the form `https://<your-domain>/` . Log in to the portal using the credentials provided by your administrator.

By default, the console opens on the overview page for your account and the *single pane of glass* summary of your EDB Postgres AI HM services. To return to this view from anywhere in the portal, select the EDB logo.

In this view, you can see summaries of your *Migration Portal Projects Owner* and HTTP 431 "Request Header Fields Too Large" error when accessing the Estate page .

Select the **Projects**, **Estate**, and **Asset Library** tabs to navigate to pages where you can monitor and create various resources.

Organizing resources with projects

To create and manage resources, you need a project to save them to. A project associates organizationally related resources together, which enables simpler administration and access.

For details, see *Migration Portal Projects Owner* .

Monitoring resources in your estate

In the HM console, *estate* refers to all the resources you have access to. Just as physical assets can be considered part of an estate, the resources pertaining to your database lifecycle are part of your HM estate. Managing your estate in HM means managing your resources, whether monitoring their condition, keeping them secure, or creating new ones as needed.

Viewing your resources as an estate differs from viewing them in projects because it gives a single unified view of all your resources.

Your resources are grouped into different categories.

- ***Postgres clusters***
- ***Migrations***
- ***Catalogs***
- **OSS Library**

Exploring your estate's capabilities

In the **Postgres** tab of the estate, you can:

- Monitor your Postgres clusters, whether HM-managed or external.
- Access any Postgres clusters created with HM. Select the database cluster in the table to open a detailed view.
- Create a destination cluster for a migration.

In the **Migrations** tab of the estate, you can:

- View an *assessment of the databases* that you connected to HM to prepare them for migrations.
- Select a specific database and view *an assessment of its schemas* .
- Select a specific database and *perform a schema migration with Migration Portal integration* .
- Create a destination cluster for a migration and perform a database migration.

In the **Catalog** tab of the estate, you can:

- View all *Apache Iceberg®* tables attached to or created with HM.

In the **Apps** tab of the estate, you can:

- Access OSS Library apps to manage and monitor your customer data.

Quick Actions menu

From anywhere in the estate pages, you have access to a set of common actions on the **Quick Actions** menu. The actions include:

- **Create EDB Postgres AI Cluster** — Create an EDB Postgres AI cluster in an existing project.
- **Add Self-managed Cluster** — Add a new self-managed cluster to an existing project.

Launchpad

In the console, Launchpad gives you access to third-party services that you can use to enhance the capabilities of HM. For example, you can access *Customizing Migration Portal secrets for secure internal communication* from Launchpad. These services are available to all users of HM, regardless of the platform on which it's deployed.

To access Launchpad, from your user menu, select **Launchpad**.

All available services are displayed in Launchpad with a brief description of each service. A search filter is available to help you find the service you're looking for. Services include:

- *Grafana dashboard setup for Agent Factory metrics*
- *Langflow FAQ*
- Prometheus
- Thanos - Query
- Thanos - Store Gateway
- Thanos - Compactor
- Thanos - Rule
- Alert Manager
- EDB Migration Portal

User menu

Your user menu gives you centralized access to some portal-related actions and some account-related options.

Portal-related options

- **Launchpad** — Opens *Launchpad* to access third-party services installed on HM.
- **Settings** — Lets you set your organization name (via **Profile**) and *set your monitoring notification preferences* .
- **Projects** — Opens the Projects overview .
- **Tags** — Opens the **Tags** page where you can add tag names and select a color for their tag/badge in the display.
- **User Management** — Opens the *User Management* page where you can add users and assign roles.
- **Usage Report** — Shows *information about each cluster's use of resources* , including disk IO, vCPU, memory, and disk as well as uptime.
- **Activity Log** — Shows a list of actions that were performed, including creating, deleting, and updating resources managed in HM.

Account options

- **My Account** — Lets you view your profile, view and manage access keys, and manage your account, organization, and project notifications.
- **Logout**

Accessing technical details

You can access technical details about your current session and environment by selecting **About** in the footer of any page in the HM console. The "Troubleshooting Information" view provides essential details often required by EDB Support to diagnose issues, including:

Version and Build — Identifies the specific Hybrid Manager (HM) software release and internal build timestamp.

Scenarios — Lists the active capabilities currently deployed in your environment (such as AI, analytics, or migration).

UI Version — Specifies the current version of the console interface.

Organization and Session IDs — Displays the unique identifiers for your organization and your current browser session.

Client details — Shows your browser type, operating system, and language settings.

5.3.2.4.2 Postgres clusters

To see information about all the clusters across all the projects in your account, from the **Estate** page, select the **Postgres** tab.

Viewing summary statistics

At the top of the **Postgres** tab, you can view cards that summarize the following aspects of your Postgres clusters:

- **Active Alerts** — The aggregated number of high-, medium-, and low-severity alerts across all of the Postgres clusters monitored in your estate.
- **Top CPU Usage** — Up to the top nine clusters with the most CPU usage across all of your Postgres clusters monitored in your estate. You can page through the clusters using the control at the bottom of the card.
- **Resource Usage** — The aggregated CPU, memory, and storage usage across all of your Postgres clusters monitored in your estate.

Clusters table

The clusters table is a list of all of the available Postgres clusters across your projects, which helps you monitor their performance, health, status, and other metrics. You can modify the view to see more or fewer columns, and you can apply filters to see only clusters of interest.

Selecting columns to display

By default, the table shows these columns:

- Project Name — Project that the cluster belongs to.
- Cluster — Cluster name.
- Management — How the cluster is managed: by appliance or self-managed.
- Cluster Type — The type of cluster: PGD, single-node, or HA.
- Status — Current status of the cluster: OK, Error, Caution, or Working.
- Engine — Current Postgres distribution type this cluster is running: Postgres, EDB Postgres Advanced Server, or EDB Extended Server.
- Version — Major version of Postgres distribution the cluster is running: 13, 14, 15, 16, or 17.
- Alerts — Count of high-, medium-, and low-severity alerts for the cluster.

- Health Scores — Scorecards for index, statistics, configuration, and security health of database instances.
- TPS — Transactions per second being made by the cluster and a line graph summarizing recent trends.
- CPU — Percentage of CPU currently used by the cluster and a line graph summarizing recent trends.
- Memory — Percentage of memory currently used by the cluster and a line graph summarizing recent trends.
- Disk — Percentage of disk space currently used by the cluster and a line graph summarizing recent trends.
- Tags — Tags associated with the cluster.

You can also select the columns to display. Select the gear to open a menu with a search box at the top. Use the search box to filter columns to display using text matching. From the list of column names that are returned, select the ones you want to appear in the table.

These are the other columns you can display:

- # of Alerts — Total number of alerts for the cluster.
- # of Cores — Total number of cores used by the cluster.
- Conn. Usage — Connection usage as a percentage.
- IO Wait % — Percentage of the wait events of type Input/Output.
- IOPS — Number of input/output operations per second for the cluster.
- Kernel Version — Platform type of a self-managed cluster.
- Last Backup — ISO 8601 timestamp of the last time the cluster was backed up.
- OS Version — Operating system version of a self-managed cluster.
- Provider — ID of the CSP provider.
- Region — Locationid for an appliance cluster.
- Rep. Lag — Replication lag in milliseconds.
- TDS — Total database size in gibabytes.

> **Tip**
>
> To reset the columns to the default set, from the gear menu, select **Reset Columns**.

Filtering the clusters to view

You can narrow the list of clusters displayed to include the ones you're interested in using either of two techniques:

- Use the table's **Search** box to search by name for the clusters you want to display.
- Use the **Filter** control to display clusters that match the following properties. Each property has a submenu for choosing the specific type or values related to the property.
- Cluster Type
- Engine
- Health Score (Index, Statistic, Configuration, Security)
- Include Deleted Clusters
- Location

- Management — Hybrid Manager - Clusters created and managed within Hybrid Manager (HM). - Cloud Hosted Databases - External clusters managed by a cloud service provider. - External CNP Clusters - External clusters running open-source or EDB versions of CloudNativePG (for example, EDB Postgres AI for CloudNativePG Global Cluster or EDB Postgres AI for CloudNativePG Cluster). - Self Managed — External clusters running on a server, VM, etc. This excludes external EDB Postgres Distributed clusters.
- Postgres Version
- Project Name
- Status

Monitoring data staleness

To see how often the data on the table is refreshed, refer to the values provided for **Estate page > Postgres table** in *Metrics latency* .

A message under the table shows the last time the table was updated with new data.

5.3.2.4.3 OSS Library

Hybrid Manager (HM) provides a set of open-source applications that can be deployed and managed directly from the HM console.

Available applications

Application	Version	License	Official documentation
Apache Airflow	v3.0.2	Apache-2.0 License	*Apache Airflow*
Apache Superset	v6.0.1rc1	Apache-2.0 License	*Asset Library apps pgAdmin4 and Apache Superset fail to deploy on RHOS*
Metabase	v0.58.7	GNU Affero general public license (AGPL)	*Metabase fails to deploy when using an EPAS cluster as its database backend*
pgAdmin 4	v8.12	*Troubleshooting*	*Troubleshooting*
pgAdmin 4	v9.13	*Troubleshooting*	*Troubleshooting*
pgBadger	v13.2	PostgreSQL License	*pgBadger*
HammerDB	v5.0	GPL-3.0 license	*HammerDB*

Deployment and architecture

Apps deploy as containerized applications within your HM Kubernetes environment using a `.yaml` package file. This deployment ensures database traffic remains entirely within your private network (VPC/VNET) and security perimeter. The architecture supports horizontal scaling and external PostgreSQL application databases for high availability.

Important

When deploying apps that require backend database connections, use a dedicated database within your Postgres cluster for each app. Apps sharing the same database risk migration failures due to schema conflicts during the initialization phase. !!!

> **Note**
>
> OSS Library apps support Community PostgreSQL as the backend application database. pgAdmin 4 also supports EDB Postgres Advanced Server (EPAS). !!!

Apache Airflow

> **Important**
>
> Regular upstream releases deliver security updates. Keep your Airflow instance up to date to receive the latest patches. !!!

Apache Airflow® is an open-source workflow orchestration platform deployed and managed within Hybrid Manager (HM). Its extensible Python framework supports integration with external technologies, and a web-based UI accessible from the HM console provides visibility into workflow execution and debugging. Airflow supports both single-node and distributed deployments.

Key features

- Web-based UI for monitoring, managing, and debugging workflow execution.
- Python-based workflow definitions using Directed Acyclic Graphs (DAGs).
- Built-in scheduler supporting cron-based and event-driven triggers.
- Extensible operator library for integrating with external systems and services.
- Git synchronization for deploying DAG files from a repository.
- Distributed task execution across multiple workers.

Workflows as code in Apache Airflow

Airflow workflows are defined entirely in Python:

- Pipelines are defined in code, supporting dynamic DAG generation and parameterization.
- Built-in operators can be extended for custom integrations.
- Jinja templating is supported for runtime variable substitution.

Supported data sources

Airflow connects to nearly any data source via Connections , including PostgreSQL, MySQL, AWS, and Snowflake. Airflow's Connection stores credentials and other information necessary for connecting to external services.

Requirements

Technical requirements

- Python: v3.10+.
- **(Required) Database:** Airflow requires a database to store workflow state, task instances, DAG definitions, and run history. You must create this database before deploying Airflow.

- PostgreSQL v13+ is recommended for production. Have the host, port, database name, username, and password ready before installation. You can obtain these from the **Connect** tab of your cluster in the HM console.
- Airflow also supports MySQL 8.0.17+ and MariaDB 10.2.2+.
- **(Optional) Storage:** Git Sync configuration for syncing DAGs from public or private Git repositories using SSH keys. If using Git sync, prepare the repository URL, branch name, and SSH private key.

Deploying Airflow

1. In the HM console, navigate to **Asset Library**, then select **Apps**.
2. Select **Apache Airflow** and select **Deploy**.
3. Under **Identity**, enter a name for your deployment and select the target project.
4. Under **Parameters**, fill in the required fields. See *HNSW parameters (Index Type = hnsw)* for field descriptions.
5. Select **Deploy** to install Airflow.

Parameters

Parameter	Description	Required
data.metadataConnection	External PostgreSQL connection details: database name, host, port, password, username, SSL mode (*require* recommended), and protocol (*postgresql*).	Yes
web.defaultUser	Default admin username for the Airflow UI.	Yes
web.password	Default admin user password for the Airflow UI.	Yes
dags.gitSync.enabled	Toggle to enable syncing DAGs from a Git repository.	No
dags.gitSync.repo	Git repository URL containing your DAGs.	No
dags.gitSync.branch	Branch to sync from. Default: *main*.	No
dags.gitSync.subPath	Subfolder in the repository containing DAG files. Default: *dags*.	No
dags.gitSync.sshKey	SSH private key for authenticating to a private Git repository.	No
dags.gitSync.wait	Seconds between Git sync intervals. Default: *60*.	No
dagProcessor.replicas	Number of DAG processor replicas. Airflow 3.0+ only. Min: 1, max: 3.	No
scheduler.replicas	Number of scheduler replicas. Min: 1, max: 3.	No

Note

Once you have deployed Apache Airflow in HM, you can access it in the HM console from **Estate > Apps** or your project's **Apps** view. !!!

Post-deployment configuration

Once Airflow is deployed, complete these steps to finish your setup:

1. Navigate to your **Project** and select the **Apps** tab, or navigate to **Estate > Apps** to check the deployment status. When the status shows **Ready**, launch the app to continue.
2. Open **Airflow** from the HM console.
3. Log in with the admin username you set in `web.defaultUser` during deployment.

4. Navigate to **Admin > Connections** to add credentials for any databases or services your DAGs will connect to.
5. If you enabled Git sync, verify that your DAGs are appearing in the Airflow UI.

Support resources

As Airflow OSS is community-supported, use these resources for technical guidance and updates:

- Airflow community — peer-to-peer support from the community.
-

Airflow documentation — self-service deployment and feature guides, as well as additional information on Airflow installation and configuration.

- Airflow GitHub repository — source code, issue tracker, and release notes.
- Best Practices — guidance for developing and maintaining efficient DAGs.
- Scheduler — guidance for fine-tuning scheduler performance.
-

Tutorials — step-by-step tutorials for getting started with Airflow.

-

How-to Guides — practical guides for common Airflow tasks.

Apache Superset

> **Important**
>
> Regular upstream releases deliver security updates. Keep your Superset instance up to date to receive the latest patches. !!!

Apache Superset is an open-source business intelligence web application that provides data visualization, dashboard creation, and SQL-based data exploration for databases.

> **Warning**
>
> Superset doesn't support Red Hat OpenShift (RHOS). !!!

Key features

- No-code interface for building charts.
- Web-based SQL Editor for querying.
- Semantic layer for defining custom dimensions and metrics.
- Support for SQL databases and data engines.
- A range of visualization types, including bar charts and geospatial visualizations.
- A configurable caching layer to reduce database load.
- Configurable security roles and authentication options.

- An API for programmatic customization.
- Cloud-native architecture.

Apache Superset components

- **Core application** — Includes a Python (Flask) backend, an API layer, and a React frontend. When a user loads a chart or dashboard, Superset queries the configured data warehouse and renders the result as a visualization.
- **Metadata database** — Stores chart definitions, dashboard configurations, user information, and logs. PostgreSQL and MySQL support production use. Avoid SQLite for production. Some installation methods use it by default.
- **Caching layer (optional)** — Reduces data warehouse load by storing query results, and acts as a message broker for the worker process. Redis is the most common choice.
- **Worker and beat (optional)** — Handles asynchronous task execution and scheduling, typically via Celery.

Enable the caching layer and worker to access these features:

- Alerts and reports
- Caching
- Async queries
- Dashboard thumbnails

Supported data sources

Superset can query data from any SQL-based datastore or data engine (Presto, Trino, Athena, and more) that has a Python DB-API driver and a SQLAlchemy dialect.

Requirements

Technical requirements

- **(Required) Database:** Superset requires an external database to store chart definitions, dashboard configurations, user information, and logs. You must create this database before deploying Superset.
- PostgreSQL v13+ is recommended for production. Have the host, port, database name, username, and password ready before installation. You can obtain these from the **Connect** tab of your cluster in the HM console.
- Superset also supports MySQL v8.x+.
- **Resources:** Worker pods default to 1Gi Memory and 250m CPU.

Deploying Superset

1. Navigate to **Asset Library**, then select **Apps** in the HM console.
2. Select **Apache Superset** and select **Deploy**.
3. Under **Identity**, enter a name for your deployment and select the target project.
4. Under **Parameters**, fill in the required fields. See *HNSW parameters (Index Type = hnsw)* for field descriptions.
5. Select **Deploy** to install Superset.

Parameters

Parameter	Description	Required
PostgreSQL Host	Hostname or IP address of your PostgreSQL server. Examples: *postgres.database.svc.cluster.local*, *prod-postgres.example.com*, *10.0.1.50*.	Yes
Database name	Name of the PostgreSQL database.	Yes
Database username	PostgreSQL user for the Superset application.	Yes
Database password	PostgreSQL password for the database connection.	Yes
PostgreSQL Port	Port number for the PostgreSQL connection. Default: *5432*.	No
replicaCount	Number of Superset replicas. Default: *1*.	No
CPU limit (node)	Maximum CPU for the Superset node. Default: *1000m*.	No
Memory limit (node)	Maximum memory for the Superset node. Default: *2Gi*.	No
CPU limit (worker)	Maximum CPU for the Superset worker. Default: *1000m*.	No
Memory limit (worker)	Maximum memory for the Superset worker. Default: *2Gi*.	No
CPU request (worker)	Requested CPU for the Superset worker. Default: *500m*.	No
Memory request (worker)	Requested memory for the Superset worker. Default: *1Gi*.	No

Post-deployment configuration

Once Superset is deployed, complete these steps to finish your setup:

1. Navigate to your **Project** and select the **Apps** tab, or navigate to **Estate > Apps** to check the deployment status. When the status shows **Ready**, launch the app to continue.
2. Open **Superset** from the HM console.
3. Navigate to **Settings** > **Database Connection** to add your data sources for analysis.

Support resources

As Superset OSS is community-supported, use these resources for technical guidance and updates:

- Superset community — peer-to-peer support from the community.
- Superset documentation — self-service deployment and feature guides, as well as additional information on Superset installation and configuration.
- Superset GitHub repository — source code, issue tracker, and release notes.

Metabase

Important

Regular upstream releases deliver security updates. Keep your Metabase instance up to date to receive the latest patches. !!!

Metabase is a community-supported, open-source business intelligence tool available as an app in Hybrid Manager.

Key features

- Question builder that doesn't require SQL.
- SQL mode for ad hoc queries.
- Interactive visualizations and dashboards.
- Data exploration and automated analysis.
- Embeddable charts and dashboards.
- Support for multiple database backends (PostgreSQL, MySQL, and more).

Requirements

Supported data sources

For production, Metabase recommends using PostgreSQL as your Metabase application database:

- PostgreSQL (Primary), plus various SQL and NoSQL databases supported by the upstream Metabase image.

Before deploying, have your PostgreSQL database connection details ready (host, port, database name, username, and password). You can obtain these from the **Connect** tab of your cluster in the HM console. See *HNSW parameters (Index Type = hnsw)* for the full list of required and optional fields, including resource limits.

Deploying Metabase

Install Metabase directly in Hybrid Manager (HM).

1. In the HM console, navigate to **Asset Library**, then select **Apps**.
2. Select **Metabase**, select **Deploy**, and fill in the required fields. See *HNSW parameters (Index Type = hnsw)* for field descriptions.

Parameters

Parameter	Description	Required
PostgreSQL Host	Hostname or IP address of your PostgreSQL server. Examples: *postgres.database.svc.cluster.local*, *prod-postgres.example.com*, *10.0.1.50*.	Yes
Database Name	Name of the PostgreSQL database. Must exist on the PostgreSQL server.	Yes
Database User Name	PostgreSQL user for the Metabase application. Must have access to the database.	Yes
Database Password	PostgreSQL password for the database connection.	Yes
PostgreSQL Port	Port number for the PostgreSQL connection. Default: *5432*.	No
CPU request	Requested CPU for the Metabase pod. Default: *500m*.	No
Memory request	Requested memory for the Metabase pod. Default: *1Gi*.	No
CPU limit	Maximum CPU for the Metabase pod. Default: *1000m*.	No

continues on next page

Table 4 – continued from previous page

Parameter	Description	Required
Memory limit	Maximum memory for the Metabase pod. Default: *2Gi*.	No

Note

Once you have deployed Metabase in HM, you can access it in the HM console from **Estate > Apps** or your project's **Apps** view. !!!

Post-deployment configuration

Once Metabase is deployed, complete these steps to finish your setup:

1. Navigate to your **Project** and select the **Apps** tab, or navigate to **Estate > Apps** to check the deployment status. When the status shows **Ready**, launch the app. A setup wizard guides you through the remaining configuration steps.
2. Create an admin account in Metabase to manage all subsequent user additions, database connections, and system configuration.
3. Choose between self-service analytics, analytics embedded in applications, or both. You can change this later.
4. Provide your database hostname, port, name, and credentials. You can also skip this step and explore the sample database included with Metabase:

•

Adding and managing databases - Connecting to supported databases

1. Choose whether to allow anonymous usage data collection. Metabase never collects anything about your data or question results.

Support resources

As Metabase OSS is community-supported, use these resources for technical guidance and updates:

- Metabase discussion forum — peer-to-peer support from the community.
- Metabase documentation — self-service deployment and feature guides, as well as additional information on Metabase installation and configuration.
- Metabase GitHub repository — source code, issue tracker, and release notes.

pgAdmin 4 (v8.12)

Important

Regular upstream releases deliver security updates. Keep your pgAdmin instance up to date to receive the latest patches. !!!

pgAdmin 4 is an open-source web-based interface for administering PostgreSQL databases. When deployed as an app in Hybrid Manager, it provides remote access to your PostgreSQL clusters for managing database objects and executing queries.

Warning

pgAdmin 4 isn't supported on Red Hat OpenShift (RHOS). !!!

Key features

- Graphical interface for managing PostgreSQL database objects.
- Query tool with syntax highlighting, query history, and execution plan visualization.
- Server monitoring including statistics, logs, and activity.
- Backup and restore tools.
- Schema, role, and permission management.

Requirements

Technical requirements

- Kubernetes cluster.
- Persistent volume: 10Gi (default) for data storage.
- Resources: Minimum 512Mi memory, 500m CPU (limits: 1Gi memory, 1000m CPU).

Supported PostgreSQL versions

- PostgreSQL 12 to 17.
- EDB Advanced Server: 12 to 16.

Deploying pgAdmin 4

1. In the HM console, navigate to **Asset Library**, then select **Apps**.
2. Select **pgAdmin 4**, select the version, and select **Deploy**.
3. Under **Identity**, enter a name for your deployment and select the target project.
4. Configure the replica count (default: `1`) and storage size if needed.
5. Select **Deploy**.

Note

Once you have deployed pgAdmin 4 in HM, you can access it in the HM console from **Estate > Apps** or your project's **Apps** view. !!!

Post-deployment configuration

Once pgAdmin 4 is deployed, complete these steps to finish your setup:

1. Navigate to your **Project** and select the **Apps** tab, or navigate to **Estate > Apps** to check the deployment status. When the status shows **Ready**, launch the app to continue.
2. Open **pgAdmin 4** from the HM console.
3. Set the master password.

4. Register your PostgreSQL servers within the pgAdmin interface to begin managing your databases.

Support resources

As pgAdmin OSS is community-supported, use these resources for technical guidance and updates:

- pgAdmin support page — get support for any queries related to pgAdmin 4.
-

pgAdmin documentation — self-service deployment and feature guides, as well as additional information on pgAdmin installation and configuration.

- pgAdmin GitHub repository — source code, issue tracker, and release notes.

pgAdmin 4 (v9.13)

> **Important**
>
> Regular upstream releases deliver security updates. Keep your pgAdmin instance up to date to receive the latest patches. !!!

pgAdmin 4 is an open-source web-based interface for administering PostgreSQL databases. When deployed as an app in Hybrid Manager, it provides remote access to your PostgreSQL clusters for managing database objects and executing queries.

> **Warning**
>
> pgAdmin 4 isn't supported on Red Hat OpenShift (RHOS). !!!

Key features

- Graphical interface for managing PostgreSQL database objects.
- Query tool with syntax highlighting, query history, and execution plan visualization.
- Server monitoring including statistics, logs, and activity.
- Backup and restore tools.
- Schema, role, and permission management.

Requirements

Technical requirements

- Kubernetes cluster.
- Persistent volume: 10Gi (default) for data storage.
- Resources: Minimum 512Mi memory, 500m CPU (limits: 1Gi memory, 1000m CPU).

Supported PostgreSQL versions

- PostgreSQL 13 to 18.
- EDB Advanced Server: 13 to 18.

Deploying pgAdmin 4

1. In the HM console, navigate to **Asset Library**, then select **Apps**.
2. Select **pgAdmin 4**, select the version, and select **Deploy**.
3. Under **Identity**, enter a name for your deployment and select the target project.
4. Configure the replica count (default: `1`) and storage size if needed.
5. Select **Deploy**.

> **Note**
>
> Once you have deployed pgAdmin 4 in HM, you can access it in the HM console from **Estate > Apps** or your project's **Apps** view. !!!

Post-deployment configuration

Once pgAdmin 4 is deployed, complete these steps to finish your setup:

1. Navigate to your **Project** and select the **Apps** tab, or navigate to **Estate > Apps** to check the deployment status. When the status shows **Ready**, launch the app to continue.
2. Open **pgAdmin 4** from the HM console.
3. Set the master password.
4. Register your PostgreSQL servers within the pgAdmin interface to begin managing your databases.

Support resources

As pgAdmin OSS is community-supported, use these resources for technical guidance and updates:

- pgAdmin support page — get support for any queries related to pgAdmin 4.
-

pgAdmin documentation — self-service deployment and feature guides, as well as additional information on pgAdmin installation and configuration.

- pgAdmin GitHub repository — source code, issue tracker, and release notes.

pgBadger

> **Important**
>
> Regular upstream releases deliver security updates. Keep your pgBadger instance up to date to receive the latest patches. !!!

pgBadger analyzes PostgreSQL logs from your HM clusters and generates HTML reports with statistics on database activity, performance, and errors.

Key features

- Parallel log parsing.
- HTML reports with interactive charts.
- Slow query identification.
- Connection and session statistics.
- Lock and checkpoint analysis.
- Temporary file and autovacuum monitoring.
- Support for stderr, csvlog, and syslog log formats.

Requirements

Technical requirements

- Kubernetes or OpenShift cluster.
- Persistence: Default 1Gi volume to store reports and uploaded logs.

Configuration requirements

- Retention: `reportRetentionDays` (default 30) controls how long reports are kept.
- Upload limits: `maxUploadSize` (default 50MB) limits the size of log files you can analyze.

Supported data sources

- PostgreSQL log files (text format).

Deploying pgBadger

1. Navigate to **Asset Library**, then select **Apps** in the HM console.
2. Select **pgBadger** and adjust resource limits based on expected log volume (default 512Mi memory).
3. Select **Deploy**.

> **Note**
>
> Once you have deployed pgBadger in HM, you can access it in the HM console from **Estate > Apps** or your project's **Apps** view. !!!

Post-deployment configuration

Once pgBadger is deployed, complete these steps to finish your setup:

1. Navigate to your **Project** and select the **Apps** tab, or navigate to **Estate > Apps** to check the deployment status. When the status shows **Ready**, launch the app to continue.
2. Open the **pgBadger Reports** interface in the HM console.
3. Upload your PostgreSQL log files to generate a new visual report.

Support resources

As pgBadger OSS is community-supported, use these resources for technical guidance and updates:

- pgBadger support — get support for any queries related to pgBadger.
- pgBadger documentation — self-service deployment and feature guides, as well as additional information on pgBadger installation and configuration.
- pgBadger GitHub repository — source code, issue tracker, and release notes.

HammerDB

Important

Regular upstream releases deliver security updates. Keep your HammerDB instance up to date to receive the latest patches. !!!

HammerDB is an open-source database load testing and benchmarking tool supporting multiple database platforms. When deployed in Hybrid Manager (HM), HammerDB runs TPROC-C benchmarks against an existing PostgreSQL cluster and generates detailed performance reports with throughput, latency, and infrastructure metrics.

Note

The HammerDB package doesn't deploy PostgreSQL. You must provide your own PostgreSQL cluster with superuser or CREATEDB privileges. !!!

Key features

- HammerDB database load testing and benchmarking tool.
- Configurable resources and replicas.

Requirements

Technical requirements

- **PostgreSQL Cluster** — An existing PostgreSQL instance accessible from the target namespace.
- Superuser or CREATEDB privileges required (HammerDB creates its own test schema).
- Connection string in the format: `postgres://user:pass@host:5432/db?sslmode=require` .

Before deploying, have your database connection details ready (host, port, database name, username, and password). You can obtain these from the **Connect** tab of your cluster in the HM console. See *HNSW parameters (Index Type = hnsw)* for the full list of required and optional fields, including resource limits.

Supported database backends

Database	Release
Oracle (TimesTen)	12c / 18c / 19c / 21c
Microsoft SQL Server	2017 / 2019 / 2022
Db2	11.1 / 12.1

continues on next page

Table 5 – continued from previous page

Database	Release
MySQL (Amazon Aurora)	5.7 / 8.0
MariaDB	10 / 11
PostgreSQL	10 / 11 / 12 / 13 / 14 / 15 / 16 / 17

Deploying HammerDB

Install HammerDB directly in HM.

1. In the HM console, navigate to **Asset Library**, then select **Apps**.
2. Select **HammerDB** and select **Deploy**.
3. Under **Identity**, enter a name for your deployment and select the target project.
4. Under **Parameters**, fill in the required fields. See *HNSW parameters (Index Type = hnsw)* for field descriptions.
5. Select **Deploy** to install HammerDB.

> **Note**
>
> Once you've deployed HammerDB in HM, you can access it in the HM console from **Estate > Apps** or your project's **Apps** view. !!!

Parameters

Parameter	Description	Required
connectionString	PostgreSQL connection string. Format: *postgres://user:pass@host:5432/db?sslmode=require*. Requires superuser or CREATEDB privileges.	Yes
CPU limit	Maximum CPU for the HammerDB pod. Default: *1000m*.	No
Memory limit	Maximum memory for the HammerDB pod. Default: *2Gi*.	No
size	Size of the persistent volume for storing HammerDB data and benchmark results. Default: *5Gi*.	No

Post-deployment configuration

Once you've deployed HammerDB, complete these steps to finish your setup:

1. Navigate to your **Project** and select the **Apps** tab, or navigate to **Estate > Apps** to check the deployment status. When the status shows **Ready**, launch the app to continue.
2. Open **HammerDB** from the HM console.
3. Configure your benchmark workload and run TPROC-C tests against your PostgreSQL cluster.

Support resources

As HammerDB OSS is community-supported, use these resources for technical guidance and updates:

- HammerDB GitHub repository — source code, issue tracker, and peer-to-peer support from the community.
-

HammerDB documentation — self-service deployment and feature guides, as well as additional information on HammerDB installation and configuration.

-

HammerDB Blog — performance updates and guides.

5.3.3 For Users of Hybrid Manager

5.3.3.1 Projects

In Hybrid Manager (HM), you use projects to associate organizationally related resources together. Projects simplify administering and accessing these resources.

To view a list of projects, select the **Projects** tab. Each project card shows:

- The project name
- The number of clusters in the project
- The number of users with access to the project
- A control for adding tags to the project, key-value pairs that help you to organize your resources
- Logs of active cloud service providers (CSPs) configured in the project.

An ellipses menu in each card gives you quick access to the functions you can perform when viewing the *Project details* .

If you have owner privileges, from this view you also can create projects.

5.3.3.1.1 Creating projects

1. Select the **Projects** tab.
2. Select **Create New Project**.
3. Enter a name for the project.
4. Optionally, add tags to the project.
5. Select **Create New Project**.

After you create the project, you see the project details. You're the owner of the new project, and you can start to add resources.

5.3.3.1.2 Project details

The detailed view of the project is where you perform your project management tasks. To open the detailed view, either create a project or select a project from the list of projects on the **Projects** page. This view includes a description of each cluster in the project and metrics including memory usage, CPU usage, storage usage, disk IOPS and transaction rate, and the database size.

In a project, you can:

- Create, view, and manage clusters .

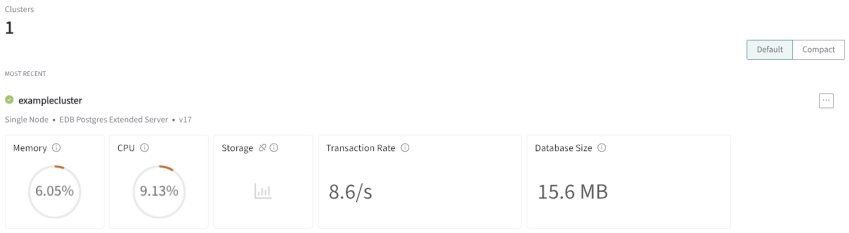


Fig. 2: Project Overview

- *Manage storage locations* .
- Add catalogs.
- *View and manage the users in the project* .
- View and download the activity log. The activity log is a record of all the actions that were performed in the current project, such as cluster creation and deletion and software installation.
- Manage project settings , including notifications, security, and alerts.
- *View a usage report* .
- View and manage database migrations .
- *Access OSS Library apps* .

For each cluster, using the ellipsis menu in the upper-right corner of the card, you can also:

- Select **Edit Cluster** to *Edit the cluster* .
- Select **Credential Information to see the connection details for the cluster, including the connection URL, username, and the database name. You can use these details to connect to the cluster from a Postgres client. Passwords aren't shown, but you can reset the password if you need to. For more information, see *Step 2: Credentials* .

5.3.3.1.3 Managing projects

You can rename and delete projects. See *Managing projects* .

5.3.3.1.4 Configure projected object storage

By default, all projects share the same object storage configured during installation. If you need to isolate project resources, you can configure dedicated object storage for each project. Refer to *Configure projected object storage* .

Managing projects

You can rename and delete projects.

Renaming a project

You can rename a project in the single-pane-of-glass view of projects and in the **Projects** tab. From either list of projects:

1. From the ellipses menu to the right of the project name, select **Edit**.
2. Enter the new project name and select **Update Project**.

Deleting a project

You can delete a project in the single-pane-of-glass view of projects and in the **Projects** tab.

> **Note**
>
> You can't delete the default project. Hybrid Manager keeps one default project per organization and protects it from deletion.

A project must have no active resources before you can delete it. Active resources include clusters, data catalogs, marketplace apps, deployed flows, knowledge bases, and pipelines, depending on the capabilities installed in your environment. If the project still contains any of these resources, Hybrid Manager blocks the deletion and lists the resources you must remove first.

You can delete clusters from within the delete project dialog. Remove any other resource types from their own management pages before you start.

From either list of projects:

1. From the ellipsis menu to the right of the project name, select **Delete**.

You're prompted to confirm the deletion.

1. If the project contains clusters, you must delete them before you can delete the project. The display shows the remaining clusters, with a checkbox next to each one. Select the check box for each cluster you want to delete, or select **Select All**.
2. After you select the clusters to delete, enter `delete clusters` in the text box to confirm the deletion. Entering this text enables the **Delete All Clusters** button.
3. Select **Delete All Clusters** to permanently delete the selected clusters.
4. When the project contains no active resources, a dialog box appears listing the project name with a check box. Select the check box and enter `delete project` in the text field. Entering this text enables the **Delete Project** button.
5. Select **Delete Project** to delete the project.

Settings

You can configure project-specific settings like notifications, security, and alerts. Use the **Settings** menu to access these settings.

- Select **Profile** to change your project's name at any time. Project names must be unique in a Hybrid Manager (HM) instance.
- Select ***Notifications*** to configure how the project's notifications are sent.
- Select **Security** to add externally managed keys for transparent data encryption (TDE) to a project. You can use these keys later during cluster creation to add data encryption. See *Enabling KMS for TDE* for KMS-specific instructions.
- Select ***Alerts*** to see the alert settings of the project and control how and when alerts are sent.

Notifications

With Hybrid Manager (HM), you can customize the types of notifications you want to receive. There are two types of notification channels:

- **User notifications** : You can receive notifications via email, in-app inbox, or both. These notifications are for all users who have enabled them if their role allows it.
- **System notifications** : You can receive notifications via email (mail group), webhook, pager duty, or any combination. These notifications are triggered once per event, primarily to automate tasks in your organization.

Your role in HM determines your access to notification preferences. The following types of notifications are available:

- Account : For user notification channels. You can be notified that:
- A new project role was assigned to you.
- A project role was unassigned from you.
- Your personal access key is about to expire.
- Global : For user and system notification channels (visible only to the organization owner). You can be notified that:
- A new project was created.
- The machine access key is about to expire.
- Projects : For user and system notification channels. You can be notified that:
- The active cluster was paused.
- A paused cluster was reactivated.

Configuring notifications

Project owners/editors and organization owners/admins can enable or disable notifications for the events they can see. They can receive notifications in the in-app inbox, by email, or both.

Notification settings made by a user apply only to that user. If an email notification is enabled, the email is sent to the email address associated with the user's login.

System notification channels can be configured at the global level by the organization owner or at the project level by the project owner. By default, each organization's project inherits the global-level system notifications channel. However, you can enable/disable or reconfigure the system notifications channel at the project level to override the default settings.

Configure notifications at the global level

Only the organization owner can set the global notifications.

For webhook:

1. From your profile menu, select **Settings**.
2. Select the **Notifications** tab.
3. Select **Webhook** and enter:

- The host URL of the webhook
- The method of the webhook - POST or PUT

1. Optionally, provide one of these authentication parameters:

- For Basic Auth: username and password
- Bearer Token

1. Select **Save**.

For PagerDuty:

Before configuring the PagerDuty notification channel:

- You need access to a PagerDuty account with elevated privileges.
- PagerDuty service must be created and configured with Events API v2 integration.
- In your PagerDuty account, navigate to the desired service and access its **Integrations** settings.
- Add and configure a new Events API v2 integration.
- Copy the Integration Key (routing key) provided by PagerDuty for the newly created integration.

To configure the PagerDuty notification channel:

1. From your profile menu, select **Settings**.
2. Select the **Notifications** tab.
3. Select **PagerDuty**.
4. Set the Routing Key (integration key).
5. Select **Save**.

For email:

1. From your profile menu, select **Settings**.
2. Select the **Notifications** tab.
3. Select **Email** and enter:

- Email To - Optionally, for the system email notifications (Email group)
- Email From - for the email which is sending the email
- Host - URL of the email service
- Port - port of the email service
- Username - username of the email service
- Password - of the email service

1. Select **Save**.

Configure notification at project level

Project-level notification settings can override global settings. Each project can have its unique configuration.

1. Select the **Projects** tab
2. Select any project.
3. Go to the Project Overview page.
4. Select **Settings** on the left-side menu.
5. Select **Notifications**.
6. Follow the steps under *Configure notifications at the global level* for each system notification channel.

Viewing notifications

Users in the following roles can view the notifications:

- Organization owners can view the global notifications.
- Project owners/editors can view the project notifications.
- All users can view their account notifications.

Each notification indicates the level and project the user belongs to with multiple roles assigned in HM.

Select the bell at the top of your HM console to view the in-app notifications. You can read a notification, mark it as unread, or archive it.

To view the email notifications, check the inbox of your configured email address.

Managing notification preferences

1. From your profile menu, select **My Account**.
2. Select the **Notifications** tab.
3. Scroll to the **Account**, **Global**, or **Project** section to view the events you want to manage.
4. Expand the section to see the list of events.
5. For each notification, you can: - Use the toggle button in front of the event name to opt in or out of receiving notifications for an event. - Use the check boxes to customize the email and in-app notifications.

Alerts

Alerting is integral to the Hybrid Manager (HM) observability ecosystem. It proactively notifies about critical events that impact the performance of the Postgres cluster. It continuously monitors essential performance metrics and triggers alerts whenever predefined thresholds are exceeded. This robust alerting system integrates seamlessly into the HM ecosystem, leveraging real-time data collected from Postgres clusters and their infrastructure across all regions. Alerts are enabled by default for each project created in HM.

The alerting system ensures timely notifications, allowing for quick and informed actions. You can configure specific thresholds and intervals to customize the alerting experience. This integration of monitoring and alerting enables swift responses to potential issues and so you can maintain optimal cluster performance.

View the alerts

You can access the alerts on the **Monitoring** tab of the cluster detail page and view the alert details for a specific cluster.

For HM clusters:

1. In the HM console, select **Projects**.
2. Select any project from the list.
3. In the left navigation, select **Clusters**.
4. Select any cluster with a ready state.
5. On the cluster detail page, select the **Monitoring** tab.

For external HM clusters:

1. In the HM console, select **Estate**.
2. Go to the table listing all the clusters.

3. Select a ready cluster with **Management** as **Self-managed**.
4. On the cluster detail page, select the **Monitoring** tab.

The **Monitoring** tab displays the Active Alerts list in a table. At the top of this tab, you can view a summary of the active alerts. Possible severity levels are High, Medium, and Low.

You can search for specific alerts using the:

- Search box for query-based search
- Date range selector to search within a specific date range
- Filter to search by severity level

Select any nodes in the Node column to view the charts as they were when the alert was triggered. Selecting a node applies the filters in the **Monitoring** tab and displays the relevant charts.

You can also:

- Select **Download CSV** to download the alert details in CSV format.
- Sort the alerts list based on severity, alert start time, and alert name.
- Select **Refresh** to refresh the list.
- Use the column selector to select the columns to display in the table.

Configure alerts

Project admins can configure alerts at the project level. Once the alerts are configured and applied, the system takes a few minutes to fully process the alerts.

To configure alerts:

1. At the top of the HM console, select **Projects**.
2. Select any project from the list.
3. In the left navigation, select **Settings**.
4. Select **Alerts**.

For each alert, select **Edit** to set:

- Enable/Disable : Activate or deactivate a specific alert.
- Notify After : Set the time duration after which the alert is triggered.
- Threshold : Specify the percentage or values that trigger an alert when exceeded.

You can configure these alerts at the database level:

- Connections Usage
- No Recent WAL Archiving Successes
- Node Down
- Replication Lag in Bytes
- Total Connections
- Total Inactive Replication Slots
- Transactions Rollback
- WAL Keeps Growing

- WAL Running Out of Disk Space

You can configure these alerts at the host level:

- CPU Utilization
- Disk IOPS
- Disk Usage
- Memory Usage

Configure dedicated object storage for a project

By default, all projects share the same object storage configured during the installation. If you need to isolate project resources, you can configure dedicated object storage for each project.

1. In the Hybrid Manager console (HM), select the **Projects** tab.
2. Select the project for which you want to configure dedicated object storage.
3. Get the project ID from the URL. The project ID is the string of characters after `/projects/` in the URL.

For example, in the URL `https://hybrid-manager.example.com/projects/prj_Abc123` , the project ID is `prj_Abc123` .

4. Normalize the project ID by replacing all underscores (_) with hyphens (-) and convert all letters to lowercase. For example, `prj_Abc123` becomes `prj-abc123` .
5. Create the object storage secret in your Kubernetes cluster. Use the provider-specific `stringData:` block from *Configuring object storage* . The secret body is identical to the default `edb-object-storage` secret — only the name differs, and it must be `edb-object-storage-<normalized-project-id>` .

```
apiVersion: v1
kind: Secret
metadata:
  name: edb-object-storage-<normalized-project-id>
  namespace: default
stringData:
  # Provider-specific fields — see Configuring object storage.
  ...
```

- Configure dedicated object storage before creating any Postgres workloads in the project. Once a project is bound to an object storage secret, the association can't be changed.
- If your HM deployment has multiple locations, apply the object storage secret to all locations. .. :: ## Verifying the binding

HM stores project-to-storage mappings in the `slo-kv-store` ConfigMap in the `default` namespace. To check which object storage a project is bound to, run:

```
kubectl get cm slo-kv-store -o yaml -n default
```

The `data` section maps each project ID to its object storage secret name. For example:

```
data:
  prj_C9cgaIh5QAZdX977: edb-object-storage-prj-c9cgaih5qazdx977 # dedicated storage
  prj_EyHfzLGVDmQCsEC2: edb-object-storage # default storage
```

If a project ID doesn't appear in the ConfigMap, the project hasn't been bound yet. Binding happens automatically when the first Postgres workload is created within the project.

Managed storage locations

Managed storage locations will be used to store data for projects in future releases. You can create a storage location, which shares the same storage as the project, in the Hybrid Manager (HM) console. However, no HM service is using managed storage locations functionality.

Usage report

The **Usage Report** page provides a view of the following historical usage metrics for clusters.

Term	Definition	Measure
Cluster Name	The name of the cluster.	•
Cluster Type	The type of the cluster.	•
Database Type	The type of the database.	•
Disk Allocated (GiB-Hour)	The amount of disk storage allocated.	GiB-Hours
Disk IO (GiB)	The amount of data read from and written to disk storage.	GiB
Disk Used (GiB-Hour)	The amount of disk storage used.	GiB-Hours
Disk Used Percent	The total amount of time that the allocated storage (disk) has been available to your database system. This does not measure activity like Disk IO, but the total duration the storage has been provisioned and available.	Hours
Memory Allocated (GiB-Hour)	The amount of memory allocated.	GiB-Hours
Memory Used (GiB-Hour)	The amount of memory used.	GiB-Hours
Memory Used Percent	The amount of RAM (memory) consumed in hours versus the amount allocated.	•
Project	The name of the project.	•
Uptime (Hour)	The total amount of time the database instance has been continuously running and available.	Hour
vCPU Allocated (Core-Hour)	The amount of vCPU allocated.	Core-Hours
vCPU Used Percent	The number of virtual CPU hours consumed versus the number allocated.	•
vCPU Used (Core-Hour)	The amount of vCPU used.	Core-Hours

5.3.3.2 Working with clusters

You can use Hybrid Manager (HM) to create and manage database clusters. The databases can be based on any of the following Postgres types:

- EDB Postgres Advanced Server

- EDB Postgres Extended Server (PGE)
- PostgreSQL

Cluster management includes creating new clusters, controlling access, monitoring cluster performance, and managing lifecycle operations such as editing, restoring, or deleting clusters. You can also create reusable templates to simplify future cluster creation.

5.3.3.2.1 What can you do?

- Create a cluster

Learn how to create a new database cluster from a predefined template or configure one from scratch.

- Manage a cluster

Perform key management tasks such as modifying configuration, restoring backups, deleting clusters, and creating templates.

- *Manage database access*

Set up users, roles, and permissions to control who can access and connect to your clusters.

- View clusters

Monitor and review the status, configuration, and activity of all your database clusters.

Creating a database cluster

You can create a database cluster from a template (if you have one) or create the cluster from scratch.

Starting points

The starting point for creating a cluster can be either the **Estate** page or a project page. When you start from the **Estate** page, you specify the project where you want to create the database. When you start from a project page, you're creating the database in that project.

To start from the **Estate** page:

1. Select **Quick Actions > Create New Database Cluster**.

If you have more than one project, you're prompted to select a project.

2. If prompted, select the project you want to create the database in.

After you select the project, or if you have only one project, the **Create Cluster** *From the Choose Your Path page* page opens.

To start from a project page, you can use either of these approaches. To get to a project page, from the **Projects** page, select the project.

- In the project page, select **Create New > Database Cluster**.
- In the project page, select **Clusters > Postgres Cluster**. On the **Clusters** page, select **Create New > Database Cluster**.

In both cases, the **Choose your path** page for creating clusters opens.

Choose your path

On the **Choose Your Path** page, you specify whether you want to create a cluster from a predefined template or from scratch as a custom build.

- To create the cluster from any available template, select **Use Template**. See *Creating a cluster from a template* .
- If no templates are available, or if you want to create a cluster from scratch, select **Custom Build**.

You can also launch template creation from here by selecting **Create New Template**. See *Create a database cluster template* .

Creating a cluster from scratch

After you select **Custom Build**, you can start to specify your new cluster.

On the ***Cluster Info*** tab, select the cluster type:

- Single node
- High availability
- Advanced high availability
- Distributed high availability

Select the tab you want to work on, or step through the tabs in order using **Next**. If you select **Advanced High Availability** or **Distributed High Availability**, instead of the **DB Configuration** and **Additional Settings** tabs, you see the ***Data Groups*** tab.

- ***Cluster Info***
- ***Cluster Settings***
- ***DB Configuration*** (optional)
- ***Additional Settings*** (optional)
- ***Replica Clusters*** (for single-node and high-availability clusters)
- ***Data Groups*** (for distributed high-availability clusters)

> **Note**
>
> Settings in optional tabs that you don't configure use default values.

Create the cluster

After you finish configuring the tabs, select **Create Cluster**. Tabs with incorrect or missing settings show a red indicator. Correct any errors and select **Create Cluster** again.

After you select **Create Cluster** with valid settings, the cluster begins provisioning, and you return to the list of clusters. A popup confirms the creation of the cluster, and the new cluster is added to the list of clusters. The cluster is created in the background. You can see the progress in the list of clusters.

If you try to view the cluster details while the cluster creation is in progress, you can see only the overview, backup, logs, and properties.

Cluster types to create

After you start to create your cluster , the **Cluster Info** tab prompts you to select the type of cluster to create. This tab also appears when you're editing a cluster.

> **Note**
>
> When editing a cluster, you can't switch between a single-node or high-availability cluster and an advanced or distributed high-availability cluster.

- ***Single Node*** creates a cluster with one primary and no standby replicas. This type of cluster is suited for test environments where high availability might not be required. You can create single-node clusters running PostgreSQL, EDB Postgres Extended Server, or EDB Postgres Advanced Server.
- ***High Availability*** creates a cluster with one primary and multiple standby replicas in different availability zones. You can create high-availability clusters running PostgreSQL, EDB Postgres Extended Server, or EDB Postgres Advanced Server.

If you select **High Availability**, you're prompted to specify the number of standby replicas for your cluster. The default is 2.

- ***Advanced High Availability*** delivers single-region, multi-AZ resilience with fast, automated failover; online upgrades; and logical replication via primary/standby nodes.

If you select **Advanced High Availability**, the tabs **DB Configuration** and **Additional Settings** no longer appear. The **Data Groups** tab appears instead.

- ***Distributed High Availability*** creates a cluster, powered by EDB Postgres Distributed (PGD), with three nodes and up to two data groups spread across three availability zones. This configuration delivers higher performance and faster recovery. You can create distributed high-availability clusters running PostgreSQL, EDB Postgres Extended Server, or EDB Postgres Advanced Server.

If you select **Distributed High Availability**, the tabs **DB Configuration** and **Additional Settings** no longer appear. The **Data Groups** tab appears instead.

See *Supported cluster types* for more information.

Cluster setup

Configure cluster settings like name and password in the **Cluster Settings** tab.

Cluster Name — Enter the name for your cluster.

Password — Enter a password for your cluster. This is the password for the user edb_admin. This password is needed to connect to the database and isn't stored in Hybrid Manager (HM).

Tags — Optionally, select **+ Add Tag** to add tags to your cluster. Tags are key-value pairs that help you organize your resources.

- To assign an existing tag, in the search bar under **Tags**, start to type the tag name until the tag you want to assign appears. Select the tag you want to assign to the cluster.
- To add a new tag, enter the full tag name and select **+ Add ``<tagname>``** . Then select a color for the tag and select **Create**. The new tag is assigned to the cluster.

Settings for single-node and high-availability clusters

The following options appear on the **Cluster Settings** tab only for single-node and high-availability clusters. You set these options in the ***Data Groups*** tab for other cluster types. (If you're creating a distributed high-availability or advanced high-availability cluster, select **Next: Data Groups** to move to that tab.)

Deployment Location — Select the location for the database cluster to be deployed into. The first location in the list is in the Kubernetes cluster hosting the EDB Postgres AI service. Where Hybrid Manager (HM) has been configured in a multi-DC environment, you can select different deployment locations for each data group to distribute the database cluster across multiple data centers.

Database Type — Select the type of Postgres you want to use:

1. In the **Postgres Type** field, select the type of Postgres you want to use:

- ** EDB Postgres Advanced Server ** is EDB's Oracle-compatible database offering. View *Migrations* . EDB Postgres Advanced Server is compatible with all the cluster types.
- ** EDB Postgres Extended Server ** is EDB's PostgreSQL-compatible database offering that uses advanced logical replication.
- ** PostgreSQL ** is the open-source, object-relational database management system. PostgreSQL is compatible with single-node and high-availability cluster types.

2. In the **Postgres Version** list, select the version of Postgres that you want to use.
3. HM supports multiple images of each Postgres database, stored in the image library. Each image is a configuration of the database that includes various extensions. Select the image that you want to use for your cluster. See *Asset Library apps require Project Owner or Project Editor role* for more information. Generally, without `-full` at the end of the name, the image has no extensions. With `-full` at the end of the name, the image has all available extensions.

Instance Size — Select the number of CPUs and the amount of memory for your cluster. The number of CPUs and the amount of memory you can select depends on available resources in your Kubernetes cluster.

Storage

You can specify the following storage settings:

Use a separate storage volume for Write-Ahead Logs — By default, HM uses a separate storage volume for database storage and write-ahead log (WAL) files. This default gives best performance and reliability. Clear the check box to store WAL files on the same volume as the database storage.

By default, HM uses a separate storage volume for database storage and WAL files. This default gives best performance and resilience. If you clear this option, WAL files will be stored on the same volume as the database storage.

Database Storage — Under **Database Storage** you can select the database storage class and size of the database storage volume in GiB. The available storage classes depend on what you have configured for your underlying infrastructure, for example, gp2 and gp3 for EKS deployments, or ssd-csi and standard-csi for RHOS deployments, etc.

Write-Ahead Logs Storage — Under **Write-Ahead Logs Storage**, you can select the WAL storage class and size of the WAL storage volume in GiB. The available storage classes depend on what you have configured for your underlying infrastructure, for example, gp2 and gp3 for EKS deployments, or ssd-csi and standard-csi for RHOS deployments, etc.

Networking

For **Connectivity Type**, select whether to use private networking (the default) or public networking.

- **Private** networking allows only IP addresses in your private network to connect to your cluster.
- **Public** networking means that any client can connect to your cluster's public IP address over the internet.

Connections (read-only)

For **High Availability** clusters, you can enable a read-only connection to route read-only operations to standby replicas, reducing the workload on the primary instance and improving overall performance.

> **Note**
>
> Available only for High Availability clusters, not single-node clusters.

Read-Only Workloads — Select the toggle to enable a dedicated read-only endpoint for your cluster.

When enabled, configure the following settings:

Read-Only Workloads Networking — Select whether to use private networking (the default) or public networking for the read-only endpoint.

- **Private** networking allows only IP addresses in your private network to connect via the read-only endpoint.
- **Public** networking means that any client can connect to the read-only endpoint's public IP address over the internet.

If you select **Public**, you can optionally **Use allowlists to restrict network access** and specify the IP ranges that are permitted to connect via the read-only endpoint.

Once enabled, the read-only connection string is available on the **Connect** tab of the cluster details.

Database configuration parameters

When you're creating or editing a single-node or high-availability cluster, use the **DB Configuration** tab to specify database configuration parameters.

The database parameters listed on the **DB Configuration** tab are also referred to as Grand Unified Configuration (GUC) variables. See What Is a GUC Variable? for more information.

The list of parameters is populated based on the type of database you selected on the **Cluster Settings** tab when you created your cluster. For more information about the parameters for your database type:

- For PostgreSQL parameters, see

Setting Parameters and Server Configuration in the

Postgres documentation.

- For EDB Postgres Advanced Server, see

Summary of configuration parameters and Configuration parameters

.

- For more information on the types of values the parameters accept, see

Parameter Names and Values . The types apply to both PostgreSQL and EDB Postgres
Advanced Server parameters.

> **Note**
>
> You can make changes to database or server parameters after cluster creation is complete. Ensure you configure memory- and CPU-related database parameters within the physical limits of the underlying hardware. Changing some parameters requires a restart. Default *wal_segment_size* is set to 64MB for PSR cluster deployment and 128MB for PGD cluster deployment.

Modifying a parameter

1. Find the parameter you want to modify using these methods:

- Search for a parameter using the search field.
- Filter the parameters that are listed by selecting an option in the **Show only** list: - **Custom Values** — Shows only parameters that changed from the default values, either in the current session or modified in a previous session and applied to the cluster. - **Currently Edited Values** — Shows only parameters that changed during the current editing session.

2. Enter the new value in the parameter value field. Your entry can be a literal value or a *Formulas for parameter values* that calculates the value based on other parameters.

> **Warning**
>
> Parameters identified with a yellow exclamation point icon trigger a restart when you save your changes. The restart terminates all open connections and takes a few minutes to complete. It takes a few more seconds for the new connection to establish. During this process, you can't connect to the database.

Backup and security settings

On the **Additional Settings** tab, you can specify the backup retention period and some security features.

Backups

Change the default database backup retention period of 30 days using the **Retention Period** controls in the **Backups** section. You can configure the retention period in days, weeks, or months. Set a retention period between 1 and 180 days, 1 and 25 weeks, or 1 and 6 months.

Backup compression

HM compresses backups by default using the Barman Cloud plugin — no additional configuration is required. Compression reduces storage consumption and can improve backup performance, helping to prevent WAL storage full incidents.

The default compression algorithms are:

- Base backups: Snappy
- WAL archiving: zstd

You can change the compression algorithm for base backups and WAL archiving independently, as each supports a different set of algorithms:

- Base backups: none, Snappy, gzip, bzip2
- WAL archiving: none, Snappy, gzip, bzip2, lz4, xz, zstd

> **Warning**
>
> Disabling WAL compression is strongly discouraged. For PGD clusters (AHA and DHA), which continuously generate WAL files, selecting "none" for WAL compression is particularly inadvisable and may lead to WAL storage full incidents.

Because performance characteristics vary by environment, test the available algorithms against your specific workload to determine which works best before changing from the defaults.

> **Note**
>
> Changing the compression algorithm applies to subsequent backups only. Existing backups aren't affected.

Security

You can encrypt your cluster's data , if required.

This option is available for EDB Postgres Advanced Server and EDB Postgres Extended Server for versions 15 and later.

Use the **Transparent Data Encryption (TDE)** toggle to enable TDE.

> **Note**
>
> You can't enable, disable, or modify the TDE configuration after the cluster is created.

You can set the following options:

- **Enter a Passphrase**. This option secures the data encryption key created by Hybrid Manager (HM) with a passphrase when you create a TDE-enabled cluster.
- If the administrator of your HM instance previously *Adding KMS support* , you can also select **Customer Key Management**. Then select one of the configured keys to secure the data encryption key created by the Hybrid Manager.

To learn more about TDE, see Transparent Data Encryption .

Creating replica clusters

The **Replica Clusters** tab is available when you create or edit a single-node or high-availability cluster. Use it to create and manage *replica clusters* for single-node and high-availability clusters.

A replica cluster is typically used to create read-only copies of the primary database cluster to offload read queries and enhance performance. You can create replica clusters in a different geographic location from the primary cluster to provide disaster recovery and high availability.

- A **replica cluster** is a separate, independent database cluster that replicates from a primary cluster. It's suited for disaster recovery or serving traffic from a different geographic location.
- A **read-only connection** is a dedicated connection endpoint within the same cluster that routes queries to standby replicas, enabling read workload distribution without creating a separate cluster.

To configure a read-only connection for your HA cluster instead, see *Cluster setup* .

1. With **Single Node** or **High Availability** selected on the **Cluster Info** tab, after assigning a name and password to the cluster, select the **Replica Clusters** tab.
2. Select the *Node settings* tab and fill out the required information.
3. Select the *Database parameter configuration* tab and modify parameters as needed.
4. Select the *Backup and security settings* tab and modify parameters settings as needed.
5. Select **Add Replica Cluster**.

The new replica cluster is listed in the **Replica Clusters** tab. Expand the replica cluster to configure the replica cluster settings. You can create replica clusters in a different geographic location from the primary cluster to provide disaster recovery and high availability.

Create as many replica clusters as needed.

Managing replica clusters

You can edit and delete replica clusters:

- Select **Edit** to make adjustments to the replica cluster.
- Select **Delete** to delete a replica cluster.
- To add a replica cluster, select **Add Replica Cluster**.

Creating data groups

The **Data Groups** tab is available when you create or edit a distributed high-availability or advanced high availability cluster. Use it to create and manage *data groups* for advanced and distributed high-availability clusters. A data group is typically a regionally close group of data nodes that are configured to replicate data within the group and respond to application queries. A distributed high-availability cluster can be made of one or more data groups that also replicate to each other, enabling geodistributed clusters.

Creating data groups

1. With **Advanced High Availability** or **Distributed High Availability** selected on the **Cluster Info** tab, select the **Data Groups** tab.
2. Select the ***Node Settings*** tab and fill out the required information.
3. Select the ***DB Configuration*** tab and modify parameters as needed.
4. Select **Create Data Group**.

The new data group is listed in the **Data Groups** tab. Expand the cluster architecture diagram to see a summary of the data group.

To add another data group, select **Add Data Group**.

> **Note**
>
> If you create two data groups, a witness group is automatically added. You can edit the witness group as you can the data groups, though the options are more limited. You can set the name of the group, deployment location, database type, version, image and instance size.

Node settings

Use the **Node Settings** tab to set up the data group.

Nodes — Select a configuration of nodes in the group.

You can select the recommended three-data-node configuration or a two-data-node configuration. For advanced high-availability clusters, you can create a one-node configuration. For more information on the node architectures see, *Distribution high availability* .

Deployment Location — Select the location for the data group to be deployed into. The first location in the list is in the Kubernetes cluster hosting the EDB Postgres AI service. Where Hybrid Manager (HM) has been configured in a

multi-DC environment, you can select different deployment locations for each data and witness group to distribute the database cluster across multiple data centers.

Database Type — Select the type of Postgres you want to use:

1. In the **Postgres Type** field, select the type of Postgres you want to use:

- EDB Postgres Advanced Server is EDB's Oracle-compatible database offering. EDB Postgres Advanced Server is compatible with all the cluster types.
- EDB Postgres Extended Server is EDB's PostgreSQL-compatible database offering that uses advanced logical replication.
- PostgreSQL is the open-source, object-relational database management system. PostgreSQL is compatible with single-node and high-availability cluster types.

2. In the **Postgres Version** list, select the version of Postgres that you want to use.
3. HM supports multiple images of each Postgres database, stored in the image library. Each image is a configuration of the database that includes various extensions. Select the image that you want to use for your cluster. See *Asset Library apps require Project Owner or Project Editor role* for more information. Generally, images without `-full` at the end of the name have no extensions. With `-full` at the end of the name, the image has all available extensions except PostGIS.

Instance Size — Select the number of CPUs and the amount of memory for your cluster. The number of CPUs and the amount of memory you can select depends on available resources in your Kubernetes cluster.

Storage

You can specify the following storage settings:

Use a separate storage volume for Write-Ahead Logs — By default, HM uses a separate storage volume for database storage and write-ahead log (WAL) files. This default gives best performance and reliability. Clear the check box to store WAL files on the same volume as the database storage.

Database Storage — Under **Database Storage** you can select the database storage class and size of the database storage volume in GiB. The available storage classes depend on what you have configured for your underlying infrastructure, for example, gp2 and gp3 for EKS deployments, or ssd-csi and standard-csi for RHOS deployments, etc.

Write-Ahead Logs Storage — Under **Write-Ahead Logs Storage**, you can select the WAL storage class and size of the WAL storage volume in GiB. The available storage classes depend on what you have configured for your underlying infrastructure, for example, gp2 and gp3 for EKS deployments, or ssd-csi and standard-csi for RHOS deployments, etc.

Networking

For **Connectivity Type**, select whether to use private networking (the default) or public networking.

- **Private** networking allows only IP addresses in your private network to connect to your cluster.
- **Public** networking means that any client can connect to your cluster's public IP address over the internet.

Connections (read-only)

For **Advanced High Availability** and **Distributed High Availability** clusters, you can enable a read-only connection per data group to route read-only operations to nodes that aren't currently designated as the write leader, reducing load on the write leader and improving overall performance.

Read-Only Workloads — Select the toggle to enable a dedicated read-only endpoint for your cluster.

When enabled, configure the following settings:

Read-Only Workloads Networking — Select whether to use private networking (the default) or public networking for the read-only endpoint.

- **Private** networking allows only IP addresses in your private network to connect via the read-only endpoint.
- **Public** networking means that any client can connect to the read-only endpoint's public IP address over the internet.

If you select **Public**, you can optionally **Use allowlists to restrict network access** and specify the IP ranges that are permitted to connect via the read-only endpoint.

Once enabled, the read-only connection string for each data group is available on the **Connect** tab of the cluster details.

Backups

Change the default database backup retention period of 30 days using the **Retention Period** controls in the **Backups** section. You can configure the retention period in days, weeks, or months. Set a retention period between 1 and 180 days, 1 and 25 weeks, or 1 and 6 months.

Security

This section appears after you select **EDB Postgres Advanced Server** or **EDB Postgres Extended Server** as the database type and select the version and image. Changes here are applied to all data groups.

Enable **Transparent Data Encryption (TDE)** to use your own encryption key. This option is available for EDB Postgres Advanced Server and EDB Postgres Extended Server for version 15 and later.

Enable **TDE** by selecting the **Enable TDE** check box.

To learn more about TDE support, see Transparent Data Encryption .

Database parameter configuration

Select the **DB Configuration tab** to configure the parameters for the database.

In the **Parameters** section, you can update the value of the database configuration parameters as needed.

The database parameters listed on the **DB Configuration** tab are also referred to as Grand Unified Configuration (GUC) variables. See What Is a GUC Variable? for more information.

The list of parameters is populated based on the type of database you selected on the **Cluster Settings** tab when you created your cluster. For more information about the parameters for your database type:

- For PostgreSQL parameters, see

Setting Parameters and Server Configuration in the

Postgres documentation.

- For EDB Postgres Advanced Server, see

Summary of configuration parameters and Configuration parameters

.

- For more information on the types of values the parameters accept, see

Parameter Names and Values . The types apply to both PostgreSQL and EDB Postgres
Advanced Server parameters.

> **Note**
>
> You can make changes to database or server parameters after cluster creation is complete. Changing some parameters requires a restart.

Modifying a parameter

1. Find the parameter you want to modify using these methods:

- Search for a specific parameter using the search field.
- Filter the parameters that are listed by selecting an option in the **Show only** list: - **Custom Values** — Shows only parameters that changed from the default values, either in the current session or modified in a previous session and applied to the cluster. - **Currently Edited Values** — Shows only parameters that changed during the current editing session.

2. Enter the new value in the parameter value field. Your entry can be a literal value or a *Formulas for parameter values* that calculates the value based on other parameters.

> **Warning**
>
> Parameters identified with a yellow exclamation point icon trigger a restart when you save your changes. The restart terminates all open connections and takes a few minutes to complete. It takes a few more seconds for the new connection to establish. During this process, you can't connect to the database.

Managing data groups

You can duplicate, edit, and delete data groups:

- To duplicate a group, select **Duplicate**. Then edit the new group to differentiate it from the original. In multi-DC environments, you can select a different deployment location for the duplicated data group.
- Select **Edit** to make adjustments to the data group.
- Select **Delete** to delete a data group. You're prompted to confirm the deletion.
- To add a data group, select **Add Data Group**.

Creating a cluster from a template

You can create a database cluster using a template, if your project has any. Using a template saves time and ensures consistency across environments.

To create a template, see *Create cluster templates* .

Starting points

You can start to create a cluster from a template from the **Choose Your Path** page or from the **Cluster Templates** page.

From the Choose Your Path page

You can get to the **Choose Your Path** page from the **Estate** page or from a project page.

From the **Estate** page:

1. Select **Quick Actions > Create New Database Cluster**.

If you have more than one project, you're prompted to select a project.

2. If prompted, select the project you want to create the database in.

After you select the project, or if you have only one project, the **Choose Your Path** page opens.

From a project page, you can use either of these approaches. To get to a project page, from the **Projects** page, select the project.

- In the project page, select **Create New > Database Cluster**.
- In the project page, select **Clusters > Postgres Cluster**. On the **Clusters** page, select **Create New > Database Cluster**.

The **Choose Your Path** page opens.

Using a template from the Choose Your Path page

1. On the **Choose Your Path** page, select **Use Template**.
2. Select a template. From the list of available templates, select the one that suits your needs.
3. Select **Use Template**.

You enter cluster creation mode. See *Customize and create the cluster* .

Start from the Cluster Templates page

1. From your project page, in the left navigation, select **Cluster > Cluster Templates**.
2. Next to the template you want to use as the basis of your cluster, select the **Use** icon.

You enter cluster creation mode. See *Customize and create the cluster* .

Customize and create the cluster

After selecting a template, you can review and modify the settings of the cluster you're creating. For details, see:

- *Cluster types to create*
- *Cluster setup*
- *Database configuration parameters*
- *Backup and security settings*
- *Creating replica clusters* (for single-node and high-availability clusters)
- *Creating data groups* (for advanced or distributed high-availability clusters)

You can also select a different template to work with. Select the **Cluster Templates** tab to see a list of available templates.

After you finish configuring the cluster using the tabs, select **Create Cluster**. Tabs with incorrect or missing settings show a red indicator. Correct any errors and select **Create Cluster** again.

After you select **Create Cluster** with valid settings, the cluster begins provisioning, and you return to the list of clusters. A popup confirms the creation of the cluster, and the new cluster is added to the list of clusters. The cluster is created in the background. You can see the progress in the list of clusters.

If you try to view the cluster details while the cluster creation is in progress, you can see only the overview, backup, logs, and properties.

Creating a cluster programmatically

By automating Postgres cluster creation you can streamline reproducible builds.

Use Bash scripts and the Hybrid Manager API to automate Postgres cluster creation for reproducible builds. Check all configurations into source control to track changes and maintain documentation.

Create a machine user

1. Go to the dropdown menu in the upper-right corner of the HM console.
2. Select **User management** from the dropdown menu.
3. Select **Add new user** in the upper-right corner.
4. In the pop-up window that shows **Add new user**, enter the name and tick off the checkbox **Create access key**.
5. Fill in the field **Expires in**.
6. In the pop-up window, select **Add user**.
7. Copy the access key from the pop-up window.
8. Select **Key stored safely**.

Give the machine user permissions

> **Note**
>
> To set the machine user permissions correctly, select the target project you want to create the Postgres clusters in from the left navigation bar.

2. In the left navigation bar, select **Users**.
3. In the search bar, look for the machine user you created.
4. Select the pencil (edit) icon on the right-hand side of **Machine user**.
5. Select **Assign Roles**.
6. Under **Assign project roles**, tick off the checkbox **Project owner**.
7. Select **Submit**.

Create an env.sh file

You can simplify wiriting shell scripts by adding the project ID to the `env.sh` file and the API host.

Add the copied access key in a file named `env.sh` :

```
export HCP_ACCESS_KEY="baak_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
export HCP_API_ACCESS_KEY=${HCP_ACCESS_KEY}
export HCP_PROJECT_ID=prj_xxxxxxxxxxxxxx
export HCP_API_HOST="https://portal-uat-eks.aws.hcp.enterprisedb.network"
```

Locate a Docker image with AIDB and PGFS

To get a list of locations and the Docker images available at the locations, create and run the script called `01-get-locations.sh` . You'll see the following content:

```
# !/bin/bash

set -e
set -u
set -o pipefail

#  list locations

source env.sh

curl \
  --insecure \
  --header "Content-Type: Application/JSON" \
  --header "x-access-key: ${HCP_API_ACCESS_KEY}" \
  --request GET \
  ${HCP_API_HOST}/api/v1/projects/${HCP_PROJECT_ID}/locations \
  | jq
```

Run the following script:

```
./01-get-locations.sh > locations.json
```

Examine `locations.json` and look for any image listed under `pgExtensions` , `aidb` and `pgfs` .

Create a Postgres cluster

Create a file named `create-psr-cluster.json` which includes the image names you have retrieved from the previous step:

```
{
  "projectId": "${HCP_PROJECT_ID}",
  "psr": {
    "clusterName": "small-ai-api-001",
    "password": "111111111111",
    "locationId": "managed-devspatcher",
    "clusterData": {
      "instances": 1,
      "resourceRequest": {
        "memory": "2Gi",
        "cpu": "2"
      },
      "storageConfiguration": {
        "primaryStorage": {
          "size": "4",
          "storageClass": "hcp-aws-gp3"
        },
        "walStorage": {
          "size": "4",
          "storageClass": "hcp-aws-gp3"
```

(continues on next page)

(continued from previous page)

```
        }
      },
      "image": {
        "url": "upmdev.azurecr.io/postgresql:17.6-2510160831",
        "digest":
↪"sha256:6bda61e6d75c3b9d6a886c547d789c46b88ed807ae28d8d06fa1510a0cc5afae"
      },
      "networkAccessType": "NetworkAccessTypePublic"
    }
  }
}
```

Write a shell script called `02-create-psr-cluster.sh` to use `create-psr-cluster.json` and create a cluster:

```
# !/bin/bash

set -e
set -o pipefail
set -u

source env.sh

cat create-psr-cluster.json | envsubst > /tmp/create-psr-cluster.json

curl \
  --insecure \
  --header "Content-Type: Application/JSON" \
  --header "x-access-key: ${HCP_API_ACCESS_KEY}" \
  --data-binary "@/tmp/create-psr-cluster.json" \
  --request POST \
  ${HCP_API_HOST}/api/v1/projects/${HCP_PROJECT_ID}/clusters
```

Run the shell script:

```
./02-create-psr-cluster.sh | tee ./psr-cluster.json
```

You can check the progress of the cluster creation in the HM console.

Create the AIDB and PGFS extensions in the new cluster

You can add the functionality of AIDB and PGFS to your programmatically created clusters as extensions.

Configure env.sh to include PostgreSQL authentication credentials

Add the following to your `env.sh` file. For example:

```
export PGPASSWORD=your_postgresql_password
export PGUSER=edb_admin
export PGHOST=your-postgresql-hostname
export PGPORT=5432
export PGDATABASE=edb_admin
export PGSSLMODE=require
```

Create cluster extensions

Create a bash file called `03-create-ai-extensions.sh` with the following content:

```
# !/bin/bash

set -e
set -o pipefail
set -u

source env.sh

psql -X <<EOF
create extension aidb cascade;
create extension pgfs cascade;
EOF
```

Run the script:

```
$ chmod +x 03-create-ai-extensions.sh
$ ./03-create-ai-extensions.sh
```

Cluster verification

After verifying the cluster creation via the HM console, you can get your new cluster via the API to verify its settings. Getting clusters in JSON format also helps specifying the required JSON fields and can serve as a blueprint for automated cluster creation, avoiding to look up each setting field individually.

> **Note**
>
> Create-cluster JSON and Get-cluster JSON are two different functionalities that can contain different information.

```
# !/bin/bash

set -e
set -u
set -o pipefail

source env.sh

HCP_PG_CLUSTER_ID=p-fj3ki6e0x0

curl \
  --insecure \
  --header "Content-Type: Application/JSON" \
  --header "x-access-key: ${HCP_API_ACCESS_KEY}" \
  --request GET \
  ${HCP_API_HOST}/api/v1/projects/${HCP_PROJECT_ID}/clusters/${HCP_PG_CLUSTER_ID} \
  | jq
```

Where `HCP_PG_CLUSTER_ID` specifies your cluster ID.

Save the content of the cluster into a JSON file:

```
./get-existing-cluster.sh > existing-cluster.json
```

Note

Check the *Using the HM API* for more details on the required fields, the naming convention, and JSON structure for cluster creation. !!!

Managing clusters

Several management tasks are associated with clusters in Hybrid Manager (HM).

Cluster creation tasks

As part of the cluster creation process, you might want to:

- *Create a cluster template* that you can use as the basis for new templates.
- *Create a formula* to use as a value for certain configuration parameters.

Working with existing clusters

After you create a cluster , you can use the three-dot menu to the right of the cluster name to:

- *Edit the cluster* to make changes to most of its configuration.
- Change a cluster password.
- *Delete a cluster* .
- Upgrade a cluster. You can perform a minor upgrade or a major upgrade.
- *Restore a deleted cluster* .

The menu includes only the options available to the current project user.

Analyze how your Postgres cluster's running

While your cluster is running, you may need to look at the status of the Kubernetes infrastructure your Postgres clusters are running on. You can do this by *tracing the cluster back to its Kubernetes resources* . For example, you can use Grafana to better understand how resources are being used.

Create a database cluster template

You can create a cluster template to use as the basis for future clusters. Creating a template is similar to creating a cluster. You need to specify all the same information.

Starting point

The starting point is the landing page of the project you want to create the template in. To get there, from the **Projects** page, select the project.

In the project page, in the left navigation, select **Clusters > Cluster Templates**.

The **Cluster Templates** page lists any templates in the project.

Select **Create Template > Create Database Cluster Template**.

From this point, specifying the cluster configuration is the same as it is for creating a cluster. You can step through the tabs in order using **Next**, or you can select the tab you want to work on. For details see:

- *Cluster types to create*
- *Cluster setup*
- *Database configuration parameters*
- *Backup and security settings*
- *Creating data groups* (for advanced or distributed high-availability clusters)

When you've finished configuring your cluster template, select **Create Template**.

The new template is added to the list of templates.

Deleting templates

You can delete a template from the **Cluster Templates** page if you no longer need it.

> **Warning**
>
> You can't recover or restore a deleted template.

1. On the page for the project whose template you want to delete, in the left navigation, select **Clusters > Cluster Templates**.
2. Next to the template you want to delete, select the **Delete** icon.
3. Type `delete template` in the confirmation box and select **Delete**.

For information on using the template, see *Creating a cluster from a template*

Deleting clusters

You can delete the cluster from the **Postgres Clusters** page and from the detailed view of the cluster.

- From the project page, select **Clusters > Postgres Clusters**. From the three-dot menu next to the cluster you want to delete, select **Delete Cluster**.
- From the project page, select **Clusters > Postgres Clusters**. Select the cluster you want to delete. From the detailed view of the cluster, select **Quick Actions > Delete Cluster**.

When you're prompted to confirm the deletion, select **Delete**.

Editing a cluster

You can start editing the cluster from the **Postgres Clusters** page or from the detailed view of the cluster:

- On the **Postgres Clusters** page, next to the cluster you want to edit, select **Edit**.
- To get to the detailed view of the cluster, on the **Postgres Clusters** page, select the cluster name. On the **Cluster Details** page, select **Quick Actions > Edit Cluster**.

> **Note**
>
> You can make changes to database or server parameters after cluster creation is complete. Changing some parameters requires a restart.

Editing a cluster

The **Edit Cluster** page is similar to the **Create Cluster** page. Select the tab you want to work on, or step through the tabs in order using **Next**:

- ***Cluster Info***
- ***Cluster Settings***
- ***DB Configuration*** (optional)
- ***Additional Settings*** (optional)
- ***Data Groups*** (for distributed high-availability clusters)

> **Note**
>
> When editing a cluster, you can't switch between a single-node or high-availability cluster and an advanced or distributed high-availability cluster

Saving and confirming edits

When you have completed your changes to the cluster configuration, select **Save**.

If the configuration is valid, a list of the changes to be made to the cluster appears. A warning might also appear saying the changes may trigger a server restart. Select **Confirm Changes** to apply the changes to the cluster and return to the page you started from.

> **Note**
>
> Selecting the **Back** arrow at the top of the page abandons your changes and returns to the page you started from.

Editing a cluster programmatically

Use the Hybrid Manager API to update cluster configuration programmatically, enabling repeatable and auditable changes to your Postgres clusters. The API uses a partial update (PATCH) model, so only fields explicitly included in the request body are changed.

Before you begin, configure a machine user and `env.sh` file. See *Creating a cluster programmatically* for the initial setup steps.

Add your cluster ID to `env.sh` :

```
export HCP_PG_CLUSTER_ID=p-fj3ki6e0x0
```

Retrieve current cluster configuration

Before patching a cluster, retrieve its current state to confirm existing field values and identifiers such as `replicaPgId` and `groupId` . These are required when managing *Managing PSR replica clusters* and *Managing PGD and AHA data groups* .

Create and run a script called `get-cluster.sh` :

```
# !/bin/bash
```

(continues on next page)

(continued from previous page)

```
set -e
set -o pipefail
set -u

source env.sh

curl \
  --insecure \
  --header "Content-Type: application/json" \
  --header "x-access-key: ${HCP_API_ACCESS_KEY}" \
  --request GET \
  ${HCP_API_HOST}/api/v1/projects/${HCP_PROJECT_ID}/clusters/${HCP_PG_CLUSTER_ID} \
  | jq
```

Patch a PSR cluster

To update a PSR cluster, create a JSON file containing only the fields you want to change. The following example scales the cluster to three instances and updates its resource allocation.

Create a file named `patch-psr-cluster.json`:

```
{
  "psr": {
    "instances": 3,
    "resourceRequest": {
      "memory": "8Gi",
      "cpu": "4"
    }
  }
}
```

Create a script called `patch-cluster.sh`:

```
# !/bin/bash

set -e
set -o pipefail
set -u

source env.sh

curl \
  --insecure \
  --header "Content-Type: application/json" \
  --header "x-access-key: ${HCP_API_ACCESS_KEY}" \
  --data-binary "@${1}" \
  --request PATCH \
  ${HCP_API_HOST}/api/v1/projects/${HCP_PROJECT_ID}/clusters/${HCP_PG_CLUSTER_ID}
```

Run the script, passing the JSON file as an argument:

```
./patch-cluster.sh patch-psr-cluster.json
```

A successful request returns HTTP 200 with an empty body. You can verify the changes in the Hybrid Manager console or by fetching the cluster with a GET request.

Managing PSR replica clusters

PSR clusters support *read replica clusters* deployed in remote locations. When you include the `replicaClusters` field in a PSR patch request, the API applies a declarative list model. The list you provide becomes the complete and authoritative set of replica clusters for that PSR cluster.

The following example makes three changes in a single patch request:

- Retains an existing replica cluster in `managed-us-east-1` with no changes to its configuration.
- Edits an existing replica cluster in `managed-eu-west-1` , scaling it to two instances.
- Adds a new replica cluster in `managed-ap-southeast-1` .

Create a file named `patch-psr-replicas.json` . Replace the `replicaPgId` values with those from your cluster:

```
{
  "psr": {
    "replicaClusters": [
      {
        "replicaPgId": "p-fj3ki6e0x0-b"
      },
      {
        "replicaPgId": "p-fj3ki6e0x0-c",
        "locationId": "managed-eu-west-1",
        "instances": 2
      },
      {
        "locationId": "managed-ap-southeast-1",
        "instances": 1,
        "resourceRequest": {
          "memory": "4Gi",
          "cpu": "2"
        },
        "storageConfiguration": {
          "primaryStorage": {
            "size": "10",
            "storageClass": "hcp-aws-gp3"
          }
        },
        "image": {
          "url": "your-registry.example.com/postgresql:17.6-xxxxxxxxxx",
          "digest":
↪"sha256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
        },
        "backupRetentionPeriod": "30d",
        "backupSchedule": "0 0 ",
        "networkAccessType": "NetworkAccessTypePrivate"
      }
    ]
  }
}
```

Run `patch-cluster.sh` from *Patch a PSR cluster* , passing the replicas file as the argument:

```
./patch-cluster.sh patch-psr-replicas.json
```

Note

When retaining an existing replica with no configuration changes, supplying only its *replicaPgId* is sufficient. All other fields remain unchanged.

Distributed High Availability (PGD) and Advanced High Availability (AHA) clusters use *EDB Postgres Distributed (PGD) cluster with 2 data groups and 1 witness group not healthy* to organize nodes across locations. When you include the `groups` field in a PGD or AHA patch request, the API applies the same declarative list model used for PSR replica clusters.

The following example patches a PGD cluster. It retains one existing data group with an updated instance count, adds a new data group in a second location, and removes any unreferenced group.

Create a file named `patch-pgd-groups.json` . Replace the `groupId` value with the one from your cluster:

```
{
  "pgd": {
    "groups": [
      {
        "groupId": "p-sdl5klb2hu2-a",
        "locationId": "managed-us-east-1",
        "groupName": "us-east-data-group",
        "dataGroup": {
          "dataInstances": 3,
          "resourceRequest": {
            "memory": "8Gi",
            "cpu": "4"
          }
        }
      },
      {
        "locationId": "managed-eu-west-1",
        "groupName": "eu-west-data-group",
        "dataGroup": {
          "dataInstances": 2,
          "resourceRequest": {
            "memory": "4Gi",
            "cpu": "2"
          },
          "storageConfiguration": {
            "primaryStorage": {
              "size": "20",
              "storageClass": "hcp-aws-gp3"
            }
          },
          "image": {
            "url": "your-registry.example.com/postgresql:17.6-xxxxxxxxxx",
            "digest":
↪"sha256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
          },
```

(continues on next page)

(continued from previous page)

```
          "backupRetentionPeriod": "30d",
          "backupSchedule": "0 0 ",
          "networkAccessType": "NetworkAccessTypePrivate"
        }
      }
    ]
  }
}
```

Run `patch-cluster.sh` from *Patch a PSR cluster* , passing the groups file as the argument:

```
./patch-cluster.sh patch-pgd-groups.json
```

- When retaining an existing PGD data group with no configuration changes, supplying only its `groupId` is sufficient. All other fields remain unchanged.
- If a PGD cluster has exactly two data groups, a witness group is automatically managed by the platform to maintain Raft quorum. See *EDB Postgres Distributed (PGD) cluster with 2 data groups and 1 witness group not healthy* for more information. !!!

Restoring clusters

You can restore a deleted cluster from a backup. The backup must be in the same project as the cluster you're restoring. The restore process technically creates a new cluster with the same configuration as the deleted cluster. Because it's actually a new cluster, you need to give it a new name and a new password.

The restored cluster is created in the same project as the deleted cluster. You can restore a deleted cluster for as long as the backup is available. The backup remains available for the retention period set for each cluster. After the retention period passes, the backup is deleted.

Restoring a cluster from a backup

1. Select **Clusters > Postgres Clusters**.
2. Select the **Deleted** tab.
3. Next to the deleted cluster you want to restore, select the **Restore** icon.
4. In the **Restore Cluster** dialog box, select the backup you want to restore and select **Restore**.

The **Restore Cluster** page opens. This page is similar to the **Create Cluster** page. It displays the cluster name, cluster type, and the backup you selected to restore. You can change the cluster name and cluster type if you want.

5. Enter a new password.
6. On the **Cluster Settings** tab, you can optionally change cluster settings. Under **Restore Type** select the type of restore you want to perform:

- **Fully Restore** — Restore the entire cluster from the last full backup of the database. After you select this option, you can select the backup you want to restore from the list below it.
- **Point In Time Restore** — Restore the cluster to a specific point in time. Select a date and time within the retention period of the backup. The range appears under the date and time picker.

7. Use the other tabs as needed. For details, see *Cluster creation settings* .
8. Select **Restore Cluster**.

The **Postgres Clusters** page opens, and you can see the progress of the restore operation. The restored cluster will be in the same project as the original cluster. It has the same configuration as that cluster but a new name and password.

Tracing clusters back to their Kubernetes resources

While managing your Postgres clusters on Hybrid Manager (HM), you may need to look at the status of the Kubernetes infrastructure your Postgres clusters are running on. These are the Kubernetes worker nodes. For example, you may want to see if a node is failing because you're seeing issues in the HM interface or you're running out of resources.

There are several ways to get information useful for tracing clusters back to their Kubernetes resources:

- From the *Using HM to collate the cluster to its namespace*
- From the *Grafana Hardware Utilization dashboard*
- Using *Loki and Fluent Bit*
- If you're an administrator, using *Using kubectl to check underlying cluster resources*

If you're tracing back a distributed high-availability cluster, you need to be aware of some differences from tracing high-availability or single-node clusters. See *Tracing back distributed high-availability clusters*

Using HM to collate the cluster to its namespace

When tracing back a Postgres cluster created using HM to its underlying Kubernetes resources, the first place to start is the cluster's **Overview** page in HM. (To get to that page, on the **Estate** page, select the cluster.) There, find the Kubernetes namespace associated with the cluster by looking at the end of the url in your browser:

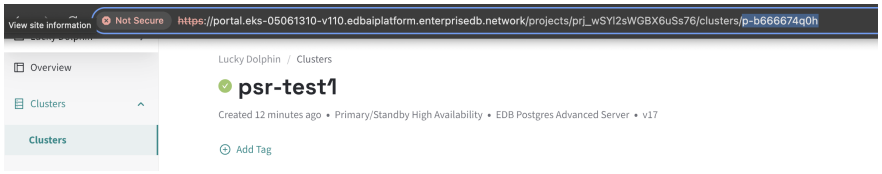


Fig. 3: Browse cluster

In this example, cluster psr-test1 is associated with namespace `p-b666674Q0h`. To further trace the cluster back to its resources, its associated namespace is an essential first piece of the puzzle. Using it, you can find even more.

After you have the namespace for the cluster you're exploring, the next stop is the built-in Grafana dashboards.

Grafana Hardware Utilization dashboard

To access the Grafana dashboards, use the HM launchpad. From the profile menu, select **Launchpad**. From the launchpad, on the **Grafana** tile, select **View Service**.

In the landing page in Grafana, in the **Search** field, enter `Hardware Utilization`. The **Hardware Utilization** dashboard lets you see underlying Kubernetes worker nodes and the Kubernetes pods with Postgres workloads running on top of some of them.

Pods

In the **Postgres Pod Status** section, use the namespace associated with the Postgres cluster you're looking into to identify the pods whose name starts with the same designation as the namespace your Postgres cluster is associated with. These pod names are suffixed with `-1`, `-2`, `-3`, and so on, as the workload is split among pods. From the Postgres Nodes table, you can also see the state of the pod in the State column and the node the pod is running on in the Node column.

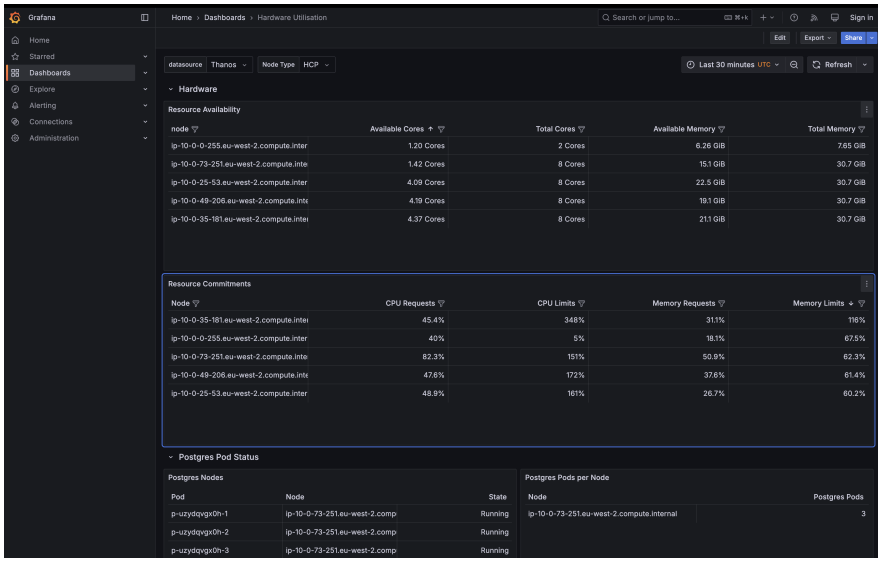


Fig. 4: Hardware Utilization dashboard

Resource availability

After you have collated your pods with the underlying Kubernetes worker nodes, you can then use the node names under **Resource Availability** to get more information about the resources available for those nodes:

- **Available Cores**: The number of cores left available after the workloads have used what they need.
- **Total Cores**: The total number of cores the node needs before any utilization from a workload or workloads. (There can be more than one pod on a worker node.)
- **Available Memory**: The amount of memory available after the workloads have utilized what they need.
- **Total Memory:** The total memory the node has without any workloads using any.

Resource commitments

When you have a Kubernetes worker node that you want to look into the Postgres workload on top, another helpful section of the dashboard is the **Resource Commitments** table. Here you can see what sort of demand is being exacted on each node, including the nodes that underlie the pods you're interested in:

- **CPU Requests**: The amount of CPU of the node being requested by workloads provisioned or being provisioned by the HM.
- **CPU Limits**: The sum total of CPU limits from pods on the node.
- **Memory Requests**: The amount of memory from the node being requested by workloads provisioned or being provisioned by the HM.
- **Memory Limits**: The sum total of memory limits from pods on the node.

Requests

Both CPU and memory requests are guaranteed-met (guaranteed given the node isn't over-provisioned) demands on the node by the pods running on top of it. This doesn't mean that each pods has to demand 25% at any one time. One or more could be idle, for example, requiring much less CPU at that time.

However, it does mean that each of the pods could potentially use 25% of the CPU resource at one time. Suppose pods 1, 2, and 3 each have CPU requests on Node A and each of the three pods have a 25% CPU request on the node. Given

that no other pods are making CPU demands on Node A, the total request demand on the node is 75%, which means that the node can handle the CPU load.

If other pods from other clusters were also making demands on Node A, for example, another pod had a 30% request on the node, then the node would be over-provisioned at 105%.

Limits

Both CPU and memory limits are the upper bounds on CPU and memory of a node's resources that a pod can use. Suppose the limits for pods 1, 2, and 3 are 50% CPU each, all on Node A. This means that Node A's CPU limit is 150%. It can't provide more than 100% CPU at one time, so if all three pods burst to 50%, they're throttled as a result. If pod 1 bursts to 50% CPU limit and the other two are using only 25% each under normal operation, then pod 1 gets the 50% it needs.

In the case of memory, consider a similar scenario where pods 1, 2, and 3 are each at 50% memory on Node A. Suppose all three try to burst to the 50% threshold. A container in a pod may be killed by the Linux kernel.

Troubleshooting potential resource issues using the Hardware Utilization dashboard

Suppose you have a Postgres cluster you requested to provision and it seems to be stuck for 15 minutes or more at 85% complete. From the HM console, you can find the namespace associated with the cluster's name using the last bit of the url in the web browser.

Suppose in this case you're looking into namespace `p-b666674Q0h` . Using the **Hardware Utilization** dashboard, you can first find the pods associated with that namespace—suppose again `p-b666674Q0h-1` , `p-b666674Q0h-2` , and `p-b666674Q0h-3` —and check the statuses of these pods. It's likely that one or more pods are in a `Pending` state. For example, `p-b666674Q0h-3` may still be pending, while `p-b666674Q0h-1` and `p-b666674Q0h-2` are running. You can then identify the node that `p-b666674Q0h-3` is running on from the Postgres Nodes table under **Postgres Pod Status**.

Suppose `p-b666674Q0h-3` is running on node `ip-10-0-73-251.eu-west-2.compute.internal` . When you check `ip-10-0-73-251.eu-west-2.compute.internal` in the **Resource Availability** table, you notice that 0.5 cores are available out of a total of 8 cores and 0.8Gi of memory available out of a total of 20.3. You can conclude from this that there aren't lot of resources left on the node for another pod.

Looking in `ip-10-0-73-251.eu-west-2.compute.internal` on the **Resource Commitments** table, suppose you notice that the node has 129% CPU requests and 106% memory requests. Given that the requests are above 100%, you can conclude that the node is being over-provisioned. This explains why your pod `p-b666674Q0h-3` is stuck in `Pending` state. Without more resources, this Postgres cluster can't complete. You can use the HM to edit the cluster you're trying to provision to need less CPU and memory, and the provisioning process continues. If making the cluster smaller doesn't meet your needs, you must allocate more database worker nodes to the Kubernetes cluster manually or up the max nodes for your autoscaler so it knows to provision more.

Loki and Fluent Bit

You can find deeper information about a pod and its status using Loki on the Fluent Bit service logs. To find this view:

1. From the navigation panel in Grafana, select **Explore**.
2. From the menu next to **Outline**, select **Loki**.
3. Use the **Label filters** menu to select **service_name** on the left and **fluent-bit** on the right.
4. Run the search query on Loki's Fluent Bit logs for pods associated with that namespace. Copy and paste the namespace associated with the cluster you're interested in analyzing further into the **Line contains** field and select **Run Query**.

You can see how long logs that contain the namespace you searched for have been collected for. Below that you can see the actual logs.

Turn on **Pretty JSON** to see each instance of the searched-for namespace highlighted throughout the logs.

You may see error messages associated with the namespace you're searching for throughout, like `Can't find target pod` associated with the pod you're searching for. Or, under a `reason: FailedScheduling`, you might see a message like:

```
"0/43 nodes are available: 1 node(s) had untolerated taint (CriticalAddonsOnly: true),
↪ 3 node(s) had untolerated taint (edbaiplatform.io/control-plane: true), 39␣
↪Insufficeint cpu, 4 Insuffcient memory. preemtion: 0/43 nodes are available: 39 No␣
↪preemption vitims found for upcoming pod, Preemption is not helpful for scheduling."
```

This message tells the exact reason why the pod is failing to complete: insufficient CPU and memory.

Tracing back distributed high-availability clusters

Tracing distributed high-availability (DHA) clusters is slightly different from tracing back high-availability or single-node clusters. That's because DHA clusters are split into one or more data groups, and each of these data groups has two to three nodes per group.

For example, suppose you have a DHA cluster with two data groups, A and B, each with three nodes. If you go to the **Monitoring** tab of the cluster's main page, then go to the **Monitoring** section of that tab, you can change the second value from the default **CLuster level** to **Node level**. After changing the value to **Node level**, you can expand the final menu to the right to show each of the six nodes of the cluster: Data Group A's 1, 2, and 3, as well as Data group B's 1, 2, and 3.

To get back to the Kubernetes level, go to the *Grafana Hardware Utilization dashboard* in Grafana.

Next, look at the **Postgres Nodes** section near the bottom-left of the **Hardware Utilization** dashboard. You can find the six nodes you just saw in the cluster's **Monitoring** tab represented by a pod for each, in the format `<id-of-cluster/namespace>-a-1-1`, `<<id-of-cluster/namespace>-a-2-1>`, and so on. The pod's designation calls out the data group the pod belongs to (A or B) and which node of the pod it is: 1, 2, or 3.

After you have the pod designations, you can see both the node it's attempting to run on and the state of the pod. With the node id, you can then use that to see both the resource availability and resource commitments of that node, in the ***Resource Availability*** and ***Resource Commitments*** sections of the dashboard respectively. You can see how many cores the node has left, how much memory, and so on.

As with single-node or HA clusters, you can troubleshoot some resource issues from here. Or, you can go into **Explore** and use *Loki and Fluent Bit* to dive down to the log level of the pods you're interested in.

Using kubectl to check underlying cluster resources

If you're an administrator with Kubernetes access, you can use kubectl to analyze the Kubernetes worker nodes underlying your Postgres clusters.

To get started, when connected to HM with kubectl, first use the `kubectl get clusterwrappers` command. This command collates the name of the Postgres cluster you're tracing back to the namespace the cluster belongs to and provides some high-level information about the age of the cluster and its current state:

```
kubectl get clusterwrappers
```

This command displays:

- The clusters you have by namespace, including the project ID the cluster is associated with
- The cluster name, which you can use to collate with that cluster's namespace
- The cluster's state and age

```
NAME           PROJECTID               CLUSTERNAME        STATE     AGE
p-6gtrowr94d   prj_X9s5sGg5Zl2tDl1v   dbcluster-1        Healthy   2d2h
p-ha6mloybrf   prj_X9s5sGg5Zl2tDl1v   dbcluster-2        Healthy   5d23h
p-p93ou61mgw   prj_X9s5sGg5Zl2tDl1v   dbcluster-3        Healthy   6d
```

Suppose you're trying to trace `dbcluster1` back to its Kubernetes resources. For example, you now know that `dbcluster1` is in the `p-6gtrowr94d` namespace. Each Postgres cluster has exactly one namespace. You can see that the state of the cluster is currently healthy and its age is two days and two hours.

When you have the name of the namespace for the Postgres cluster you're interested in, you can use that to see the pods running for that namespace:

```
kubectl get pods --namespace p-6gtrowr94d
```

Given that everything is functioning right, you see something like this as output:

```
NAME             READY   STATUS    RESTARTS   AGE
p-6gtrowr94d-1   3/3     Running   0          2d2h
p-6gtrowr94d-2   3/3     Running   0          2d2h
p-6gtrowr94d-3   3/3     Running   0          2d2h
```

Postgres clusters pending

Suppose you have a new cluster that you made using HM in a Provisioning state at 92% complete. The new cluster is named `psrtest-1` , and you want to see what's going on at the Kubernetes level. First use `kubectl get clusterwrappers` to collate the name of the PG cluster in HM with its namespace:

```
kubectl get clusterwrappers
__OUTPUT__
NAME           PROJECTID               CLUSTERNAME        STATE     AGE
p-gime4xeamp   prj_X9s5sGg5Zl2tDl1v   psrtest-1          Pending   9m40s
```

You already see that the status of the cluster is Pending, which makes sense as HM showed it was still provisioning. You can look deeper and also see how the pod provisioning is going now that you have the namespace for the cluster:

```
kubectl get pods --namespace p-gime4xeamp
__OUTPUT__
AME                           READY   STATUS      RESTARTS   AGE
p-gime4xeamp-1                 0/3     Pending     0          7m17s
p-gime4xeamp-1-initdb-g6d9b    0/1     Completed   0          9m46s
```

You see that one pod was successfully created and another is still being created. At this point, it doesn't seem like this cluster is actually stalled. It's just still being provisioned.

After a while, if the pod is still has Pending status, then there's good reason to think you ran out of resources. For example:

```
kubectl get pods --namespace p-gime4xeamp
__OUTPUT__
NAME                           READY   STATUS      RESTARTS   AGE
p-gime4xeamp-1                 0/3     Pending     0          28m
p-gime4xeamp-1-initdb-g6d9b    0/1     Completed   0          32m
```

To verify whether you ran out of resources, you can use `kubectl describe pod` to see exactly what's going on with it:

```
kubectl --namespace p-gime4xeamp describe pod p-gime4xamp-1
```

This command returns a lot of information about the pod that you don't need. But the very end of the output gives you more specifics about what's going wrong. For example, in this case you see:

```
Normal   NotTriggerScaleUp  4m15s (x181 over 34m)  cluster-autoscaler  pod didnt␣
↪trigger scale-up: 7 max node group size reached
```

This tells you that the autoscaler hit its max node group size. As a result, the pod can't be completed.

Warning

Even if you have enough resources for another cluster, if you try to create too many simultaneously, the system tries to create all of them at once. In this case, it's possible that none can be fully provisioned as they spread the load.

Adding resources

After you've learned that you hit the max node group size such that a pod can't be completed, change the max node group size so that it's larger. In an AWS EKS deployment, you do this by going to your AWS Console and then to EKS.

1. Select the EKS cluster you're working with, and then select the **Compute** tab.
2. In the **Node groups** section, select a `PGNodeGroup` , and then select **Edit**.
3. Change value of **Maximum Size** for the node group to something higher than what it was. HM then continues to provision the cluster.

If you tried to create many clusters at once, you may have to expand the **Maximum** size accordingly.

Formulas for parameter values

In addition to entering specific values for parameters in the ***DB Configuration** tab* , for some parameters you can specify formulas to calculate a value. You can use formulas for parameters of type integer and real in ternary formulas, such as the *shared buffer example* , using the following operators: `+ - / * > >= < <= == != && || ! ? : ( )` . Use `?` and `:` . Use `( )` to specify *Order of operations* , if needed. GUCs used in formulas must also be of type integer or real. All arithmetic is done on 64-bit floating point values rounded to an integer result if the target GUC is of type integer and not real.

Hybrid Manager (HM) also has what's refer to as *pseudo GUCs* to help with creating equations. These read-only GUCs are:

- `cpu_cores` — Number of cores provided by the VM running the cluster nodes.
- `disk_iops` — Number of IOPS allocated to the disks backing the primary data volume.
- `disk_size` — Amount of memory backing the primary data volume, in KB.
- `ram` — Amount of memory provided by the VM running the cluster nodes, in KB.
- `replica_count` — Total number of nodes in the cluster.

Formulas can calculate a value relative to the specifications you selected for the cluster or relative to other non-pseudo GUCs. If you resize your cluster, HM recalculates formulas using the new values for the pseudo GUCs.

Order of operations

`*` and `/` are at the same operator precedence. `+` and `-` are at the same operator precedence. `*` and `/` are evaluated before `+` and `-` . Use parentheses to specify additional order of operations.

Units

GUCs can either be a bare number, such as 8, or have a unit, such as GB. Units are categorized into kinds of units, such as size or duration.

The kind of unit of the target GUC must match the kind of unit used in the formula. For example, you can't use a formula that tries to multiply two seconds with three kilobytes (`2s * 3KB`), but you can use a formula that multiplies two sizes even if they have different scales (`2KB * 8GB`).

The unit of the target GUC must match the unit used in the formula. For example, you can multiply two seconds by four days but you can't assign the result to a GUC of unit KB.

If a target GUC has a unit, a bare number in an equation is assumed to be in the default unit for the target GUC.

Recursion

We recommend no more than approximately 100 levels of recursion depth.

Examples

Instead of entering a specific value for the `shared_buffers` parameter, with GUC equations you can specify that the value for `shared_buffers` is a certain percentage of RAM but no higher than a certain size. For example:

```
((ram * 0.25) > 8 GB) ? 8 GB : (ram * 0.25)
```

Suppose you want `wal_buffers` to be 3% of `shared_buffers` , up to a maximum of 16MB. The formula is:

```
((shared_buffers * 0.03) > 16MB) ? 16MB : (shared_buffers * 0.03)
```

The formula refers to the `shared_buffers` , which, in turn, has its own formula in this example. This example shows how GUCs can refer to other GUCs.

Common errors and error handling

Malformed formulas result in an error. Here are some examples:

- Following a number with two units, for example:

```
work_mem = 10 GB kB
```

This syntax gives the error:

```
unexpected token "kB" at end of input (line 1, col 7)
```

- Creating an infinite evaluation loop by having a GUC referring to itself, for example:

```
effective_cache_size = effective_cache_size
```

This syntax gives the error:

```
cycle in equation; GUC "effective_cache_size" already seen
```

- Mixing unit types, for example:

```
work_mem = 10kB * 8s
```

This syntax gives the error:

```
both size and time units used in calculation; cannot proceed
```

Managing database cluster access

If you want to configure database cluster access (as opposed to Hybrid Manager access) with an identity provider, see *LDAP authentication for database cluster access* .

LDAP authentication for database cluster access

You can configure LDAP for database cluster authentication.

- Applies to: Hybrid Manager version 1.2 and later

Prerequisites

- Before you deploy a cluster with LDAP configurations using the API, an LDAP server must be present in your environment.
- Before a user can authenticate via LDAP, the user must exist in PostgreSQL.

Configure LDAP via the API

After setting up the LDAP server, deploy the cluster using the API to configure LDAP authentication.

Creating a new cluster to use LDAP

Before using the REST API, make sure you have a valid key and the appropriate role is assigned to perform cluster CRUD operations.

For a new cluster, use curl to configure the LDAP server during cluster creation. Update the `${URL}` , `KEY` , `URL` , `project_id` , and other parameters in the request:

```
curl -H "x-access-key: ${KEY}" -H "Content-Type: application/json" --insecure -s
↪"https://.${URL} edbaiplatform.enterprisedb.network/api/v1/projects/prj_
↪pJ9Ug36492SQhdEQ/clusters" -d {"projectId": "prj_pJ9Ug36492SQhdEQ",
 "psr": {
   "clusterName": "ldap-cluster",
   "password": "111111111111",
   "locationId": "managed-devspatcher",
   "clusterData": {
     "instances": 1,
     "resourceRequest": {
       "requests": {
         "memory": "2Gi",
         "cpu": "1"
       }
     },
     "storageConfiguration": {
       "primaryStorage": {
         "size": "4"
       }
     },
     "image": {
       "url": "<pg_image_url>",
       "digest": "<pg_image_sha>"
     },
     "ldapConfig": {
```

(continues on next page)

(continued from previous page)

```
      "enabled": true,
      "config": {
        "hostname": "ldap-service.default.svc.cluster.local",
        "scheme": "ldap",
        "bindAuth": {
          "prefix": "uid=",
          "suffix": ",dc=example,dc=org"
        }
      }
    }
  }
}
```

Modify an existing cluster to use LDAP

For an existing cluster, use `PATCH` to configure it for LDAP authentication:

```
curl -X PATCH -H "x-access-key: ${KEY}" -H "Content-Type: application/json" --
↪insecure -s "https://portal.eks-05220612-main.edbaiplatform.enterprisedb.network/
↪api/v1/projects/prj_sXNNRhwj1YddHslR/clusters/p-lfdghq5gxe" -d {"projectId": "prj_
↪sXNNRhwj1YddHslR","clusterId": "p-lfdghq5gxe",
"psr": {
  "ldapConfig": {
   "enabled": true,
   "config": {
    "hostname": "ldap-service.default.svc.cluster.local",
    "scheme": "ldap",
    "bindAuth": {
     "prefix": "uid=",
     "suffix": ",dc=example,dc=org"
    }
   }
  }
 }
}
```

Verification

Verify the `pg_hba.conf` file was updated with an LDAP entry for authentication:

```
#
#  LDAP CONFIGURATION (optional)

#
host all all 0.0.0.0/0 ldap ldapserver="ldap-server.default.svc.cluster.local"␣
↪ldapscheme="ldap" ldapprefix="uid=" ldapsuffix=",dc=example,dc=org"
```

Bind-and-search mode configuration

The following parameters are required when using search+bind mode:

- `server`
- `prefix`
- `suffix`

The following parameters are optional when using search+bind mode:

- `searchFilter`
- `searchAttribute` : If `searchAttribute` is not specified, the default value is set to `uid` .
- `scheme` : an LDAP scheme
- `port`

This example shows configuring search+bind mode:

```
postgresql:
  ldap:
    server: openldap.default.svc.cluster.local
    bindSearchAuth:
      baseDN: ou=org,dc=example,dc=com
      bindDN: cn=admin,dc=example,dc=com
      bindPassword:
        name: ldapBindPassword
        key: data
      searchAttribute: uid
```

Bind mode configuration

The following parameters are required when using bind mode:

- `server`
- `baseDN`
- `binDN`
- `bindPassword` : a secret containing the ldap password

The following parameters are optional when using bind mode:

- `scheme` : an LDAP scheme
- `port`

This example shows configuring bind mode:

```
postgresql:
  ldap:
    server: "openldap.default.svc.cluster.local"
    bindAuth:
      prefix: "uid="
      suffix: ",dc=example,dc=org"
```

LDAP over TLS

You can use TLS to encrypt traffic between the PostgreSQL server and the LDAP server. You can configure your LDAP server to use TLS certificates.

The connection between the PostgreSQL server and the PostgreSQL client remains unencrypted unless SSL is configured.

More information

- LDAP Authentication
-

LDAP Configuration

Automating database cluster operations

You can automate routine database operations directly from Hybrid Manager (HM). Instead of manually monitoring your HM-managed clusters and reacting to issues, you configure rule-based automated operations that run on a schedule, detect conditions, and take action. It offers built-in controls to keep you in charge.

Automated operations are designed for teams managing production databases at scale. They reduce manual intervention, enforce consistent operations, and provide full visibility through the integrated **Task Manager**. They're available for HM-managed clusters running any supported distribution, including PostgreSQL, EDB Postgres Extended (PGE), and EDB Postgres Advanced Server (EPAS).

You configure automated operations on a per-cluster basis. For clusters with multiple replicas or data groups — such as HA, AHA and DHA clusters — configure the automation separately for each one.

Overview

This feature provides a shared set of capabilities across all automated operations:

- **Scheduled execution** — Define how often an automation runs using a configurable interval (for example, `5m` , `1h`).
- **Cooldown periods** — Set a minimum time between successive executions to prevent rapid, repeated actions. The minimum cooldown period is 5 minutes.
- **Optional maintenance windows** — If required, you can restrict automation execution to a defined time window to align with your operational schedule. Otherwise, the operation runs as soon as conditions are met.
- **Human-in-the-loop approvals** — Optionally require a user to review and approve a task before it runs. Recommended for production environments.
- **Role-based access control** — HM's RBAC system ensures only users with the appropriate permissions in the target database can approve tasks.
- **Notifications integration** — Task events in the **Activity Log** trigger notifications to users or third-party systems. Refer to *Set up notifications (Observability)* to learn how to configure project-level webhooks.
- **Task Manager integration** — All automation activity is tracked in Task Manager, giving you a centralized audit trail of every action taken.
- **Pause and resume** — Temporarily disable any automation without deleting its configuration.

When to configure automations

Automating cluster operations is a good fit when:

- **You need consistent, repeatable operations.** Automations apply the same logic on every run. This reduces human error and ensures that maintenance tasks execute according to policy, not on an ad hoc basis.
- **You want to maintain control without hands-on execution.** The approval workflow lets you require human sign-off before any action is taken, giving your team visibility and control without requiring them to perform the work manually.

Available automations

- ** *Apply index recommendation* ** — Automatically applies the most beneficial pending index recommendation on a cluster. HM determines the best recommendation based on the total query time saved relative to write performance impact.
- ** *Automatic image upgrade* ** — Automatically upgrades your distribution of Postgres to the latest available minor image version.
- ** *CPU auto-scaling* ** — Automatically scales CPU when average utilization exceeds a threshold.
- ** *Database storage auto-scaling* ** — Automatically expand your cluster's storage when utilization exceeds a threshold, preventing disk full outages.
- ** *Memory auto-scaling* ** — Automatically scales memory when average utilization exceeds a threshold.

Configuring an automated operation

You must configure automated operations after a cluster has been created, not during provisioning.

1. Open the target cluster and navigate to the **Settings** tab.
2. If your cluster has multiple replicas or data groups, use the dropdown at the top of the **Automations** section to select the replica or data group you want to configure.
3. Select **Configure** next to the automation you want to set up.
4. Fill in the general automation fields:

Field	Description
Display Name	A descriptive name for this automation (for example, *Database storage auto-scaling – Production*).
Interval	How often HM evaluates disk utilization (for example, *5m*, *1h*).
Cooldown Period	Minimum time to wait after a scaling event before evaluating again. This prevents rapid back-to-back expansions during usage spikes. For AWS deployments, set this to at least *6h* due to a platform restriction on disk expansion frequency.
Paused	Toggle to temporarily disable the automation without deleting its configuration.

1. Refer to the *Available automations* for additional configuration parameters.
2. Optionally, configure a **Maintenance Window** to restrict when the automated operation can run. See *Create a maintenance window* for instructions.
3. Optionally, configure approval settings:

- **Approval Required** — The automation generates a pending task in **Task Manager** that a user must explicitly approve before the database operation runs. Recommended for production clusters. - **Approval Settings > Approval Description** — A message shown to approvers that explains the purpose of the operation request. -

Approval Settings > Approval Timeout — How long to wait for approval before the task expires (for example, `24h` , `72h`). - **Require Reason** — When enabled, approvers must provide a written reason when approving or declining.

- **Approval Not Required** — The task runs automatically without manual review.

You can combine a maintenance window with approval requirements. In that case, the task must be approved *and* fall within an open window before it executes.

1. Select **Create Automation** to activate the automation.

Shared configuration

- *Create a maintenance window* — Restrict when automation tasks are allowed to run by defining one-time or recurring time windows at the project level.
- *Reviewing and approving automation tasks* — Review pending automation tasks, approve or reject them, and monitor task history in **Task Manager**.
- *Set up notifications (Observability)* — Configure webhooks, email, or Pager Duty to receive automation activity events in your external tools.

Limitations

- You can only configure automations after the cluster has been provisioned.
- You currently can't delete automation tasks in the HM console after creation.

Apply index recommendation

Apply index recommendation is an automation that selects the most beneficial pending index recommendation for an HM-managed cluster and applies it at each configured interval. HM determines the best recommendation based on the total query time saved relative to the write performance impact.

This automation reduces the manual effort of identifying, reviewing, and executing index changes, and ensures your performance tuning keeps pace with changing workloads. Unlike CPU and memory scaling, index creation doesn't require a database restart, but because index creation can affect query performance on large or resource-constrained databases, plan this automation carefully. Use a maintenance window to control when index creation runs, and enable approval requirements so your team can review each recommendation before it's applied.

> **Note**
>
> Apply index recommendation only works with HM-managed clusters. External databases aren't supported.

For **HA clusters**, configure it on each replica separately. Logical replication doesn't sync DDL changes, so indexes won't propagate between replicas automatically.

For **AHA and DHA clusters**, you only need to configure it on one data group. The automation applies the index to the write leader, and PGD replicates the change to all other data groups automatically.

Apply index recommendation parameters

When you configure an automated operation , there are specific parameters for the apply index recommendation automation in addition to the general automation fields:

Field	Description
Database Filter	Restrict the automation to specific databases within the cluster. You can add multiple values. Leave empty to consider all databases.
Schema Filter	Restrict the automation to specific schemas. You can add multiple values. Leave empty to consider all schemas. If multiple databases are selected, HM looks for a matching schema name across all of them and applies recommendations wherever a match is found.
Table Filter	Restrict the automation to specific tables. You can add multiple values. Leave empty to consider all tables. If multiple databases or schemas are selected, HM looks for a matching table name across all of them and applies recommendations wherever a match is found.
Maintenance Window	Schedule index creation during this maintenance window. Leave empty for immediate execution. See *Create a maintenance window* .
Approval	Select **Approval Required** to create a pending task in **Task Manager** that a user must approve before index creation runs, or **Approval Not Required** to run tasks automatically without manual review. **Approval Required** is recommended for production clusters.

Each time the automation runs, HM evaluates the pending index recommendations for the database cluster and selects the single most beneficial one. It then creates a task in **Task Manager** and executes it according to your configuration — either immediately or within the next open maintenance window, with or without approval. On subsequent runs, HM selects the next most beneficial pending recommendation and continues until no pending recommendations remain.

Tasks appear in Task Manager where you can monitor task status — pending, completed, or rejected — from either the cluster-level **Task Manager** tab or the project-level **Task Manager** in the left navigation. All executions are logged in the **Activity Log** for full auditability.

How the Recommendations tab and this automation work together

The automation draws from the same recommendation engine as the *Cluster recommendations* on the cluster view. The status of a recommendation in that tab directly affects what the automation picks up:

- To prevent the automation from applying a specific recommendation, open the **Recommendations** tab and mark it as **Ignored**. Ignored recommendations are excluded from the automation and won't reappear.
- Not approving a pending task in **Task Manager** doesn't mark the recommendation as ignored — the automation recreates the task after the cooldown period expires. To permanently skip a recommendation, mark it as **Ignored** in the **Recommendations** tab instead.

Automatic image upgrade

Automatic image upgrade is an automation that keeps your HM-managed cluster up to date by automatically applying the latest available minor Postgres image version whenever one becomes available. This automation ensures your clusters stay current without requiring manual intervention, helping you maintain security patches and bug fixes consistently across your estate.

The automation detects new minor version upgrades only (for example, 18.1 → 18.2). Newer image builds or patches of the same minor version aren't detected automatically.

> **Note**
>
> We recommend configuring a *Create a maintenance window* to control when upgrades run. Depending on your cluster type and configuration, an image upgrade may involve a rolling restart of cluster nodes.

Automatic image upgrade parameters

When you configure an automated operation , there are specific parameters for the automatic image upgrade automation in addition to the general automation fields:

Field	Description
Maintenance Window	Schedule the upgrade during this maintenance window. Leave empty for immediate execution when a new image version becomes available. See *Create a maintenance window* .
Approval	Select **Approval Required** to create a pending task in **Task Manager** that a user must approve before the upgrade runs, or **Approval Not Required** to run tasks automatically without manual review.

Once active, HM checks for a new minor image version at each interval. When a newer version is available, it creates a task in **Task Manager** and executes it according to your configuration — either immediately or within the next open maintenance window, with or without approval.

You can monitor task status — pending, completed, or rejected — from either the cluster-level **Task Manager** tab or the project-level **Task Manager** in the left navigation. All upgrade events are logged in the **Activity Log** for full auditability.

For information on what happens if an upgrade fails, including rollback behavior, see *Rollback considerations* .

CPU auto-scaling

CPU auto-scaling monitors the average CPU utilization of your HM-managed cluster and automatically scales up CPU resources when utilization exceeds a configured threshold. This automation prevents performance degradation and potential outages as your database resource consumption grows, without requiring manual intervention.

This automation only scales CPU up, not down. If you need to reduce CPU allocation, you must do so manually.

> **Warning**
>
> CPU scaling triggers a database restart. Configure a *Create a maintenance window* to control when the restart occurs and avoid unexpected downtime on production clusters.

You configure automations per replica or data group using the dropdown in the **Automations** section. Scaling applies to all nodes within the selected replica or data group. It isn't coordinated across replicas or data groups — configure each one separately if you need the same behavior elsewhere.

CPU auto-scaling parameters

When you configure an automated operation , there are specific parameters for the CPU auto-scaling automation in addition to the general automation fields:

Field	Description
CPU Threshold	The average CPU utilization percentage that triggers a scaling event (for example, *80*). Min: *1*, Max: *99*.
Lookback Window (minutes)	The number of minutes over which HM averages CPU utilization before comparing it to the threshold (for example, *5*). This smooths out short-lived spikes. Min: *1*, Max: *60*.

continues on next page

Table 11 – continued from previous page

Field	Description
Scaling Method	Choose how to increase CPU resources: **Scale by Quantity** adds a fixed amount each time, expressed in *Troubleshooting* , **Scale by Percentage** increases by a percentage of the current value.
Maximum CPU	The upper limit for CPU scaling. The automation never scales beyond this value — if a full increment would exceed it, it scales to the maximum instead. Once reached, no further scaling occurs.
Maintenance Window	The maintenance window during which the scaling operation and required database restart are allowed to run. Because CPU scaling requires a restart, leaving this empty isn't recommended for production clusters. See *Create a maintenance window* .
Approval	Select **Approval Required** to create a pending task in **Task Manager** that a user must approve before it runs, or **Approval Not Required** to run tasks automatically without manual review.

Once active, HM evaluates average CPU utilization at each interval using the configured lookback window. When the threshold is crossed, a task is created in **Task Manager**. You can monitor task status — pending, completed, or rejected — from either the cluster-level **Task Manager** tab or the project-level **Task Manager** in the left navigation.

> **Note**
>
> After a CPU scaling event, review your Postgres configuration to ensure it's optimized for the new resource allocation. Parameters such as *max_parallel_workers* , *max_parallel_workers_per_gather* , and *max_worker_processes* may need adjustment to take full advantage of additional CPU.

Database storage auto-scaling

Database storage auto-scaling monitors the storage utilization of your HM-managed cluster's data volumes. When usage exceeds a configured threshold, HM automatically expands the underlying `PersistentVolumeClaim` (PVC) — keeping your databases online and preventing outages caused by disk full events.

You configure the trigger threshold, the scaling method (fixed amount or percentage), and a maximum disk size to prevent runaway growth. Expansion is performed as an online operation (if your storage driver supports it), so your Postgres database remains accessible throughout.

> **Note**
>
> Database storage auto-scaling applies to the primary data volume only. If your cluster uses separate WAL volumes, the WAL volume is not in scope. Backup storage is also not affected.

You configure automations per replica or data group using the dropdown in the **Automations** section. HM monitors and scales each node's data volume within the selected replica or data group. Automations aren't coordinated across replicas or data groups — configure each one separately if you need the same behavior elsewhere.

Database storage auto-scaling parameters

When you configure an automated operation , there are specific parameters for the database storage auto-scaling automation in addition to the general automation fields:

Field	Description
Disk Threshold	The disk utilization percentage that triggers a scaling event (for example, *80*).
Scaling Method	Choose how to increase the disk size: **Scale by Quantity** adds a fixed size each time (for example, *10Gi*, *50Gi*); **Scale by Percentage** increases by a percentage of the current capacity (for example, *20%*).
Maximum Disk Size	The upper limit for scaling. The automation stops expanding once this size is reached, preventing runaway growth.
Maintenance Window	If your storage driver supports online expansion — meaning the operation causes no downtime — you can leave the maintenance window field empty and the task will execute as soon as the threshold is crossed.
Approval	Select **Approval Required** to create a pending task in **Task Manager** that a user must approve before it runs, or **Approval Not Required** to run tasks automatically without manual review.

Once active, HM evaluates disk utilization at each interval. When the threshold is crossed, a task is created in **Task Manager**. You can monitor task status — pending, completed, or rejected — from either the cluster-level **Task Manager** tab or the project-level **Task Manager** in the left navigation.

Memory auto-scaling

Memory auto-scaling monitors the average memory utilization of your HM-managed cluster and automatically scales up memory resources when utilization exceeds a configured threshold. This automation prevents performance degradation as your database resource consumption grows, without requiring manual intervention.

> **Warning**
>
> Memory scaling triggers a database restart. Configure a *Create a maintenance window* to control when the restart occurs and avoid unexpected downtime on production clusters.

You configure automations per replica or data group using the dropdown in the **Automations** section. Scaling applies to all nodes within the selected replica or data group. It isn't coordinated across replicas or data groups — configure each one separately if you need the same behavior elsewhere.

Memory auto-scaling parameters

When you configure an automated operation , there are specific parameters for the memory auto-scaling automation in addition to the general automation fields:

Field	Description
Memory Threshold	The average memory utilization percentage that triggers a scaling event (for example, *80*). Min: *1*, Max: *99*.
Lookback Window (minutes)	The number of minutes over which HM averages memory utilization before comparing it to the threshold (for example, *5*). This smooths out short-lived spikes. Min: *1*, Max: *60*.
Scaling Method	Choose how to increase memory resources: **Scale by Quantity** adds a fixed amount each time, expressed in *Troubleshooting* , **Scale by Percentage** increases by a percentage of the current value.

continues on next page

Table 13 – continued from previous page

Field	Description
Maximum Memory	The upper limit for memory scaling (for example, *32Gi*). The automation never scales beyond this value — if a full increment would exceed it, it scales to the maximum instead. Once reached, no further scaling occurs.
Maintenance Window	The maintenance window during which the scaling operation and required database restart are allowed to run. Because memory scaling requires a restart, leaving this empty isn't recommended for production clusters. See *Create a maintenance window* .
Approval	Select **Approval Required** to create a pending task in **Task Manager** that a user must approve before it runs, or **Approval Not Required** to run tasks automatically without manual review.

Once active, HM evaluates average memory utilization at each interval using the configured lookback window. When the threshold is crossed, a task is created in **Task Manager**. You can monitor task status — pending, completed, or rejected — from either the cluster-level **Task Manager** tab or the project-level **Task Manager** in the left navigation.

Note

After a memory scaling event, review your Postgres configuration to ensure it's optimized for the new resource allocation. Parameters such as *shared_buffers* , *effective_cache_size* , and *work_mem* may need adjustment to take full advantage of additional memory.

Create a maintenance window

Maintenance windows let you restrict automation execution to a defined time period. Instead of running a task the moment a condition is met, Hybrid Manager (HM) waits until the cluster falls within an open window before acting.

You can configure maintenance windows at the **project level**, where they can be shared and reused across all clusters in the project. Once created, you reference a window when configuring any individual automation.

If a task is triggered while a window is open, it executes immediately. If no window is open at that moment, the task waits and runs at the next occurrence.

Note

Maintenance windows are optional. If the operation doesn't require downtime (for example, auto-increase storage when the storage driver supports online expansion), you can leave the maintenance window field empty when configuring an automation and the task will execute as soon as the condition is met.

Create a maintenance window

1. In the HM console, navigate to your **Project**.
2. Go to **Settings > Maintenance Windows**.
3. Select **Create New Window** and fill in the following fields:

Field	Description
Display Name	A descriptive name for the window (for example, *Weekend maintenance* or *Nightly – pre-prod*).
Type	Select **Recurring (Cron)** for a repeating schedule, or **One-Time** for a single occurrence.

If you select **Recurring (Cron)**:

Field	Description
Cron Expression	Defines when the window starts and how often it repeats (for example, *0 22 ** 6* for every Saturday at 10 PM).
Duration	How long the window stays open each time it starts (for example, *2h*).

If you select **One-Time**:

Field	Description
Start Time	The date and time when the window opens (UTC).
End Time	The date and time when the window closes (UTC).

1. Save the maintenance window. It's now available to select when configuring automations for any cluster in the project.

Usage and scope

To apply a window to an automation, select it from the **Maintenance Window** dropdown in the automation settings. You can combine a maintenance window with approval requirements — in that case, the task must be both approved and fall within an open window before it executes.

Maintenance windows are project-scoped. You can reuse a single window across all automations and clusters within a project, but it isn't shared across projects. If you want the same schedule in multiple projects, create the window separately in each one. In the future, maintenance windows are expected to be usable for other types of cluster operations beyond automations.

Reviewing and approving automation tasks

When you configure an automation with **Approval Required**, Hybrid Manager (HM) doesn't run the operation immediately. Instead, it creates a pending task and waits for a team member to review it. **Task Manager** is where that review happens — it gives you a centralized view of all pending, running, and completed automation tasks across your project, so you can stay in control of what runs and when. You can also manage tasks programmatically using the HM API, which lets you integrate the approval flow with third-party tools and workflows.

Accessing Task Manager

Task Manager is available at two levels:

- **Project level** — Within a project, in the left navigation, select **Task Manager**. This view shows all pending, running, failed, and completed tasks across every cluster in the project.
- **Cluster level** — Within a project, in the left navigation, select **Clusters** > **Postgres Clusters**, then select a cluster. In the cluster view, select the **Task Manager** tab. This view shows only the tasks for that specific cluster.

Reading the task list

Each row in the task list represents a single automation task. The following columns appear by default:

Column	Description
Task Name	The name of the automation task. Select the name to open the task detail sidebar.
Status	The current state of the task. See *Understanding task status* .
Created	When the task was created.
Duration	How long the task took to execute. Displays a dash (-) for tasks that haven't run yet.
Source	The automation rule that triggered the task.
Target	The cluster targeted by the task. Task Manager hides this column when you view it from within a specific cluster.

You can customize which columns you display using the settings icon at the top of the task list. **Priority** is an optional column that doesn't appear by default.

Understanding task status

Each task moves through a lifecycle that you can track in the **Status** column:

Status	Meaning
Pending	The task is waiting — either for approval (if the automation requires approval) or for its maintenance window to open.
Gate Satisfied	All conditions are met. The task executes when its maintenance window opens, or immediately if you didn't configure a window.
Running	The task is currently executing.
Completed	The task finished successfully.
Failed	The task couldn't run or encountered an error. When a specific failure reason is available, the status badge shows an error code instead of *Failed*. See *Failure error codes* .
Cancelled	A user or process canceled the task before it ran.

Failure error codes

When a task fails, the status badge shows a specific error code to indicate why:

Error code	Meaning
MaintenanceWindowFailed	The maintenance window closed before the task could run. If the automation's cooldown period has passed, HM recreates the task on the next evaluation cycle.
ApprovalFailed	A user rejected the task, or the approval timeout expired before anyone approved it. If the automation's cooldown period has passed, HM recreates the task on the next evaluation cycle.
ExecutionError	The task started but encountered an error during execution. Check the task detail sidebar for the error message.
Timeout	The task exceeded the maximum allowed execution time.

Approving or rejecting a task

When a task is in **Pending** status and requires your approval:

1. Select the task name or its status badge to open the task detail sidebar.
2. Review the task details — including the automation type, target cluster or data group, and the approval description that the automation owner configured.
3. Select **Approve** to allow the task to proceed, or **Deny** to reject it.
4. If you configured the automation with **Require Reason**, add an explanation in the **Reason** field before confirming your decision.

After approval, the task moves to **Running** if you didn't configure a maintenance window, or to **Gate Satisfied** if a maintenance window exists. After rejection, HM closes the task. If the automation's cooldown period has passed, HM recreates the task on the next evaluation cycle.

> **Note**
>
> To approve a task, you need the same permissions required to perform the underlying operation on the cluster. For example, approving an automatic image upgrade requires the same permissions as performing a manual upgrade. HM's RBAC system enforces this.

Notifications for pending approval tasks

When an automation task transitions to **Pending** status and requires approval, HM sends a notification through any channels you've configured at the project or organization level.

- **In-app** — A notification appears in the bell icon menu in the top-right corner of the HM console. The notification reflects the task's status change and indicates that a system action requires attention. Select the bell icon to view it, then navigate to **Task Manager** to review and approve the task.
- **External channels** — If you've configured email, Pager Duty, or a webhook, HM sends a notification with additional context about the pending task through those channels. Navigate to **Task Manager** to approve or reject the task.

In both cases, approval must be completed in **Task Manager** — the notification itself doesn't include a direct approval action.

Monitoring task history

Task Manager retains a history of all completed, failed, and canceled tasks. Use it to audit what ran, when, and with what outcome.

To narrow down the list, use the available filters:

- **Status** — Filter by one or more task phases (for example, show only `Pending` or `Failed` tasks).
- **Date range** — Limit results to a specific time window. Preset ranges of 7, 15, 30, and 90 days are available, or set a custom range.
- **Search** — Find tasks by name.

Select any task to open its detail sidebar. The *Auditing platform activity* also records all task activity and provides an org- or project-wide audit trail across all resource types, not just tasks.

Cluster view

The Cluster view provides a detailed overview of the cluster, including its configuration, health, monitoring, query diagnostics, backups, logs, and other properties. To open the Cluster view, select a cluster from the Clusters table.

The Cluster view displays the name of the cluster, the creation time, the type of cluster, the Postgres distribution used, and the version of the Postgres distribution. (The creation time shows when the cluster was created relative to the current time.) Select the cluster name to open the cluster's connection information page.

Tags are used to group clusters for easier management. On the Cluster view, the tags appear as a list of links. Select a tag to open a detailed view that shows all the resources that have that tag.

To add or remove tags, select +. In the dialog box, you can add and create tags and remove tags from the cluster.

The tabs in the cluster view are:

- *Overview* — A brief overview of the cluster.
- *Connection* — Connection information for the cluster.
- *Monitoring execution* — Monitoring information for the cluster.
- *Diagnostics (Query Diagnostics and Recommendations tabs)* — Query diagnostics information for the cluster.
- *HM backups and DR for multi-DC environments* — Backup information for the cluster.
- *Logs and observability* — Logs for the cluster.
- *Cluster Properties* — Properties of the cluster.
- *Health assessment reports* — Generate, view, download, and compare health assessment reports for your database clusters.

Cluster overview

The **Cluster Overview** tab shows a summary of the cluster's connection string, active alerts, and recent cluster activity.

If the cluster is a single node or primary/standby high-availability cluster, the overview first shows the *Getting your connection string* section.

If the cluster is a distributed high-availability cluster, the overview first shows the *EDB Postgres Distributed (PGD) cluster with 2 data groups and 1 witness group not healthy* section.

Getting your connection string

The **Quick Connect** section provides a full psql command line with a connection string configured for the cluster and a user with the edb_admin role to connect to the `edb_admin` database. You can copy this command and paste it into a terminal to connect to the database. (You need the password for the edb_admin user to connect to the database.)

Data groups

Under **Data Groups**, view a summary of the data groups and witness groups in the cluster. Each group is listed with a status badge, the data group's name, and a **Data Group** or **Witness Group** label.

The status badge indicates the status of the data group. A green circle with a white tick indicates the data group is healthy. The other information displayed includes the time since creation, Postgres server type and version, and number of nodes in the data group.

- To edit a data group's node settings and database configurations, next to the data group name, select **Edit Data Group**.
- When you have more than one data group, you can delete data groups. (You can't delete all the data groups.) From the ellipsis menu, select **Delete Data Group**.

- To restore a data group you deleted, from the ellipsis menu, select **Restore Data Group**.
- To perform a write leader switchover, from the ellipsis menu, select *Switchover Write Leader* .

Under **Quick Connect** for the data group, you can get the psql command line connection string for read-write and read-only connections.

Select **Read-write** or **Read-only** to see the psql command line with a connection string configured for the data group and a user with the edb_admin role to connect to the `bdrdb` database. To connect to the database, you can copy this command and paste it into a terminal. To do so, you need the password for the edb_admin user.

> **Note**
>
> The **Read-only** connection string isn't available in Hybrid Manager (HM) 1.1.

To toggle the display of a cluster architecture diagram, select the **Cluster Architecture Diagram** heading. (The diagram is hidden by default.) This diagram shows the architecture of the cluster, including the data groups, witness groups, and proxies, and the interconnection between them.

Switchover Write Leader

Initiate a write leader switchover within an Advanced High Availability or Distributed High Availability cluster data group to seamlessly transfer the write leader role from one node to another. This zero-data-loss operation is designed to minimize downtime, ensuring that running applications experience a smooth transition with no lost transactions.

The write leader switchover serves several key purposes:

- **Planned maintenance**: Before taking a node offline for patching, an OS update, or hardware replacement, shift the write leader role to a healthy node, perform the necessary maintenance, and then switch the role back.
- **Routine testing**: Perform switchovers as part of regular disaster recovery drills to validate the process and ensure expected functionality during a real event.
- **Load balancing**: Move the write leader role to a different node to balance the workload across the cluster or to place the write leader closer to a specific group of applications.

More details are available in Switching the write leader .

Active alerts

The number of currently active alerts are listed in three categories: High Severity, Medium Severity, and Low Severity. Selecting the displayed counts opens the *Cluster monitoring, alerts, and observability* tab, where you can view the alert details.

Cluster activity log

You can view the recent activity logged on the cluster, including creation, configuration updates, and other major events in the cluster's lifecycle. Active elements don't appear here. For more details on the events, select the *Cluster Logs* tab.

When HM detects a schema change in an HM-managed database, it automatically inserts an event entry in the Cluster Activity Log. You can use these entries to track when schema changes occurred and correlate them with changes in cluster performance.

> **Note**
>
> The Cluster Activity Log isn't available for self-managed clusters. Schema change events for self-managed clusters appear as event markers on the *Cluster monitoring, alerts, and observability* tab instead.

Cluster connections

Use the **Connect** tab to get the connection information for the cluster and a user with the edb_admin role to connect to the `edb_admin` database in the cluster.

If the cluster is a distributed high-availability cluster, the connection information is provided for each data group in the cluster.

Connection information

You can see the following connection information:

- **Read/Write URI**: The connection string for connecting to the cluster with read/write access. This is the same as the psql command line in the **Quick Connect** section.
- **Dbname**: The name of the database to connect to. This is always `edb_admin` .
- **Read/write Host**: The hostname or IP address of the cluster for read/write access.
- **Port**: The port to connect to. This is always `5432` .
- **User**: The user to connect as. This is always edb_admin.

You can copy each of these fields to use for starting connection attempts to the database.

Read-only connection information

If you enabled a read-only connection when creating or editing your cluster, the **Connect** tab also displays a read-only connection string. Use this connection string to route read-only operations to standby replicas (for HA clusters) or to nodes that aren't currently designated as the write leader (for AHA and DHA clusters), reducing load on the primary instance.

You can see the following read-only connection information:

- **Read-only URI**: The connection string for connecting to the cluster for read-only access.
- **Read-only Host**: The hostname or IP address of the read-only endpoint.

For DHA clusters, read-only connection information is provided separately for each data group.

To enable or configure a read-only connection, see *Cluster setup* for HA clusters, or *EDB Postgres Distributed (PGD) cluster with 2 data groups and 1 witness group not healthy* for AHA and DHA clusters.

`.pgpass` blob

You can copy and paste a `.pgpass` blob into your `.pgpass` file so you can connect to the cluster without entering a password. This is useful for automating connections to the cluster, such as in scripts or applications. Replace the placeholder with the actual password. To prevent unauthorized access, ensure that you have the correct permissions set on the `.pgpass` file. Set the permissions to `0600` (read and write for the user only).

For more information about the `.pgpass` file, see the PostgreSQL documentation .

`.pg_service.conf` blob

You can copy and paste a `.pg_service.conf` blob into your `.pg_service.conf` file so you can connect to the cluster by referring to it by a *service name*. Doing so makes it easier to remember connections to the cluster. In combination with the `.pgpass` blob, you can connect by specifying the service name. For more information about the `.pg_service.conf` file, see the PostgreSQL documentation .

The `pgpass` and `.pg_service.conf` files are supported by any Postgres client application that uses libpq.

Cluster monitoring

The **Monitoring** tab provides a detailed view of node and cluster health.

Monitoring

Review the **Monitoring** section for a summary of the cluster's health status. Use the monitoring control at the top of the table to control what the table shows.

You can specify when you want the displayed data to start. The default is 15 minutes, which means to display data from the last 15 minutes. You can specify a value between 15 days and 5 minutes. Alternatively, use the date and time pickers to select a custom time range to sample the monitoring data from. You can select time ranges only where data for the cluster is available. Select a time from the first menu clears any date and time you specified.

You can also specify whether to display the data as an aggregate of all the nodes in the cluster (**Cluster level**) or for each node (**Node level**). The default is to show the data at the cluster level.

Enable **Display Events** for each monitoring category (**Host**, **Connections**, **Transactions**, etc.) to overlay event markers, including schema change events detected by HM. For self-managed clusters monitored through the EDB Postgres AI agent, schema change events appear as event markers rather than in the Cluster Activity Log.

At the cluster level, each ring chart is a single ring showing the average of the data for all nodes in the cluster and graphs are based on a similar aggregation.

At the node level, each chart is a concentric ring chart. Each ring represents a node in the cluster. Graphs are displayed separately for each node.

Comparing time periods

The comparison view lets you compare monitoring metrics and query statistics between two non-adjacent time periods side by side, so you can understand the impact of changes, incidents, or workload shifts. To open it, select **+ New Comparison** from the **Monitoring** tab.

Define a **Reference Period** and a **Comparison Period**. For each period, set the start and end time using the date and time pickers, or by selecting a deployment marker event. Use the **Monitoring** and **Query Diagnostics** toggles to include or exclude those sections from the comparison.

The view displays charts for both periods side by side, with a delta badge between them showing the percentage difference. The **Reference vs Comparison Period** section below the charts compares query performance across both periods. Results are scoped to a single cluster and support both cluster-level and node-level views.

> **Note**
>
> Data availability is subject to retention limits: monitoring metrics are retained for up to 30 days, and query statistics for up to 7 days.

Active alerts

The summary of active alerts displays the number of active alerts in three categories: high severity, medium severity, and low severity.

In the searchable table view of alerts, the search/filter bar lets you:

- Filter by a text search term. Enter a value in the **Search** box.
- Filter by a time range. Select the **From** and **To** date/time pickers.
- Filter by severity. Select **Filter > Severity** and then one or more of the available severities.
- Sort by start time, severity, or alert in ascending or descending order.
- Select auto-refreshing by selecting the lightning icon.
- Select the displayed columns. Select the cog and then, from the menu, select the columns to display in the table.
- Download the currently displayed alerts in CSV format. Select the download icon.

Host

Review the **Host** section for summary of operating system statistics displayed in charts:

- **Memory** : The average memory usage percentage of memory for Postgres primary node.
- **CPU** : The average CPU percentage for Postgres primary node.
- **Storage** : Total storage used for Postgres primary node.
- **Disk IOPS** : Total number of reads, writes, and total operations on the disk per second over a time period.
- **Disk Throughput** : Total amount of data transferred to and from the disk per second for Postgres nodes.
- **Network Activity** : Total amount of data transferred to and from the network card per second over a time period for Postgres nodes.

Connections

- **Connections** : The current number of connections between the client applications and Postgres database by type.
- **Average Active Sessions by Wait Type** : Time spent on each wait event type by the primary Postgres nodes is calculated using Average Active Sessions (AAS).
- **Number of blocked backends** : Total number of backends waiting on locks across all Postgres nodes in the cluster.

Transactions

- **Tuples In** : Total number of tuples inserted, updated, and deleted per second for Postgres nodes.
- **Tuples Out** : Total number of tuples fetched and returned per second for Postgres nodes.
- **Transaction Rate** : Total number of committed/rolled-back/total transactions per second for Postgres nodes.
- **Buffer Cache Hit Ratio** : Average buffer cache hit percentage across all Postgres nodes in the cluster.
- **Longest Running Transaction** : Total number of longest running transactions per second for Postgres nodes.

Queries

- **Query Rate** : Total number of queries per second for Postgres nodes.
- **Query Latency** : Average query latency in milliseconds across all Postgres nodes in the cluster.

Storage

- **Database size** (line chart) : Total database size across the primary Postgres nodes.
- **Disk Usage** : Disk usage in percentage for primary Postgres nodes.
- **WAL Size** : Total WAL directory size across the primary Postgres nodes.
- **WAL Usage** : WAL usage in percentage for primary Postgres nodes.
- **Live/Dead Tuples** : Total number of Live/Dead tuples for Postgres nodes.
- **Index, Table, and Temp estimated size** : Estimated size of index, table and temp for Postgres nodes.

Internals

- **Time Since Last Autovacuum** : Time since the last autovacuum in seconds for Postgres nodes.
- **Autovacuum Stats** : Total number of autovacuum operations per second for Postgres nodes.
- **Time Since Last Checkpoint** : Time since the last checkpoint in seconds for Postgres nodes.
- **Checkpoint Stats** : Total number of checkpoints per second for Postgres nodes.
- **Time Since Last Successful WAL Archive** : Time since the last successful WAL archive in seconds for Postgres nodes.
- **WAL Archiving Stats** : Total number of WAL archiving operations per second for Postgres nodes.

Cluster Query Diagnostics

Query Diagnostics

The Query Diagnostics tab provides insight into the workload running on your cluster. It collects detailed statistics on the queries that have been executed, enabling users to quickly identify problem areas.

The Query Diagnostics tab comprises four charts and a table of queries, providing helpful statistics for diagnosing performance issues with query execution.

Charts

The charts are the following:

- “Query latency”, a line chart with query latency percentiles (50, 90 and 99),
- “Number of queries executed”, a line chart with the raw count of executed queries,
- “Query latency histogram”, a bar chart with query latency distributions,
- “Number of active sessions”, a stacked area chart with the number of active sessions and the related wait states for I/O, CPU, IPC, and locks.

The filters allow users to restrict data by:

- A time interval,
- A start and end date,

- Specific nodes.

Query Table

Below the charts, a table lists all the collected queries along with their info and statistics:

- "Query ID", the ID of the query,
- "Query", the truncated query text,
- "# of Executions", the number of executions for this query,
- "50th Percentile", the 50th percentile for query latency,
- "90th Percentile", the 90th percentile for query latency,
- "99th Percentile", the 99th percentile for query latency,
- "CPU %", the percentage of time spent working on the CPU,
- "Wait %", the percentage of time spent waiting (i.e. active, but not on CPU),
- "Min Execution Time",
- "Max Execution Time",
- "CPU AAS", average active sessions spent on CPU by the query
- "Wait AAS", average active sessions spent in wait states by the query,
- "Total AAS", average active sessions spent running the query,
- "Count", number users, databases and nodes that executed this query at least once.

With the cog icon, it's possible to show/hide columns.

The filters allow users to restrict data by:

- Databases,
- Users.

The drop-down menu on the right side allows you to sort rows in the table by column in ascending or descending order.

The search bar on the left side lets you search for a substring in the query text.

Understanding Query Diagnostics

What is a query?

For the purpose of Query Diagnostic, we defined a query as being a unique query text after predicates are removed (also known as normalization). For example, `SELECT id FROM users WHERE name = "Bob"` ia the same query as `SELECT id FROM users WHERE name = "Alice"` . This is true even if the queries run in different databases against different versions of the `users` table.

What does Average Active Sessions (AAS) mean?

Average Active Sessions is a metric of how much time was spent in a particular state across all sessions. It is calculated as the proportion of time in each session spent in the state of interest, summed over all sessions.

For example, if a single session runs the same query continuously for an hour, that query would be said to occupy 1 AAS for that hour. If it only spent half of the hour running the query, that would be 0.5 AAS.

If there were instead 10 sessions running the query, each for half of the hour, that would be 5 AAS.

Average Active Sessions is a useful metric to answer the question "what were my top ten queries yesterday?". By definition, the ten queries with the highest AAS will be those that occupied the most time across the whole day, across all sessions, . For AAS, whether the query executed a million times in the day or only once is irrelevant: it measures the total execution time.

Where does the data come from?

Data for Query Diagnostics is sourced from two extensions. If only one is available, functionality is reduced. If neither is available Query Diagnostics will be disabled.

- EDB Wait States collects sampled data from the session state. This is used to populate the AAS and related metrics.
- EDB Stat Monitor collects data at the query execution level. By default it records every query execution. This is used to populate the number of executions, and query latency metrics.

Does Query Diagnostics show every query executed?

Yes, if EDB Stat Monitor is enabled and `edbsm_sample_rate` is set to its default of `1` then Query Diagnostics will include every query executed with the exception of utility queries such as `SET` and queries executed inside PGPLSQL procedures.

If EDB Stat Monitor is not enabled, the queries shown will only be those which have been observed by EDB Wait States. This is likely to include any query that contributes a significant amount of AAS, but this cannot be guaranteed.

Cluster recommendations

Recommendations

The **Recommendations** tab shows a scorecard and a list of recommendations that can help to improve the Postgres cluster performance.

The scorecard and recommendation are directly related. A cluster scores lower than Excellent only if any recommendations appear. After you apply or dismiss the recommendations, the score shows Excellent.

Scorecard

In the scorecard, a score is assigned to a given cluster for each grading category:

- Indexes
- Statistics
- Configuration
- Security
- Workload

> **Note**
>
> Currently, Indexes, Statistics, Configuration, and Security are available.

Generally, grades are assigned on an A-F scale.

For the Indexes category, the possible grades are:

- Excellent (A) if no recommendations were generated to tune indexes

- Good (B), Fair (C), and Poor (D), depending on how many recommendations were generated

For Indexes, the scores are computed by considering only primary nodes in the case of PSR and write leaders in the case of PGD/AHA. Furthermore, the score is related to the database in the PG cluster that is performing the worst, not an average over all databases created in such a cluster).

For the Statistics category, the possible grades are:

- Excellent (A) if no recommendations were generated to add statistics
- Good (B) if at least one recommendation was generated

For the Configuration category, the possible grades are:

- Excellent (A) if no recommendations were generated to tune configuration parameters
- Good (B), Fair (C), Poor (D), and Fail (F), depending on how many recommendations were generated

For the Security category, the possible grades are:

- Excellent (A) if no recommendations were generated to improve security
- Good (B), Fair (C), Poor (D), and Fail (F), depending on how many recommendations were generated

Support for Workload scorecards will be added in a later release.

> **Note**
>
> "Unknown" (U) means that information needed to evaluate a cluster is unavailable.

When you select a category, the displayed text helps you to understand the relevance of the recommendations in addition to providing context on how the score was computed.

A radar chart shows the global score of the cluster across all categories.

Recommendations list

Below the scorecard, a list of recommendations is displayed, depending on the category selected in the scorecard:

- Status: Status of a recommendation (New, Ignored, Applied).
- Database Name.
- Node ID.
- Size Increase (for indexes): Estimated index size.
- Absolute Benefit (for indexes): Absolute cost reduction that applying this index can produce.
- Total Cost (for indexes): Execution time plus plan time for running this query.
- Cost Reduction (for indexes): Estimated percentage cost reduction computed as (abs_benefit/total_cost) * 100, which is relative to all the workloads executed in a database.
- Weight (for statistics): Rough estimation of the potential benefit of applying the statistics.
- Recommended Time: Creation time for this recommendation.
- Query: Text of the query to create an index or a statistic.

Use the cog icon to show/hide columns.

CTA buttons in each row let you:

- Mark the recommendation as completed

- Ignore the recommendation
- Copy the text of the recommendation

> **Note**
>
> You can't apply recommendations from the web interface. You must manually copy and execute them in the terminal or an admin UI.

Use the filters to restrict data by database, node, or status.

Understanding recommendations

Mark the recommendation as applied/ignored

Use the **applied** option to tell Hybrid Manager (HM) that you applied the recommended change. If you mark a recommendation as applied, it disappears from the list. When the page is next reloaded, the score increases to reflect the change. If the same recommendation appears again, this means the change was not applied or it was undone. The score will drop again.

Use the **ignored** option to tell HM that you reviewed the recommended change and decided not to apply it. If you mark a recommendation as ignored, it disappears from the list. When the page is next reloaded, the score increases. If the same recommendation is received again, it's ignored and doesn't reappear.

Where does this data come from?

The Index and Statistics recommendations are sourced from the Query Advisor extension. Make sure this extension is enabled in the monitored database.

The Configuration and Security recommendations are sourced from the Postgres configuration.

In future releases, recommendations will be generated from other data sources, including historical metrics and Query Diagnostics.

Backup Management

The Backups tab under a specific cluster in Hybrid Manager provides a centralized view and control panel for managing the backups of that PostgreSQL cluster. This page helps users track, validate, and manage the backup lifecycle efficiently and securely.

Purpose

Backups tab is designed to provide visibility into all backup activities associated with the selected cluster. It supports both monitoring past backups and initiating new ones. It is especially useful for operations and site reliability engineers (SRE) teams to ensure data protection and facilitate recovery processes when needed.

Key features

- **Monitor backup status**

Easily check whether backups were successfully completed or failed, along with timestamps and durations.

- **Initiate manual backups** Use the Create Backup button to trigger an immediate backup of the cluster.
- **Review backup metadata**

Get detailed insights such as start and stop time, method of backup (e.g., VolumeSnapshot), and the storage location used.

- **Restore from backups**

Use the restore icon (□□) to initiate recovery from a specific backup point.

- **Copy backupID**

Use the copy icon (□) to copy the backup identifier for referencing in automation, support cases, or logs.

- **Delete backups**

Use the delete icon (□□) to delete the backup manually which are no longer required.

The backups tab provides detailed information of backup jobs with the following columns:

Column	Description
Status	Indicates whether the backup completed successfully (Completed), failed, or is still running.
Name	System-generated name of the backup job.
Start Time / Stop Time	Timestamps showing when the backup started and ended.
Duration	How long the backup took to complete.
Location	The storage destination for the backup (e.g., managed-devspatcher).
Method	The technology used to take the backup (e.g., VolumeSnapshot).

Usability

Use this tab when you need to:

- Verifying successful completion of automated or scheduled backups
- Performing a backup prior to a major upgrade or maintenance task
- Restoring the database to a previous known-good state
- Auditing historical backup activity

Cluster Logs

The Logs tab for a selected cluster in the Hybrid Manager provides real-time and historical insights into the cluster's internal operations, system events, and extension-level activities. It is a critical tool for debugging, diagnostics, and monitoring the health of individual pods in the cluster.

Purpose

Logs tab serves as a centralized logging interface that enables users to trace events, warnings, and system activity for each pod within a cluster. It helps database administrators and Kubernetes operators quickly identify operational issues, such as misconfigured extensions, heartbeat failures, or missing components.

Key features

- **View Pod-Level Logs** Displays logs per pod, making it easy to isolate issues affecting specific parts of the cluster.
- **Filter by Time Range**

Use the date selector to filter logs within a specific time frame, improving focus during incident investigation.

- **Search & Filter Events** Easily locate relevant log entries by filtering on event type or keyword.
- **Log Level Indicators**

Quickly assess the severity of each event through labels like Info or Warn.

- **Detailed Error Descriptions**

Logs include detailed error messages such as missing extensions or failed queries, which help in root cause analysis.

The logs tab displays a detailed information of the logs with the following cloumns:

Column	Description
Timestamp	Indicates when the log event occurred (UTC).
Pod	The pod where the event originated.
Event	A brief description of the event, such as extension errors or heartbeat checks. Also provides the level of severity label like info, warn, etc. of the event.

Usability

- Verifying if monitoring extensions (like query_advisor, edb_stat_monitor) are active and correctly configured.
- Investigating errors related to missing or non-functional database extensions.
- Tracking regular system-level events such as pod heartbeats.
- Auditing recent changes or anomalies in cluster behavior.

Cluster Properties

The **Properties** tab provides a comprehensive overview of the configuration settings and resource allocations for the selected PostgreSQL cluster. This tab is essential for users who need to review the technical specifications or validate setup parameters before performing cluster operations such as scaling, editing, or troubleshooting.

Purpose

This tab serves as a single point of reference for all static properties related to a cluster, including compute resources, storage configuration, network access, and PostgreSQL version. It helps database administrators, support engineers, and developers:

- Understand the cluster's capacity and limitations
- Verify that the environment meets workload requirements
- Facilitate compliance and audit checks
- Identify configuration mismatches or deployment errors

The properties tab provides detailed information about cluster configuration settings with the following properties:

Property	Description
Cluster Type	Indicates whether the cluster is single-node or high-availability (HA).
Instances	Number of PostgreSQL instances deployed in the cluster.
PgType	Specifies the database type (e.g., PostgreSQL).
PostgreSQL Version	Version of PostgreSQL currently installed in the cluster.
Image	The container or VM image tag used for provisioning the cluster.
CPU Cores	Total number of virtual CPU cores allocated to the cluster.

continues on next page

Table 22 – continued from previous page

Property	Description
Memory	Total memory assigned to the cluster.
Primary Storage Class	The underlying storage class used by the persistent volume claim (e.g., *gp3*).
Primary Storage Size	Size of the storage volume attached to the primary instance.
Network Access	Indicates whether the cluster is accessible publicly or privately.
Connections	Describes the connection mode (e.g., *Read/write*).
Retention Period	Duration for which backups and logs are retained (e.g., 30 days).

Additional features

- **Edit Cluster**: Clicking the "Edit Cluster" link allows users to modify editable cluster properties such as resource allocation or network configuration. Some properties may require a restart or re-provisioning of the cluster.
- **Quick Actions**: Access context-aware options relevant to the cluster, such as scaling or deletion.

Health assessment reports

Health assessment reports provide a point-in-time snapshot of the performance and health of a PG cluster node over a given period. You can use them to identify anomalies, track how your system evolves over time, and share a consistent artifact with others.

Reports are generated automatically: one report per node immediately after a cluster is connected to Hybrid Manager (HM), then once every 24 hours thereafter. All nodes are tracked, including witness nodes — reports aren't scoped to a cluster or data group. Reports are stored persistently, so they remain available even after the underlying data used to generate them has been purged.

Report contents

Each report covers a fixed time window and includes aggregate statistics for that period, such as session counts, transaction stats, wait events, top SQL statements, WAL and buffer metrics.

Viewing and downloading reports

To access reports:

1. In the HM console, select **Reports**.

The **Reports** page lists all reports across your estate, sorted by date. You can filter by project, cluster, date range, and report interval.

2. To download a report, select the report row and select **Download Report**. Reports are available as PDF files.

You can also view reports scoped to a single cluster from the cluster detail page.

Comparing two reports

You can ask Chat Agent to compare two reports for the same cluster. This highlights the changes between two snapshots — useful for tracking how performance has changed over time.

To compare reports using Chat Agent:

1. Open **Chat Agent** in HM.
2. Ask Chat Agent to compare reports for a specific node and cluster — for example: *"Compare the last two reports for the primary node of cluster X."*

Chat Agent returns a summary of the differences between the two reports.

> **Note**
>
> To use the chatbot comparison feature, Chat Agent must be enabled in HM and the model must be configured as at least a Large (L) model. See *EDB Hybrid Manager Chat Agent* for the instructions.

5.3.3.3 Upgrading database clusters in Hybrid Manager

Upgrades are an important part of database administration for both major or minor upgrades of your Postgres database.

- *Postgres major version upgrades*
- *Postgres minor version upgrades in Hybrid Manager*

> **Tip**
>
> If you want to upgrade your **Hybrid Manager installation** instead, see *Upgrading Hybrid Manager* .

5.3.3.3.1 Postgres minor version upgrades in Hybrid Manager

Minor version upgrades use the same procedure for all cluster types and Postgres database distributions, including community PostgreSQL, EDB Postgres Extended Server, and EDB Postgres Advanced Server.

Minor version releases include bug fixes, security patches, and backward-compatible improvements. These are typically issued at least once every three months.

Minor releases of Postgres don't introduce new features or breaking changes, so we recommend that you adopt them as soon as they're released. Even if your Postgres cluster is running multiple minor versions behind, we recommend adopting the latest minor version available.

You have full control over when new versions become available in Hybrid Manager (HM). Once validated and published, new version images are available in HM's image library and are available for upgrades on any existing cluster.

> **Note**
>
> Postgres minor version upgrades require a data restart. Expect minimal downtime.

Although minor upgrades are ideally frequent and low risk, best practice is to always have up-to-date backups and a recovery plan in place before proceeding with the upgrade.

The prerequisites to get started are to have a cluster created with HM up and in a healthy state.

To ensure the cluster is in a healthy state, from the cluster's main page, select the **Monitoring** tab. Check the cluster and database metrics. Make sure everything is operating and all metrics are within the defined acceptable parameters.

Upgrade the Postgres minor version of an existing instance/cluster created with HM

1. From the cluster's main page, select **Quick Action > Upgrade Cluster**. Then select **Minor Upgrade**.
2. Select your preferred new minor version image.
3. Review the upgrade path. To confirm, select **Continue**.
4. Review the specific package changes expected with the upgrade. If everything looks correct, select **Upgrade Cluster**.

Use the **Monitoring** and **Logs** tabs to monitor the process and troubleshoot any unexpected behavior.

> **Note**
>
> On rare occasions, additional steps might be recommended after the minor upgrade is completed for certain product release. Consult the release notes to see if this is necessary in your case.

Rollback considerations

With this approach, no automatic rollback is available, and we don't recommend performing a downgrade. To undo the minor upgrade, restore from the latest backup in a new cluster running the preferred minor version and switch application workloads over to the new cluster. You can then delete the Postgres minor upgraded cluster or troubleshoot it further.

5.3.3.3.2 Postgres major version upgrades

Major version upgrades let you access new features, enhancements, and performance improvements introduced in the latest version of PostgreSQL. However, they may include changes that aren't backward-compatible, requiring careful planning and validation.

Major version upgrades typically occur annually and can bring significant advancements, such as new SQL functionality, enhanced performance, or improved compatibility with modern standards.

To ensure a smooth upgrade process, it's essential to test your applications and extensions against the new version and review the release notes for deprecated or modified features.

With Hybrid Manager (HM), you can manage major version upgrades at your discretion. Once a new major version has been validated and published, it soon becomes available in HM's image library, enabling controlled upgrades for existing clusters.

> **Note**
>
> Postgres major version upgrades require a data restart. Expect downtime.

Before proceeding, we strongly recommended that you perform a full backup of your cluster and validate a comprehensive recovery plan. This ensures you can restore your data if any compatibility or operational issues arise during the upgrade process.

Major version upgrades by database cluster type

You can upgrade Postgres major versions on any supported cluster type:

- You can upgrade single-node and high-availability clusters using the in-place, offline *upgrade procedure* .
- Advanced high-availability (AHA) and distributed high-availability (DHA) clusters support an in-place, online *upgrade procedure* .
- Alternatively, DHA clusters also support zero-downtime major upgrades using a *blue-green deployment approach* .

Rollback considerations

No automatic rollback is available for major version upgrades, and we don't recommend performing a downgrade. To undo the major upgrade, restore from the latest backup in a new cluster running the preferred major version and switch application workloads over to the new cluster. You can then delete the Postgres major upgraded cluster or troubleshoot it further.

Postgres major version upgrades on Hybrid Manager single-node or high-availability clusters

Prerequisites

- Have a cluster created with Hybrid Manager (HM) up and in a healthy state. To ensure the cluster is in a healthy state, from the cluster's main page, select the **Monitoring** tab. Check the cluster and database metrics. Make sure everything is operating and all metrics are within the defined acceptable parameters.
- We strongly recommend that you perform a full backup of your cluster and validate a comprehensive recovery plan before proceeding.

Upgrade the Postgres major version of an existing cluster created with HM

1. From the cluster's main page, select **Quick Actions > Upgrade Cluster**. Then choose **Major Upgrade**.
2. Select your preferred new major version image.
3. Review the upgrade path. Select **Continue**.
4. Review the specific package changes expected with the upgrade. If everything looks correct, select **Upgrade Cluster**.
5. Use the **Monitoring** and **Logs** tabs to monitor the process and troubleshoot any unexpected behavior.

> **Note**
>
> On rare occasions, additional steps might be recommended after the major upgrade is completed for certain product releases. Consult the release notes to see if this is necessary in your case.

Postgres major version upgrades on Hybrid Manager for advanced high availability or distributed high availability

Upgrade considerations

- **Upgrade skips:** Direct major version skips aren't supported. All upgrades must proceed sequentially, one major version at a time.

Correct: 16 > 17 > 18

Incorrect: 16 > 18

- **Image tag consistency:** Maintain consistency in image tags. For example, upgrade from `<old-image-version>-<distribution>_full` to `<new-image-version>-<distribution>_full` . Failure to do so may result in an altered set of packaged extensions.
- **Physical join:** While advanced high-availability (AHA) and distributed high-availability (DHA) clusters use logical replication for day-to-day data synchronization, the process of a major version upgrade involves a *physical join* (physical replication) to facilitate the version change between data groups. This allows data groups to be upgraded in place, where they temporarily run different major versions. However, you can't add a new data group with a different major version.

Prerequisites

- Have a cluster created with Hybrid Manager (HM) up and in a healthy state. To ensure the cluster is in a healthy state, from the cluster's main page, select the **Monitoring** tab. Check the cluster and database metrics. Make sure everything is operating and all metrics are within the defined acceptable parameters.
- We strongly recommend that you perform a full backup of your cluster and validate a comprehensive recovery plan before proceeding.

Upgrade the Postgres major version of an existing cluster created with HM

For major upgrades of AHA or DHA clusters (those with multiple data groups), you must upgrade all data and witness groups sequentially. While the order of groups isn't critical, ensure each group's upgrade completes entirely before starting the next.

1. From the cluster's main page, next to the first data group you want to upgrade, from the ellipsis menu, select **Upgrade Data Group**. Then select **Major Upgrade**.
2. Select your preferred new major version image.
3. Review the upgrade path. Select **Continue**.
4. Review the specific package changes expected with the upgrade. If everything looks correct, select **Upgrade Cluster**.

5. After the upgrade is applied, you can use the PGD CLI to ensure

   the data group becomes healthy. The upgrade is successful when all checks have the Status `Ok` .

6. After the first data group is successfully upgraded and healthy, upgrade the remaining data and witness groups one by one until the entire cluster is on the new version.

> **Note**
>
> On rare occasions, additional steps might be recommended after the major upgrade is completed for certain product releases. Consult the release notes to see if this is necessary in your case.

Seamless major upgrades with blue-green deployments

A blue-green approach for major upgrades

Reliable, safe, and instant major upgrades are possible with distributed high-availability (DHA) clusters. The method consists of using a blue-green deployment approach for your distributed database based on different data groups:

- The existing *blue* data group represents the current stable environment (for example, your production database).
- A new *green* data group is created in the database cluster and upgraded to the next major version.
- You perform comprehensive tests in your application by connecting to the green data group. (A new connection string is available.)
- You redirect all client workloads to the green data group.
- You delete the blue data group after the upgrade is fully validated.

Prerequisites

- Have a DHA cluster with the old version (blue) data group on Hybrid Manager (HM) up and running in a healthy state.
- Have the next Postgres major version available in the asset library.

General recommendations

- Always perform a full backup before starting the upgrade process.
- If feasible, test your upgrade process in a preproduction environment.
- We recommend completing the upgrade process in a short period of time (for example, the same day). Running different major versions in data groups can lead to unexpected incompatibility issues.

Check the cluster status

To ensure the cluster is in a healthy state, from the cluster main page, select the **Health Status** tab. Then check the cluster's:

- **Raft Status**. Make sure it reads **OK**.
- **Replication Slot Status**. Make sure it reads **OK**.
- **Clock Skew**. Make sure it reads **OK**.
- **Proxy Status**. Make sure all the proxies are up.
- **Node Status**. Make sure all the nodes are up.
- **Transaction Rate**. Make sure it's within acceptable/expected parameters.

You can explore more details about each data group and node in the data group including:

- The data group's **Proxy Status**
- The data group's **Node Status**
- The following details of each node in the group:
- Number of connections
- WAL size
- Memory percentage used
- Storage percentage used

You can look at the latest **Replication Status** details to see the replication lag between nodes.

To ensure all data is being synchronized as expected, you can use the PGD CLI to view data group state or perform deeper monitoring on the replication peers.

If any of these metrics aren't within acceptable/expected parameters, address these issues before proceeding with the upgrade process.

Edit the cluster

To begin the upgrade process, from the **Quick Actions** menu in the upper-right corner, select **Edit Cluster**. Then:

1. Select the **Cluster Settings** tab and set your password again.
2. Select the **Data Groups** tab, find your old version (blue) data group, and select **Duplicate**. This brings up a new data group below the ones you already have.

You can change the name of the group from the default, change the **Nodes** architecture (two to three nodes or three to two nodes), as well as the **Deployment Location**, **Database Type**, **Instance Size**, **Storage**, **Networking**, **Backups**, and **Security** settings. However, in most cases using this upgrade process, you want to leave everything but possibly the **Group Name** the same, since you are trying to replace your old version data group with another new version data group that can handle the same load. However, if you do need to change something for a particular circumstance, you can do that at this point in the upgrade process.

3. Select **Create Data Group**. If you had only an odd number of data groups, a witness group with the default name **Witness Group** is added to your set of data groups. The witness group serves to keep Raft consensus possible.

This is necessary with two data groups because there could be what's known as a "split brain" scenario with an even number of data groups. In this case, half of the groups want to make one decision and the other half want to do something else. The witness group prevents such a scenario by being the tie breaker whenever this happens.

4. Select **Save** to save the witness group's configuration.

You can see both your old (blue) data groups, new (green) data group, and witness group in the **Data Groups** tab.

5. On the **Data Groups** tab, select **Save**.

The **Confirm Changes** pop-up lists the changes to be made to the cluster and a warning stating that enacting the changes could trigger database restarts across the cluster. While this is the case, it doesn't mean that you incur downtime from enacting the changes. At least one node in your old version data group stays active during this time.

6. Select **Confirm Changes**. The changes can take quite some time depending on your underlying hardware and database size.

Verify the data in the green data group

Ensure the entire cluster is in *Check the cluster status* .

From the **Overview** tab, you can copy the connection string for your new data group. Each data group has a different connection string exposed so that any applications connected to the existing data group aren't impacted until the process is complete. You can use the new connection string to perform tests such as validating that all data is synchronized.

Upgrade the green data group to the next major version

1. From the **Overview** tab, identify the new data group. From the ellipsis menu, select **Upgrade data group**.
2. From the **Software Image** list, select the target version. We recommend upgrading to the latest minor available for the next major version.
3. Select **Continue**. You see the details of the upgrade.
4. Select **Upgrade Cluster** to start the data group upgrade. Wait a few minutes until the process is complete.

Verify the upgrade

The **Overview** and **Properties** tabs show the new version for the data group. Ensure the cluster is in *Check the cluster status* .

Plan your tests

We strongly recommend that you rigorously test applications and clients using the new upgraded version before switching them from your blue environment.

If you require write operations or schema changes to perform the tests, those are then replicated to the old version data group. To manage this, you can modify the replication configuration of your cluster by enabling or disabling the replication subscriptions .

For example, you can use the following SQL command to disable changes in your new upgraded version data group to your old version data groups. The SQL command, which must be executed when connected to a node in the old version data group, disables the subscriptions replicating changes from the new data group.

```
SELECT bdr.alter_subscription_disable(sub_name, true, true)
    FROM bdr.subscription s, bdr.node n, bdr.node_group ng
  WHERE n.node_group_id = ng.node_group_id
    AND s.source_node_id = n.node_id
    AND ng.node_group_name = <new version data group name>;
```

Plan the switchover

After you've performed all tests, you're ready to switch workloads to the green data group using the new connection string. This might require changes in your applications, clients, or infrastructure services.

If you previously disabled the two-way synchronization between the new and old version data groups, you need to reenable it:

```
SELECT bdr.alter_subscription_enable(sub_name, true)
    FROM bdr.subscription s, bdr.node n, bdr.node_group ng
  WHERE n.node_group_id = ng.node_group_id
    AND s.source_node_id = n.node_id
    AND ng.node_group_name = <old version data group name>;
```

Ensure the cluster is in *Check the cluster status* .

Remove the blue data groups

The last step in the upgrade procedure is to remove any old version data groups. In the **Overview** tab from the cluster page, from the ellipsis menu for the old version data group you want to remove, select **Delete Data Group**.

Confirm the deletion. Remove any other old version data groups you have or the witness data group if you don't need it. The upgrade is then complete.

Ensure the cluster is in *Check the cluster status* .

Rollback considerations

With this approach, after you delete the old version data group, no automated rollback is available. To undo the major upgrade, consider restoring the latest backup in a new data group running the desired major version and switching application and client workloads to that new data group.

5.3.3.4 Migrating databases with Hybrid Manager

Hybrid Manager (HM) provides an integrated solution to perform migrations from external databases into your environment. This capability allows you to bring your existing data, schemas, and applications into a centralized management hub, facilitating easier administration and monitoring.

For data migration, the migration process primarily leverages the EDB Data Migration Service (DMS). It works with other integrations like the Migration Portal and AI Copilot to offer a comprehensive workflow for assessing, converting, and migrating your databases.

5.3.3.4.1 How do I use migration capabilities in HM?

You perform the migration process using a series of steps that connect your source database and your destination database to the HM console. To understand the workflow, see *Agent Factory Architecture on Hybrid Manager* .

5.3.3.4.2 What kind of migrations does the EDB Data Migration Service support?

The EDB DMS supports migrations from external databases into HM-managed databases, from external databases to other external databases, from HM-managed databases to external databases, and more. To see end-to-end examples of possible migration paths, see *Choosing your migration path* .

For supported source and destination databases, see the *Supported databases and operating systems* .

For supported migration methods, see *Migration types* .

5.3.3.4.3 What kind of technologies does the EDB Data Migration Service use?

DMS is built on Apache Kafka and the open-source Debezium CDC platform. Using change data capture (CDC) and event streaming, source database row changes are replicated to the migration destination.

Planning

Migration architecture

Hybrid Manager (HM) provides a unified platform for migrating databases from various sources to HM-managed and externally managed Postgres clusters. The migration process relies on several key components that work together to handle schema conversion, data transfer, and continuous replication.

From Oracle

Fig. 5: Migrations from Oracle databases

Migrations from Oracle are best suited for targeting EDB Postgres Advanced Server clusters (HM-managed or externally managed). Given the differences between the two database systems, these migrations require a specific set of components to handle both schema and data migration, with EDB Postgres Advanced Server being optimized for this purpose.

Oracle database: The source database containing the data and schemas to be migrated.

Agent: A component that connects to the Oracle source to read and extract schemas for migration. Also, registering the source with HM provides a view of the source database in the HM console for migration assessment and to enable schema migration.

DMS agent in reader mode: A component that connects to the Oracle source to read and extract data for migration.

Hybrid Manager (HM): The central control plane.

HM console: The web-based interface where users manage and monitor all migration activities.

Migration Portal: A service in HM that handles Oracle-specific schema assessment, conversion, and import to the destination database, while ensuring compatibility with EDB Postgres Advanced Server. It can also assist with schema-related issues using an AI Copilot.

Data Migration Service (DMS): The core service that performs the actual data transfer (snapshot) and continuous change replication (streaming). It includes a built-in writer for migrations to HM-managed databases.

HM-managed EDB Postgres Advanced Server database: A destination database cluster that's created and managed by HM.

DMS agent in writer mode: A component that you require only when the migration destination database is a self-managed database, external to HM. It consumes CDC records to replicate and apply changes on the destination database.

External EDB Postgres Advanced Server database: An externally managed destination for migrations, also called self-managed databases. This scenario requires an additional DMS agent component in writer mode that you connect to the destination host to receive and apply data.

See Oracle to HM-managed database for an end-to-end example that shows how to perform a migration.

From Postgres

Fig. 6: Migrations from Postgres databases

Migrating between Postgres databases is a more streamlined process, as the schema is already compatible. This migration is primarily focused on efficient data transfer and replication and also requires a specific set of components for migration:

External Postgres database: The source Postgres database containing the data to be migrated.

Migration Toolkit or pg_dump: External tools used for the initial schema migration, as the schema itself doesn't require conversion.

Agent: A component that registers the source with HM to provide a view of the source database within the HM console for monitoring.

DMS agent in reader mode: A component that connects to the source Postgres to read and extract data.

Hybrid Manager (HM): The central control plane.

HM console: The web-based interface where users manage and monitor all migration activities.

Data Migration Service (DMS): The core service that performs the actual data transfer (snapshot) and continuous change replication (streaming). It includes a built-in writer for migrations to HM-managed databases.

HM-managed Postgres database: A destination database cluster that's created and managed by HM. It can be PostgreSQL, EDB Postgres Advanced Server, or EDB Postgres Extended Server.

DMS agent in writer mode: A component that you require only when the migration destination database is a self-managed database, external to HM. It consumes CDC records to replicate and apply changes on the destination database.

External Postgres database: An externally managed destination for migrations, also called self-managed databases. This scenario requires an additional DMS agent component in write mode that you connect to the destination host to receive and apply data.

See Self-managed Postgres to HM-managed database for an end-to-end example that shows how to perform a migration.

Terminology

This terminology is important to understand EDB Data Migration Service (DMS) functionality.

Change data capture (CDC)

CDC is a set of software design patterns used to determine and track changes in data sets by calculating *deltas*. EDB DMS uses CDC to stream changes from one database cluster to another.

EDB DMS agent

The DMS agent is a tool that can register databases for migration. In reader mode, it registers databases as a source for migrations. In writer mode, it registers them as a destination.

After you register the source and destination databases, the DMS agent works with DMS to perform data migrations.

> **Note**
>
> In Hybrid Manager (HM) version 1.3 and later, the *Installing EDB Data Migration Service agent on Linux* , packaged as *cdcagent* , replaces legacy DMS Reader and DMS Writer.

To ensure compatibility, use the DMS agent on all HM 1.3 and later deployments. For HM deployments on version 1.2 and earlier, use the separately packaged EDB DMS Reader and EDB DMS Writer.

EDB DMS agent in reader mode

When running in reader mode, the DMS agent uses Debezium to perform CDC operations on the source database and Kafka/Object storage to produce messages containing the change events.

Install the DMS agent on your source machine. Alternatively, if your source database runs on a machine managed by a cloud provider that you don't have admin access to, you can also install it on an auxiliary machine that has access to your source database.

Then configure and launch it in reader mode when performing any type of migrations, as this mode allows you to connect to and read the source.

EDB DMS agent in writer mode

When running in writer mode, the DMS agent consumes CDC records from Kafka/Object storage to replicate and apply changes on the destination database.

If the destination for your migration is an HM-managed database, you don't need to install and launch the EDB DMS agent in writer mode because the DMS component that comes with HM already includes a built-in writer.

If the destination is a self-managed database, install an additional DMS agent on your destination machine. Then configure and launch it in writer mode as that mode allows you to connect to and migrate data to self-managed destinations.

EDB Postgres AI agent

The *Setting up EDB Postgres AI agent to monitor AWS RDS deployments on HM* , packaged as `beacon-agent` , and formerly known as HM or beacon agent, is a tool that provides database observability, monitoring, usage reporting, and more. In the context of migrations, the EDB Postgres AI agent performs a migration assessment for each database you register with it. It also performs schema extraction for subsequent schema migration — for Oracle sources, this feeds the *Enhancements to the Migration Portal integration* ; for Postgres sources, it enables the unified schema migration workflow when you select the **Schema and data** or **Schema only** scope.

Auxiliary machine

An auxiliary machine is a Linux-based server (bare metal, VM, or local) dedicated to running the EDB DMS agent for database migrations. This machine acts as the migration engine, capable of connecting to various source and destination databases, whether local or remote.

This machine can be the same host as where you have a source or destination database and the same host where you have the HM running. The only requirement is a Linux operating system, since that's the only platform currently supporting the EDB DMS agent.

Migration types

When *creating migrations* , you can choose whether to perform the migration using the **Snapshot** or **Snapshot + Streaming** option.

Snapshot migration

When you select the **Snapshot** option, DMS captures an image of your database at a specific moment. This point-in-time copy is then used for the migration. Any new inserts and updates performed on the source database after the snapshot is taken aren't replicated to the destination database.

Snapshot + Streaming migration

When you select the **Snapshot + Streaming** option, DMS begins with a snapshot to establish a baseline copy and then constantly employs CDC to apply subsequent changes. This approach ensures data consistency during migration. DMS continues to stream any data updates performed on the source database to the destination database until you *mark the migration as completed* .

Migration scope

When creating a migration, you select a scope that determines whether the migration includes schema, data, or both.

Data only

Migrates data only. Use this option when you have already imported the schema to the destination database manually. The schema must be in place before the migration begins.

Schema and data

Migrates schema and data in a single managed workflow. HM imports the schema automatically from the Migration Portal project created when you registered your source database, handles constraint ordering during migration, and reapplies deferred constraints after the migration completes. This scope requires the source database to be registered with the EDB Postgres AI agent.

Schema only

Migrates schema only, without transferring data. Useful for setting up the destination database structure before running a data-only migration.

Migration destination types

When *creating migrations* , you can choose whether to perform a migration to managed Postgres or to external Postgres.

Managed Postgres

Managed Postgres destinations are databases created and managed through HM. You can select any of the database clusters created in the project where you're performing the migration as possible migration destinations.

External Postgres

External Postgres destinations are databases created outside of HM that you fully manage. To choose an external Postgres database as a migration destination, you must first install and configure the *Configuring and running the EDB DMS agent* and configure it in writer mode to allow the DMS to connect to the self-managed database and perform write operations on it.

Apache Kafka

Apache Kafka is an open-source, distributed-event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. EDB DMS uses Kafka to manage data streaming from the source to the destination database.

Debezium

Debezium is a Java-based, open-source platform for CDC that's supported by the Red Hat community. When running in reader mode, the EDB DMS agent uses Debezium to perform reading operations and capture data changes on the source database.

Legacy terminology

EDB DMS Reader

In HM versions 1.2 and earlier, EDB Data Migration Service Reader, packaged as `cdcreader` , is a tool that uses Debezium to perform CDC operations on the source database and Kafka to produce messages containing the change events.

Install and configure DMS Reader when performing any type of migrations, as it allows you to connect to and read the source. If the destination for your migration is an HM-managed database, you don't need to install *EDB DMS Writer* . If the destination is another self-managed database, install *EDB DMS Writer* .

EDB DMS Writer

In HM versions 1.2 and earlier, EDB Data Migration Service Writer, packaged as `cdcwriter` , is a tool that uses CDC records to replicate and apply changes on the destination database.

Install and configure DMS Writer on your destination machine when performing migrations to self-managed destination databases, as it allows you to connect to self-managed destinations. Also install *EDB DMS Reader* for any type of migrations.

Migration features

Hybrid Manager (HM) provides a comprehensive, centralized solution for database migrations, offering a wide range of features designed to simplify and accelerate your data transfer process.

Sovereign migration solution

Perform migrations from a single platform deployed entirely within your on-premises or cloud environments. This approach provides you with complete control over your data and infrastructure, ensuring your migrations are sovereign and secure within your own network.

Migration assessment

HM leverages the EDB Postgres AI agent to generate an assessment of the complexity and estimated effort (LOE) required for Oracle database migrations. By analyzing the registered source database, the platform provides early insights into potential issues and the scale of the migration, helping you plan effectively.

Bulk migration assessment

When you have a large database estate to evaluate, the EDB Postgres AI agent's `configure-onprem-dbs` command lets you enroll many databases for migration assessment in a single step. Instead of manually constructing a connection

string for each database, you can import your inventory from a CSV file, an Oracle `tnsnames.ora` file, or discover databases directly from AWS RDS. See *Assessing multiple databases at once* .

Database and schema management

Access detailed information about your source database through the database details page. There, you'll find information about schemas, as well as a clear breakdown of migration assessment factors.

Migration Portal with AI Copilot

The Migration Portal is a powerful tool for Oracle-to-Postgres migrations. It automates the assessment of schemas, flags and incompatibilities. It also provides an integrated AI Copilot to help you quickly identify and resolve complex schema issues, streamlining the conversion process. Originally a web tool hosted by EDB, the Migration Portal has been integrated into HM. This integration deploys the service in your environment, supporting the sovereignty of your deployment.

Flexible schema and data migration options

Migration scope

HM supports automated schema migration as part of the data migration workflow for both Oracle and Postgres sources. When creating a migration, you can choose to migrate **Schema and data**, **Data only**, or **Schema only**, giving you full control over what gets transferred.

Migration method

Choose the migration method that best suits your needs:

- **Snapshot** — A fast, one-time data transfer for bulk migrations.
- **Snapshot and streaming** — Suitable for migrations that require minimal downtime. It uses change data capture (CDC) to continuously capture and replicate ongoing changes from the source to the destination database.

Diverse migration destinations

The solution offers flexibility in choosing your destination. You can migrate to databases provisioned in HM (HM-managed databases) or to externally managed Postgres databases (externally managed databases).

Granular migration control

Maintain precise control over your data migration with the ability to select specific database tables and columns to migrate. This feature allows you to migrate only the data you need, reducing migration time and resource usage.

Schema and column mapping

The migration service allows for flexible schema mapping, enabling you to migrate data to tables and columns with different names in the destination database. This feature is particularly useful for restructuring data during the migration process.

Supported databases and operating systems

The EDB Data Migration Service (EDB DMS) is available only as part of Hybrid Manager (HM).

Supported source databases

The following database versions are supported.

Database version	Supported source
PostgreSQL 12 - 18	Y
EDB Postgres Advanced Server 12 - 18	Y
EDB Postgres Extended Server 12 - 18	Y
Oracle 19c	Y
Oracle 21c	Y
Oracle 23ai	Y
AWS RDS PostgreSQL 13-17	Y
AWS RDS Aurora (PostgreSQL compatible) 13-17	Y

The supported Postgres source databases include only externally managed (self-managed) databases.

Note

The DMS agent supports migrating Oracle 21c and 23ai databases. However, Migration Portal doesn't. This means that to migrate 21c and 23ai databases, you must manually perform the schema migration.

Oracle

The EDB DMS stack requires the Oracle database to have archive log mode enabled and supplemental logging data enabled at the table and database level. For details, see *Preparing Oracle source database* .

Container databases (CDB/PDB) and non-CDB sources are supported.

Postgres

When migrating from Postgres toward Postgres, cross-version and cross-distribution migrations are supported. You can migrate data from one version of PostgreSQL to another, such as from PostgreSQL 16 to PostgreSQL 17 or from PostgreSQL 17 to EDB Postgres Advanced Server 17. The exception is that you can't migrate from EDB Postgres Advanced Server as a source to a different distribution, for example, to PostgreSQL or EDB Postgres Extended Server.

Source Postgres instances can be self-managed instances on bare metal machines, on virtual machines, or hosted by a cloud service provider like AWS (RDS or Aurora databases).

Postgres sources require a database role or user that can manage replications. For details, see *Preparing your AWS RDS/Aurora PostgreSQL source database* .

Supported destination databases

DMS supports migrating to both HM-managed databases (on clusters created with HM) and self-managed databases running Postgres.

Database version	Supported destination
PostgreSQL 11 - 18	Y
EDB Postgres Advanced Server 11 - 18	Y
EDB Postgres Extended Server 11 - 18	Y

> **Note**
>
> For HM-managed database clusters, support is only for versions 16 and 17.

DMS supports migrating to all databases listed in the table. However, Migration Portal supports schema migrations only for EDB Postgres Advanced Server versions 16 and 17. For version 15 and earlier, you must perform schema migration manually.

Oracle

EDB DMS doesn't yet support migrations to Oracle destinations.

Postgres

When migrating from Postgres toward Postgres, cross-version and cross-distribution migrations are supported, for example from PostgreSQL 16 to 17 or from PostgreSQL 17 to EDB Postgres Advanced Server 17.

Postgres destinations require a database role or user that can manage replications. For details, see *Preparing your HM-managed destination database* .

Supported operating systems

The DMS agent runs on Linux. For details, see Installing EDB DMS agent .

Known issues, limitations, and notes

These are the known issues, limitations, and notes for:

- *Data migration limitations*
- *Oracle limitations*
- *Postgres limitations*
- *Schema ingestion known issues*
- *HM Migration Portal known issues*
- *Other known issues*

Data migration limitations

Case sensitivity in object names

> **Tip**
>
> Resolved in HM versions 2025.11 and later.

EDB DMS doesn't currently support migrating schemas, tables, and columns that have case-sensitive names.

Upgraded destinations clusters

You can't use clusters provisioned in Hybrid Manager (HM) versions 1.2 and earlier as migration destinations after you upgrade the HM instance to version 1.3 or later. You must create a new cluster in the upgraded environment to use it as a valid migration destination.

Considerations and limitations for multiple migrations to a single destination

The DMS supports running multiple data migrations from one source to one destination database, but certain restrictions apply.

- **Table usage**: You can actively use each destination table only in a single migration at any given time.
- **Unique resourceID (applies to migrations to external destinations)**: When running multiple migrations to an external destination, you must assign a unique resourceID to each instance of the EDB DMS agent in writer mode with the `DBCONFIG_DATABASES_0__RESOURCEID=${Resource_ID}` parameter. Setting this parameter prevents concurrent migrations from attempting to use the same table simultaneously, which can cause data loss or duplication.

> **Note**
>
> This resourceID requirement doesn't apply to migrations targeting HM-managed Postgres instances. !!!

- **Performance planning (applies to migrations to external destinations)**: While performance generally remains strong with the *System specifications* , if you run five or more migrations to the same external destination database, to prevent resource contention, consider allocating additional CPU, memory, and volume IOPS/throughput to the destination. This is especially important if the destination is an external database.
- **Resource management**: Each migration job requires a dedicated, non-concurrent use of the necessary DMS agent components.

DMS agent in reader mode (source): Don't use a single running instance of the EDB DMS agent in reader mode for more than one concurrent migration from the same source database. The HM console removes a source database from the available list once it's active in a migration. However, you can use a single DMS agent in reader mode for subsequent migrations or for concurrent migrations from different source databases.

DMS agent in writer mode (external destination): Each migration requires its own instance of the DMS agent in writer mode, even if the migration shares the same destination as another migration.

Postgres-specific schema object types aren't migrated by DMS at schema migration time

> **Tip**
>
> Resolved in HM versions 2026.5 and later.

Description: Postgres schema migration support is limited to simple databases with tables only. DMS is currently unable to migrate some object types and has issues with the order in which objects are migrated.

Workaround: Migrate the schema manually and use DMS in data-only migration mode.

Oracle limitations

Unsupported Oracle data types

A limited number of Oracle data types and features aren't supported by EDB DMS.

See the Debezium documentation for detailed comments on supported data types.

Unsupported Oracle data types include:

- BFILE
- LONG
- LONG RAW
- RAW
- UROWID
- User-defined types (REF, Varrays, Nested Tables)
- ANYDATA
- XMLTYPE
- Spatial

EDB DMS supports replicating Oracle tables that contain BLOB, CLOB, or NCLOB columns only if these also have the primary key constraint. If the tables don't have the primary key constraint, the streaming replication supports only INSERT operations.

`BINARY_FLOAT` and `BINARY_DOUBLE` types in Oracle that might contain `Nan` , `+INF` , and `-INF` values aren't supported by EDB DMS.

EDB DMS doesn't support values containing a null character (0x00) for text field types like `VARCHAR` and `CLOB` in Oracle.

Oracle 21c databases

The DMS agent supports migrating Oracle 21c databases. However, Migration Portal doesn't. This means that to migrate 21c databases, you must manually perform the schema migration.

Oracle XE compatibility restrictions

Oracle XE (Express Edition) does not support enabling supplemental logging at the database level. Because this is a core requirement for Change Data Capture (CDC), EDB DMS is not fully compatible with Oracle XE as a source database.

- Expected errors : When using Oracle XE, the DMS Agent (reader) will typically fail and log exceptions similar to - Debezium consumer $NAME stopped with exception: an exception occurred in the change event producer. This connector will be stopped.
- Limited functionality : Only snapshot-only migrations are expected to function. Even during successful snapshots, Debezium error messages may still appear in the logs.
- Usage policy : Snapshot-only migrations on Oracle XE are suitable only for testing or evaluation purposes. This configuration is not supported for production use cases.

Unqualified object references in migrated PL/SQL bodies

Description: When DMS translates Oracle triggers, procedures, functions, and packages to EDB Postgres Advanced Server (EPAS), the translated PL/SQL bodies retain unqualified object references from the Oracle source.

Oracle and EPAS resolve unqualified names differently:

- **Oracle**: Unqualified references inside a PL/SQL body resolve to the definer's schema (the schema that owns the object), regardless of who invokes it.
- **EPAS**: Unqualified references resolve at execution time using the caller's `search_path`. If the calling session's `search_path` doesn't include the migrated schema, references fail at runtime.

The schema migration itself succeeds because the DMS agent sets `search_path` explicitly when executing each DDL statement. The failure surfaces after migration, when applications or users invoke migrated PL/SQL objects from sessions with a different `search_path`.

This limitation applies only to Oracle to EPAS migrations with a migration scope of **Schema only** or **Schema and data**. It doesn't affect **Data only** migrations, where schema objects are created by the customer before migration runs and DMS doesn't generate or translate any PL/SQL bodies.

Example:

This Oracle trigger references `test_orders_seq` without a schema prefix:

```
CREATE TRIGGER "DEBEZIUM"."TEST_ORDERS_TRG"
    BEFORE INSERT ON test_orders_legacy
    FOR EACH ROW
BEGIN
    IF :NEW.order_id IS NULL THEN
        SELECT test_orders_seq.NEXTVAL INTO :NEW.order_id FROM dual;
    END IF;
END;
```

The DDL produced by DMS on EPAS preserves the unqualified reference:

```
CREATE OR REPLACE TRIGGER TEST_ORDERS_TRG
    BEFORE INSERT ON test_orders_legacy
    FOR EACH ROW
BEGIN
    IF :NEW.order_id IS NULL THEN
        SELECT test_orders_seq.NEXTVAL INTO :NEW.order_id FROM dual;
    END IF;
END;
```

After migration, behavior varies by caller:

- Session with `search_path = debezium`: trigger executes successfully.
- Session with default `search_path = "$user", public`: fails with `sequence "test_orders_seq" does not exist`.

Workarounds:

Choose one of the following approaches based on your environment:

- **Schema-qualify the DDL (recommended)**: Before cutover, review all translated trigger, procedure, function, and package bodies and prefix unqualified object references with the source schema name. Apply the modified DDL to the destination. This approach removes the runtime dependency on `search_path`.

The following fix schema-qualifies the sequence reference inside the trigger body. The table reference in the trigger header is also qualified to make the object ownership explicit:

```
CREATE OR REPLACE TRIGGER TEST_ORDERS_TRG
    BEFORE INSERT ON debezium.test_orders_legacy
    FOR EACH ROW
BEGIN
    IF :NEW.order_id IS NULL THEN
        SELECT debezium.test_orders_seq.NEXTVAL INTO :NEW.order_id FROM dual;
    END IF;
END;
```

- **Set a default ``search_path`` for the role or database**: Set a persistent default so new sessions automatically include the migrated schema in `search_path` :

```
ALTER ROLE app_user SET search_path = debezium, public;
-- or, database-wide:
ALTER DATABASE mydb SET search_path = debezium, public;
```

- **Set ``search_path`` at connection time**: Configure all application connections to include the migrated schema in `search_path` , either in the connection string (`options=-c search_path=debezium,public`) or as the first statement after connecting:

```
SET search_path = debezium, public;
```

Ad-hoc sessions that connect without an explicit `search_path` (DBA sessions, monitoring tools, or BI clients) remain at risk of failures regardless of which workaround you choose.

Postgres limitations

Unsupported domain type definitions in columns

> **Tip**
>
> Partially resolved in HM versions 2026.4 and later.

The EDB DMS doesn't support migrating tables with columns that have user-defined domains as data types for the following data type domains:

- DATE (Support added in Innovation Release HM 2026.4)
- TIME (Support added in Innovation Release HM 2026.4)
- TIMESTAMP (Support added in Innovation Release HM 2026.4)
- INTERVAL (Support added in Innovation Release HM 2026.4)
- UUID (Support added in Innovation Release HM 2026.4)
- ENUM
- JSON (Support added in Innovation Release HM 2026.4)
- XML (Support added in Innovation Release HM 2026.4)
- POINT (Support added in Innovation Release HM 2026.4)
- LTREE

- TSVECTOR

TSVECTOR type migration workaround

Although we don't support migrating `TSVECTOR` type columns, you can apply this workaround to enable their migration. For example, this `messages` table in source database contains a `TSVECTOR` column:

```
CREATE TABLE messages (
    title       text,
    body        text,
    tsv         tsvector
);
```

Before starting the migration, create a table in the destination database without the `TSVECTOR` column:

```
CREATE TABLE messages (
    title       text,
    body        text,
);
```

When you perform the migration, you must exclude the `TSVECTOR` column in the HM console when you review the tables on the **Review your Tables** page by clearing the `tsv` column check box.

After the migration finishes, the `title` and `body` columns are migrated to the target database. Now you can add the `TSVECTOR` column to the target table:

```
ALTER TABLE messages ADD COLUMN tsv TSVECTOR DEFAULT NULL;
```

Then run the following `UPDATE` statement to populate the column for all migrated data:

```
UPDATE messages SET tsv = to_tsvector(english, CONCAT(title,  , body)) WHERE tsv IS␣
↪NULL;
```

Extension migration limitations

When migrating between any Postgres distributions, the EDB Postgres AI agent extracts all extensions installed on the source database and applies them to the destination as part of schema migration. The agent extracts whatever is installed on the source with no allowlist or denylist. The following limitations apply.

- **Per-database scope**: The agent extracts extensions on a per-database basis. Extensions installed in other databases on the same Postgres server aren't included. To migrate extensions from multiple databases, run one migration per database.
- **Version pinning**: The agent installs each extension at exactly the version found on the source. If that exact version isn't available on the destination, schema migration fails. When migrating between different Postgres versions or distributions, extension behavior may vary unexpectedly even if the version is available — extension compatibility across versions isn't guaranteed.
- **FDW object migration**: The agent installs the foreign data wrapper (FDW) extension itself (for example, `postgres_fdw` or `oracle_fdw`), but doesn't migrate FDW objects. Foreign servers, user mappings, and foreign tables are stripped and must be recreated manually on the destination after migration.
- **Extensions requiring superuser**: Extensions that require superuser to install fail silently at apply time on managed Postgres services (such as Amazon RDS or Aurora) that don't grant superuser. The migration assessment flags these extensions as `RequiresSuperuser` , but this doesn't prevent the migration from proceeding.

- **Fixed-schema extensions**: Some extensions can only be installed into a specific schema and can't be placed into a different one. If the source has such an extension in a non-default schema, the agent can't replicate that schema placement on the destination.
- **Extension superuser display on Postgres 10 to 12**: On Postgres 10 to 12, the migration assessment always shows extensions as requiring superuser to install, even if they don't. This is a display-only limitation with no impact on the migration itself.

Schema ingestion known issues

Reserved schema names for Postgres DDL extraction

The EDB Postgres AI agent uses two synthetic schema names internally during Postgres DDL extraction:

- `__extensions__` — stores `CREATE EXTENSION` DDL statements
- `__globals__` — stores global objects (roles, permissions, and memberships from `pg_dumpall`)

These names are reserved and the agent rejects any user schema that conflicts with them. The double-underscore convention is intentionally chosen to avoid collision with real schema names, so this is unlikely to affect most databases.

Additional schemas ingested for EDB Postgres Advanced Server sources

> **Tip**
>
> Resolved in HM versions 2025.11 and later.

When viewing the list of schemas for a source EDB Postgres Advanced Server database that you registered with the EDB Postgres AI agent, you may observe additional schemas that don't directly correspond to schemas you explicitly created. These extra schemas are generated internally by EDB Postgres Advanced Server as part of its Oracle compatibility feature set to represent Oracle PACKAGE objects. This is a known issue that we expect to correct in a future release. In the meantime, you can disregard these generated schemas for migration purposes.

HM Migration Portal known issues

Disabling the automatic Migration Portal project creation requires a workaround

> **Tip**
>
> Resolved in HM versions 2026.1 and later. See *Disabling the automatic creation of Migration Portal projects* if you have version 2026.1 or later.

There is a known issue in 2025.12 and earlier versions affecting the internal `beacon-server` service where setting the `autocreate_mp_projects: false` parameter in the ConfigMap doesn't prevent the automatic creation of Migration Portal projects.

Workaround: To prevent the automatic creation of Migration Portal projects and enforce manual creation, set the parameter `max_automatic_project_creation_bytes` to an extremely low value, for example 1, within the `beacon-server` ConfigMap. This low threshold ensures that all schemas exceed the limit, successfully bypassing the automatic creation process.

1. Edit the `beacon-server` ConfigMap:

```
kubectl edit configmap beacon-server-config -n upm-beacon
```

1. Locate the `beacon_server.yaml` block and insert or modify the `max_automatic_project_creation_bytes` section under `server` . Set the value to an extremely low number, like 1, to ensure automatic creation is bypassed:

```
apiVersion: v1
data:
  beacon_server.yaml: |
  # [...]
    server:
    # [...]
      migration_portal_configuration:
        max_automatic_project_creation_bytes: 1
```

1. Apply the changes by restarting the `beacon_server` deployment:

```
kubectl rollout restart deployment upm-beacon-server -n upm-beacon
```

Warning

The *upm-beacon-server* supports all database registration and schema ingestion activities across the entire HM instance using EDB Postgres AI agents. Restarting the *upm-beacon-server* will halt all _ongoing_ database registrations and schema ingestions, regardless of the project. Ensure you coordinate this restart with other users.

Databases that already appear in the Estate > Migrations tab, for which schema ingestions have finished, are not affected by restarts. .. ::

1. You can monitor the status of your restarted service:

```
kubectl get pods -n upm-beacon
```

Note

There is another currently known issue where the HM console warning for manual Migration Portal project creation persists after manually creating projects.

HM warning persists after manual Migration Portal project creation

Tip

Resolved in HM versions 2026.1 and later.

In versions 2025.12 and earlier, when the automatic creation of a Migration Portal project fails (for example, due to the ingested schemas exceeding the default size limit of 300 MB or a different specified limit), the HM console displays a persistent warning instructing the user to manually create the projects. This warning continues to display in the HM console even after the user has successfully created the necessary Migration Portal projects for their Oracle schemas.

Automatic Migration Portal project creation fails with invalid `resource_id` characters

The automatic creation of Migration Portal projects fails (with error message **Failed to create**) if the `resource_id` specified for the registered Oracle database contains characters that are invalid.

This failure occurs because the `resource_id` value is used in the construction of the Migration Portal project name, which currently doesn't support special characters like hyphens (-), which prevent project creation.

Workaround: To ensure successful automatic creation of Migration Portal projects, the `resource_id` value specified in the EDB Postgres AI agent configuration for Oracle databases must conform to the following naming rules:

- It can include alphanumeric characters (A–Z, a–z, 0–9).
- It can include the underscore (_) character, but it cannot begin with an underscore.
- It must not start with a number.
- It must not contain special characters, such as the hyphen (-).

Migration Portal database pods fail to start after upgrade

> **Tip**
>
> Resolved in HM versions 2026.2 and later.

Description: Following an upgrade to the 2026.1 Innovation Release, the PostgreSQL pods for the Migration Portal (for example, `mp-epas-16` , `mp-epas-17`) may enter a `CrashLoopBackOff` or `ImagePullBackOff` state. This occurs because the deployment is referencing an incorrect image digest or tag that is no longer available in the repository.

Workaround: To resolve the error and restore the Migration Portal to a healthy state, manually delete the affected pods using `kubectl delete pod <pod-name>` . The deployment will automatically recreate the pods using the correct, updated image.

SSL connection error (x25519) when migrating schema to EPAS 18 using Migration Portal

Description: When using the Migration Portal's **online schema migration** feature to target an EDB Postgres Advanced Server (EPAS) database version 18 in a FIPS-enforced environment, the connection may fail with the Migration Portal UI error: `Failed to connect: Failed to initialize pool: SSL error: Cannot decode named group: x25519` . This is due to the FIPS environment enabled for the Hybrid Manager Migration Portal.

This issue occurs because Postgres 18 defaults to the `x25519` SSL group, which is currently not supported by certain components used by the Migration Portal application. This issue specifically affects the **Online Migration** option, where the destination database is either an HM-managed EPAS database cluster or self-managed EPAS database.

Workaround: To resolve the connection failure, you must manually update the supported SSL groups list on the destination database cluster to include only FIPS-compliant NIST curves (such as `prime256v1`).

Configuration example on an HM-managed EPAS database cluster version 18

1. In the HM console, select the project that hosts the destination cluster.
2. Select **Clusters** > **Postgres Clusters** and select your version 18 cluster.
3. Select **Quick Actions** > **Edit Cluster** and then navigate to **DB Configuration** tab.
4. Locate the `ssl_groups` parameter.
5. Change the value from the default to `prime256v1` .
6. Save the changes and allow the cluster to reconcile.

After the cluster is updated, retry the connection test in the Migration Portal online schema migration.

Configuration example on a self-managed EPAS database version 18

1. Edit the `postgresql.conf` file, locate the `ssl_groups` parameter and change it to `prime256v1` .
2. Load the new configuration by restarting the Postgres service with `systemctl` , using `psql` , or a method of your preference.

After the cluster is updated, retry the connection test in the Migration Portal online schema migration.

Automatic Migration Portal project creation fails with "UNAVAILABLE: unresolved address"

> **Tip**
>
> Resolved in HM versions 2026.4 and later.

Description: The automatic creation of Migration Portal projects may fail with an internal server error: `rpc error: code = Internal desc = Internal server error occurred: UNAVAILABLE: unresolved address` . This is typically caused by a transient network resolution issue between the Hybrid Manager components and the Migration Portal service.

Workaround: If you encounter this error, use one of the following methods to resolve it:

- Manual creation: Wait 20–40 minutes for network states to stabilize, then manually create the Migration Portal project through the HM console.
- Resource ID reset: Modify the `resource_id` in your `beacon_agent.yaml` configuration file (to any new ID assigned by yourself) and restart the EDB Postgres AI agent, or create a new project and map the agent to that new project to trigger a fresh synchronization.

Upgrade to version 2026.4 fails due to immutable `ragchew-ai-vendor-secrets` secret

Description: When upgrading HM to version 2026.4, the upgrade process may fail with a `Failed to install component` error for `edb-migration-copilot` . This occurs because the upgrade attempts to change the `type` field of the existing `ragchew-ai-vendor-secrets` Kubernetes Secret in the `edb-migration-copilot` namespace. The `type` field is immutable once a Secret is created, causing the Kubernetes API server to reject the update with the error: `Secret "ragchew-ai-vendor-secrets" is invalid: type: Invalid value: "Opaque": field is immutable` .

Workaround: Before upgrading to 2026.4, manually delete the secret `ragchew-ai-vendor-secrets` in the `edb-migration-copilot` namespace. The operator will recreate it with the correct type during the upgrade:

```
kubectl delete secret ragchew-ai-vendor-secrets -n edb-migration-copilot
```

SQL file assessment defaults to `Oracle_19c` as the source database type

> **Tip**
>
> Resolved in HM versions 1.4.0 and later.

Description: When uploading a SQL file for an Application Migration assessment, the HM console doesn't associate the source database resource ID with the uploaded file. Because no resource ID is provided, `upm-beacon` sends the assessment request to the Migration Portal with the source database version set to `UNKNOWN` . The Migration Portal then defaults to `Oracle_19c` as the source database type when processing the assessment. As a result, assessments may

not accurately reflect the compatibility rules for your actual source database version. Additionally, at least one project must exist in the Migration Portal for the application SQL assessment to complete without errors.

Workaround: Ensure at least one project exists in the Migration Portal before initiating an Application Migration assessment via SQL file upload. A fix that associates source database resource IDs with uploaded SQL files is being tracked for a future release.

Other known issues

Data migrations supported without a Load Balancer in RHOS environments only

> **Tip**
>
> Resolved in HM versions 2026.1 and later.

Description: The primary method for exposing Kafka (which is required for data migrations) within Hybrid Manager is via Load Balancers (an exception is the RHOS environment, where a native route is used to expose Kafka). Therefore, in non-RHOS environments where the Load Balancer is disabled, the data migration workflows will not function, as there is no fallback mechanism to support exposing Kafka using NodePort mode.

Workaround: To enable data migration workflows in non-RHOS environments such as RKE, you must configure a Load Balancer.

Optimizing migration performance

The EDB Data Migration Service (DMS) is included with Hybrid Manager (HM). You can leverage it to migrate your data from existing databases (Oracle, self-managed Postgres, and so on) to an HM-managed database cluster. EDB DMS supports two migration methods: snapshot and snapshot + streaming, allowing you to migrate your data with minimal-to-no downtime.

Understanding migration workloads in HM

The EDB DMS runs within the same Kubernetes environment where your HM is deployed (on-premises, AWS, Red Hat OpenShift, and Google Cloud). This means that most of the workloads required for migration operations are managed and optimized by your Kubernetes cluster.

Generally, you don't need to extensively fine-tune resources, as your Kubernetes environment meeting the general requirements for running HM helps with a smooth migration. However, for large datasets, optimizing specific components can significantly accelerate the process.

The key components to consider for resource allocation are:

- **EDB DMS Agent in reader mode:** This component can run on the same machine as the source database or on an auxiliary machine that can connect to the source database and to HM.
- **Destination database cluster:** This can be the database managed by HM or a different external self-managed database. However, we'll focus on an HM-managed database. This is where your data will reside.
- **EDB DMS Agent in writer mode:** This component is required only if your migration destination is an external (self-managed) database cluster and it runs on that database's host machine.

Whether you perform a snapshot or snapshot + streaming migration also affects performance differently:

- **Snapshot migrations:** They primarily focus on efficiently transferring a large volume of existing data from a source database to the destination. Performance for these is heavily influenced by total data size and sustained disk I/O.

- **Snapshot + streaming migrations:** They first transfer a snapshot of existing data and then continuously stream ongoing changes (transactions) from the source to the destination. Performance here is driven by the sustained transaction rate (TPS) and the ability to keep up with the source's write activity.

Optimizing migration performance for large datasets

HM and DMS are designed to handle resource usage. For datasets smaller than 100 GB, they're configured to perform fast migrations without further tuning. However, for datasets larger than 100 GB, a few targeted alterations can lead to significantly better performance and quicker migrations.

Tuning destination database performance

Optimizing the destination database cluster (whether an HM-managed database or a self-managed database) provides the most significant performance benefits from resource tuning.

> **Note**
>
> These recommendations are based on current testing and are subject to change. We work continuously to improve migration performance and reduce migration times.

Storage recommendations

During a migration, storage IOPS can peak at 2000, with an average throughput of roughly 300 MB/s and peaks up to 350 MB/s. To accommodate these demands:

- Separate WAL and data volumes. Aim for storage volumes that can handle a minimum of 2500-3000 IOPS and at least 200-250 MB/s of throughput for both your data and write-ahead log (WAL) volumes. We highly recommend using separate volumes for WAL and data for optimal performance.
- Unified storage and WAL volume. A unified storage and WAL volume can adversely affect performance. For example, using a single volume with 3000 IOPS and 125 MB/s throughput can increase migration time by over 40%. If you must use a unified storage and WAL volume, we recommend a throughput of 300-400 MB/s. Keep in mind that if the WAL becomes full, the migration will fail.

How to adjust this

For HM-managed database clusters:

- Adjust WAL and data volume allocation by *Editing a published bundle* your cluster's storage.
- By default, HM-managed clusters use separate volumes for data and WAL volumes, but if you clear the **Use a separate storage volume for Write-Ahead Logs** option while creating the cluster, you can't enable this setting later.
- Volume speeds (IOPS or throughput) are determined by the cluster's underlying infrastructure. When migrating to an HM-managed database cluster, the specific storage classes available for your cluster, and their associated IOPS and throughput characteristics, are determined by your HM administrator (and the underlying cloud or on-premise environment). Therefore, review the storage options available for your clusters and work with your HM administrators to ensure the selected storage meets the recommended performance criteria. For more information on cloud provider infrastructure provisioning, see Request Amazon EBS volume modifications or Google Cloud - About Persistent Disk performance .

Memory recommendations

While 5 GB of memory is sufficient, a larger amount facilitates optimal performance during a migration. Use at least 10 GB of memory for the destination database.

How to adjust this

For HM-managed database clusters:

- Adjust memory allocation by *Editing a published bundle* your cluster's instance size (under **Cluster Settings**).
- You can also adjust the `shared_buffers` , `effective_cache_size` , and `wal_buffers` parameters in **DB Configuration**.

CPU recommendations

The destination database cluster needs at least 4 CPU cores. A lower core count will negatively impact migration performance.

How to adjust this

For HM-managed database clusters, adjust CPU allocation by *Editing a published bundle* your cluster's instance size.

> **Note**
>
> Apply these optimizations during the migration process. After the migration is complete, you can readjust these values as needed based on your ongoing workload requirements.

Extensive testing has shown that the biggest factor affecting migration performance is volume speed (IOPS and throughput). While decreasing CPU below recommendations can reduce performance, it's not nearly to the degree that low IOPS and throughput for storage will. Halving the storage throughput can increase migration time by over 40%, whereas halving the CPU count to 2 cores might increase it by 10-15%. These recommendations are designed to yield the best migration performance. Using lower values may still result in successful migrations but at a slower speed.

WAL configuration

WAL checkpoints are one of the most resource-intensive tasks during a data migration. By default, `max_wal_size` is set to 1 GB, which means that a checkpoint is made for every GB written to the WAL file. During a migration, this can result in checkpoints occurring every 30-60 seconds, putting a substantial load on the destination database.

By increasing the `max_wal_size` during the migration, you can reduce the load on the destination database. Our tests found that a larger `max_wal_size` doesn't impact migration speed. For instance, a migration with a 20 GB `max_wal_size` takes the same amount of time as one with a 1 GB setting. However, the migration using the 20 GB setting consumes approximately 50% fewer resources.

Given that WAL logging ensures that all migrated data is properly saved via checkpoints, disabling `max_wal_size` isn't a viable option for most EDB databases, as several critical features depend on it.

How to adjust this

For HM-managed database clusters:

Adjust WAL configurations by *Editing a published bundle* your cluster's DB configuration. Search for the `max_wal_size` parameter, provide the new size in MB, and save to apply the configuration.

Tuning EBD DMS Agent performance

Optimal performance for the EDB DMS Agents is achieved when the host machines where these components are installed follow these specifications.

EDB DMS Agent in reader mode

CPU: 3 CPU cores

Memory: 4 GB

EDB DMS Agent in writer mode

CPU: 2 CPU cores

Memory: 2.5 GB

Shared host considerations

For optimal migration performance, carefully consider resource allocation when EDB DMS components and source database run on the same machine/host.

- **DMS agents share a host:** If both agents are on the same machine, one in reader mode and a second one in writer mode, provision combined resources (at least 5 CPU cores and 6.5 GB memory).
- **DMS component and source database share a host:** If a DMS agent is on the same host as the source database, ensure the host's resources are significantly increased to accommodate both the database's operational workload and the migration's demands.

Optimizing migration performance for bulk migrations

HM and DMS are designed to handle multiple concurrent migrations. However, when performing bulk migrations (that is, high volumes of concurrent or large migrations), increasing the number of Kafka brokers can significantly enhance data migration performance.

> **Tip**
>
> If you observe high CPU and memory use on the Kafka brokers during a data migration, the default DMS configuration is insufficient for bulk migration. Use *Grafana dashboard setup for Agent Factory metrics* to check the resource usage for the Kafka pods in the *transporter-kafka-operator* namespace.

Change Kafka broker count

1. Connect to your HM Kubernetes cluster using your `kubeconfig` file. (The specific command varies by cloud environment.)

```
# AWS example:
aws eks update-kubeconfig --name eks-1234567890-main --region eu-west-2 --profile app-
↪dev
```

1. Before making any changes, check if a Kafka cluster was already provisioned. This is the case if you previously ran a data migration.

```
kubectl get kafka -n transporter-kafka-operator
```

1. Remove existing migrations and Kafka clusters. If the command output lists a Kafka resource, you must first delete all migrations to start with a clean state.

> **Warning**
>
> The Kafka cluster is generally deployed to support all data migration activities across the entire HM instance. Removing the Kafka resource will halt all active data migrations, regardless of the project. Coordinate this change with other users. !!!

In the HM console, delete all active and completed migrations in your project under **Migrate > Migrations**.

For testing environments

For testing or development environments, you can also remove the Kafka cluster using `kubectl` . However, we recommend this method only in non-production environments, as it deletes all resources other users may be using.

```
kubectl -n transporter-kafka-operator delete $(kubectl get strimzi -o name -n␣
↪transporter-kafka-operator)
```

1. Get the name of the ConfigMap in the `transporter-dp-provisioner` namespace:

```
kubectl get cm -n transporter-dp-provisioner
```

1. Open the ConfigMap for editing. Replace `<config_map_name>` with the name found in the previous step:

```
kubectl edit cm transporter-dp-provisioner-<config_map_name> -n transporter-dp-
↪provisioner
```

1. Locate the `broker_number` field in the ConfigMap and change the value to your desired number of Kafka brokers (for example, `broker_number: 3`).
2. Save the ConfigMap. The new configuration is reloaded and applied within about 10 seconds.
3. You can start new data migrations that will use the changed broker capacity.

Migratiog

Choose your migration path

For end-to-end instructions on how to perform each type of database migration, see:

- *AWS RDS PostgreSQL or AWS RDS Aurora (PostgreSQL compatible) database to an HM-managed database*
- *Migrate your Oracle database to an HM-managed database*
- *Migrate your self-managed Postgres database to a different self-managed Postgres database*
- *Migrate your self-managed Postgres database to an HM-managed database*

Migrate your Oracle database to an HM-managed database

These are the tasks involved in an end-to-end migration from an Oracle database to an EDB Postgres Advanced Server cluster managed by Hybrid Manager (HM). The process uses two agents: the EDB Postgres AI agent (`beacon-agent`) to register your Oracle database with HM and ingest schema for assessment and conversion, and the EDB DMS agent (`cdcagent`) to perform the actual data migration.

1. ** *Configuring your auxiliary machine or bastion host* ** — Configure the auxiliary machine that acts as the bridge between your Oracle source and HM. This machine hosts both agents.

2. ** *Preparing Oracle source database* ** — Create a dedicated migration user and configure your Oracle database to support schema ingestion and Change Data Capture (CDC).
3. ** *Configuring the EDB Postgres AI agent to register your Oracle database* ** — Install and configure the EDB Postgres AI agent to connect your Oracle database to HM. This enables migration assessment and automatic schema DDL extraction for use with Migration Portal.
4. ** *Preparing your HM-managed destination database* ** — Set up the HM-managed EDB Postgres Advanced Server cluster that will receive the migrated schema and data.
5. ** *Migrating Postgres schemas* ** — Use Migration Portal to convert your Oracle DDL to be compatible with EDB Postgres Advanced Server, then import the schema to the destination database before data migration begins.
6. ** *Configuring and running the EDB DMS agent* ** — Configure the EDB DMS agent in reader mode to read changes from your Oracle source for data migration.
7. ** *Migrating data from an AWS RDS or Aurora PostgreSQL database to an HM-managed database* ** — Create and run the data migration in the HM console.

Configuring your auxiliary machine or bastion host

The EDB Postgres AI agent (`beacon-agent`) and the DMS agent (`cdcagent`) can run on the Oracle database server itself or on a separate auxiliary machine or bastion host. In most production environments, running the agents on a dedicated auxiliary machine is preferred — it avoids installing additional software on the database server, and a single agent instance can connect to databases on multiple servers.

Required software

Ensure you install the following software on your auxiliary machine before running the agents:

- **Java 17 or higher** — required by the DMS agent (`cdcagent`). Verify with `java -version` .
- **Oracle Instant Client with SQL*Plus** — required by both agents. The EDB Postgres AI agent uses SQL*Plus to extract DDL from the Oracle source. The DMS agent uses the Oracle JDBC libraries for Change Data Capture (CDC) connectivity.

Download Oracle Instant Client and install both the Basic
and SQL*Plus packages. Ensure you download a version compatible with your source Oracle database.

- **EDB Postgres AI agent** (`beacon-agent`) — handles database registration and schema assessment. Check the *beacon-agent version compatibility matrix* . Then install a compatible version of the EDB Postgres AI agent .
- **DMS agent** (`cdcagent`) — handles data migration and CDC. Check the *DMS agent version compatibility* . Then *install a compatible version of the DMS agent* .

Configuring the environment

Both agents must be able to locate the Oracle libraries and binaries at runtime. Set the following environment variables on your auxiliary machine, replacing `</path/to/instantclient>` with the actual Instant Client directory (for example, `/opt/oracle/instantclient_19_19`). These variables must be set in the same shell session where you run the agents. For persistent configuration, add them to your shell profile (for example, `~/.bashrc` or `~/.bash_profile`). If you run the agents as systemd services, set these variables in the service file instead.

```
export ORACLE_HOME=</path/to/instantclient>
export PATH=$ORACLE_HOME:$PATH
export LD_LIBRARY_PATH=$ORACLE_HOME:$LD_LIBRARY_PATH
```

> **Note**
>
> For Oracle Instant Client, the SQL*Plus binary and the shared libraries (*.so* files) are both located directly in the Instant Client root directory — not in a *bin/* or *lib/* subdirectory. Set both *PATH* and *LD_LIBRARY_PATH* to the Instant Client root.

Verify the setup before proceeding.

```
which sqlplus
```

The output should show the path to the SQL*Plus binary inside the Instant Client directory with the version number. Alternatively, run:

```
sqlplus -V
```

Which shows the SQL*Plus version.

Network connectivity

Ensure network connectivity between the auxiliary machine and the source database host, and between the auxiliary machine and Hybrid Manager. You'll verify the specific ports required later, when configuring the agents.

Next step

Preparing Oracle source database

Preparing Oracle source database

Configure your Oracle source database and create a dedicated migration user before starting your migration. How much you need to configure depends on what you plan to do:

- **Schema ingestion and assessment only** (EDB Postgres AI agent): You don't need to make database-level changes. You only need to specify a migration user with read access to Oracle data dictionary views — specifically those granted via `SELECT_CATALOG_ROLE` or `SELECT ANY DICTIONARY` — and the `CREATE TABLE` privilege. The assessment script uses the `CREATE TABLE` privilege to create temporary tables during execution. No permanent changes are made to your database. If your goal is to, for example, evaluate a large Oracle estate before committing to migration, skip directly to *Configuring the migration user* .
- **Data migration** (DMS agent): Requires additional database-level configuration — `ARCHIVELOG` mode, supplemental logging, dedicated tablespaces, and expanded LogMiner privileges on the migration user. Complete all sections on this page.

If you plan to run both schema assessment and data migration, complete all sections. The migration user you create here serves both agents.

Prerequisites

- Identify your Oracle architecture (CDB/PDB multitenant vs. non-CDB).
- Collect the names of the schemas and tables you want to migrate.

Configuring the database

Enabling ARCHIVELOG mode

Migration tools require the database to be in ARCHIVELOG mode to ensure that data changes are captured during the transfer.

1. Connect to the database as SYSDBA to perform administrative tasks:

Connect to the Root Container (CDB$ROOT) using the CDB service name:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<CDB_NAME> as sysdba
```

For example:

```
sqlplus sys/password123@//localhost:1521/ORCLCDB as sysdba
```

Connect to the Database Instance using the Service Name or SID:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<SID_OR_SERVICE> as sysdba
```

For example:

```
sqlplus sys/password123@//localhost:1521/ORCL as sysdba
```

1. Verify the status:

```
archive log list;
```

If the status is NOARCHIVELOG , you must enable it.

1. Configure the Fast Recovery Area (FRA) and enable logging:

> **Tip**
>
> ***2_blank_5_*- Create a directory for Fast Recovery Area (FRA) in advance.`1_blank_4_-` Ensure the *db_recovery_file_dest_size* is large enough to hold at least 24 hours of redo logs during peak migration loads to prevent the database from hanging. .. Tip::**
> edb_notranlate_5

> **Note**
>
> Setting *db_recovery_file_dest* and *db_recovery_file_dest_size* configures Oracle's FRA, which is a required storage location for the archived logs Debezium reads.

These parameters are critical for database stability. Always consult your DBA before setting them in a production environment. .. ::
2_blank_1_1. Restart the database and enable *ARCHIVELOG* :

```
SHUTDOWN IMMEDIATE;
STARTUP MOUNT;
ALTER DATABASE ARCHIVELOG;
ALTER DATABASE OPEN;
```

1. Confirm the change:

```
archive log list;
```

Enabling supplemental logging

Supplemental logging ensures that the redo logs contain enough information for the DMS agent to identify changed rows.

1. Connect to the database as `SYSDBA` to perform administrative tasks:

Connect to the Root Container (CDB$ROOT) using the CDB service name:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<CDB_NAME> as sysdba
```

Connect to the Database Instance using the Service Name or SID:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<SID_OR_SERVICE> as sysdba
```

1. Enable supplemental logging:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

Optionally, ensure adequate redo log space

Undersized redo logs cause frequent log switching, which can slow down or destabilize the migration. If your redo logs are small relative to your database's change volume, increase their size and count before starting the migration.

More information…

The data migration process consists of two main phases: a consistent snapshot of the data, followed by a continuous stream of changes. This change stream is powered by LogMiner and the Oracle database redo logs.

The lifetime of a change in the redo logs is limited. It's determined by the size and number of the logs, as well as the database's change throughput. Redo logs that are too small can cause frequent log switching, which negatively impacts migration performance.

1. Connect to the root container or database instance as `SYSDBA` :

Connect to the root container:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<CDB_NAME> as sysdba
```

Connect to the database instance:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<SID_OR_SERVICE> as sysdba
```

1. To view the current log file configuration, run the following SQL queries:

```
SELECT GROUP#, TYPE, MEMBER FROM V_$LOGFILE;
__OUTPUT__
    GROUP# TYPE    MEMBER
---------- ------- --------------------------------------------------
         1 ONLINE  /opt/oracle/oradata/ORCLCDB/redo03.log
         2 ONLINE  /opt/oracle/oradata/ORCLCDB/redo01.log
         3 ONLINE  /opt/oracle/oradata/ORCLCDB/redo04.log
```

```
SELECT GROUP#, ARCHIVED, BYTES/1024/1024 MB, STATUS FROM V_$LOG;
__OUTPUT__
    GROUP# ARC  MB STATUS
---------- --- --- ----------------
         1 YES 2000 INACTIVE
         3 YES 2000 INACTIVE
         3 NO  2000 CURRENT
```

For example, the output might show three log groups, each with a size of 2000MB, which may be too small for many production environments.

1. If needed, you can adjust the redo logs using commands like these:

```
ALTER DATABASE ADD LOGFILE GROUP 4 (/opt/oracle/oradata/ORCLCDB/redo04.log) SIZE 8G;
ALTER DATABASE ADD LOGFILE GROUP 5 (/opt/oracle/oradata/ORCLCDB/redo05.log) SIZE 8G;
ALTER DATABASE ADD LOGFILE GROUP 6 (/opt/oracle/oradata/ORCLCDB/redo06.log) SIZE 8G;
ALTER DATABASE ADD LOGFILE GROUP 7 (/opt/oracle/oradata/ORCLCDB/redo07.log) SIZE 8G;
ALTER SYSTEM ARCHIVE LOG CURRENT;
ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM ARCHIVE LOG CURRENT;
ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM ARCHIVE LOG CURRENT;
ALTER SYSTEM CHECKPOINT;
ALTER DATABASE DROP LOGFILE GROUP 1;
ALTER DATABASE DROP LOGFILE GROUP 2;
ALTER DATABASE DROP LOGFILE GROUP 3;
```

These commands create four 8GB redo log groups, each with a single redo log file.

> **Note**
>
> Always consult your DBA to determine the appropriate redo log size for your production environment.

Configuring tables

Enabling table-level supplemental logging

While database-level logging (enabled in the previous step) provides the foundation, you must also specify which individual tables require full column logging. This ensures that the DMS agent receives a *before* and *after* image of every row change.

- For Oracle, the terms *schema* and *user* are used interchangeably. When you run the SQL commands for data migration, to avoid errors, be sure to set the schema name to the correct user account.
- For Oracle XE compatibility restrictions, see *Limitations* . .. :: <details><summary>More information...</summary>

Supplemental logging refers to the capture of additional information in Oracle redo logs, such as *before* state. This extra redo log information is needed for log-based applications like EDB DMS to capture change events. See Oracle's Supplemental Logging documentation for more information.

To use Debezium for data migration, you must enable supplemental logging at both the database and table levels. This is crucial for Debezium to correctly capture change data:

- **Database-level supplemental logging** is the foundation for change data capture (CDC). It ensures that every change message includes a tracking number, which is typically the primary key of the modified row. This allows

you to uniquely identify which specific record was changed. Without this, Debezium can't link a change event to its source row.

- **Table-level supplemental logging** provides the detailed information. It ensures that the change message includes a before image of the entire row, that is, a complete copy of all columns before the change occurred. This is especially useful for understanding exactly what was changed, as Debezium can compare the before and after states. Without this, Debezium captures only the updated column and not the full context of the change.

1. Connect to the database:

Table-level logging must be enabled within the Pluggable Database (PDB) where your data resides. Connect to the PDB where your data resides using the CDB service name as `SYSDBA` :

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<PDB_NAME> as sysdba
```

Connect to the Database Instance using the Service Name or SID:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<SERVICE_NAME_OR_SID> as sysdba
```

1. Enable logging for your specific tables. You must run this command for every table you intend to migrate:

```
ALTER TABLE <SCHEMA_NAME>.<TABLE_NAME> ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

For example:

```
ALTER TABLE HR_APP.EMPLOYEES ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
ALTER TABLE FINANCE_APP.SALARIES ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

```
SELECT ALTER TABLE  || owner || . || table_name ||  ADD SUPPLEMENTAL LOG DATA (ALL)␣
↪COLUMNS;
FROM dba_tables
WHERE owner = YOUR_SCHEMA_1;
```

Copy the output and execute it to update the entire schema at once.

The Data Migration Service (DMS) and the underlying Debezium engine require dedicated tablespaces to store internal metadata, log-mining details, and tracking offsets. This keeps migration-related overhead separate from your application data.

Initial size: 25MB is the minimum for testing. Production environments typically require 512MB or more.

File paths: Always consult your Database Administrator (DBA) to determine the correct directory for `.dbf` files on your server.

In a multitenant environment, you must create a tablespace in the Root Container (CDB) for global log mining and a tablespace in the Pluggable Database (PDB) for table-specific tracking.

1. Connect as sysdba to the CDB:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<CDB_NAME> as sysdba;
```

1. Create the CDB tablespace:

```
CREATE TABLESPACE <TS_NAME>
DATAFILE <CDB_PATH>/<TS_NAME>.dbf
SIZE 25M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

1. Switch to the PDB:

```
ALTER SESSION SET CONTAINER = <PDB_NAME>;
```

1. Create the PDB tablespace:

```
CREATE TABLESPACE <TS_NAME>
DATAFILE <PDB_PATH>/<TS_NAME>.dbf
SIZE 25M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

> **Tip**
>
> Using the same name for both tablespaces (for example, *LOGMINER_TBS*) simplifies configuration later. !!!

Create a tablespace to handle both log mining and metadata tracking.

```
CREATE TABLESPACE <TS_NAME>
DATAFILE <PATH>/<TS_NAME>.dbf
SIZE 25M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

More information…

To perform data migration, both DMS and its underlying Debezium library require a dedicated user. Before creating this user, set up the required tablespaces.

For CDB architecture, you need two tablespaces:

- **CDB-level tablespace** : This tablespace is for Debezium's internal processes. It's used for log mining and other management operations that must run at the container database level.
- **PDB-level tablespace** : This one is for Debezium operations that are specific to your pluggable database. It handles processes related to the individual tables you're capturing within that PDB.

Configuring the migration user

To simplify the migration process, we recommend creating a single Common User (C## for multitenant CDB/PDB configurations) or a dedicated system user (for non-CDB configurations). This unified user handles both schema assessment via the EDB Postgres AI agent and live data migration via the DMS agent.

1. Connect to the root container or database instance as SYSDBA:

Connect to the Root Container (CDB$ROOT) using the CDB service name:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<CDB_NAME> as sysdba
```

For example:

```
sqlplus sys/password123@//localhost:1521/ORCLCDB as sysdba
```

Connect to the Database Instance using the Service Name or SID:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<SID_OR_SERVICE> as sysdba
```

1. Create a new user for migrations:

```
CREATE USER <C##_MIGRATION_USER> IDENTIFIED BY <MIGRATION_USER_PASSWORD> DEFAULT␣
↪TABLESPACE <TS_NAME> QUOTA UNLIMITED ON <TS_NAME> CONTAINER=ALL;
```

```
CREATE USER <MIGRATION_USER> IDENTIFIED BY <MIGRATION_USER_PASSWORD> DEFAULT␣
↪TABLESPACE <TS_NAME> QUOTA UNLIMITED ON <TS_NAME>;
```

1. Grant all of the following privileges:

```
-- Grant core session and container privileges
-- These grants are required for schema ingestion and assessment via the EDB Postgres␣
↪AI agent.
GRANT CREATE SESSION TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SET CONTAINER TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$DATABASE TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT CONNECT TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT_CATALOG_ROLE TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT CREATE TABLE TO <C##_MIGRATION_USER> CONTAINER=ALL;

-- Grant migration and log mining permissions
-- These additional grants are required for data migration via the DMS agent.
GRANT FLASHBACK ANY TABLE TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ANY TABLE TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT EXECUTE_CATALOG_ROLE TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ANY TRANSACTION TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ANY DICTIONARY TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT LOGMINING TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT LOCK ANY TABLE TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT CREATE SEQUENCE TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT EXECUTE ON DBMS_LOGMNR TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT EXECUTE ON DBMS_LOGMNR_D TO <C##_MIGRATION_USER> CONTAINER=ALL;

-- Grant access to system views.
-- Note: several of these views are also accessible via SELECT_CATALOG_ROLE,
-- granted above. The explicit grants are intentional to ensure compatibility
-- across Oracle patch levels and to make the required access unambiguous.
GRANT SELECT ON V_$LOGMNR_LOGS TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$LOGMNR_CONTENTS TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$LOGFILE TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$ARCHIVED_LOG TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$ARCHIVE_DEST_STATUS TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$TRANSACTION TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$VERSION TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$INSTANCE TO <C##_MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$LICENSE TO <C##_MIGRATION_USER> CONTAINER=ALL;

ALTER USER <C##_MIGRATION_USER> QUOTA UNLIMITED ON <TS_NAME> CONTAINER=ALL;
```

Switch to the PDB:

```
ALTER SESSION SET CONTAINER = <PDB_NAME>;
```

Set the default tablespace for the user:

```
ALTER USER <C##_MIGRATION_USER> DEFAULT TABLESPACE <TS_NAME>;
```

```
-- Grant core session privileges
-- These grants are required for schema ingestion and assessment via the EDB Postgres
↪AI agent.
GRANT CREATE SESSION TO <MIGRATION_USER>;
GRANT SELECT ON V_$DATABASE to <MIGRATION_USER>;
GRANT CONNECT TO <MIGRATION_USER>;
GRANT SELECT_CATALOG_ROLE TO <MIGRATION_USER>;
GRANT CREATE TABLE TO <MIGRATION_USER>;

-- Grant migration and log mining permissions
-- These additional grants are required for data migration via the DMS agent.
GRANT FLASHBACK ANY TABLE TO <MIGRATION_USER>;
GRANT SELECT ANY TABLE TO <MIGRATION_USER>;
GRANT EXECUTE_CATALOG_ROLE TO <MIGRATION_USER>;
GRANT SELECT ANY TRANSACTION TO <MIGRATION_USER>;
GRANT SELECT ANY DICTIONARY TO <MIGRATION_USER>;
GRANT LOGMINING TO <MIGRATION_USER>;
GRANT LOCK ANY TABLE TO <MIGRATION_USER>;
GRANT CREATE SEQUENCE TO <MIGRATION_USER>;
GRANT EXECUTE ON DBMS_LOGMNR TO <MIGRATION_USER>;
GRANT EXECUTE ON DBMS_LOGMNR_D TO <MIGRATION_USER>;

-- Grant access to system views.
-- Note: several of these views are also accessible via SELECT_CATALOG_ROLE,
-- granted above. The explicit grants are intentional to ensure compatibility
-- across Oracle patch levels and to make the required access unambiguous.
GRANT SELECT ON V_$LOGMNR_LOGS TO <MIGRATION_USER>;
GRANT SELECT ON V_$LOGMNR_CONTENTS TO <MIGRATION_USER>;
GRANT SELECT ON V_$LOGFILE TO <MIGRATION_USER>;
GRANT SELECT ON V_$ARCHIVED_LOG TO <MIGRATION_USER>;
GRANT SELECT ON V_$ARCHIVE_DEST_STATUS TO <MIGRATION_USER>;
GRANT SELECT ON V_$TRANSACTION TO <MIGRATION_USER>;
GRANT SELECT ON V_$VERSION TO <MIGRATION_USER>;
GRANT SELECT ON V_$INSTANCE TO <MIGRATION_USER>;
GRANT SELECT ON V_$LICENSE TO <MIGRATION_USER>;

ALTER USER <MIGRATION_USER> QUOTA UNLIMITED ON <TS_NAME>;
```

1. Verify the migration user can see your source tables from the CDB root. If this returns no rows, the grants weren't applied with `CONTAINER=ALL` — regrant all privileges as shown above.

Connect as the migration user to the CDB root:

```
sqlplus <C##_MIGRATION_USER>/<MIGRATION_USER_PASSWORD>@//<HOST>:<PORT>/<CDB_NAME>
```

Then verify your source tables are visible:

```
SELECT owner, table_name FROM cdb_tables WHERE owner = <SCHEMA_NAME>;
```

This step does not apply to non-CDB configurations.

1. Grant `SELECT` on source tables:

Switch to the PDB, still as `SYSDBA` :

```
ALTER SESSION SET CONTAINER = <PDB_NAME>
```

Ensure you are connected to the database instance as sysdba.

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<SID_OR_SERVICE> as sysdba
```

1. Grant SELECT permissions to your specific tables. You must run this command for every table you intend to migrate:

```
GRANT SELECT ON <SCHEMA_NAME>.<TABLE_NAME> TO <C##_MIGRATION_USER/MIGRATION_USER>
```

1. Set the default tablespace:

Change the default tablespace in both the CDB and the PDB:

```
-- In the CDB ROOT
ALTER SESSION SET CONTAINER = CDB$ROOT;
ALTER USER <C##_MIGRATION_USER> DEFAULT TABLESPACE <TS_NAME>;

-- In the PDB
ALTER SESSION SET CONTAINER = <PDB_NAME>;
ALTER USER <C##_MIGRATION_USER> DEFAULT TABLESPACE <TS_NAME>;
```

```
ALTER USER <MIGRATION_USER> DEFAULT TABLESPACE <TS_NAME>;
```

1. Exit sql, and perform a quick connection test with your migration user:

```
sqlplus <C##_MIGRATION_USER>@//<HOST>:<PORT>/<PDB_NAME>
```

```
sqlplus <MIGRATION_USER>/<MIGRATION_USER_PASSWORD>@//<HOST>:<PORT>/<SID_OR_SERVICE>
```

1. Verify the migration user can read data from the source tables:

```
SELECT * FROM <SCHEMA_NAME>.<TABLE_NAME> WHERE ROWNUM <= 5;
```

```
SELECT * FROM <SCHEMA_NAME>.<TABLE_NAME> WHERE ROWNUM <= 5;
```

Validating your database and user configurations

Ensure your database configurations and created user meet all requirements to function with the agents:

1. Navigate to the EDB DMS agent folder in `/opt/cdcagent/reader/` .
2. Create an array that contains the name of all the tables that need to be migrated in `<schema_name>.<table_name>` format. For example:

```
arr=(test1.table1 test1.table2 test1.table3 test1.table4)
```

1. Run the script with the necessary parameters. For example:

```
ODB_SYSDBA_USERNAME=sys ODB_SYSDBA_PASSWORD=password ODB_HOST=localhost ODB_PORT=1521␣
↪ORACLE_SERVICENAME=ORCLPDB1 ODB_DBZ_USERNAME=C##MIG_USER2 ODB_DBZ_PASSWORD=migrator_
↪password LOG_MINER_TBS_NAME=LOGMINER_TBS ./oracleConfigValidation.sh "${arr[@]}"
```

> **Note**
>
> Set *ORACLE_SERVICENAME* to the PDB service name. !!!

```
ODB_SYSDBA_USERNAME=sys ODB_SYSDBA_PASSWORD=password ODB_HOST=localhost ODB_PORT=1521␣
↪ORACLE_SERVICENAME=ORCL ODB_DBZ_USERNAME=MIG_USER2 ODB_DBZ_PASSWORD=migrator_
↪password LOG_MINER_TBS_NAME=LOGMINER_TBS ./oracleConfigValidation.sh "${arr[@]}"
```

> **Note**
>
> Set *ORACLE_SERVICENAME* to the SID or service name. !!!

You get a response similar to the following:

```
*** [Transporter] - Validate system user connectivity
[Pass] Provide username and password of system user can connect to database.

 *** [Transporter] - Validate debezium user connectivity
[Pass] Provide username and password of debezium user can connect to database.

 *** [Transporter] - Validate Archive Log mode
[Pass] Database log mode is Archive Mode.

 *** [Transporter] - Validate fast recovery area parameters
[Pass] Recovery area path was set.
[Pass] Recovery area size limit meets requirements.
[Pass] Recovery area remaining size meets requirements.

 *** [Transporter] - Validate if database supplemental logging is enabled
[Pass] Supplemental logging check for database is passed.

 *** [Transporter] - Validate if table supplemental logging is enabled
[Pass] All source tables enable supplemental logging.

 *** [Transporter] - Validate redo log count&size
[Pass] Redo log count meets the requirements.
[Pass] Redo log size meets the requirements.
[Pass] Redo logs are all active.

 *** [Transporter] - Validate if tablespace for migration with Limited Privileges␣
↪were created
[Pass] LOGMINER_TBS tablespace has enough datafiles.
[Pass] LogMiner DataFile Size meet the requirements.
[Pass] LogMiner DataFile Max Size meet the requirements.
[Pass] LogMiner DataFile are auto-extendable.

 *** [Transporter] - Validate if user for migration with Limited Privileges were␣
↪created
Oracle version is 19, which is 12c or newer.
[Pass] This user has all sys privileges that migration needs.
[Pass] This user has all role privileges that migration needs.
```

(continues on next page)

(continued from previous page)

```
[Pass] This user has all select privilege on specific tables that migration needs.
[Pass] This user has all execute privilege on specific tables that migration needs.

 *** [Transporter] - Validate if user for migration with LOGMINER_TBS tablespace
[Pass] User for migration use LOGMINER_TBS tablespace.
```

Your database is ready for CDC migration.

> **Note**
>
> Address any *[Failed]* or *[Suggestion]* statuses by modifying the source database settings as recommended here. *[Failed]* checks are blocking issues that you must resolve before the DMS can execute the data migration. *[Suggestion]* checks allow migration but can negatively affect performance if left unaddressed.

More information

- See the

Debezium Oracle Connector documentation to understand how Debezium captures changes from Oracle redo logs and the specific database configurations it requires.

- See Oracle's

Enabling the Fast Recovery Area .

Next step

Configuring the EDB Postgres AI agent to register your Oracle database

Configuring the EDB Postgres AI agent to register your Oracle database

Configure the EDB Postgres AI agent (packaged as `beacon-agent`) to establish a secure connection with HM and retrieve schema and migration assessment data from your Oracle source.

Prerequisites

- **Machine user and access key**: Created in *Creating a machine user* . Store the access key in an environment variable:

```
export BEACON_AGENT_ACCESS_KEY="baak_your_key_here"
```

- **Oracle Instant Client with SQL*Plus**: The EDB Postgres AI agent uses SQL*Plus to extract DDL from your Oracle source. Ensure you install Oracle Instant Client and SQL*Plus, and configure the required environment variables on your auxiliary machine. See *Configuring your auxiliary machine or bastion host* for instructions.

> **Note**
>
> The agent substitutes environment variables referenced in *beacon_agent.yaml* at runtime, so setting *$BEACON_AGENT_ACCESS_KEY* in your environment before running the agent is sufficient. Environment variables always take precedence over static values in the YAML.

Recommendations for setting environment variables

We recommend handling sensitive data, such as `$BEACON_AGENT_ACCESS_KEY` or `$DSN` , as environment variables managed through a secrets manager. This strategy prevents sensitive credentials from being stored in plaintext configuration files. If you use environment variables, make sure they're available to the EDB Postgres AI agent while it's running.

Consult your organization's IT department on the safest approach for production environments.

For short-lived migrations or testing and demo purposes, you can use the `export` command in an active terminal session.

Obtaining HM connection parameters

The agent requires specific values from your HM installation. Contact your administrator for:

- **Beacon server hostname**: The internal endpoint for the `beacon_server` .

When configuring the HM installation, administrators/installers must specify an internal endpoint for the `beacon_server` service. Ask the administrator or installer of your HM instance to provide the hostname (URL) they set for the `beacon_server` service in the Helm chart configuration file used for installation. For this, they'll look up the value they set for `parameters.upm-beacon.server_host` (or `BEACON_SERVICE_DOMAIN_NAME`) in the `values.yaml` file. You'll need this value later to configure the EDB Postgres AI agent.

- **Root certificate path (if required)**: If your HM instance uses a self-signed or internal CA, obtain the `ca_public.crt` file. Store it on the agent machine (For example, `/etc/beacon/ca_public.crt`).

The EDB Postgres AI agent requires a trusted connection to the `beacon_server` service in your HM instance. If the server running the EDB Postgres AI agent doesn't inherently trust HM's TLS certificate (managed by your organization's security infrastructure), ask the administrator or installer of your HM instance to provide the file with the certificate. Store this certificate on the machine running the EDB Postgres AI agent. You'll need to provide the directory path to this certificate later to configure the EDB Postgres AI agent.

See *Administrative tasks* for more information about the `beacon_server` and root certificate.

Verifying connectivity

Once you have the HM connection parameters, verify that your auxiliary machine can reach both the Oracle source and Hybrid Manager. The following ports must be open:

Destination	Port	Used by
Oracle host	1521	Both agents (database connection)
HM *beacon_server*	9443	EDB Postgres AI agent
HM DMS service (*RW_SERVICE_HOST*)	443 (HTTPS)	DMS agent

Verify connectivity to each host and port with `nc` :

```
#  Oracle Listener

nc -zv <oracle-host-name-or-ip> 1521

#  HM beacon server

nc -zv <beacon_server_hostname> 9443

#  HM DMS service
```

(continues on next page)

(continued from previous page)

```
nc -zv <rw-service-hostname> 443
```

A successful connection returns `succeeded!` . If you see `Connection refused` or `Operation timed out` , the port isn't reachable and must be opened before proceeding.

Defining source database connection parameters

The Data Source Name (DSN) is the connection string for your Oracle source.

Connect using a Common User (starting with C##) to a specific Pluggable Database (PDB) service.

> **Note**
>
> The # characters in the username must be URL-encoded as %23.

Format:

```
DSN="oracle://<C##_MIGRATION_USER>:<MIGRATION_USER_PASSWORD>@<HOST>:<PORT>/<PDB_
↪SERVICE_NAME>"
```

Example:

```
DSN="oracle://C%23%23_MIG:password@localhost:1521/ORCLPDB1"
```

Connect using a standard local user to the database instance SID or Service Name.

Format:

```
DSN="oracle://<MIGRATION_USER>:<MIGRATION_USER_PASSWORD>@<HOST>:<PORT>/<SERVICE_NAME_
↪OR_SID>"
```

Example:

```
DSN="oracle://MIG_USER:password@localhost:1521/ORCL"
```

```
DSN="oracle://oracle:pa%24%24word@localhost:1521/ORCLPDB1"
```

Preparing a configuration file

Prepare a configuration file for your agent. The agent searches for `beacon_agent.yaml` in `/etc/beacon/` or `${HOME}/.beacon/` .

1. Create an EDB Postgres AI agent configuration directory in your home directory:

```
mkdir ${HOME}/.beacon
```

> **Note**
>
> The agent searches for *beacon_agent.yaml* in */etc/beacon/* or *${HOME}/.beacon/* . As a final fallback, it searches the directory from which it's executed. !!!

1. Inside this directory, create a file `beacon_agent.yaml` :

```
touch ${HOME}/.beacon/beacon_agent.yaml
```

1. Copy and paste the following template into the new file.

In this template, `$BEACON_AGENT_ACCESS_KEY` and `$DSN` are the environment variables you configured previously. Replace all < > placeholders in the template. See *EDB DMS agent parameter reference* for details.

```
---
agent:
  access_key: $BEACON_AGENT_ACCESS_KEY
  beacon_server: <beacon_server_hostname>:9443
  project_id: <your_project_id>
  providers:
    - "onprem"
  root_ca_path: ""
  # In most HM deployments, you need to set this to the path of the ca_public.crt
  # file obtained from your HM administrator (for example, /etc/beacon/ca_public.crt).
  # Leave empty only if the HMs certificate is already trusted by the systems CA␣
↪bundle.
  schema_providers:
    - "onprem-schema"
    # Include "onprem-schema" to enable schema ingestion and migration assessment.
    # Remove this section if you dont want schema information to be collected.
provider:
  onprem:
    databases:
      - resource_id: "<database_resource_identifier>"
        # Controls how the auxiliary machine appears in HM Estate.
        dsn: $DSN
        schema:
          enabled: true
          poll_interval: 15s
        tags:
          - "<tag_names>"
general:
  logging:
    level: info
    # Use debug during initial setup or troubleshooting for more verbose output.
```

1. Save the file.

Running the EDB Postgres AI agent

1. Ensure the Oracle Instant Client environment variables are set in the same shell session where you'll run the agent:

```
export ORACLE_HOME=</path/to/instantclient>
export PATH=$ORACLE_HOME:$PATH
export LD_LIBRARY_PATH=$ORACLE_HOME:$LD_LIBRARY_PATH
```

1. Before starting the service, verify connectivity and syntax:

```
beacon-agent validate
```

1. Run the agent:

```
beacon-agent
```

> **Tip**
>
> See agent CLI for other agent modes and options. !!!

1. Monitor the ingestion process as the agent starts sending data to Hybrid Manager (HM). After a few minutes, your Oracle database appears in the HM console under **Estate > Migrations**.

> **Important**
>
> The agent dynamically detects changes to tables and columns. However, changes to views, procedures, or functions require an agent restart to re-ingest metadata.

For production use, see *Running the agent as a service* .

Verifying the migration assessment

To confirm your Oracle database was recognized by the agent and is ready for migration, go to **Estate > Migrations** in the HM console.

If the database appears in the table, you have access to the following resources:

- A **migration assessment** on the *Migrations* of the **Estate** page, providing a summarized view of database schema count, core count, migration complexity, and level of effort.
- A ** *Viewing migration assessment details* ** accessible by selecting a specific database from the **Migrations** tab. This view provides an expanded analysis of **Schemas**, a global **Assessment**, and a **Migration** hub with direct connections to other HM tools for performing migrations.
- **Automatic schema DDL extraction** that you can use for schema migrations with the built-in Migration Portal. Once ingestion is complete, *Migration Portal automatically creates a project* per database, which you can use to convert incompatible schemas and migrate them to the destination database.

Troubleshooting

There are several reasons why the agent might fail to connect to HM or your Oracle database. Here are some tools you can use for troubleshooting.

`beacon-agent validate` command

The `beacon-agent validate` command is a helpful step to identify common connectivity and configuration issues.

```
beacon-agent validate
```

The validate command checks for the following:

- **Server connectivity**: Verifies the agent can connect and authenticate to the configured `beacon_server` (usually on port `9443`).

- **Database connectivity**: Attempts to connect to each configured database DSN. This is a full connection test, not just a syntax or URL-encoding check.
- **Configuration display**: Shows TLS settings, project ID, providers, and per-database settings to help identify misconfiguration.

`beacon-agent` logs

If the agent reports errors during ingestion, you can check the logs for more details. If you're running the agent in the foreground, logs will be printed to the terminal. If you set up the agent as a systemd service, use `journalctl` to view logs.

Here are some common error messages you might encounter in the logs:

Error	Cause	Fix
sqlplus is not installed or not found in PATH	Oracle Instant Client environment variables are not set or set incorrectly in the current session.	Set *PATH* and *LD_LIBRARY_PATH* to the Instant Client root directory and verify with *which sqlplus*. See *Configuring your auxiliary machine or bastion host* .
extractor permissions check failed: sqlplus execution failed: exit status 127	*sqlplus* binary is found but cannot load its shared libraries.	Ensure *LD_LIBRARY_PATH* points to the Instant Client root directory (not a *lib/* subdirectory).
tls: failed to verify certificate: x509: certificate signed by unknown authority	The agent cannot verify HM's TLS certificate because the CA is not trusted by the system.	Set *root_ca_path* in *beacon_agent.yaml* to the path of the *ca_public.crt* file obtained from your HM administrator. If you cannot obtain the certificate, you can set *plaintext: true* in the agent config as a temporary workaround (not recommended for production).
rpc error: code = Unknown desc = no rows in result set	The beacon-agent version is incompatible with the server.	Upgrade to beacon-agent v1.384.0 or later.

Running the agent as a service

For continuous metric streaming and to ensure your Oracle database remains visible in **HM console > Estate**, configure the agent to run as a systemd service. Running as a service ensures it starts automatically on system boot and restarts after any errors.

Without a service, the agent streams metrics only while you run it manually. Your database will disappear from the HM estate 72 hours after the agent stops streaming.

> **Note**
>
> Ensure that any environment variables referenced in *beacon_agent.yaml* (for example, *$BEACON_AGENT_ACCESS_KEY* and *$DSN*) are accessible to the service at runtime.

1. Create the service file. This example uses `vi` , but you can use any text editor:

```
sudo vi /usr/lib/systemd/system/beacon-agent.service
```

Default service file paths by operating system:

Operating system	Default path
RHEL	*/usr/lib/systemd/system/beacon-agent.service*
Rocky Linux	*/usr/lib/systemd/system/beacon-agent.service*
AlmaLinux	*/usr/lib/systemd/system/beacon-agent.service*
Ubuntu	*/etc/systemd/system/beacon-agent.service*
Debian	*/lib/systemd/system/beacon-agent.service*

1. Paste the following template into the file and replace all < > placeholders:

```
[Unit]
Description=Agent
After=network.target

[Service]
Type=simple
User=<user>
WorkingDirectory=</home/user/>
Environment="PATH=</path/to/oracle/instantclient>:</user/binaries>:</system/binaries>"
Environment="ORACLE_HOME=</path/to/oracle/instantclient>"
ExecStart=</usr/local/bin/beacon-agent>
Restart=on-failure
RestartSec=60

[Install]
WantedBy=multi-user.target
```

Setting	Placeholder	Description
User=	*<user>*	System user that runs the agent.
WorkingDirectory=	*</home/user/>*	Directory where the *.beacon* folder is stored. The path depends on the operating system.
Environment=	*"PATH=</path/to/oracle/instantclient>:..."*	All directories containing Oracle executables, so the agent can locate *sqlplus* for schema extraction at runtime.
Environment=	*"ORACLE_HOME=</path/to/oracle/instantclient>"*	Root directory of the Oracle Instant Client, required by *sqlplus* for schema extraction.
ExecStart=	*</usr/local/bin/beacon-agent>*	Full path to the *beacon-agent* binary. The path depends on how and where the agent was installed.

1. Reload systemd, then enable and start the service:

```
sudo systemctl daemon-reload
sudo systemctl enable beacon-agent.service
sudo systemctl start beacon-agent.service
```

1. Confirm the service started successfully by checking the logs:

```
journalctl -u beacon-agent.service
```

Related topics

- For more granular control of schema ingestion (for example, to exclude or include schemas), see *Controlling schema ingestion* .
- If your administrator needs guidance on how to obtain the values for the root certificate and `beacon_server` , see *Administrative tasks* .

Next step

Preparing your HM-managed destination database

Preparing your HM-managed destination database

Prepare an HM-managed database cluster to serve as the destination for your Oracle migration.

> **Note**
>
> The destination database cluster must use the **EDB Postgres Advanced Server** distribution. HM's built-in schema migration features — Migration Portal and EDB DMS — require EDB Postgres Advanced Server when migrating from Oracle.

If you don’t have a destination database cluster on Hybrid Manager (HM) yet, create one:

1. In the HM console, select the "Can't select a project" — the portal domain defaults to the API endpoint where you want to create the database cluster.
2. In your project, select **Create New** and then **Database Cluster** to *Create a database cluster template* . When prompted to select a distribution, choose **EDB Postgres Advanced Server**.

> **Note**
>
> If you're creating an EDB Postgres Advanced Server 18 or later cluster, you must update the *ssl_groups* database parameter to remove the *X25519* group from the default value. *X25519* isn't FIPS-compliant, and Migration Portal's connection to the destination database requires FIPS. !!!

1. In the **Clusters** page, select your cluster, and use the **Quick Connect** option to access your database from your terminal.
2. Create an empty database to use as a destination for the migration. Alternatively, you can use the default database `edb_admin` .

If you already have an HM-managed EDB Postgres Advanced Server database cluster, you can use it as the destination for your migration.

Before proceeding, ensure:

- The cluster uses the **EDB Postgres Advanced Server** distribution. If it uses a different distribution, create a new EDB Postgres Advanced Server cluster instead.
- The tables you migrate and the load generated during migration don’t interfere with existing workloads on the cluster.

Next step

Migrating Postgres schemas

Migrating schema from Oracle to an HM-managed database

Before migrating data, your schema must be present on the destination database. There are two ways to handle this, and both start with opening your Migration Portal project and resolving any compatibility issues:

- **Simplified (recommended)**: Use Migration Portal to resolve incompatibilities only, then let HM import the schema as part of the migration. HM manages constraint ordering automatically — no manual constraint handling required. Select **Schema and data** as the migration scope when creating the migration.
- **Manual**: Use Migration Portal to resolve incompatibilities and import the schema directly to the destination. You must manage constraint ordering yourself and reapply deferred constraints after data migration completes. Select **Data only** as the migration scope when creating the migration.

A third scope option, **Schema only**, is also available when creating a migration in HM. Use this if you want to migrate the database structure without migrating any table data — for example, to validate the schema on the destination before committing to a full migration.

Opening your Migration Portal project

The EDB Postgres AI agent automatically extracts the DDL from your Oracle source and makes it available in Migration Portal. Use Migration Portal to convert and migrate your Oracle schemas.

Prerequisites

- You *Configuring the EDB Postgres AI agent to register your Oracle database* via the EDB Postgres AI agent.
- Optional: Your HM administrator *enabled and configured the AI Copilot* to help resolve schema incompatibilities in Migration Portal.

Accessing your project

After registering your Oracle database with the EDB Postgres AI agent, a Migration Portal project is **created automatically** — one per registered database, based on the `resource_id` in your `beacon_agent.yaml` .

> **Warning**
>
> Any DDL changes made after the Migration Portal project was created — including new schemas, new tables, updated tables, or deleted tables — are not reflected in the auto-created project.

To find your project:

1. Navigate to your Oracle database's detailed view. You can get there from either of these paths:

a. **Estate > Migrations > [your database]**.

b. **[your project] > Migrate > Databases > [your database]**.

1. Select the **Schemas** tab.
2. Select the project link in the **MP Project** column to open the Migration Portal's project-scoped workspace for this database.

If no project link appears in the **MP Project** column, the project wasn't created automatically. Expand the section below to create one manually.

Creating a Migration Portal project manually

Automatic project creation won't trigger if:

- The ingested DDL exceeds the 300 MB default limit (or an administrator-configured *customized limit*).
- Your Hybrid Manager administrators *disabled automatic project creation* .

In these cases, create a project manually:

1. In the **Estate** page, select the **Migrations** tab, then select your database.
2. In the detailed view, select the **Schemas** tab, then select **New MP Project** or the plus sign next to a schema.

> **Tip**
>
> The **Dependencies** column shows which schemas the current one relies on. Group a schema with all its dependencies in a single project to ensure referenced objects exist in the destination. !!!

1. Enter the project details and select **Continue**.
2. Select the schemas to include and select **Create New Project**.

> **Note**
>
> You can't add schemas to the project after creation. Use a separate project to migrate additional schemas to the same destination. !!!

HM loads the schemas into Migration Portal in the background. Once ready, a project link appears — select it to open Migration Portal and begin resolving incompatibilities using the AI Copilot or QuickHelp.

Converting schema and resolving compatibility issues

Use Migration Portal to convert your Oracle DDL to be compatible with EDB Postgres Advanced Server. Use the AI Copilot and QuickHelp to resolve any compatibility issues that arise during conversion. Additional resources:

- Using Migration Portal
- *Enabling and configuring the AI Copilot in HM* (administrator task)
- Migration Portal's AI Copilot
-

Performing a schema assessment

> **Note**
>
> Migrated PL/SQL bodies (triggers, procedures, functions, and packages) retain unqualified object references from the Oracle source. Review these references in your converted DDL before importing to avoid runtime failures on the destination. See *Unqualified object references in migrated PL/SQL bodies* for details and workarounds.

Importing schema

After resolving all incompatibilities in Migration Portal, proceed to *Configuring and running the EDB DMS agent* . When you create the migration, select **Schema and data** as the scope — HM imports the converted schema with constraints applied in the correct order, and automatically reapplies deferred constraints after data migration completes.

After resolving all compatibility issues in Migration Portal, import the schema into the destination database.

Handling constraints

Import constraints in two stages to avoid performance degradation during migration:

- **Include now** (before data migration): primary key, unique, and `NOT NULL` constraints. EDB DMS requires primary key and unique constraints to guarantee exactly-once delivery. `NOT NULL` constraints have negligible performance impact.
- **Exclude now, reapply later** (after data migration): foreign key, check, and exclusion constraints. These cause unnecessary CPU and memory overhead while data is actively being migrated.

When following the steps below, disable **Constraints** in Migration Portal and keep a record of what you excluded so you can reapply them after the migration completes.

> **Note**
>
> For tables without primary key or unique constraints, EDB DMS provides at-least-once delivery only. You can perform deduplication during migration verification.

Use this option when you don't need to rename any schemas, tables, or columns.

1. Select **Migrate to** > **Online migration**.
2. Select your schemas. In the object list beneath the database, disable **Constraints**. Select **Next**.
3. Enter the details of your destination cluster.

Find your cluster's information under **Clusters > Postgres Clusters > [cluster name] > Connect**.

1. Select **Test connection** to verify connectivity to the destination, then select **Save and continue**.
2. Select **Next** and wait for the import to complete.
3. Connect to the destination database and confirm the schemas are present.

Use this option when you need to map schemas, tables, or columns to different names in the destination.

1. Select **Migrate to** > **Offline migration**.
2. Select your schemas. In the object list beneath the database, disable **Constraints**. Select **Next**.
3. Select **Download SQL file**.
4. Edit the object names directly in the downloaded DDL. Ensure that renamed objects are also updated in any dependent views, constraints, or other references. See *Mapping schemas, tables, and columns* for details.
5. Connect to the destination database and apply the modified DDL file.

Related topics

- *Migration Portal's project-scoped workspace*

Next step

Configuring and running the EDB DMS agent

Configuring and running the EDB DMS agent

The EDB DMS agent connects to your Oracle source and works with the Data Migration Service (DMS) to perform data and schema migrations. It must be running for **Data Only**, **Schema Only**, and **Schema and Data** migrations. Before you can run it, you need to create migration credentials in HM and pass them to the agent alongside your Oracle connection details.

Creating migration credentials

Migration credentials authenticate the DMS agent to Hybrid Manager (HM). You need one set of credentials per HM project.

1. In the HM console, select the project you're using for your migration.
2. Select **Migrate > Credentials**.
3. Select **Create Migration Credential**, enter a name, and select **Create Migration Credential**. A `credentials.zip` file is downloaded to your machine.

> **Note**
>
> Store the credentials safely. If you lose them, you'll need to revoke this set and create new ones. !!!

1. Unzip the downloaded file and copy the credentials folder to the auxiliary machine where you installed the DMS agent — for example, to `$HOME/credentials/` . If you use a different path, update `CREDENTIAL_DIRECTORY_PATH` in the script to match.

The folder contains:

- `int.crt`
- `int.truststore.p12`
- `client.keystore.p12`
- `client-key.pem`
- `client-cert.pem`

Configuring the DMS agent

1. Open `/opt/cdcagent/run-cdcagent.sh` with write permissions.
2. Set the required variables. Optional parameters are commented out by default — uncomment and configure any that apply to your environment.

See the *EDB DMS agent parameter reference* for details.

```
#!/bin/bash -e
# run-cdcagent.sh
#
# This script provides a convenient place to specify
# environment variables used to configure the
# EDB Data Migration Service Agent.
#

########################################
# DMS Agent General Configuration      #
```

(continues on next page)

(continued from previous page)

```
########################################

# This ID is used to identify DMS Agent
# and is specified by the user.
export DBCONFIG_ID=<source_name_of_choice>

# Determine the run mode of the DMS Agent
# The run mode can be set to either reader or writer
# reader mode is used to read changes from a source database
# writer mode is used to write changes to a target database
export RUN_MODE=reader

# This is the DMS backend service used by the Agent
# Obtain this value from the HM console Sources or Destinations migration pages
export RW_SERVICE_HOST=<rw_service_host>

########################################
# DMS Agent TLS Configuration          #
########################################

# You need to create migration credentials in EDB Postgres AI platform
# and set the path of the credentials directory
export CREDENTIAL_DIRECTORY_PATH=$HOME/credentials

########################################
# DMS Agent DB Configuration           #
########################################

# You can increase the index to configure more than
# one database for the DMS Agent
export DBCONFIG_DATABASES_0__TYPE=ORACLE
export DBCONFIG_DATABASES_0__HOSTNAME=<oracle_host>
export DBCONFIG_DATABASES_0__PORT=1521

# For CDB/PDB: use CDB_NAME/PDB_NAME (for example, ORCLCDB/ORCLPDB1).
# The agent splits on "/" to identify the CDB and PDB separately.
# Using only the PDB name will cause it to be treated as the CDB, leaving the PDB␣
↪unset.
# For non-CDB: use the SID or service name (for example, ORCL)
export DBCONFIG_DATABASES_0__CATALOG=<CDB_NAME/PDB_NAME or SID>

# Use your Oracle migration user (not SYS)
export DBCONFIG_DATABASES_0__USERNAME=<migration_user>

# The password env can be set without specifying it here
# but the env structure looks like this
export DBCONFIG_DATABASES_0__PASSWORD=<password>

# Required for Schema and data or Schema only scope.
# Must match resource_id in beacon_agent.yaml.
export DBCONFIG_DATABASES_0__RESOURCEID=<resource_id>
```

(continues on next page)

(continued from previous page)

```
# Table filters for databases, only tables passing the filter will be discovered and␣
↪therefore can be chosen for migration:
# - Format: a comma-separated list of tokens in the form `schema.table`
# - Wildcard: use `*` for schema or table (for example: `SCHEMA1.*` or `*.TABLE1`)
# - Semantics:
#    * If `includeList` is absent/empty -> all tables are included
#    * If `includeList` is present -> only tables matching at least one include␣
↪pattern are considered
#    * If `excludeList` is absent/empty -> no table is excluded
#    * If `excludeList` is present -> only tables matching at least one exclude␣
↪pattern are considered
#    * `excludeList` has higher priority: a table matching any exclude pattern is␣
↪always excluded
# Examples:
#export DBCONFIG_DATABASES_0__INCLUDELIST=SCHEMA1.*,SCHEMA2.TABLE1
#export DBCONFIG_DATABASES_0__EXCLUDELIST=SCHEMA1.IGNORED_TABLE

# SSL mode: STRICT (verify-full), DEFAULT (require), WEAK (prefer/allow plaintext)
#export DBCONFIG_DATABASES_0__SSLMODE=DEFAULT

# TCP connection settings (for dead connection detection)
# Socket timeout in seconds (default: 600 for writer, 60 for reader)
#export DBCONFIG_DATABASES_0__SOCKETTIMEOUT=600
# Enable TCP keepalive probes (default: true)
#export DBCONFIG_DATABASES_0__TCPKEEPALIVE=true
# Writer-mode TCP keepalive timing for long-running operations
# used during PK application and post-snapshot deduplication
# Seconds before first keepalive probe (default: 60)
#export DBCONFIG_DATABASES_0__TCPKEEPALIVESIDLE=60
# Seconds between keepalive retries (default: 10)
#export DBCONFIG_DATABASES_0__TCPKEEPALIVESINTERVAL=10
# Max failed probes before connection is dead (default: 6)
#export DBCONFIG_DATABASES_0__TCPKEEPALIVESCOUNT=6

# Escape hatch for arbitrary JDBC driver properties (format: key=value;key=value)
# These are applied after defaults but before explicit settings (SSLMODE,␣
↪SOCKETTIMEOUT, etc.)
# Oracle examples: oracle.net.encryption_client=REJECTED;v$session.program=DMSAgent
#export DBCONFIG_DATABASES_0__JDBCOPTIONS=oracle.net.encryption_client=REJECTED;v
↪$session.program=DMSAgent

# You can increase the index to configure more than one database for the DMS Agent:
#export DBCONFIG_DATABASES_1__TYPE=ORACLE
#export DBCONFIG_DATABASES_1__HOSTNAME=<oracle_host>
#export DBCONFIG_DATABASES_1__PORT=1521
#export DBCONFIG_DATABASES_1__CATALOG=<pdb_or_sid>
#export DBCONFIG_DATABASES_1__USERNAME=<migration_user>
#export DBCONFIG_DATABASES_1__PASSWORD=<password>
#export DBCONFIG_DATABASES_1__RESOURCEID=<resource_id>

########################################
# Optional Parameters Below            #
```

(continues on next page)

(continued from previous page)

```
##########################################

# FIPS mode can be enabled by setting the FIPS_MODE env variable to true
#export FIPS_MODE=true

# Enable transformation for null character(0x00) in Oracle text-based types: VARCHAR2,
↪ LONG, CLOB, NCLOB
#export TRANSFORMATION_ENABLE_NULLCHAR_REPLACEMENT=false
# Customize the replacement string for null character(0x00)
#export TRANSFORMATION_NULLCHAR_REPLACEMENT=""
# Enable transformation for special characters in column names such as: $ and -
#export TRANSFORMATION_ENABLE_SPECIAL_COLUMN_NAME_ENCODING=false

# Configure logging
# Loglevel for the entire Quarkus application and underlying components
# WARNING: Enabling this loglevel is extremely verbose
#export QUARKUS_LOG_LEVEL=DEBUG
# Loglevel for the EDB package
# Preferred mode for debugging issues related to the logic of the agent
#export QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL=DEBUG

# Passthrough debezium configuration properties
# You can set any Debezium configuration property by prefixing it with DBZ_
# for example, to change max.batch.size:
# debezium supported properties can be found here:
# for oracle: https://debezium.io/documentation/reference/stable/connectors/oracle.
↪html#oracle-connector-properties
# for postgres: https://debezium.io/documentation/reference/stable/connectors/
↪postgresql.html#postgresql-connector-properties
#export DBZ_MAX_BATCH_SIZE=204800

cd "$(dirname "$0")"

echo "Starting in mode: $RUN_MODE"

case "$RUN_MODE" in
  reader)
    java -jar ./reader/quarkus-run.jar
    ;;
  writer)
    java -jar ./writer/quarkus-run.jar
    ;;
  *)
    echo "Unknown RUN_MODE: $RUN_MODE"
    exit 1
    ;;
esac
```

Running the DMS agent

Navigate to the DMS agent directory and run the configuration script:

```
cd /opt/cdcagent
./run-cdcagent.sh
```

The script exports all required environment variables and starts the agent process. Once running, the agent registers your Oracle database as a source in HM. Continue to the next section to verify it appears correctly.

Verifying the source in HM

Once the agent is running, verify that your Oracle database appears as a source in HM:

1. In the HM console, select your project.
2. Select **Migrate > Sources**.

Your Oracle database should be listed with its name, type, agent ID, and status. For a description of all fields on this page, see *Understanding deployment resources* .

Next step

Migrating data from an AWS RDS or Aurora PostgreSQL database to an HM-managed database

Migrating data from Oracle to an HM-managed database

With the EDB DMS agent running and your schema either imported or ready for HM to import, you're ready to create a migration in the HM console. This page walks through the full migration lifecycle: creating the migration, monitoring its progress, finalizing it, and handling post-migration constraints.

The steps below vary depending on the approach you chose on the previous page. Select the tab that matches your choice.

Creating a migration

A migration establishes a sync between your Oracle source and a destination database cluster managed by HM.

1. In the HM console, select the project you're using for the migration.
2. Select **Migrate > Migrations**.
3. Select **Create Migration**.
4. Assign a meaningful name to the migration — for example, `oracle-prod-migration` .
5. Select the source of the migration. This corresponds to the `DBCONFIG_ID` you set when configuring the DMS agent.
6. In **Scope**, select the option that matches your approach:

Select **Schema and data**. HM imports the converted schema from the resolved Migration Portal project, creates tables without constraints to keep the initial snapshot fast, applies primary key and unique constraints after the snapshot completes (required for exactly-once delivery), and applies the remaining schema either concurrently with streaming (via **Finalize Schema Concurrently**) or automatically in the post-cutover **Schema Finalization** stage. See *Finalizing a streaming migration* .

Select **Data only**. The schema was already imported to the destination via Migration Portal.

> **Note**
>
> You can use **Schema only** if you want to migrate the database structure without migrating any table data. !!!

1. Select the migration type:

- ***Snapshot*** — copies data once and completes. Use this for a one-time, offline migration.
- ***Snapshot + Streaming*** — copies an initial snapshot, then continuously streams changes from the source. Use this to minimize downtime.

1. Under **Destination Type**, select **HM-managed Postgres** and select your destination database cluster and database. Then select **Next**.
2. On the **Review your Tables** tab, select the tables and columns to include in the migration. Expand a table to view its columns and optionally *map source columns to different destination column names* . Column mapping is only available for the **Data only** scope.
3. Select **Create Migration**.

The HM console displays the *new migration* . If you chose **Snapshot + Streaming**, the EDB DMS agent streams data changes continuously while the migration shows a **Running** state.

> **Note**
>
> The EDB DMS agent streams data changes only — not DDL changes (schema modifications).

Monitoring progress

Migration list view

The **Migrations** page shows a summary of all migrations in the project. The **Migration progress** column displays three metrics per migration:

- **Total number of tables** — total tables included in this migration
- **Snapshot tables completed** — tables that have finished the snapshot stage
- **Migrated rows** — cumulative rows migrated across all tables

Migration details view

Select a migration from the list to open the details page, which has three tabs:

- **Overview** — shows the current migration state and general information about the source and destination databases. For **Snapshot + Streaming** migrations, the migration is caught up when the status is **Streaming**. For **Snapshot** migrations, it's finished when the status is **Completed**.
- **Tables** — shows all tables selected for migration. For each table, you can see source and destination table and schema names, snapshot status, mapped columns, migrated rows, and migration rate. Select the + icon to expand a table and view column-level mappings.
- **Error history** — lists all errors logged for this migration. Select the **i** icon to open the full error log for a specific error. For stuck or failed migrations, active errors here indicate where to start troubleshooting.

DMS agent logs

The HM console doesn't surface all DMS agent errors. If a migration is stuck or no tables are progressing, check the terminal where you ran `run-cdcagent.sh` for `ERROR` log entries — these often contain the root cause. Common issues include version incompatibility between the DMS agent and HM, and connectivity or credential failures.

Finalizing a streaming migration

> **Note**
>
> This section applies only to **Snapshot + Streaming** migrations. For **Snapshot** migrations, the DMS marks the migration as completed automatically.

Monitoring catch-up before cutover

Before stopping writes on the source, give the destination time to catch up. Use the **Migration details** page to track row counts and replication lag. Begin the cutover only after the destination is consistently close to the source.

Finalizing schema concurrently (recommended)

If you used the **Schema and data** scope, HM can apply most schema objects while the stream is still running: primary keys, unique constraints, indexes (built with `CREATE INDEX CONCURRENTLY` , so live writes aren't blocked), foreign keys (added as `NOT VALID`), check constraints, `NOT NULL` constraints, and similar. Foreign key validation, sequence synchronization, and other lightweight tasks run later in the post-cutover **Schema Finalization** stage.

Running this work concurrently shrinks the read-only cutover window. The larger your tables and the more indexes they have, the greater the time saved.

Once every table has completed its snapshot and the migration enters the streaming phase, the **Migration details** page shows a **Schema Finalization Available** alert with a **Finalize Schema Concurrently** button. The button stays available throughout streaming, so you can pick the moment based on how close the destination is to catching up.

1. Select **Finalize Schema Concurrently** on the **Migration details** page.
2. HM applies the deferred objects while streaming continues. The migration remains in the **Running** state, with a progress sub-step labeled **Finalizing Schema Concurrently**.

When the sub-step completes, proceed to the cutover.

> **Note**
>
> **Finalize Schema Concurrently** is available only for migrations created with the **Schema and data** scope with streaming enabled.

If you skip this step, the same objects are applied serially after you mark the migration as completed. See *Schema finalization stage* . The cutover window is larger in that case because indexes are built non-concurrently and constraints are validated under exclusive locks.

Not applicable to the manual path. You applied the schema before migration and will reapply deferred constraints manually after cutover. See *Applying deferred constraints and verifying data* .

Performing the cutover

When you're ready to switch traffic from the source to the destination:

1. Stop all writes on the source database.
2. Verify in the HM console that the destination has applied the last source change.
3. In the HM console, select **Migrate > Migrations** in your project.
4. On the **Migration details** page, select **Mark as Completed**.

This stops the streaming process.

Schema Finalization stage

For migrations created with the **Schema and data** scope, the migration enters a **Schema Finalization** stage automatically after **Mark as Completed**. What runs in this stage depends on whether you ran concurrent finalization first:

- **If you ran concurrent finalization**: HM runs the lightweight remaining work — foreign key validation, sequence sync, and any other deferred metadata tasks. This stage is short.
- **If you skipped concurrent finalization**: HM applies the full deferred set now (primary keys, unique constraints, indexes, foreign keys, check constraints, `NOT NULL` , and similar) followed by the lightweight tasks above. This stage takes substantially longer and runs while the source is read-only.

The migration is fully complete only after the **Schema Finalization** stage finishes. Progress is shown on the **Migration details** page.

No automatic schema finalization runs for the **Data only** scope. Continue to *Applying deferred constraints and verifying data* to reapply the foreign key, check, and exclusion constraints you deferred before migration.

Applying deferred constraints and verifying data

Reapplying foreign key, check, and exclusion constraints

HM applied all deferred constraints and synced sequences automatically during the **Schema Finalization** stage. No manual action is needed. See *Schema finalization stage* .

During the schema import, you excluded foreign key, check, and exclusion constraints to avoid performance overhead during data migration. Now that migration is complete, connect to the destination database and reapply those constraints.

Primary key, unique, and `NOT NULL` constraints were already applied before migration and don't require any action.

Deduplicating rows when needed

For tables that had no primary key or unique constraints, the EDB DMS agent provides at-least-once delivery, which can result in duplicate rows. Perform deduplication as part of your verification step.

Verifying migrated data

Compare the source and destination databases to confirm all data was migrated successfully. You can use LiveCompare for this.

Assessing your application SQL before cutover

With the database migration complete, the next step before switching your application to EPAS is to confirm that the SQL queries your application executes are compatible with the target database. Even a successful schema and data migration doesn't guarantee that application-level SQL — embedded in your code, ORM, or scripts — will work without changes.

Use HM's application assessment to identify incompatible queries and review suggested transformations before application cutover. See *Application assessment* for an overview, or go directly to *Creating an application record* to get started.

Related topics

- LiveCompare — row-level data comparison tool for verifying migration completeness
- *Terminology* — definitions for snapshot, streaming, and other migration concepts
- *Application assessment* — assess application SQL compatibility with EPAS before cutting over your application

Migrate your AWS RDS/Aurora PostgreSQL database to an HM-managed database

These are the tasks involved in an end-to-end migration from an AWS RDS for PostgreSQL or Aurora PostgreSQL database to a database cluster managed by Hybrid Manager (HM). The process uses two agents: the EDB Postgres AI agent (`beacon-agent`) to register your database with HM and ingest schema for assessment, and the EDB DMS agent (`cdcagent`) to perform the actual data migration.

1. ** *Configuring your auxiliary machine or bastion host* ** — Configure the machine that hosts both migration agents. This machine acts as the bridge between your AWS RDS or Aurora PostgreSQL source and HM.
2. ** *Preparing your AWS RDS/Aurora PostgreSQL source database* ** — Create the required users, configure schema extraction permissions, enable logical replication via an AWS DB cluster parameter group, and validate your configuration.
3. ** *Configuring the EDB Postgres AI agent to register your AWS RDS/Aurora PostgreSQL database* ** — Connect the agent to your source database and to HM to enable schema ingestion and migration assessment.
4. ** *Preparing your HM-managed destination database* ** — Create or identify the HM-managed database cluster that will receive the migrated schema and data.
5. ** *Migrating Postgres schemas* ** — Choose between the simplified path, where HM handles schema migration and constraint ordering automatically, or the manual path, where you export, prepare, and import the schema yourself.
6. ** *Configuring and running the EDB DMS agent* ** — Create migration credentials, configure the DMS agent in reader mode to read changes from your AWS RDS or Aurora PostgreSQL source, and verify the source appears in HM.
7. ** *Migrating data from an AWS RDS or Aurora PostgreSQL database to an HM-managed database* ** — Create and run the data migration in the HM console, monitor its progress, finalize streaming, and apply post-migration constraints.

Configuring your auxiliary machine or bastion host

The EDB Postgres AI agent (`beacon-agent`) and the Data Migration Service (DMS) agent (`cdcagent`) can run on the Postgres database server itself or on a separate auxiliary machine or bastion host. In most production environments, running the agents on a dedicated auxiliary machine is preferred — it avoids installing additional software on the database server, and a single agent instance can connect to databases on multiple servers.

Required software

Ensure you install the following software on your auxiliary machine before running the agents:

- **Java 17 or higher** — required by the DMS agent (`cdcagent`). Verify with `java -version` .
- **EDB Postgres AI agent** (`beacon-agent`) — required for database registration, schema assessment, and schema migration. Check the *beacon-agent version compatibility matrix* . Then install a compatible version of the EDB Postgres AI agent .
- **DMS agent** (`cdcagent`) — handles data migration and change data capture (CDC). Before installing, check the *DMS agent version compatibility* . Then *install a compatible version of the DMS agent* .

Configuring the environment

When you enable schema ingestion, `beacon-agent` uses `pg_dump` to extract DDL from your Postgres source. Postgres client tools packages include the `pg_dump` utility — it might already be available on your machine. Verify first:

```
pg_dump --version
```

If the command isn't found, install the client tools that match your source database distribution.

Once installed, ensure your `pg_dump` binary meets two compatibility rules:

- **Version rule** — The `pg_dump` major version must be greater than or equal to the major version of your source database. For example, if your source runs Postgres 16, you need `pg_dump` 16 or later. The agent validates this at startup and reports an error if the versions are incompatible.
- **Distribution rule** — The `pg_dump` binary must come from the same Postgres distribution as the source database. Each distribution's `pg_dump` accesses distribution-specific catalog tables. Using a mismatched binary (for example, community PostgreSQL `pg_dump` against an EDB Postgres Advanced Server source) will fail because the expected catalog objects won't exist.

Managing multiple Postgres versions on a host

If you have multiple Postgres versions on the same machine, ensure the correct `pg_dump` appears first in your `PATH` :

```
which pg_dump
pg_dump --version
```

On Debian and Ubuntu systems, use `update-alternatives` to manage the active version:

```
sudo update-alternatives --config pg_dump
```

Registering databases from multiple distributions or versions

A single `beacon-agent` instance can register databases from different Postgres distributions (for example, community PostgreSQL and EDB Postgres Advanced Server) or different major versions at the same time. Because the version and distribution rules apply per database, each database connection may require a different `pg_dump` binary.

Install all the required `pg_dump` binaries on the auxiliary machine. Later you can use the `pg_bin_path` setting in your `beacon_agent.yaml` configuration file to point each database entry to the correct binary directory. When set, it overrides the system `PATH` lookup for that specific database. When omitted, the agent falls back to the first `pg_dump` found in `PATH` . For example:

```
provider:
  onprem:
```

(continues on next page)

(continued from previous page)

```
  databases:
    - resource_id: "pg16-community"
      dsn: $DSN_PG16
      pg_bin_path: /usr/lib/postgresql/16/bin
      schema:
        enabled: true
    - resource_id: "epas17"
      dsn: $DSN_EPAS17
      pg_bin_path: /usr/edb/as17/bin
      schema:
        enabled: true
```

Understanding credentials and pgpass behavior

The OS user that runs `beacon-agent` is the same user that executes `pg_dump` . To avoid exposing database credentials in process arguments, the agent writes the password to a temporary `pgpass` file with restrictive permissions (`0600`) and passes a password-free connection string to `pg_dump` . The agent creates the file in the OS temporary directory and removes it automatically after each extraction completes.

> **Note**
>
> If the agent process is killed unexpectedly (for example, by *SIGKILL*), the agent may not clean up the temporary pgpass file. Only the OS user that runs the agent can read the file. The OS removes it on reboot, or on container restart if the agent runs in a container. To check for orphaned files, look for files matching *.pgpass-** in */tmp* .

Next step

Preparing your AWS RDS/Aurora PostgreSQL source database

Preparing your AWS RDS/Aurora PostgreSQL source database

Configure your AWS RDS or Aurora PostgreSQL source database and create the required users before starting your migration. What you need to configure depends on your goal:

- **Schema ingestion only** with the EDB Postgres AI agent: Create a machine user in Hybrid Manager (HM), set up a dedicated OS user for the agent on your auxiliary machine, and create a migration user with schema extraction permissions. No WAL-level database changes are required.
- **Data migration** with the Data Migration Service (DMS) agent: Enable logical replication via an AWS DB cluster parameter group and grant the required replication privileges.

If you plan to run both schema ingestion and data migration, complete all sections. If you only plan to do schema or data migration, each section explains whether it's applicable for your use case.

> **Note**
>
> Because AWS RDS and Aurora PostgreSQL are managed services, you can't modify database configuration directly with *psql* or by editing *postgresql.conf* . WAL-level settings must be applied through AWS DB cluster parameter groups.

Creating the migration user

> **Tip**
>
> Required for both schema ingestion with the EDB Postgres AI agent and data migration with the DMS agent.

Create a single migration user that serves both the EDB Postgres AI agent and the DMS agent. Using one user simplifies credential management across agents.

Using a database admin account, connect to your RDS or Aurora PostgreSQL source database and create the migration user:

```
CREATE USER <migration_user> WITH LOGIN PASSWORD <migration_user_password>;
```

Granting schema ingestion permissions

> **Tip**
>
> Required for schema ingestion with the EDB Postgres AI agent.

1. Grant the EDB Postgres AI agent connect access to the source database:

```
GRANT CONNECT ON DATABASE <source_database> TO <migration_user>;
```

Where `<source_database>` is the name of the database you want to assess. If you plan to connect the agent to additional databases on the same server, grant `CONNECT` on each one.

1. Grant the roles needed for a complete schema extraction:

```
GRANT pg_monitor TO <migration_user>;
GRANT pg_read_all_data TO <migration_user>;
GRANT pg_read_all_settings TO <migration_user>;
```

Together, these three roles enable a full schema extraction, including ownership and privilege information. With only `pg_read_all_data` granted, the agent performs a partial extraction limited to schema object definitions, without ownership and privilege information.

`pg_read_all_data` isn't available in Postgres 10 to 13:

```
GRANT pg_monitor TO <migration_user>;
GRANT pg_read_all_settings TO <migration_user>;
```

Granting data migration permissions

> **Tip**
>
> Required for data migration with the DMS agent.

1. Grant the `rds_replication` role, which provides replication privileges on RDS and Aurora:

```
GRANT rds_replication TO <migration_user>;
```

1. Grant the migration user usage on each schema containing tables to migrate:

```
GRANT USAGE ON SCHEMA <db_schema> TO <migration_user>;
```

1. Grant `SELECT` on source tables. For an entire schema:

```
GRANT SELECT ON ALL TABLES IN SCHEMA <db_schema> TO <migration_user>;
```

For individual tables:

```
GRANT SELECT ON <table_name> TO <migration_user>;
```

1. Grant `CREATE` on the source database so the DMS agent can create publications:

```
GRANT CREATE ON DATABASE <pg_db_name> TO <migration_user>;
```

Preparing for schema ingestion (EDB Postgres AI agent)

> **Tip**
>
> Required for schema ingestion with the EDB Postgres AI agent.

Creating a dedicated OS user for the agent

Create a dedicated OS user on your auxiliary machine to run `beacon-agent` . This strategy avoids granting the agent permissions it doesn't need. This is a Linux-specific example, the exact command might vary by distribution:

```
sudo useradd <agent_os_user>
```

Preparing for extension migration (EDB Postgres AI agent)

> **Tip**
>
> Required for schema migration with the EDB Postgres AI agent when your source database has extensions installed.

As part of schema migration, the EDB Postgres AI agent automatically extracts all extensions installed on the source database and applies them to the destination before any other schema DDL runs. If any extension fails to install, the entire schema migration aborts.

Before running schema migration, check which extensions are installed on the source and confirm they're available on the destination at the same version. If an extension isn't available at the matching version, schema migration fails. When migrating across different major Postgres versions, the extension versions available on the destination may not match those on the source, which can cause migration failures.

> **Note**
>
> On managed Postgres services (such as Amazon RDS or Aurora), extensions that require superuser to install fail silently at apply time. Review your source extensions for superuser requirements before migration.

```
SELECT name, installed_version
FROM pg_available_extensions
WHERE installed_version IS NOT NULL;
```

Run the same query on the destination to compare versions.

After migration completes, if your source uses foreign data wrapper (FDW) extensions, the agent installs the FDW extension itself but doesn't migrate FDW objects. Recreate foreign servers, user mappings, and foreign tables on the destination manually.

For a full list of extension migration limitations, see *Extension migration limitations* .

Preparing for data migration (DMS agent)

> **Tip**
>
> Required for data migration with the DMS agent.

Enabling logical replication

The DMS agent requires logical replication to be enabled on your source instance. Because RDS and Aurora are managed services, you enable this feature through an AWS DB cluster parameter group rather than by editing `postgresql.conf` directly.

1. Create a DB cluster parameter group using the AWS Management Console, AWS CLI, or RDS API. Ensure the parameter group family matches the Postgres version your instance is running.
2. Associate the parameter group with your source RDS or Aurora instance .
3. Modify the parameters in the parameter group:

- Set `rds.logical_replication` to `1` . This automatically sets `wal_level` to `logical` .
- Set `max_wal_size` to at least 8 GB for adequate WAL LSN lifetime.

> **Note**
>
> Aurora PostgreSQL doesn't allow modifying *max_wal_size* — it's managed by AWS. This parameter is only configurable on RDS PostgreSQL instances. !!!

Setting `max_wal_size` too small can allow Postgres to drop WAL entries before the DMS agent can stream them, interfering with or slowing down the migration.

1. Reboot your RDS instance to apply the changes.
2. Verify the settings took effect by connecting to the database and running:

```
SHOW wal_level;
SHOW max_wal_size;
```

`wal_level` should return `logical` .

Creating the replication group role

The migration user must own the source tables to auto-create Postgres publications. Because those tables are already owned by another role, create a replication group role that both the current table owner and the migration user belong to:

```
CREATE ROLE <replication_group>;
GRANT <replication_group> TO <migration_user>;
GRANT <replication_group> TO <original_owner>;
ALTER TABLE <table_name> OWNER TO <replication_group>;
```

Where:

- `<migration_user>` is the Postgres user you created for migration.
- `<original_owner>` is the original owner of the tables.
- `<replication_group>` is the shared role that owns the source tables for publication auto-creation.
- `<table_name>` is the name of a table to migrate (case-sensitive).

You must run `ALTER TABLE ... OWNER TO <replication_group>` for every table you plan to migrate, across all schemas. To transfer ownership of all tables in the database at once:

```
DO $$
DECLARE
  s text;
  t text;
BEGIN
  FOR s, t IN
    SELECT <schemaname>, <tablename> FROM pg_tables
    WHERE <schemaname> NOT IN (pg_catalog, information_schema)
  LOOP
    EXECUTE ALTER TABLE  || quote_ident(s) || . || quote_ident(t) ||  OWNER TO
↪<replication_group>;
  END LOOP;
END;
$$;
```

Validating your configuration

The DMS agent includes a validation script that checks whether your source database meets all requirements for CDC migration.

1. Navigate to the DMS agent folder:

```
cd /opt/cdcagent/reader/
```

1. Create an array of all tables to migrate in `<schema_name>.<table_name>` format:

```
arr=(schema1.table1 schema1.table2 schema2.table1)
```

1. Run the validation script:

```
PG_USERNAME=postgres PG_PASSWORD=<password> PG_HOST=<rds-endpoint> PG_PORT=5432 DB_
↪NAME=<db_name> DBZ_USERNAME=<migration_user> DBZ_PASSWORD=<migration_user_password>␣
↪./postgresConfigValidation.sh "${arr[@]}"
```

A passing result looks like this:

```
 *** [Transporter] - Get PostgreSQL Major Version
PostgreSQL major version: 17
 *** [Transporter] - Validate WAL Level
wal_level:  logical
[Pass] wal_level is logical.
 *** [Transporter] - Validate max WAL senders
max_wal_senders: 10
[Pass] max_wal_senders is at least 1.
 *** [Transporter] - Validate max replication slots
max_replication_slots: 10
[Pass] max_replication_slots is at least 1.
 *** [Transporter] - Validate max WAL size
max_wal_size: 8 GB
[Pass] max_wal_size is set to the recommended value of 8GB or higher.
 *** [Transporter] - Validate checkpoints
checkpoints_timed: 100
checkpoints_req: 5
[Pass] Timed checkpoints are more frequent than requested checkpoints.
 *** [Transporter] - Check <migration_user> user role
[Pass] User <migration_user> is present
[Pass] User <migration_user> has replication permission
 *** [Transporter] - Check SELECT privilege on the tables to be migrated
[Pass] User <migration_user> has select privilege on all tables to be migrated.
 *** [Transporter] - Check CREATE privilege on database
[Pass] Role <migration_user> has CREATE privilege on database <db_name>.
```

Note

Address any *[Failed]* or *[Suggestion]* statuses before proceeding. *[Failed]* checks are blocking issues you must resolve before the DMS agent can execute the data migration. *[Suggestion]* checks allow migration but can negatively affect performance if left unaddressed.

Next step

Configuring the EDB Postgres AI agent to register your AWS RDS/Aurora PostgreSQL database

Configuring the EDB Postgres AI agent to register your AWS RDS/Aurora PostgreSQL database

Configure the EDB Postgres AI agent (packaged as `beacon-agent`) to establish a secure connection with HM and retrieve schema and migration assessment data from your Postgres source.

Prerequisites

- **Machine user and access key**: Created in *Creating a machine user* . Store the access key in an environment variable:

```
export BEACON_AGENT_ACCESS_KEY="baak_your_key_here"
```

- **pg_dump**: The EDB Postgres AI agent uses `pg_dump` to extract DDL from your Postgres source. Ensure `pg_dump` is installed and configured on your auxiliary machine. See *Configuring your auxiliary machine or bastion host* for installation instructions, compatibility rules, and credential setup.

> **Note**
>
> The agent substitutes environment variables referenced in *beacon_agent.yaml* at runtime, so setting *$BEACON_AGENT_ACCESS_KEY* in your environment before running the agent is sufficient. Environment variables always take precedence over static values in the YAML.

Recommendations for setting environment variables

We recommend handling sensitive data, such as `$BEACON_AGENT_ACCESS_KEY` or `$DSN` , as environment variables managed through a secrets manager. This strategy prevents sensitive credentials from being stored in plaintext configuration files. If you use environment variables, make sure they're available to the EDB Postgres AI agent while it's running.

Consult your organization's IT department on the safest approach for production environments.

For short-lived migrations or testing and demo purposes, you can use the `export` command in an active terminal session.

Obtaining HM connection parameters

The agent requires specific values from your HM installation. Contact your administrator for:

- **Beacon server hostname**: The internal endpoint for the `beacon_server` .

When configuring the HM installation, administrators must specify an internal endpoint for the `beacon_server` service. Ask the administrator or installer of your HM instance to provide the hostname (URL) they set for the `beacon_server` service in the Helm chart configuration file used for installation. They'll look up the value they set for `parameters.upm-beacon.server_host` (or `BEACON_SERVICE_DOMAIN_NAME`) in the `values.yaml` file. You'll need this value later to configure the EDB Postgres AI agent.

- **Root certificate path (if required)**: If your HM instance uses a self-signed or internal CA, obtain the `ca_public.crt` file. Store it on the agent machine (for example, `/etc/beacon/ca_public.crt`).

The EDB Postgres AI agent requires a trusted connection to the `beacon_server` service in your HM instance. If the server running the EDB Postgres AI agent doesn't inherently trust HM's TLS certificate (managed by your organization's security infrastructure), ask the administrator or installer of your HM instance to provide the file with the certificate. You'll need to provide the directory path to this certificate later to configure the EDB Postgres AI agent.

See *Administrative tasks* for more information about the `beacon_server` and root certificate.

Verifying connectivity

Once you have the HM connection parameters, verify that your auxiliary machine can reach both the Postgres source and HM. The following ports must be open:

Destination	Port	Used by
Postgres host	5432 (default for PostgreSQL, AWS RDS, and Aurora); 5444 (default for EDB Postgres Advanced Server)	Both agents (database connection)
HM *beacon_server*	9443	EDB Postgres AI agent
HM DMS service (*RW_SERVICE_HOST*)	443 (HTTPS)	DMS agent

> **Note**
>
> Ports 5432 and 5444 are defaults. If your Postgres database is configured to listen on a different port, use that port instead and ensure it's open on your network.

Verify connectivity to each host and port with `nc`:

```
#  Postgres source

nc -zv <postgres-host-name-or-ip> 5432

#  HM beacon server

nc -zv <beacon_server_hostname> 9443

#  HM DMS service

nc -zv <rw-service-hostname> 443
```

A successful connection returns `succeeded!`. If you see `Connection refused` or `Operation timed out`, the port isn't reachable and must be opened before proceeding.

Defining source database connection parameters

The data source name (DSN) is the connection string for your Postgres source.

Format:

```
DSN="postgresql://<migration_user>:<migration_user_password>@<host>:<port>/<database_
↪name>"
```

Set the DSN as an environment variable before running the agent or configuring the service:

```
export DSN="postgresql://migration_user:password@postgres-host:5432/mydb"
```

If your database has SSL disabled, append `?sslmode=disable`:

```
export DSN="postgresql://migration_user:password@postgres-host:5432/mydb?
↪sslmode=disable"
```

Use the RDS instance endpoint as the host. RDS enforces SSL by default, so append `?sslmode=require`:

```
export DSN="postgresql://migration_user:password@myinstance.abc123.us-east-1.rds.
↪amazonaws.com:5432/mydb?sslmode=require"
```

Use the Aurora cluster endpoint as the host. Aurora enforces SSL by default, so append `?sslmode=require`:

```
export DSN="postgresql://migration_user:password@my-cluster.cluster-abc123.us-east-1.
↪rds.amazonaws.com:5432/mydb?sslmode=require"
```

```
DSN="postgresql://migration_user:pa%24%24word@postgres-host:5432/mydb"
```

Preparing a configuration file

1. Create an EDB Postgres AI agent configuration directory in your home directory:

```
mkdir ${HOME}/.beacon
```

> **Note**
>
> The agent searches for *beacon_agent.yaml* in */etc/beacon/* or *${HOME}/.beacon/* . As a final fallback, it searches the directory from which it's executed. !!!

1. Inside this directory, create a file `beacon_agent.yaml` :

```
touch ${HOME}/.beacon/beacon_agent.yaml
```

1. Copy and paste the following template into the new file.

In this template, `$BEACON_AGENT_ACCESS_KEY` and `$DSN` are the environment variables you configured previously. Replace all < > placeholders in the template. See *EDB DMS agent parameter reference* for details.

```
---
agent:
  access_key: $BEACON_AGENT_ACCESS_KEY
  beacon_server: <beacon_server_hostname>:9443
  project_id: <your_project_id>
  providers:
    - "onprem"
  # In most HM deployments, set `root_ca_path` to the path of the ca_public.crt
  # file obtained from your HM administrator (for example, /etc/beacon/ca_public.crt).
  # Leave empty only if HMs certificate is already trusted by the systems CA bundle.
  root_ca_path: ""
  schema_providers:
    # Include "onprem-schema" to enable schema ingestion and migration assessment.
    # Remove this section if you dont want schema information to be collected.
    - "onprem-schema"
provider:
  onprem:
    databases:
      # `resource_id` controls how this database appears in HM Estate.
      - resource_id: "<database_resource_id>"
        dsn: $DSN
        # pg_bin_path: /usr/lib/postgresql/16/bin
        # Optional. Absolute path to the directory containing pg_dump and pg_dumpall.
        # Set when registering databases from different Postgres distributions/
↪versions.
        schema:
          enabled: true
          poll_interval: 15s
general:
  logging:
    # `info` is simplified, use debug during initial setup or
    # troubleshooting for more verbose output.
    level: info
```

1. Save the file.

Running the EDB Postgres AI agent

1. Before starting the agent, verify connectivity and configuration syntax:

```
beacon-agent validate
```

1. Run the agent:

```
beacon-agent
```

> **Tip**
>
> See Agent CLI for other agent modes and options. !!!

1. Monitor the ingestion process as the agent starts sending data to HM. After a few minutes, your Postgres database appears in the HM console under **Estate > Migrations**.

> **Important**
>
> The agent dynamically detects changes to tables and columns. However, changes to views, procedures, or functions require an agent restart to re-ingest metadata.

For production use, see *Running the agent as a service* .

Verifying the migration assessment

To confirm your Postgres database was recognized by the agent and is ready for migration, go to **Estate > Migrations** in the HM console.

If the database appears in the table, you have access to the following resources:

- A **migration assessment** on the *Migrations* of the **Estate** page, providing a summarized view of database schema count, core count, migration complexity, and level of effort.
- A *Viewing migration assessment details* accessible by selecting a specific database from the **Migrations** tab. This tab provides an expanded analysis of **Schemas**, a global **Assessment**, and a **Migration** hub with direct connections to other HM tools for performing migrations.
- **Automatic schema DDL extraction** that you can use for schema migrations with the built-in Migration Portal. Once ingestion is complete, the Migration Portal automatically creates a project per database, which you can use to prepare and migrate schemas to the destination database.

Troubleshooting

There are several reasons why the agent might fail to connect to HM or your Postgres database. Here are some tools you can use for troubleshooting.

`beacon-agent validate` command

The `beacon-agent validate` command is a helpful step to identify common connectivity and configuration issues.

```
beacon-agent validate
```

The validate command checks for the following:

- **Server connectivity**: Verifies the agent can connect and authenticate to the configured `beacon_server` (usually on port `9443`).
- **Database connectivity**: Attempts to connect to each configured database DSN. This is a full connection test, not just a syntax or URL-encoding check.
- **Configuration display**: Shows TLS settings, project ID, providers, and per-database settings to help identify misconfiguration.

`beacon-agent` logs

If the agent reports errors during ingestion, you can check the logs for more details. If you're running the agent in the foreground, logs print to the terminal. If you set up the agent as a systemd service, use `journalctl` to view logs.

Here are some common error messages you might encounter in the logs:

Error	Cause	Fix
tls: failed to verify certificate: x509: certificate signed by unknown authority	The agent can't verify HM's TLS certificate because the CA isn't trusted by the system.	Set *root_ca_path* in *beacon_agent.yaml* to the path of the *ca_public.crt* file obtained from your HM administrator. If you can't obtain the certificate, you can set *plaintext: true* in the agent config as a temporary workaround (not recommended for production).
rpc error: code = Unknown desc = no rows in result set	The beacon-agent version is incompatible with the server.	Upgrade to beacon-agent v1.384.0 or later.

Running the agent as a service

For continuous metric streaming and to ensure your Postgres database remains visible in **HM console > Estate**, configure the agent to run as a systemd service. Running as a service ensures it starts automatically on system boot and restarts after any errors.

Without a service, the agent streams metrics only while you run it manually. Your database will disappear from the HM estate 72 hours after the agent stops streaming.

> **Note**
>
> Ensure that any environment variables referenced in *beacon_agent.yaml* (for example, *$BEACON_AGENT_ACCESS_KEY* and *$DSN*) are accessible to the service at runtime. If you configured a *.pgpass* file for schema extraction credentials (as described in *Configuring the environment*), ensure that file is present in the home directory of the user specified in *User=* .

1. Create the service file. This example uses `vi` , but you can use any text editor:

```
sudo vi /usr/lib/systemd/system/beacon-agent.service
```

Default service file paths by operating system:

Operating system	Default path
RHEL	*/usr/lib/systemd/system/beacon-agent.service*
Rocky Linux	*/usr/lib/systemd/system/beacon-agent.service*
AlmaLinux	*/usr/lib/systemd/system/beacon-agent.service*
Ubuntu	*/etc/systemd/system/beacon-agent.service*
Debian	*/lib/systemd/system/beacon-agent.service*

1. Paste the following template into the file and replace all < > placeholders:

```
[Unit]
Description=EDB Postgres AI Agent
After=network.target

[Service]
Type=simple
User=<agent_os_user>
WorkingDirectory=</home/agent_os_user/>
ExecStart=</usr/local/bin/beacon-agent>
Restart=on-failure
RestartSec=60

[Install]
WantedBy=multi-user.target
```

Setting	Placeholder	Description
User=	*<agent_os_user>*	OS user that runs the agent. Use the dedicated OS user you created when preparing the source database.
WorkingDirectory=	*</home/agent_os_user/>*	Directory where the *.beacon* folder is stored. This is typically the home directory of *<agent_os_user>*.
ExecStart=	*</usr/local/bin/beacon-agent>*	Full path to the *beacon-agent* binary. The path depends on how and where the agent was installed.

1. Reload systemd, then enable and start the service:

```
sudo systemctl daemon-reload
sudo systemctl enable beacon-agent.service
sudo systemctl start beacon-agent.service
```

1. Confirm the service started successfully by checking the logs:

```
journalctl -u beacon-agent.service
```

Related topics

- For more granular control of schema ingestion (for example, to exclude or include schemas), see *Controlling schema ingestion* .
- For the full set of configuration parameters, see *EDB DMS agent parameter reference* .
- If your administrator needs guidance on obtaining the root certificate or `beacon_server` value, see *Administrative tasks* .

Next step

Preparing your HM-managed destination database

Preparing your HM-managed destination database

Prepare an HM-managed database cluster to serve as the destination for your migration.

If you don't have a destination database cluster on Hybrid Manager (HM) yet, create one:

1. In the HM console, select the "Can't select a project" — the portal domain defaults to the API endpoint where you want to create the database cluster.
2. In your project, select **Create New** and then **Database Cluster** to *Create a database cluster template* .
3. In the **Clusters** page, select your cluster, and use the **Quick Connect** option to access your database from your terminal.
4. Create an empty database to use as a destination for the migration. Alternatively, you can use the default database `edb_admin` .

If you already have an HM-managed database cluster, you can use it as the destination for your migration.

Before proceeding, ensure the tables you migrate and the load generated during migration don't interfere with existing workloads on the cluster.

Next step

Migrating Postgres schemas

Migrating Postgres schemas

Before starting a data migration, choose how to handle schema migration. HM supports two approaches:

- **Simplified (recommended)**: HM manages schema migration and constraint handling automatically as part of the data migration setup.
- **Manual**: Export and import the schema yourself using Migration Toolkit or `pg_dump` , and handle constraints based on the key structure of each source table.

No manual schema preparation is required. HM handles schema migration and constraint ordering automatically when you select the **Schema and data** scope later. Before proceeding, confirm:

- The EDB Postgres AI agent has ingested your source schema and a Migration Portal project exists for the database. To verify, go to **Estate > Migrations** in the HM console, select the database in the **Database Name** column, and confirm a project appears in the **MP Project** column.
- When you configure the DMS reader agent in a later step, you'll set `DBCONFIG_DATABASES_<N>__RESOURCEID` to the same `resource_id` value as in `beacon_agent.yaml` . The **Schema and data** and **Schema only** scopes will fail without it.

Continue to the next step.

Before you use EDB DMS for data migration, export and import your schema to the destination database. DMS migrates data only — the destination database must have schemas in place before migration begins.

Handling constraints

How to handle constraints depends on the constraint type and the key structure of each source table.

Constraint type	Action	Notes
Primary key	Exclude now — DMS applies automatically	The DMS writer applies the primary key when the migration enters the streaming phase, using it to deduplicate snapshot data.
Unique key (source table has a primary key)	Exclude now, apply after streaming starts	Primary key-based deduplication ensures no duplicate rows remain when you apply unique constraints after streaming starts.
Unique key (source table has no primary key)	Not currently supported	Use the simplified path instead — select the **Schema and data** scope when creating the migration to let HM handle schema migration automatically.
NOT NULL	Include now	*NOT NULL* validation adds no meaningful overhead to the data load, unlike foreign key and check constraint validation.
Foreign key, check, and exclusion	Exclude now, reapply after migration	These cause unnecessary CPU and memory overhead while data is actively being migrated.

Note

For tables with no primary key or unique key, no key constraints are needed before or after migration. DMS treats these tables as keyless and provides at-least-once delivery. Deduplicate rows as part of your post-migration verification.

Exporting your schema

We recommend using EDB Migration Toolkit to export your schema. MTK's offline migration capability generates SQL scripts you can apply on the destination without direct connectivity. Other valid export methods include `pg_dump` /`pg_restore` , pgAdmin, or `psql` .

When using Migration Toolkit:

- Use the `-offlineMigration` option to generate SQL scripts.
- Use the `-schemaOnly <schema_scope>` option to export schema only.
- Exclude (comment out) primary key, unique, foreign key, check, and exclusion constraints from the generated script before importing.
- If you plan to map objects to different names in the destination, edit them directly in the DDL script before importing. Update all dependent objects — views, constraints, and other references — that reference renamed objects. See *Mapping schema, tables, and columns* for details.

Importing your schema

After preparing the DDL and excluding the deferred constraints, connect to the destination database and apply the SQL file by executing the generated offline migration script .

Note

Apply the excluded unique, foreign key, check, and exclusion constraints after migration completes. The DMS writer applies primary key constraints automatically — don't reapply them manually. Keep track of excluded constraints so you can reapply them when the time comes.

Next step

Configuring and running the EDB DMS agent

Configuring and running the EDB DMS agent

The EDB DMS agent connects to your AWS RDS or Aurora PostgreSQL source and works with the Data Migration Service (DMS) to perform data migrations. Before running it, create migration credentials in HM and configure the agent with your source database details.

Prerequisites

- Ensure the machine where you're running the DMS agent has Java/OpenJDK 17 or later installed.
- *Install the EDB DMS agent* .

Creating migration credentials

Migration credentials authenticate the DMS agent to Hybrid Manager (HM). You need one set of credentials per HM project.

1. In the HM console, select the project you're using for your migration.
2. Select **Migrate > Credentials**.
3. Select **Create Migration Credential**, enter a name, and select **Create Migration Credential**. A `credentials.zip` file is downloaded to your machine.

Note

Store the credentials safely. If you lose them, you'll need to revoke this set and create new ones. !!!

1. Unzip the downloaded file and copy the credentials folder to the machine where you installed the DMS agent — for example, to `$HOME/credentials/` . If you use a different path, update `CREDENTIAL_DIRECTORY_PATH` in the script to match.

The folder contains:

- `int.crt`
- `int.truststore.p12`
- `client.keystore.p12`
- `client-key.pem`
- `client-cert.pem`

Configuring the DMS agent

1. Open `/opt/cdcagent/run-cdcagent.sh` with write permissions.
2. Set the required variables. Optional parameters are commented out by default — uncomment and configure any that apply to your environment.

See the *EDB DMS agent parameter reference* for details.

```
#!/bin/bash -e
# run-cdcagent.sh
#
# This script provides a convenient place to specify
# environment variables used to configure the
# EDB Data Migration Service Agent.
#

########################################
# DMS Agent General Configuration      #
########################################

# This ID is used to identify DMS Agent
# and is specified by the user.
export DBCONFIG_ID=<source_name_of_choice>

# Determine the run mode of the DMS Agent
# The run mode can be set to either reader or writer
# reader mode is used to read changes from a source database
# writer mode is used to write changes to a target database
export RUN_MODE=reader

# This is the DMS backend service used by the Agent
# Obtain this value from the HM console Sources or Destinations migration pages
export RW_SERVICE_HOST=<rw_service_host>

########################################
# DMS Agent TLS Configuration          #
########################################

# You need to create migration credentials in EDB Postgres AI platform
# and set the path of the credentials directory
export CREDENTIAL_DIRECTORY_PATH=$HOME/credentials

########################################
# DMS Agent Source DB Configuration    #
########################################

# A sample configuration to create a single Postgres database connection:
export DBCONFIG_DATABASES_0__TYPE=POSTGRES
export DBCONFIG_DATABASES_0__HOSTNAME=<RDS_endpoint>
export DBCONFIG_DATABASES_0__PORT=<5432>
# The CATALOG is the database name
export DBCONFIG_DATABASES_0__CATALOG=<source>
export DBCONFIG_DATABASES_0__USERNAME=<postgres>
# The password env can be set without specifying it here
```

(continues on next page)

(continued from previous page)

```
# but the env structure looks like this
export DBCONFIG_DATABASES_0__PASSWORD=<password>
# Required for Schema and data or Schema only scope.
# Must match resource_id in beacon_agent.yaml.
#export DBCONFIG_DATABASES_0__RESOURCEID=<resource_id>

# You can increase the index to configure more than
# one database for the DMS Agent
#export DBCONFIG_DATABASES_1__TYPE=POSTGRES
#export DBCONFIG_DATABASES_1__HOSTNAME=<RDS_endpoint>
#export DBCONFIG_DATABASES_1__PORT=<5432>
#export DBCONFIG_DATABASES_1__CATALOG=<source>
#export DBCONFIG_DATABASES_1__USERNAME=<postgres>
#export DBCONFIG_DATABASES_1__PASSWORD=<password>
#export DBCONFIG_DATABASES_1__RESOURCEID=<resource_id>

########################################
# Optional Parameters Below            #
########################################

# FIPS mode can be enabled by setting the FIPS_MODE env variable to true
#export FIPS_MODE=true

# Configure logging
# Loglevel for the entire Quarkus application and underlying components
# WARNING: Enabling this loglevel is extremely verbose
#export QUARKUS_LOG_LEVEL=DEBUG
# Loglevel for the EDB package
# Preferred mode for debugging issues related to the logic of the agent
#export QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL=DEBUG

cd "$(dirname "$0")"

echo "Starting in mode: $RUN_MODE"

case "$RUN_MODE" in
  reader)
    java -jar ./reader/quarkus-run.jar
    ;;
  writer)
    java -jar ./writer/quarkus-run.jar
    ;;
  *)
    echo "Unknown RUN_MODE: $RUN_MODE"
    exit 1
    ;;
esac
```

Note

Set *DBCONFIG_DATABASES_0__HOSTNAME* to the RDS endpoint found under **Connectivity & security** in the

AWS Console. A redacted example is *foo_database.cb7asw1xxxxx.eu-west-0.rds.amazonaws.com* . !!!

Running the DMS agent

Navigate to the DMS agent directory and run the configuration script:

```
cd /opt/cdcagent
./run-cdcagent.sh
```

The script exports all required environment variables and starts the agent process. Once running, the agent registers your AWS RDS or Aurora PostgreSQL database as a source in HM. Continue to the next section to verify it appears correctly.

Verifying the source in HM

Once the agent is running, verify that your database appears as a source in HM:

1. In the HM console, select your project.
2. Select **Migrate > Sources**.

Your database is listed with its name, type, agent ID, and status. For a description of all fields on this page, see *Understanding deployment resources* .

Next step

Migrating data from an AWS RDS or Aurora PostgreSQL database to an HM-managed database

Migrating data from an AWS RDS or Aurora PostgreSQL database to an HM-managed database

With the EDB DMS agent running and your schema either imported or ready for HM to import, you're ready to create a migration in the HM console. This page covers the full migration lifecycle: creating the migration, monitoring its progress, finalizing it, and applying post-migration constraints.

The steps below vary depending on the approach you chose on the previous page. Select the tab that matches your choice.

Creating a migration

A migration establishes a sync between your Postgres source and a destination database cluster managed by HM.

1. In the HM console, select the project from which you want to manage the migration.
2. In the project, select **Migrate > Migrations**.
3. In the **Migrations** page, select **Create New Migration**.
4. In the **Create OLTP Migration** page, assign a name to the migration.
5. Select the source of the migration. The ID for the DMS agent is listed in the **Source** menu.
6. Under **Scope**, select the option that matches your approach:

Select **Schema and data**. When you select this option, HM:

1. Sources the schema from the Migration Portal project that was created automatically when you registered your database with the EDB Postgres AI agent.

2. Creates tables on the destination without constraints, so the initial snapshot copy isn't slowed by constraint validation.
3. After the snapshot completes, applies primary key and unique constraints (for tables that have them) so DMS can deliver streaming changes exactly-once. Tables without a PK or UK fall back to at-least-once delivery and require deduplication during verification.
4. Defers the remaining schema (indexes, foreign keys, check constraints, `NOT NULL`, and similar) until cutover. You can optionally apply most of this work concurrently with streaming via **Finalize Schema Concurrently** to shrink the cutover window — see *Finalizing a streaming migration* .
5. After you mark the migration as completed, validates deferred foreign keys, syncs sequences, and applies any remaining lightweight tasks in a final **Schema Finalization** stage.

> **Note**
>
> Use **Schema only** if you want to migrate the database structure without migrating any table data. !!!

Select **Data only**. The schema was already imported manually in the previous step.

1. Select the snapshot type: ***Snapshot*** or ***Snapshot + Streaming*** .
2. Under **Destination Type**, select an HM-managed destination database cluster and database for the migration. Then, select **Next**.
3. On the **Review your Tables** tab, select the tables and columns to include in the migration. Expand a table to view its columns and optionally *map source columns to different destination column names* . Column mapping is only available for the **Data only** scope.
4. Select **Create Migration**.

The HM console displays the *new migration* . If you chose **Snapshot + Streaming**, the DMS agent streams data changes continuously while the migration shows a **Running** state.

> **Note**
>
> The DMS agent streams data changes only — not DDL changes (schema modifications).

Monitoring progress

Migration list view

The **Migrations** page shows a summary of all migrations in the project. The **Migration progress** column displays three metrics per migration:

- **Total number of tables** — total tables included in this migration
- **Snapshot tables completed** — tables that have finished the snapshot stage
- **Migrated rows** — cumulative rows migrated across all tables

Migration details view

Select a migration from the list to open the details page, which has three tabs:

- **Overview** — shows the current migration state and general information about the source and destination databases. For **Snapshot + Streaming** migrations, the migration is caught up when the status is **Streaming**. For **Snapshot** migrations, it's finished when the status is **Completed**.

- **Tables** — shows all tables selected for migration. For each table, you can see source and destination table and schema names, snapshot status, mapped columns, migrated rows, and migration rate. Select the + icon to expand a table and view column-level mappings.
- **Error history** — lists all errors logged for this migration. Select the **i** icon to open the full error log for a specific error. For stuck or failed migrations, active errors here indicate where to start troubleshooting.

DMS agent logs

The HM console doesn't surface all DMS agent errors. If a migration is stuck or no tables are progressing, check the terminal where you ran `run-cdcagent.sh` for `ERROR` log entries — these entries often contain the root cause. Common issues include version incompatibility between the DMS agent and HM and connectivity or credential failures.

Finalizing a streaming migration

> **Note**
>
> This section applies only to **Snapshot + Streaming** migrations. For **Snapshot** migrations, DMS marks the migration as completed automatically.

Monitoring catch-up before cutover

Before stopping writes on the source, give the destination time to catch up. Use the **Migration details** page to track row counts and replication lag. Begin the cutover only after the destination is consistently close to the source.

Finalizing schema concurrently (recommended)

If you used the **Schema and data** scope, HM can apply most schema objects while the stream is still running: primary keys, unique constraints, indexes (built with `CREATE INDEX CONCURRENTLY` , so live writes aren't blocked), foreign keys (added as `NOT VALID`), check constraints, `NOT NULL` constraints, and similar. Foreign key validation, sequence synchronization, and other lightweight tasks run later in the post-cutover **Schema Finalization** stage.

Running this work concurrently shrinks the read-only cutover window. The larger your tables and the more indexes they have, the greater the time saved.

Once every table has completed its snapshot and the migration enters the streaming phase, the **Migration details** page shows a **Schema Finalization Available** alert with a **Finalize Schema Concurrently** button. The button stays available throughout streaming, so you can pick the moment based on how close the destination is to catching up.

1. Select **Finalize Schema Concurrently** on the **Migration details** page.
2. HM applies the deferred objects while streaming continues. The migration remains in the **Running** state, with a progress sub-step labeled **Finalizing Schema Concurrently**.

When the sub-step completes, proceed to the cutover.

> **Note**
>
> **Finalize Schema Concurrently** is available only for migrations created with the **Schema and data** scope with streaming enabled.

If you skip this step, the same objects are applied serially after you mark the migration as completed. See *Schema finalization stage* . The cutover window is larger in that case because indexes are built non-concurrently and constraints are validated under exclusive locks.

Not applicable to the manual path. You applied the schema before migration and will reapply deferred constraints manually after cutover. See *Applying deferred constraints and verifying data* .

Performing the cutover

When you're ready to switch traffic from the source to the destination:

1. Stop all writes on the source database.
2. Verify in the HM console that the destination has applied the last source change.
3. On the **Migration details** page, select **Mark as Completed**.

This stops the streaming process.

Schema finalization stage

For migrations created with the **Schema and data** scope, the migration enters a **Schema Finalization** stage automatically after **Mark as Completed**. What runs in this stage depends on whether you ran concurrent finalization first:

- **If you ran concurrent finalization**: HM runs the lightweight remaining work — foreign key validation, sequence sync, and any other deferred metadata tasks. This stage is short.
- **If you skipped concurrent finalization**: HM applies the full deferred set now (primary keys, unique constraints, indexes, foreign keys, check constraints, `NOT NULL` , and similar) followed by the lightweight tasks above. This stage takes substantially longer and runs while the source is read-only.

The migration is fully complete only after the **Schema Finalization** stage finishes. Progress is shown on the **Migration details** page.

No automatic schema finalization runs for the **Data only** scope. Continue to *Applying deferred constraints and verifying data* to reapply the foreign key, unique, check, and exclusion constraints you deferred before migration.

Applying deferred constraints and verifying data

Reapplying deferred constraints

HM applied all deferred constraints and synced sequences automatically during the **Schema Finalization** stage. No manual action is needed. See *Schema finalization stage* .

At the beginning of your migration, you prepared and imported the schema excluding foreign key, unique, check, and exclusion constraints. Connect to the destination database and reapply those constraints now.

Primary key constraints were applied automatically by the DMS writer when the migration entered the streaming phase. `NOT NULL` constraints were applied before migration and don't require any action now.

Deduplicating rows when needed

For tables that had no primary key or unique constraints, the EDB DMS agent provides at-least-once delivery, which can result in duplicate rows. Perform deduplication as part of your verification step.

Verifying migrated data

Compare the source and destination databases to confirm all data was migrated successfully. You can use LiveCompare for this.

Related topics

- LiveCompare — row-level data comparison tool for verifying migration completeness
- *Terminology* — definitions for snapshot, streaming, and other migration concepts

Migrate your self-managed Postgres database to a different self-managed Postgres database

These are the tasks involved in an end-to-end migration from a self-managed Postgres database to a different self-managed Postgres database using the Data Migration Service (DMS) of Hybrid Manager (HM). Because the destination isn't HM-managed, this migration requires two DMS agent instances: a reader to stream changes from the source, and a writer to apply them to the destination.

1. ** *Configuring your auxiliary machine or bastion host* ** — Configure the machine that hosts both migration agents. This machine acts as the bridge between your Postgres source and HM.
2. ** *Preparing your AWS RDS/Aurora PostgreSQL source database* ** — Create the required users and configure your Postgres database for schema ingestion with the EDB Postgres AI agent and data migration with the DMS reader agent.
3. ** *Configuring the EDB Postgres AI agent to register your AWS RDS/Aurora PostgreSQL database* ** — Connect the agent to your source database and to HM to enable schema ingestion and migration assessment.
4. ** *Preparing your HM-managed destination database* ** — Create a self-managed Postgres database and configure it with SSL and a CDC migration role to receive the migrated data.
5. ** *Migrating Postgres schemas* ** — Choose between the simplified path, where HM handles schema migration and constraint ordering automatically, or the manual path, where you export, prepare, and import the schema yourself.
6. ** *Configuring and running the EDB DMS agent* ** — Create migration credentials, configure the reader agent for your source and the writer agent for your destination, and verify both appear in HM.
7. ** *Migrating data from an AWS RDS or Aurora PostgreSQL database to an HM-managed database* ** — Create and run the data migration in the HM console, monitor its progress, finalize streaming, and apply post-migration constraints.

Configuring your auxiliary machine or bastion host

The EDB Postgres AI agent (`beacon-agent`) and the Data Migration Service (DMS) agent (`cdcagent`) can run on the Postgres database server itself or on a separate auxiliary machine or bastion host. In most production environments, running the agents on a dedicated auxiliary machine is preferred — it avoids installing additional software on the database server, and a single agent instance can connect to databases on multiple servers.

Required software

Ensure you install the following software on your auxiliary machine before running the agents:

- **Java 17 or higher** — required by the DMS agent (`cdcagent`). Verify with `java -version` .
- **EDB Postgres AI agent** (`beacon-agent`) — required for database registration, schema assessment, and schema migration. Check the *beacon-agent version compatibility matrix* . Then install a compatible version of the EDB Postgres AI agent .
- **DMS agent** (`cdcagent`) — handles data migration and change data capture (CDC). Before installing, check the *DMS agent version compatibility* . Then *install a compatible version of the DMS agent* .

Configuring the environment

When you enable schema ingestion, `beacon-agent` uses `pg_dump` to extract DDL from your Postgres source. Postgres client tools packages include the `pg_dump` utility — it might already be available on your machine. Verify first:

```
pg_dump --version
```

If the command isn't found, install the client tools that match your source database distribution.

Once installed, ensure your `pg_dump` binary meets two compatibility rules:

- **Version rule** — The `pg_dump` major version must be greater than or equal to the major version of your source database. For example, if your source runs Postgres 16, you need `pg_dump` 16 or later. The agent validates this at startup and reports an error if the versions are incompatible.
- **Distribution rule** — The `pg_dump` binary must come from the same Postgres distribution as the source database. Each distribution's `pg_dump` accesses distribution-specific catalog tables. Using a mismatched binary (for example, community PostgreSQL `pg_dump` against an EDB Postgres Advanced Server source) will fail because the expected catalog objects won't exist.

Managing multiple Postgres versions on a host

If you have multiple Postgres versions on the same machine, ensure the correct `pg_dump` appears first in your `PATH` :

```
which pg_dump
pg_dump --version
```

On Debian and Ubuntu systems, use `update-alternatives` to manage the active version:

```
sudo update-alternatives --config pg_dump
```

Registering databases from multiple distributions or versions

A single `beacon-agent` instance can register databases from different Postgres distributions (for example, community PostgreSQL and EDB Postgres Advanced Server) or different major versions at the same time. Because the version and distribution rules apply per database, each database connection may require a different `pg_dump` binary.

Install all the required `pg_dump` binaries on the auxiliary machine. Later you can use the `pg_bin_path` setting in your `beacon_agent.yaml` configuration file to point each database entry to the correct binary directory. When set, it overrides the system `PATH` lookup for that specific database. When omitted, the agent falls back to the first `pg_dump` found in `PATH` . For example:

```
provider:
  onprem:
    databases:
      - resource_id: "pg16-community"
        dsn: $DSN_PG16
        pg_bin_path: /usr/lib/postgresql/16/bin
        schema:
          enabled: true
      - resource_id: "epas17"
        dsn: $DSN_EPAS17
        pg_bin_path: /usr/edb/as17/bin
        schema:
          enabled: true
```

Understanding credentials and pgpass behavior

The OS user that runs `beacon-agent` is the same user that executes `pg_dump`. To avoid exposing database credentials in process arguments, the agent writes the password to a temporary `pgpass` file with restrictive permissions (`0600`) and passes a password-free connection string to `pg_dump`. The agent creates the file in the OS temporary directory and removes it automatically after each extraction completes.

> **Note**
>
> If the agent process is killed unexpectedly (for example, by *SIGKILL*), the agent may not clean up the temporary pgpass file. Only the OS user that runs the agent can read the file. The OS removes it on reboot, or on container restart if the agent runs in a container. To check for orphaned files, look for files matching *.pgpass-** in */tmp*.

Next step

Preparing your AWS RDS/Aurora PostgreSQL source database

Preparing your Postgres source database

Configure your Postgres source database and create the required users before starting your migration. What you need to configure depends on your goal:

- **Schema ingestion only** with the EDB Postgres AI agent: Create a machine user in Hybrid Manager (HM), set up a dedicated OS user for the agent on your auxiliary machine, and create a migration user with schema extraction permissions. No WAL-level database changes are required.
- **Data migration** with the Data Migration Service (DMS) agent: Configure WAL settings and grant the required replication privileges.

If you plan to run both schema ingestion and data migration, complete all sections. If you only plan to do schema or data migration, each section explains whether it's applicable for your use case.

Creating the migration user

> **Tip**
>
> Required for both schema ingestion with the EDB Postgres AI agent and data migration with the DMS agent.

Create a single migration user that serves both the EDB Postgres AI agent and the DMS agent. Using one user simplifies credential management across agents.

Using a database admin account, connect to your source Postgres database and create the migration user:

```
CREATE USER <migration_user> WITH REPLICATION LOGIN PASSWORD <migration_user_password>
↪;
```

`REPLICATION` is required for the DMS agent to create replication slots and read from the WAL. If you're performing schema ingestion only, you can omit it — but including it now means you won't need to alter the user later if you add data migration.

Granting schema ingestion permissions

> **Tip**
>
> Required for schema ingestion with the EDB Postgres AI agent.

1. Grant the EDB Postgres AI agent connect access to the source database:

```
GRANT CONNECT ON DATABASE <source_database> TO <migration_user>;
```

Where `<source_database>` is the name of the database you want to assess. If you plan to connect the agent to additional databases on the same server, grant `CONNECT` on each one.

1. Grant the roles needed for a complete schema extraction:

```
GRANT pg_monitor TO <migration_user>;
GRANT pg_read_all_data TO <migration_user>;
GRANT pg_read_all_settings TO <migration_user>;
```

Together, these three roles enable a full schema extraction, including ownership and privilege information. With only `pg_read_all_data` granted, the agent performs a partial extraction limited to schema object definitions, without ownership and privilege information.

`pg_read_all_data` isn't available in Postgres 10 to 13:

```
GRANT pg_monitor TO <migration_user>;
GRANT pg_read_all_settings TO <migration_user>;
```

1. If you're using Postgres 16 or earlier, also grant the following to allow the agent to discover the Postgres log file location:

```
GRANT EXECUTE ON FUNCTION pg_catalog.pg_current_logfile() TO <migration_user>;
```

Granting data migration permissions

> **Tip**
>
> Required for data migration with the DMS agent.

1. Grant the migration user usage on each schema containing tables to migrate:

```
GRANT USAGE ON SCHEMA <db_schema> TO <migration_user>;
```

1. Grant `SELECT` on source tables. For an entire schema:

```
GRANT SELECT ON ALL TABLES IN SCHEMA <db_schema> TO <migration_user>;
```

For individual tables:

```
GRANT SELECT ON <table_name> TO <migration_user>;
```

1. Grant `CREATE` on the source database so the DMS agent can create publications:

```
GRANT CREATE ON DATABASE <pg_db_name> TO <migration_user>;
```

Preparing for schema ingestion (EDB Postgres AI agent)

> **Tip**
>
> Required for schema ingestion with the EDB Postgres AI agent.

Creating a dedicated OS user for the agent

Create a dedicated OS user on your auxiliary machine to run `beacon-agent` . This strategy avoids granting the agent permissions it doesn't need. This is a Linux-specific example, the exact command might vary by distribution:

```
sudo useradd <agent_os_user>
```

Preparing for extension migration (EDB Postgres AI agent)

> **Tip**
>
> Required for schema migration with the EDB Postgres AI agent when your source database has extensions installed.

As part of schema migration, the EDB Postgres AI agent automatically extracts all extensions installed on the source database and applies them to the destination before any other schema DDL runs. If any extension fails to install, the entire schema migration aborts.

Before running schema migration, check which extensions are installed on the source and confirm they're available on the destination at the same version. If an extension isn't available at the matching version, schema migration fails. When migrating across different major Postgres versions, the extension versions available on the destination may not match those on the source, which can cause migration failures.

```
SELECT name, installed_version
FROM pg_available_extensions
WHERE installed_version IS NOT NULL;
```

Run the same query on the destination to compare versions.

After migration completes, if your source uses foreign data wrapper (FDW) extensions, the agent installs the FDW extension itself but doesn't migrate FDW objects. Recreate foreign servers, user mappings, and foreign tables on the destination manually.

For a full list of extension migration limitations, see *Extension migration limitations* .

Preparing for data migration (DMS agent)

> **Tip**
>
> Required for data migration with the DMS agent.

Configuring WAL settings

The DMS agent uses Postgres logical decoding to capture changes from the write-ahead log (WAL). Configure the following parameters in `postgresql.conf` :

1. Set `wal_level` to `logical` to enable logical decoding:

```
wal_level = logical
```

1. Set `max_wal_senders` to at least `1` . If other streaming clients connect to your database, consult your DBA for the appropriate value:

```
max_wal_senders = 1
```

1. Set `max_replication_slots` to at least `1` . This can be higher if your organization uses Postgres replication:

```
max_replication_slots = 1
```

1. Set `max_wal_size` large enough that production traffic generates mostly timed checkpoints rather than requested checkpoints based on WAL size. We recommend at least 8 GB:

```
max_wal_size = 8GB
```

Setting `max_wal_size` too small can allow Postgres to drop WAL entries before the DMS agent can stream them, interfering with or slowing down the migration. Consult your DBA to determine the appropriate size for your environment.

1. Restart the database server to apply the changes.

Creating the replication group role

The migration user must own the source tables to auto-create Postgres publications. Because those tables are already owned by another role, create a replication group role that both the current table owner and the migration user belong to:

```
CREATE ROLE <replication_group>;
GRANT <replication_group> TO <migration_user>;
GRANT <replication_group> TO <original_owner>;
ALTER TABLE <table_name> OWNER TO <replication_group>;
```

Where:

- `<migration_user>` is the Postgres user you created for migration.
- `<original_owner>` is the original owner of the tables.
- `<replication_group>` is the shared role that owns the source tables for publication auto-creation.
- `<table_name>` is the name of a table to migrate (case-sensitive).

You must run `ALTER TABLE ... OWNER TO <replication_group>` for every table you plan to migrate, across all schemas. To transfer ownership of all tables in the database at once:

```
DO $$
DECLARE
  s text;
  t text;
BEGIN
```

(continues on next page)

(continued from previous page)

```
  FOR s, t IN
    SELECT <schemaname>, <tablename> FROM pg_tables
    WHERE <schemaname> NOT IN (pg_catalog, information_schema)
  LOOP
    EXECUTE ALTER TABLE  || quote_ident(s) || . || quote_ident(t) ||  OWNER TO
↪<replication_group>;
  END LOOP;
END;
$$;
```

Configuring SSL

Configure your source database server to accept SSL connections . The DMS agent requires SSL to connect to the source. Create a server certificate and server private key — you can generate the private key with a tool like OpenSSL.

Validating your configuration

The DMS agent includes a validation script that checks whether your source database meets all requirements for CDC migration.

1. Navigate to the DMS agent folder:

```
cd /opt/cdcagent/reader/
```

1. Create an array of all tables to migrate in `<schema_name>.<table_name>` format:

```
arr=(schema1.table1 schema1.table2 schema2.table1)
```

1. Run the validation script:

```
PG_USERNAME=postgres PG_PASSWORD=<password> PG_HOST=<host> PG_PORT=5432 DB_NAME=<db_
↪name> DBZ_USERNAME=<migration_user> DBZ_PASSWORD=<migration_user_password> ./
↪postgresConfigValidation.sh "${arr[@]}"
```

A passing result looks like this:

```
 *** [Transporter] - Get PostgreSQL Major Version
PostgreSQL major version: 17
 *** [Transporter] - Validate WAL Level
wal_level:  logical
[Pass] wal_level is logical.
 *** [Transporter] - Validate max WAL senders
max_wal_senders: 10
[Pass] max_wal_senders is at least 1.
 *** [Transporter] - Validate max replication slots
max_replication_slots: 10
[Pass] max_replication_slots is at least 1.
 *** [Transporter] - Validate max WAL size
max_wal_size: 8 GB
[Pass] max_wal_size is set to the recommended value of 8GB or higher.
 *** [Transporter] - Validate checkpoints
checkpoints_timed: 100
checkpoints_req: 5
```

(continues on next page)

(continued from previous page)

```
[Pass] Timed checkpoints are more frequent than requested checkpoints.
 *** [Transporter] - Check <migration_user> user role
[Pass] User <migration_user> is present
[Pass] User <migration_user> has replication permission
 *** [Transporter] - Check SELECT privilege on the tables to be migrated
[Pass] User <migration_user> has select privilege on all tables to be migrated.
 *** [Transporter] - Check CREATE privilege on database
[Pass] Role <migration_user> has CREATE privilege on database <db_name>.
```

Note

Address any *[Failed]* or *[Suggestion]* statuses before proceeding. *[Failed]* checks are blocking issues you must resolve before the DMS agent can execute the data migration. *[Suggestion]* checks allow migration but can negatively affect performance if left unaddressed.

Next step

Configuring the EDB Postgres AI agent to register your AWS RDS/Aurora PostgreSQL database

Configuring the EDB Postgres AI agent to register your Postgres database

Configure the EDB Postgres AI agent (packaged as `beacon-agent`) to establish a secure connection with HM and retrieve schema and migration assessment data from your Postgres source.

Prerequisites

- **Machine user and access key**: Created in *Creating a machine user* . Store the access key in an environment variable:

```
export BEACON_AGENT_ACCESS_KEY="baak_your_key_here"
```

- **pg_dump**: The EDB Postgres AI agent uses `pg_dump` to extract DDL from your Postgres source. Ensure `pg_dump` is installed and configured on your auxiliary machine. See *Configuring your auxiliary machine or bastion host* for installation instructions, compatibility rules, and credential setup.

Note

The agent substitutes environment variables referenced in *beacon_agent.yaml* at runtime, so setting *$BEACON_AGENT_ACCESS_KEY* in your environment before running the agent is sufficient. Environment variables always take precedence over static values in the YAML.

Recommendations for setting environment variables

We recommend handling sensitive data, such as `$BEACON_AGENT_ACCESS_KEY` or `$DSN` , as environment variables managed through a secrets manager. This strategy prevents sensitive credentials from being stored in plaintext configuration files. If you use environment variables, make sure they're available to the EDB Postgres AI agent while it's running.

Consult your organization's IT department on the safest approach for production environments.

For short-lived migrations or testing and demo purposes, you can use the `export` command in an active terminal session.

Obtaining HM connection parameters

The agent requires specific values from your HM installation. Contact your administrator for:

- **Beacon server hostname**: The internal endpoint for the `beacon_server` .

When configuring the HM installation, administrators must specify an internal endpoint for the `beacon_server` service. Ask the administrator or installer of your HM instance to provide the hostname (URL) they set for the `beacon_server` service in the Helm chart configuration file used for installation. They'll look up the value they set for `parameters.upm-beacon.server_host` (or `BEACON_SERVICE_DOMAIN_NAME`) in the `values.yaml` file. You'll need this value later to configure the EDB Postgres AI agent.

- **Root certificate path (if required)**: If your HM instance uses a self-signed or internal CA, obtain the `ca_public.crt` file. Store it on the agent machine (for example, `/etc/beacon/ca_public.crt`).

The EDB Postgres AI agent requires a trusted connection to the `beacon_server` service in your HM instance. If the server running the EDB Postgres AI agent doesn't inherently trust HM's TLS certificate (managed by your organization's security infrastructure), ask the administrator or installer of your HM instance to provide the file with the certificate. You'll need to provide the directory path to this certificate later to configure the EDB Postgres AI agent.

See *Administrative tasks* for more information about the `beacon_server` and root certificate.

Verifying connectivity

Once you have the HM connection parameters, verify that your auxiliary machine can reach both the Postgres source and HM. The following ports must be open:

Destination	Port	Used by
Postgres host	5432 (default for PostgreSQL, AWS RDS, and Aurora); 5444 (default for EDB Postgres Advanced Server)	Both agents (database connection)
HM *beacon_server*	9443	EDB Postgres AI agent
HM DMS service (*RW_SERVICE_HOST*)	443 (HTTPS)	DMS agent

Note

Ports 5432 and 5444 are defaults. If your Postgres database is configured to listen on a different port, use that port instead and ensure it's open on your network.

Verify connectivity to each host and port with `nc` :

```
#  Postgres source

nc -zv <postgres-host-name-or-ip> 5432

#  HM beacon server

nc -zv <beacon_server_hostname> 9443

#  HM DMS service
```

(continues on next page)

(continued from previous page)

```
nc -zv <rw-service-hostname> 443
```

A successful connection returns `succeeded!` . If you see `Connection refused` or `Operation timed out` , the port isn't reachable and must be opened before proceeding.

Defining source database connection parameters

The data source name (DSN) is the connection string for your Postgres source.

Format:

```
DSN="postgresql://<migration_user>:<migration_user_password>@<host>:<port>/<database_
↪name>"
```

Set the DSN as an environment variable before running the agent or configuring the service:

```
export DSN="postgresql://migration_user:password@postgres-host:5432/mydb"
```

If your database has SSL disabled, append `?sslmode=disable` :

```
export DSN="postgresql://migration_user:password@postgres-host:5432/mydb?
↪sslmode=disable"
```

Use the RDS instance endpoint as the host. RDS enforces SSL by default, so append `?sslmode=require` :

```
export DSN="postgresql://migration_user:password@myinstance.abc123.us-east-1.rds.
↪amazonaws.com:5432/mydb?sslmode=require"
```

Use the Aurora cluster endpoint as the host. Aurora enforces SSL by default, so append `?sslmode=require` :

```
export DSN="postgresql://migration_user:password@my-cluster.cluster-abc123.us-east-1.
↪rds.amazonaws.com:5432/mydb?sslmode=require"
```

```
DSN="postgresql://migration_user:pa%24%24word@postgres-host:5432/mydb"
```

Preparing a configuration file

1. Create an EDB Postgres AI agent configuration directory in your home directory:

```
mkdir ${HOME}/.beacon
```

> **Note**
>
> The agent searches for *beacon_agent.yaml* in */etc/beacon/* or *${HOME}/.beacon/* . As a final fallback, it searches the directory from which it's executed. !!!

1. Inside this directory, create a file `beacon_agent.yaml` :

```
touch ${HOME}/.beacon/beacon_agent.yaml
```

1. Copy and paste the following template into the new file.

In this template, `$BEACON_AGENT_ACCESS_KEY` and `$DSN` are the environment variables you configured previously. Replace all < > placeholders in the template. See *EDB DMS agent parameter reference* for details.

```
---
agent:
  access_key: $BEACON_AGENT_ACCESS_KEY
  beacon_server: <beacon_server_hostname>:9443
  project_id: <your_project_id>
  providers:
    - "onprem"
  # In most HM deployments, set `root_ca_path` to the path of the ca_public.crt
  # file obtained from your HM administrator (for example, /etc/beacon/ca_public.crt).
  # Leave empty only if HMs certificate is already trusted by the systems CA bundle.
  root_ca_path: ""
  schema_providers:
    # Include "onprem-schema" to enable schema ingestion and migration assessment.
    # Remove this section if you dont want schema information to be collected.
    - "onprem-schema"
provider:
  onprem:
    databases:
      # `resource_id` controls how this database appears in HM Estate.
      - resource_id: "<database_resource_id>"
        dsn: $DSN
        # pg_bin_path: /usr/lib/postgresql/16/bin
        # Optional. Absolute path to the directory containing pg_dump and pg_dumpall.
        # Set when registering databases from different Postgres distributions/
↪versions.
        schema:
          enabled: true
          poll_interval: 15s
general:
  logging:
    # `info` is simplified, use debug during initial setup or
    # troubleshooting for more verbose output.
    level: info
```

1. Save the file.

Running the EDB Postgres AI agent

1. Before starting the agent, verify connectivity and configuration syntax:

```
beacon-agent validate
```

1. Run the agent:

```
beacon-agent
```

> **Tip**
>
> See Agent CLI for other agent modes and options. !!!

1. Monitor the ingestion process as the agent starts sending data to HM. After a few minutes, your Postgres database appears in the HM console under **Estate > Migrations**.

Important

The agent dynamically detects changes to tables and columns. However, changes to views, procedures, or functions require an agent restart to re-ingest metadata.

For production use, see *Running the agent as a service* .

Verifying the migration assessment

To confirm your Postgres database was recognized by the agent and is ready for migration, go to **Estate > Migrations** in the HM console.

If the database appears in the table, you have access to the following resources:

- A **migration assessment** on the *Migrations* of the **Estate** page, providing a summarized view of database schema count, core count, migration complexity, and level of effort.
- A *Viewing migration assessment details* accessible by selecting a specific database from the **Migrations** tab. This tab provides an expanded analysis of **Schemas**, a global **Assessment**, and a **Migration** hub with direct connections to other HM tools for performing migrations.
- **Automatic schema DDL extraction** that you can use for schema migrations with the built-in Migration Portal. Once ingestion is complete, the Migration Portal automatically creates a project per database, which you can use to prepare and migrate schemas to the destination database.

Troubleshooting

There are several reasons why the agent might fail to connect to HM or your Postgres database. Here are some tools you can use for troubleshooting.

`beacon-agent validate` command

The `beacon-agent validate` command is a helpful step to identify common connectivity and configuration issues.

```
beacon-agent validate
```

The validate command checks for the following:

- **Server connectivity**: Verifies the agent can connect and authenticate to the configured `beacon_server` (usually on port `9443`).
- **Database connectivity**: Attempts to connect to each configured database DSN. This is a full connection test, not just a syntax or URL-encoding check.
- **Configuration display**: Shows TLS settings, project ID, providers, and per-database settings to help identify misconfiguration.

`beacon-agent` logs

If the agent reports errors during ingestion, you can check the logs for more details. If you're running the agent in the foreground, logs print to the terminal. If you set up the agent as a systemd service, use `journalctl` to view logs.

Here are some common error messages you might encounter in the logs:

Error	Cause	Fix
tls: failed to verify certificate: x509: certificate signed by unknown authority	The agent can't verify HM's TLS certificate because the CA isn't trusted by the system.	Set *root_ca_path* in *beacon_agent.yaml* to the path of the *ca_public.crt* file obtained from your HM administrator. If you can't obtain the certificate, you can set *plaintext: true* in the agent config as a temporary workaround (not recommended for production).
rpc error: code = Unknown desc = no rows in result set	The beacon-agent version is incompatible with the server.	Upgrade to beacon-agent v1.384.0 or later.

Running the agent as a service

For continuous metric streaming and to ensure your Postgres database remains visible in **HM console > Estate**, configure the agent to run as a systemd service. Running as a service ensures it starts automatically on system boot and restarts after any errors.

Without a service, the agent streams metrics only while you run it manually. Your database will disappear from the HM estate 72 hours after the agent stops streaming.

> **Note**
>
> Ensure that any environment variables referenced in *beacon_agent.yaml* (for example, *$BEACON_AGENT_ACCESS_KEY* and *$DSN*) are accessible to the service at runtime. If you configured a *.pgpass* file for schema extraction credentials (as described in *Configuring the environment*), ensure that file is present in the home directory of the user specified in *User=* .

1. Create the service file. This example uses `vi` , but you can use any text editor:

```
sudo vi /usr/lib/systemd/system/beacon-agent.service
```

Default service file paths by operating system:

Operating system	Default path
RHEL	*/usr/lib/systemd/system/beacon-agent.service*
Rocky Linux	*/usr/lib/systemd/system/beacon-agent.service*
AlmaLinux	*/usr/lib/systemd/system/beacon-agent.service*
Ubuntu	*/etc/systemd/system/beacon-agent.service*
Debian	*/lib/systemd/system/beacon-agent.service*

1. Paste the following template into the file and replace all `< >` placeholders:

```
[Unit]
Description=EDB Postgres AI Agent
After=network.target

[Service]
Type=simple
User=<agent_os_user>
WorkingDirectory=</home/agent_os_user/>
ExecStart=</usr/local/bin/beacon-agent>
Restart=on-failure
```

(continues on next page)

(continued from previous page)

```
RestartSec=60

[Install]
WantedBy=multi-user.target
```

Setting	Placeholder	Description
User=	*<agent_os_user>*	OS user that runs the agent. Use the dedicated OS user you created when preparing the source database.
WorkingDirectory=	*</home/agent_os_user/>*	Directory where the *.beacon* folder is stored. This is typically the home directory of *<agent_os_user>*.
ExecStart=	*</usr/local/bin/beacon-agent>*	Full path to the *beacon-agent* binary. The path depends on how and where the agent was installed.

1. Reload systemd, then enable and start the service:

```
sudo systemctl daemon-reload
sudo systemctl enable beacon-agent.service
sudo systemctl start beacon-agent.service
```

1. Confirm the service started successfully by checking the logs:

```
journalctl -u beacon-agent.service
```

Related topics

- For more granular control of schema ingestion (for example, to exclude or include schemas), see *Controlling schema ingestion* .
- For the full set of configuration parameters, see *EDB DMS agent parameter reference* .
- If your administrator needs guidance on obtaining the root certificate or `beacon_server` value, see *Administrative tasks* .

Next step

Preparing your HM-managed destination database

Preparing your self-managed destination database

Prepare your self-managed Postgres database to serve as the destination for your migration.

If you already have a self-managed database you want to use as the destination, you can skip to *Configuring SSL* . Before proceeding, ensure that the tables you migrate and the load generated during migration won't interfere with existing workloads on the server.

Creating the destination database

If you don't have a destination database yet, create one. This example assumes you have installed the Postgres database distribution of your choice and initialized the server.

1. Connect to the destination Postgres server:

```
psql -p <PG_PORT> -U <PG_USERNAME>
```

1. Create an empty database to use as the migration destination. In this example the destination database is `tr_target`:

```
CREATE DATABASE tr_target;
```

Configuring SSL

Configure your destination Postgres server to accept SSL connections so the EDB DMS agent in writer mode can connect to it securely. To enable SSL, create a server certificate and a server private key (for example, with OpenSSL).

The following example creates a self-signed certificate and key with OpenSSL for a PostgreSQL 17 server on Ubuntu. The bin and data directories may vary depending on your OS, *Supported Postgres distributions* , and version.

1. Generate an SSL certificate and key:

```
openssl req -new -x509 -days 365 -nodes -out server.crt -keyout server.key
```

Follow the terminal prompts to create the files.

1. Move the created files to the Postgres data directory:

```
sudo cp server.crt server.key /var/lib/postgresql/17/<data_directory>
```

1. Set the required permissions:

```
sudo chmod 600 /var/lib/postgresql/17/<data_directory>/server.key
sudo chmod 600 /var/lib/postgresql/17/<data_directory>/server.crt
sudo chown postgres:postgres /var/lib/postgresql/17/<data_directory>/server.crt /var/
↪lib/postgresql/17/<data_directory>/server.key
```

1. Enable SSL in `postgresql.conf`:

```
ssl = on
ssl_cert_file = server.crt
ssl_key_file = server.key
```

1. Allow local SSL connections in `pg_hba.conf`:

```
hostssl all             all             0.0.0.0/0               md5
```

1. Restart your server to apply the changes:

```
/usr/lib/postgresql/17/bin/pg_ctl -D /var/lib/postgresql/17/<data_directory> restart
```

Creating a migration role

The EDB DMS agent in writer mode requires write access to the destination database. You can use an existing superuser or create a dedicated migration role. This example creates a dedicated role:

1. Create a role with login access:

```
CREATE ROLE <MIGRATION_ROLE_WRITER> WITH LOGIN PASSWORD <ROLE_PASSWORD>;
```

Where `<MIGRATION_ROLE_WRITER>` is the name of the role to use for destination database access.

1. Grant control over replication sessions:

```
GRANT SET ON PARAMETER session_replication_role TO <MIGRATION_ROLE_WRITER>;
```

> **Note**
>
> Granting control over replication sessions is supported only on Postgres 15 and later. If your destination database is on Postgres 14 or earlier, use a superuser instead. !!!

Supply this role to the DMS writer agent via the `DBCONFIG_DATABASES_0__USERNAME` and `DBCONFIG_DATABASES_0__PASSWORD` values when you configure it in the next steps.

Next step

Migrating Postgres schemas

Migrating Postgres schemas

Before starting a data migration, choose how to handle schema migration. HM supports two approaches:

- **Simplified (recommended)**: HM manages schema migration and constraint handling automatically as part of the data migration setup.
- **Manual**: Export and import the schema yourself using Migration Toolkit or `pg_dump` , and handle constraints based on the key structure of each source table.

No manual schema preparation is required. HM handles schema migration and constraint ordering automatically when you select the **Schema and data** scope later. Before proceeding, confirm:

- The EDB Postgres AI agent has ingested your source schema and a Migration Portal project exists for the database. To verify, go to **Estate > Migrations** in the HM console, select the database in the **Database Name** column, and confirm a project appears in the **MP Project** column.
- When you configure the DMS reader agent in a later step, you'll set `DBCONFIG_DATABASES_<N>__RESOURCEID` to the same `resource_id` value as in `beacon_agent.yaml` . The **Schema and data** and **Schema only** scopes will fail without it.

Continue to the next step.

Before you use EDB DMS for data migration, export and import your schema to the destination database. DMS migrates data only — the destination database must have schemas in place before migration begins.

Handling constraints

How to handle constraints depends on the constraint type and the key structure of each source table.

Constraint type	Action	Notes
Primary key	Exclude now — DMS applies automatically	The DMS writer applies the primary key when the migration enters the streaming phase, using it to deduplicate snapshot data.
Unique key (source table has a primary key)	Exclude now, apply after streaming starts	Primary key-based deduplication ensures no duplicate rows remain when you apply unique constraints after streaming starts.

continues on next page

Table 38 – continued from previous page

Constraint type	Action	Notes
Unique key (source table has no primary key)	Not currently supported	Use the simplified path instead — select the **Schema and data** scope when creating the migration to let HM handle schema migration automatically.
NOT NULL	Include now	*NOT NULL* validation adds no meaningful overhead to the data load, unlike foreign key and check constraint validation.
Foreign key, check, and exclusion	Exclude now, reapply after migration	These cause unnecessary CPU and memory overhead while data is actively being migrated.

Note

For tables with no primary key or unique key, no key constraints are needed before or after migration. DMS treats these tables as keyless and provides at-least-once delivery. Deduplicate rows as part of your post-migration verification.

Exporting your schema

We recommend using EDB Migration Toolkit to export your schema. MTK's offline migration capability generates SQL scripts you can apply on the destination without direct connectivity. Other valid export methods include `pg_dump` /`pg_restore` , pgAdmin, or `psql` .

When using Migration Toolkit:

- Use the `-offlineMigration` option to generate SQL scripts.
- Use the `-schemaOnly <schema_scope>` option to export schema only.
- Exclude (comment out) primary key, unique, foreign key, check, and exclusion constraints from the generated script before importing.
- If you plan to map objects to different names in the destination, edit them directly in the DDL script before importing. Update all dependent objects — views, constraints, and other references — that reference renamed objects. See *Mapping schema, tables, and columns* for details.

Importing your schema

After preparing the DDL and excluding the deferred constraints, connect to the destination database and apply the SQL file by executing the generated offline migration script .

Note

Apply the excluded unique, foreign key, check, and exclusion constraints after migration completes. The DMS writer applies primary key constraints automatically — don't reapply them manually. Keep track of excluded constraints so you can reapply them when the time comes.

Next step

Configuring and running the EDB DMS agent

Configuring and running the EDB DMS agent

Because the destination is a self-managed Postgres database, this migration requires two DMS agent instances: one in reader mode to read changes from the source, and one in writer mode to apply them to the destination. Install and configure both before creating a migration.

Prerequisites

- Ensure the machine where you're running the DMS agents has Java/OpenJDK 17 or later installed.
- *Install the EDB DMS agent* .

Creating migration credentials

Migration credentials authenticate the DMS agent to Hybrid Manager (HM). You need one set of credentials per HM project, and both the reader and writer agents use the same credentials.

1. In the HM console, select the project you're using for your migration.
2. Select **Migrate > Credentials**.
3. Select **Create Migration Credential**, enter a name, and select **Create Migration Credential**. A `credentials.zip` file is downloaded to your machine.

> **Note**
>
> Store the credentials safely. If you lose them, you'll need to revoke this set and create new ones. !!!

1. Unzip the downloaded file and copy the credentials folder to each machine where you installed a DMS agent — for example, to `$HOME/credentials/` . If you use a different path, update `CREDENTIAL_DIRECTORY_PATH` in each script to match.

The folder contains:

- `int.crt`
- `int.truststore.p12`
- `client.keystore.p12`
- `client-key.pem`
- `client-cert.pem`

Configuring the reader DMS agent

The reader agent connects to your source Postgres database and streams changes to HM.

1. Open `/opt/cdcagent/run-cdcagent.sh` on the reader machine with write permissions.
2. Set the required variables. Optional parameters are commented out by default — uncomment and configure any that apply to your environment.

See the *EDB DMS agent parameter reference* for details.

```
#!/bin/bash -e
# run-cdcagent.sh
#
# This script provides a convenient place to specify
```

(continues on next page)

(continued from previous page)

```
# environment variables used to configure the
# EDB Data Migration Service Agent.
#

########################################
# DMS Agent General Configuration      #
########################################

# This ID is used to identify DMS Agent
# and is specified by the user.
export DBCONFIG_ID=<source_name_of_choice>

# Determine the run mode of the DMS Agent
# The run mode can be set to either reader or writer
# reader mode is used to read changes from a source database
# writer mode is used to write changes to a target database
export RUN_MODE=reader

# This is the DMS backend service used by the Agent
# Obtain this value from the HM console Sources or Destinations migration pages
export RW_SERVICE_HOST=<rw_service_host>

########################################
# DMS Agent TLS Configuration          #
########################################

# You need to create migration credentials in EDB Postgres AI platform
# and set the path of the credentials directory
export CREDENTIAL_DIRECTORY_PATH=$HOME/credentials

########################################
# DMS Agent DB Configuration           #
########################################

# A sample configuration to create a single Postgres database connection:
export DBCONFIG_DATABASES_0__TYPE=POSTGRES
export DBCONFIG_DATABASES_0__HOSTNAME=<localhost/host>
export DBCONFIG_DATABASES_0__PORT=<5432>
# The CATALOG is the database name
export DBCONFIG_DATABASES_0__CATALOG=<source>
export DBCONFIG_DATABASES_0__USERNAME=<postgres>
# The password env can be set without specifying it here
# but the env structure looks like this
export DBCONFIG_DATABASES_0__PASSWORD=<password>
# Required for Schema and data or Schema only scope.
# Must match resource_id in beacon_agent.yaml.
#export DBCONFIG_DATABASES_0__RESOURCEID=<resource_id>

# You can increase the index to configure more than
# one database for the DMS Agent
#export DBCONFIG_DATABASES_1__TYPE=POSTGRES
#export DBCONFIG_DATABASES_1__HOSTNAME=<localhost/host>
```

(continues on next page)

(continued from previous page)

```
#export DBCONFIG_DATABASES_1__PORT=<5432>
#export DBCONFIG_DATABASES_1__CATALOG=<source>
#export DBCONFIG_DATABASES_1__USERNAME=<postgres>
#export DBCONFIG_DATABASES_1__PASSWORD=<password>
#export DBCONFIG_DATABASES_1__RESOURCEID=<resource_id>

########################################
# Optional Parameters Below            #
########################################

# FIPS mode can be enabled by setting the FIPS_MODE env variable to true
#export FIPS_MODE=true

# Configure logging
# Loglevel for the entire Quarkus application and underlying components
# WARNING: Enabling this loglevel is extremely verbose
#export QUARKUS_LOG_LEVEL=DEBUG
# Loglevel for the EDB package
# Preferred mode for debugging issues related to the logic of the agent
#export QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL=DEBUG

cd "$(dirname "$0")"

echo "Starting in mode: $RUN_MODE"

case "$RUN_MODE" in
  reader)
    java -jar ./reader/quarkus-run.jar
    ;;
  writer)
    java -jar ./writer/quarkus-run.jar
    ;;
  *)
    echo "Unknown RUN_MODE: $RUN_MODE"
    exit 1
    ;;
esac
```

Running the reader DMS agent

Navigate to the DMS agent directory and run the configuration script:

```
cd /opt/cdcagent
./run-cdcagent.sh
```

The script exports all required environment variables and starts the agent process. Once running, the reader agent registers your Postgres source in HM. Continue to the next section to verify it appears correctly.

Verifying the source in HM

1. In the HM console, select your project.
2. Select **Migrate > Sources**.

Your database is listed with its name, type, agent ID, and status. For a description of all fields on this page, see *Understanding deployment resources* .

Configuring the writer DMS agent

The writer agent connects to your destination Postgres database and applies streamed changes from HM. Configure it on the machine where you installed the writer instance of the DMS agent.

1. Open `/opt/cdcagent/run-cdcagent.sh` on the writer machine with write permissions.
2. Set the required variables. Optional parameters are commented out by default — uncomment and configure any that apply to your environment.

See the *EDB DMS agent parameter reference* for details.

```
#!/bin/bash -e
# run-cdcagent.sh
#
# This script provides a convenient place to specify
# environment variables used to configure the
# EDB Data Migration Service Agent.
#

########################################
# DMS Agent General Configuration      #
########################################

# This ID is used to identify DMS Agent
# and is specified by the user.
export DBCONFIG_ID=<destination_name_of_choice>

# Determine the run mode of the DMS Agent
# The run mode can be set to either reader or writer
# reader mode is used to read changes from a source database
# writer mode is used to write changes to a target database
export RUN_MODE=writer

# This is the DMS backend service used by the DMS Agent
# Obtain this value from the HM console Sources or Destinations migration pages
export RW_SERVICE_HOST=<rw_service_host>

########################################
# DMS Agent TLS Configuration          #
########################################

# You need to create migration credentials in EDB Postgres AI platform
# and set the path of the credentials directory
export CREDENTIAL_DIRECTORY_PATH=$HOME/credentials

########################################
```

(continues on next page)

(continued from previous page)

```
# DMS Agent DB Configuration            #
########################################

# A sample configuration to create a single Postgres database connection:
export DBCONFIG_DATABASES_0__TYPE=POSTGRES
export DBCONFIG_DATABASES_0__HOSTNAME=<localhost/host>
export DBCONFIG_DATABASES_0__PORT=<5432>
# The CATALOG is the database name
export DBCONFIG_DATABASES_0__CATALOG=<destination_db_name>
export DBCONFIG_DATABASES_0__USERNAME=<MIGRATION_ROLE_WRITER>
# The password env can be set without specifying it here
# but the env structure looks like this
export DBCONFIG_DATABASES_0__PASSWORD=<password>

# You can increase the index to configure more than
# one database for the DMS Agent
#export DBCONFIG_DATABASES_1__TYPE=POSTGRES
#export DBCONFIG_DATABASES_1__HOSTNAME=<localhost/host>
#export DBCONFIG_DATABASES_1__PORT=<5432>
#export DBCONFIG_DATABASES_1__CATALOG=<destination_db_name>
#export DBCONFIG_DATABASES_1__USERNAME=<MIGRATION_ROLE_WRITER>
#export DBCONFIG_DATABASES_1__PASSWORD=<password>

########################################
# Optional Parameters Below             #
########################################

# FIPS mode can be enabled by setting the FIPS_MODE env variable to true
#export FIPS_MODE=true

# Configure logging
# Loglevel for the entire Quarkus application and underlying components
# WARNING: Enabling this loglevel is extremely verbose
#export QUARKUS_LOG_LEVEL=DEBUG
# Loglevel for the EDB package
# Preferred mode for debugging issues related to the logic of the agent
#export QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL=DEBUG

cd "$(dirname "$0")"

echo "Starting in mode: $RUN_MODE"

case "$RUN_MODE" in
  reader)
    java -jar ./reader/quarkus-run.jar
    ;;
  writer)
    java -jar ./writer/quarkus-run.jar
    ;;
  *)
    echo "Unknown RUN_MODE: $RUN_MODE"
    exit 1
```

(continues on next page)

(continued from previous page)

```
    ;;
esac
```

Running the writer DMS agent

Navigate to the DMS agent directory and run the configuration script:

```
cd /opt/cdcagent
./run-cdcagent.sh
```

Once running, the writer agent registers your Postgres destination in HM. Continue to the next section to verify it appears correctly.

Verifying the destination in HM

1. In the HM console, select your project.
2. Select **Migrate > Destinations**.

Your database is listed with its name, type, agent ID, and status. For a description of all fields on this page, see *Diverse migration destinations* .

Next step

Migrating data from an AWS RDS or Aurora PostgreSQL database to an HM-managed database

Migrating data from a self-managed Postgres database to a different self-managed Postgres database

With both DMS agents running and your schema either imported or ready for HM to import, you're ready to create a migration in the HM console. This page covers the full migration lifecycle: creating the migration, monitoring its progress, finalizing it, and applying post-migration constraints.

The steps below vary depending on the approach you chose on the previous page. Select the tab that matches your choice.

Creating a migration

A migration establishes a sync between your Postgres source and a self-managed destination database via HM.

1. In the HM console, select the project from which you want to manage the migration.
2. In the project, select **Migrate > Migrations**.
3. In the **Migrations** page, select **Create New Migration**.
4. In the **Create OLTP Migration** page, assign a name to the migration.
5. Select the source of the migration. The ID for the reader DMS agent is listed in the **Source** menu.
6. Under **Scope**, select the option that matches your approach:

Select **Schema and data**. When you select this option, HM:

1. Sources the schema from the Migration Portal project that was created automatically when you registered your database with the EDB Postgres AI agent.

2. Creates tables on the destination without constraints, so the initial snapshot copy isn't slowed by constraint validation.
3. After the snapshot completes, applies primary key and unique constraints (for tables that have them) so DMS can deliver streaming changes exactly-once. Tables without a PK or UK fall back to at-least-once delivery and require deduplication during verification.
4. Defers the remaining schema (indexes, foreign keys, check constraints, `NOT NULL`, and similar) until cutover. You can optionally apply most of this work concurrently with streaming via **Finalize Schema Concurrently** to shrink the cutover window — see *Finalizing a streaming migration* .
5. After you mark the migration as completed, validates deferred foreign keys, syncs sequences, and applies any remaining lightweight tasks in a final **Schema Finalization** stage.

> **Note**
>
> Use **Schema only** if you want to migrate the database structure without migrating any table data. !!!

Select **Data only**. The schema was already imported manually in the previous step.

1. Select the snapshot type: ***Snapshot*** or ***Snapshot + Streaming*** .
2. Under **Destination Type**, select a self-managed instance as the destination for the migration. The ID for the writer DMS agent is listed in the **Destination** menu. Then, select **Next**.
3. On the **Review your Tables** tab, select the tables and columns to include in the migration. Expand a table to view its columns and optionally *map source columns to different destination column names* . Column mapping is only available for the **Data only** scope.
4. Select **Create Migration**.

The HM console displays the *new migration* . If you chose **Snapshot + Streaming**, the DMS agents stream data changes continuously while the migration shows a **Running** state.

> **Note**
>
> The DMS agent streams data changes only — not DDL changes (schema modifications).

Monitoring progress

Migration list view

The **Migrations** page shows a summary of all migrations in the project. The **Migration progress** column displays three metrics per migration:

- **Total number of tables** — total tables included in this migration
- **Snapshot tables completed** — tables that have finished the snapshot stage
- **Migrated rows** — cumulative rows migrated across all tables

Migration details view

Select a migration from the list to open the details page, which has three tabs:

- **Overview** — shows the current migration state and general information about the source and destination databases. For **Snapshot + Streaming** migrations, the migration is caught up when the status is **Streaming**. For **Snapshot** migrations, it's finished when the status is **Completed**.

- **Tables** — shows all tables selected for migration. For each table, you can see source and destination table and schema names, snapshot status, mapped columns, migrated rows, and migration rate. Select the + icon to expand a table and view column-level mappings.
- **Error history** — lists all errors logged for this migration. Select the **i** icon to open the full error log for a specific error. For stuck or failed migrations, active errors here indicate where to start troubleshooting.

DMS agent logs

The HM console doesn't surface all DMS agent errors. If a migration is stuck or no tables are progressing, check the terminal where you ran `run-cdcagent.sh` for `ERROR` log entries — these often contain the root cause. Common issues include version incompatibility between the DMS agent and HM, and connectivity or credential failures.

Finalizing a streaming migration

> **Note**
>
> This section applies only to **Snapshot + Streaming** migrations. For **Snapshot** migrations, DMS marks the migration as completed automatically.

Monitoring catch-up before cutover

Before stopping writes on the source, give the destination time to catch up. Use the **Migration details** page to track row counts and replication lag. Begin the cutover only after the destination is consistently close to the source.

Finalizing schema concurrently (recommended)

If you used the **Schema and data** scope, HM can apply most schema objects while the stream is still running: primary keys, unique constraints, indexes (built with `CREATE INDEX CONCURRENTLY` , so live writes aren't blocked), foreign keys (added as `NOT VALID`), check constraints, `NOT NULL` constraints, and similar. Foreign key validation, sequence synchronization, and other lightweight tasks run later in the post-cutover **Schema Finalization** stage.

Running this work concurrently shrinks the read-only cutover window. The larger your tables and the more indexes they have, the greater the time saved.

Once every table has completed its snapshot and the migration enters the streaming phase, the **Migration details** page shows a **Schema Finalization Available** alert with a **Finalize Schema Concurrently** button. The button stays available throughout streaming, so you can pick the moment based on how close the destination is to catching up.

1. Select **Finalize Schema Concurrently** on the **Migration details** page.
2. HM applies the deferred objects while streaming continues. The migration remains in the **Running** state, with a progress sub-step labeled **Finalizing Schema Concurrently**.

When the sub-step completes, proceed to the cutover.

> **Note**
>
> **Finalize Schema Concurrently** is available only for migrations created with the **Schema and data** scope with streaming enabled.

If you skip this step, the same objects are applied serially after you mark the migration as completed. See *Schema finalization stage* . The cutover window is larger in that case because indexes are built non-concurrently and constraints are validated under exclusive locks.

Not applicable to the manual path. You applied the schema before migration and will reapply deferred constraints manually after cutover. See *Applying deferred constraints and verifying data* .

Performing the cutover

When you're ready to switch traffic from the source to the destination:

1. Stop all writes on the source database.
2. Verify in the HM console that the destination has applied the last source change.
3. On the **Migration details** page, select **Mark as Completed**.

This stops the streaming process.

Schema finalization stage

For migrations created with the **Schema and data** scope, the migration enters a **Schema Finalization** stage automatically after **Mark as Completed**. What runs in this stage depends on whether you ran concurrent finalization first:

- **If you ran concurrent finalization**: HM runs the lightweight remaining work — foreign key validation, sequence sync, and any other deferred metadata tasks. This stage is short.
- **If you skipped concurrent finalization**: HM applies the full deferred set now (primary keys, unique constraints, indexes, foreign keys, check constraints, `NOT NULL` , and similar) followed by the lightweight tasks above. This stage takes substantially longer and runs while the source is read-only.

The migration is fully complete only after the **Schema Finalization** stage finishes. Progress is shown on the **Migration details** page.

No automatic schema finalization runs for the **Data only** scope. Continue to *Applying deferred constraints and verifying data* to reapply the foreign key, unique, check, and exclusion constraints you deferred before migration.

Applying deferred constraints and verifying data

Reapplying deferred constraints

HM applied all deferred constraints and synced sequences automatically during the **Schema Finalization** stage. No manual action is needed. See *Schema finalization stage* .

At the beginning of your migration, you prepared and imported the schema excluding foreign key, unique, check, and exclusion constraints. Connect to the destination database and reapply those constraints now.

Primary key constraints were applied automatically by the DMS writer when the migration entered the streaming phase. `NOT NULL` constraints were applied before migration and don't require any action now.

Deduplicating rows when needed

For tables that had no primary key or unique constraints, the EDB DMS agent provides at-least-once delivery, which can result in duplicate rows. Perform deduplication as part of your verification step.

Verifying migrated data

Compare the source and destination databases to confirm all data was migrated successfully. You can use LiveCompare for this.

Related topics

- LiveCompare — row-level data comparison tool for verifying migration completeness
- *Terminology* — definitions for snapshot, streaming, and other migration concepts

Migrate your self-managed Postgres database to an HM-managed database

These are the tasks involved in an end-to-end migration from a self-managed Postgres database to a database cluster managed by Hybrid Manager (HM). The process uses two agents: the EDB Postgres AI agent (`beacon-agent`) to register your Postgres database with HM and ingest schema for assessment, and the EDB DMS agent (`cdcagent`) to perform the actual data migration.

1. ** *Configuring your auxiliary machine or bastion host* ** — Configure the machine that hosts both migration agents. This machine acts as the bridge between your Postgres source and HM.
2. ** *Preparing your AWS RDS/Aurora PostgreSQL source database* ** — Create the required users and configure your Postgres database for schema ingestion with the EDB Postgres AI agent and data migration with the DMS agent.
3. ** *Configuring the EDB Postgres AI agent to register your AWS RDS/Aurora PostgreSQL database* ** — Connect the agent to your source database and to HM to enable schema ingestion and migration assessment.
4. ** *Preparing your HM-managed destination database* ** — Create or identify the HM-managed database cluster that will receive the migrated schema and data.
5. ** *Migrating Postgres schemas* ** — Choose between the simplified path, where HM handles schema migration and constraint ordering automatically, or the manual path, where you export, prepare, and import the schema yourself.
6. ** *Configuring and running the EDB DMS agent* ** — Create migration credentials, configure the DMS agent in reader mode to read changes from your Postgres source, and verify the source appears in HM.
7. ** *Migrating data from an AWS RDS or Aurora PostgreSQL database to an HM-managed database* ** — Create and run the data migration in the HM console, monitor its progress, finalize streaming, and apply post-migration constraints.

Configuring your auxiliary machine or bastion host

The EDB Postgres AI agent (`beacon-agent`) and the Data Migration Service (DMS) agent (`cdcagent`) can run on the Postgres database server itself or on a separate auxiliary machine or bastion host. In most production environments, running the agents on a dedicated auxiliary machine is preferred — it avoids installing additional software on the database server, and a single agent instance can connect to databases on multiple servers.

Required software

Ensure you install the following software on your auxiliary machine before running the agents:

- **Java 17 or higher** — required by the DMS agent (`cdcagent`). Verify with `java -version` .
- **EDB Postgres AI agent** (`beacon-agent`) — required for database registration, schema assessment, and schema migration. Check the *beacon-agent version compatibility matrix* . Then install a compatible version of the EDB Postgres AI agent .
- **DMS agent** (`cdcagent`) — handles data migration and change data capture (CDC). Before installing, check the *DMS agent version compatibility* . Then *install a compatible version of the DMS agent* .

Configuring the environment

When you enable schema ingestion, `beacon-agent` uses `pg_dump` to extract DDL from your Postgres source. Postgres client tools packages include the `pg_dump` utility — it might already be available on your machine. Verify first:

```
pg_dump --version
```

If the command isn't found, install the client tools that match your source database distribution.

Once installed, ensure your `pg_dump` binary meets two compatibility rules:

- **Version rule** — The `pg_dump` major version must be greater than or equal to the major version of your source database. For example, if your source runs Postgres 16, you need `pg_dump` 16 or later. The agent validates this at startup and reports an error if the versions are incompatible.
- **Distribution rule** — The `pg_dump` binary must come from the same Postgres distribution as the source database. Each distribution's `pg_dump` accesses distribution-specific catalog tables. Using a mismatched binary (for example, community PostgreSQL `pg_dump` against an EDB Postgres Advanced Server source) will fail because the expected catalog objects won't exist.

Managing multiple Postgres versions on a host

If you have multiple Postgres versions on the same machine, ensure the correct `pg_dump` appears first in your `PATH` :

```
which pg_dump
pg_dump --version
```

On Debian and Ubuntu systems, use `update-alternatives` to manage the active version:

```
sudo update-alternatives --config pg_dump
```

Registering databases from multiple distributions or versions

A single `beacon-agent` instance can register databases from different Postgres distributions (for example, community PostgreSQL and EDB Postgres Advanced Server) or different major versions at the same time. Because the version and distribution rules apply per database, each database connection may require a different `pg_dump` binary.

Install all the required `pg_dump` binaries on the auxiliary machine. Later you can use the `pg_bin_path` setting in your `beacon_agent.yaml` configuration file to point each database entry to the correct binary directory. When set, it overrides the system `PATH` lookup for that specific database. When omitted, the agent falls back to the first `pg_dump` found in `PATH` . For example:

```
provider:
  onprem:
    databases:
      - resource_id: "pg16-community"
        dsn: $DSN_PG16
        pg_bin_path: /usr/lib/postgresql/16/bin
        schema:
          enabled: true
      - resource_id: "epas17"
        dsn: $DSN_EPAS17
        pg_bin_path: /usr/edb/as17/bin
        schema:
          enabled: true
```

Understanding credentials and pgpass behavior

The OS user that runs `beacon-agent` is the same user that executes `pg_dump`. To avoid exposing database credentials in process arguments, the agent writes the password to a temporary `pgpass` file with restrictive permissions (`0600`) and passes a password-free connection string to `pg_dump`. The agent creates the file in the OS temporary directory and removes it automatically after each extraction completes.

> **Note**
>
> If the agent process is killed unexpectedly (for example, by *SIGKILL*), the agent may not clean up the temporary pgpass file. Only the OS user that runs the agent can read the file. The OS removes it on reboot, or on container restart if the agent runs in a container. To check for orphaned files, look for files matching *.pgpass-** in */tmp*.

Next step

Preparing your AWS RDS/Aurora PostgreSQL source database

Preparing your Postgres source database

Configure your Postgres source database and create the required users before starting your migration. What you need to configure depends on your goal:

- **Schema ingestion only** with the EDB Postgres AI agent: Create a machine user in Hybrid Manager (HM), set up a dedicated OS user for the agent on your auxiliary machine, and create a migration user with schema extraction permissions. No WAL-level database changes are required.
- **Data migration** with the Data Migration Service (DMS) agent: Configure WAL settings and grant the required replication privileges.

If you plan to run both schema ingestion and data migration, complete all sections. If you only plan to do schema or data migration, each section explains whether it's applicable for your use case.

Creating the migration user

> **Tip**
>
> Required for both schema ingestion with the EDB Postgres AI agent and data migration with the DMS agent.

Create a single migration user that serves both the EDB Postgres AI agent and the DMS agent. Using one user simplifies credential management across agents.

Using a database admin account, connect to your source Postgres database and create the migration user:

```
CREATE USER <migration_user> WITH REPLICATION LOGIN PASSWORD <migration_user_password>
↪;
```

`REPLICATION` is required for the DMS agent to create replication slots and read from the WAL. If you're performing schema ingestion only, you can omit it — but including it now means you won't need to alter the user later if you add data migration.

Granting schema ingestion permissions

> **Tip**
>
> Required for schema ingestion with the EDB Postgres AI agent.

1. Grant the EDB Postgres AI agent connect access to the source database:

```
GRANT CONNECT ON DATABASE <source_database> TO <migration_user>;
```

Where `<source_database>` is the name of the database you want to assess. If you plan to connect the agent to additional databases on the same server, grant `CONNECT` on each one.

1. Grant the roles needed for a complete schema extraction:

```
GRANT pg_monitor TO <migration_user>;
GRANT pg_read_all_data TO <migration_user>;
GRANT pg_read_all_settings TO <migration_user>;
```

Together, these three roles enable a full schema extraction, including ownership and privilege information. With only `pg_read_all_data` granted, the agent performs a partial extraction limited to schema object definitions, without ownership and privilege information.

`pg_read_all_data` isn't available in Postgres 10 to 13:

```
GRANT pg_monitor TO <migration_user>;
GRANT pg_read_all_settings TO <migration_user>;
```

1. If you're using Postgres 16 or earlier, also grant the following to allow the agent to discover the Postgres log file location:

```
GRANT EXECUTE ON FUNCTION pg_catalog.pg_current_logfile() TO <migration_user>;
```

Granting data migration permissions

> **Tip**
>
> Required for data migration with the DMS agent.

1. Grant the migration user usage on each schema containing tables to migrate:

```
GRANT USAGE ON SCHEMA <db_schema> TO <migration_user>;
```

1. Grant `SELECT` on source tables. For an entire schema:

```
GRANT SELECT ON ALL TABLES IN SCHEMA <db_schema> TO <migration_user>;
```

For individual tables:

```
GRANT SELECT ON <table_name> TO <migration_user>;
```

1. Grant `CREATE` on the source database so the DMS agent can create publications:

```
GRANT CREATE ON DATABASE <pg_db_name> TO <migration_user>;
```

Preparing for schema ingestion (EDB Postgres AI agent)

> **Tip**
>
> Required for schema ingestion with the EDB Postgres AI agent.

Creating a dedicated OS user for the agent

Create a dedicated OS user on your auxiliary machine to run `beacon-agent` . This strategy avoids granting the agent permissions it doesn't need. This is a Linux-specific example, the exact command might vary by distribution:

```
sudo useradd <agent_os_user>
```

Preparing for extension migration (EDB Postgres AI agent)

> **Tip**
>
> Required for schema migration with the EDB Postgres AI agent when your source database has extensions installed.

As part of schema migration, the EDB Postgres AI agent automatically extracts all extensions installed on the source database and applies them to the destination before any other schema DDL runs. If any extension fails to install, the entire schema migration aborts.

Before running schema migration, check which extensions are installed on the source and confirm they're available on the destination at the same version. If an extension isn't available at the matching version, schema migration fails. When migrating across different major Postgres versions, the extension versions available on the destination may not match those on the source, which can cause migration failures.

```
SELECT name, installed_version
FROM pg_available_extensions
WHERE installed_version IS NOT NULL;
```

Run the same query on the destination to compare versions.

After migration completes, if your source uses foreign data wrapper (FDW) extensions, the agent installs the FDW extension itself but doesn't migrate FDW objects. Recreate foreign servers, user mappings, and foreign tables on the destination manually.

For a full list of extension migration limitations, see *Extension migration limitations* .

Preparing for data migration (DMS agent)

> **Tip**
>
> Required for data migration with the DMS agent.

Configuring WAL settings

The DMS agent uses Postgres logical decoding to capture changes from the write-ahead log (WAL). Configure the following parameters in `postgresql.conf`:

1. Set `wal_level` to `logical` to enable logical decoding:

```
wal_level = logical
```

1. Set `max_wal_senders` to at least `1`. If other streaming clients connect to your database, consult your DBA for the appropriate value:

```
max_wal_senders = 1
```

1. Set `max_replication_slots` to at least `1`. This can be higher if your organization uses Postgres replication:

```
max_replication_slots = 1
```

1. Set `max_wal_size` large enough that production traffic generates mostly timed checkpoints rather than requested checkpoints based on WAL size. We recommend at least 8 GB:

```
max_wal_size = 8GB
```

Setting `max_wal_size` too small can allow Postgres to drop WAL entries before the DMS agent can stream them, interfering with or slowing down the migration. Consult your DBA to determine the appropriate size for your environment.

1. Restart the database server to apply the changes.

Creating the replication group role

The migration user must own the source tables to auto-create Postgres publications. Because those tables are already owned by another role, create a replication group role that both the current table owner and the migration user belong to:

```
CREATE ROLE <replication_group>;
GRANT <replication_group> TO <migration_user>;
GRANT <replication_group> TO <original_owner>;
ALTER TABLE <table_name> OWNER TO <replication_group>;
```

Where:

- `<migration_user>` is the Postgres user you created for migration.
- `<original_owner>` is the original owner of the tables.
- `<replication_group>` is the shared role that owns the source tables for publication auto-creation.
- `<table_name>` is the name of a table to migrate (case-sensitive).

You must run `ALTER TABLE ... OWNER TO <replication_group>` for every table you plan to migrate, across all schemas. To transfer ownership of all tables in the database at once:

```
DO $$
DECLARE
  s text;
  t text;
BEGIN
```

(continues on next page)

(continued from previous page)

```
  FOR s, t IN
    SELECT <schemaname>, <tablename> FROM pg_tables
    WHERE <schemaname> NOT IN (pg_catalog, information_schema)
  LOOP
    EXECUTE ALTER TABLE  || quote_ident(s) || . || quote_ident(t) ||  OWNER TO
↪<replication_group>;
  END LOOP;
END;
$$;
```

Configuring SSL

Configure your source database server to accept SSL connections . The DMS agent requires SSL to connect to the source. Create a server certificate and server private key — you can generate the private key with a tool like OpenSSL.

Validating your configuration

The DMS agent includes a validation script that checks whether your source database meets all requirements for CDC migration.

1. Navigate to the DMS agent folder:

```
cd /opt/cdcagent/reader/
```

1. Create an array of all tables to migrate in `<schema_name>.<table_name>` format:

```
arr=(schema1.table1 schema1.table2 schema2.table1)
```

1. Run the validation script:

```
PG_USERNAME=postgres PG_PASSWORD=<password> PG_HOST=<host> PG_PORT=5432 DB_NAME=<db_
↪name> DBZ_USERNAME=<migration_user> DBZ_PASSWORD=<migration_user_password> ./
↪postgresConfigValidation.sh "${arr[@]}"
```

A passing result looks like this:

```
 *** [Transporter] - Get PostgreSQL Major Version
PostgreSQL major version: 17
 *** [Transporter] - Validate WAL Level
wal_level:  logical
[Pass] wal_level is logical.
 *** [Transporter] - Validate max WAL senders
max_wal_senders: 10
[Pass] max_wal_senders is at least 1.
 *** [Transporter] - Validate max replication slots
max_replication_slots: 10
[Pass] max_replication_slots is at least 1.
 *** [Transporter] - Validate max WAL size
max_wal_size: 8 GB
[Pass] max_wal_size is set to the recommended value of 8GB or higher.
 *** [Transporter] - Validate checkpoints
checkpoints_timed: 100
checkpoints_req: 5
```

(continues on next page)

(continued from previous page)

```
[Pass] Timed checkpoints are more frequent than requested checkpoints.
 *** [Transporter] - Check <migration_user> user role
[Pass] User <migration_user> is present
[Pass] User <migration_user> has replication permission
 *** [Transporter] - Check SELECT privilege on the tables to be migrated
[Pass] User <migration_user> has select privilege on all tables to be migrated.
 *** [Transporter] - Check CREATE privilege on database
[Pass] Role <migration_user> has CREATE privilege on database <db_name>.
```

Note

Address any *[Failed]* or *[Suggestion]* statuses before proceeding. *[Failed]* checks are blocking issues you must resolve before the DMS agent can execute the data migration. *[Suggestion]* checks allow migration but can negatively affect performance if left unaddressed.

Next step

Configuring the EDB Postgres AI agent to register your AWS RDS/Aurora PostgreSQL database

Configuring the EDB Postgres AI agent to register your Postgres database

Configure the EDB Postgres AI agent (packaged as `beacon-agent`) to establish a secure connection with HM and retrieve schema and migration assessment data from your Postgres source.

Prerequisites

- **Machine user and access key**: Created in *Creating a machine user* . Store the access key in an environment variable:

```
export BEACON_AGENT_ACCESS_KEY="baak_your_key_here"
```

- **pg_dump**: The EDB Postgres AI agent uses `pg_dump` to extract DDL from your Postgres source. Ensure `pg_dump` is installed and configured on your auxiliary machine. See *Configuring your auxiliary machine or bastion host* for installation instructions, compatibility rules, and credential setup.

Note

The agent substitutes environment variables referenced in *beacon_agent.yaml* at runtime, so setting *$BEACON_AGENT_ACCESS_KEY* in your environment before running the agent is sufficient. Environment variables always take precedence over static values in the YAML.

Recommendations for setting environment variables

We recommend handling sensitive data, such as `$BEACON_AGENT_ACCESS_KEY` or `$DSN` , as environment variables managed through a secrets manager. This strategy prevents sensitive credentials from being stored in plaintext configuration files. If you use environment variables, make sure they're available to the EDB Postgres AI agent while it's running.

Consult your organization's IT department on the safest approach for production environments.

For short-lived migrations or testing and demo purposes, you can use the `export` command in an active terminal session.

Obtaining HM connection parameters

The agent requires specific values from your HM installation. Contact your administrator for:

- **Beacon server hostname**: The internal endpoint for the `beacon_server` .

When configuring the HM installation, administrators must specify an internal endpoint for the `beacon_server` service. Ask the administrator or installer of your HM instance to provide the hostname (URL) they set for the `beacon_server` service in the Helm chart configuration file used for installation. They'll look up the value they set for `parameters.upm-beacon.server_host` (or `BEACON_SERVICE_DOMAIN_NAME`) in the `values.yaml` file. You'll need this value later to configure the EDB Postgres AI agent.

- **Root certificate path (if required)**: If your HM instance uses a self-signed or internal CA, obtain the `ca_public.crt` file. Store it on the agent machine (for example, `/etc/beacon/ca_public.crt`).

The EDB Postgres AI agent requires a trusted connection to the `beacon_server` service in your HM instance. If the server running the EDB Postgres AI agent doesn't inherently trust HM's TLS certificate (managed by your organization's security infrastructure), ask the administrator or installer of your HM instance to provide the file with the certificate. You'll need to provide the directory path to this certificate later to configure the EDB Postgres AI agent.

See *Administrative tasks* for more information about the `beacon_server` and root certificate.

Verifying connectivity

Once you have the HM connection parameters, verify that your auxiliary machine can reach both the Postgres source and HM. The following ports must be open:

Destination	Port	Used by
Postgres host	5432 (default for PostgreSQL, AWS RDS, and Aurora); 5444 (default for EDB Postgres Advanced Server)	Both agents (database connection)
HM *beacon_server*	9443	EDB Postgres AI agent
HM DMS service (*RW_SERVICE_HOST*)	443 (HTTPS)	DMS agent

Note

Ports 5432 and 5444 are defaults. If your Postgres database is configured to listen on a different port, use that port instead and ensure it's open on your network.

Verify connectivity to each host and port with `nc` :

```
#  Postgres source

nc -zv <postgres-host-name-or-ip> 5432

#  HM beacon server

nc -zv <beacon_server_hostname> 9443

#  HM DMS service
```

(continues on next page)

(continued from previous page)

```
nc -zv <rw-service-hostname> 443
```

A successful connection returns `succeeded!` . If you see `Connection refused` or `Operation timed out` , the port isn't reachable and must be opened before proceeding.

Defining source database connection parameters

The data source name (DSN) is the connection string for your Postgres source.

Format:

```
DSN="postgresql://<migration_user>:<migration_user_password>@<host>:<port>/<database_
↪name>"
```

Set the DSN as an environment variable before running the agent or configuring the service:

```
export DSN="postgresql://migration_user:password@postgres-host:5432/mydb"
```

If your database has SSL disabled, append `?sslmode=disable` :

```
export DSN="postgresql://migration_user:password@postgres-host:5432/mydb?
↪sslmode=disable"
```

Use the RDS instance endpoint as the host. RDS enforces SSL by default, so append `?sslmode=require` :

```
export DSN="postgresql://migration_user:password@myinstance.abc123.us-east-1.rds.
↪amazonaws.com:5432/mydb?sslmode=require"
```

Use the Aurora cluster endpoint as the host. Aurora enforces SSL by default, so append `?sslmode=require` :

```
export DSN="postgresql://migration_user:password@my-cluster.cluster-abc123.us-east-1.
↪rds.amazonaws.com:5432/mydb?sslmode=require"
```

```
DSN="postgresql://migration_user:pa%24%24word@postgres-host:5432/mydb"
```

Preparing a configuration file

1. Create an EDB Postgres AI agent configuration directory in your home directory:

```
mkdir ${HOME}/.beacon
```

> **Note**
>
> The agent searches for *beacon_agent.yaml* in */etc/beacon/* or *${HOME}/.beacon/* . As a final fallback, it searches the directory from which it's executed. !!!

1. Inside this directory, create a file `beacon_agent.yaml` :

```
touch ${HOME}/.beacon/beacon_agent.yaml
```

1. Copy and paste the following template into the new file.

In this template, `$BEACON_AGENT_ACCESS_KEY` and `$DSN` are the environment variables you configured previously. Replace all < > placeholders in the template. See *EDB DMS agent parameter reference* for details.

```
---
agent:
  access_key: $BEACON_AGENT_ACCESS_KEY
  beacon_server: <beacon_server_hostname>:9443
  project_id: <your_project_id>
  providers:
    - "onprem"
  # In most HM deployments, set `root_ca_path` to the path of the ca_public.crt
  # file obtained from your HM administrator (for example, /etc/beacon/ca_public.crt).
  # Leave empty only if HMs certificate is already trusted by the systems CA bundle.
  root_ca_path: ""
  schema_providers:
    # Include "onprem-schema" to enable schema ingestion and migration assessment.
    # Remove this section if you dont want schema information to be collected.
    - "onprem-schema"
provider:
  onprem:
    databases:
      # `resource_id` controls how this database appears in HM Estate.
      - resource_id: "<database_resource_id>"
        dsn: $DSN
        # pg_bin_path: /usr/lib/postgresql/16/bin
        # Optional. Absolute path to the directory containing pg_dump and pg_dumpall.
        # Set when registering databases from different Postgres distributions/
↪versions.
        schema:
          enabled: true
          poll_interval: 15s
general:
  logging:
    # `info` is simplified, use debug during initial setup or
    # troubleshooting for more verbose output.
    level: info
```

1. Save the file.

Running the EDB Postgres AI agent

1. Before starting the agent, verify connectivity and configuration syntax:

```
beacon-agent validate
```

1. Run the agent:

```
beacon-agent
```

> **Tip**
>
> See Agent CLI for other agent modes and options. !!!

1. Monitor the ingestion process as the agent starts sending data to HM. After a few minutes, your Postgres database appears in the HM console under **Estate > Migrations**.

Important

The agent dynamically detects changes to tables and columns. However, changes to views, procedures, or functions require an agent restart to re-ingest metadata.

For production use, see *Running the agent as a service* .

Verifying the migration assessment

To confirm your Postgres database was recognized by the agent and is ready for migration, go to **Estate > Migrations** in the HM console.

If the database appears in the table, you have access to the following resources:

- A **migration assessment** on the *Migrations* of the **Estate** page, providing a summarized view of database schema count, core count, migration complexity, and level of effort.
- A *Viewing migration assessment details* accessible by selecting a specific database from the **Migrations** tab. This tab provides an expanded analysis of **Schemas**, a global **Assessment**, and a **Migration** hub with direct connections to other HM tools for performing migrations.
- **Automatic schema DDL extraction** that you can use for schema migrations with the built-in Migration Portal. Once ingestion is complete, the Migration Portal automatically creates a project per database, which you can use to prepare and migrate schemas to the destination database.

Troubleshooting

There are several reasons why the agent might fail to connect to HM or your Postgres database. Here are some tools you can use for troubleshooting.

`beacon-agent validate` command

The `beacon-agent validate` command is a helpful step to identify common connectivity and configuration issues.

```
beacon-agent validate
```

The validate command checks for the following:

- **Server connectivity**: Verifies the agent can connect and authenticate to the configured `beacon_server` (usually on port `9443`).
- **Database connectivity**: Attempts to connect to each configured database DSN. This is a full connection test, not just a syntax or URL-encoding check.
- **Configuration display**: Shows TLS settings, project ID, providers, and per-database settings to help identify misconfiguration.

`beacon-agent` logs

If the agent reports errors during ingestion, you can check the logs for more details. If you're running the agent in the foreground, logs print to the terminal. If you set up the agent as a systemd service, use `journalctl` to view logs.

Here are some common error messages you might encounter in the logs:

Error	Cause	Fix
tls: failed to verify certificate: x509: certificate signed by unknown authority	The agent can't verify HM's TLS certificate because the CA isn't trusted by the system.	Set *root_ca_path* in *beacon_agent.yaml* to the path of the *ca_public.crt* file obtained from your HM administrator. If you can't obtain the certificate, you can set *plaintext: true* in the agent config as a temporary workaround (not recommended for production).
rpc error: code = Unknown desc = no rows in result set	The beacon-agent version is incompatible with the server.	Upgrade to beacon-agent v1.384.0 or later.

Running the agent as a service

For continuous metric streaming and to ensure your Postgres database remains visible in **HM console > Estate**, configure the agent to run as a systemd service. Running as a service ensures it starts automatically on system boot and restarts after any errors.

Without a service, the agent streams metrics only while you run it manually. Your database will disappear from the HM estate 72 hours after the agent stops streaming.

> **Note**
>
> Ensure that any environment variables referenced in *beacon_agent.yaml* (for example, *$BEACON_AGENT_ACCESS_KEY* and *$DSN*) are accessible to the service at runtime. If you configured a *.pgpass* file for schema extraction credentials (as described in *Configuring the environment*), ensure that file is present in the home directory of the user specified in *User=* .

1. Create the service file. This example uses `vi` , but you can use any text editor:

```
sudo vi /usr/lib/systemd/system/beacon-agent.service
```

Default service file paths by operating system:

Operating system	Default path
RHEL	*/usr/lib/systemd/system/beacon-agent.service*
Rocky Linux	*/usr/lib/systemd/system/beacon-agent.service*
AlmaLinux	*/usr/lib/systemd/system/beacon-agent.service*
Ubuntu	*/etc/systemd/system/beacon-agent.service*
Debian	*/lib/systemd/system/beacon-agent.service*

1. Paste the following template into the file and replace all `< >` placeholders:

```
[Unit]
Description=EDB Postgres AI Agent
After=network.target

[Service]
Type=simple
User=<agent_os_user>
WorkingDirectory=</home/agent_os_user/>
ExecStart=</usr/local/bin/beacon-agent>
Restart=on-failure
```

(continues on next page)

(continued from previous page)

```
RestartSec=60

[Install]
WantedBy=multi-user.target
```

Setting	Placeholder	Description
User=	*<agent_os_user>*	OS user that runs the agent. Use the dedicated OS user you created when preparing the source database.
WorkingDirectory=	*</home/agent_os_user/>*	Directory where the *.beacon* folder is stored. This is typically the home directory of *<agent_os_user>*.
ExecStart=	*</usr/local/bin/beacon-agent>*	Full path to the *beacon-agent* binary. The path depends on how and where the agent was installed.

1. Reload systemd, then enable and start the service:

```
sudo systemctl daemon-reload
sudo systemctl enable beacon-agent.service
sudo systemctl start beacon-agent.service
```

1. Confirm the service started successfully by checking the logs:

```
journalctl -u beacon-agent.service
```

Related topics

- For more granular control of schema ingestion (for example, to exclude or include schemas), see *Controlling schema ingestion* .
- For the full set of configuration parameters, see *EDB DMS agent parameter reference* .
- If your administrator needs guidance on obtaining the root certificate or `beacon_server` value, see *Administrative tasks* .

Next step

Preparing your HM-managed destination database

Preparing your HM-managed destination database

Prepare an HM-managed database cluster to serve as the destination for your migration.

If you don't have a destination database cluster on Hybrid Manager (HM) yet, create one:

1. In the HM console, select the "Can't select a project" — the portal domain defaults to the API endpoint where you want to create the database cluster.
2. In your project, select **Create New** and then **Database Cluster** to *Create a database cluster template* .
3. In the **Clusters** page, select your cluster, and use the **Quick Connect** option to access your database from your terminal.
4. Create an empty database to use as a destination for the migration. Alternatively, you can use the default database `edb_admin` .

If you already have an HM-managed database cluster, you can use it as the destination for your migration.

Before proceeding, ensure the tables you migrate and the load generated during migration don't interfere with existing workloads on the cluster.

Next step

Migrating Postgres schemas

Migrating Postgres schemas

Before starting a data migration, choose how to handle schema migration. HM supports two approaches:

- **Simplified (recommended)**: HM manages schema migration and constraint handling automatically as part of the data migration setup.
- **Manual**: Export and import the schema yourself using Migration Toolkit or `pg_dump` , and handle constraints based on the key structure of each source table.

No manual schema preparation is required. HM handles schema migration and constraint ordering automatically when you select the **Schema and data** scope later. Before proceeding, confirm:

- The EDB Postgres AI agent has ingested your source schema and a Migration Portal project exists for the database. To verify, go to **Estate > Migrations** in the HM console, select the database in the **Database Name** column, and confirm a project appears in the **MP Project** column.
- When you configure the DMS reader agent in a later step, you'll set `DBCONFIG_DATABASES_<N>__RESOURCEID` to the same `resource_id` value as in `beacon_agent.yaml` . The **Schema and data** and **Schema only** scopes will fail without it.

Continue to the next step.

Before you use EDB DMS for data migration, export and import your schema to the destination database. DMS migrates data only — the destination database must have schemas in place before migration begins.

Handling constraints

How to handle constraints depends on the constraint type and the key structure of each source table.

Constraint type	Action	Notes
Primary key	Exclude now — DMS applies automatically	The DMS writer applies the primary key when the migration enters the streaming phase, using it to deduplicate snapshot data.
Unique key (source table has a primary key)	Exclude now, apply after streaming starts	Primary key-based deduplication ensures no duplicate rows remain when you apply unique constraints after streaming starts.
Unique key (source table has no primary key)	Not currently supported	Use the simplified path instead — select the **Schema and data** scope when creating the migration to let HM handle schema migration automatically.
NOT NULL	Include now	*NOT NULL* validation adds no meaningful overhead to the data load, unlike foreign key and check constraint validation.
Foreign key, check, and exclusion	Exclude now, reapply after migration	These cause unnecessary CPU and memory overhead while data is actively being migrated.

> **Note**
>
> For tables with no primary key or unique key, no key constraints are needed before or after migration. DMS treats these tables as keyless and provides at-least-once delivery. Deduplicate rows as part of your post-migration verification.

Exporting your schema

We recommend using EDB Migration Toolkit to export your schema. MTK's offline migration capability generates SQL scripts you can apply on the destination without direct connectivity. Other valid export methods include `pg_dump` /`pg_restore` , pgAdmin, or `psql` .

When using Migration Toolkit:

- Use the `-offlineMigration` option to generate SQL scripts.
- Use the `-schemaOnly <schema_scope>` option to export schema only.
- Exclude (comment out) primary key, unique, foreign key, check, and exclusion constraints from the generated script before importing.
- If you plan to map objects to different names in the destination, edit them directly in the DDL script before importing. Update all dependent objects — views, constraints, and other references — that reference renamed objects. See *Mapping schema, tables, and columns* for details.

Importing your schema

After preparing the DDL and excluding the deferred constraints, connect to the destination database and apply the SQL file by executing the generated offline migration script .

> **Note**
>
> Apply the excluded unique, foreign key, check, and exclusion constraints after migration completes. The DMS writer applies primary key constraints automatically — don't reapply them manually. Keep track of excluded constraints so you can reapply them when the time comes.

Next step

Configuring and running the EDB DMS agent

Configuring and running the EDB DMS agent

The EDB DMS agent connects to your Postgres source and works with the Data Migration Service (DMS) to perform data migrations. Before running it, create migration credentials in HM and configure the agent with your source database details.

Prerequisites

- Ensure the machine where you're running the DMS agent has Java/OpenJDK 17 or later installed.
- *Install the EDB DMS agent* .

Creating migration credentials

Migration credentials authenticate the DMS agent to Hybrid Manager (HM). You need one set of credentials per HM project.

1. In the HM console, select the project you're using for your migration.
2. Select **Migrate > Credentials**.
3. Select **Create Migration Credential**, enter a name, and select **Create Migration Credential**. A `credentials.zip` file is downloaded to your machine.

> **Note**
>
> Store the credentials safely. If you lose them, you'll need to revoke this set and create new ones. !!!

1. Unzip the downloaded file and copy the credentials folder to the machine where you installed the DMS agent — for example, to `$HOME/credentials/` . If you use a different path, update `CREDENTIAL_DIRECTORY_PATH` in the script to match.

The folder contains:

- `int.crt`
- `int.truststore.p12`
- `client.keystore.p12`
- `client-key.pem`
- `client-cert.pem`

Configuring the DMS agent

1. Open `/opt/cdcagent/run-cdcagent.sh` with write permissions.
2. Set the required variables. Optional parameters are commented out by default — uncomment and configure any that apply to your environment.

See the *EDB DMS agent parameter reference* for details.

```
#!/bin/bash -e
# run-cdcagent.sh
#
# This script provides a convenient place to specify
# environment variables used to configure the
# EDB Data Migration Service Agent.
#

########################################
# DMS Agent General Configuration      #
########################################

# This ID is used to identify DMS Agent
# and is specified by the user.
export DBCONFIG_ID=<source_name_of_choice>

# Determine the run mode of the DMS Agent
```

(continues on next page)

(continued from previous page)

```
# The run mode can be set to either reader or writer
# reader mode is used to read changes from a source database
# writer mode is used to write changes to a target database
export RUN_MODE=reader

# This is the DMS backend service used by the Agent
# Obtain this value from the HM console Sources or Destinations migration pages
export RW_SERVICE_HOST=<rw_service_host>

########################################
# DMS Agent TLS Configuration          #
########################################

# You need to create migration credentials in EDB Postgres AI platform
# and set the path of the credentials directory
export CREDENTIAL_DIRECTORY_PATH=$HOME/credentials

########################################
# DMS Agent DB Configuration           #
########################################

# A sample configuration to create a single Postgres database connection:
export DBCONFIG_DATABASES_0__TYPE=POSTGRES
export DBCONFIG_DATABASES_0__HOSTNAME=<localhost/host>
export DBCONFIG_DATABASES_0__PORT=<5432>
# The CATALOG is the database name
export DBCONFIG_DATABASES_0__CATALOG=<source>
export DBCONFIG_DATABASES_0__USERNAME=<postgres>
# The password env can be set without specifying it here
# but the env structure looks like this
export DBCONFIG_DATABASES_0__PASSWORD=<password>
# Required for Schema and data or Schema only scope.
# Must match resource_id in beacon_agent.yaml.
#export DBCONFIG_DATABASES_0__RESOURCEID=<resource_id>

# You can increase the index to configure more than
# one database for the DMS Agent
#export DBCONFIG_DATABASES_1__TYPE=POSTGRES
#export DBCONFIG_DATABASES_1__HOSTNAME=<localhost/host>
#export DBCONFIG_DATABASES_1__PORT=<5432>
#export DBCONFIG_DATABASES_1__CATALOG=<source>
#export DBCONFIG_DATABASES_1__USERNAME=<postgres>
#export DBCONFIG_DATABASES_1__PASSWORD=<password>
#export DBCONFIG_DATABASES_1__RESOURCEID=<resource_id>

########################################
# Optional Parameters Below            #
########################################

# FIPS mode can be enabled by setting the FIPS_MODE env variable to true
#export FIPS_MODE=true
```

(continues on next page)

(continued from previous page)

```
# Configure logging
# Loglevel for the entire Quarkus application and underlying components
# WARNING: Enabling this loglevel is extremely verbose
#export QUARKUS_LOG_LEVEL=DEBUG
# Loglevel for the EDB package
# Preferred mode for debugging issues related to the logic of the agent
#export QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL=DEBUG

cd "$(dirname "$0")"

echo "Starting in mode: $RUN_MODE"

case "$RUN_MODE" in
  reader)
    java -jar ./reader/quarkus-run.jar
    ;;
  writer)
    java -jar ./writer/quarkus-run.jar
    ;;
  *)
    echo "Unknown RUN_MODE: $RUN_MODE"
    exit 1
    ;;
esac
```

Running the DMS agent

Navigate to the DMS agent directory and run the configuration script:

```
cd /opt/cdcagent
./run-cdcagent.sh
```

The script exports all required environment variables and starts the agent process. Once running, the agent registers your Postgres database as a source in HM. Continue to the next section to verify it appears correctly.

Verifying the source in HM

Once the agent is running, verify that your Postgres database appears as a source in HM:

1. In the HM console, select your project.
2. Select **Migrate > Sources**.

Your database is listed with its name, type, agent ID, and status. For a description of all fields on this page, see *Understanding deployment resources* .

Next step

Migrating data from an AWS RDS or Aurora PostgreSQL database to an HM-managed database

Migrating data from a self-managed Postgres database to an HM-managed database

With the EDB DMS agent running and your schema either imported or ready for HM to import, you're ready to create a migration in the HM console. This page covers the full migration lifecycle: creating the migration, monitoring its progress, finalizing it, and applying post-migration constraints.

The steps below vary depending on the approach you chose on the previous page. Select the tab that matches your choice.

Creating a migration

A migration establishes a sync between your Postgres source and a destination database cluster managed by HM.

1. In the HM console, select the project from which you want to manage the migration.
2. In the project, select **Migrate > Migrations**.
3. In the **Migrations** page, select **Create New Migration**.
4. In the **Create OLTP Migration** page, assign a name to the migration.
5. Select the source of the migration. The ID for the DMS agent is listed in the **Source** menu.
6. Under **Scope**, select the option that matches your approach:

Select **Schema and data**. When you select this option, HM:

1. Sources the schema from the Migration Portal project that was created automatically when you registered your database with the EDB Postgres AI agent.
2. Creates tables on the destination without constraints, so the initial snapshot copy isn't slowed by constraint validation.
3. After the snapshot completes, applies primary key and unique constraints (for tables that have them) so DMS can deliver streaming changes exactly-once. Tables without a PK or UK fall back to at-least-once delivery and require deduplication during verification.
4. Defers the remaining schema (indexes, foreign keys, check constraints, `NOT NULL` , and similar) until cutover. You can optionally apply most of this work concurrently with streaming via **Finalize Schema Concurrently** to shrink the cutover window — see *Finalizing a streaming migration* .
5. After you mark the migration as completed, validates deferred foreign keys, syncs sequences, and applies any remaining lightweight tasks in a final **Schema Finalization** stage.

> **Note**
>
> Use **Schema only** if you want to migrate the database structure without migrating any table data. !!!

Select **Data only**. The schema was already imported manually in the previous step.

1. Select the snapshot type: **Snapshot** or **Snapshot + Streaming** .
2. Under **Destination Type**, select an HM-managed destination database cluster and database for the migration. Then, select **Next**.
3. On the **Review your Tables** tab, select the tables and columns to include in the migration. Expand a table to view its columns and optionally *map source columns to different destination column names* . Column mapping is only available for the **Data only** scope.
4. Select **Create Migration**.

The HM console displays the *new migration* . If you chose **Snapshot + Streaming**, the DMS agent streams data changes continuously while the migration shows a **Running** state.

> **Note**
>
> The DMS agent streams data changes only — not DDL changes (schema modifications).

Monitoring progress

Migration list view

The **Migrations** page shows a summary of all migrations in the project. The **Migration progress** column displays three metrics per migration:

- **Total number of tables** — total tables included in this migration
- **Snapshot tables completed** — tables that have finished the snapshot stage
- **Migrated rows** — cumulative rows migrated across all tables

Migration details view

Select a migration from the list to open the details page, which has three tabs:

- **Overview** — shows the current migration state and general information about the source and destination databases. For **Snapshot + Streaming** migrations, the migration is caught up when the status is **Streaming**. For **Snapshot** migrations, it's finished when the status is **Completed**.
- **Tables** — shows all tables selected for migration. For each table, you can see source and destination table and schema names, snapshot status, mapped columns, migrated rows, and migration rate. Select the + icon to expand a table and view column-level mappings.
- **Error history** — lists all errors logged for this migration. Select the **i** icon to open the full error log for a specific error. For stuck or failed migrations, active errors here indicate where to start troubleshooting.

DMS agent logs

The HM console doesn't surface all DMS agent errors. If a migration is stuck or no tables are progressing, check the terminal where you ran `run-cdcagent.sh` for `ERROR` log entries — these entries often contain the root cause. Common issues include version incompatibility between the DMS agent and HM and connectivity or credential failures.

Finalizing a streaming migration

> **Note**
>
> This section applies only to **Snapshot + Streaming** migrations. For **Snapshot** migrations, DMS marks the migration as completed automatically.

Monitoring catch-up before cutover

Before stopping writes on the source, give the destination time to catch up. Use the **Migration details** page to track row counts and replication lag. Begin the cutover only after the destination is consistently close to the source.

Finalizing schema concurrently (recommended)

If you used the **Schema and data** scope, HM can apply most schema objects while the stream is still running: primary keys, unique constraints, indexes (built with `CREATE INDEX CONCURRENTLY`, so live writes aren't blocked), foreign keys (added as `NOT VALID`), check constraints, `NOT NULL` constraints, and similar. Foreign key validation, sequence synchronization, and other lightweight tasks run later in the post-cutover **Schema Finalization** stage.

Running this work concurrently shrinks the read-only cutover window. The larger your tables and the more indexes they have, the greater the time saved.

Once every table has completed its snapshot and the migration enters the streaming phase, the **Migration details** page shows a **Schema Finalization Available** alert with a **Finalize Schema Concurrently** button. The button stays available throughout streaming, so you can pick the moment based on how close the destination is to catching up.

1. Select **Finalize Schema Concurrently** on the **Migration details** page.
2. HM applies the deferred objects while streaming continues. The migration remains in the **Running** state, with a progress sub-step labeled **Finalizing Schema Concurrently**.

When the sub-step completes, proceed to the cutover.

> **Note**
>
> **Finalize Schema Concurrently** is available only for migrations created with the **Schema and data** scope with streaming enabled.

If you skip this step, the same objects are applied serially after you mark the migration as completed. See *Schema finalization stage*. The cutover window is larger in that case because indexes are built non-concurrently and constraints are validated under exclusive locks.

Not applicable to the manual path. You applied the schema before migration and will reapply deferred constraints manually after cutover. See *Applying deferred constraints and verifying data*.

Performing the cutover

When you're ready to switch traffic from the source to the destination:

1. Stop all writes on the source database.
2. Verify in the HM console that the destination has applied the last source change.
3. On the **Migration details** page, select **Mark as Completed**.

This stops the streaming process.

Schema finalization stage

For migrations created with the **Schema and data** scope, the migration enters a **Schema Finalization** stage automatically after **Mark as Completed**. What runs in this stage depends on whether you ran concurrent finalization first:

- **If you ran concurrent finalization**: HM runs the lightweight remaining work — foreign key validation, sequence sync, and any other deferred metadata tasks. This stage is short.
- **If you skipped concurrent finalization**: HM applies the full deferred set now (primary keys, unique constraints, indexes, foreign keys, check constraints, `NOT NULL`, and similar) followed by the lightweight tasks above. This stage takes substantially longer and runs while the source is read-only.

The migration is fully complete only after the **Schema Finalization** stage finishes. Progress is shown on the **Migration details** page.

No automatic schema finalization runs for the **Data only** scope. Continue to *Applying deferred constraints and verifying data* to reapply the foreign key, unique, check, and exclusion constraints you deferred before migration.

Applying deferred constraints and verifying data

Reapplying deferred constraints

HM applied all deferred constraints and synced sequences automatically during the **Schema Finalization** stage. No manual action is needed. See *Schema finalization stage* .

At the beginning of your migration, you prepared and imported the schema excluding foreign key, unique, check, and exclusion constraints. Connect to the destination database and reapply those constraints now.

Primary key constraints were applied automatically by the DMS writer when the migration entered the streaming phase. `NOT NULL` constraints were applied before migration and don't require any action now.

Deduplicating rows when needed

For tables that had no primary key or unique constraints, the EDB DMS agent provides at-least-once delivery, which can result in duplicate rows. Perform deduplication as part of your verification step.

Verifying migrated data

Compare the source and destination databases to confirm all data was migrated successfully. You can use LiveCompare for this.

Related topics

- LiveCompare — row-level data comparison tool for verifying migration completeness
- *Terminology* — definitions for snapshot, streaming, and other migration concepts

References

DMS agent version compatibility

The DMS agent (`cdcagent`) is versioned and released independently of Hybrid Manager (HM). Using an incompatible version can cause silent failures — the HM console recognizes the database correctly, but issues appear only when a migration starts.

The table shows the minimum `cdcagent` version required for each HM release. Using a newer version than the minimum is supported and recommended.

HM version	Minimum *cdcagent* version
1.4.0 (LTS)	1.13.12
2026.5 (IR)	1.12.40
2026.4 (IR)	1.12.8
2026.3 (IR)	1.10.21
2026.2 (IR)	1.9.4
1.3.x (LTS)	1.3.23

To install or upgrade the DMS agent, see Installing the DMS agent .

EDB DMS agent parameter reference

DMS agent general configuration

These parameters define the information the EDB DMS agent requires to:

- Access the Hybrid Manager (HM) instance and project that you're using for the migration
- Connect to the source database and collect data or connect to the destination database and migrate data

> **Note**
>
> A DMS Agent instance can only run in either reader or writer mode.

DBCONFIG_ID

The name you assign to identify a source/destination. This name will later appear either as a source in the **Migrate** section in the **Sources** tab of the HM console or as a destination in the **Destinations** tab.

For the ID:

- The maximum character length is 255 characters.
- You can use lowercase and uppercase characters, numbers, underscores(_) and hyphens(-). Other special characters aren't supported.
- The ID must be unique in a project. Don't use the same ID for two or more sources/destinations in the same project.

RUN_MODE

Determines how the EDB DMS agent runs. Set it to `reader` if you're using the DMS agent to connect HM with a migration source. Set it to `writer` if you're connecting HM with an external, self-managed migration destination.

Run the DMS agent in writer mode only if the destination for your migration is a self-managed Postgres database.

RW_SERVICE_HOST

Specifies the URL of the DMS that the EDB DMS agent uses to perform migrations. This value is deployment-scoped — the same URL applies across all projects within a given HM deployment. Obtain this value from the HM console **Sources** or **Destinations** migration pages.

To obtain your `RW_SERVICE_HOST` :

1. In the HM console, select the project you want to use for migrations.
2. In the left navigation select **Migrate** and then select **Sources** or **Destinations**.

Both **Sources** and **Destinations** pages display the `RW_SERVICE_HOST` in a banner at the top of the page.

1. Use the copy icon to get the value and use it for the configuration of your EDB DMS agent.

CREDENTIAL_DIRECTORY_PATH

The directory path of the credentials folder you *Configuring and running the EDB DMS agent* .

DMS agent database configuration

These parameters (`DBCONFIG_DATABASES` section) define a list of source/destination database information the EDB DMS agent requires to connect to the correct database for the migration.

You can configure the EDB DMS agent to migrate multiple databases. The `DBCONFIG_DATABASES_0__TYPE` section delimits the information for the first database. You can use `DBCONFIG_DATABASES_1__TYPE` to provide data for a second database. Add more sections to the EDB DMS agent (`DBCONFIG_DATABASES_2__TYPE` , `DBCONFIG_DATABASES_3__TYPE`) by increasing the index manually.

DBCONFIG_DATABASES_0__TYPE

The source/destination database type.

The EDB DMS agent supports `ORACLE` and `POSTGRES` as source database types (in reader mode), but only `POSTGRES` as a destination type (in writer mode).

DBCONFIG_DATABASES_0__HOSTNAME

The hostname of the source/destination database.

> **Note**
>
> For RDS databases, ensure the *DBCONFIG_DATABASES_0__HOSTNAME* value points to the RDS endpoint you can find under **Connectivity & security** in the AWS console. A redacted example is *foo_database.cb7asw1xxxxx.eu-west-0.rds.amazonaws.com* .

DBCONFIG_DATABASES_0__PORT

The port of the source/destination database.

DBCONFIG_DATABASES_0__CATALOG

The database name in the source/destination database server.

For Oracle sources:

- **CDB/PDB**: use `CDB_NAME/PDB_NAME` (for example, `ORCLCDB/ORCLPDB1`). The agent splits on `/` to identify the CDB and PDB separately. Using only the PDB name causes it to be treated as the CDB, leaving the PDB unset.
- **Non-CDB**: use the SID or service name (for example, `ORCL`)

DBCONFIG_DATABASES_0__USERNAME

The database username of the source/destination database. Use your dedicated migration user, it can be the same user you created for the EDB Postgres AI agent.

> **Note**
>
> For Oracle sources, use your dedicated migration user — not *SYS* or other privileged system accounts. See *Preparing Oracle source database* for instructions on creating the required migration user.

DBCONFIG_DATABASES_0__PASSWORD

The password for the database username of the source/destination database. You can set an environment variable for the password without specifying it in the script, but the variable must follow this format: `export DBCONFIG_DATABASES_0__PASSWORD=password`

DBCONFIG_DATABASES_0__RESOURCEID

User-defined identifier that associates this database with its registered counterpart in HM.

If you previously registered this database with the EDB Postgres AI agent, this value **must match** the `resource_id` set for this database in `beacon_agent.yaml` . This is how HM correlates the DMS agent source with the registered database — a mismatch will cause them to appear as separate, unrelated entries.

DBCONFIG_DATABASES_0__INCLUDELIST

Comma-separated list of `schema.table` tokens that controls which tables are available for migration. Use `*` as a wildcard for schema or table name (for example, `SCHEMA1.*` or `*.TABLE1`).

If absent or empty, all tables are included. If `EXCLUDELIST` is also set, exclusions take priority.

DBCONFIG_DATABASES_0__EXCLUDELIST

Comma-separated list of `schema.table` tokens for tables to always exclude from migration. Use `*` as a wildcard. Takes priority over `INCLUDELIST` — a table matching any exclude pattern is always excluded regardless of include rules.

DBCONFIG_DATABASES_0__SSLMODE

Controls the SSL/TLS behavior for the database connection. Accepted values:

- `STRICT` — verify-full; requires a valid, trusted certificate
- `DEFAULT` — require encryption but don't verify the certificate
- `WEAK` — prefer encryption but allow plaintext fallback

DBCONFIG_DATABASES_0__SOCKETTIMEOUT

Socket timeout in seconds for the database connection. Used for dead connection detection. Default: `600` for writer mode, `60` for reader mode.

DBCONFIG_DATABASES_0__TCPKEEPALIVE

Enables TCP keepalive probes on the database connection. Default: `true` .

DBCONFIG_DATABASES_0__TCPKEEPALIVESIDLE

Seconds of inactivity before the first TCP keepalive probe is sent. Used during long-running writer operations such as PK application and post-snapshot deduplication. Default: `60` .

DBCONFIG_DATABASES_0__TCPKEEPALIVESINTERVAL

Seconds between TCP keepalive retry probes. Default: `10` .

DBCONFIG_DATABASES_0__TCPKEEPALIVESCOUNT

Maximum number of failed TCP keepalive probes before the connection is considered dead. Default: `6`.

DBCONFIG_DATABASES_0__JDBCOPTIONS

Arbitrary JDBC driver properties to pass to the database connection, in `key=value;key=value` format. These are applied after defaults but before explicit settings like `SSLMODE` and `SOCKETTIMEOUT`.

Oracle examples:

- Disable network encryption negotiation: `oracle.net.encryption_client=REJECTED`
- Set session program name: `v$session.program=DMSAgent`

Optional parameters

FIPS_MODE

If you need to comply with FIPS standards, enable FIPS mode for the DMS Agent by setting the environment variable to true: `export FIPS_MODE=true`.

TRANSFORMATION_ENABLE_NULLCHAR_REPLACEMENT

Oracle-specific. Enables replacement of null characters (`0x00`) in text-based column types: `VARCHAR2`, `LONG`, `CLOB`, and `NCLOB`. Default: `false`.

Use together with `TRANSFORMATION_NULLCHAR_REPLACEMENT` to specify the replacement string.

TRANSFORMATION_NULLCHAR_REPLACEMENT

Oracle-specific. The string used to replace null characters (`0x00`) when `TRANSFORMATION_ENABLE_NULLCHAR_REPLACEMENT` is enabled. Defaults to an empty string.

TRANSFORMATION_ENABLE_SPECIAL_COLUMN_NAME_ENCODING

Oracle-specific. Enables encoding of special characters (`$` and `-`) in column names. Default: `false`.

DBZ_*

Debezium property passthrough. You can override any Debezium connector property by prefixing it with `DBZ_`. For example, to set `max.batch.size`, use `export DBZ_MAX_BATCH_SIZE=204800`.

For a full list of supported properties, see the Debezium documentation:

- Oracle connector properties
-

PostgreSQL connector properties

QUARKUS_LOG_LEVEL

Sets the default log level for the entire Quarkus application, including all underlying frameworks and third-party libraries.

Avoid setting this to `DEBUG` or `TRACE` unless you need to diagnose issues within the entire application ecosystem. Using this method produces logs that are excessively verbose and difficult to analyze. Use `QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL` instead for focused debugging.

QUARKUS_LOG_CATEGORYCOM_ENTERPRISEDBLEVEL

Sets the log level specifically for classes within the `com.enterprisedb` package.

This is the recommended parameter for debugging. Set this parameter to `DEBUG` or `TRACE` to obtain detailed logging for the DMS agent's core logic (the components relevant to schema extraction, migration assessment, etc.). This isolates the useful output and excludes messages from third-party libraries.

Installing EDB Data Migration Service agent on Linux

> **Note**
>
> Install and configure EDB DMS Agent with Hybrid Manager versions 1.3 and later. For Hybrid Manager 1.2 and earlier versions, use EDB DMS Reader and EDB DMS Writer.

Select a link to access the applicable installation instructions:

Linux x86-64 (amd64)

Red Hat Enterprise Linux (RHEL) and derivatives

- *Installing EDB Data Migration Service agent on RHEL 9 or OL 9 x86_64* , *Installing EDB Data Migration Service agent on RHEL 8 or OL 8 x86_64*
- *Installing EDB Data Migration Service agent on RHEL 9 or OL 9 x86_64* , *Installing EDB Data Migration Service agent on RHEL 8 or OL 8 x86_64*
- *Installing EDB Data Migration Service agent on AlmaLinux 9 or Rocky Linux* , *Installing EDB Data Migration Service agent on AlmaLinux 8 or Rocky Linux*
- *Installing EDB Data Migration Service agent on AlmaLinux 9 or Rocky Linux* , *Installing EDB Data Migration Service agent on AlmaLinux 8 or Rocky Linux*

SUSE Linux Enterprise (SLES)

- *Installing EDB Data Migration Service agent on SLES 15 x86_64*

Debian and derivatives

- *Installing EDB Data Migration Service agent on Ubuntu 24.04 x86_64* , *Installing EDB Data Migration Service agent on Ubuntu 22.04 x86_64*
- *Installing EDB Data Migration Service agent on Debian 12 arm64* , *Installing EDB Data Migration Service agent on Debian 11 x86_64*

Linux Installing EDB Data Migration Service agent on Linux AArch64 (ARM64)

Debian and derivatives

- *Installing EDB Data Migration Service agent on Debian 12 arm64*

Installing EDB Data Migration Service agent on Linux x86 (amd64)

Operating system-specific install instructions are described in the corresponding documentation:

Red Hat Enterprise Linux (RHEL) and derivatives

- *Installing EDB Data Migration Service agent on RHEL 9 or OL 9 x86_64*
- *Installing EDB Data Migration Service agent on RHEL 8 or OL 8 x86_64*
- *Installing EDB Data Migration Service agent on RHEL 9 or OL 9 x86_64*
- *Installing EDB Data Migration Service agent on RHEL 8 or OL 8 x86_64*
- *Installing EDB Data Migration Service agent on AlmaLinux 9 or Rocky Linux*
- *Installing EDB Data Migration Service agent on AlmaLinux 8 or Rocky Linux*
- *Installing EDB Data Migration Service agent on AlmaLinux 9 or Rocky Linux*
- *Installing EDB Data Migration Service agent on AlmaLinux 8 or Rocky Linux*

SUSE Linux Enterprise (SLES)

- *Installing EDB Data Migration Service agent on SLES 15 x86_64*

Debian and derivatives

- *Installing EDB Data Migration Service agent on Ubuntu 24.04 x86_64*
- *Installing EDB Data Migration Service agent on Ubuntu 22.04 x86_64*
- *Installing EDB Data Migration Service agent on Debian 12 arm64*
- *Installing EDB Data Migration Service agent on Debian 11 x86_64*

Installing EDB Data Migration Service agent on Debian 11 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

Setting up the repository is a one-time task. If you already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter:

```
apt-cache search enterprisedb
```

If no output is generated, the repository is installed.

To set up the EDB repository:

1. Go to EDB repositories .
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

Install the EDB DMS agent (packaged as `cdcagent`):

```
sudo apt-get install cdcagent
```

Installing EDB Data Migration Service agent on Debian 12 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

Setting up the repository is a one-time task. If you already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter:

```
apt-cache search enterprisedb
```

If no output is generated, the repository is installed.

To set up the EDB repository:

1. Go to EDB repositories .
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

Install the EDB DMS agent (packaged as `cdcagent`):

```
sudo apt-get install cdcagent
```

Installing EDB Data Migration Service agent on AlmaLinux 8 or Rocky Linux

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

Setting up the repository is a one-time task. If you already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter:

```
dnf repolist | grep enterprisedb
```

If no output is generated, the repository is installed.

To set up the EDB repository:

1. Go to EDB repositories .
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

Install the EDB DMS agent (packaged as `cdcagent`):

```
sudo dnf install cdcagent
```

Installing EDB Data Migration Service agent on AlmaLinux 9 or Rocky Linux

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

Setting up the repository is a one-time task. If you already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter:

```
dnf repolist | grep enterprisedb
```

If no output is generated, the repository is installed.

To set up the EDB repository:

1. Go to EDB repositories .
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

Install the EDB DMS agent (packaged as `cdcagent`):

```
sudo dnf install cdcagent
```

Installing EDB Data Migration Service agent on RHEL 8 or OL 8 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

Setting up the repository is a one-time task. If you already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter:

```
dnf repolist | grep enterprisedb
```

If no output is generated, the repository is installed.

To set up the EDB repository:

1. Go to EDB repositories .
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

Install the EDB DMS agent (packaged as `cdcagent`):

```
sudo dnf install cdcagent
```

Installing EDB Data Migration Service agent on RHEL 9 or OL 9 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

Setting up the repository is a one-time task. If you already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter:

```
dnf repolist | grep enterprisedb
```

If no output is generated, the repository is installed.

To set up the EDB repository:

1. Go to EDB repositories .
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

Install the EDB DMS agent (packaged as `cdcagent`):

```
sudo dnf install cdcagent
```

Installing EDB Data Migration Service agent on SLES 15 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

Setting up the repository is a one-time task. If you already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter:

```
zypper lr -E | grep enterprisedb
```

If no output is generated, the repository is installed.

To set up the EDB repository:

1. Go to EDB repositories .
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

- Activate the required SUSE module:

```
sudo SUSEConnect -p PackageHub/15.7/x86_64
```

- Refresh the metadata:

```
sudo zypper refresh
```

Install the package

Install the EDB DMS agent (packaged as `cdcagent`):

```
sudo zypper install cdcagent
```

Installing EDB Data Migration Service agent on Ubuntu 22.04 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

Setting up the repository is a one-time task. If you already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter:

```
apt-cache search enterprisedb
```

If no output is generated, the repository is installed.

To set up the EDB repository:

1. Go to EDB repositories .
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

Install the EDB DMS agent (packaged as `cdcagent`):

```
sudo apt-get install cdcagent
```

Installing EDB Data Migration Service agent on Ubuntu 24.04 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

Setting up the repository is a one-time task. If you already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter:

```
apt-cache search enterprisedb
```

If no output is generated, the repository is installed.

To set up the EDB repository:

1. Go to EDB repositories .
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

Install the EDB DMS agent (packaged as `cdcagent`):

```
sudo apt-get install cdcagent
```

Installing EDB Data Migration Service agent on Linux AArch64 (ARM64)

Operating system-specific install instructions are described in the corresponding documentation:

Debian and derivatives

- *Installing EDB Data Migration Service agent on Debian 12 arm64*

Installing EDB Data Migration Service agent on Debian 12 arm64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

Setting up the repository is a one-time task. If you already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter:

```
apt-cache search enterprisedb
```

If no output is generated, the repository is installed.

To set up the EDB repository:

1. Go to EDB repositories .
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

Install the EDB DMS agent (packaged as `cdcagent`):

```
sudo apt-get install cdcagent
```

Using the HM console to view migration assessments

One of the main uses of Hybrid Manager (HM) is that it helps with planning and performing database migrations.

You can register an external Oracle or Postgres database with the EDB Postgres AI agent (for example, *Configuring the EDB Postgres AI agent to register your Oracle database*). The **Migrations** tab that you can access from the **Estate** page then lists the external databases you registered along with a migration assessment that you can use for planning.

This is your first stop in the HM console when migrating external databases to HM.

Migrations table

Default columns

The default view of the table contains the following information for each analyzed database:

Database Name: Name of the assessed database.

Database Type: Database type. See *Schema compatibility summary* for currently supported database types.

Resource ID: Resource ID or `databases.resource_id` you assigned to a database in the `beacon_agent.yaml` configuration. See the EDB Postgres AI agent's *EDB DMS agent parameter reference* for more information.

Version: Database version. See *Schema compatibility summary* for currently supported versions.

 # or Cores: Number of cores allocated to the database.

# of Schemas: Number of schemas in the assessed database that the DSN user has access to. Contains at least one table on which the DSN user has `SELECT` privileges. The DSN user is the source database user you specified while configuring the EDB Postgres AI agent.

- In Oracle databases, built-in `SYS` and `SYSTEM` schemas are excluded from the column count.
- The DSN user's schema is usually excluded if it has no tables. However, a known issue can cause it to be included when the DDL extractor creates temporary tables during extraction.
- The EDB Postgres AI agent detects schema changes to tables and columns while running. However, changes to views or procedures aren't detected until you restart the EDB Postgres AI agent.

Engine: This column applies to Postgres databases only and indicates the database server distribution, such as PostgreSQL, EDB Postgres Advanced Server, or EDB Postgres Extended Server.

Complexity: This value is generated only for Oracle schemas and provides the level of complexity to migrate the database to HM. Possible values are:

- **Very Easy:** The Oracle schema doesn't contain incompatible features or objects, and the data model is fully compatible with EDB Postgres Advanced Server.
- **Easy:** The migration is relatively straightforward. The database structure and features are compatible and require minimal adjustment.
- **Medium:** Requires moderate migration effort. Some database structures or features may require modification to get similar functionality in EDB Postgres Advanced Server.
- **Hard:** Requires significant effort and expertise. The database structures, features, or custom code in Oracle have significant differences or complexities that you need to address during the migration.
- **Very hard:** The migration is highly complex and demanding. Substantial differences in structures, features, and custom code between the Oracle and EDB Postgres Advanced Server databases require expertise so data, objects, and statements can be transformed. The code might have to be rewritten, or architectural redesign may be required.

Migration LoE: This value is generated only for Oracle schemas and represents the level of effort, in 8-hour working days, required to convert the schema and migrate data to HM. The estimation corresponds to the effort applied by members of EDB's professional services who specialize in Oracle-to-Postgres migrations. This value considers database and schema-level assessments that include database features, schema complexity, data volume, and other factors to estimate the number of days.

Status: Current state of the migration. You can manually update the status from the database's *Viewing migration assessment details* to reflect its progress.

Other columns

You can use the gear icon to display additional columns:

Last assessed: Period of time that's transpired since the database was last assessed to determine if the analysis is up to date.

Management: Shows whether the database is managed by you (self-managed), a cloud service provider, or HM.

Project Name: Project that was used for configuring the EDB Postgres AI agent. It's the project you assign to an agent in `agent.project_id`. See the EDB Postgres AI agent *EDB DMS agent parameter reference* for more information.

Tags: Labels assigned to the migration of a specific database in `databases.tags`. See the EDB Postgres AI agent *EDB DMS agent parameter reference* for more information.

TDS: Total database size in GB. This includes database data only. Plan some additional space in the migration destination databases for metaobjects, temp and log files, and so on.

Viewing and filtering the migrations table view

From the **Estate** page, the **Migration** tab displays a table-based analysis of all databases that you can migrate with HM after registering it with the EDB Postgres AI agent. To customize how assessment results are displayed or filter databases, you can:

- Enter a keyword to search for specific databases. The **Search** function uses the keyword to display databases with matching **Resource ID**, **Project Name**, and **Database Type** values.
- Use the **Filter** selector to display databases by Postgres version and project name.
- Order the results by newest to oldest, oldest to newest, and alphabetically.
- Use the lightning icon to select how often to refresh the table. If the icon isn't enabled, the table isn't updated unless you manually refresh it. If **Auto refresh** is enabled, the table updates every 30 seconds. Alternatively, you can select one of the preconfigured intervals.
- Customize the columns displayed in the **Migrations** tab using the gear icon. You can also reset the columns to the default state.
- Adjust the size of the table to your screen.

Related topics

- See *Schema compatibility summary* to understand which databases are supported for migration assessment.
- See *Data security and sovereignty* to understand how your data is stored by the EDB Postgres AI agent.

Migration assessment support

After registering a self-managed database with the EDB Postgres AI agent, HM generates a migration assessment. Assessments are currently supported for the following database versions.

> **Note**
>
> Currently, the Postgres assessments offer a high-level migration overview. A more detailed analysis is available for Oracle databases.

Oracle

- Oracle 11g
- Oracle 12c
- Oracle 18c
- Oracle 19c

Later versions (for example, 21c, 23c) might also be assessed, but they aren't yet fully certified.

Postgres

- PostgreSQL 13-18
- EDB Postgres Advanced Server 13-18
- EDB Postgres Extended Server 13-18

Viewing migration assessment details

When you select a database from the **Migrations** tab of the **Estate** page in Hybrid Manager (HM), a detailed view opens. Use this view to plan and execute migrations from external databases. The **Schemas** and **Assessment** tabs display different information depending on whether your source database is Postgres or Oracle.

Status

At the top of the database details page, a status selector appears next to the database name. Use it to update the current state of a database migration and track your migration workflow, providing clarity for you and your team members.

You can select from the following statuses:

- Assessing (default)
- Planning
- Decided Not To Migrate
- In Progress
- Done

Schemas

The **Schemas** tab offers an overview of the schemas found in your connected database. It displays summary metrics for the database, along with a list of schemas where you granted `SELECT` access on at least one table. Only schemas containing at least one table are displayed.

This information can help you scope your migration effort and effectively plan the migration of your schemas.

Important

The EDB Postgres AI agent detects schema changes to tables and columns while running. However, changes to views or procedures aren't detected until you restart the EDB Postgres AI agent.

Summary cards

- Total schemas for the selected database.
- Total tables for the selected database.
- Total objects for the selected database.

Schema table

- **Name:** Name of the schema.
- **# of Tables:** Number of tables in the schema.
- **# of Objects:** Number of objects in the schema.
- **MP Project:** Link to the Migration Portal project associated with this schema. Migration Portal projects for Postgres databases are created automatically by the EDB Postgres AI agent.

Selecting a row expands it to show the following:

Schema Compatibility Summary: Shows the compatibility status of schema objects as reported by the Migration Portal, including the count of failed, system-repaired, user-repaired, and passed objects, along with the total object count and passed percentage.

Migration Portal project table: Lists the Migration Portal project associated with the schema, including the project name, target EDB Postgres Advanced Server (EPAS) version, creation date, and available actions.

Summary cards

- Total schemas for the selected database.
- Total tables for the selected database.
- Total objects for the selected database.
- Complexity overview for all schemas, color-coded by complexity (Very Easy, Easy, Medium, Hard, Very Hard).
- Migration level of effort, color-coded by day ranges (0–5, 6–15, 16–25, 26–40, 40+ days), defined by 8-hour working days.

Schema table

- **Name:** Name of the schema.
- **Latest Assessment:** Status of the latest assessment. Possible values are Not Started, In Progress, Complete, and Error.
- **Last Assessed:** Timestamp of the last assessment. The assessment is regenerated or updated if the EDB Postgres AI agent discovers a new schema or an update to an existing one.
- **# of Tables:** Number of tables in the schema.
- **# of Objects:** Number of objects in the schema.

- **Complexity:** Indicates how challenging it will be to convert the schema to EDB Postgres Advanced Server, based on a detailed, object-level analysis. Possible values are Very Easy, Easy, Medium, Hard, and Very Hard. A single schema's complexity may be higher than the overall database complexity, as schema-level analysis is more granular.
- **Migration LoE:** Indicates how many days it will take to migrate the schema, in 8-hour working days, based on a detailed, object-level analysis.
- **Dependencies:** The names of any schemas the current schema depends on. Migrate schemas with their dependencies in the same batch (using the same Migration Portal project) to ensure that all referenced objects exist in the destination database.
- **MP Project:** Link to the Migration Portal project associated with this schema. To create a new project, select **+ New MP Project** above the table and provide the required details. HM creates the project in the background using the Migration Portal Project-scoped workspace vs. user's workspace . You can access it later from the ***Migration*** tab.

Selecting a row expands it to show the following:

Schema Compatibility Summary: Shows the compatibility status of schema objects as reported by the Migration Portal, including the count of failed, system-repaired, user-repaired, and passed objects, along with the total object count and passed percentage.

Migration Portal project table: Lists the Migration Portal project associated with the schema, including the project name, target EDB Postgres Advanced Server (EPAS) version, creation date, and available actions. You can delete a project from the Actions column.

Assessment

The **Assessment** tab provides a migration assessment of your connected database, offering insights to help you scope the effort and plan the migration of your database objects. This section contains a summary of the assessment, including key metrics and detailed breakdowns of the factors that contribute to the generated analysis.

To export the assessment results in `.json` formatting, select **Download Report**.

Assessment Details: Shows a high-level migration assessment summary of the database and the key information used in the assessment. For Postgres databases, this includes database name, engine, version, size, object count, table count, and schema count. For Oracle databases, it also includes migration complexity, Oracle database features used, Oracle PL/SQL features used, database migration LoE, number of CPUs, number of schemas assessed, assessment type, and last assessment timestamp.

Extensions Installed: Lists the extensions, including foreign database wrappers, that are installed in the selected database. Check the EDB Postgres extensions documentation to determine which extensions are supported for each EDB supported database prior to migrating your database.

Foreign Servers: Lists the foreign servers created in the selected database and the corresponding Foreign Data Wrapper (FDW) used to configure them. Before migrating, verify that each FDW is compatible with your destination database and that the destination can reach the foreign server.

Foreign Tables: Lists the count of foreign tables in schemas included in the assessment for the selected database, grouped by schema and foreign server.

Tablespaces: Lists the tablespaces in the selected database, including tablespace name, owner, and location.

Tablespace Schema Usage Summary: Lists the non-built-in Postgres tablespaces (excluding `pg_default` and `pg_global`) in the selected database, with the count of table, index, or materialized view objects created in those tablespaces per schema included in the assessment.

Schemas Without Full Replica Identity: Lists the count of tables in each schema included in the assessment that don't have a primary key, a unique key, or replica identity set to `FULL` . This condition impacts the ability to perform change data capture (CDC) streaming data migration.

Installed Procedural Languages: Lists the procedural languages installed in the selected database. If these languages are needed to support application requirements, check the EDB Postgres extensions documentation to determine which languages are supported for use in EDB supported databases prior to migrating your database. Also, if any of the installed languages are untrusted languages, determine if their use complies with security policies and is supported.

Compatibility Summary: Shows a high-level compatibility summary of database features and schema objects, along with other factors that contribute to the overall migration complexity and level of effort assessment. Metrics include total schemas, total assessed PL/SQL objects, compatible objects, incompatible objects and occurrences with known workarounds, incompatible objects and occurrences that need further analysis, Oracle database features used, database size, PL/SQL lines, large objects usage, partition types used, aggregate queries, and SQL count.

Oracle PL/SQL Features Used: Lists all PL/SQL objects that contain Oracle built-in features. For each feature, the table shows the object count, occurrence count, whether it's EPAS compatible (Yes/No), and migration complexity.

Oracle Database Features Used: Lists all Oracle database features in use in the database, including their migration complexity and migration level of effort (LoE) in days.

Incompatible PL/SQL Features in Objects: Lists all incompatible PL/SQL features found in database objects. For each entry, the table shows the schema name, object type (function, package, and so on), object name, the incompatible PL/SQL feature, number of occurrences, and migration complexity.

Migration

The **Migration** tab helps you perform migration-related tasks for the selected database.

Migration Workflows

Create a Destination: Opens the cluster creation workflow, where you can set up a new database cluster to serve as the destination for your migration. See *Creating a database cluster* for more guidance.

Migrate Schema(s): Defines a migration that moves both your database schemas and data to your target cluster. Select **Migrate Schema(s)** to start.

Migrate Data: Transfers your complete data set to a new cluster while maintaining data integrity and minimizing downtime. Select **Migrate Data** to start. See *Choose your migration path* for end-to-end examples.

Active Migrations

Lists the migrations associated with the selected database. Select **View All Migrations** to see all migrations across your estate.

The table includes the following columns:

- **Name:** Name of the migration, linked to the migration details page.
- **Source:** The external source database being migrated.
- **Destination:** The target database and cluster.
- **Snapshot Type:** The type of snapshot used. Possible values are **Snapshot** and **Snapshot + Streaming**.
- **Scope:** The scope of the migration. Possible values are **Schema Only**, **Data Only**, and **Schema + Data**.
- **Migration Progress:** Shows three metrics for the migration: total tables, snapshot tables completed, and migrated rows. Not shown for schema-only migrations.
- **Status:** Current status of the migration.

Assessing multiple databases at once

When you have a large database estate — dozens of Oracle or Postgres databases — manually constructing a DSN string and adding a `beacon_agent.yaml` entry for each one is time-consuming and error-prone. The EDB Postgres AI agent's `configure-onprem-dbs` command solves this problem by reading your existing database inventory from a file (or discovering databases directly from AWS) and automatically populating `beacon_agent.yaml` with the correct DSN entries.

Once you start the agent with the populated configuration, the agent registers all enrolled databases with HM and generates migration assessments for each one. Bulk enrollment with `configure-onprem-dbs` is the recommended starting point when you want an estate-wide migration readiness report before committing to individual migration projects.

> **Note**
>
> *configure-onprem-dbs* only populates the *provider.onprem.databases* section of *beacon_agent.yaml* with DSN entries. It doesn't generate the full agent configuration, start the agent, or connect to HM.

Supported input sources

The `configure-onprem-dbs` command can populate your configuration from three sources:

- CSV file
- Oracle `tnsnames.ora` file
- AWS RDS discovery

Full command reference

For all available flags — including RDS-specific options such as `--region` , `--rds-engine` , and `--tag-filter` — see Populating DSN entries using `configure-onprem-dbs <Populating DSN entries using *configure-onprem-dbs*>` in the EDB Postgres AI agent CLI reference.

Related topics

- See *Migration assessment support* for the Oracle and Postgres versions supported for migration assessment.
- To get started, see *Enrolling multiple databases for assessment* .

Enrolling multiple databases for assessment

Set up and run the EDB Postgres AI agent to enroll multiple databases for migration assessment in a single workflow.

Prerequisites

These prerequisites apply regardless of your source database type:

- **EDB Postgres AI agent installed** on an auxiliary machine or bastion host. How you configure your auxiliary machine depends on your source database type — see *Configuring your auxiliary machine or bastion host* , *Configuring your auxiliary machine or bastion host* , and *Configuring your auxiliary machine or bastion host* .
- **Machine user with access key**. You need an HM machine user with the `estate ingester` role and a valid access key. See *Creating a machine user* .

The following requirements must be completed **for each database individually** before it can be enrolled. The agent installation and machine user are set up once and reused across all databases.

- **Oracle sources** — Each source database requires a dedicated migration user with the appropriate read privileges. See *Preparing Oracle source database* .
- **Postgres sources** — Each source database requires a user with sufficient read access. See *Configuring the EDB Postgres AI agent to register your AWS RDS/Aurora PostgreSQL database* (self-managed Postgres) or *Configuring the EDB Postgres AI agent to register your AWS RDS/Aurora PostgreSQL database* (AWS RDS/Aurora-managed PostgreSQL).

> **Note**
>
> A database will appear in HM under **Estate > Migrations** as long as the agent can connect to it. Without the full privilege set, it will show no schema count, no complexity score, and no assessment data. !!!

Preparing `beacon_agent.yaml`

How you handle this step depends on whether you already have a configuration file:

You already have a ``beacon_agent.yaml`` — skip this step. In the next step, point `--output-yaml` at your existing file and `configure-onprem-dbs` will merge the new database entries into it, leaving all other settings untouched.

You don't have a ``beacon_agent.yaml`` yet — you have two options:

- Set up the full configuration file first (HM connection parameters, project ID, and so on), leaving the `provider.onprem.databases` section empty. Then use `configure-onprem-dbs` in the next step to populate it. Follow the configuration guide for your source database type to create the file:
- **Oracle** — See *Configuring the EDB Postgres AI agent* .
- **Self-managed Postgres** — See *Configuring the EDB Postgres AI agent to register your AWS RDS/Aurora PostgreSQL database* .
- **AWS RDS / Aurora** — See *Configuring the EDB Postgres AI agent to register your AWS RDS/Aurora PostgreSQL database* .
- Run `configure-onprem-dbs` first and add the agent configuration afterward. The command creates a minimal file containing only the `provider.onprem.databases` section with your DSN entries. You then need to add the HM connection parameters and other settings manually, following the same guides above.

Populating DSN entries using `configure-onprem-dbs`

Run `configure-onprem-dbs` to read your database inventory and append the `provider.onprem.databases` entries to the configuration file you created. Choose the example that matches your input source.

During the prompt, you can enter credentials for each database individually, skip an entry to leave placeholders, apply the same credentials to all remaining entries, or skip all remaining entries at once. Any entries still using placeholder credentials after the run are listed in a warning — replace them in the output file before starting the agent.

A CSV file lets you enroll databases from any source — exported from Oracle Enterprise Manager, another Configuration Management Database (CMDB) or asset management tool, or assembled manually. The file must follow a specific format. See CSV file format in the EDB Postgres AI agent CLI reference.

Prepare a CSV file with your database inventory. For example:

```
#  Database inventory

resource_id,db_type,host,port,db_name,username,password,schema_enabled,poll_interval,
↪tags
ORACLE_PROD_01,oracle,oracle-prod.example.com,1521,PRODPDB,<username>,<password>,true,
```

(continues on next page)

(continued from previous page)

```
↪30s,production|critical
ORACLE_STAGING_01,oracle,oracle-staging.example.com,1521,STAGEPDB,<username>,
↪<password>,true,1m,staging
ORACLE_DEV_01,oracle,oracle-dev.example.com,1521,DEVPDB,<username>,<password>,false,
↪5m,development
PG_PROD_01,postgresql,postgres-prod.example.com,5432,appdb,<username>,<password>,true,
↪1m,production|postgres
```

Then run:

```
beacon-agent configure-onprem-dbs \
  --source-type csv \
  --input databases.csv \
  --output-yaml ${HOME}/.beacon/beacon_agent.yaml \
  --interactive
```

Use `--interactive` to be prompted for credentials for any entries that have placeholder values.

From an Oracle `tnsnames.ora` file

If you use Oracle Net Services, you can pass a `tnsnames.ora` file directly. The command parses each entry and extracts the host, port, and service name or SID to construct the DSN. If your Oracle environment uses `tnsnames.ora` , pass it directly. Since `tnsnames.ora` doesn't include credentials, use `--interactive` to supply them during the run:

```
beacon-agent configure-onprem-dbs \
  --source-type tns \
  --input /etc/oracle/network/admin/tnsnames.ora \
  --output-yaml ${HOME}/.beacon/beacon_agent.yaml \
  --interactive
```

The command parses each entry and constructs the DSN from the host, port, and service name or SID. For example, given this entry:

```
PROD_DB =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = oracle-prod.example.com)(PORT = 1521))
    (CONNECT_DATA =
      (SERVICE_NAME = PRODPDB)
    )
  )
```

The agent generates: `oracle://username:password@oracle-prod.example.com:1521/PRODPDB`

From AWS RDS discovery

If your databases run on AWS RDS or Aurora, the command can discover them automatically without a local file. It uses the standard AWS credential chain (environment variables, `~/.aws/credentials` , or an IAM role) and scans one or all regions. You can filter by engine type or by resource tags.

Supported AWS RDS engines: `postgres` , `aurora-postgresql` , `oracle-ee` , `oracle-ee-cdb` , `oracle-se2` , `oracle-se2-cdb` .

To discover all supported AWS RDS and Aurora instances in a specific region:

```
beacon-agent configure-onprem-dbs \
  --source-type rds \
  --region us-east-1 \
  --output-yaml ${HOME}/.beacon/beacon_agent.yaml \
  --interactive
```

To narrow discovery to specific engine types and filter by resource tags:

```
beacon-agent configure-onprem-dbs \
  --source-type rds \
  --region us-east-1 \
  --rds-engine oracle-ee \
  --rds-engine oracle-se2 \
  --tag-filter Environment=Production \
  --output-yaml ${HOME}/.beacon/beacon_agent.yaml \
  --interactive
```

Reviewing and starting the agent

Before starting the agent, check the output file for any entries still using placeholder credentials:

```
grep -n "CHANGEME" ${HOME}/.beacon/beacon_agent.yaml
```

If placeholders remain, you have two options for replacing them:

- **All databases share the same credentials** — use a `sed` command to replace all placeholders in one step:

```
sed -i "s/CHANGEME_USERNAME:CHANGEME_PASSWORD/<username>:<password>/g" ${HOME}/.
↪beacon/beacon_agent.yaml
```

If your password is stored in an environment variable, you can reference it directly:

```
sed -i "s/CHANGEME_USERNAME:CHANGEME_PASSWORD/<username>:$MY_DB_PASSWORD/g" ${HOME}/.
↪beacon/beacon_agent.yaml
```

- **Each database has different credentials** — edit the output file manually and replace each placeholder with the correct credentials, or re-run `configure-onprem-dbs` with `--interactive` to step through each database individually.

Replace any remaining placeholders with real credentials, then follow the guide for your source database type to validate your configuration and start the agent:

- **Oracle** — See *Running the EDB Postgres AI agent* .
- **Self-managed Postgres** — See *Running the EDB Postgres AI agent* .
- **AWS RDS / Aurora** — See *Running the EDB Postgres AI agent* .

Using the HM console to assess your applications

The Hybrid Manager (HM) console includes the capability to assess SQL statements of your applications. This assessment allows you to evaluate the compatibility of Oracle application SQL statements with EDB Postgres Advanced Server (EPAS) in the scope of Oracle database migrations.

While HM's migration tools focus on database schemas and data, this feature addresses the application side of the migration. It provides a structured way to manage the relationship between your application code and the databases they

inhabit, ensuring that SQL queries executed by your application will function correctly in an EDB Oracle-compatible environment.

Overview

Database migration is rarely limited to the database itself, it's an application migration. Even if a schema is successfully migrated, the SQL queries embedded within application code (Java, .NET, etc.) or external scripts might contain Oracle-specific syntax that fails on the target database.

By identifying these incompatibilities early, you can:

- **Reduce risk:** Discover code-level issues before the application testing phase.
- **Accurate planning:** Estimate the effort required for application refactoring.
- **Automate transformation:** Leverage EDB's transformation engine to automatically convert complex Oracle SQL to EPAS-compatible syntax.

Application records in HM

In HM, you can *Creating an application record* . The record acts as a logical container that stores metadata about the software using your databases and holds the SQL artifacts (trace files, scripts, or extracted queries) that need assessment.

Key capabilities include:

- **Many-to-many mapping:** Associate one application with multiple schemas/databases, reflecting real-world microservices or monolithic architectures.
- **SQL transformation:** Automatically converts Oracle DML, DCL, TCL, and DDL into EPAS-compatible versions.

Application assessment

HM performs application assessments automatically after you create an application record and upload the SQL artifacts.

1. **Application record creation with metadata definition:** You define the application's purpose, its tech stack, and the databases it accesses.
2. **SQL ingestion:** You upload SQL artifacts (Oracle trace `.trc` , EDB Query Extractor files, or plain SQL). See *Preparing SQL files for assessment* for instructions on generating each file type.
3. **Assessment:** The system uses the EDB PLTE engine to parse queries, and identify incompatibilities. You can then *Reviewing application assessment results* .

Supported import formats

HM supports three primary methods for gathering SQL statements:

File Type	Description	Extension
Oracle trace	Standard files generated by the Oracle trace utility.	*.trc*
EDB Query Extractor	SQL statements extracted from Oracle Unified Auditing using EDB scripts.	*.out*
Plain SQL	Manual scripts or utility files containing standard SQL statements.	*.sql*

For step-by-step instructions on preparing each file type, see *Preparing SQL files for assessment* .

Preparing SQL files for assessment

Hybrid Manager (HM) accepts three types of SQL input files for application assessment. Each captures your application's SQL differently, and the best choice depends on your application's architecture and the Oracle features it uses.

File type	Best for	Extension
Generating Oracle trace files	Applications where you can enable Oracle SQL tracing against a test or nonproduction database. Provides the most complete picture of runtime SQL, including bind variable values.	*.trc*
Using the EDB Query Extractor	Applications where Oracle tracing isn't available but Unified Auditing is enabled. Captures SQL persistently across restarts.	*.out*
Preparing plain SQL files	Existing SQL scripts — initialization scripts, maintenance jobs, or SQL extracted from source code — that don't require any Oracle-side configuration.	*.sql*

Known limitations

- **SQL*Plus commands (plain SQL files only):** HM recognizes the following SQL*Plus commands:
- `EXIT`
- `SHOW ERRORS` (`SHOW ERR`)
- `START`
- `WHENEVER`
- `EXEC`
- `DEFINE`
- `SET`
- `COLUMN`
- `VARIABLE`

HM doesn't support any other SQL*Plus commands when processing plain SQL files — they might cause parsing errors or be silently skipped. HM doesn't validate supported commands against an actual database, since there's no active session to resolve them against.

- **No schema context:** HM doesn't currently use schema context to validate transformed SQL statements. The **Schema Name** column in the application's **Overview** tab appears empty for all file types. A future release will resolve this for trace and EDB Query Extractor files. For plain SQL files, it remains permanent, because files don't link to a live Oracle session.
- **MP Project column:** The **MP Project** column in the application's **Overview** tab appears empty for all file types. A future release will resolve this.
- **Procedure and function queries (EDB Query Extractor files only):** EDB Query Extractor captures queries executed from within stored procedures or functions as separate individual statements in the assessment results.

Related topics

- *Creating an application record* — Set up an application record and upload SQL files for assessment.
- *Reviewing application assessment results* — Interpret the compatibility report and plan remediation.
- Application assessment overview — Understand how HM assesses application SQL and the supported file formats.

Generating Oracle trace files

Oracle SQL trace files record all SQL statements executed by a session or user, including bind variable values and wait events. They give Hybrid Manager (HM) the most complete picture of your application's runtime SQL.

Prerequisites

- You have DBA-level access to the Oracle database.
- Your Oracle diagnostic destination (`diagnostic_dest`) has sufficient disk space. Tracing can generate large volumes of data quickly, especially for busy applications.

Connecting to the database as SYSDBA

Connect to the Pluggable Database (PDB) where your application data resides:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<PDB_NAME> as sysdba
```

For example:

```
sqlplus sys/password123@//localhost:1521/ORCLPDB1 as sysdba
```

Connect to the database instance using the service name or SID:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<SID_OR_SERVICE> as sysdba
```

For example:

```
sqlplus sys/password123@//localhost:1521/ORCL as sysdba
```

Creating a logon trigger

A logon trigger automatically starts tracing every time a specific database user connects. Replace `YOUR_APP_USER` with the Oracle schema the application connects as.

```
CREATE OR REPLACE TRIGGER trace_user_logon
AFTER LOGON ON DATABASE
BEGIN
    IF (USER = YOUR_APP_USER) THEN
        DBMS_SESSION.SET_IDENTIFIER(YOUR_APP_USER);
        EXECUTE IMMEDIATE ALTER SESSION SET TRACEFILE_IDENTIFIER = APP_TRACE;
        DBMS_SESSION.SESSION_TRACE_ENABLE(waits => TRUE, binds => TRUE);
    END IF;
END;
/
```

To trace multiple users, use an `IN` list:

```
CREATE OR REPLACE TRIGGER trace_user_logon
AFTER LOGON ON DATABASE
BEGIN
    IF (USER IN (APP_USER_1, APP_USER_2)) THEN
        EXECUTE IMMEDIATE ALTER SESSION SET TRACEFILE_IDENTIFIER =  || USER || _TRACE;
        DBMS_SESSION.SESSION_TRACE_ENABLE(waits => TRUE, binds => TRUE);
    END IF;
END;
/
```

Verifying tracing is active

Before running the full application, confirm the trigger is working. Connect as `YOUR_APP_USER` and run:

```
SELECT sysdate FROM dual;
```

Then check the trace directory for a new or updated `.trc` file. If one appears, tracing is working correctly.

Running the application

With tracing confirmed, run your application against the Oracle database. To get representative coverage, run the full test suite or exercise the key workflows the application uses in production.

Collecting the trace files

To find the trace directory, run:

```
SELECT value FROM v$diag_info WHERE name = Diag Trace;
```

This returns the full path to Oracle's trace directory, which follows the structure:

```
<ORACLE_BASE>/diag/rdbms/<DB_NAME>/<INSTANCE_NAME>/trace/
```

For example: `/u01/app/oracle/diag/rdbms/orcl/orcl/trace/`

You'll find the trace files from your application run in that directory, named after the instance and your `TRACEFILE_IDENTIFIER` — for example, `orcl_ora_12345_APP_TRACE.trc` . Collect those files and upload them to HM.

> **Note**
>
> The logon trigger stays active after this step. Drop it when you no longer need to capture traces, to avoid accumulating trace files on future connections:

```
DROP TRIGGER trace_user_logon;
```

- *Preparing SQL files for assessment* — Overview of all supported SQL file types.
- *Using the EDB Query Extractor* — Alternative capture method using Oracle Unified Auditing.
- *Preparing plain SQL files* — Upload existing SQL scripts without Oracle-side configuration.
- *Creating an application record* — Upload your trace files and start the assessment.

Using the EDB Query Extractor

EDB Query Extractor captures SQL executed by your application using Oracle Unified Auditing, which provides persistent storage of audit records — unlike `V$SQL` , which is in-memory and can be cleared between restarts. The extractor exports the captured SQL to a `.out` file compatible with Hybrid Manager (HM)'s assessment engine.

> **Note**
>
> EDB Query Extractor is currently available on request. Contact EDB Support to obtain the script. A self-serve download will be included in a future HM release.

Prerequisites

- **Oracle Unified Auditing is enabled on the database.** In Oracle 12c and later, Unified Auditing is available by default in mixed mode. To verify it's active on your database, run:

```
SELECT VALUE FROM V$OPTION WHERE PARAMETER = Unified Auditing;
```

A result of `TRUE` means pure unified auditing is enabled. A result of `FALSE` means mixed mode is active, which still supports Unified Auditing and is sufficient for this workflow. If Unified Auditing isn't available, refer to the Oracle documentation on enabling Unified Auditing .

- **You (or a DBA) have the ``AUDIT SYSTEM`` privilege.** You need this privilege to create and enable the audit policy in the steps below. The SYSDBA user has it by default. If you're not connecting as SYSDBA, ask a DBA to either grant you the `AUDIT SYSTEM` privilege or run the policy setup on your behalf.

Connecting to the database

Connect to the Pluggable Database (PDB) where your application data resides:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<PDB_NAME> as sysdba
```

For example:

```
sqlplus sys/password123@//localhost:1521/ORCLPDB1 as sysdba
```

Connect to the database instance using the service name or SID:

```
sqlplus sys/<PASSWD>@//<HOST>:<PORT>/<SID_OR_SERVICE> as sysdba
```

For example:

```
sqlplus sys/password123@//localhost:1521/ORCL as sysdba
```

Setting up the audit policy

1. Create an audit policy that captures all SQL actions:

```
CREATE AUDIT POLICY trace_style_policy
    ACTIONS ALL;
```

1. Enable the policy for your application's database user. Replace `YOUR_APP_USER` with the Oracle schema the application connects as. Enabling the policy for a specific user (rather than for all users) is strongly recommended to limit performance impact and output volume.

```
AUDIT POLICY trace_style_policy BY YOUR_APP_USER;
```

Confirming the audit policy is working

Before running your application, confirm that Oracle is recording SQL for the audited user.

1. Connect to the database as `YOUR_APP_USER` :

```
sqlplus YOUR_APP_USER/<PASSWD>@//<HOST>:<PORT>/<PDB_NAME>
```

```
sqlplus YOUR_APP_USER/<PASSWD>@//<HOST>:<PORT>/<SID_OR_SERVICE>
```

1. Run a pair of test statements to generate audit records:

```
CREATE TABLE junk(a int);
DROP TABLE junk;
```

1. Reconnect as SYSDBA and query `UNIFIED_AUDIT_TRAIL` to confirm the records were captured:

```
SELECT SQL_TEXT, ACTION_NAME, STATEMENT_ID, SESSIONID, EVENT_TIMESTAMP
FROM UNIFIED_AUDIT_TRAIL
WHERE DBUSERNAME = UPPER(YOUR_APP_USER)
  AND (OBJECT_SCHEMA IS NULL
       OR OBJECT_SCHEMA NOT IN (SYS, SYSTEM, AUDSYS, USERENV, GSMADMIN_INTERNAL))
  AND SQL_TEXT IS NOT NULL
  AND SESSIONID IS NOT NULL
  AND STATEMENT_ID IS NOT NULL;
```

If the `CREATE TABLE` and `DROP TABLE` statements appear in the results, the policy is working correctly.

Granting EDB Query Extractor privileges

Before running EDB Query Extractor, grant the audited user the privileges it needs to read the audit trail:

```
GRANT AUDIT_VIEWER TO YOUR_APP_USER;
GRANT SELECT ON UNIFIED_AUDIT_TRAIL TO YOUR_APP_USER;
GRANT SELECT ON sys.v_$instance TO YOUR_APP_USER;
```

Running the application

With the audit policy confirmed, run your application or test suite. Oracle will record all SQL statements executed by the audited user into `UNIFIED_AUDIT_TRAIL` .

Extracting the SQL

Run the EDB Query Extractor script (`edb_queries_extractor.sql`) as `YOUR_APP_USER` to export the captured SQL from `UNIFIED_AUDIT_TRAIL` to a `.out` file.

```
sqlplus YOUR_APP_USER/<PASSWD>@//<HOST>:<PORT>/<PDB_NAME>
```

```
sqlplus YOUR_APP_USER/<PASSWD>@//<HOST>:<PORT>/<SID_OR_SERVICE>
```

Then run the script:

```
@/path/to/edb_queries_extractor.sql
```

The script prompts for the following information:

- **Output file location** — The directory where the `.out` file will be written. Defaults to the current directory.
- **Username** — The Oracle user whose audit records to export. Defaults to the current user.
- **Audit policies** — A comma-separated list of unified audit policy names to filter by. Leave blank to include all policies.
- **Client program or identifier** — The client program name or client identifier set in your application's source code (for example, `MyPythonApp`). Leave blank to include all.
- **Start timestamp** — The start of the capture window, in `DD-MM-YYYY HH24:MI:SS` format. Defaults to `01-01-1970`.
- **End timestamp** — The end of the capture window, in `DD-MM-YYYY HH24:MI:SS` format. Defaults to the current time.

The script writes the output to a file named `USERNAME_YYMMDDHHMMSS.out` in the specified directory — for example, `/tmp/YOUR_APP_USER_260414133704.out`.

Disabling the audit policy

After extraction, disable the audit policy to stop capturing new records:

```
NOAUDIT POLICY trace_style_policy BY YOUR_APP_USER;
```

Upload the generated `.out` file to HM for assessment.

Related topics

- *Preparing SQL files for assessment* — Overview of all supported SQL file types.
- *Generating Oracle trace files* — Alternative capture method using Oracle SQL tracing.
- *Preparing plain SQL files* — Upload existing SQL scripts without Oracle-side configuration.
- *Creating an application record* — Upload your extractor output and start the assessment.

Preparing plain SQL files

If you already have SQL scripts — for example, scripts used for database initialization, maintenance jobs, or SQL embedded in source code files — you can upload them directly as plain `.sql` files without any Oracle-side configuration.

Format requirements

Plain SQL files must follow these formatting rules for Hybrid Manager (HM) to parse them correctly:

- Every SQL statement must end with a semicolon (`;`).
- Every PL/SQL block (anonymous blocks, stored procedures, and triggers) must end with a forward slash (`/`) on its own line, preceded by a blank line.
- Files must use UTF-8 encoding.

Example:

```
SELECT employee_id, first_name FROM employees WHERE department_id = 10;

BEGIN
    UPDATE employees SET salary = salary * 1.1 WHERE department_id = 10;
    COMMIT;
END;

/
```

Related topics

- *Preparing SQL files for assessment* — Overview of all supported SQL file types.
- *Generating Oracle trace files* — Capture runtime SQL using Oracle SQL tracing.
- *Using the EDB Query Extractor* — Capture runtime SQL using Oracle Unified Auditing.
- *Creating an application record* — Upload your SQL files and start the assessment.

Creating an application record

An application record is how Hybrid Manager (HM) represents a software application. It maps the application to the source databases and schemas it depends on, and holds the SQL files you upload to assess their compatibility with EDB Postgres Advanced Server (EPAS). Creating an application record is the first step in the application assessment workflow.

Prerequisites

Before creating an application record, ensure the following:

- You have access to the HM project where you want to create the application.
- Your user role includes the `migration_applications` permission for that HM project.
- You have your SQL artifacts ready to upload. See *Supported import formats* for the accepted file types. If you need to generate them first, see *Preparing SQL files for assessment* .

Creating an application record

The HM console guides you through defining the application's general metadata and associating it with your migration databases.

1. In HM, open your project.
2. In the left navigation, go to **Migrate** > **Applications**.
3. Select +**New Application**.
4. Enter the application's metadata:

Field	Required	Description
Name	Yes	A descriptive name for the application. This name appears in the **Applications** table and on the details page.
Description	No	A short explanation of what the application does or its business purpose.

continues on next page

Table 47 – continued from previous page

Field	Required	Description
Target database version	Yes	The EPAS version your SQL will be assessed against, ensuring the correct compatibility rules are applied. Becomes the default for every file you upload to this application, so you don't have to select it again at upload time.
Programming language	Yes	The primary language or framework the application uses to interact with the database, for example, Java, Go (sqlx), or Python.

1. If you want to link an application to source databases, optionally continue to the **Next: Databases (Optional)** step.
2. Optionally, select the checkbox next to each database you want to include. You can select multiple databases across pages.

Note

Linking a source database is optional. If you don't link one, HM assumes Oracle 19c as the source database version. This can affect assessment accuracy if your SQL contains Oracle 21c-specific syntax. To register an Oracle database first, see *Configuring the EDB Postgres AI agent to register your Oracle database* . .. Note:: 1. Select **Create application**.

HM creates the application record and takes you to your application details page.

Uploading SQL files

After creating the application record, upload your SQL artifacts for assessment.

1. On your application details page, go to the **Overview** tab.
2. In the **SQL File Assessments** section, select **Upload New File(s)**.
3. Drag and drop your files into the upload area, or select them using the file picker. You can select up to 20 files in a single upload operation. Individual files must be 10 MB or smaller. If a file exceeds the limit, it's rejected at upload with an error message.
4. Review the list of files. Any pre-validation errors appear here.
5. Select **Next: Databases** to continue.
6. For each uploaded file, configure the following settings:

Field	Required	Description
Source database	No	The migration database the file's SQL was extracted from. Linking a file to a source database lets HM map queries to specific schemas for more precise assessment.
Target database version	Yes	The EPAS version to assess against, for example, EPAS 18. This action prefills the target version for all assessed databases at the application record level.

continues on next page

Table 48 – continued from previous page

Field	Required	Description
Oracle User UIDs	No	One or more Oracle user IDs whose objects appear in the SQL file. Used to scope the assessment to the relevant schema owners.

1. Select **Confirm** to upload the files and queue them for assessment.

HM automatically runs an assessment after you upload the files. See *Reviewing application assessment results* to understand how to interpret the compatibility report, identify incompatible queries, and plan remediation before migrating your application.

Related topics

- *Preparing SQL files for assessment* — Generate Oracle trace files, EDB Query Extractor output, or prepare plain SQL files for upload.
- *Reviewing application assessment results* — Interpret the assessment output, explore incompatible queries, and plan remediation before application migration.
- Application assessment overview — Understand how HM assesses application SQL and the supported file formats.
- *Migrations* — Review schema- and database-level assessments alongside application assessments.

Reviewing application assessment results

After you *Creating an application record* , Hybrid Manager (HM) automatically runs an assessment and produces a compatibility report. This page explains how to read those results and use them to remediate incompatibilities before you migrate your application.

Opening the SQL Assessment tab

1. In HM, go to **Migrate** > **Applications**.
2. Select the application you want to review.
3. On your application details page, select the **SQL Assessment** tab.

The **SQL Assessment** tab aggregates findings across all SQL files uploaded to the application.

Reviewing assessment details

The **Assessment Details** section at the top of the **SQL Assessment** tab gives you a feature-compatibility summary and key metadata about the assessment run.

Feature compatibility cards

Compatibility cards show how Oracle-specific features in your SQL files break down by compatibility status:

Card	Description
Compatible features	Features that work on EDB Postgres Advanced Server (EPAS) without any changes.
Features with available workaround	Features that aren't natively compatible but have a known, established workaround you can apply.

continues on next page

Table 49 – continued from previous page

Card	Description
Features with new workaround	Features for which a new workaround has been identified, which may require additional review before adopting.
Features with no workaround	Features that can't be automatically transformed and have no known workaround. These items require manual remediation.

Assessment metadata

Below the cards, a metadata strip shows context for the assessment run:

Field	Description
Application name	The name of the application record.
Source database(s)	The migration databases linked to this application.
Target database version(s)	The EPAS version the assessment was run against.
Total input files provided	The number of SQL files uploaded to this application.
Total queries parsed	The number of SQL statements successfully parsed across all files.
Total queries with suggested repairs	The number of queries for which HM produced a suggested transformation.
Last assessment timestamp	When the most recent assessment completed.

Downloading the assessment report

To share or archive the full assessment, select **Download Report**. The report contains a complete summary of all findings across files and is useful for offline review, team handoffs, or migration planning documentation.

Downloading transformed queries

Select **Download Transformed Queries** to export all transformed queries for the application as a single archive. You can use this output to:

- Inspect how HM adapted specific queries for EPAS.
- Feed transformed statements into a test environment before completing the migration.
- Seed a review process with your development team.

Reviewing incompatible features

The **Incompatible Features** table identifies Oracle-specific features found in your SQL files that aren't directly compatible with EPAS. Use this table to understand categories of issues across your codebase before drilling into individual queries.

Column	Description
Feature	The Oracle-specific syntax, function, or construct that's incompatible.
File	The SQL file where this feature appears.
Oracle User UIDs	The Oracle schema owner(s) associated with this feature, if provided at upload.
Workaround Status	The current remediation state for this feature.

continues on next page

Table 51 – continued from previous page

Column	Description
Queries	The number of queries in the file that use this feature.
Occurrences	The total number of times this feature appears across all queries in the file.
Query ID	A reference to the specific query where the feature is found.

Reviewing incompatible queries

The **Incompatible Queries** table gives you a query-by-query breakdown of SQL statements that need changes before your application can run against EPAS. This table is your primary tool for understanding the scope of application-side remediation work.

Each row represents a single SQL statement with one or more incompatibilities. Queries listed here won't run correctly as-is and require action before application cutover.

Column	Description
Query ID	A stable identifier for the query in the assessment results.
File	The SQL file that contains this query.
Oracle User UIDs	The Oracle schema owner(s) associated with this query, if provided at upload.
Query	A preview of the SQL statement.
# of Incompatible Features	The number of distinct incompatible features detected in this query.
Occurrences	How many times this query appears across the uploaded files.

Expanding a query to see transformations and repair guidance

If HM was able to repair the query, use the + sign for that query row to expand it. The expanded panel shows:

- **Transformed query** — HM's EPAS-compatible version of the original SQL. This output is the suggested replacement you can use to update your application code.
- **Repair handlers** — details about the specific changes made during transformation, including:
- What Oracle construct was changed.
- Which transformation rules were applied.
- Any remaining steps that require manual attention.

Not every incompatible query can be transformed automatically. Where HM can't produce a transformation, the repair handler notes describe what needs to be changed, so you can make the fix manually in your application code.

Fixing incompatibilities before application migration

After reviewing the assessment results, use the findings to adapt your application before cutover. A typical remediation workflow looks like this:

1. **Triage using incompatible features.** Start with the **Incompatible Features** table to understand the categories of issues. Use the **Workaround Status** column to prioritize: features marked **None / Needs Investigation** have no known workaround and require the most attention, while **Available** and **New Workaround** entries have established remediation paths. A small number of feature types often account for a large share of incompatible queries — identifying these patterns lets you plan remediation in bulk rather than query by query.
2. **Review transformed queries.** For each incompatible query in the **Incompatible Queries** table, expand the row to compare the original Oracle SQL with the transformed EPAS version. Where HM's transformation is correct, you can adopt it directly in your application code.

3. **Handle queries without automatic transformations.** Where no transformation is available, the **Repair Handler Applied** field in the expanded row describes what was attempted and what still needs manual attention. Update your application code accordingly and retest against a non-production EPAS environment.
4. **Test on a non-production environment.** Before migrating production, run your updated application against a non-production EPAS cluster with migrated data and schema. Use the **# Leftover Incompatible Features** count in each expanded query row to confirm all issues have been resolved. Resolve any remaining runtime errors before cutover.

> **Note**
>
> Addressing SQL incompatibilities before the production cutover window reduces risk and shortens the maintenance window. See Oracle to HM-managed migration guide for guidance on sequencing these steps.

Related topics

- *Preparing SQL files for assessment* — Generate Oracle trace files, EDB Query Extractor output, or prepare plain SQL files for upload.
- *Creating an application record* — Set up an application record and upload SQL files for assessment.
- Application assessment overview — Understand how HM assesses application SQL and the supported file formats.
- *Migrations* — Review schema- and database-level assessments alongside application-level findings.

Migrating schemas

Schema migration is an essential step in database migrations. As the complexity of the process varies between different database systems, the tools you use depend on your source database.

Oracle sources

For Oracle sources, schema migration is more complex due to the differences between Oracle and PostgreSQL. The process requires a dedicated conversion step to handle syntax, data types, and functions that aren't directly compatible.

To streamline this process, Hybrid Manager (HM) provides an integrated Migration Portal. This service automates the assessment of your Oracle schema, flags incompatibilities, and includes an AI Copilot to help you resolve complex schema issues.

Oracle schema migration

Postgres source databases

Migrating a schema from a Postgres source (PostgreSQL, EDB Postgres Advanced Server, or EDB Postgres Extended Server on-premises or cloud distributions) is a more straightforward process. Since both the source and the HM-managed destination are based on the same technology, their schemas are compatible, eliminating the need for conversion.

You can use standard PostgreSQL tools for this task. We recommend using Migration Toolkit (MTK) or pg_dump to manually export your schema from the source database and import it into your HM-managed destination cluster.

3. Restore Postgres clusters

Using Migration Portal to assess, convert, and migrate Oracle schemas

Migration Portal helps you to assess, analyze, convert, and migrate Oracle database schemas to EDB Postgres Advanced Server. It's available as a web-based service or as an application in Hybrid Manager (HM).

Hybrid Manager Migration Portal vs web-based Migration Portal

Some features in the HM deployment of Migration Portal are unique or function differently from the web-based version.

AI Copilot

> **Note**
>
> You must manually enable this feature in Migration Portal in HM. In the web-based version, it's included out-of-the-box.

AI Copilot is an AI-driven chatbot tool that assists you with issues that can occur during the schema assessment and conversion in Migration Portal. It uses third-party AI services and is trained on both EDB's product documentation and support knowledge base to deliver use-case-specific solutions.

The AI Copilot isn't enabled by default in HM.

- To use the AI Copilot for Migration Portal queries, ask the HM administrator to enable and configure it with the AI/LLM of your organization .
- See AI Copilot for more information.

Project collaboration

> **Note**
>
> The project collaboration feature is available only in the Hybrid Manager Migration Portal.

Migration Portal supports collaborative DDL editing, enabling multiple users in the same HM project to work on the same schema file simultaneously. If another user modifies a DDL you're also editing, your changes will trigger a conflict resolution view. This view displays the other user's update in an **Update** view and your own edit in a **Target** view. You can then select **Use Updated** to adopt the other user's fix or **Reassess** to apply and save your own changes, overwriting the previous update. This approach ensures efficient, concurrent schema compatibility adjustments.

Portal access

Accessing the Migration Portal interface differs based on the version you're using.

- Access the web-based Migration Portal through the EDB portal .
- Access the Migration Portal built into HM from the HM console.
- Use *Assessing schemas for HM projects* to migrate schemas loaded by the EDB Postgres AI agent.
- Use the *Assessing schemas for external projects* to migrate externally provided schemas.

User permissions and role-based access

- The web-based Migration Portal provides each EDB account with a dedicated workspace, meaning you can't share projects with other users.
- Migration Portal built into HM offers project-scoped, collaborative access. Users with access to a specific HM project gain access to its associated Migration Portal project.
- In HM, Migration Portal user permissions are managed directly through HM. For instance, a user with Viewer permissions can't migrate schemas or reassess DDL.

Accessing Migration Portal

Migration Portal in Hybrid Manager (HM) offers two different workspaces that you access differently depending on how you want to use it.

Project-scoped workspace vs. user's workspace

Deploying HM includes a built-in instance of Migration Portal that has two types of workspaces:

- The *project-scoped workspace* in the Migration Portal is where you manage Migration Portal projects created with the HM console for schemas loaded by the *Setting up EDB Postgres AI agent to monitor AWS RDS deployments on HM* .

This shared portal environment aligns your Migration Portal projects directly with your HM projects. In this shared portal, you can see all Migration Portal projects created by you and other users who are part of the same HM project. You can't, however, see Migration Portal projects from migrations performed in other HM projects.

- The *user's workspace* in the Migration Portal is where you create and manage migration projects for external schemas that you manually upload for analysis, conversion, and migration.

This isn't a shared environment. Only you have access to the Migration Portal projects created in this workspace.

This separation allows Migration Portal to cater to different types and scales of migration efforts:

- Organized, large-scale, enterprise-driven projects (project-scoped)
- More flexible, smaller-scale, or individual-driven migrations (user's workspace)

Accessing

A Migration Portal has two type of workspaces for different purposes:

- To create and manage projects for schemas loaded into HM by the EDB Postgres AI agent, see *Assessing schemas for HM projects* . This uses the project-scoped access to Migration Portal.
- To manually upload externally provided DDL files and assess schemas with the Migration Portal, see *Assessing schemas for external projects* . This uses the user's workspace access to Migration Portal.

Assessing schemas for HM projects

Use the collaborative Project-scoped workspace vs. user's workspace of Migration Portal to convert and migrate schemas in the context of a database migration. HM provides an integration that lets you use the HM console to create a Migration Portal project. In this project, you can add source database schemas preloaded by the *Setting up EDB Postgres AI agent to monitor AWS RDS deployments on HM* while performing a database migration.

Only Oracle databases include an integration of Migration Portal into a migration workflow, as their schemas require compatibility assessment and often conversion before they're migrated to the destination database.

To upload and assess externally provided DDL schema files, *use the user's workspace* .

Prerequisites

- *Register an Oracle database* with HM.
- **Optional:** To facilitate resolving schema incompatibilities in Migration Portal, the HM administrator must enable and configure the AI Copilot with the AI/LLM of your organization .

Accessing Migration Portal

For Oracle databases only, use the Migration Portal integration to assess and migrate schemas:

1. In the **Estate** page, select the **Migrations** tab.
2. To open the detailed view of the database, select the name of the database you want to migrate.

If you already created a schema migration project for one, several, or all schemas, the HM console displays a link with the name of the project in the **Migration Portal Project** column. Select the link to open a Migration Portal workspace that contains all the schemas in that project.

If you haven't created a project for your schema migrations yet, select either the global **New Migration Portal Project** button or the schema-scoped + sign. This selection starts a new workflow to *create a Migration Portal* project.

Creating a Migration Portal project

If you registered an Oracle database with Hybrid Manager via the EDB Postgres AI agent, a Migration Portal project is **created automatically** for all the database schemas and dependencies. It creates a project per registered database (per `resource_id` established in the `beacon-agent.yaml`). Take into account that any schemas added post-registration aren't reflected in the Migration Portal project.

If the Migration Portal didn't automatically create projects for your registered databases, create projects manually. This can be the case when:

- The size of the ingested DDL exceeds the default limit of 300 MB.
- The size of the ingested DDL exceeds the *Customizing the maximum size of Migration Portal projects* .
- You disabled the automatic creation of Migration Portal projects.

Creating a Migration Portal project manually

Create Migration Portal projects manually when the automatic project creation is disabled or failed.

1. In the **Estate** page, select the **Migrations** tab.
2. To open the detailed view of the database, select the name of the database you want to migrate.
3. In the detailed view of the database, select the **Schemas** tab.
4. Select **New Migration Portal Project** or the the plus sign next to one of the schemas you want to migrate.

> **Tip**
>
> The **Dependencies** column shows the other schemas the current schema relies on. We recommend grouping a schema with all its dependencies in a single Migration Portal project to ensure that all referenced objects exist in the destination database.

1. In the pop-up window, enter the details for the Migration Portal project that will host the assessment, conversion, and migration of the schemas. Select **Continue**.

2. Select all the schemas you want to add to this Migration Portal project. You can select only schemas that the EDB Postgres AI agent has ingested successfully and schemas that aren't already part of a different Migration Portal project.

> **Note**
>
> You can't add schemas to the existing project later. However, you can use a different project to migrate other schemas to the same destination database.

1. Select **Create New Project**.

In the background, this creates a Migration Portal project and loads the source schemas into it for analysis.

After HM finishes loading the schemas and creating a project in Migration Portal, a link is added with the name of the project. Select the link to open Migration Portal.

Working in Migration Portal

The selected schemas were uploaded and assessed by Migration Portal. Use the AI Copilot or QuickHelp to help you resolve incompatibilities.

1. After you resolve all compatibility issues, select **Migrate to**.
2. Select the **Online migration** option (recommended).
3. Select the schemas you want to migrate, but clear **Constraints** from the list of objects underneath the database. Select **Next**.

> **Note**
>
> See *Migrating Postgres schemas* to learn more about why constraints must be disabled and other considerations.

1. In the **Connect to your cluster** page, enter the host name of the database cluster you want to use as a destination.

> **Note**
>
> If you're migrating to an HM-managed database cluster, you can find the read/write host and other connection data from the **Clusters** page. Select your cluster, and then select **Connect**.

1. Select **Next** and wait until the schemas were imported successfully.
2. Connect to the target database cluster and ensure the schemas were migrated.

Deleting a Migration Portal project

When using the project-scoped workspace of Migration Portal, you can delete projects only through the HM console, not through the integrated portal.

1. From the **Estate** page, select the **Migrations** tab.
2. Select the name of the database you were migrating.
3. In the database's detailed view, select the **Schemas** tab. You see a list of schemas and the assigned Migration Portal projects.
4. Locate the schema assigned to the Migration Portal project you want to delete, and select the + (plus) sign in the same line.

> **Note**
>
> If you're using a project to assess several schemas, deleting the project from any of the assigned schemas deletes the project for all schemas.

5. In the expanded view, under **Actions**, select the trash can icon.
6. Type **delete project** and select **Confirm Delete** to permanently delete the project.

Switching to user's workspace

While you're working on schemas in Migration Portal, you can switch to the Project-scoped workspace vs. user's workspace for manual DDL upload.

To do so, select **Switch to user's workspace**.

However, this workspace doesn't display the schemas you selected during project creation. It serves only as a way to quickly switch between tasks. See *Managing external schemas for external DDL* for more information.

Related topics

•

Perform an online or offline migration of the converted schemas to a destination database.

- Migration Portal documentation for recommendations and guidelines on how to use the portal.
- *Controlling Migration Portal project creation* by customizing the maximum size of projects or disabling automatic project creation.

Assessing schemas for external projects

Use the Project-scoped workspace vs. user's workspace of Migration Portal to assess, convert, and migrate externally provided DDL files for Oracle database migrations.

To access the project-scoped workspace of Migration Portal in the scope of Hybrid Manager (HM) migrations that leverage the EDB Postgres AI agent, *access Migration Portal from the detailed view of the Oracle databases* .

Prerequisites

- **Optional:** To facilitate resolving schema incompatibilities in Migration Portal, the HM administrator must enable and configure the AI Copilot with the AI/LLM of your organization .

Accessing Migration Portal

Use the Launchpad to access Migration Portal.

1. *Access the Launchpad* .
2. To open the Migration Portal tab, hover over the **Migration Portal** tile and select **View Service**.

Next steps

Use Migration Portal to:

- Resolve any incompatibilities and complete the schema conversion.
- Perform an online or offline migration of the converted schemas to a destination database.

See the Migration Portal documentation for recommendations and guidelines on how to use it.

Controlling Migration Portal project creation

If you registered an Oracle database with Hybrid Manager (HM) via the EDB Postgres AI agent, a Migration Portal project is **created automatically** for all the database schemas and dependencies.

It creates a project per registered database (per `resource_id` established in the `beacon-agent.yaml`) as long as the schema size per database doesn't exceed the default maximum size of 300 MB.

> **Note**
>
> Take into account that any schemas added post-registration aren't reflected in the Migration Portal project.

Customizing the maximum size of Migration Portal projects

The default maximum size of Migration Portal projects is 300 MB. If the schema ingested is larger than that, the Migration Portal will require you to *Creating a Migration Portal project manually* . However, there are scenarios where setting a different maximum size for automatically created projects is preferred, for example, for:

- Project size control
- Cost and resource management
- Workflow control

To alter the maximum size of Migration Portal projects created automatically, you must edit the `beacon_server` ConfigMap in your HM Kubernetes cluster and restart the server deployment. The `beacon_server` is the service that controls Migration Portal project configurations.

1. Edit the ConfigMap:

```
kubectl edit configmap beacon-server-config -n upm-beacon
```

1. Locate the `beacon_server.yaml` block and insert or modify the `max_automatic_project_creation_bytes` section under `server` :

```
apiVersion: v1
data:
  beacon_server.yaml: |
  # [...]
    server:
    # [...]
      migration_portal_configuration:
        max_automatic_project_creation_bytes: 150000000
```

> **Note**
>
> The value set for *max_automatic_project_creation_bytes* must be equal or less than the value configured for *schema_max_file_size* (which by default is also 300 MB), for example:

```
apiVersion: v1
data:
  beacon_server.yaml: |
```

(continues on next page)

(continued from previous page)

```
  # [...]
    server:
    # [...]
      stream_configuration:
        schema_max_file_size: 150000000
      migration_portal_configuration:
        max_automatic_project_creation_bytes: 150000000
```

```
kubectl rollout restart deployment upm-beacon-server -n upm-beacon
```

> **Warning**
>
> The *upm-beacon-server* supports all database registration and schema ingestion activities across the entire HM instance using EDB Postgres AI agents. Restarting the *upm-beacon-server* will halt all _ongoing_ database registrations and schema ingestions, regardless of the project. Ensure you coordinate this restart with other users.

Databases that already appear in the Estate > Migrations tab, for which schema ingestions have finished, are not affected by restarts. .. ::

1. You can monitor the status of your restarted service:

```
kubectl get pods -n upm-beacon
```

Disabling the automatic creation of Migration Portal projects

While automatic project creation is convenient for getting started quickly, there are scenarios where manual management is preferred, for example, for:

- Cost and resource management
- Staging and development environments
- Workflow control and auditoring

To disable automatic Migration Portal project creation, you must edit the `beacon_server` ConfigMap of your HM Kubernetes cluster and restart the server deployment. The `beacon_server` is the service that controls Migration Portal project configurations.

1. Edit the ConfigMap:

```
kubectl edit configmap beacon-server-config -n upm-beacon
```

1. Locate the `beacon_server.yaml` block and insert or modify the `migration_portal_configuration` section under `server` :

```
apiVersion: v1
data:
  beacon_server.yaml: |
  # [...]
    server:
    # [...]
      migration_portal_configuration:
        disable_auto_create_mp_projects: true
```

1. Apply the changes by restarting the `beacon_server` deployment:

```
kubectl rollout restart deployment upm-beacon-server -n upm-beacon
```

> **Warning**
>
> The *upm-beacon-server* supports all database registration and schema ingestion activities across the entire HM instance using EDB Postgres AI agents. Restarting the *upm-beacon-server* will halt all _ongoing_ database registrations and schema ingestions, regardless of the project. Ensure you coordinate this restart with other users.

Databases that already appear in the Estate > Migrations tab, for which schema ingestions have finished, are not affected by restarts. .. ::
Related topics

- *Migrating Oracle databases with HM* .
- *Controlling schema ingestion* .
- Migration Portal documentation for recommendations and guidelines on how to use the portal.

Using the HM Console to perform data migration

To perform a data migration, you can leverage the data migration features that are provided by the HM Console. This section provides reference material to explain the options available under the **Migrate** section of the left navigation in a project.

- *Understanding deployment resources* : view and manage migration database sources that you have added with the DMS Agent in reader mode.
- *Diverse migration destinations* : view and manage migration database [destinations] that you have added with the DMS Agent in writer mode.
- *Migrations* : view existing data migrations or start new data migrations from existing sources to existing destinations.
- *Step 2: Credentials* : Obtain project-scoped credentials required to connect sources and destinations with the DMS Agent.

Sources

Sources are databases you connected to the Hybrid Manager using the *Data Migration Service or DMS* so you can use them as sources for migrations. For a database to appear in this view, you must install, configure, and run the *EDB DMS Agent in reader mode* .

This view displays all the configured sources in the project.

Each source's view shows the database name, database type (Postgres or Oracle), `cdcagent` ID, the current *Step 4: Monitor Deployment Status* , the source type, and more.

Status

HM can display the following statuses for a source.

Status	Description
Creating	The database source connection is being initialized. This is a transitional state.

continues on next page

Table 53 – continued from previous page

Status	Description
Waiting for tables metadata	The EDB DMS Agent is retrieving information about the tables (for example, names, columns, data types). If the source is stuck in this state, there might be connectivity issues with the database.
Ready to start a new migration	The EDB DMS Agent can successfully read the source database and you can use it for a migration.
Running	A migration process using this source is currently active and data is being transferred.
Paused	An ongoing migration process that's using this source was temporarily stopped.
Stopping	A currently running migration for this source is in the process of being terminated.
Connectivity error	The EDB DMS Agent can't establish a connection to the externally managed database. This can be due to incorrect connection details (host, port, database name, username, password) or network issues.
Error	Unspecified problem. Investigate *cdcagent* logs to understand the specific cause.

Note

You can't use a source or destination that's part of an active migration (in any state, including running or paused) for another migration. The existing migration must be completed or deleted before you can reuse the source or destination.

Destinations

Destinations are externally managed databases you connected to the Hybrid Manager using the *Data Migration Service or DMS* so you can use them as destinations for migrations. For a database to appear in this view, you must install and configure the *EDB DMS Agent in writer mode* .

This view displays all the configured destinations in the project.

Each destination's view shows the database name, database type, `cdcagent` ID, the current *Step 4: Monitor Deployment Status* , the destination type, and more.

Status

HM can display the following statuses for a destination.

Status	Description
Waiting for tables metadata	The EDB DMS Agent is retrieving information about the tables (for example, names, columns, data types). If the destination is stuck in this state, there might be connectivity issues with the database.
Ready to start a new migration	The EDB DMS Agent can successfully access the destination database, and you can use it for a migration.
Running	A migration process using this destination is currently active and data is being transferred.
Paused	An ongoing migration process that's using this destination was temporarily stopped.
Connectivity error	The EDB DMS Agent can't establish a connection to the externally managed database. This can be due to incorrect connection details (host, port, database name, username, password) or network issues.
Error	Unspecified problem. Investigate HM Console and *cdcagent* logs to understand the specific cause.

> **Note**
>
> You can't use a source or destination that's part of an active migration (in any state, including running or paused) for another migration. The existing migration must be completed or deleted before you can reuse the source or destination.

Migrations

Migrations are processes you can start in a project of the Hybrid Manager (HM) to move data from one database to another. These processes include migrating an external Oracle database to an HM-managed cluster and to an externally managed Postgres instance, among others.

Viewing existing migrations

After creating one or several migrations, this view displays all the migrations in the project in a table format. The table contains the columns:

- **Name** : Name you assigned to a migration.
- **Source** : Name of the source, configured through the **Sources** page.
- **Destination** : Name of the destination, either a database in an HM-managed cluster or a destination configured through the **Destinations** page.
- **Snapshot type** : The replication mode configured for each migration:
- **Snapshot** : The migration runs a snapshot only. Any subsequent changes to the source aren't replicated to the destination.
- **Snapshot + Streaming** : The migration runs an initial snapshot, then continues streaming ongoing changes from the source database.
- **Migration progress** : This column summarizes how far a migration has progressed. It displays three metrics for each of the migrations:
- **Total number of tables** : The total number of tables included in this migration.
- **Snapshot tables completed** : The number of tables that have finished the snapshot stage.
- **Migrated rows** : The cumulative number of rows that have been migrated across all tables in this migration.
- **Status** : Current status of the migration. The values can be one of the following: `Creating` , `Starting` , `Running` , `Pausing` , `Paused` , `Error` , `Completed` , `Failed` , `Deleting` , or `Deleted` .
- **Delete** : At the end of each row, the trash can icon allows you to delete a migration.

At the top right of the page, you can use the **Sort** dropdown to sort the list of migrations by **Creation date**. By default, the list is sorted by newest to oldest created date.

You can use **Create Migration** to create a new migration. See *Creating a migration* for more details.

Status

HM can display the following statuses for a migration.

Status	Description
Creating	HM is creating resources required to perform a migration (for example, Kafka nodes).

continues on next page

Table 55 – continued from previous page

Status	Description
Starting	The initial phase of execution started, where the EDB DMS Agent connects to the source and/or destination databases.
Running	The migration is active and data is being transferred.
Pausing	The migration is being temporarily stopped.
Paused	The migration was temporarily stopped and can be resumed.
Error	Unspecified problem. Under **Migrations** , select the name of the migration to open the details page. Look at the **Error log** section for more information about the cause.
Completed	The migration finished successfully.
Failed	The migration encountered an error and could not be completed. Investigate HM Console logs to understand the specific cause.
Deleting	The migration is being terminated.
Deleted	The migration was successfully terminated and removed.

Note

You can't use a source or destination that's part of an active migration (in any state, including running or paused) for another migration. The existing migration must be completed or deleted before you can reuse the source or destination.

Migration details page

For each migration that appears on this page, you can open a more detailed view by selecting the name of the migration.

Mark as Completed option: If you selected the ***Snapshot + Streaming*** option when *Creating a migration* , you can use **Mark as Completed** to stop the streaming procedure.

Overview tab

The **Overview** tab contains additional information on the status of the migration, type of migration, source and destination database information, and error logs that can help you troubleshoot.

Tables tab

The **Tables** tab provides detailed information about the progress of individual tables within the migration.

1. **Migration summary**

At the top of the tab, a summary section displays key metrics for the migration, including:

- **Total schemas** : The total number of schemas included in this migration.
- **Total tables** : The total number of tables included in this migration.
- **Snapshot completed** : The number of tables that have finished the snapshot stage and are ready for streaming (if applicable).

2. **Search and filtering bar**

The filtering bar allows you to narrow down the table list to focus on specific datasets or troubleshoot migration bottlenecks.

- **Global search** : Use the search bar to find specific tables by entering keywords or phrases contained in the schema or table names.

- **Snapshot status filter** : Use the status filter dropdown to view tables based on their migration status, such as **Pending**, **In Progress**, **Completed**, or **Failed**.

3. **Sorting dropdown**

The sorting dropdown adjusts the display order of the table list as per the selected order. By default, the tables are sorted alphabetically by schema and table name. You can change the sorting order based on different criteria, such as:

- **Table name** : Sort tables alphabetically by their names.
- **Snapshot status** : Sort tables based on their current snapshot status.
- **Migrated rows** : Sort tables by the number of rows that have been migrated.

4. **Tables detail view**

This section provides row-level granularity for every table in the migration.

For each table, you can view the source and destination schema and table names, the snapshot status, the number of mapped columns and migrated rows, as well as the migration rate for ongoing migrations. The snapshot status reflects the overall migration state for snapshot-only jobs, or the snapshot phase status for snapshot + streaming jobs. Each row is expandable, allowing you to view the detailed column-level mapping from the source to the destination.

Error history tab

The **Error history** tab displays a list of all the errors the DMS has logged during this migration. For each error, you can view a summarized error message, the error source (DMS agent in reader or writer mode, Kafka, etc.), whether the error is active or not, and the error timestamp. To obtain more details for the error, select the information icon to open a view with the full error log. To help you isolate specific issues, the **Error history** tab includes advanced filtering and time-range selection capabilities.

- **Global search** : Use the search bar to find specific errors by entering keywords or phrases contained in the error messages.
- **Flexible time ranges** : Use the time-range selector to filter errors based on predefined intervals (last 7 days, last 30 days, last 90 days) or specify a custom date and time range.
- **Advanced filtering** : Use the filter bar to refine the list based on error attributes. You can filter using:
- **Error source** : View only errors originating from specific components, such as the DMS agent in reader or writer mode, Kafka, etc.
- **Active** : Status toggles to quickly filter between active and resolved errors.

Creating a migration

Select **Create Migration** to create migrations. For a step-by-step example, see *Creating a migration* in the self-managed Postgres to HM guide, or the equivalent section in your migration guide.

Mapping schema, tables and columns during a migration

When you create a migration with the **Data only** scope, the Data Migration Service (DMS) allows you to map your source schema, table, and column names to different destination schemas, tables, and columns. This method is particularly useful for restructuring your database during a migration, enabling you to align table names with new naming conventions or improve clarity.

Column mapping isn't available for the **Schema and data** or **Schema only** scopes. In those scopes, HM drives the schema migration automatically using the Migration Portal project created from your source database. Because HM creates the destination objects itself, it controls their names — any column renaming you configure is preserved in the migration definition but won't take effect unless you switch the scope to **Data only**.

Prerequisites

The DMS requires that all destination objects already exist in the destination database before the data migration begins. If you plan to restructure data by mapping a source object (schema, table, or column) to a different name in the destination (for example, source table `loc` to destination table `location`), you must ensure the destination object (`location`) has been manually created or renamed in the destination database prior to initiating data migration.

- **Schema migration tools** : If you use a utility like Migration Toolkit or pg_dump/pg_restore for schema import, the objects are created with the same name as the source. You must then manually rename the objects in the destination to match your desired mapping. For example, you can use Migration Toolkit's offline option or pg_dump to generate an SQL script, edit the object names directly in the script, and then apply the modified script to the destination.
- **Manual/alternate creation** : If you create the destination objects manually, ensure the objects are defined with compatible data types for all mapped columns. While DMS can migrate data to differently named objects, the structure of the destination object must match the source data.

Object mapping

1. Open the **Create Migration** window from **Migrate** > **Migrations** > **Create Migration** in the HM Console. See Creating Migration Portal secrets using `edbctl <Creating Migration Portal secrets using *edbctl*>` for more details.
2. Navigate to the **Tables** tab for your migration.
3. To map a schema to a renamed destination schema:
4. Find a table within the list (`<schema>.<table>`) whose schema you want to rename.
5. Select the pen icon next to the `<schema>.<table>` name.
6. Enter the new name for the schema.
7. To map a table to a renamed destination table:
8. Find table within the list (`<schema>.<table>`) whose table you want to rename.
9. Select the pen icon next to the `<schema>.<table>` name.
10. Enter the new name for the table.
11. To map a column to a renamed destination column:
12. Expand one of the items from the list (`<schema>.<table>`) with the dropdown arrow.
13. Find the column you wish to rename in the **Source Column** list.
14. Enter the new name for the column in the **Destination Column**.

This process allows you to map schemas, tables and columns as part of your migration definition.

Credentials

Credentials are required so you can perform migrations. Before you can add a source or destination for a migration to the Hybrid Manager (HM), you must create migration credentials. You can then use them to allow the EDB DMS Agent to connect to the HM.

The **Credentials** view shows all the credentials in the project in a table format. The entry for each credential displays its name, status, when it was created, and expiry date.

Creating migration credentials

You create migration credentials as part of configuring the EDB DMS agent. For a step-by-step example, see *Creating migration credentials* in the self-managed Postgres to HM guide, or the equivalent section in your migration guide.

The decompressed credentials folder contains these files:

- `int.crt`
- `int.truststore.p12`
- `client.keystore.p12`
- `client-key.pem`
- `client-cert.pem`

Using the Migration API

The Migration API is a crucial tool for automating your database migration tasks. It allows you to programmatically manage and monitor migrations, offering a powerful way to integrate migration workflows into your existing CI/CD pipelines or automated scripts. Instead of navigating the UI, you can use the API to manage your migration projects, view the detailed schema reports, see data migration status, and perform many more administrative tasks.

This API is for developers, DevOps engineers, and DBAs who need a scalable and automated way to handle a large number of migrations or integrate them into broader automation strategies.

The Hybrid Manager (HM) platform provides a broader set of APIs to support end-to-end database migration, including assessment, schema conversion, and data migration. However, only a subset is shown in the *Examples* .

Examples

Create Migration Portal project

This API endpoint is used to create a new migration project in the Migration Portal. It's a `POST` request that sends a JSON payload to define the project's properties and configurations.

- **Path** : `/api/v1/projects/prj_LTIfqp81u6MslAk4/migration-portal-project`
- **Method** : `POST`

URL path parameters:

- `prj_LTIfqp81u6MslAk4` : The unique identifier for the project's scope.

```
Request Body:
{
    "projectId": "prj_LTIfqp81u6MslAk4",
    "resourceId": "oracleSource",
    "schemaIds": [
        "cea06ec8-8641-4dff-9631-4b19fcba5212",
        "5770b6fa-295c-40af-bd02-1acbe02c9eb6"
    ],
    "mpProjectName": "my_project_name",
    "applicationInterface": "JDBC",
    "targetDbVersion": "EPAS_17",
    "useUniqueSchemaObjectSuffixes": true,
    "useOracleDefaultCase": true,
    "assessDefaultProfile": true,
```

(continues on next page)

(continued from previous page)

```
    "description": "My Project Description (Optional)"
}
```

Request body description:

- `projectId` : The unique identifier of the project in HM.
- `resourceId` : The unique identifier for the source database (for example, `oracleSource`).
- `schemaIds` : An array of unique IDs for the schemas you want to include in the migration project.
- `mpProjectName` : The custom name you want to assign to the new Migration Portal project.
- `applicationInterface` : The type of interface used to connect to the source database (for example, `JDBC` , `ODBC` , `.NET` , `OCI` , `ProC` , Other).
- `targetDbVersion` : The version of the target database you're migrating to (for example, `EPAS_17` for EDB Postgres Advanced Server version 17).
- `useUniqueSchemaObjectSuffixes` : A Boolean value that, when set to `true` , ensures that object names in the target schema have unique suffixes to avoid conflicts.
- `useOracleDefaultCase` : A Boolean value that, when set to `true` , respects Oracle's default behavior of using uppercase for object names.
- `assessDefaultProfile` : A Boolean value that, when set to `true` , includes the default profile in the assessment.
- `description` : A custom, optional description for the migration project.

Delete Migration Portal project

This API endpoint deletes an existing migration project from the Migration Portal. It's a `DELETE` request that targets the specific project by its ID.

- **Path** : `/api/v1/projects/prj_LTIfqp81u6MslAk4/migration-portal-project/42`
- **Method** : `DELETE`

URL path parameters:

- `prj_LTIfqp81u6MslAk4` : The unique identifier for the project's scope.
- `42` : The unique ID of the specific Migration Portal project you want to delete. This ID is assigned after the project is successfully created.

5.3.3.5 Asset management

The Asset Management section of the Hybrid Manager (HM) console provides tools for managing the assets that are available for use in your clusters. This includes managing container images and AI models, as well as repositories that contain these assets.

5.3.3.5.1 Managing HM Postgres and model assets

- *Asset Library apps require Project Owner or Project Editor role*

The main page for managing assets in the HM console, where you can view and manage container images and knowledge bases.

- *Container images*

You can explore and manage container images in the asset library by searching, filtering, and viewing image details.

- *Manage repositories in the asset library*

Control the visibility of repositories and images in the asset library by managing repository registrations and rules.

- *Knowledge bases and retrieval*

Manage AI models in the asset library, including creating and managing knowledge bases that contain AI models for use in clusters.

Asset library

The asset library is where you manage the container images and models that are available for use in your Hybrid Manager clusters.

Accessing the asset library

To access the asset library, select the **Asset Library** tab at the top of the Hybrid Manager (HM) console.

Asset Library page

The **Asset Library** page displays a summary count of the images available in the library, a search and filter bar, and a list of the major images available in the library.

The asset library is divided into two main sections: **Container Images** and **Models**. Each section allows you to view and manage the assets available for use in your HM clusters.

Container images

The **Container Images** section of the asset library is where you can view and manage the container images available for use in your HM clusters. This includes the ability to search for specific images, filter the list of images by various criteria, and view detailed information about each image. See *Container images* for more information on how to use this tab.

Models

A model is an AI model that can be used in HM. You can use these models with specific datasets to perform various tasks, including natural language processing, image recognition, and more. See *Knowledge bases and retrieval* for more information on how to use this tab.

Container images in the asset library

The **Container Images** section of the asset library is where you can view and manage the container images available for use in your Hybrid Manager (HM) clusters.

Container images

A container image is a configuration of the database that includes various extensions. Each image is identified by its name and an image tag.

For each name, there can be multiple image tags representing different versions and different combinations of extensions loaded in the image.

For example, official HM images have an image name `postgresql` and image tag `17.4-2503191025` that represents a community PostgreSQL image for version 17.4. The `2503191025` is the build date and time of the image (2025/03/19 10:25). The postfix `-full` indicates that this image includes all the extensions available in the HM Postgres distribution.

This tag format is specific to the official HM images and may differ for other images. The image name is used to identify the type of image, such as a particular flavor of Postgres. The image tag is used to identify the specific version of the image.

Image statistics

A summary of the number of images available in the library is listed by type. The types of images are:

- **Total Image Tags**: The total number of image tags available in the library, where a tag represents a specific version of an image.
- **PostgreSQL Images**: The number of Postgres images available in the library.
- **AI Model Images**: The number of AI model images available in the library.

Container images search and filter bar

In this view, use the **Search** field to enter a search term that's part of the name or description of the image repository. Use the **Filter** field to filter the repositories by the type (Postgres or AI Model).

Use the **Sort** field to sort the repositories by name, by description, by type, or by the date the repository was updated.

On the search and filter bar, select **Manage Repositories** to manage the repositories that are available in the library. Use the dialog box to add, edit, or delete repositories. In the **Manage Repositories** page, you can manage the repositories and images that are available in the asset library. For more information on how to manage repositories, see *Manage repositories in the asset library* .

Container images cards

Each image repository is displayed in a card format with the following information:

- **Name**: The name of the image repository.
- **Description**: A short description of the image repository.
- **Tags**: HM tags associated with this image repository. If no tags are associated with the image repository, **Add Tag** appears.
- **Updated**: The number of days since the image repository was last updated.

Select the name of an image repository to go to the page with the details of that image.

Image details

The **Image Details** page for an image has two tabs: **Overview** and **Image Tags**. An **Add Tag** button lets you add an HM tag to the image repository. Any existing tags are displayed before the button, and you can select a tag to view its details.

Overview tab

The **Overview** tab provides a summary of the image repository, including:

- **README**: A textual description of the image repository, which can include any relevant information about the image, such as how to use it, what it contains, and other useful details.
- **Recent Image Tags**: The 10 most recently created image tags for this image repository. You can select each tag to go to the **Image Tag Details** page for that tag.

The **Clusters Using This Image** section lists any clusters in the estate that are using an image from this repository. If no clusters are using this image, the message "No Data" appears.

Use the search and filter bar to search for clusters. Use the **Search** field to enter a search term that's part of the name or description of the cluster. Use the **Filter** field to filter the clusters by their attributes (cluster type, cluster architecture, Postgres flavor, and Postgres version). Use the **Sort** field to display clusters to sort by name, description, or the date the cluster was created.

Select a cluster name to go to the **Cluster Details Overview** page for that cluster.

Image tags

Select the **Image Tags** tab to see a list of all the image tags associated with this image repository.

Each tag is displayed in a table format with the following information:

- **Name**: The name of the image tag.
- **Created**: The date and time when the image tag was created.
- **Last Pushed**: The date and time when the image tag was pushed to the repository.
- **Platform Architecture**: The platform architectures for which the image tag is available, either X86 or AMD64.
- **SHA**: The SHA256 hash of the image, which is a unique identifier for the image. You can use it to verify the image against other repositories. The last 8 digits of the hash are displayed. Hover over those digits to see the full hash. You can copy the full hash using the **Copy** button.

To navigate through the list of image tags, you can use the search and filter bar at the top of the table. In the **Search** field, enter a search term that's part of the image tag name. Use the **Filter** field to filter the image tags by their attributes (EDB Official Release, Location Available, Platform Architecture). Use the **Sort** field to sort the image tags by version, date created, or last pushed date.

Select the name of an image tag to go to the **Image Tag Details** page for that tag.

Image tag details

The **Image Tag Details** page shows the most detailed information about a specific image tag.

At the top of the page is the name of the image tag. Below that are any HM tags associated with this image tag, which can include the EDB Official tag, the Transparent Data Encryption tags generated by the platform, and any tags you created. Select **Add Tag** to add or remove tags to the image tag and create new tags if needed.

Next are the following:

- **README**: A textual description of the image repository, which can include any relevant information about the image, such as how to use it, what it contains, and any other details that may be useful.
- An image attributes panel that displays the following information:
- **Version**: The version of the image tag.
- **Base Image**: The underlying OS image the image is based on.
- **Image SHA**: The SHA256 hash of the image.
- **Platform Architecture**: The platform architectures for which the image tag is available, either X86 or AMD64.
- **Last Pushed**: The date and time when the image tag was pushed to the repository.
- **Created**: The date and time when the image tag was created.
- **Platform Architecture**: The platform architectures for which the image tag is available, either X86 or AMD64.
- **Transparent Data Encryption**: Indicates whether the image tag supports Transparent Data Encryption (TDE).

A field labeled **Deployment Locations** lists the locations where the image tag is available.

A table shows the extensions that are included in this image tag. Each extension is displayed with its name, version, and description. Use the **Search** field to search for a specific extension by name or description. You can sort this table by name.

The **Clusters Using This Image Tag** section lists any clusters in the estate that are using this image tag.

Manage repositories in the asset library

While the connection to the underlying repositories is configured by the system administrator, you can manage the repositories that are available in the asset library.

You can achieve this by allowing the system to include repositories and tagged images that match a specific pattern or patterns. Each pattern is defined as a rule. You can then control which repositories are included in the asset library and which images are available for use in your Hybrid Manager (HM) clusters.

By default, the asset library includes all the official HM repositories and all the tagged images in those repositories.

For more control over the repositories and images available in the asset library, you can manage your own private repository, which can be hosted on a private registry. This gives you complete control over what the registry contains and allows you to add your own images and manage them. Whichever route you choose, the asset library's repository inclusion rules still operate to control what's available in the asset library.

Managing repositories

In the **Container Images** tab of the asset library, select **Manage Repositories**. Use the **Manage Repositories** page to add, edit, or delete repository registrations that are available in the asset library.

A table of the repositories that are currently registered in the asset library has the following columns:

- **Repository Name Regex**: The regular expression that the name of the repository must match.
- **Number of Rules**: The number of rules that are associated with this repository registration.
- **Created**: The date and time when the repository registration was created.

When populated, each row includes an **Edit** button to edit the repository registration and a **Delete** button to delete the repository registration.

Use the search field to search for a specific repository registration by name. Use the sort control to sort the table by the repository name or created date.

In the top-right corner of the screen, select **Register New Repository** to add a repository registration.

Registering new repository

Select **Register New Repository** to open a dialog box where you can enter the details of the new repository registration:

- **Repository Name Regex**: A regular expression that matches the name of the repository. This is used to filter which repositories are included in the asset library.

This field must match the name of the repositories that you want to include in the asset library. For example, if you want to include all repositories that start with `edb-postgres-advanced-` , enter `^edb-postgres-advanced-.*$` as the regular expression. You can enter plain text, but it's treated as a regular expression. We recommend using a regular expression to ensure that the repository name matches the pattern you want.

- **Rules**: A list of rules that are associated with this repository registration. Each rule consists of a regular expression that matches a tag that's associated with the image. The rules are used to filter which images are included in the asset library. For example, if you want to include all images with the tag `latest` , enter `^latest$` as

the regular expression. You can enter plain text, but it's treated as a regular expression. We recommend using a regular expression to ensure that the tag exactly matches the pattern you want.

To add a rule, select **Add Rule**, which adds a new row to the list of rules. To delete a rule, select **Delete** next to the rule.

Once you have entered the details of the new repository registration, select **Register Repository** to save the registration. The new repository registration is then added to the list of registered repositories and is available in the asset library. This may take a few minutes to complete as the system must scan the repository for images that match the rules you defined.

When the operation is complete, you return to the **Manage Repositories** page, where you can see the new repository registration in the list of registered repositories.

Return to the *Asset Library apps require Project Owner or Project Editor role* to see the effect of new repository rules and the images that are now available.

Knowledge bases in the asset library

The **Knowledge Bases** page in the asset library lets you manage your collections of AI models that can be used in your Hybrid Manager (HM) clusters.

The **Knowledge Bases** page consists of a search and filter bar, a sort control, and a table of knowledge bases. Each knowledge base is displayed in its own row with the following information:

- **Name**: The name of the knowledge base.
- **Source Data**: The source data that the knowledge base is built from.
- **Database**: The database that the knowledge base is built from.
- **Cluster**: The cluster that the knowledge base is built from.
- **Model**: The AI model that the knowledge base is built from.

5.3.3.6 Backups and restores

The fundamental tasks of backing up and restoring are:

- *HM backups and DR for multi-DC environments*
- *Backups and restores*

5.3.3.6.1 Backups

Hybrid Manager (HM) features backup and recovery capabilities for its components using various technologies. It offers two backup strategies:

- Postgres backups, which back up your data
- HM backups, which back up the HM system's core functionality

Postgres backups (data plane)

Postgres backups focus on protecting your user data stored in the Postgres database:

- Base backups are available in all HM deployments.
- Snapshot backups are available in cloud service provider (CSP) environments and on-premises deployments where appropriate storage and drivers are configured.

Postgres backups leverage robust database features, including write-ahead log (WAL) archiving, to provide granular point-in-time recovery (PITR). This approach allows for a fine-grained recovery point objective (RPO).

> **Note**
>
> Local storage isn't currently supported for snapshot backups.

HM backups (control plane)

HM backups protect the HM system, including its metadata and system databases. They operate on a separate schedule and produce different backup files from Postgres backups. HM backups are file-system based, with hourly backups of metadata and backups of the system's Postgres databases by default.

Unlike Postgres backups, HM backups don't offer continuous backup through WAL archiving. Recovery is therefore limited to the hourly or user-defined backup intervals.

Key differences and recovery implications

Because of these differences, the RPO and recovery time objective (RTO) vary between your user data (Postgres) and the HM system.

- Postgres (data) offers strong RPO capabilities due to PITR and WAL archiving, potentially allowing for recovery to a very recent point in time.
- For HM (system), RPO is limited to the user-modifiable backup schedule, which is hourly by default.

Component	RPO	RTO	Method	Location
HM	1 hour (default)	30 mins or less	Velero	Object storage
Postgres data plane	5 minutes or less	Based on data size, backup method, and hardware.	Snapshot or base backup	Object storage

Data size

For Postgres clusters using base backup, a data size of less than 500 GB is expected to be the effective size limit. (Though it might be up 1 TB, depending on hardware performance.) A Postgres cluster with more than 1 TB of data may encounter backup durations and performance loads that negatively affect quality of service.

For Postgres clusters using snapshot backups, data sizes larger than 1 TB are expected to be supported without the backup and restore processes negatively affecting service quality.

Backup transportability and disaster recovery (DR)

In the simplest configurations, for example using TopoLVM (not available at GA in version 1.1), snapshot backup and recovery is expected to be local only, meaning that snapshot backups can't be moved to another appliance, data center, or location. This limitation effectively rules out DR with the most basic implementation of snapshots.

When HM is configured with enterprise storage and CSI drivers that support snapshots, HM can take advantage of that functionality to provide cross-location DR.

Snapshot availability

Volume snapshots are supported across all environments that provide Kubernetes-compatible volume snapshot functionality.

Snapshot transportability

Transportability of snapshots depends on the specific CSI driver implementation.

AWS specifics

- By default, AWS volume snapshots aren't transportable.
- While restoration to a new host in the same AWS region is possible, cross-region restoration requires additional manual steps.

TopoLVM specifics

TopoLVM snapshots aren't transportable and can't be restored to a different host.

Configuration

You can configure storage, object storage, and backups for:

- Single-node cluster
- Primary/standby cluster
- EDB Postgres Distributed (PGD) cluster

Single-node cluster

Backups of a single-node cluster can have the following characteristics:

- A single-node cluster is deployed using HM.
- You can modify the retention period and backup time or use the default values.
- Automated backups occur after cluster deployment and then daily.
- You can run manual backups.

Back up primary/standby cluster

Backups of a primary/standby cluster can have the following characteristics:

- A primary/standby cluster is deployed using HM.
- You can modify the retention period and backup time or use the default values.
- Automated backups occur after cluster deployment and then daily.
- You can run manual backups.
- Backups are taken from a replica (primary/secondary).

Backup PGD cluster

Backups of a PGD cluster can have the following characteristics:

- A PGD cluster is deployed using HM.
- You can modify the retention period and backup time or use the default values.
- Automated backups occur after cluster deployment and then daily.
- You can run manual backups.
- Backups are taken from a replica (primary/secondary).

Backup compression

Backup compression reduces the size of backup data using a compression algorithm before storing it to object storage. HM compresses Postgres backups by default using algorithms from the Barman Cloud plugin. Compression is applied separately to base backups and WAL archiving:

- Base backups use Snappy compression by default.
- WAL archiving uses zstd compression by default.

Compression can improve backup performance and reduces storage consumption. It's enabled automatically for all new clusters — no additional configuration is required to activate it.

You can change the compression algorithm for each backup type independently in the *Backup and security settings* tab when creating or editing a cluster.

HM backups

HM backups include metadata needed to restore the HM system. They don't include backups of user databases. HM backups use Velero to back up to object storage. The default backup schedule is hourly. You can adjust the schedule.

When using the default schedule, the RPO is one hour. This means you can expect to lose up to one hour of HM data in the event of a crash. This doesn't mean data loss at the user database level since user databases are protected with continuous backups of Postgres.

Taking a Postgres backup

You can use the HM console, API, or CLI to take a Postgres backup. To take a Postgres backup with the API or CLI, refer to the API documentation link in the HM console.

To take a Postgres backup using the HM console:

1. Select the **Backups** tab to view existing backups.
2. Select **Create Backup**.
3. Enter the location, backup name, and backup method, then select **OK**.

Taking an HM backup

To configure backups of the HM system, see *Hybrid Manager disaster recovery* .

Terminology

The following terminology is used for backup and recovery with Postgres and HM.

Backup

A copy of one or more files from the original location to a secondary location, like object storage. Backups vary by component type (for example, control plane versus data plane) and deployment type (for example, CSP, engineered system, or RHOS).

Backup retention period

The length of time that backup copies of data are stored before being deleted or overwritten. It defines how long you keep your backups available for recovery purposes.

Barman

An open-source Postgres backup tool that's primarily maintained and supported by EDB. When used alone, the term Barman refers to the Barman server or core backup tool, not the *Barman cloud utility scripts* . Barman automates many of the Postgres backup and recovery primitives that are required for enterprise-class backup and recovery. Barman provides innovative features like incremental backups (versus block-level incremental backups that were offered by Postgres), WAL streaming to achieve RPO=0 (or near zero), backup reporting, automated backup file retention management, and other useful capabilities.

Barman cloud utility scripts

A collection of scripts that can be used separately from the Barman server. Their purpose is to copy Postgres backups to S3-compatible object storage. Barman for the cloud is used by the CNP operator and EDB Cloud Service to copy Postgres backups to object storage.

Compression algorithm

The method used to compress backup data. The supported algorithms differ between base backups and WAL archiving:

- Base backups: none, Snappy, gzip, bzip2
- WAL archiving: none, Snappy, gzip, bzip2, lz4, xz, zstd

Performance characteristics vary by environment and workload.

Base backup

The Postgres backup utility pg_basebackup that automates the backup, or file copy, of the required files needed to restore a Postgres instance.

Cloud

A public cloud provider, that is, AWS, Microsoft Azure, and Google Compute Platform (GCP).

Cluster

A collection of databases in a single Postgres instance (also known as the Postgres cluster). In HM, it also refers to a deployment type (for example, single-node, PSR, PGD cluster).

Cloud-native Postgres (CNP)

The open-source Postgres operator developed and maintained by EDB. It provides the foundation for Postgres on Kubernetes in HM.

Postgres cluster

A Postgres instance manages a Postgres cluster. A database cluster is a collection of databases that are stored in a common file system location (the data directory).

pg_dump

A data export and import utility (also referred to as *dump*). Although it offers capabilities to export databases in part or in whole, it isn't an enterprise-class, scalable backup tool. However, pg_dump is widely used to export data for a variety of purposes, including creating test databases, data archiving, and migrating from one database to another (or one DBaaS provider to another). HM users use pg_dump to access their Postgres data. The pg_dump utility is installed with the Postgres client and can be used from a user's client machine.

Recovery

The process of making the restore data consistent and up to date by applying backup data files and changes recorded in the WAL files.

Recovery point objective (RPO)

The point or time (for example, 5 minutes) in a database timeline that can be used for recovery. It is commonly described as answering the question, "How much data can I afford to lose?" RPO is the maximum tolerable amount of data loss that an organization can withstand after a disruption, measured in time. It defines the point in time to which systems and data must be recovered.

Recovery time objective (RTO)

The maximum tolerable amount of downtime that an organization can withstand after a disruption. It defines the target time within which systems and data must be restored. For example, a production database may have an RTO of 30 minutes, which means that the amount of downtime to perform restore and recovery must not exceed 30 minutes.

Restore

The process of copying files from a backup location to the original or an alternative location to prepare for recovery. An example is copying backup files from object storage to the database server.

Snapshot

A volume-level snapshot or file-system-level backup of data. Snapshot is the fastest and most scalable backup and recovery method available. Snapshot capabilities vary by HM deployment type and the availability of storage features like transportability.

Transportable volume snapshot

A point-in-time copy of data that can be moved or copied to a different storage system, location, or cloud region. This copy allows for disaster recovery, data migration, and offsite backups.

Non-transportable volume snapshot

A point-in-time copy of data that's confined to the specific storage system or location where it was created. It's primarily used for local data recovery in that system.

5.3.3.6.2 Restores

Restoring a Postgres backup

You can restore using:

- The Hybrid Manager (HM) console
- The API
- The CLI

Restoring using the HM console

1. Select the **Backups** tab to view existing backup that you can restore from.
2. Select a backup to restore from.
3. Select a cluster type to restore to.
4. Optionally, configure the restore:

- In **Cluster Settings**, provide a unique name for the restored cluster.
- For **Restore Type**, choose **Fully Restore** or **Point In Time Restore**. If you choose **Point In Time Restore**, use the date picker to select a point in time to restore to.
- For **DB Configuration** and **Additional Settings**, select your preferred database configuration.

1. Select **Restore Cluster**. The restored cluster will appear in the **Clusters** list in the console.

Restoring using the API

To restore using the API, you need:

- A JSON file to specify the restore configuration.
- A shell script that uses curl to call the JSON file.

> **Tip**
>
> You can get the file templates for point-in-time restores in the EDB GitHub repository . The templates for restores have a *pitr_* prefix.

To restore to a point in time using the API:

1. In your JSON file, supply the following parameters:

`projectID` : Obtain the `projectID` from the URL of your cluster. For example, if the URL is https://portal/projects/prj_6EOsaHpfS8RLshWH/clusters, the `projectID` is `prj_6EOsaHpfS8RLshWH` .

`source_cluster_id` : Get the `source_cluster_id` from the URL of your cluster. For example, if the URL is https://portal/projects/prj_6EOsaHpfS8RLshWH/clusters/p-nubw4gnbv9>, the `source_cluster_id` is `p-nubw4gnbv9` .

`point_in_time` : Provide a timestamp value for the point in time that you want to restore the cluster to.

Supply additional information for the other configuration parameters as appropriate for the cluster you want to restore. For example:

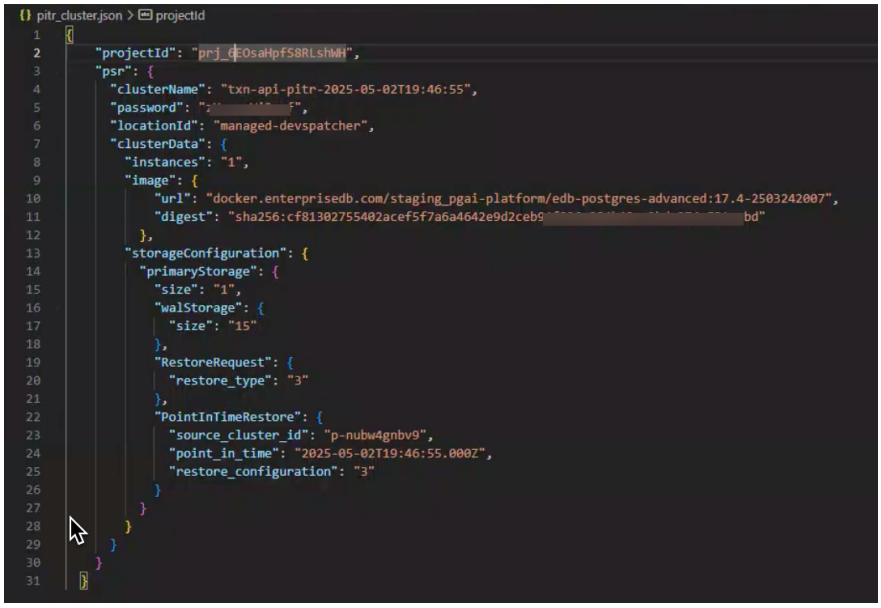
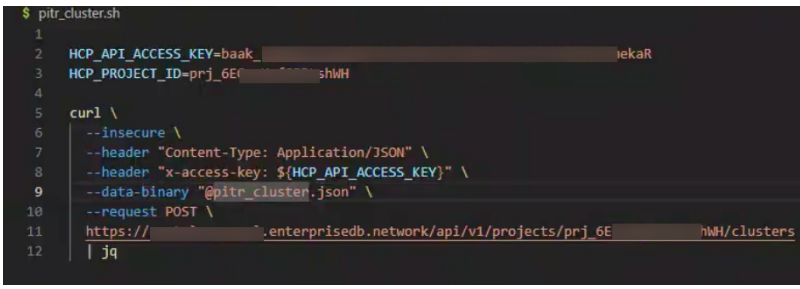


Fig. 7: PGAIHM Restore via API JSON file

1. Create a shell script to call the JSON file. For example:



2. Execute the shell script. If the script returns a clusterID as output, the restored cluster will appear in the **Clusters** list in the console. For example:

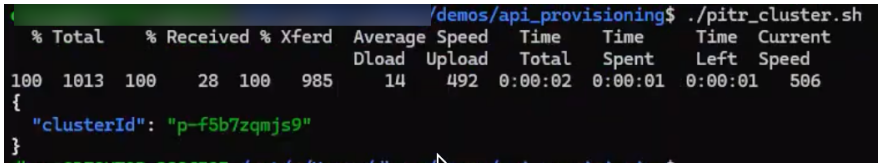


Fig. 8: PGAIHM Restore via API shell file conf

Restoring using the CLI

You can restore to a point in time using the CLI. To restore with the CLI, you need:

- A YAML file to specify the restore configuration.
- To use `edbctl` at the CLI to call the YAML file.

1. In your YAML file, supply the following parameters:

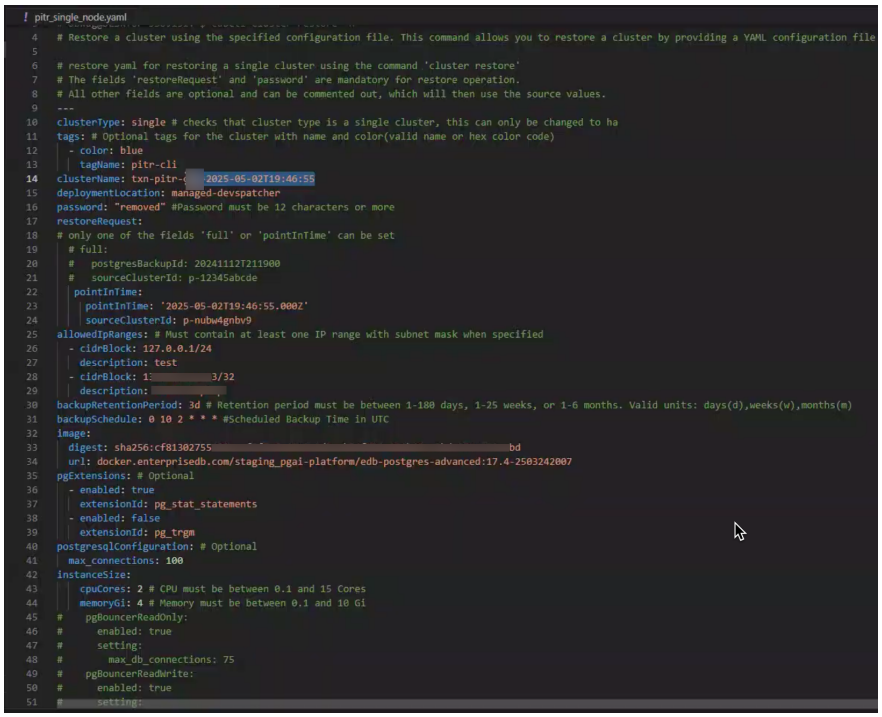


Fig. 9: HM Restore via CLI YAML file

> **Tip**
>
> For more information on the parameters you can use with the restore command, use *edbctl cluster restore -h* in the CLI to view the help.

1. Execute the CLI command by pointing it to your YAML file. A message is returned to indicate the cluster restore started. For example:

```
$ edbctl cluster restore -F $DEMOS/cli_pitr/pitr_single_node.yaml
```

Fig. 10: PGAIHM Restore via CLI YAML file

5.3.3.7 EDB Hybrid Manager Chat Agent

The EDB Hybrid Manager (HM) **Chat Agent** is a conversational assistant that helps you operate HM in plain language. Ask it about your estate, provision and inspect Postgres clusters, look up active alerts, troubleshoot performance, and consult the HM documentation — all without leaving the HM console.

5.3.3.7.1 Supported operations

Chat Agent covers estate browsing, cluster lifecycle management, monitoring and alerts, configuration recommendations, cluster health reports, Oracle-to-Postgres SQL translation, Data Migration Service status, and documentation Q&A — all scoped to your HM permissions.

For the complete catalog with example prompts and scope boundaries, see *What the Chat Agent can do* .

5.3.3.7.2 Model-dependent capabilities

Chat Agent's effective capability scales with the strength of the model your administrator configures. A small model answers HM documentation questions. Stronger models add estate browsing, monitoring, Postgres cluster lifecycle management, cluster health reports, and schema design recommendations. See *Model strength tiers* for guidance on model selection.

5.3.3.7.3 Documentation scope

Chat Agent answers HM questions from a versioned documentation snapshot shipped with each release. For details on citation behavior and the knowledge cutoff, see *HM documentation Q&A* .

5.3.3.7.4 Next steps

- *What the Chat Agent can do* — full capability catalog grouped by area, with example prompts.
- *Configuring the Chat Agent* — for HM administrators setting up the inference service and choosing a model.
- *Chat Agent security and data handling* — authorization model, audit, and how your data is treated.
- *Troubleshooting Langflow in Hybrid Manager* — common issues and how to resolve them.

What the Chat Agent can do

Chat Agent helps you operate Hybrid Manager (HM) in plain language, grouped by HM area. Available capabilities depend on the configured model's strength tier — see *Model strength tiers* . All capabilities respect your HM permissions and project scope — see *Chat Agent security and data handling* .

HM documentation Q&A

Answer conceptual or how-to questions about HM, including model serving, Lakehouse Analytics, and migrations. Answers come from a snapshot of the official HM documentation that ships with each HM release. Chat Agent cites the pages it read. If Chat Agent can't find an answer in the bundled snapshot, it says so and points you to the official EDB Hybrid Manager documentation site, where you can check for content added after this release.

Try:

- *How are log files controlled in HM?*
- *What are commit scopes in PGD and when should I use them?*

Manage estate, projects, users, and roles

Browse and manage the HM estate, projects, organization and project users, role assignments, and activity feeds. You can't delete users or configure identity providers with Chat Agent — use *Configuring your IdP in the HM console* for IdP setup.

Try:

- *List the projects.*
- *Add the ``Editor`` role to ``alice@example.com`` on project ``payments`` .*

Postgres cluster lifecycle management

List, inspect, create, delete, and restore Postgres clusters in these topologies: Single Node, High Availability (HA), and Distributed High Availability (DHA). Restore supports point-in-time recovery (PITR) and targeted-database restore.

Chat Agent inspects backups and lists supported Postgres image versions. Long-running actions run as background tasks — Chat Agent returns a task reference and can check progress.

Try:

- *I want to create a small Postgres cluster for development.*
- *Restore cluster ``orders-prod`` from the latest backup, and name the new cluster ``orders-restored``.*

Cluster monitoring, alerts, and observability

Get current or historical values of standard cluster metrics. Available metrics cover CPU, memory, storage, I/O, throughput, network, connections, transactions, query rate and latency, buffer cache hit ratio, top databases, write-ahead logs, and checkpoints. Render metrics over time as charts. View active alerts per cluster or per project. Chat Agent mirrors the **Cluster Monitoring** page in the HM console. It doesn't create or modify dashboards, acknowledge alerts, or silence alerts — use the HM console for those.

Try:

- *Plot CPU usage on ``customers-prod`` over the past 24 hours.*
- *How is storage usage on my cluster ``customers-prod`` ?*

Kubernetes workload observability

Inspect the Kubernetes workloads behind your clusters through HM's observability stack. Query pod and namespace metrics from Prometheus, read logs through Loki, and list the available Grafana data sources.

Try:

- *Tail the recent errors in namespace ``p-0hrzzicgpq-a``.*

Recommendations and advisories (read-only)

Get configuration, indexing, statistics, and security recommendations for a cluster, plus aggregated health scores across a project. Chat Agent surfaces the recommendations for review — apply them from the HM console.

Try:

- *Are there any pending index recommendations on ``orders-prod`` ?*
- *Summarize the security recommendations for cluster ``orders-prod``.*

Cluster health reports

List, fetch, analyze, and compare cluster health reports. HM generates one report per cluster per day — newly created clusters have no reports for the day of creation.

Try:

- *Compare yesterday's and today's health reports for ``customers-prod`` and tell me what changed.*

Schema design recommendations

Get table-level recommendations for data types, normalization, indexing, partitioning, and constraints. Chat Agent reads the table's structure and the top resource-consuming queries recorded against it (from `pg_stat_statements`), then applies a Postgres and EDB best-practices catalog. With no recorded query activity, recommendations fall back to schema-only fixes from the table definition.

Try:

- *Suggest schema improvements for table ``public.orders`` on ``customers-prod``, database ``app``.*

Browse Lakehouse Analytics catalogs (read-only)

List and inspect the Iceberg and Lakehouse catalogs registered in your projects. Create, update, and delete catalogs from the HM console.

Try:

- *List the Iceberg catalogs.*

AI Factory inventory (read-only)

Browse the AI Factory resources in your projects — registered models, inference services, knowledge bases, pipelines, and flows. Chat Agent lists and inspects these resources. Create and configure them from the HM console.

Try:

- *List my inference services*
- *List my knowledge bases in project ``payments``.*

Usage analytics

Review aggregated resource-usage analytics for your organization and projects — the compute hours, memory, and storage your Postgres clusters consume — including period-over-period comparison. Usage covers resource consumption only, not cost.

Try:

- *How much compute and storage did we use this month across the org?*
- *Show the resource usage for project ``payments`` this month, broken down by cluster.*

Data Migration Service (read-only)

View existing migrations, sources, destinations, table mappings, error history, and assessment reports. To start, pause, resume, or delete migrations, use the HM console — Chat Agent points you to the right page.

Try:

- *Show me the failing migrations in project ``migrations-prod`` and the most recent error for each.*

Oracle-to-Postgres SQL translation

Translate Oracle SQL to EDB Postgres-compatible SQL. Useful when preparing queries or schema definitions ahead of a migration from Oracle.

Try:

- *Translate this Oracle SQL: ``SELECT NVL(name, 'unknown') FROM customers ...``*

General constraints

These rules apply to every capability:

- **HM-only scope.** Off-topic questions (general knowledge, math, other products) are out of scope.
- **Confirmation gates.** Create, update, delete, drop, DDL, and DML operations require an explicit *yes* before execution.

- **Permission-honest.** On a permission-denied error, Chat Agent stops and reports the error rather than retrying or routing around it.
- **Status comes from a status check.** Listing a resource doesn't imply it's healthy or active — Chat Agent calls a status endpoint when needed.

For the security model behind these guarantees, see *Chat Agent security and data handling* .

See also

- *Configuring the Chat Agent* — model configuration, supported providers, and tier selection.
- *Chat Agent security and data handling* — the authorization model, audit, and data handling behind these capabilities.
- *Troubleshooting Langflow in Hybrid Manager* — common issues and how to resolve them.

Configuring the Chat Agent

The EDB Hybrid Manager (HM) Chat Agent is delivered with HM and powered by a chat-completion model that you provide through an HM-managed inference service. As an HM administrator, you create or register an inference service for that model. After that, anyone in your organization can use Chat Agent from the HM console.

This page is for HM administrators. To learn how to use Chat Agent, see the *Overview* .

Before you begin

Confirm these prerequisites:

- HM is installed and reachable.
- The **AI scenario** is enabled in your HM deployment. Chat Agent is part of the AI scenario, and the sparkle icon stays hidden when the AI scenario isn't installed. To enable it, make sure `ai` is listed under `scenarios` in your `HybridControlPlane` manifest. See *HM installation scenarios* for details.
- HM AI Factory is available, with Model Serving and the Model Library reachable from the HM console. See *Model Serving* and *Step 2: Access Model Library* for the AI Factory documentation.
- You have the **AI Model Manager** organization role. This role allows you to register AI models, apply model functions, and create inference services. See *User role permissions by use case* for the full role matrix.
- You have the credentials and connection details for at least one chat-completion model — either a third-party API or a self-hosted endpoint. See *Supported model providers* .
- The chat-completion model supports a **context window of at least 128K tokens**. Chat Agent loads skill instructions, tool schemas, and conversation history on every turn — models with smaller context windows don't have enough context.

> **Important**
>
> Chat Agent answers questions about your live HM estate. Choose a provider whose data-handling policy matches your organization's requirements. Self-hosted models keep prompts and responses inside your infrastructure.

Chat Agent is powered by a chat-completion model served through an HM-managed inference service. To make a model available to Chat Agent, go to **Estate > Inference Services** in the HM console, open the **Quick Actions** menu, and choose one of the following two paths depending on where your model runs:

- **Create Local Inference Service** — deploy and serve a model from your HM local **Asset Library**. Recommended when you want to keep prompts and responses inside your infrastructure.

- **Register External Inference Service** — point HM to a model endpoint running outside your cluster. Examples include a third-party cloud API (OpenAI, Anthropic, Gemini) or a self-hosted endpoint on another HM cluster. Recommended when you already operate the model elsewhere and want HM to use it.

Creating a local inference service

Add a model to your HM **Asset Library**, then create the inference service from that entry.

1. In **Asset Library > Models**, select **+ Add New Model** and fill in the model details. Add ``**chatbot-gen-content**`` to the **Functions** field. For the full form reference, see *How to deploy AI Models from the Model Library* .
2. Go to **Estate > Inference Services**.
3. From **Quick Actions**, select **Create Local Inference Service** (or the **Create Local Inference Service** button in the empty state).
4. Pick the model you added in the previous step and complete the deployment form.
5. Submit and wait until the inference service reaches the **Ready** state.
6. (Optional) To customize Chat Agent behavior for this model, apply additional model functions. See *Configuring model functions* .

The next time a user opens Chat Agent, the model is listed in the model selector.

Registering an external inference service

Point HM to a model endpoint running outside your cluster — such as OpenAI's API, an Anthropic-compatible endpoint, a Google Gemini endpoint, or a self-hosted endpoint on another HM cluster.

1. Go to **Estate > Inference Services**.
2. From **Quick Actions**, select **Register External Inference Service**.
3. Provide the connection details for the external endpoint. Required fields are the URL, the API protocol (OpenAI v1, Anthropic Messages, or Google Gemini), and API key.
4. Confirm the ``**chatbot-gen-content**`` function is applied.
5. Submit and wait until the registration reaches the **Ready** state.
6. (Optional) To customize Chat Agent behavior for this model, apply additional model functions. See *Configuring model functions* .

For more on external services, see *Model Serving* in the AI Factory documentation.

Configuring model functions

This optional step is referenced at the end of *Creating a local inference service* and *Registering an external inference service in HM* . These functions change Chat Agent behavior for the model they're applied to. Apply or update them in **Asset Library > Models** by editing the model's **Functions** field:

Function	Value	Purpose
chatbot-default-thinking-l	*low*, *medium*, or *high*	Sets the model's default reasoning effort.
chatbot-model-tier	*S*, *M*, *L*, or *XL*	Overrides the auto-detected strength tier.

Set `chatbot-default-thinking-level` only on models that support thinking or reasoning modes. If you don't set it, the model uses its own default thinking level (typically `medium` , per the model's specification). That default

works well for most tasks. Leave the function unset unless you see inconsistent Chat Agent behavior on complex tasks or you need faster responses. Raise it to `high` for more complex tasks, or lower it to `low` to speed up responses.

> **Warning**
>
> Some reasoning models don't accept a per-request thinking-level setting. On those models, applying *chatbot-default-thinking-level* causes the model to return an error and Chat Agent stops responding. Confirm the model accepts your chosen level before setting this function.

Verifying your configuration

1. In the HM console, sign in as a user that belongs to at least one project.
2. Select the sparkle icon at the top right.
3. Confirm the panel opens with the header **HM AI** and the placeholder **Ask me anything…** in the input.
4. If you registered more than one model, select the **model** picker and confirm your model appears.
5. Send a quick prompt, such as: *What can you do?*
6. Confirm a streamed response begins within a few seconds.

If the sparkle icon is missing or the panel can't be opened, see *Troubleshooting Langflow in Hybrid Manager* .

Accessing the Chat Agent

Chat Agent is available on every authenticated page in the HM console.

1. Look for the rainbow sparkle icon at the top right of the HM header.
2. Select the icon to open Chat Agent.

If the icon is missing or disabled, see *Troubleshooting Langflow in Hybrid Manager* .

Chat Agent opens in two layouts:

- **Full screen** — the default. The conversations list appears on the left.
- **Floating panel** — a compact, draggable panel docked at the bottom right of the HM console. Use the expand or shrink button to switch between modes.

Using the Chat Agent

Ask a question

1. Select **Model** from the picker if your administrator registered more than one. The model determines which capabilities are available — see *Model-dependent capabilities* .
2. Type your question in the **Ask me anything** input and press **Enter**.
3. Use **Shift+Enter** to add a line break within your message.
4. Chat Agent streams a response in the chat window. Documentation answers include inline citations so you can open the source page directly.
5. Select the stop button to interrupt generation early.

If Chat Agent needs more information before completing your request — for example, which project to target or a connection password — it pauses and presents a structured form. Fill in the form to continue, or cancel it (or type an unrelated message) to dismiss it.

Use page context

When you open Chat Agent from a cluster, project, or other resource page, it picks up that page's context automatically. The project and cluster IDs appear as small tags above the input, so you can ask about that resource without naming it — for example, *Show me the active alerts* applies to the cluster you're currently viewing.

Manage conversations

Each conversation is saved to your HM organization so you can return to it later. Select **New Chat** to start a fresh conversation, or open the sidebar with the menu icon to return to an earlier one.

Chat Agent gives each new conversation a short auto-generated title. From the sidebar you can also **Rename** a conversation or **Delete** one — deletion can't be undone.

For conversation privacy details, see *Conversation storage and isolation* .

Recommendations

- Start with a Large model — Large is the lowest tier that covers Postgres Cluster lifecycle management with structured forms and the full set of Medium-tier capabilities. `gemini-2.5-pro` , `gpt-5` , and `gpt-oss-120b` sit in this tier.
- Pick a self-hosted model if your prompts contain regulated or proprietary data — this choice keeps Chat Agent prompts and responses inside your infrastructure.
- Register more than one model to let users pick a tier per conversation — for example, register a Small model for documentation lookups and a Large model for cluster-creation flows.

Supported model providers

Chat Agent communicates with models through three API protocols. Any provider — third-party or self-hosted — that speaks one of these protocols works. EDB does not build or ship AI models. All HM-hosted models are third-party models downloaded by HM from Hugging Face or NVIDIA, such as `gpt-oss-20b` and `nvidia/llama-3.3-nemotron-super-49b-v1.5` .

Protocol	Compatible providers and runtimes
OpenAI v1	OpenAI, Azure OpenAI, NVIDIA NIM containers, Amazon Bedrock (via OpenAI-compatible gateway), vLLM and other OpenAI-compatible self-hosted servers
Anthropic Messages	Anthropic API, Anthropic-compatible gateways and proxies that preserve the Messages API surface
Google Gemini	Google AI Studio (Gemini API), Vertex AI with the Gemini API surface

Model strength tiers

Chat Agent classifies every registered model into a **strength tier** — Small, Medium, Large, or Extra Large — based on the model's capability. A model's tier determines which Chat Agent skills are available when that model is selected.

Strength tiers

The four tiers correspond to approximate model scale and capability:

- Small — roughly 7–30B parameter class.
- Medium — roughly 30–100B parameter class.

- Large — `gpt-4.1` , `gemini-2.5-pro` , `gpt-5` , and `gpt-oss-120b` class.
- Extra Large — current flagship models: the Claude 4.5+ family (including `claude-haiku-4-5`), `gpt-5.2` and later, `gpt-5.3-codex` , `gemini-3.1-pro-preview` , and comparable frontier models.

A model that isn't on the recognized list below defaults to Small.

Recognized models

Model	Tier
gemini-2.5-flash-lite	Small
gemini-3.1-flash-lite	Small
gpt-4.1-nano	Small
gpt-4o	Small
gpt-4o-mini	Small
gpt-5-nano	Small
gpt-oss-20b	Small
nvidia/llama-3.3-nemotron-super-49b-v1	Small
nvidia/llama-3.3-nemotron-super-49b-v1.5	Small
nvidia/NVIDIA-Nemotron-3-Nano-30B-A3B-BF16	Small
nvidia/Nemotron-3-Nano-Omni-30B-A3B-Reasoning-BF	Small
nvidia/Nemotron-Cascade-2-30B-A3B	Small
gemini-2.5-flash	Medium
gpt-4.1-mini	Medium
gpt-5-mini	Medium
gpt-5.4-nano	Medium
o4-mini	Medium
qwen/qwen3-next-80b-a3b-instruct	Medium
qwen/qwen3-next-80b-a3b-thinking	Medium
gemini-2.5-pro	Large
gemini-3-flash-preview	Large
gpt-4.1	Large
gpt-5	Large
gpt-5.1	Large
gpt-5.4-mini	Large
gpt-oss-120b	Large
o3	Large
nvidia/llama-3.1-nemotron-ultra-253b-v1	Large
nvidia/NVIDIA-Nemotron-3-Super-120B-A12B-BF16	Large
qwen/qwen3.5-122b-a10b	Large
claude-haiku-4-5	Extra Large
claude-sonnet-4-5	Extra Large
claude-sonnet-4-6	Extra Large
claude-opus-4-1	Extra Large
claude-opus-4-5	Extra Large
claude-opus-4-6	Extra Large
claude-opus-4-7	Extra Large
gemini-3.1-pro-preview	Extra Large
gpt-5.2	Extra Large
gpt-5.3-codex	Extra Large
gpt-5.4	Extra Large
gpt-5.4-pro	Extra Large

continues on next page

Table 59 – continued from previous page

Model	Tier
gpt-5.5	Extra Large
gpt-5.5-pro	Extra Large
deepseek-ai/deepseek-v4-flash	Extra Large
deepseek-ai/deepseek-v4-pro	Extra Large
meta/llama-4-maverick-17b-128e-instruct	Extra Large
minimaxai/minimax-m2.7	Extra Large
qwen/qwen3-coder-480b-a35b-instruct	Extra Large
qwen/qwen3.5-397b-a17b	Extra Large
z-ai/glm5	Extra Large
z-ai/glm-5.1	Extra Large

The following aliases are recognized as the same canonical model:

- Anthropic models on Amazon Bedrock — cross-region prefixes (`us.` , `eu.` , `jp.` , `au.` , `global.`) and provider-prefixed variants (`anthropic.<model>-<date>-v1:0`).
- OpenAI `gpt-oss-20b` and `gpt-oss-120b` — provider-prefixed variants `openai/<model>` , `openai.<model>-1:0` , `gpt-oss:<size>` .

Assigning a tier to an unrecognized model

Models whose names don't match any row in the table above (and aren't covered by the alias rules) default to Small. Many of the more advanced skills are unavailable as a result.

To override this setting, add the ``**chatbot-model-tier**`` function to the model in the Asset Library with one of these values:

Function value	Tier
S	Small
M	Medium
L	Large
XL	Extra Large

Use the value that best matches your model's actual capability. After saving, Chat Agent honors the tier you set instead of defaulting to Small.

Skills available at each tier

A skill is available when the selected model's tier is at or above the skill's minimum tier. Higher tiers add capabilities without removing the existing ones.

Skill / capability	Minimum tier
HM documentation Q&A (concepts, architecture, how-to, with citations)	Small
Browse and manage estate, projects, organization users, project users, organization roles, and project roles	Medium
Project and organization activity feeds	Medium
Iceberg and Lakehouse catalog inventory (list and inspect, read-only)	Medium

continues on next page

Table 61 – continued from previous page

Skill / capability	Minimum tier
Kubernetes workload observability — Grafana data sources, Prometheus queries, Loki queries	Medium
Cluster realtime and historical metrics, charts, and available metric metadata	Medium
Active alerts (per cluster, per project), aggregated health scores	Medium
Configuration, index, statistics, and security recommendations	Medium
Data Migration Service status — sources, destinations, mappings, errors, assessments	Medium
Oracle-to-Postgres SQL translation	Medium
Resource-usage analytics (compute, memory, and storage consumption)	Medium
Postgres Cluster lifecycle management	Medium
AI Factory inventory — models, inference services, knowledge bases, pipelines, and flows	Large
Image library and deployable Postgres versions per location	Large
Read-only database, schema, and cluster-configuration inspection	Large
Cluster health reports — list, fetch, analyze, compare	Large
Postgres Cluster lifecycle management with structured follow-up questions	Large
Schema design recommendations driven by top running queries	Extra Large

A Small model offers only HM documentation Q&A. Postgres Cluster lifecycle management starts at the Medium tier, with structured follow-up questions added at Large and above. Cluster health reports need a Large model, and schema design recommendations need an Extra Large model. For a full breakdown of what each tier supports, see *What the Chat Agent can do* .

Disabling Chat Agent for an organization

To remove Chat Agent from your HM console, remove the `chatbot-gen-content` function from every model that has it. Once no Chat Agent provider is registered, the sparkle icon is shown but disabled, with a tooltip directing users to configure a model service.

To remove the sparkle icon entirely, disable the AI scenario at install time. See *HM installation scenarios* .

Next steps

- *What the Chat Agent can do* — the capability catalog users see, organized by area.
- *Chat Agent security and data handling* — the authorization model, audit, and data treatment.
- *Troubleshooting Langflow in Hybrid Manager* — common configuration issues and how to resolve them.

Chat Agent security and data handling

The Hybrid Manager (HM) Chat Agent is a convenience layer over the same APIs that power the HM console. It doesn't introduce a new privilege model and doesn't act on its own authority.

Authorization model

Chat Agent isn't a privileged service account. Your HM identity and project context travel with every request it makes. Chat Agent runs as the authenticated user, against the same APIs you can already call. It can't see resources you can't see, or perform actions you don't have permission to perform. Requests without a verified user identity are rejected.

When your organization removes a permission, revokes a project membership, or disables a user, those changes apply to Chat Agent immediately. No Chat Agent-specific configuration is needed.

Secret handling

When Chat Agent needs a secret as input, it collects the value through a structured form with a hidden input. An example is a database password during Postgres Cluster lifecycle management. The form-collected value stays inside HM. HM never sends it to the chat-completion model or includes it in the conversation transcript.

This guarantee applies only when the configured model is Large or Extra Large tier. Those tiers ask structured follow-up questions for inputs like passwords. At lower tiers, Chat Agent uses free-form prompts. Any secret you type into chat then passes through the model with the rest of your prompt. See *Skills available at each tier* .

Audit logging

Every action Chat Agent performs is recorded in the HM activity log, alongside HM console and API activity. See *Activity log* .

Conversation storage and isolation

Conversations are stored inside your HM organization so you can resume them later. Each conversation is private to its creator. No one else, including organization owners and project administrators, can list, load, or download its messages through Chat Agent or its API. The HM activity log records actions, not the prompts or responses.

You can delete a conversation from Chat Agent's conversations sidebar and rename the auto-generated subject. You can't recover a conversation after you've deleted it.

Ad-hoc SQL queries you run from a cluster detail page aren't stored — refreshing or closing the page clears them.

Prompts and the model provider

Your prompts and Chat Agent's responses pass through the chat-completion model your administrator configured. With a self-hosted model, both stay inside your infrastructure. With a third-party model, they leave your infrastructure to reach the vendor — choose providers whose data-handling and retention policies match your organization's requirements.

Treat the model as part of your data path. Sensitive values collected through structured forms stay inside HM — see *Secret handling* . Anything else you type into chat reaches the model.

Knowledge cutoff

Chat Agent answers HM questions from a versioned documentation snapshot shipped with each release. For how the snapshot works, citation behavior, and knowledge cutoff handling, see *HM documentation Q&A* .

See also

- *What the Chat Agent can do* — full capability catalog with scope boundaries.
- *Configuring the Chat Agent* — model configuration, including self-hosted versus third-party provider considerations.

- *Troubleshooting Langflow in Hybrid Manager* — common issues and how to resolve them.

Chat Agent schema design recommendations

Hybrid Manager (HM) Chat Agent can analyze your table schemas and query workload and suggest improvements to your schema design. Recommendations are evidence-based and grounded in Postgres and EDB best practices.

Recommendations are delivered conversationally — ask Chat Agent about a specific table and it returns actionable suggestions. Schema design recommendations don't appear in the **Recommendations** tab; they're available only through Chat Agent.

Prerequisites

- The target cluster is registered with HM. Both HM-managed and self-managed clusters are supported.
- Monitoring is enabled on the cluster. Chat Agent reads query statistics and schema metadata through HM's monitoring data, which is available for both managed and self-managed clusters. Without monitoring data, recommendations fall back to schema-only analysis.

Recommendation types

Chat Agent focuses on DDL-level improvements for both managed and self-managed clusters — changes to how your data is defined and stored, rather than to the queries that access it. It applies a Postgres and EDB best-practices catalog, so the suggestions you get back depend on the table and its workload. Common areas include:

- Normalization and structure — flags issues such as repeating groups, partial dependencies, or over-denormalization that may be affecting performance or maintainability.
- Data types — suggests more appropriate types for a column, for example `TIMESTAMPTZ` over `TIMESTAMP` , `NUMERIC` for money, or `TEXT` over `CHAR(n)` /`VARCHAR(n)` .
- Constraints and keys — recommends primary keys, foreign keys, `NOT NULL` , `CHECK` , and `UNIQUE` constraints where they're missing or could be tightened.
- Indexing — identifies missing indexes for your access paths (such as unindexed foreign-key columns) and the appropriate index type, and flags unused or redundant ones.
- Partitioning — identifies tables that would benefit from range, list, or hash partitioning based on query patterns and data distribution.
- Workload-specific tuning — adjustments for update-heavy, insert-heavy, or upsert-oriented tables, such as `fillfactor` settings or separating hot and cold columns.
- EDB extensions — where it adds value, recommends EDB extensions for the use case, for example the EDB Advanced Storage Pack table access methods for analytical or append-only workloads.

For curated, auto-applicable index recommendations, see *Recommendations and advisories (read-only)* .

How schema recommendations work

When you ask for schema recommendations, Chat Agent:

- Reads the table's DDL (column definitions, constraints, indexes, and storage settings).
- Retrieves the top queries running against the table. By default it looks at the last 6 hours; you can specify a different window in your prompt.
- Applies a Postgres and EDB best-practices catalog to the schema and query data.
- Returns specific, evidence-based recommendations with an explanation for each.

If there's no query workload in the specified window, Chat Agent falls back to schema-only recommendations derived from the table definition alone.

Example prompts

- *Suggest schema improvements for table ``public.orders`` on ``customers-prod``, database ``app``.*
- *Review the schema for ``public.line_items`` on ``orders-prod`` using the last 24 hours of query data.*
- *Are there any partitioning opportunities on ``public.events`` in ``analytics-db``?*

Limitations

- Recommendations cover DDL-level improvements only. Query-level (DML) improvements are out of scope.
- Schema recommendations aren't auto-applied. Unlike index recommendations, you must implement them manually or through your own deployment pipeline.
- Recommendations aren't displayed in the **Recommendations** tab or included in the cluster health score.
- Accuracy improves with more query data. For newly created tables or clusters with little workload history, recommendations are based on schema structure alone.

Troubleshooting the Chat Agent

Use these fixes to resolve common Hybrid Manager (HM) Chat Agent issues for users and administrators. If you don't see your issue here, contact EDB Support with the request-tracing ID shown in the HM console.

The sparkle icon doesn't appear

Problem: the rainbow sparkle icon is missing from the HM header.

Cause: the AI scenario isn't installed. Chat Agent is part of the AI scenario.

Fix: check with your HM administrator that the AI scenario was selected at install or upgrade time. See *HM installation scenarios* .

The sparkle icon is disabled with a tooltip

Problem: the icon is visible but grayed out. Hovering shows a tooltip such as *The administrator needs to configure an inference service to enable the Chat Agent* — or another reason.

Cause 1: no chat-completion model is registered as a Chat Agent provider.

Fix 1: the administrator must register a chat-completion model and apply the `chatbot-gen-content` function. See *Configuring the model* .

Cause 2: the registered inference service isn't ready.

Fix 2: open **Estate > Inference Services** and confirm the **Status** column shows **Ready**. Chat Agent only surfaces inference services in the **Ready** state — services in any other state (for example, **Unknown**) can't be used by Chat Agent.

Cause 3: the Chat Agent service is unavailable.

Fix 3: the tooltip includes a reason. Wait a moment and try again, or check with your administrator that the supporting HM workloads are healthy.

Generation stalls or fails

Problem: the **Waiting…** placeholder bubble stays for a long time, or the response ends with `Request failed, please try again!`

Cause 1: the configured model is unreachable or has reached a usage limit.

Fix 1: try a different model from the model selector if more than one is registered. Otherwise, check the model server's status in **AI Factory > Model Serving** (*Monitor InferenceService*).

Cause 2: the prompt is too long. Each user message is capped at 10,240 characters.

Fix 2: trim the prompt and retry.

Cause 3: network egress is blocked.

Fix 3: for third-party models, confirm that the HM cluster has outbound access to the provider's API.

Cause 4: the model rejects the configured `chatbot-default-thinking-level` . Not every reasoning model accepts a per-request thinking-level setting, so the turn fails when one is passed.

Fix 4: remove `chatbot-default-thinking-level` from the model's **Functions** field to fall back to the model's default, or switch to a model that documents support for per-request thinking-level configuration. See *Configuring model functions* .

Chat Agent doesn't answer my question

Problem: Chat Agent says it can only help with HM topics.

Cause: Chat Agent answers only HM-related questions. Off-topic questions — general knowledge, math, creative writing, and other products — are out of scope.

Fix: rephrase your question to be about your HM estate, an HM feature, or HM documentation.

Chat Agent says it can't perform an action

Problem: Chat Agent reports a `403` or `not authorized` error and stops, or says you don't have permission to perform the action.

Cause: Chat Agent runs as your user account. If you don't have the role to perform the action in the HM console, Chat Agent gets the same answer.

Fix:

- Confirm with an administrator that you hold the right organization or project role.
- Check that you are a member of the project the action targets.
- For action types that require approval workflow (such as some automation tasks), wait for an administrator to approve.

See *Authorization model* .

Chat Agent doesn't see my cluster or project

Problem: Chat Agent says it cannot find a project or cluster you know exists.

Cause 1: you aren't a member of the project that owns the resource. Chat Agent only sees resources within projects you can access.

Fix 1: ask an administrator to add you to the project.

Cause 2: you misspelled the resource name. Resource names are unique within a project. Chat Agent resolves a name to an ID by listing the resources first, and a typo causes it to find no match.

Fix 2: copy the resource name from the HM console and retry.

Chat Agent says a topic post-dates its knowledge

Problem: Chat Agent says the topic you asked about is newer than its bundled documentation snapshot.

Cause: each HM release ships with a snapshot of the official HM documentation, with a stamped knowledge cutoff date. Chat Agent reports when the documentation snapshot does not contain an answer.

Fix: refer to the official EDB Hybrid Manager documentation site for newer information. To benefit from a fresher in-product snapshot, upgrade HM to a more recent release.

Chat Agent keeps asking follow-up questions

Problem: Chat Agent keeps asking for more information before completing your request.

Cause: some Chat Agent flows follow a predefined sequence of inputs to complete a specific task — Postgres Cluster lifecycle management is one example. When Chat Agent doesn't have all the required parameters, it keeps asking until it does, rather than guessing or proceeding with incomplete data.

Fix: fill in the requested information to continue, or cancel the form or type an unrelated message in chat — the question clears.

Charts don't render in the response

Problem: the response shows raw chart syntax or a malformed placeholder instead of a rendered chart.

Cause: chart generation requires the model to emit precise structured output. Smaller models often struggle with the syntax and produce charts the renderer can't display.

Fix: switch to a Large or Extra Large model from the model picker if your administrator registered more than one. Otherwise, ask the administrator to register a stronger model. See *Model strength tiers* .

Reduced capability

Problem: Chat Agent doesn't offer cluster creation, doesn't ask structured follow-up questions, or doesn't start the full interactive flow you've seen before.

Cause: Chat Agent capability scales with the strength of the configured model. A Small or Medium model handles fewer flows than a frontier model, and structured follow-up forms are only used when the model supports them.

Fix: select a stronger model from the model picker if your administrator registered more than one, or ask the administrator to register a stronger model. See *Model strength tiers* .

Contacting support

If the issue persists, capture these details before contacting EDB Support :

- HM troubleshooting information — open **About** from the HM console footer and select **Copy troubleshooting information** to capture the version, build, scenarios, organization, session, and platform details.
- The model and model provider in use, if known.
- A copy of the prompt you sent and the response you received.

5.3.3.8 Governance in Hybrid Manager

EDB Agent Governance is a standalone application that audits and governs how AI agents interact with your Postgres data — reconstructing agent sessions, inspecting every SQL statement an agent ran, and presenting the full audit trail. It works with several upstream log sources; Hybrid Manager (HM) is one of them.

This page covers the HM-specific integration only. For how to use the application itself — connecting data sources, reading the sessions list, inspecting session detail, and deploying it — see the EDB Agent Governance documentation .

5.3.3.8.1 What HM provides

When you point EDB Agent Governance at an HM instance, HM supplies the pieces the application needs to discover and read AI agent activity:

- **Project and cluster discovery** — the application auto-discovers the projects and Postgres clusters in your HM dataplane, so they appear as audit targets without manual configuration.
- **Log access through HM's Loki pipeline** — HM already collects Postgres logs into Loki as part of *Monitoring execution* . The application queries that same pipeline to retrieve the query logs AI agents generate.
- **Machine user API key authentication** — the application authenticates to HM with a machine user API key, so it reads only what that key is authorized to read.

5.3.3.8.2 Prerequisites

For an HM-managed cluster's AI agent activity to appear in EDB Agent Governance:

- The cluster has Postgres statement logging enabled (`log_statement = 'all'` or `log_min_duration_statement = 0`).
- AI agent workloads run through the Airman Model Context Protocol (MCP) server with tracing enabled, so each query is tagged and attributable to a session.
- You have a machine user API key for the HM instance. Create it in the HM admin portal.

5.3.3.8.3 Connecting to HM

In EDB Agent Governance, add an instance of type **HM**, supplying the HM base URL and the machine user API key. The application then discovers your projects and clusters and begins syncing session data. For the full procedure, see Connecting data sources .

5.3.3.8.4 Learn more

-

EDB Agent Governance — full product documentation.

-

Connecting data sources — registering an HM instance.

-

Securing access and handling credentials — authentication and credential handling.

- *Monitoring execution* — HM's Loki logging pipeline that the application reads from.

5.3.3.9 Hybrid Manager CLI - edbctl

edbctl is the dedicated command-line interface (CLI) for EDB Postgres AI Hybrid Manager (HM). It serves as the primary administrative tool for managing HM resources and executing sophisticated database cluster operations through a terminal environment.

By providing a unified entry point for both configuration and resource management, edbctl allows administrators to automate and streamline the lifecycle of database assets across diverse environments.

It is important to note that edbctl is distributed as a standalone package with its own independent versioning lifecycle. Consequently, its version numbers do not mirror those of the Host Manager (HM).

For detailed information about installation, compatibility, usage and more, see edbctl documentation .

5.3.3.10 Interacting programmatically with HM

System administrators and developers can use the EDB Postgres AI Hybrid Manager (HM) API to integrate directly with HM for management tasks. These tasks involve managing clusters - (provisioning, deprovisioning, and scaling), automating administrative operations, and handling migrations.

For more information, see API docs from the EDB Postgres AI Portal:

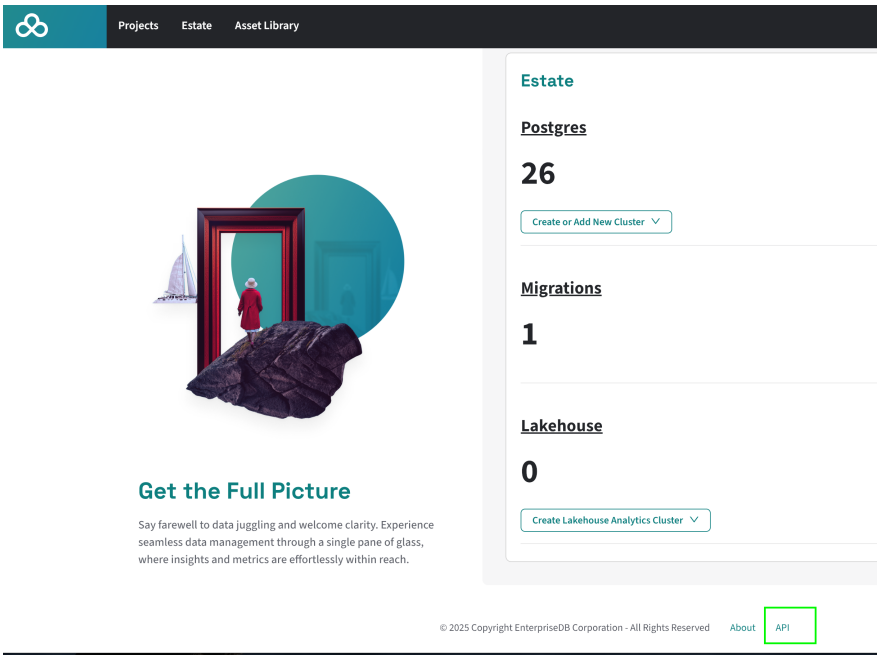


Fig. 11: API Docs

1. Go to your Hybrid Manager console URL and log in with your credentials.
2. Look for an `API` link at the bottom of the page.
3. Select the `API` link to open the interactive API explorer.

You can view all available endpoints, see required parameters, and even execute sample requests directly from your browser.

5.3.3.10.1 Access Key

An access key provides secure, programmatic access to the HM API. It enables users to authenticate and interact with HM services without requiring interactive login each time.

Key purposes:

- Authentication — Verifies the identity of the user or system calling the API.
- Automation — Allows scripts, tools, or applications to securely perform tasks such as provisioning clusters, scaling, or monitoring—without manual intervention.
- Access control — Ties actions to a specific user or machine, enabling audit trails and permission enforcement.
- Security — Avoids the need to embed usernames and passwords in automation scripts.

For more information, see *Create and manage Access key*

Access key

An access key provides authentication for users to interact directly with the Hybrid Manager (HM) API. Each access key is linked to a single user and is immutable after creation.

You can create an access key for a machine user or a normal user. The organization owner can manage the access key for a machine user. Normal users can manage their own access keys.

Each key must have an expiration period between 1 and 365 days, set at creation time. Once an access key expires, or if it's lost, then you must delete it and create a new one.

User type	Quota - max keys	Keys created and managed
Machine user	2	Two access keys, created while *Creating a machine user* . Optionally, can be individually added from the **Access keys** tab on the machine user's details page using the **Create New Key** option.
Normal user	1	One access key, *Tables in user-created schemas* and managed by the user from their home page.

An access key can be revoked from a user. Revoking the key doesn't affect the role or any other authentication process of the linked user.

Create your personal access key

1. Log in to the HM console.
2. From the menu next to your organization name at the top right, select **My Account**.
3. From the **My Account** page, select the **Access Keys** tab.
4. On the **Access Keys** tab, select **Create New Key**.
5. Enter a value for **Access Key Name**.
6. Enter a value for **Expiry days** in the range of 1 to 365.
7. Select **Create Key**.

The generated access key is provided to the user.

Copy this access key and save it in a secure location. The access key is available only when you create it. If you lose your access key, you must delete it and create a new one.

You can enable in-app inbox or email notifications so that you're alerted when your personal key is about to expire. For more information, see *Managing notifications* .

Manage access key

You can manage the access keys for a machine user on the **Users** page.

To view the details of an access key:

1. From the menu next to your organization name at the top right, select **User Management**.
2. Select **Edit** next to the user whose access key details you want to view.
3. View the **Access Keys** tab on the user's page. Each row provides the access key details:

- **Name** — Access key name.
- **Access Key** — Access key in half encrypted format.
- **Expiry at** — The expiry date for the access key.
- **Created at** — The creation date for the access key.

To delete a key, select **Delete** next to the key. You're prompted to confirm the deletion.

To create a key, select **Create New Key** at the top right of the **Access Keys** tab.

Use the access key

The HM API detects if the caller is using an access key in the request. If the caller uses the access key, the API checks:

- The validity of the access key.
- The organization of the user requesting access.
- The user permissions.

It then authorizes the access. For more information, see *Using the access key* .

Using Access Keys to access GenAI Builder

To interact with the GenAI Builder API, you must authenticate your requests using an access key. This key can be provided in the request header as either a bearer token or an `x-access-key` .

1. Generate an Access Key

Create an access key in Hybrid Manager via the UI or CLI.

Important: Store the key securely, as it won't be shown again after generation.

2. Use the Access Key in API Requests

The base URL for the GenAI Builder API endpoints is typically structured as `${BASE_URL}/launchpad/apps/genai-builder/api` . Here `${BASE_URL}` represents the domain or IP address of your Hybrid Manager instance.

You can use tools like `curl` to test your API access. The access key must be sent in the HTTP header in one of two ways:

- **As a Bearer Token:** `Authorization: Bearer ${access_key}`

- **As an x-access-key:** `x-access-key: ${access_key}`

Here's an example using the `x-access-key` header to retrieve a list of users:

```
curl -H "x-access-key: ${access_key}" \
     -k "${BASE_URL}/launchpad/apps/genai-builder/api/users"
```

Where,

- `${access_key}` is your actual access key.
- `${BASE_URL}` is your domain or IP address of your Hybrid Manager instance. **Note:** The `-k` option allows `curl` to work with self-signed TLS certificates. Remove it if you're using a trusted certificate.

Example API Response: Retrieving Users

Upon a successful request to the users endpoint as shown in the `curl` example above, you might receive a JSON response similar to this:


```
{
    "users": [
        {
            "user_id": "d9e03312-638d-52b6-8d6f-f5b99f5f2480",
            "created_at": "2025-06-25T19:27:32.642823Z",
            "updated_at": "2025-06-26T08:50:42.122521Z",
            "organizations": [
                "060e360b-0364-47b6-adc2-f62ac0feae41"
            ],
            "name": "owner@mycompany.com",
            "email": "owner@mycompany.com"
        }
    ]
}
```


Using the HM API

Using the HM API to integrate directly with HM for management activities such as cluster provisioning, deprovisioning, and scaling.

The API reference documentation is available from the EDB Postgres AI portal.

Prerequisite

Before calling API, create an access key. For more information, see *Access key* .

Call the API

To authenticate with the HM API, your application must include the retrieved access key in the x-access-key header of each HTTP request. For example:

```
curl --request GET \
  --url "https://portal.edb-hcp.com/api/v1/projects" \
  --header "x-access-key: $ACCESS_KEY"
```

Where the `ACCESS_KEY` variable is the access key's text copied from the console.

Example response:

```
{
  "data": [
    {
      "projectId": "string",
      "projectName": "string",
      "userCount": 0,
      "clusterCount": 0,
      "tags": []
    }
  ]
}
```

Using MCP tools

Hybrid Manager exposes a Model Context Protocol (MCP) server that allows AI agents and MCP-compatible clients to interact with the platform programmatically. The MCP server enables agentic workflows (such as automated migrations, cluster management, and observability queries) to be driven by large language models (LLMs) or AI pipelines.

The MCP server is self-documenting. When an MCP client connects to the entry point, it can automatically discover all available tools and their descriptions without requiring separate API reference documentation.

Connecting to the MCP server

The MCP server is available via the Portal API Gateway. Use the following entry point URL, replacing `<portal-url>` with your HM environment's portal URL:

```
https://<portal-url>/api/mcp
```

Requests to the MCP server must include the following headers:

Header	Value
Authorization	*Bearer <access-key>*
Accept	*application/json, text/event-stream*

Use your portal access key. You must have sufficient permissions. For information on how to create an access key, see *Access key* .

Available tool categories

The MCP server exposes tools across these functional areas:

- Cluster management — Create, view, scale, update, and delete Postgres clusters (PG, PGE, and EPAS) in Single Node, High Availability, Advanced High Availability, and Distributed High Availability topologies, manage cluster templates, backups, and maintenance windows.
- Observability and monitoring — Retrieve health scores, current and historical metrics, active alerts, logs, and Grafana dashboards.
- Recommendations — Get index, statistics, configuration tuning, and security recommendations for clusters.
- Migrations — Manage database migrations end-to-end, including assessment, schema conversion, Oracle SQL translation, and migration lifecycle (create, pause, resume, complete).
- Lakehouse and analytics — Manage catalogs, tiered tables, and data offload for lakehouse workloads.

- Agent Factory — Configure inference services, models, model server clusters, knowledge bases, pipelines, and Langflow-based AI flows.
- Identity and access — Manage access keys, machine users, identity providers (IdPs), and organization and project roles.
- Projects and organization — Create and manage projects, tags, locations, and view usage and activity logs.
- Database introspection — Browse schemas, tables, partitions, DDL, and analyze query performance directly through the MCP interface.

Pg Airman MCP .

5.3.3.11 Managing identity providers

Hybrid Manager (HM) supports external identity providers (IdPs) — such as LDAP, SAML, and OIDC — for authenticating users in the HM console.

- See *Configuring your IdP in the HM console* to set up and manage your IdP integrations.
- See *Rotating IdP connector encryption keys* to rotate the encryption key that protects your IdP connector secrets.

5.3.3.11.1 Configuring your IdP in the HM console

The identity provider (IdP) configuration workflow in the Hybrid Manager (HM) console automates the integration process. When you use this workflow, you don't need to manually modify the Helm chart or restart the workload. The workflow makes deployment consistent and reduces downtime.

> **Important**
>
> You must be an HM organization owner to configure an IdP in HM.

Configuring a SAML IdP

You can use any IdP that supports SAML 2.0, such as Okta, Microsoft Entra ID, Google Workspace, or Keycloak.

> **Important**
>
> When you begin adding a SAML identity provider in HM, the system generates unique **Assertion Consumer Service URL** and **Audience URI** values. These values change each time you start a new configuration. You must complete both the IdP configuration and the HM configuration in a single session without navigating away from the page.

Before you begin

1. Open your IdP admin console in a separate browser tab.
2. In HM, start the SAML IdP configuration to obtain the **Assertion Consumer Service URL** and **Audience URI**.
3. Copy these values to your IdP configuration immediately.
4. Complete the IdP setup and obtain the Single Sign-On (SSO) URL and certificate.
5. Return to the HM tab (keep it open) and complete the configuration.

Starting the SAML configuration in HM

1. In the HM console, select **Settings** > **Identity Providers**.
2. Select **Add Identity Provider**.
3. Select the **SAML** tab.
4. Copy the following values (you need these for your IdP):

- **Assertion Consumer Service URL**
- **Audience URI (SP Entity ID)**

> **Note**
>
> Don't close or navigate away from this page until you complete the HM configuration.

Configuring your identity provider

In a separate browser tab, create a new SAML 2.0 application in your IdP with the following settings:

1. Enter a name for the SAML integration.
2. Enter the **Assertion Consumer Service URL** from HM in the **Single Sign-On URL** field.
3. Enter the **Audience URI** from HM in the **Audience URI (SP Entity ID)** field.
4. Set the **NameID Format** based on your requirements. See *NameID formats* for details.
5. Configure attribute mappings to include user information in the SAML assertion.

Where: - **Username** is the user's unique identifier. - **Email** is the user's email address.

> **Note**
>
> Attribute names vary by IdP. Refer to your IdP's documentation for configuring SAML attribute statements. !!!

1. Save the integration.
2. Copy the **Single Sign-On URL** (also called SSO URL or Login URL) that your IdP provided.
3. Download the **Signing Certificate** that your IdP provided (as a `.pem` or `.cer` file).

Completing the SAML configuration in HM

Return to the HM browser tab and enter the following:

1. In the **Name** field, enter a unique name for this identity provider.
2. In the **Description** field, enter an optional description for this identity provider.
3. In **Single Sign-On URL**, enter the SSO URL from your IdP.
4. Under **Signature Certificate File**, upload the signing certificate from your IdP.
5. Under **NameID Format**, select the format that matches your IdP configuration.
6. Enter the values for **Username Attribute** and **Email Attribute**. Where:

- **Username Attribute** is the attribute name containing the username (for example, `name`).
- **Email Attribute** is the attribute name containing the email address (for example, `email`).

1. (Optional) To enable *Automatic role assignment with role mapping rules* , configure group attribute mapping: Where:

- **Groups Attribute** is the SAML attribute name containing the user's group memberships (for example, `groups` or `memberOf`).
- **Groups Delimiter** is the delimiter that separates multiple groups in the attribute value. Leave empty if your IdP sends groups as separate attribute values.

> **Note**
>
> Configure your IdP to include group membership in the SAML assertion. The exact configuration varies by IdP. Refer to your IdP's documentation for configuring group attribute statements. !!!

1. Select **Save**.

NameID formats

Select the NameID format that matches your IdP configuration:

Format	Description	Use case
Email Address	User's email address	Most common. Use when your IdP identifies users by email.
Persistent	A persistent, opaque identifier for the user	When the IdP assigns stable user identifiers that aren't email addresses.
Transient	A temporary identifier valid only for the current session	When you require privacy and can't share a persistent identifier.
X.509 Subject Name	The subject name from an X.509 certificate	When using certificate-based authentication.

> **Tip**
>
> If you're unsure which format to use, start with **Email Address** as it's the most commonly used format.

Configuring an LDAP IdP

You can configure HM to authenticate users against an LDAP directory server, such as OpenLDAP, Microsoft Active Directory, or other LDAP-compatible directories.

Requirements

Before configuring LDAP in HM, gather the following information from your LDAP administrator:

- LDAP server hostname and port
- Bind DN and password for a service account with read access
- Base DN for user searches
- User attribute names (username, email, and name)

Configuring an LDAP IdP in HM

1. In the HM console, select **Settings** > **Identity Providers**.
2. Select **Add Identity Provider**.
3. Select the **LDAP** tab.
4. In the **Name** field, enter a unique name for this identity provider.
5. In the **Description** field, enter an optional description.
6. Enter the LDAP server connection settings:

Where: - **Host** is the LDAP server hostname. For example: `ldap.example.com` . - **Port** is the LDAP server port. Use `389` for LDAP or `636` for LDAPS. - **Scheme** is the connection protocol. Select `ldap` for unencrypted or `ldaps` for TLS-encrypted connections. - **Use Custom Root CA** is a toggle that appears only when **Scheme** is `ldaps` . Enable it when HM must trust an internal or self-signed CA to connect to the LDAP server. - **Custom Root CA File** is a text file containing the CA certificate used to verify the LDAP server's TLS certificate. It can contain a single CA certificate or a bundle of certificates — multiple `-----BEGIN CERTIFICATE-----` / `-----END CERTIFICATE-----` blocks concatenated together, which is useful when the LDAP server presents a chain that includes intermediate CAs. Upload only the CA or CA chain that signed the server certificate. Don't upload the server certificate or any private keys.

1. Enter the bind credentials: Where:

- **Bind DN** is the distinguished name of the service account used to search the directory. For example: `cn=admin,dc=example,dc=com` .
- **Bind Password** is the password for the bind DN service account.

1. Enter the user search settings: Where:

- **Base DN** is the base distinguished name for user searches. For example: `ou=People,dc=example,dc=com` .
- **Filter** is the LDAP filter for finding user entries. For example: `(objectClass=inetOrgPerson)` .
- **Username** is the attribute used for username matching. For example: `uid` or `cn` .
- **ID Attribute** is the attribute to map to the HM user ID. For example: `uid` or `cn` .
- **Email Attribute** is the attribute to map to the HM user email. For example: `mail` .
- **Name Attribute** is the attribute to map to the HM user display name. For example: `cn` or `displayName` .

1. (Optional) To enable *Automatic role assignment with role mapping rules* , configure group search settings: Where:

- **Base DN** is the base distinguished name for group searches. For example: `ou=Groups,dc=example,dc=com` .
- **Filter** is the LDAP filter for finding group entries. For example: `(objectClass=groupOfNames)` .
- **Name Attribute** is the attribute to map to the group name. For example: `cn` .
- **User Matchers** define how to match users to groups. Each matcher specifies: - **User Attribute** is the user attribute to match. For example: `DN` or `uid` . - **Group Attribute** is the group attribute that contains user references. For example: `member` or `memberUid` .

Example configuration for OpenLDAP with `groupOfNames` : - Base DN: `ou=Groups,dc=example,dc=com` - Filter: `(objectClass=groupOfNames)` - Name Attribute: `cn` - User Matcher: User Attribute `DN` , Group Attribute `member`

Example configuration for Active Directory: - Base DN: `CN=Users,DC=example,DC=com` - Filter: `(objectClass=group)` - Name Attribute: `cn` - User Matcher: User Attribute `DN` , Group Attribute `member`

1. Select **Save**.

Configuring an OIDC IdP

You can use any identity provider that supports OpenID Connect (OIDC), such as Okta, Microsoft Entra ID, Google Workspace, or Keycloak.

Starting the OIDC configuration in HM

1. In the HM console, select **Settings** > **Identity Providers**.
2. Select **Add Identity Provider**.
3. Select the **OIDC** tab.
4. Copy the **Redirect URI** that HM displays on the page (you need this for your IdP).

Configuring your identity provider

In a separate browser tab, register a new application in your OIDC provider:

1. Create a new OIDC or OAuth 2.0 application.
2. Enter a name for the application.
3. Set the **Redirect URI** (also called Callback URL) to the value you copied from HM.
4. Configure the application to include `name` and `email` claims in the ID token or UserInfo endpoint.
5. Save the application.
6. Record the following values from your IdP:

- **Issuer URL** (for example, `https://accounts.google.com`)
- **Client ID**
- **Client Secret**

1. Assign the users you want to have access to HM to the registered application.

Completing the OIDC configuration in HM

Return to the HM browser tab and enter the following:

1. In the **Name** field, enter a unique name for this identity provider.
2. In the **Description** field, enter an optional description.
3. Enter the values from your OIDC provider: Where:

- **Issuer URL** is the issuer URL of your OIDC provider.
- **Client ID** is the client ID that your IdP assigns to your registered application.
- **Client Secret** is the client secret that your IdP assigns to your registered application.

1. (Optional) To enable *Automatic role assignment with role mapping rules* , configure group claim settings: Where: - **Groups Key** is the claim name that contains the user's group memberships in the ID token or UserInfo response. For example: `groups` .

> **Note**
>
> Configure your OIDC provider to include group membership claims in the ID token or UserInfo response. The exact configuration varies by provider. Refer to your provider's documentation for configuring group claims. !!!

1. Select **Save**.

Testing the IdP integration

After configuring an identity provider, you can verify whether the integration is working correctly using the built-in test feature. The test performs a sign-in attempt through the configured IdP and validates that the IdP returns user attributes such as `email` and `name` correctly.

1. In the HM console, select **Settings** > **Identity Providers**.
2. Select the IdP you want to test.
3. Select **Test Connection**.
4. Follow the sign-in prompts from your IdP. After authentication is complete, the HM console displays the test result indicating whether the integration works correctly.

Managing the IdP

In HM, manage IdPs by editing or deleting them, and assign users to restrict access to authorized users only.

Granting users access through the IdP

HM automatically associates users with an IdP when they first log in through it. To grant users access:

1. In your IdP's admin console, assign the users you want to give access to HM.
2. Share the **Quick Sign-in URL** with your users:

- In the HM console, select **Settings** > **Identity Providers**.
- Select the IdP and copy the **Quick Sign-in URL**.

1. When users log in using this URL, HM automatically registers them and associates them with the IdP.

Removing the IdP from HM

You must first delete all users who logged in through that IdP before you can delete the IdP from HM. HM disables the delete option for an IdP until you delete all associated users.

1. In the HM console, select **User Management**.
2. Identify and delete all users associated with the IdP you want to remove.
3. Select **Settings** > **Identity Providers**.
4. Select the IdP you want to remove and select **Delete**.
5. Confirm the deletion.

5.3.3.11.2 Rotating IdP connector encryption keys

Hybrid Manager (HM) encrypts the sensitive fields of your identity provider (IdP) connector configuration — values such as client secrets and bind passwords — using a Fernet key stored in the `fernet-key` field of the `hm-portal-bootstrap` Kubernetes secret. Rotate this key periodically as part of your security posture, or immediately if you suspect a key has been exposed.

Because `fernet-key` accepts a comma-separated list of keys, you can rotate without downtime. HM always encrypts with the first key in the list but can decrypt data protected by any key in the list. This means you can add a new key to the front, let HM start using it, re-encrypt your existing connectors, and only then remove the old key — keeping the service running throughout.

Before you begin

- You need `kubectl` access to the cluster running HM, with permission to read and edit secrets in the `default` namespace.
- You need `openssl` installed on the machine you use to generate the new key.
- Perform a backup of the current Fernet key, and store it somewhere secure. An example is provided, but you can use any method you prefer.
- Because rotation requires you to re-save each connector and restarts HM's identity service, perform it during a maintenance window.

Important

A Fernet-encrypted value can only be read with the key that produced it. Keep a key in the list for as long as any data is still protected by it — removing a key before re-encrypting all connectors makes those connector secrets permanently unrecoverable.

Rotating the key

1. Read and back up the current key. Copy this value into your secrets manager — it's both your backup and the "current key" referenced in the next steps:

```
kubectl get secret hm-portal-bootstrap -n default \
  -o jsonpath={.data.fernet-key} | base64 -d
```

1. Generate a new key. A Fernet key is a URL-safe, base64-encoded, 32-byte value. Copy the output:

```
openssl rand -base64 32 | tr +/ -_
```

1. Add the new key while keeping the current one. Open the secret for editing:

```
kubectl edit secret hm-portal-bootstrap -n default
```

When the editor opens, you'll see `fernet-key` listed under `data:` as a base64-encoded value. To supply plain-text keys without manually encoding them, remove the `fernet-key` line from the `data:` block and add a top-level `stringData:` block instead, listing the new key **first** and the current key second:

```
apiVersion: v1
kind: Secret
metadata:
  name: hm-portal-bootstrap
  # …
data:
  static-passwords.yaml: <unchanged>
stringData:
  fernet-key: "<new-key>,<current-key>"
```

Save and close the editor. HM uses the new key (first in the list) to encrypt and still reads existing data with the current key.

Note

stringData overrides only *fernet-key* , so the rest of the secret — including *static-passwords.yaml* — is unchanged. The API server moves *stringData* into *data* and encodes it, so the next time you edit the secret you'll again see *fernet-key* under *data:* (now holding the updated value), not *stringData* . !!!

1. HM's identity service restarts automatically once the change is replicated to the `upm-dex` namespace. Verify the pods have restarted and are running:

```
kubectl get pods -n upm-dex -l app.kubernetes.io/name=dex
```

Authentication continues to work uninterrupted.

1. Re-encrypt your existing connector secrets with the new key. In the HM console, open each configured IdP connector and save it again. (See *Configuring your IdP in the HM console* .) Saving a connector re-encrypts its sensitive fields with the current (new) key.

Note

Until you re-save a connector, its secret is still protected by the old key. Re-save every connector before continuing to the next step. !!!

1. Remove the old key. Once every connector has been re-saved, edit the secret again:

```
kubectl edit secret hm-portal-bootstrap -n default
```

Remove the `fernet-key` line from the `data:` block and add a `stringData:` block containing only the new key:

```
stringData:
  fernet-key: "<new-key>"
```

Save and close the editor.

Warning

Don't remove the old key until you've re-saved every connector. Removing it while a connector's secret is still protected by the old key prevents HM from reading that secret, and the affected IdP login stops working. !!!

Verifying the rotation

After HM's identity service restarts, confirm the pods are running:

```
kubectl get pods -n upm-dex -l app.kubernetes.io/name=dex
```

Then sign in through each configured IdP to confirm authentication succeeds.

If a login fails after you removed the old key, that connector wasn't re-encrypted in time. To recover, edit the secret again and set `stringData.fernet-key` back to `"<new-key>,<old-key>"` (new key first), using your backed-up old key. HM's identity service restarts automatically. Sign in to confirm the login works, re-save the affected connector in the HM console, then remove the old key again.

5.3.4 For Kubernetes Administrators

5.3.4.1 Managing users in Hybrid Manager

Hybrid Manager (HM) supports authentication for the HM console using either native users or external identity providers (IdPs) such as LDAP, SAML, or OIDC.

During the initial bootstrapping of HM, an administrator sets up a native superuser for HM console access. Use this native superuser to sign in to the console for the first time.

For testing or demo environments, you can simplify operations by adding more native users as needed.

- See *Creating native users for testing* for more information.

For assigning system roles to different users, HM uses a role-based access control (RBAC) model.

- See *User roles and authorization* for more information.

For production deployments, we strongly recommend configuring an external IdP like LDAP, SAML, OIDC, or another supported connector . This ensures robust and secure user management. Configure your IdP at install time. After configuring your chosen IdP and adding new administrators for the organization, disable the native user login.

- See *Configuring your IdP in the HM console* for more information.

To enforce security and governance requirements across your organization, HM lets organization owners configure authentication policy settings such as access key lifetimes, web session durations, and the maximum number of machine users and access keys.

- See *Managing authentication policy* for more information.

5.3.4.1.1 Managing native users for testing

For testing or demo environments, you can use native users to simplify operations. To manage the lifecycle of native users you can:

- *Changing password for native users* .
- *Creating native users for testing* .

For production deployments, we strongly recommend that you *Configuring your IdP in the HM console* . After configuring an IdP, you can:

- *Disabling native user login* .

Changing password for native users

You can change any native user's password in the case of a security breach.

HM stores native user credentials in the `hm-portal-bootstrap` Kubernetes secret. To change a password, update this secret with a new bcrypt-hashed password.

Changing a native user password

1. Generate a new bcrypt-hashed password, where `$password` contains the new password:

```
NEW_HASH=$(echo -n "${password}" | htpasswd -BinC 10 admin | cut -d: -f2)
echo "Password hash: $NEW_HASH"
```

The output displays the new hashed password, for example:

```
Password hash: $2y$10$YDHMEnc8VZKnhvze538ePOLKol7dFH.S7IgpqC3.X3R/3P2Q9zRcG
```

1. Retrieve the current secret content:

```
kubectl get secret hm-portal-bootstrap -n default -o jsonpath={.data.static-passwords\
↪.yaml} | base64 -d > static-passwords.yaml
```

1. Edit the `static-passwords.yaml` file and replace the `hash` value for the user you want to update:

```
staticPasswords:
  - email: "owner@mycompany.com"
    hash: "$2y$10$YDHMEnc8VZKnhvze538ePOLKol7dFH.S7IgpqC3.X3R/3P2Q9zRcG"
    username: "Owner MyCompany"
    userID: "0fa134d4-b3ee-4503-ad90-91044dc37061"
```

Important

Change only the *hash* value. Do not change the *userID* . Changing the *userID* creates a new user identity, resulting in a duplicate user with the same email and name, and orphans any existing data associated with the original user. !!!

1. Update the secret with the modified content:

```
kubectl patch secret hm-portal-bootstrap -n default \
  --type merge \
  -p "{\"data\":{\"static-passwords.yaml\":\"$(base64 < static-passwords.yaml | tr -d␣
↪\n)\"}}"
```

Kubernetes automatically replicates the secret to the `upm-dex` namespace, and Dex — an IdP proxy service that connects HM with your identity provider — automatically restarts to apply the changes.

1. Verify the Dex pods have restarted and are running:

```
kubectl get pods -n upm-dex -l app.kubernetes.io/name=dex
```

You can now sign in with the new password.

Creating native users for testing

For testing or demo environments, you can simplify operations by adding native users. For production deployments, we strongly recommend *Configuring your IdP in the HM console* .

HM stores native user credentials in the `hm-portal-bootstrap` Kubernetes secret. To add a new user, append their credentials to this secret.

Adding a native user

1. Generate a unique user ID and bcrypt-hashed password for the new user, where `${password}` contains the password:

```
NEW_USER_ID=$(uuidgen | tr [:upper:] [:lower:])
NEW_HASH=$(echo -n "${password}" | htpasswd -BinC 10 admin | cut -d: -f2)
echo "User ID: $NEW_USER_ID"
echo "Password hash: $NEW_HASH"
```

1. Retrieve the current secret content:

```
kubectl get secret hm-portal-bootstrap -n default -o jsonpath={.data.static-passwords\
↪.yaml} | base64 -d > static-passwords.yaml
```

1. Edit the `static-passwords.yaml` file and append the new user to the `staticPasswords` list:

```
staticPasswords:
  - email: "owner@mycompany.com"
    hash: "$2y$10$FWXeALAmT/axI/FIos/NtOVEldynfCx/bqDOAJ9EbiKeq5NuDR.XO"
    username: "Owner MyCompany"
    userID: "c234d951-dee1-4e3d-9f1d-108dabe339ce"
  - email: "newuser@example.com"
    hash: "$2y$10$FIvL7GWymJtGZo5uTMrWsOTuifUmwJQg8JtHJQxuLkT3os2Ozpo1C"
    username: "New User"
    userID: "60138e10-9552-4922-afe7-044c2250a23d"
```

Important

Do not change a user's *userID* after it's assigned. Changing it creates a new user identity, resulting in a duplicate user with the same email and name, and orphans any existing data associated with the original user. !!!

1. Update the secret with the modified content:

```
kubectl patch secret hm-portal-bootstrap -n default \
  --type merge \
  -p "{\"data\":{\"static-passwords.yaml\":\"$(base64 < static-passwords.yaml | tr -d␣
↪\n)\"}}"
```

1. Kubernetes automatically replicates the secret to the `upm-dex` namespace, and the Dex deployment automatically restarts to apply the changes.
2. Verify the Dex pods have restarted and are running:

```
kubectl get pods -n upm-dex -l app.kubernetes.io/name=dex
```

The new user can sign in to the HM console with their email and password.

Parameters

These settings are for the user in the `portal.authentication.staticPasswords` entry. Each user needs values for `email` , `hash` , `username` , and `userID` .

Parameter	Description
email	Email address of the user. Also serves as the user's login identifier for accessing the HM console.
hash	Bcrypt-hashed user password for password store. To generate this value, use *echo ${password} I htpasswd -BinC 10 userA I cut -d: -f2*, where you store the actual password in the *${password}* variable. *userA* represents the username used during the password hashing process. It can be any arbitrary text, as HM doesn't use it elsewhere in the configuration. HM uses only the resulting hash.

continues on next page

Table 65 – continued from previous page

Parameter	Description
userID	Each static-password user requires a unique *userID*. You can generate one with a UUID tool or assign a random string manually — but once set, never change it. Changing the *userID* creates a new identity, duplicating the user and orphaning their existing data.
username	Unique username for the user. This is the primary identifier for logging in to the HM console. It can be the same as *email*.

Disabling native user login

Before disabling native user login, confirm that at least one other IdP user has the organization owner role. Assign a new organization owner if you haven't done that yet.

Assigning a new organization owner

1. Log in at least once to the Hybrid Manager (HM) console as the IdP user that will become the organization owner. This ensures the HM console indexes the new user. Then, log out.
2. Log in as the current native owner of HM (for example, `owner@mycompany.com`).
3. From the top-right profile menu, select **User management**. Verify the user that will become the new owner is in the list.
4. Edit the user and assign them the owner role.

Disabling native users

1. Retrieve the current secret content:

```
kubectl get secret hm-portal-bootstrap -n default -o jsonpath={.data.static-passwords\
↪.yaml} | base64 -d > static-passwords.yaml
```

1. Edit the `static-passwords.yaml` file and set `staticPasswords` to an empty list:

```
staticPasswords: []
```

1. Update the secret with the modified content:

```
kubectl patch secret hm-portal-bootstrap -n default \
  --type merge \
  -p "{\"data\":{\"static-passwords.yaml\":\"$(base64 < static-passwords.yaml | tr -d␣
↪\n)\"}}"
```

1. Kubernetes automatically replicates the secret to the `upm-dex` namespace, and Dex automatically restarts to apply the changes.

You've now disabled native user login, but the user account still appears in the **User management** view. Only users who authenticate through the configured IdP can access the HM console.

5.3.4.1.2 Managing authentication policy

As an organization owner, you can configure authentication policies to control the security settings for your organization. These settings let you manage how users and machine users access your account. For example, limiting access key lifetimes reduces exposure from compromised keys, and capping the number of machine users supports least-privilege access.

Configuring your authentication policy

Important

Only users with the **Organization owner** role can view and update authentication policy settings.

To update your authentication policy settings:

1. In the Hybrid Manager (HM) console, navigate to **Settings**.
2. Select the **Authentication** tab.
3. Enter the values you want to configure. For details on each setting, see *Parameter values* .
4. Select **Save** to apply the changes.

Note

Policy changes don't affect existing sessions, machine users, or access keys. Changes take effect immediately for new sessions and newly created machine users and access keys. Decreasing the access key or machine users quota does not impact already provisioned machine users and access keys.

Parameter values

These settings are available under your organization's **Settings > Authentication**.

Setting	Description	Default	Valid range
Maximum Machine Users	Maximum number of machine users allowed in your organization.	4096	1 to 2147483647
Maximum Access Keys per Machine User	Maximum number of access keys that can be created for each machine user.	2	1 to 2147483647
Maximum Access Keys per User	Maximum number of access keys that can be created for each normal (human) user account.	1	1 to 2147483647
Maximum Access Key Duration	Maximum lifetime of an access key, in days. Keys expire after this period.	365	1 to 2147483647
Maximum Web Session Duration	Maximum duration of a web session, in seconds. Users are logged out after this period.	86400	1 to 2147483647

5.3.4.1.3 User roles and authorization

You can assign several user roles to Hybrid Manager (HM) users. See:

- *User role permissions by use case*
- *User role descriptions*
- *Automatic role assignment with role mapping rules*

User role descriptions

Hybrid Manager (HM) has two types of users: human users (native users or IDP users) and machine users.

You can assign any of the default roles to both types of users, except that only machine users can be estate ingesters.

The two main categories of roles are organization-level and project-level.

Organization-level roles

You can assign these roles to human or machine-users using the **User Management** option of your user profile menu at the top-right of the navigation bar in the HM console.

Organization Administrator

This role provides read access at the organization level, allowing the user to view information and settings in the organization as a whole.

Organization Owner

The purpose of this role is to manage the organization at a high level. Organization owners can create projects and assign organization-level roles to other users.

Platform Administrator

This role is for users who need to access and manage the underlying platform components of HM. They can access platform management and monitoring tools.

AI Model Manager

This role is for users who work with the AI model and model service feature of HM, granting them full access to its functions. It also grants access to the Langflow flow builder UI.

Project-level roles

These roles can be assigned to human or machine-users (except for Estate Ingester). Select **Users** in the left navigation when viewing a project.

Project Owner

In a specific project, the Project Owner role has the highest level of authority. They can take all actions in that project and are responsible for assigning project-level roles to other users.

Project Editor

This role is for users who need to actively work with the data in a project. Users with the Project Editor role have read and write access to data.

Project Viewer

For users who only need to observe the data in a project, the Project Viewer role provides read-only access.

Estate Ingester

This role is for machine-users and provides access to perform estate ingests in a project.

Pipeline Editor

This role is for users who need to create and manage AI pipelines and knowledge bases in a project.

Catalog Data Reader

Users with this role can read the Iceberg namespace/table/view of project-scoped catalogs.

Catalog Data Writer

This role allows users to read, write, and delete the Iceberg namespace/table/view of all project-scoped catalogs in a project.

Migration Portal Projects Owner

This role is for users who manage Migration Portal projects in an HM project. They can create, read, update, and delete these projects.

Migration Portal Projects Editor

Users with this role can read and update Migration Portal projects in an HM project.

Migration Portal Projects Viewer

This role provides read-only access to Migration Portal projects in an HM project.

Automatic role assignment with role mapping rules

Role mapping rules allow you to automatically assign roles to users based on their identity provider (IdP) attributes. When a user logs in, Hybrid Manager (HM) evaluates the configured rules and assigns the appropriate roles without manual intervention.

This feature is useful for organizations that want to:

- Automatically grant roles based on IdP group membership.
- Reduce manual role assignment overhead.
- Ensure consistent role assignments across users from the same IdP groups.

How role mapping works

When a user authenticates through an external IdP, HM receives information about their IdP identity and group memberships. Role mapping rules define which roles HM assigns based on these attributes.

Each rule consists of:

- **Conditions**: Criteria that must match for the rule to apply (for example, IdP identifier and group membership).
- **Roles**: The roles to assign when all conditions match.

Rule evaluation

- All conditions within a single rule must match (AND logic).
- Multiple rules are evaluated independently (OR logic).
- A user receives the union of roles from all matching rules.

Role assignment sources

Users can have roles from two sources:

- **Manual**: Roles that an administrator assigns directly through the HM console. An administrator can modify or remove these roles.
- **Mapping**: Roles that HM assigns automatically through role mapping rules. You can't modify these roles in the HM console. To change a user's mapped roles, update the role mapping rules or the user's group membership in the IdP.

Prerequisites

Before configuring role mapping rules:

1. Configure an external IdP (SAML, OIDC, or LDAP) with group attribute mapping enabled.
2. Ensure users have group memberships configured in your IdP.
3. Have organization owner or appropriate administrative permissions.

> **Important**
>
> If you configure the HM IdP integration to retrieve group membership information, each user must be a member of at least one group in the IdP. Users with no group memberships can't sign in to HM.

Configuring role mapping rules

You can configure role mapping rules at two levels:

- **Organization level**: Assigns organization-scoped roles (Organization Administrator, Organization Owner, etc.)
- **Project level**: Assigns project-scoped roles (Project Owner, Project Editor, Project Viewer, etc.)

Creating organization-level rules

> **Important**
>
> Only an Organization Owner can create or edit organization-level role mapping rules.

1. In the HM console, navigate to **User Management**.
2. Select the **Role Mapping** tab.
3. Select **Add Role Mapping Rule**.
4. Configure the rule conditions:

Condition	Description
Identity Provider	Select the IdP this rule applies to.
Group	Enter the IdP group name that users must belong to.

1. Select the organization roles to assign when the conditions match.
2. Select **Save**.

Creating project-level rules

Important

Only an Organization Owner or Project Owner can create or edit project-level role mapping rules.

You can add project-level role mapping rules to an existing project or during project creation.

Adding rules to an existing project

1. In the HM console, navigate to the project.
2. Go to **Users**.
3. Select the **Role Mapping** tab.
4. Select **Add Role Mapping Rule**.
5. Configure the rule conditions (Identity Provider and Group).
6. Select the project roles to assign when the conditions match.
7. Select **Save**.

Adding rules during project creation

1. When creating a new project, enable **Automated Role Assignments**.
2. Configure the role mapping rules as part of the project setup.

Note

Roles that HM assigns through role mapping rules can't be modified in the HM console. To change a user's mapped roles, update the role mapping rules or the user's group membership in the IdP.

Example configurations

Assigning organization owner role to IT administrators

Create an organization-level rule:

- **Identity Provider**: Your corporate SAML IdP
- **Group**: `IT-Admins`
- **Roles**: Organization Owner

HM automatically assigns all users in the `IT-Admins` group the Organization Owner role when they log in.

Assigning project roles based on team membership

Create project-level rules for a data analytics project:

Rule 1 - Data engineers get editor access:

- **Identity Provider**: Corporate LDAP
- **Group**: `data-engineering`
- **Roles**: Project Editor

Rule 2 - Data analysts get viewer access:

- **Identity Provider**: Corporate LDAP
- **Group**: `data-analysts`
- **Roles**: Project Viewer

Users who are members of both `data-engineering` and `data-analysts` groups receive both Project Editor and Project Viewer roles.

Role reconciliation

When you modify role mapping rules, HM automatically reconciles role assignments for affected users:

- **Adding a rule**: Users matching the new rule receive the configured roles on their next login.
- **Removing a rule**: HM revokes mapping-assigned roles from the removed rule; manually assigned roles remain unchanged.
- **Modifying a rule**: HM removes the previous assignments and applies new assignments based on the updated rule.

HM immediately reconciles roles for users who have previously logged in when rules change. Users who have never logged in receive the appropriate role assignments on their first login.

Invalidating user sessions

When you update group memberships in your IdP (for example, adding a user to a new group), currently logged-in users don't automatically receive the updated group information until they log in again.

To ensure that users receive the correct roles based on their current group memberships, you can invalidate their sessions to force re-authentication.

1. In the HM console, go to **Settings** > **Identity Providers**.
2. Select the IdP whose users need to be re-authenticated.
3. Select **Invalidate Sessions**.
4. Confirm the action.

HM terminates all active sessions for users from that IdP. Users must log in again, at which point their authentication token includes the current group memberships from the IdP.

> **Note**
>
> Session invalidation affects all users from the selected IdP, not just users whose groups changed. Use this feature after making significant group membership changes in your IdP.

Removing role mapping rules

When you remove a role mapping rule:

1. HM automatically revokes roles assigned only through mapping from affected users.
2. Manually assigned roles remain in place.
3. The change takes effect immediately for users who have previously logged in.

> **Note**
>
> Removing a role mapping rule doesn't affect users' ability to log in. It only removes the automatically assigned roles from that rule.

Activity log

Changes to role mapping rules are recorded in the activity log. When you add or remove rules, HM logs an update action indicating whether the scope is organization or project, and the number of rules configured.

You can view role mapping activity in the organization activity log by navigating to **Settings** > **Activity Log**. Project-specific role mapping changes can also be viewed in the project's activity log by navigating to the project and selecting **Activity Log**.

User role permissions by use case

Authorization of these user roles follows a role-based access control (RBAC) model with the restrictions applying to a specific scope :either within one project or within one account.

The following list doesn't cover Postgres cluster database authorization.

Currently, you can't create custom roles. Only these 14 predefined roles are available.

Only machine-users can be assigned to ingest self-managed cluster data.

The following functionality is available to all authenticated users, regardless of assigned role:

- Access to the chatbot. Users must still have roles that grant access to the relevant resources (for example, Postgres clusters, AI models, and pipelines) to interact with them through the chatbot.
- Manage Postgres image repositories and repository rules.

5.3.4.2 Monitoring with Hybrid Manager

With Hybrid Manager (HM), you can collect a wide range of metrics (host metrics, Kubernetes metrics, Postgres database data, query data, and logging data). You can visualize the collected data with the HM console or Grafana instance to analyze trends, configure alerting, improve the efficiency of databases, perform troubleshooting, and much more.

5.3.4.2.1 Understanding the monitoring architecture

HM has three main internal pipelines that provide monitoring capabilities:

- Prometheus and Thanos collect metrics.
- Loki collects logs.
- Stats Collector collects query data.

All these services come with their own dedicated stores, which means that the collected data is kept in HM. These services and stores are deployed by default.

These collected metrics become available to you through two interfaces:

- The **Grafana** dashboard.
- The **HM console** , for example, in the **Estate** tab or **Cluster** view .

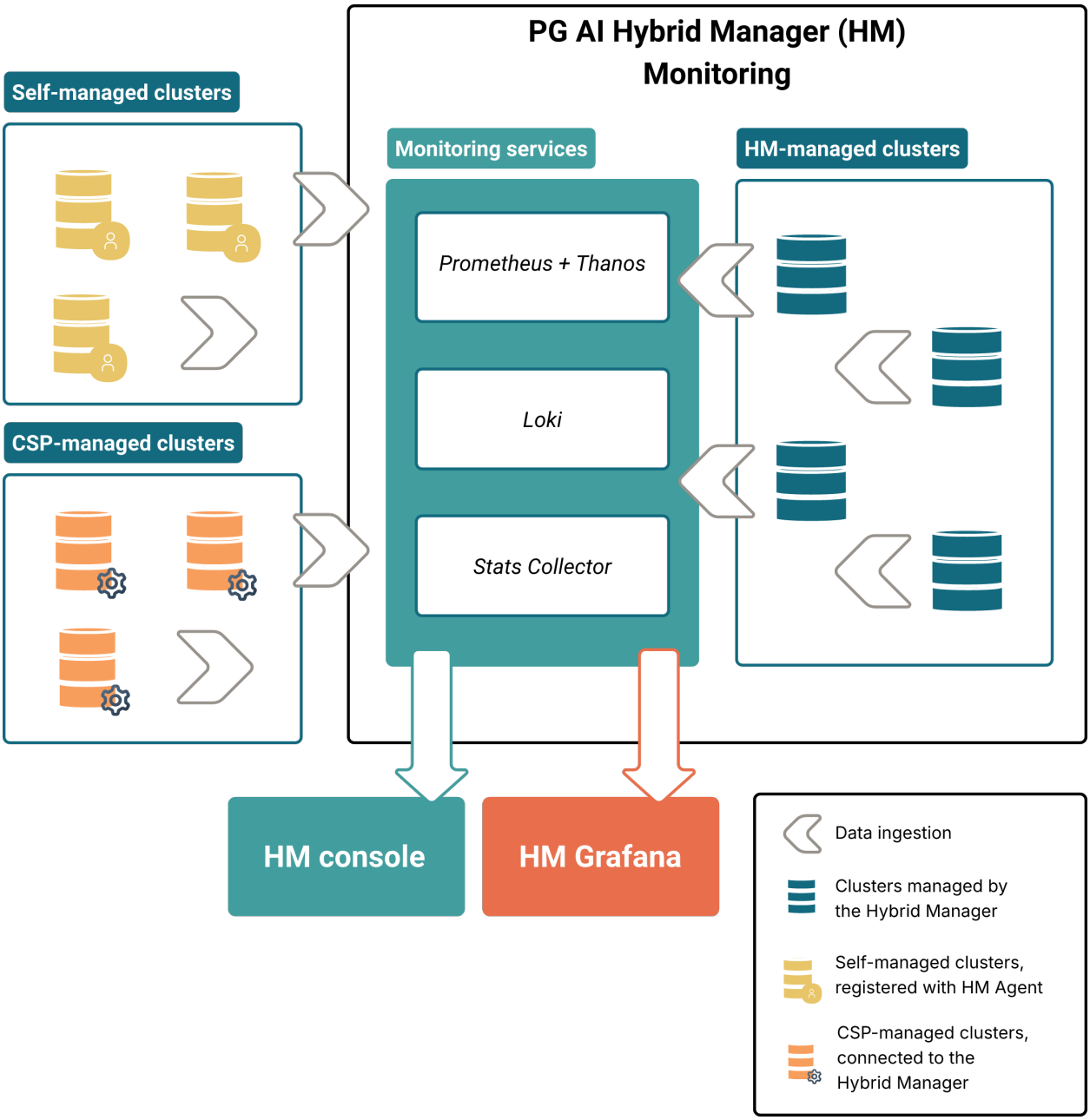


Fig. 12: Monitoring infrastructure in Hybrid Manager

Enabling database monitoring

The monitoring services that come with HM require access to the databases you want to monitor, as well as some extensions to enable Stats Collector.

- For HM-managed clusters (created with the HM), the default setup provides access to the monitoring stack.
- For external self-managed clusters, each of the databases must be registered with the EDB Postgres AI agent to enable data collection. By default, the agent provides the configuration to collect Prometheus, Thanos, and Loki data. Additionally, the agent provides configuration parameters so you can control whether to enable or disable the extensions required for Stats Collector (collection of query data), namely `edb_wait_states_data` , `edb_stat_monitor` , `edb_query_advisor` , and `pg_stat_statements` .
- For external cloud service provider (CSP)-managed databases, such as AWS RDS, HM needs to get access to the database through a Kubernetes `secret` resource to enable data collection. See Monitoring CSP-managed databases for more information. Query data with Stats Collector isn't currently available for CSP-managed databases.

Monitoring data ingestion and scope

Because of the different levels of access HM has to the variety of cluster types, the way that data is ingested and the scope of collected data varies.

- For HM-managed clusters (created with HM), the databases push host metrics, Postgres metrics, cluster metrics, query metrics, data required for recommendations, and logs to the corresponding Prometheus, Thanos, Loki, and Stats Collector stores.
- For external self-managed databases, the agent pushes host metrics, Postgres metrics, cluster metrics, query metrics, data required for recommendations, and logs to the corresponding Prometheus, Thanos, and Stats Collector stores at configurable intervals.
- For external CSP-managed databases, such as AWS RDS, HM leverages the HM-to-database connection to pull host and Postgres metrics to the central stores. Query data with Stats Collector isn't currently available for CSP-managed databases.
- For automated compliance: HM server transmits usage report to EDB for core-based licensing and entitlement tracking across all on-premises servers, eliminating manual audits.

For more information, see *Metrics support scope* .

Monitoring refresh frequency

The refresh intervals for the metrics that are collected by HM vary depending on the metric and database type. This difference is due to the varying methods of data collection and integration of Prometheus/Thanos, Loki, and Stats Collector services.

See *Metrics refresh intervals* to understand how often metrics are updated and how to customize the frequency.

5.3.4.2.2 Connecting data into external tools

Currently, EDB doesn't support plugging metrics and log data collected by HM into external tools or enterprise systems.

5.3.4.2.3 Related topics

- *Enabling monitoring* ► Install the agent or connect your database to HM to enable monitoring on external databases.
- *Enabling HM server monitoring* ► Enable usage reporting in the HM server.

- *Metrics refresh intervals* ► Understand how often metrics are updated and how to customize the frequency.
- Monitoring with Grafana ► Use Grafana to monitor your databases.
- Monitoring with HM console ► Use the HM console to monitor your databases.
- *Set up notifications (Observability)* ► Configure notifications.
- *Activity log* ► Audit platform events across your organization or project.

Enabling monitoring

On HM-managed database clusters

Since *Hybrid Manager (HM)-managed database clusters* are created in the same environment as HM, their resources are already accessible. Therefore, these clusters are monitored by default, and monitoring starts as soon as their deployment is finished.

On external database clusters

On external database clusters are either managed by a cloud service provider (CSP) or by you. For this reason, additional steps are required to create a connection between your database cluster and your HM instance.

There are two types of external databases that you can monitor:

- **Monitoring self-managed deployments**

To enable monitoring on self-managed deployments , you must install the agent and register external self-managed databases.

Examples: Self-managed Postgres databases or Cluster managers like EDB Failover Manager you have on bare metal, or those hosted in the cloud that aren't managed by the CSP (like a Postgres database on an EC2 instance).

- **Monitoring CSP-managed databases**

To enable monitoring on CSP-managed databases , connect the database to HM.

Examples: Amazon RDS.

> **Note**
>
> You can also register databases with the agent to *assess them for migration* .

Monitoring self-managed deployments

To monitor self-managed external Postgres instances from Hybrid Manager (HM) that are deployed on bare metal or virtual machines, you must first install EDB Postgres AI agent. The agent provides monitoring and observability metrics to your HM.

To have monitoring, query diagnostics, and recommendations for your self-managed Postgres instance:

1. *Create an OS user to run the agent.*
2. *Prepare your database.*
3. *Create a machine user.*
4. Install the agent.
5. *Configure the agent.*
6. *Run the agent.*

7. *Configure the agent to run as a service.*

> **Note**
>
> To leverage full observability capabilities, we recommend deploying one agent per host machine with self-managed database clusters. Then, you can configure the agent on that host to connect to each database with a dedicated DSN string. Details on DSN configuration are provided in *Configuring your agent* .

For extra help with administrative tasks needed to configure the agent, see *Administrative tasks* .

See *Troubleshooting Langflow in Hybrid Manager* for help with general troubleshooting and *Data security and sovereignty* to understand how your data is stored by the agent.

Create an OS user to run the agent

We recommend that you create a dedicated operating system user to configure and run the agent. This is a security best practice because it avoids granting permissions to the agent that it doesn't need to function.

To create a user:

```
sudo useradd <agent_os_user>
```

We recommend that you add the agent OS user to the Postgres user's group. This facilitates granting the agent access to the Postgres log files later.

```
sudo usermod -aG postgres <agent_os_user>
```

Preparing your Postgres database for monitoring

Monitoring user

When configuring the agent, you must first connect to the self-managed database you want to monitor and create a user that the agent will use to connect to the database. This user can be any Postgres database user with the pg_monitor role and the pg_read_all_data role who has CONNECT privilege on the target database.

For this example, create a user `agent_user` :

```
CREATE USER agent_user WITH PASSWORD <password>;
GRANT CONNECT ON DATABASE <target_database> TO agent_user;
GRANT pg_monitor, pg_read_all_data TO agent_user;
```

Where `<target_database>` is the name of the database you want to monitor.

If you're using Postgres 16 or earlier, you must also grant the following permission to allow the agent to discover the Postgres log file location:

```
GRANT EXECUTE ON FUNCTION pg_catalog.pg_current_logfile() TO agent_user;
```

You must ensure that the agent user can log in to the database to which you plan to connect. For example, if you plan to connect to a database called `hr` :

```
GRANT CONNECT ON DATABASE hr TO USER agent_user;
```

The edb_internals schema and required extensions

To leverage the full monitoring and optimization features of Hybrid Manager (HM), you must enable extensions that add instrumentation to Postgres. You need these extensions to enable the **Query Diagnostics** and **Recommendations** tabs as well as some metrics.

To install the extensions:

1. Refer to the per-extension installation instructions for the correct commands for your platform and Postgres distribution:
    - EDB Wait States
    - EDB Query Advisor
    - EDB Stat Monitor

In addition, pg_stat_statements must be enabled in the public schema. pg_stat_statements is installed with Postgres, so no additional installation is required.

For example, on RHEL 9, to monitor EDB Advanced Server 17:

```
dnf install edb-as17-server-edb_wait_states \
            edb-as17-query-advisor \
            edb-as17-stat_monitor
```

1. While connected to the database, add the required preload modules to the Postgres `shared_preload_libraries` . For example, if the current value of `shared_preload_libraries` is empty:

```
ALTER SYSTEM SET
shared_preload_libraries="$libdir/pg_stat_statements","$libdir/edb_wait_states", "
↪$libdir/query_advisor","$libdir/edb_stat_monitor";
```

1. As a superuser, create the schema in the database to which the agent will be connected and enable the `pg_stat_statements` extension as well as the extensions installed in the previous step:

```
CREATE SCHEMA edb_internals;
CREATE EXTENSION edb_wait_states WITH SCHEMA edb_internals;
CREATE EXTENSION query_advisor WITH SCHEMA edb_internals;
CREATE EXTENSION edb_stat_monitor WITH SCHEMA edb_internals;
CREATE EXTENSION pg_stat_statements;
```

1. Add the edb_internals schema to the search path. For example, as a superuser:

```
ALTER SYSTEM SET
search_path="$user", public, edb_internals;
```

1. Grant the agent user access to this schema:

```
GRANT USAGE ON SCHEMA edb_internals TO agent_user;
GRANT SELECT ON ALL TABLES IN SCHEMA edb_internals TO agent_user;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA edb_internals TO agent_user;
```

Postgres logging configuration

To collect Postgres logs, the Postgres logging collector must be enabled and the agent must have permission to read the log files. The simplest way to do this is to run the agent as the Postgres user. However, as a matter of security best practice, we recommend running the agent using a *Create an OS user to run the agent* .

To allow the agent OS user to read the log files, logs must be located outside of the data directory and have appropriate permissions. For example, setting the following parameters in `postgresql.conf` causes logs to be written to `/var/log/postgres-lc` and be readable by any user who's a member of the owner's group:

```
logging_collector=on
log_directory=/var/log/postgres-lc
log_file_mode=0640
log_destination = jsonlog
```

The new logging location must be owned by the Postgres user and readable by that user's group. Since the agent OS user is a member of the same group, it then also gets access to the log directory. For example:

```
mkdir /var/log/postgres-lc
chown postgres:postgres /var/log/postgres-lc
chmod 0750 /var/log/postgres-lc
```

Restart Postgres after completing these changes.

Creating a machine user

In the Hybrid Manager (HM) console, create a machine user that the agent can use to ingest data into your estate.

Creating a machine user

1. Select your profile icon in the upper-right corner.
2. Select **User Management > Add New User**.
3. On the **Add New User** page, enter a name for the new user.
4. Optionally, enter an email address for the user.
5. Select **Create Access Key**. The form expands where you can enter an **Access Key Name** and an **Expiry** time.
6. Select **Add User**.
7. Store the access key safely. You won't be able to access the key again. Select **Key stored safely**.

Assigning roles

1. Select the **Projects** tab to view all your projects.
2. Select the project from which you want to monitor the database.
3. Select the **Users** option in the left navigation, and locate the machine user you just created.
4. Select the edit button (pen icon) of the user entry.
5. Select **Assign Roles**.
6. On the **Assign Project Roles** window, select **estate ingester**.
7. Select **Submit**.

Your new machine user is ready to ingest data from the agent.

Configuring the EDB Postgres AI agent

Configure the EDB Postgres AI agent (deployed as `beacon-agent`) to allow it to connect to HM and fetch monitoring data from the source database.

> **Note**
>
> You can also use the agent to *assess your Oracle database for migration* .

Prerequisites

- Install the agent .
- You *Creating a machine user* assigned to it. Store the access key in an environment variable named `$BEACON_AGENT_ACCESS_KEY` so the EDB Postgres AI agent can access it later.
- If you plan to use schema ingestion as part of a migration workflow, `pg_dump` must be installed on the machine running the agent. See *Configuring your auxiliary machine or bastion host* for installation instructions and compatibility rules.

Obtain EDB Postgres AI connection parameters

The EDB Postgres AI agent requires specific configuration values that are set during the HM installation. You cannot access these yourself. Contact the administrator or installer of your HM instance to request the following values:

- **Beacon server hostname**: When configuring the HM installation, administrators/installers must specify an internal endpoint for the `beacon_server` service. Request the administrator or installer of your HM instance to provide the hostname (URL) they set for the `beacon_server` service in the Helm chart configuration file used for installation. For this, they'll have to look up the value they set for `parameters.upm-beacon.server_host` (or `BEACON_SERVICE_DOMAIN_NAME`) in the `values.yaml` file. You will need this value later to configure the EDB Postgres AI agent.
- **Root certificate path (if required)**:The EDB Postgres AI agent requires a trusted connection to the `beacon_server` service in your HM instance. If the server running the agent doesn't inherently trust the HM's TLS certificate (managed by your organization's security infrastructure), request the administrator or installer of your HM instance to provide the file with the certificate. Store this certificate on the machine running the EDB Postgres AI agent. You will need to provide the directory path to this certificate later to configure the EDB Postgres AI agent.

See *Administrative tasks* for more information about the `beacon_server` and root certificate.

Define source database connection parameters

The DSN is the connection string the EDB Postgres AI agent uses to connect to your source database. Set the `clusters.nodes.dsn` parameter using an environment variable `$DSN` (recommended) or direct URL format.

A DSN string for connections to databases must use this format:

```
DSN="<postgresql/oracle>://<user>:<password>@<host>:<port>/<database_name>"
```

Where:

- `<postgresql/oracle>` is the type of database you are registering, either `postgresql` for all types of Postgres databases (EDB Postgres, AWS RDS for PostgreSQL, etc.) or `oracle` for Oracle databases.
- The `<username>` and `<password>` correspond to the user you granted permissions when preparing the database.
- `<host>` is the hostname of the instance where your database is running.

- <port> ist the port number of the database you want to fetch data from.
- <database_name> is the name of the database you want to fetch data from.

Examples:

In most production environments, SSL is enabled to ensure network traffic is encrypted.

To connect as user postgres authenticated with password password to a database postgres on port 5432 with SSL enabled, you would specify:

```
DSN="postgresql://postgres:password@localhost:5432/postgres"
```

If your Postgres database is configured with SSL disabled, or if you require an unencrypted connection, append ?sslmode=disable to the DSN string.

For example, to connect as user postgres authenticated with password password to a database postgres on port 5432 with SSL disabled, you would specify:

```
DSN="postgresql://postgres:password@localhost:5432/postgres?sslmode=disable"
```

```
DSN="postgresql://postgres:pa%24%24word@localhost:5432/postgres?sslmode=disable"
```

Prepare a configuration file

1. Change to the *OS user you previously created* to perform the agent configuration.
2. Create a configuration directory in your home directory:

```
mkdir ${HOME}/.beacon
```

If this location isn't convenient, you can also use either:

```
- `/etc/beacon`  - The directory from which you execute any `beacon-agent`  command.
```

The agent looks for its configuration file starting with /etc/beacon , then ${HOME}/.beacon . As a final fallback, it searches the directory from which it's executed.

1. Inside this directory, create a new file beacon_agent.yaml :

```
touch .beacon/beacon_agent.yaml
```

1. Copy and paste the following template into the new file. This template is designed to enable all monitoring options, and generate a migration assessment.

In this template, $BEACON_AGENT_ACCESS_KEY and $DSN are the environment variables you configured previously. Replace all < > placeholders in the template following the *EDB DMS agent parameter reference* .

``yaml # Configuring the EDB Postgres AI agent`

agent.beacon_server	*<beacon_server>:9443*	Host URL that allows the agent to connect to the correct service endpoint. Obtain the host URL of your beacon server from an administrator, the port is fixed as 9443. See *Administrative tasks* for more information.

continues on next page

Table 68 – continued from previous page

agent.project_id	*<project_id>*	To obtain your project's ID, go to the HM console, select **Projects** , and select your project. Your browser displays a URL like *https://portal.example.com/projects/prj_5Wuqyl5JtpvxjiYC/clusters*. The project ID is the identifier starting with "prj_" in the URL of that page. In this example: *prj_5Wuqyl5JtpvxjiYC*.
agent.providers	N/A	Set to - *"onprem"*.
agent.settings_providers	N/A	Set to - *"onprem-settings"* to enable recommendations for Postgres configurations.
agent.schema_providers	N/A	Set to - *"onprem-schema"* to enable schema ingestion and migration assessment capabilities. Remove this section if you don't want to allow schema information to be collected.
agent.root_ca_path	N/A	Set to *""* if your organization has configured the machine where EDB Postgres AI agent runs to trust HM's TLS certificate. Otherwise, obtain the root certificate from an administrator and store it locally. Then, set this value to the path where you stored it. See *Administrative tasks* for more information.
agent.plaintext	*false*	Disables TLS when set to *true*. Defaults to *false*. In almost all cases, this should remain *false* or unset. Only set to *true* for local testing or when connecting to an endpoint that does not use TLS.
agent.feature_flag_interval	*10m*	How often the agent syncs feature flags with the server. Defaults to *10m*. In almost all cases, the default is preferred.
general.logging.level	N/A	Set to *"info"* or *"trace"* to obtain logging information. Start with *"info"* and only use *"trace"* if required, as it is extremelly verbose.
general.metrics.push.[...]	N/A	Enables pushing metrics to the HM console. The *push_endpoint* is the same as *agent.beacon_server*. The *root_ca_path* is the same as *agent.root_ca_path*.
general.metrics.usage.[...]	N/A	Enables storing usage data (and reporting usage data, *Troubleshooting*). Your *company_code* can be found on the EDB Support Portal .
provider.onprem.check_databases_D	N/A	Enables verifying all DSN connections and reports any errors to the terminal by indicating the source of the error with the resource ID during runtime. The check is performed when the EDB Postgres AI agent runs without any CLI subcommands/options , or when it runs with the *--config* subcommand. The default is set to false.
dispatcher.location_id	*<location_id>*	A unique identifier used by the EDB Postgres AI for this agent/location. Consider setting this to the same value as *provider.onprem.host.resource_id* for convenience.

These settings are relevant to the database configuration. All `clusters.*` parameters are nested under `provider.onprem` in the YAML file.

YAML database settings	Placeholder	Guidelines
clusters.resource_id	*<cluster_resource_id>*	Assign a unique identifier to your database cluster. Encase in double quotes. You must begin identifiers with an alphanumeric character (*A-Z, a-z, 0-9*); in between you may also include hyphens (-), underscores (_) and periods (.). You must not use any other characters. Example: *An_I.D.-1*
clusters.name	*"<instance_name>"*	Assign a unique name to your database cluster.
clusters.manager	*"other"*	Collects statistics for cluster manager. Possible values are *"patroni"*, *"repmgr"*, *"efm"*, *"pgd"*, *"other"*.
clusters.pgd_cluster.bdr_database	*<bdr_database_name>*	Required when *clusters.manager* is set to *"pgd"*. Specifies the name of the BDR database in the PGD cluster.
clusters.nodes.resource_id	*<node_resource_id>*	Assign a unique identifier to your database node. Single-node instances require a single entry here. Multi-node instances require an entry per node. Encase in double quotes. Reuse the same id when configuring the DMS agent for migrations. This consistency is essential for the DMS to correctly correlate the resource across both the EDB Postgres AI agent and DMS agent components. You must begin identifiers with an alphanumeric character (*A-Z, a-z, 0-9*); in between you may also include hyphens (-), underscores (_) and periods (.). You must not use any other characters. Example: *An_I.D.-1*
clusters.nodes.dsn	N/A	Connection string or DSN that provides access to your remote database. The DSN must follow the format: *<database_type>://<user>:<password>@<host>:<port>/<database_name>*. The *database_type* can be postgresql or oracle.
clusters.nodes.disable_host_association	N/A	Set to *true* to prevent this database being associated with the host on which the agent is running. For example, if the agent is running on a different server.
clusters.nodes.tags	N/A	Assign tags for resource labeling. Encase in double quotes and introduce each tag with a dash.
clusters.nodes.settings.enabled	N/A	Set to *true* to enable collecting database configuration data.
clusters.nodes.metrics.[...]	N/A	Controls *Fine-tuning monitoring data collection* the agent ingests, and how they are ingested.

These settings are relevant to host data. All `host.*` parameters are nested under `provider.onprem` in the YAML file.

YAML database settings	Placeholder	Guidelines
host.metrics.disabled	N/A	Set to *true* to disable the collection of host metrics from the host on which the agent is located. For example, when the agent host is not the same as the database cluster host.

continues on next page

Table 70 – continued from previous page

YAML database settings	Placeholder	Guidelines
host.resource_id	*<host_id>*	Assign an identifier to the host on which the agent is running, by default this will be populated by the hostname detected during setup. This will link your host machine to an identifier. Encase in double quotes. Must be unique within the project.

These settings are relevant to schema ingestion. `clusters.nodes.schema.*` parameters are nested under `provider.onprem` in the YAML file.

YAML database schema settings	Placeholder/Value	Guidelines
provider.onprem.schema_export_max_workers	N/A	Maximum number of workers the EDB Postgres AI agent starts in parallel to extract schema from the source database server(s). You can control CPU, memory and disk space usage by the EDB Postgres AI agent witht this parameter. See *Performing multiple concurrent schema ingestions* for more information. The default is set to 10.
clusters.nodes.schema.enabled	N/A	Set to *true* to enable schema ingestion for the database. Set to *false* to disable it.
clusters.nodes.schema.poll_interval	N/A	Defines how frequently the agent checks for schema changes. Provide a value in seconds or minutes (for example, 15 s or 1 m).
clusters.nodes.schema.filter	N/A	Use this block to specify which schemas to include or exclude from ingestion. It contains the *mode* and *names* parameters.
clusters.nodes.schema.filter.mode	*include* or *exclude*	Specifies the filtering behavior. Use *include* to only ingest listed schemas, or *exclude* to ingest all schemas except those listed.
clusters.nodes.schema.filter.names	*<schema_name>*	A list of strings representing the names of the schemas to be filtered. This list works in conjunction with the mode setting.

Related topics

- *Fine-tuning monitoring data collection* ► Learn how to customize the monitoring features and enable/disable a subset of them.
- *Configuring multiple databases* ► Learn how to use the same agent config file to connect and fetch data from more than one database.
- *Monitoring the EFM Cluster* ► Learn how to use the same agent config file to monitor the EDB Failover Manager (EFM) Cluster.
- *Adding usage reporting for Hybrid Manager-monitored databases* ► Learn how to complement monitoring efforts by providing EDB with basic insights into product usage.
- *Administrative tasks* ► Learn how an HM instance's administrator can obtain the values for the root certificate and `beacon_server` .

- *Use the agent to assess your Oracle database for migration* ▶ Use the EDB Postgres AI agent to asses Oracle databases and extract schema for conversion with the Migration Portal.
- *Perform multiple concurrent schema ingestions* ▶ Adjust the EDB Postgres AI agent configuration to tune EDB Postgres AI agent's resource utilization and support the ingestion of a large number of databases.

Monitoring the EFM Cluster

To monitor an EDB Failover Manager, deploy one EDB Postgres AI agent per efm node (excluding the witness node). EDB Postgres AI agent leverages the `efm` cli and run the `efm cluster-status-json <cluster_name>` command to collect the monitoring info of the failover manager.

Configuring clusters and nodes with manager type as `efm`

- Set manager type as `efm`
- Use the `cluster_name` field to specify the name of the Failover Manager cluster. The cluster name is the prefix of the name of the cluster properties file. For example, if your cluster properties file is named efm.properties, your cluster name is efm.

```
#  efm node1 beacon agent config

...
clusters:
  - resource_id: selfmanaged_efm_cluster_id
    name: "<pg_cluster_name1>"
    manager: "efm"
    efm:
      cluster_name: "<efm_cluster_name>"
    nodes:
      - resource_id: <efm_node_resource_id_1>
        dsn: "$DSN1.1"
        ...
```

Since we need to deploy one EDB Postgres AI agent per efm node, the nodes list will contain only one entry.

```
#  efm node2 beacon agent config

...
clusters:
  - resource_id: selfmanaged_efm_cluster_id
    name: "<pg_cluster_name1>"
    manager: "efm"
    efm:
      cluster_name: "<efm_cluster_name>"
    nodes:
      - resource_id: <efm_node_resource_id_2>
        dsn: "$DSN1.2"
        ...
```

Note

Ensure the *clusers* - *resource_id* is same in all the agent config of efm nodes.

Fine-tuning monitoring data collection

The template provided in *Configuring the agent* enables all monitoring features. You can fine-tune the features that are enabled.

Time-series metrics (the Monitoring tab)

The `general.metrics.push` section of the configuration file controls pushing time series metrics back to the Prometheus instance in Hybrid Manager (HM). These metrics populate the **Monitoring** tab. To disable this, set `general.metrics.push.enabled` to `false`.

Postgres logs (the Logs tab)

Enable log collecting by setting `general.metrics.push.include_logs` to `true`. Disable it at the agent level by setting this value to `false`. In addition, set `logs.disabled` to `false` for each entry in `provider.onprem.clusters.nodes` for which you want to collect Postgres logs.

By default, the agent attempts to discover the location of the log files by calling the Postgres function `pg_current_logfile()`. If you want to explicitly specify the location of the log files because, for example, you configured your Postgres instance to store logs at a different location, substitute `<logs_directory>` with the path to the directory where you're storing log files.

For example, to collect logs with the extension `.log` from filepath `/var/log/postgres-lc`, you can use the pattern `/var/log/postgres-lc/*.log`.

```
clusters:
  - resource_id: selfmanaged_postgres
    ...
    nodes:
      - resource_id: selfmanaged_postgres_node1
        ...
        logs:
          disabled: false
          mode: file
          file:
            paths: <logs_directory>/*log
```

Diagnostics (Query Diagnostics and Recommendations tabs)

The `metrics.stats` section under each entry in `provider.onprem.clusters.nodes` controls collecting more detailed diagnostic data for query analysis and optimization.

You can disable all such data collection by setting `metrics.stats.enabled` to `false`. Alternatively, you can enable or disable each element by setting the appropriate `enabled` parameter:

- `metrics.stats.wait_states` controls the collection of query data based on sampling the session state from EDB Wait States.
- `metrics.stats.query_text` controls the collection of query statistics from EDB Stat Monitor.

If both of these options are disabled, the **Query Diagnostics** tab becomes disabled.

- `metrics.stats.recommendations` controls collecting index and statistics recommendations from EDB Query Advisor. If disabled, the **Recommendations** tab doesn't display index and statistics recommendations, but still displays other recommendations.
- `metrics.stats.query_text` controls collecting query texts from both EDB Wait States and EDB Stat Monitor. If disabled, no query texts are collected and you see only query ID in **Query Diagnostics**.

Configuring multiple databases

To leverage full observability capabilities, we recommend deploying one agent per host machine with self-managed database clusters. Then, you can configure this single agent to connect to multiple database clusters on the same host.

Configuring multiple clusters and nodes

The agent uses the `clusters` section to define all monitored database clusters on the host.

- **Cluster entries**: Each item in the clusters list represents a single database cluster (for example, a primary database and its replicas, or a single-node instance).
- **Nodes entries**: Within each cluster, the nodes list specifies the individual database instances that belong to that cluster.

For a single-instance database, the nodes list will contain one entry.

For multi-node clusters (like a primary/replica setup), the nodes list can contain an entry for every instance.

```
clusters:
  - resource_id: selfmanaged_postgres_1
    name: "<instance_name1>"
    manager: "other"
    nodes:
      - resource_id: <node_resource_id_1.1>
        disable_host_association: false
        dsn: "$DSN1.1"
        ...
      - resource_id: <node_resource_id_1.2>
        disable_host_association: false
        dsn: "$DSN1.2"
        ...
  - resource_id: selfmanaged_postgres_2
    name: "<instance_name2>"
    manager: "other"
    nodes:
      - resource_id: <node_resource_id_2.1>
        disable_host_association: true
        dsn: "$DSN2.1"
    ...
```

While a single agent instance can track databases across different host machines, that setup provides limited host-level metrics for databases not on the same machine as the agent.

> **Note**
>
> Currently, monitoring data is collected only from the database specified in the DSN. For example, if you specify the *postgres* database, then the database size reflects only the size of this database. To monitor other databases in the same Postgres instance, you must specify them individually the same way you specify multiple instances.

Schema change detection

Schema change detection automatically identifies modifications to your database schema such as added or removed tables, altered columns, new or dropped indexes, constraint changes and reports them as activity events in the Hybrid Manager (HM) console.

Schema change detection operates in two stages:

1. **Schema collection:** The EDB Postgres AI Agent periodically polls the monitored database and sends the current schema state to the beacon server. The poll frequency is controlled by `schema.poll_interval` (default: `1m`).
2. **Change detection:** The beacon server compares each incoming schema snapshot against the previously stored version. When differences are found, the server emits schema change events to the activity service (`accm-server`), which records them in the HM Activity Log.

A **10-minute grace period** applies after a database is first registered. During this window, newly discovered tables aren't reported as changes, and this prevents the initial schema ingestion from generating a flood of "table added" events.

Detected change types

- Table added or removed
- Column added, removed, or modified (data type, nullability, default value)
- Index added or removed
- Constraint added or removed
- Table type changed

Enabling schema change detection by deployment type

Schema change detection requires schema collection to be active. How you enable it depends on your deployment type:

On-prem databases

For databases monitored with the on-prem EDB Postgres AI Agent, schema collection must be **explicitly enabled** in your `beacon-agent.yaml` . Set `schema.enabled` to `true` for each node entry:

```
provider:
  onprem:
    clusters:
      - resource_id: "<cluster_resource_id>"
        nodes:
          - resource_id: "<node_resource_id>"
            dsn: $DSN
            schema:
              enabled: true
              poll_interval: 1m
```

Once schema collection is enabled, change detection is automatic on the server side irrespective of deployment type, and no additional configuration is required. If `schema.enabled` isn't set or is `false` , no schema data is collected and changes can't be detected.

HM-managed clusters and external CNPG clusters

For HM-managed Postgres clusters and external CloudNativePG clusters, the EDB Postgres AI Agent runs as a sidecar container alongside the database. Schema collection is **enabled by default**.

External cloud databases such as RDS

For external cloud-hosted databases monitored via the EDB Postgres AI Agent, follow the same configuration as on-prem databases. Set `schema.enabled: true` in the EDB Postgres AI Agent configuration for the relevant node entry.

Limitations

- The EDB Postgres AI Agent detects changes to **tables and columns** while running. Changes to views or procedures aren't detected until you restart the EDB Postgres AI Agent.
- Schema change detection relies on the activity service (`accm-server`) being available to receive events. See *Activity service unavailability* for details.

Activity service unavailability

Schema change events and other beacon telemetry are delivered to the platform through the activity service (`accm-server`). If the activity service is degraded or unavailable, events fail to send from the EDB Postgres AI Agent. This is expected platform behavior and not a product defect. Event delivery resumes automatically once the activity service recovers.

> **Note**
>
> No schema change data is lost on the monitored database itself. Only the reporting of detected changes to the HM console is delayed until the activity service is available again. The EDB Postgres AI Agent continues to collect schema snapshots, and the server detects and reports changes once the activity service recovers.

Controlling schema ingestion

The *beacon-agent.yaml* allows the Agent to ingest all schemas that the migration user has permissions to and contain at least one table.

You can add further configurations to control schema ingestion.

Defining a subset of schemas for ingestion

If you don't want the Agent to ingest all schemas, you can define a subset with the `databases.schema.filter` option.

Using the schema filter

To control which schemas the Agent ingests from your database, add the `filter` parameter within the `databases.schema` configuration block in your `beacong-agent.yaml` . This allows you to specify a set of schemas to either include or exclude.

The filter parameter requires two settings:

- `mode` : Defines the filtering behavior. It must be set to either `include` or `exclude` . When set to `include` , the Agent only ingests schemas listed in the names list. When set to `exclude` , the Agent ingests all schemas except those listed in the names list.
- `names` : A list of schema names to apply the filter to. Because the names are case-sensitive, ensure you use the same casing for the names as how the schemas are stored in the source database.

Note

The Agent has a default *Excluded system schemas* for both Oracle and Postgres that are automatically excluded from ingestion because they are irrelevant for the database migration. When an *exclude* filter is used, this default list is combined with any additional schemas you specify. For the *include* filter, the Agent will ingest any schemas you explicitly name, including system schemas that would have otherwise been excluded by default.

Include example

```
[...]
provider:
  onprem:
    clusters:
      - resource_id: "<cluster_resource_id>"
        [...]
        nodes:
          - resource_id: "<node_resource_id>"
            dsn: $DSN
            schema:
              enabled: true
              poll_interval: 15s
              filter:
                mode: "include"
                names:
                  - "<schema_name1>"
                  - "<schema_name2>"
            tags:
              - "<tag_names>"
```

Exclude example

```
[...]
provider:
  onprem:
    clusters:
      - resource_id: "<cluster_resource_id>"
       [..]
       nodes:
         - resource_id: "<node_resource_id>"
           dsn: $DSN
           schema:
             enabled: true
             poll_interval: 15s
             filter:
               mode: "exclude"
               names:
                 - "<schema_name1>"
                 - "<schema_name2>"
           tags:
             - "<tag_names>"
```

Excluded system schemas

The Agent excludes the following system schemas by default.

Postgres and EDB Postgres Extended Server

- information_schema
- pg_catalog
- pg_toast

EDB Postgres Advanced Server

- sys

Oracle

- ANONYMOUS
- APEX_PUBLIC_USER
- APEX_030200
- APEX_040000
- APEX_040200
- APPQOSSYS
- AUDSYS
- CTXSYS
- DMSYS
- DBSNMP
- DBSFWUSER
- DEMO
- DIP
- DVF
- DVSYS
- EXFSYS
- FLOWS_FILES
- FLOWS_020100
- FRANCK
- GGSYS
- GSMADMIN_INTERNAL
- GSMCATUSER
- GSMROOTUSER
- GSMUSER
- LBACSYS
- MDDATA
- MDSYS

- MGMT_VIEW
- OJVMSYS
- OLAPSYS
- ORDPLUGINS
- ORDSYS
- ORDDATA
- OUTLN
- ORACLE_OCM
- OWBSYS
- OWBYSS_AUDIT
- PDBADMIN
- RMAN
- REMOTE_SCHEDULER_AGENT
- SI_INFORMTN_SCHEMA
- SPATIAL_CSW_ADMIN_USR
- SPATIAL_WFS_ADMIN_USR
- SQLTXADMIN
- SQLTXPLAIN
- SYS
- SYSBACKUP
- SYSDG
- SYSKM
- SYSRAC
- SYSTEM
- SYSMAN
- TSMSYS
- WKPROXY
- WKSYS
- WK_TEST
- WMSYS
- XDB
- XS$NULL

Performing multiple concurrent schema ingestions

When you configure the EDB Postgres AI agent to ingest schemas from several Oracle databases, you can use the `schema_export_max_workers` parameter of the `beacon-agent.yaml` configuration file to control how many schema export jobs the EDB Postgres AI agent runs in parallel.

The `schema_export_max_workers` sets the maximum number of concurrent schema exports the EDB Postgres AI agent runs across all configured Oracle databases. Because active workers consume CPU, memory, and some disk space on the auxiliary machine that runs EDB Postgres AI agent, you can use this parameter to tune resource consumption, while speeding up or slowing down the bulk schema ingestion.

The `schema_export_max_workers` is in the `provider.onprem` section of the `beacon-agent.yaml` :

```
... # other beacon-agent.yaml configurations
provider:
  onprem:
    runner:
      enabled: true
    schema_export_max_workers: 10
```

Using the max workers parameter

Start with the default `schema_export_max_workers: 10` when configuring initial schema ingestion. Then, monitor the auxiliary machine running the EDB Postgres AI agent (CPU, memory, disk I/O, network).

If the auxiliary machine is underutilized and you need faster ingestion for multiple databases, or if you notice that the databases are taking too long to appear in the **Estate > Migrations** tab, increase the value and observe the impact.

If resource usage is consistently high or causing contention (for example, you notice spikes in CPU, memory, and disk usage), decrease the value and confirm that the system stabilizes. This slows down total ingestion time but reduces the peak resource usage on the EDB Postgres AI agent host.

After each change, restart the Agent if it is not running as a service with automatic reload.

Adding usage reporting for Hybrid Manager-monitored databases

You can complement your monitoring efforts by providing EDB with valuable insights into product usage, all from a unified agent deployment. When usage reporting is enabled, the agent collects basic usage data to enable EDB to better understand our customers. It sends data to EDB only if you enable it.

In internet-connected environments

1. In the *agent configuration example* , the `beacon_agent.yaml` file already has `general.metrics.usage.output.file.enabled` set to `true` . This setting allows the agent to save a local copy of the usage data in the file `beacon_usage.json` .
2. Review this file to ensure that the information is accurate and doesn't contain any data you don't want to share with EDB.
3. Enable reporting to EDB by setting `general.metrics.usage.output.http.enabled` to `true` .

In air-gapped environments

1. In the *agent configuration example* , the `beacon_agent.yaml` file already has `general.metrics.usage.output.file.enabled` set to `true` . This setting allows the agent to save a local copy of the usage data in the file `beacon_usage.json` .

2. Review this file to verify that the information is accurate and doesn't contain any data you don't want to share with EDB.
3. Copy the `beacon_usage.json` report file to a machine that has internet access.

Running the agent

Run the agent:

```
beacon-agent
```

> **Tip**
>
> See Agent CLI for other agent modes and options.

The agent starts sending data to Hybrid Manager (HM). Follow the logs to monitor the data ingestion process. After a few minutes, the data from the agent appears in the HM console:

- In the **Estate** section, the **Postgres** tab lists your Postgres databases along with summarized database information. Selecting one of the databases provides a more detailed database overview and monitoring capabilities.
- The **Estate > Migrations** tab lists your external databases (for example, Oracle) along with summarized database and migration information. Selecting one of the databases provides a more detailed migration assessment and migration capabilities.

> **Important**
>
> The agent detects schema changes to tables and columns while running. However, changes to views or procedures aren't detected until you restart the agent.

For production use, configure the agent to *run as a service* .

Running the agent as a service

For continuous metric streaming and to ensure your database remains visible in your Hybrid Manager (HM) estate, configure the agent to run as a service. Running as a service allows it to start automatically on system startup and restart after any errors.

If you don't set up the agent as a service, it streams metrics only when you manually run it. Furthermore, your database will disappear from your HM estate 72 hours after the agent stops streaming.

Finish installing and configuring the agent before setting it up as a service.

> **Note**
>
> Future versions of the agent package may set this up automatically.

This example configures the agent to run on an RHEL 9 machine. However, we provide operating system aids so you know when to replace the values.

1. Create a file using your text editor of choice. This example command uses the vi editor:

```
sudo vi /usr/lib/systemd/system/beacon-agent.service
```

Other default system directories on AWS EC2 instances are:

Operating system	Default system directory
RHEL	/usr/lib/systemd/system/beacon-agent.service
Rocky Linux	/usr/lib/systemd/system/beacon-agent.service
AlmaLinux	/usr/lib/systemd/system/beacon-agent.service
Ubuntu	/etc/systemd/system/beacon-agent.service
Debian	/lib/systemd/system/beacon-agent.service

1. Paste the following template into the `beacon-agent.service` file and replace the information in placeholders `< >`.

```
# TEMPLATE

[Unit]
Description=Agent

After=network.target

[Service]
Type=simple

User=<user>

WorkingDirectory=</home/user/>

ExecStart=</usr/local/bin/beacon-agent>

Restart=on-failure
RestartSec=60

[Install]
WantedBy=multi-user.target
```

Service settings	Placeholder to replace	Guidelines
User=	*<user>*	Provide the name of the system user that will run the agent.
WorkingDirectory=	*</home/user/>*	Set the working directory to where you stored the *.beacon* directory. The path depends on the operating system.
ExecStart=	*</usr/local/bin/beacon-agent>*	Provide the path to the directory where the agent was installed. The path depends on the operating system.

1. Run the following commands to reload, enable, and start your new service. These commands apply to all mentioned operating systems:

```
sudo systemctl daemon-reload
sudo systemctl enable beacon-agent.service
sudo systemctl start beacon-agent.service
```

1. Your agent is now running as a service. Check the logs:

```
journalctl -u beacon-agent.service
```

Troubleshooting

If you're having issues configuring the EDB Postgres AI Agent or are getting errors when trying to run it, you can troubleshoot. Test each EDB Postgres AI Agent feature in isolation to narrow down the source of the issue.

Testing connectivity from the EDB Postgres AI Agent to Hybrid Manager

To ensure there's a connection between the EDB Postgres AI Agent and Hybrid Manager (HM), run the EDB Postgres AI Agent:

```
beacon-agent --config
```

See EDB Postgres AI Agent CLI for other EDB Postgres AI Agent modes and options.

If the output displays no errors, your connection to HM is working properly. The issue might have to do with the connection to your databases.

If the output displays errors, your connection to HM isn't working properly. Check the syntax of the YAML file, the access key, the CA path and contents, DSN string, and project ID one after the other to ensure the settings are correct.

Error messages and solutions

`"fail fetching server version: pq: SSL is not enabled on the server"`

If the database you are connecting to has SSL disabled, you must explicitly instruct the EDB Postgres AI Agent to use an unencrypted connection. To do this, append the setting `?sslmode=disable` to the DSN used in the *EDB Postgres AI Agent's configuration* , for example:

```
export DSN="postgresql://postgres:password@localhost:5432/postgres?sslmode=disable"
```

Schema change events not appearing in Hybrid Manager

If schema change events or other EDB Postgres AI Agent telemetry aren't appearing in the HM console, the activity service (`accm-server`) may be degraded or unavailable.

To check the `accm-server` pod status, run:

```
kubectl get pods -A | grep accm
upm-beaco-ff-base                accm-server-8c5cc7676-f2zv2                                ␣
↪           2/2     Running     0             8h
```

Symptoms:

- Schema changes aren't reflected in the HM console despite the EDB Postgres AI Agent running normally.
- Other event-driven features (such as new query detection) also aren't updating.
- The EDB Postgres AI Agent logs show errors similar to the following while sending events to the platform.

```
{"level":"warn","error":"rpc error: code = Unavailable desc = no healthy upstream",
↪"project_id":"prj_xxxx","cluster_id":"p-xxxxx","table":"edb_admin.public.test_18653
↪","time":1781518934,"msg":"failed to emit schema change activity"}
```

Resolution:

This is expected behavior when the activity service is down, it's not a defect in the EDB Postgres AI Agent or the monitored database. The EDB Postgres AI Agent continues to detect changes locally and queues events for delivery. Once the activity service recovers, queued events are delivered automatically and the HM console updates.

If the issue persists, verify that the `accm-server` pod is running and healthy in your HM Kubernetes deployment.

Administrative tasks

To configure the agent, users need access to some resources that they can't obtain by themselves. However, these resources are available to administrators, as they have the rights and permissions to configure certificate and host-related settings while installing Hybrid Manager (HM) or after.

What is the `beacon_server` ?

To configure the agent, users must set the `beacon_server` value in the `beacon_agent.yaml` file. This is a string consisting of two values: host and port.

The host is the URL provided in the Helm chart configuration file used for installation or `values.yaml` . Look up the value set for `upm-beacon.server_host` (or `BEACON_SERVICE_DOMAIN_NAME`) to obtain the URL. The port is fixed as 9443.

What is the root certificate authority (CA)?

The agent makes a secure connection to the `beacon_server` URL using TLS. You therefore need the machine on which the agent is running to trust the server certificate provided by `beacon_server` . Generally, this trust is already managed by your organization's existing security infrastructure. The certificate authority (CA) responsible for signing your internal server certificates must be configured as a trusted CA across your network. This ensures that servers within your environment, including the one running the agent, inherently trust certificates issued by this CA.

If this isn't the case, users can set the `root_ca_path` value in the `beacon_agent.yaml` file. This is a directory where the user stored the root certificate, which is capable of validating the EDB Postgres AI Platform server certificate. However, non-admin users don't have access to the certificate, so you need to provide it to them.

To see the Kubernetes secret containing the root certificate:

```
#  The certificate is base64 encoded as the data field "public.crt"

kubectl get secret -n upm-beacon beaconator-ca-bundle -o json
```

For a more readable output that prints only the root certificate, ensure you have `jq` and `base64` installed and run:

```
#  If `jq` and `base64` are available

kubectl get secret -n upm-beacon beaconator-ca-bundle -o json | jq -r .data."public.
↪crt" | base64 --decode
```

This command outputs the contents of the certificate. Send them to the user that requires them through a secure channel. The user must store it locally in crt format.

Data security and sovereignty

When you deploy the agent to monitor your external databases with Hybrid Manager (HM), you retain control over the data collected for monitoring and assessment, aligning with your security and compliance requirements.

Limited data collection

The agent gathers the following data:

- General database metadata to display metrics in the **Estate** view.

For Postgres databases:

- Database logs and metrics to provide observability features via the *External cloud-hosted OCR models fail to register via the HM console* and *Grafana dashboard setup for Agent Factory metrics* .

For Oracle databases:

- Database data to generate the automatic assessment of migration complexity and level of effort displayed in **Estate** section in the **Migrations** tab.
- Schema DDL for eventual retrieval and use by Migration Portal if you decide to use that tooling for schema conversion.

> **Note**
>
> Currently, migration assessment data and schema DDL data are collected only for Oracle databases. Later releases will also gather migration assessment and schema DDL data for external Postgres and cloud service provider (CSP) databases.

We don't collect data beyond what's strictly necessary for these purposes.

Data storage location

Your collected data stays within your chosen environment. HM respects your data sovereignty. For example, if you deployed HM on your own premises, all collected data from your monitored databases remains within your on-premises infrastructure. If your HM instance is deployed on a cloud platform like EKS, the collected data will reside securely in your specific cloud environment.

Installing the Agent

See *Installing Vault* for instructions on how to install the Agent.

Monitoring cloud service provider-managed Postgres deployments

You can easily monitor PostgreSQL databases running outside Hybrid Manager (HM) in an environment managed by a cloud service provider (CSP) with full estate representation support.

Currently, you can monitor the following external CSPs:

- *Monitoring AWS RDS deployments*

Monitoring AWS RDS deployments

You can monitor your Amazon Web Service (AWS) Relational Database Service (RDS) deployments with Hybrid Manager (HM).

Prerequisites

You must have access to the Kubernetes environment where HM resides and also have sufficient administrator privileges to change the settings on the entire HM cluster.

This access requirement is different from other external database monitoring situations with HM, such as *Monitoring self-managed deployments* deployments of Postgres, where fewer privileges are required.

You must also ensure that the user credentials stored in a Kubernetes secret, for example `example-beacon-csp-credentials` , have sufficient permissions for the AWS RDS environment being monitored. Without these permissions, the system can't retrieve all available metrics.

To do this, you need to obtain an AWS access key of an IAM user with the following minimal permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowInstanceAndLogDescriptions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles"
            ],
            "Resource": [
                "arn:aws:rds:*:*:db:*"
            ]
        },
        {
            "Sid": "AllowMaintenanceDescriptions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribePendingMaintenanceActions"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowGettingCloudWatchMetrics",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowRDSUsageDescriptions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeAccountAttributes"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowQuotaDescriptions",
            "Effect": "Allow",
```

(continues on next page)

(continued from previous page)

```
            "Action": [
                "servicequotas:GetServiceQuota"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowInstanceTypeDescriptions",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowInstanceFilterByTags",
            "Effect": "Allow",
            "Action": [
                "tag:GetResources"
            ],
            "Resource": "*"
        }
    ]
}
```

Setting up EDB Postgres AI agent to monitor AWS RDS deployments on HM

You can set up the agent to monitor all your AWS RDS deployments on HM.

> **Note**
>
> Once you register an RDS region, the system adds every database you have in that RDS region. This can result in several resources being added to your estate. They are all under one project, so you either have access as a user to all the RDS databases or you don't have access to any of them.

1. Add your `example-beacon-csp-credentials` secret.

Add your `example-beacon-csp-credentials` secret in your HM Kubernetes cluster's `upm-beacon` namespace.

Example `yaml` sample:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
    name: example-beacon-csp-credentials
    namespace: upm-beacon
stringData:
    AWS_ACCESS_KEY_ID: <your_access_key_id>
    AWS_SECRET_ACCESS_KEY: <your_secret_access_key>
    AWS_SESSION_TOKEN: <your_session_token>
    AWS_REGION: example-us-east-1
    BEACON_PROVIDER_AWS_CONSTANT_REGIONS: example-us-east-1
```

(continues on next page)

(continued from previous page)

```
    BEACON_PROVIDER_AWS_CONSTANT_PROJECT_ID: <your_project_id>
    BEACON_AGENT_PROVIDERS: appliance,aws
    BEACON_PROVIDER_AWS_MODE: constant
    BEACON_PROVIDER_AWS_CONSTANT_SERVICES: rds
EOF
```

2. Restart the agent and external metrics receiver:

```
kubectl -n upm-beacon rollout restart deployment upm-beacon-agent-k8s
kubectl -n upm-beacon rollout restart deployment upm-beacon-external-metrics-receiver
```

You now see your AWS RDS deployments on your **Estate** page. After the databases start populating, you can see five metrics:

- CPU
- Network throughput
- Connections
- Disk IOPS
- Disk throughput

Disabling AWS RDS monitoring in HM

You can turn off monitoring of your AWS RDS deployments.

1. Delete your `beacon-csp-credentials` secret:

```
kubectl -n upm-beacon delete secret beacon-csp-credentials
```

2. Restart the agent and external metrics receiver:

```
kubectl -n upm-beacon rollout restart deployment upm-beacon-agent-k8s
kubectl -n upm-beacon rollout restart deployment upm-beacon-external-metrics-receiver
```

You no longer see your AWS RDS deployments on your estate and are no longer monitoring them with HM.

Monitoring standalone Kubernetes clusters

Use the EDB Postgres AI agent for centralized monitoring, metrics collection, and resource management for your EDB Postgres® AI for CloudNativePG™ or CloudNativePG™ clusters and integrate them on your Hybrid Manager (HM) console. By integrating these clusters with your HM console, you gain a unified view of your cloud-native environment. This integration is deployed using a Helm chart.

You can configure the EDB Postgres AI agent to run anywhere. To activate specific capabilities, assign one or more roles to it: Webhooks, Providers, Dispatchers, or Runners. You can configure a single agent to perform any combination of these functions simultaneously.

Architecture

The architecture to implement monitoring Kubernetes deployments consists of the following components.

EDB Postgres AI agent

The EDB Postgres AI agent serves as a centralized tool for resource discovery and monitoring, while also streamlining the collection and forwarding of your metrics.

OpenTelemetry collector (optional)

The OpenTelemetry (OTEL) collector acts as a centralized data pipeline that gathers logs and exports telemetry data from your environment.

Key functions:

- Container metrics (cAdvisor): The cAdvisor service collects and processes resource usage data (CPU, memory) from active containers.
- Kubelet metrics: Monitors the health and performance of Kubernetes nodes and pods.
- Application logs: Aggregates and forwards logs from your database applications.
- Custom metrics: Allows for the collection of user-defined metrics tailored to specific application needs.
- (Optional) Kube state metrics: Generates metrics based on the state of Kubernetes objects.

Requirements to install the EDB Postgres AI agent

This section lists the requirements to configure your Hybrid Manager (HM) installation to monitor external CloudNativePG™, EDB Postgres® AI for CloudNativePG™ Cluster, and EDB Postgres® AI for CloudNativePG™ Global Cluster clusters.

HM console requirements

You need access to a HM console instance with:

- Machine user: After your HM cluster is ready, *Create a machine user* .
- Estate Ingest Role: Assign the *estate ingest role* to the machine user you created.
- Access key: Save the access key generated during the machine user creation.
- Project ID: Note the project ID where the agent will report metrics.

Required executables

You will need the following executables to run the commands in this documentation:

- kubectl
- helm
- envsubst

The `cnp` and `ns` plugins for kubectl are also recommended.

Supported operators

You must have an existing external Kubernetes cluster (other than the Kubernetes cluster where HM is installed) with a Postgres cluster managed by one of the following operators:

-

EDB Postgres® AI for CloudNativePG™ Cluster (formerly EDB Postgres for Kubernetes)

-

EDB Postgres® AI for CloudNativePG™ Global Cluster (formerly EDB Postgres Distributed for Kubernetes)

- CloudNativePG™

Kubernetes cluster requirements

Ensure the Kubernetes cluster where you are running the Postgres databases also aligns with these requirements.

- Kubernetes version: 1.21 or higher.
- Helm version: 3.8 or higher.
- Cluster access: Admin privileges to create namespaces, RBAC resources, and deployments.
- Required Kubernetes operators to monitor the metrics.

Cert-manager

You must also have cert-manager installed in your cluster to complete the deployment.

To install cert-manager:

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.9.
↪1/cert-manager.crds.yaml
```

Required environment values

Export the following environment values that you will need to complete deployment. You can skip this step if you prefer to provide the environment values explicitly during the deployment.

- `TARGET_NS` - The namespace of your existing CNPG cluster (or PGD Group for PGD4K).
- `AGENT_NS` - The namespace to install the EDB Postgres AI agent.
- `HM_ACCESS_KEY` - The access key for Hybrid Manager.
- `HM_PROJECT_ID` - The Project ID in HM.
- `HM_CACERT` - The root CA certificate of HM. Required when `rootCAEnabled` is `true` (the default), which is the configuration described in this guide. It works with any certificate provided from the chain. If you instead set `rootCAEnabled: false` to trust HM via the system CA pool (or skip verification), you can omit this value and the `beacon-server-ca` secret in *Namespace and secrets* — see the *TLS configuration options* .
- `HM_URL` - the beacon server URL including port
- `EDB_REPO` - the name of the EDB repo matching your subscription: enterprise, standard or community360
- `EDB_SUBSCRIPTION_TOKEN` - Your EDB token.
- `POD_SELECTOR` - Set to `k8s.enterprisedb.io/podRole=instance` for EDB operators, or `cnpg.io/podRole=instance` for the community CNPG operator.
- `AGENT_VERSION` - Set to v1.337.18 for HM 2026.2.

For example:

```
export TARGET_NS=cnpg
export AGENT_NS=upm-beacon
export HM_ACCESS_KEY=baak_1DMvAduJeigwIcCJyk2pc6BxllJxAbjFa5Nlc1c680lqSj0sj
export HM_PROJECT_ID=prj_3AlRuOvlbUZafFY2
```

(continues on next page)

(continued from previous page)

```
export HM_CACERT=$(cat source/k8s-agent/ca.cert)
export HM_URL=beacon-pm-test.edbhcp.com:9443
export EDB_REPO=enterprise
export POD_SELECTOR=k8s.enterprisedb.io/podRole=instance
export AGENT_VERSION=v1.337.18
```

Namespace and secrets

Create the following namespaces and secrets before installation:

- Namespace: The logical partition `${AGENT_NS}` used to isolate the EDB Postgres AI agent.
- Beacon server CA Secret: Contains the TLS certificate for connecting to HM.
- Beacon access key aecret: Contains the HM access key for the cluster.
- Image pull secret: Contains credentials for pulling the agent and OpenTelemetry (OTEL) collector images.

Create the namespaces and secrets required for the configuration:

```
kubectl create namespace "${AGENT_NS}"
kubectl create secret generic beacon-server-ca --from-literal=ca.crt="${HM_CACERT}" -
↪n "${TARGET_NS}"
kubectl create secret generic beacon-server-ca --from-literal=ca.crt="${HM_CACERT}" -
↪n "${AGENT_NS}"
kubectl create secret generic beacon-access-key --from-literal=access-key="${HM_
↪ACCESS_KEY}" -n "${TARGET_NS}"
kubectl get secrets -n "$AGENT_NS" -oyaml beacon-agent-pull-secret | grep -v ^\
↪s*namespace:\s | kubectl apply -n "$TARGET_NS" -f -
```

Note

The two *beacon-server-ca* secrets are only required when *rootCAEnabled* is *true* (the default). If you set *rootCAEnabled: false* , skip creating them — see the *TLS configuration options* . !!!

Installing the EDB Postgres AI agent

Installation methods

Important

Set the *Required environment values* before proceeding with the installation of the EDB Postgres AI agent for external Kubernetes cluster.

Environment	Configuration	OpenShift Flag
KIND, GKE, EKS, AKS	Uses image defaults (numeric or named)	openshift.enabled: false (default)
OpenShift (RHOS)	Compatible via runAsUser: null (SCC assigns ID)	openshift.enabled: false/true (optional)

Key points:

- **Universal Compatibility:** The chart defaults to `runAsUser: null`, allowing OpenShift SCCs to assign valid UIDs automatically, while standard Kubernetes falls back to the image's default user.
- **Security Context:** runAsNonRoot strategy varies by component to ensure compatibility with standard Kubernetes validation logic vs OpenShift named-user support.
- **OTEL collector:** Uses the image default user (nonroot) without explicit numeric IDs to satisfy both strict K8s validation and OpenShift SCCs.

Downloading the Helm chart and customizing the values file

1. Obtain an EDB subscription token.
2. Download the EDB Postgres AI agent chart from the EDB repository:

```
helm repo add enterprisedb-$EDB_REPO "https://downloads.enterprisedb.com/$EDB_
↪SUBSCRIPTION_TOKEN/$EDB_REPO/helm/charts/"
```

3. Create a `custom-values.yaml` file that includes your Hybrid Manager (HM) access credentials:

```
# Required: Your HM access credentials
agentConfig:
  accessKey: ${HM_ACCESS_KEY}
  projectID: ${HM_PROJECT_ID}
  beaconServer:
    host: ${HM_URL}
    plaintext: false
  metrics:
    push:
      endpoint: ${HM_URL}
  webhook:
    enabled: true
    podSelector: ${POD_SELECTOR}
# Required: Docker registry credentials for EDB images
image:
  repository: docker.enterprisedb.com/${EDB_REPO}/upm-beacon-agent
  tag: ${AGENT_VERSION}
  imageCredentials:
    registry: docker.enterprisedb.com
    username: ${EDB_REPO}
    password: ${EDB_SUBSCRIPTION_TOKEN}

# OTEL collector image
otelImage:
  repository: "docker.enterprisedb.com/${EDB_REPO}/upm-beacon-external-metrics-
↪receiver"
  tag: ${AGENT_VERSION}

# Optional: Enable additional monitoring components
additionalCharts:
  cert-manager:
    install: false
```

(continues on next page)

(continued from previous page)

```
  opentelemetry-collector:
    install: true # Enable for metrics collection
  kube-state-metrics:
    install: true # Enable for Kubernetes state metrics
  prometheus-node-exporter:
    install: true # Enable for node metrics (requires privileged access)

# Security context for the agent
securityContext:
  runAsUser: 65532
  runAsGroup: 65532
podSecurityContext:
  fsGroup: null

# kube-state-metrics uses the image default user
kube-state-metrics:
  podSecurityContext: null

# Required if opentelemetry-collector.install is true:
opentelemetry-collector:
  image:
    repository: "docker.enterprisedb.com/${EDB_REPO}/upm-beacon-external-metrics-
↪receiver"
    tag: ${AGENT_VERSION}
  podSecurityContext:
    seccompProfile:
      type: RuntimeDefault
  securityContext:
    runAsUser: 65532
    runAsGroup: 65532
    runAsNonRoot: true
    allowPrivilegeEscalation: false
    capabilities:
      drop:
        - ALL
```

For other configuration parameters and options, see the *Configuration reference* .

4. Apply the helm chart:

   - For standard Kubernetes (GKE, EKS, AKS, KIND):

```
helm upgrade --install beacon-agent-k8s \
 enterprisedb-$EDB_REPO/beacon-agent-k8s \
 --create-namespace \
 --namespace upm-beacon \
 --values custom-values.yaml
```

```
- For OpenShift clusters, add the `--set openshift.enabled=true`  flag:
```

```
helm upgrade --install beacon-agent-k8s \
 enterprisedb-$EDB_REPO/beacon-agent-k8s \
 --create-namespace \
```

(continues on next page)

(continued from previous page)

```
 --namespace upm-beacon \
 --set openshift.enabled=true \
 --values custom-values.yaml
```

Important

Grant SCC if using OTEL collector:

```
oc adm policy add-scc-to-user privileged \
  system:serviceaccount:upm-beacon:beacon-agent-k8s-opentelemetry-collector
kubectl -n upm-beacon rollout restart daemonset beacon-agent-k8s-opentelemetry-
↪collector-agent
```

```
kubectl get secrets -n "$AGENT_NS" -oyaml beacon-agent-pull-secret | grep -v ^\
↪s*namespace:\s | kubectl apply -n "$TARGET_NS" -f -
```

Deploying the external cluster with monitoring extensions

The EDB Postgres AI agent collects query and wait-state metrics from extensions that must be present in the monitored cluster. Deploy your external cluster with the following `bootstrap.initdb.postInitApplicationSQL` commands in your database (`${PG_DATABASE}`) and `postgresql` settings so the monitoring extensions are created and preloaded when the cluster is initialized.

Note

Because the manifest doesn't set *bootstrap.initdb.owner* , CNPG defaults the owner to the database name. As a result, CNPG creates an unprivileged Postgres role (user) with the exact same name as the target database (*${PG_DATABASE}*) and makes it the owner of that database. The grants in the init SQL reference this role by the *${PG_DATABASE}* name.

- Creates the `edb_internals` schema in `${PG_DATABASE}` and grants `USAGE` on it to the database owner role (`${PG_DATABASE}` , which CNPG creates with the same name as the database) and `pg_monitor` .
- Creates the `edb_stat_monitor` , `query_advisor` , and `edb_wait_states` extensions in that schema.
- Sets the database `search_path` and `timezone` .

The `postgresql` block enables `pg_stat_statements` and preloads `aidb` , `edb_stat_monitor` , `query_advisor` , and `edb_wait_states` via `shared_preload_libraries` .

Important

${PG_IMAGE} must be an EDB Postgres AI image that bundles the *aidb* , *edb_stat_monitor* , *query_advisor* , and *edb_wait_states* extensions. The community manifest uses the upstream CloudNativePG operator API but still requires these EDB extensions.

The image name can be `postgresql` , `edb-postgres-advanced` , or `edb-postgres-extended` , depending on the Postgres distribution you want to run.

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: ${PG_CLUSTER_NAME}
  namespace: ${TARGET_NS}
  labels:
    k8s.enterprisedb.io/beacon: enabled
spec:
  backup:
    target: primary
  plugins:
    - name: barman-cloud.cloudnative-pg.io
      isWALArchiver: true
      parameters:
        barmanObjectName: ${PG_CLUSTER_NAME}-bck
  bootstrap:
    initdb:
      database: ${PG_DATABASE}
      postInitApplicationSQL:
        - CREATE SCHEMA IF NOT EXISTS edb_internals
        - GRANT USAGE ON SCHEMA edb_internals TO ${PG_DATABASE}
        - GRANT USAGE ON SCHEMA edb_internals TO pg_monitor
        - CREATE EXTENSION IF NOT EXISTS edb_stat_monitor WITH SCHEMA edb_internals
        - CREATE EXTENSION IF NOT EXISTS query_advisor WITH SCHEMA edb_internals
        - CREATE EXTENSION IF NOT EXISTS edb_wait_states WITH SCHEMA edb_internals
        - ALTER DATABASE ${PG_DATABASE} SET search_path TO "\$user", public, edb_
↪internals
        - ALTER DATABASE ${PG_DATABASE} SET timezone TO UTC
  imageName: ${PG_IMAGE}
  postgresql:
    parameters:
      pg_stat_statements.max: "10000"
      pg_stat_statements.track: all
    shared_preload_libraries:
      - aidb
      - edb_stat_monitor
      - query_advisor
      - edb_wait_states
  instances: 2
  storage:
    size: 1Gi
  inheritedMetadata:
    labels:
      k8s.enterprisedb.io/beacon: enabled
```

EDB Postgres AI for CloudNativePG (enterprise operator)

The enterprise manifest uses the EDB operator API, adds an `imagePullSecrets` entry for the EDB registry, and additionally grants `USAGE` on the `public` schema to the database owner role (`${PG_DATABASE}`) and `pg_monitor` .

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
```

(continues on next page)

(continued from previous page)

```
metadata:
  name: ${PG_CLUSTER_NAME}
  namespace: ${TARGET_NS}
  labels:
    k8s.enterprisedb.io/beacon: enabled
spec:
  backup:
    target: primary
  plugins:
    - name: barman-cloud.cloudnative-pg.io
      isWALArchiver: true
      parameters:
        barmanObjectName: ${PG_CLUSTER_NAME}-bck
  bootstrap:
    initdb:
      database: ${PG_DATABASE}
      postInitApplicationSQL:
        - CREATE SCHEMA IF NOT EXISTS edb_internals
        - GRANT USAGE ON SCHEMA edb_internals TO ${PG_DATABASE}
        - GRANT USAGE ON SCHEMA edb_internals TO pg_monitor
        - GRANT USAGE ON SCHEMA public TO ${PG_DATABASE}
        - GRANT USAGE ON SCHEMA public TO pg_monitor
        - CREATE EXTENSION IF NOT EXISTS edb_stat_monitor WITH SCHEMA edb_internals
        - CREATE EXTENSION IF NOT EXISTS query_advisor WITH SCHEMA edb_internals
        - CREATE EXTENSION IF NOT EXISTS edb_wait_states WITH SCHEMA edb_internals
        - ALTER DATABASE ${PG_DATABASE} SET search_path TO "\$user", public, edb_
↪internals
        - ALTER DATABASE ${PG_DATABASE} SET timezone TO UTC
  imagePullSecrets:
    - name: edb-pull-secret
  imageName: ${PG_IMAGE}
  postgresql:
    parameters:
      pg_stat_statements.max: "10000"
      pg_stat_statements.track: all
    shared_preload_libraries:
      - aidb
      - edb_stat_monitor
      - query_advisor
      - edb_wait_states
  inheritedMetadata:
    labels:
      k8s.enterprisedb.io/beacon: enabled
  instances: 2
  storage:
    size: 1Gi
```

Apply the manifest with `kubectl apply -f <manifest>.yaml` (use `envsubst` first if you parameterized it with environment values).

If you do use it on OpenShift, the upstream plugin manifest is rejected by the `restricted-v2` SCC because it sets legacy seccomp annotations and `runAsUser: 10001`. Install it with both stripped. With the enterprise (EDB) operator you must also grant the plugin `get` /`list` /`watch` on `clusters` and `patch` /`update` on `clusters/finalizers`

for the `postgresql.k8s.enterprisedb.io` API group; the upstream plugin only grants this on the community `postgresql.cnpg.io` group, so without it the cluster never reconciles (`cannot set blockOwnerDeletion … can't set finalizers`).

This section shows how to patch a single cluster. You can adapt the instructions below to patch many clusters:

1. Define the target resource you wish to patch. Export the TARGET_NAME for your CloudNativePG™ cluster or PGD group (for EDB Postgres® AI for CloudNativePG™ Cluster) to which you wish to add the agent.

```
export TARGET_UID="$(kubectl get "${TARGET_TYPE}" "${TARGET_NAME}" -n "${TARGET_NS}" -
↪o jsonpath={.metadata.uid})"
```

2. Create the configuration patch. This template adds the necessary labels and ensures the agent can pull the required images:

```
PATCH_CONTENT=$(cat <<EOF
metadata:
  labels:
    k8s.enterprisedb.io/beacon: enabled
spec:
  imagePullSecrets:
  - name: beacon-agent-pull-secret
  inheritedMetadata:
    labels:
      k8s.enterprisedb.io/beacon: enabled
      k8s.enterprisedb.io/beacon-project-id: "${HM_PROJECT_ID}"
      k8s.enterprisedb.io/beacon-resource-id: "${TARGET_NAME}-${TARGET_UID}"
EOF
)
```

3. Run the following command to merge the configuration changes into your resource:

```
kubectl patch pgdgroup "${TARGET_NAME}" -n "${TARGET_NS}" --type=merge -p "$PATCH_
↪CONTENT"
```

4. Restart the cluster to bring the agent online.

   - **For CNPG clusters:**

```
kubectl cnp restart "$TARGET_NAME"
```

```
- **For EDB Postgres® AI for CloudNativePG™ Cluster clusters:**
```

Restart each node in the group:

```
kubectl cnp restart region-a-1
kubectl cnp restart region-a-2
kubectl cnp restart region-a-3
```

Configuring security contexts for external Kubernetes clusters

For standard Kubernetes (KIND, GKE, EKS, AKS)

Configure your `custom-values.yaml` with default secure settings which include user and group IDs:

```
securityContext:
  allowPrivilegeEscalation: false
  capabilities:
    drop:
      - ALL
  privileged: false
  readOnlyRootFilesystem: true
  runAsNonRoot: true
  runAsUser: 10000
  runAsGroup: 10000

podSecurityContext:
  seccompProfile:
    type: RuntimeDefault
```

For OpenShift Kubernetes clusters

1. Enable the OpenShift mode to automatically configure security contexts for SCC compatibility.
2. Enable OpenShift support in your `custom-values.yaml` file:

```
openshift:
  enabled: true
```

When the value is set to `openshift.enabled: true`, the chart automatically:

```
- Omits `runAsUser` and `runAsGroup` from the container security contexts, allowing␣
↪SCC to assign them. - Stores other security settings like `capabilities.drop` ,␣
↪`readOnlyRootFilesystem` , etc. - Uses numeric UID 65532 for the OTEL collector␣
↪that is required for distroless image validation.
```

3. Alternatively, specify your configuration via the command line using the `-f custom-values.yaml` flag:

```
securityContext:
  allowPrivilegeEscalation: false
  capabilities:
    drop:
      - ALL
  privileged: false
  readOnlyRootFilesystem: true
  runAsNonRoot: true
```

4. Leave the fields below empty as OpenShift will assign them automatically:

```
runAsUser:
runAsGroup:
```

5. When deploying to OpenShift, ensure the `fsGroup` field remains undefined to allow the platform to manage volume permissions automatically:

```
podSecurityContext:
  seccompProfile:
    type: RuntimeDefault
```

6. To ensure OpenShift compatibility, configure the OTEL collector subchart with a restricted security context:

```
opentelemetry-collector:
  podSecurityContext:
    seccompProfile:
      type: RuntimeDefault
  securityContext:
    runAsNonRoot: true
    allowPrivilegeEscalation: false
    capabilities:
      drop:
        - ALL
```

7. Configure the security context to prohibit root access, manage volume permissions via file system groups, and restrict system calls using the default seccomp profile.

```
securityContext:
  enabled: true
  runAsNonRoot: true
  runAsUser:
  runAsGroup:
  fsGroup:
  seccompProfile:
    type: RuntimeDefault
```

Note

Implementing a standardized security context for your deployments ensures seamless compatibility and enhanced security across diverse container platforms.

```
- Distroless image support: The UID/GID `65532` is the standard "nonroot" user for␣
↪distroless images, which lacks a traditional `/etc/passwd` file.

- Numeric validation: Setting a specific numeric ID is required for the␣
↪`runAsNonRoot: true` directive to verify the user identity correctly.
```

- **OpenShift compatibility: While OpenShift's Privileged SCC allows any UID, using 65532 ensures a consistent, secure identity across both OpenShift and standard Kubernetes environments. .. ::**
  Key configuration notes:

```
- OpenShift ID management: The `runAsUser` , `runAsGroup` , and `fsGroup` fields are␣
↪left empty to allow OpenShift's SCC to automatically assign the correct UID and GID␣
↪ranges for the project.

- Security context activation: The setting `enabled: true` ensures the subchart␣
↪applies the specified security settings rather than falling back to its internal␣
↪defaults.

- Non-root enforcement: The `runAsNonRoot: true` flag remains active to guarantee␣
↪that the container never runs with root privileges, even when IDs are dynamically␣
↪assigned by the platform.

- Seccomp hardening: Utilizing the `RuntimeDefault` profile ensures the container␣
↪uses the container runtime's default secure syscall filter.
```

8. Harden the container-level security context by enforcing a read-only filesystem, disabling privilege escalation, and dropping all default Linux capabilities:

```
containerSecurityContext:
  readOnlyRootFilesystem: true
  allowPrivilegeEscalation: false
  capabilities:
    drop:
      - ALL
```

Alternatively, you can apply these settings via the command line:

```
helm upgrade --install beacon-agent-k8s \
  . \
  --namespace upm-beacon \
  --create-namespace \
  # ... your other flags ...
  --set securityContext.runAsUser=null \
  --set securityContext.runAsGroup=null \
  --set podSecurityContext.fsGroup=null \
  --set opentelemetry-collector.securityContext.runAsUser=null \
  --set opentelemetry-collector.securityContext.runAsGroup=null \
  --set kube-state-metrics.podSecurityContext=null
```

Additional requirements for OpenShift

OpenShift assigns a specific UID range to each namespace. For example: 1000880000-1000889999.

Specifying an explicit runAsUser (like 10000) often falls outside the UID range, triggering SCC violations. Leave these values empty so OpenShift's admission controller automatically assigns appropriate values from the allowed range.

Recommended runAsUser configurations by component

This table outlines the required runAsUser settings for EDB Postgres AI agent components to ensure compatibility with different SCC.

Component	runAsUser	Reason
beacon-agent	Empty	Uses restricted SCC; OpenShift assigns UID from namespace range
kube-state-metrics	Empty	Uses restricted SCC; OpenShift assigns UID from namespace range
opentelemetry-collector	65532	Uses privileged SCC for *hostPath* or *hostPorts*.
opentelemetry-collector	65532	Requires explicit numeric UID because *runAsNonRoot: true* cannot verify non-numeric users like *nonroot*. Privileged SCC allows any UID, so *65532* works.

Granting SCC for Daemonsets

To collect node-level metrics and logs on OpenShift, the `opentelemetry-collector` and `prometheus-node-exporter` subcharts require elevated permissions to access `hostPath` , `hostPorts` , and `hostNetwork` . Grant the privileged SCC to their respective service accounts using the commands below.

> **Note**
>
> Since OpenShift includes a built-in node-exporter, your deployment may conflict with existing monitoring; consider using the platform's native exporter instead.

1. Create the namespace (if not already present):

```
oc new-project upm-beacon
```

2. Grant the privileged SCC:

```
oc adm policy add-scc-to-user privileged \
  system:serviceaccount:upm-beacon:beacon-agent-k8s-opentelemetry-collector

oc adm policy add-scc-to-user privileged \
  system:serviceaccount:upm-beacon:beacon-agent-k8s-prometheus-node-exporter
```

3. Deploy or update the Helm chart:

```
helm upgrade --install beacon-agent . -f custom-values.yaml -n upm-beacon
```

4. Restart the Daemonsets:

```
kubectl -n upm-beacon rollout restart daemonset beacon-agent-k8s-opentelemetry-
↪collector-agent
kubectl -n upm-beacon rollout restart daemonset beacon-agent-k8s-prometheus-node-
↪exporter
```

> **Note**
>
> On OpenShift, *prometheus-node-exporter* may conflict with OpenShift's built-in *node-exporter* . For example, if port 9100 is already in use. We recommend setting *additionalCharts.prometheus-node-exporter.install: false* and relying on the *opentelemetry-collector* for container metrics via cadvisor/kubelet scraping.

On a fully-loaded OpenShift cluster, the per-node kubelet and cAdvisor scrape can produce an OTLP batch larger than the HM metrics receiver's 4 MiB gRPC message limit. When that happens the receiver rejects the batch with `ResourceExhausted` and **every datapoint in it is dropped**, so HM shows no CPU, memory, or network metrics even though the collector pods are healthy.

Cap the collector's batch processor so each push stays under the limit:

```
opentelemetry-collector:
  config:
    processors:
      batch:
```

(continues on next page)

(continued from previous page)

```
        send_batch_size: 1000
        send_batch_max_size: 1500
```

Or via the command line:

```
- -set opentelemetry-collector.config.processors.batch.send_batch_size=1000 \
- -set opentelemetry-collector.config.processors.batch.send_batch_max_size=1500
```

Configuration reference

Core configuration options:

Parameter	Description	Default	Required
agentConfig.accessKey	HM machine user access key	*""*	Yes
agentConfig.projectID	HM project ID	*""*	Yes
agentConfig.beaconServer.host	HM server hostname	*""*	Yes
agentConfig.locationID	Agent location identifier	*"standalone-k8s"*	Yes
namespace	Kubernetes namespace	*"upm-beacon"*	No
image.repository	Beacon agent image	*""*	Yes
image.tag	Beacon agent version	*""*	Yes
otelImage.repository	OTEL collector image	*""*	Yes (if OTEL enabled)
otelImage.tag	OTEL collector version	*""*	Yes (if OTEL enabled)
additionalCharts.cert-manager.instal	Enable cert-manager subchart	*false*	No
additionalCharts.opentelemetry-coll	Enable opentelemetry-collector subchart	*false*	No
additionalCharts.kube-state-metrics.	Enable kube-state-metrics subchart	*false*	No
additionalCharts.prometheus-node-e	Enable prometheus-node-exporter subchart	*false*	No
opentelemetry-collector.ima	OTEL collector image repository	*""*	Yes (if OTEL enabled)
opentelemetry-collector.ima	OTEL collector image tag	*""*	Yes (if OTEL enabled)

TLS configuration options

These options control how the agent trusts the Hybrid Manager (HM) beacon server certificate. By default (`rootCAEnabled: true`), the agent mounts a Certificate Authority (CA) secret and verifies HM using that certificate.

Parameter	Description	Default	Required
rootCAEnabled	When *true*, mounts the CA secret and sets the agent's *root_ca_path*. Set to *false* to trust HM via the system CA pool (or skip verification) without mounting a CA secret.	*true*	No

continues on next page

Table 77 – continued from previous page

Parameter	Description	Default	Required
rootCASecretName	Name of the secret holding the HM root CA (*ca.crt*). Must exist when *rootCAEnabled* is *true*.	*beacon-server-ca*	No
rootCA	PEM-encoded root CA. When set, the chart creates the CA secret for you instead of requiring a pre-existing one.	*""*	No
agentConfig.beaconServer.rootCAPath	Path inside the pod where the CA certificate is mounted and read from. Only used when *rootCAEnabled* is *true*.	*/etc/ssl/certs/ca-certif*	No
agentConfig.beaconServer.tlsInsecureSkipVerify	When *true*, the agent skips TLS verification of the HM beacon server certificate. Typically paired with *rootCAEnabled: false* for testing.	*false*	No
agentConfig.beaconServer.plaintext	When *true*, the agent connects to HM without TLS.	*false*	No

Note

When *rootCAEnabled: false* , you don't need to create the *beacon-server-ca* secret described in the *Namespace and secrets* . The agent relies on the system CA pool (or skips verification when *tlsInsecureSkipVerify: true*), and the CA volume mount is not injected into the agent, OpenTelemetry (OTEL) collector, or CloudNativePG™ sidecar pods. !!!

Monitoring with the Hybrid Manager console

You can access a comprehensive set of database metrics using the Hybrid Manager (HM) console, with availability varying by metric and cluster type.

- *Support scope* : Learn about the type of metrics that are supported for your clusters.
- *Metrics latency* : Learn about how often metrics are updated.
- *Event markers overlapping when displaying two different events* : Learn about Event markers and how they provide additional information on the data monitored in your clusters.

Metrics support

Hybrid Manager (HM) collects and displays a wide range of metrics to give you comprehensive visibility into your database clusters.

The availability of metrics you can see in the HM console depends on the metric and cluster type.

Summarized overview

This overview shows the available metrics per database cluster type.

	Host metrics	*Postgres metrics*	*Cluster metrics*	*Query metrics*	*GPU Recommendations for Default NIM Models*	*Logs and observability*
HM-managed cluster	Yes	Yes	Yes	Yes	Yes	Yes
External: **Self-managed cluster**	Yes	Yes	Yes	Yes	Yes	Yes
External: **CNPG database cluster**	Yes	Yes	Yes	Yes	Yes	Yes
External: **AWS RDS database**	Yes	partial support	No	No	No	No

- Currently, the Hybrid Manager supports monitoring for EDB CloudNativePG and for open-source CloudNativePG clusters.
- Monitoring isn't supported for external self-managed EDB Postgres Distributed clusters. .. :: ## Detailed overview

Host metrics

Host metrics display resource utilization for the underlying server infrastructure hosting your database instances. You can see them in the **Estate** view and in the cluster view's ***Monitoring*** tab in the **Host** section.

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Memory	Yes	Yes	Yes	Yes
CPU	Yes	Yes	Yes	Yes
Storage	Yes	Yes	Yes	Yes
Disk IOPS	Yes	Yes	Yes	Yes
Disk throughput	Yes	Yes	Yes	Yes
Network activity	Yes	Yes	Yes	Yes

Important

Memory, CPU, and storage metrics show the average percentage usage for Postgres processes on the hosting node. To monitor system-wide usage on HM-managed clusters, use *Grafana dashboard setup for Agent Factory metrics* . System-wide monitoring isn't currently supported for external clusters.

Postgres metrics

Postgres metrics show performance and operational statistics specific to the Postgres database server. You can see them in the **Estate** view and in the cluster view's ***Monitoring*** tab.

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Connections				
Connections	Yes	Yes	Yes	Yes
AAS by Wait Type	Yes	Yes	Yes	No
Nr. of blocked backends	Yes	Yes	Yes	No
Transactions				
Tuples in	Yes	Yes	Yes	Yes
Tuples out	Yes	Yes	Yes	Yes
Transactions rate	Yes	Yes	Yes	Yes
Buffer cache hit ration	Yes	Yes	Yes	Yes
Longest running transaction	Yes	Yes	Yes	No
Queries				
Query latency	Yes	Yes	Yes	No
Query rate	Yes	Yes	Yes	No
Storage				
Database size	Yes	Yes	Yes	No
Disk usage	Yes	Yes	Yes	No
WAL size	Yes	Yes	Yes	Yes
WAL usage	Yes	Yes	Yes	No
Live/Dead tuples	Yes	Yes	Yes	No
Index/table/temp estimated size	Yes	Yes	Yes	Yes
Internals				
Time since last autovacuum	Yes	Yes	Yes	Yes
Autovacuum Stats	Yes	Yes	Yes	Yes
Time since last checkpoint	Yes	Yes	Yes	Yes
Checkpoint stats	Yes	Yes	Yes	Yes
Time since last successful WAL archive	Yes	Yes	Yes	No
WAL archiving stats	Yes	Yes	Yes	Yes

Cluster metrics

Cluster metrics are host and postgres metrics, individual and aggregated for database deployments spanning multiple nodes (HA, AHA, and DHA topologies). You can see them in the **Estate** view and in the cluster view's ***Monitoring*** tab in the **Topology** and **Replication** sections. Cluster metrics aren't displayed for single node deployments.

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
RAFT status (AHA, and DHA)	Yes	No	No	No
Replication slot status	Yes	Yes	Yes	No
Clock skew	Yes	Yes	Yes	No
Node status	Yes	Yes	Yes	No
Transaction rate	Yes	Yes	Yes	No
Connections (total)	Yes	Yes	Yes	No
Connections (per node)	Yes	Yes	Yes	No
Memory (per node)	Yes	Yes	Yes	No

continues on next page

Table 81 – continued from previous page

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Storage (per node)	Yes	Yes	Yes	No
WAL size (per node)	Yes	Yes	Yes	No
Replication status	Yes	Yes	Yes	No

Query metrics

Query metrics offer insights into the performance and characteristics of the database Postgres queries being executed. You can see them in the cluster view's ***Query diagnostics*** tab.

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Query latency	Yes	Yes	Yes	No
Nr. of queries executed	Yes	Yes	Yes	No
Query latency histogram	Yes	Yes	Yes	No
Nr. of active sessions	Yes	Yes	Yes	No

Recommendations

Recommendations suggest actions and optimizations based on observed metrics to improve your database environment (for example, improve indexes). You can see them in the cluster view's **Recommendations** tab.

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Indexes	Yes	Yes	Yes	No
Statistics	Yes	Yes	Yes	No
Configuration	Yes	Yes	Yes	No
Security	Yes	Yes	Yes	No
Workload	No	No	No	No

Logs

Logs present raw and processed log data from various components of your database system for troubleshooting and auditing. You can see them in the cluster view's ***Logs*** tab.

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Logs	Yes	Yes	Yes	No

Metrics latency

Hybrid Manager (HM) collects and displays a wide range of observability data to give you comprehensive visibility into your database clusters.

The refresh intervals for these metrics vary depending on the metric and database type. This difference is due to the varying methods of data collection and integration.

Understanding data latency

When you view your database metrics in the HM console, the data you see is the result of a two-step process, each with its own refresh interval. Understanding both steps will help you understand how up to date your metrics are.

Backend refresh: This is the first step, where HM's internal services actively collect updated metrics and data from your database clusters. Whether your cluster is managed by HM or an external database, the backend systems are constantly gathering the latest information. This data is then stored in HM's central data stores.

HM console refresh: This is the second step, where the HM console makes calls to your central data stores to retrieve the most recently collected data and display it to you.

The actual latency of the data you see in the HM console is a combination of these two refresh intervals. The data on the screen reflects information that isn't older than the sum of these two intervals. In many cases, it's much fresher, sometimes just seconds old.

Refresh intervals

Backend refresh: HM's internal services update the data stores as follows. The interval varies depending on the database cluster type and the metrics context.

	HM-managed	External: self-managed	External: CSP-managed	External: EDB CNPG
Estate page ► Postgres table	20s	5m	5m	5m
Cluster view ► Monitoring tab	15s	15s	30s	15s
Cluster view ► Query Diagnostics tab	1m	1m	na	1m
Cluster view ► Recommendations tab	30m	30m	na	30m
Cluster view ► Logs tab	1s	1s	na	1s

These intervals are configurable for self-managed and CSP-managed clusters. To customize the frequency of data collection for self-managed clusters, define the intervals in the *EDB Postgres AI agent configuration YAML* . In the case of CSP-managed clusters, define the intervals for the supported metrics through the *beacon-csp-credentials* .

HM console refresh: The HM console updates the data for all cluster types with the following frequency.

	Console refresh interval
Estate page ► Postgres table	1m
Cluster view ► Monitoring tab	30s
Cluster view ► Query Diagnostics tab	Manual browser refresh
Cluster view ► Recommendations tab	Manual browser refresh

continues on next page

Table 86 – continued from previous page

	Console refresh interval
Cluster view ▶ Logs tab	10s

Note

Data is updated only when you explicitly reload the browser page.

Event markers

Event markers show activity log events as visual indicators on time-series charts in Hybrid Manager (HM). Use them to correlate metric changes with specific cluster events, or as boundaries when selecting a time period to analyze.

Marker types

Event markers are displayed as a vertical line at the event's timestamp. Use them to mark a single moment, such as a configuration change or an alert.

Viewing event markers

Event markers appear on monitoring charts in the **Monitoring** tab. You can view them at the cluster or node level.

To view event markers:

1. In the HM console, select your project.
2. Select **Clusters > Postgres Clusters**.
3. Select the **Monitoring** tab.
4. From the **All Events** filter, select the event types you want to display.

Hovering over a marker displays its metadata — name, type, and description.

Toggling markers

Use the **All Events** filter to show or hide markers by section, chart, or marker type. Marker visibility can be controlled independently per type.

Note

Alerts appear as event markers but aren't logged in the activity log. Alert markers reflect only the alert's start time. When the underlying issue is resolved, the marker disappears.

Using markers as time delimiters

Instead of the date and time pickers, you can use event markers to define the time range displayed on monitoring charts. Use a marker as the start or end of the time range, with an optional time buffer to extend the range on either side. You can also pair a marker on one end with a datetime on the other.

To use an event marker as a time delimiter:

1. In the monitoring toolbar, select the **Start** or **End** picker.

2. Select the **Event** tab.
3. Search for or select an event from the list.
4. (Optional) Repeat for the other picker, or enter a datetime manually using the **Date** tab.
5. (Optional) Select **Time buffer** and select a duration to extend the range on either side.
6. Select **Apply**.

Viewing system-generated annotations

HM automatically generates annotations for certain events so they appear on your monitoring charts without manual action. These events include cluster configuration updates — whether triggered by a user, autoscaling, or an agentic automation.

Creating custom event markers

You can create a custom event marker at any point in time by sending a `POST` request to `/api/v1/projects/<project_id>/custom-event` :

```
{
  "message": "The text to attach to your event",
  "clusterId": "<cluster_id>"
}
```

Note

The *clusterId* field is optional. If included, the event appears in the activity log and as an annotation on that cluster's charts. If omitted, the event appears only in the activity log.

Grafana

Grafana allows you to visualize and analyze metrics about Hybrid Manager (HM) deployments, HM-managed database clusters and any other external clusters connected to HM. You can monitor performance, troubleshoot issues, and gain insights into resource utilization.

For HM-managed clusters, Grafana gathers metrics out of the box. For external clusters, you must first install EDB Postgres AI agent, as explained in Monitoring external databases .

Grafana support

Grafana dashboards are supported as follows:

- HM-managed clusters : Full support out of the box.
- Self-managed clusters : Partial support after the cluster is connected via EDB Postgres AI agent.
- CNP clusters : Partial support after the cluster is connected via EDB Postgres AI agent.
- RDS clusters : Partial support after the cluster is connected via Kubernetes secret.

Accessing Grafana

You can access the built-in Grafana instance via the *Launchpad* .

Filtering results

Every cluster managed by HM has an ID, which can be traced back to a namespace and its pods. You can filter and analyze metrics on Grafana by cluster ID (equivalent to a namespace) or by pod name to perform troubleshooting and resource planning operations.

See *Tracing clusters back to their Kubernetes resources* to understand how Kubernetes infrastructure relates to your cluster. More specifically:

- Understand how to obtain the *ID and namespace for a cluster* .
- Understand how to recognize the *Pods check is failing* in a cluster/namespace.

Related topics

- *Using Grafana dashboards*
- Grafana's Explore option

Using Grafana dashboards

The Grafana service shipped with Hybrid Manager (HM) comes with default dashboards designed to monitor different HM resources. Alternatively, create your own custom dashboard to display specific data that serves your organization's requirements.

Default Grafana dashboards

The Grafana service shipped with HM comes with default dashboards that are grouped into the following categories:

- *Kubernetes dashboards*
- *Loki dashboards for logging monitoring*
- *Postgres dashboards*
- *Telemetry dashboards*
- *Other dashboards*

Note

The dashboard descriptions offer a generalized and exemplified overview of the default dashboards included with HM's Grafana. The availability of specific metrics depends on your environment.

Custom dashboards

See Grafana's documentation to create your own dashboard .

Warning

Ensure you have a backup of any customized Grafana dashboards and have them stored on a version control repository. Any customized Grafana dashboards are deleted whenever the Grafana instance is updated or redeployed, for example, due to an HM upgrade, node Kubernetes version upgrade, or general node failure.

Create, use, and manage dashboards

See the Grafana documentation for information on how to create, use, and manage dashboards. Here are a few useful links:

•

Create a dashboard .

•

Import dashboards .

•

Modify dashboard settings .

- Manage dashboards .

•

Share dashboards .

- Use Explore to query, collect and analyze data.
- Use Alerting to set up your system to watch and flag specific events.

Kubernetes dashboards

These dashboards provide insights into the health and performance of the underlying Kubernetes infrastructure that powers Hybrid Manager (HM).

These are prebuilt dashboards called Kubernetes mixins from Grafana's official library.

Kubernetes / API server

Note

This dashboard displays data only for on-premises HM deployments, as Grafana requires node-level access to the API server.

Description: Monitors the performance and availability of the Kubernetes API server, a critical component for all interactions with the cluster.

Metrics These metrics include read and write availability, work queue latency, read and write errors, resource utilization, among others.

Usage: Use this to identify API latency issues or availability problems.

Kubernetes / Compute resources *

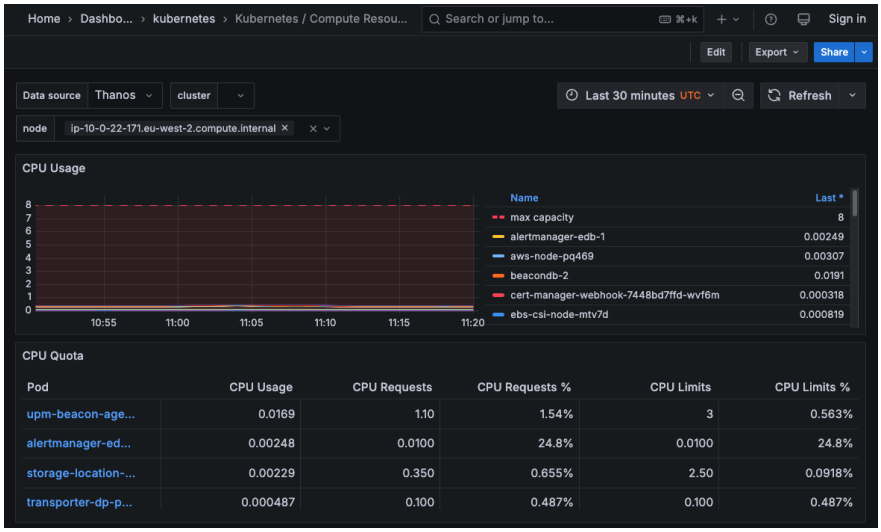


Fig. 13: Compute Resources by Node and Pods

Compute resources are displayed in multiple dashboards that provide different views: **Cluster** , **Namespace (Pods)**, **Namespace (Workloads)**, **Node (Pods)**, **Pod**, or **Workload**.

- **Cluster** displays resource utilization for the entire Kubernetes cluster, as well as utilization per namespace.
- **Namespace (Pods)** displays resource utilization that you can filter by Kubernetes namespaces, focusing on pod/service consumption.
- **Namespace (Workloads)** displays resource utilization that you can filter by Kubernetes namespaces and by workload type (deployments, daemonsets, StatefulSets, and so on), focusing on individual workload consumption.
- **Node (Pods)** shows resource utilization that you can filter by individual Kubernetes nodes, broken down by the pods running on them.
- **Pod** provides detailed resource usage for individual Kubernetes pods in a specific namespace.
- **Workload** shows detailed resource usage for a specific Kubernetes workload (for example, a deployment workload) in a specific namespace.

Description: Shows the resource utilization at the Kubernetes cluster level. Each of the subdashboards presents the data differently, focusing on different Kubernetes levels and offering different filtering options.

Metrics: Metrics include CPU, memory, bandwidth, packet rates, IOPS, throughput, and more.

Usage: Use these dashboards to:

- Understand the resource consumption across all workloads
- Identify resource-intensive applications or teams
- Understand the resource footprint of specific workloads in an application and of specific pods in a node
- Troubleshoot performance issues with specific application instances
- Monitor the performance of a particular application component

Kubernetes / Kubelet

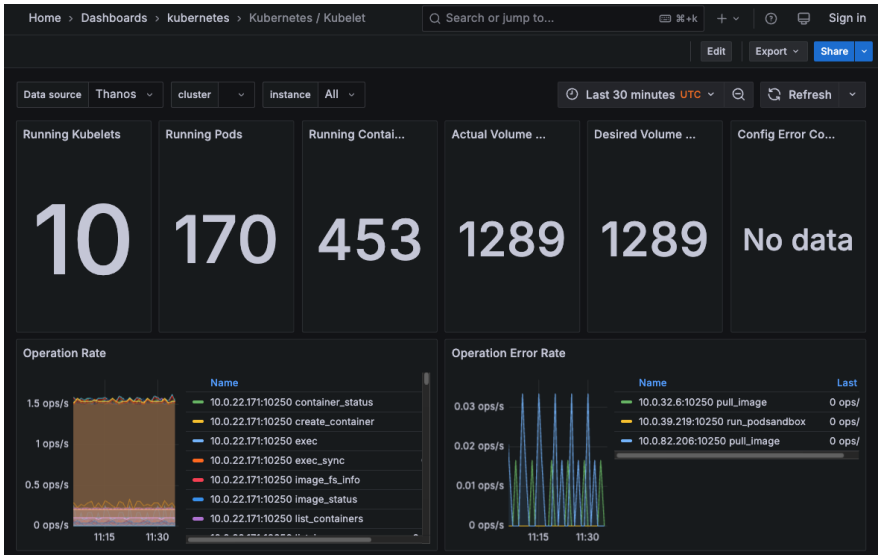


Fig. 14: Kubelet dashboard

Description: Monitors the health and performance of the Kubelet service running on each Kubernetes node, responsible for managing containers. You can see the aggregated performance of all kubelets, filter by kubelet instance and drill down into specific kubelet operations.

Metrics: Metrics include kubelet count, pods count, containers count, operation rates, operations durations, pod lifecycle operation rates, operation error rates, and more.

Usage: Use this to identify node-level agent activity, operation behavior, and latency and troubleshoot performance issues on a particular node.

Kubernetes / Networking *

Networking resources are displayed in multiple dashboards that provide different views: **Cluster** , **Namespace (Pods)**, **Namespace (Workloads)**, **Pod**, or **Workload**.

- **Cluster** displays network utilization for the entire Kubernetes cluster, as well as per namespace.
- **Namespace (Pods)** shows network traffic metrics that you can filter by Kubernetes namespaces, focusing on the pods in that namespace.
- **Namespace (Workloads)** shows network traffic metrics that you can filter by Kubernetes namespaces and by workload type (deployments, daemonsets, StatefulSets, and so on), focusing on individual workload traffic.
- **Pod** provides detailed network traffic metrics for individual Kubernetes pods in a specific namespace.
- **Workload** shows detailed network traffic metrics for a specific Kubernetes workload (for example, a deployment workload).

Description: Shows the network usage at the Kubernetes cluster level. Each of the subdashboards presents the data differently, focusing on different Kubernetes levels and offering different filtering options.

Metrics: Metrics include rates of bytes received and transmitted, rates of packets received and transmitted, receive and transmit bandwidth, and more.

Usage: Use these dashboards to:

- Understand overall network communication patterns, traffic, throughput, and potential bottlenecks

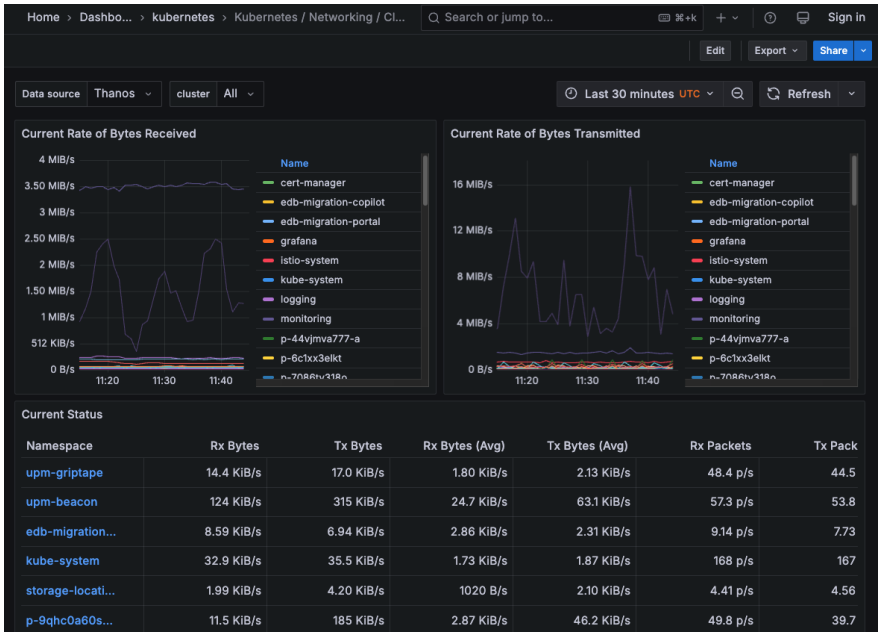


Fig. 15: Networking overall cluster dashboard

- Analyze network patterns for specific applications
- Understand the resource footprint of specific workloads in an application
- Troubleshoot network connectivity issues with specific pods in an application
- Monitor the network performance of a particular application component

Kubernetes / Persistent Volumes

Description: Monitors the usage and performance of persistent volumes (PVs) in the Kubernetes cluster. You can filter the results per namespace and persistent volume claim (PVCs).

Metrics: Metrics include free and used space.

Usage: Use this to identify storage bottlenecks and capacity issues, plan for storage expansions, and monitor storage costs.

Kubernetes / StatefulSets

Description: Provides specific monitoring for Kubernetes StatefulSets , which manage stateful applications. You can filter results by namespace (=application) and StatefulSet name.

Metrics: Metrics include CPU, memory, and network consumption, as well as replicas.

Usage: Use this to monitor the stability, performance, and resource consumption of your stateful applications and track whether your stateful applications have met their desired replicas.

Kubernetes CoreDNS

Description: Monitors the performance and health of the CoreDNS service, which provides DNS resolution in the Kubernetes cluster, enabling communication within the pods and services.

Metrics: The metrics include DNS request rates by protocol, query type, DNS zone, size, response code, response duration, response size, and others.

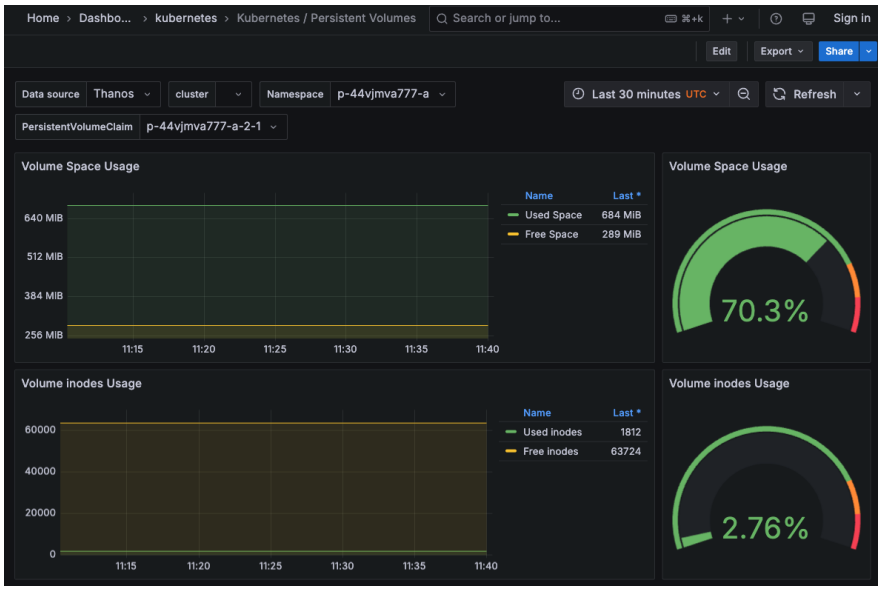


Fig. 16: Persistent Volumes dashboard

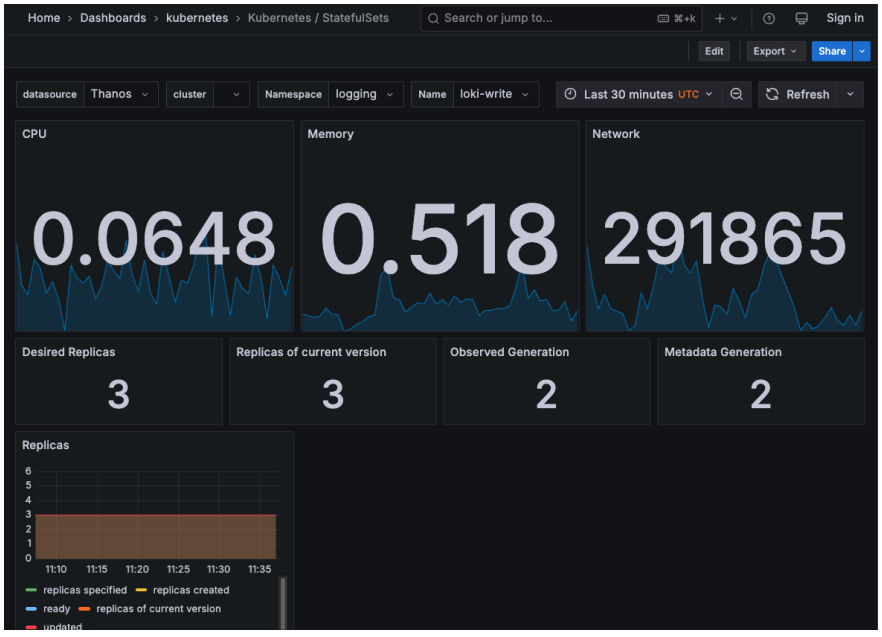


Fig. 17: StatefulSets dashboard

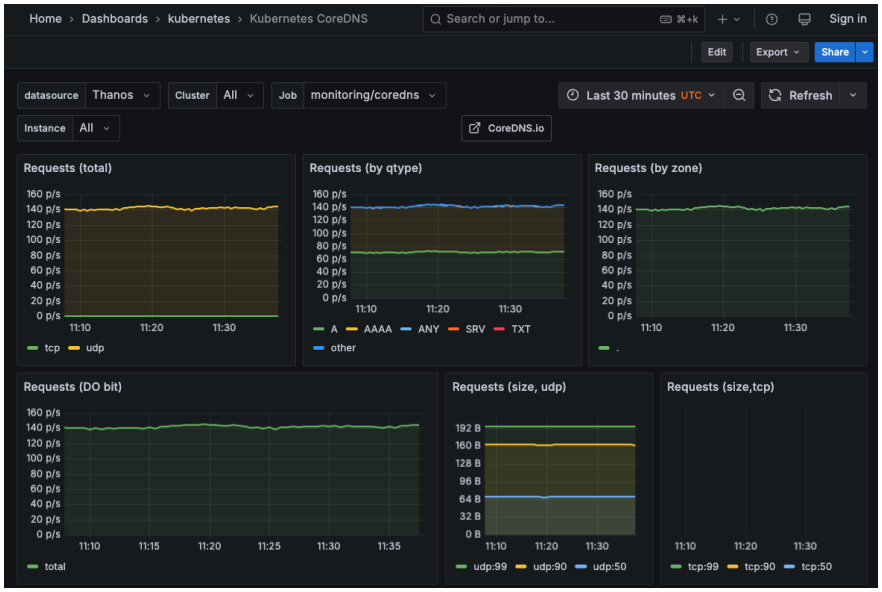


Fig. 18: CoreDNS dashboard

Usage Use this to monitor the reliability and efficiency of the CoreDNS component and identify any discovery and communication issues among services.

Loki dashboards for logging monitoring

These dashboards are related to the integrated Loki logging system, allowing you to visualize and analyze your logs. Visualization is limited to the `logging` namespace only, as it's where Loki performs its operations.

Loki dashboards are prebuilt dashboards called Loki mixins from Grafana's official library.

Loki / Chunks

Description: Monitors the storage and management of log data chunks (a grouping of log entries that belong to a single log stream) in Loki.

Metrics: Metrics include active chunks, percentile utilization, log entries per chunk, chunk lifecycle events (flush operations), chunk size, chunk age, bytes per chunk, and more.

Usage: Helps identify potential storage inefficiencies, understand chunk lifecycle patterns, and troubleshoot issues related to chunk management and query performance.

Loki / Deletion

Description: Tracks the performance and status of Loki's log deletion processes based on retention policies. The default Loki retention period is 30 days.

Metrics: Deletion duration, deletion errors, number of chunks deleted, number of series affected by deletion, and deletion queue length.

Usage: Allows you to verify that your retention policies are being enforced correctly, monitor the efficiency of the deletion process, and identify any errors preventing log deletion. Use this to ensure logs are being pruned as expected.

Fig. 19: Loki Chunks dashboard

Loki / Logs

Description: You can view and filter logs ingested by Loki.

Metrics: Logs, rates of logs ingested, and more.

Usage: Provides a visual overview of log data, enabling quick identification of trends and errors. Use this for real-time log analysis and troubleshooting.

Loki / Operational

Description: Provides insights into the operational health and performance of the Loki system.

Metrics: Queries and pushes per second, query and push latency, log lines per tenant, MBs usage per tenant, CPU utilization of Loki components (ingesters, distributors, queriers), memory usage of Loki components, streams and chunks over time, and more.

Usage: Helps ensure the stability and performance of the Loki logging system by monitoring its resource consumption and internal health. Identify potential bottlenecks or resource exhaustion in Loki.

Loki / Reads

Description: Monitors the performance of log read queries against Loki.

Metrics: Loki read queries per second, latency per read operation and per pod.

Usage: Allows you to assess Loki's responsiveness to log queries.

Loki / Reads Resources

Description: Shows resource consumption (CPU, memory) by Loki read operations.

Metrics: Memory in use by individual Loki read pods and by Loki-ingested pods.

Usage: Helps understand the resource overhead of log read querying.

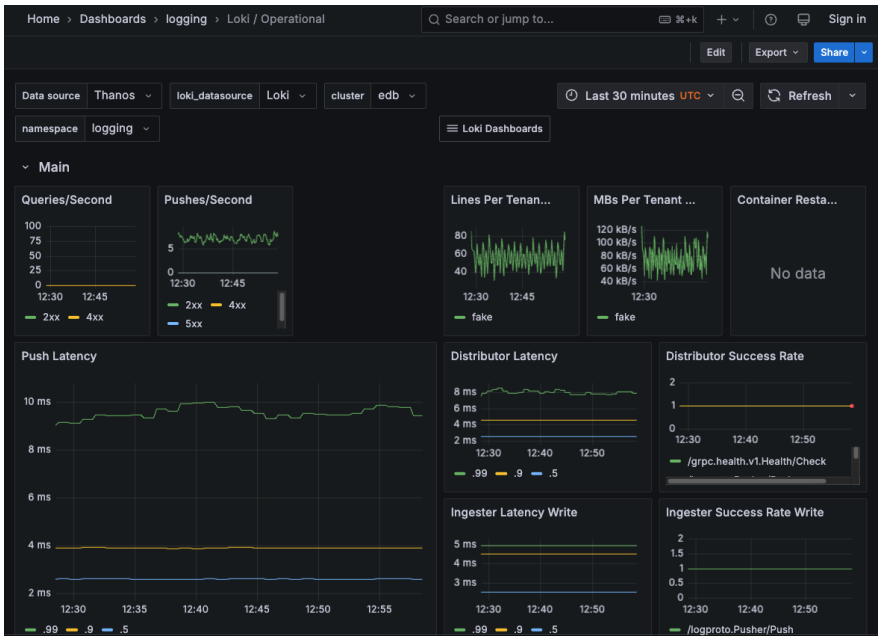


Fig. 20: Loki Operational dashboard

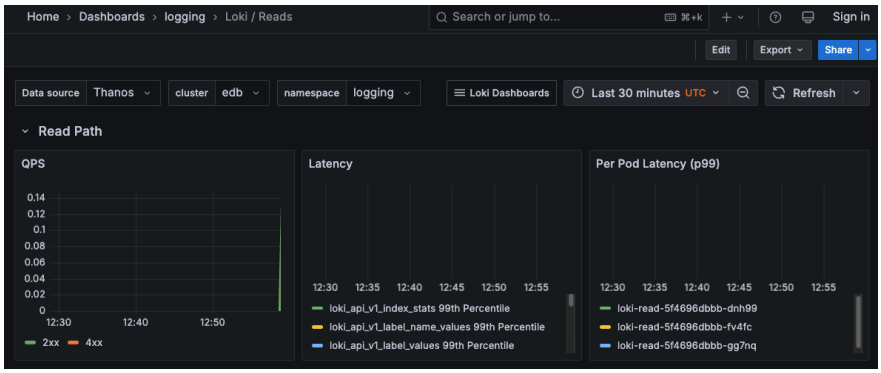


Fig. 21: Loki Reads dashboard

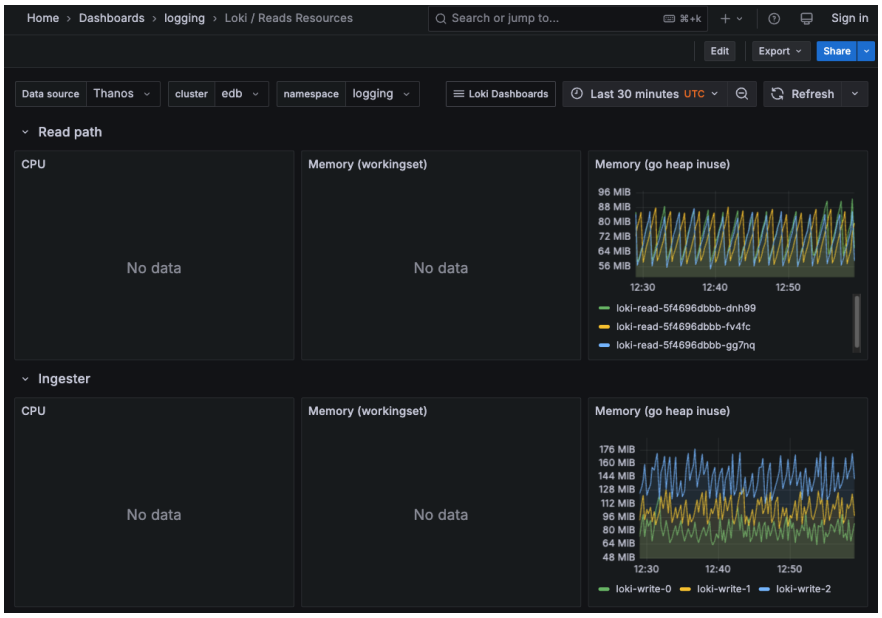


Fig. 22: Loki Reads Resources dashboard

Loki / Retention

Description: Provides detailed metrics on Loki's log retention policies and their enforcement.

Metrics: Last successful compaction operation, compaction operations by duration and status, memory in use by Loki backend pods (for retention).

Usage: Allows you to verify that your configured retention policies are active and understand their impact on storage consumption in Loki.

Loki / Writes

Description: Monitors the performance of log ingestion (writes) queries into Loki.

Metrics: Loki write queries per second, latency per write operation and per pod.

Usage: Helps you identify slow ingestion rates or dropped logs in the log forwarding pipeline from your applications to Loki.

Loki / Writes Resources

Description: Shows resource consumption (CPU, memory) by Loki write operations.

Metrics: Memory in use by individual Loki write pods and by Loki ingesting pods.

Usage: Helps you understand the resource overhead of log write querying.

Loki Data

Description: This is a dashboard designed by EDB that provides a high-level view of the amount of log data Loki is managing.

Metrics: Loki ingest rate and total stored chunks in bytes.

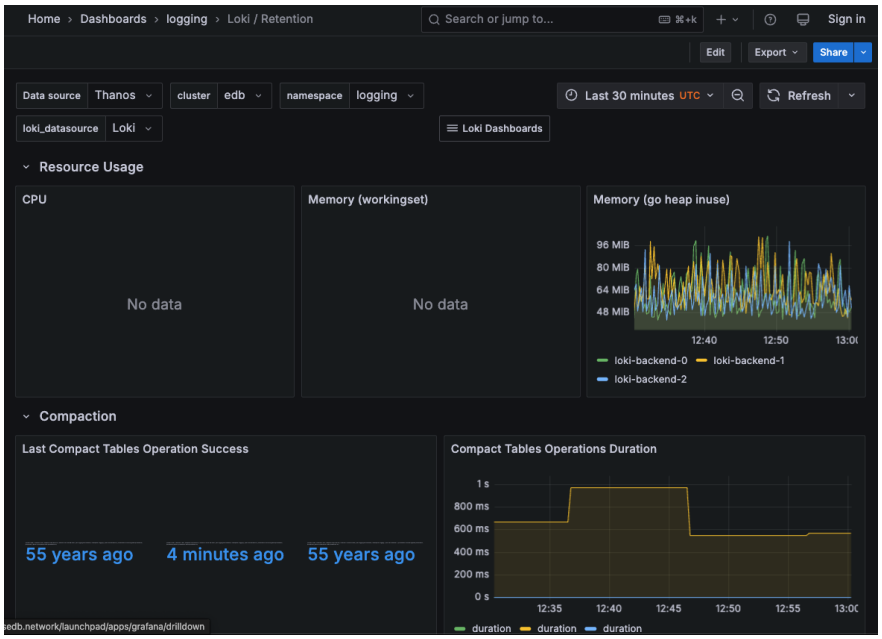


Fig. 23: Loki Retention dashboard

Fig. 24: Loki Writes dashboard

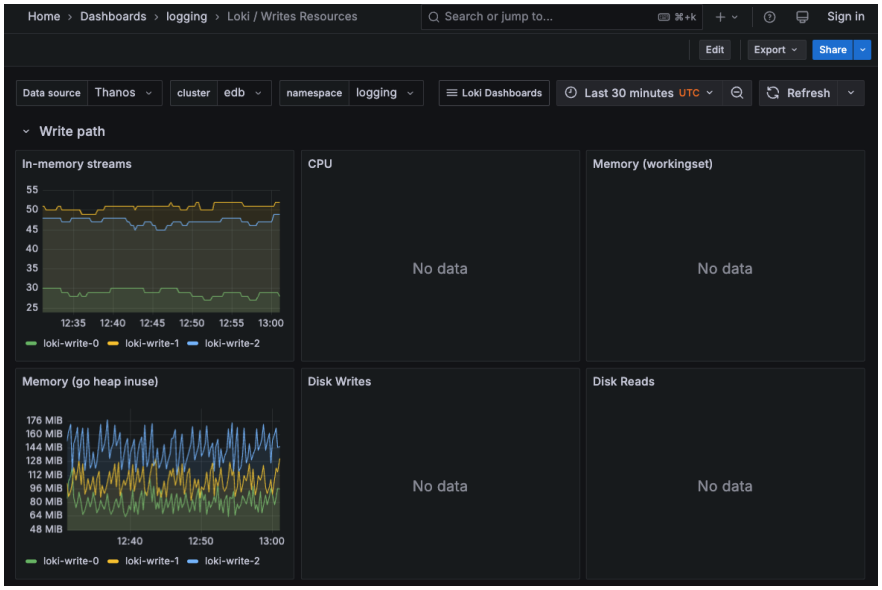


Fig. 25: Loki Writes Resources dashboard

Fig. 26: Loki Data dashboard

Usage: You can use this dashboard for capacity planning, understanding data growth, and managing logging-related storage costs.

Postgres dashboards

These dashboards were designed by EDB to provide insights into the performance and health of Postgres databases you manage or monitor with Hybrid Manager (HM).

Hardware Utilisation

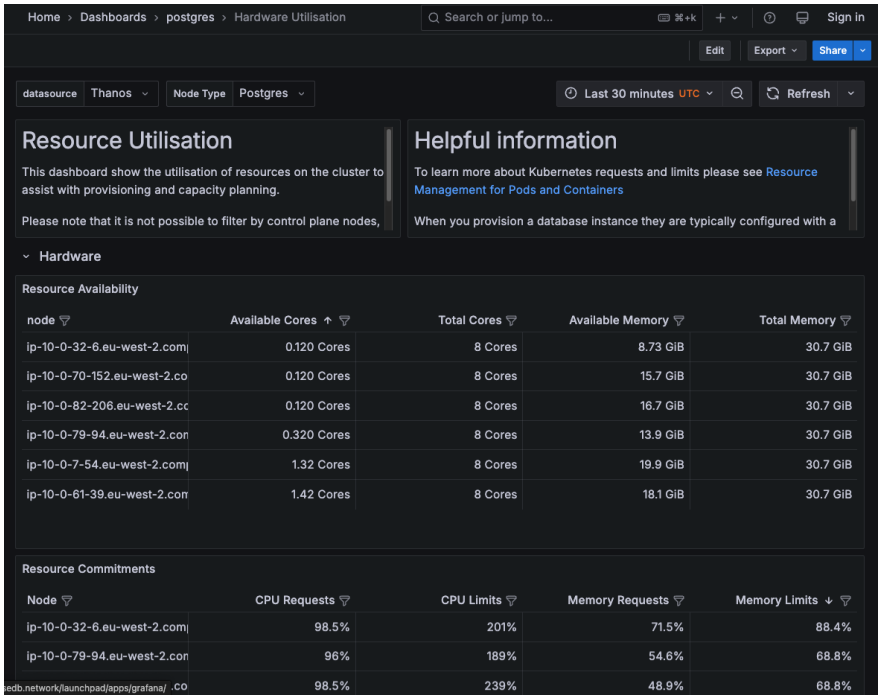


Fig. 27: Postgres Hardware Utilisation dashboard

Description: Shows the underlying hardware resource utilization (cores, memory, CPU) of the nodes hosting your Postgres instances.

Metrics: Available and total cores per node, available and total memory per node, CPU requests and limits per node, Postgres pods per node count, pod status, and more.

Usage: Use this to identify capacity bottlenecks and see if you have oversubscribed and if nodes have been overscheduled.

Kubernetes Events

Description: Displays Kubernetes events related to your Postgres pods and deployments. It displays events of the last hour. Therefore, it's possible that no events are shown for the specified namespace, for example, if workloads are running smoothly and no Kubernetes events are required to restore a pod.

Metrics: Type of event (for example, pod creation, deletion, error) in YAML format.

Usage: Provides context into Kubernetes operations that might be impacting your Postgres instances. Useful for diagnosing failed deployments, pod restarts, or other Kubernetes-level issues affecting database availability.

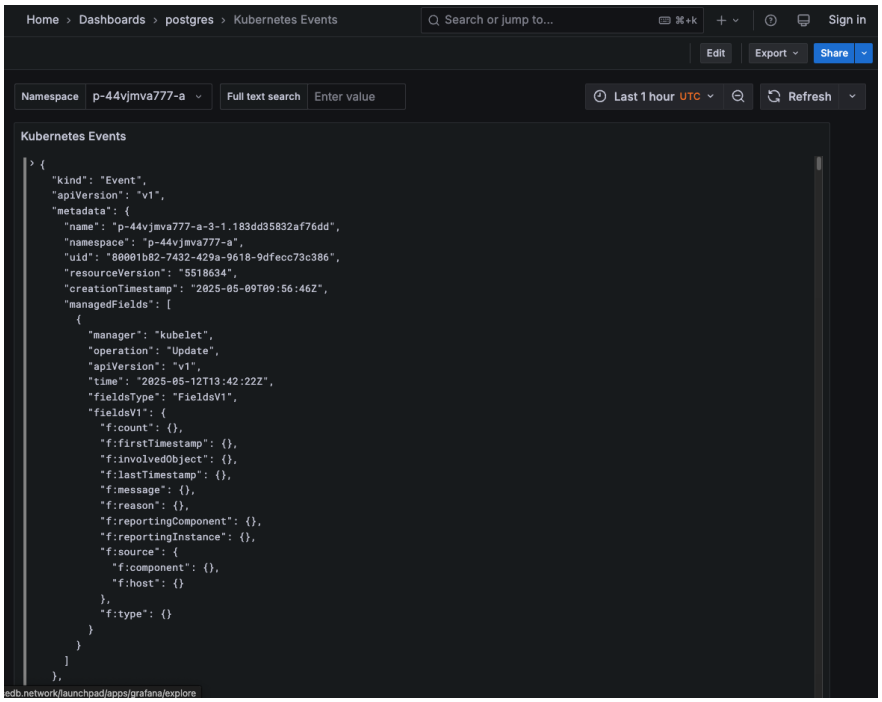


Fig. 28: Postgres Kubernetes Events dashboard

PGD Overview

For distributed high availability (DHA or PGD) clusters only.

Description: Provides an overview of how your distributed high availability PGD cluster is performing, focusing on its replication capabilities.

Metrics: PGD and BDR versioning, data node status, version, Raft state and metrics, BDR worker state and metrics, replication status, connections, throughput, and more.

Usage: Gives a central view of the health and synchronization status of your cluster. Identify nodes that are lagging or unhealthy and monitor the overall performance of the distributed setup.

Postgres Estate Statistics

Description: Offers a fleet-wide view of the clusters you're running.

Metrics: List and count of all running clusters, their age, and latest restore stamp.

Usage: Provides a high-level overview of the clusters under your estate.

Postgres logs

Description: You can view Postgres server logs.

Metrics: Logs in text form.

Usage: Provides a visual overview of Postgres log data, enabling quick identification of trends and errors. Use this for real-time log analysis and troubleshooting.

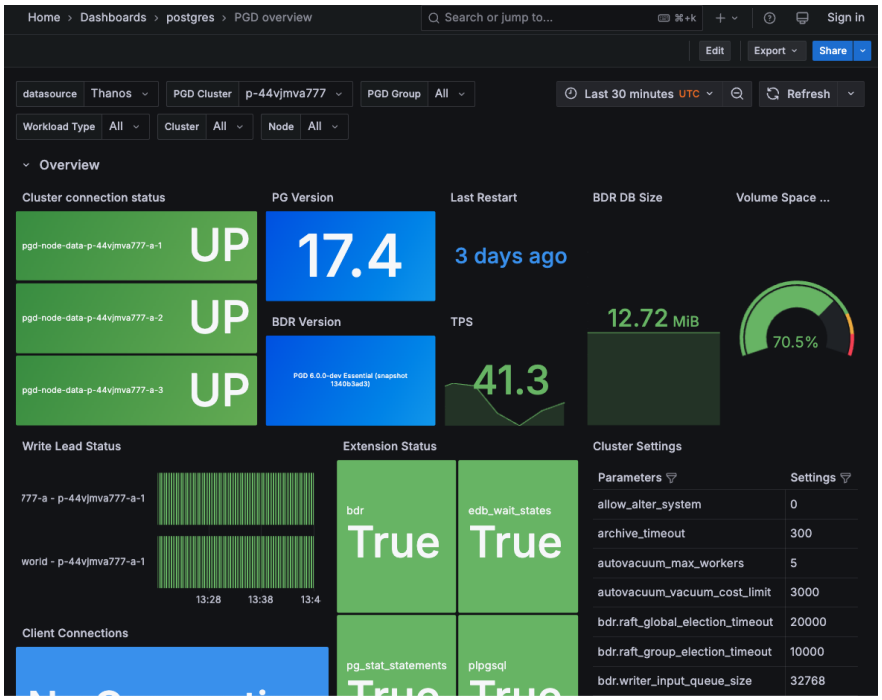


Fig. 29: Postgres PGD Overview dashboard

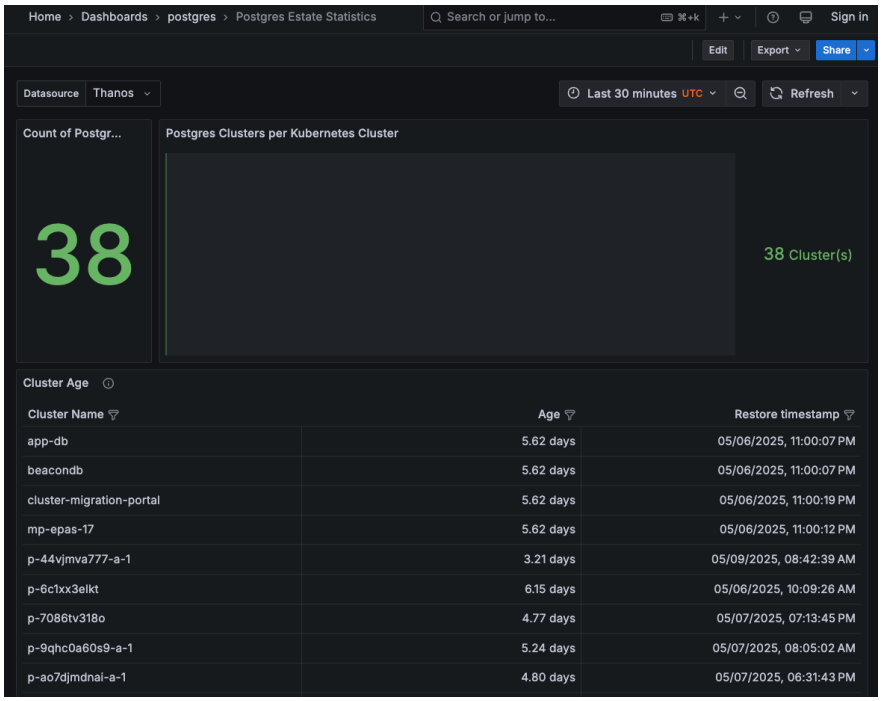


Fig. 30: Postgres Estate Statistics dashboard

Postgres Status

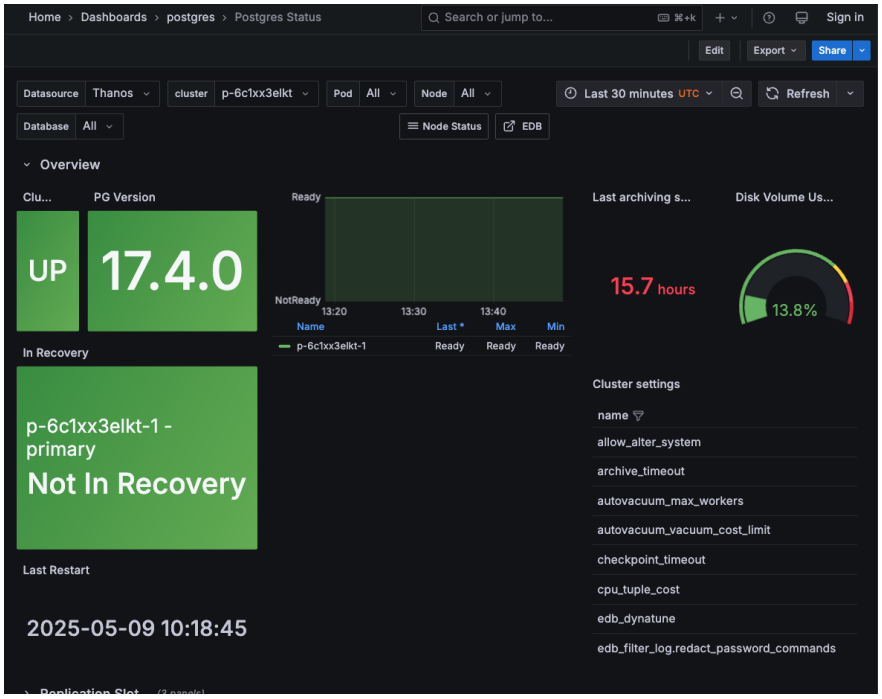


Fig. 31: Postgres Status dashboard

For all cluster types (single-node and multiple nodes).

Description: Provides an overview of how your individual Postgres instances are performing.

Metrics: Cluster status, Postgres version, pod status, replication status and lag, connections, throughput, resource utilization, WAL filing status, and more.

Usage: Provides a central view of the health and activity of individual Postgres databases. Identify instances with high load, connection issues, replication lag, or other performance issues that may require investigation.

Telemetry dashboards

These dashboards provide insights into the Hybrid Manager (HM) platform and its underlying components.

FluentBit

This is a prebuilt Fluent Bit mixin from Grafana's official library.

Description: Provides metrics about the FluentBit instance, which is used to collect logs from the nodes, and ships them to Loki, which in turn informs Hybrid Manager.

Metrics: Input and output processing rates, input and output records rate, failed and error rates, and more.

Usage: Helps identify if FluentBit is experiencing performance bottlenecks, resource exhaustion, or errors in collecting and sending logs from HM components.

Grafana Overview

Description: Provides metrics about the Grafana instance, the platform that collects metrics from different sources and displays them in a single place.

Metrics: Firing alerts, requests per second, latency, and more.

Usage: Allows you to monitor the health and performance of the Grafana instance that you're using to view these dashboards. Identify if Grafana is overloaded or experiencing issues.

Memcached

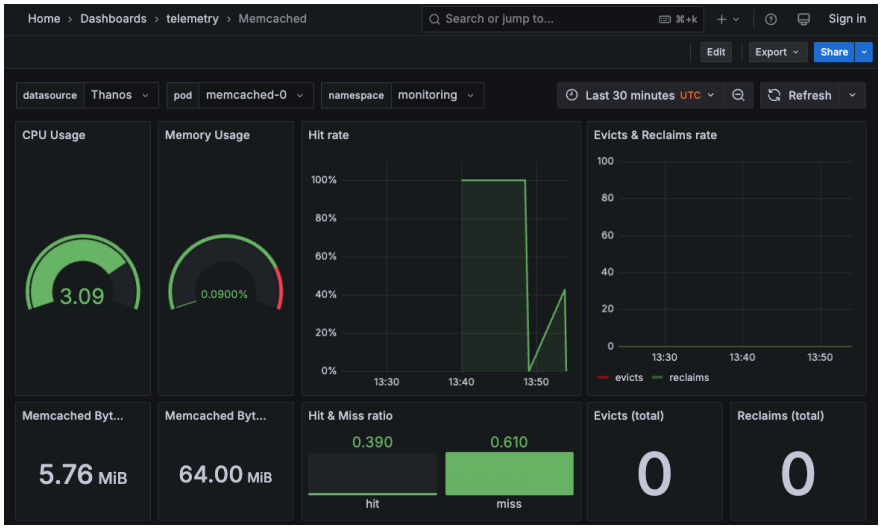


Fig. 32: Telemetry Memcached dashboard

Description: Monitors the health and performance of Memcached monitoring services in the HM infrastructure, providing insights into resource utilization, cache efficiency, and eviction rates.

Metrics: CPU and memory usage, hit and miss ratios, evicts and reclaims rate, memcached bytes, and more.

Usage: Use this dashboard to understand the resource consumption of your Memcached instances, evaluate the effectiveness of the cache (hit rate), and monitor how frequently items are being evicted or reclaimed due to memory pressure.

Node Exporter *

These are prebuilt node mixins from Grafana's official library.

Node Exporter metrics are displayed in multiple dashboards that provide different views: **Nodes**, **USE Method / Cluster**, and **USE Method / Node**.

- **Nodes** provides the aggregated resource utilization for a specific node, broken down into types of CPU usage, loads, type of memory, and more.
- **USE Method / Cluster** displays the aggregated resource utilization for the entire Kubernetes cluster, with per-node colorization.
- **USE Method / Node** displays the total resource utilization per node and some metrics for non-volatile memory devices.

Description: Shows hardware metrics for nodes running any of the HM components.

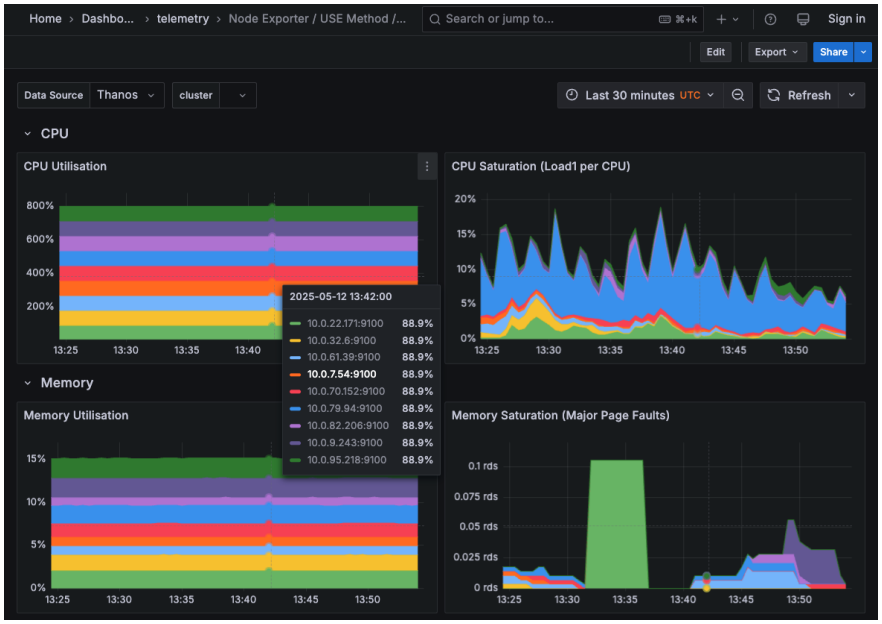


Fig. 33: Telemetry Node Exporter dashboards

Metrics: CPU utilization, load average, memory usage, disk usage, and more.

Usage: Provides several views of the hardware resources of the servers hosting HM. Useful for identifying resource bottlenecks at the node level.

Prometheus Overview

This is a prebuilt Prometheus mixin from Grafana's official library.

Description: Monitors the health and performance of the Prometheus monitoring system, which is used to collect and ship metrics in HM.

Metrics: Prometheus stats, discovery and targets, scrape intervals and failures, query rates, and more.

Usage: Allows you to monitor the health of the Prometheus metric collection engine. Issues with Prometheus can lead to incomplete or delayed monitoring data.

Thanos *

These are prebuilt Thanos mixins from Grafana's official library.

Thanos metrics are displayed in multiple dashboards that provide different views: **Compact**, **Overview**, **Query**, **Receive**, and **Store**.

- **Compact** monitors the Thanos compactor, a component used for long-term storage and downsampling of Prometheus metrics.
- **Overview** provides a high-level overview of the Thanos deployment in HM.
- **Query** monitors the Thanos query API, providing rate and duration metrics as well as errors and more.
- **Receive** monitors Thanos incoming requests, providing rate and duration metrics as well as errors and more.
- **Store** monitors Thanos storage jobs, bucket, block, cache operations, and any related errors.

Description: Monitors the performance and health of the Thanos long-term metric storage system.

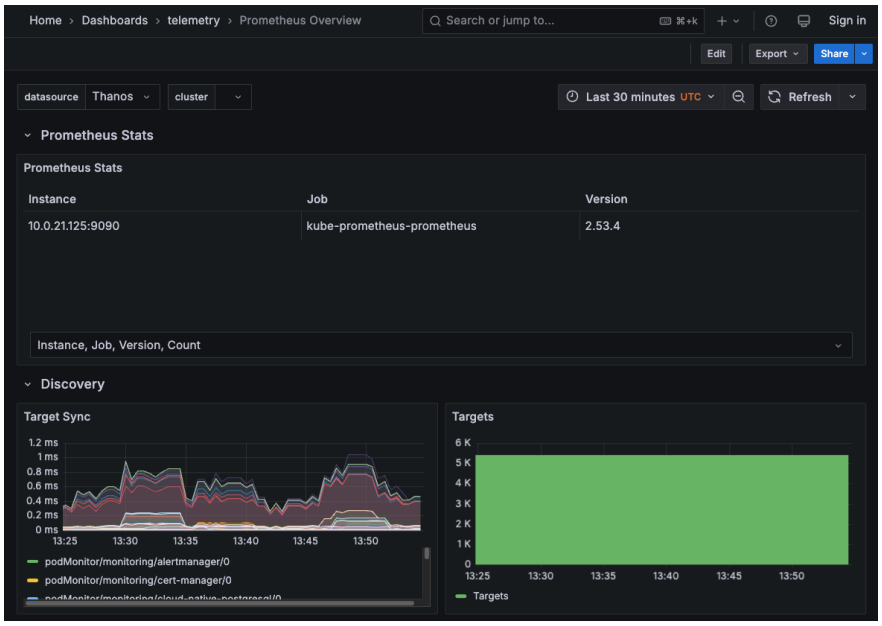


Fig. 34: Telemetry Prometheus dashboard

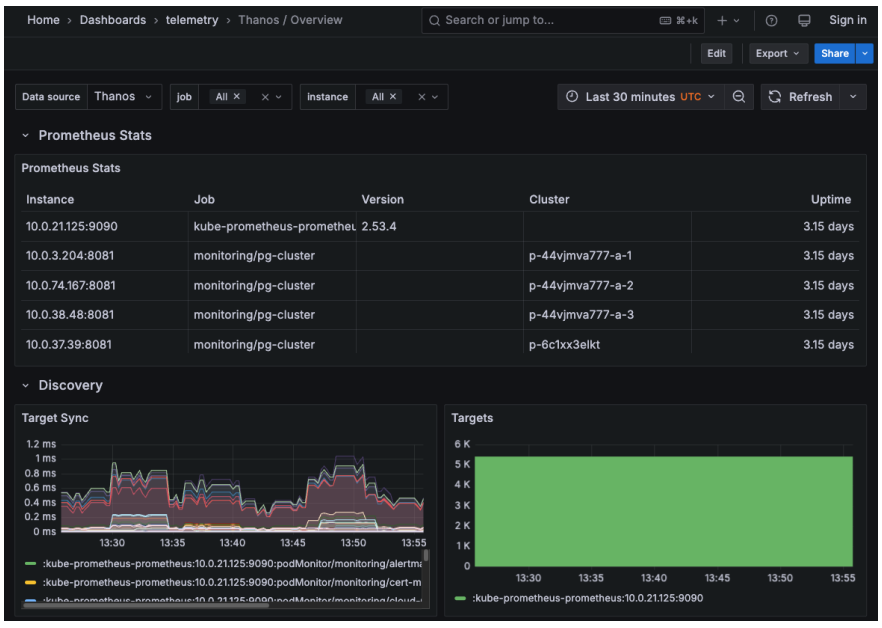


Fig. 35: Telemetry Thanos dashboards

Metrics: Statistics of Thanos jobs, scrape interval durations, metrics on compaction, query, incoming request and store procedures, procedure latencies, errors, and CPU and memory usage.

Usage: Helps ensure that the long-term storage of metrics is being managed efficiently by Thanos. Issues with the compactor can lead to storage inefficiencies or data loss. Slow Thanos queries and processing of incoming requests can affect the responsiveness of Grafana dashboards and so on.

Other dashboards

Power Consumption: Power Usage & Carbon Footprint

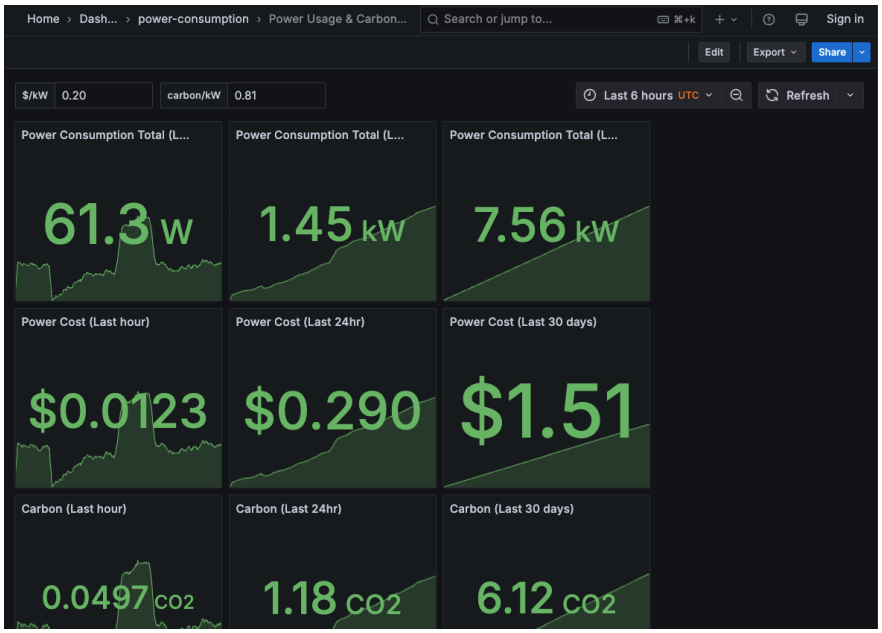


Fig. 36: Power usage and carbon footprint dashboard

Description: Monitors the energy consumption and estimated carbon footprint of the Hybrid Manager (HM) infrastructure. Use this for tracking and potentially optimizing environmental impact.

This dashboard was designed by EDB following a similar procedure as re:cinq's Cloud CPU Energy Consumption calculation methodology and uses the freely available power measurements from TEADS dataset as source data.

Metrics: Power consumption (in watts or kilowatts), estimated carbon footprint, and estimated power cost, broken down into graphs for the last hour, 24 hours, or 30 days.

Usage: Allows you to visualize the environmental impact of the HM infrastructure, track trends in energy usage, and potentially identify areas for optimization to reduce costs and environmental impact.

Infrastructure: cert-manager

Description: Monitors the health and performance of cert-manager , a Kubernetes add-on to automate managing and issuing TLS certificates. Use this to ensure the proper functioning of certificate management in HM.

Metrics: Number of issued certificates, certificate request status (pending, ready, failed), expiry times of certificates, and more.

Usage: Helps ensure that TLS certificates in HM are being managed correctly and that applications relying on them have valid and up-to-date certificates. Identify potential issues with certificate issuance or renewal before they cause outages.

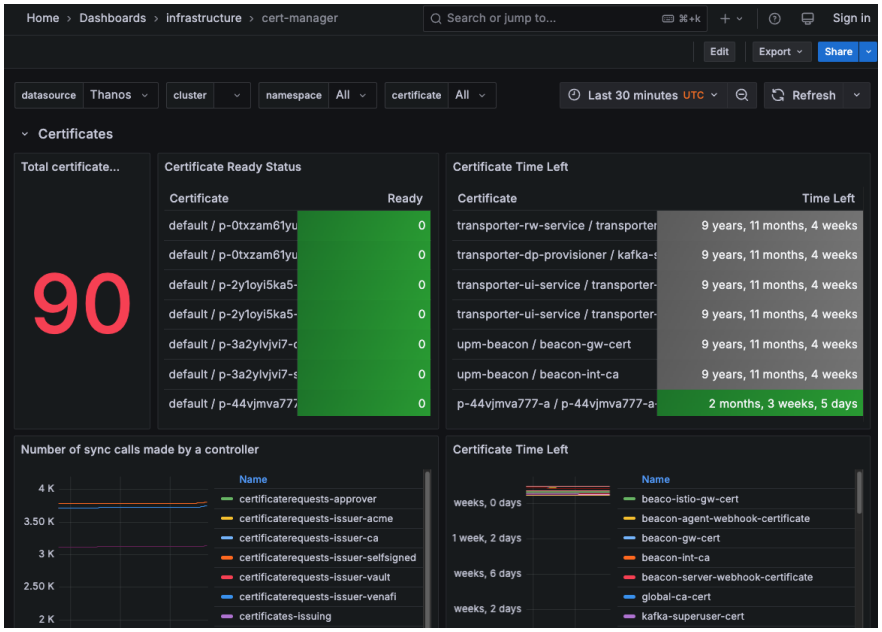


Fig. 37: Cert-manager dashboard

Infrastructure: Velero Stats

Description: Monitors the health and performance of Velero , a backup and restore tool for Kubernetes resources and persistent volumes. Use this to track the status and success of your backups in HM.

Metrics: Number of backups created, number of restores performed, status of backup and restore operations (completed, failed, in progress), backup size, restore duration, errors encountered during backups or restores, schedule status.

Usage: Provides visibility into the backup and restore operations in HM. Allows you to verify the success of backups, monitor restore processes, and identify any issues with your disaster recovery strategy.

Grafana's Explore option

Grafana offers an **Explore** option that allows you to query logs gathered by Loki.

To query the logs from a specific cluster:

1. Obtain your *cluster's ID and namespace* . Select `cnp_instance_id` .
2. In the **Label filters** menu, select **cnp_instance_id**.
3. In the **Select value** menu, select the pod's ID.
4. Craft your query to look for certain key words in that instance's logs, for example, "error" or "exception."

Troubleshooting in Grafana

The `no healthy upstream` warning in Grafana dashboards indicates that a service can't connect to its intended backend. This issue can be temporary or persist for a longer duration, impacting the usability of your monitoring and debugging tools. Understanding the common causes can help you to diagnose and resolve these issues.

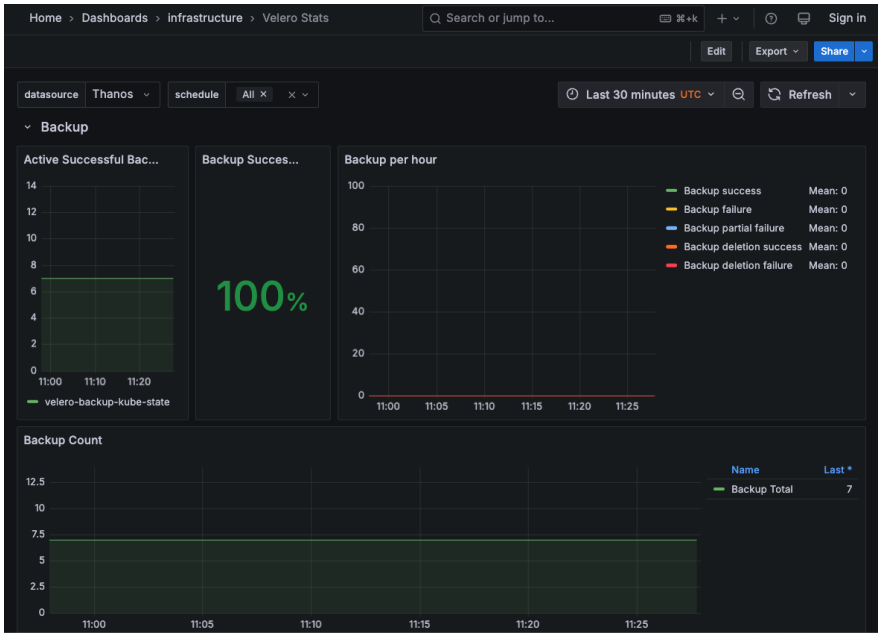


Fig. 38: Velero dashboard

Potential causes

Pod eviction or movement

Dynamic node provisioning and pod scheduling tools like Karpenter can cause pods to be moved between nodes. During this process, a pod might temporarily become unavailable, leading to `no healthy upstream` errors until it's rescheduled and becomes ready on a new node.

Resource overscheduling and out-of-memory (OOM) errors

Services, especially those handling complex or large queries (like Loki), can consume significant memory. These tools generally don't constrain queries. They will attempt to execute a query that might return terabytes of logs or gigabytes of metrics and OOM rather than identifying and flagging queries that may not succeed with the memory allocations.

If the cluster nodes are oversubscribed or if pods aren't configured with appropriate resource limits, Kubernetes can kill them due to memory pressure on the host. Even if the pod recovers quickly, the brief unavailability during the OOM event can trigger `no healthy upstream` warnings.

Transient network issues or connection termination

Intermittent network problems, connection resets, or abrupt connection terminations before headers are received can also appear as `no healthy upstream.`

Service unavailability or misconfiguration

While less common for transient issues, a `no healthy upstream` message can also indicate that the backend service is unhealthy, not running, or misconfigured, preventing Istio from routing traffic to it successfully.

Troubleshooting steps

1. **Optimize queries:** Address the most common and direct cause of OOMs for these specific backends. Since Loki, Thanos, and Prometheus don't guardrail queries, the first thing to look at when these backends start failing is whether overly broad or complex queries are causing them to have OOM issues.

Try narrowing your queries by reducing their time window, selecting a smaller set of log metrics, or using more specific filtering (stream selectors, line matching, JSON queries, avoiding regex expressions). See Query best practices for more information.

1. **Monitor resource utilization:** Check the resource usage of your monitoring workloads (Loki, Thanos, Prometheus). Review *resource utilization metrics for the affected pods* and their underlying nodes (CPU, memory). This helps to identify whether resource overscheduling or lack of capacity is the root cause. For example, check the *Loki dashboards for logging monitoring* . Verify the configured *Resource commitments* , such as resource *Recommended memory limits* and *Requests* for the pods.

If resource utilization looks high and OOM errors persist, then consider adjusting the pod's resource requests/limits or the node's capacity. If the node has enough memory capacity but the pod is undersubscribed, consider increasing pod memory limits. If the pods are oversubscribed and the node needs more memory capacity to host the subscribed pods, consider increasing the node memory.

1. **Check Kubernetes events and logs:** Check Kubernetes events (Grafana's Explore option) for pod evictions, scheduling failures, or node changes. Logs are a good source for looking at a pod's health and specific OOM events. They help you understand which pods and processes are failing.
2. **Verify network health:** This is a more general check and less likely to be the primary cause of OOM issues. Verify the health of the underlying *network infrastructure* .

Enabling usage reporting in the HM server

The Hybrid Manager (HM) server transmits usage telemetry to EDB to automate your core-based licensing and entitlements. This ensures your environment stays compliant with provisioning limits without the need for manual audits or tracking. This reporting works for all on-premises servers, even if you are not using HM for active monitoring.

Usage reporting with the HM server is an opt-in service, hence it is not enabled by default. Data is only sent to EDB when you explicitly enable it. When enabled, the HM server sends a new report every 24 hours by default, starting with the first time the HM server runs.

What type of data is collected?

When configured to send usage data, the HM server generates a small report containing environment metadata and resource usage. This includes information such as:

- Operating system details
- Memory and CPU usage
- Installed database distribution (for example, PostgreSQL, EDB Postgres Advanced Server, EDB Postgres Extended Server) and version

Important

This report does not include sensitive data or user-identifiable information.

Where is the data sent to?

When configured, reporting data goes to https://hm-usage.enterprisedb.com, an EDB-managed endpoint.

How do I enable usage reporting with the HM server?

If you are using the Hybrid Manager, you can enable reporting as part of your HM server's configuration. See *Enabling HM server usage reporting* .

If you have a standalone deployment of a database server (PostgreSQL, EPAS, PGE) and you don't wish to use HM, see Enabling usage reporting for standalone databases .

If you are operating in an air-gapped environment, you can create a usage report as a JSON file within the EDB Postgres AI agent's configuration page and upload it through the option **PGAI Agent Reporting** under the **Tools** section in the left side pane of the **EDB Customer Portal**.

Enabling HM server usage reporting

You can enable usage reporting for HM via the following methods:

Enable HM server usage reporting through the operator

To enable telemetry and server metrics collection, you must configure the `HybridControlPlane` custom resource with your specific company and installation details.

Requirements

- Access to the cluster where the operator is installed.
- Your unique company code is provided by the administrator.

Enable usage reporting (operator)

1. Open the `HybridControlPlane` resource for editing, using `kubectl` to edit your existing control plane configuration:

```
kubectl edit hybridcontrolplane <resource-name> -n <namespace>
```

1. Under the `spec` section, ensure the `globalParameters` block includes your company code:

```
spec:
  globalParameters:
    company_code: "YOUR_COMPANY_CODE"
```

1. Add the `enableEDBUsageReporting` and `hybridManagerInstallation` fields to the `beaconServer` section:

```
spec:
  beaconServer:
    enableEDBUsageReporting: true
    hybridManagerInstallation: <install-name>
```

1. Save your changes. The operator reconciles the updated spec automatically; once `HybridControlPlane` is in a healthy state, HM server will begin initializing the telemetry services.

Important

***2_blank_4_*- If *enableEDBUsageReporting* is set to *false* , usage data is not collected or sent to EDB.`1_blank_3_-` If *enableEDBUsageReporting* is set to true, the fields *company_code* and *hybridManagerInstallation* are required. If either field is missing, the HM custom resource moves to a failure state and returns an error indicating that these fields are required when usage reporting is enabled.**
.. important::
Enable HM server usage reporting through the Helm chart

To enable telemetry and server metrics collection during a Helm deployment, you must define the global company identity and the Beacon Server status in your configuration.

Requirements

- A functional Helm environment.
- Your unique company code is provided by the administrator.

Enable usage reporting (Helm)

1. Locate your `values.yaml` configuration file used for your specific deployment.
2. Under the `parameters.global` section, add your company code:

```
parameters:
  global:
    company_code: "YOUR_COMPANY_CODE"
```

1. Enable server metrics collection by setting `enableEDBUsageReporting` to `true` and `hybridManagerInstallation` to `<install-name>` :

```
beaconServer:
  enableEDBUsageReporting: true
  hybridManagerInstallation: <install-name>
```

1. For first-time installations, apply the changes:

```
helm install <release-name> <chart-path> -f values.yaml
```

For updating an existing installation, run the `helm upgrade` command:

```
helm upgrade <release-name> <chart-path> -f values.yaml
```

- If `enableEDBUsageReporting` is set to `false` , usage data is not collected or sent to EDB.
- If `enableEDBUsageReporting` is set to `true` , the fields `company_code` and `hybridManagerInstallation` are required. Helm fails with an error if either value is missing while usage reporting is enabled. !!!

Notifications

With Hybrid Manager (HM), you can customize the types of notifications you want to receive. The two types of notification channels are:

User notifications : You can receive notifications via email, HM inbox, or both. These notifications are for all users who have enabled them and if their role allows it.

System notifications : You can receive notifications via email (mail group), webhook, Pager Duty, or any combination. These notifications are triggered once per event. System notifications are driven by activity log events, including cluster lifecycle changes (create, update, delete), security and index recommendations, automation task activity, and other platform events.

The user's role in HM determines the extent of their access to notification preferences. The notifications are categorized into the following preference sections:

- Account : For user notification channels.
- Global : For user and system notification channels.
- Projects : For user and system notification channels.

The notifications under the Account preference section are:

- New project role assigned to you
- Project role unassigned from you
- The personal access key is about to expire

The notifications under the Global preference section are:

- New project created
- The machine access key is about to expire

> **Note**
>
> This section is visible only to the organization owner.

The notifications under the Projects preference section are:

- Active cluster paused
- Paused cluster reactivated
- Failed cluster maintenance upgrade
- User invited to project
- Failed third-party monitoring integration

Configuring notifications

Project owners/editors and organization owners/admins can enable/disable notifications for the events visible to them. They can receive notifications in the HM inbox, by email, or both.

Notification settings made by a user apply only to that user. If an email notification is enabled, the email is sent to the email address associated with the user's login.

For org-level and project-level notifications, HM determines who receives each notification by cross-referencing the event's target audience with each user's role and their enabled notification preferences. Both conditions must be met: the user's role must qualify them to receive that notification type, and they must have it enabled in their preferences. Account-level notifications are always delivered only to the individual user they concern.

System notification channels can be configured at the global level by the organization owner or at the project level by the project owner. By default, each organization's project inherits the global-level system notifications channel. However, you can enable/disable or reconfigure the system notifications channel at the project level to override the default settings.

Notification settings follow a scope hierarchy: global settings apply across all projects, and project-level settings override them. Configure at the lowest scope that fits your needs — global for organization-wide defaults, project-level for project-specific routing.

Configure notifications at the global level

Only the organization owner can set the global-level notifications.

Webhook

1. In the HM console, from your profile menu, select **Settings**.
2. Select the **Notifications** tab.
3. Select the edit icon next to **Webhook** and provide the details:

- **Host** : The endpoint URL of the webhook.
- **Method** : HTTP method for the request: POST or PUT.
- **HTTP Headers** (optional) : Custom HTTP headers to include in the request, as key-value pairs.
- **Authentication** : Select an authentication type and provide credentials: - **Basic Auth** : Enter an optional username and password. - **Bearer Token** : Provide a bearer token.

1. Select **Save**.

Pager Duty

Before configuring the Pager Duty notification channel:

- You need access to a Pager Duty account with elevated privileges.
- A Pager Duty service must be created and configured with Events API v2 integration.
- In your Pager Duty account, navigate to the desired service and access its **Integrations** settings.
- Add an Events API v2 integration and configure it according to your requirements.
- Copy the integration key (routing key) provided by Pager Duty for the newly created integration.

Then you can configure the channel:

1. In the HM console, from your profile menu, select **Settings**.
2. Select the **Notifications** tab.
3. Select the edit icon next to **Pager Duty** and set the **Routing Key**.
4. Select **Save**.

Email

1. In the HM console, from your profile menu, select **Settings**.
2. Select the **Notifications** tab.
3. Select the edit icon next to **Email** and enter the details:

- **Email To** : Optionally, a group email address for system notifications. HM sends every system notification for this scope to this address — not just events relevant to the person who configured the channel. Use a shared inbox or distribution list, not a personal email address.
- **Email From** : The email address that sends the notification.

- **Host** : URL of the email service.
- **Port** : Port of the email service.
- **Username** : Username of the email service.
- **Password** : Password of the email service.

> **Note**
>
> If **Email To** is set, HM delivers notifications to that address in addition to any eligible individual users — not instead of them. If **Email To** is left blank, only eligible individual users receive email notifications at their HM account email address. !!!

1. Select **Save**.

Configure notifications at project level

Project-level notification settings can override global settings. Each project can have a unique configuration.

1. In the HM console, select the **Projects** tab.
2. Select any project.
3. Go to the Project Overview page.
4. In the left navigation, select **Settings**.
5. Select **Notifications**.
6. Specify settings as described in *Step 9: Transfer Helm ownership - Webhook Configurations* , *Pager Duty* , or *Email* .

Viewing notifications in the HM console

Users in the following roles can view the notifications:

- Organization owners can view the global notifications.
- Project owners/editors can view the project notifications.
- All users can view their account notifications.

Each notification indicates the level and project the user belongs to with multiple roles assigned in HM.

Select the bell icon at the top of the HM console to view notifications. You can read a notification, mark it as unread, or archive it.

Some notifications — particularly task-related ones — include a direct link to the affected resource. Selecting the link takes you to the relevant page, provided you have the required access permissions.

When you have unread broadcast notifications — platform-wide announcements such as maintenance windows — a persistent banner appears at the top of the console. If there are multiple unread broadcasts, banners stack. Select **Mark as read** on a banner to dismiss it.

Managing notification preferences

1. In the HM console, from your profile menu, select **My Account**.
2. Select the **Notifications** tab. Notifications are grouped by account, organizations, and projects available to you.
3. Scroll to the **Account**, **Global**, or **Project** section to view the events you want to manage.

4. Select the right arrow next to the entity to expand the list of events.
5. For each notification, you can: - Use the toggle next to the event name to opt in or out of receiving notifications for that event. - When the event toggle is enabled, select the check box for each channel (email, in-app, webhook, Pager Duty) to control which channels deliver that notification.

Alerts

Alerting is integral to the Hybrid Manager (HM) observability ecosystem, which proactively notifies about critical events that impact the performance of the Postgres cluster. It continuously monitors essential performance metrics and triggers alerts whenever predefined thresholds are exceeded. This robust alerting system integrates seamlessly into the HM ecosystem, leveraging real-time data collected from Postgres clusters and their infrastructure across all regions. Alerts are enabled by default for each project created in HM.

The alerting system ensures timely notifications through intuitive notifications, allowing for quick and informed actions. Comprehensive alert configuration options allow specific thresholds and intervals to be configured to customize the alerting experience. This seamless integration of monitoring and alerting offers complete visibility, enabling swift responses to potential issues and maintaining optimal cluster performance.

View the alerts

You can access the alerts on the **Monitoring** tab of the cluster detail page and view the alert details for a specific cluster.

- For HM clusters:

1. In the HM console, select **Projects**.
2. Select any project from the list.
3. In the left navigation, select **Clusters**.
4. Select any cluster with a ready state.
5. On the cluster detail page, select the **Monitoring** tab.

- For external HM clusters:

1. In the HM console, select **Estate**.
2. Go to the table listing all the clusters.
3. Select a ready cluster with Management as self-managed.
4. On the cluster detail page, select the **Monitoring** tab.

The **Monitoring** tab displays the Active Alerts list in a table. At the top of this tab, you can view a summary of the active alerts with their severity levels set to High, Medium, and Low.

You can search for specific alerts using the:

- Search box for query-based search
- Date range selector for a specific date range-based search
- Filter for a severity-level-based search
- Sort utility to sort the alerts list based on severity, alert start time, and alert name
- Refresh button to refresh the alerts list
- Column selector to select the columns to display in the table according to your preferences
- Download button to download the alerts list in a CSV format

Select **Refresh** to manually refresh the alerts. The last refresh interval is displayed at the bottom of the alert table.

Select any nodes in the Node column to view the charts as they were when the alert was triggered. Selecting a node applies the filters in the **Monitoring** tab and displays the relevant charts.

Download CSV — You can download the alert details in CSV format using the Download button in the top-right corner. The CSV file includes these columns:

- Alert Description
- Start Time
- Alert Severity
- Metric Triggering the Alert
- Node on which the alert has occurred
- Alert Threshold
- Cluster ID
- Project ID

Configure alerts

You can configure alerts at the project level. The project admins can configure the alerts. Once the alerts are configured and applied, the system requires a few minutes to fully process the alerts.

To configure alerts:

1. At the top of the HM console, select **Projects**.
2. Select any project from the list.
3. In the left navigation, select **Settings**.
4. Select **Alerts** to access/edit the alerts configuration.

For each alert, select the **Edit** button to:

- Enable/Disable — To activate or deactivate a specific alert as required.
- Notify After — To set the time duration after which the alert is triggered.
- Threshold — To specify the percentage or values that will trigger an alert when they're exceeded.

The list of the alerts with the default configuration:

These alerts can be configured at database level:

- Connections Usage
- No Recent WAL Archiving Successes
- Node Down
- Replication Lag in Bytes
- Total Connections
- Total Inactive Replication Slots
- Transactions Rollback
- WAL Keeps Growing
- WAL Running Out of Disk Space

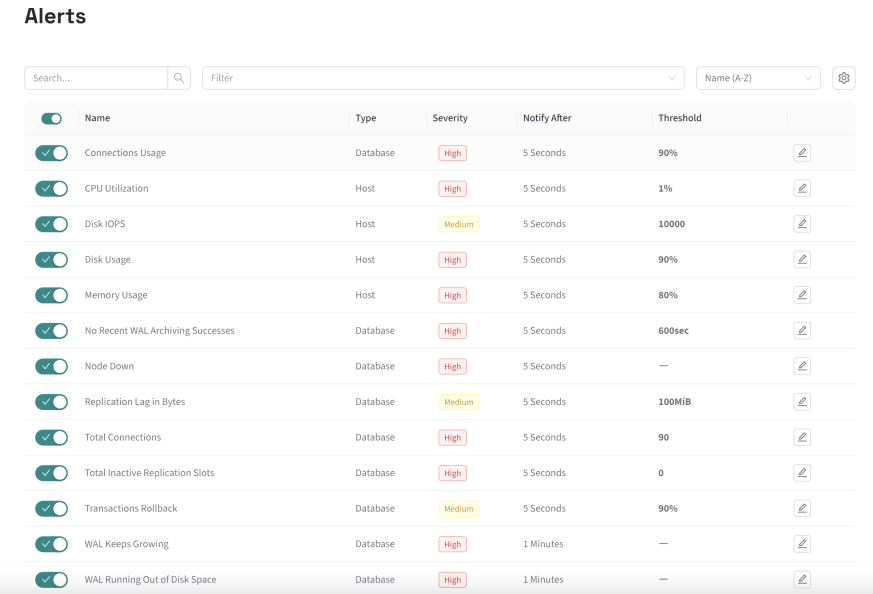


Fig. 39: Alerts

These alerts can be configured at host level:

- CPU Utilization
- Disk IOPS
- Disk Usage
- Memory Usage
- Any changes made to the alert configuration are saved automatically.
- All the alerts aren't available for external HM clusters due to technical limitations.

Auditing platform activity

The **Activity Log** is a chronological record of everything that happens in your Hybrid Manager (HM) environment — cluster changes, automation task executions, user actions, access control changes, and more. Use it to understand what changed, who made the change, and when.

The **Activity Log** is also the event source for *Set up notifications (Observability)* . When the **Activity Log** records a platform event — such as an automation task completing or HM applying a recommendation — HM uses that event to trigger any notification channels you've configured at the project or organization level.

Accessing the Activity Log

The **Activity Log** is available at two levels:

- **Project level** — Within a project, in the left navigation, select **Activity Log**. This view shows all events scoped to that project and is available to all users with a project role.
- **Organization level** — In the organization navigation, select your profile name and then select **Activity Log**. This view shows events across all projects in your organization, with a **Project** column to identify the source. It requires the **Organization Owner** or **Organization Administrator** role.

Reading the log

Each row in the **Activity Log** represents a single event. The columns are:

Column	Description
Activity Name	What happened — for example, a cluster creation, a task approval, or a user role change. For task-related events, the activity name includes the task name for easy identification.
User	The user or system process that triggered the event.
Date	When the event occurred.
Resource	The type of object affected. Common values include: **Cluster** , **Cluster Template** , **Catalog** , **Task** , **Task Approval** , **User Roles** , **Project** , **Identity Provider** , **Notification Credentials** , and **Role Mapping Rule** . The organization-level view also shows additional resources, like **Access Key** , **User** , **Organization** , and **Tag** events.
Project	(Organization-level view only) The project the event belongs to.

Filtering and searching

Use the filters at the top of the log to narrow down results:

- **Date range** — Preset ranges of 7, 15, and 30 days are available, or set a custom range.
- **Resource** — Filter by resource type (for example, show only **Cluster** or **Task events**) and optionally by a specific resource within that type.
- **Project** — (Organization-level view only) Filter events to one or more specific projects.
- **Search** — Find events by name or resource.

Exporting activity data

Select **Download CSV** to download the current filtered view as a CSV file. This is useful for compliance audits or sharing activity history with stakeholders outside HM.

Connecting activity to notifications

Any event the **Activity Log** records can trigger a system notification if you've configured a notification channel (webhook, Pager Duty, or email) at the project or organization level. This means you don't need to monitor the **Activity Log** manually — configure your channels and HM pushes the relevant events to your external tools automatically.

See *Set up notifications (Observability)* for steps on configuring channels and managing which events you receive.

5.3.4.3 Upgrading Hybrid Manager

Hybrid Manager (HM) offers a *Hybrid Manager dual release strategy* : **LTS (Long-Term Support)** and **Innovation Releases (IR)**. Because these streams follow different versioning logic, the supported paths for moving among them depend on your current version and the target environment.

> **Tip**
>
> If you want to upgrade your **Postgres database clusters** instead, see *Upgrading database clusters in Hybrid Manager* .

Innovation Releases follow a date-based versioning scheme (e.g., **2025.11**). These releases provide the latest features and platform hardening at a faster cadence.

However, these releases are not infinite, they are designed to converge into the next Long-Term Support (LTS) minor version.

Sequential requirement: You must upgrade month-to-month within the IR stream (e.g., 2025.11 → 2025.12).

Consolidation point: Twice a year, the IR stream "consolidates." To continue receiving updates after a consolidation point, you must transition to the newly released LTS version (e.g., 1.4.0), which then serves as the foundation for the next month's IR.

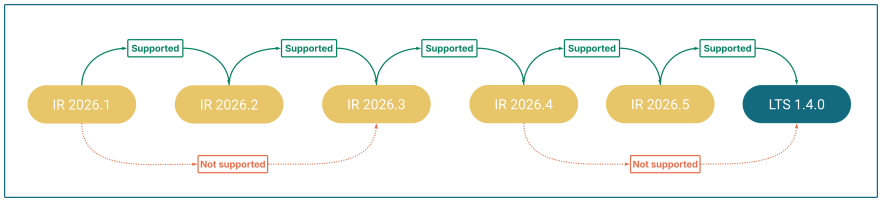


Fig. 40: IR sequential upgrade

5.3.4.3.1 LTS upgrade paths

LTS releases follow standard semantic versioning (`major.minor.patch` , e.g., **1.3.x**). These versions focus on stability and are intended for production environments requiring a long support lifecycle. Each LTS release is supported for 15 months from its release date, receiving monthly patch releases with bug and security fixes.

Minor and patch version upgrades

You can upgrade from one minor version to the immediate next minor release (e.g., **1.3.x** to **1.4.y**). However, you must first ensure you are running the latest available patch version of your current minor release before moving forward.

Within the same minor version, patch upgrades are flexible: you may upgrade from any patch version to any higher patch version (e.g., 1.3.1 to 1.3.5) without the need to install intermediate patches.

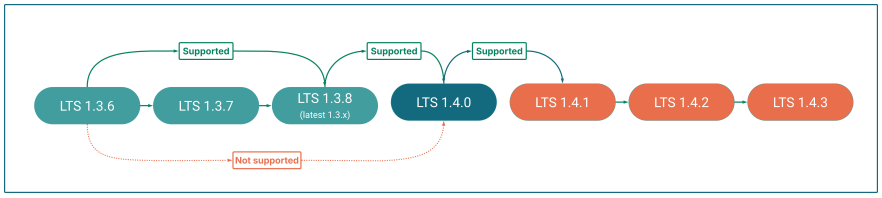


Fig. 41: LTS minor and patch releases

Major version upgrades

When a new major version (e.g., **2.0**) is released, you must be running the latest available **1.x** minor release to perform the upgrade.

5.3.4.3.2 Cross-stream upgrade paths

Moving between LTS and Innovation Releases is possible, but subject to specific constraints regarding version numbering and platform limitations.

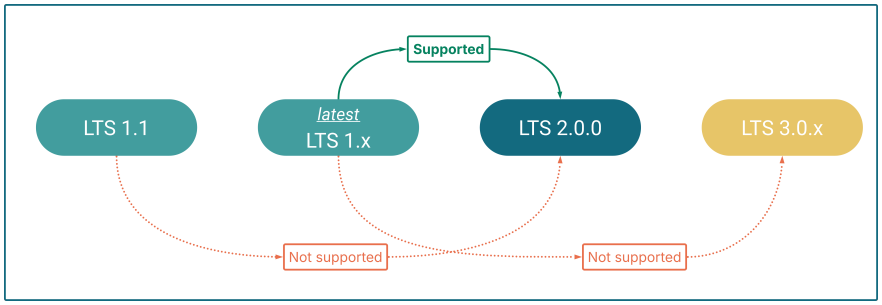


Fig. 42: IR sequential upgrade

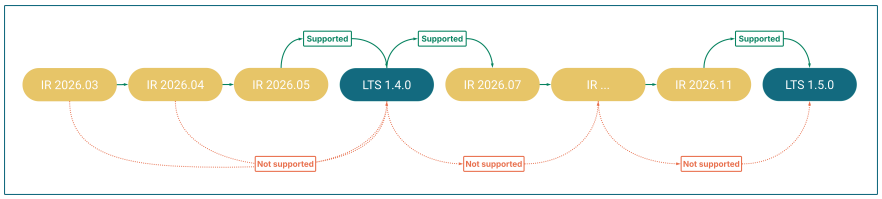


Fig. 43: IR sequential upgrade

Innovation Release to LTS

You can move from the **final Innovation Release** of a cycle (two cycles in a year) to the **LTS release** that consolidates those features, but you can't move to an LTS release from any other Innovation Release.

To determine if your current Innovation Release can move to an LTS version, refer to the documented upgrade instructions provided with each new release. These release-specific instructions will explicitly state if a version is an Innovation Release and whether it supports a transition to the LTS stream.

LTS to Innovation Release

You can upgrade an LTS release to its immediate succeeding Innovation Release.

It isn't possible to jump from an LTS version to an arbitrary or later Innovation Release. You must move to the immediate successor first, and then follow the sequential IR upgrade path.

> **Warning**
>
> Once you move from an LTS release to an Innovation Release, you cannot move back to an LTS release until the next consolidation point (see Innovation Release to LTS below). Consolidation points occur twice a year.

The impact of upgrades on your HM-managed Postgres database clusters depends on your specific upgrade path and the components being updated.

While many upgrades are designed to be non-disruptive to database traffic, certain system updates may trigger an automatic restart of your HM-managed Postgres database clusters. To avoid unexpected service interruptions, verify the specific requirements of your upgrade path.

Upgrading from 2026.3 to the 2026.4 Innovation Release

Upgrade from 2026.3 to 2026.4.

Prerequisites

- Administrative access to the Kubernetes cluster (`kubectl` , `helm`).
- Access to the HM installation configuration file or `values.yaml` file used for the current installation. To save the contents of your current installation configuration:

```
helm get values edbpgai-bootstrap -n edbpgai-bootstrap > values.yaml
```

- Required tools:
- `helm` (Latest version recommended)
- `kubectl`
- `edbctl` version `1.4.0` or later (or a build from `main`)

Upgrade 2026.3 → 2026.4

This release introduces the `dbaas` scenario, requires new secrets for `upm-system-db` , and includes mandatory pre-upgrade steps to prevent data corruption and resource conflicts.

Create new secrets for upm-system-db

Before upgrading, create the new secrets required by the `upm-system-db` component:

```
edbctl setup create-install-secrets --version v2026.4.0
```

Note

This command requires *edbctl* version *1.4.0* or a build from *main* . Earlier versions don't support the *--version* flag.

Label changes in 2026.4 make the existing `fluent-bit` StatefulSet and DaemonSet immutable. Delete both resources before upgrading to prevent the upgrade from failing:

```
kubectl delete statefulset fluent-bit-k8s-events -n edb-observability
kubectl delete daemonset fluent-bit -n edb-observability
```

The operator recreates them automatically during the upgrade.

Delete the statistics collector database

Warning

Complete this step before running the upgrade. The *upm-api-stats-collector* component bumps the Bluefin extension version in this release to address a critical WAL files issue. Existing data is incompatible with the new version and can't be migrated — upgrading without deleting the cluster first causes data corruption.

The statistics collector retains only seven days of telemetry data, so deleting the cluster is non-disruptive for most deployments. If you need to preserve this data, contact EDB Support before proceeding.

```
kubectl -n upm-api-stats-collector delete cluster stats-collector-db
```

If the cluster doesn't terminate promptly, force-delete the pods:

```
kubectl delete pod --force -n upm-api-stats-collector -l cnpg.io/cluster=stats-
↪collector-db
```

1. Verify all associated resources are removed before proceeding:

```
kubectl -n upm-api-stats-collector get cluster stats-collector-db
kubectl -n upm-api-stats-collector get pod stats-collector-db-1
kubectl -n upm-api-stats-collector get pvc stats-collector-db-1
```

Each command returns `NotFound` when the resource is gone. Proceed only once all three return `NotFound`.

Perform the upgrade

For Helm-based installations

1. Update your Helm repository:

```
helm repo update
```

1. If your SKU permits, add `dbaas` to the `scenarios` field in your `values.yaml`:

```
scenarios: "core,migration,ai,analytics,dbaas"
```

1. Perform the upgrade:

```
helm upgrade --install edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap \
  --version 2026.4.0 \
  --values values.yaml \
  --namespace edbpgai-bootstrap \
  --wait
```

For operator-based (CRD) installations

The `dbaas` scenario wasn't present in the 2026.3 CRD schema and can't be added while still running the 2026.3 operator. Upgrade the operator first, then update the `HybridControlPlane` resource.

1. Upgrade the `edb-hcp-operator` Helm chart to version `2026.4.0`:

```
helm template \
  --namespace edbpgai-bootstrap \
  --install \
  --version 2026.4.0 \
  --include-crds \
  edb-hcp-operator enterprisedb-edbpgai/edb-hcp-operator \
  --set "controllerManager.manager.image.repository=docker.enterprisedb.com/pgai-
↪platform/edb-hcp-operator/manager" \
  --set "controllerManager.manager.image.tag=2026.4.0" \
  --set "imagePullSecrets[0].name=edb-cred" \
  | kubectl apply -f -
```

1. Update the version and, f your SKU permits, add the `dbaas` scenario in a single patch command:

```
kubectl patch hcp edbpgai --type json \
  --patch [{"op":"add", "path":"/spec/scenarios/-", "value":"dbaas"}, {"op":"replace",
↪ "path":"/spec/version", "value":"v2026.4.0"}]
```

If you aren't enabling `dbaas`, update only the version:

```
kubectl patch hcp edbpgai --type json \
  --patch [{"op":"replace", "path":"/spec/version", "value":"v2026.4.0"}]
```

See *The dbaas scenario* for details on what happens if `dbaas` isn't enabled.

1. Trigger the upgrade by annotating the resource:

```
kubectl annotate hybridcontrolplane edbpgai --overwrite edbpgai.com/ready-for-
↪upgrade=true
```

1. The operator detects the version change and begins the rolling update. Monitor the progress with:

```
kubectl get hybridcontrolplane edbpgai -w
```

Enable beacon server usage telemetry

2026.4 introduces a new boolean field for enabling usage server telemetry collection. To enable it, set `enableEDBUsageReporting` to `true` alongside the existing required fields.

For operator-based installations, update your `HybridControlPlane` CR:

```
spec:
  beaconServer:
    enableEDBUsageReporting: true
    hybridManagerInstallation: "<your-installation-name>"
  globalParameters:
    company_code: "<your-company-code>"
```

For Helm-based installations, add the equivalent values to your `values.yaml`:

```
beaconServer:
  enableEDBUsageReporting: true
  hybridManagerInstallation: "<your-installation-name>"
parameters:
  global:
    company_code: "<your-company-code>"
```

After updating your configuration, re-annotate to apply the changes:

```
kubectl annotate hybridcontrolplane edbpgai --overwrite edbpgai.com/ready-for-
↪upgrade=true
```

The dbaas scenario

The `dbaas` scenario is new in 2026.4 and is separate from `core`. It enables database cluster management — including cluster provisioning, backups, storage locations, and templates. For an overview of all available scenarios, see *Phase 1: Planning your architecture*.

> **Warning**
>
> Existing installations upgrading from 2026.3 must explicitly add *dbaas* to their scenarios list to retain Postgres cluster management functionality. Without it, the Clusters UI is hidden and cluster management APIs return *404* — existing clusters continue running but are no longer manageable through the platform.

- Cluster management APIs return `404` .
- The **Clusters** section is hidden in the HM console.
- Existing Postgres clusters continue running but are no longer manageable through the platform.

You can add `dbaas` at any time after the upgrade by patching the CR and re-triggering reconciliation:

```
kubectl patch hcp edbpgai --type json \
  -p [{"op":"add", "path":"/spec/scenarios/-", "value":"dbaas"}]

kubectl annotate hybridcontrolplane edbpgai \
  --overwrite edbpgai.com/ready-for-upgrade=true
```

Troubleshooting

edb-migration-copilot fails to install

If the upgrade fails with an error similar to:

```
failed to execute kapp command, error: update secret/ragchew-ai-vendor-secrets (v1)␣
↪namespace: edb-migration-copilot
```

Delete the conflicting secret and re-trigger the upgrade:

```
kubectl delete secret -n edb-migration-copilot ragchew-ai-vendor-secrets
kubectl annotate hybridcontrolplane edbpgai --overwrite edbpgai.com/ready-for-
↪upgrade=true
```

Upgrading from 2026.2 to the 2026.3 Innovation Release

Upgrade from 2026.2 to 2026.3.

Prerequisites

- Administrative access to the Kubernetes cluster (`kubectl` , `helm`).
- Access to the HM installation configuration file or `values.yaml` file used for the current installation. To save the contents of your current installation configuration:

```
helm get values edbpgai-bootstrap -n edbpgai-bootstrap > values.yaml
```

- Required Tools:
- `helm` (Latest version recommended)
- `kubectl`

Upgrade 2026.2 → 2026.3

This release moves telemetry components to a new namespace, which requires a manual step before upgrading.

Prepare the telemetry namespace

Before upgrading, annotate the existing `logging` namespace to ensure telemetry components move cleanly to their new namespace:

```
kubectl annotate namespace logging kapp.k14s.io/delete-strategy=orphan --
↪overwrite=true
```

Perform the upgrade

For Helm-based installations

1. Update your Helm repository:

```
helm repo update
```

1. Perform the upgrade:

```
helm upgrade --install edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap \
  --version 2026.3.0 \
  --values values.yaml \
  --namespace edbpgai-bootstrap \
  --wait
```

For operator-based (CRD) installations

For installations managed via the operator, update the version specification within your `HybridControlPlane` resource.

1. Edit the `HybridControlPlane` resource:

```
kubectl edit hybridcontrolplane <resource-name>
```

Where `<resource-name>` is the name you assigned during installation.

1. In the YAML editor, locate the `spec.version` field and update it to `2026.3.0`.
2. Save the changes and force a reconciliation by annotating the resource:

```
kubectl annotate hybridcontrolplane <resource-name> --overwrite edbpgai.com/ready-for-
↪upgrade=true
```

1. The operator detects the version change and begins the rolling update. Monitor the progress with:

```
kubectl get hybridcontrolplane <resource-name> -w
```

Clean up

After the upgrade is complete, delete the `logging` namespace:

```
kubectl delete namespace logging
```

Upgrading from 2026.1 to the 2026.2 Innovation Release

Upgrade from 2026.1 to 2026.2.

Prerequisites

- Administrative access to the Kubernetes cluster (`kubectl` , `helm`).
- Access to the HM installation configuration file or `values.yaml` file used for the current installation. To save the contents of your current installation configuration:

```
helm get values edbpgai-bootstrap -n edbpgai-bootstrap > values.yaml
```

- Required Tools:
- `helm` (Latest version recommended)
- `kubectl` , `yq` , `base64` , `jq`

Upgrade 2026.1 → 2026.2

This release introduces a redesigned Identity Provider (IdP) configuration system and new Custom Resource Definitions (CRDs) that require manual pre- and post-upgrade steps.

Changes from previous release: IdP migration

- the `config.yaml` , which previously loaded from the `upm-dex` secret, is now mounted from a ConfigMap. Static connector configurations are no longer supported — connectors must be provisioned using the IdP Management API. Env var values like `$DB_USER` and `$DB_PASSWORD` are now obtained from the `upm-db-url` secret, which is propagated to the `upm-dex` namespace automatically.
- A new `hm-portal-bootstrap` secret must be created in the `default` namespace. It is replicated to the `upm-dex` namespace automatically via the reloader annotation. It contains:
- `fernet-key` — encryption key for sensitive connector config fields
- `static-passwords.yaml` — initial user accounts for bootstrap access
- New CRDs are introduced in 2026.2 and must be applied manually before the upgrade, as Helm skips CRD changes.
- After a successful migration, manually remove the `upm-dex` and `dex-fernet-key` secrets.

1. Back up the current IdP settings

From the HM portal, log in with an owner account and go to `https://<portal_dns_name>/settings/identity-providers` . For each IdP, save all settings including fields that are only partially visible on screen.

> **Warning**
>
> The HM IdP API does not return sensitive fields such as *clientSecret* (OIDC) or *BindPassword* (LDAP). Retrieve these values directly from your actual Identity Provider (e.g., Okta, Google) before proceeding. !!!

- For EDB Internal: back up the `upm-dex` secret directly:

```
kubectl get secret upm-dex -n upm-dex -o yaml > upm-dex-backup.yaml
```

- Optionally, export the IdP connector configuration from the CLI:

```
kubectl get hcp edbpgai -o json | jq .spec.pgai.portal.authentication.idpConnectors >␣
↪idps.json
```

1. Generate a certificate file from the existing IdP configuration

Extract the certificate from your existing IdP connector:

```
kubectl get hybridcontrolplane edbpgai -o yaml \
    | yq .spec.pgai.portal.authentication.idpConnectors[0].config.caData \
    | base64 -d > okta-cert.pem
```

Note

Adjust the array index if you have multiple connectors. Keep *okta-cert.pem* — you will need it in step 8. !!!

1. Extract the fernet key from the `dex-fernet-key` secret:

```
DEX_FERNET_KEY=$(kubectl get secret -n upm-dex dex-fernet-key \
    -o jsonpath={.data.fernet-key} | base64 -d)
```

1. Extract the static user passwords from the `upm-dex` secret's decoded `config.yaml` :

```
staticPasswords=$(kubectl get secret -n upm-dex upm-dex -o yaml \
    | yq .data."config.yaml" | base64 -d \
    | yq .staticPasswords)
```

Important

Migrate each user's *userID* exactly as extracted. Changing it creates a new user identity, resulting in a duplicate user with the same email and name, and orphans any existing data associated with the original user. !!!

1. Create the `hm-portal-bootstrap` secret in the `default` namespace using the values from steps 3 and 4:

```
kubectl create secret generic hm-portal-bootstrap \
    --namespace default \
    --from-literal=fernet-key="$DEX_FERNET_KEY" \
    --from-literal=static-passwords.yaml="staticPasswords:
$staticPasswords"

# Add the replicator annotation
kubectl annotate secret hm-portal-bootstrap --namespace default \
    replicator.v1.mittwald.de/replicate-to=upm-dex
```

The replicator annotation ensures it is propagated to the `upm-dex` namespace.

1. Apply the new CRDs using the appropriate method for your environment:

> **Warning**
>
> Helm skips CRD changes during upgrades. These must be applied before upgrading. !!!

If using Helm packages from the remote CS Helm repo:

```
helm pull edbpgai/edb-hcp-operator --version 2026.2.0 --untar
kubectl apply -f edb-hcp-operator/crds/preflight-crd.yaml
kubectl apply -f edb-hcp-operator/crds/postflight-crd.yaml
```

For EDB Internal — ephemeral clusters:

```
git clone -b 2026.2 git@github.com:EnterpriseDB/edb-hcp-operator.git
cd edb-hcp-operator/deploy/charts/edb-hcp-operator/crds
kubectl apply -f preflight-crd.yaml
kubectl apply -f postflight-crd.yaml
```

- Annotate the CRDs for Helm management:

```
kubectl annotate crd preflights.edbpgai.edb.com postflights.edbpgai.edb.com \
meta.helm.sh/release-name=edbpgai-bootstrap \
meta.helm.sh/release-namespace=edbpgai-bootstrap \
--overwrite

kubectl label crd preflights.edbpgai.edb.com postflights.edbpgai.edb.com \
app.kubernetes.io/managed-by=Helm \
--overwrite
```

1. Perform the upgrade

Bootstrap method installations

1. Update your Helm repository:

```
helm repo update
```

1. Perform the upgrade:

```
helm upgrade --install edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap \
--version 2026.2.0 \
--values values.yaml \
--namespace edbpgai-bootstrap \
--wait
```

Operator method installations

For installations using the Operator method, you must update the version specification within your `HybridControlPlane` resource.

1. Edit the `HybridControlPlane` resource:

```
kubectl edit hybridcontrolplane <resource-name>
```

Where `<resource-name>` is the name you assigned during installation.

1. In the YAML editor, locate the `spec.version` (or equivalent version field) and update it to `2026.2.0`.
2. Save the changes. Force a reconciliation by annotating the resource:

```
kubectl annotate hybridcontrolplane <resource-name> --overwrite edbpgai.com/ready-for-
↪upgrade=true
```

1. The operator will then detect the version change and begin the rolling update. You can monitor the progress with:

```
kubectl get hybridcontrolplane <resource-name> -w
```

1. Delete the auto-discovered IdP configuration:

After the upgrade, connect to the application database and run:

```
kubectl exec svc/app-db-rw -c postgres -n upm-beaco-ff-base -- \
  psql -U postgres upm -c \
  "UPDATE upm_api_admin.idp SET deleted_at = NOW() WHERE deleted_at IS NULL;"
```

1. Migrate connectors using the IdP Management API:

Before proceeding, confirm that `hm-portal-bootstrap` exists in the `upm-dex` namespace and that `upm-dex` pods are running:

```
kubectl get pods -n upm-dex
```

Using the portal UI

1. Log in with native credentials and go to **Settings → Identity Providers**.
2. Create a new IdP configuration with the following values:

Field	Value
Name	*okta* _(ensures the existing user config is maintained)_
Assertion Consumer Service URL	*https://<Portal URL>/auth/callback/okta*
Audience URI	*https://<Portal URL>/auth/samlentity/okta*
Connector ID	*okta*
Single Sign-On URL	From your backup in step 1
Signature Certificate File	*okta-cert.pem* — generated in step 2
NameID Format	*PERSISTENT*
User Attribute	*name*
Email Attribute	*email*

1. Click **Save**.

> **Note**
>
> **Save & Test** often fails until IT updates the ACS URL and Audience URI in your IdP with the new */okta* suffix. Raise a ticket with IT to request this update before testing. !!!

Using the REST API

Use the settings saved in step 1 to populate the request body:

```
curl -i -k -X POST https://<portal_url>/api/v1/idp \
--header x-access-key: <HM_org_owner_access_key> \
--header Content-Type: application/json \
--data {
    "general": {
    "name": "okta",
    "description": "Okta Identity Provider",
    "protocol": "OIDC"
    },
    "oidcSettings": {
    "issuer": "https://<your_okta_domain>.okta.com",
    "clientId": "your_client_id",
    "clientSecret": "your_client_secret"
    },
    "dexConnectorId": "okta"
}
```

1. Restart Dex:

Restart the Dex deployment to apply all configuration changes:

```
kubectl rollout restart deploy upm-dex -n upm-dex
```

1. Validate the migration:
2. Check that static user (native credentials) login works.
3. Check that the IdP-configured login works.

Clean up

After confirming both logins work, remove the old secrets:

```
kubectl delete secret upm-dex -n upm-dex
kubectl delete secret dex-fernet-key -n upm-dex
```

Upgrading from 2025.12 to the 2026.1 Innovation Release

Upgrade from 2025.12 to 2026.1.

Prerequisites

- Administrative access to the Kubernetes cluster (`kubectl` , `helm`).
- Access to the HM installation configuration file or `values.yaml` file used for the current installation. To save the contents of your current installation configuration:

```
helm get values edbpgai-bootstrap -n edbpgai-bootstrap > values.yaml
```

- Required Tools: - `helm` (Latest version recommended)

Upgrade 2025.12 → 2026.1

This release introduces new authentication secrets, and deprecates several Beacon parameters.

Create Langflow secret

Before installing the 2026.1 chart, you must manually create the `langflow-secret` secret in the `default` namespace.

1. Create a Langflow secret:

```
kubectl create secret generic langflow-secret -n default \
  --from-literal LANGFLOW_SUPERUSER=langflow \
  --from-literal LANGFLOW_SECRET_KEY="$(python3 -c "from secrets import token_urlsafe;
↪ print(f{token_urlsafe(32)})")" \
  --from-literal LANGFLOW_SUPERUSER_PASSWORD="$(openssl rand -base64 24 | tr -dc a-zA-
↪Z0-9 | cut -c1-16)"
```

This command:

```
- Generates a base64url-encoded application secret key (`LANGFLOW_SECRET_KEY` )␣
↪required for Langflow encryption (a secure token with 32 bytes is required). -␣
↪Associates the secret with the `langflow`  superuser (`LANGFLOW_SUPERUSER` ). -␣
↪Generates a secure password for the `langflow`  user (`LANGFLOW_SUPERUSER_PASSWORD`␣
↪).
```

1. Ensure the created secret is synchronized with the `upm-langflow` and `upm-beaco-ff-base` namespaces:

```
kubectl annotate secret langflow-secret -n default replicator.v1.mittwald.de/
↪replicate-to="upm-langflow,upm-beaco-ff-base"
```

Alternatively, you can create the secret manually.

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: langflow-secret
  namespace: default
  annotations:
    replicator.v1.mittwald.de/replicate-to: "upm-langflow,upm-beaco-ff-base"
type: Opaque
stringData:
  LANGFLOW_SECRET_KEY: <generated_langflow_secret_key>
  LANGFLOW_SUPERUSER: langflow
  LANGFLOW_SUPERUSER_PASSWORD: <generated_superuser_password>
    EOF
```

Update the `values.yaml`

In your HM installation configuration file or `values.yaml` , replace the following deprecated fields with new fields:

Remove field:	Replace with:
beaconAgent.provisioning.provider	*parameters.global.load_balancer_provider*
beaconAgent.provisioning.loadBalancersEnabled	*parameters.global.load_balancer_mode*
beaconAgent.provisioning.nodePortDomain	*parameters.global.node_port_domain*

Note

If you don't replace these fields, the HM installation upgrade will still suceed, but if in the future you have differing values set for both the old and new parameter, the new parameter will take preference over the old one.

Remove the `scenarios` stanza (only for multi-DC environments)

If you are running a multi-DC environment (if you deployed HM across multiple data centers), you must remove the `scenarios` stanza before upgrading. If your HM is a single-DC instance, skip this section and go to *Perform the upgrade* directly.

Due to schema changes in `scenarios` options for `v2026.1.0` , the operator fails to map values from `v2025.12.0` correctly. This triggers an unintended cleanup routine. To mitigate this, if your installation uses the `scenarios` parameter, the scenarios stanza must be removed prior to initiating the version upgrade. You can reapply the stanza after the upgrade.

You can find the new schema changes and scenario installation options in *Deploying Hybrid Manager across multiple data centers* .

Failing to follow these steps will cause the operator to ignore existing configurations and uninstall all components, including Postgres clusters.

For Helm-based installations

1. Open the `values.yaml` file of your primary instance (Hub).
2. Remove the entire `scenarios:` stanza.
3. Re-apply the configuration to your secondary instance (Spoke).

Do not proceed with the `v2026.1.0` upgrade until you have confirmed the secondary instance has finished reconciling and is in a healthy `Ready` or `Deployed` state.

For operator-based (CRD) installations

1. Edit your `HybridControlPlane` resource:

```
kubectl edit hybridcontrolplane <resource-name>
```

Where `<resource-name>` is the name you assigned during installation.

1. Delete the scenarios section from the specification.
2. Force a reconciliation by annotating the resource:

```
kubectl annotate hybridcontrolplane <resource-name> --overwrite edbpgai.com/ready-for-
↪upgrade=true
```

Do not proceed with the `v2026.1.0` upgrade until you have confirmed the secondary instance has finished reconciling and is in a healthy `Ready` or `Deployed` state.

Perform the upgrade

For Helm-based installations

1. Update your Helm repository:

```
helm repo update
```

1. Perform the upgrade:

```
helm upgrade --install edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap \
  --version 2026.1.0 \
  --values values.yaml \
  --namespace edbpgai-bootstrap \
  --wait
```

For operator-based (CRD) installations

For installations managed via the operator, you must update the version specification within your `HybridControlPlane` resource.

1. Update the `edbpgai-bootstrap` project from the catalog:

Openshift example:

- Navigate to **Operators** > **OperatorHub** in the OpenShift Console.
- Search for **EDB Postgres AI**.
- Follow the on-screen wizard to deploy into the `edbpgai-bootstrap` project.

1. Edit the `HybridControlPlane` resource:

```
kubectl edit hybridcontrolplane <resource-name>
```

Where `<resource-name>` is the name you assigned during installation.

1. In the YAML editor, locate the `spec.version` (or equivalent version field) and update it to `2026.1.0`.
2. Save the changes. Force a reconciliation by annotating the resource:

```
kubectl annotate hybridcontrolplane <resource-name> --overwrite edbpgai.com/ready-for-
↪upgrade=true
```

1. The operator will then detect the version change and begin the rolling update. You can monitor the progress with:

```
kubectl get hybridcontrolplane <resource-name> -w
```

Re-add the scenarios stanza (only for multi-DC environments)

If you removed the scenarios: stanza earlier to avoid the uninstallation bug, you can now re-integrate it. Update the installation configuration for the primary (Hub) and secondary locations (Spokes).

Upgrading from 2025.11 to the 2025.12 Innovation Release

Upgrade from 2025.11 to 2025.12.

Prerequisites

- Administrative access to the Kubernetes cluster (`kubectl` , `helm`).
- Access to the HM installation configuration file or `values.yaml` file used for the current installation. To save the contents of your current installation configuration:

```
helm get values edbpgai-bootstrap -n edbpgai-bootstrap > values.yaml
```

- Required Tools:
- `helm` (Latest version recommended)
- `cmctl` (Cert-Manager CLI) - *Required for 2025.11+*

Upgrade 2025.11 → 2025.12

This release introduces new authentication secrets, updates the Data Migration Service (DMS) configuration, and deprecates several Beacon parameters.

Create Fernet secret

Before installing the 2025.12 chart, you must manually create the `dex-fernet-key` .

```
FERNET_KEY=$(head -c 32 /dev/urandom | base64 | tr +/ -_)

kubectl create secret generic dex-fernet-key \
  --from-literal=fernet-key="$FERNET_KEY" \
  -n upm-dex
```

Update `values.yaml`

1. **Add DMS domain to ``values.yaml`` .**

Add a new global parameter `dms_domain_name` .

```
parameters:
global:
  dms_domain_name: <your-hm-dms-domain-name>
```

1. **Remove deprecated fields.**

Remove the following fields from your `values.yaml` :

- `beaconAgent.plaintext`
- `beaconAgent.beaconServerAddress`
- `beaconAgent.tlsInsecure`

- `beaconAgent.inCluster`

Critical infrastructure workaround

To ensure Postgres workloads schedule correctly, you must add a taint to a specific node pool.

- **Taint key:** `edbaiplatform.io/postgres`
- **Value:** `'true'`
- **Effect:** `NoSchedule`

Perform the upgrade

1. **Update your Helm repo.**

```
helm repo update
```

1. **Perform the upgrade.**

```
helm upgrade --install edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap \
  --version 2025.12.0 \
  --values values.yaml \
  --namespace edbpgai-bootstrap \
  --wait
```

Post-upgrade fixes

1. **Re-initialize SPIRE (if trust domain changed).**

In 2025.12, the `trust_domain` parameter is removed and defaults to `portal_domain_name` . If the `trust_domain` you set in version 1.3.0 differs from your Portal Domain, SPIRE breaks.

- **Fix:**

```
kubectl scale statefulset spire-server -n spire-system --replicas=0
kubectl delete pvc -l app=spire-server -n spire-system
kubectl scale statefulset spire-server -n spire-system --replicas=1
```

1. **Reconfigure models**

Knowledge Bases (KBs) may disappear from the UI after upgrade. To workaround, log in to the Portal, navigate to the **Model Configuration** page, and re-save your model settings.

Upgrading from 1.3.x to the 2025.12 Innovation Release

Upgrade from 1.3.x to 2025.11.

Prerequisites

- Administrative access to the Kubernetes cluster (`kubectl` , `helm`).
- Access to the HM installation configuration file or `values.yaml` file used for the current installation. To save the contents of your current installation configuration:

```
helm get values edbpgai-bootstrap -n edbpgai-bootstrap > values.yaml
```

- Required Tools:

- `helm` (Latest version recommended)
- `cmctl` (Cert-Manager CLI) - *Required for 2025.11+*

Upgrade 1.3.x → 2025.11

This release increases the Certificate Authority (CA) TTL to 10 years and refactors the Istio implementation on OpenShift.

Certificate renewal (required)

The internal self-signed CA has been updated. You must regenerate certificates to apply the new 10-year TTL.

1. Install `cmctl` (for example, `brew install cmctl`).
2. Regenerate certificates:

```
# Global CA
cmctl renew -n cert-manager global-ca-cert

# Leaf Certificates
cmctl renew -n upm-beacon beacon-int-ca
cmctl renew -n istio-system beaco-istio-gw-cert
```

OpenShift specifics: Istio Migration

Note

You can skip this step if you are not running on Red Hat OpenShift (RHOS).

HM is no longer using the OLM-managed Maistra Service Mesh with RHOS. You must manually clean up the old resources before the upgrade.

1. Scale down the operator:

```
kubectl scale deployment --replicas 0 edb-hcp-operator-controller-manager -n edbpgai-
↪bootstrap
```

1. Remove old Maistra resources:

```
kubectl delete servicemeshmemberrolls.maistra.io -n istio-system default
kubectl delete servicemeshcontrolplanes.maistra.io -n istio-system upm
```

1. Bulk delete members:

```
kubectl get servicemeshmember -A -ojsonpath={range .items[*]}{"kubectl delete␣
↪servicemeshmember "}{@.metadata.name}{" -n "}{@.metadata.namespace}{"\n"} {end} |␣
↪while read -r line; do eval $line; done
```

Important

Do not resume the operator at this point. Proceed with the upgrade. The Helm upgrade automatically scales the operator back up.

Perform the upgrade

1. Update the repo:

```
helm repo update
```

1. Perform the upgrade:

```
helm upgrade --install edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap \
  --version 2025.11.0 \
  --values values.yaml \
  --namespace edbpgai-bootstrap \
  --wait
```

5.3.4.4 Hybrid Manager backup and disaster recovery (DR)

Learn about disaster recovery (DR) for your Hybrid Manager (HM) installation and clusters. This section includes the following topics:

- *Multi-DC for high availability* : Provides an overview of the disaster recovery procedure and requirements.
- *Configuring backups to support disaster recovery (DR)* : Explains how to ensure the backup requirements for DR are met, while considering the recovery time objective (RTO) and recovery point objective (RPO) times.
- *Hybrid Manager disaster recovery* : Provides a step-by-step guide to perform a disaster recovery run of your HM instance and HM-managed Postgres clusters using Velero. This includes restoring the HM control plane and data plane from backups, as well as verifying the restore process.

5.3.4.4.1 Disaster recovery overview

Disaster can affect Hybrid Manager (HM) and make it unusable. For example, unavailability of the underlying Cloud Service Provider (CSP) region or an outage in a data center that makes hardware unusable can occur.

A disaster recovery (DR) option allows you to restore your databases to a point-in-time from your available backups. It involves recovering the **Control plane** (the HM console and HM components) and **Data plane** (your HM-managed Postgres clusters) after a catastrophic failure.

Unlike High Availability (HA), which provides Postgres database real-time failover, DR is a manual restoration process initiated when the original HM environment is unrecoverable.

Recovery scenarios

- **Restore HM to the original location**: You have two data centers (DC1, DC2), and HM runs in DC1. You need to restore HM from object storage to DC1.
- **Restore HM to an alternative location**: You have two data centers (DC1, DC2), and HM runs in DC1. You need to restore HM from object storage to DC2 (because DC1 is down for an unexpected period of time). The alternative location (DC2) must have access to the same (or replicated) object storage used by DC1.

Core components

DR relies on three components:

1. **Velero** — An open-source tool included with your HM installation that backs up Kubernetes resources (Definitions, Secrets, and ConfigMaps). It captures the configuration and state of your HM installation, as well as the definition for Postgres database clusters created with it.

2. **Object storage** — An S3-compatible bucket (AWS S3, GCS, etc.) that holds the Velero backups for the HM installation and its Postgres database clusters.
3. **HM console** — Use it to re-provision Postgres database clusters and bring back their data.

Scope: the two-layer restore strategy

To understand DR, you must distinguish between the two layers of data being recovered:

Layer	Content	Recovery method
Control plane	HM console and its applications, general settings, user accounts, and database definitions.	Velero restore, reinstalls Kubernetes definitions into the new HM installation
Data plane	Actual Postgres tables, rows, and indexes from database clusters created with HM.	HM console restore, leverages Point-in-Time Recovery (PITR) to roll data forward.

The backups for both data layers are stored in an S3-compatible bucket managed with Velero. You define the location and frequency of the backups during the initial HM configuration and installation.

Data recovery considerations

- **Migrations:** DR procedures don't restore active migration functionality. While the HM-internal `transporter-db` and `migration-db` databases are recovered, preserving migration history for audit and record-keeping, migrations that were active and impacted by the disaster will need to be re-created.
- **Query diagnostics and recommendations:** Data displayed in the HM console > **Query Diagnostics** and **Recommendations** tabs of a cluster's view is not recoverable after a disaster. This data is intentionally excluded from backups as it is transient. After HM is restored, it will automatically begin generating new data that reflects the current state of your Postgres clusters.
- **Monitoring:** Historical metrics (Thanos data)—displayed in the HM console > **Monitoring** tab of a cluster's view—are recoverable. Metrics will reappear automatically following the disaster recovery procedure.

Workflow: from outage to recovery

When a disaster occurs, the recovery follows this sequence:

1. **Infrastructure preparation** — Deploy a fresh Kubernetes cluster in a healthy region or data center, and deploy HM to it.
2. **Environment restore** — Use Velero to pull the HM configuration and Postgres database cluster definitions from your bucket. This brings back the HM console and the records of your previous databases.
3. **Database re-provisioning** — Log in to the restored HM console. Databases will appear as **Deleted** or **Missing**. Use the **Restore** function in the HM console to spin up new Postgres database clusters that point to the original data in object storage.

For this architecture to work, the following must be true **before** a disaster:

- **Offsite backups:** Your S3-compatible bucket must be replicated to a different region or physical site. If the bucket dies with the data center, DR is impossible. See *Object storage across locations* for more information.
- **Consistent credentials:** The new HM instance must have the same IAM permissions or secret access to the original object storage bucket to "see" the old backups.

Configuring backups to support disaster recovery (DR) .

5.3.4.4.2 Configuring backups to support disaster recovery (DR)

A successful disaster recovery (DR) outcome is directly tied to a robust backup strategy. While HM provides a default backup configuration, you should evaluate and implement supplemental schedules to ensure the environment aligns with your specific recovery time objective (RTO) and recovery point objective (RPO) requirements. HM backups are handled with Velero .

RTO and RPO

The ability to do any restore, and the associated RTO and RPO depend on the frequency and size of the backups.

- The **RTO** will be determined by the time it takes to deploy a new HM instance and restore the HM control plane and data plane (Postgres database clusters) from the available backups. The RTO will be affected by the size of the Postgres clusters, the number of Postgres clusters, and the size of the HM control plane data.
- The **RPO** will be determined by the frequency of the backups and the backup lifecycle policies. Critical data, such as the definition of the Postgres clusters, is stored as Kubernetes objects and included in the Velero backup. By default, this backup runs daily at 23:00, as defined by the default `velero-backup-kube-state` schedule. If your RPO requires more frequent backups, complement the default schedule by creating an additional custom backup schedule.

> **Note**
>
> While RTO and RPO targets are defined by your organization's service level requirements, the actual time and data recovery points achieved during a disaster will vary based on data volume and the manual nature of this procedure. We strongly recommend that you perform periodic disaster recovery exercises to validate that these manual steps can meet your defined RTO and RPO objectives.

During installation, you configure an *Other S3-compatible storage* for your HM instance that stores:

- Internal backups (HM control plane data)
- Postgres backups (Postgres database backups or data plane data)

There is a default storage location in your bucket for these backups. However, HM supports storing backup data into a custom storage location (**Managed Storage Locations**) in the same bucket. If you use custom storage locations, ensure these directories are included in your cross-region replication strategy to remain available during a regional outage.

All of this data needs to be available after a disaster. Depending on the criticality of the data and the level of disaster that you want to be able to recover from, you'll need to replicate this data outside of the CSP region or physical data center where the HM instance resides.

> **Tip**
>
> When using a bucket, you can achieve replication by using cross-region replication. For example, for AWS S3 buckets, see cross-region replication .

Default backup configuration

HM employs a dual-layered backup strategy to protect both the control plane and your data plane:

- **Postgres database data** — Managed via continuous backups. This ensures that once a cluster is restored, your data can be recovered to a specific point-in-time, limited only by your backup lifecycle policies.
- **HM configuration data** — Managed via Velero. This captures the "blueprints" of your environment, including the definitions of your Postgres clusters. By default, this occurs daily at 23:00 via the `velero-backup-kube-state` schedule.

If your RPO requires more frequent snapshots of your cluster definitions (for example, if you create or delete clusters frequently), you can create a custom Velero schedule.

Danger

Don't modify the default schedule, as it may be overwritten by a HM software update.

Adding a custom backup schedule

The following command creates a supplemental backup schedule that runs every 6 hours in addition to the default system backups. This ensures more frequent snapshots of your metadata without interfering with the built-in `velero-backup-kube-state` schedule. Note that `--snapshot-volumes=false` is used because the actual Postgres data is already protected by the continuous backup stream in your object storage.

```
velero create schedule <backup-name> --schedule="0 */6 " --include-namespaces=* --
↪include-resources=* --snapshot-volumes=false --ttl=168h --skip-immediately=false
```

To check how the backup schedule was configured in the backend, run the following command:

```
velero get schedules <backup-name> -o yaml | yq .spec
```

Which should return the following output:

```
schedule: 0 */6
skipImmediately: false
template:
  csiSnapshotTimeout: 0s
  hooks: {}
  includedNamespaces:
    - "*"
  includedResources:
    - "*"
  itemOperationTimeout: 0s
  metadata: {}
  snapshotVolumes: false
  ttl: 168h0m0s
useOwnerReferencesInBackup: false
```

Next topic

Hybrid Manager disaster recovery .

5.3.4.4.3 Hybrid Manager disaster recovery

The Disaster Recovery (DR) procedure is defined as the series of manual steps that you need to take to recover your HM installation and your HM-managed Postgres clusters.

> **Warning**
>
> You must constantly test and update your organization's DR procedure for it to remain valid.

Before you start

Before starting the DR procedure, ensure you have made yourself familiar with:

- *Multi-DC for high availability*
- *Configuring backups to support disaster recovery (DR)*

Prerequisites

- A new HM instance deployed and running. It must be running the same version as the old instance that failed or became unavailable.
- The container images used to build the clusters in the old, unavailable HM instance are available to the new one.

Required tools

Ensure the following tools are available on your workstation environment or *1. Management workstation (Bastion host)* :

- Velero CLI
- `edbctl` (EDB Hybrid Manager CLI - see the *Hybrid Manager CLI - edbctl*)
- `jq` (Command-line JSON processor)
- `yq` (Command-line YAML processor)

1. Make backups available in the new HM instance

The first step ensures the backups of the unavailable HM instance (“old backups”) are reachable from the new HM instance by copying the backups of the damaged HM instance to the linked storage (new bucket) of the new HM instance.

1. Obtain the bucket names, the backup ID, and the region of the new bucket. Store these as environment variables to be used in the commands throughout this guide:

> **Important**
>
> These variables are session-specific. If you open a new terminal tab or your session times out, the variables will be lost, and subsequent commands will fail. To avoid re-typing these values, you could save these export commands into a small shell script (e.g., *dr-env.sh*). You can then "source" that file in any new terminal window to instantly reload your environment .. Important:: edb_notranlate_0

How do I obtain the old bucket values?

To obtain the old bucket values:

1. Go to the console/dashboard of your CSP > buckets.
2. Find and select the bucket linked to the backups of your old HM instance.
3. Browse through to the `edb-internal-backups` folder. Inside that folder you will find a subfolder with the backup ID, e.g. 4be7a1c8c9f0.

EKS Example

This is an example for setting the environment variables for an HM instance deployed on EKS:

```
export OLD_BUCKET=eks-1105143903-2511-edb-postgres
export OLD_BACKUP_ID=a7462dbc7106
export NEW_BUCKET=eks-1105155418-2511-edb-postgres
export NEW_REGION=eu-west-3
```

2. To copy the data from the old bucket to the new bucket, you first need to locate and note the names of the source and target folders. You need to copy the following folders and their content:

- **Internal EDB backups folder** — The internal backups folder in the old bucket `edb-internal-backups/<random-string>` is different in the new HM instance, as it will have a different `<random-string>` .
- **Postgres clusters backups folder** — `customer-pg-backups` .
- **Folder corresponding to any defined custom storage locations** — If you utilize **Managed Storage Locations** in the HM console (e.g., for offloading Postgres queries), you must ensure the corresponding folders are copied from the old S3-compatible bucket to the new one. While the definitions are restored via Velero, the actual data inside those custom folders must be manually migrated to the new target bucket.

3. Copy the old backups to the new bucket using your preferred tools. Here are some examples using cloud service provider CLIs to move data between buckets:

```
aws s3 cp --recursive s3://${OLD_BUCKET}/edb-internal-backups/${OLD_BACKUP_ID} s3://$
↪{NEW_BUCKET}/edb-internal-backups
aws s3 cp --recursive s3://${OLD_BUCKET}/customer-pg-backups s3://${NEW_BUCKET}/
↪customer-pg-backups
```

If you have configured additional **Managed Storage Locations**, use the same method to copy those folders.

```
gcloud storage cp gs://${OLD_BUCKET}/edb-internal-backups/${OLD_BACKUP_ID} gs://${NEW_
↪BUCKET}/edb-internal-backups --recursive
gcloud storage cp gs://${OLD_BUCKET}/customer-pg-backups gs://${NEW_BUCKET}/customer-
↪pg-backups --recursive
```

If you have configured additional **Managed Storage Locations**, use the same method to copy those folders.

4. Load the backups you just copied to your new HM instance by creating a new custom resource definition and applying it to the new HM instance:

```
kubectl apply -f - <<EOF
apiVersion: velero.io/v1
kind: BackupStorageLocation
metadata:
  annotations:
    appliance.enterprisedb.com/s3-prefixes: edb-internal-backups/velero
  labels:
    appliance.enterprisedb.com/storage-credentials: bound
  name: recovery
```

(continues on next page)

(continued from previous page)

```
  namespace: velero
spec:
  accessMode: ReadOnly
  config:
    insecureSkipTLSVerify: "false"
    region: ${NEW_REGION}
    s3ForcePathStyle: "true"
  default: false
  objectStorage:
    bucket: ${NEW_BUCKET}
    prefix: edb-internal-backups/velero
  provider: aws
EOF
```

```
kubectl apply -f - <<EOF
apiVersion: velero.io/v1
kind: BackupStorageLocation
metadata:
  annotations:
    appliance.enterprisedb.com/s3-prefixes: edb-internal-backups/velero
  labels:
    appliance.enterprisedb.com/storage-credentials: bound
  name: recovery
  namespace: velero
spec:
  accessMode: ReadOnly
  credential:
    key: gcp
    name: gcs-credentials
  default: false
  objectStorage:
    bucket: ${NEW_BUCKET}
    prefix: edb-internal-backups/velero
  provider: gcp
EOF
```

5. Confirm that the new storage location is available:

```
velero get backup-locations
```

If the status is not `Available` , check the Velero pod logs for permission errors on the S3 bucket.

6. Confirm that the backups are available as well:

```
velero get backups --selector velero.io/storage-location=recovery
```

7. Choose the backup you want to restore from. You can have multiple backups available, so choose the one that best suits your needs, e.g. the most recent backup before the disaster happened. Note the Velero backup name, as well as the date and time (UTC), as both are required for a restore, for example:

```
NAME                                    STATUS      ERRORS   WARNINGS   CREATED     ␣
↪                    EXPIRES   STORAGE LOCATION   SELECTOR
```

(continues on next page)

(continued from previous page)

```
velero-backup-kube-state-**20241216154403**   Completed   0        0          2024-12-
↪16 16:44:03 0100 CET   5d        recovery           \<none\>
```

> **Note**
>
> The timestamp value is referred to as the *recovery date* in the instructions that follow.

8. (Optional) If you were using HM to manage AI workloads, e.g. with the GenAI Builder, also copy the object store files and CORS configuration from the old bucket to the new one:

```
export OLD_BUCKET_DATALAKE=<old-bucket-name>
export NEW_BUCKET_DATALAKE=<new-bucket-name>
```

```
# Copy data lake objects from old bucket to new bucket
aws s3 cp --recursive s3://${OLD_BUCKET_DATALAKE}/ s3://${NEW_BUCKET_DATALAKE}/
# Copy CORS configuration from old bucket to new bucket
aws s3api get-bucket-cors --bucket ${OLD_BUCKET_DATALAKE} --output json > cors-config.
↪json
aws s3api put-bucket-cors --bucket ${NEW_BUCKET_DATALAKE} --cors-configuration file://
↪cors-config.json
```

```
# Copy data lake objects from old bucket to new bucket
gcloud storage cp "gs://${OLD_BUCKET_DATALAKE}/**" gs://${NEW_BUCKET_DATALAKE}/ --
↪recursive
# Copy CORS configuration from old bucket to new bucket
gcloud storage buckets describe gs://${OLD_BUCKET_DATALAKE} --format="json" | jq .
↪cors_config > cors-config.json
gcloud storage buckets update gs://${NEW_BUCKET_DATALAKE} --cors-file=cors-config.json
```

2. Recovery steps

Restore HM-internal databases

After the old backups are available in the new bucket, you can restore the HM-internal databases. These are back-end services used by HM and are required to fully restore the HM instance. Depending on the HM version you are using and on the installation scenario you have deployed, the list of databases may vary.

1. To simplify these process, run following script with your kubeconfig pointing to your new HM installation:

patch-clusters.sh

Script details

This patch script takes care of:

```
- Saving the HM-internal database cluster manifests to YAML files, while generating␣
↪two directories: - One directory `old-cluster-configs`  with the current state of␣
↪the database clusters in the new HM installation (default configuration after␣
↪installation). - Another one called `new-cluster-configs`  with the same files,␣
↪where the script will perform the patches required so the HM-internal databases␣
↪start using the data from the backups.
```

2. Suspend the reconciliation of all HM-internal database clusters, so that you can safely remove the old Custom Resource Definitions (CRDs) of the database clusters without having the operator recreating them by default:

```
for CLUSTER in $(kubectl get clusters.postgresql.k8s.enterprisedb.io -A -o json | jq -
↪rc .items[].metadata | select((.name | test("^p-") | not) and (.name != "stats-
↪collector-db")) | {namespace: .namespace} | uniq)
do
  NAMESPACE=$(echo "${CLUSTER}" | jq -rc .namespace)
  if [[ "${NAMESPACE}" == "upm-system-db" ]]
  then
    COMPONENT="upm-app-db"
  else
    COMPONENT="${NAMESPACE}"
  fi
  edbctl hm suspend-component "${COMPONENT}"
done
```

3. Verify that the components have been suspended correctly:

```
edbctl hm status
```

4. Delete the HM-internal database clusters that were created during installation of the new HM instance to make room for the HM-internal database clusters that will be recovered from the backup:

```
for CONFIG in $(find new-cluster-configs -type f)
do
    kubectl delete -f $CONFIG
done
```

5. Clean the backup area that was created during the installation of the new HM instance to avoid confusion with the old backups that you want to restore:

```
for CONFIG in $(find new-cluster-configs -type f)
do
    NAME=$(yq .metadata.name $CONFIG)
    NAMESPACE=$(yq .metadata.namespace $CONFIG)
    # Try to get PREFIX from cluster config first, fallback to ObjectStore if it fails
    PREFIX=$(yq .spec.backup.barmanObjectStore.destinationPath | downcase $CONFIG 2>/
↪dev/null)
    if [ -z "$PREFIX" ] || [ "$PREFIX" = "null" ]; then
      # destinationPath doesnt exist in the cluster config (after CNPG-I barman␣
↪plugin migration)
      # Get it from the ObjectStore resource instead
      PREFIX=$(kubectl -n $NAMESPACE get objectstores.barmancloud.cnpg.io $NAME -o␣
↪yaml | yq .spec.configuration.destinationPath | downcase)
    fi
    aws s3 rm --recursive ${PREFIX}/${NAME}
done
```

```
for CONFIG in $(find new-cluster-configs -type f)
do
    NAME=$(yq .metadata.name $CONFIG)
    NAMESPACE=$(yq .metadata.namespace $CONFIG)
    # Try to get PREFIX from cluster config first, fallback to ObjectStore if it fails
```

(continues on next page)

(continued from previous page)

```
    PREFIX=$(yq .spec.backup.barmanObjectStore.destinationPath | downcase $CONFIG 2>/
↪dev/null)
    if [ -z "$PREFIX" ] || [ "$PREFIX" = "null" ]; then
      # destinationPath doesnt exist in the cluster config (after CNPG-I barman␣
↪plugin migration)
      # Get it from the ObjectStore resource instead
      PREFIX=$(kubectl -n $NAMESPACE get objectstores.barmancloud.cnpg.io $NAME -o␣
↪yaml | yq .spec.configuration.destinationPath | downcase)
    fi
    gsutil -m rm -r ${PREFIX}/${NAME}
done
```

6. Replace the `hm-portal-bootstrap` secret with the version from backup.

> **Note**
>
> This step is only applicable to versions of HM from *2026.2* and later.

Run the command `kubectl get secret hm-portal-bootstrap -n default` against your new HM instance, and if the secret doesn't exist, then you can skip this step because you are using a version of HM from before `2026.2`.

The `hm-portal-bootstrap` secret on the new HM instance needs to be replaced with the version that was on the old HM instance. This secret contains a Fernet key used for encrypting sensitive configuration data, and the static passwords used for initial bootstrap access. Without the correct Fernet key from the old environment, it isn't possible for HM to decrypt some of the data restored in the next step.

This step involves restoring a single secret found in the `default` namespace. Velero doesn't support this well, as it is typically used to restore all resources of a given type for a given namespace. Therefore, the only way to perform this restoration is to download the entire Velero backup and pull out a specific file.

To select a specific file, you first need to select a backup to use. Start by listing out all the available backups with the following command:

```
velero get backups -o json --selector velero.io/storage-location=recovery \
| jq -rc (["Name", "Timestamp"]), (.items // [.] | .[] | [.metadata.name, .metadata.
↪creationTimestamp]) | @tsv \
| column -t -s "$(printf \t)"
```

Select a backup from this list :typically the latest :and export the following variable pointing to the file's selected name.

```
export BACKUP_NAME=<selected-name>
```

Now download and extract the full backup:

```
mkdir "${BACKUP_NAME}"
velero backup download "${BACKUP_NAME}" -o "${BACKUP_NAME}/backup.tar.gz"
tar -xzf "${BACKUP_NAME}/backup.tar.gz" -C "${BACKUP_NAME}"
```

Confirm that the file `${BACKUP_NAME}/resources/secrets/namespaces/default/hm-portal-bootstrap.json` exists. Now delete the existing `hm-portal-boostrap` secret, and replace it with the one from the backup you just downloaded:

```
kubectl delete secret hm-portal-bootstrap -n default
kubectl apply -f "${BACKUP_NAME}/resources/secrets/namespaces/default/hm-portal-
↪bootstrap.json"
```

Confirm the secret is there using the following command:

```
kubectl get secret hm-portal-bootstrap -n default
```

And you can now remove the local copy of the backup you downloaded, as you will not be needing these files again:

```
rm -rf "${BACKUP_NAME}"
```

7. Apply the YAML file for all the HM-internal database clusters to be re-created with the backup data:

```
for CONFIG in $(find new-cluster-configs -type f)
do
    kubectl apply -f $CONFIG
done
```

You can monitor the restore progress using `kubectl get clusters -A`.

Restart HM services

After all HM-internal database clusters are successfully restored and reporting a healthy state, perform this one-time restart of the management server to refresh the HM console:

```
kubectl delete pods $(kubectl get pods -n upm-beaco-ff-base | grep ^accm-server | awk
↪{print $1}) -n upm-beaco-ff-base
```

Wait for the new pod to reach the **Running** state. At this point, the HM console is available, though it won't yet show your HM-managed Postgres clusters.

Configure the Velero plugin

The Velero plugin handles the transformation of Kubernetes resources during the restore. Most importantly, it ensures Postgres clusters are restored in a state that allows you to manually trigger their data recovery.

1. List the available backups and note the **Name** and **Timestamp** of your preferred recovery point:

```
velero get backups -o json --selector velero.io/storage-location=recovery \
| jq -rc (["Name", "Timestamp"]), (.items // [.] | .[] | [.metadata.name, .metadata.
↪creationTimestamp]) | @tsv \
| column -t -s "$(printf \t)"
```

2. Export the environment variables:

```
export BACKUP_TIMESTAMP=<recovery-date in YYYY-MM-DDTHH:MM:SSZ format>
# The BACKUP_NAME may already be available in your terminal from an earlier step.
export BACKUP_NAME=<selected-name>

# These environment variables should already be available in your terminal:
export OLD_BUCKET=<old-bucket-name>
export NEW_BUCKET=<new-bucket-name>
```

Important

The *BACKUP_TIMESTAMP* must be the exact ISO timestamp found in the previous step.

For example: ***2024-12-16T15:44:03Z . .. ::***

3. Create and apply a *ConfigMap* to configure the Velero plugin:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: velero-plugin-for-edbpgai
  namespace: velero
  labels:
    velero.io/plugin-config: ""
    enterprisedb.io/edbpgai-plugin: RestoreItemAction
data:
  # configure disaster recovery mode, so restored items are transformed as needed
  drMode: "true"
  # configure a date corresponding to the velero backup date. Note the format!
  drDate: "${BACKUP_TIMESTAMP}"
  # old and new buckets for internal custom storage locations
  oldBucket: ${OLD_BUCKET}
  newBucket: ${NEW_BUCKET}
EOF
```

Restore resources

1. Restore **Managed Storage Locations** by applying the following Velero restore. This includes the default `managed-devspatcher` location as well as any additional custom-defined locations.

```
kubectl apply -f - <<EOF
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-1-storagelocations
  namespace: velero
spec:
  backupName: "${BACKUP_NAME}"
  includedResources:
   - storagelocations.biganimal.enterprisedb.com
  includeClusterResources: true
  labelSelector:
    matchLabels:
      biganimal.enterprisedb.io/reserved-by-biganimal: "false"
EOF
```

2. Configure and apply the following Velero restore resource manifest to restore the cluster wrappers:

```
kubectl apply -f - <<EOF
apiVersion: velero.io/v1
kind: Restore
```

(continues on next page)

(continued from previous page)

```
metadata:
  name: restore-2-clusterwrappers
  namespace: velero
spec:
  backupName: "${BACKUP_NAME}"
  includedResources:
   - clusterwrappers.beacon.enterprisedb.com
  restoreStatus:
    includedResources:
     - clusterwrappers.beacon.enterprisedb.com
EOF
```

3. Monitor the restore progress. You must wait until `clusterwrappers` is restored first, because the following custom resources (CR) depend on it. If the corresponding `clusterwrapper` isn't found, HM could delete the other CRs.

```
velero get restore restore-2-clusterwrappers
```

4. After the cluster wrappers are restored, configure and apply the following Velero resource manifest to restore the backup wrappers:

```
kubectl apply -f - <<EOF
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-3-backupwrappers
  namespace: velero
spec:
  backupName: "${BACKUP_NAME}"
  includedResources:
   - backupwrappers.beacon.enterprisedb.com
  restoreStatus:
    includedResources:
     - backupwrappers.beacon.enterprisedb.com
EOF
```

5. Configure and apply the following Velero resource manifest to restore Griptape, Lakekeeper and Dex secrets:

```
kubectl apply -f - <<EOF
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-4-required-secrets
  namespace: velero
spec:
  backupName: "${BACKUP_NAME}"
  includedNamespaces:
  - upm-griptape
  - upm-lakekeeper
  - upm-dex
  includedResources:
  - secrets
  includeClusterResources: false
```

(continues on next page)

(continued from previous page)

```
EOF
```

6. (Optional) If you are running AI workloads, configure and apply the following Velero restore resource manifest to restore kserve resources:

```
kubectl apply -f - <<EOF
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-5-kservecrs
  namespace: velero
spec:
  backupName: "${BACKUP_NAME}"
  includedResources:
  - clusterservingruntimes.serving.kserve.io
  - inferenceservices.serving.kserve.io
EOF
```

7. Monitor all restores and wait for them to be completed:

```
velero get restores
```

3. Restore Postgres clusters

The cluster metadata has been restored, but the HM-managed Postgres clusters must be manually re-provisioned to link back to your data.

1. In the HM console, navigate to the databases section. You will see your original clusters listed with a status of **Deleted**.
2. Select the desired cluster and locate the **Restore** button. Follow the prompts to create a new cluster. During this process, the system will use your previous backups to populate the new instance.
3. After provisioning is complete, verify that the data matches your original state.

You can apply the same procedure to restore any Postgres clusters you had configured on a secondary location.

> **Note**
>
> AI components (such as the GenAI Builder UI in the Launchpad section) will automatically reappear in the HM console once the restore is initiated. Due to the large size of container images and profiles, synchronization may take some time.

The restoration procedure is now complete. To ensure a successful recovery, we recommend checking for data integrity. Log in to the newly provisioned Postgres cluster and run a few test queries to confirm your data is current and accessible.

> **Tip**
>
> If you are performing this as part of a DR drill, internally document the total "Time to Restore" (TTR) for both the database and AI layers to help refine your recovery objectives (RTO). !!!

5.3.4.5 Enabling the Migration Portal AI Copilot

After deploying Hybrid Manager (HM), Migration Portal , a Launchpad tool for migrating external database schemas, works out of the box.

However, the AI Copilot , a chat-bot tool that helps you resolve schema migration compatibility issues within Migration Portal, isn't enabled by default, so you need additional configuration to enable it.

You can configure the AI Copilot using OpenAI, Azure OpenAI, or an alternative AI vendor. Or, you can configure the AI Copilot to work with a self-hosted model you previously created in HM with EDB Agent Factory capabilities. This gives you the flexibility of using ready-to-use models from a third-party vendor or of having complete control over your data and environment.

Fig. 44: Model options for the Migration Portal AI Copilot

5.3.4.5.1 Third-party model (cloud)

Workflow: While using the Migration Portal, you can submit a prompt to the Migration Portal AI Copilot to get information and help on schema conversion. The AI Copilot then communicates with an external, third-party cloud vendor of your choice (for example, OpenAI) over the internet. The third-party vendor uses its own model to process the prompt and generate an answer.

Models: Third-party vendors provide the core AI capabilities, including a chat completion model and an embeddings model. (For Azure OpenAI, you still have to deploy an embedding model.)

Benefits: This approach offers access to highly powerful and continuously updated models without the need for you to manage any AI infrastructure.

Considerations: This option requires uninterrupted internet connectivity. Depending on the license, data may be stored with the vendor. Also, consider costs caused by periodic health check requests that the chat completion and embeddings models need and per-query charges.

5.3.4.5.2 Self-hosted model (on-premises)

Workflow: While using Migration Portal, you can submit a prompt to the Migration Portal AI Copilot to get information and help on schema conversion. The AI Copilot then communicates with models that are hosted locally in the HM environment. (These models were previously created by your organization with EDB Agent Factory capabilities.) The self-hosted models process the prompt and generate an answer.

Models: You create and manage these models using the EDB Agent Factory. They can be models like Llama 3, which is served using a NVIDIA NIM microservice.

Benefits: This approach is better suited for organizations that require a highly secure, air-gapped environment. It ensures that all data processing remains within your infrastructure, meeting strict compliance and privacy requirements.

Considerations: The AI infrastructure is managed by you, which means you need AI expertise in your organization. Also, self-hosted models lack the built-in content filtering and safeguards provided by third-party vendors. The responsibility and potential liability for any unsafe or harmful content generated by the model falls on the user or organization. You also must consider the resource consumption of costly GPU nodes required to run AI workloads.

Enabling a third-party model for the Migration Portal AI Copilot

To leverage a third-party cloud vendor for AI Copilot interactions, these examples use OpenAI and Azure OpenAI.

Prerequisites

Prepare the resources your environment needs to deploy the Migration Portal AI Copilot with your vendor's AI solution.

- You need access credentials to the specific AI vendor's services.
- If you're using OpenAI, you need your organization's ID and API key.
- If you're using Azure OpenAI, you need your organization's endpoint URL and an API key. Also, in your Azure OpenAI instance, create the following deployments: - gpt-4o deployment:
  - Deployment name: `gpt-4o`
  - Suggested version: `2024-11-20` - text-embedding-3-small deployment:
  - Deployment name: `text-embedding-3-small`
  - Suggested version: `1`

Enabling the AI Copilot

You can enable the AI Copilot before or after you install Hybrid Manager (HM).

1. Check if the `edb-migration-copilot` exists:

```
kubectl get namespaces edb-migration-copilot
```

The namespace is created during HM installation. If you're enabling the AI Copilot before installing HM, you must create the namespace in advance.

2. If the `edb-migration-copilot` namespace doesn't exist yet, create it:

```
kubectl create ns edb-migration-copilot
```

3. Set the following environment variables to link the AI Copilot to an existing AI account. Replace the placeholders with your AI vendor's account information:

```
export OPENAI_ORG_ID=<openai_org_id>    # organisation id for the openai account
export OPENAI_API_KEY=<openai_api_key> # api key for accessing the openai account
```

```
export AZURE_OPENAI_ENDPOINT=<azure_openai_endpoint> # endpoint for the azure openai␣
↪instance
export AZURE_OPENAI_API_KEY=<azure_openai_api_key>    # api key for accessing the␣
↪azure openai instance
```

4. Configure your vendor's credentials by adding a secret to the `edb-migration-copilot` namespace:

```
kubectl create secret generic ai-vendor-secrets \
  --namespace=edb-migration-copilot \
  --type=opaque \
  --from-literal=AI_VENDOR=OPENAI \
  --from-literal=OPENAI_ORG_ID="${OPENAI_ORG_ID}" \
  --from-literal=OPENAI_API_KEY="${OPENAI_API_KEY}"
```

```
kubectl create secret generic ai-vendor-secrets \
  --namespace=edb-migration-copilot \
  --type=opaque \
  --from-literal=AI_VENDOR=AZURE_OPENAI \
```

(continues on next page)

(continued from previous page)

```
  --from-literal=AZURE_OPENAI_ENDPOINT="${AZURE_OPENAI_ENDPOINT}" \
  --from-literal=AZURE_OPENAI_API_KEY="${AZURE_OPENAI_API_KEY}"
```

5. Create a file called `migration-portal-values.yaml` with the following Helm value to override the default AI vendor secrets with the secret created in the previous step:

```
parameters:
  edb-migration-copilot:
    ai_vendor_secrets: ai-vendor-secrets
```

```
- The secret referenced in `ai_vendor_secrets`  must match the name of the secret␣
↪previously created.
```

```
parameters:
  edb-migration-copilot:
    ai_vendor_secrets: ai-vendor-secrets
    chat_model: gpt-4o
```

```
- The secret referenced in `ai_vendor_secrets`  must match the name of the secret␣
↪previously created.  - The LLM version specified in `chat_model`  is `gpt-4o`  by␣
↪default as it's the recommended version for the AI Copilot.
```

6. Reference this file when running the `helm upgrade` command during the *Installation and Setup* , so it uses the customized values for Migration Portal. Use this option each subsequent time you invoke `helm upgrade` . Otherwise, your values will be overridden with the vendor's defaults.

```
helm upgrade \
  -n edbpgai-bootstrap \
  --install \
  ...
  -f migration-portal-values.yaml \
  ...
```

Enabling a self-hosted model for the Migration Portal AI Copilot

You can use a self-hosted Agent Factory model to serve the AI Copilot. This example uses NVIDIA NIM to serve the requests and Llama 3 to process it and generate an answer.

- Chat completions: nvidia/llama-3_3-nemotron-super-49b-v1
- Embeddings: nvidia/llama-3.2-nv-embedqa-1b-v2

> **Warning**
>
> There are significant safety implications to consider when using self-hosted models with Migration Portal AI Copilot.

The models provided by third-party vendors like OpenAI amd Azure OpenAI include content filtering and other safeguards that are designed to reduce the risk of the model responding to, generating, or contributing to unsafe content. When you use self-hosted models, these protections aren't present.

In addition, because you're hosting the models, you bear responsibility for the risks and potential liability associated with any unsafe behavior.

Prerequisites

Prepare the resources your environment requires to deploy the Migration Portal AI Copilot with a self-hosted solution:

- You have administrative access to the HM environment.
- Your organization has created a *chat completion and a text embeddings model* with the HM *Getting started with Agent Factory on Hybrid Manager* . They have provided the endpoints for each model, which you can set as environment variables.

```
export COMPLETIONS_SVC=llama-3-3-nemotron-super-49b-v1
export EMBEDDINGS_SVC=llama-3-2-nv-embedqa-1b-v2

export COMPLETIONS_ENDPOINT=$(kubectl get inferenceservice $COMPLETIONS_SVC -o␣
↪jsonpath={.status.url})
export EMBEDDINGS_ENDPOINT=$(kubectl get inferenceservice $EMBEDDINGS_SVC -o␣
↪jsonpath={.status.url})
```

Enabling the AI Copilot

1. Check if the `edb-migration-copilot` namespace exists:

```
kubectl get namespaces edb-migration-copilot
```

The namespace is created during HM installation. If you're enabling the AI Copilot before installing HM, you must create the namespace in advance.

1. If the `edb-migration-copilot` namespace doesn't exist yet, create it:

```
kubectl create ns edb-migration-copilot
```

1. Set the following environment variables to link the secret with the model endpoints:

```
export OPENAI_API_BASE=${COMPLETIONS_ENDPOINT}/v1
export OPENAI_EMBEDDINGS_API_BASE=${EMBEDDINGS_ENDPOINT}/v1
export OPENAI_API_KEY=<openai api key> # set to a placeholder value like `noop` if␣
↪models are deployed in a way that no key is required
```

Note

The AI Copilot uses OpenAI-compatible APIs to communicate with all models, including self-hosted ones. That's why some configuration parameters contain *openai* in their names, even when you're using a different model to serve queries.

1. Create the `ai-vendor-secrets` secret and configure it to point at the models' endpoints:

```
kubectl create secret generic ai-vendor-secrets \
    --namespace=edb-migration-copilot \
    --type=opaque \
    --from-literal=AI_VENDOR=NIM \
    --from-literal=RAGCHEW_OPENAI_API_BASE="${OPENAI_API_BASE}" \
    --from-literal=RAGCHEW_OPENAI_EMBEDDINGS_API_BASE="$    {OPENAI_EMBEDDINGS_API_
↪BASE}" \
    --from-literal=OPENAI_API_KEY="${OPENAI_API_KEY}"
```

1. Create a file called `migration-portal-values.yaml` with the following Helm value to override the default AI vendor secrets with the secret created in the previous step.

```
parameters:
  edb-migration-copilot:
    ai_vendor_secrets: ai-vendor-secrets
```

1. Update the HM installation file to include the AI Copilot configuration. Either update the YAML values you used for installation or run the `helm upgrade` command with the AI Copilot configuration parameters.

Add this configuration block to either the `values.yaml` file you used to install HM with Helm or to the CRD you used to install HM with the operator.

```
[...]
parameters:
  edb-migration-copilot:
    ai_vendor_secrets: ai-vendor-secrets                    # Allows the default AI␣
↪vendor secrets to be overridden by the secret created above. This *must* match the␣
↪name of the above secret.
    chat_model: nvidia/llama-3.3-nemotron-super-49b-v1      # This must match the␣
↪model name as listed by `${COMPLETIONS_ENDPOINT}/v1/models`.
    embeddings_model: nvidia/llama-3.2-nv-embedqa-1b-v2     # This must match the␣
↪model name as listed by `${EMBEDDINGS_ENDPOINT}/v1/models`.
    chat_model_profile: llama3                              # Enables prompt rules␣
↪and other parameters which can help improve the quality of responses from chat␣
↪completion models of the llama3 family.
    embeddings_dimension: "2048"                           # This must match the size␣
↪of the vectors generated by the embeddings model.
    tokenizer: huggingface                                  # Causes tokenizers from␣
↪the Hugging Face Transformers library to be used for token counting.
    tokenizer_model: nvidia/llama-3.3-nemotron-super-49b-v1 # Sets the specific␣
↪tokenizer to use, ensuring it corresponds to a pretrained model on Hugging Face Hub.
    chat_request_max_tokens: "1024"                        # Limits the total number␣
↪of requested tokens to prevent excessively-verbose responses.
    default_similarity_limit: "10"                         # Limits the number of␣
↪contextual chunks included in queries to prevent overwhelming the model with input.
    stream_no_stop_sequence: "true"                        # Prevents OpenAI-specific␣
↪stop sequences being sent to the model. This should always be set when using NIM␣
↪models in order to avoid duplicated final chunks in response streams.
[...]
```

Run the `helm upgrade` command while including the following configuration parameters. Remember to include these parameters each subsequent time you invoke `helm upgrade` . Otherwise, your values will be overridden with defaults.

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    [...]
    --set parameters.edb-migration-copilot.ai_vendor_secrets=ai-vendor-secrets \
    --set parameters.edb-migration-copilot.chat_model=nvidia/llama-3.3-nemotron-super-
↪49b-v1 \
    --set parameters.edb-migration-copilot.embeddings_model=nvidia/llama-3.2-nv-
↪embedqa-1b-v2 \
```

(continues on next page)

(continued from previous page)

```
    --set parameters.edb-migration-copilot.chat_model_profile=llama3 \
    --set parameters.edb-migration-copilot.embeddings_dimension="2048" \
    --set parameters.edb-migration-copilot.tokenizer=huggingface \
    --set parameters.edb-migration-copilot.tokenizer_model=nvidia/llama-3.3-nemotron-
↪super-49b-v1 \
    --set parameters.edb-migration-copilot.chat_request_max_tokens="1024" \
    --set parameters.edb-migration-copilot.default_similarity_limit="10" \
    --set parameters.edb-migration-copilot.stream_no_stop_sequence="true" \
   [...]
```

1. Restart the `edb-migration-copilot` services to trigger a reconciliation of the new values with the system.

```
kubectl rollout restart edb-migration-copilot -n edb-migration-copilot
```

Additional configuration for air-gapped installations

When running in an air-gapped environment, the Migration Portal AI Copilot will fail when it tries to fetch the pretrained tokenizer data from Hugging Face Hub.

Set the following parameter to use a local snapshot of tokenizer data instead:

Include the following `airgapped_mode` parameter to the previously documented configuration block to either the `values.yaml` file you used to install HM with Helm or to the CRD you used to install HM with the operator.

```
[...]
parameters:
  edb-migration-copilot:
    airgapped_mode: "true" # Forces a local snapshot of tokenizer data to be used␣
↪instead of fetching from Hugging Face Hub.
                           # Only works if `tokenizer` and `tokenizer_model` are␣
↪set to `huggingface` and
                           #`nvidia/llama-3.3-nemotron-super-49b-v1` respectively.
[...]
```

Include the following `airgapped_mode` parameter to the previously documented `helm upgrade` command. Remember to include all parameters each subsequent time you invoke `helm upgrade` . Otherwise, your values will be overridden with defaults.

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    [...]
    --set parameters.edb-migration-copilot.airgapped_mode="true" \
    [...]
```

Restart the `edb-migration-copilot` services.

Important

The Migration Portal AI Copilot ships with tokenizer data only for the *nvidia/llama-3.3-nemotron-super-49b-v1* pretrained tokenizer. Using *airgapped_mode: '"true"'* with *tokenizer_model* set to any other model will result in Migration Portal AI Copilot failing.

5.3.4.6 Troubleshooting Hybrid Manager

You can troubleshoot common problems in Hybrid Manager (HM).

5.3.4.6.1 Error codes

Error Code	Category	Error Message	Explanation	Potential solution
na	Installation	[upm-istio] - failed to install components: [upm-istio]	An error from bootstrap: the LoadBalancer service is pending.	Fix the loadbalancer IAM role first, then delete the LB pod to trigger re-creation.
na	Installation	upm-beaco-ff-base install failed		kubectl edit sc gp2
na	Installation	upm-thanos installation failed		
na	Installation	Failed to pull image		
na	Installation	Loki pods crash	The loki pods crashed and the installation failed.	
na	Portal Login	No healthy upstream	"No healthy upstream" is returned after the login.	
401	Portal Login	401 Access Denied	401 error returned when logging into the portal after the installation.	
500	Portal Login	HTTP Error 500	The back ingress gateway can't access incoming connections.	
na	Database Provisioning	Postgres Cluster Provisioning Stuck at X% Complete		
na	Database Provisioning	CPU and Memory range is much smaller than the nodegroup		
na	Database Update	Disk scale up	The new value is shown on cluster yaml but not applied on pvc/pv.	Edit the storageclass.

5.3.4.6.2 Errors in detail

[upm-istio] - failed to install components: [upm-istio]

Detailed error

```
5:03:42AM: ---- waiting on 1 changes [4/5 done] ----
5:03:42AM: ongoing: reconcile service/istio-ingressgateway (v1) namespace: istio-
↪system
5:03:42AM:  ^ Load balancer ingress is empty
{"level":"error","msg":"failed to install component: failed to execute kapp command,␣
```

(continues on next page)

(continued from previous page)

```
↪error: Timed out waiting after 15m0s for resources: [service/istio-ingressgateway␣
↪(v1) namespace: istio-system], message: "}
{"level":"error","msg":"Failed to install components: [upm-istio]"}
{"level":"error","msg":"Installation failed, error: failed to install components:␣
↪[upm-istio]"}
```

```
k get svc -n istio-system
```

```
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          ␣
↪                            AGE
istio-ingressgateway    LoadBalancer   172.20.184.80    <pending>     80:30270/TCP,
↪443:31858/TCP,9443:32692/TCP   3h3m
istiod                  ClusterIP      172.20.146.223   <none>        15010/TCP,15012/
↪TCP,443/TCP,15014/TCP       3h3m
```

```
k describe svc istio-ingressgateway -n istio-system
```

```
Events:
  Type     Reason            Age    From     Message
  ----     ------            ----   ----     -------
  Warning  FailedBuildModel  53m    service  Failed build model due to operation␣
↪error Elastic Load Balancing v2: DescribeLoadBalancers, get identity: get␣
↪credentials: failed to refresh cached credentials, failed to retrieve credentials,␣
↪operation error STS: AssumeRoleWithWebIdentity, https response error StatusCode:␣
↪403, RequestID: 07cc71c0-3113-4b4e-be7f-3de3b475c49b, api error AccessDenied: Not␣
↪authorized to perform sts:AssumeRoleWithWebIdentity
```

Resolution

Fix the loadbalancer IAM role, then delete the LB pod.

```
k delete pod aws-load-balancer-controller-57ccd8bc77-f8lrk -n kube-system
k delete pod aws-load-balancer-controller-57ccd8bc77-j58c2 -n kube-system
k get svc -n istio-system
```

```
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP                    ␣
↪                                                    PORT(S)                        ␣
↪         AGE
istio-ingressgateway    LoadBalancer   172.20.184.80    k8s-istiosys-istioing-
↪38d88046aa-37454e78c9648556.elb.us-east-1.amazonaws.com   80:30270/TCP,443:31858/
↪TCP,9443:32692/TCP   3h8m
istiod                  ClusterIP      172.20.146.223   <none>                         ␣
↪                                                    15010/TCP,15012/TCP,443/TCP,15014/
↪TCP       3h8m
```

upm-beaco-ff-base install failed

Detailed error

```
NAME                      READY   STATUS    RESTARTS   AGE
app-db-1-initdb-k4nbl     0/1     Pending   0          35m
```

```
kc get pvc -n upm-beaco-ff-base
```

```
NAME       STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS  ␣
↪VOLUMEATTRIBUTESCLASS   AGE
app-db-1   Pending                                      <unset>       ␣
↪          35m
```

```
kc describe pvc app-db-1 -n upm-beaco-ff-base
```

```
Name:          app-db-1
Namespace:     upm-beaco-ff-base
StorageClass:
Status:        Pending
Volume:
.......
Events:
  Type     Reason         Age                  From                         Message
  ----     ------         ----                 ----                         -------
  Normal   FailedBinding  64s (x142 over 36m)  persistentvolume-controller  no␣
↪persistent volumes available for this claim and no storage class is se
```

Resolution

Run `kubectl edit sc gp2` and add the following annotation: `storageclass.kubernetes.io/is-default-class: "true"`

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
```

Upm-thanos installation failed

Detailed error

Resolution

1. Check for any pods in crash loop.
2. Check the logs to look for any permission-related errors.

```
kubectl get pod -n monitoring
```

Check if the secret is correct:

```
kubectl get secret -n monitoring
```

The secret must match what you defined in `edb-object-storage` :

```
kubectl get secret -n monitoring  thanos-objstore-secret -o yaml
```

```
 apiVersion: v1
data:
  objstore.yml:␣
↪dHlwZTogUzMKY29uZmlnOgogIGJ1Y2tldDogYXBwbGlhbmNlLXNoYW50ZXN0LWVrcy1jbHVzdGVyLWVkYi1wb3N0Z3JlcwogIHs

~ ⍰  echo␣
↪dHlwZTogUzMKY29uZmlnOgogIGJ1Y2tldDogYXBwbGlhbmNlLXNoYW50ZXN0LWVrcy1jbHVzdGVyLWVkYi1wb3N0Z3JlcwogIHs
↪| base64 -d
```

To correct the secret:

```
kubectl delete secret thanos-objstore-secret -n monitoring

kubectl delete pod storage-location-operator-controller-manager-xxxxxx -n storage-
↪location-operator
```

Rerun helm upgrade and ensure the secret `thanos-objstore-secret` is correct:

```
kubectl delete pod <pod in crashloop> -n monitoring
```

Failed to pull image

Detailed error

```
NAME                                                READY   STATUS                  ␣
↪RESTARTS   AGE
edbpgai-bootstrap-job-v1.0.6-appl-d12cvm-q2pbg   0/1     Init:AssetPullBackOff   0   ␣
↪      2m49s

Events:
  Type     Reason     Age               From               Message
  ----     ------     ----              ----               -------
  Normal   Scheduled  53s               default-scheduler  Successfully assigned␣
↪edbpgai-bootstrap/edbpgai-bootstrap-job-v1.0.6-appl-d12cvm-q2pbg to i-
↪001f23a316e13e905
  Warning  Failed     22s               kubelet            Failed to pull image
↪"docker.enterprisedb.com/staging_pgai-platform/edbpgai-bootstrap/bootstrap-eks:v1.0.
↪6-appl": rpc error: code = DeadlineExceeded desc = failed to pull and unpack image
↪"docker.enterprisedb.com/staging_pgai-platform/edbpgai-bootstrap/bootstrap-eks:v1.0.
↪6-appl": failed to resolve reference "docker.enterprisedb.com/staging_pgai-platform/
↪edbpgai-bootstrap/bootstrap-eks:v1.0.6-appl": failed to do request: Head "https://
↪docker.enterprisedb.com/v2/staging_pgai-platform/edbpgai-bootstrap/bootstrap-eks/
↪manifests/v1.0.6-appl": dial tcp 18.67.76.32:443: i/o timeout
  Warning  Failed     22s               kubelet            Error: ErrAssetPull
  Normal   BackOff    22s               kubelet            Back-off pulling image
↪"docker.enterprisedb.com/staging_pgai-platform/edbpgai-bootstrap/bootstrap-eks:v1.0.
```

(continues on next page)

(continued from previous page)

```
↪6-appl"
  Warning  Failed     22s                kubelet            Error: AssetPullBackOff
  Normal   Pulling    7s (x2 over 52s)   kubelet            Pulling image "docker.
↪enterprisedb.com/staging_pgai-platform/edbpgai-bootstrap/bootstrap-eks:v1.0.6-appl"
```

Resolution

1. In your AWS console, edit your related subnet and enable **Auto-assign public IPv4 address**.
2. Delete nodes to let EKS pick up the change:

```
k get node
```

```
NAME                  STATUS   ROLES    AGE   VERSION
i-001f23a316e13e905   Ready    <none>   19h   v1.31.4-eks-0f56d01
i-012fd44b2f575a514   Ready    <none>   20h   v1.31.4-eks-0f56d01
i-076d6a447c6551a21   Ready    <none>   20h   v1.31.4-eks-0f56d01
```

```
k delete node i-001f23a316e13e905 i-012fd44b2f575a514 i-076d6a447c6551a21
```

```
node "i-001f23a316e13e905" deleted
node "i-012fd44b2f575a514" deleted
node "i-076d6a447c6551a21" deleted
```

```
kgp -n edbpgai-bootstrap
```

```
NAME                                                READY   STATUS    RESTARTS   AGE
edbpgai-bootstrap-job-v1.0.6-appl-5ee7l7-xzfqt   1/1     Running   0          118s
```

Loki pods crash

Detailed error

```
k get pods -n logging
```

```
NAME                           READY   STATUS             RESTARTS          AGE
loki-backend-0                 1/2     CrashLoopBackOff   16 (2m1s ago)     59m
loki-read-5b8555fb64-9srmq     0/1     CrashLoopBackOff   16 (2m40s ago)    59m
loki-read-5b8555fb64-fw6ls     0/1     CrashLoopBackOff   16 (2m7s ago)     59m
loki-read-5b8555fb64-nw2vq     0/1     CrashLoopBackOff   16 (2m17s ago)    59m
loki-write-0                   0/1     CrashLoopBackOff   16 (2m37s ago)    59m
loki-write-1                   0/1     CrashLoopBackOff   16 (2m5s ago)     59m
loki-write-2                   0/1     CrashLoopBackOff   16 (2m16s ago)    59m
```

```
k logs loki-write-0 -n logging
```

```
level=info ts=2025-02-26T09:32:35.173785968Z caller=main.go:126 msg="Starting Loki"␣
↪version="(version=release-3.1.x-89fe788, branch=release-3.1.x, revision=89fe788d)"
level=info ts=2025-02-26T09:32:35.173821129Z caller=main.go:127 msg="Loading␣
```

(continues on next page)

(continued from previous page)

```
↪configuration file" filename=/etc/loki/config/config.yaml
level=info ts=2025-02-26T09:32:35.174421237Z caller=server.go:352 msg="server␣
↪listening on addresses" http=[::]:3100 grpc=[::]:9095
level=info ts=2025-02-26T09:32:35.175391511Z caller=memberlist_client.go:435 msg=
↪"Using memberlist cluster label and node name" cluster_label= node=loki-write-0-
↪a0cfbc5d
level=info ts=2025-02-26T09:32:35.17599149Z caller=shipper.go:160 index-store=tsdb-
↪2020-05-15 msg="starting index shipper in WO mode"
level=info ts=2025-02-26T09:32:35.176072171Z caller=table_manager.go:136 index-
↪store=tsdb-2020-05-15 msg="uploading tables"
level=info ts=2025-02-26T09:32:35.176269663Z caller=head_manager.go:308 index-
↪store=tsdb-2020-05-15 component=tsdb-head-manager msg="loaded wals by period"␣
↪groups=0
level=info ts=2025-02-26T09:32:35.176305024Z caller=manager.go:86 index-store=tsdb-
↪2020-05-15 component=tsdb-manager msg="loaded leftover local indices" err=null␣
↪successful=true buckets=0 indices=0 failures=0
level=info ts=2025-02-26T09:32:35.176325994Z caller=head_manager.go:308 index-
↪store=tsdb-2020-05-15 component=tsdb-head-manager msg="loaded wals by period"␣
↪groups=1
level=info ts=2025-02-26T09:32:35.180393332Z caller=module_service.go:82 msg=starting␣
↪module=server
level=info ts=2025-02-26T09:32:35.180477643Z caller=module_service.go:82 msg=starting␣
↪module=memberlist-kv
level=error ts=2025-02-26T09:32:35.180526964Z caller=loki.go:524 msg="module failed"␣
↪module=memberlist-kv error="starting module memberlist-kv: invalid service state:␣
↪Failed, expected: Running, failure: service memberlist_kv failed: failed to create␣
↪memberlist: Failed to get final advertise address: no private IP address found, and␣
↪explicit IP not provided"
level=error ts=2025-02-26T09:32:35.180549544Z caller=loki.go:524 msg="module failed"␣
↪module=ring error="failed to start ring, because it depends on module memberlist-kv,
↪ which has failed: invalid service state: Failed, expected: Running, failure:␣
↪starting module memberlist-kv: invalid service state: Failed, expected: Running,␣
↪failure: service memberlist_kv failed: failed to create memberlist: Failed to get␣
↪final advertise address: no private IP address found, and explicit IP not provided"
level=error ts=2025-02-26T09:32:35.180582375Z caller=loki.go:524 msg="module failed"␣
↪module=store error="failed to start store, because it depends on module memberlist-
↪kv, which has failed: invalid service state: Failed, expected: Running, failure:␣
↪starting module memberlist-kv: invalid service state: Failed, expected: Running,␣
↪failure: service memberlist_kv failed: failed to create memberlist: Failed to get␣
↪final advertise address: no private IP address found, and explicit IP not provided"
level=error ts=2025-02-26T09:32:35.180588655Z caller=loki.go:524 msg="module failed"␣
↪module=ingester error="failed to start ingester, because it depends on module␣
↪analytics, which has failed: context canceled"
level=error ts=2025-02-26T09:32:35.180593435Z caller=loki.go:524 msg="module failed"␣
↪module=distributor error="failed to start distributor, because it depends on module␣
↪analytics, which has failed: context canceled"
level=error ts=2025-02-26T09:32:35.180597415Z caller=loki.go:524 msg="module failed"␣
↪module=analytics error="failed to start analytics, because it depends on module␣
↪memberlist-kv, which has failed: invalid service state: Failed, expected: Running,␣
↪failure: starting module memberlist-kv: invalid service state: Failed, expected:␣
↪Running, failure: service memberlist_kv failed: failed to create memberlist: Failed␣
↪to get final advertise address: no private IP address found, and explicit IP not␣
```

(continues on next page)

(continued from previous page)

```
↪provided"
level=info ts=2025-02-26T09:32:35.180687877Z caller=modules.go:1832 msg="server␣
↪stopped"
level=info ts=2025-02-26T09:32:35.180702097Z caller=module_service.go:120 msg="module␣
↪stopped" module=server
level=info ts=2025-02-26T09:32:35.180708417Z caller=loki.go:508 msg="Loki stopped"␣
↪running_time=85.855531ms
failed services
github.com/grafana/loki/v3/pkg/loki.(*Loki).Run
    /src/loki/pkg/loki/loki.go:566
main.main
    /src/loki/cmd/loki/main.go:129
runtime.main
    /usr/local/go/src/runtime/proc.go:271
runtime.goexit
    /usr/local/go/src/runtime/asm_amd64.s:1695
level=error ts=2025-02-26T09:32:35.180772078Z caller=log.go:216 msg="error running␣
↪loki" err="failed services\ngithub.com/grafana/loki/v3/pkg/loki.(*Loki).Run\n\t/src/
↪loki/pkg/loki/loki.go:566\nmain.main\n\t/src/loki/cmd/loki/main.go:129\nruntime.
↪main\n\t/usr/local/go/src/runtime/proc.go:271\nruntime.goexit\n\t/usr/local/go/src/
↪runtime/asm_amd64.s:1695"
```

Resolution

There is a known issue from Loki due to the CIDR not 10.*

1. Check the IPv4 CIDR in your VPC. For example, if it's set to 11.1.0.0/20, set it to 10.1.0.0/20.

No healthy upstream

Detailed error

The pod at `upm-dex` is in a crash loop. Check the pod logs to view the error details. In this case, `pgai.portal.authentication.staticPasswords` isn't configured correctly, so update the correct configuration and rerun the deployment:

```
upm-dex                       upm-dex-66b96b5857-pdcg2                                  ␣
↪       1/2     CrashLoopBackOff   206 (3m34s ago)   17h

kubectl logs upm-dex-66b96b5857-pdcg2 -n upm-dex
error parse config file /tmp/dex.config.yaml-565347154: error unmarshaling JSON:␣
↪malformed bcrypt hash: crypto/bcrypt: hashedSecret too short to be a bcrypted␣
↪password
```

Resolution

Added the 4 parameter on the upgrade command:

```
helm upgrade \
- n edbpgai-bootstrap \
- -install \
- -set parameters.global.portal_domain_name="${PORTAL_DOMAIN_NAME}" \
```

(continues on next page)

(continued from previous page)

```
- -set parameters.global.dms_domain_name="$DMS_DOMAIN_NAME" \
- -set parameters.global.load_balancer_provider="$LOAD_BALANCER_PROVIDER" \
- -set parameters.global.load_balancer_mode="$LOAD_BALANCER_MODE" \
- -set parameters.global.node_port_domain="$NODE_PORT_DOMAIN" \
- -set parameters.upm-beacon.server_host="${BEACON_SERVICE_DOMAIN_NAME}" \
- -set parameters.upm-beaco-ff-base.cookie_aeskey="${AES_256_KEY}" \
- -set system="eks" \
- -set remoteContainerRegistryURL="${REGISTRY_PACKAGE_URL}" \
- -set internalContainerRegistryURL="${REGISTRY_PACKAGE_URL}" \
- -set bootstrapAsset="${REGISTRY_PACKAGE_URL}/edbpgai-bootstrap/bootstrap-eks:$
↪{EDBPGAI_BOOTSTRAP_IMAGE_VERSION}" \
- -set pgai.portal.authentication.staticPasswords[0].email="owner@mycompany.com" \
- -set pgai.portal.authentication.staticPasswords[0].hash=$2y$10
↪$STTzuGJa3PydqHvlzi2z6OgDU1G/JLTqiuGblH.RemIutWxkztN5m \
- -set pgai.portal.authentication.staticPasswords[0].username="owner@mycompany.com" \
- -set pgai.portal.authentication.staticPasswords[0].userID="c5998173-a605-449a-a9a5-
↪4a9c33e26df7" \
- -version "${EDBPGAI_BOOTSTRAP_HELM_CHART_VERSION}" \
edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap
```

401 Access Denied

Detailed error

Resolution

Check the `config.yaml` file. The `staticPasswords` section isn't set properly in this case. Rerun `helm upgrade` with the correct values.

```
k get secret upm-dex -n upm-dex -o yaml
```

```
apiVersion: v1
data:
  config.yaml:␣
↪aXNzdWVyOiBodHRwczovL3BvcnRhbC5iYXN1cHBvcnQub3JnL2F1dGgKc3RvcmFnZToKICB0eXBlOiBwb3N0Z3JlcwogIGNvbm
echo "xxx" | base64 -d
staticPasswords:
  - email: owner@mycompany.com
    hash: XXX
    userID: c5998173-a605-449a-a9a5-4a9c33e26df7
    username: owner@mycompany.com
```

As an alternative, use this one-line command:

```
k get secret -n upm-dex upm-dex -ojsonpath={.data.config\.yaml} | base64 -d
```

HTTP Error 500

Detailed error

The back ingress gateway can't access incoming connections. If you're accessing the webpage from a public network, it must be through port 443 since https is the only allowed protocol.

Resolution

Check the status of the `ingressgateway` . Tne `ingressgateway` is an `istio-ingressgateway` .

```
k get all -n istio-system
```

```
NAME                                        READY   STATUS    RESTARTS   AGE
pod/istio-ingressgateway-7956f6d57c-jd8jm   1/1     Running   0          14m
pod/istio-ingressgateway-7956f6d57c-jmmv8   1/1     Running   0          14m
pod/istio-ingressgateway-7956f6d57c-r65tk   1/1     Running   0          14m
pod/istiod-7f7bbcdcbb-fzggv                 1/1     Running   0          3h26m
pod/istiod-7f7bbcdcbb-mxjpz                 1/1     Running   0          3h26m

NAME                           TYPE           CLUSTER-IP       EXTERNAL-IP
↪                                                              PORT(S)
↪                  AGE
service/istio-ingressgateway   LoadBalancer   172.20.92.132    k8s-istiosys-istioing-
↪987da680df-8452f790b4602f2c.elb.us-east-1.amazonaws.com   80:30950/TCP,443:32604/
↪TCP,9443:31744/TCP   3h26m
service/istiod                 ClusterIP      172.20.112.158   <none>
↪                                                              15010/TCP,15012/TCP,443/
↪TCP,15014/TCP       3h26m

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/istio-ingressgateway    3/3     3            3           3h26m
deployment.apps/istiod                  2/2     2            2           3h26m

NAME                                               DESIRED   CURRENT   READY   AGE
replicaset.apps/istio-ingressgateway-68fcd46fff    0         0         0       84m
replicaset.apps/istio-ingressgateway-7956f6d57c    3         3         3       14m
replicaset.apps/istio-ingressgateway-866d9bd74b    0         0         0       3h26m
replicaset.apps/istiod-7f7bbcdcbb                  2         2         2       3h26m
```

Check the gateway configuration for configuration errors. The gateway rule is already set to listen on 443 ports:

```
k get gateway/upm-portal -n istio-system -oyaml
```

```
apiVersion: networking.istio.io/v1
kind: Gateway
….
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - portal.basupport.org
    port:
      name: http
```

(continues on next page)

(continued from previous page)

```
      number: 80
      protocol: HTTP
    tls:
      httpsRedirect: true
  - hosts:
    - portal.basupport.org
    port:
      name: https-443
      number: 443
      protocol: HTTPS
  …
```

Then check the deployment logs:

```
2025-02-20T03:42:32.203112Z     warning envoy main external/envoy/source/server/
↪server.cc:835   Usage of the deprecated runtime key overload.global_downstream_max_
↪connections, consider switching to `envoy.resource_monitors.downstream_connections`␣
↪instead.This runtime key will be removed in future.      thread=15
2025-02-20T03:42:32.203389Z     warning envoy main external/envoy/source/server/
↪server.cc:928   There is no configured limit to the number of allowed active␣
↪downstream connections. Configure a limit in `envoy.resource_monitors.downstream_
↪connections` resource monitor.   thread=15
2025-02-20T03:42:32.211729Z     info    xdsproxy        connected to delta upstream␣
↪XDS server: istiod.istio-system.svc:15012   id=1
2025-02-20T03:42:32.268369Z     info    ads     ADS: new connection for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system-1
2025-02-20T03:42:32.268437Z     info    cache   returned workload certificate from␣
↪cache        ttl=23h59m59.731565472s
2025-02-20T03:42:32.268890Z     info    ads     SDS: PUSH request for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system resources:1 size:4.0kB resource:default
2025-02-20T03:42:32.279500Z     info    ads     ADS: new connection for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system-2
2025-02-20T03:42:32.279654Z     info    cache   returned workload certificate from␣
↪cache        ttl=23h59m59.72034871s
2025-02-20T03:42:32.279711Z     info    ads     SDS: PUSH request for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system resources:1 size:4.0kB resource:default
2025-02-20T03:42:32.280287Z     info    ads     ADS: new connection for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system-3
2025-02-20T03:42:32.280372Z     info    cache   returned workload trust anchor from␣
↪cache       ttl=23h59m59.719629366s
2025-02-20T03:42:32.280503Z     info    ads     SDS: PUSH request for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system resources:1 size:1.1kB resource:ROOTCA
2025-02-20T03:42:32.320657Z     info    wasm    fetching image staging_pgai-platform/
↪upm-beaco-filters/upm-oidc from registry docker.enterprisedb.com with tag v1.1.44
2025-02-20T03:42:32.323034Z     info    wasm    fetching image staging_pgai-platform/
↪upm-beaco-filters/upm-error-transformer from registry docker.enterprisedb.com with␣
↪tag v1.1.44
2025-02-20T03:42:32.323073Z     info    wasm    fetching image staging_pgai-platform/
↪upm-beaco-filters/upm-authz-checker from registry docker.enterprisedb.com with tag␣
↪v1.1.44
2025-02-20T03:42:35.074684Z     warning envoy wasm external/envoy/source/extensions/
↪common/wasm/context.cc:1198 wasm log: error parsing plugin configuration: Error(
```

(continues on next page)

(continued from previous page)

```
↪"aes_key: Invalid Length, got 3 bytes, expected 32", line: 1, column: 12)  ␣
↪thread=15
2025-02-20T03:42:35.074712Z     error   envoy wasm external/envoy/source/extensions/
↪common/wasm/wasm.cc:110     Wasm VM failed Failed to configure base Wasm plugin    ␣
↪thread=15
2025-02-20T03:42:35.075923Z     critical        envoy wasm external/envoy/source/
↪extensions/common/wasm/wasm.cc:474     Plugin configured to fail closed failed to␣
↪load thread=15
2025-02-20T03:42:35.076559Z     warning envoy config external/envoy/source/extensions/
↪config_subscription/grpc/delta_subscription_state.cc:269  delta config for type.
↪googleapis.com/envoy.config.core.v3.TypedExtensionConfig rejected: Unable to create␣
↪Wasm HTTP filter istio-system.upm-oidc        thread=15
2025-02-20T03:42:35.076572Z     warning envoy config external/envoy/source/extensions/
↪config_subscription/grpc/grpc_subscription_impl.cc:138    gRPC config for type.
↪googleapis.com/envoy.config.core.v3.TypedExtensionConfig rejected: Unable to create␣
↪Wasm HTTP filter istio-system.upm-oidc thread=15
2025-02-20T03:42:35.076577Z     warning envoy config external/envoy/source/extensions/
↪config_subscription/grpc/grpc_subscription_impl.cc:138    gRPC config for type.
↪googleapis.com/envoy.config.core.v3.TypedExtensionConfig rejected: Unable to create␣
↪Wasm HTTP filter istio-system.upm-oidc thread=15
2025-02-20T03:42:35.076584Z     warning envoy config external/envoy/source/extensions/
↪config_subscription/grpc/grpc_subscription_impl.cc:138    gRPC config for type.
↪googleapis.com/envoy.config.core.v3.TypedExtensionConfig rejected: Unable to create␣
↪Wasm HTTP filter istio-system.upm-oidc thread=15
2025-02-20T03:42:35.189419Z     info    Readiness succeeded in 3.054667763s
2025-02-20T03:42:35.189690Z     info    Envoy proxy is ready
```

The error here was because the `wasm plugin` wasn't bootstrapped successfully, so the `ingressgateway` isn't really ready:

```
k get pod -listio=ingressgateway -n istio-system
```

```
Enter MFA code for arn:aws:iam::121858151946:mfa/zane.zhou@enterprisedb.com:
NAME                                    READY   STATUS    RESTARTS   AGE
istio-ingressgateway-7956f6d57c-jd8jm   1/1     Running   0          72m
istio-ingressgateway-7956f6d57c-jmmv8   1/1     Running   0          72m
istio-ingressgateway-7956f6d57c-r65tk   1/1     Running   0          72m
```

But in the pod, you can see the readiness check actually failed:

```
k describe pod/istio-ingressgateway-7956f6d57c-jd8jm -n istio-system
```

```
Warning  Unhealthy  20m (x2 over 20m)  kubelet            Readiness probe failed: Get
↪"http://10.0.30.134:15021/healthz/ready": dial tcp 10.0.30.134:15021: connect:␣
↪connection refused
```

This issue is caused by a misconfiguration of `wasm plugin` . The `aes_key` isn't correctly set. Fix the configuration and redeploy the `ingressgateway` :

```
k edit WasmPlugin/upm-oidc
```

Then set the following `aes_key` to the default value (or a custom-generated value in a real production environment):

```
pluginConfig:
  aes_key: rzkutHl8NJNztPMEJYykZouHslNiA7xmIXH+58ISUVo=
```

Then restart the deployment:

```
k rollout restart deployment/istio-ingressgateway -n istio-system
```

Postgres Cluster Provisioning Stuck at X% Complete

Detailed error

A single-node cluster creation stuck at 87% completion.

Resolution

The CPU and memory range is much smaller than the nodegroup

Detailed error

The nodegroup is m5.4xlarge, which is 16vCPU and 64GB RAM with no node. At cluster creation, the range shown in the console are CPU (0-3.92 Cores) and Memory (0-14.3 Gi)

Resolution

This is due to 0 size nodepool. If the nodepool has node, it's fine. But when the problem occurs and then you create the node, it doesn't work. The console still shows the same value after the node created.

Pending: https://enterprisedb.atlassian.net/browse/UPM-45883

Disk scale up

Detailed error

Scale disk on the console. The new value is shown on cluster yaml but not applied on pvc/pv.

```
kubectl get cluster p-2hgee2782y -o yaml
```

```
storage:
  resizeInUseVolumes: true
  size: 5Gi
```

```
k get pv
```

```
pvc-4a35b3ba-507e-4d4c-9017-360326e57b9d   2Gi    RWO            Delete          ␣
↪Bound    p-2hgee2782y/p-2hgee2782y-1
```

Resolution

CNP operator logs contains the error:

```
kubectl logs postgresql-operator-controller-manager-7cc67597df-4ggdf -n postgresql-
↪operator-system
```

```
{
  "level": "error",
  "ts": "2025-01-17T10:02:35.739693483Z",
  "msg": "Reconciler error",
  "controller": "cluster",
  "controllerGroup": "postgresql.k8s.enterprisedb.io",
  "controllerKind": "Cluster",
  "Cluster": { "name": "p-2hgee2782y", "namespace": "p-2hgee2782y" },
  "namespace": "p-2hgee2782y",
  "name": "p-2hgee2782y",
  "reconcileID": "2de2ade7-2bf7-45f4-a2af-3f135a4f7ad9",
  "error": persistentvolumeclaims "p-2hgee2782y-1" is forbidden: only dynamically␣
↪provisioned pvc can be resized and the storageclass that provisions the pvc must␣
↪support resize,
  "stacktrace": "sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller[.
↪..]).reconcileHandler\n\tcloud-native-postgres/vendor/sigs.k8s.io/controller-
↪runtime/pkg/internal/controller/controller.go:316\nsigs.k8s.io/controller-runtime/
↪pkg/internal/controller.(*Controller[...]).processNextWorkItem\n\tcloud-native-
↪postgres/vendor/sigs.k8s.io/controller-runtime/pkg/internal/controller/controller.
↪go:263\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller[...]).
↪Start.func2.2\n\tcloud-native-postgres/vendor/sigs.k8s.io/controller-runtime/pkg/
↪internal/controller/controller.go:224",
}
```


Check and edit `storageclass` to `allowVolumeExpansion: true`.

```
kubecl edit storageclass gp2
```

```
k get storageclass gp2 -o yaml
```

```
allowVolumeExpansion: true
```

Delete the CNP operator pod to apply the change:

```
kubectl  delete pod postgresql-operator-controller-manager-7cc67597df-tgl7b -n␣
↪postgresql-operator-system
```

```
kubectl get pv
```

```
pvc-4a35b3ba-507e-4d4c-9017-360326e57b9d   5Gi        RWO            Delete           ␣
↪Bound    p-2hgee2782y/p-2hgee2782y-1
```

stats-collector-db disk issues

In most cases, if you're having disk issues with `stats-collector-db`, then you are getting either:

1. An alert in Alert Manager saying your disk is close to capacity. This happens if your cluster is still running as the disk isn't yet full.
2. An alert in Alert Manager saying a pod is in a crash loop. This happens if your cluster is crashed due to full disk. You also can't get relevant disk stats in your cluster's **Monitoring** tab.

Scenario 1: High disk usage while the cluster is still running

If the cluster is operating normally but you notice high disk usage, you have two options.

Solution A: Reduce data retention period

Reducing the data retention period is a quick way to lower disk usage by storing less data. The default retention period is 168 hours (7 days). Using this solution, you reduce the retention period to 96 hours (4 days).

Reducing the retention period frees up approximately 42% more available disk. To reduce the retention rate:

1. Edit the `ConfigMap` using the following command from a terminal with `kubectl` access to the platform:

```
kubectl -n upm-api-stats-collector edit cmd
upm-api-stats-collector-configmap-<your-config-map-string>
```

1. In the `ConfigMap` , find the `retention-period` parameter and change its value from `168h` to `96h` .
2. Restart `upm-api-stats-collector` .

```
kubectl rollout restart deploy/upm-api-stats-collector -n upm-api-stats-collector
```

After completing these steps, DB usage will stay within the stable range and the alert in Alert Manager will disappear.

Solution B: Increase disk volume size

If you need to keep 7 days of data, you must increase the disk's storage capacity to solve the issue.

1. Edit the cluster configuration.

```
kubectl edit cluster stats-collector-db -n upm-api-stats-collector
```

1. Locate the `stats-collector-db` specification and increase the value of `.spec.storage.size` to the preferred size.
2. Restart `upm-api-stats-collector` .

```
kubectl rollout restart deploy/upm-api-stats-collector -n upm-api-stats-collector
```

Scenario 2: Disk is full and cluster is crashing

If the disk is full, it can cause the cluster to get stuck in a crash loop, for example:

```
upm-api-stats-collector      stats-collector-db-1 0/1     CrashLoopBackOff   197␣
↪(4m34s ago)   4d7h
```

You have two options for addressing this issue.

Solution A: Re-create the database

If you don't need to keep the historical stats data:

1. Delete the `stats-collector-db` cluster:

```
kubectl delete cluster stats-collector-db -n upm-api-stats-collector
```

1. Delete the `stats-collector-db` backup folder from your S3 bucket:

```
S3/edb-internal-backups/<your-S3-string>/databases/transporter-db/
```

1. Run `helm upgrade` to re-create an empty database with the default settings:

```
helm upgrade -n edbpgai-bootstrap \
- -install -f ./values.yaml \
- -version "${EDB_PLATFORM_VERSION}" \
edbpgai-bootstrap "${HCP_HELM_REPO_NAME}/edbpgai-bootstrap
```

1. Restart `upm-api-stats-collector` :

```
kubectl rollout restart deploy/upm-api-stats-collector -n upm-api-stats-collector
```

Solution B: Increase disk volume size

If you need to preserve the existing data, you must increase the disk's volume size.

1. Edit the cluster configuration:

```
kubectl edit cluster stats-collector-db -n upm-api-stats-collector
```

1. Locate the `stats-collector-db` specification and increase the value of `.spec.storage.size` .
2. Delete the pod in the crashing loop.
3. Restart `ump-api-stats-collector` :

```
kubectl rollout restart deploy/upm-api-stats-collector -n upm-api-stats-collector
```

After the restart, the database has a larger volume, which resolves the crash loop.

> **Warning**
>
> Increasing the volume size above the default blocks patch version upgrades for clusters running version *1.2.x* .

If you increased the disk volume size but now need to perform a minor version upgrade on HM v1.2.x, then you must first resize the database back to the default size (50GB). Contact the Support team for assistance with database resizing. !!!

5.3.4.7 Using query recommendations in Hybrid Manager

5.3.4.8 Using query recommendations in Hybrid Manager

Query recommendations in Hybrid Manager help you continuously improve the performance of your database workloads.

By correlating detailed query diagnostics with actionable recommendations, Hybrid Manager enables you to:

- Identify which queries are slowing down your systems.
- Understand how to optimize those queries.
- Take action to reduce query latency and resource consumption.

5.3.4.8.1 Who uses query recommendations?

Query recommendations are valuable for:

- **DBAs and platform engineers** tuning database performance.
- **Data engineers** optimizing data pipelines and batch jobs.
- **Developers** diagnosing application performance bottlenecks.
- **SREs and observability teams** monitoring database health across environments.

5.3.4.8.2 What are query recommendations?

Hybrid Manager collects in-depth query diagnostics and analyzes them. The platform identifies opportunities to optimize your workload, including:

- **Index recommendations** to improve data retrieval performance.
- **Statistics recommendations** to improve query planning and execution accuracy.

5.3.4.8.3 Why use query recommendations?

Query recommendations help you:

- Improve query performance with minimal manual tuning.
- Focus tuning efforts on queries that will provide the highest impact.
- Continuously adapt to evolving workloads — as query patterns and data volumes change, so do your optimization needs.
- Reduce database load, infrastructure cost, and user-facing latency.
- Proactively manage performance rather than reacting to production issues.
- Build a structured tuning workflow backed by data-driven insights.

5.3.4.8.4 When to use query recommendations

Use query recommendations as part of your regular tuning and performance management process:

- During **initial onboarding** of new applications.
- When **migrating databases** to new platforms or versions.
- After significant **schema or application logic changes**.
- When monitoring for **performance regressions** or **capacity trends**.
- As part of **ongoing database health reviews** — query patterns evolve even without code changes.
- During **release validation** — check whether new workload patterns have introduced tuning opportunities.

5.3.4.8.5 How query recommendations work

Hybrid Manager collects query diagnostics data, including:

- Query latency and execution patterns.
- Wait state analysis
- Workload trends and query frequency.
- The **Query Diagnostics** tab — where recommendations are shown inline with individual queries.

- The **Recommendations** tab — where all current recommendations for the selected database or cluster are listed.

You can move seamlessly from diagnosing a problematic query to understanding and applying the relevant recommendation.

5.3.4.8.6 Integrating query recommendations into your tuning process

You can incorporate Hybrid Manager recommendations into your broader tuning workflow:

- **Daily or weekly reviews:** Regularly check the Recommendations tab to identify new opportunities.
- **Targeted tuning:** Focus on high-latency or high-frequency queries flagged in Query Diagnostics. Start by applying high-impact recommendations (such as missing indexes).
- **Trend analysis:** Monitor how query performance evolves after applying recommendations. Hybrid Manager helps you track whether tuning efforts are effective.
- **Cross-team collaboration:** Developers, DBAs, and platform engineers can use the same recommendations to prioritize and coordinate tuning work.
- **Optimization backlog:** Treat unresolved recommendations as a backlog of tuning work to tackle during capacity planning cycles.

5.3.4.8.7 Workflow: From query to optimization

To use query recommendations:

1. Open the **Query Diagnostics** tab for your cluster.
2. Select a time range and review query performance.
3. Look for queries with associated recommendations — these are visually indicated in the query list.
4. Select the recommendation to open the **Recommendations** tab, with the relevant recommendation highlighted.
5. Review the recommendation details, including:
   - The impact on the selected query.
   - Any other queries that will benefit.
   - Guidance on applying the optimization.
6. Apply the recommendation (for example, by creating an index or updating statistics).
7. Monitor the query over time to validate performance improvements.

5.3.4.8.8 Requirements for external clusters

To view query recommendations for external (non-Hybrid Manager-managed) clusters:

- The **Query Diagnostics* extension must be enabled.
- The **Query Advisor** extension must be enabled.
- The Hybrid Manager **the Agent** must be installed and running.

Without these components, Hybrid Manager cannot collect the necessary data to generate recommendations.

5.3.4.8.9 Example scenarios

Here are typical tuning scenarios where query recommendations provide value:

- **Slow application pages:** A frequently used query with high latency is identified in **Query Diagnostics**. A recommendation suggests adding an index on a filter column. Once applied, response times improve significantly.
- **ETL job optimization:** A batch process is running slower after recent data growth. Query Diagnostics shows an aggregate query performing poorly. A statistics refresh recommendation improves planner accuracy, restoring job performance.
- **Capacity planning:** Before scaling hardware, you review Recommendations to see whether tuning can alleviate bottlenecks without adding compute.
- **Proactive tuning:** During a routine tuning cycle, you review recommendations across critical databases and systematically apply improvements. As a result, you reduce overall database load by optimizing expensive queries.
- **Release validation:** After deploying new application features, you check whether new query patterns have surfaced tuning opportunities. Applying recommendations early prevents future performance issues.

5.3.4.8.10 Supported recommendation types

Hybrid Manager currently supports:

- **Index recommendations** — such as adding a missing index on a frequently used column.
- **Statistics recommendations** — such as refreshing outdated statistics to improve planner accuracy.

Additional recommendation types are planned for future releases.

5.3.4.8.11 Related information

- *Cluster Query Diagnostics*
- *Cluster recommendations*

CHAPTER 6

Agent Factory in Hybrid Manager

The **Agent Factory workload** in Hybrid Manager brings scalable AI and machine learning capabilities to your Hybrid Manager platform. It enables you to operationalize AI across your Hybrid Manager–managed clusters and data — with deep integration across Postgres, vector search, model serving, and AI pipelines.

With Agent Factory in Hybrid Manager, you can:

- Build AI flows and agents with Langflow for internal or external-facing use.
- Serve AI models at scale with integrated KServe-powered inferencing and GPU acceleration.
- Create knowledge bases and perform Retrieval-Augmented Generation (RAG) with vector search.
- Manage your model library and deploy trusted models under Hybrid Manager governance.
- Integrate AI features directly into your applications and data pipelines.

6.1 Example AI solutions you can build

Agent Factory in Hybrid Manager enables solutions across many domains:

- Enterprise search and knowledge assistants Build RAG-based assistants that integrate with corporate documents, databases, and intranet content.
- Customer support chatbots Deploy assistants powered by your own data and domain-specific models, combining conversational AI with Knowledge Bases.
- AI-driven data apps Expose AI-powered endpoints for apps — semantic search, recommendations, similarity search, or natural language querying.
- Operational AI for internal tools Build AI agents and tools to assist with DevOps, customer success, HR automation, sales enablement, and more.
- Domain-specific model serving Serve proprietary or fine-tuned models (LLMs, embedding models, ranking models) as scalable inference services within your Hybrid Manager cluster.

You can start small — with a single assistant or model endpoint — and scale to full AI-powered applications using the Agent Factory architecture.

6.2 Agent Factory in Hybrid Manager workloads

Hybrid Manager supports a full set of Agent Factory capabilities, tightly integrated into its control plane:

6.2.1 Build AI flows

- *Langflow FAQ* — Build and deploy AI pipelines and flows using EDB components connected to HM-managed models, knowledge bases, and Postgres clusters
- *Pipeline Designer concepts* — Create AIDB data pipelines and knowledge bases through a no-code visual interface

6.2.2 Model management and serving

- *Step 2: Access Model Library* — Manage and govern your model assets
- *Model Serving* — Deploy and scale model inference services on Kubernetes
- *GPU resource management* — Configure and allocate GPU capacity for serving

---

6.3 How Agent Factory fits within Hybrid Manager

Hybrid Manager is the control plane for your databases, analytics, and AI workloads. Agent Factory complements other HM capabilities:

- Databases: Provision and manage Postgres clusters; Agent Factory connects to these for vector search, RAG, and application endpoints.
- Analytics: Offload data to Iceberg/Delta and query via SQL; Agent Factory's pipelines and Vector Engine integrate with these datasets for retrieval.
- Operations: Use HM observability and governance to monitor AI flows, pipelines, and model serving alongside databases and analytics.

Prerequisites for Agent Factory in HM:

- A Hybrid Manager project with Agent Factory enabled and required entitlements for Langflow, Pipeline Designer, and Model Serving.
- Access to GPU resources where needed for serving. See *GPU resource management* .

6.4 Sovereign AI in Hybrid Manager

Agent Factory on Hybrid Manager runs AI workloads co-located with your data:

- **Your environment.** Model serving, pipelines, and vector search run directly inside your Hybrid Manager project and Kubernetes cluster.
- **Your data.** Source documents, embeddings, and inference results never leave your environment unless you explicitly configure it.

- **Your models.** Model images are curated in the Model Library and served from endpoints inside your cluster.
- **Your visibility.** Model logs and AI workload metrics are observable through the same platform you use for Postgres and analytics.

This is distinct from external AI services, where prompts, documents, and embeddings leave your network and are processed in a third-party environment. With Hybrid Manager, all retrieval, generation, and storage happen inside your project and infrastructure.

For high-security deployments, Hybrid Manager can operate fully air-gapped — model images are preloaded, and all workloads stay entirely within your controlled cluster. See *Prerequisites* for air-gapped setup.

6.5 Get started

- New to Agent Factory on HM? Start here: *Getting started with Pipeline Designer*
- Review environment requirements: *Prerequisites*
- Explore real-world patterns: *Use Cases and Personas for Agent Factory on Hybrid Manager*
- Understand the architecture: *Agent Factory Architecture on Hybrid Manager*

6.5.1 Getting Started with AI Factory

6.5.1.1 Getting started with Agent Factory on Hybrid Manager

Agent Factory is the AI and machine learning workload in Hybrid Manager. It lets you deploy and serve NVIDIA NIM models on your own GPU infrastructure, build AI flows and agents with Langflow, create knowledge bases and RAG pipelines with Pipeline Designer, and expose model endpoints to your applications, all within your sovereign Kubernetes environment.

This page walks you through the fastest path to get something running.

6.5.1.1.1 What you'll build

Agent Factory on Hybrid Manager supports two starting paths:

- Building an AI flow in Langflow backed by a knowledge base.
- Creating a knowledge base pipeline through Pipeline Designer.

Both paths require a deployed model cluster, start there if you haven't already.

6.5.1.1.2 Before you start

Complete these steps in order before following either path below. Agent Factory must be enabled and a model cluster must be running before Langflow and Pipeline Designer can use models.

- Enable Agent Factory in your project: *Enabling Agent Factory*
- Provision GPU node groups: *GPU Recommendations for Default NIM Models*
- Deploy a model cluster from the Model Library: *Deploy Agent Factory with Hybrid Manager UI*

6.5.1.1.3 Option A — Build an AI flow with Langflow

Langflow is a visual flow builder that ships with HM. Use it to wire together EDB components — knowledge bases, embedding models, LLMs, and Postgres databases — into AI flows and agents you can run interactively or deploy as callable services.

- *Langflow quickstart* — end-to-end walkthrough: knowledge base setup, component wiring, and running the flow in the playground.

6.5.1.1.4 Option B — Create a knowledge base pipeline

Pipeline Designer is a no-code wizard in the HM console for building AIDB data pipelines. Use it to ingest documents, generate embeddings, and store them in a vector knowledge base ready for semantic search or RAG.

- *Getting started with Pipeline Designer*
- *Creating pipelines*

6.5.1.1.5 Access model endpoints from your applications

Any model deployed through HM exposes an OpenAI-compatible REST endpoint. You can call it directly from applications, scripts, or external tools without going through Langflow or Pipeline Designer.

- *Access KServe endpoints*
- *Quickstart (Python)*

6.5.1.1.6 Monitor and iterate

Once your models and flows are running, use HM's built-in observability to track inference latency, GPU utilization, and flow health.

- *Observability (Agent Factory on HM)*
- *Monitor InferenceService*

6.5.1.2 Prerequisites for Agent Factory on Hybrid Manager

6.5.1.2.1 Enabling Agent Factory

Before you can use Agent Factory in Hybrid Manager, you need to enable it during the installation of HM.

See *Creating other necessary secrets* for the Agent Factory-specific enablement steps.

6.5.1.2.2 Infrastructure Requirements

Kubernetes Cluster Foundation

Agent Factory requires a properly configured Hybrid Manager Kubernetes cluster with sufficient resources for AI workloads. The cluster must support GPU scheduling and have appropriate node groups configured for different workload types.

Cluster Requirements

- Kubernetes 1.27 or later with GPU device plugin support
- NVIDIA GPU operator installed for GPU node management
- Sufficient CPU and memory for orchestration components
- Network policies supporting service mesh communication

GPU Node Configuration

GPU resources are essential for model serving operations. Node configuration must align with model requirements and expected workload characteristics.

GPU Node Requirements

- NVIDIA GPUs with CUDA 12.1+ support
- GPU nodes labeled with `nvidia.com/gpu=true`
- GPU taint `nvidia.com/gpu` for dedicated scheduling
- Sufficient GPU memory for target model sizes

For per-model GPU requirements and cloud node recommendations, see *GPU Recommendations for Default NIM Models* .

6.5.1.2.3 Registry Configuration

Internet-connected deployments

For clusters with internet access, no additional image mirroring is required. Ensure your cluster has egress to `nvcr.io` so HM can pull NVIDIA NIM images during model deployment.

Air-gapped deployments

For environments without internet access, use the hub guides to prepare images and deployment assets in advance, and configure private registry access in Hybrid Manager.

Model image migration

Prepare and mirror required model images to your private registry following these hub references:

- Private registries and image governance: *Step 2: Access Model Library*
- KServe manifests and deployment flow: *Using NVIDIA NIM in your environment*

Model registry updates

Update default model references to point to your private registry. Work with your Model Library configuration and HM API/console paths.

Profile caching (NIM)

Some NVIDIA NIM models use runtime profiles that must be available locally for offline operation. Follow NVIDIA's documentation for profile discovery and caching strategies.

- NVIDIA NIM docs: https://docs.nvidia.com/nim/
- Hub usage patterns: *Using NVIDIA NIM in your environment*
- *Air-gapped cache*

6.5.1.2.4 Next Steps

With prerequisites satisfied, proceed to:

1. Review *GPU Recommendations for Default NIM Models* to plan your node groups.
2. Deploy your first model: *Deploy Agent Factory with Hybrid Manager UI* .

3. Build your first AI flow: *Getting started with Pipeline Designer* .

For architecture context, see *Agent Factory Architecture on Hybrid Manager* .

6.5.1.3 GPU Recommendations for Default NIM Models

6.5.1.3.1 Overview

From Hybrid Manager, there are two primary consumers of AI models:

- PG.AI Knowledge Base (AIDB Postgres extension) for creating and maintaining AI Knowledge Bases.
- Langflow for building AI pipelines and agentic flows.

6.5.1.3.2 Default NIM Models

Model type	NIM model	NVIDIA NIM documented resource requirements
Text completion	nvidia/llama-3.3-nemotron-super-49b-v1	4 × L40S
Text embeddings	nvidia/llama-3.2-nemoretriever-300m-embed-v1	1 × L40S
Image embeddings	nvclip	1 × L40S
OCR	paddleocr	1 × L40S
Text reranking	llama-3.2-nv-rerankqa-1b-v2	1 × L40S

6.5.1.3.3 Minimum GPU Requirement

Based on the default models above, the minimum to run them concurrently is 8 × L40S GPUs.

6.5.1.3.4 Cloud Mappings

- AWS EKS: recommend a node group with 2 × `g6e.12xlarge` nodes.
- GCP GKE: recommend a node pool with 2 × `a2-highgpu-4g` nodes.

Note: GCP does not offer L40S GPUs. The recommended A2 nodes with A100 GPUs are supported and documented for the NIM models listed above.

6.5.2 Understand the AI Factory Architecture

6.5.2.1 Agent Factory Architecture on Hybrid Manager

6.5.2.1.1 Architectural Overview

Agent Factory deploys as a collection of containerized services within Hybrid Manager's Kubernetes infrastructure, delivering sovereign AI capabilities through integrated *model governance* , *inference serving* , and Langflow-based AI flow development. The architecture ensures complete data sovereignty by processing all AI workloads within customer-controlled Kubernetes clusters, leveraging local GPU resources and object storage.

The system operates across three architectural layers: a control plane for governance and orchestration, a runtime layer for model serving and flow execution, and a storage layer for model artifacts and *Knowledge bases and retrieval* . These layers integrate through Kubernetes APIs and custom resources, providing unified management while maintaining isolation between projects and workloads.

6.5.2.1.2 Core Components

Model Library

The *Step 2: Access Model Library* operates as a control plane service managing model lifecycle and governance across the platform. This service maintains a centralized registry of approved models while enforcing security and compliance policies before models reach production environments.

The library consists of several interconnected services:

- Registry synchronization service that monitors external container registries
- Policy engine evaluating models against organizational governance rules
- Metadata service tracking model versions, performance benchmarks, and approvals
- Storage interface managing model artifacts in object storage backends

Model metadata persists in PostgreSQL databases managed by Hybrid Manager, ensuring consistency with other platform data. The library exposes models to project namespaces through Kubernetes custom resources, enabling declarative model deployment while maintaining centralized governance. See also: *Model Library explained* .

Inference Server Infrastructure

Inference servers deploy as KServe *InferenceServices* within project namespaces, providing scalable *Model Serving* through specialized container pods. These pods encapsulate model runtime engines optimized for different frameworks and hardware configurations.

Inference pod configurations include:

- Model runtime containers
- Resource specifications defining GPU allocation, memory limits, and CPU requirements (see *Setup GPU* and *Update GPU resources*)
- Volume mounts connecting to model storage and configuration data
- Environment variables containing endpoint configurations and runtime parameters
- Health check definitions for liveness and readiness probes

Autoscaling configurations respond to metrics including request latency, GPU utilization, and queue depth, ensuring optimal resource utilization while meeting performance targets. For deployment options, see *Model Deployment* and *Configure ServingRuntime* .

Langflow Runtime

Langflow FAQ runs as a managed workload in Hybrid Manager, providing a visual flow builder and a deployment lifecycle for turning flows into callable services. The runtime is containerized and deployed flows are hosted in isolated Kubernetes namespaces.

The Langflow architecture within HM includes:

- A shared Langflow editor environment where flows are built and tested
- Per-deployment runtime containers that host published flows as long-running services
- EDB components (EDB Model Server, EDB Embeddings, EDB Knowledge Base, and others) that wire flows to HM-managed resources
- State and credentials managed as Kubernetes secrets, scoped to each deployment's namespace

Flows access model endpoints through cluster-local service DNS, and all traffic between the Langflow runtime, model server pods, and Postgres clusters stays within the project namespace. See *Langflow FAQ* for the full component and deployment reference.

Storage

Agent Factory uses object storage for model artifacts, datasets, and knowledge bases, with MinIO or cloud provider services (S3, Azure Blob, GCS) as primary storage backends. This separates compute from storage, enabling independent scaling and cost optimization.

Storage access occurs through standardized S3 APIs with authentication via service account credentials or cloud provider identity mechanisms. Persistent volume claims provide local caching for frequently accessed models, reducing network overhead and improving inference latency.

6.5.2.1.3 Infrastructure

Agent Factory runs on standard Kubernetes primitives — GPU device plugins, KServe, object storage, and a service mesh. For deep-dives on GPU setup, network policies, HA configuration, and monitoring, see the hub references:

- *GPU setup*
- *Create InferenceService*
- *Configure ServingRuntime*
- *Monitor InferenceService*

6.5.2.2 Use Cases and Personas for Agent Factory on Hybrid Manager

6.5.2.2.1 Overview

Agent Factory on Hybrid Manager enables diverse organizational use cases through sovereign AI infrastructure. Organizations leverage these capabilities to build intelligent applications while maintaining complete control over data and models within their Kubernetes environments.

6.5.2.2.2 Key Use Cases

Internal Knowledge Management

Organizations deploy AI assistants that provide employees with natural language access to internal documentation, policies, and institutional knowledge.

Implementation Pattern

Financial services firms index compliance documentation, regulatory filings, and internal procedures into knowledge bases. Employees query these systems using conversational interfaces, receiving accurate answers with source citations while maintaining data sovereignty.

Technical Components

- Knowledge bases indexing SharePoint, Confluence, and database content
- LLM endpoints for query understanding and response generation
- Retrieval systems with role-based filtering for access control

See *Getting started with Pipeline Designer* for implementation guidance.

Customer Service Automation

AI-powered customer support systems handle routine inquiries while escalating complex issues to human agents.

Implementation Pattern

E-commerce companies deploy assistants that access order systems, product catalogs, and return policies through secure APIs. These assistants resolve a significant portion of customer inquiries automatically while maintaining full audit trails for quality assurance.

Technical Components

- REST API tools for order management system integration
- Knowledge bases containing product documentation and FAQs

See *Langflow quickstart* for a working example.

Enterprise RAG Systems

Retrieval-augmented generation applications ground LLM responses in organizational data, reducing hallucination while maintaining accuracy.

Implementation Pattern

Healthcare organizations implement RAG systems over clinical guidelines, research papers, and treatment protocols. Medical professionals receive evidence-based recommendations with citations to source materials, supporting clinical decision-making while ensuring traceability.

Technical Components

- Vector embeddings stored in PostgreSQL with pgvector
- Hybrid search combining semantic and keyword matching
- Reranking models for result optimization

See *Knowledge bases and retrieval* for detailed architecture.

AI-Enabled APIs

Organizations expose AI capabilities through secure APIs for integration with existing applications.

Implementation Pattern

Software companies provide intelligent features within their products by deploying model endpoints accessible through standard REST interfaces. These APIs handle tasks like document summarization, sentiment analysis, and content generation while maintaining predictable performance characteristics.

Technical Components

- InferenceServices exposing OpenAI-compatible endpoints
- API gateways with authentication and rate limiting
- Load balancing across model replicas for high availability

See *Model Serving Guide* for deployment instructions.

6.5.2.2.3 User Personas

Platform Engineers

Platform engineers manage Agent Factory infrastructure, ensuring reliable model serving and optimal resource utilization.

Primary Responsibilities

- Configure GPU nodes and resource allocation
- Deploy and monitor InferenceServices
- Manage model library and registry connections
- Implement security policies and network configurations

Key Workflows

1. Provision GPU resources following *GPU Setup Guide*
2. Configure model registries per *Model Library Documentation*
3. Monitor inference performance using *Observability (Agent Factory on HM)*

Data Scientists

Data scientists leverage Agent Factory to deploy models and build intelligent applications without managing infrastructure complexity.

Primary Responsibilities

- Create and optimize knowledge bases
- Configure retrieval strategies for RAG applications
- Evaluate model performance and accuracy
- Design assistant behaviors and tool integrations

Key Workflows

1. Build knowledge base pipelines using *Getting started with Pipeline Designer*
2. Configure retrieval in a Langflow flow: *EDB Knowledge Base*
3. Deploy flows as services: *Flow deployment*

Application Developers

Developers integrate AI capabilities into business applications through APIs and SDKs.

Primary Responsibilities

- Integrate model endpoints into applications
- Build custom tooling for AI-powered features
- Handle error conditions and fallback strategies

Key Workflows

1. Access model endpoints via *KServe Integration*
2. Call models from Python: *Python Client Guide*
3. Invoke deployed Langflow flows over HTTP or MCP: *Accessing deployed flows*

Business Analysts

Business analysts use Langflow's visual flow builder to create AI applications without coding expertise.

Primary Responsibilities

- Define flow behaviors and knowledge domains
- Configure data source connections
- Test and validate flow responses
- Monitor usage patterns and effectiveness

Key Workflows

1. Build flows using the Langflow visual editor: *Langflow quickstart*
2. Use pre-built templates as starting points: *Langflow templates*
3. Publish and deploy flows for others to use: *Flow deployment*

6.5.2.2.4 Industry-Specific Implementations

Financial Services

Use Case: Regulatory compliance assistant analyzing transaction patterns and generating required reports.

Components

- Knowledge base of regulatory documentation
- Tools accessing transaction databases
- Audit logging for compliance tracking

Healthcare

Use Case: Clinical decision support system providing evidence-based treatment recommendations.

Components

- RAG over medical literature and guidelines
- Integration with electronic health records
- Strict access controls for patient data

Manufacturing

Use Case: Maintenance assistant diagnosing equipment issues using sensor data and technical manuals.

Components

- Knowledge base of equipment documentation
- Real-time data integration from IoT sensors
- Predictive maintenance model endpoints

6.5.2.2.5 Implementation Considerations

Scalability Planning

Different use cases require varying levels of scale:

- **High-volume customer service**: Multiple model replicas with auto-scaling
- **Internal knowledge systems**: Moderate scaling with caching optimization
- **Specialized assistants**: Single instances with dedicated resources

Security Requirements

Use cases determine security configurations:

- **Public-facing applications**: External ingress with authentication
- **Internal systems**: Cluster-local access with RBAC
- **Regulated industries**: Enhanced audit logging and encryption

Performance Optimization

Optimize based on use case characteristics:

- **Real-time applications**: Low-latency model serving with edge caching
- **Batch processing**: Higher throughput with larger batch sizes
- **Interactive assistants**: Balanced latency and throughput

6.5.2.2.6 Getting Started by Use Case

For Knowledge Management

1. Create a knowledge base pipeline: *Getting started with Pipeline Designer*
2. Query the knowledge base from a flow: *EDB Knowledge Base*
3. Build a RAG flow end-to-end: *Langflow quickstart*

For Customer Service

1. Build a conversational flow: *Langflow quickstart*
2. Try the Simple Migration Agent template as a reference for agentic flows: *Langflow templates*
3. Deploy the flow as a callable service: *Flow deployment*

For API Development

1. Deploy models via *Deploy Agent Factory with Hybrid Manager UI*
2. Access endpoints: *Python Client Guide*
3. Monitor performance: *Observability (Agent Factory on HM)*

6.5.2.2.7 Additional Resources

Related Documentation

- *Hybrid Manager Overview*
- *Agent Factory Architecture on Hybrid Manager*
- *Langflow FAQ*
- *Pipeline Designer concepts*

6.5.2.3 Frequently Asked Questions - Agent Factory on Hybrid Manager

6.5.2.3.1 Table of Contents

- *Platform Capabilities*
- *Installation and Setup*
- *Model Management*
- *Langflow FAQ*
- *Operations and Maintenance*
- *Troubleshooting Langflow in Hybrid Manager*
- *Security and Compliance*
- *Performance and Scaling*

6.5.2.3.2 Platform Capabilities

What types of models does Hybrid Manager support?

Hybrid Manager supports Large Language Model (LLM) deployments exclusively through NVIDIA NIM containers. Traditional machine learning models (classification, regression, time-series forecasting) are not supported in this release.

Supported NVIDIA NIM model categories:

- **Text Generation**: Large language models for chat and completion tasks
- **Text Embeddings**: Models for semantic search and RAG applications
- **Text Reranking**: Models for search result optimization
- **Multimodal Models**: Vision models including CLIP and OCR capabilities

Can I deploy custom models?

Custom models must be packaged as NVIDIA NIM containers to be compatible with Hybrid Manager. Standard machine learning frameworks (scikit-learn, XGBoost, TensorFlow for traditional ML) are not supported. Custom LLMs can be deployed if they conform to NIM container specifications and API standards.

See *Private Registry Integration* for custom NIM deployment procedures.

What distinguishes Agent Factory from cloud AI services?

Agent Factory provides complete sovereignty over AI operations:

- Models execute within your Kubernetes infrastructure
- Data remains within organizational boundaries
- No external API dependencies for inference
- Complete audit trails for regulatory compliance

6.5.2.3.3 Installation and Setup

What are the minimum infrastructure requirements?

Core Requirements

- Kubernetes 1.27+ with NVIDIA GPU operator
- NVIDIA GPUs compatible with NIM containers (L40S, A100, H100)
- 100GB+ object storage for model artifacts
- Network connectivity to NVIDIA NGC registry (or air-gapped configuration)

GPU Requirements by NIM Model Type

- Text completion (Nemotron Super 49B): 4 x L40S GPUs
- Text embeddings: 1 x L40S GPU
- Text reranking: 1 x L40S GPU
- Vision models: 1 x L40S GPU

See *Prerequisites for Agent Factory on Hybrid Manager* for comprehensive specifications.

How do I configure GPU nodes for NIM models?

GPU node preparation involves:

1. Install NVIDIA GPU operator on the cluster
2. Label GPU nodes with `nvidia.com/gpu=true`
3. Apply GPU taints for dedicated scheduling
4. Verify CUDA compatibility for NIM requirements

Detailed instructions available in *GPU Setup Documentation* .

Can Agent Factory operate in air-gapped environments?

Air-gapped deployments require advance preparation:

1. Mirror NVIDIA NIM images to private registry
2. Download and cache model profiles
3. Upload profiles to object storage
4. Configure Model Library for private registry access

Complete procedures documented in *Air-gapped deployments* .

6.5.2.3.4 Model Management

How do I deploy NVIDIA NIM models?

NIM model deployment workflow:

1. Access Model Library in HM console
2. Select NVIDIA NIM model from catalog
3. Configure resources (GPU allocation, memory, replicas)
4. Deploy InferenceService to project namespace
5. Access through generated endpoints

Step-by-step guide: *Create InferenceService* .

Which NVIDIA NIM models are available by default?

Default NIM models:

- **llama-3.3-nemotron-super-49b-v1**: Advanced reasoning and chat
- **llama-3.2-nemoretriever-300m-embed-v1**: Text embeddings
- **llama-3.2-nv-rerankqa-1b-v2**: Query-document reranking
- **nvclip**: Multimodal embeddings
- **paddleocr**: Optical character recognition

How do I manage NIM model versions?

Version management strategies:

- Model Library maintains version tags for each NIM image
- Blue-green deployments enable zero-downtime updates
- Canary deployments allow gradual traffic shifting
- Rollback through InferenceService configuration updates

6.5.2.3.5 Langflow

Langflow is the AI flow builder in Hybrid Manager, replacing the previous Griptape-based Gen AI Builder. You build AI pipelines and flows using EDB components that connect directly to your HM-managed models, knowledge bases, and Postgres clusters.

For Langflow-specific questions — components, Global Variables, knowledge bases, model usage, flow promotion, and common errors — see the *Langflow FAQ* .

6.5.2.3.6 Operations and Maintenance

How do I monitor NIM model performance?

Monitoring encompasses:

Metrics Collection

- Prometheus metrics for inference latency
- GPU utilization and memory consumption

- Token generation throughput
- Request success rates

Visualization

- Grafana dashboards integrated in HM console
- Custom panels for model-specific metrics
- Alert configuration for SLA breaches

Reference: *Observability (Agent Factory on HM)* .

How should I handle NIM model updates?

Update procedure for production deployments:

1. **Validation**: Deploy new version in development namespace
2. **Testing**: Execute performance and accuracy tests
3. **Deployment**: Implement canary or blue-green strategy
4. **Monitoring**: Track metrics during transition
5. **Decision**: Complete rollout or rollback based on metrics

6.5.2.3.7 Troubleshooting

NIM model fails to start - diagnostic steps?

Common initialization failures:

1. **GPU unavailability**: Verify GPU resources match model requirements
2. **Image pull failures**: Check NGC credentials and network connectivity
3. **Profile cache missing**: Ensure profiles available in air-gapped setups
4. **Insufficient memory**: Validate memory allocation for model size

Diagnostic commands:

```
kubectl describe inferenceservice <name> -n <namespace>
kubectl logs <pod-name> -n <namespace>
kubectl get events -n <namespace>
```

High inference latency - optimization strategies?

Performance optimization approaches:

- **Batch processing**: Increase batch size for throughput optimization
- **Model quantization**: Use INT8 quantization where supported
- **Response caching**: Cache frequent queries at application layer
- **Horizontal scaling**: Deploy additional replicas for load distribution

Poor retrieval quality in RAG applications?

Check that the correct embedding model is in use, chunk sizes are appropriate for your content, and all documents have been indexed. For Langflow-specific guidance including AIDB knowledge base diagnostics, see *How do I improve retrieval quality in a RAG flow?* .

6.5.2.3.8 Security and Compliance

How do I implement access control?

Role-based access control for AI resources involves:

- Kubernetes RBAC for namespace and resource permissions
- Model Library access controls for deployment authorization
- API key management for external endpoint access
- Network policies for inter-service communication

What encryption is implemented?

Encryption coverage:

- **At rest**: Kubernetes secrets encryption, database encryption
- **In transit**: TLS for API calls, mTLS within service mesh
- **Model artifacts**: Encrypted object storage
- **Knowledge bases**: Encrypted vector storage in PostgreSQL

Which operations are audited?

Audit logging captures:

- NIM model deployment and configuration changes
- Inference requests (configurable detail level)
- Knowledge base queries and updates
- Administrative operations on AI resources

6.5.2.3.9 Performance and Scaling

How do I configure resource quotas?

Resource quotas prevent resource exhaustion at the namespace level. Configure GPU quotas, memory limits, and storage constraints based on project requirements and available infrastructure capacity.

When should I scale horizontally versus vertically?

Horizontal Scaling (additional replicas):

- High concurrent request volume
- Stateless inference workloads
- Load distribution requirements

Vertical Scaling (increased resources per instance):

- Large model memory requirements
- Batch processing optimization
- Single-request latency minimization

What model sizes can Hybrid Manager support?

Model size constraints:

- **Single GPU**: Models up to 13B parameters
- **Multi-GPU**: Models up to 70B+ parameters using tensor parallelism
- **Memory limits**: 80GB (A100), 48GB (L40S) per GPU

NVIDIA NIM handles model sharding and parallelism automatically based on available resources.

6.5.2.3.10 Additional Resources

Getting Started

- *Prerequisites for Agent Factory on Hybrid Manager*
- *Agent Factory Architecture on Hybrid Manager*

Implementation Guides

- *Model Serving*
- *Langflow FAQ*
- *Knowledge bases and retrieval* (AIDB)
- Data pipelines (AIDB)

Troubleshooting Resources

- *Model Verification*

For issues not addressed here, contact EDB support or see *Troubleshooting Langflow in Hybrid Manager* .

6.5.3 Model Management

6.5.3.1 Model Management on Hybrid Manager

6.5.3.1.1 Purpose and Benefits

Model management within Hybrid Manager provides centralized governance and deployment capabilities for AI models running on your Kubernetes infrastructure. This system enables organizations to maintain complete control over their AI capabilities while leveraging enterprise-grade *Model Serving* infrastructure.

The integration addresses critical requirements for organizations deploying AI at scale: model governance through approved registries, scalable inference serving with GPU acceleration, and unified management through Hybrid Manager's control plane. By running models within your controlled infrastructure, you maintain data sovereignty while accessing state-of-the-art AI capabilities.

6.5.3.1.2 Core Concepts

Model Library

The *Step 2: Access Model Library* serves as your centralized governance system for AI model images. Operating within Hybrid Manager's Asset Library infrastructure, it provides a curated view of validated models ready for production deployment.

The library implements multi-stage governance:

- Automated synchronization from trusted container registries
- Security scanning and vulnerability assessment
- Approval workflows based on organizational policies
- Metadata management for versioning and documentation

Models in the library power all Agent Factory capabilities including Langflow flows, Pipeline Designer knowledge bases, and custom inference applications. Only models validated through the library's governance framework can reach production environments.

Model Serving

Model Serving transforms approved models into scalable inference endpoints using KServe within your Kubernetes clusters. This infrastructure provides production-grade model deployment with automatic scaling, health management, and resource optimization.

Key serving capabilities include:

- ***InferenceService*** resources that define deployed model endpoints
- ***ServingRuntime*** configurations optimized for different model frameworks
- GPU allocation and scheduling for high-performance inference — see *GPU Recommendations for Default NIM Models*
- Internal and external endpoint access with authentication

Management Interface

Hybrid Manager provides unified management through its web console, abstracting Kubernetes complexity while maintaining full configurability. The interface enables:

- Visual workflows for model deployment from library to serving
- Resource allocation and scaling configuration
- Monitoring dashboards for inference metrics and GPU utilization
- Access control and endpoint management

6.5.3.1.3 Implementation Workflow

Model Registration

Organizations begin by configuring repository connections to trusted model sources. The Model Library synchronizes with external registries based on defined rules, automatically discovering and validating new model versions.

```
External Registry → Repository Rules → Security Scanning → Model Library
```

Repository rules determine which models enter your environment, implementing organizational policies at the point of ingestion. This automated approach reduces manual overhead while maintaining governance standards.

Model Deployment

Validated models deploy through guided workflows that configure serving infrastructure:

1. **Model Selection**: Browse available models in the library with metadata including version, performance characteristics, and resource requirements
2. **Runtime Configuration**: Select or create *ServingRuntimes* optimized for the model framework (vLLM, TensorRT-LLM, custom)
3. **Resource Allocation**: Define GPU, memory, and CPU requirements based on expected workload
4. **Endpoint Configuration**: Set up internal cluster access or external API endpoints with authentication (see *Access KServe endpoints*)

The system creates InferenceService resources that KServe manages, handling pod scheduling, health monitoring, and traffic routing automatically.

Operational Management

Deployed models operate under continuous monitoring with automatic scaling based on demand. Hybrid Manager provides visibility through:

- Real-time inference metrics including latency and throughput
- GPU utilization tracking for resource optimization
- Error rates and health status for proactive maintenance
- Cost analysis based on resource consumption

6.5.3.1.4 Using deployed models

Once a model cluster is running, there are three ways to consume it.

From applications

Applications call deployed model endpoints directly via KServe InferenceServices. Each endpoint exposes a standard OpenAI-compatible REST API for chat completions, embeddings, or other inference tasks.

- *Access KServe endpoints*
- *Quickstart (Python)*

From AIDB (SQL patterns)

AIDB lets you call models from SQL, making them available directly inside Postgres — useful for embedding pipelines or enabling in-database inference.

- *Using models via OpenAI-compatible SQL calls*
- *Using NVIDIA NIM in AIDB*

Hybrid Manager specifics

When models are deployed through Hybrid Manager:

- **Service URLs.** Each model is exposed as an internal KServe endpoint within your HM project. The URL is visible in the Model Library or the Model Serving details page.

- **Authentication.** Endpoints are protected by the platform. Applications running inside the same project can reach them directly. For external access, configure authentication using the HM ingress and project-scoped credentials.
- **Observability.** Requests and logs flow into HM observability, giving you usage metrics, latency, and error tracking.

6.5.3.2 Deploy Agent Factory with Hybrid Manager UI

6.5.3.2.1 Overview

This guide provides deployment procedures for Agent Factory components through the Hybrid Manager (HM) web interface. The deployment process covers GPU infrastructure setup, NVIDIA NIM model deployment, and Gen AI application configuration.

6.5.3.2.2 Prerequisites

Infrastructure Requirements

Before deploying Agent Factory components:

- Kubernetes cluster with GPU nodes configured
- NVIDIA GPU operator installed
- Access to NVIDIA NGC registry or private registry
- Object storage for model profiles (air-gapped deployments)

GPU Node Configuration

Verify GPU nodes meet NIM requirements — see *GPU Recommendations for Default NIM Models* .

6.5.3.2.3 Step 1: Configure Registry Authentication

Internet-Connected Deployments

For clusters with internet access, configure NVIDIA NGC authentication:

1. Obtain NGC API key from NVIDIA NGC Portal

2. Create authentication secrets via HM UI: - Navigate to **Project Settings** > **Secrets**. - Create `nvidia-nim-secrets` with NGC_API_KEY. - Create `ngc-cred` Docker registry secret.

Alternatively, use kubectl:

```
NGC_API_KEY=<your-ngc-api-key>

#  Create runtime secret

kubectl -n default create secret generic nvidia-nim-secrets \
    --from-literal=NGC_API_KEY=${NGC_API_KEY}

kubectl -n default annotate secret nvidia-nim-secrets \
    replicator.v1.mittwald.de/replicate-to=m-.*

#  Create image pull secret

kubectl -n default create secret docker-registry ngc-cred \
```

(continues on next page)

(continued from previous page)

```
    --docker-server=nvcr.io \
    --docker-username=$oauthtoken \
    --docker-password=${NGC_API_KEY}

kubectl -n default annotate secret ngc-cred \
    replicator.v1.mittwald.de/replicate-to=m-.*
```

Air-Gapped Deployments

For environments without internet access:

1. **Mirror NIM Images**: Copy required images to private registry using skopeo.
2. **Update Model URLs**: Configure HM to reference private registry locations.
3. **Cache Profiles**: Download and store model profiles in object storage.
4. **Configure Storage Path**: Reference cached profiles during model deployment.

See *Air-Gapped Configuration* for detailed procedures.

6.5.3.2.4 Step 2: Access Model Library

Navigate to the Model Library interface:

1. Log into Hybrid Manager console.
2. Select your project from the project selector.
3. Navigate to **Agent Factory → Model Library**.

The Model Library displays available NVIDIA NIM models:

- **llama-3.3-nemotron-super-49b-v1**: Advanced reasoning and chat capabilities (128K context)
- **llama-3.2-nemoretriever-300m-embed-v1**: High-quality text embeddings
- **llama-3.2-nv-rerankqa-1b-v2**: Multilingual query-document reranking
- **paddleocr**: Ultra-lightweight OCR system
- **nvclip**: Multimodal embeddings for image and text

6.5.3.2.5 Step 3: Deploy Model Server Cluster

Select Model for Deployment

1. Click **Deploy Model** in Model Library.
2. Select target model from available options.
3. Review model requirements and documentation links.

Configuring Deployment Parameters

Configure the model server cluster settings:

Instance Configuration

- **Server Instances**: Default 1 (increase for high availability)
- **Minimum Instances**: 1 (0 for scale-to-zero when supported)

- **Maximum Instances**: Based on load requirements

Resource Allocation

- **Memory**: Configure based on model requirements
- Text completion models: 64-128 GB
- Embedding models: 32-64 GB
- **CPU**: Default values typically sufficient
- **GPU**: Match documented requirements
- llama-3.3-nemotron: 4 GPUs
- Other models: 1 GPU typical

Scaling Configuration

- **Concurrent Request Threshold**: Configure autoscaling trigger
- **Scale to Zero**: Enable when supported (may be unavailable in initial release)

Deploy Model

1. Review configuration summary.
2. Click **Deploy** to initiate deployment.
3. Monitor deployment status in Model Server Clusters view.

Deployment creates:

- KServe InferenceService resource.
- Predictor pods with configured resources.
- Service endpoints for model access.

6.5.3.2.6 Step 4: Monitor Deployment Status

View Model Server Clusters

Navigate to **Agent Factory** → **Model Server Clusters** to view:

- Cluster display name
- Load balancer ingress URI
- Deployment status (Active/Healthy, Pending, Failed)
- Model details and tags
- Resource utilization metrics

Access Cluster Details

Click on a cluster name to view:

- Detailed configuration parameters
- Real-time health metrics
- Grafana dashboard integration
- Inference latency charts

- System health indicators

6.5.3.2.7 Step 5: Configure API Access

Generate API Tokens

For external access to deployed models:

1. Navigate to **Agent Factory → API Token Management**
2. Click **Create Token**
3. Provide token reference name
4. Store generated token securely

API tokens enable:

- RAG application authentication
- External service integration
- Programmatic model access

Access Endpoints

Models expose OpenAI-compatible endpoints:

Internal Access (within cluster):

```
http://<service-name>.<namespace>.svc.cluster.local/v1/chat/completions
```

External Access (with ingress):

```
https://<ingress-url>/v1/chat/completions
```

6.5.3.2.8 Step 6: Build Gen AI Applications

Gen AI application building in Hybrid Manager is done through **Langflow**, accessible via **Launchpad** in the HM console. Langflow provides a visual flow editor with EDB components that connect directly to your HM-managed models, knowledge bases, and Postgres clusters.

For a step-by-step walkthrough of building your first Gen AI application, see *Langflow quickstart* .

For a full reference of EDB components, flow deployment, and sharing flows between instances, see *Langflow FAQ* .

6.5.3.2.9 Step 7: Update and Manage Models

Edit Model Server Cluster

To modify running clusters:

1. Navigate to **Model Server Clusters**
2. Select cluster to edit
3. Adjust parameters:
   - Instance counts
   - Resource allocations
   - Scaling thresholds

1. Apply changes (triggers rolling update)

Rolling Updates

Updates maintain availability through:

- Connection draining before restart
- Health verification between instances
- Automatic rollback on failure

6.5.3.2.10 High availability and isolation for Langflow flows

High availability, project-level isolation, stable routing, and access control for a Langflow flow are properties of a *deployed* flow, not of the Langflow flow builder. The flow builder is a single shared instance: flows you run there are not isolated per project and you can't scale them independently. To run a flow as an isolated, highly available, independently scalable service with its own endpoints, publish and deploy it. See *Flow deployment* , which covers replicas, endpoints, access levels, and credential handling for deployed flows.

For the model server clusters that Langflow flows depend on, adjust the number of running instances in **Estate** → **InferenceServices** → select the inference service → **Edit** → **Inference Service Instances**.

6.5.3.2.11 Troubleshooting

Common Deployment Issues

Model Fails to Start

- Verify GPU availability matches requirements
- Check registry authentication secrets
- Review pod events and logs

High Inference Latency

- Adjust batch size parameters
- Increase GPU allocation
- Scale replicas for load distribution

API Token Issues

- Verify token hasn't expired
- Check network policies
- Confirm ingress configuration

For detailed diagnostics, see *Troubleshooting Langflow in Hybrid Manager* .

6.5.3.2.12 Additional Resources

Reference Documentation

- *GPU Setup Guide*
- *Create InferenceService*
- *Model Serving Concepts*

NVIDIA Model Documentation

- Llama 3.3 Nemotron
- Arctic Embed
- NV-ReRankQA
- PaddleOCR
- NV-CLIP

6.5.3.3 External inference services

6.5.3.3.1 Context

An external inference service connects Hybrid Manager (HM) to a remote model provider hosted outside the cluster — such as OpenAI, Google Gemini, Anthropic, or NVIDIA NIM. You configure the provider's URL, model name, and API key once; HM stores the credentials in Kubernetes secrets and handles authentication transparently for every downstream request.

This page covers:

- Viewing the **Inference Services list**
- Registering a new external inference service
- Getting the details of a registered service
- Updating a registered service
- Deregistering a service

6.5.3.3.2 Inference Services list

Open the **Inference Services list** page from the **Estate → Inference Services** menu in your HM project.

The list displays each service's name, model, and current status. When the status isn't **Ready**, a short **Status Message** explains the cause.

Status

Status	Meaning
Ready	The service is healthy and accepting requests.
Failed	The most recent health check failed. The service detail page shows a **Status Message** with the diagnosis (for example, *API key is missing or invalid*, *endpoint unreachable*, or *account quota exhausted*).
Pending	The endpoint hasn't yet been verified by the background health check. Typically seen during the first few minutes after an HM restart, while the cache warms up. The **Status Message** reads *Remote connection check is pending — upstream has not yet been verified*.
Unknown	The underlying Kubernetes resources haven't reported a status yet — usually a transient state right after creation.

Status is refreshed every 5 minutes by a background health check. After you create or edit a service, the connectivity probe runs inline so the result is reflected immediately.

6.5.3.3.3 Register an external inference service

Navigate to **Estate → Quick Actions → Register External Inference Service** in your HM project.

> **Tip**
>
> You can also reach the form from the Inference Services list page via the **Quick Actions** menu.

Prerequisites

Before registering, confirm you have:

- The provider's base URL (scheme and host, without a trailing `/v1`).
- The model name exactly as the provider expects it (case-sensitive).
- A valid API key for the provider.
- Network reachability from the HM cluster to the upstream hostname. Your HM administrator may need to allow egress to the provider's domain.

Quick reference by provider

Use this table to fill in the form for the most common providers. The columns map directly to the fields described below.

For providers not listed here, see the *Form fields* below.

Form fields

External Service Name (required)

A unique identifier for this service within HM. Must follow DNS-style naming rules:

- Lowercase letters and digits only.
- Hyphens (`-`) are allowed within segments but not at the start or end.
- Dots (`.`) are allowed as segment separators.
- No uppercase letters, underscores, or spaces.
- Maximum 63 characters.

Example: `openai-gpt-4o-mini` , `azure.gpt-4o.prod` .

Tags (optional)

Reuse existing HM tags to group and filter services. Tags have no effect on request routing or authentication.

Model name (required)

The exact identifier the upstream provider expects, as documented by the provider. This value is case-sensitive. Slash-separated names (such as `meta/llama-3.1-8b-instruct` or `openai/gpt-4o-mini`) are supported. See the *Quick reference by provider* table above for examples.

API Key (required for most providers)

The API key only — don't include the `Authorization: Bearer …` prefix. HM adds the correct auth header automatically based on the **API Protocol Version** you select.

Model Base URL (required)

The scheme and host (plus any required path prefix) for the provider's API. See the *Quick reference by provider* table above for the correct value for common providers.

Don't include `/v1` — consumer applications append `/v1` (or `/v1beta` for Gemini) themselves. Including `/v1` here causes duplicated paths such as `/v1/v1/chat/completions` , which returns a 404. A trailing slash is tolerated and stripped automatically.

> **Tip**
>
> Self-hosted vLLM servers return *404* on the per-model endpoint */v1/models/{name}* , which normally causes the registration probe to fail. HM handles this by falling back to *GET /v1/models* and matching the model name in the list, so vLLM-served models register successfully.

Functions (optional, multi-select)

Capability tags that consumer applications filter on when discovering available models. Use the predefined values below for HM's built-in consumers; for your own applications, any string is valid.

Built-in consumer	Required function tag
HM chatbot	*chatbot-gen-content*
AIDB pipeline step	The matching *aidb-** tag (see your AIDB pipeline documentation)

Leave this field empty if you're exposing the service exclusively to custom applications that perform their own model selection.

API Protocol Version (required)

Controls both the request body format and the outbound authentication header. Choose the option that matches the provider's native API.

Option	Request body shape	Auth header sent	Use for
OPENAI_V1	OpenAI Chat Completions	*Authorization: Bearer <key>*	OpenAI, NVIDIA NIM, vLLM, OpenRouter, any OpenAI-compatible endpoint
GEMINI_V1_BETA	Google Gemini	*x-goog-api-key: <key>*	Google Gemini native API only
ANTHROPIC_V1	Anthropic Messages	*x-api-key: <key>* + *anthropic-version: 2023-06-01*	Anthropic Claude

Allow Insecure Connection (optional, default off)

Disables TLS certificate verification on outbound calls to the upstream. Enable this only if the upstream uses a self-signed certificate or a certificate signed by a CA not trusted by the HM cluster.

Use this toggle when registering another HM's external inference endpoint (for example, HM2 federating through HM1). HM1's ingress typically presents a self-signed certificate, which causes TLS errors for both the registration probe and runtime forwarding. HM-to-HM federation also requires both ends to use `OPENAI_V1` as the API Protocol Version.

> **Warning**
>
> This setting is create-only. You can't toggle it after registration. If you need to change it, delete the service and re-register. Only enable this for development environments or trusted self-signed certificates — disabling TLS verification reduces security.

After clicking Register

HM validates the endpoint before creating any infrastructure. For OpenAI (`OPENAI_V1`), Anthropic (`ANTHROPIC_V1`), and Google Gemini (`GEMINI_V1_BETA`), HM performs a live connectivity probe that checks both reachability and credential validity. If the endpoint is unreachable or the API key is rejected, registration fails immediately with an error — no resources are created.

> **Note**
>
> For some *OPENAI_V1* providers — such as NVIDIA NIM, HuggingFace, OpenRouter, and self-hosted vLLM running without authentication — the models endpoint doesn't require an API key. A connectivity probe is still performed, but a wrong API key may still return HTTP 200. Key validity isn't guaranteed at registration time for these providers.

After registration, the status is refreshed every 5 minutes by a background health check. If the upstream becomes unreachable or starts rejecting credentials, the service flips to **Failed** (with an explanatory **Status Message**) at the next refresh tick.

Use the service

Once the service is ready, it's available to:

- **HM chatbot** — The chatbot picks up services tagged with `chatbot-gen-content` automatically.
- **Pipeline Designer** — Registered external models appear in the model picker alongside HM-hosted models. For details, see *External inference services* .
- **Gen AI Builder** — Models are available as inference targets in Gen AI Builder pipelines once registered.

6.5.3.3.4 Retrieve inference service details

Click a service name in the **Inference Services list** to open its detail view, which shows the service's configuration, current status, and available actions.

Details

Field	Description
External Service Name	The unique identifier assigned at registration.
Model name	The model identifier forwarded to the upstream provider.
Model Base URL	The upstream endpoint the proxy routes requests to.
API Protocol Version	The request format and authentication header in use (*OPENAI_V1*, *GEMINI_V1_BETA*, or *ANTHROPIC_V1*).
Functions	The capability tags currently assigned to the service.
Allow Insecure Connection	Whether TLS certificate verification is disabled for outbound calls.
Status	Current health of the service: **Ready** , **Failed** , **Pending** , or **Unknown** .
Status Message	A human-readable explanation that accompanies the status. Empty when the service is **Ready** ; carries the probe diagnosis (for example, *API key is missing or invalid (status 401 from ...)* or *endpoint unreachable: ...*) when **Failed** ; carries the cache warm-up text when **Pending** .

Note

The API key isn't displayed in the service detail page. To replace it, open **Quick Actions → Edit Service** and enter a new value. HM rotates the underlying secret automatically.

Troubleshooting from the Status Message

When a service is **Failed**, the **Status Message** tells you what to fix. Common patterns:

Status message	Likely cause	What to do
API key is missing or invalid	Wrong or expired API key.	Open **Edit** and replace the **API Key** value.
API key is missing/invalid or account credits are exhausted (Anthropic)	Anthropic returns the same status for a bad key and for depleted credits.	Verify the key first; if it's correct, top up the Anthropic account.
account quota exhausted — check billing and credits (OpenAI)	OpenAI billing balance is empty.	Top up the OpenAI account; no change to the HM service is needed.
API key lacks permission for this endpoint	Key is valid but doesn't have access to the model (project scoping, tier).	Grant model access on the provider's console, or use a different key.
endpoint not found — check the base URL and model name	Typo in **Model Base URL** or wrong **model name** .	Verify both values. Make sure Base URL doesn't end with */v1*.
endpoint unreachable: ... x509: ...	TLS error against an untrusted certificate.	Delete and re-register with **Allow Insecure Connection** enabled (the toggle is create-only).

continues on next page

Table 6 – continued from previous page

Status message	Likely cause	What to do
endpoint unreachable: ... (other)	Network egress blocked, DNS failure, or upstream down.	Check egress rules from the HM cluster and confirm the provider is online.
rate limited by upstream	Provider is throttling requests.	Wait and retry; recurring rate limits indicate the upstream account tier needs upgrading.
upstream is temporarily unavailable / *upstream server error*	Provider-side outage.	Wait; the next refresh tick will recover automatically once upstream is healthy.
Remote connection check is pending — upstream has not yet been verified	Background health check hasn't run yet (typically after an HM restart).	Wait up to 5 minutes for the next refresh tick.

6.5.3.3.5 Update inference service parameters

Some parameters can be updated after registration; others require deregistering and re-registering the service.

What can and cannot be changed

Field	Editable after registration?
Functions	Yes
API Key	Yes — HM rotates the underlying Kubernetes secret automatically
External Service Name	No — delete and re-register
Model name	No — delete and re-register
Model Base URL	No — delete and re-register
API Protocol Version	No — delete and re-register
Allow Insecure Connection	No — delete and re-register

Note

API Protocol Version is locked after registration. Each protocol probes a different endpoint path with a different authentication header, and the connectivity probe that ran at registration only passed because the original protocol matched the **Model Base URL** and **Model name**. Since those two fields can't be changed, any attempt to switch to a different protocol fails the connectivity probe and the update is rejected. To use a different protocol, deregister the service and register a new one.

How to edit a service parameter

Open the service detail page and select **Quick Actions → Edit Service**, or click the pencil icon on the **Inference Services** list.

Note

HM runs a connectivity probe before applying the update. If the endpoint is unreachable or the new API key is rejected, the update fails and no changes are applied.

6.5.3.3.6 De-register an external inference service

Warning

Deregister is permanent. All associated Kubernetes resources (namespace, secret, *ServingRuntime* , *InferenceService*) are removed immediately. This action can't be undone.

How to deregister

To delete a service, either open the service detail page and select **Quick Actions → Deregister External Inference Service**, or click the trash icon on the **Inference Services** list.

HM blocks deletion if the service is currently referenced by one or more pipelines. Remove or update those pipelines first, then retry.

When deletion succeeds, HM:

1. Removes all Kubernetes resources backing the service (including the API key secret).
2. Removes the service record from the database.
3. Clears all tags associated with the service.

6.5.4 Build with AI

6.5.4.1 Langflow

Langflow is an open-source tool to build and deploy AI pipelines and Model Context Protocol (MCP) servers. It supports important AI functionality like agents and the Model Context Protocol (MCP), and it doesn't require the use of specific large language models (LLMs) or vector stores.

EDB ships Langflow in Hybrid Manager with two additions on top of upstream:

- ** *EDB components* **: a set of Langflow components in the** EDB** folder that connect flows to HM-managed resources (knowledge bases, vector indexes, databases, model server clusters, local `aidb` models, and HM platform tools).
- ** *Flow deployment* **: a publishing and deployment lifecycle for turning flows into long-running, callable services hosted on HM.

6.5.4.1.1 Prerequisites

Before using EDB components in Langflow, you need the required HM roles and an access key. You can manage users and permissions in the HM console.

Roles:

- **EDB Embeddings** and **EDB Model Server** require the AI Model Manager role at organization level:

```
-   Profile > **User management** > **Assign Organization Roles** > **AI Model␣
↪Manager**.
```

- **EDB Knowledge Base**, **EDB Database**, **EDB Vector Index**, **EDB Embedded Models**, and **EDB Airman MCP** require the Project Viewer role on the project that owns the relevant cluster:

```
-   Project > **Users** > **Assign Project Roles** > **Project Viewer**.
```

- **EDB Platform** does not require a specific HM role assignment. It uses the identity of whoever is running the flow — the logged-in user in the flow builder, or the caller's identity in a deployed flow. Tool calls execute with that identity's permissions.

Access key:

- Create an access key by creating a machine user or via the My Account option in your profile. Store it in Langflow as a Global Variable named `HM_API_KEY` so EDB components pick it up automatically.

6.5.4.1.2 What's next

- *Langflow quickstart* : Build your first RAG flow in Langflow on Hybrid Manager.
- *EDB components* : Reference for each component in the EDB folder, with prerequisites, fields, and usage.
- *Using models in Langflow* : How to consume EDB-hosted model endpoints from Langflow flows.
- *Flow deployment* : Publish a bundle of flows, deploy published flows, and access deployed flows.
- *Langflow templates* : Pre-built flow templates for common Hybrid Manager workflows.
- *Langflow FAQ* : Common questions about building, configuring, and operating Langflow flows.
- *Troubleshooting Langflow in Hybrid Manager* : Diagnostic steps and fixes for common issues with Langflow flows and EDB components.

Langflow quickstart

This quickstart walks you through building a simple retrieval-augmented generation (RAG) flow in Langflow on Hybrid Manager. By the end you'll have a working flow that accepts a user question, retrieves relevant content from a knowledge base, and returns a grounded answer from an LLM, all running within your HM cluster.

Prerequisites

Complete the Langflow prerequisites for roles and access key setup, then make sure you also have:

- A GPU node deployed for model serving. See *GPU Recommendations for Default NIM Models* .
- At least one model cluster deployed from the Model Library with an LLM for generation. See *Model Management on Hybrid Manager* . The knowledge base in this quickstart uses the AIDB `bert` embedded model, which runs on the database CPU — no embedding model cluster is required.
- A Postgres cluster in your HM project with a table containing the content you want to query. Have the database username and password for that cluster ready — you'll need them in Step 2.

Prepare your data: create a knowledge base

Before opening Langflow, create a knowledge base in your HM Postgres cluster so the flow has content to retrieve from.

You can run this using psql or any SQL client connected to your cluster. To find the connection details, go to your cluster in the HM console and select **Connect**.

Run the following, replacing the table name and column with your own data:

```
CREATE EXTENSION IF NOT EXISTS aidb CASCADE;

SELECT aidb.create_table_knowledge_base(
    name               => my_knowledge_base,
    model_name         => bert,
    source_table       => public.my_content_table,
    source_data_column => content,
    source_data_format => Text
);

SELECT aidb.bulk_embedding(my_knowledge_base);
```

Verify the knowledge base is ready:

```
SELECT * FROM aidb.kbstat;
```

Note

This quickstart creates the knowledge base using AIDB SQL directly. If you prefer a UI-driven approach, you can also create knowledge bases through *Pipeline Designer concepts* .

Step 1: Accessing Langflow

1. Log into the HM console and select your project.
2. Select **Launchpad** in the navigation and open **Langflow**.

Step 2: Setting up Global Variables

EDB components read credentials from Langflow Global Variables. Set these up once and reuse them across all components in the flow.

1. In Langflow, go to **Profile → Settings → Global Variables**.
2. Create the following variables:

Variable name	Value
HM_API_KEY	Your HM machine user access key.
HM_DB_USER	Postgres user for your HM cluster.
HM_DB_PASSWORD	Password for the Postgres user.

See Configure Global Variables for instructions.

Step 3: Creating a blank flow

1. On the Langflow home screen, select **New Flow → Blank Flow**.

Step 4: Adding a chat input

1. In the Langflow canvas, select **Inputs** in the component panel.
2. Drag **Chat Input** onto the canvas. This captures the user's question.

Step 5: Adding an EDB Knowledge Base component

1. Select **EDB** in the component panel and drag **EDB Knowledge Base** onto the canvas.
2. Configure the component:

- **HM Machine User Key** → select the `HM_API_KEY` Global Variable.
- **Hybrid Manager Project** → select your project.
- **Hybrid Manager Database** → select the cluster containing your knowledge base.
- **Database Connection Type** → select the appropriate connection type.
- **User for the database** → select the `HM_DB_USER` Global Variable.
- **Password for the database** → select the `HM_DB_PASSWORD` Global Variable.
- **Database Name** → select the database where the knowledge base was created.
- **Knowledge Base** → select `my_knowledge_base` (created in the Prerequisites).

1. Connect **Chat Input → EDB Knowledge Base**. This wires the user's question into the **Query Value** field automatically.

Step 6: Adding a Prompt Template

Without a prompt template, the model receives a raw dump of retrieved chunks alongside the question. A template structures the input into a prompt the model can follow.

1. Select **Prompts** in the component panel and drag **Prompt Template** onto the canvas.
2. In the **Template** field, enter:

```
Answer the question using only the context below. If the answer isnt in the context,␣
↪say you dont know.

Context:
{context}

Question:
{question}
```

1. Connect **EDB Knowledge Base → Prompt Template** (`context` variable).
2. Connect **Chat Input → Prompt Template** (`question` variable).

Step 7: Adding an EDB Model Server component

1. Select **EDB** in the component panel and drag **EDB Model Server** onto the canvas.
2. Configure the component:

- **HM Machine User Key** → select the `HM_API_KEY` Global Variable.
- **Hybrid Manager Model Server Cluster Instance** → select the LLM model cluster deployed in the prerequisites.

1. Connect **Prompt Template → EDB Model Server**.

Step 8: Adding a Chat Output

1. Select **Outputs** in the component panel and drag **Chat Output** onto the canvas.
2. Connect **EDB Model Server → Chat Output**.

Step 9: Running and testing the flow

1. Select **Playground** in the top toolbar to open the interactive test panel.
2. Type a question that relates to your source data and press **Enter**.
3. Verify the response is grounded in your content — you should see the EDB Knowledge Base retrieving relevant rows, the Prompt Template formatting the context and question, and the EDB Model Server generating a contextualised answer.
4. Check that all components show green status indicators after running.

If any component shows an error, select it and review the configuration. Common issues:

- **Can't select a project in the EDB Knowledge Base component** — the portal domain is defaulting to the API endpoint. Select **Controls** on the component and set **Hybrid Manager URL** to `http://istio-internalgateway.istio-system.svc.cluster.local:7080` .
- **Incorrect or expired access key** — verify the `HM_API_KEY` Global Variable holds a valid machine user key.
- **Mismatched database connection** — confirm the database user, password, and database name match what's configured in the cluster.
- **Model cluster not active** — check the cluster status in **Agent Factory → Model Server Clusters**.

For further help, see *Troubleshooting Langflow in Hybrid Manager* .

Next steps

- **Add more components** — extend the flow with an **EDB Database** component to query structured data alongside the knowledge base, or an **EDB Vector Index** component to manage vector indexes. See *EDB components* for the full set, including EDB Airman MCP, EDB Platform, and EDB Embedded Models.
- **Deploy the flow** — publish it as a bundle and deploy it as a managed, callable service with its own endpoints, project-level isolation, and independent scaling. See *Flow deployment* .
- **Monitor the flow's dependencies** — set up Prometheus alert rules and Grafana dashboards for the model cluster and Langflow pods. See *Observability (Agent Factory on HM)* .

EDB components

The **EDB** folder in Langflow contains components that connect flows to resources managed by Hybrid Manager (HM): knowledge bases, databases, vector indexes, embedding and model server clusters, embedded models, and MCP tooling.

Component	Connects flow to	Required roles
EDB Knowledge Base	An AIDB knowledge base in an HM cluster	Project Viewer
EDB Vector Index	Vector indexes on Postgres tables (pgvector)	Project Viewer
EDB Database	A Postgres database in an HM cluster	Project Viewer
EDB Airman MCP	The Airman MCP server's tools	None (requires valid PG connection string)
EDB Platform	HM platform MCP tools for the current user	Depends on operations invoked
EDB Embeddings	An HM-hosted embedding model	AI Model Manager
EDB Model Server	An HM-hosted inference model (LLM or other)	AI Model Manager
EDB Embedded Models	AIDB embedded models for CPU-based inference	Project Viewer

See the parent *Langflow FAQ* page for role requirements and Global Variables setup.

EDB Knowledge Base

The **EDB Knowledge Base** component runs a semantic similarity search against an existing AIDB knowledge base in a Hybrid Manager (HM) cluster, with optional structured filtering on the knowledge base's source-table columns.

Choosing the right component

Use the EDB Knowledge Base component when:

- You're building a RAG flow that retrieves relevant passages by similarity before passing them to a generation step.
- You want an agent tool that looks up information in a knowledge base on demand. The component is tool-mode compatible.
- You want hybrid search: semantic ranking combined with a structured filter on the knowledge base's source-table columns.

Use a different component if:

- You need to manage pgvector index structure (create, drop, rebuild HNSW or IVFFlat indexes) on an existing vector column. Use *EDB Vector Index* .
- You need direct SQL access to a database, not similarity search. Use *EDB Database* .
- The knowledge base doesn't exist yet. Create one through *Pipeline Designer concepts* or by calling `aidb.create_table_knowledge_base()` against the cluster.

Prerequisites

- The Project Viewer role on the project that owns the cluster:

```
-   Project > **Users** > **Assign Project Roles** > **Project Viewer**.
```

- An HM cluster with at least one AIDB knowledge base. Create one through *Pipeline Designer concepts* or directly via `aidb.create_table_knowledge_base()`. The knowledge base's source can be a table (filtering supported) or a volume (semantic-only).
- An HM machine-user access key and the database user/password saved in Langflow as Global Variables .

Inputs

Connection

Field	Type	Required	Default	Notes
Connection Mode	Dropdown	Yes	Hybrid Manager	*Hybrid Manager* or *Direct Connection*. Switches the connection fields that follow.
Hybrid Manager URL	Text	No		Advanced. Override the default HM URL.
HM Machine User Key	Secret	Yes	*HM_API_KEY*	Defaults to the global variable named *HM_API_KEY*.
Hybrid Manager Project	Dropdown	Yes		Populated from your HM projects.
Hybrid Manager Database	Dropdown	Yes		The cluster containing the knowledge base.
Database Group	Dropdown	No		Cluster group, for example primary or reader.
Database Connection Type	Dropdown	Yes		For example, read or read/write.
User for the database	Text	Yes	*HM_DB_USER*	Postgres role. Defaults to the *HM_DB_USER* global variable.
Password for the database	Secret	Yes	*HM_DB_PASSWORD*	Defaults to the *HM_DB_PASSWORD* global variable.
Default database name override	Text	No		Advanced. Override the cluster default when listing databases.
Database Name	Dropdown	Yes		The database holding the AIDB knowledge bases.

In **Direct Connection** mode the HM cluster fields are replaced with **Database Host** and **Database Port**. The user, password, and name fields remain.

Knowledge base and query

Field	Type	Required	Default	Notes
Knowledge Base	Dropdown	Yes		Populated from *aidb.knowledge_bases* in the selected database. Use the refresh button if you've just created one.

continues on next page

Table 11 – continued from previous page

Field	Type	Required	Default	Notes
Limit results	Integer	Yes	*10*	Maximum rows to return.
Query Value	Multiline text	Yes		The natural-language query to search for. Tool-mode compatible: agents call the component with this as the tool argument.

Search mode

Field	Type	Required	Default	Notes
Search Mode	Dropdown	Yes	*Vector Only*	*Vector Only* runs pure semantic search via *aidb.retrieve_**. *Vector + Filter* adds a *WHERE* clause against the knowledge base's source table. Only *Vector Only* is offered when the KB's source is a volume. See *Hybrid search* .

Filter (Vector + Filter only)

These fields appear only when **Search Mode** is **Vector + Filter** and the knowledge base has a table source.

Field	Type	Required	Default	Notes
Filter Column	Dropdown	Yes		Populated from the source table's columns.
Operator	Dropdown	Yes	=	See *Filter operators* .
Filter Value	Text	Yes		Value to compare against. For *IN*, supply a comma-separated list. Ignored for *IS NULL* / *IS NOT NULL*.

Output options

Field	Type	Required	Default	Notes
Include Text	Boolean	No	*false*	Applies in **Vector Only** mode: when on, the result includes the source-text column for each hit (uses *aidb.retrieve_text*); when off, only keys and distances are returned (*aidb.retrieve_key*). Has no effect under **Vector + Filter**, where the source-text column is always returned.
Include Columns	Boolean	No	*true*	Advanced. Include source-table column metadata in the output. Tool-mode compatible.
Add Error	Boolean	No	*false*	Advanced. Surface query errors in the result instead of raising. Tool-mode compatible.

Outputs

Output	Type	Carries
Result Table	DataFrame	One row per match, with *key*, optional *value* (text), and *distance*. Distance is computed using the knowledge base's configured metric (Cosine, L2, or Inner Product).

Hybrid search

Vector + Filter combines semantic similarity with a structured `WHERE` clause against the knowledge base's source table. Use it when you want the top-K semantic matches *within* a subset of your data. For example, "the closest match in documents tagged `kind = 'policy'`".

How it works:

- The component JOINs the KB's vector table with its source table on the source key column.
- The filter is applied in the `WHERE` clause before the order-by-distance.
- For HNSW-indexed vectors the component sets `hnsw.iterative_scan = relaxed_order` so the filtered search still returns useful results when the index would otherwise prune too aggressively.

Hybrid search is only available when the knowledge base's source is a table. If the source is a volume, the **Search Mode** dropdown is restricted to **Vector Only** and the filter fields stay hidden.

Filter operators

Operator	Behavior
=, *!=*, >, <, >=, <=	Standard SQL comparison. Numeric values are passed unquoted. Non-numeric values are quoted.
LIKE	SQL *LIKE* pattern match. Use *%* wildcards in **Filter Value** .
IN	Comma-separated values in **Filter Value** , for example *policy, contract, memo*.
IS NULL, *IS NOT NULL*	**Filter Value** is ignored.

EDB Vector Index

The **EDB Vector Index** component manages pgvector indexes on an existing table. It can create a new index, drop an existing one, rebuild it, or just check whether it exists.

Choosing the right component

Use the EDB Vector Index component when:

- You already have a Postgres table with a `vector` column and want to add (or change) an index to accelerate similarity search.
- You want to switch index type (HNSW vs IVFFlat) or change distance metric on an existing column.
- You want to verify from a flow whether a particular index is in place before running other operations.

Use a different component if:

- You want to query a knowledge base, not manage index structure. Use *EDB Knowledge Base* .
- You want raw SQL access to the database. Use *EDB Database* .
- The table doesn't have a `vector` column yet. The component only manages indexes on existing vector columns. Add the column first (with `aidb` or your own DDL).

Prerequisites

- A Postgres role with the `CREATE INDEX` privilege on the target table.
- An HM cluster with the `pgvector` extension installed and a table containing at least one `vector` column.
- An HM machine-user access key and the database user/password saved in Langflow as Global Variables .

Inputs

Connection

Field	Type	Required	Default	Notes
Hybrid Manager URL	Text	No		Advanced. Override the default HM URL.
HM Machine User Key	Secret	Yes	*HM_API_KEY*	Defaults to the global variable named *HM_API_KEY*.
Hybrid Manager Project	Dropdown	Yes		Populated from your HM projects.

continues on next page

Table 17 – continued from previous page

Field	Type	Required	Default	Notes
Hybrid Manager Database	Dropdown	Yes		The cluster containing the target table.
Database Group	Dropdown	No		Cluster group, for example primary or reader.
Database Connection Type	Dropdown	Yes		For example, read or read/write.
User for the database	Text	Yes	*HM_DB_USER*	Defaults to the *HM_DB_USER* global variable.
Password for the database	Secret	Yes	*HM_DB_PASSWORD*	Defaults to the *HM_DB_PASSWORD* global variable.
Default database name override	Text	No		Advanced. Override the cluster default when listing databases.
Database Name	Dropdown	Yes		Database containing tables with vector columns.

Target

Field	Type	Required	Default	Notes
Table	Dropdown	Yes		Tables in the selected database, excluding system schemas.
Vector Column	Dropdown	Yes		Populated from columns of type *vector* in the selected table.
Distance Operation	Dropdown	Yes	*Cosine*	One of *Cosine*, *L2*, *InnerProduct*. Maps to pgvector operator classes *vector_cosine_ops*, *vector_l2_ops*, *vector_ip_ops*.
Index Type	Dropdown	Yes	*hnsw*	*hnsw* or *ivfflat*.
Operation	Dropdown	Yes	*Create*	One of *Create*, *Drop*, *Rebuild*, *Check Exists*. See *Operations* .

HNSW parameters (Index Type = hnsw)

Field	Type	Required	Default	Notes
HNSW M (Max Connections)	Integer	No	*16*	Advanced. Maximum number of connections per node.
HNSW ef_construction	Integer	No	*64*	Advanced. Size of the dynamic candidate list during index construction.

IVFFlat parameters (Index Type = ivfflat)

Field	Type	Required	Default	Notes
IVFFlat Lists	Integer	No	*100*	Advanced. Number of inverted lists.

Outputs

Output	Type	Carries
Result	Data	Operation outcome. Always includes *success*, *operation*, *table*, *column*, *index_name*, and *message*. *Check Exists* adds *exists*. *Create* adds *index_type* and the generated *sql* when it builds a new index, and *already_existed* when an index was already present (set both on an idempotent same-type match and on a different-type conflict).

Operations

Operation	Behavior
Create	Creates the index if it doesn't exist. Idempotent: if an index of the same type already exists on this column, the operation returns success without re-creating. If an index of a different type exists on the column, the operation fails and asks you to *Drop* it first.
Drop	Drops the index. Idempotent (uses *DROP INDEX IF EXISTS*).
Rebuild	Runs *REINDEX INDEX* on the existing index. Fails if the index doesn't exist.
Check Exists	Returns *exists: true/false*. Does not modify the database.

Index naming

The component generates a stable, predictable index name from the table and column: `idx_<table>_<column>_<index_type>` (without schema prefix). Choose your indexes through this component if you want that naming convention. Otherwise, manage them in SQL.

EDB Database

The **EDB Database** component resolves an HM project, cluster, and connection type to a Postgres connection URL and exposes that URL as an output. Downstream components (SQL query, MCP-tool wrappers, custom code) use the URL to connect.

Choosing the right component

Use the EDB Database component when:

- You need a Postgres connection URL for a database in an HM cluster, derived from your HM project and connection settings.
- You want to wire that URL into another component that connects to the database directly (an *EDB Airman MCP* for SQL-as-tools, or a custom flow component).

Use a different component if:

- You want similarity search against an existing knowledge base. Use *EDB Knowledge Base* .
- You want to manage pgvector index structure. Use *EDB Vector Index* .
- You want SQL as MCP tools for an agent. Use *EDB Airman MCP* (it accepts the URL output from this component).

Prerequisites

- The Project Viewer role on the project that owns the cluster:
- Project > **Users** > **Assign Project Roles** > **Project Viewer**.
- An HM cluster the user has access to. See *Create clusters* .
- An HM machine-user access key and the database user/password saved in Langflow as Global Variables .

Inputs

Field	Type	Required	Default	Notes
Hybrid Manager URL	Text	No		Advanced. Override the default HM URL.
HM Machine User Key	Secret	Yes	*HM_API_KEY*	Defaults to the global variable named *HM_API_KEY*.
Hybrid Manager Project	Dropdown	Yes		Populated from your HM projects.
Hybrid Manager Database	Dropdown	Yes		The cluster containing the database.
Database Group	Dropdown	No		Cluster group, for example primary or reader.
Database Connection Type	Dropdown	Yes		For example, read or read/write.
User for the database	Text	Yes	*HM_DB_USER*	Defaults to the *HM_DB_USER* global variable.
Password for the database	Secret	Yes	*HM_DB_PASSWORD*	Defaults to the *HM_DB_PASSWORD* global variable.
Default database name override	Text	No		Advanced. Override the cluster default when listing databases.
Database Name	Dropdown	No		The database to connect to. Leave blank to use the user's default.

Note

If you can't select a project because the portal domain defaults to the API, select **Controls** at the top of the EDB Database and Knowledge Base component to configure *http://istio-internalgateway.istio-system.svc.cluster.local:7080* as the **Hybrid Manager URL**.

Output	Type	Carries
Database URL	Message	The fully resolved Postgres connection string (*postgresql://user:password@host:port/database*), URL-encoded as needed. Pass this into components that take a database URL, such as *EDB Airman MCP* .

EDB Airman MCP

The **EDB Airman MCP** component starts a Postgres MCP server backed by

pg-airman-mcp and surfaces its tools to an agent. The agent can then call those tools to read from (or, in unrestricted mode, write to) the database without you wiring every individual SQL operation by hand.

Choosing the right component

Use the EDB Airman MCP component when:

- You want an agent to be able to interrogate a database during a flow run (look up a row, summarize a table, answer a question by querying).
- You want safe-by-default behavior: the agent only runs read-only queries unless you explicitly opt into unrestricted access.
- You want each flow execution to track back to the calling HM session for audit.

Use a different component if:

- You need a fixed database connection to pass into other components (not a tool for an agent). Use *EDB Database* .
- You want platform-level operations (projects, clusters, knowledge bases) as agent tools rather than SQL. Use *EDB Platform* .
- You want similarity search against a knowledge base, not arbitrary SQL. Use *EDB Knowledge Base* .

Prerequisites

- The Project Viewer role on the project that owns the database cluster.
- A reachable Postgres database URL (from an *EDB Database* component's `Database URL` output, or supplied directly).
- An agent in the flow that consumes **Tools** (the component's only output is a Tool).

Inputs

Field	Type	Required	Default	Notes
Database URL	Text	Yes		A Postgres connection string. Typically wired from the *Database URL* output of an *EDB Database* component.
Purpose	Text	No		Optional free-text hint passed to the MCP server as *AIRMAN_MCP_PURPOSE*. Use it to describe what the agent should do with the database.
Unrestricted Access Mode	Boolean	No	*false*	Advanced. When off (default), the MCP server runs in restricted mode and only safe read-only queries are permitted. When on, the agent can execute any SQL the database role allows.
MCP Server Name	Text	No		Advanced. Override the registered MCP server name. By default the name is derived from the flow node ID so it stays stable across builds.

Outputs

Output	Type	Carries
Toolset	Tool	The MCP tools exposed by *pg-airman-mcp* for the configured database. Connect to an agent's Tools input.

Access mode

The **Unrestricted Access Mode** toggle controls what the agent can do:

Mode	Behavior
Restricted (default)	The MCP server only accepts safe read-only queries. Mutating statements (*INSERT*, *UPDATE*, *DELETE*, DDL) are rejected at the MCP layer regardless of what the database role would otherwise allow.
Unrestricted	The MCP server accepts any SQL the connected database role can run. The role's own privileges become the only ceiling.

Restricted mode is the safe default. Turn it off only when the agent has a legitimate reason to write, and pair it with a database role that has the minimum privileges needed.

Session propagation

When the component runs inside an HM-deployed flow, the calling user's HM session ID is forwarded to the MCP server with each tool call. Tool calls are traceable back to the originating session, which is useful for audit and for correlating MCP activity with HM gateway logs.

EDB Platform

The **EDB Platform** component connects a flow to the Hybrid Manager (HM) platform's MCP tool surface. It surfaces HM's platform-level operations to an agent on behalf of the calling user, so the agent can list, inspect, and act on the user's HM resources.

Choosing the right component

Use the EDB Platform component when:

- You want an agent to be able to look up information about the caller's HM environment (projects, clusters, knowledge bases) during a flow run.
- You want platform operations available as MCP tools, scoped to the current HM user's permissions.

Use a different component if:

- You want the agent to execute SQL against a specific Postgres database. Use *EDB Airman MCP* .
- You want a knowledge base similarity search, not platform navigation. Use *EDB Knowledge Base* .

Prerequisites

- The identity running the flow must have the HM permissions for whichever platform operations the agent invokes. In the flow builder this is the logged-in user; in a deployed flow it is the caller's identity (user or machine user). Tool calls execute with that identity's permissions.

Inputs

Field	Type	Required	Default	Notes
Tool	Dropdown	Yes		Populated from the HM platform MCP server when you select the refresh button. Select the specific tool to invoke. Additional fields appear depending on the tool's arguments.

When you select a **Tool**, the component reads the tool's input schema from the MCP proxy and renders one field per argument under the dropdown. Field types and labels come from the tool definition.

Outputs

Output	Type	Carries
Response	DataFrame	The selected tool's result, parsed as a DataFrame. JSON outputs are flattened to columns. Errors are returned as a row with an *error* column rather than raised.

When the component is used as a tool provider for an agent (instead of being invoked directly), every tool the proxy exposes is registered with the agent and the **Tool** dropdown is not consulted.

Notes

- Tools are fetched from a running HM-side MCP proxy. Available tools and their argument schemas can change as HM platform features evolve.
- Each tool call carries the calling user's HM identity and session, so platform-side audit logs attribute the action to the user, not to a shared service account.

EDB Embeddings

The **EDB Embeddings** component connects a flow to an embedding model running on a Hybrid Manager (HM) model server cluster, and produces an Embeddings model object that downstream components (vector stores, retrievers) can use to encode text.

Choosing the right component

Use the EDB Embeddings component when:

- You want vector embeddings produced by a model running on an HM-hosted model server cluster (OpenAI-compatible or NVIDIA NIM).
- You're feeding a vector store, retriever, or similarity-search component that expects an Embeddings model.

Use a different component if:

- You want CPU-based embeddings produced locally on the database cluster using `aidb` . Use *EDB Embedded Models* .
- You want generative responses from a model, not embeddings. Use *EDB Model Server* .

Prerequisites

- An HM model server cluster with an embedding model deployed. See *Model Management on Hybrid Manager* for how to create one. A *GPU Recommendations for Default NIM Models* is typically required.
- An HM machine-user access key saved in Langflow as a Global Variable (default name `HM_API_KEY`).

Inputs

Connection

Field	Type	Required	Default	Notes
Hybrid Manager URL	Text	No		Advanced. Override the default HM URL.
HM Machine User Key	Secret	Yes	*HM_API_KEY*	Defaults to the global variable named *HM_API_KEY*.
Hybrid Manager Model Server Cluster Instance	Dropdown	Yes		The model server cluster to call. Populated from your HM model clusters.
External Ingress	Boolean	No		Advanced. Route through the external ingress instead of the in-cluster service.

Model

Field	Type	Required	Default	Notes
API Client	Dropdown	No	*OpenAI*	Advanced. *OpenAI* for the EDB Model Server's OpenAI-compatible API. *NVIDIA* only when connecting to a native NIM endpoint that needs NVIDIA-specific embedding features.
Model	Dropdown	Yes		Populated from the selected model server cluster. Use the refresh button if the list is empty.

API Client = OpenAI (default)

Field	Type	Required	Default	Notes
Dimensions	Integer	No		Advanced. Number of dimensions for the output vectors. Only supported by certain models. Leave blank to use the model's default.
Max Retries	Integer	No	*3*	Advanced. Maximum retries on a failed embedding call.
Chunk Size	Integer	No	*1000*	Advanced. Maximum texts to send in a single batch.
Model Kwargs	Dict	No		Advanced. Additional keyword arguments passed through to the OpenAI embeddings client.

API Client = NVIDIA

Field	Type	Required	Default	Notes
Model Temperature	Float	No	*0.1*	Advanced. Model temperature.

The OpenAI-specific fields hide when **API Client** is `NVIDIA` , and vice versa. Field values are preserved when you switch between clients, so a round-trip doesn't wipe your settings.

Outputs

Output	Type	Carries
Embedding Model	Embeddings	A LangChain *Embeddings* instance pointing at the selected HM model server cluster. Pass this into a vector store or retriever.

EDB Model Server

The **EDB Model Server** component connects a flow to an inference model running on a Hybrid Manager (HM) model server cluster. Use it for chat-style language models, reasoning models, and other inference endpoints that you've deployed to HM.

Choosing the right component

Use the EDB Model Server component when:

- You want a language-model response in your flow from a model running on an HM-hosted model server cluster.
- You want a `LanguageModel` object to plug into an agent or downstream chain.
- You're using an OpenAI-compatible endpoint (the default) or a native NVIDIA NIM endpoint.

Use a different component if:

- You want embeddings rather than text responses. Use *EDB Embeddings* .
- You want CPU-based inference running on the database cluster rather than on a model server. Use *EDB Embedded Models* .

Prerequisites

- An HM model server cluster with the target model deployed. See *Model Management on Hybrid Manager* for how to create one. A *GPU Recommendations for Default NIM Models* is typically required.
- An HM machine-user access key saved in Langflow as a Global Variable (default name `HM_API_KEY`).

Inputs

Prompt

Field	Type	Required	Default	Notes
Input	Message	No		The user message to send to the model. Connect from upstream Chat Input or other text-producing components.
System Message	Multiline text	No		System prompt passed to the model.
Stream	Boolean	No	*false*	Advanced. Stream the model's response tokens as they arrive. Only supported by models that expose a streaming endpoint.

Connection

Field	Type	Required	Default	Notes
Hybrid Manager URL	Text	No		Advanced. Override the default HM URL.
HM Machine User Key	Secret	Yes	*HM_API_KEY*	Defaults to the global variable named *HM_API_KEY*.
Hybrid Manager Model Server Cluster Instance	Dropdown	Yes		The model server cluster to call. Populated from your HM model clusters.
External Ingress	Boolean	No		Advanced. Route through the external ingress instead of the in-cluster service.

Model

Field	Type	Required	Default	Notes
API Client	Dropdown	No	*OpenAI*	Advanced. *OpenAI* is required for correct tool-calling behavior in Agent flows. *NVIDIA* only when connecting to a native NIM endpoint that needs NVIDIA-specific features like detailed thinking.
Model Name	Dropdown	Yes		Populated from the selected model server cluster. Use the refresh button if the list is empty.

continues on next page

Table 37 – continued from previous page

Field	Type	Required	Default	Notes
Temperature	Slider	No	*0.1*	Controls randomness. Range *0–1*, step *0.01*.
Max Tokens	Integer	No		Advanced. Maximum tokens to generate. Range *0–128000*. Set to *0* for unlimited.
Seed	Integer	No	*1*	Advanced. Controls reproducibility.
Default model query	Text	No		Advanced. Sent when no input is connected to the component. Useful for testing a model that does not accept empty input.

API Client = OpenAI (default)

Field	Type	Required	Default	Notes
Model Kwargs	Dict	No		Advanced. Additional keyword arguments passed to the OpenAI chat client.
JSON Mode	Boolean	No	*false*	Advanced. Force the model to return JSON.
Max Retries	Integer	No	*5*	Advanced. Maximum retries on a failed request.
Timeout	Integer	No	*700*	Advanced. Per-request timeout in seconds.

API Client = NVIDIA

Field	Type	Required	Default	Notes
Detailed Thinking	Boolean	No	*false*	Advanced. Return the model's detailed thought process. Only supported by NVIDIA reasoning models.

The OpenAI-specific fields hide when **API Client** is `NVIDIA` , and vice versa. Field values are preserved when you switch between clients.

Outputs

Output	Type	Carries
Model Response	Message	The model's text response to the input prompt.

continues on next page

Table 40 – continued from previous page

Output	Type	Carries
Language Model	LanguageModel	A LangChain *LanguageModel* instance pointing at the selected HM model server cluster. Pass this into an agent or chain that needs a model handle.

EDB Embedded Models

The **EDB Embedded Models** component runs inference inside a Postgres cluster using the AIDB extension's built-in models. No external inference service is involved: embeddings and small-model text generation execute on the database CPU, producing data that's already where you want it for downstream Postgres operations.

Choosing the right component

Use the EDB Embedded Models component when:

- You want text or image embeddings produced locally on the cluster, with no external network call.
- You want small-model text generation (T5, Llama-Instruct, Qwen2, SmolLM2) close to the data, for tasks that don't need a large frontier model.
- You're building a flow that combines a single embedding step with vector storage in the same cluster.

Use a different component if:

- You need a Hybrid Manager-hosted large-model inference endpoint. Use *EDB Model Server* or *EDB Embeddings* .
- You want to retrieve from an existing knowledge base rather than compute new embeddings. Use *EDB Knowledge Base* .

Prerequisites

- An HM cluster with the aidb extension installed.
- At least one supported embedded model registered in `aidb.list_models()` . Supported providers: `bert_local` , `clip_local` , `t5_local` , `llama_instruct_local` , `qwen2_local` , `smollm2_local` .
- An HM machine-user access key and the database user/password saved in Langflow as Global Variables .

Inputs

Connection

Field	Type	Required	Default	Notes
Connection Mode	Dropdown	Yes	Hybrid Manager	*Hybrid Manager* or *Direct Connection*. Switches the connection fields that follow.
Hybrid Manager URL	Text	No		Advanced. Override the default HM URL.

continues on next page

Table 41 – continued from previous page

Field	Type	Required	Default	Notes
HM Machine User Key	Secret	Yes	*HM_API_KEY*	Defaults to the global variable named *HM_API_KEY*.
Hybrid Manager Project	Dropdown	Yes		Populated from your HM projects.
Hybrid Manager Database	Dropdown	Yes		The cluster running *aidb*.
Database Group	Dropdown	No		Cluster group, for example primary or reader.
Database Connection Type	Dropdown	Yes		For example, read or read/write.
User for the database	Text	Yes	*HM_DB_USER*	Defaults to the *HM_DB_USER* global variable.
Password for the database	Secret	Yes	*HM_DB_PASSWORD*	Defaults to the *HM_DB_PASSWORD* global variable.
Default database name override	Text	No		Advanced. Override the cluster default when listing databases.
Database Name	Dropdown	Yes		The database with *aidb* installed.

In **Direct Connection** mode the HM cluster fields are replaced with **Database Host** and **Database Port**.

Model selection

Field	Type	Required	Default	Notes
Model	Dropdown	Yes		Populated from *aidb.list_models()* against the selected database. Use the refresh button if you've just registered a model.
Model Capability	Dropdown	Yes		Available capabilities depend on the selected model. See *Capabilities* .

Capability-specific inputs

The fields that follow the capability dropdown depend on what **Model Capability** is set to.

When **Model Capability** is **Text Embedding**:

Field	Type	Required	Default	Notes
Text Input	Message/Data	No		Text to embed. If empty, the component outputs the embeddings model itself (useful for wiring into a vector store, see *Outputs*).

When **Model Capability** is **Text Generation**:

Field	Type	Required	Default	Notes
Prompt	Message	Yes		The prompt to send to the model.
Temperature	Slider	No	*0.1*	Advanced. Controls randomness. Range *0–1*.
Max Tokens	Integer	No		Advanced. Maximum tokens to generate. Range *1–4096*.

When **Model Capability** is **Image Embedding**:

Field	Type	Required	Default	Notes
Image	Data/Message	Yes		The image to embed. Connect a File component or upstream image-producing component.

Outputs

Outputs vary by capability:

Capability	Output	Type	Carries
Text Embedding (with text supplied)	**Embedding**	Data	*{input, embedding, model, embedding_dim}* for the supplied text.
Text Embedding (text empty)	**Embedding**	Data	The embeddings model itself, suitable for passing to a vector store component.
Image Embedding	**Embedding**	Data	*{input, embedding, model, embedding_dim}* for the supplied image.
Text Generation	**Response**	Message	The model's generated text.

Capabilities

The **Model Capability** dropdown lists only the capabilities of the selected model. Mapping of the built-in models:

Model	Capabilities
bert_local	Text Embedding

continues on next page

Table 47 – continued from previous page

Model	Capabilities
clip_local	Text Embedding, Image Embedding
t5_local	Text Embedding, Text Generation
llama_instruct_local	Text Generation
qwen2_local	Text Generation
smollm2_local	Text Generation

If a model has a single capability the dropdown is hidden and the capability is selected automatically.

Note

- Inference runs on the database cluster's CPU, not on the HM control plane and not on a model server cluster. Large workloads will compete with regular database queries on that cluster's CPU and memory. - The component uses *aidb.encode_text* , *aidb.encode_text_batch* (batch embedding on the vector-store output path, when **Text Input** is empty), *aidb.encode_image* , and *aidb.decode_text* under the hood. See the AIDB documentation for details on each function.

Using models in Langflow

How models are consumed in Langflow

In Hybrid Manager, Langflow connects to AI models through two EDB components: **EDB Model Server** and **EDB Embeddings**. Both components communicate with model endpoints that are governed and served by Hybrid Manager — no external API calls are made by default.

- **EDB Model Server** — sends prompts to a deployed LLM and returns generated responses. Use this for conversational steps, summarisation, and reasoning within a flow.
- **EDB Embeddings** — converts text into vector representations using a deployed embedding model. Use this when building retrieval steps or populating knowledge bases from within a flow.

Both components require a model cluster to be deployed in your HM project before use. See *Model Management on Hybrid Manager* for setup steps.

Using the EDB Model Server component

1. In Langflow, select **EDB** in the left-side component panel.
2. Drag **EDB Model Server** onto the canvas.
3. In the component, set **HM Machine User Key** to your access key Global Variable.
4. Under **Hybrid Manager Model Server Cluster Instance**, select the deployed model cluster you want to use.
5. Connect the component's input to your flow's upstream step (for example, a **Chat Input** or a retrieval result).
6. Connect the output to a **Chat Output** or the next step in the flow.
7. Select the **Play** icon to run the component. Select **Model Response** in the lower-right corner to inspect the output.

Models available in the dropdown are those deployed from the Model Library into your HM project. Hybrid Manager manages their internal endpoints — you don't need to configure URLs manually.

Using the EDB Embeddings component

1. In Langflow, select **EDB** in the left-side component panel.
2. Drag **EDB Embeddings** onto the canvas.
3. Set **HM Machine User Key** to your access key Global Variable.
4. Under **Hybrid Manager Model Server Cluster Instance**, select the embedding model cluster.
5. Connect the component's **Embedding Model** output to a downstream Langflow vector store or retriever component that expects an embeddings model.

EDB Embeddings produces a LangChain `Embeddings` model object, not a vector directly. Downstream components use it to encode text at query or indexing time. It has no text input of its own.

Knowledge bases and retrieval

To ground model responses in your organisation's data, combine the EDB Model Server with the **EDB Knowledge Base** component:

1. Set up a knowledge base in your HM project using AIDB SQL or *Pipeline Designer concepts* .
2. Add an **EDB Knowledge Base** component to the flow and configure it with your production project, cluster, and knowledge base name.
3. Add a **Prompt Template** component between the EDB Knowledge Base and the EDB Model Server. Connect the EDB Knowledge Base output to the template's context variable, and the user's query to its question variable. Connect the template output to the EDB Model Server input.

This pattern implements retrieval-augmented generation (RAG) entirely within your HM cluster — source documents, embeddings, and model calls all stay inside your environment.

See *EDB Knowledge Base* for the full configuration walkthrough.

Environment and service discovery

When Langflow runs inside Hybrid Manager:

- Model endpoints are automatically discovered from the HM Model Library — no external URLs to configure.
- Authentication is handled through the HM Machine User Key you set as a Global Variable.
- All traffic between Langflow, model server pods, and Postgres clusters stays within the project namespace.

If a component can't resolve the project because the portal domain defaults to the API endpoint, select **Controls** on the component and set **Hybrid Manager URL** to `http://istio-internalgateway.istio-system.svc.cluster.local:7080` .

Flow deployment

Hybrid Manager (HM) extends Langflow with a deployment lifecycle that turns flows you've built in the editor into long-running, callable services. A flow moves through three stages:

1. ** *Publishing bundles* **: package one or more flows from a Langflow project into an immutable bundle in the HM Asset Library.
2. ** *Deploying flows* **: instantiate a published bundle as a managed workload in an HM project, with its own parameter values, resource limits, and endpoints.
3. ** *Accessing deployed flows* **: invoke the flow over HTTP or MCP. Access is authenticated by the same HM machine-user access keys used elsewhere in HM.

Flows in the Langflow editor are private to the user who built them and run in a shared editor environment. Compared to that, a deployed flow has:

- A stable HTTP endpoint and an MCP server endpoint, both routed through the HM gateway.
- Its own parameter values, supplied at deploy time independently of whatever the builder used in the editor. Any parameter values that are secrets (passwords, API keys, or tokens the flow uses to reach databases, model APIs, or other downstream resources) are stored as Kubernetes secrets in the deployment's namespace.
- An independent lifecycle (deploy, update, undeploy) that doesn't require opening the editor.
- Versioning: each deployment is created from an immutable bundle version. Updating the bundle creates a new version. The deployment continues to run the version it was deployed from until you update it.
- Access control: a deployed flow can be made available globally or per-project, with access limited to a specific set of Hybrid Manager projects.

Concepts

Term	Meaning
Published flow	A versioned bundle of one or more Langflow flows in the HM Asset Library. Visible at **Asset Library > Published Flows** (*/asset-library/published-flows*). Immutable per version.
Bundle	Synonym for "published flow" when emphasizing that it can contain multiple flows.
Parameter	A value in the flow that uses a Langflow *Step 2: Setting up Global Variables* . Parameters are deduplicated across all flows in a bundle by variable name.
Credential	A parameter whose input is a Langflow secret field (*SecretStrInput*), typically a password, API key, or token the flow uses to reach a downstream resource such as a database or model API. Not a credential for accessing the deployed flow itself (that is HM machine-user authentication, see *Accessing deployed flows*). Credentials never carry defaults at publish time. Their values are supplied at deploy time and stored as Kubernetes secrets.
Access level	Either **project** (visible only to users with access to one of a chosen list of HM projects) or **global** (visible to all projects). Configured at publish time, editable later.
Deployed flow	A running instantiation of a published flow in an HM project. Visible at **Estate > Flows** (*/estate/flows*). Each deployed flow is isolated in its own Kubernetes namespace.
Runtime	The Langflow runtime container that hosts a deployed flow. Exposes an HTTP API and an MCP server, both routed through the HM gateway.

Required roles

Permissions for the publish/deploy lifecycle are defined in *User role permissions by use case* :

Action	Required role
View published flows	Project Viewer (project-scoped) or any organization role (global)
Publish a flow / create new version	Project Admin or Project Member, for each project granted access
Edit a published flow (defaults, access list)	Project Admin or Project Member
Delete a published flow version	Organization Admin
Deploy a published flow	Project Admin or Project Member

continues on next page

Table 49 – continued from previous page

Action	Required role
View deployed flows	Project Viewer
Edit / undeploy a deployed flow	Project Admin or Project Member

What's next

- *Publishing bundles* : Package flows from Langflow into an Asset Library bundle.
- *Deploying flows* : Turn a published bundle into a running deployment with its own parameter values and resources.
- *Accessing deployed flows* : Invoke a deployed flow over HTTP or MCP.

Publishing bundles

A **bundle** (also called a "published flow") is an immutable package of one or more Langflow flows in the HM Asset Library, identified by a name and a version. Publishing is the first step in the flow deployment lifecycle .

Opening the publish dialog

The publish dialog has two entry points, both opening the same three-step sidebar:

- **From the Langflow editor.** While editing a flow, select **Share > Publish to Asset Library**. The HM UI opens with the current flow already selected in step 1.
- **From the Asset Library.** Navigate to **Asset Library > Published Flows** in the HM console and select **+ Add New Flow**. No flows are pre-selected.

Publish dialog

Step 1: Select flows

1. The sidebar opens with a checklist of every flow in the current Langflow project. Each row shows the flow name and a checkbox on the left.
2. If you opened the dialog from the Langflow editor, the flow you were editing is already ticked. Confirm it is the flow you want to publish.
3. To include additional flows in the same bundle, tick their checkboxes. A bundle can contain multiple flows — they are all deployed together as a single unit.
4. To remove a flow from the selection, untick its checkbox.
5. Confirm that at least one flow is selected. The **Next** button remains disabled until at least one flow is ticked.
6. Select **Next** to proceed to step 2.

> **Note**
>
> When multiple flows in a bundle reference the same Langflow global variable, they share a single parameter in the published bundle. Any default you set for that parameter applies to every flow that uses it.

Step 2: Configure

Field	Required	Notes
Name	Yes	Up to 64 characters. Used as the display name in the Asset Library and as the basis for the deployed flow's identity.
Description	No	Free-text. Shown in the published-flow detail view.
Version Tag	Yes	Up to 128 characters. An opaque string — HM does not interpret it as semver. Avoid using *latest* as a literal tag; HM reserves it as an alias that always resolves to the most recently published version. Pick a scheme (*1.0.0*, *2026-05-28*, *rc1*) and use it consistently.
Parameters	Conditional	One entry per deduplicated global variable. See *Parameters and credentials* .
Access Level	Yes	**Project** (default) or **Global** . See *Access level* .

Note that **Name** is immutable after publishing — to rename, publish a new version. When all required fields are filled, select **Next** to review before publishing.

Step 3: Review

The review step shows everything you've configured. Selecting **Publish** writes the bundle to the Asset Library and returns you to the published-flow list. The new bundle is the first version. Subsequent publishes with the same name create new versions of the same bundle.

Parameters and credentials

Parameters are the values in your flow that use a Langflow global variable . HM extracts them from each flow's definition at publish time and presents them in step 2 of the dialog.

How a parameter is treated depends on its Langflow input type:

Langflow input type	Treated as	Default at publish time	Value at deploy time
SecretStrInput	**Credential**	Not allowed. The input is read-only in the publish dialog.	**Required** . Supplied per deployment, stored as a Kubernetes secret in the deployment's namespace.
Anything else	**Non-credential parameter**	Optional default value	If no default was set and no value is supplied at deploy time, the flow runs without one.

Credential defaults are never persisted, even if you supply them via the API. Hybrid Manager strips any defaults for credential parameters at publish time. This keeps the bundle safe to share across projects.

Deduplication

Parameters are deduplicated across the bundle by global-variable name. If two flows in the bundle each reference `DB_USER` , the publish dialog shows a single `DB_USER` parameter, and any default you set is reused by both flows. The bundle records which flows reference each parameter, so the deployment can wire the same value into every consumer.

Access level

The access level controls who can see the bundle in the Asset Library and who can deploy it.

- **Project** (default). You must select at least one project. The bundle is visible to users with the appropriate role in any of the listed projects. Promoting a project bundle to global, or moving it between projects, requires that you have access to every project currently on the list.
- **Global**. Visible to everyone in the organization. Use sparingly: once global, anyone with the deploy role on any project can instantiate the bundle.

Access is enforced server-side: lists and reads only return bundles whose `allowed_projects` overlap the requesting user's authorized projects, or whose access level is global.

Editing a published bundle

From **Asset Library > Published Flows**, select the pencil icon on the row for the version you want to change. The edit sidebar opens. The following are editable:

- **Description**
- **Access level** (project ↔ global) and the **allowed projects** list
- **Parameter defaults** for non-credential parameters

The following are immutable for a given version:

- **Name**
- **Version tag**
- **The set of flows in the bundle and their definitions**

To change immutable fields, publish a new version of the bundle.

Deleting a published bundle

Deleting a bundle version removes it from the Asset Library. Existing deployments created from the version continue to run: they hold a downloaded copy of the bundle and do not re-fetch unless the deployment is explicitly updated. Deletion is a permissioned operation (see *Required roles*).

To turn a published bundle into a running deployment, see *Deploying flows* .

Deploying flows

Deploying turns a *Publishing bundles* into a running Langflow runtime hosted in a Hybrid Manager (HM) project. Each deployment is an independent workload with its own parameter values, resource configuration, and endpoints.

Starting a deployment

Navigate to **Asset Library > Published Flows**. In the row for the bundle you want to deploy, select the **Deploy** icon. The Deploy dialog opens.

You can also reach the same action from **Estate > Flows** by selecting **+ Add Flow** and choosing the bundle.

Deploy dialog

The dialog has four steps.

Step 1: Configure

Field	Required	Notes
Display Name	Yes	Up to 63 characters. Used as the deployed flow's name in the Estate UI and in Kubernetes resource names.
Project	Yes	A single project. The drop-down lists only projects where you have permission to create deployed flows. A deployed flow lives in exactly one project.
Network Access Type	Yes	Public or private. Determines whether the HM gateway routes for the flow are reachable from outside the HM network.
Runtime Configuration	Yes	Settings that apply to the Langflow runtime itself, separate from your flow's parameters (for example, service-level configuration).

Note that a deployment belongs to exactly one project and cannot be moved after creation.

Select **Next** to proceed to step 2.

Step 2: Credentials

Every credential parameter in the bundle (any parameter whose Langflow input type is `SecretStrInput`) is listed here. Each must be set before the deployment can proceed.

Credential values are stored as a Kubernetes Secret in the deployment's namespace and mounted as environment variables into the runtime container. They are not visible after deployment. The Estate UI redacts them.

Non-credential parameters that were given defaults at publish time inherit those defaults, but can be overridden here.

Enter the production value for each credential field — the field name matches the Langflow Global Variable name used in the flow (for example, `HM_DB_PASSWORD` or `HM_API_KEY`). The **Next** button remains disabled until every required field has a value.

Select **Next** to proceed to step 3.

Step 3: Task Configuration

Set the deployment's resource constraints. The fields surface a subset of the underlying Kubernetes resource model:

Setting	Default	Range
Replicas	1	1–5
CPU request / limit	500m request, 1000m limit	Configurable
Memory request / limit	512Mi request, 1Gi limit	Configurable

For production workloads, set **Replicas** to at least `2` so the flow stays available if a pod restarts. If unsure on CPU and memory values, start with the defaults and adjust based on observed usage in **Estate > Flows**.

Select **Next** to proceed to the review step.

Step 4: Review

A summary of every setting from steps 1–3, with passwords masked. Before selecting **Deploy**, confirm that every credential field shows a masked value — a blank field means the credential was not filled in step 2 and the deployment will fail to start. Use **Back** to return and fill any missing values.

Selecting **Deploy** submits the deployment. Credential values are written to a Kubernetes Secret immediately and the workload begins reconciling. The new deployment appears in **Estate > Flows** with status **Provisioning**. Provisioning typically completes within a few minutes; once the status changes to **Ready**, the flow's HTTP and MCP endpoints are live.

Understanding deployment resources

Each deployment provisions a set of Kubernetes resources in its own isolated namespace:

Resource	Purpose
Deployment	Runs the Langflow runtime container. Replicated per the configured replica count.
Service	Exposes the runtime's HTTP API and MCP server inside the cluster. The HM gateway routes external traffic to this Service.
Secret	Holds all parameter values (credentials and non-credentials). Mounted into the runtime container as environment variables.
ServiceAccount	Pod identity used for secure communication with internal HM services.
NetworkPolicy	Restricts pod egress to DNS, the HM services the runtime needs at startup, public HTTPS (port 443), and Postgres (port 5432).

Note

Flows that need to reach hosts on protocols or ports outside the NetworkPolicy's allowlist (DNS, HTTPS/443, Postgres/5432) will fail to connect. Contact your HM administrator if your flow requires additional egress.

At pod startup the runtime downloads the bundle (pinned to the version the deployment was created from) before the HTTP and MCP endpoints become ready. Cold start therefore has a small delay. Subsequent restarts repeat the same download step.

Managing deployments

- **Edit deployment parameters.** From **Estate > Flows**, open the deployed flow and edit the parameter values. Saving rewrites the deployment's parameter Secret and triggers a rolling restart of the runtime pods. The runtime re-downloads the bundle at startup (still pinned to the version the deployment was created from).

- **Update the bundle version.** A deployed flow is pinned to the version it was deployed from. To run a newer version of the same bundle, edit the deployment and change the version, or undeploy and deploy again from the new version. Existing deployments are not auto-upgraded when a new bundle version is published.
- **Undeploy.** From the Estate UI, delete the deployed flow. HM deletes the deployment's resources, including the Secret holding its parameter values. The HM gateway routes for the flow are removed.

To invoke the deployed flow over HTTP or MCP, see *Accessing deployed flows* .

Accessing deployed flows

A *Deploying flows* exposes two endpoints, a Langflow runtime API and an MCP server, both routed through the Hybrid Manager (HM) gateway. Clients authenticate to the gateway with an HM machine-user access key.

Finding the endpoints

1. In the HM console, go to **Estate > Flows** (`/estate/flows`).
2. Wait until the **Status** column shows **Ready**. Earlier statuses such as **Provisioning** or **Starting** mean the runtime is not yet accepting requests.
3. Select the row to open the detail sidebar.
4. Scroll to the **Endpoints** section and copy the URL for the endpoint you need.

> **Note**
>
> Do not manually construct these URLs — the route IDs are assigned by HM at deploy time and are not predictable.

Endpoint label	Purpose
Langflow	The Langflow runtime API for listing and running flows in the bundle.
MCP	Model Context Protocol server endpoint. Used by MCP-capable clients (LLM agents, IDEs) to discover and invoke flows as tools.

Each route has its own route ID under the same project and app:

```
https://<hm-portal-host>/projects/<project-id>/apps/<app-id>/routes/<route-id>
```

For example, a single deployed flow might expose:

```
https://<hm-portal-host>/projects/prj_abc/apps/app-xyz/routes/a3528a9d   # HTTP␣
↪runtime
https://<hm-portal-host>/projects/prj_abc/apps/app-xyz/routes/538275d3   # MCP server
```

`<project-id>` , `<app-id>` , and the two `<route-id>` values are assigned by HM at deploy time and are stable for the lifetime of the deployment.

Authentication

Every request to a deployed flow must include an HM machine-user access key in the `x-access-key` header. The HM gateway enforces this. The runtime pod itself does not handle auth.

```
curl -L https://<hm-portal-host>/projects/<project-id>/apps/<app-id>/routes/<route-id>
↪ \
  -H x-access-key: <your-machine-user-key>
```

To create an access key, see *Create a machine user* . The machine user must have the appropriate role in the deployed flow's project. See *Required roles* .

HTTP runtime API

A `GET` against the root of the HTTP runtime route returns a plain-text index of the deployed bundle and the available endpoints:

```
Welcome to Langflow Runtime
Loaded flows:
  - My Flow (4944038b-f66c-4358-bb31-0f0b5bb865fd)
  - Another Flow (5cb0ac73-8de9-4f65-96f5-0fab4dd6a195)
HTTP endpoints:
  GET  /flows              - List available flows
  POST /flows/{id}/run     - Execute a flow
  GET  /health             - Liveness probe
  GET  /ready              - Readiness probe
  GET  /docs               - OpenAPI docs
```

Endpoint	Method	Purpose
/	GET	Index. Human-readable list of loaded flows and endpoints.
/flows	GET	JSON list of flows in the deployed bundle, with their IDs and names.
/flows/{id}/run	POST	Execute the flow with the given ID. Request body carries the flow's inputs. Response carries its outputs. See */docs* for the exact schema.
/health	GET	Liveness probe. Used by Kubernetes. Safe for external monitoring.
/ready	GET	Readiness probe. Returns 200 when the runtime has loaded all flows and is ready to serve requests.
/docs	GET	Interactive OpenAPI documentation for the runtime API.

For the authoritative request/response shapes for `POST /flows/{id}/run` , open `/docs` against your specific deployment. The schema reflects the inputs and outputs of the flows you published.

Example: list flows

```
curl -L https://<hm-portal-host>/projects/<project-id>/apps/<app-id>/routes/<route-id>
↪/flows \
  -H x-access-key: <your-machine-user-key>
```

Example: run a flow

```
curl -L -X POST \
  https://<hm-portal-host>/projects/<project-id>/apps/<app-id>/routes/<route-id>/
↪flows/<flow-id>/run \
  -H x-access-key: <your-machine-user-key> \
  -H content-type: application/json \
  -d { "inputs": { "input_value": "hello" } }
```

The exact request body depends on the input components in your flow. Open `/docs` in a browser (with `x-access-key` set via a browser extension or proxy) to see the generated schema.

MCP server endpoint

The MCP server route exposes the deployed bundle's flows as Model Context Protocol tools. Point an MCP-capable client (an LLM agent, an IDE integration) at the MCP route URL and supply the same `x-access-key` header as for HTTP. The client will discover each flow as an MCP tool and can invoke it with structured arguments.

For the MCP protocol itself, see the MCP specification .

Logs and observability

Logs and metrics from a deployed flow go to HM's observability stack and are surfaced through the standard HM Grafana dashboards. See *Observability (Agent Factory on HM)* for Prometheus alert rules and Grafana dashboard setup for Langflow pods and model cluster performance.

Common errors

Symptom	Likely cause
401 Unauthorized from the gateway	Missing or invalid *x-access-key*, or the machine user does not have the role required to access the project's deployed flows.
404 Not Found on the runtime route	Wrong route ID, or the deployment has been undeployed. Re-check the URL in the Estate UI.
/ready returns 503	The runtime is still downloading the bundle at startup, or hasn't finished loading flows yet. Wait and retry. If it persists, check the deployment's logs.
Flow run returns a credential error from a downstream resource	A credential parameter (such as a DB password or model API key) was not supplied at deploy time, or the secret it carries is no longer valid against the downstream system. Edit the deployed flow's parameters from the Estate UI.

Langflow templates

Langflow in Hybrid Manager ships with pre-built flow templates for common workflows. Templates give you a working starting point — open one from the template gallery, configure a few inputs, and run it in the Playground.

To access templates, open Langflow from **Launchpad** in the HM console, select **New Flow**, and then select **All Templates**.

Template	What it does
Simple Migration Agent	Automates database migration into Hybrid Manager: creates a destination cluster, migrates the schema, and kicks off data migration.

Simple Migration Agent

The Simple Migration Agent is a pre-built Langflow template that automates the process of migrating a database into Hybrid Manager. It uses an LLM to orchestrate the migration steps: it creates a destination cluster in your HM project, migrates the schema, and kicks off the data migration. Once the data migration is running, the agent's job is done — you monitor progress directly in Hybrid Manager.

Important

The Simple Migration Agent is not recommended for production use. It is best suited for Postgres-to-Postgres migrations into Hybrid Manager in non-production environments.

All four prerequisites must be in place before running the agent.

1. **Configure an LLM API key** in Langflow. The agent requires an LLM to interpret and orchestrate migration steps. OpenAI is recommended. In Langflow, go to **Profile** > **Settings** > **Model Provider** and enter your OpenAI API key before creating a new project.
2. **Install and register the Beacon Agent** on the source database host. The Beacon Agent discovers the source database and makes it visible in the HM console as a registered resource. Confirm the source database appears under your project in the HM console before proceeding.
3. **Install the DMS agent** and confirm the source database appears as a migration source. The DMS (Data Migration Service) agent handles the actual data transfer. Verify the source is listed and available as a migration source in HM.
4. **Verify matching resource IDs.** The resource ID used for the DMS agent connection and the EDB Postgres AI Agent connection must be the same. Mismatched resource IDs will cause the agent to fail. Confirm this before running the template.

Run the Simple Migration Agent

1. Open Langflow from **Launchpad** in the HM console.
2. Select **New Flow** > **All Templates**.
3. Select **Simple Migration Agent** from the template gallery.
4. Before creating the project, confirm that the **Model Provider** is set to **OpenAI** (or your chosen provider) and that your API key is configured. The agent needs a working LLM connection to proceed.
5. Select **Create** to open the flow on the canvas.
6. In the **Text Input** component on the left side of the canvas, provide the required inputs in the following format:

```
project id: <your-project-id>
source database resource id: <your-source-database-resource-id>
```

- **project id** — the ID of the HM project where the destination cluster will be created. - **source database resource id** — the resource ID of the source database as registered by the Beacon Agent and DMS agent. This must match the ID used in both agent configurations (see *Prerequisites*).

Optionally, add a `target cluster name` field to specify an existing cluster as the migration target instead of creating a new one:

```
project id: <your-project-id>
source database resource id: <your-source-database-resource-id>
target cluster name: <your-target-cluster-name>
```

1. Optionally, in the **Cluster Creation Wrapper** component, enter a cluster template name in the provided textbox to use as the destination cluster configuration. If left empty or if the template doesn't exist, the agent defaults to a single-node EDB PaaS Postgres 18 cluster.
2. Select **Run**. The agent orchestrates the following steps automatically:

- Creates a new destination cluster in your HM project using the specified template, or a single-node EDB PaaS Postgres 18 cluster if no valid template is provided.
- Migrates the schema from the source database to the destination cluster.
- Kicks off the data migration.

1. Once the data migration starts, navigate to **Hybrid Manager** to monitor migration progress. The active migration appears in your project under the migration status view.

> **Note**
>
> The Langflow agent's role ends when data migration is initiated. All subsequent monitoring and management happens in Hybrid Manager, not in Langflow.

Review these limitations before running the agent.

Schema compatibility is required. The agent does not validate or repair schema compatibility. If the source schema has incompatible objects (for example, unsupported data types, extensions not available in the destination, or naming conflicts), the migration will fail. Resolve all schema issues before running the agent — use Migration Portal AI or EDB Copilot to assess and fix schema compatibility first.

Creates a new destination cluster by default. The agent provisions a new single-node EDB PaaS cluster (Postgres 18) as the migration target unless a valid cluster template name is provided in the **Cluster Creation Wrapper** component. You can also specify an existing cluster as the target by adding a `target cluster name` to the **Text Input** component.

Destination must be within Hybrid Manager. The agent cannot migrate to external destinations outside of HM. The destination cluster must exist within your HM project.

Not for production use. The template is designed for evaluation and non-production migrations. It is best suited for straightforward Postgres-to-Postgres migrations into Hybrid Manager.

Security considerations for Langflow

This page covers how credentials are handled as a flow moves from the flow builder to a deployed flow.

Credentials in Global Variables

Global Variables in the flow builder are used for storing credentials and non-sensitive configuration values. Understand where credentials live at each stage and the risk that carries.

In the flow builder:

- Global Variables are encrypted and stored in the Langflow database. That database is shared by all flows in the flow builder, and Global Variables for all flows and all users are encrypted with the same key. Users can't see

each other's Global Variables, but there is no isolation: any compromise of the database or the encryption key exposes every user's Global Variables.

When a flow is published:

- Global Variable values are not included in the published bundle. The variable names are extracted and stored in the bundle as parameters. You can set default values for generic parameters at publish time; no value can be set for a credential parameter.

When a flow is deployed:

- You provide values for parameters at deploy time. Generic parameters can use their defaults, but credential parameters must be given a value. Those values are stored as Kubernetes secrets in an isolated namespace used only by that deployed flow.

Recommendations:

- In the flow builder, use Global Variables only for credentials to non-sensitive resources (for example developer or test databases and model deployments).
- Add production credentials only when deploying a flow, where each deployment's credential values are stored as Kubernetes secrets isolated to that flow.

Related pages

- *Sovereign AI on Hybrid Manager* — data residency and air-gapped architecture concepts.
- *Flow deployment* — project-level isolation, scaling, and access control for deployed Langflow flows.

Langflow FAQ

How do I improve retrieval quality in a RAG flow?

Poor retrieval results are usually caused by one of the following:

- **Wrong embedding model** — the model used to create the knowledge base embeddings must match the model used at query time. Check the `model_name` used in `aidb.create_table_knowledge_base`.
- **Chunk size** — if source documents were ingested as large chunks, retrieval scores may be noisy. Re-index with smaller chunks using `aidb.chunk_text_config`.
- **Top-k too low** — the EDB Knowledge Base component returns the top-k most similar results. Increase this if relevant content is being missed.
- **Index not up to date** — if source data has changed since the last `aidb.bulk_embedding` run, retrieved content may be stale. Re-run the bulk embedding or set up incremental syncing.

What's the difference between EDB Model Server and EDB Embeddings?

EDB Model Server sends a prompt to an LLM and returns a text response — use it for generation, summarisation, and reasoning steps. **EDB Embeddings** converts text into a vector using an embedding model — use it when you need to compute similarity or store vectors in Postgres. Both require a model cluster to be deployed in your HM project, but they use different model types (LLM vs embedding model).

"Can't select a project" — the portal domain defaults to the API endpoint

Select **Controls** at the top of the affected EDB Database or Knowledge Base component and set **Hybrid Manager URL** to:

```
http://istio-internalgateway.istio-system.svc.cluster.local:7080
```

Troubleshooting Langflow in Hybrid Manager

Component configuration errors

Components show empty or invalid fields after importing a flow

Cause: Exported flow `.json` files don't include HM Machine User Keys, passwords, or Global Variable values. These are scoped to the Langflow instance they were exported from.

Fix:

1. In the new Langflow instance, go to **Profile → Settings → Global Variables** and create variables for the machine user key, database user, and database password using production values.
2. Open each EDB component and set **HM Machine User Key** to the correct Global Variable.
3. Reconfigure project, cluster, and database selections to point to the correct resources.

"Can't select a project" — portal domain defaults to the API endpoint

Cause: The EDB component is resolving the Hybrid Manager URL to the internal API endpoint instead of the frontend.

Fix: Select **Controls** at the top of the affected EDB Database or Knowledge Base component and set **Hybrid Manager URL** to:

```
http://istio-internalgateway.istio-system.svc.cluster.local:7080
```

EDB Knowledge Base component

Empty or irrelevant retrieval results

Symptoms: The component runs without error but returns no results, or returns results that don't match the query.

Diagnostic steps:

1. Verify the knowledge base exists and is populated:

```
SELECT * FROM aidb.kbstat WHERE name = your_knowledge_base_name;
```

1. Check that `embeddings` is greater than zero. If not, run:

```
SELECT aidb.run_pipeline(your_knowledge_base_name);
```

1. Confirm the **Database Name** and **Knowledge Base** fields in the component match exactly what was created in the cluster.

Common causes:

- Knowledge base was created but `aidb.run_pipeline` was never run.
- Source data was updated after the last embedding run — re-run `aidb.run_pipeline` to sync.

- The embedding model used to create the knowledge base doesn't match the one currently deployed. The model passed to `aidb.knowledge_base_config(...)` in `aidb.create_pipeline(...)` must correspond to a model cluster that's still active.

Connection failure to the Postgres cluster

Symptoms: Component shows a connection error when run.

Diagnostic steps:

1. Verify the machine user has **Project Viewer** role on the project:
   - **Project → Users → Assign Project Roles → Project Viewer**
2. Check that the Postgres cluster is active in the HM console.
3. Confirm **Database Connection Type**, **User for the database**, **Password for the database**, and **Database Name** all match what's configured in the cluster.

EDB Model Server component

Model cluster not available in the dropdown

Cause: No model cluster is deployed in the current HM project, or the machine user doesn't have the **AI Model Manager** role.

Fix:

1. Deploy a model cluster from **Agent Factory → Model Library** in the HM console. See *Model Management on Hybrid Manager* .
2. Confirm the machine user has **AI Model Manager** at organization level:
   - **Profile → User management → Assign Organization Roles → AI Model Manager**

Component runs but returns no response or an error

Symptoms: The EDB Model Server component finishes with an error, or returns an empty response.

Diagnostic steps:

1. Check the model cluster status in **Agent Factory → Model Server Clusters**. The cluster must show **Active** / **Healthy** before it can serve requests.
2. If the cluster is starting up (status: Pending), wait for it to become ready and retry.
3. Check the cluster's pod logs for initialization errors using the Hybrid Manager Grafana instance. See *Observability (Agent Factory on HM)* for dashboard access.
4. Verify the **Input** field in the component is connected to an upstream component and isn't empty when the flow runs.

Context window exceeded

Symptoms: The model returns an error mentioning token limits or truncated input.

Fix: Reduce the amount of context being passed to the model. If the flow pipes EDB Knowledge Base results directly into the model:

- Lower the **Limit results** value on the EDB Knowledge Base component to retrieve fewer chunks.

- Use a chunking step (AIDB `aidb.chunk_text_config`) on source documents to reduce individual chunk size before indexing.

EDB Embeddings component

No model clusters available in the dropdown

Same cause and fix as EDB Model Server — see *Model cluster not available in the dropdown* above.

Embedding generation fails

Diagnostic steps:

1. Confirm the model cluster uses an embedding model (not an LLM) — the two are not interchangeable.
2. Check that the cluster is Active in **Agent Factory → Model Server Clusters**.
3. Review pod logs using the Hybrid Manager Grafana instance.

Flow-level issues

Flow works in Playground but fails after moving to another Hybrid Manager instance

Likely causes:

- Global Variables in the production Langflow instance aren't set up, or reference incorrect values.
- The production machine user has different (or missing) role assignments.
- The production knowledge base or model cluster has a different name than the development one.

Confirm every EDB component points to the correct project, cluster, and database, and that its Global Variables hold valid values. To run the flow as a managed service with parameters and credentials set at deploy time, see *Flow deployment* .

Flow runs but gives inconsistent or low-quality answers

Start with retrieval quality — most answer quality issues in RAG flows trace back to what the EDB Knowledge Base component is returning, not the model. See *Empty or irrelevant retrieval results* and the retrieval quality guidance in *Langflow FAQ* .

Further help

- *Langflow FAQ* — common questions about components, Global Variables, and flow deployment.
- *Observability (Agent Factory on HM)* — Grafana dashboards and Prometheus monitoring for Langflow pod health and model cluster performance.
- *Troubleshooting Langflow in Hybrid Manager* — infrastructure-level issues (GPU, InferenceService, storage).

6.5.4.2 Pipeline Designer

The **Pipeline Designer** is a visual interface in Hybrid Manager (HM) for building and managing AI Database (AIDB) data pipelines. It provides a guided, no-code workflow for creating pipelines that process data through AI and ML transformation steps, optionally storing the results in a vector knowledge base for retrieval-augmented generation (RAG).

With Pipeline Designer you can:

- Create single-step or multi-step data pipelines through a visual wizard.
- Build knowledge bases with embedded vector storage for semantic search, including multi-pipeline knowledge bases where multiple pipelines feed a single knowledge base.
- Configure processing modes (Live, Background, or On Demand).
- Monitor pipeline status, record counts, and error conditions.
- Test knowledge base retrieval with built-in query tools.
- Use HM-hosted models, HM external inference service proxies, AIDB-native models, or built-in on-database models.
- Run pipelines on primary HM clusters, secondary HM locations, or self-managed Postgres instances.

6.5.4.2.1 How Pipeline Designer relates to AIDB

Pipeline Designer is a graphical front end for the AIDB extension . Every pipeline you create through the designer translates directly to AIDB SQL calls `aidb.create_pipeline` , `PipelineStepOperation` , `PipelineAutoProcessingMode` , and related functions running in your Postgres database. Pipeline Designer doesn't introduce its own conceptual vocabulary. It presents AIDB pipelines, knowledge bases, and models through a visual interface. For the underlying concepts, see the AIDB documentation .

Pipeline Designer manages only the pipelines it creates. Pipelines created directly through SQL (outside the Pipeline Designer UI) don't appear in the designer interface. See *VPU and permissions* for details on why this separation exists.

6.5.4.2.2 Prerequisites

Before creating your first pipeline, complete the setup steps in *Setup prerequisites* . The setup checklist covers extension installation, cluster configuration, platform roles, model registration, and source table permissions. The *Getting started with Pipeline Designer* page also includes example source tables you can use to try Pipeline Designer straight away.

Pipeline Designer provides a subset of the full AIDB feature set. Key limitations include volume-based sources not yet being supported in the UI. For the full list, see *Limitations* .

6.5.4.2.3 Supported cluster types

Pipeline Designer supports the following cluster configurations:

- **HM-managed clusters** (Primary/Standby Replication (PSR) and Postgres Distributed (PGD)/AHA): Clusters provisioned and managed by Hybrid Manager on the primary location.
- **Secondary HM locations**: HM-managed clusters running on additional HM locations in a multi-location deployment.
- **Self-managed Postgres instances**: Single-instance Postgres servers registered with HM through the EDB Postgres AI agent (`beacon-agent`), with AIDB installed. External PGD clusters and Cloud Native Postgres (CNP) clusters aren't supported.

The minimum supported Postgres major version is 16, and the minimum supported AIDB extension version is 7.2.0. For details on how execution works across these cluster types, see *Executing pipelines* .

- *Getting started with Pipeline Designer* : Set up your environment and create your first pipeline.
- *Pipeline Designer concepts* : Learn about pipeline structure, multi-step design constraints, and processing mode selection.
- *Creating pipelines* : Walk through the pipeline creation wizard.
- *Knowledge bases and retrieval* : Understand vector storage, monitoring, and querying.
- *External inference services* : Register external model providers.
- *Executing pipelines* : Run pipelines on different cluster types.
- *VPU and permissions* : Understand the permission model.
- *Limitations* : Design constraints and current limitations.

Getting started with Pipeline Designer

Follow the setup steps and two complete pipeline examples below to get Pipeline Designer running. The examples cover a text summarization pipeline and a knowledge base pipeline, taking you through the core workflow from source table preparation through result verification.

Setup prerequisites

Pipeline Designer requires a dedicated Postgres role called `visual_pipeline_user` (VPU) - the role Pipeline Designer uses to execute all pipeline operations, including querying the model registry, creating destination tables, and running pipeline steps. This role isn't created automatically. You must create it manually before using Pipeline Designer. The model picker queries the AI Database (AIDB) model registry as VPU, so if the role doesn't exist, the model picker is empty and pipeline creation can't proceed. See Creating VPU role for the setup steps. The AIDB extension may or may not be pre-installed, depending on the cluster type:

Deployment type	Extension installed at provisioning	Model picker populated before first pipeline
Postgres Distributed (PGD) managed	Yes	No (empty until VPU exists)
Primary/Standby Replication (PSR) managed	No	No (empty until extension and VPU exist)
Self-managed (any)	No	No (empty until extension and VPU exist)

The steps below are ordered by dependency, so you can identify which steps apply when setting up a new database or adding a source table to an existing deployment.

Step	Scope	Applies to	Re-run when
shared_preload_libraries	Instance	Self-managed	PG major upgrade
max_worker_processes	Instance	All	Pipeline concurrency growth
CREATE EXTENSION aidb CASCADE	Database	PSR managed, self-mgd	New database
aidb.bdr_setup()	Database	PGD self-managed	New database, extension upgrade
GRANT CREATE ON SCHEMA public	Database	All	New database
beacon-agent DDL privileges	Database	Self-managed	New cluster registration

continues on next page

Table 60 – continued from previous page

Step	Scope	Applies to	Re-run when
CREATE ROLE visual_pipeline_user + *aidb_users* grant	Instance	All	Never (one-time)
Hybrid Manager (HM) roles	Platform	All	New user onboarding
Inference service registration	Platform	All	New model provider
Source table grants	Table	All	New tables, schema changes
Pipeline object creation	Pipeline	All	Automated (no manual action)

If this is your first time setting up Pipeline Designer, follow all sections below in order.

If you are adding Pipeline Designer to a new database on an existing cluster, start at *Cluster-level setup* if pipeline concurrency has changed, then proceed through *Per-database setup* . Skip Creating VPU role if the role already exists on the instance.

If you are only adding a new source table to an existing pipeline deployment, go directly to *Per-source-table setup* .

Cluster-level setup

Apply cluster-level settings once per Postgres instance. Revisit these settings after a Postgres major upgrade or when scaling pipeline concurrency.

- On self-managed clusters, you must add `aidb` to `shared_preload_libraries` in `postgresql.conf` and restart Postgres before creating the AIDB extension:

```
shared_preload_libraries = aidb
```

See the AIDB configuration documentation for the full self-managed configuration procedure.

AIDB uses Postgres background workers to execute pipelines in Background processing mode. The default `max_worker_processes` value (8) may be insufficient for workloads with multiple concurrent pipelines. Insufficient worker slots cause pipelines to silently fail to start.

> **Tip**
>
> Pipeline Designer doesn't surface an error or retry, so the pipeline appears permanently stuck. If a pipeline is unexpectedly idle, check *max_worker_processes* early.

```
max_worker_processes = 64
```

The recommended value depends on the number of concurrent pipelines and other extensions that consume background workers. A value of 64 is a reasonable starting point for Pipeline Designer workloads.

Per-database setup

Repeat every step in this section for each database where you use Pipeline Designer. When you create a new database on an existing cluster, return here.

On PGD managed clusters, the AIDB extension is created automatically during cluster provisioning. No manual extension installation is needed.

On PSR managed and self-managed clusters, the extension isn't created at provisioning. The model sync that populates the Pipeline Designer model picker needs the extension to discover available models. Without it, the model picker is empty and pipeline creation can't proceed. Install the extension manually before using Pipeline Designer:

```
CREATE EXTENSION IF NOT EXISTS aidb CASCADE;
```

```
SELECT aidb.bdr_setup();
```

On managed PGD clusters this step runs automatically at provisioning. Re-run `bdr_setup()` after upgrading the AIDB extension with `ALTER EXTENSION aidb UPDATE;` . The function is a no-op if the migration script already invoked it, so running it unconditionally after each upgrade is safe.

```
GRANT CREATE ON SCHEMA public TO visual_pipeline_user;
```

Per-table grants are configured in *Per-source-table setup* . For On Demand mode, this is a `SELECT` grant. For Live and Background modes, VPU must be granted the source table's owner role. See *Configuring VPU permissions* for the full set of required grants. Without these grants, pipeline creation or execution will fail with a permission error.

AIDB documentation for self-managed configuration requirements.

The `visual_pipeline_user` role is a cluster-wide Postgres role that Pipeline Designer uses for all pipeline operations. It must exist before you open Pipeline Designer, because the model picker queries the AIDB model registry as VPU. If the role is missing, the model picker is empty and pipeline creation can't proceed.

Create the role once per Postgres instance. The `aidb_users` role that VPU must join is created by the AIDB extension, so the extension must exist in at least one database first.

- On **Postgres 16 and earlier**, VPU requires `LOGIN` privileges because background worker authentication uses role-based login:

```
CREATE ROLE visual_pipeline_user LOGIN;
GRANT aidb_users TO visual_pipeline_user;
GRANT visual_pipeline_user TO edb_beacon;
```

- On **Postgres 17 and later**, `LOGIN` isn't required because background worker authentication no longer depends on it:

```
CREATE ROLE visual_pipeline_user;
GRANT aidb_users TO visual_pipeline_user;
GRANT visual_pipeline_user TO edb_beacon;
```

This step is one-time and instance-level, applying to all deployment types (PGD managed, PSR managed, and self-managed). You don't need to repeat it when adding databases or tables. For the full set of per-table grants that VPU needs, see *Configuring VPU permissions* .

Platform-level setup

Platform configuration is performed in the EDB Postgres AI portal, not in Postgres. These steps don't need to be repeated when you add databases or tables.

Two HM roles control access to Pipeline Designer features:

1. **Pipeline editor project role.** Required to create, edit, and delete pipelines within a project.
2. **AI Model Manager organization role.** Required to access the **Settings** tab, where external inference services are registered and managed.

Ensure the relevant users have been assigned these roles before they begin working with Pipeline Designer.

At least one inference service must be registered before you can create pipelines that use model-backed steps (such as SummarizeText or KnowledgeBase). Without a registered inference service, the model picker in Pipeline Designer has no models to offer. See *External inference services* for the available registration paths (HM external inference proxy and AIDB-native registration).

Per-source-table setup

Grant VPU access each time you add a new source table to a pipeline, and update grants when schemas change (for example, when a table moves to a different schema or new tables are added).

See *Configuring VPU permissions* for the required grants (a `SELECT` grant for On Demand mode, or the source table's owner role for Live and Background modes) and topology-specific instructions (public schema, custom schemas, and PGD clusters).

Per-pipeline setup

Pipeline-level database objects are managed automatically. When you create a pipeline through Pipeline Designer, beacon-agent handles all per-pipeline database operations, including creating destination tables, knowledge base vector tables, trigger functions, and pipeline state tracking tables. No manual per-pipeline setup is required. For details on the objects VPU creates, see *Objects created by VPU* .

Example: Summarize text

This example creates a single-step pipeline that takes text content from a source table, runs it through a text completion model for summarization, and writes the summaries to a destination table.

Create the source table

Connect to your database and create a table with some sample text content:

```
CREATE TABLE source_table_st (
    id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    content TEXT NOT NULL
);

INSERT INTO source_table_st (content) VALUES
(I dont want a babysitter. I am eleven years old. My babysitter is only three years␣
↪older than I am," she loudly yelled to her Mom. Now, she really wished she had␣
↪somebody with her as she heard the clicking, scratching noises outside of the␣
↪living room window. "This is silly. Its probably the storm," the girl said. She␣
↪regretted watching the horror show she had been tuned into for the last half hour.␣
↪As she searched for the remote to turn off the vampire movie, the front door blew␣
↪open with a thunderous noise. Carla whirled around to see a dark image.),
(There are times when the night sky glows with bands of color. The bands may begin as␣
↪cloud shapes and then spread into a great arc across the entire sky. They may fall␣
↪in folds like a curtain drawn across the heavens. The lights usually grow brighter,␣
↪then suddenly dim. During this time the sky glows with pale yellow, pink, green,␣
↪violet, blue, and red. These lights are called the Aurora Borealis. Some people␣
↪call them the Northern Lights. Scientists have been watching them for hundreds of␣
↪years. They are not quite sure what causes them. In ancient times people were␣
↪afraid of the Lights. They imagined that they saw fiery dragons in the sky. Some␣
↪even concluded that the heavens were on fire.);
```

Grant VPU access to the source table. This example uses Background processing, so VPU must be granted the source table's owner role:

```
GRANT <source_table_owner_role> TO visual_pipeline_user;
```

Replace `<source_table_owner_role>` with the role that owns `source_table_st` . If you instead use On Demand mode, grant only `SELECT ON source_table_st TO visual_pipeline_user` . This example covers the public schema. For custom schemas or PGD clusters, see *Configuring VPU permissions* .

Create the pipeline

1. In the EDB Postgres AI portal, navigate to **Sovereign AI > Pipelines** and click **Create Pipeline**.
2. Enter a pipeline name, for example `summarize_demo` .
3. Select your **cluster** and **database**.
4. Set the **processing type** to **Background**.
5. Select **Next**.
6. Select the **schema** (`public`), **table** (`source_table_st`), **key column** (`id`), and **data column** (`content`).
7. Select the + button on the arrow between the source and destination and select **SummarizeText**.
8. In the **SummarizeText** step configuration, select a **text completion model** from the model picker. The available models depend on which inference services you registered during *Platform-level setup* .
9. Select **Deploy**.

The pipeline appears in the pipeline list with a **Creating** status that transitions to **Up To Date** once setup is complete.

> **Note**
>
> Background processing runs automatically after deployment, processing new and existing rows without manual intervention. For Live and On Demand modes, see *Processing modes* .

With Background processing, the pipeline begins processing automatically after deployment. Monitor the pipeline list until the status changes from **Creating** to **Up To Date**, which indicates that all source records have been processed.

Once the pipeline reaches **Up To Date**, verify the results by querying the destination table:

```
SELECT * FROM pipeline_summarize_demo;
```

The destination table contains one row per source record, with the `content` column replaced by the model's summary. For example, the passage about the Aurora Borealis produces a condensed summary capturing the key points about the Northern Lights, their appearance, and their history.

Example: Build a knowledge base

This example creates a two-step pipeline that chunks product descriptions into segments, generates vector embeddings, and stores them in a knowledge base for semantic search. This workflow is the standard pattern for building RAG-ready vector stores with Pipeline Designer.

Create the source table

Connect to your database and create a table with product data:

```
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    product_name TEXT NOT NULL,
    description TEXT
);

INSERT INTO products (product_name, description) VALUES
(Classic Burger, A juicy beef patty with lettuce, tomato, and pickles on a toasted␣
↪sesame bun),
(Margherita Pizza, Wood-fired pizza with San Marzano tomatoes, fresh mozzarella, and␣
↪basil),
(Caesar Salad, Crisp romaine lettuce with parmesan, croutons, and house-made Caesar␣
↪dressing),
(Fish Tacos, Grilled mahi-mahi with cabbage slaw, lime crema, and pickled onions),
(Falafel Wrap, Crispy chickpea falafel with tahini, pickled turnips, and fresh herbs␣
↪in warm pita),
(Pad Thai, Stir-fried rice noodles with shrimp, peanuts, bean sprouts, and tamarind␣
↪sauce),
(Beef Burrito, Seasoned beef with rice, black beans, cheese, and salsa in a flour␣
↪tortilla),
(Mushroom Risotto, Creamy arborio rice with mixed wild mushrooms, parmesan, and␣
↪truffle oil),
(Lamb Kebab, Spiced lamb skewers with grilled vegetables, garlic yogurt, and crisp␣
↪pitta);
```

Grant VPU access to the source table. This example uses Background processing, so VPU must be granted the source table's owner role:

```
GRANT <source_table_owner_role> TO visual_pipeline_user;
```

Replace `<source_table_owner_role>` with the role that owns `products`. If you instead use On Demand mode, grant only `SELECT ON products TO visual_pipeline_user`. This covers the public schema. For custom schemas or PGD clusters, see *Configuring VPU permissions*.

Create the pipeline

1. In the EDB Postgres AI portal, navigate to **Sovereign AI > Pipelines** and select **Create Pipeline**.
2. Enter a pipeline name, for example `product_kb`.
3. Select your **cluster** and **database**.
4. Set the **processing type** to **Background**.
5. Select **Next**.
6. Select the **schema** (`public`), **table** (`products`), **key column** (`id`), and **data column** (`description`).
7. Select the + button and select **ChunkText**. Set the **desired chunk length** (for example, `200`). For short descriptions like these the chunking step passes them through without splitting, but it is good practice to include it for pipelines that may later process longer text.
8. Select the + button after the ChunkText step and select **KnowledgeBase**.
9. In the **KnowledgeBase** step configuration, select a **text embedding model** from the model picker and set the **data format** to **Text**.
10. Select **Deploy**.

> **Note**
>
> Background processing runs automatically after deployment, processing new and existing rows without manual intervention. For Live and On Demand modes, see *Processing modes* .

With Background processing, the pipeline runs automatically after deployment. Wait for the pipeline status to reach **Up To Date**, then verify the knowledge base using the built-in query tool:

1. Navigate to **Sovereign AI > Knowledge Bases**.
2. Select the knowledge base created by your pipeline (its name is derived from the pipeline name).
3. Select **Query**.
4. Enter a search term, for example `seafood` or `spicy` .
5. Select **Run**.

The query tool returns the most semantically similar product descriptions with their distance scores. For example, searching for **seafood** returns the fish tacos entry as the closest match, even though the word **seafood** doesn't appear in its description. This demonstrates how vector embeddings capture semantic meaning rather than relying on keyword matching.

You can also verify the raw pipeline output at the database level:

```
SELECT * FROM pipeline_product_kb;
```

The knowledge base is now available for use by downstream applications such as *Langflow FAQ* agents. See *Using knowledge bases with Langflow* for integration details.

Next steps

- *Pipeline Designer concepts* : Learn about pipeline structure, multi-step design constraints, and processing mode selection.
- *Creating pipelines* : Full reference for the pipeline creation wizard, including all step types and model selection.
- *Knowledge bases and retrieval* : Monitor knowledge bases, understand record counts, and test retrieval.
- *VPU and permissions* : Understand the `visual_pipeline_user` role, grant details, and security considerations.
- *Limitations* : Current design constraints and known issues.

Pipeline Designer concepts

Pipeline Designer is a visual interface for AI Database (AIDB) pipelines . It presents and constrains AIDB functionality through a guided wizard. The concepts below cover what the designer enforces and why. For the underlying pipeline, knowledge base, and model concepts, see the AIDB documentation .

Pipeline structure

A pipeline is a sequence of one to ten processing steps that transform data from a source table into a destination. You configure the source, steps, and destination through the *Creating pipelines* .

Pipeline names must be lowercase, start with a letter, and contain only letters, digits, and underscores. The maximum length is 46 characters (AIDB reserves up to 17 characters of the Postgres identifier limit for internal suffixes). These constraints are validated when you enter the pipeline name in the *Creating pipelines* .

Step types

Pipeline Designer exposes the AIDB pipeline steps and knowledge base functionality as pipeline steps. For detailed descriptions of what each operation does, its parameters, and its input/output types, see the AIDB pipeline steps documentation .

Step type	Input	Output	Use case
ParseHTML	Text (HTML)	Text	Strip HTML markup and extract clean text
ParsePDF	Bytes (PDF)	Text	Extract text content from PDF files
PdfToImage	Bytes (PDF)	Bytes (image)	Convert PDF pages to images, typically before OCR
PerformOCR	Bytes (image)	Text	Extract text from scanned images or image-based PDFs
ChunkText	Text	Text	Split long text into smaller segments
SummarizeText	Text	Text	Condense text using a completion model
KnowledgeBase	Text	Vector	Generate embeddings and store in a vector knowledge base

Multi-step pipelines

Pipelines can chain up to ten steps. In the pipeline canvas, steps appear as cards arranged left to right between the **Table Source** and **Destination Table** endpoints, with a + control between each pair of cards for inserting a step at that position.

Ordering constraints enforced by the designer:

- Steps that consume raw document data (ParsePDF, PdfToImage) must be placed first. PerformOCR also accepts binary input but can appear later in the pipeline (for example, after PdfToImage).
- The KnowledgeBase step, if present, must be placed last.
- Middle steps (ChunkText, SummarizeText, ParseHTML) can appear in any order between the first and last positions.

Type compatibility:

Each step produces output of a specific type (text or bytes), and the next step must accept that type as input. The designer validates this automatically and prevents you from deploying incompatible step sequences. For example, you can't place a ChunkText step (which expects text input) immediately after a PdfToImage step (which produces bytes output) without an intermediate OCR step to convert bytes to text.

Common multi-step patterns:

Pattern	Steps	Use case
PDF to knowledge base	ParsePDF, ChunkText, KnowledgeBase	Index PDF documents for semantic search
Image OCR to knowledge base	PerformOCR, ChunkText, KnowledgeBase	Extract and index text from scanned documents
HTML processing	ParseHTML, ChunkText, SummarizeText	Clean, chunk, and summarize web content
Full document pipeline	ParsePDF, ChunkText, SummarizeText, KnowledgeBase	Parse, chunk, summarize, and index documents

Pipelines are limited to 10 steps, and the step structure can't be changed after creation. See *Pipeline creation and structure* for details.

Processing modes

During pipeline creation, select one of three AIDB auto-processing modes. You can change the mode after creation without recreating the pipeline.

Mode	How it works	When to use
On Demand (default)	Processes rows only when triggered manually via SQL	Testing, ad-hoc runs, or pipelines where you control exactly when processing occurs
Live	Processes rows immediately as they are inserted or updated, via database triggers	Low-latency use cases where results must reflect writes with minimal delay
Background	Processes rows asynchronously in batches on a configurable interval	Most production workloads. Handles large backlogs and continuous ingestion without blocking writes

In Background mode, the designer exposes two additional controls, batch size (default: 100) and sync interval (default: 30 seconds, ranging from 1 second to 2 days). For a full explanation of how each mode works, see the AIDB auto-processing documentation .

Data flow and step results

When a pipeline executes, data flows through AIDB's internal envelope from one step to the next. For details on how AIDB handles data lineage, chunking identifiers, and step-to-step data transfer, see the AIDB pipelines documentation .

In the pipeline creation wizard, each intermediate step has a **Save Step Results** toggle (enabled by default). When enabled, the step's output is persisted to a destination table, letting you inspect intermediate results at each stage of the pipeline and not just the final output. The toggle is hidden on the last step, whose output is always written to the pipeline's destination. See *Creating pipelines* for where this toggle appears in the wizard.

Creating pipelines

Pipeline Designer uses a two-phase wizard to create pipelines. The first phase configures the pipeline's target cluster, database, and processing options. The second phase defines the source table, processing steps, and destination.

Phase 1 - Pipeline options

1. Navigate to **Sovereign AI > Pipelines** in your Hybrid Manager (HM) project.
2. Select **Create New Pipeline**.
3. Enter a **pipeline name**. Names must be lowercase, start with a letter, contain only letters, digits, and underscores, and be at most 46 characters long.
4. Select a **cluster**. The dropdown shows all eligible clusters in your project:

- **HM-managed clusters** (Primary/Standby Replication (PSR) or Postgres Distributed (PGD)/AHA): For PGD clusters, you also select a data group. Witness groups are excluded. Only the write leader node is eligible for pipeline execution.
- **Self-managed Postgres instances**: Single-instance Postgres servers registered with HM through the EDB Postgres AI agent (`beacon-agent`). External PGD clusters and Cloud Native Postgres (CNP) clusters aren't supported.
- Only clusters running Postgres 16 or later appear. Replica clusters are excluded.

1. Select a **database** on the chosen cluster.
2. Configure the **processing type**:

- **On Demand** (default): No automatic processing. You trigger execution manually. After deployment, an On Demand pipeline shows a status of "No results" until you run it for the first time.
- **Live**: Real-time processing triggered by row changes. See *Processing modes* .
- **Background**: Asynchronous batch processing at a configurable interval.

1. If you selected Background, configure the **Batch Processing Size** (default: 500) and **Sync Interval** (default: 15 hours).
2. Select **Next** to proceed to step configuration.

Phase 2 - Step configuration

The step designer shows a visual pipeline with two fixed endpoints: a **Table Source** card on the left and a **Destination Table** card on the right. You build your pipeline by adding steps between these endpoints.

Configuring the source

1. Select the **Table Source** card.
2. Select the **Schema** containing your source table.
3. Select the **Source Table** to use as the pipeline source.
4. Select the **Source Key Column**: a column with unique values that identifies each row (typically a primary key or unique identifier).
5. Select the **Source Data Column**: the column containing the content to process (text, HTML, PDF bytes, or image bytes, depending on your first step).

> **Note**
>
> The *visual_pipeline_user* role must have access to the source table. In On Demand mode, it requires *SELECT* . In Live and Background modes, *visual_pipeline_user* must be granted the source table's owner role. If you encounter a permission error during pipeline creation, refer to *VPU and permissions* for the required permissions.

1. Select the + button on the arrow connecting two nodes in the pipeline.
2. Select a step type from the dropdown. Available step types depend on the position in the pipeline and the output type of the preceding step:

- **First step**: ParsePDF, PerformOCR, PdfToImage, ParseHTML, ChunkText, SummarizeText, or KnowledgeBase (if it is the only step).
- **Middle steps**: ParseHTML, ChunkText, SummarizeText (provided the preceding step produces text output), or PerformOCR (provided the preceding step produces binary image output).

- **Last step**: Any compatible step. If you want vector storage, KnowledgeBase must be the last step.

1. Configure the step's parameters:

- **ChunkText**: Set the desired chunk length (required) and optionally the maximum length.
- **SummarizeText**: Select a text completion model from the model picker.
- **PerformOCR**: Select an OCR model from the model picker.
- **KnowledgeBase**: Optionally select an existing **Knowledge Base Table Name** to append data to an existing knowledge base (enabling *Multi-pipeline knowledge bases*). Leave the field blank to create a new knowledge base. Choose the **Data Format** (Text or Image). - When creating a new knowledge base, select a **Model Service** (embedding model) and optionally configure the **Distance Operator** and **Index Type**. - When appending to an existing knowledge base, the model, distance operator, and index type are inherited from the existing knowledge base and aren't shown.

1. Repeat to add more steps, up to the 10-step maximum.

Each intermediate step has a **Save Step Results** toggle (enabled by default) that persists the step's output to a destination table, letting you inspect results at each stage. See *Data flow and step results* for details.

Managing step ordering

Pipeline Designer validates step ordering in real time. Invalid step placements are flagged visually. The rules are:

- Steps consuming raw document input (ParsePDF, PdfToImage) must be the first step. PerformOCR accepts binary image input and can appear first or after a step that produces image output (such as PdfToImage).
- KnowledgeBase must be the last step.
- Each step's output type must match the next step's expected input type.

During pipeline creation, you can reorder steps using the left and right arrow controls on each step card, and remove steps by selecting the delete control. Once deployed, the step structure is fixed — see *Managing a pipeline after deployment* .

Selecting models

Steps that require a model (KnowledgeBase, SummarizeText, PerformOCR) display a model picker that lists all models registered in the AI Database (AIDB) model registry on the target database. The picker filters by function type (embedding, text completion, OCR) based on the step being configured. The picker presents a flat list of model names and doesn't categorize models by source (HM-hosted, external, or on-database).

Not all registered models are reachable from every cluster. HM-hosted and HM-proxied models require network connectivity to the primary location's model-serving endpoints, which is available by default only on primary-location clusters. Clusters on secondary locations or self-managed instances must use *Registering AIDB-native models* (registered directly via SQL, calling an external provider) or *Built-in models* (shipped with AIDB, running locally). See *How models reach pipeline steps* for the full model availability breakdown by deployment type.

Deploying the pipeline

1. Verify that all steps are configured and the pipeline is valid (no red indicators).
2. The **destination table** name defaults to `pipeline_{your_pipeline_name}` . For pipelines ending with a KnowledgeBase step, this becomes the vector storage table.
3. Select **Deploy**.

Pipeline Designer creates the pipeline, associated database objects (destination tables, triggers or background processing artifacts), and begins monitoring its status. The pipeline appears in the pipeline list with a "Creating" status that transitions to its operational status once setup is complete.

Creating a KnowledgeBase pipeline

Most pipelines end with a KnowledgeBase step, which is the standard pattern for building RAG-ready vector stores. When you deploy such a pipeline:

1. AIDB creates a vector table to store embeddings.
2. A knowledge base entry is registered in AIDB's knowledge base registry.
3. The knowledge base is available for consumption by downstream applications such as *Langflow FAQ* agents. Pipeline Designer's *Knowledge bases and retrieval* view provides a query interface for testing and validation.

To create a pipeline with a KnowledgeBase step:

1. Follow the wizard above.
2. Add a **ChunkText** step to split your source text into segments.
3. Add a **KnowledgeBase** step as the final step.
4. In the KnowledgeBase step, leave **Knowledge Base Table Name** blank to create a new knowledge base (or select an existing one to append to it). Select a **text embedding model** and set the data format to **Text**.
5. Deploy the pipeline.

Pipelines without a KnowledgeBase final step are also possible (for example, a pipeline that only chunks or summarizes text into a destination table), but are less common.

For PDF or image sources, add a ParsePDF or PerformOCR step before ChunkText to convert binary data to text.

Managing a pipeline after deployment

Once a pipeline is created:

- **Step structure is fixed.** See *Steps cannot be modified after creation* .
- **Processing parameters are editable.** You can change the processing type, batch size, and sync interval by selecting **Actions > Edit Pipeline Options** from the pipeline detail page.
- **Manual execution is available via SQL.** Regardless of the processing mode, you musts trigger a manual run by calling `SELECT aidb.run_pipeline('pipeline_name');` on the pipeline's database. Pipeline Designer doesn't currently provide a UI button for manual execution.

Integrating with Langflow

Langflow FAQ is the flow-building interface for AI agents in EDB Postgres AI. Knowledge bases created through Pipeline Designer can serve as data sources for those agents: the Langflow KB component queries AIDB knowledge bases using the same vector retrieval mechanism as Pipeline Designer's built-in query tool. Langflow flows use user-configured database credentials, so the Postgres role configured in the flow must have access to the knowledge base's underlying tables. See *Using knowledge bases with Langflow* for permission details and the *Langflow documentation* for details on building agent flows.

Knowledge bases

A knowledge base is a vector store created by a pipeline whose last step is KnowledgeBase. It holds embeddings generated from your source data and supports semantic similarity search for retrieval-augmented generation (RAG) and other AI workloads.

Knowledge base list

Navigate to **Sovereign AI > Knowledge Bases** to see all knowledge bases in your project. Each entry shows:

- **Name**: The knowledge base identifier, derived from the pipeline name.
- **Origin**: The cluster and database where the knowledge base resides.
- **Model**: The embedding model used to generate vectors.
- **Source Pipelines**: The pipelines feeding data into this knowledge base. A knowledge base can have one or more source pipelines. When more than three pipelines are associated, the list shows the first three with a **+N more** link to the knowledge base detail page.
- **Source Records**: Total rows across source tables.
- **Result Records**: Number of vector embeddings stored.
- **Unprocessed Records**: Source rows that haven't yet been processed.

Knowledge base detail

Select a knowledge base to view its detail page.

Record counts

Metric	Description
Source records	Total rows in the source table
Embeddings	Number of vector embeddings stored in the knowledge base
Unprocessed records	Source rows that haven't yet been processed

When the pipeline is operating as expected, the unprocessed count is zero (for Background and Live modes) or equal to the source count minus embeddings (for On Demand mode).

Source pipelines

The detail page lists all pipelines that feed data into this knowledge base. A knowledge base can receive data from multiple pipelines (N:1 relationship), enabling you to aggregate data from different sources and processing chains into a single searchable knowledge base.

Multi-pipeline knowledge bases

Pipeline Designer supports multiple pipelines writing to the same knowledge base. This lets you aggregate data from different source tables or apply different processing chains (for example, one pipeline for PDF documents and another for HTML content) into a single knowledge base for unified semantic search.

To add a pipeline to an existing knowledge base:

1. Create a new pipeline following the *Creating pipelines* .
2. Add a **KnowledgeBase** step as the final step.

3. In the **Knowledge Base Table Name** field, search for and select the existing knowledge base you want to append to.
4. Select the **Data Format** (Text or Image). The model, distance operator, and index type are inherited from the existing knowledge base.
5. Deploy the pipeline.

The new pipeline's data is appended to the existing knowledge base's vector table. Both the knowledge base list and detail pages show all source pipelines associated with each knowledge base.

Testing retrieval

The knowledge base detail page includes a built-in query tool for testing semantic search without writing SQL.

Running a test query

1. On the knowledge base detail page, select **Run Test Query**.
2. In the **Perform Test Query Retrieval** dialog, enter your search text in the **Search Query** field.
3. Optionally adjust the **Number of Results** parameter.
4. Click **Run Query**.

The query tool encodes your text using the knowledge base's embedding model, performs a vector similarity search, and returns the top matching records with their distance scores. Lower distance scores indicate closer semantic matches. The results appear in the same dialog beneath the query fields, with each matching record shown as a row containing the source key and its distance score.

Interpreting results

Each result includes:

- **Source key**: The key column value from the original source row, allowing you to trace results back to source data.
- **Distance**: The similarity distance between the query vector and the result vector, using the knowledge base's configured distance operator (L2 by default).

Test queries are useful for validating that your pipeline is producing meaningful embeddings and that the chunking strategy produces segments of appropriate granularity. If results seem poor, consider adjusting the ChunkText step's desired length or trying a different embedding model.

Knowledge base lifecycle

Knowledge bases are created automatically when you deploy a pipeline with a KnowledgeBase terminal step. They can't be created independently through Pipeline Designer.

When you delete a pipeline that has a KnowledgeBase step, the knowledge base is only deleted if no other pipelines are feeding into it. If other pipelines still reference the knowledge base, only the deleted pipeline's association is removed. If the deleted pipeline is the last one associated with the knowledge base, the knowledge base and its vector table are also deleted, and all stored embeddings are permanently removed.

> **Warning**
>
> Deleting a knowledge base is irreversible. All vector data is permanently removed. If the knowledge base is used by *Langflow FAQ* flows or other consumers, those integrations will break.

Using knowledge bases with Langflow

Knowledge bases created through Pipeline Designer are accessible from the Langflow-based Gen AI builder . The Langflow KB component connects to AIDB knowledge bases using the same *Knowledge bases and retrieval* functions that power the Pipeline Designer query tool.

To use a Pipeline Designer knowledge base in Langflow:

1. Create and populate a knowledge base through Pipeline Designer as described above.
2. In Langflow, add a KB component to your flow.
3. Configure the component with the database connection details and the knowledge base name.
4. The component queries the knowledge base for semantically similar content based on the flow's input.

Troubleshooting

No embeddings after pipeline runs

If the source record count is positive but the embedding count remains zero:

1. Check the pipeline status for errors. "Has Errors" or "Failed" status indicates processing problems.
2. Verify that the embedding model is reachable from the pipeline's cluster. Hybrid Manager (HM)-hosted and HM-proxied models are only reachable from primary-location clusters. If the pipeline runs on a secondary location or self-managed cluster, those models will fail. See *How models reach pipeline steps* .
3. If using an HM-hosted model on the primary location, confirm the KServe InferenceService is healthy.
4. Check that the source data column contains valid content for the configured pipeline steps.
5. Review the AIDB error log for detailed error messages. Connect to the database and run `SELECT * FROM aidb.get_error_logs('your_pipeline_name');` .

Unprocessed count not decreasing

If the unprocessed count remains static:

1. Verify the processing mode isn't On Demand. If On Demand, trigger a manual run or switch to Background or Live.
2. For Background mode, confirm the sync interval has elapsed since the last data change.
3. Check for pipeline-blocking errors that halt processing entirely.

Poor retrieval quality

If test queries return irrelevant results:

1. Review your chunking strategy. Chunks that are too large or too small produce suboptimal embeddings.
2. Consider a different embedding model. Some models perform better on specific content types.
3. Verify that the source data is clean and relevant. HTML tags, boilerplate text, or corrupt content degrade embedding quality. Consider adding a ParseHTML step before chunking.

External inference services

Pipeline Designer can use models from external AI providers (OpenAI, AWS Bedrock, NVIDIA NIM, and other API-compatible endpoints) in addition to Hybrid Manager (HM)-hosted models. Two distinct mechanisms make external models available, and which one applies depends on whether the cluster has connectivity to the HM control plane.

Understanding the delivery mechanism matters because it determines which clusters can use a given model and how credentials are managed.

How external models reach AIDB

AI Database (AIDB), the Postgres extension that powers Pipeline Designer, maintains a model registry. Pipeline steps reference models by name from this registry. External models can enter the registry through two independent paths.

HM external inference service proxy

When you register an external inference service through the Hybrid Manager UI (Sovereign AI > Model Serving, when available), HM deploys a managed proxy on the primary location that forwards inference requests to the external provider's API endpoint. HM manages the provider's API credentials, which aren't stored in the database.

From AIDB's perspective, the proxy is indistinguishable from an HM-hosted model: AIDB discovers it automatically and routes inference requests to it in the same way. The fact that inference ultimately reaches an external provider is transparent to AIDB and to Pipeline Designer.

Key characteristics of this path:

- **Managed by the HM control plane.** Registration, credential storage, and lifecycle management happen through the HM UI and backend services.
- **Credentials managed by HM.** The provider's API key (or equivalent) is stored and injected by HM infrastructure. AIDB doesn't handle external provider credentials directly.
- **Same reachability constraint as HM-hosted models.** The proxy runs on the primary location, so a cluster needs network connectivity to the primary to use this path. The external provider itself is internet-accessible, but traffic from AIDB goes through the proxy, not to the provider directly. In a default deployment, secondary locations and self-managed clusters don't have this connectivity and must use AIDB-native registration instead.

Registering an external inference service in HM

1. Navigate to *Model Management on Hybrid Manager* in your Hybrid Manager project.
2. Select **Register External Service**.
3. Fill in the registration form:

- **Service name**: A descriptive name for this service connection.
- **Provider**: The external provider type (for example, OpenAI, AWS Bedrock, NVIDIA NIM).
- **Endpoint URL**: The API endpoint for the external service.
- **API Protocol Version**: The protocol version to use when communicating with the provider. For most providers, the default value is correct. Select a specific version only if your provider requires it or you need to pin to a particular API revision.
- **Credentials**: The authentication credentials for the provider. This is typically an API key. For AWS Bedrock, this may include access key and secret.

1. Select **Register**.

The service is registered and its models become available after the next model sync cycle (typically within a few minutes).

Managing external services in HM

From the Sovereign AI model serving interface, you can:

- **View registered services**: See all external inference service connections, including their status and activity logs.
- **Edit a service**: Update the endpoint URL, credentials, or API protocol version.
- **Deregister a service**: Remove the external service connection. Models are cleaned up during the next model sync cycle.

> **Warning**
>
> Unregistering an external service will cause any pipelines using that service's models to fail on their next execution. Before unregistering, ensure no active pipelines depend on the service's models, or update those pipelines to use alternative models.

Registering AIDB-native models

AIDB has its own model registration mechanism that works independently of HM. The `aidb.create_model()` SQL function creates a model entry backed by a Foreign Data Wrapper (FDW), pointing directly at an external provider's API endpoint. When a pipeline step uses an AIDB-native model, the Postgres process contacts the provider directly, with no intermediary proxy.

Key characteristics of this path:

- **SQL-level, works anywhere.** Any Postgres cluster running the AIDB extension can register models this way, regardless of whether HM infrastructure is present or reachable.
- **Credentials stored as FDW user mappings.** API keys are stored using `CREATE USER MAPPING FOR PUBLIC`, making them accessible to all database users. Credentials are scoped to the provider (foreign server), not to individual models. All models using the same provider share a single set of credentials.
- **Direct network access required.** The Postgres process must be able to reach the external provider's API endpoint. There is no proxy or intermediary.
- **No HM UI management.** Models registered this way don't appear in the HM model serving interface. They are managed entirely through SQL.

For details on `aidb.create_model()`, provider types, and credential configuration, see the AIDB model registry reference documentation.

Choosing the right mechanism for your cluster

The HM external inference proxy runs on the primary location. Clusters outside the primary location can use it only if they have network connectivity to the primary's model-serving endpoints, which isn't available by default. AIDB-native registration works on any cluster with the AIDB extension, because the Postgres process contacts the provider directly.

Deployment type	HM external inference proxy	AIDB-native registration
Primary location	Yes	Yes

continues on next page

Table 65 – continued from previous page

Deployment type	HM external inference proxy	AIDB-native registration
Secondary location or self-managed	Not by default [1]	Yes

[1] Requires network connectivity to the primary location's model-serving endpoints, which is not present in a default deployment.

For the full model availability picture (including HM-hosted KServe models and built-in AIDB models), see *How models reach pipeline steps* .

Using external models in pipeline steps

Regardless of which mechanism registered them, external models appear in Pipeline Designer's model picker alongside HM-hosted and on-database models. Pipeline Designer doesn't distinguish between registration paths and shows every model in the AIDB model registry that's compatible with the step being configured.

When configuring a step that requires a model:

- **KnowledgeBase step**: External text embedding models (such as OpenAI embedding models) appear in the embedding model list.
- **SummarizeText step**: External text completion models (such as OpenAI completion models) appear in the language model list.
- **PerformOCR step**: External OCR models appear in the OCR model list, if the provider offers compatible endpoints.

The model picker filters by function type, so only models compatible with the step being configured are shown.

The function types that a model supports are determined by its provider. When you register an external inference service through the HM UI, the **Provider** you select (for example, OpenAI or NVIDIA NIM) determines which function types the service's models expose. Similarly, when you register a model with `aidb.create_model()` , the provider argument (the second parameter) controls the function types. For example, a model registered with the `openai_embeddings` provider supports text embedding, while one registered with `nim_paddle_ocr` supports OCR. There is no separate step to configure function types; they follow from the provider.

Handling credentials

External service credentials are shared across all database users, regardless of registration path. HM-registered external services store credentials in Kubernetes secrets (used by the proxymodel) and, after sync, as FDW user mappings in the database. AIDB-native models store credentials as FDW user mappings directly. In both cases, there is no per-user or per-pipeline credential isolation. For a full discussion of the credential sharing model and its security implications, see *Shared model credentials* .

Executing pipelines

Pipeline execution is the process of reading source data, running it through the configured processing steps, and writing results to the destination. The sections below cover how to trigger execution, how processing modes affect execution behaviour, and how execution works across different cluster types.

Execution flow

When a pipeline executes (whether triggered automatically or manually), the following sequence occurs:

1. Pipeline operations run under the *VPU and permissions* role. The mechanism depends on how the pipeline is triggered.

- For manual execution and pipeline management operations, the EDB Postgres AI agent (`beacon-agent`) opens a transaction and switches to VPU using `SET LOCAL ROLE` .
- For Background mode, the AI Database (AIDB) background worker connects to the database as the pipeline's owner role directly.
- For Live mode, per-pipeline SECURITY DEFINER wrapper functions owned by VPU execute with VPU's privileges regardless of the invoking session's role.

1. AIDB reads pending rows from the source table.
2. Each step processes the data sequentially. The output of one step becomes the input for the next.
3. If any step encounters a record-level error (for example, a malformed PDF), the error is logged and the pipeline continues with the remaining records. Pipeline-level errors (for example, model unreachable) halt execution.
4. The final output is written to the destination table, or to the knowledge base vector table if the last step is KnowledgeBase.

Running a pipeline manually

You can trigger pipeline execution manually regardless of the processing mode:

1. Connect to the pipeline's database.
2. Run `SELECT aidb.run_pipeline('your_pipeline_name');` .
3. The pipeline processes all pending source records.

> **Note**
>
> Pipeline Designer doesn't currently provide a UI button for manual execution. Use the SQL function above to trigger pipeline runs on demand.

Manual execution is useful for:

- Testing a newly created pipeline before enabling automatic processing.
- Processing a backlog of records after re-enabling a previously On Demand pipeline.
- One-time batch operations where automatic processing isn't needed.

Automatic execution modes

Live mode

In Live mode, AIDB installs a trigger on the source table that fires on every `INSERT` , `UPDATE` , or `DELETE` . The pipeline processes the affected row within the triggering transaction, so the application writing to the source table incurs the full latency of pipeline processing before the write returns.

Characteristics:

- Writes to the source table are blocked until the pipeline finishes processing the affected row. The application pays the cost of pipeline processing on every write.
- A slow inference model call (for example, embedding or summarization) adds latency directly to the application's write path.
- Suitable for low-volume, latency-sensitive scenarios where data must be embedded or processed immediately after writing.

Background mode

In Background mode, AIDB's background worker (running inside the Postgres instance) processes unprocessed rows in batches on a configurable schedule. Application transactions complete immediately. Pipeline processing happens independently.

Characteristics:

- Application writes to the source table are unaffected by pipeline processing. There is a delay (determined by the sync interval) before results appear.
- Batch size and sync interval are configurable (defaults: 500 rows per batch, 15-hour interval).
- Recommended for most production workloads.
- The background worker runs on the database cluster, not in the HM control plane. On Postgres Distributed (PGD) clusters, it runs on the write leader node.

Deployment types

Pipeline Designer can create and manage pipelines on clusters across all three Hybrid Manager (HM) deployment types. Understanding these categories is useful because they determine which model access paths are available. See *How models reach pipeline steps* below for the full breakdown.

An HM **location** is a Kubernetes environment managed by the HM control plane. HM supports multiple locations in a hub-and-spoke topology:

- **Primary location.** The initial HM-managed environment where the full platform stack runs, including the control plane, KServe model serving, the external inference service proxy, and all Sovereign AI services. Clusters on the primary location have full access to every feature.
- **Secondary locations.** Additional HM-managed environments in a multi-datacenter deployment. Secondary locations run the `core` installation scenario, which includes database clusters and essential services but excludes the AI scenario (KServe, model serving). They maintain control-plane connectivity back to the primary location.
- **Self-managed clusters.** Customer-managed single-instance Postgres servers registered with HM through the EDB Postgres AI agent (`beacon-agent`). External PGD clusters and Cloud Native Postgres (CNP) clusters are not supported. Self-managed clusters are not provisioned by HM and do not have network connectivity to the primary location by default. Without that connectivity, only AIDB-native and built-in models are available. See *How models reach pipeline steps* for details.

Supported clusters

Pipeline Designer manages pipelines on HM-managed clusters (both primary and secondary locations) and on self-managed Postgres instances. When you create a pipeline, you select the target cluster from the estate dropdown regardless of its deployment type. Pipeline Designer discovers databases, schemas, tables, and available models through the EDB Postgres AI agent on each cluster.

Self-managed clusters must meet additional prerequisites. They need Postgres 16 or later, the AIDB extension installed and configured, and the EDB Postgres AI agent connected to the HM control plane.

How models reach pipeline steps

Pipeline steps that perform inference (KnowledgeBase, SummarizeText, PerformOCR) reference a model by name from the AIDB model registry on the target database. Pipeline Designer does not distinguish between model sources at the pipeline level: the model picker shows every registered model whose function type matches the step being configured. However, models enter the registry through three distinct paths, each with different availability characteristics depending on where the pipeline's cluster sits in the deployment topology.

HM-hosted KServe models (Path 1)

Models deployed through HM's Model Serving run on the primary location's Kubernetes cluster. AIDB automatically discovers and registers these models, making them available for use when creating pipeline steps. When a pipeline step invokes such a model, AIDB routes the inference request to the model's serving endpoint on the primary location.

Availability: Available by default on the primary location. Clusters on secondary locations or self-managed instances can use these models only if they have network connectivity to the primary location's model-serving endpoints. This connectivity isn't present in a default deployment.

HM external inference service proxy (Path 2)

When you register an external provider (OpenAI, AWS Bedrock, NVIDIA NIM) through the HM UI, HM deploys a managed proxy on the primary location that forwards inference requests to the external provider, handling credential management on the user's behalf. AIDB discovers and addresses this proxy exactly as it would a locally hosted model (path 1).

Availability: Same constraint as path 1. The proxy runs on the primary location, so a cluster needs network connectivity to the primary to use it. Although the external provider itself is internet-accessible, AIDB's traffic goes through the proxy, not to the provider directly. For registration and management details, see *HM external inference service proxy (Path 2)* .

AIDB-native model registration (Path 3)

The `aidb.create_model()` SQL function registers a model directly in the AIDB model registry, backed by a Foreign Data Wrapper (FDW) pointing at the provider's API endpoint. When a pipeline step invokes an AIDB-native model, the Postgres process contacts the provider directly, with no intermediary proxy. Credentials are stored as FDW user mappings (`CREATE USER MAPPING FOR PUBLIC`).

Availability: Any cluster running the AIDB extension, regardless of HM connectivity. This is the only model access path available to clusters that can't reach the primary location. For details on providers, configuration, and credential management, see External inference services: AIDB-native model registration and the *AIDB model registry reference documentation* .

Model availability by deployment type

Deployment type	HM-hosted KServe (path 1)	HM external inference proxy (path 2)	AIDB-native registration (path 3)
Primary location	Yes	Yes	Yes
Secondary location (or self-managed with HM network)	Not by default [1]	Not by default [1]	Yes
Self-managed without HM network	Not by default [1]	Not by default [1]	Yes

[1] Requires network connectivity to the primary location's model-serving endpoints, which isn't present in a default deployment.

Paths 1 and 2 require network connectivity to the primary location's model-serving endpoints, which isn't present by default on secondary locations or self-managed clusters. Path 3 works anywhere because the Postgres process contacts the provider endpoint directly.

Built-in models

AIDB also ships with several built-in models (bert, clip, t5, llama) that run locally within the Postgres process and require no network access. These are available on every cluster with the AIDB extension installed.

Replication and failover

Pipeline execution behavior depends on the cluster architecture, not on the deployment type. The same mechanics apply whether the cluster is on the primary location, a secondary location, or self-managed.

- **PGD clusters** execute pipelines on the write leader node. Pipeline metadata (configuration, state, and metrics) replicates to all nodes through BDR (Bi-Directional Replication) logical replication, but execution occurs only on the write leader. On a write leader switchover, the new leader resumes execution: in Background mode, AIDB's background worker starts on the new leader and picks up unprocessed rows; in Live mode, trigger functions are replicated to all nodes and fire on the new write leader automatically. There may be a brief gap during the switchover while the new leader is elected.
- **Primary/Standby Replication (PSR) clusters** execute pipelines on the single writable node. On failover to a promoted standby, background processing resumes once the new primary is operational and AIDB's background worker starts. Live mode triggers are present on the promoted standby and fire normally. Rows written during the failover window (if any) are processed as part of the normal unprocessed-row backlog.

Monitoring execution

Pipeline execution status is visible in the pipeline list and detail pages:

- **Status indicator**: Shows whether the pipeline is Up To Date, Processing, Stale, Has Errors, or Failed.
- **Record counts**: Source records, destination records (or embeddings for KB pipelines), and unprocessed records.

A pipeline in an error state displays a Configuration Error badge in the list view:

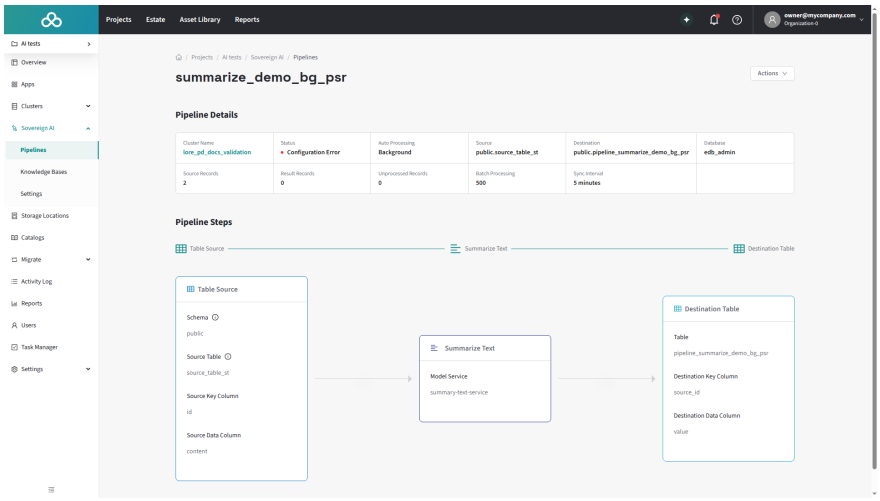


Fig. 1: Pipeline list entry showing a Configuration Error status badge

- **Error log**: For detailed error information, query the AIDB error log directly:

```
SELECT * FROM aidb.get_error_logs(your_pipeline_name);
```

For more on error log queries and error categories, see the AIDB error log documentation .

Errors are categorized as:

- **RecordTemporary / RecordPermanent**: Errors affecting individual records. The pipeline continues processing other records.
- **PipelineTemporary / PipelinePermanent**: Errors affecting the entire pipeline. Execution halts until the issue is resolved.

VPU and permissions

Pipeline Designer uses a dedicated Postgres role called `visual_pipeline_user` (VPU) to execute all pipeline operations against the AI Database (AIDB) extension. Understanding how VPU works is important because it directly affects what pipelines are visible in the Pipeline Designer, what tables pipelines can access, and how permissions must be configured.

What is VPU?

`visual_pipeline_user` is a Postgres role that you create manually as a prerequisite before using Pipeline Designer. The role is not created automatically. See *Creating VPU role* for the setup steps. The role has basic connectivity once created, but it does not have the grants needed to read your data or create pipeline objects. You must configure those permissions yourself (see *Configuring VPU permissions*). Pipeline management operations (creating, updating, and deleting pipelines) run under this role - the agent opens a transaction and switches identity using `SET LOCAL ROLE visual_pipeline_user` . Pipeline execution also runs as VPU, though the mechanism varies by processing mode (see *Execution flow* for details).

VPU exists for two reasons:

1. **Privilege isolation.** Pipeline SQL doesn't run with superuser or administrator privileges. If a pipeline operation triggers unexpected behavior in the AIDB extension, the blast radius is limited to what VPU is authorized to access.
2. **Pipeline Designer visibility control.** Pipeline Designer displays only the pipelines and knowledge bases owned by VPU. This separation keeps pipelines managed through Pipeline Designer distinct from those created through other means (SQL, scripts, third-party tools). Pipeline Designer doesn't attempt to manage pipelines it didn't create, avoiding potential conflicts with incompatible configurations.

For environment setup steps (extension installation, cluster configuration, platform roles, and source table grants), see *Setup prerequisites* .

How VPU affects visibility

When a pipeline is created through Pipeline Designer, AIDB stamps the pipeline's `owner_role` as `visual_pipeline_user` . The EDB Postgres AI agent periodically collects pipeline and knowledge base data from AIDB and filters for `owner_role = 'visual_pipeline_user'` . Only matching objects are reported to the Hybrid Manager (HM) control plane and displayed in the Pipeline Designer.

This has a direct consequence: pipelines and knowledge bases created outside Pipeline Designer aren't visible in Pipeline Designer. If you create a pipeline through SQL (for example, using `psql` as the `edb_admin` role), it will function correctly at the database level but won't appear in Pipeline Designer's pipeline list or knowledge base views.

There is no AIDB function to change a pipeline's `owner_role` after creation. If you need a SQL-created pipeline to appear in the Portal, you must create it while assuming the VPU role. Your Postgres role must have been granted VPU (`GRANT visual_pipeline_user TO your_role`) to perform the role switch:

```
-- SET LOCAL ROLE scopes the identity change to the current transaction.
BEGIN;
SET LOCAL ROLE visual_pipeline_user;
SELECT aidb.create_pipeline(...);
COMMIT;
```

> **Note**
>
> All pipelines created this way are stamped with *owner_role = 'visual_pipeline_user'* , regardless of which user executed the *SET LOCAL ROLE* . There is no per-user attribution in the pipeline metadata. Use *SET LOCAL ROLE* (transaction-scoped) rather than *SET ROLE* (session-scoped) to avoid accidentally running subsequent SQL as VPU.

Configuring VPU permissions

VPU doesn't automatically have access to your data tables or the ability to create pipeline objects. The permissions VPU needs depend on the pipeline's processing mode, as described in the overview below. If any required permission is missing, pipeline creation or execution will fail with a permission error.

Required VPU permissions

1. **Create destination tables.** AIDB creates a destination table (named `pipeline_<name>`) in the same schema as the source table. VPU needs `CREATE ON SCHEMA` for that schema. This grant is required for every processing mode.
2. **Read source data and create processing triggers.** The grant VPU needs to read the source table depends on the processing mode:
   - **On Demand mode** reads the source table but doesn't attach triggers. VPU only needs `SELECT` on the source table:

```
GRANT SELECT ON <schema>.<source_table> TO visual_pipeline_user;
```

```
- **Live and Background modes** attach triggers to the source table so that new or␣
↪updated rows are processed automatically. For these modes, VPU must be granted␣
↪membership in the source table's owner role:
```

```
GRANT <source_table_owner_role> TO visual_pipeline_user;
```

This single grant gives VPU both the read access and the privileges needed to create and delete the processing triggers on the source table. Replace `<source_table_owner_role>` with the role that owns the source table.

Membership in the owner role is required rather than a narrower `TRIGGER` grant because Postgres lets only a table's owner drop triggers on it. The `TRIGGER` privilege allows creating triggers but not deleting them, so VPU would be unable to clean up its triggers when a pipeline is updated or deleted.

The subsections below show the exact `GRANT` statements for each topology.

Tables in the public schema

On Postgres 15 and later, `CREATE` on the `public` schema is no longer granted to `PUBLIC` by default. VPU therefore needs an explicit `CREATE ON SCHEMA public` grant regardless of whether the cluster is managed or self-managed. The `aidb_users` role grants access to AIDB extension objects but doesn't cover `SELECT` on user-created tables, schema-level `CREATE` , or the privileges needed to create and delete processing triggers.

Connect to the database as an administrator and run the grants for your processing mode.

For On Demand mode:

```
GRANT SELECT ON public.<source_table> TO visual_pipeline_user;
GRANT CREATE ON SCHEMA public TO visual_pipeline_user;
```

For Live and Background modes:

```
GRANT <source_table_owner_role> TO visual_pipeline_user;
GRANT CREATE ON SCHEMA public TO visual_pipeline_user;
```

Replace `<source_table>` with the actual table name and `<source_table_owner_role>` with the role that owns the source table.

Tables in user-created schemas

For tables in custom schemas, you must grant VPU access explicitly. Connect to the database as an administrator and run the grants for your processing mode.

For On Demand mode:

```
GRANT USAGE ON SCHEMA <schema> TO visual_pipeline_user;
GRANT SELECT ON <schema>.<source_table> TO visual_pipeline_user;
GRANT CREATE ON SCHEMA <schema> TO visual_pipeline_user;
```

For Live and Background modes:

```
GRANT USAGE ON SCHEMA <schema> TO visual_pipeline_user;
GRANT <source_table_owner_role> TO visual_pipeline_user;
GRANT CREATE ON SCHEMA <schema> TO visual_pipeline_user;
```

Replace `<schema>` and `<source_table>` with your actual schema and table names, and `<source_table_owner_role>` with the role that owns the source table. `USAGE` allows VPU to see objects in the schema. `CREATE` allows it to create destination tables there.

Important

For Live and Background modes, grant only the role that owns the source table (*GRANT <source_table_owner_role> TO visual_pipeline_user*). Do not grant VPU membership in broad administrative roles. A common but overly broad shortcut is *GRANT edb_admin TO visual_pipeline_user* , which effectively gives VPU administrator-level access to the entire database, defeating the privilege isolation that VPU is designed to provide.

Tables on PGD clusters

Postgres Distributed (PGD) clusters require the same grants as any other topology, plus additional BDR (Bi-Directional Replication) considerations. All grants must be consistent across every BDR node, not just the write leader. A grant that exists on the write leader but is missing on a subscriber node will cause replication failures when the trigger fires on that node.

Apply the base grants on every BDR node (or use a replicated DDL mechanism).

For On Demand mode:

```
- - Run on each BDR node:
GRANT USAGE ON SCHEMA <schema> TO visual_pipeline_user;
GRANT SELECT ON <schema>.<source_table> TO visual_pipeline_user;
GRANT CREATE ON SCHEMA <schema> TO visual_pipeline_user;
```

For Live and Background modes on PGD clusters running AIDB 7.3.2 or later:

```
-- Run on each BDR node:
GRANT USAGE ON SCHEMA <schema> TO visual_pipeline_user;
GRANT <source_table_owner_role> TO visual_pipeline_user;
GRANT CREATE ON SCHEMA <schema> TO visual_pipeline_user;
```

For Live and Background modes on PGD clusters running AIDB earlier than 7.3.2, transfer ownership of the source table to VPU instead of granting the owner role:

```
-- Run on each BDR node:
GRANT USAGE ON SCHEMA <schema> TO visual_pipeline_user;
ALTER TABLE <schema>.<source_table> OWNER TO visual_pipeline_user;
GRANT CREATE ON SCHEMA <schema> TO visual_pipeline_user;
```

Replace `<schema>` and `<source_table>` with your actual schema and table names, and `<source_table_owner_role>` with the role that owns the source table. For tables in `public` , omit the `USAGE` grant.

Trigger handler ownership on PGD clusters

When you enable Live or Background processing, AIDB creates a per-pipeline SECURITY DEFINER wrapper function and sets its owner to the pipeline's `owner_role` (which is `visual_pipeline_user` for Portal-created pipelines). BDR enforces ownership constraints on trigger functions attached to replicated tables: it checks that the trigger function owner satisfies replication constraints for the relation on all nodes.

On AIDB 7.3.2 and later, granting VPU the source table's owner role (`GRANT <source_table_owner_role> TO visual_pipeline_user`) satisfies these constraints. On AIDB versions earlier than 7.3.2, granting ownership of the source table to VPU by having it inherit the owner role is not sufficient. It is necessary to explicitly make VPU owner of the source table:

```
ALTER TABLE <schema>.<source_table> OWNER TO visual_pipeline_user;
```

This is broader than the owner-role grant because it makes VPU the owner of the source table. Use it only as a PGD-specific workaround on AIDB versions earlier than 7.3.2. See *Trigger function ownership on PGD clusters* for more detail on the underlying constraint.

Security and isolation considerations

The preceding sections cover VPU setup and permissions grants. The topics below address the architectural implications of VPU's shared-identity model for security, credential isolation, and object ownership.

Shared identity

All pipelines created through Pipeline Designer run as the same Postgres role. There's no per-HM-user isolation at the database level. HM user A's pipeline and HM user B's pipeline both execute as `visual_pipeline_user` and have access to the same set of tables. Isolation between users' pipelines is managed at the HM project level (through Pipeline Designer access controls), not at the Postgres level.

This also means there is no per-user audit trail at the Postgres level. All pipeline activity in the database logs appears as operations by `visual_pipeline_user` .

Shared model credentials

Model credentials (API keys for external inference services) are stored as FDW user mappings and are shared across all database users. Any pipeline on the same database, regardless of which HM user created it, can use any registered model. There is no per-pipeline or per-user credential isolation.

Objects created by VPU

When Pipeline Designer creates a pipeline, VPU becomes the owner of several database objects:

- The pipeline's destination table (for example, `pipeline_my_pipeline`).
- The knowledge base vector table (if the pipeline includes a KnowledgeBase step).
- Per-pipeline SECURITY DEFINER wrapper functions (for Live and Background processing modes). These wrappers call the actual AIDB trigger handler and are owned by VPU so that they execute with VPU's privileges regardless of the invoking role.
- Pipeline state tracking tables used internally by AIDB.

These objects persist until the pipeline is deleted through Pipeline Designer.

Common VPU issues

"Permission denied for table" during pipeline creation

Cause: VPU is missing one or more of the required permissions for the source table or its schema. For On Demand mode, this is `SELECT` or `CREATE ON SCHEMA` . For Live and Background modes, it is membership in the source table's owner role (`GRANT <source_table_owner_role> TO visual_pipeline_user`) or `CREATE ON SCHEMA` .

Fix: Grant the required permissions as described in *Configuring VPU permissions* .

Pipeline created via SQL does not appear in Pipeline Designer

Cause: The pipeline was created under a role other than `visual_pipeline_user` , so its `owner_role` doesn't match the Pipeline Designer's filter.

Fix: Recreate the pipeline using `SET LOCAL ROLE visual_pipeline_user` before calling `aidb.create_pipeline()` . See *How VPU affects visibility* for the full procedure.

Pipeline works but knowledge base is not visible

Cause: Same as above. The knowledge base inherits the `owner_role` from the pipeline. If the pipeline wasn't created under VPU, neither the pipeline nor the KB will appear in Pipeline Designer.

Trigger creation fails on PGD clusters

Cause: When AIDB enables Live or Background processing, it creates a per-pipeline SECURITY DEFINER wrapper function owned by VPU. BDR requires that the trigger function owner satisfies replication constraints on the source table. If VPU lacks the necessary grants on any BDR node, trigger creation fails.

Fix: Apply consistent `GRANT` statements on all PGD nodes as described in *Tables on PGD clusters* . On AIDB versions earlier than 7.3.2, the owner-role grant isn't sufficient; see the *Trigger handler ownership on PGD clusters* fallback for transferring source table ownership to VPU.

Limitations

The sections below cover intentional design boundaries of the current Pipeline Designer implementation and known issues with available workarounds. For release-specific details, see the *Hybrid Manager release notes* .

Pipeline creation and structure

Maximum of 10 steps per pipeline

A pipeline supports a maximum of 10 processing steps.

Steps cannot be modified after creation

Once a pipeline is deployed, its step structure (step types, order, and configuration) is fixed. You can't add, remove, or reorder steps. To change a pipeline's structure, delete it and create a new one.

Pipeline name length limit

Pipeline names are limited to 46 characters. AIDB reserves up to 17 characters of the Postgres identifier limit for internal object name suffixes (state tables, triggers, error logs).

Destination table name collision

If a table named `pipeline_{your_pipeline_name}` already exists in the target schema and isn't a registered knowledge base vector table, pipeline creation will fail. Either rename the existing table or choose a different pipeline name.

External PGD and CNP clusters not supported

Self-managed cluster support is limited to single-instance Postgres servers. External PGD (Postgres Distributed) clusters and Cloud Native Postgres (CNP) operator-managed clusters aren't eligible as pipeline targets, even if registered with Hybrid Manager (HM) through the EDB Postgres AI agent.

Multi-step pipelines

ChunkText overlap setting not available

The `overlap_length` parameter for the ChunkText step (which controls how much text overlaps between consecutive chunks) is supported by AI Database (AIDB) but not yet configurable through Pipeline Designer. If you need chunk overlap, create the pipeline through SQL directly. See the ChunkText step reference for parameter details.

Inference routing on remote clusters

HM-hosted and HM-proxied models require primary-location connectivity

HM-hosted KServe models and HM external inference service proxies both run on the primary location. Clusters on secondary locations or self-managed instances don't have network connectivity to the primary location's model-serving endpoints by default. Models from these two paths appear in the model picker (because they are registered in the AIDB model registry) but fail at execution time when the pipeline runs on a cluster that can't reach the primary location.

Resolution: For pipelines on clusters without primary-location connectivity, use AIDB-native models registered via `aidb.create_model()` (which contact the provider directly from the Postgres process) or built-in AIDB models that run locally. See *How models reach pipeline steps* for the complete model availability breakdown.

Processing modes

Live mode blocks application writes

Live processing executes within the triggering transaction. An `INSERT` , `UPDATE` , or `DELETE` on the source table doesn't return until the pipeline finishes processing the affected row. For steps involving inference model calls (SummarizeText, KnowledgeBase embedding), this can add significant latency to the application's write path.

Recommendation: Use Background mode for production workloads unless real-time processing is strictly required.

Background worker overhead

The background processing worker polls all databases on the cluster for pipelines in Background mode. On deployments with many databases or many pipelines, this polling can create measurable overhead.

Recommendation: Reduce polling frequency by increasing the `background_sync_interval` for pipelines where near-real-time processing isn't required. See the AIDB background worker documentation for configuration details.

Knowledge bases

No standalone knowledge base creation

Knowledge bases can only be created as a side effect of deploying a pipeline with a KnowledgeBase terminal step. There is no standalone "create knowledge base" operation in Pipeline Designer.

A KnowledgeBase step alone is rarely sufficient for useful results. Raw text data typically needs at least a ChunkText step before the KnowledgeBase step to split content into appropriately sized segments for embedding. A pipeline with only a KnowledgeBase step would attempt to embed entire source rows as single vectors, which can exceed model token limits or produce poor retrieval quality.

Model and access control

No per-user model isolation

Model credentials are shared across all database users on a cluster. There is no per-user or per-pipeline credential isolation. See *Shared model credentials* for details.

Known issues

Trigger function ownership on PGD clusters

On AIDB earlier than 7.3.2, `visual_pipeline_user` must own the trigger handler functions directly, rather than acquiring ownership through role inheritance. Without direct ownership, the AIDB trigger handler functions (`aidb.pipeline_background_trigger_handler` and `aidb.pipeline_live_trigger_handler`) cause Live and Background processing modes to fail on PGD clusters.

Workaround: Transfer ownership of the trigger handler functions to `visual_pipeline_user` :

```
ALTER FUNCTION aidb.pipeline_background_trigger_handler OWNER TO visual_pipeline_user;
ALTER FUNCTION aidb.pipeline_live_trigger_handler OWNER TO visual_pipeline_user;
```

This issue will be resolved in a future AIDB release.

Knowledge base replication on PGD clusters

On AIDB earlier than 7.3.2, the `bdr_setup()` function doesn't include knowledge base tables in BDR replication. Without these tables, knowledge base registry entries and knowledge base–pipeline associations created on the write leader aren't replicated to other PGD nodes. This limitation can cause inconsistencies after a write leader switchover, where the new leader has no record of knowledge bases or their pipeline bindings.

Workaround: Run the following SQL on the PGD cluster to add the missing tables and sequence to replication:

```
SELECT bdr.alter_sequence_set_kind(aidb.knowledge_base_registry_id_seq::regclass,␣
↪galloc, 1);
SELECT bdr.replication_set_add_table(aidb.knowledge_base_registry);
SELECT bdr.replication_set_add_table(aidb.knowledge_base_pipeline);
```

This bug is resolved in AIDB 7.3.2.

6.5.5 Operate & Observe AI Factory

6.5.5.1 Observability (Agent Factory on HM)

6.5.5.1.1 Observability in Hybrid Manager

When you deploy Agent Factory components on Hybrid Manager, the platform gives you integrated observability for both **Model Serving** (KServe) and **Langflow** workloads.

Model Serving (KServe)

- Use the platform dashboards and logs to monitor **InferenceServices** and GPU workloads.
- Metrics include request latency, throughput, GPU/CPU utilization, and error rates.
- See: *Monitor InferenceService*

Langflow

- Monitor Langflow logs, pod health, replica count, and flow availability via Prometheus and Grafana.
- See: *Troubleshooting Langflow in Hybrid Manager*

6.5.5.1.2 Accessing metrics and logs

- **Dashboards.** Hybrid Manager surfaces Agent Factory metrics through the built-in observability dashboards (Grafana).
- **Logs.** Use `kubectl logs` or the HM UI log viewer to inspect Langflow pods, KServe InferenceServices, and retriever jobs.

6.5.5.1.3 Typical commands

For cluster-level troubleshooting or custom dashboards, you can also pull metrics and logs directly:

```
#  Check logs for a running InferenceService

kubectl logs -n <project-namespace> svc/<model-service-name>

#  Port-forward Grafana if running locally
```

(continues on next page)

(continued from previous page)

```
kubectl port-forward -n observability svc/grafana 3000:3000
```

6.5.5.1.4 Grafana dashboard setup for Agent Factory metrics

Hybrid Manager includes a Grafana instance in the observability stack. You can access it through the Launchpad or configure dashboards for Agent Factory-specific metrics.

Access Grafana

Navigate to the Launchpad in the Hybrid Manager Portal and click on the Grafana tile.

Recommended dashboard panels

Build or import a dashboard with the following panels to cover Agent Factory health at a glance:

Panel	Metric	Visualisation
Inference request rate	*rate(kserve_request_duration_seconds_count[5m])*	Time series
P95 inference latency	*histogram_quantile(0.95, rate(kserve_request_duration_seconds_bucket[5m]))*	Time series
Inference error rate	*rate(kserve_request_duration_seconds_count{response_code!~"2.."}[5m])*	Time series
Model server replica count	*kube_deployment_status_replicas_ready{deployment=~".*inferenceservice.*"}*	Stat
Langflow replica count	*kube_deployment_status_replicas_ready{deployment=~".*langflow.*"}*	Stat
GPU utilisation	*DCGM_FI_DEV_GPU_UTIL*	Gauge

Import a pre-built dashboard

KServe publishes a community Grafana dashboard. To import it:

1. In Grafana, go to **Dashboards → Import**.
2. Enter the dashboard ID or paste the JSON from the KServe community dashboards .
3. Select your Prometheus data source.
4. Select **Import**.

6.5.5.1.5 Key takeaway

- **Model Serving** → monitored as KServe services with full metrics.
- **Langflow** → pod health, replica count, and flow availability monitored via Prometheus and Grafana.
- **Hybrid Manager** provides a unified observability layer — no separate monitoring stack to configure.
- **Enterprise monitoring** → extend the built-in stack with custom Grafana dashboards.

6.5.5.2 Troubleshooting Agent Factory on Hybrid Manager

6.5.5.2.1 Diagnostic Overview

This guide provides systematic troubleshooting procedures for Agent Factory components within Hybrid Manager environments. Issues typically fall into three categories: infrastructure problems, model serving failures, and application-level errors.

6.5.5.2.2 Quick Reference Links

- *Langflow FAQ*
- *GPU Setup Guide*
- *InferenceService Creation*
- *Troubleshooting Langflow in Hybrid Manager*

6.5.5.2.3 Infrastructure Issues

GPU Resource Problems

Symptoms

- InferenceService pods remain pending
- "Insufficient nvidia.com/gpu" events
- Model initialization timeouts

Diagnostic Steps

```
#  Check GPU node availability

kubectl get nodes -l nvidia.com/gpu=true

#  Verify GPU resource allocation

kubectl describe node <gpu-node-name> | grep -A 5 "Allocated resources"

#  Check GPU driver status

kubectl logs -n gpu-operator nvidia-driver-daemonset-<pod>
```

Common Resolutions

Missing GPU Labels Nodes with GPUs must be properly labeled:

```
kubectl label nodes <node-name> nvidia.com/gpu=true
```

GPU Taint Issues Verify taint configuration for dedicated GPU scheduling:

```
kubectl taint nodes <node-name> nvidia.com/gpu=NoSchedule
```

Driver Compatibility Ensure NVIDIA driver version matches CUDA requirements for NIM containers. Check driver logs in the `gpu-operator` namespace for initialization errors.

Storage Access Failures

Symptoms

- Model download failures
- "ImagePullBackOff" status
- Profile cache errors in air-gapped environments

Diagnostic Procedures

```
#  Check secret configuration

kubectl get secret nvidia-nim-secrets -n default -o yaml

#  Verify image pull secret

kubectl get secret ngc-cred -n <namespace> -o yaml

#  Test registry connectivity

kubectl run test-pull --image=nvcr.io/nim/nvidia/nvclip:latest --dry-run=client
```

Resolution Strategies

Registry Authentication Recreate NGC credentials if authentication fails:

```
kubectl delete secret ngc-cred -n default
kubectl create secret docker-registry ngc-cred \
  --docker-server=nvcr.io \
  --docker-username=$oauthtoken \
  --docker-password=<NGC_API_KEY> \
  -n default
```

Air-Gapped Environments Verify profile cache availability in object storage and correct path configuration in model deployment specifications.

6.5.5.2.4 Model Serving Failures

InferenceService Not Ready

Symptoms

- InferenceService shows "NotReady" status
- Predictor pods crash or restart
- Health check failures

Investigation Commands

```
#  Check InferenceService status

kubectl get inferenceservice <name> -n <namespace>
```

(continues on next page)

(continued from previous page)

```
#  Examine detailed conditions

kubectl describe inferenceservice <name> -n <namespace>

#  Review pod logs

kubectl logs <predictor-pod> -n <namespace> -c kserve-container
```

Common Causes and Fixes

Insufficient Memory NIM models require substantial memory. Check pod resource requests:

```
kubectl describe pod <predictor-pod> -n <namespace> | grep -A 3 "Requests"
```

Increase memory allocation in InferenceService specification if necessary.

Model Loading Timeout Large models may exceed default initialization timeouts. Adjust readiness probe settings in the InferenceService configuration.

Profile Mismatch Ensure cached profiles match GPU architecture. List compatible profiles and verify cache contents.

High Inference Latency

Symptoms

- Response times exceed SLA requirements
- Token generation rates below expectations
- GPU underutilization

Performance Analysis

```
#  Monitor GPU utilization

kubectl exec <predictor-pod> -n <namespace> -- nvidia-smi

#  Check request metrics

kubectl port-forward -n <namespace> svc/<service-name> 9090:9090

#  Access metrics at localhost:9090/metrics
```

Optimization Approaches

Batch Size Tuning Adjust batch processing parameters in ServingRuntime configuration for improved throughput.

Replica Scaling Add InferenceService replicas to distribute load:

```
kubectl scale inferenceservice <name> -n <namespace> --replicas=3
```

Resource Allocation Verify GPU memory allocation matches model requirements. Insufficient GPU memory forces CPU fallback, severely impacting performance.

6.5.5.2.5 Langflow and Gen AI application issues

For issues with Langflow flows and EDB components — including EDB Knowledge Base retrieval failures, EDB Model Server errors, component configuration after flow import, and flow-level quality issues — see *Troubleshooting Langflow in Hybrid Manager* .

6.5.5.2.6 Log Analysis

Log Locations

Agent Factory components generate logs at multiple levels:

System Logs

- Kubernetes events: `kubectl get events -n <namespace>`
- Node logs: `/var/log/messages` or journalctl on GPU nodes

Application Logs

- InferenceService: Predictor pod container logs
- Langflow flows: Application pod logs
- Model Library: HM control plane logs

Metrics and Monitoring

- Prometheus metrics: Available through HM monitoring stack
- Custom dashboards: Accessible via Grafana in HM console

Log Aggregation

Configure centralized logging for comprehensive analysis:

```
#  Stream logs from multiple components

kubectl logs -f -l app=inference-service -n <namespace> --all-containers

#  Export logs for analysis

kubectl logs <pod> -n <namespace> --since=1h > diagnostic.log
```

6.5.5.2.7 Alert Configuration

Critical Alerts

Configure monitoring alerts for critical conditions:

GPU Availability Alert when GPU nodes become unavailable or GPU allocation fails.

Model Health Monitor InferenceService readiness and restart frequency.

Performance Degradation Track inference latency percentiles and token generation rates.

Resource Exhaustion Alert on memory pressure, GPU memory saturation, or storage capacity.

Alert Integration

Integrate alerts with organizational notification systems through HM alert manager configuration.

6.5.5.2.8 Escalation Procedures

Support Resources

When internal troubleshooting proves insufficient:

1. **Documentation Review**
   - Consult *Agent Factory Hub*
   - Review model-specific documentation
2. **Community Resources**
   - NVIDIA NIM documentation for model-specific issues
   - KServe community for serving infrastructure problems
3. **EDB Support**
   - Collect diagnostic bundles using HM support tools
   - Include relevant logs, configurations, and error messages
   - Reference specific component versions and deployment specifications

Diagnostic Information Collection

Prepare comprehensive diagnostic information:

```
#  Generate support bundle

kubectl cluster-info dump --output-directory=./cluster-dump

#  Collect Agent Factory specifics

kubectl get inferenceservice -A -o yaml > inferenceservices.yaml
kubectl get pods -A -l serving.kserve.io/inferenceservice -o wide > serving-pods.txt
kubectl describe nodes -l nvidia.com/gpu=true > gpu-nodes.txt
```

6.5.5.2.9 Preventive Measures

Regular Health Checks

Implement proactive monitoring:

- Weekly GPU driver and operator status verification
- Daily InferenceService health assessment
- Continuous performance baseline tracking

Capacity Planning

Monitor resource trends:

- GPU utilization patterns
- Memory consumption growth
- Storage usage projections
- Request volume trends

Update Management

Maintain component currency:

- Track NVIDIA NIM model updates
- Monitor security advisories
- Plan maintenance windows for updates
- Test updates in non-production environments

CHAPTER 7

Analytics in Hybrid Manager

Perform analytics management for your Advanced High Availability (AHA) and Distributed High Availability (DHA) clusters directly from the Hybrid Manager (HM) console, including table management, tiered storage configuration, and catalog exploration.

AHA and DHA clusters include Postgres Analytics Accelerator (PGAA), a high-performance extension that enables Postgres to query large-scale data stored in open table formats like Delta Lake, Apache Iceberg, and Parquet.

Manage your cluster's analytics end-to-end from the HM console by deploying a single-node AHA or a minimal DHA cluster in your project. A dedicated cluster gives you a unified view of all analytical tables across your AHA and DHA clusters in a HM project, with table management, tiered storage configuration, and catalog exploration available from a single interface.

For concepts, architecture, and reference documentation, see the PGAA documentation .

7.1 Configuring tiered storage

Replicate and offload Postgres tables to object storage, and query them with standard Postgres SQL without migrating data out of your cluster. Advanced High Availability (AHA) and Distributed High Availability (DHA) are EDB Postgres Distributed (PGD)-based cluster types managed by Hybrid Manager (HM), with Postgres Analytics Accelerator (PGAA) pre-installed and configured automatically.

For concepts, architecture, and reference documentation, see *Using tiered storage* in the PGD documentation.

7.1.1 Managing analytics for your cluster

Object storage for your AHA and DHA clusters is configured at HM installation when *creating a storage secret* . HM automatically provisions an Iceberg catalog for each data group backed by that object storage, and configures it as the analytics write destination. PGAA is pre-installed and the catalog is ready to use, with no manual setup required. You can view and manage catalogs under **Estate > Catalogs**, or create additional shared catalogs for cross-cluster querying. See *Managing catalogs* for details.

Use **Enable Analytics** and **Disable Analytics** from the **Actions** menu to control analytics at the data group level. Disabling analytics prevents new replication and offload operations on all tables in that data group. Existing offloaded tables remain accessible and the catalog configuration is preserved.

7.1.2 Viewing tables

The **Tables** tab in your cluster's detail view lists all tables in the cluster, together with their type, replication status, and available actions.

!!!note By default, the `edb.collect_table_statistics` configuration parameter is set to `pgaa` , which collects statistics for Iceberg tables only. To include Heap and HTAP tables, set it to `all` via **Edit Cluster > Data Groups > DB Configuration**. No cluster restart is required.

Each table has one of the following types:

- **Heap** — a standard Postgres table with no PGAA metadata.
- **HTAP** (Hybrid Transactional and Analytics Processing) — a heap table with analytics replication enabled. The table remains fully transactional while data is continuously replicated to object storage. See *Replicating to analytics* .
- **Iceberg** — a cold analytics table stored entirely in object storage. The local copy has been truncated. See *Offloading to analytics* .
- **Tiered** — a partitioned table managed by PGD AutoPartition , where cold partitions are Iceberg. Hot partitions are HTAP if real-time analytics replication is enabled, or Heap if not. See *Converting to a tiered table* .

Selecting a table row displays a summary panel at the top of the tab with storage metrics for that table: the number of rows and total size on disk, and the number of rows and total size in object storage. These metrics reflect the current state of the table across both storage tiers.

7.1.3 Converting to a tiered table

Convert a heap table to a tiered table to restructure it as a range-partitioned table managed by PGD AutoPartition . Cold partitions are offloaded to Iceberg format automatically when they age past the analytics offload period. Hot partitions are HTAP and replicated to object storage continuously if real-time analytics replication is enabled, or remain as Heap if not.

The table must meet the following requirements before conversion:

- It has a primary key.
- The partition column is a `DATE` or `TIMESTAMP` column that is also part of the primary key.
- It has no foreign key constraints or sequences. Drop and recreate these after conversion.

To convert a table:

1. In the HM console, go to **Estate > Postgres** and select your cluster.
2. Select the **Tables** tab.
3. In the table row, open the three-dot menu and select **Convert to Tiered Table**.
4. Fill in the partitioning options:

- **Partition Column** — the `DATE` or `TIMESTAMP` column to partition by.
- **Initial Lower Bound** — the start date for the first partition.
- **Analytics Offload Period** — how long partitions stay hot before being offloaded to Iceberg.
- **Enable real-time analytics replication** — when enabled, hot partitions are replicated to object storage continuously as HTAP tables. When disabled, hot partitions remain as Heap tables and are bulk-offloaded when they age out of the offload window.
- **Partition Increment** — the time interval for each partition.
- **Minimum Advance Partitions** — the minimum number of future partitions to keep created in advance.

- **Maximum Advance Partitions** — the maximum number of future partitions to maintain.

5. Select **Convert to Tiered Table** and wait for the conversion to complete.

After conversion, the table type changes to **Tiered**. Expanding the table row shows the individual partitions with their type (Heap, HTAP, or Iceberg) and boundary expressions. The **Tables** tab also shows a partition distribution chart with the breakdown across partition types. Partition lifecycle is managed automatically by PGD AutoPartition after conversion.

The HM console doesn't provide actions to modify a tiered table after conversion. To change settings such as the offload period or partition increment, rerun `bdr.autopartition()` directly in your database client. See *Implementing tiered tables* in the PGD documentation for details on modifying tiered table settings.

7.1.4 Replicating to analytics

Enable replication on a heap table to start continuous replication to object storage. The table type changes to **HTAP** and data changes are streamed to the cluster's pre-configured data group catalog in near real-time. The catalog name is shown in the confirmation dialog.

The table must meet the following requirements before enabling replication:

- It has a primary key.

To enable replication:

1. In the HM console, go to **Estate > Postgres** and select your cluster.
2. Select the **Tables** tab.
3. In the table row, open the three-dot menu and select **Enable Replication**.
4. Review the confirmation dialog and select **Enable Replication**.

The table type changes to **HTAP** and the replication status shows **Enabled** once the initial sync is complete.

Once a table is HTAP, the following actions are available:

- **Disable Replication** — Stops replication and removes PGAA metadata, reverting the table to Heap. Data previously replicated to object storage isn't deleted automatically and remains in the catalog.
- **Offload Table** — Converts the table to a cold Iceberg table and truncates the local copy.
- **Trigger Compaction** — Consolidates fragmented Parquet files in object storage to improve query performance and reduce storage overhead.

!!!note Enabling replication on a table that previously had replication disabled fails due to leftover catalog metadata. To resolve the error, re-enable replication directly in your database client using the `purge_analytics_target` parameter:

```
CALL pgaa.enable_analytics_replication(<schema>.<table>::regclass, true);
```

See pgaa.enable_analytics_replication in the PGAA documentation.

7.1.5 Offloading to analytics

Offload an HTAP table to convert it to a cold Iceberg table, truncate the local copy, and leave only the analytics copy in object storage. The confirmation dialog shows the destination catalog (the cluster's pre-configured data group catalog) where the data will be stored.

The table must meet the following requirements before offloading:

- No other tables can have foreign key constraints pointing to it.

To offload a table:

1. In the HM console, go to **Estate > Postgres** and select your cluster.
2. Select the **Tables** tab.
3. In the HTAP table row, open the three-dot menu and select **Offload Table**.
4. Review the confirmation dialog and select **Offload**.

The table type changes to **Iceberg** with replication disabled. Iceberg tables are read-only and don't support direct writes. `INSERT` , `UPDATE` , and `DELETE` operations aren't supported. To write to the table again, restore it to **HTAP** first.

From an Iceberg table, you can:

- **Restore Table** — Moves the data back to local storage. The table reverts to **HTAP**. The copy in object storage isn't deleted automatically.
- **Trigger Compaction** — Consolidates fragmented Parquet files in object storage to improve query performance and reduce storage overhead.

7.1.6 Triggering compaction

Use compaction to consolidate fragmented Parquet files in object storage into larger, optimized files. Compaction improves query performance and reduces storage overhead. Compaction is available on HTAP and Iceberg tables.

To trigger compaction:

1. In the HM console, go to **Estate > Postgres** and select your cluster.
2. Select the **Tables** tab.
3. In the table row, open the three-dot menu and select **Trigger Compaction**.
4. Review the confirmation dialog and select **Trigger Compaction**.

7.2 Managing catalogs

Attach Iceberg catalogs to your clusters to query analytical data across your estate without having to move data between clusters. Hybrid Manager (HM) automatically provisions an Iceberg catalog for each data group in an Advanced High Availability (AHA) or Distributed High Availability (DHA) cluster, backed by the *object storage configured at installation* . All replication and offload operations write to this catalog. View your catalogs under **Estate > Catalogs**.

For Iceberg catalog concepts, see Integrating with Iceberg catalogs in the Postgres Analytics Accelerator (PGAA) documentation.

7.2.1 Managing your cluster's catalog

View the catalogs associated with your AHA or DHA cluster from the **Catalogs** tab in the cluster's detail view. Use the following actions to make table metadata available for querying:

- **Import** — Imports the table metadata from the catalog into your cluster. Use **Import** for a point-in-time snapshot and run it again whenever you want to refresh the metadata.
- **Enable Sync** — Continuously synchronizes table metadata from the catalog into your cluster, creating analytical tables on the cluster whenever new ones are added to the catalog. The sync status changes to **Enabled**. Use **Enable Sync** when tables are actively changing and you want queries to always reflect the latest state.

7.2.2 Querying analytical data across clusters

Attach a catalog from another cluster within the same project to query its data from your cluster using PGAA. For example, offload data from an AHA cluster and query it from a DHA cluster, or vice versa.

To attach a catalog:

1. Go to the cluster's **Catalogs** tab, select **Add Catalog > Add Managed Catalog**, and search for the catalog. Only catalogs that already exist in the project appear in the search results.
2. Use **Import** or **Enable Sync** to make the catalog's tables available for querying.

Catalog-attached tables appear as **Iceberg** type on the importing cluster, regardless of their type on the source cluster. They're read-only and don't support `INSERT` , `UPDATE` , or `DELETE` .

Use **Restore Table** on an Iceberg table managed by another cluster's catalog to copy the data into your own cluster as a local heap table. The source cluster and its data are unaffected, but the restored table is no longer linked to the catalog.

!!!warning Restoring a table from another cluster's catalog doesn't preserve the original schema. Because Iceberg's type system is a subset of Postgres's, constraints (primary keys, `NOT NULL` , unique) are lost, column types are coerced to generic equivalents, and default values are removed.

CHAPTER 8

Sovereign Data and Agent Factory

Sovereign Data and Agent Factory is a fully integrated hardware and software platform co-designed by EDB and Supermicro®. It enables organizations to deploy production-grade Postgres® and AI workloads in their own data centers without external dependencies.

This engineered system is optimized for sovereignty, security, and scale. It's preassembled, lifecycle-managed, and ready to run advanced Postgres, vector, and AI pipelines as soon as it's installed.

8.1 Why choose a hardware-based system?

Cloud-first isn't always an option. Sovereign Data and Agent Factory exists for environments where:

- Data and model control are legally or operationally required
- Cloud GPU access is inconsistent, oversubscribed, or restricted
- Sovereign AI demands zero external inference or data movement
- Unified observability across Postgres, vector, and AI is critical
- Lifecycle reliability matters across hardware and software

This system removes friction from modern workloads by tightly coupling the hardware, orchestration platform, and supported runtimes so you can focus on outcomes, not infrastructure.

8.2 What's inside

Delivered as a rack-mounted system into your data center, the platform includes:

- Validated Supermicro servers
- Lifecycle-managed Kubernetes and Hybrid Manager
- Built-in HA Postgres orchestration
- pgvector, embedding pipelines, and model-serving capabilities
- Optional GPU-based AI inference and agent runtime

- Full monitoring, patching, and version upgrade automation

All configuration is handled before delivering access. Once installed, use the Hybrid Manager web interface to begin deploying Postgres clusters, RAG pipelines, and agentic applications immediately.

8.3 Supported workloads

You can run a wide variety of sovereign or offline-capable workloads:

- High-availability Postgres databases
- Hybrid internal DBaaS environments
- Vector search and embedding pipelines
- Retrieval-augmented generation (RAG) using local LLMs
- On-premises AI inference with GPU acceleration
- Internal chatbots, copilots, and agentic orchestration

See *Common workloads and use cases* for examples.

8.4 Hardware configurations

We offer two validated configurations:

- **Core Compute** — For high-performance Postgres and vector workloads.
- **Advanced AI** — Adds GPU-enabled nodes for inference and agent-based systems.

Both configurations are expandable and fully supported. See *System specifications* for details.

8.5 Explore capabilities

- *Concepts and architecture*
- *Glossary and key terms*
- *What to expect during setup*
- *Common workloads and use cases*
- *Langflow FAQ*

To learn more about lifecycle operations, see *Using Hybrid Manager* .

8.5.1 Concepts and architecture

Sovereign Data and Agent Factory combines physical infrastructure, orchestration software, and lifecycle automation into a single system, preassembled and installed in your data center.

Understanding the system architecture and core roles helps clarify how Postgres workloads and AI pipelines are deployed, managed, and scaled.

As a prerequisite, see configuration details in *System specifications* .

8.5.1.1 Node types and roles

The system includes several node roles, each with distinct responsibilities and configurations.

8.5.1.1.1 Administration node

The administration node provides additional software packages, licensing, and support tooling for your Sovereign Data and Agent Factory. You need one administration node per installation.

8.5.1.1.2 Hybrid Manager nodes

Three Hybrid Manager nodes form a high-availability control plane. They manage Kubernetes scheduling, cluster lifecycle, observability, and API services.

Functions include:

- Managing database cluster deployments
- Monitoring 200+ metrics from Postgres, OS, and GPU workloads
- Coordinating updates and patches
- Serving configuration UIs and APIs

These nodes aren't used for running data workloads.

8.5.1.1.3 Compute nodes

Compute nodes are used to run Postgres databases, analytics pipelines, or vector workloads. These nodes support:

- EDB Postgres Advanced Server or Extended Server
- EDB Postgres Distributed (optional)
- Lakehouse formats, such as Apache Iceberg®
- Python-based embedding or transformation pipelines

8.5.1.1.4 AI nodes (optional)

AI nodes are GPU-enabled servers that support large-scale model inference and agentic execution. The system includes two types:

- **2-GPU nodes** for mid-scale model serving (NVIDIA Ada L40S)
- **8-GPU nodes** for LLM and agent runtime (NVIDIA HGX with B200 GPUs)

These nodes integrate with KServe to provide on-premises model serving and inference routing.

8.5.1.2 Orchestration and lifecycle

The system runs a preinstalled Kubernetes cluster managed by Hybrid Manager. You don't directly access Kubernetes. Instead, Hybrid Manager provides a web-based interface to:

- Deploy Postgres clusters with high availability
- Install and monitor vector pipelines or AI workflows
- Scale workloads to additional nodes
- Monitor performance and resource usage

EDB provides lifecycle services for both software and hardware:

- Major and minor version upgrades
- Security patches
- Hardware firmware and BIOS updates
- Automated cluster rebalancing and restart handling

8.5.1.3 What is "sovereign"?

Sovereign in this context means:

- All data and models reside in your physical environment
- No external cloud dependencies or data sharing
- You own the hardware and control network access
- Fully audit-ready, isolated environment for regulated industries

Sovereign Data and Agent Factory is suitable for:

- Government or national data infrastructure
- Financial services subject to local data regulations
- Enterprises with strict cloud restrictions or sensitive IP

8.5.1.4 Supported workload patterns

Each configuration supports a range of use cases:

Workload | Description | Core Compute | Advanced AI ———|————-|————|———- High availability Postgres | Run 3-node HA clusters | □ | □ Embeddings | Convert internal data into vector form | □ | □ Retrieval-augmented generation (RAG) | Combine Postgres + LLMs | □ | □ Inference | Serve models locally using GPUs | □ | □ Agentic workflows | Build AI agents that act autonomously | □ | □

For configuration details, see *System specifications* .

8.5.2 Frequently asked questions

8.5.2.1 Why use this system instead of a cloud service?

Sovereign Data and Agent Factory gives you complete physical and operational control over your infrastructure. It's designed for:

- Environments where data must remain on premises
- Workloads that require consistent, predictable performance
- Organizations that need AI capabilities but can't use cloud APIs or GPU services
- Teams that want unified Postgres and AI orchestration from a single platform

8.5.2.2 Who installs and maintains the hardware?

Supermicro and EDB jointly install the system at your site. All hardware, firmware, and OS-level maintenance is covered under your agreement, along with support for Postgres, Hybrid Manager, and AI tools.

8.5.2.3 What software does it run?

The system runs:

- EDB Postgres Advanced Server or Extended Server
- Hybrid Manager for orchestration and observability
- Agent Factory components
- pgvector and Apache Arrow/Iceberg for vector and analytics workloads
- Containers and KServe for model serving
- Optional: EDB Postgres Distributed for multi-node HA

8.5.2.4 Can I connect this system to cloud services?

You can, but it's not required or assumed. The platform is designed to be fully self-contained. You can selectively enable outbound traffic, depending on your internal policies.

8.5.2.5 What models are supported for AI use?

The system supports standard containerized models, including:

- Embedding models like `bge-base` , `instructor-xl` , `e5`
- Language models like `LLaMA2` , `Mistral` , `Mixtral`
- Task-specific models (summarization, classification, and so on

Custom models can be deployed as long as they conform to the inference container format.

8.5.2.6 Can I monitor external databases?

Yes. Hybrid Manager can monitor and display metrics for external Postgres clusters in observer mode. However, only approved Sovereign Data and Agent Factory nodes can be added to the internal cluster and managed through lifecycle workflows.

8.5.2.7 How do I scale the system?

You can order and rack additional compute nodes or GPU-enabled AI nodes.

8.5.2.8 What's included in support?

- 24/7 global support for software stack (EDB)
- Hardware and firmware support
- Security patches and version upgrades managed through Hybrid Manager
- Lifecycle assistance for expansion, updates, or custom configuration

8.5.3 Glossary and key terms

These key terms and architectural components are used throughout the Sovereign Data and Agent Factory documentation.

8.5.3.1 Administration node

The central control node of the system. Hosts the Hybrid Manager interface, authentication services, and orchestration logic.

8.5.3.2 AI node

A GPU-enabled server used for model inference, vector processing, and agentic execution. Part of the optional AI workload configuration.

8.5.3.3 Agentic workflow

A multi-step automation pattern where AI agents perform tasks independently using internal tools, memory, and feedback loops. Often built on top of LLMs and vector data.

8.5.3.4 Control plane

The set of three Hybrid Manager nodes responsible for cluster scheduling, monitoring, lifecycle operations, and access control. Operates independently of data workloads.

8.5.3.5 Compute node

A general-purpose high-performance node used for running Postgres clusters, analytics, or vector pipelines. Can be scaled out as needed.

8.5.3.6 Embedding

The process of converting structured or unstructured data into vector representations suitable for AI workflows such as retrieval and classification.

8.5.3.7 EDB Hybrid Manager

EDB's management layer for orchestrating database and AI workloads in Kubernetes environments. Used to deploy clusters, monitor systems, and manage lifecycle operations.

8.5.3.8 Inference

The act of generating output from a machine-learning model. In this system, inference is performed on dedicated GPU nodes using KServe and NVIDIA NIM containers.

8.5.3.9 KServe

An open-source model serving framework used to run inference workloads. Integrated into Hybrid Manager and GPU nodes for serving LLMs and embedding models.

8.5.3.10 Lifecycle services

Ongoing support and management for both hardware and software, including security patches, OS updates, Postgres upgrades, and firmware management.

8.5.3.11 EDB Postgres Distributed (PGD)

EDB's multi-node, high-availability Postgres solution. Optional for customers who require globally distributed or synchronous multi-region clusters.

8.5.3.12 Retrieval-augmented generation (RAG)

A workload pattern where structured data (often in Postgres) is retrieved and passed to an LLM to generate natural language responses. Requires embedding and inference capabilities.

8.5.3.13 Sovereign AI

An AI deployment model where all data, models, and execution environments remain physically and logically within a customer-controlled space that's fully air-gapped if needed.

8.5.3.14 Vector store

A database or index that stores embedding vectors and supports similarity search. Postgres and pgvector are used in this system for local vector storage.

8.5.4 System specifications

The Sovereign Data and Agent Factory system is available in two configuration tiers: Core Compute and Advanced AI. Each system is preassembled and lifecycle-managed with full software and hardware support.

8.5.4.1 Configuration options

Configuration	Description	Total nodes
Core Compute	Designed for high-performance Postgres and hybrid DBaaS workloads	7
Advanced AI	Extends Core Compute with GPU nodes for AI and inference workloads	11

8.5.4.2 Configuration #1: Core Compute

Includes:

- 1x Administration node
- 3x Hybrid Manager / Control plane nodes
- 3x Hyper 2U compute nodes

8.5.4.2.1 Administration node

Category	Product / Model	Description	Qty
System	SYS-111C-NR	1U X13SEDW-F Server	1
Processor	P4X-EMR4514Y-SRN6J-MC	Intel Xeon Gold 4514Y 16-Core @ 2.0GHz	1
Memory	MEM-DR516L-SL01-ER56	16GB DDR5-5600 1RX8 LP ECC RDIMM (Total 128GB)	8
Storage	HDS-MUN-MTFDKCC3T2T	Micron 7450 MAX 3.2TB NVMe PCIe 4.0 U.3	2
Storage Controller	AOC-VROCSTNMOD	Intel VROC (RSTe) Standard HW Key	1

continues on next page

Table 2 – continued from previous page

Category	Product / Model	Description	Qty
Networking	AOC-AG-I2M-O	AIOM Dual-Port 1GbE RJ45	1
Networking	AOC-S25G-I2S-O	Std Low Profile 25G Dual Port SFP28	1

8.5.4.2.2 Core Compute – Hyper 2U Server (per server x6)

Category	Product / Model	Description	Qty
System	SYS-221H-TNR	Hyper 2U Server Chassis, 8x2.5" Bays, X13DEM Motherboard	1
Processor	R6538N-SRN6V-MCC	Intel Xeon Platinum 6538N 32-Core @ 2.1GHz	2
Cooling	SNK-P0087V	Passive CPU Heatsink for X13 Platform	2
Memory	MEM-DR532L-SL01-ER56	32GB DDR5-5600 2Rx8 LP ECC RDIMM (Total 1024GB)	32
Storage	HDS-SMN0-MZ1L2960HCJR	Samsung PM9A3 960GB NVMe PCIe 4.0 M.2	2
Storage	HDS-TUN-KCD8XPUG15T3	Kioxia CD8-P 15.3TB NVMe PCIe 5.0 2.5"	4
Storage	HDS-TUN-KCD8XPUG1T92	Kioxia CD8-P 1.92TB NVMe PCIe 5.0 2.5"	2
Storage Controller	AOC-VROCPREMOD	Intel VROC Premium Controller (RAID 0, 1, 5, 10)	1
Networking	AOC-A100G-M2CM-O	AIOM 2-port 100GbE QSFP28, Mellanox CX-6 DX	2
Power	PWS-2K63A-1R	1U 2600W Redundant Power Supply	2
Cooling	FAN-0239L4	80x80x38 mm, 16.8K RPM, Middle Cooling Fan	4
Expansion	RSC-H-68G5 / RSC-H-6G5L	1U LHS Hyper Riser cards (PCIe 5.0 x16/x8, PCIe 5.0 x16)	2
Security	AOM-TPM-9672V-O	TPM 2.0 Module	1

8.5.4.3 Configuration #2: Advanced AI

Includes all Core Compute components, plus:

- 2x 2-GPU Hyper 2U Servers
- 2x 8-GPU B200 10U Servers

8.5.4.3.1 2-GPU Hyper 2U Server (per server x2)

Category	Product / Model	Description	Qty
System	SYS-221H-TNR	Hyper 2U Server Chassis, 8x2.5" Bays, X13DEM Motherboard	1
Processor	R6538N-SRN6V-MCC	Intel Xeon Platinum 6538N 32-Core @ 2.1GHz	2
Cooling	SNK-P0087V	1U Passive CPU Heatsink for X13 Platform	2
Memory	MEM-DR532L-SL01-ER56	32GB DDR5-5600 2Rx8 LP ECC RDIMM (Total 1024GB)	32
GPU	GPU-NVL40S	NVIDIA Ada L40S 48GB GDDR6 PCIe Gen 4	2
Storage	HDS-SMN0-MZ1L2960HCJR	Samsung PM9A3 960GB NVMe PCIe 4.0 M.2	2
Storage	HDS-TUN-KCD8XPUG15T3	Kioxia CD8-P 15.3TB NVMe PCIe 5.0 2.5"	4
Storage	HDS-TUN-KCD8XPUG1T92	Kioxia CD8-P 1.92TB NVMe PCIe 5.0 2.5"	2
Storage Controller	AOC-VROCPREMOD	Intel VROC Premium Controller (RAID 0, 1, 5, 10)	1
Networking	AOC-A100G-M2CM-O	AIOM 2-port 100GbE QSFP28, Mellanox CX-6 DX	2
Power	PWS-2K63A-1R	1U 2600W Redundant Power Supply	2
Cooling	FAN-0239L4	80x80x38 mm, 16.8K RPM, Middle Cooling Fan	4
Cooling	MCP-310-8294...	GPU Plastic Air Shrouds	2
Expansion	RSC-H-68G5 / RSC-H-6G5L	1U LHS Hyper Riser cards (PCIe 5.0 x16/x8, PCIe 5.0 x16)	2
Security	AOM-TPM-9672V-O	TPM 2.0 Module	1

8.5.4.3.2 8-GPU B200 10U Server (per server x2)

Category	Product / Model	Description	Qty
System	SYS-A21GE-NBRT-G1	Supermicro 10U B200 Gold Series GPU Server	1
Processor	Intel® Xeon® Platinum 8570	56-Core @ 2.10GHz (350W)	1
Memory	DDR5 ECC RDIMM	96GB DDR5 5600MHz ECC RDIMM (Total 3072 GB)	32
GPU	NVIDIA HGX B200	Baseboard with 8x B200 180GB GPUs	1
DPU	NVIDIA BlueField-3	BF3220 DPU 200GbE Dual Port, Crypto Enabled	1

continues on next page

Table 5 – continued from previous page

Category	Product / Model	Description	Qty
Storage	PM9A3 M.2	1.9TB M.2 NVMe	1
Storage	7450 PRO 2.5"	3.8TB NVMe PCIe 4.0 SSD (1 DWPD)	1
*Networking	AOC-1	*400-Gigabit MCX75310AAS-NEAT OSFP Adapter	1
Onboard	—	2x RJ45 10Gbase-T Ports	1
Security	AOM-TPM-9670V-P	TPM 2.0 Module	1

8.5.5 Common workloads and use cases

Sovereign Data and Agent Factory is purpose-built for organizations that require strict control over data, infrastructure, and AI execution. It supports a wide range of workloads while remaining locally managed.

8.5.5.1 Why use the engineered system over cloud?

Many modern workloads can run in cloud environments—but not all organizations can or should rely on them. Sovereign Data and Agent Factory exists to meet the needs of the following.

8.5.5.1.1 Data sovereignty and compliance

- All data, models, and workloads remain inside your physical infrastructure.
- No external APIs or third-party storage are required.
- Enables deployment in financial, government, healthcare, and national security sectors.

8.5.5.1.2 Regulated environments

- Operates with limited internet dependency.
- Supports deployment lifecycles.
- Enables LLM and AI use cases in environments previously closed to them.

8.5.5.1.3 Unified platform for Postgres + AI

- Combines HA Postgres, vector search, and GPU inference in one system, controlled by your organization.
- Simplifies orchestration, monitoring, and support.
- Eliminates fragmented solutions and duplicate infrastructure.

8.5.5.1.4 Predictable performance and cost

- No egress fees, throttling, or unpredictable GPU queues.
- Optimized I/O and compute across known hardware.
- Total cost of ownership is clear and controlled.

8.5.5.1.5 Lifecycle-managed infrastructure

- Joint EDB + Supermicro support agreements.
- Hardware and software patches integrated and coordinated.
- No reliance on customer-led infrastructure management or DevOps.

8.5.5.2 Supported workload patterns

The following workloads are supported end to end on the Sovereign Data and Agent Factory system.

8.5.5.2.1 High-availability Postgres

- Run mission-critical systems of record with automatic failover.
- Observability and lifecycle are baked in.

8.5.5.2.2 Hybrid DBaaS

- Serve internal teams with multiple clusters from one interface.
- Role-based access control and cluster-level visibility.

8.5.5.2.3 Embedding and vector pipelines

- Transform data into embeddings using local models.
- Store and query with pgvector in Postgres.
- No API keys or cloud endpoints required.

8.5.5.2.4 Retrieval-augmented generation (RAG)

- Query structured internal data and pass it to an LLM.
- Host both retrieval and inference within your own boundary.

8.5.5.2.5 On-premises inference

- Serve generative and embedding models on dedicated GPU hardware.
- Use standard containers (for example, NVIDIA NIM) to abstract model serving.

8.5.5.2.6 Agentic AI workflows

- Build AI agents that execute multi-step tasks using internal tools and data.
- No external APIs or data leakage.
- Deploy interfaces in Slack, custom dashboards, or internal tools.

8.5.6 What to expect

Here's what to expect when receiving, setting up, and operating your Sovereign Data and Agent Factory system during the first 90 days—from delivery through active use and support.

8.5.6.1 Days 0–21: Site readiness and installation

After your order is confirmed, you'll complete a site readiness survey to guide installation. The survey includes:

- Power, rack, and cabling details
- Networking and security preferences
- Physical access and contact coordination

8.5.6.1.1 What happens

- Supermicro ships the full system to your selected data center.
- EDB and Supermicro teams perform on-site racking, power-up, and validation.
- Configuration is completed based on your preferences.
- You receive login credentials and URL access to the Hybrid Manager portal.

8.5.6.2 Days 21–28: Deploying your first workloads

Once the system is live, you can immediately begin launching workloads. All operations take place through the Hybrid Manager portal.

8.5.6.2.1 What you can do

- Deploy a high-availability 3-node Postgres cluster.
- Launch embedding jobs and vector pipelines using Postgres and pgvector.
- Use GPU nodes to serve models locally via NVIDIA NIM Microservices .
- Create RAG-style applications and internal chatbots.

You don't need to access Kubernetes directly. You manage everything through the HM portal.

8.5.6.3 Days 28–90: Lifecycle management and scaling

As workloads evolve, the system is ready to scale and update with full lifecycle support from EDB and Supermicro.

8.5.6.3.1 What's included:

- Regular OS and software updates, including Postgres and AI tools.
- Optional expansion by adding compute or GPU nodes.
- Hardware support with Supermicro's 4-hour Gold SLA.
- Built-in observability and diagnostics across all layers.

You can continue operating the system through Hybrid Manager while both software and hardware remain covered under joint support agreements.