

EDB Postgres AI Hybrid Manager

Release 1.3.5

EnterpriseDB Corporation

Feb 17, 2026

Contents

1 One platform, full control **3**
- 1.1 What you can do . . . 3
- 1.2 Start here . . . 4

2 Hybrid Manager Release Notes **5**
- 2.1 Hybrid Manager 1.3.5 release notes . . . 5
- 2.2 Hybrid Manager 1.3.4 release notes . . . 6
- 2.3 Hybrid Manager 1.3.3 release notes . . . 6
- 2.4 Hybrid Manager 1.3.2 release notes . . . 7
- 2.5 Hybrid Manager 1.3.1 release notes . . . 8
  - 2.5.1 HM console . . . 8
  - 2.5.2 Advanced High Availability and Distributed High Availability (PGD) . . . 8
  - 2.5.3 Backup, recovery & storage . . . 8
  - 2.5.4 Core platform functionality & API . . . 9
- 2.6 Hybrid Manager 1.3 release notes . . . 9
  - 2.6.1 Release Highlights . . . 9
  - 2.6.2 New Features and Enhancements . . . 9
  - 2.6.3 Fixes . . . 12
- 2.7 Hybrid Manager 1.2 release notes . . . 12
  - 2.7.1 Release Highlights . . . 13
  - 2.7.2 Deployment Options . . . 13
  - 2.7.3 New Features & Enhancements . . . 13
  - 2.7.4 Observability, Monitoring & Insights . . . 14
  - 2.7.5 Sustainability: ESG Reporting (Preview) . . . 14
  - 2.7.6 Platform Improvements . . . 15
  - 2.7.7 Resources . . . 15

3 EDB Postgres AI Hybrid Manager **17**
- 3.1 Why Hybrid Manager? . . . 17
- 3.2 Key capabilities . . . 17
- 3.3 Supported environments . . . 18
- 3.4 Integrated solutions . . . 18
- 3.5 Learn more . . . 18
- 3.6 Get started . . . 19
  - 3.6.1 Hybrid Manager architecture . . . 19
  - 3.6.2 Supported database types . . . 21
  - 3.6.3 Known issues . . . 27

3.6.4 Supported Postgres distributions . . . 36

4 Installing and configuring Hybrid Manager 39
4.1 Overview . . . 39
4.2 Phase 1: Planning your architecture . . . 39
4.3 Phase 2: Gathering your system requirements . . . 40
4.4 Phase 3: Deploying your Kubernetes cluster . . . 40
4.5 Phase 4: Preparing your environment . . . 40
4.6 Phase 5: Installing Hybrid Manager . . . 41
4.7 Phase 6: Exploring your post-installation options . . . 41
4.8 Configuring HM . . . 42
4.8.1 Planning your architecture . . . 42
4.8.2 Gathering your system requirements . . . 50
4.8.3 Deploying your Kubernetes cluster . . . 68
4.8.4 Preparing your environment . . . 73
4.8.5 Install Hybrid Manager . . . 84
4.8.6 Exploring your post-installation options . . . 87
4.8.7 Configuring Hybrid Manager . . . 88

5 Using Hybrid Manager 129
5.1 Where to start . . . 129
5.2 For users . . . 129
5.3 For Kubernetes administrators . . . 130
5.3.1 Hybrid Manager Terminology . . . 130
5.3.2 Hybrid Manager console . . . 134
5.3.3 Projects . . . 142
5.3.4 Working with clusters . . . 150
5.3.5 Upgrading database clusters in Hybrid Manager . . . 188
5.3.6 Migrating databases with Hybrid Manager . . . 195
5.3.7 Asset management . . . 306
5.3.8 Backups and restores . . . 311
5.3.9 Hybrid Manager CLI - edbctl . . . 318
5.3.10 Using the API . . . 385
5.3.11 For Kubernetes Admin . . . 389

6 AI Factory in Hybrid Manager 495
6.1 Example AI solutions you can build . . . 495
6.2 Learning paths . . . 496
6.3 Use cases and solutions . . . 496
6.4 AI Factory in Hybrid Manager workloads . . . 496
6.4.1 Gen AI workloads . . . 496
6.4.2 Model management and serving . . . 496
6.4.3 Vector Engine . . . 496
6.5 Hybrid Manager learn content . . . 496
6.6 How AI Factory fits within Hybrid Manager . . . 497
6.7 Get started . . . 497
6.7.1 Getting Started . . . 497
6.7.2 Manual . . . 515
6.7.3 How To Builders . . . 529
6.7.4 Setup . . . 530
6.7.5 How To Models . . . 539
6.7.6 Operate & Observe . . . 540

7 Analytics in Hybrid Manager 549
7.1 Why Analytics in Hybrid Manager . . . 549

7.2 Core Concepts and Capabilities . . . 550
7.2.1 Apache Iceberg® . . . 550
7.2.2 Delta Lake . . . 550
7.2.3 Tiered Tables with PGD . . . 550
7.2.4 Lakehouse Clusters . . . 550
7.3 How These Capabilities Work Together . . . 551
7.4 Getting Started: Quickstart Checklist . . . 551
7.5 Explore Further . . . 551
7.5.1 Analytics Concepts in Hybrid Manager . . . 551
7.5.2 Delta Lake in Hybrid Manager . . . 553
7.5.3 Apache Iceberg® in Hybrid Manager . . . 555
7.5.4 EDB Postgres Lakehouse Clusters on Hybrid Manager . . . 557
7.5.5 Tiered Tables in Hybrid Manager . . . 559

8 Sovereign Data and AI Factory 563
8.1 Why choose a hardware-based system? . . . 563
8.2 What's inside . . . 563
8.3 Supported workloads . . . 564
8.4 Hardware configurations . . . 564
8.5 Explore capabilities . . . 564
8.5.1 Concepts and architecture . . . 564
8.5.2 Frequently asked questions . . . 566
8.5.3 Glossary and key terms . . . 567
8.5.4 System specifications . . . 569
8.5.5 Common workloads and use cases . . . 571
8.5.6 What to expect . . . 573

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

One platform, full control

HM brings together the infrastructure, data, and application layers required to support modern Postgres-based data platforms:

- **Sovereign AI** — Build and serve AI-powered workloads using your own infrastructure, with full control over GPU resources and GenAI pipelines.
- **Postgres-native analytics** — Offload to open data formats like Iceberg and Delta Lake while preserving Postgres compatibility and control.
- **Operational excellence** — Simplify backup, scaling, monitoring, and cluster lifecycle management for Postgres clusters.

Whether deployed in the cloud or on-premises, Hybrid Manager ensures **observability, governance, and resilience** across your data and AI stack.

1.1 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.

continues on next page

Table 1 – continued from previous page

Area	Capabilities
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.

1.2 Start here

- *Install Hybrid Manager* → A detailed guide to walk you through a full HMinstallation.
- *Sovereign Data and AI Factory (Hardware Appliance)* → Sovereign Data and AI Factory hardware
- *Using Hybrid Manager* → backup/restore, monitoring, migrations, cluster creation, multi-DC deployment and more
- *AI Factory Architecture on Hybrid Manager* → sovereign GenAI pipelines, model serving, and retrieval systems
- *Analytics Concepts in Hybrid Manager* → Delta Lake, Iceberg, tiered storage, and lakehouse patterns
- *Hybrid Manager 1.2 release notes*

CHAPTER 2

Hybrid Manager Release Notes

The EDB Hybrid Manager documentation describes the latest version of EDB Hybrid Manager, including minor releases and patches. The release notes provide information on what was new in each release. For new functionality introduced in a minor or patch release, the content also indicates the release that introduced the feature.

Version	Release date
Hybrid Manager 1.3.5 release notes	12 Feb 2026
Hybrid Manager 1.3.4 release notes	15 Jan 2026
Hybrid Manager 1.3.3 release notes	15 Dec 2025
Hybrid Manager 1.3.2 release notes	27 Nov 2025
Hybrid Manager 1.3.1 release notes	28 Oct 2025
Hybrid Manager 1.3 release notes	10 Oct 2025
Hybrid Manager 1.2 release notes	17 Jun 2025

2.1 Hybrid Manager 1.3.5 release notes

Release Date: February 12, 2026

Hybrid Manager 1.3.5 is a maintenance release that includes bug fixes and stability updates.

Type	Description
Bug fix	Resolved various upstream stability issues in Spark Connect by upgrading to version 3.5.8. This update incorporates critical fixes and stability improvements into HM Analytics to improve platform performance.
Bug fix	Fixed an issue where replicas could fail to start after a system database major version upgrade. The system now correctly differentiates between WAL files from the old and new timelines in object storage, preventing startup crashes caused by timeline divergence.

continues on next page

Table 2 – continued from previous page

Type	Description
Bug fix	The internal statistics database (*stats-db*) no longer performs backups. This change optimizes storage and resource usage, as the telemetry data stored here is transient and not critical for Disaster Recovery.
Bug fix	The statistics collector has been patched to properly decode *query_text*, ensuring compatibility and reliable metric collection across different versions of the HM agent.

2.2 Hybrid Manager 1.3.4 release notes

Release Date: January 15, 2026

Hybrid Manager (HM) 1.3.4 is a maintenance release that includes bug fixes, updates to internal operands and third-party dependencies and updates to improve overall platform stability.

Type	Description
Enhancement	Improved automated WAL retention and cleanup for Postgres clusters. To ensure that Write-Ahead Logs (WAL) are correctly pruned from object storage, HM now utilizes barman-cloud as the default scheduled backup method for newly created Advanced High Availability and Distributed High Availability (PGD) clusters. For existing clusters and newly created Single Node, High Availability (PSR), or Analytics clusters, HM now automatically performs periodic Barman backups alongside volume snapshots.
Bug fix	Fixed an issue where creating a model cluster with an invalid name (such as one containing spaces) resulted in a generic 500 error. The HM console now validates the model cluster name during input to prevent spaces. Additionally, HM now correctly returns a 400 error with a descriptive message if an invalid name is provided.
Bug fix	Fixed an issue where image discovery failed when using a private Red Hat Quay registry in air-gapped environments. Previously, *upm-beacon-agent* couldn't successfully retrieve images even when configured with the correct registry URL and authentication tokens. This fix ensures that image discovery now works correctly for both secure and insecure private registries in restricted network environments.
Bug fix	Fixed an issue in the *cnp_pg_replication_in_recovery* alert where Persistent Volume Claims (PVCs) were incorrectly mapped to pods. Previously, the alerting logic attempted to reuse pod names for PVCs, which caused failures for WAL volumes that used a *-wal* suffix. The monitoring logic now correctly uses *cnpg_instance_id* to join metrics, ensuring accurate replication recovery status and alerting for all volume types.
Bug fix	Fixed a "missing module" error in the *upm-griptape* component. Previously, versions v1.5.16 and v1.7.6 of the *upm-griptape* service failed to start due to missing Python dependencies, specifically the *fastapi* module. This fix resolves the dependency conflicts and ensures all required modules are correctly bundled in the service image.

2.3 Hybrid Manager 1.3.3 release notes

Release Date: December 15, 2025

Hybrid Manager 1.3.3 is a maintenance release that includes bug fixes and stability updates.

Type	Description
Bug fix	You can now successfully click the Restore Cluster button even when Barman is configured as the default backup solution.
Bug fix	AWS STS requests now correctly route to the regional endpoint when a VPC endpoint is in use, rather than defaulting to the global endpoint.
Bug fix	Introduced a Quick-Signin URL to allow users to navigate directly to the Identity Provider (IdP).
Bug fix	Improved platform security against CVE-2025-14038 by ensuring all internal gRPC API endpoints have enhanced authentication and authorization to prevent unauthorized access.

2.4 Hybrid Manager 1.3.2 release notes

Release Date: November 27, 2025

Hybrid Manager 1.3.2 is a maintenance release that includes bug fixes and stability updates.

Type	Description
Bug fix	Fixed an issue where backup retention and compression settings couldn't be changed during installation. Users can now define the *internal_db_backup_compression* and *internal_db_backup_retention* (defaulting to 30 days) during installation, preventing issues with uncontrolled internal storage consumption and data lifecycle policies.
Bug fix	Corrected *beacon-agent* configuration to include missing release annotations. The *org.enterprisedb.operandset.version* is now properly sent to the *beacon-server* to fix release reporting inaccuracies.
Bug fix	Fixed an issue where the *ImageLibrary* pulled unnecessary images, resulting in degraded performance. Added filtering logic and corrected the Helm chart configuration to ensure these limits are properly applied during installation.
Bug fix	Added a fix to the *spire-agent* component which caused a connection error with the kubelet in RHOS deployments.
Bug fix	Added a fix to *kube-state-metrics* which prevented the service from exporting custom metrics.
Bug fix	Added a fix that caused *beacondb* backups to fail.
Bug fix	Added a fix for the *barman-plugin-v0-8-0-migrator* plugin which caused issues in RHOS deployments.
Bug fix	Fixed an issue that caused errors when configuring the Hybrid Manager to use a custom certificate issuer.
Bug fix	Corrected sorting behavior for AI Models, allowing users to sort by name in descending order.
Bug fix	Resolved filtering issues in the **Estate** table, ensuring accurate results when applying filters to PostgreSQL databases.
Bug fix	Resolved button visibility for the **Delete** and **Dismiss** buttons on Migration Portal, ensuring these actions are now correctly restricted based on user permissions.
Bug fix	Resolved an issue preventing PGD Knowledge Bases from displaying in the GenAI Builder interface.

2.5 Hybrid Manager 1.3.1 release notes

Release Date: October 28, 2025

This release includes bug fixes and security updates.

2.5.1 HM console

- The total number of Postgres clusters on the HM console home page now correctly excludes individual cluster nodes (e.g., EFM or CNP replicas) to prevent inflated and inaccurate totals.
- Fixed an issue where the database connection information in the HM console's **Connect** tab incorrectly displayed 'edb_admin' instead of the required 'bdrdb' for Advanced High Availability and Distributed High Availability (PGD) clusters.
- The database size in the **Monitoring** tab of a cluster's detailed page now correctly displays storage metrics for self-managed clusters.
- Tags are now correctly displayed for active clusters on the **Deployment Details** page of the HM console.

2.5.2 Advanced High Availability and Distributed High Availability (PGD)

- Corrected validation logic for `max_worker_processes` to ensure that GUC values for replica groups aren't incorrectly validated as lower than the primary group.
- Fixed an issue where submitting an empty GUCs configuration payload didn't preserve the existing settings, instead incorrectly triggering a reset to default values.
- Ensured that when duplicating a data group, the GUCs (such as `max_connections`) are properly included in the payload, preventing unnecessary validation failures.
- Resolved a bug that prevented the successful addition of a new PGD group.
- Fixed an issue that could cause PGD-X cluster creation to stall during the reconcile phase.
- When adding a new PGD data group, the database settings are now correctly duplicated from the first data group to maintain consistency.
- PGD cluster status visibility has been enhanced with new control plane resource (CR) status panels for cluster, clusterwrapper, and pgdgroup resources.
- The console now correctly reflects the effective `max_wal_size` of 8 GB in the HM console, resolving a discrepancy where the console displayed 1024 MB.
- The required `shared_preload_libraries` setting for PGD now correctly includes pgaa, or the HM console provides appropriate indication when the setting is incorrect.

2.5.3 Backup, recovery & storage

- Fixed a bug where cluster metadata wasn't completely removed after the retention period, leaving orphaned Barman files in S3/object storage.
- WAL files marked as completed are no longer incorrectly retained beyond the configured limits.
- Resolved an issue with the upm-beacon agent component to stabilize the backup deletion process.
- The cross-location cluster restore functionality now correctly uses pg_basebackup as the method.

2.5.4 Core platform functionality & API

- Fixed a bug that caused the `accm-server` to crash due to an invalid pod name format.
- Resolved an issue that could prevent cluster creation due to upper bounds limits on EKS.
- Fixed a bug that prevented the secondary location beacon dispatcher runner from connecting to the primary beacon-server.
- The **Model Profile Paths** field in the HM console is now correctly marked as optional during configuration.
- Fixed an API bug that caused certain status fields, such as `status.backupStatus` , to appear empty.
- Improved cluster metadata availability across the platform for enhanced reliability.
- Fixed an issue that caused an error when calling an LLM remotely, ensuring successful external API communication.
- Increased the default memory allocation for the beacon agent and server and made the limits configurable in the bootstrap values.
- Fixed a bug where a process group would cease retrying to connect after an hour, ensuring continuous attempts for connection and stability.

2.6 Hybrid Manager 1.3 release notes

Release Date: October 10, 2025

We are thrilled to announce the release of EDB Postgres AI Hybrid Manager v1.3.

2.6.1 Release Highlights

- **Upgrade reliability**: We've successfully hardened platform upgrades across all supported minor and patch releases. This includes a fully validated, self-service pathway for existing EDB Cloud Service (former Big Animal) customers to transition to Hybrid Manager.
- **Secured crucial backup and recovery capabilities**: Introduced multi-location disaster recovery, simplifying complex switchover and HA management.
- **Unified observability**: Achieved feature parity for monitoring self-managed bare-metal and CloudNativePG clusters (including Community CNPG), providing a true single pane of glass for all EDB instances.
- **Platform stability & flexibility**: Validated EDB Cloud Service (formerly Big Animal) to Hybrid Manager migrations, expanded deployment support to SUSE Rancher, and secured crucial backup & recovery flows.
- **Data-Sovereign AI for migration**: Migration AI Copilot can now use models hosted within Hybrid Manager, ensuring migration queries remain secure and within platform boundaries.
- **Secure AI model access**: Enabled secure, authenticated external access to deployed KServe models via a new dynamic proxy layer and centralized AI Model Catalogue management.

2.6.2 New Features and Enhancements

Platform management & reliability

SUSE Rancher platform support: Hybrid Manager is now certified and supported for deployment on SUSE Rancher environments.

Support for EDB Postgres Distributed (PGD) 6.1: Hybrid Manager now supports provisioning and managing clusters running EDB Postgres Distributed (PGD) version 6.1. This upgrade enables customers to utilize the latest features, performance optimizations, and security enhancements provided by the newest PGD release within the unified Hybrid Manager platform.

Decoupled operand lifecycle: We have de-coupled the Postgres operand releases from the main Hybrid Manager release cycle. This foundational improvement enhances platform agility by enabling faster, more frequent, and less disruptive updates to the database engine components.

Local storage support for Engineered System deployments: We have introduced support for provisioning high-performance, low-latency persistent volumes directly from local disks on worker nodes within the Engineered System. This uses the LVM Storage Operator (TopoLVM), enabling robust data persistence for on-premise deployments that do not utilize shared external storage.

Administrator broadcast messaging: Administrators (including project owners) can now broadcast important platform-wide or project-specific messages to all users within Hybrid Manager. This feature is ideal for communicating planned maintenance, outages, or other critical operational updates. These forced notifications are displayed in the user interface via the inbox and information pane to ensure all users are informed.

Disaster recovery and high availability

Multi-location high availability and disaster recovery (DR): Hybrid Manager now supports building highly resilient Postgres architectures

across distinct Kubernetes clusters (e.g., across regions) by leveraging CloudNativePG's distributed topology features. This allows users to configure and manage multi-location clusters using native Postgres streaming replication and WAL archiving for robust DR and high availability.

Backup frequency: We've introduced the ability to customize the backup frequency for the Hybrid Manager control plane. Customers can now define the Velero backup schedule directly through the Helm values file, allowing them to precisely control their Recovery Point Objective (RPO) and meet specific operational requirements.

Volume snapshot restore validation: Hybrid Manager now includes enhanced validation to prevent restore operations from stalling and failing due to incompatible storage configurations. The system will now block or issue a clear warning when attempting to restore a cluster from a volume snapshot if the new cluster's storage configuration (e.g., changing from a single data volume to split data/WAL volumes) is not compatible with the source snapshot.

Cross-namespace volume snapshot restore support on RHOS: We have successfully engineered a solution to overcome Red Hat OpenShift's default security context constraints (SCC) that previously prevented restoring a volume snapshot into a different namespace. Hybrid Manager now automatically configures the required security settings to allow cross-namespace restore operations, significantly enhancing disaster recovery and multi-tenancy capabilities for clusters deployed on OpenShift environments. This ensures volume snapshot backups can be reliably restored for data migration or recovery to any target namespace.

Backup support for RHOS: Hybrid Manager now supports volume snapshot-based backup and recovery operations for clusters deployed on Red Hat OpenShift (RHOS) and Engineered System. This provides a faster, platform-native B&R experience for these critical environments.

Upgrade experience

Seamless Hybrid Manager upgrades: We've implemented foundational work to make upgrades between all supported minor and patch releases highly dependable. This enhances platform stability, minimizes the risk of regressions, and provides users with clear expectations and a reliable path for platform transitions.

Transition from EDB Cloud Service: We now support a non-disruptive, self-service path for existing Cloud Service (formerly Big Animal) Bring Your Own Account (BYOA) customers to easily and reliably transition to Hybrid Manager.

Observability, monitoring, and insights

Unified observability: Hybrid Manager supports the collection of essential telemetry and health metrics from self-managed, bare-metal Postgres instances, as well as self-managed CloudNativePG (CNP) and EDB Community CloudNativePG instances, bringing them to parity with HM-managed clusters. The new metrics include crucial operational data such as host metrics, WAL activity, storage utilization, and IOPS statistics. This allows for unified,

comprehensive observability within the Hybrid Manager console, regardless of where your Postgres instance is deployed.

Postgres configuration recommendations: Hybrid Manager provides recommendations to help you tune Postgres settings, improve database performance, and optimize resource usage. You can access these suggestions in the **Recommendations** tab of the cluster's detailed view.

GenAI workloads with AI Factory

Secure external access to deployed AI models: Hybrid Manager now supports secure, authenticated access to deployed KServe models from outside the EDB Postgres AI Platform. This is achieved by implementing a dynamic proxy and leveraging Istio for robust authentication and authorization (AuthN/AuthZ). This enhancement allows developers to securely integrate deployed AI models (including OpenAI-compatible endpoints) into external applications while maintaining full platform governance.

Centralized AI model catalogue: Hybrid Manager now includes a dedicated model catalogue to serve as the central repository for your organization's AI models. This catalogue provides authorized users with the necessary tools for AI governance and rapid deployment. Users can now list, create, import, edit, and view their entire inventory of AI models. The catalogue integrates directly with KServe, allowing users to deploy an AI model to the model serving infrastructure with a single click from the Hybrid Manager interface.

Dedicated management for AI Model serving clusters (KServe): Hybrid Manager now provides centralized management for the underlying AI model serving infrastructure. Users and administrators can now gain unified visibility and control over deployed KServe clusters and models:

- Model Cluster Overview: A new user interface displays a list of all running model server clusters, along with dedicated detail pages for monitoring and administration.
- REST API Endpoints: Dedicated REST APIs are available for listing running model clusters and retrieving detailed cluster information, facilitating external integration and automation.

Migrations

Data-Sovereign AI for Migration Portal AI Copilot: AI Copilot can now be configured to exclusively utilize AI Factory Models hosted within Hybrid Manager. This ensures that sensitive migration queries and data remain within the boundary of the platform and are not transmitted to third-party vendors.

Postgres migration assessment details in console: Hybrid Manager's migration assessment capabilities have been expanded to include Postgres as a supported source database. The existing migration assessment details page has been updated to provide Postgres-specific information, offering users a unified and detailed view of your Postgres migration readiness alongside their Oracle assessments.

Agent schema selection and filtering: Users can now precisely control which schemas the migration agent ingests from a source database using the new databases.schema.filter option in the `beacon-agent.yaml` configuration. This allows platform users to define a subset of schemas to either include or exclude,

Updated Data Migration Service Agent: The underlying components for data migration, previously known as the EDB DMS Reader and Writer, have been unified and updated to the EDB DMS Agent . This change, effective in Hybrid Manager 1.3, provides a streamlined and unified agent for all data ingestion and migration workflows.

Enhanced database migration documentation: We've significantly overhauled all database migration content for improved organization and clarity. All content is now unified under the Migrating databases section, grouped into clear PLANNING, MIGRATING, and REFERENCE categories.

2.6.3 Fixes

- **Resolved UI 500 error on database migration page**: Fixed an issue where the Migration Databases view would incorrectly return a HTTP 500 error when the number of database items exceeded the default pagination size.
- **Resolved cross-version restore issue:** Fixed an issue that previously blocked users from selecting the correct Postgres image version during a restore operation following an attempted major version upgrade. Users can now successfully select the original Postgres version image (e.g., v16) and restore a pre-upgrade backup on a new cluster, providing a reliable rollback path if the major upgrade fails or needs to be abandoned.
- **Adjusted default log level for beacon-agent sidecar**: The default logging level for the `beacon-agent` sidecar in Postgres pods was unnecessarily set to `DEBUG` , resulting in excessively noisy logs, especially in GKE deployments. The default log level has been changed to `INFO` to reduce log clutter and focus on relevant operational information.
- **Corrected duplicate metrics after CNP failover**: Fixed an issue where connecting external, multi-node CloudNativePG clusters to Hybrid Manager would result in duplicate metric series being displayed on the pg_node level charts following a primary node failover or promotion. Metric collection for these external clusters is now accurately reported after topology changes.
- **Eliminated erroneous recommendations for PGD/BDR internal tables**: Fixed an issue where the Query Recommendations feature would incorrectly surface suggestions for internal EDB Postgres Distributed (PGD) or BDR management tables. The underlying mechanism (`query_advisor` extension) has been updated to allow excluding internal system schemas, ensuring that recommendations focus only on end-user application tables and prevent false alarms about internal infrastructure health.
- **Initial backup for cluster health**: The cluster provisioning workflow has been improved to ensure that a newly deployed cluster is not marked as "Deployed" or "Healthy" in Hybrid Manager until the initial mandatory backup has successfully completed. This change prevents situations you might rely on a seemingly healthy cluster that lacks an immediate restore point.
- **Increased minimum resource requirements for EDB Postgres Distributed (PGD)**: To improve the stability and reliability of PGD deployments, we have increased the minimum required storage and memory allocations. The minimal disk size has been raised to 10 GB (from 1 GB), and the default minimum memory has been set to 2 GB. This change prevents critical operational issues, such as RAFT consensus failures in low-memory environments, and applies consistently across all PGD components, including witness nodes.
- **API validation for location-specific networking parameters**: Implemented validation in the provisioning API to reject cluster creation requests if networking parameters, such as `allowedIpRanges` , are specified for a deployment location that doesn't support IP `allowlisting` . This ensures that users receive immediate feedback on invalid configurations and promotes consistent use of location capabilities.
- **Restored tags visibility on the Estate page**: Fixed an issue where tags assigned to Postgres clusters were not displayed in the **Tags** column on the main **Estate** page. Tags are now correctly retrieved from the API and shown consistently across all relevant user interface views.
- **Resolved installation failure on Kubernetes 1.33**: Fixed an issue where the `transporter` component failed to install on Kubernetes version 1.33 due to an incompatibility in the underlying client libraries, ensuring successful installations on modern K8s environments.

2.7 Hybrid Manager 1.2 release notes

Release Date: June 17, 2025

We are thrilled to announce the first public availability release of **EDB Postgres AI Hybrid Manager v1.2**.

EDB Postgres AI Hybrid Manager is the control plane of the EDB Postgres AI Platform—a sovereign AI, analytics, and Postgres management solution. This release revolutionizes how you work with data by unifying advanced analytics, generative AI, and operational management into a single, cohesive platform.

2.7.1 Release Highlights

- **Sovereign AI & Analytics Platform**: Hybrid Manager is now a complete platform for building modern data and AI applications. Deploy a Postgres-native lakehouse and sovereign GenAI workloads with unified governance, all within your own infrastructure.
- **New Deployment Option: Sovereign Data and AI Factory**: Introducing a fully integrated hardware and software appliance, co-designed by EDB and Supermicro®, for deploying production-grade Postgres and AI workloads in your own data center with no external dependencies.
- **Postgres-Native Lakehouse**: A groundbreaking analytics stack is now built-in. Deploy **Lakehouse Clusters** to run high-speed, vectorized queries directly against open data formats like **Apache Iceberg®** and the **Delta Lake format** in your object storage.
- **AI Factory for GenAI Workloads**: Build and serve powerful AI applications with a low-code **GenAI Builder**, integrated model serving with GPU acceleration, in-database **Semantic Search**, and a comprehensive Python SDK.
- **Unified Management & Proactive Diagnostics**: Experience a completely redesigned monitoring interface, a new cluster-level scorecard, **proactive query recommendations**, and deep query latency statistics.

2.7.2 Deployment Options

EDB Postgres AI Hybrid Manager v1.2 is available in two ways, providing maximum flexibility for your operational needs.

Sovereign Data and AI Factory (Hardware Appliance)

For organizations requiring maximum control and zero external dependencies, the **Sovereign Data and AI Factory** is a fully integrated, pre-assembled hardware and software platform. Delivered as a rack-mounted system to your data center, it is the ideal choice for environments where data sovereignty, security, and lifecycle reliability are paramount.

This engineered system is optimized for scale and is ready to run advanced Postgres, vector, and AI pipelines the moment it's installed. It includes validated Supermicro servers, lifecycle-managed Kubernetes, and the full Hybrid Manager software suite with optional GPU nodes for AI inference.

Software-Only Deployment

For organizations with existing infrastructure, EDB Postgres AI Hybrid Manager can be deployed as a software-only solution on your preferred Kubernetes distribution. It is certified for **Amazon EKS**, **Google GKE**, and **Red Hat OpenShift (RHOS)**.

2.7.3 New Features & Enhancements

Sovereign AI & Analytics Platform

Hybrid Manager v1.2 introduces a powerful, integrated suite of tools for building and managing next-generation analytics and AI workloads directly on your Postgres platform.

Postgres-Native Analytics

Build a modern, cost-effective data architecture that combines the power of Postgres with the scale of object storage. Hybrid Manager provides the tools to manage your entire analytics stack.

- **Lakehouse Clusters**: Provision and manage dedicated compute clusters that provide scalable, vectorized query execution on object storage. They serve as the query engine for your Postgres-native lakehouse.

- **Apache Iceberg® Support**: Natively manage large analytical tables in Iceberg format. Use **Tiered Tables** with EDB Postgres Distributed (PGD) to automatically offload cold data partitions to Iceberg, making them queryable via Lakehouse Clusters while reducing storage costs.
- **Delta Lake Format Integration**: Seamlessly integrate with existing data lakes. Lakehouse Clusters can directly query tables using the Delta Lake open format, enabling fast analytics on data produced by other big data pipelines like Spark.
- **Tiered Tables with PGD**: Efficiently manage large time-series datasets by automatically partitioning tables and offloading cold partitions to object storage in Iceberg format. A single query can span both hot transactional data in Postgres and cold historical data in your data lake.

GenAI Workloads with AI Factory

EDB Postgres AI Hybrid Manager v1.2 introduces the **AI Factory**, a comprehensive GenAI capability stack designed for enterprise-grade deployments.

- **GenAI Builder**: A web-based low-code interface to rapidly prototype, iterate, and deploy AI applications—from simple one-shot prompts to complex RAG pipelines.
- **Model Serving**: Deploy and manage **NVIDIA NIM** containers for AI inferencing on GPU-accelerated infrastructure, with built-in support for availability and autoscaling.
- **Semantic Search**: Use vector search capabilities built into Postgres to power retrieval for your GenAI applications. Ingest and embed text, HTML, PDFs, or images—then query them via similarity search.
- **Python SDK for AI Development**: The EDB AI Builder Framework SDK enables developers to assemble AI workflows using Python. Integrate retrieval, reranking, embeddings, and generative responses with fewer lines of code.

2.7.4 Observability, Monitoring & Insights

v1.2 enhances the platform's observability with expanded metrics, sustainability insights, and performance diagnostics.

- **Redesigned Monitoring Dashboard**: Gain deeper insight into cluster health with a revamped UI, improved layout, and more granular control over metrics and thresholds.
- **Query Latency Metrics**: Visualize query latency statistics collected via the `pg_stat_monitor` extension directly in the **Query Diagnostics** tab.
- **Proactive Query Recommendations**: Receive optimization tips, including new index suggestions and stale statistics alerts, based on real workload patterns.
- **Cluster Scorecard**: View a unified summary of diagnostics, recommendations, and risk indicators for each cluster in one place.
- **Self-Managed Postgres Support**: Connect external Postgres instances to Hybrid Manager to collect query diagnostics, recommendations, and monitoring metrics.

2.7.5 Sustainability: ESG Reporting (Preview)

EDB Hybrid Manager now includes **experimental support for ESG dashboards**, helping organizations track energy usage and carbon emissions for their data infrastructure.

- **Carbon and Energy Metrics**: Visualize CPU-based energy usage and estimated CO□ emissions using configurable emission factors.
- **Region-Aware Reporting**: Customize dashboards with local energy cost ($/kWh) and emission rate (kg CO□/kWh) based on the deployment region.

- **Integrated Grafana Dashboards**: Access the new **Power Usage & Carbon Footprint** dashboards directly from the console.
- **Preview Phase Goals**: Validate demand for ESG observability and collect feedback from interested users.

2.7.6 Platform Improvements

Area	Highlights
Deployment options	New Sovereign Data and AI Factory hardware appliance.
Analytics	Lakehouse clusters with Iceberg and Delta Lake query support.
AI application tooling	GenAI Builder, model serving, semantic search, and SDK.
Monitoring	Refreshed console, scorecard view, extended query diagnostics.
Postgres coverage	Supports external/self-managed Postgres instances.
Sustainability	Preview of ESG metrics and dashboards for power and carbon tracking.

2.7.7 Resources

- *Phase 5: Installing Hybrid Manager*
- *Using Hybrid Manager*
-

AI Factory Overview

- Analytics Lakehouse Architecture
-

Power & ESG Reporting

- Release Notes Archive

Released: 17 June 2025

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 sovereign GenAI pipelines , 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 enterprise cloud platforms:

- Amazon Elastic Kubernetes Service (EKS)
- Google Kubernetes Engine (GKE)

HM also supports on-premises hosting 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:

- *AI Factory Architecture on Hybrid Manager* – Build and run GenAI workloads using private data.
- *Analytics Concepts in Hybrid Manager* – Enable open-format analytics with Iceberg, Delta Lake, and tiered offloading.
- *Model Image Migration* – Accelerate adoption of Postgres with in-platform migration tools.
- *Operational Monitoring* – Monitor operations using built-in observability stack with Kubernetes and database insights.
- Cluster lifecycle – Create templates, perform upgrades, configure HA settings, and restore data.

3.5 Learn more

- *AI Factory Architecture on Hybrid Manager*
- *Supported Postgres distributions*
- *Supported database types*

3.6 Get started

- *Install Hybrid Manager*
- Use Hybrid Manager

3.6.1 Hybrid Manager architecture

The outermost containers of the diagram illustrate a Kubernetes cluster running on a cloud service provider's infrastructure, such as 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.

Working inward through the diagram, the three main logical groupings 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, which isn't as simply as pictured in the diagram.

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, which may not be as simply as pictured.

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 .

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>` .

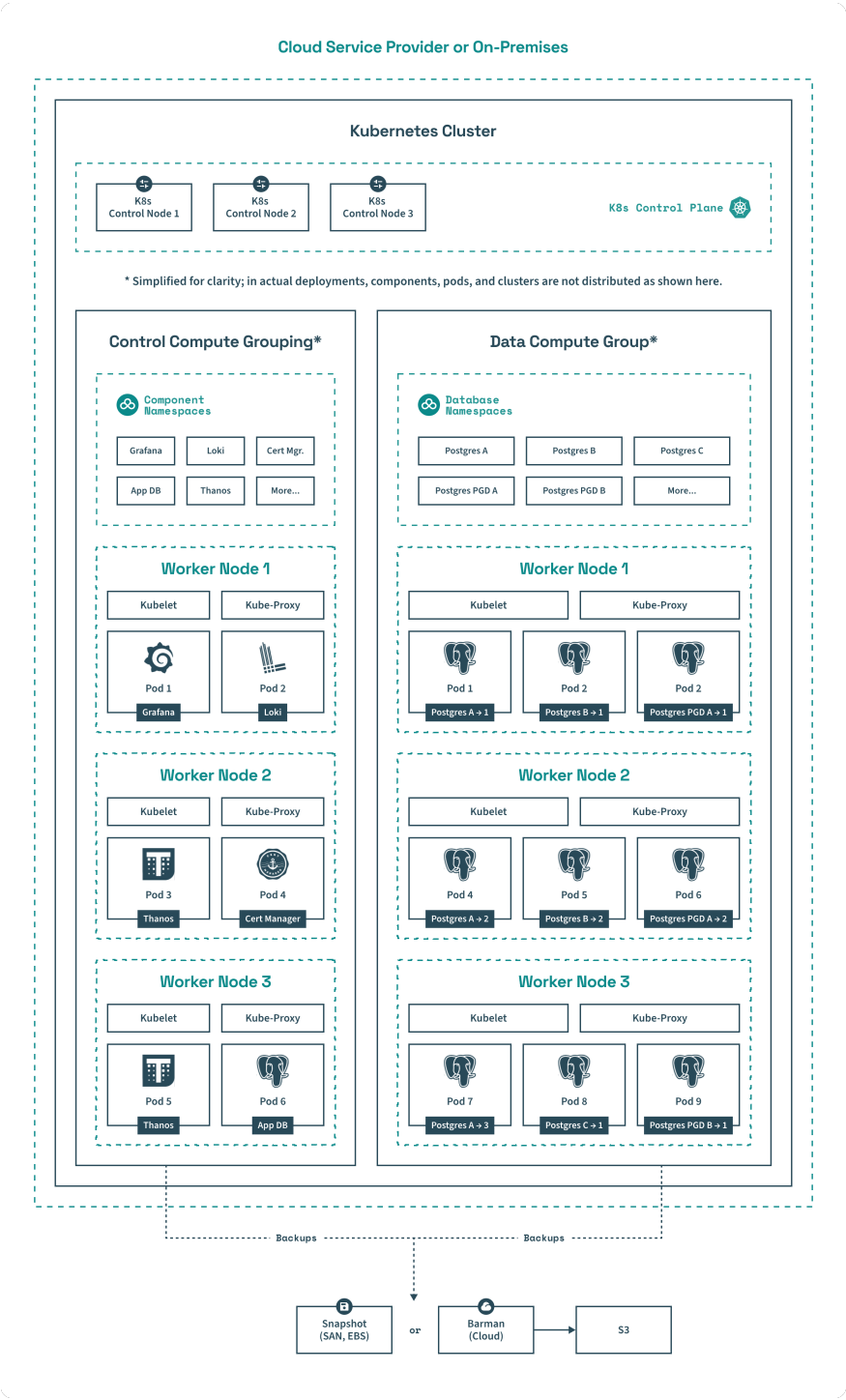


Fig. 1: High-level diagram of Hybrid Manager's architecture

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.

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.

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.

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.

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.

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.

Cluster type architectures

- *Distributed high availability*
- *Advanced high availability*
- *High Availability Considerations*
- *Single node*

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

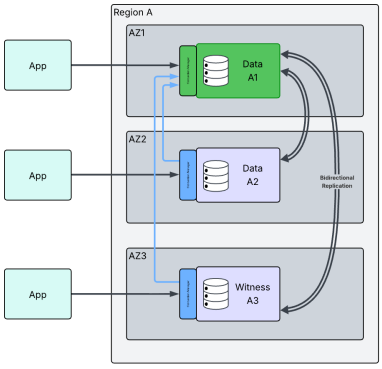


Fig. 2: region(2 data + 1 witness)

- 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

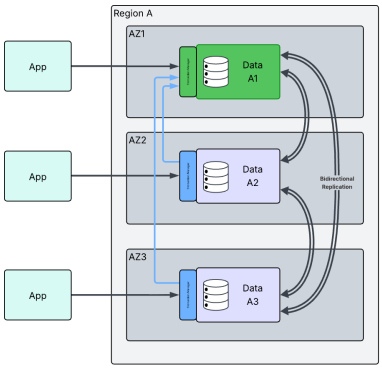


Fig. 3: region(3 data)

- 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.

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.

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 .

Advanced high availability

Advanced high-availability (AHA) clusters are powered by EDB Postgres Distributed and are based on the Essential edition of PGD. Unlike *Distributed high availability* , 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

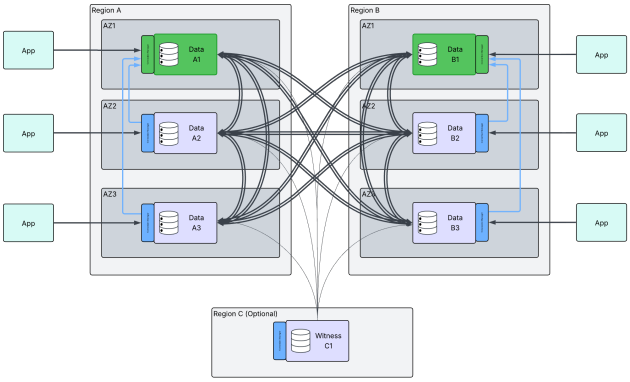


Fig. 4: groups(2 x 3 data + 1 witness)

ensure connections go to the current write leader node. This eliminates the need to manage conflicts across the cluster, making application development easier.

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.

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, then 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.

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.

Incoming client connections are always routed to the current primary. In case of failure of the primary, a standby replica

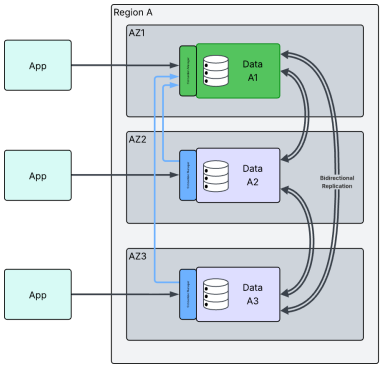


Fig. 5: Advanced high-availability cluster architecture

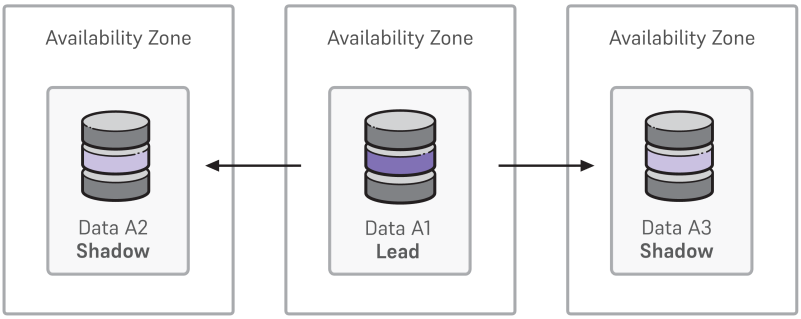


Fig. 6: Primary/standby high availability

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.

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. 7: Single node cluster

3.6.3 Known issues

These are the currently known issues and limitations identified in the Hybrid Manager 1.3 release. Where applicable, we have included workarounds to help you mitigate the impact of these issues. These issues are actively tracked and are planned for resolution in a future release.

> **Note**
>
> Some of these issues have been resolved in a later Innovation release. Please refer to *Known issues* if you want to know the latest status of these issues or if you are using an Innovation release.

Multi-DC

Multi-DC configuration loss after upgrade

> **Tip**
>
> Resolved in an *Known issues* .

Description: Configurations applied by the multi-DC setup scripts (for cross-cluster communication) do not persist after a Hybrid Manager platform upgrade or operator reconciliation.

Workaround: After every Hybrid Manager upgrade or component reversion, the multi-DC setup scripts must be run again to re-apply the necessary configurations.

Location dropdown list is empty for multi-DC setups

> **Tip**
>
> Resolved in an *Known issues* .

Description: In multi-data center environments, the API call to retrieve available locations fails with a gRPC message size error (429/4.3MB limit exceeded). This is due to the large amount of image set information included in the API response, resulting in an empty location list in the console.

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 beacon 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
```

1. Edit the ConfigMap found in the previous step 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",
```

(continues on next page)

(continued from previous page)

```
      "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

This step must be performed on the primary Hybrid Manager cluster if you are running a multi-DC setup.

1. 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 Agent configuration:

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

1. 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
    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 Agent pod:

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

After completing these steps, the reduced image data size should allow the location API call to succeed, and the locations should appear correctly in the Hybrid Manager console.

Core platform and resources

Cross-stream upgrade issues for OpenShift users

Description: Due to the OpenShift Operator Lifecycle Manager (OLM) enforcing strict semantic versioning, date-based versions (like `2026.1`) are interpreted as being significantly "higher" than standard versions (like `1.4`). This can prevent the OLM from recognizing a move to an LTS version as an upgrade.

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

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

> **Tip**
>
> Resolved in HM versions 1.3.1 and later.

Description: In environments with many databases and backups, the default 1GB memory allocation for the `upm-beacon-agent` pod is insufficient, which can lead to frequent OOMKill or crashloop issues. This resource limit is currently not configurable via the standard Helm values or HybridControlPlane CR.

Workaround: The user must manually patch the Kubernetes deployment to increase the memory resource limits for the `upm-beacon-agent` pod.

Database cluster engine

Incorrect database name displayed for EDB Postgres Distributed (PGD) clusters

> **Tip**
>
> Resolved in HM versions 1.3.1 and later.

Description: The Hybrid Manager console's **Connect** tab and connection string incorrectly show the default database name for PGD clusters as edb_admin. PGD clusters require connection to the bdrdb database.

Workaround: For PGD cluster connection information, use one of the following reliable sources: the `.PGPASS BLOB` , the `.PG_SERVICE.CONF` file, or the full connection string from the cluster details page.

PGD-X cluster creation stuck in the "PGD - Reconcile application user" phase

> **Tip**
>
> Resolved in HM versions 1.3.1 and later.

Description: PGD-X cluster creation, particularly when involving a witness-only region, may stall due to: - global RAFT leadership being unexpectedly held by the witness-only node, or due to - a subgroup's `enable_routing` being disabled.

Workaround for RAFT Leadership issue: Manually trigger the transfer of the global RAFT lead to a node within a data group. Connect to the PGD cluster's bdrdb and execute:

```
bdr.raft_leadership_transfer(node_name:=<target node>, wait_for_completion:=true,␣
↪node_group_name:=world);
```

Workaround for enable_routing issue: Manually enable routing for the subgroup. Connect to bdrdb and execute:

```
SELECT bdr.alter_node_group_option(<subgroup name>,enable_routing,true);
```

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

Description: Creating a 3-data node EDB Postgres Distributed (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, and then update the value after the cluster has been successfully provisioned.

PGD database settings are not duplicated when creating or duplicating a second data group

> **Tip**
>
> Resolved in HM versions 1.3.1 and later.

Description: When creating or duplicating a second data group in a PGD cluster, Postgres settings (like `max_connections` , `max_worker_processes` , etc.) are not automatically copied from the first data group. This can lead to inconsistent settings and cluster health issues, as the replica group settings cannot be lower than the primary group.

Workaround: Manually edit the configuration for the second PGD group to ensure the database settings are identical to the first data group before provisioning.

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.

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 do not support IP Subject Alternative Names (IP SANs), which are required for `verify-full` in certain environments (like GKE Load Balancers).

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

Second node is too slow to join large HA clusters

> **Tip**
>
> Resolved in an *Known issues* .

Description: For large clusters, the `pg_basebackup` process used by a second node (standby) to join an HA cluster is too slow. This can cause the standby node to fail to join, which prevents scaling a single node to HA and also causes issues when restoring a cluster directly into an HA configuration.

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 is no workaround for restoring a large cluster into an HA configuration.

Unable to query Knowledge Bases for self-managed clusters

Description: You are unable to query Knowledge Bases for self-managed clusters.

Workaround: n/a

Backup and recovery

Replica cluster creation fails when using volume snapshot recovery across regions

> **Tip**
>
> Resolved in HM versions 1.3.1 and later.

Description: Creating a replica cluster in a second location that is in a different region fails with an `InvalidSnapshot.NotFound` error because volume snapshot recovery does not support cross-region restoration.

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

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 is not yet configurable via 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 (e.g., to 20) to accelerate archiving.

Volume snapshot clusters don't automatically prune WAL files

> **Tip**
>
> Resolved in HM versions 1.3.4 and later.

Description: Due to a current limitation, the automatic Write-Ahead Log (WAL) retention policies don't work for VolumeSnapshot and are only compatible with the barman-cloud backup method. Because VolumeSnapshot is the default backup method for all supported environments, clusters created using VolumeSnapshot will not automatically prune old WAL files, resulting in an accumulation of WAL files in your object storage.

Workaround: To prevent storage accumulation and invoke WAL retention policies, you must perform or schedule an on-demand Barman backup for clusters originally created with VolumeSnapshot at least once per week.

Volume snapshot restoration is limited to the same region

Description: While Hybrid Manager (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.

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

Description: The DR process for the internal `transporter-db` service may fail when restoring from the latest available backup. This occurs in low-activity scenarios where a backup was completed, but no subsequent write-ahead log (WAL) file was archived immediately following that backup. This gap prevents the restore process from successfully completing a reliable point-in-time recovery.

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.

AI Factory and model management

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

Description: Model creation for the `nim-nvidia-nvclip` model will fail within the AI Factory when the profile cache is utilized during the deployment process.

Workaround: The workaround requires an administrator to manually download the necessary model profile from the NVIDIA registry to a local machine. They must then upload the profile files directly to the Hybrid Manager's object storage path, and finally, 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
```

1. Pull the Docker image to your local machine:

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

1. Prepare a local directory for the downloaded profiles:

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

1. Select the profile for your target GPU.

For example, A100 GPU profile: `9367a7048d21c405768203724f863e116d9aeb71d4847fca004930b9b9584bb6`

1. 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 will not 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.

1. Upload the profiles from your local machine to the object storage bucket used by your Hybrid Manager 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.

1. Create the model `nim-nvidia-nvclip` using the HM console, specifying the Model Profiles Path field as the previous location (e.g., `/model-cache/nim-nvidia-nvclip`). The deployment will initially fail or become stuck.
2. Export the InferenceService YAML from the Hybrid Manager Kubernetes cluster.
3. 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 ...
      runtime: nim-nvidia-nvclip
      storageUri: gs://uat-gke-edb-object-storage/model-cache/nim-nvidia-nvclip
# ------------------------------------------------
```

1. 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 should now proceed to start successfully, using the model profiles you manually provided via object storage.

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

> **Tip**
>
> Resolved in an *Known issues* .

Description: Creating an AI Model cluster and specifying an object storage path in the “Model Profiles Path field” will cause the deployment to stall at the pending stage if the specified path contains no content (i.e., the model profile does not yet exist).

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.

Error page appears when editing knowledge base credentials

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.

Incorrect model name causes 404 error when calling LLM remotely

> **Tip**
>
> Resolved in HM versions 1.3.1 and later.

Description: When calling a deployed NVIDIA NIM model via the API endpoint, the model name displayed on the model card may not be the correct name required by the API. This results in a "404 Not Found error".

Workaround: To find the exact model name required for the API call (e.g., `nvidia/llama-3.3-nemotron-super-49b-v1`), query the `/v1/models` API endpoint first.

Analytics and tiered tables

Updating tiered, partitioned tables fails with PGAA error

Description: When attempting to execute an `UPDATE` statement on a large, tiered and partitioned table, the operation fails with the message `ERROR: system columns are not supported by PGAA scan`. This issue occurs even when the target partition for the update uses a standard heap access method (that is, it isn't an actively tiered Iceberg table), indicating a conflict in how the Analytics Accelerator (PGAA) processes the partitioned table structure during a modification query.

HM console and observability

Tags for active model clusters are not displayed on the model details screen

> **Tip**
>
> Resolved in HM versions 1.3.1 and later.

Description: In the table showing active model clusters utilizing a specific model on the **Model Cluster Details** screen, the **Tags** field is empty.

Workaround: Model cluster tags can still be viewed correctly on the dedicated **Model Cluster Details** page.

Chat model cluster metrics are missing from the Grafana model overview dashboard

> **Tip**
>
> Resolved in HM versions 1.3.1 and later.

Description: Metrics for deployed chat model clusters are not displayed on the Grafana model overview dashboard, impacting observability for these specific AI components.

User-created Grafana dashboards do not persist after platform redeployment/upgrade

Description: Dashboards created by users directly within the Grafana application are not stored in persistent storage. They disappear when the Grafana pods are updated, redeployed, or restarted (e.g., during an EKS auto-update or a Hybrid Manager upgrade).

Workaround: Any custom dashboards must be backed up externally by

exporting the dashboard as JSON . After an upgrade, they must be manually imported back into Grafana.

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

Description: Users who have access to a large number of projects (typically more than 50) may encounter an HTTP 431 error when navigating to the **Estate** page in the HM console. This error occurs because the authorization headers generated for the request exceed the size limits of the upstream proxy or load balancer. This issue prevents the HM console from loading estate results and can also affect machine users or automation scripts calling the `/api/v1/estate` endpoint directly.

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.

Migrations

Known issues pertaining to the HM Migration Portal, data migration workflows, and schema ingestion workflows are maintained on a dedicated page within the Migrating databases documentation. This allows for focused communication within the relevant section of the 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.

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 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.

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 .

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.

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.

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*

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 and configuring Hybrid Manager

4.1 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—particularly in *Phase 2: Gathering your system requirements* or *Preparing your environment* are the most common cause of installation failure.

4.2 Phase 1: Planning your architecture

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 Hybrid Manager.
- **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, 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.

4.3 Phase 2: Gathering your system requirements

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 file (`values.yaml`).
- **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 entries in the ``**values.yaml**`` file for Hybrid Manager.
- **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.

4.4 Phase 3: Deploying your Kubernetes cluster

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 update critical `values.yaml` 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.

4.5 Phase 4: Preparing your environment

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

- **Goal:** Provision necessary security and configuration resources.

- **Key activities:**
  - Provision essential **secrets and configuration resources** as specified in `values.yaml` .
  - 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.

4.6 Phase 5: Installing Hybrid Manager

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 environment and deployment settings.
  - Deploy Hybrid Manager using customized `values.yaml` entries.
  - Confirm successful installation and operational readiness.
- **Documentation:** *See Hybrid Manager Installation*

> **Note**
>
> **Deployment responsibility:** Installation of Hybrid Manager is optimally executed by the **customer's SRE team**, and the customer owns its lifecycle operation. If you are using an EDB managed service (CX), 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.

4.7 Phase 6: Exploring your post-installation options

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*

4.8 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
2. Configuring image discovery
3. Enabling KMS for TDE
4.

4.8.1 Planning your architecture

Overview

Prerequisites

- *A list of the business goals that your architecture plan should enable (examples: bugdet constraints, desired uptime, 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 Helm chart configuration file: ``values.yaml``.*

> **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.

Next phase: Phase 2: *Phase 2: Gathering your system requirements*

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 immmediately, and where do you plan to expand over the next 1-2 years?

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 (e.g., 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 (i.e., 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 (e.g., 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)
- **Google GKE** (Google Kubernetes Engine)
- **Rancher RKE2** (Rancher Kubernetes Engine)
- **Red Hat OpenShift** (RHOS)

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

HM distributed reference architecture

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 (e.g., “City 1” and “Data Center 1”).
- **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 (e.g., control-plane).
- **Compute Machine:** The virtual machines (e.g., 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).

Deployment architectures

Use reference architectures A-D below as reference models to decide which topology matches your ” Target state .”

> **Note**
>
> This legend above also applies to reference architectures A-D below.

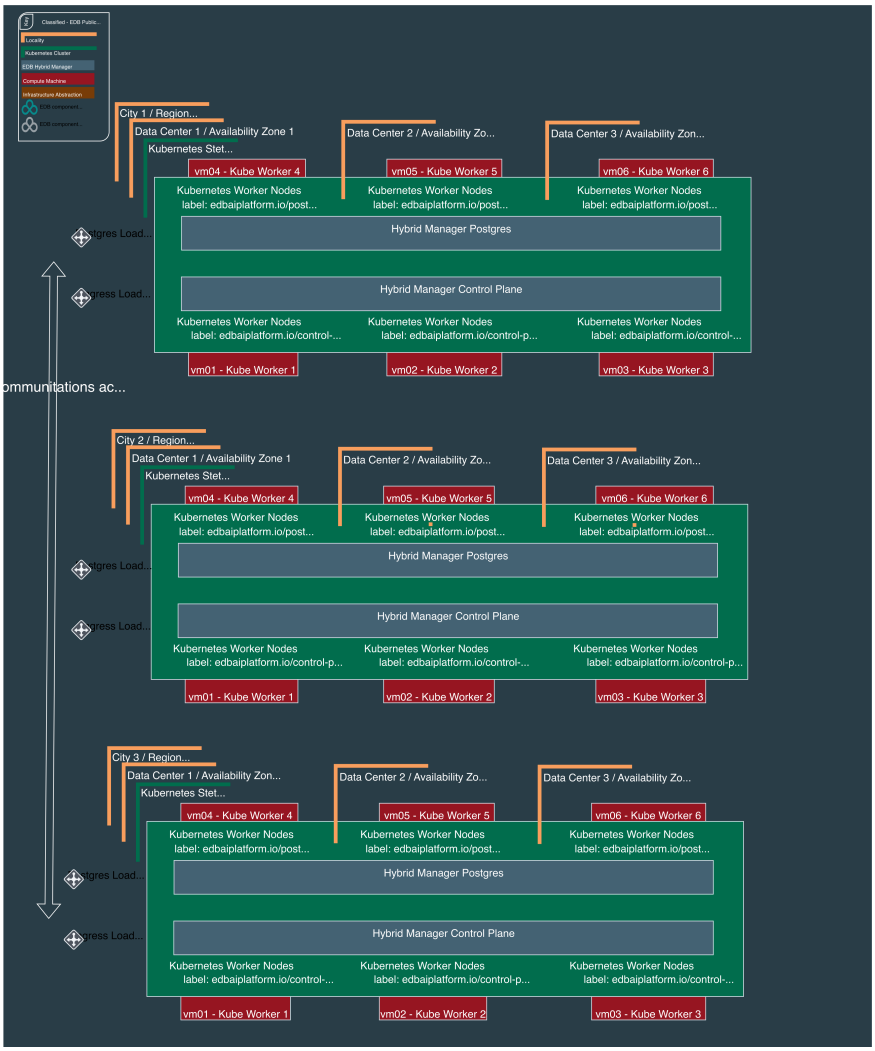


Fig. 1: HM reference architecture

A. Minimum Control Plane

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).

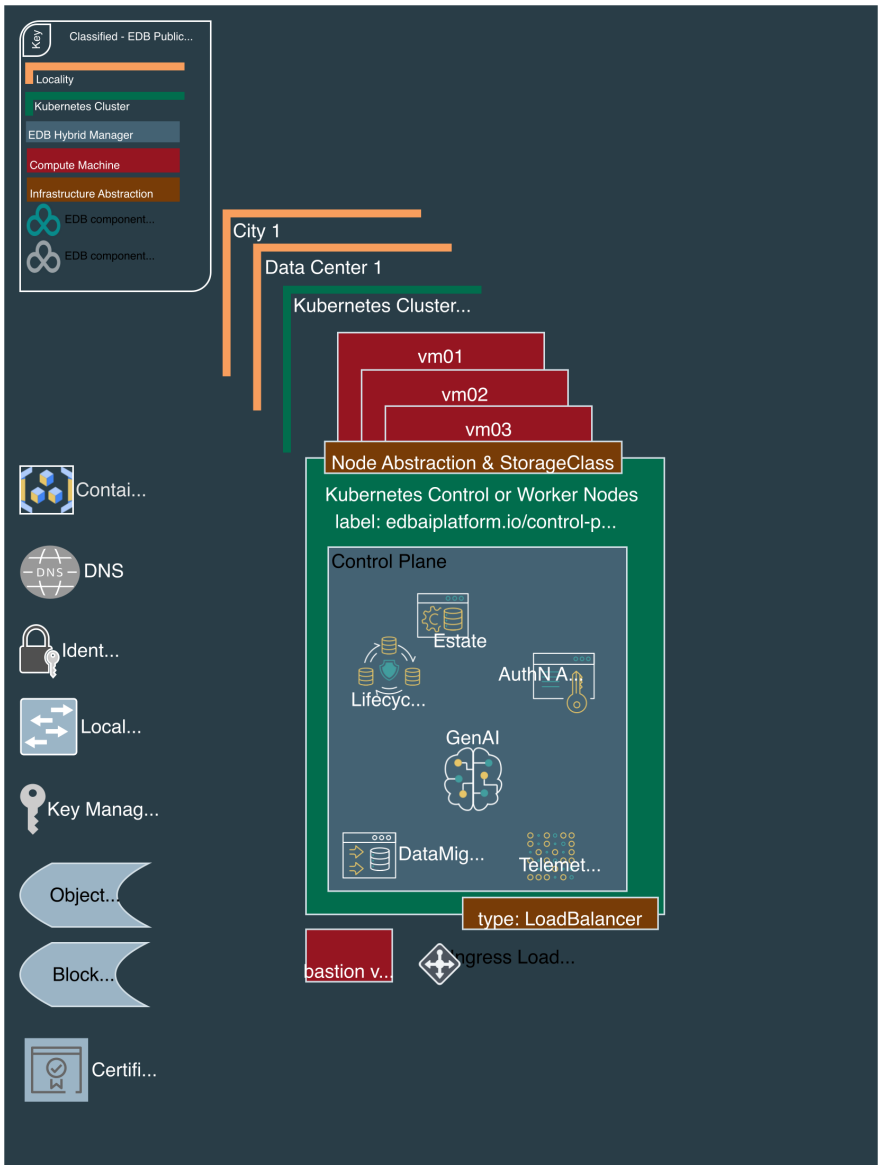


Fig. 2: HM minimum

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 Hybrid Manager
- **Database Migration Assistant:** Facilitates the movement of data from external sources into the platform. * *See:* Migrating databases with Hybrid Manager
- **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 Hybrid Manager

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

B. HM Data Plane (Postgres life cycle orchestration)

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.

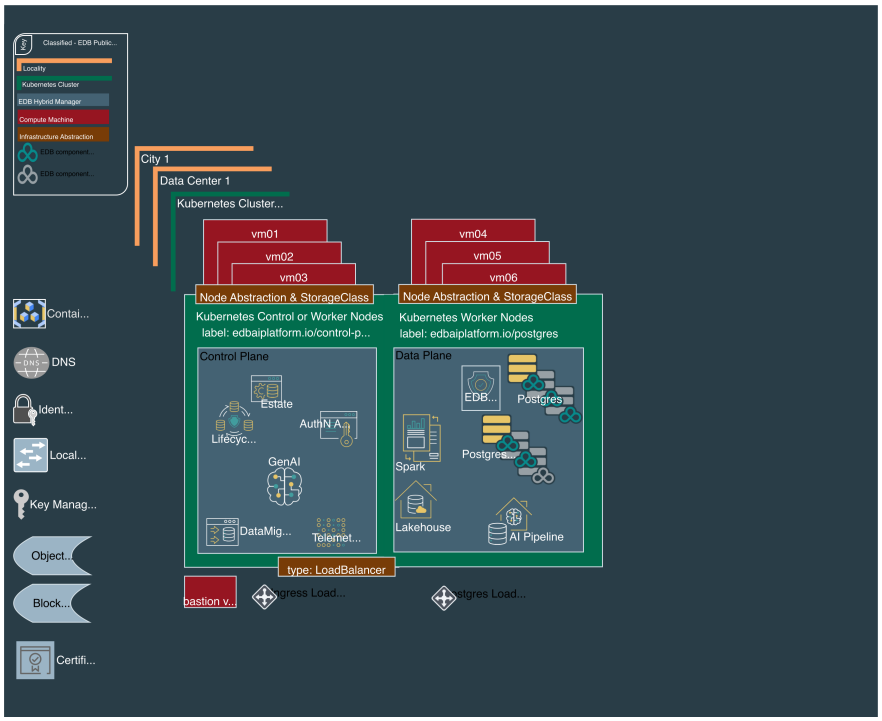


Fig. 3: HM Data Plane

C. Fully featured deployment

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

D. Multi-Location (Hub and spoke)

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 for example.

Impact on configuration

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, either recording them or continuing/beginning the configuration file in Phase 2: Gathering your system requirements and/or uses these specs to build the configuration file, `values.yaml` in ** *Preparing your environment* **.

Configuration details

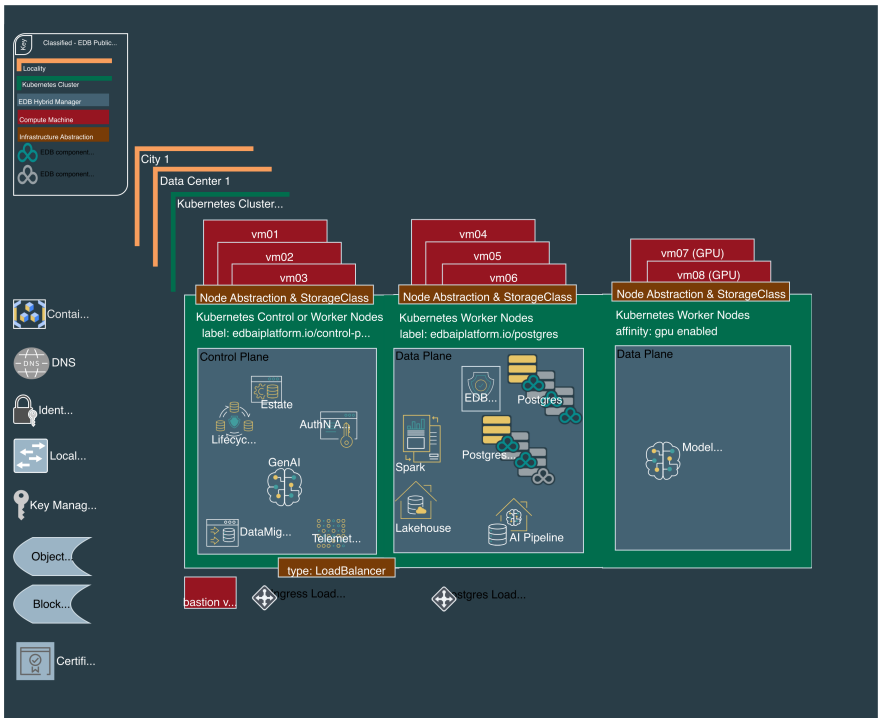


Fig. 4: HM fully featured

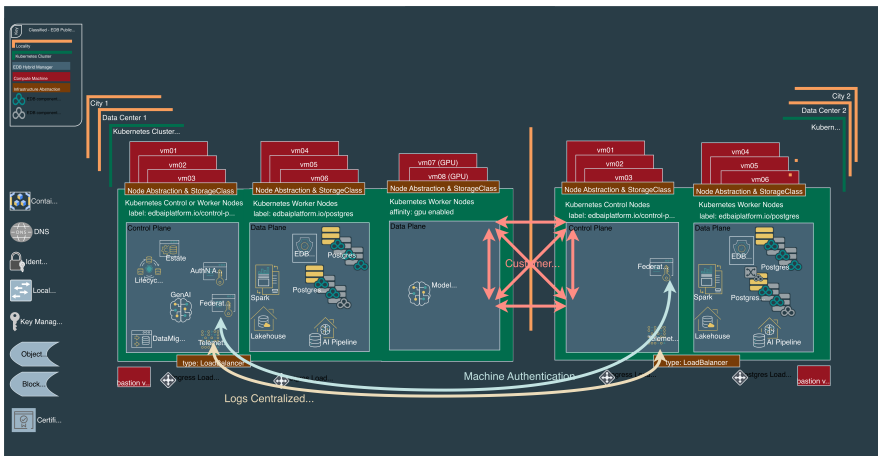


Fig. 5: HM multi-location

Architecture decision	Config parameter (*values.yaml*)	Example value
Kubernetes Platform	*system*	*eks*, *gke*, *rhos*
Target location	*parameters.upm-beacon.beacon_locatic*	*aws-us-east-1*
Provisioning mode	*beaconAgent.provisioning.provider*	*aws* or *gcp*

Impact on configuration file

Here is how your decisions map to the final configuration file structure of the HM Helm chart `values.yaml` you create in Phase 3:

```
system: <Kubernetes_Flavor> # e.g., rhos, rke2, eks, gke
bootstrapImageName: [https://docker.enterprisedb.com/pgai-platform/edbpgai-bootstrap/
↪bootstrap-](https://docker.enterprisedb.com/pgai-platform/edbpgai-bootstrap/
↪bootstrap-)<Kubernetes_Flavor>
bootstrapImageTag: <Version>
parameters:
  upm-beacon:
    beacon_location_id: <Deployment_Location_Name> # Identified in Phase 1: a simple␣
↪string which will be a hint in the UI to identify this location.
beaconAgent:
    provisioning:
        provider: <Provider_Name> # AWS or 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 <Phase 2: Gathering your system requirements>` to verify that your infrastructure can match your design in your ADR.

4.8.2 Gathering your system requirements

Overview

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 a number of inital inputs defined for the HM Helm chart ``values.yaml`` to start this file now, continue adding to it from Phase 1 if already started, or to use to populate the ``values.yaml`` at the end of Phase 4 (:ref:`Preparing your environment <Preparing your environment>`).*

> **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:` <#2-kubernetes-platform-verification>`
- 3. :ref:` <#3-compute-node-requirements>`
- 3.1 *3.1 Control plane nodes*
- 3.2 *3.2 Data plane nodes*
- 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:` <#6-block-storage-database--logs>`
- 6.1 *6.1 Volume snapshots*
- 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)

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 * Amazon EKS * Google GKE

- On premises platforms: * Rancher RKE2 * Red Hat OpenShift (RHOS)

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 values in the `values.yaml` file.

Record these now for the next phase:

Parameter	YAML Key	Required Value
System Flavor	*system*	*eks*, *gke*, *rke2*, or *rhos*
Provider	*beaconAgent.provisioning.provider*	*AWS* or *GCP* (If on cloud)
OpenShift	*beaconAgent.provisioning.openshift*	Set to *true* **only** if using Red Hat OpenShift.

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 ** *Phase 1: Planning your 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"*
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.

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

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

3.2 Data plane nodes

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: Planning your architecture* .

3.3 AI model nodes

- **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* .

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:** Cilium is known to be more efficient at very high numbers of pods. However, because Postgres is the center of Hybrid Manager, extreme pod density is rarely the bottleneck compared to database I/O. Extra effort to optimize the stack to acheive slightly better results. Depending on the customers use case it may be justified. However, more often than not, good solution design can adapt to most infrastructure limitations.
- **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. 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

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

- **TCP:** The load balancer must be configured for **TCP passthrough (TCP-only)** and must **not** terminate SSL. SSL is terminated by the HM Ingress Gateway.
- **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*:

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

Configuration dependencies

Parameter	YAML Key	Value / Format
Enable LB	*beaconAgent.provisioning.loadBalancersEnabled*	*true*

continues on next page

Table 6 – continued from previous page

Parameter	YAML Key	Value / Format
Annotations	*resourceAnnotations*	Provider-specific list (see below).

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 are the Hybrid Manager Control Plane ports:

Port	Protocol	Description
443	HTTPS	HM Portal (HTTPS ingress)
8444	TCP	HM internal API
9443	gRPC	Beacon gRPC API
9445	TCP	Spire TLS

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

5.1.2 NodePort alternative

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

- **Configuration:** You must configure `beaconAgent.provisioning.loadBalancersEnabled: false` and define a `nodePortDomain` in your HM Helm chart `values.yaml` 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 `nodePortDomain` (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	YAML Key	Value / Format
Enable LB	*beaconAgent.provisioning.loadBalancersEnabled*	*false*

continues on next page

Table 9 – continued from previous page

Parameter	YAML Key	Value / Format
Domain	*nodePortDomain*	Base DNS domain (e.g., *nodes.myorg.com*).

Default NodePort assignments

NodePort Variable	Port	Description
ingress_http_node_port	**32542**	HM Portal (HTTP)
ingress_https_node_port	**30288**	HM Portal (HTTPS)
ingress_grpc_tls_node_p	**30290**	Beacon gRPC
ingress_spire_tls_node_p	**30292**	Spire TLS
ingress_beacon_spire_tls_node_port	**30294**	Beacon Spire TLS
ingress_thanos_query_tls	**30296**	Thanos-query TLS (inter-cluster metrics)
ingress_fluent_bit_tls_node_port	**30298**	Fluent-bit TLS (inter-cluster logs)
enable_server_session	n/a	Enable server stored session ("true" or "false")

Postgres NodePorts

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

5.2 DNS

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 configuration in the HM Helm chart `values.yaml` in the next phase:

Domain Purpose	YAML key	Description
Portal Domain	*parameters.global.portal_domain_name*	The host name for the HM Portal UI.
Agent Domain	*parameters.upm-beacon.server_host*	The host name through which the Beacon Server API is reachable.
Migration Domain	*parameters.transporter-rw-service:dom*	The domain name for the internal Transporter migration service.

continues on next page

Table 11 – continued from previous page

Domain Purpose	YAML key	Description
Migration URL	*parameters.transporter-dp-agent:rw_se*	Full URL derived from the domain above: *https://<Migration Domain>/transporter*
NodePort Base	*beaconAgent.provisioning.nodePortDomain*	**(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 `beaconAgent.provisioning.loadBalancersEnabled:` is `true` in the values.yaml`.
- **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 `nodePortDomain` 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 `nodePortDomain` 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

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.
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

Once you select a strategy, record the corresponding values for the `values.yaml` later, or in the file if you have already started it:

Option A: Custom cert-manager issuer

You can leverage your cert-manager for automatic certificate handling by specifying an `Issuer` or `ClusterIssuer` kind and name in the `values.yaml` :

Parameter	YAML Key	Value / Format
Issuer kind	*parameters.global.portal_certificate_issuer_kind:*	*<ClusterIssuer or Issuer>*
Issuer name	*portal_certificate_issuer_name:*	*<my_issuer>*

Option B: Custom CA (BYO CA)

If you do not have your own custom cert-manager issuer, you can instead use your own Certificate Authority (CA).

- With this strategy, after deploying your Kubernetes cluster, you create a Kubernetes secret and specify in the `values.yaml` :

Parameter	YAML Key	Value / Format
CA Secret	*parameters.global.ca_secret_name*	Name of the pre-created K8s Secret.

Option C: Custom certificate (BYO cert)

If you don't choose to use a customer cert-manager issuer or custom CA, you should use a custom certificate.

- With this strategy, after deploying your Kubernetes cluster, you create a Kubernetes secret and specify in the `values.yaml` :

Parameter	YAML Key	Value / Format
Cert Secret	*parameters.global.portal_certificate_secret*	Name of the pre-created K8s Secret.

This secret requires an export of the entire certificate chain in the public certificate file and the unencrypted private key.

Option D: Self signed certificates

Configuration dependencies

N/A, default

6. Block storage (Database & logs)

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

The configuration in `values.yaml` defines the default class used for the **HM Control Plane**. 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

Configuration dependencies

Parameter	YAML Key	Value / Format
Storage Class	*parameters.global.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
```

7. Object storage

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

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 *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 keys that are affected in the `values.yaml` you create in the Phase 3.

Parameter	YAML Key	Value / Format
Storage Class	*parameters.global.storage_class*	The exact name of your K8s StorageClass.

8. Container registry

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` .

Supported providers & authentication

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!)

Configuration dependencies

Provisioning actions

- **Sync images to your local container registry:** You must use `edbctl` to synchronize images if using a private registry. * *Syncing images to a local private 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.*

You need to collect these specific values to populate your `values.yaml` in the next phase (Phase 3).

Parameter	YAML Key	Value / Format
Bootstrap Image	*bootstrapImageName*	*<Registry Domain>/pgai-platform/edbpgai-bootstrap/bootstrap*
Global Registry	*containerRegistryURL*	*<Registry Domain>/pgai-platform*
Discovery Registry	*beaconAgent.provisioning.imagesetDiscoveryContainerRegistryURL*	*<Registry Domain>/pgai-platform*
Discovery Auth	*beaconAgent.provisioning.imagesetDiscoveryAuthenticationType*	*<Auth Type>* (e.g., *token*, *basic*, *eks_managed_identity*)
TLS Validation	*imagesetDiscoveryAllowInsecureRegistry*	Set to *true* if using TL without certificate validation.

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.

9.1 External IdP (Recommended)

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):

Configuration dependencies

Configuration file

Parameter	YAML Key / Resource Name	Value / Format
Client Secret	*pgai.portal.authentication.clientSecret*	The OIDC client secret string.
Connector Config	*pgai.portal.authentication.idpConnectors*	List of connector configurations.
Connector Type	*...idpConnectors[0].type*	Must be *ldap* or *saml* (parameters vary by type).

Provisioning actions

- **CA Bundle Secret**: K8s Secret: `beaconator-ca-bundle` : Create in **all namespaces** containing the IdP trust chain.
- **Dex Secret**: K8s Secret: `upm-dex` : Create in `upm-dex` namespace.

9.2 Static user alternative

> **Warning**
>
> This method is **strongly discouraged** for production and should be used for pilot or proof of concept activity only.

- **Context:** There is a single mandatory static `User-0` created for installation purposes.

While you *can* create additional static users at provisioning time, there is **no UI provided** to manage them, as this is not a secure pattern.

- **Minimum action:** You must set the password for the required static user (`User-0`).

Configuration dependencies

Configuration file

If choosing this path, you need to configure the following in `values.yaml` :

Parameter	YAML Key	Value / Format
Password Hash	*pgai.portal.authentication.staticPasswords.hash*	Bcrypt hash string of the password.
User Email	*pgai.portal.authentication.staticPasswords.email*	A valid email address (e.g., *admin@example.com*).
Username	*pgai.portal.authentication.staticPasswords.username*	The login username (e.g., *admin*).
User ID	*pgai.portal.authentication.staticPasswords.userID*	A unique string identifier (e.g., *user-0*).

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 for the `values.yaml` for *Phase 4 guide* :

Parameter	YAML Key	Value / Format
KMS Providers	*beaconAgent.transparentDataEncryptionMethods*	List of enabled providers (e.g., *aws-kms*, *google-cloud-kms*).

continues on next page

Table 23 – continued from previous page

Parameter	YAML Key	Value / Format
Auth Strategy	*auth_type* (nested under provider)	*workloadIdentity* or *credentials*.

- **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.

Use this **Master Inventory** to ensure you have every required value before proceeding to Phase 3.

General & Platform

Source	Requirement	YAML Key	Condition
Phase 1	**K8s Flavor**	*system*	**Required** (e.g., *eks*, *gke*, *rhos*)
Phase 1	**Provider**	*beaconAgent.provisioning.provider*	**Required** (e.g., *AWS*, *GCP*)
Phase 1	**OpenShift**	*beaconAgent.provisioning.openshift*	**Required** if using RHOS (*true*)
Phase 1	**Location**	*parameters.upm-beacon.beac*	**Required** (String ID)

Networking & Ingress

Source	Requirement	YAML Key	Condition
Phase 2	**Portal Domain**	*global.portal_domain_name*	**Required**
Phase 2	**Agent Domain**	*upm-beacon.server_h*	**Required**
Phase 2	**Migration Domain**	*transporter-rw-service*	**Required**
Phase 2	**Migration URL**	*transporter-dp-agent:*	**Required**
Phase 2	**Load Balancer**	*beaconAgent.provisioning.loadBalancersEnabled*	**Required** (*true* or *false*)
Phase 2	**LB Annotations**	*resourceAnnotations*	**Required** if using CSP LB (e.g., AWS)
Phase 2	**NodePort Domain**	*nodePortDomain*	Required if *loadBalancersEnabled* is *false*

Storage & Registry

Source	Requirement	YAML Key	Condition
Phase 2	**Storage Class**	*parameters.global.storage_class*	**Required**

continues on next page

Table 26 – continued from previous page

Source	Requirement	YAML Key	Condition
Phase 2	**Global Registry**	*containerRegistryURL*	**Required**
Phase 2	**Bootstrap Image**	*bootstrapImageName*	**Required**
Phase 2	**Discovery Reg.**	*beaconAgent.provisioning.imagesetDiscoveryContainerRegistryURL*	**Required**
Phase 2	**Registry Auth**	`beaconAgent.provisioning.imagesetDiscoveryAuthenticationType`	**Required**
Phase 2	**Insecure TLS**	*imagesetDiscoveryAllowInsecureRegistry*	Optional (If using self-signed registry)

Security & Identity

Source	Requirement	YAML Key	Condition
Phase 2	**IDP Secret**	*pgai.portal.authentication.clientSecret*	**Required** (if using OIDC)
Phase 2	**IDP Config**	*pgai.portal.authentication.idpConnectors*	**Required** (if using OIDC)
Phase 2	**Static User**	*pgai.portal.authentication.staticPasswords.**	**PoC Only** (if no IdP)
Phase 2	**TDE Keys**	*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	YAML Key	Description
A. Cert Manager	*parameters.global.portal_certificate_issuer_kind*	**Required.** The Resource Kind (e.g., *ClusterIssuer*).
A. Cert Manager	*portal_certificate_issuer_name*	**Required.** The Resource Name (e.g., *letsencrypt-prod*).
B. BYO CA	*parameters.global.ca_secret_name*	**Required.** The name of your CA Secret.
C. BYO Cert	*parameters.global.portal_certificate_secret*	**Required.** The name of your Certificate Secret.
D. Self-signed	N/A	**Default behavior.** (No configuration required).

Example `values.yaml`

According to the decisions made in previous steps, your `values.yaml` , if you have already started it, should reflect those choices.

Here is an example of a production oriented `values.yaml` for installation which using all options "required for optimal experience". If you are starting now, or have started your `values.yaml` already, your configuration file may be different if you are not employing the options "required for optimal experience above.

```
system: <Kubernetes>
bootstrapImageName: <Container Registry Domain>/pgai-platform/edbpgai-bootstrap/
↪bootstrap-<Kubernetes>
bootstrapImageTag: <Version>
containerRegistryURL: "<Container Registry Domain>/pgai-platform"
parameters:
  global:
    portal_domain_name: <Portal Domain>
    storage_class: <Block Storage>
    portal_certificate_issuer_kind: <ClusterIssuer>
    portal_certificate_issuer_name: <my-issuer>
    trust_domain: <Portal Domain>
  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"
  transparentDataEncryptionMethods:
    - <available_encryption_method>
pgai:
  portal:
    authentication:
      idpConnectors:
        - config:
            caData: <base64 encyrption of Certificate Authority from SSO provider>
            emailAttr: email
            groupsAttr: groups
            entityIssuer: https://<Portal Domain>/auth/callback
            redirectURI: https://<Portal Domain>/auth/callback
            ssoURL: https://login.microsoft.com/<azure service identifier>/saml2
            usernameAttr: name
          id: azure
          name: Azure
          type: saml
resourceAnnotations:
  - name: istio-ingressgateway
    kind: Service
    annotations:
```

(continues on next page)

(continued from previous page)

```
    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 some inputs to your HM Helm chart `values.yaml` to be added now if you are building your file as you go, or used as inputs to build the file at the end of *Preparing your environment* .

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

4.8.3 Deploying your Kubernetes cluster

Overview

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](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. * **AWS:** Install AWS CLI

```
* **Google Cloud:**  `Install gcloud SDK <https://cloud.google.com/sdk/docs/install>`_

* **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/](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 *Phase 2: Gathering your system requirements* .

Public cloud providers

Provider	Official documentation	EDB Knowledge Base	Key considerations
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-Gui	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-cl	` **HM on GKE Guide** <https://knowledge.enterprisedb.com/hc/en-us/articles/24212699192732-Gui	Requires Workload Identity enabled and specific VPC firewall rules.

Private cloud and on-premises

Distribution	Official documentation	EDB Knowledge Base	Key considerations
Rancher RKE2	*Troubleshooting AI Factory on Hybrid Manager*	` **HM on RKE2 Guide** <https://knowledge.enterprisedb.com/hc/en-us/articles/24212371387420-Gui	Requires manual setup of Longhorn (or similar) storage and MetalLB for ingress.
Red Hat OpenShift	*Troubleshooting AI Factory on Hybrid Manager*	` **HM on OpenShift Guide** <https://knowledge.enterprisedb.com/hc/en-us/articles/24212799018780-Gui	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`

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

Data plane

Required labels & taints:

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

```
#  Example Node Pool Specification

spec:
    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

- **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 *Set up your management workstation* can reach all relavent endpoints for configuration in the next phase: *Preparing your environment* .
2. Confirm that your deployed cluster matches the relevant specifications you defined in *Phase 2: Gathering your system requirements* :

Compute resources

Source: *Gathering your system requirements>Compute (Node 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.*

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:**

```
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 apply the HM configuration file: `values.yaml` .

** *Preparing your environment* →**

4.8.4 Preparing your environment

Overview

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 ``values.yaml`` configuration file ready for Hybrid Manager (HM) installation*

> **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.

Next phase: Phase 5: *Install Hybrid Manager*

Phase 4 guide

Now that your Kubernetes cluster is running, you must prepare it for the HM platform.

This involves establishing administrative access through a remote management workstation, staging required artifacts and secrets into the cluster, and translating your architectural decisions into the final configuration file.

Preparing your environment, Phase 4, covers:

- **Creating necessary infrastructure secrets and implementing TLS:**
- *Create Image Pull secret* * *Create Object Storage secret* . * *Implement TLS* (chosen in *Phase 2: Gathering your system requirements*) * *(A) self-managed Cert-Manager issuer*

```
*  :ref:`(B) custom CA <#option-b-custom-certificate-authority-ca>`

*  :ref:`(C) custom certificate <#option-c-custom-certificate>`

*  :ref:`(D) self-signed default <#option-d-self-signed-certificates-default>`
```

- **Creating other necessary secrets**
- *Creating GenAI Builder secret* (requires Fernet secret) * *Creating Catalog secret* (requires Fernet secret) * *Creating Catalog secret* .
- **Configuring advanced features:**
- *Configuring multi-location architecture*
- Constructing complex YAML configurations for your **Identity Provider** . * *Implementing KMS for TDE* .
-

Completing environment preparation:

- Configuring the internal backup folder and securing the mandatory **User-0** admin account.
- **Finalizing the cluster configuration (`values.yaml):** <#finalizing-the-cluster-configuration-valuesyaml>` * Assembling all the relevant above inputs into the final Helm chart `values.yaml` file.
- *Validating the configuration file and runtime*
- Validating the `values.yaml` file and runtime environment, so that they are ready for HM installation.

Creating necessary secrets

> **Warning**
>
> The following are not optional configurations. These preparatory steps **must be completed** in order to successfully complete your installation. This configurations do not require you utilize the associated features; it only prepares a secure implementation in case it's needed.

Create Kubernetes secrets for any required credentials, such as object storage credentials, database access tokens, or any other sensitive information.

Create Image Pull secret

> **Warning**
>
> It is necessary to create an Image Pul secret for HM to install successfully.

Dependencies

- k8s secret `edb-cred` (defaults to `edb-cred` in `values.yaml`)

Use `edbctl` to create the `ImagePullSecret` namespace and required secrets.

1. Create the necessary namespaces and pull secrets:

```
edbctl image-pull-secret create \
  --username <container registry username> \
  --password <container registry passowrd> \
  --registry <local registry URI>
```

1. When prompted with `Proceed? [y/N]` with the current Kubernetes context, select `y` .

You should see output similar to:

```
2025/02/10 10:10:10 Creating Kubernetes Namespaces and ImagePullSecrets with the␣
↪provided credentials...
2025/02/07 15:29:08 Namespaces and ImagePullSecrets creation completed
```

1. Verify the secret creation:

```
edbctl image-pull-secret list
```

```
*Output example:*
```

```
Current Kubernetes context is: <your-KubeContext>
Namespace edbpgai-bootstrap: exists, all set!
  Secret edb-cred: exists, all set!
Namespace upm-replicator: exists, all set!
  Secret edb-cred: exists, all set!
```

Spot validation

1. Check that the `edb-cred` secret exists in the target namespace:

```
kubectl get secret edb-cred -n edbpgai-bootstrap
```

1. Check that the default service account references the `edb-cred` secret:

```
kubectl get serviceaccount default -n edbpgai-bootstrap -o yaml | grep edb-cred
```

1. Deploy a test pod (lasso) to validate image pull:

```
kubectl run lasso \
  --rm -it \
  --image=[docker.enterprisedb.com/pgai-platform/lasso:latest](https://docker.
↪enterprisedb.com/pgai-platform/lasso:latest) \
  --restart=Never \
  -n edbpgai-bootstrap \
  --image-pull-policy=Always \
  -- bash
```

1. (Optional) Describe the pod to confirm imagePullSecrets are set:

```
kubectl describe pod lasso -n edbpgai-bootstrap | grep -i imagepull
```

Create Object Storage secret

> **Warning**
>
> It is necessary to create an Object Storage secret for HM to install successfully. 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.

To implement the *Phase 2: Gathering your system requirements* , you must create a secret named `edb-object-storage` in the `default` namespace.

Select the configuration matching your provider:

AWS IAM (EKS or ROSA)

```
apiVersion: v1
kind: Secret
metadata:
  name: edb-object-storage # the name cannot be changed
  namespace: default # the namespace 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>
```

AWS / Other K8s (Static Credentials)

```
apiVersion: v1
kind: Secret
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>
```

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>
```

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>
```

Other S3 Compatible Storage

```
apiVersion: v1
kind: Secret
metadata:
  name: edb-object-storage
  namespace: default
stringData:
    auth_type: credentials
    # Optional: Base64 CA bundle if not using a well-known CA
    aws_ca_bundle_base64: <CA_BUNDLE_BASE64>
    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>
    # Set to true if server-side encryption is disabled on the bucket
    server_side_encryption_disabled: <true|false>
```

Creating other necessary secrets

> **Warning**
>
> The following are not optional configurations. These preparatory steps **must be completed** in order to successfully complete your installation. This configuration does not require you utilize the associated feature; it only prepares a secure implementation in case it's needed.

There are other secrets that must be created for the HM installation to be successful.

Creating GenAI Builder secret

> **Warning**
>
> It is necessary to create a GenAI Builder secret for HM to install successfully.

Even if you are not planning to use GenAI Builder, creating a secret is a necessary step.

You must manually set up encryption keys and a dedicated S3-compatible bucket with specific CORS policies.

Preparation checklist:

1. **Namespace:** Create the `upm-griptape` namespace.
2. **Encryption:** Generate a Fernet key and create the `fernet-secret` inside that namespace.
3. **Storage:** Create a dedicated Object Storage bucket (DataLake).
4. **Networking:** Apply CORS configuration to the bucket to allow access from the Portal domain.
   •

SOP: Enabling GenAI Builder

- *Guidance: Scripts for generating Fernet keys and CORS JSON configurations.*

Creating Catalog secret

> **Warning**
>
> It is necessary to create a Catalog secret for HM to install successfully.

Similar to GenAI, even if not using the Catalog service, it is required to create encrypted storage credentials.

Preparation checklist:

1. **Namespace:** Create the `upm-lakekeeper` namespace.
2. **Encryption:** Generate a Fernet key and create the secret in that namespace.
3. **Configuration:** Configure the *installation-time* catalogs (Infrastructure/OS images) in `values.yaml` .

•

SOP: Enabling data catalogs

- *Guidance: Commands for creating the confounding key, namespace, and secret.*

Customizing Migration Portal secrets

> **Warning**
>
> It is necessary to customize your Migration Portal secrets for HM to install successfully.

Create secrets in namespaces `edb-migration-portal` and `edb-migration-copilot` .

•

SOP: Customizing Migration Portal secrets for secure internal communication

- *Guidance: Commands for creating custom secrets for using Migration Portal.*

Implement TLS

You must implement the TLS strategy chosen during the *Phase 2: Gathering your system requirements* phase, some of which require secrets to be created.

Option A: Custom cert-manager issuer

- **Use case:** You have an existing `ClusterIssuer` or `Issuer` (e.g., Let's Encrypt, HashiCorp Vault, or a corporate CA) running in the cluster.

> **Note**
>
> For this option, you do not need to pass a secret to HM. The existing Issuer already manages its own credentials.

Dependencies

- Ensure your existing `ClusterIssuer` or `Issuer` is configured and available in the target cluster.
- Specify the issuer name and kind in `values.yaml` : * `parameters.global.portal_certificate_issuer_kind:<ClusterIssuer or Issuer>` * `portal_certificate_issuer_name:<my_issuer>`
- **SOP: Set up a custom cert-manager issuer for the HM Portal **

Option B: Custom certificate authority (CA)

- **Use case:** You want HM to create an cert-manager issuer for you and use your own corporate CA to sign certificates.

Dependencies

- Set up your own custom CA.
- Create a Kubernetes secret for `<ca_secret_name>` containing your CA signing keypair (cert and key).
- Specify the `<ca_secret_name>` in the `values.yaml` : `parameters.global.ca_secret_name:(<ca_secret_name)` .
- **SOP: :ref:`Bring your own private certificate authority <Bring your own private certificate authority>` .**

Option C: Custom certificate

Use case: You have your own x.509 certificate and would like to change it out for the default self-signed HM certificate

Dependencies

- Create a Kubernetes secret for `<my_portal_certificate>` including an export of the entire certificate chain in the public certificate file and the unencrypted private key.
- Specify the `<my_portal_certificate>` in the `values.yaml` : `parameters.global.portal_certificate_secret:<my_portal_certificate>` .
- **SOP: Set up a custom x.509 certificate for the Hybrid Manager Portal **

Option D: Self-signed certificates (Default)

- **Use case:** Non-production testing.

HM automatically generates self-signed certificates by **default**.

Dependencies

N/A

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 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 `values.yaml` .

•

SOP: Configuring multiple data centers for Hybrid Manager

- *Guidance: Detailed steps for setting up SPIRE federation and cross-DC wiring.*

Configuring identity providers (IdP)

Action: Construct the `idpConnectors` configuration block.

Configuring an IdP requires constructing a complex array in your `values.yaml` .

You must gather specific values from your IdP provider (Okta, Active Directory, etc.) to populate the `portal.authentication.idpConnectors` section.

•

SOP: Configuring your own identity provider

- *Guidance: Detailed YAML examples for LDAP and SAML integration.*

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 identify the necessary `values.yaml` flags and create the keys.

- HashiCorp Vault
- AWS Key Management Service (KMS)
- Google Cloud Key Management Service (Cloud KMS)

Implement the configuration strategies decided upon during the *Phase 1: Planning your architecture* and *Phase 2: Gathering your system requirements* phases.

Completing environmental preparation

Set up an internal backup folder

This is a universally unique folder name for the HM backups for disaster recovery and multi-location.

Dependencies

- `values.yaml` parameter `global.internal_backup_folder` .

Change mandatory static User-0 password

Dependencies

You must configure the following four values under `pgai.portal.authentication.staticPasswords` in your `values.yaml` :

1. ``**userID**`` : Must be set to `c5998173-a605-449a-a9a5-4a9c33e26df7` .
2. ``**username**`` : The login username for the admin account.
3. ``**email**`` : The email address associated with the admin account.
4. ``**hash**`` : The bcrypt hash of your desired password.

Use a hash for your new password

To generate the hash for your new password:

Option 1:

```
bcrypt hash of the string "password": $(echo password | htpasswd -BinC 10 admin | cut␣
↪-d: -f2)
```

Option 2:

```
echo password | htpasswd -BinC 10 admin | cut -d: -f2
```

Finalizing the cluster configuration (`values.yaml`)

You now construct (or update if you have previously created) the final `values.yaml` file.

This file references your **system requirements** (Storage, Network, etc), **environment settings**, and the **feature configurations** you just prepared.

Below is a template configuration with all of the keys you should have determined for a successful HM install:

```
system: <Kubernetes>
bootstrapImageName: <Container Registry Domain>/pgai-platform/edbpgai-bootstrap/
↪bootstrap-<Kubernetes>
bootstrapImageTag: <Version>
containerRegistryURL: "<Container Registry Domain>/pgai-platform"
parameters:
  global:
    internal_backup_folder: <twelveCharacterString>
    portal_domain_name: <Portal Domain>
    storage_class: <Block Storage>
    portal_certificate_issuer_kind: <ClusterIssuer>
    portal_certificate_issuer_name: <my-issuer>
    trust_domain: <Portal Domain>
  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
```

(continues on next page)

(continued from previous page)

```
beaconAgent:
  provisioning:
    imagesetDiscoveryAuthenticationType: <Authentication Type for the Container␣
↪Registry>
    imagesetDiscoveryContainerRegistryURL: "<Container Registry Domain>/pgai-platform"
  transparentDataEncryptionMethods:
    - <available_encryption_method>
pgai:
  portal:
    authentication:
      idpConnectors:
        - config:
            caData: <base64 encyrption of Certificate Authority from SSO provider>
            emailAttr: email
            groupsAttr: groups
            entityIssuer: https://<Portal Domain>/auth/callback
            redirectURI: https://<Portal Domain>/auth/callback
            ssoURL: [https://login.microsoft.com/](https://login.microsoft.com/)
↪<azure service identifier>/saml2
            usernameAttr: name
          id: azure
          name: Azure
          type: saml
      staticPasswords:
        - email: <email>
          hash: <hashed_password>
          userID: c5998173-a605-449a-a9a5-4a9c33e26df7
          username: <email-or-username>
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
```

Validating the configuration file and runtime

Before proceeding to deployment, validate both your configuration file and the runtime environment.

Validate configuration syntax

Ensure your `values.yaml` is valid and can be processed by Helm.

Replace `<your-registry-url>` with the registry you defined in `values.yaml` (e.g., `docker.enterprisedb.com` or your internal Harbor/Artifactory).

```
helm template edb-pgai oci://<your-registry-url>/pgai-platform/edb-pgai \
  --values values.yaml \
  --version <target-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.

```
kubectl get secrets -n default
```

Check for: `edb-object-storage` , `edb-cred` , and your TLS 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, `values.yaml` is finalized, and then that YAML is validated, you are ready to install HM.

** *Install Hybrid Manager* **

4.8.5 Install Hybrid Manager

Overview

Prerequisites

- *Phase 1: Planning your architecture* (Completed)
- *Phase 2: Gathering your system requirements* (Completed)
- *Phase 3: Deploying your Kubernetes cluster* (Running and validated)
- *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: *Phase 6: Exploring your post-installation options*

With your environment prepared and your configuration validated, you are ready to deploy HM.

This phase involves applying the Helm chart (`values.yaml`) to your cluster and monitoring the operator as it bootstraps the platform services.

Verify final configuration

Before running the install command, perform one final sanity check on the critical values in your `values.yaml` .

These are the most common causes of deployment failures.

Open your ``values.yaml`` and confirm:

Global & image settings

1. ``**containerRegistryURL**`` : Ensure this points to the registry containing your images (e.g., `docker.enterprisedb.com/pgai-platform` or your private mirror).
2. ``**bootstrapImageName**`` : Must match the bootstrap image path defined in *Preparing your environment* .
3. ``**bootstrapImageTag**`` : Must match the target version you are installing.

Domain & network parameters

1. ``**parameters.global.portal_domain_name**`` : The FQDN for the H< UI (e.g., `portal.my-company.com`).
2. ``**parameters.transporter-rw-service.domain_name**`` : The FQDN for the migration service.
3. ``**parameters.transporter-dp-agent.rw_service_url**`` : The full URL for the data plane agent (e.g., `https://<migration-domain>/transporter`).
4. ``**parameters.upm-beacon.server_host**`` : The hostname the beacon agent will use to connect back to the control plane.
5. ``**beaconAgent.provisioning.nodePortDomain**`` : If using NodePort instead of a load balancer, ensure this domain is set.

Security & Authentication

1. ``**pgai.portal.authentication.staticPasswords**`` : You must have the `email` , `hash` , `username` , and `userID` set for the initial admin user (“User-0”).
2. ``**parameters.upm-istio-gateway.cookie_aeskey**`` : Ensure this 16/32-byte key is set for session encryption.
3. ``**transporter-data-operator.fips_enabled**`` : Set this to ``**false**`` . (FIPS support is not yet universal for all components).

> **Note**
>
> Ensure *transporter-data-operator:fips_enabled* is set to *false* unless you have explicitly configured a FIPS-compliant environment.

Install Hybrid Manager

Use the following Standard Kubernetes installation flow, unless you are deploying on RHOS and prefer to *Use OpenShift Console* in the RHOS UI.

Standard Kubernetes

Use the Helm client to install the Hybrid Manager chart (values.yaml). We use the `--install` flag (or `upgrade --install`) to ensure the command is idempotent.

Create namespace

Create a dedicated namespace for the bootstrap operator.

```
kubectl create namespace edbpgai-bootstrap --dry-run=client -o yaml | kubectl apply -
↪f -
```

Run installation

Execute the deployment using the OCI registry path.

- **``[REGISTRY]`` :** Use `docker.enterprisedb.com` (Public) or your private registry domain.
- **``<"VERSION">`` :** The target version tag (e.g., `1.3.0`).

```
helm upgrade
  --namespace edbpgai-bootstrap \
  --install \
  --version <"VERSION"> \
  -f <path>/values.yaml \
  edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap
```

Use OpenShift Console

If you are deploying on RHOS, you can install the operator directly from the Catalog:

1. **Adapt Configuration:** Ensure your `values.yaml` is compatible with OpenShift Security Context Constraints (SCC).
2. **Install from Catalog:** * 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.

Monitor installation progress

Once the Helm command completes, the **bootstrap operator** takes over. It interprets your configuration and spins up the HM platform.

Check HM control plane status

Monitor the custom resource to see the high-level status of the platform:

```
kubectl get hybridcontrolplane edbpgai -o yaml
```

- **Expected output:** You should see an object named `edbpgai` (or similar).
- **Status column:** Look for `Running` or `Ready` . If it says `Reconciling` , wait a few moments.

Check pod health

Verify that all pods in the namespace are coming up healthy.

```
kubectl get pods -n edbpgai-bootstrap
```

What to look for:

- **``edbpgai-operator`` :** Must be `Running` .
- **``edb-portal`` :** The web interface.
- **``edb-beacon`` :** The agent handling cluster communication.
- **``transporter`` :** Services related to migration.

Verify Portal access

Once the pods are running, verify that you can access the user interface.

1. **Open your browser** and navigate to the domain defined in `portal_domain_name` (e.g., `https://portal.my-pgai.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 defined in `values.yaml` (under `staticPasswords`).

Success: You should see the HM 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.

** *Phase 6: Exploring your post-installation options* →**

4.8.6 Exploring your post-installation options

Overview

Prerequisites

- *Phase 1: Planning your architecture* (Completed)
- *Phase 2: Gathering your system requirements* (Completed)
- *Phase 3: Deploying your Kubernetes cluster* (Running and validated)
- *Preparing your environment* (Completed and validated)
- *Install 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.

Unlock AI capabilities: Complete GenAI Builder setup

-

Guide: Gen AI Builder

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 **Beacon 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

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 `Using Hybrid Manager <https://www.enterprisedb.com/docs/edb-postgres-ai/latest/hybrid-manager/using_hybrid_m
→

4.8.7 Configuring Hybrid Manager

Syncing images to a local private registry

Related installation phase: *Preparing your environment*

Prerequisites

-

**`*edbctl* CLI tool** <https://www.enterprisedb.com/docs/edb-postgres-ai/latest/hybrid-manager/using_hybrid_manager/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.

Configure sync variables

Required Information:

- **Release Version:** The tag of Hybrid Manager (HM) you intend to install (example:`v1.3.0`).
- **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>
```

1. Define EDB credentials (Source)

```
export CS_EDB_TOKEN=<your-edb-repos-token>
export EDB_SOURCE_REGISTRY=pgai-platform
```

1. 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.

Sync platform images

Execute the following command to sync the core platform images (examples: Portal, Beacon, Transporter).

```
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}"
```

Sync Operator image

The HM Bootstrap Operator is packaged separately. You must run a second sync command to mirror it.

```
edbctl operator 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}"
```

Update configuration

Once the sync is complete, you must configure the Helm chart to use your private registry.

Open your `values.yaml` file and update the `containerRegistryURL` to point to your internal URI.

Next steps

With your images synced and your `values.yaml` updated, you are ready to proceed with the installation.

Return to :ref:`Preparing your environment <Preparing your environment>` or :ref:`Install Hybrid Manager <Install Hybrid Manager>` .

Configuring image discovery

Related installation phase: *Preparing your environment*

Prerequisites

- *Syncing images to a local private 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 **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 Helm Chart

To enable this feature, you must update your `values.yaml` to tell the Beacon where to look.

```
beaconAgent:
  provisioning:
    imageDiscovery: true
    imagesetDiscoveryContainerRegistryURL: "<your-private-registry-uri>"
    # Set to true only for self-signed/internal CA scenarios
    imagesetDiscoveryAllowInsecureRegistry: false
```

imageDiscovery

This boolean option enables or disables the image discovery feature. It defaults to `false` .

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*

imagesetDiscoveryAllowInsecureRegistry

If this option enabled, it allows establishing a TLS connection without certificate validation.. It defaults to `false` .

imagesetDiscoveryAuthenticationType

The supported authentication type by registry provider is as follows:

Registry provider	Supported	Recommended
EDB Repo 2.0	*token*	*token*
Azure Container Registry (ACR)	*token*, *basic*	*token*

continues on next page

Table 32 – continued from previous page

Registry provider	Supported	Recommended
Amazon Elastic Container Registry (ECR)	*eks_managed_identity*	*eks_managed_identity*
Google Artifact Registry (GAR)	*token*, *basic*	*token*

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 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.

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.

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",
```

(continues on next page)

(continued from previous page)

```
            "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}"
```

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}" \
```

(continues on next page)

(continued from previous page)

```
    --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"
```

Configure Kubernetes Secret

The final step is ensuring the Helm chart knows which Kubernetes Secret contains the credentials (if not using Cloud Identity).

In your `values.yaml` , confirm the `image_discovery_secret_name` .

By default, this is set to `edb-cred` .

```
parameters:
  upm-beacon:
    image_discovery_secret_name: "edb-cred"
```

Verification:

After applying these changes via `helm upgrade` , check the logs of the **Beacon Agent** pod. You should see messages indicating that it is scanning the registry and discovering image tags.

Implementing TLS certificates

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:
```

(continues on next page)

(continued from previous page)

```
  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
```

Edit the Helm chart

Edit your Helm values.yaml file by adding the input variable `portal_certificate_secret` under `parameters.global` , setting its value to the name of the secret you have just created.

For example:

```
parameters:
  global:
    portal_certificate_secret: "my-portal-certificate"
```

Update the Helm chart

Run the Helm `upgrade` command to apply the changes on HM:

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    --f my-values.yaml
```

Warning

Make sure you always provide the set of input variables you have used to install the Helm Chart. If in a subsequent execution of the Helm upgrade command you don't specify one of the custom inputs, its value is going to be set to its default.

For more information on how the secret can be formatted, consult Istio documentation .

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 you can pass its name to HM by updating your `values.yaml` and run the Helm `upgrade` again.

The section that you need to edit in your `values.yaml` is:

```
parameters:
  global:
    portal_certificate_issuer_kind: "ClusterIssuer" # Valid values are Issuer and␣
↪ClusterIssuer
    portal_certificate_issuer_name: "my-issuer" # Your Issuer name
```

Finally, run the Helm upgrade command to apply the changes on HM:

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    --f my-values.yaml
```

If you want to take advantage of this feature, make sure you don't set `parameters.global.portal_certificate_secret`.

Warning

Make sure you always provide the set of input variables you have used to install the Helm Chart. If in a subsequent execution of the Helm upgrade command you don't specify one of the custom inputs, its value is going to be set to its default.

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.

You can run an helm upgrade to set the new CA, either by passing it via your values.yaml:

```
parameters:
  global:
    ca_secret_name: my-custom-ca
Or using the --set CLI argument:
```

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    ...
    --set parameters.global.ca_secret_name="my-custom-ca" \
    ...
```

Enabling GenAI Builder

GenAI Builder is EDB's integration of Griptape , a tool that allows you to build AI agents.

To enable the GenAI Builder launchpad application in your Hybrid Manager instance, you must create a secret for the application and configure dedicated object storage.

Creating a secret for GenAI Builder

For EKS installations using the `eks-install-secrets.sh` script, you can skip key and secret creation and go to *Configuring DataLake object storage for GenAI builder* , *Configuring CORS* , and *Backing up the Fernet key* . Other EKS setups and operating systems require manual secret creation with the Fernet key.

1. Create a Fernet key and store it in a variable:

```
FERNET_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64)
```

> **Note**
>
> Fernet is a cryptographic library used by Python. It provides symmetric encryption/decryption and is required to store secret data. .. Note:: 2. Create a namespace for the integration:

```
kubectl create namespace upm-griptape
```

3. Create a secret that references the new Fernet key and is stored in the dedicated namespace:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
   name: fernet-secret
   namespace: upm-griptape
```

(continues on next page)

(continued from previous page)

```
stringData:
   FERNET_KEY: ${FERNET_KEY}
EOF
```

Configuring DataLake object storage for GenAI builder

1. Create 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-griptape-datalake-bucket-name> -region     <your-region>
```

```
gsutil mb -l <your-region> gs://<your-griptape-datalake-bucket-name>
```

Use your provider's management console or CLI to create a bucket with a unique name for your Griptape DataLake bucket.

2. 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.

Configuring CORS

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:

```
cat cors-config.json
[
  {
    "origin": ["https://${PORTAL_DOMAIN_NAME}"],
    "method": ["GET", "PUT", "POST", "DELETE", "HEAD"],
    "responseHeader": ["*"],
    "maxAgeSeconds": 3600
  }
]
```

Where `https://${PORTAL_DOMAIN_NAME}` is the domain configured for your Hybrid Manager.

5. Apply the CORS configuration to the previously created bucket:

```
gsutil cors set cors-config.json gs://<bucket name>
```

Use your provider's management console or CLI to configure cross-origin resource sharing (CORS) with Hybrid Manager.

After you've configured the secret and created a dedicated bucket, continue with Hybrid Manager installation .

Backing up the Fernet key

The Hybrid Manager administrator must keep the Fernet key safe and back it up.

Fetch the key:

```
kubectl get secret -n upm-griptape fernet-secret -o yaml
```

Store the key safely.

Enabling data catalogs

The Hybrid Manager (HM) data catalog requires additional configuration to manage Lakehouse data storage. This involves creating a Kubernetes secret with a confounding key that the Lakekeeper service can use to store encrypted data.

Creating a secret with the key

For EKS installations using the `eks-install-secrets.sh` script, you don't need to create a key and secret. In that case, skip this step and go to *Backing up the confounding key* .

Other EKS setups and operating systems require manual secret creation with the confounding key.

1. Create a confounding key and store it in a variable:

```
PG_CONFOUNDING_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64)
```

Note

- A confounding key is a randomized string that's at least 32 bytes long. - Create a confounding key for each Hybrid Manager deployment. .. Note:

2. Create a namespace for the service:

```
kubectl create namespace upm-lakekeeper
```

3. Create a secret that references the new confounding key and is stored in the dedicated namespace:

```
kubectl apply -f - <<EOF
  apiVersion: v1
  kind: Secret
  metadata:
    name: pg-confounding-key
    namespace: upm-lakekeeper
  stringData:
    PG_CONFOUNDING_KEY: ${PG_CONFOUNDING_KEY}
EOF
```

After you've configured the secret, continue with Hybrid Manager installation .

Backing up the confounding key

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:

```
kubectl get secrets -n upm-lakekeeper pg-confounding-key -o yaml
```

Store the key safely.

Customizing Migration Portal secrets for secure internal communication

Migration Portal uses several secrets for internal component communication. While these secrets are hard-coded in the Hybrid Manager release for a quick, functioning installation, for production environments we strongly recommend overriding them with custom secrets.

Using custom secrets ensures the secrets aren't easily discoverable, reducing risk of breaches.

Understanding Migration Portal secrets

For each secret, **Description** explains why you need it. **Helm value** shows where in the configuration file to reference it. **Parameters** shows on how to set non-default values.

See *Overriding the default secrets* 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.

Helm value: `parameters.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 Helm value.
- `stringData.username` : User that connects to the database. Also enter this user in the `parameters.edb-migration-portal.db_owner` *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.

Helm value: `parameters.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 Helm value.
- `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.

Helm value: `parameters.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 Helm value.
- `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.

Helm value: `parameters.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.

Helm value: `parameters.edb-migration-copilot.metrics_auth_secrets`

```
apiVersion: v1
kind: Secret
metadata:
  name: <copilot-metrics-secret-name>
  namespace: edb-migration-copilot
```

(continues on next page)

(continued from previous page)

```
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.

Overriding the default secrets

First, create custom secrets for internal component communication. Then, depending on the installation method you're using, override the values by referencing the custom secrets in either the `values.yaml` config file or the `HybridControlPlane` custom resource.

This example shows how to create custom secrets and configure Hybrid Manager to use them by working directly in the terminal. 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 *Migration Portal database account* :

```
kubectl create secret generic custom-db-secrets \
  --namespace=edb-migration-portal \
  --from-literal=username="${PORTAL_DB_USERNAME}" \
```

(continues on next page)

(continued from previous page)

```
  --from-literal=password="${PORTAL_DB_PASSWORD}" \
  --type=kubernetes.io/basic-auth
```

1. Create the secret that determines the *Migration Portal database superuser account* :

```
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 *Migration Copilot and Migration Portal database account* :

```
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 and Migration Copilot* :

```
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 that come with the Hybrid Manager installation, reference the new secrets in either the `values.yaml` or `HybridControlPlane` custom resource (depending on the installation method you used):

```
parameters:
  edb-migration-copilot:
    db_secrets: custom-ragchew-db-secrets
    metrics_auth_secrets: custom-edb-migration-copilot-metrics-auth
  edb-migration-portal:
    copilot_secrets: custom-edb-migration-copilot-auth
    db_owner: ${PORTAL_DB_USERNAME}
    db_secrets: custom-db-secrets
    db_superuser_secrets: custom-db-superuser-secrets
```

Continue with the installation process. Once you perform the bootstrapping process by either invoking the `helm upgrade` command to apply the `values.yaml` config file or apply the `HybridControlPlane` , the installation will take care of using the secret overrides.

If you're overriding the secrets after installation, ensure you bootstrap again 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.

Configuring multiple data centers for Hybrid Manager

Overview

Why run Hybrid Manager (HM) across multiple data centers?

Multi-DC gives you high availability and disaster recovery for Postgres workloads and the Hybrid Manager (HM) control plane (CP):

- Survive a site loss (DR): Keep a warm Secondary site ready. If the Primary DC is unavailable, promote replicas in the Secondary and restore service.
- Minimize downtime (HA): Perform maintenance or migrations on one site while workloads continue on the other.
- Protect data (RPO): Continuous replication to a second DC reduces potential data loss compared to single-site backups only.
- Reduce blast radius: Faults, misconfigurations, or noisy neighbors in one DC 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 DCs.

RTO/RPO at a glance

- **RTO (time to restore service)**: Typically minutes, driven by your promotion/cutover runbook and DNS/LB changes.

RPO (data loss window):

- Async replication (common across DCs): very low, but not zero (best-effort seconds).
- Sync replication (latency-sensitive): can approach zero data loss, but adds cross-DC latency and requires robust low-latency links.

What this guide helps you do

- Connect two HM clusters (Primary ↔ Secondary) on the same provider/on-prem family.
- Align object storage (identical edb-object-storage secret) so backups/artifacts are usable in both DCs.
- Enable SPIRE federation (trust domains via 8444/TCP) so platform identities work cross-site.
- Wire the Agent (Beacon) so the Primary can register the Secondary as a managed location and provision there (9445/TCP).
- (Optional) Federate telemetry (Thanos/Loki) for cross-site metrics and logs.

- Prepare a Postgres topology with a primary in one DC and replicas in the other; perform manual failover by promoting replicas.

Current limitations

- Two sites (Primary and Secondary).
- Manual failover: Promote replicas in the Secondary if the Primary is down.
- Same cloud/on-prem family: Cross-CSP multi-DC is not supported.

Architecture at a glance

- Control plane: Two HM clusters, federated via SPIRE; Primary "manages" the Secondary as a Location through Beacon.
- Data nodes: Postgres primary in DC-A, replicas in DC-B (async by default).
- Storage: Shared/consistent object store config across sites for backups/artifacts.
- Telemetry (optional): Thanos/Loki federation to view metrics/logs across sites.

Who is this for?

This is for teams that need higher resilience than a single DC can provide, and are comfortable running a manual, well-rehearsed failover playbook with clearly defined RTO/RPO targets.

Prerequisites

Architecture prereqs

- Two HM clusters available: Primary and Secondary (Kubernetes contexts configured).
- Each cluster has a unique SPIRE trust domain.
- Network connectivity:
  - 8444/TCP open between clusters (SPIRE bundle endpoint).
  - 9445/TCP from Secondary → Primary (Beacon gRPC).
  - Tooling: kubectl, jq (and yq if editing YAML locally).
  - Same provider/on-prem family (no cross-cloud).
- Tools: - `jq` , `yq` , AWS command line
- Scripts: - There are a number of scripts referred to use throughout this procedure. Please contact EDB Professional Services or EDB Support Center to get access to the scripts.

Discover and set necessary environment variables

1. Tell `kubectl` which clusters are Primary vs. Secondary

Set the kube contexts you'll use for all subsequent discovery commands:

```
export KUBE_CONFIG_PRIMARY_CONTEXT="<primary-kube-ctx-or-arn>"
export KUBE_CONFIG_SECONDARY_CONTEXT="<secondary-kube-ctx-or-arn>"
```

1. Discover the Primary portal FQDN (used by Beacon/telemetry)

Query the Primary cluster for the Beacon gateway host, then export it as `PRIMARY_PORTAL_URL` :

```
export PRIMARY_PORTAL_URL="$(
kubectl --context "$KUBE_CONFIG_PRIMARY_CONTEXT" \
    -n upm-beacon get gw beacon-server -o json \
| jq -r .spec.servers[1].hosts[0]
)"
```

1. (Optional) Discover the Secondary portal FQDN

If you'll also federate telemetry, capture the Secondary portal host as `SECONDARY_PORTAL_URL` and set the environment variable accordingly:

```
export SECONDARY_PORTAL_URL="$(
kubectl --context "$KUBE_CONFIG_SECONDARY_CONTEXT" \
    -n upm-beacon get gw beacon-server -o json \
| jq -r .spec.servers[1].hosts[0]
)"
```

1. Derive the Beacon gRPC endpoint for the Primary

Beacon listens on `:9445` on the Primary portal host.

```
export BEACON_SERVER_ENDPOINT_PRIMARY="${PRIMARY_PORTAL_URL}:9445"
```

1. Discover the SPIRE trust domain for the Primary

Read the SPIRE server config from the Primary and export `TRUST_DOMAIN_PRIMARY` .

```
export TRUST_DOMAIN_PRIMARY="$(
kubectl --context "$KUBE_CONFIG_PRIMARY_CONTEXT" \
    -n spire-system get cm spire-server \
    -o jsonpath="{[data][server\.conf]}" \
| jq -r .server.trust_domain
)"
```

1. Discover the SPIRE trust domain for the Secondary

Do the same for the Secondary and export `TRUST_DOMAIN_SECONDARY` .

```
export TRUST_DOMAIN_SECONDARY="$(
kubectl --context "$KUBE_CONFIG_SECONDARY_CONTEXT" \
    -n spire-system get cm spire-server \
    -o jsonpath="{[data][server\.conf]}" \
| jq -r .server.trust_domain
)"
```

1. Choose a label for the managed Secondary location

Pick any unique label (it will show up as `managed-<label>` on the Primary).

```
export SECONDARY_LOCATION_NAME="secondary"
```

1. (Optional for EKS/IRSA helpers) Set the EKS identifiers

Only do this if you'll use the helper to update S3 trust policy and copy the object-store secret. This step requires `PRIMARY_EKS` , `SECONDARY_EKS` , and `AWS_PROFILE` .

```
export PRIMARY_EKS="<region>:<primary-eks-name>"
export SECONDARY_EKS="<region>:<secondary-eks-name>"
export AWS_PROFILE="<aws-profile>"
```

1. Sanity check before proceeding

Verify the key variables are set (will error if any are missing):

```
: "${KUBE_CONFIG_PRIMARY_CONTEXT:?}"; : "${KUBE_CONFIG_SECONDARY_CONTEXT:?}"
: "${PRIMARY_PORTAL_URL:?}"; : "${BEACON_SERVER_ENDPOINT_PRIMARY:?}"
: "${TRUST_DOMAIN_PRIMARY:?}"; : "${TRUST_DOMAIN_SECONDARY:?}"
: "${SECONDARY_LOCATION_NAME:?}"
```

Here are some optional checks if you're doing telemetry or EKS helper

```
: "${SECONDARY_PORTAL_URL:?Set if doing telemetry federation}" || true
: "${PRIMARY_EKS:?Set if using EKS helper}" || true
: "${SECONDARY_EKS:?Set if using EKS helper}" || true
: "${AWS_PROFILE:?Set if using EKS helper}" || true
```

Note

If you open a new shell or run a new CI step later, re-run these exports (or save them to a file and source it).

Object storage across locations

HM uses an object store for backups, artifacts, WAL, and internal bundles. **In multi-DC, both clusters must use the same object store configuration**.

Key requirement

Each cluster must have an identical Kubernetes secret named `edb-object-storage` in the default namespace.

Note

Create/sync *edb-object-storage* before installing HM at any secondary location.

You create the initial secret when setting up object storage during installation for the Primary. Replicate it now to the Secondary:

```
#  Clean slate on Secondary

kubectl delete secret \
- -context=$KUBE_CONFIG_SECONDARY_CONTEXT \
- n default edb-object-storage || true

#  Copy Primary → Secondary

kubectl get secret \
- -context=$KUBE_CONFIG_PRIMARY_CONTEXT \
- n default edb-object-storage -o yaml | \
```

(continues on next page)

(continued from previous page)

```
kubectl apply \
- -context=$KUBE_CONFIG_SECONDARY_CONTEXT \
- n default -f -
```

EKS (IRSA) trust policy

If you use S3 + IAM Roles for Service Accounts (IRSA), the role must trust both clusters' OIDC providers.

1. Either update the trust policy manually to include both OIDC issuers, or
2. Retrieve the helper `object-storage-on-multi-dc.sh` helper script:
3. Use the helper script to copy the secret and append OIDC providers to the role's trust policy automatically:

```
./object-storage-on-multi-dc.sh \
-p <region>:<primary-eks-name> \
-s <region>:<secondary-eks-name>[,<region>:<another-eks>] \
-a <aws-profile>
```

The `object-storage-on-multi-dc.sh` helper script reads the IAM role ARN from the Primary's edb-object-storage secret, copies the identical secret to each Secondary, and appends the Secondary cluster's OIDC provider to the role trust policy if missing.

1. Validation checklist
   - Secrets identical (compare .data only).
   - Both clusters can list/write the bucket (quick Pod/Job test).
   - IRSA role trust includes both OIDC providers (if EKS).

Setup options

Choose Option A (one-shot) or Option B (step-by-step). You'll reach the same end state.

Option A — Quick start (master script)

The master script runs object storage sync, SPIRE federation, Beacon wiring, and optional telemetry in one pass.

1. Retrieve the `master-install.sh` script.
2. Run the script:

```
cd scripts/multi-dc
./master-install.sh
```

1. Useful flags:

```
./master-install.sh --dry-run \
--skip-object-store --skip-federation --skip-beacon --skip-telemetry
```

1. Verify:

- Verify that SPIRE federation is listed:

```
kubectl -n spire-system exec svc/spire-server -c spire-server -- \
/opt/spire/bin/spire-server federation list
```

- Verify the secondary location registered in Primary

```
kubectl get location
```

Option B — Manual setup (advanced/customizable)

Retrieve the necessary scripts

You need to retrieve these scripts in order to manually setup multi-DC:

- `update_objectstore_secrets.sh`
- `apply-federated-domain.sh`
- `configure-beacon-primary.sh`
- `configure-beacon-secondary.sh`
- `install.sh` (if setting up telemetry federation)

Run the scripts to setup multi-DC

Run these scripts in order:

- `update_objectstore_secrets.sh`
- `apply-federated-domain.sh`
- `configure-beacon-primary.sh`
- `configure-beacon-secondary.sh`
- `install.sh` (if setting up telemetry federation)

1. Object storage sync:

```
./update_objectstore_secrets.sh
# (EKS only, if IRSA)
./eks-object-storage-on-multi-dc.sh -p $PRIMARY_EKS -s $SECONDARY_EKS -a $AWS_PROFILE
```

1. SPIRE federation:

```
./apply-federated-domain.sh $KUBE_CONFIG_PRIMARY_CONTEXT $KUBE_CONFIG_SECONDARY_
↪CONTEXT
```

1. Validate:

```
kubectl -n spire-system exec svc/spire-server -c spire-server -- \
/opt/spire/bin/spire-server federation list
```

1. Beacon wiring:

```
./configure-beacon-primary.sh   $TRUST_DOMAIN_SECONDARY
./configure-beacon-secondary.sh $BEACON_SERVER_ENDPOINT_PRIMARY $TRUST_DOMAIN_PRIMARY
↪$SECONDARY_LOCATION_NAME
```

1. Validate:

```
kubectl get location
```

1. (Optional) Telemetry federation

```
cd thanos && ./install.sh -l secondary -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL
./install.sh -l primary  -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL

cd ../fluent-bit && ./install.sh -l primary -p $PRIMARY_PORTAL_URL -s $SECONDARY_
↪PORTAL_URL
./install.sh -l secondary -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL
```

SPIRE federation details (what/why/how)

SPIRE federation lets each SPIRE server trust the peer trust domain and continuously refresh its bundle (requires 8444/TCP). It can be configured via CRDs or spire-server federation CLI.

Typical flow (CRD-based):

- Generate ClusterFederatedTrustDomain manifest from each cluster (helper script).
- Cross-apply them (A → B, B → A).
- Validate with spire-server federation list.

Post-federation: Any workload identity that needs to cross sites must have a `ClusterSPIFFEID` with `federatesWith: "<peer-trust-domain>"`.

Beacon configuration (Primary & Secondary)

Beacon enables the Primary HM to register the Secondary as a managed "location" and provision there.

Primary: allow Secondary trust domain

1. Export current values and edit:

```
kubectl get configmap -n edbpgai-bootstrap -l app=edbpgai-bootstrap -o yaml \
| yq .items[0].data["values.yaml"] > /tmp/primary-boot-values.yaml
```

1. Edit /tmp/primary-boot-values.yaml:

```
beaconServer:
additionalTrustDomains:
    - "<secondary-location-trust-domain>"
```

1. Retrieve the Agent (beacon) install script, `install-dev.sh`.
2. Reinstall Beacon:

```
./install-dev.sh -f <provider> -a install -c upm-beacon -v <beacon-version> -p /tmp/
↪primary-boot-values.yaml
```

Secondary: point agent to Primary Beacon

1. From the Secondary:

```
kubectl get configmap -n edbpgai-bootstrap -l app=edbpgai-bootstrap -o yaml \
| yq .items[0].data["values.yaml"] > /tmp/secondary-boot-values.yaml
```

1. Edit `/tmp/secondary-boot-values.yaml`:

```
parameters:
upm-beacon:
    beacon_location_id: "secondary"   # unique label
beaconAgent:
beaconServerAddress: "<primary-portal-fqdn>:9445"
beaconServerTrustDomain: "<primary-trust-domain>"
plaintext: false
tlsInsecure: false
inCluster: false
```

1. Retrieve the Agent (beacon) install script `install-dev.sh` .
2. Reinstall Beacon:

```
./install-dev.sh -f <provider> -a install -c upm-beacon -v <beacon-version> -p /tmp/
↪secondary-boot-values.yaml
```

Federate Agent (Beacon) SPIFFE IDs (required)

1. Retrieve the `federate-beacon-spiffe-ids.sh` script.
2. Run from Secondary (include Primary trust domain):

```
./federate-beacon-spiffe-ids.sh "$TRUST_DOMAIN_PRIMARY"
kubectl rollout restart -n upm-beacon deploy/upm-beacon-server
kubectl rollout restart -n upm-beacon deploy/upm-beacon-agent-k8s
```

1. Run from Primary (include Secondary trust domain):

```
./federate-beacon-spiffe-ids.sh "$TRUST_DOMAIN_SECONDARY"
kubectl rollout restart -n upm-beacon deploy/upm-beacon-server
kubectl rollout restart -n upm-beacon deploy/upm-beacon-agent-k8s
```

1. Validate registration (Primary):

```
kubectl get location
# Expect: managed-<SECONDARY_LOCATION_NAME> with recent LASTHEARTBEAT
```

Telemetry federation (optional)

If you need cross-site metrics/logs:

Thanos (metrics)

1. Retrieve the Thanos install script, `install.sh` .
2. Secondary:

```
./install.sh -l secondary -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL
```

1. Primary:.

```
./install.sh -l primary -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL
```

1. Validate: port-forward `thanos-query` and hit `/api/v1/stores` for entries containing `thanos-query-federated`.

Fluent Bit/Loki (logs)

1. Retrieve the Loki install script, `install.sh`.
2. Primary:

```
./install.sh -l primary -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL
```

1. Secondary:

```
./install.sh -l secondary -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL
```

1. Validate: port-forward `loki-read` and query for `{app="fluent-forward"}`.

> **Note**
>
> Use distinct prefixes for metrics and logs in values:

- "`global.metrics_storage_prefix`"
- "`upm-loki.logs_storage_prefix`"

Goal: Wire Beacon so the Primary HM can treat the Secondary as a managed location and you can provision PG primary/replicas across DCs.

Prereqs recap

- Two HM Kubernetes clusters (same provider/on-prem family), with SPIRE federation already configured.
- Shared object store (edb-object-storage secret) present and identical in both clusters.
- Network open: 8444/TCP (SPIRE bundle endpoint), 9445/TCP (Beacon gRPC), plus your Postgres replication ports.
- Tools installed: `jq`, `yq`.

Discover important values (run against each cluster where noted):

1. Retrieve Primary portal host used by Beacon (append :9445)

```
kubectl get gw beacon-server -n upm-beacon -o json \
| jq -r .spec.servers[1].hosts[0]
```

1. Retrieve trust domain (run in each cluster)

```
kubectl get cm spire-server -n spire-system -o jsonpath="{[data][server\.conf]}" \
| jq -r .server.trust_domain
```

Export environment variables you'll reuse:

```
export KUBE_CONFIG_PRIMARY_CONTEXT="<primary-kube-ctx-or-arn>"
export KUBE_CONFIG_SECONDARY_CONTEXT="<secondary-kube-ctx-or-arn>"

export TRUST_DOMAIN_PRIMARY="<primary-trust-domain>"
export TRUST_DOMAIN_SECONDARY="<secondary-trust-domain>"

export BEACON_SERVER_ENDPOINT_PRIMARY="<primary-portal-fqdn>:9445"
export SECONDARY_LOCATION_NAME="secondary"     # any unique label
```

Multi-DC Beacon Helm install (Primary)

1. Extract the current `values.yaml` for Primary:

```
kubectl get configmap -n edbpgai-bootstrap -l app=edbpgai-bootstrap -o yaml \
| yq eval .items.0.data["values.yaml"] > /tmp/primary-boot-values.yaml
```

1. Edit `/tmp/primary-boot-values.yaml` to allow the Secondary trust domain:

```
beaconServer:
# ...
additionalTrustDomains:
    - "<secondary-location-trust-domain>"
```

1. Install/Reinstall Beacon on Primary (example):

```
./install-dev.sh -f <provider> -a install -c upm-beacon -v <upm-beacon-version> -p /
↪tmp/primary-boot-values.yaml
```

Multi-DC Beacon Helm install (Secondary)

1. Extract the current `values.yaml for` Secondary:

```
kubectl get configmap -n edbpgai-bootstrap -l app=edbpgai-bootstrap -o yaml \
| yq eval .items.0.data["values.yaml"] > /tmp/secondary-boot-values.yaml
```

1. Edit `/tmp/secondary-boot-values.yaml` to register this cluster as a managed location and to point the agent back to Primary:

```
parameters:
upm-beacon:
    # name can be anything, just not the same as Primary
    beacon_location_id: "secondary"

beaconAgent:
beaconServerAddress: "<primary-portal-fqdn>:9445"
beaconServerTrustDomain: "<primary-trust-domain>"
plaintext: false
tlsInsecure: false
inCluster: false
```

1. Install/Reinstall Beacon on Secondary:

```
./install-dev.sh -f <provider> -a install -c upm-beacon -v <upm-beacon-version> -p /
↪tmp/secondary-boot-values.yaml
```

Beacon ClusterSPIFFEID federation (both directions)

Each Beacon SPIFFE ID that crosses DCs must include the peer trust domain in `federatesWith`.

Use your helper script on both clusters.

1. From Secondary (add Primary trust domain):

```
./federate-beacon-spiffe-ids.sh "$TRUST_DOMAIN_PRIMARY"

kubectl rollout restart -n upm-beacon deploy/upm-beacon-server
kubectl rollout restart -n upm-beacon deploy/upm-beacon-agent-k8s
```

1. From Primary (add Secondary trust domain):

```
./federate-beacon-spiffe-ids.sh "$TRUST_DOMAIN_SECONDARY"

kubectl rollout restart -n upm-beacon deploy/upm-beacon-server
kubectl rollout restart -n upm-beacon deploy/upm-beacon-agent-k8s
```

The script finds Beacon ClusterSPIFFEID objects, creates a federatesWith list if missing, and appends the peer trust domain if not present.)

Validate wiring

1. On Primary: should list the managed Secondary location

```
kubectl get location
```

1. Validate SPIRE federation present on each cluster

```
kubectl -n spire-system exec svc/spire-server -c spire-server -- \
/opt/spire/bin/spire-server federation list
```

1. Expected:

- `kubectl get location` shows `managed-<SECONDARY_LOCATION_NAME>` with recent `LASTHEARTBEAT`.
- `federation list` shows 1 relationship (the peer trust domain) with `bundle endpoint profile: https_spiffe` and the peer's `:8444` URL.

Create the cross-DC Postgres topology

At this point HM can provision into the Secondary location. You still choose and create the actual DB topology.

Typical flow

1. From Primary HM, create the Postgres Primary in the Primary DC.
2. From Primary HM, create replica cluster(s) in the Secondary DC (select the managed Secondary location).
3. Confirm replication mode (sync/async) and monitor replication lag meets your SLOs.
4. Ensure backups are writing to the shared object store from both DCs, and test a restore.

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)

1. Validate federation relationships

```
kubectl -n spire-system exec svc/spire-server -c spire-server -- \
/opt/spire/bin/spire-server federation list
```

1. Validate that the Secondary location is registered (Primary)

```
kubectl get location
```

1. Validate provisioning to Secondary

- From Primary HM, deploy a small test workload to the 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 identical.

Manual failover runbook

Manual failover procedure from Primary to Secondary

1. Quiesce writes to Primary (maintenance mode / LB cutover).
2. Promote replicas in Secondary to Primary (per your HM workflow / scripts).
3. Redirect clients (DNS/LB) to Secondary.
4. Observe: confirm writes succeed; replication role updated.
5. When original Primary returns: re-seed it as a replica of the new Primary; optionally plan a later cutback.

Operator tips

- 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 A — SPIRE federation via CLI (optional)

You can manage federation with spire-server federation (create, list, update, delete, show, refresh) instead of CRDs. Use this if you prefer direct server control or for debugging.

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.

Configuring your own identity provider

For production deployments, we strongly recommend configuring an external IdP like LDAP, SAML, or another supported Dex connector . Doing so ensures robust and secure user management. Here are some examples:

- *Configuring LDAP*
- *Configuring SAML*

For testing or demo environments, you can simplify operations by

creating additional native users .

Configuring LDAP

Configure LDAP to work with Dex authentication to enable end users to log in to the Hybrid Manager (HM) console using LDAP server credentials. This example uses Okta for LDAP configuration.

Creating an Okta user for LDAP integration

Log in to Okta and create a new user with the read-only administrator role. Dex will use this user to communicate with the Okta LDAP interface.

Okta provides an LDAP interface for client integration. For more information, see the Okta LDAP Interface documentation .

Configuring Dex to connect to Okta via LDAP connector

Add an entry for the Okta LDAP configuration in which you specify the username and password (BindDN and BindPW) for the created read-only administrator user. Add the entry either in your `values.yaml` file or `HybridControlPlane` custom resource, depending on the installation method you use.

```
portal:
  authentication:
    idpConnectors:
      - type: ldap
        name: okta
        id: oktaLdap
        config:
          host: <ldap_server>:<ldap_server_port>
          insecureNoSSL: false
          insecureSkipVerify: false
          bindDN: uid=<okta_admin@example.com>, dc=<org_subdomain>, dc=<domain>,␣
↪dc=com
          bindPW: <okta_admin_password>
          usernamePrompt: Email Address
          userSearch:
            baseDN: ou=users, dc=<org_subdomain>, dc=<domain>, dc=com
            filter: "(objectclass=inetOrgPerson)"
            username: mail
            idAttr: DN
            emailAttr: mail
            nameAttr: cn
          groupSearch:
            baseDN: ou=groups, dc=<org_subdomain>, dc=<domain>, dc=com
            filter: "(objectclass=groupOfUniqueNames)"
            userMatchers:
              - userAttr: memberOf
                groupAttr: uniqueMember
            nameAttr: cn
    staticPasswords:
      - [...]
```

Parameters

The table shows the needed identity provider settings in the `portal.authentication` configuration. It shows the parameters that require customization based on your environment. Any other parameters not mentioned here can be configured as shown in the provided example.

Parameter	Description
idpConnectors	Enter an array of data per IdP connector you want to use for authentication. Refer to the official *Troubleshooting AI Factory on Hybrid Manager* for more guidance.
idpConnectors.config	Configuration for the connection to the LDAP server. Use the format *<ldap_server>:<ldap_server_port>*.
idpConnectors.config.bindDN	Distinguished Name of the read-only admin user you created in Okta that Dex will use to communicate with the LDAP server. Also specify where you want Dex to look for this user in the LDAP server.
idpConnectors.config.bindPW	Password for the read-only admin user you created in Okta that Dex will use to communicate with the LDAP server.
idpConnectors.config.usernamePrompt:	The text displayed above the username/email input field on the login screen.
idpConnectors.config.userSearch.baseDN:	Specify where you want Dex to look for users in your LDAP server. The administrator of your Okta LDAP instance will know the baseDN for users.

continues on next page

Table 33 – continued from previous page

Parameter	Description
idpConnectors.config.groupSearch.baseDN:	Specify where you want Dex to look for groups in your LDAP server. The administrator of your Okta LDAP instance will know the baseDN for users.
idpConnectors.config.groupSearch.baseDN.us Matchers:	Define how Dex determines if a user is a member of a group. Match attributes between user entries and group entries.

Applying Dex configuration

After changing the authentication settings, you need to reconcile the Dex configuration and restart the Dex services for your changes to take effect. See Applying Dex for instructions.

Assign new owner and disable native user

After assigning a new organization owner in the HM console, disable the native user. See Disabling a native user .

Configuring SAML

Configure SAML to work with Dex authentication to enable end users to log in to the Hybrid Manager (HM) console using SAML server credentials. This example uses Okta for SAML configuration.

Creating a SAML application

1. Log in to Okta and create a new app integration.
2. Select **SAML 2.0**.
3. Assign an app name, specify other general settings, and select **Next**.
4. Enter the Dex callback URL in both the **Single sign-on URL** (ACS URL) and **Audience URI** fields. The format is `https://<hybrid_manager_URL>/auth/callback` .
5. Map the **Attribute Statements** for username and email. For example:

Name	Name format	Value
name	Unspecified	"user.firstName+" "+user.lastName"
email	Unspecified	user. email

1. After the application is created, assign the users or groups you want to have access to HM to the created app integration. You can do this on the **Assignments** tab.

Configuring Dex to connect to Okta via SAML connector

Add an entry for the Okta SAML configuration where you specify the Okta and Dex endpoints for communication. Add the entry either in your `values.yaml` file or `HybridControlPlane` custom resource, depending on the installation method you use.

```
portal:
  authentication:
    idpConnectors:
      - type: saml
```

(continues on next page)

(continued from previous page)

```
      name: Okta
      id: okta
      config:
        ssoURL: <https://saml.example.com/sso>
        redirectURI: <https://<hybrid_manager_URL>/auth/callback>
        usernameAttr: name
        emailAttr: email
        groupsAttr: groups
        caData: <dNekV4TURSYUZ3MHpOREV3T01Rc3dDUVlEVlFRR1CRU=>
  staticPasswords:
    - [...]
```

Parameters

The table shows the essential identity provider settings in the `portal.authentication` configuration. It lists only the parameters you must customize for your environment. Any other parameters not detailed here can be configured as shown in the provided example.

Parameter	Description
idpConnectors	Enter an array of data per IdP connector you want to use for authentication. Refer to the official *Troubleshooting AI Factory on Hybrid Manager* for more guidance.
idpConnectors.config.ssoURL	The Single Sign-On URL is the endpoint where Dex will send authentication requests. You can obtain it from your Okta Sign On configuration.
idpConnectors.config.redirectURI	ACS URL of Hybrid Manager, which is the endpoint where the Okta SAML instance will send replies to. Same as the Dex callback URL you provided when creating the SAML application.
idpConnectors.config.usernameAttr	Tells Dex which attribute of the reply contains the user's username. Value must match the attribute names you specified when creating the SAML application.
idpConnectors.config.email	Tells Dex which attribute of the reply contains the user's email. Value must match the attribute names you specified when creating the SAML application.
idpConnectors.config.groupsAttr	Tells Dex which attribute of the reply contains information about the groups a user's belongs to.
idpConnectors.config.caData	Certificate from your Okta application's SAML configuration. It allows HM to authenticate replies from the SAML server. You can obtain it from your Okta Sign On configuration, SAML setup options.

Applying Dex configuration

After changing the authentication settings, you need to reconcile the Dex configuration and restart the Dex services for your changes to take effect. See Applying Dex .

Assigning new owner and disabling native user

After assigning a new organization owner in the HM console, disable the native user. See Disabling a native user .

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.

First, ensure the Key Management System provider of your choice has been enabled in the `values.yaml` :

- *Adding KMS support*

Then, configure your Key Management System provider:

- *HashiCorp Vault*
- *AWS KMS*
- *Creating the Google Cloud KMS key*

Adding KMS support

When you deploy the Hybrid Manager (HM), the bootstrap kit configuration (`values.yaml`) 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, 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 helm chart with the altered `values.yaml` to populate the system with the new configuration. See the installation guides for infrastructure-specific instructions on how to apply the helm chart.

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, gcp_kms, hashicorp_vault.
#     Default values: passphrase, hashicorp_vault.
transparentDataEncryptionMethods:
  - passphrase
  - hashicorp_vault
  - aws_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, gcp_kms, hashicorp_vault.
#     Default values: passphrase, hashicorp_vault.
transparentDataEncryptionMethods:
  - 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 Google Cloud Key Management Service as the KMS for TDE*

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:
```

(continues on next page)

(continued from previous page)

```
    - preference:
        matchExpressions:
        - key: edbaiplatform.io/control-plane
          operator: Exists
      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* .

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 `values.yaml` 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* .

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 `values.yaml` 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* .

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.

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:

- *Hybrid Manager 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 HM:

- *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.
- *edbctl backup* — 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.
- *Model Image Migration* — 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.

5.3 For Kubernetes administrators

For those administering the Kubernetes environment of HM:

- *Managing users in Hybrid Manager* — configure native users.
- *Operational Monitoring* — set up agents, explore built-in dashboards, and extend observability with Prometheus and Grafana.
- Hybrid Manager upgrades - upgrade the HM platform to the next minor/major/innovation release version.
- *HA/DR planning and best practices* — configure high availability and disaster recovery for production clusters.
- *Troubleshooting AI Factory on 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.

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.

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.

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.

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* .

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.

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.

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.

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.

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* .

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.

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* .

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* .

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.

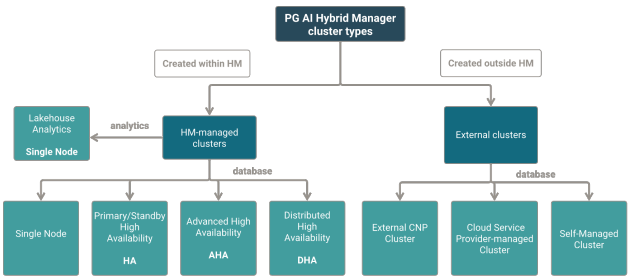


Fig. 1: Cluster Types in Hybrid Manager

Currently, you can:

- Create *HM-managed clusters*

or,

- Connect *Requirements for external clusters* .

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.

Lakehouse Analytics - Single Node: You can also create *Lakehouse Analytics* clusters with the Hybrid Manager. PG AI Lakehouse Analytics clusters provide an environment for querying and analyzing data stored in open lakehouse formats (like Delta Lake) within object storage. Leveraging Hybrid Manager's unified control plane, users can easily deploy and manage these specialized clusters optimized for fast analytical queries.

Hybrid Manager simplifies the deployment and operation of these analytics engines, accelerating data-driven decision-making.

External clusters

You can connect clusters that were provisioned externally.

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 Agent and perform migrations with the *Installing EDB Data Migration Service agent on Linux* .

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**.

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 *Installing EDB Data Migration Service agent on Linux* .

Examples: Postgres on AWS RDS.

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.

Other terms

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.

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

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.

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
- Create, view, and manage Lakehouse clusters
- View and manage catalogs
- View the status of model clusters
- Manage your asset library
- Monitor HM-managed and external clusters

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.

Launching third-party services

You can use the HM *Launchpad* to launch related third-party services without leaving HM.

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 *Monitoring resources in your estate* .

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

- ***Lakehouses***
-

Catalogs

- **Model clusters**

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 **Lakehouse** tab of the estate, you can:

- Monitor your

analytics clusters , which are HM clusters specialized for analytics workloads.

In the **Catalog** tab of the estate, you can:

- View all Apache Iceberg® tables attached to or created with HM.

In the **Model clusters** tab of the estate, you can:

- View and access clusters that serve AI models.

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 New Lakehouse Cluster** — *Add a new Lakehouse cluster* to 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**.

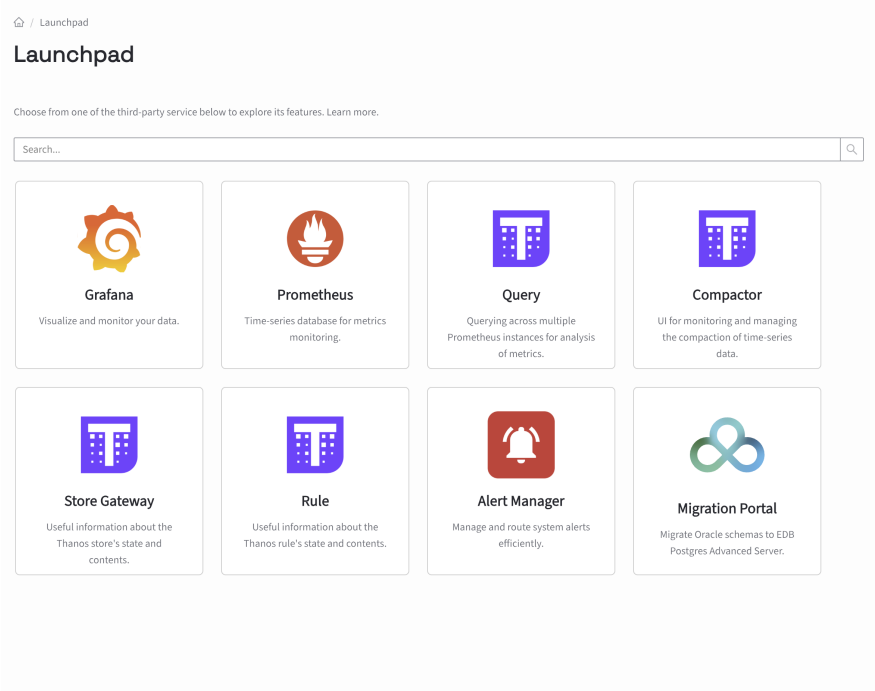


Fig. 2: Launchpad

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
- 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**

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.
- 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:

- Number of of Alerts — Total number of alerts for the cluster.
- Number of 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 gibibytes.

> **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
- Health Score Statistic
- Include Deleted Clusters
- Location
- Postgres Version
- Project Name
- Resource Type, including: - Appliance — Clusters managed by Hybrid Manager (HM). - Self Managed Postgres — Clusters that are self-managed. - Self Managed Postgres Databases — Clusters with databases that are self-managed.
- 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.

Lakehouse clusters

To view your Lakehouse clusters, from the **Estate** page, select **Lakehouse**. When viewing your Lakehouse clusters, you can apply filters to narrow your results. You can see the number of Lakehouse clusters you have access to and the metadata and metrics for the cluster.

Adding a Lakehouse cluster

To add a Lakehouse cluster to your estate, from the **Quick Actions** menu, hover over **Add New Lakehouse Cluster** and select a project to associate with the new cluster. The **Create Analytics Cluster** page opens, which offers two ways to add a cluster: build from a template or create a custom build.

- To build from a template, if no templates are available, select **Create New Template** to create one. To use an existing template, select **Use Template**.
- To create a custom build, select **Start from Scratch**.

The template creation workflow and custom build workflow for analytic clusters are the same as the regular cluster creation workflow. See Creating clusters .

To learn more about how to use the Analytics/Lakehouse feature of Hybrid Manager (HM), see Analytics/Lakehouse .

Lakehouse clusters table

The clusters table is a list of all of the available Lakehouse clusters across your projects. It 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
- 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 field at the top. Use the search field to filter columns to show column names using text matching. From the list of column names, select the ones you want to appear in the table.

These are the other columns you can display:

- Number of of Alerts : Total number of alerts for the cluster.
- Number of 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.
- OS Version : Operating system version of a self-managed cluster.
- Provider : ID of the CSP provider.
- Region : Locationid for an appliance cluster.

- TDS : Total database size in gibibytes.

> **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 to choose the specific type or values related to that property.
  - Engine
  - Include Deleted Clusters
  - Location
  - Management
  - Postgres Version
  - Project Name
  - Status

Monitoring data staleness

The data in the table is refreshed every 30 seconds. A message under the table shows the last time the table was updated with new data.

5.3.3 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.

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.

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.

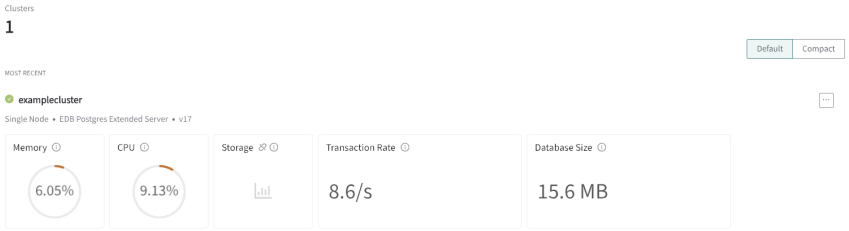


Fig. 3: Project Overview

In a project, you can:

- Create, view, and manage clusters .
- *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 .

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 *image_discovery_credentials_secret_name* .

Managing projects

You can rename and delete projects. See *Managing projects* .

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. 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 there are any clusters in the project, you must delete them before you can delete the project. The display shows any remaining clusters with a check box 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, you must 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 there are no clusters in the project, 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 completely 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 key management systems 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
4. Optionally, provide one of these authentication parameters:
   - For Basic Auth: username and password
   - Bearer Token
5. 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:
6. From your profile menu, select **Settings**.
7. Select the **Notifications** tab.
8. 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
9. 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

Note

- HM saves any changes you make to the alert configuration. - Not all the alerts are available for external HM clusters due to technical limitations.

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

5.3.4 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.

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)
- ***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 cluster to be deployed into. The default location is in the Kubernetes cluster hosting the EDB Postgres AI service. Where HM has been configured in a multi-HM environment, you can select the location for the cluster to be deployed into.

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* 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.

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.

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

enabled a key from a key management system (KMS) ,
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 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 cluster to be deployed into. The default location is in the Kubernetes cluster hosting the EDB Postgres AI service. Where Hybrid Manager (HM) has been configured in a multi-HM environment, you can select the location for the cluster to be deployed into.

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* 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.

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.
- 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 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.

1. In the upper-left corner, select **Projects** and navigate to your project.
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_xxxxxxxxxxxxx
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
```

(continues on next page)

(continued from previous page)

```
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"
        }
      },
      "image": {
        "url": "upmdev.azurecr.io/postgresql:17.6-2510160831",
        "digest":
↪"sha256:6bda61e6d75c3b9d6a886c547d789c46b88ed807ae28d8d06fa1510a0cc5afae"
      },
```

(continues on next page)

(continued from previous page)

```
      "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
```

(continues on next page)

(continued from previous page)

```
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. !!!

Create a file called `get-existing-cluster.sh` that includes the following:

```
# !/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

API documentation 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, including Lakehouse Analytics templates.

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.

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:

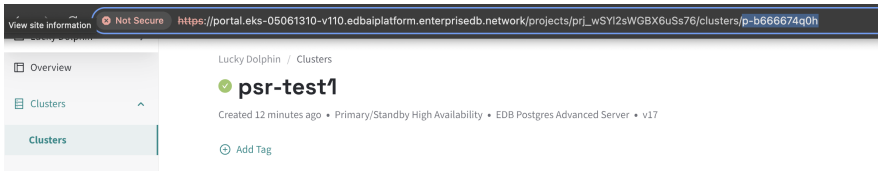


Fig. 4: Browse cluster

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.

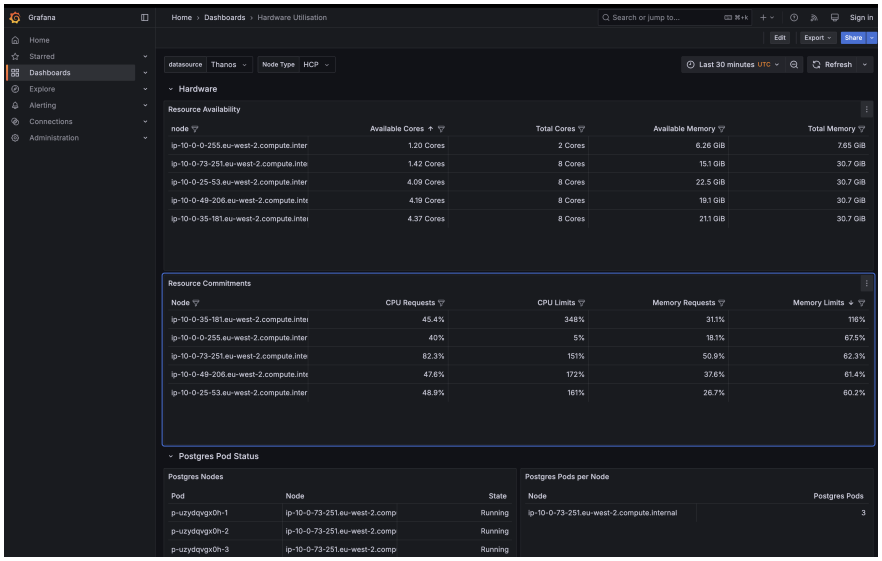


Fig. 5: Hardware Utilization dashboard

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.

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 *Configuring LDAP* .

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": {
       "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",
```

(continues on next page)

(continued from previous page)

```
    "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

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:

- *Architectural Overview* — A brief overview of the cluster.
- *Connectivity options* — Connection information for the cluster.

- *Operational Monitoring* — Monitoring information for the cluster.
- *Diagnostics (Query Diagnostics and Recommendations tabs)* — Query diagnostics information for the cluster.
- *HM backups and DR* — Backup information for the cluster.
- *Accessing metrics and logs* — Logs for the cluster.
- *Cluster Properties* — Properties of the cluster.

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 *Quick Connect* section.

If the cluster is a distributed high-availability cluster, the overview first shows the *Creating data groups* 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**.

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 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.

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 ***Monitoring*** 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 ***Logs*** tab.

Cluster connections

Use the **Cluster 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.

`.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.

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.

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**
>
> Only the first Indexes and Statistics are currently 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

These 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's 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

> **Note**
>
> - Fail (F) isn't currently used. It's reserved for unhealthy cluster states that require immediate action. - Unknown (U) means 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.

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.
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.

5.3.5 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* .

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.

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.

1. 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.

1. 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.

1. 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.

1. 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.

5.3.6 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.

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 *AI Factory Architecture on Hybrid Manager* .

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* .

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. 6: 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. 7: 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 EDB Data Migration Service agent , 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.

HM agent

The HM agent , packaged as `beacon-agent` , is a tool that provides database observability, monitoring, usage reporting, and more. In the context of migrations, the HM agent performs a migration assessment for each database you register with it. In the case of registered Oracle databases, it also performs schema extraction for subsequent handling with the *Migration Portal integration* .

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 a data migration with Hybrid Manager* , 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 *Finalizing a streaming data migration* .

Migration destination types

When *Creating a data migration with Hybrid Manager* , 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 in reader mode* 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 HM 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.

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 migration options

Choose the migration method that best suits your needs:

- Snapshot migration : 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 - 17	Y
EDB Postgres Advanced Server 12 - 17	Y
EDB Postgres Extended Server 12 - 17	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 your Oracle source for data migration* .

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 Postgres source databases .

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 - 17	Y
EDB Postgres Advanced Server 11 - 17	Y
EDB Postgres Extended Server 11 - 17	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 Postgres destination databases .

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*

Data migration limitations

Case sensitivity in object names

> **Tip**
>
> Resolved in an *Known issues, limitations, and notes* .

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.

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

limitation: # 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.

Postgres limitations

Unsupported domain type definitions in columns

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
- TIME
- TIMESTAMP
- INTERVAL
- UUID
- ENUM
- JSON
- XML
- POINT
- 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;
```

Schema ingestion known issues

Additional schemas ingested for EDB Postgres Advanced Server sources

Tip

Resolved in an *Known issues, limitations, and notes* .

When viewing the list of schemas for a source EDB Postgres Advanced Server database that you registered with the HM 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.

Other known issues

Data migrations supported without a Load Balancer in RHOS environments only

Tip

Resolved in an *Known issues, limitations, and notes* .

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: This issue is currently targeted for implementation to ensure Kafka can be accessed even when standard Kubernetes Load Balancers are unavailable. To perform data migrations in RKE environments, you currently must have Load Balancers configured.

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 *Error page appears when editing knowledge base credentials* 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 *Error page appears when editing knowledge base credentials* 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 *Error page appears when editing knowledge base credentials* 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 *Error page appears when editing knowledge base credentials* 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.

Migrating

Migration Guides

Migrate your AWS RDS/Aurora PostgreSQL database to an HM-managed database

These are all the steps you need to perform a migration from an AWS RDS for PostgreSQL or Aurora PostgreSQL database to a database managed by Hybrid Manager (HM).

Prepare your source database

Preparing your source database for data migration involves creating a dedicated user for the migration and altering the database configuration to enable the replication features.

1. *Preparing your AWS RDS/Aurora PostgreSQL source database* .

Assess the database (optional)

Install and configure the HM agent with your database to obtain observability and monitoring data for your Postgres database. This optional step also provides high-level migration metrics, like schema count and names, table count, and object count.

1. *Configure the HM agent to assess your database* .

Create a destination database

Create a cluster in the HM instance to host the destination database.

1. *Creating your HM-managed destination database* .

Migrate your schema

Handle incompatibilities and transfer your schemas.

1. *Understanding schema handling* .
2. *Migrating Postgres schemas with Migration Toolkit* .

Migrate your data

Execute the data transfer and complete the final cleanup steps.

1. *Creating Hybrid Manager project credentials for the migration* .
2. *Configuring and running the EDB DMS agent in reader mode* .
3. *Verifying your source appears in the HM console* .
4. *Creating a data migration with Hybrid Manager* .
5. *Finalizing a streaming data migration* (for streaming migrations only).
6. *Applying constraints after the migration is finalized* .

Prepare Source

Preparing your AWS RDS/Aurora PostgreSQL source database

Before migrating data from an AWS RDS for PostgreSQL or Aurora PostgreSQL database using the EDB Data Migration Service, you must prepare the source database to allow the EDB DMS Agent to connect to the source and perform replication.

Enable logical replication

To perform logical replication, the EDB DMS Agent in reader mode requires:

- `rds.logical_replication` set to `1`
- `wal_level` set to `logical`
- Optionally, `max_wal_size` set to the recommended setting of 8 GB or higher for

improved migration performance

Because AWS RDS databases are managed by a cloud service provider, you can't change their default configurations with psql. Instead, create a DB cluster parameter group with replication enabled and associate it with your RDS instance.

1. Create a DB cluster parameter group using either the AWS Management Console, AWS CLI, or RDS API. Ensure the DB cluster parameter group family matches the Postgres version your RDS database is using.

See Parameter groups for Amazon RDS for additional configuration information.

2. Associate the created DB cluster parameter group with your source Amazon RDS database using either the AWS Management Console, AWS CLI, or RDS API.
3. Modify parameters in a DB cluster parameter group using either the AWS Management Console, AWS CLI, or RDS API.

- For this DB cluster parameter group, you must set `rds.logical_replication` to `1` . After you configure this, the database sets `wal_level` to `logical` .
- Ensure `max_wal_size` is configured for adequate WAL LSN lifetime.

> **Note**
>
> If your source database is an Amazon Aurora PostgreSQL instance, you won't be able to modify *max_wal_size* as it is managed by AWS. Only Amazon RDS PostgreSQL instances allow you to edit the *max_wal_size* parameter. !!!

Set the `max_wal_size` value large enough that production traffic generates mostly timed checkpoints rather than requested checkpoints based on WAL size.

The streaming migration process also requires changes to be available in the WAL until they can be streamed to durable message storage in the cloud infrastructure of EDB DMS. Setting `max_wal_size` too small can affect performance. It can also interfere with the migration process by allowing Postgres LSNs to be dropped from the WAL before they can be streamed. We recommend setting `max_wal_size` to at least 8 GB. Consult your DBA to determine the appropriate size.

4. To apply the changes, reboot your AWS RDS database .
5. Verify that the settings were set correctly by connecting to the database and checking whether `wal_level` is now set to `logical` and that `max_wal_size` is set as configured.

```
SHOW WAL_LEVEL;
SHOW MAX_WAL_SIZE;
```

The output is logical.

Create a role for CDC migration

1. To create a role for the migration, connect to the source database as postgres or a user with superuser access:

```
psql -h <PG_HOST> -p <PG_PORT> -U <PG_USERNAME> -d <PG_DB_NAME>
```

Where:

```
-   `<PG_DB_NAME>`  is the name of the Postgres database source to connect to.

-   `<PG_HOST>`  is the Postgres database host.

-   `<PG_PORT>`  is the Postgres database port.

-   `<PG_USERNAME>`  is an administrative user or superuser who can create and grant␣
↪roles, alter ownership of tables to migrate, and create a replication slot.
```

This command prompts you for the password associated with `<PG_USERNAME>` .

2. Create a role for migration with `LOGIN` :

```
CREATE ROLE <MIGRATION_ROLE> WITH LOGIN PASSWORD <MIGRATION_ROLE_PASSWORD>;
```

3. Grant the role `RDS_REPLICATION` abilities:

```
GRANT RDS_REPLICATION to <MIGRATION_ROLE>;
```

4. `<MIGRATION_ROLE>` must be the owner of the source tables you want to migrate so it can autocreate Postgres publications.

A way to do this is to create an additional role (here `<REPLICATION_GROUP>`) which will be the new table owner, and grant this role to both the original owner of the table and `<MIGRATION_ROLE>` .

```
CREATE ROLE <REPLICATION_GROUP>;
GRANT <REPLICATION_GROUP> TO <MIGRATION_ROLE>;
GRANT <REPLICATION_GROUP> TO <ORIGINAL_OWNER>;
ALTER TABLE <TABLE_NAME> OWNER TO <REPLICATION_GROUP>;
```

Where:

```
-   `<MIGRATION_ROLE>`  is the name of the Postgres role or user to use for migration␣
↪database access.

-   `<ORIGINAL_OWNER>`  is the original production owner of the table.

-   `<REPLICATION_GROUP>`  is the name of a role used to own the source tables to␣
↪migrate for publication autocreation.

-   `<TABLE_NAME>`  is the name of a table to migrate and is case sensitive.
```

5. Grant `USAGE` of the source schemas containing tables to be migrated to the migration role:

```
GRANT USAGE ON SCHEMA <DB_SCHEMA> TO <MIGRATION_ROLE>;
```

The new <MIGRATION_ROLE> has schema access but still needs SELECT access to the source tables of those schemas.

6. Grant SELECT on source tables to the migration role. You can grant access across a schema or for each table.

For all tables in a schema, use this command:

```
GRANT SELECT ON ALL TABLES IN SCHEMA <DB_SCHEMA> TO <MIGRATION_ROLE>;
```

For each table, use:

```
GRANT SELECT ON <TABLE_NAME> TO <MIGRATION_ROLE>
```

Where:

```
-   `<DB_SCHEMA>`  is the database schema name for the tables to migrate.

-   `<MIGRATION_ROLE>`  is the name of the Postgres role or user to use for migration␣
↪database access.

-   `<TABLE_NAME>`  is the name of a table to migrate and is case sensitive.
```

7. Grant CREATE privileges on the source database to the migration role. This is required by the pgoutput plugin so the user can create publications.

```
GRANT CREATE ON DATABASE <PG_DB_NAME> to <MIGRATION_ROLE>
```

Validate your configuration

After you configure the source database, validate your Postgres configuration. For this, you need the configuration validation script packaged with the DMS Agent.

Prerequisites

1. Ensure the machine where you're running the DMS Agent has Java/OpenJDK 17 or later installed.
2. Install the EDB DMS Agent .
3. Navigate to the EDB DMS Agent folder in /opt/cdcagent/ .

Run the config validation script

This example shows how to run the validation script.

1. 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:

```
PG_USERNAME=postgres PG_PASSWORD=password PG_HOST=localhost PG_PORT=5432 DB_
↪NAME=postgres DBZ_USERNAME=debezium DBZ_PASSWORD=dbz ./postgresConfigValidation.sh "
↪${arr[@]}"
```

You get a response similar to the following:

```
 *** [Transporter] - Get PostgreSQL Major Version
PostgreSQL major version: 16

 *** [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 at least 8 GB.

 *** [Transporter] - Validate checkpoints
checkpoints_timed: 424
checkpoints_req: 1
[Pass] Timed checkpoints are more frequent than requested checkpoints.

 *** [Transporter] - Check debezium user role
[Pass] User debezium is present
[Pass] User debezium has replication permission

 *** [Transporter] - Check SELECT privilege on the tables to be migrated
[Pass] User debezium has select privilege on all tables to be migrated.

 *** [Transporter] - Check CREATE privilege on database
[Pass] Role debezium has CREATE privilege on database postgres.
```

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 you must resolve before the DMS can execute the data migration. *[Suggestion]* checks allow migration but can negatively affect performance if left unaddressed.

Assess Database

Configuring the HM agent to assess your database

You can install and configure the Hybrid Manager (HM) agent to obtain observability and monitoring data for your Postgres database. This optional step provides high-level metrics like schema count and names, table count, and object count, which can aid in migration planning.

> **Note**
>
> Configuring the HM agent isn't required to perform the migration, and it currently doesn't ingest the actual schema objects (DDL).

To register a database with the HM agent, see:

- Monitoring self-managed deployments
-

Monitoring Cloud Service Provider managed Postgres deployments

Create Destination

Creating your HM-managed destination database

Create new database

If you don’t have a destination database cluster on Hybrid Manager (HM) yet, create one that HM can use as destination for migrations:

1. In the HM console, select the project where you want to create the database cluster.
2. In your project, select **Create New** and **Database Cluster** to create a database cluster to serve as the destination for the migration.
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` .

Use existing database

Alternatively, you can use an existing HM-managed database as a destination for your migration. Before you use an existing database cluster as a destination for the migration, ensure the tables you migrate and the load generated on the destination don’t interfere with existing workloads.

If you want to use an existing HM-managed database as a destination, continue with the next step.

Migrate Schema

Understanding schema handling

Before you use EDB Data Migration Service (EDB DMS) to configure a data migration that uses the `cdcagent` , you must prepare and import your schema to the destination database.

Importing the schema to the destination database before the migration takes place is important as EDB DMS migrates only data. The destination database must have the schemas in place for the migration to populate them with data.

Some of your schema’s constraints must be included before the data migration takes place. Others must be applied after the data migration is completed. This approach ensures you can migrate without performance degradation.

Schema integrity and performance considerations

The presence of destination database constraints, triggers, and WAL logging can impact the data migration performance. When this approach is possible, we recommend a two-step import of schema constraints.

Primary key and unique constraints

EDB DMS leverages primary key/unique constraints to provide an exactly-once delivery when migrating data to the destination database. Therefore, you must include primary key/unique constraints in the schema import that you perform before the data migration begins. Exclude other types of constraints from the schema import.

For rows in tables that don't have primary key/unique constraints you can achieve only at-least-once delivery. You can perform deduplication during the data migration verification.

> **Note**
>
> Not null constraints don't represent a significant performance impact for destination servers and can also be included in the schema import.

Foreign key, check, and exclusion constraints

EDB DMS can apply change events in parallel against destination database clusters. However, migrating some constraint types can negatively affect migration performance. These type of constraints lead to unnecessary CPU and memory utilization in the context of an in-flight data migration from a consistent and referentially integral source database.

Therefore, we recommend applying foreign key, check, and exclusion constraints on the destination database only after the migration is complete.

Migrating Postgres schemas with Migration Toolkit

Schema handling for Postgres databases

You can use several different methods to prepare your schema, exclude unsupported constraints, and import it to the destination database. The following are guidelines for how to export and import your schemas for Postgres migrations using Migration Toolkit.

Other valid routes for migrating DDL to import Postgres schemas include:

- Manually creating the schemas in the destination database
- Performing a pg_dump to obtain the schemas and

pg_restore to
restore them

- Using

pgAdmin or psql

Export

For data migrations from Postgres, we recommend using EDB Migration Toolkit to manage the schema. MTK's offline migration capability provides an easy way to extract a database's schema and separate constraints. Here are some guidelines:

- Use the `-offlineMigration` option, so Migration Toolkit can generate SQL scripts that you can apply on destination without having to configure connectivity.

- Use the `-schemaOnly <schema_scope>` option, so Migration Toolkit can generate scripts that focus only on the export of schemas.
- After you run Migration Toolkit, exclude (comment out) the following constraints from the generated SQL script before applying it on the destination database:
- Foreign key constraints
- Check constraints
- Exclusion constraints
- After you run Migration Toolkit, and if you plan to map objects to different names in the destination, modify the schema SQL script to reflect the new names. After the SQL script is generated, edit the object names directly in the script (DDL) before applying it to the destination. Also make sure that the structure and data types of the renamed object in the script exactly match the source. If you manually update object names in the DDL scripts, also update the names in other objects that reference the updated objects. For example, if you rename a table, also update any views or constraints that reference the original table name.

Later, during data migration, select the newly assigned object names. See Mapping schema, tables and columns during a migration for more information.

Import

After you prepare the DDL and exclude foreign key, check, and exclusion constraints, connect to the destination database and import the SQL-formatted DDL file.

To start the import, execute the generated offline migration script .

> **Note**
>
> You will have to apply the excluded foreign key, check, and exclusion constraints on the destination database later in the migration process, after the migration is performed successfully. Therefore, keep track of the excluded constraints, so you can apply them when the time comes.

Migrate Data

Creating Hybrid Manager project credentials for the migration

Credentials are required so you can perform data migrations. Before you can add a source or destination for a migration to Hybrid Manager (HM), you must create migration credentials. You can then use them to allow the EDB DMS agent to connect to HM.

You need one set of credentials per HM project . For example, if you're using a single HM project for all migrations, you need only one set of credentials.

1. On the HM console, select the project you're using for your migration activities.
2. In the project, select **Migrate > Credentials**.

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.

3. Select **Create Migration Credential**, enter a name for your credentials, and select **Create Migration Credential**.
4. After the credentials are generated, select **Download Credential**.

> **Note**
>
> This is the only time you can download the credentials. If you don't download them now and you need your credentials, you'll have to revoke this set and create new ones.

5. Unzip the downloaded zip file and copy the generated folder to the host where the EDB DMS agent is installed.

The decompressed credentials folder contains these files:

```
- `int.crt` - `int.truststore.p12` - `client.keystore.p12` - `client-key.
↪pem` - `client-cert.pem`
```

Configuring and running the EDB DMS agent in reader mode

Configure and run the DMS agent in reader mode to perform a data migration.

Prerequisites

1. Ensure the machine where you're running the DMS agent has Java/OpenJDK 17 or later installed.
2. Install the EDB DMS agent .
3. Open the EDB DMS agent located in `/opt/cdcagent/run-cdcagent.sh` with write permissions.

Configuring the EDB DMS agent for migrations from AWS RDS for PostgreSQL or Aurora PostgreSQL

Set the variables in the script according to your environment and uncomment the edited lines. See parameters for more guidance. The following shows the unedited script:

```
# !/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>
```

(continues on next page)

(continued from previous page)

```
#  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 destination database

export RUN_MODE=reader

#  This is the DMS backend service used by the DMS Agent

#  Consult your system administrators

export RW_SERVICE_HOST=https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter

##########################################
#  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

#  but the env structure looks like this

export DBCONFIG_DATABASES_0__PASSWORD=<password>

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent
```

(continues on next page)

(continued from previous page)

```
#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_0__RESOURCEID=1234567890abcdef

#  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=localhost
# export DBCONFIG_DATABASES_1__PORT=1521

#  The CATALOG is the database name

# export DBCONFIG_DATABASES_1__CATALOG=ORCLCDB/ORCLPDB1
# export DBCONFIG_DATABASES_1__USERNAME=oracle

#  The password env can be set without specifying it here

#  but the env structure looks like this

# export DBCONFIG_DATABASES_1__PASSWORD=password

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_1__RESOURCEID=1234567890abcdef

########################################
#  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
```

(continues on next page)

(continued from previous page)

```
#  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 the *DBCONFIG_DATABASES_0__HOSTNAME* value to point 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* .

Running the EDB DMS agent

Start the DMS agent:

```
cdcagent
```

Verifying your source appears in the HM console

- Sources* are databases you connect to Hybrid Manager (HM) using the EDB DMS agent in reader mode so you can use them for migrations.

To verify a source was recognized by the EDB DMS agent and is available for a migration:

1. In the HM console, select the project you're using for the current migration.
2. From the left navigation menu, select **Migrate > Sources**.

Each source's view shows the database name, database type (Postgres or Oracle), `cdcagent` ID, the current status, the source type, and more.

More information

For a description of the available data on this page of the HM console, see Sources .

Creating a data migration with Hybrid Manager

After you configured the EDB DMS agent (in reader mode) to establish connections to Hybrid Manager (HM), create a migration in the HM console. A migration establishes a sync between the source database and a destination database with the help of EDB DMS.

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 EDB DMS agent is listed in the **Source** menu.
6. Select the migration type: **Snapshot** or **Snapshot + Streaming** .
7. Under **Destination Type**, select an HM-managed destination database cluster and database for the migration. Then, select **Next**.
8. Select the tables and columns to migrate. Modify the table and column names if needed.
9. Select **Create Migration**.

The HM console displays a new migration.

If you chose the **Snapshot + Streaming** option, the EDB DMS agent is constantly streaming data when the migration displays the **Running** state. Changes to data are replicated from the source to the destination database as long as the migration is running.

> **Note**
>
> The EDB DMS agent in reader mode streams data changes. It doesn't stream changes in the DDL objects.

Finalizing a streaming data migration

If you selected the **Snapshot + Streaming** option when creating a migration, the EDB DMS agent continues to stream any data updates performed on the source database to the destination database until you mark the migration as completed (for example, to perform a cutover).

If you selected the **Snapshot** option, you can skip this page, as the DMS marks the migration as completed after it finishes copying the entire snapshot to the destination.

Stop the streaming process

This process applies only to **Snapshot + Streaming** migrations.

It's important that you carefully select the best time to stop the data migration stream. Ensure no new inserts are executed in the source and verify that the last known inserts were synchronized in the destination.

After checking that your data is present in the destination database, you can stop the stream by marking the migration as completed.

1. In the Hybrid Manager (HM) console, in your project, select **Migrate > Migrations**.
2. Select the checkmark next to your migration to mark the migration as completed.

Alternatively, select the migration from the list to open the Migration details page . Then select **Mark as Completed**.

This stops the streaming procedure.

Applying constraints after the migration is finalized

At the beginning of your data migration journey with EDB Data Migration Service (EDB DMS), you prepared and imported the schema of your source database. Now, connect to the destination database and reapply the constraints that were excluded from the schema and data migration.

Primary key and unique constraints

For primary key and unique constraints, you already created the tables and constraints in the destination Postgres database. This allowed EDB DMS to map them to the source objects and migrate data successfully. You don't need do to anything else.

The same applies to not null constraints if you included them in your schema import.

Foreign key, check, and exclusion constraints

You can now reapply any foreign key, check, or exclusion constraints you excluded during the schema preparation and import.

Ensuring data integrity

Rows in tables that don't have primary key or unique constraints were migrated with at-least-once delivery, Therefore, it's possible that these rows are duplicates.

Deduplication can be performed as part of the verification.

Verification

To verify that all data was migrated, compare the source and destination databases.

You can use LiveCompare .

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 a database cluster managed by Hybrid Manager (HM).

Prepare your source database

Preparing your source database for schema ingestion, migration assessment, and subsequent data migration involves creating a dedicated user for both the HM agent and DMS agent and altering the database configuration to enable replication features.

1. *Preparing your Oracle source database for schema ingestion* .
2. *Preparing your Oracle source for data migration* .

Ingest source schema and assess the database

Install and configure the HM agent with your Oracle database. This process performs a migration assessment to help you scope the effort, complexity, and time required for the migration. Also, the HM agent extracts schema DDL that you can use to perform schema migrations with the built-in Migration Portal.

1. Create a machine user for registration with the HM agent .
2. Install the HM agent .
3. *Configuring the HM agent to register your Oracle database* .

4. Run the HM agent .
5. *Running the HM agent as a service* .
6. *Verifying your Oracle database was assessed for migration* .

Create a destination database

Create a cluster in the HM instance to host the destination database.

1. *Creating your HM-managed destination database* .

Migrate your schema

Use the Migration Portal integration to handle incompatibilities and transfer your schemas.

1. *Understanding schema handling* .
2. *Migrating Postgres schemas with Migration Toolkit* .

Migrate your data

Execute the data transfer and complete the final cleanup steps.

1. *Creating Hybrid Manager project credentials for the migration* .
2. *Configuring and running the EDB DMS agent in reader mode* .
3. *Verifying your source appears in the HM console* .
4. *Creating a data migration with Hybrid Manager* .
5. *Finalizing a streaming data migration* (for streaming migrations only).
6. *Applying constraints after the migration is finalized* .

Prepare Source

Preparing your Oracle source for data migration

> **Note**
>
> This configuration is required for data migration, as it allows Hybrid Manager (HM) capabilities to replicate data.

To prepare your Oracle source for the migration, you need sysdba privileges.

Configure an Oracle source database to:

- *Step 1: Enable archivelog mode* .
- Enable supplemental logging for the database and table of interest .
- *Step 3: Prepare tablespaces for migration user* .
- Create migration user and grant necessary privileges to carry out the data migration .
- Grant SELECT on source tables .
- Optionally, for better performance, ensure adequate redo log space is available .
- Validate the configuration .

Prerequisites

To alter the database's configuration, you must connect to your Oracle source database with sysdba privileges. The following examples use SQL*Plus to establish this connection.

Step 1: Enable archivelog mode

For Debezium to capture all database changes in Oracle, the database must be configured in archivelog mode. This is essential because Debezium reads the database's redo logs to track changes.

In archivelog mode, Oracle saves a copy of each redo log file before it's overwritten. This ensures a continuous and complete history of every transaction, which Debezium needs to maintain a reliable stream of change data without any gaps.

Log in to Oracle container database (CDB) as sysdba. Or, if you're using a non-CDB architecture, log in to your non-CDB database as sysdba:

```
sqlplus <DBA_USER>/<PASSWD>@<HOST>:<PORT>/<DB_IDENTIFIER> as sysdba
```

Where:

- <DBA_USER> is the Oracle user with sysdba privileges.
- <PASSWD> is the password for the DBA_USER.
- <HOST> is the hostname of your Oracle database.
- <PORT> is the port of your Oracle database.
- <DB_IDENTIFIER> is, for CDB, the container database name (for example, CDB1), For non-CDB, this is either the service name (recommended) or SID (for example, ORCL).

Run the following SQL command to check whether archivelog mode is enabled:

```
archive log list;
```

The returned content indicates the database mode:

```
-- below output indicates archive log mode is enabled
Database log mode   Archive Mode -- (or "Archive Log Mode" in new versions of oracle)
-- or below output indicates archive log mode is not enabled
Database log mode   No Archive Mode
```

If you've already enabled archivelog mode, you can skip this step. If not, log out from your current SQL session, then run the following commands to enable it.

Connect to the SQL session:

```
ORACLE_SID=<DB_IDENTIFIER> sqlplus /nolog
```

Where:

- <DB_IDENTIFIER> is, for CDB, the container database name (for example, CDB1). For non-CDB, this is either the service name (recommended) or SID (for example, ORCL),

Run following SQL commands:

```
CONNECT <DBA_USER>/<PASSWORD> AS sysdba;
-- Set the location and size for Fast Recovery Area (FRA)
ALTER SYSTEM SET db_recovery_file_dest_size = <RECOVERY_FILE_DEST_SIZE>;
```

(continues on next page)

(continued from previous page)

```
ALTER SYSTEM SET db_recovery_file_dest = <RECOVERY_FILE_DEST> scope=spfile;
- - Restart the database in the mount phase to enable archiving
SHUTDOWN IMMEDIATE;
STARTUP MOUNT
- - Enable archive log mode and open the database for use
ALTER DATABASE archivelog;
ALTER DATABASE open;
- - Verify the change was successful
archive log list;
exit;
```

Where:

- `<DBA_USER>` is the Oracle user with sysdba privileges.
- `<PASSWD>` is the password for DBA_USER.
- `<RECOVERY_FILE_DEST>` is the directory path of your Oracle fast recovery area (FRA), where Oracle redo logs are stored.
- `<RECOVERY_FILE_DEST_SIZE>` is the amount of space allocated to FRA, for example, 100G for 100 gigabytes.

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.

Step 2: Enable supplemental logging for the database and tables of interest

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.

Log in to Oracle CDB as sysdba. Or, if you're using a non-CDB architecture, log in to your non-CDB database as sysdba:

```
sqlplus <DBA_USER>/<PASSWD>@<HOST>:<PORT>/<DB_IDENTIFIER> as sysdba
```

Where:

- <DBA_USER> is the Oracle user with sysdba privileges.
- <PASSWD> is the password for DBA_USER.
- <HOST> is the hostname of your Oracle database.
- <PORT> is the port of your Oracle database.
- <DB_IDENTIFIER> is, for CDB, the container database name (for example, CDB1). For non-CDB, this is either the service name (recommended) or SID (for example, ORCL).

Enable supplemental logging at the database level:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

If you're using a CDB architecture, switch to the PDB that you want to migrate. (If you're using a non-CDB architecture, you can skip this step.)

```
ALTER SESSION SET CONTAINER = <PDB_NAME>;
```

Where <PDB_NAME> is the name of your Oracle PDB you want to migrate.

Enable supplemental logging for all the tables that you want to migrate:

```
ALTER TABLE <SCHEMA_NAME>.<TABLE_NAME> ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

Where:

- <SCHEMA_NAME> is the schema to which your table belongs.
- <TABLE_NAME> is the name of a table you want to migrate.

Note

- Make sure supplemental logging is enabled on tables that you want to migrate. Without it, the synchronization doesn't work, as it's a requirement for Debezium.

```
- 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 <https://www.
↪enterprisedb.com/docs/edb-postgres-ai/latest/hybrid-manager/using_hybrid_manager/
↪migration/limitations/#oracle-xe-compatibility-restrictions>`_  .
```

Step 3: Prepare tablespaces for migration user

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.

For non-CDB architecture, only one tablespace is needed. This single tablespace combines both log mining and table operations since there's no container/PDB separation.

Log in to Oracle CDB as sysdba. Or, if you're using a non-CDB architecture, log in to your non-CDB database as sysdba:

```
sqlplus <DBA_USER>/<PASSWD>@<HOST>:<PORT>/<DB_IDENTIFIER> as sysdba
```

Where:

- <DBA_USER> is the Oracle user with sysdba privileges.
- <PASSWD> is the password for DBA_USER.
- <HOST> is the hostname of your Oracle database.
- <PORT> is the port of your Oracle database.
- <DB_IDENTIFIER> is, for CDB, the container database name (for example, CDB1). For non-CDB, this is either the service name (recommended) or SID (for example, ORCL).

Create a tablespace using the SQL command. For CDB architecture, this tablespace is for CDB, while for non-CDB architecture this is for a normal database:

```
CREATE TABLESPACE <TABLESPACE_NAME> DATAFILE <TABLESPACE_PATH>.dbf SIZE 25M REUSE␣
↪AUTOEXTEND ON MAXSIZE UNLIMITED;
```

Where:

- <TABLESPACE_NAME> is the name of the tablespace. You can use any valid name here.
- <TABLESPACE_PATH>.dbf is the file path of this tablespace. Always consult your DBA to determine the correct and appropriate file path for your environment.

Next, if you're using a CDB architecture, switch to the pluggable database (PDB) you want to migrate. (If you're using a non-CDB Oracle database, you can skip this step.)

```
ALTER SESSION SET CONTAINER = <PDB_NAME>;
```

Where <PDB_NAME> is the name of your Oracle's PDB you want to migrate. Create a tablespace for PDB:

```
CREATE TABLESPACE <TABLESPACE_NAME> DATAFILE <PDB_TABLESPACE_PATH>.dbf SIZE 25M REUSE␣
↪AUTOEXTEND ON MAXSIZE UNLIMITED;
  -- 25M is a minimal example size. Production systems typically require larger␣
↪initial allocations (e.g., 512M+).
  -- Consult your DBA for organization-specific sizing policies.
- - continue adding more datafiles if necessary
- - ALTER TABLESPACE <TABLESPACE_NAME> ADD DATAFILE <PDB_TABLESPACE_PATH2>.dbf SIZE␣
↪25M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

Where:

- <TABLESPACE_NAME> is the name of the PDB tablespace. For consistency, you can use the same name you used for the CDB tablespace.
- <PDB_TABLESPACE_PATH>.dbf is the file path for the tablespace. Always consult your DBA to determine the correct path for your environment.

Step 4: Create migration user and grant the privileges

DMS needs a dedicated migration user (a *common user* for CDBs, regular user for non-CDBs).

Log in to Oracle CDB as sysdba. Or, if you're using a non-CDB architecture, log in to your non-CDB database as sysdba:

```
sqlplus <DBA_USER>/<PASSWD>@<HOST>:<PORT>/<DB_IDENTIFIER> as sysdba
```

Where

- <DBA_USER> is the Oracle user with sysdba privileges.
- <PASSWD> is the password for DBA_USER.
- <HOST> is the hostname of your Oracle database.
- <PORT> is the port of your Oracle database.
- <DB_IDENTIFIER> is, for CDB, the container database name (for example, CDB1). For non-CDB, this is either the service name (recommended) or SID (for example, ORCL)

Now, create a dedicated Oracle user for the migration. The way you create it and the permissions it has depend on whether you are using a CDB or a non-CDB database.

> **Note**
>
> If you previously created a dedicated Oracle user for schema extraction and assessment with the HM Agent, you can reuse the same user here. In that case, ensure it's configured in the same way and has the following permissions.

CDB database

If you're using a CDB architecture, run the following SQLs one by one to create a migration user (which must be a common user) and grant necessary permissions:

```
CREATE USER <MIGRATION_USER> IDENTIFIED BY <MIGRATION_USER_PASSWORD>
    DEFAULT TABLESPACE <TABLESPACE_NAME>
    QUOTA UNLIMITED ON <TABLESPACE_NAME>
    CONTAINER=ALL;
GRANT CREATE SESSION TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SET CONTAINER TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$DATABASE to <MIGRATION_USER> CONTAINER=ALL;
GRANT FLASHBACK ANY TABLE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ANY TABLE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT_CATALOG_ROLE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT EXECUTE_CATALOG_ROLE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ANY TRANSACTION TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ANY DICTIONARY TO <MIGRATION_USER> CONTAINER=ALL;
GRANT LOGMINING TO <MIGRATION_USER> CONTAINER=ALL;
GRANT CREATE TABLE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT LOCK ANY TABLE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT CREATE SEQUENCE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT EXECUTE ON DBMS_LOGMNR TO <MIGRATION_USER> CONTAINER=ALL;
GRANT EXECUTE ON DBMS_LOGMNR_D TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$LOGMNR_LOGS TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$LOGMNR_CONTENTS TO <MIGRATION_USER> CONTAINER=ALL;
```

(continues on next page)

(continued from previous page)

```
GRANT SELECT ON V_$LOGFILE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$ARCHIVED_LOG TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$ARCHIVE_DEST_STATUS TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$TRANSACTION TO <MIGRATION_USER> CONTAINER=ALL;
```

Where:

- <MIGRATION_USER> is the name of the common user you're creating for DMS.
- <MIGRATION_USER_PASSWORD> is the password for the new migration user.
- <TABLESPACE_NAME> is the name of the CDB tablespace you created earlier. For simplicity, we recommend using the same name for both the CDB and PDB tablespaces.

If you used different names for your CDB and PDB tablespaces in the previous step, you must also log in to the PDB and run the following command to set the default tablespace for the user:

```
ALTER USER <MIGRATION_USER> DEFAULT TABLESPACE <PDB_TABLESPACE_NAME>
```

non-CDB database

If you're using a non-CDB database, use the following SQL command instead:

```
CREATE USER <MIGRATION_USER> IDENTIFIED BY <MIGRATION_USER_PASSWORD>
    DEFAULT TABLESPACE <TABLESPACE_NAME>
    QUOTA UNLIMITED ON <TABLESPACE_NAME>;
GRANT CREATE SESSION TO <MIGRATION_USER>;
GRANT SELECT ON V_$DATABASE to <MIGRATION_USER>;
GRANT FLASHBACK ANY TABLE TO <MIGRATION_USER>;
GRANT SELECT ANY TABLE TO <MIGRATION_USER>;
GRANT SELECT_CATALOG_ROLE TO <MIGRATION_USER>;
GRANT EXECUTE_CATALOG_ROLE TO <MIGRATION_USER>;
GRANT SELECT ANY TRANSACTION TO <MIGRATION_USER>;
GRANT SELECT ANY DICTIONARY TO <MIGRATION_USER>;
GRANT LOGMINING TO <MIGRATION_USER>;
GRANT CREATE TABLE TO <MIGRATION_USER>;
GRANT LOCK ANY TABLE TO <MIGRATION_USER>;
GRANT CREATE SEQUENCE TO <MIGRATION_USER>;
GRANT EXECUTE ON DBMS_LOGMNR TO <MIGRATION_USER>;
GRANT EXECUTE ON DBMS_LOGMNR_D TO <MIGRATION_USER>;
GRANT SELECT ON V_$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>;
```

Where: - <MIGRATION_USER> is the name of the user to create for CDC migration table access. - <MIGRATION_USER_PASSWORD> is the password for the migration user. - <TABLESPACE_NAME> is the tablespace for <MIGRATION_USER> .

Step 5: Grant SELECT on source tables

Log in to PDB as sysdba. Or, if you're using a non-CDB architecture, log in to your non-CDB database as sysdba:

```
sqlplus <DBA_USER>/<PASSWD>@<HOST>:<PORT>/<DB_IDENTIFIER> as sysdba
```

Where:

- <DBA_USER> is an Oracle user that can log in as sysdba.
- <PASSWD> is the password for <DBA_USER> .
- <HOST> is the Oracle database hostname.
- <PORT> is the Oracle database port.
- <DB_IDENTIFIER> is, for CDB, the pluggable database name (for example, PDB1). For non-CDB, this is either the service name (recommended) or SID (for example, ORCL).

For each table you want to migrate, you must grant SELECT permission to the migration user you created:

```
GRANT SELECT ON <SCHEMA_NAME>.<TABLE_NAME> TO <MIGRATION_USER>
```

Where:

- <SCHEMA_NAME> is the name of the schema that contains the table you want to migrate.
- <TABLE_NAME> is the name of an individual table to migrate and is case sensitive.
- <MIGRATION_USER> is the name of the user being granted SELECT access on the table for CDC migration purposes.

> **Note**
>
> For Oracle, the terms *schema* and *user* are interchangeable. To prevent migration errors, be sure to use the correct schema name.

(Optional) Step 6: Ensure adequate redo log space is available for better performance

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.

To check the state of your database's redo logs, log in to Oracle CDB as sysdba. Or, if you're using a non-CDB architecture, log in to your non-CDB database as sysdba:

```
sqlplus <DBA_USER>/<PASSWD>@<HOST>:<PORT>/<DB_IDENTIFIER> as sysdba
```

Where:

- <DBA_USER> is the Oracle user with sysdba privileges.
- <PASSWD> is the password for DBA_USER.
- <HOST> is the hostname of your Oracle database.
- <PORT> is the port of your Oracle database.

- <DB_IDENTIFIER> is, for CDB, the container database name (for example, CDB1). For non-CDB, this is either the service name (recommended) or SID (for example, ORCL).

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.

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.

Step 7: Validate configuration

After you configure the source database, validate your Oracle configuration. For this, you can use the configuration validation script packaged with the DMS Agent.

Validation prerequisites

1. Ensure the machine where you're running the DMS Agent has Java/OpenJDK 17 or later installed.
2. Install the EDB DMS Agent .
3. Navigate to the EDB DMS Agent folder in `/opt/cdcagent/` .

Run the config validation script

The EDB DMS Reader installation (packaged as `cdcreader`) comes with a helper script that validates the Oracle configuration and helps you identify any issues. After you configure the database, we recommend running the script to ensure all checks pass.

1. 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=5432␣
↪ORACLE_SID=postgres ODB_DBZ_USERNAME=debezium ODB_DBZ_PASSWORD=dbz LOG_MINER_TBS_
↪NAME=LOGMINER_TBS ./oracleConfigValidation.sh "${arr[@]}"
```

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.
```

(continues on next page)

(continued from previous page)

```
[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.
[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.

SSL configuration

Ensure you configure your source database server to accept SSL connections to allow the EDB DMS Reader to connect to it. You must create a server certificate and a server private key to enable this configuration. You can use a tool like OpenSSL to create the key.

Your database is ready for CDC migration.

More information

- See the

Debezium Oracle Connector documentation.

- See Oracle's Enabling the Fast Recovery Area .

Preparing your Oracle source database for schema ingestion

> **Note**
>
> This configuration isn't required for data migration. However, it provides a helpful migration assessment and facilitates schema conversion by ingesting schema.

Registering an Oracle database with the Hybrid Manager (HM) agent provides migration assessment and schema extraction capabilities.

To assess an Oracle database, the HM agent needs an Oracle user with access to the database and the necessary permissions to retrieve assessment data. These permissions are required for automated schema extraction and database migration assessment. The HM agent will still function if these roles aren't assigned but in reduced feature capacity.

Prerequisites

Install Oracle's SQL Plus utility on the auxiliary machine you're using to run the HM agent.

Create and configure an Oracle user

Create an Oracle user (a *common user* for CDBs, regular user for non-CDBs) for schema extraction and assessment with the HM agent. You can use this same Oracle user later for data migration with the DMS agent. Create the user following your organization's usual provisioning process, and ensure the user receives the recommended privileges.

Alternatively, you can use an existing user if their current permissions meet the following requirements and aren't overly broad.

To create the user, log in to Oracle CDB as sysdba. Or, if you're using a non-CDB architecture, log in to your non-CDB database as sysdba.

The Oracle user you create needs access to these resources that allow it to connect, read schema metadata, and perform the assessment:

```
V$VERSION
V$INSTANCE
V$LICENSE
sys.all_tables
sys.all_tab_privs
```

The user must have the following privileges:

- `CONNECT` for database connection.
- `CREATE TABLE` to create temporary working tables during the assessment process.
- `SELECT_CATALOG_ROLE` or `SELECT ANY DICTIONARY` to read database metadata and schema information.

> **Note**
>
> These permissions are the minimum required for schema extraction and assessment. Granting additional permissions may be necessary depending on your specific Oracle database configuration and security policies.

The `CREATE TABLE` privilege is used by the assessment script to create temporary tables during execution. No permanent changes are made to your database.

`SELECT_CATALOG_ROLE` provides read access to data dictionary views (recommended for least privilege). Alternatively, `SELECT ANY DICTIONARY` is a more comprehensive system privilege that ensures access to all dictionary objects. Either will work for the assessment.

Avoid logging in as `SYS AS SYSDBA` , `SYSTEM` , or any other user with the `DBA` role unless absolutely necessary, as this can pose security risks.

Assess Database

Configuring the HM agent to register your Oracle database

Configure the Hybrid Manager (HM) agent (deployed as `beacon-agent`) to establish a secure connection with HM and retrieve necessary schema and migration data from your Oracle source database.

Prerequisites

*Create a machine user and access key with the `estate ingester* role <https://www.enterprisedb.com/docs/edb-postgres-ai/latest/hybrid-manager/using_hybrid_manager/migration/migration_guides/oracle_to_hm_managed/agent_creating_machine_user/>`_ assigned to it.

Store the access key in an environment variable named `$BEACON_AGENT_ACCESS_KEY` so the HM agent can access it later.

Obtain HM connection parameters

The HM agent requires specific configuration values that are set during the HM installation. Only an administrator or installer can access these. 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. 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 will need this value later to configure the HM agent.
- **Root certificate path (if required)**:The HM agent requires a trusted connection to the *beacon_server* service in your HM instance. If the server running the HM 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 HM agent. You'll need to provide the directory path to this certificate later to configure the HM 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 HM agent uses to connect to your Oracle source database. Set the `databases.dsn` parameter using an environment variable `$DSN` or direct URL format.

A DSN string for connections to Oracle databases must use this format:

```
DSN="oracle://<username>:<password>@<host>:<port>/<oracle_db_name>"
```

Where:

- `<username>` and `<password>` correspond to the user you granted permissions when preparing the database.
- `<host>` is the hostname of the instance where your Oracle database is running.
- `<port>` ist the port number of the database you want to fetch data from.
- `<oracle_db_name>` is the catalog or name of the database you want to fetch data from. For example, to connect as user `oracle` authenticated with password `password` to a database `ORCLPDB1` on port `1521` (assuming the agent is running on the same instance as the Oracle database):

```
DSN="oracle://oracle:password@localhost:1521/ORCLPDB1"
```

```
DSN="oracle://oracle:pa%24%24word@localhost:1521/ORCLPDB1"
```

Prepare a configuration file

Prepare a configuration file for your agent.

1. Create an HM agent configuration directory in your home directory:

```
mkdir ${HOME}/.beacon
```

Note

If this location isn't convenient, you can also use either:

- ***/etc/beacon* - The directory from which you execute any *beacon-agent* command The HM 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 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 the HM console to generate a migration assessment and ingest all schemas for assessment and import.

In this template, `$BEACON_AGENT_ACCESS_KEY` and `$DSN` are the environment variables you configured previously. Replace all < > placeholders in the template. See Parameter reference for details.

```
---
agent:
  access_key: $BEACON_AGENT_ACCESS_KEY
  beacon_server: <beacon_server>:9443
  project_id: <project_id>
  providers:
    - "onprem"
  root_ca_path: ""
  schema_providers:
    - "onprem-schema"
provider:
  onprem:
    clusters:
      - resource_id: "<cluster_resource_id>"
        name: "<cluster_name>"
        nodes:
          - resource_id: "<node_resource_id>"
            dsn: $DSN
            schema:
              enabled: true
            poll_interval: 15s
            tags:
              - "<tag_names>"
```

1. Save the file.

After configuring the HM agent, you can run it or configure it to run as a service.

Recommendations for setting environment variables

Although you can set all parameters directly in the config file, 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 HM 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.

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 .

Running the HM 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 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. Also, your database will disappear from your HM estate 72 hours after the agent stops streaming.

Before setting the agent up as a service, install and configure it.

> **Note**
>
> Future versions of the agent package may set this up automatically.

This example configures the HM agent to run on an RHEL 9 machine. Operating system aids help you to 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 new `beacon-agent.service` file and replace the information in placeholders `< >` .

```
# TEMPLATE

[Unit]
Description=Agent

After=network.target

```

(continues on next page)

(continued from previous page)

```
[Service]
Type=simple

User=<user>

WorkingDirectory=</home/user/>
Environment="PATH=</path/to/Oracle/db/binary>:</Oracle/user/binaries>:</system/
↪binaries>" # e.g. /u01/app/oracle/product/19.0.0/dbhome_1/bin:/home/oracle/.local/
↪bin:/home/oracle/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
Environment="ORACLE_HOME=</path/to/Oracle/db/binary>" # e.g. /u01/app/oracle/product/
↪19.0.0/dbhome_1

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 which 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.
Environment=	*"PATH=</path/to/Oracle/db/binary>:</Oracle/user/binaries>:</system/binaries>"*	Set to all user and system directories that contain Oracle-related executables to ensure the Agent can use *sqlplus* for schema extraction during runtime.
Environment=	*"ORACLE_HOME=</path/to/Oracle/db/binary>"*	Set to the directory of the Oracle database executables to enable *sqlplus* to extract schemas.
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
```

Verifying your Oracle database was assessed for migration

To verify the Oracle database was recognized by the Hybrid Manager (HM) agent and is available for assessment, on the HM console, select **Estate** and then select **Migrations**.

If the database appears in the table, you have access to the following resources:

- A migration assessment, which provides a summarized view of database schema count, core count, migration complexity, and level of effort. It's available on the

Migrations tab of the **Estate** page.

- A migration database detailed view , which provides an expanded analysis of schemas, a global and more detailed assessment, as well as a **Migrations** hub that provides direct connections to other tools in the HM console that can help you perform migrations. These resources are available when you select a specific Oracle database from the **Migrations** tab of the **Estates** page.
- Automatic schema DDL extraction that you can leverage to perform schema migrations with the built-in Migration Portal.

Create Destination

Creating your HM-managed destination database

Create new database

If you don't have a destination database cluster on Hybrid Manager (HM) yet, create one that HM can use as destination for migrations:

1. In the HM console, select the project where you want to create the database cluster.
2. In your project, select **Create New** and **Database Cluster** to create a database cluster to serve as the destination for the migration.
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` .

Use existing database

Alternatively, you can use an existing HM-managed database as a destination for your migration. Before you use an existing database cluster as a destination for the migration, ensure the tables you migrate and the load generated on the destination don't interfere with existing workloads.

If you want to use an existing HM-managed database as a destination, continue with the next step.

Migrate Schema

Understanding schema handling

Before you use EDB Data Migration Service (EDB DMS) to configure a data migration that uses the `cdcagent` , you must prepare and import your schema to the destination database.

Importing the schema to the destination database before the migration takes place is important as EDB DMS migrates only data. The destination database must have the schemas in place for the migration to populate them with data.

Some of your schema's constraints must be included before the data migration takes place. Others must be applied after the data migration is completed. This approach ensures you can migrate without performance degradation.

Schema integrity and performance considerations

The presence of destination database constraints, triggers, and WAL logging can impact the data migration performance. When this approach is possible, we recommend a two-step import of schema constraints.

Primary key and unique constraints

EDB DMS leverages primary key/unique constraints to provide an exactly-once delivery when migrating data to the destination database. Therefore, you must include primary key/unique constraints in the schema import that you perform before the data migration begins. Exclude other types of constraints from the schema import.

For rows in tables that don't have primary key/unique constraints you can achieve only at-least-once delivery. You can perform deduplication during the data migration verification.

> **Note**
>
> Not null constraints don't represent a significant performance impact for destination servers and can also be included in the schema import.

Foreign key, check, and exclusion constraints

EDB DMS can apply change events in parallel against destination database clusters. However, migrating some constraint types can negatively affect migration performance. These type of constraints lead to unnecessary CPU and memory utilization in the context of an in-flight data migration from a consistent and referentially integral source database.

Therefore, we recommend applying foreign key, check, and exclusion constraints on the destination database only after the migration is complete.

Migrating Oracle schemas with Migration Portal

Schema handling for Oracle databases

You can use several different methods to prepare your schema, exclude unsupported constraints, and import it to the destination database. The following are guidelines for how to export and import your schemas for Oracle using Migration Portal.

Other valid routes for migrating DDL to import Postgres schemas include:

- Manually creating the schemas in the destination database
- Performing a pg_dump to obtain the schemas and

pg_restore to
restore them

- Using

pgAdmin or psql

Export

Use the collaborative project-scoped workspace of Migration Portal to convert and migrate schemas. Hybrid Manager (HM) provides an integration that lets you use the HM console to create a Migration Portal project. In that project, you can easily add source database schemas preloaded by the HM agent while you're performing a database migration.

Prerequisites

- You

registered an Oracle database with Hybrid Manager.

- Optional: The HM administrator

enabled and configured the AI Copilot with the AI/LLM of your organization . This configuration helps to resolve schema incompatibilities in Migration Portal.

Creating a Migration Portal project

You can create Migration Portal projects from Hybrid Manager.

1. In the **Estate** page, select the **Migrations** tab.
2. To open the database's detailed view, 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 MP Project** or 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 HM 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. Instead, use a different project to migrate other schemas to the same destination database.

1. Select **Create New Project**.

- A Migration Portal project is created in the background with the source schemas loaded 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.

The selected schemas were uploaded and assessed by the Migration Portal. Use the AI Copilot or QuickHelp to help you resolve incompatibilities.

Import

Standard schema import procedure

If you plan to keep schema, table, and column names unaltered, use the **Online migration** option to directly connect to the destination database and migrate schema.

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 disable **Constraints** from the list of objects underneath the database. Select **Next**.

> **Note**
>
> Ensure you exclude foreign key, check, and exclusion constraints from the SQL-formatted DDL before importing the schema to the destination database.

1. In the **Connect to your cluster** page, enter the host name of the database cluster you want to use as a destination.

> **Tip**
>
> If you're migrating to an HM-managed database cluster, you can find the read/write host and other connection data by selecting **Clusters**, the cluser name, and then **Connect**. !!!

1. Select **Next** and wait until the schemas are imported successfully.
2. Connect to the target database cluster and ensure the schemas were migrated.

> **Note**
>
> You must reapply the excluded foreign key, check, and exclusion constraints on the destination database later in the migration process, after the migration is performed successfully. Therefore, keep track of the excluded constraints. The documentation will guide you on when and how to reapply them.

Schema import procedure with object renaming (optional)

If you plan to map schemas, tables, or columns to different names in the destination, use the **Offline migration** option. This allows you to download a DDL file to edit.

1. After you resolve all compatibility issues, select **Migrate to**.
2. Select the **Offline migration** option.
3. Select the schemas you want to migrate, but disable **Constraints** from the list of objects underneath the database. Select **Next**.

> **Note**
>
> Ensure you exclude foreign key, check, and exclusion constraints from the SQL-formatted DDL before importing the schema to the destination database.

1. Select **Download SQL file**.
2. After the SQL script is generated, edit the object names directly in the script (DDL) before applying it to the destination. Ensure that the structure and data types of the renamed object in the script exactly match the source. See

Mapping schema, tables and columns during a migration for more information.

If you manually update object names in the DDL scripts, ensure that you also update the names in other objects that reference the updated objects. For example, if you rename a table, also update any views or constraints that reference the original table name.

1. Connect to the destination database, and apply the modified DDL file.

> **Note**
>
> You must reapply the excluded foreign key, check, and exclusion constraints on the destination database later in the migration process, after the migration is performed successfully. Therefore, keep track of the excluded constraints. The documentation will guide you on when and how to reapply them.

Migrate Data

Creating Hybrid Manager project credentials for the migration

Credentials are required so you can perform data migrations. Before you can add a source or destination for a migration to Hybrid Manager (HM), you must create migration credentials. You can then use them to allow the EDB DMS agent to connect to HM.

You need one set of credentials per HM project . For example, if you're using a single HM project for all migrations, you need only one set of credentials.

1. On the HM console, select the project you're using for your migration activities.
2. In the project, select **Migrate > Credentials**.

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.

3. Select **Create Migration Credential**, enter a name for your credentials, and select **Create Migration Credential**.
4. After the credentials are generated, select **Download Credential**.

> **Note**
>
> This is the only time you can download the credentials. If you don't download them now and you need your credentials, you'll have to revoke this set and create new ones.

5. Unzip the downloaded zip file and copy the generated folder to the host where the EDB DMS agent is installed.

The decompressed credentials folder contains these files:

```
- `int.crt` - `int.truststore.p12` - `client.keystore.p12` - `client-key.
↪pem` - `client-cert.pem`
```

Configuring and running the EDB DMS agent in reader mode

Configure and run the DMS agent in reader mode to perform a data migration.

Prerequisites

1. Ensure the machine where you're running the DMS agent has Java/OpenJDK 17 or later installed.
2. Install the EDB DMS agent .
3. Open the EDB DMS agent located in `/opt/cdcagent/run-cdcagent.sh` with write permissions.

Configuring the EDB DMS Agent for migrations from Oracle

Set the variables in the script according to your environment and uncomment the edited lines. See parameters for more guidance. The following shows the unedited script:

```
# !/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

#  Consult your system administrators

export RW_SERVICE_HOST=https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter

########################################
#  DMS Agent TLS Configuration           #

########################################

#  You need to create migration credentials in EDB Postgres AI platform

#  and set the path of the credentials directory
```

(continues on next page)

(continued from previous page)

```
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=ORACLE
export DBCONFIG_DATABASES_0__HOSTNAME=<host>
export DBCONFIG_DATABASES_0__PORT=<1521>

#  The CATALOG is the database name

export DBCONFIG_DATABASES_0__CATALOG=<source>
export DBCONFIG_DATABASES_0__USERNAME=<SYS>

#  The password env can be set without specifying it here

#  but the env structure looks like this

export DBCONFIG_DATABASES_0__PASSWORD=<password>

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_0__RESOURCEID=1234567890abcdef

#  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=localhost
# export DBCONFIG_DATABASES_1__PORT=1521

#  The CATALOG is the database name

# export DBCONFIG_DATABASES_1__CATALOG=ORCLCDB/ORCLPDB1
# export DBCONFIG_DATABASES_1__USERNAME=oracle

#  The password env can be set without specifying it here

#  but the env structure looks like this

# export DBCONFIG_DATABASES_1__PASSWORD=password
```

(continues on next page)

(continued from previous page)

```
#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_1__RESOURCEID=1234567890abcdef

########################################
#  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 EDB DMS agent

Start the migration:

```
cdcagent
```

Verifying your source appears in the HM console

- Sources* are databases you connect to Hybrid Manager (HM) using the EDB DMS agent in reader mode so you can use them for migrations.

To verify a source was recognized by the EDB DMS agent and is available for a migration:

1. In the HM console, select the project you're using for the current migration.
2. From the left navigation menu, select **Migrate > Sources**.

Each source's view shows the database name, database type (Postgres or Oracle), `cdcagent` ID, the current status, the source type, and more.

More information

For a description of the available data on this page of the HM console, see Sources .

Creating a data migration with Hybrid Manager

After you configured the EDB DMS agent (in reader mode) to establish connections to Hybrid Manager (HM), create a migration in the HM console. A migration establishes a sync between the source database and a destination database with the help of EDB DMS.

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 EDB DMS agent is listed in the **Source** menu.

6. Select the migration type: **Snapshot** or **Snapshot + Streaming** .

7. Under **Destination Type**, select an HM-managed destination database cluster and database for the migration. Then, select **Next**.
8. Select the tables and columns to migrate. Modify the table and column names if needed.
9. Select **Create Migration**.

The HM console displays a new migration.

If you chose the **Snapshot + Streaming** option, the EDB DMS agent is constantly streaming data when the migration displays the **Running** state. Changes to data are replicated from the source to the destination database as long as the migration is running.

> **Note**
>
> The EDB DMS agent in reader mode streams data changes. It doesn't stream changes in the DDL objects.

Finalizing a streaming data migration

If you selected the **Snapshot + Streaming** option when creating a migration, the EDB DMS agent continues to stream any data updates performed on the source database to the destination database until you mark the migration as completed (for example, to perform a cutover).

If you selected the **Snapshot** option, you can skip this page, as the DMS marks the migration as completed after it finishes copying the entire snapshot to the destination.

Stop the streaming process

This process applies only to **Snapshot + Streaming** migrations.

It's important that you carefully select the best time to stop the data migration stream. Ensure no new inserts are executed in the source and verify that the last known inserts were synchronized in the destination.

After checking that your data is present in the destination database, you can stop the stream by marking the migration as completed.

1. In the Hybrid Manager (HM) console, in your project, select **Migrate > Migrations**.
2. Select the checkmark next to your migration to mark the migration as completed.

Alternatively, select the migration from the list to open the Migration details page . Then select **Mark as Completed**.

This stops the streaming procedure.

Applying constraints after the migration is finalized

At the beginning of your data migration journey with EDB Data Migration Service (EDB DMS), you prepared and imported the schema of your source database. Now, connect to the destination database and reapply the constraints that were excluded from the schema and data migration.

Primary key and unique constraints

For primary key and unique constraints, you already created the tables and constraints in the destination Postgres database. This allowed EDB DMS to map them to the source objects and migrate data successfully. You don't need do to anything else.

The same applies to not null constraints if you included them in your schema import.

Foreign key, check, and exclusion constraints

You can now reapply any foreign key, check, or exclusion constraints you excluded during the schema preparation and import.

Ensuring data integrity

Rows in tables that don't have primary key or unique constraints were migrated with at-least-once delivery, Therefore, it's possible that these rows are duplicates.

Deduplication can be performed as part of the verification.

Verification

To verify that all data was migrated, compare the source and destination databases.

You can use LiveCompare .

Migrate your self-managed Postgres database to an HM-managed database

These are all the steps you need to perform a migration from a self-managed Postgres database to a database cluster managed by Hybrid Manager (HM).

Prepare your source database

Preparing your source database for data migration involves creating a dedicated user for the migration and altering the database configuration to enable replication features.

1. *Preparing your AWS RDS/Aurora PostgreSQL source database* .

Assess the database (optional)

Install and configure the HM agent with your database to obtain observability and monitoring data for your Postgres database. This optional step also provides high-level migration metrics, like schema count and names, table count, and object count.

1. *Configure the HM agent to assess your database* .

Create a destination database

Create a cluster in the HM instance to host the destination database.

1. *Creating your HM-managed destination database* .

Migrate your schema

Handle incompatibilities and transfer your schemas.

1. *Understanding schema handling* .
2. *Migrating Postgres schemas with Migration Toolkit* .

Migrate your data

Execute the data transfer and complete the final cleanup steps.

1. *Creating Hybrid Manager project credentials for the migration* .
2. *Configuring and running the EDB DMS agent in reader mode* .
3. *Verifying your source appears in the HM console* .
4. *Creating a data migration with Hybrid Manager* .
5. *Finalizing a streaming data migration* (for streaming migrations only).
6. *Applying constraints after the migration is finalized* .

Prepare Source

Preparing your Postgres source database

Connect to the source database

To prepare your Postgres source for the migration, you need administrative privileges. Create a change data capture (CDC) migration role with limited privileges for data migration.

Execute SQL statements with psql or a similar client.

To connect to the source database using `psql` :

```
psql -h <PG_HOST> -p <PG_PORT> -U <PG_USERNAME> -d <PG_DB_NAME>
```

Where:

- `<PG_DB_NAME>` is the name of the Postgres database source to connect to.
- `<PG_HOST>` is the Postgres database host.
- `<PG_PORT>` is the Postgres database port.
- `<PG_USERNAME>` is an administrative user who can create and grant roles, alter ownership of tables to migrate, and create a replication slot.

This command prompts you for the password associated with `<PG_USERNAME>` .

Set the Postgres configuration

Ensure the following configuration entries for Postgres are set in the `postgresql.conf` file as follows:

1. Configure `wal_level` as `logical` .

The CDC migration process leverages Postgres logical decoding. Setting `wal_level` to `logical` enables logical decoding of the Postgres write-ahead log (WAL).

1. Configure `max_wal_senders` appropriately.

If EDB Data Migration Service migration is the first streaming client for your database, set `max_wal_senders` to at least `1` . Other streaming clients might be present. Consult your DBA for the appropriate value for streaming client connectivity.

1. Configure `max_replication_slots` appropriately.

`max_replication_slots` must be at least `1` for the CDC migration process. This value can be higher if your organization uses Postgres replication.

1. Configure `max_wal_size` for adequate WAL LSN lifetime.

Set the `max_wal_size` value large enough that production traffic is generating mostly timed checkpoints and not requested checkpoints based on WAL size.

The streaming migration process also requires changes to be available in the WAL until they can be streamed to durable message storage in the cloud infrastructure of EDB DMS. Setting `max_wal_size` too small can affect performance. It can also interfere with the migration process by allowing Postgres LSNs to be dropped from the WAL before they can be streamed. We recommend setting `max_wal_size` to at least 8 GB for

improved migration performance . Consult your DBA to determine the appropriate size.

1. Restart the database server to apply these changes.

Create new roles and grant access for CDC migration

Create a new role for CDC migration with `LOGIN` and `REPLICATION` abilities granted:

```
CREATE ROLE <MIGRATION_ROLE> WITH REPLICATION LOGIN PASSWORD <MIGRATION_ROLE_PASSWORD>
↪;
```

`<MIGRATION_ROLE>` needs to own the source tables to autocreate Postgres publications. Because the source tables are already owned by another role, you create a role/user that can act as the new owner and grant the specified replication group role to both the current table owner and to `<MIGRATION_ROLE>` :

```
CREATE ROLE <REPLICATION_GROUP>;
GRANT <REPLICATION_GROUP> TO <MIGRATION_ROLE>;
GRANT <REPLICATION_GROUP> TO <ORIGINAL_OWNER>;
ALTER TABLE <TABLE_NAME> OWNER TO <REPLICATION_GROUP>
```

Where:

- <MIGRATION_ROLE> is the name of the Postgres role or user to use for CDC migration database access.
- <ORIGINAL_OWNER> is the original production owner of the table.
- <REPLICATION_GROUP> is the name of a role used to own the source tables to migrate for publication autocreation.
- <TABLE_NAME> is the name of a table to migrate and is case sensitive.

Grant usage of the source schemas to the CDC migration role

To select tables in a schema, you must grant usage on the schema in addition to granting SELECT on the tables of the schema. This usage is necessary for all source schemas containing tables to be migrated.

You can grant usage on a schema using the following command:

```
GRANT USAGE ON SCHEMA <DB_SCHEMA> TO <MIGRATION_ROLE>;
```

Grant SELECT on source tables to the CDC migration role

The new <MIGRATION_ROLE> now has schema access but still needs SELECT access to the source tables of those schemas. You can grant access across a schema or for each table.

For an entire schema's tables, use:

```
GRANT SELECT ON ALL TABLES IN SCHEMA <DB_SCHEMA> TO <MIGRATION_ROLE>;
```

For each table, use:

```
GRANT SELECT ON <TABLE_NAME> TO <MIGRATION_ROLE>
```

Where:

- <DB_SCHEMA> is the database schema name for the tables to migrate.
- <MIGRATION_ROLE> is the name of the Postgres role or user to use for CDC migration database access.
- <TABLE_NAME> is the name of a table to migrate and is case sensitive.

Grant CREATE privileges on the source database to the CDC migration role

Since the pgoutput plugin is being used, the migration user needs to be granted CREATE privileges on the source database for Debezium to be able to create publications. To grant the CREATE privilege:

```
GRANT CREATE ON DATABASE <PG_DB_NAME> to <MIGRATION_ROLE>
```

SSL configuration

Ensure you configure your source database server to accept SSL connections to allow the EDB DMS Agent in reader mode to connect to it. You must create a server certificate and a server private key to enable this configuration. You can create the private key with a tool like OpenSSL.

Validate your configuration

After you configure the source database, validate your Postgres configuration. For this, you need the configuration validation script packaged with the DMS Agent.

Prerequisites

1. Ensure the machine where you're running the DMS Agent has Java/OpenJDK 17 or later installed.
2. Install the EDB DMS Agent .
3. Navigate to the EDB DMS Agent folder in `/opt/cdcagent/` .

Run the config validation script

This example shows how to run the validation script:

1. 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. DBZ_USER here refers to the user for cdc (the value of MIGRATION_ROLE above); DBZ_PASSWORD here refers to the password you set for the user of the cdc (the value of MIGRATION_ROLE_PASSWORD above) For example:

```
PG_USERNAME=postgres PG_PASSWORD=password PG_HOST=localhost PG_PORT=5432 DB_
↪NAME=postgres DBZ_USERNAME=debezium DBZ_PASSWORD=dbz ./postgresConfigValidation.sh "
↪${arr[@]}"
```

You get a response similar to the following:

```
 *** [Transporter] - Get PostgreSQL Major Version
PostgreSQL major version: 16

 *** [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: 1 GB
```

(continues on next page)

(continued from previous page)

```
[Pass] max_wal_size is at least 8 GB.

 *** [Transporter] - Validate checkpoints
checkpoints_timed: 424
checkpoints_req: 1
[Pass] Timed checkpoints are more frequent than requested checkpoints.

 *** [Transporter] - Check debezium user role
[Pass] User debezium is present
[Pass] User debezium has replication permission

 *** [Transporter] - Check SELECT privilege on the tables to be migrated
[Pass] User debezium has select privilege on all tables to be migrated.

 *** [Transporter] - Check CREATE privilege on database
[Pass] Role debezium has CREATE privilege on database postgres.
```

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 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

Debezium Postgres Connector for information on CDC configuration.

- See the

Postgres replication documentation for more information on `max_replication_slots` configuration.

- See the

*EDB blog post on tuning `max_wal_size* <https://www.enterprisedb.com/blog/tuning-maxwalsize-postgresql>`_ and Postgres WAL documentation .

Assess Database

Configuring the HM agent to assess your database

You can install and configure the Hybrid Manager (HM) agent to obtain observability and monitoring data for your Postgres database. This optional step provides high-level metrics like schema count and names, table count, and object count, which can aid in migration planning.

> **Note**
>
> Configuring the HM agent isn't required to perform the migration, and it currently doesn't ingest the actual schema objects (DDL).

To register a database with the HM agent, see:

- Monitoring self-managed deployments
-

Monitoring Cloud Service Provider managed Postgres deployments

Creating Destination

Creating your HM-managed destination database

Create new database

If you don't have a destination database cluster on Hybrid Manager (HM) yet, create one that HM can use as destination for migrations:

1. In the HM console, select the project where you want to create the database cluster.
2. In your project, select **Create New** and **Database Cluster** to create a database cluster to serve as the destination for the migration.
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`.

Use existing database

Alternatively, you can use an existing HM-managed database as a destination for your migration. Before you use an existing database cluster as a destination for the migration, ensure the tables you migrate and the load generated on the destination don't interfere with existing workloads.

If you want to use an existing HM-managed database as a destination, continue with the next step.

Migrate Schema

Understanding schema handling

Before you use EDB Data Migration Service (EDB DMS) to configure a data migration that uses the `cdcagent`, you must prepare and import your schema to the destination database.

Importing the schema to the destination database before the migration takes place is important as EDB DMS migrates only data. The destination database must have the schemas in place for the migration to populate them with data.

Some of your schema's constraints must be included before the data migration takes place. Others must be applied after the data migration is completed. This approach ensures you can migrate without performance degradation.

Schema integrity and performance considerations

The presence of destination database constraints, triggers, and WAL logging can impact the data migration performance. When this approach is possible, we recommend a two-step import of schema constraints.

Primary key and unique constraints

EDB DMS leverages primary key/unique constraints to provide an exactly-once delivery when migrating data to the destination database. Therefore, you must include primary key/unique constraints in the schema import that you perform before the data migration begins. Exclude other types of constraints from the schema import.

For rows in tables that don't have primary key/unique constraints you can achieve only at-least-once delivery. You can perform deduplication during the data migration verification.

> **Note**
>
> Not null constraints don't represent a significant performance impact for destination servers and can also be included in the schema import.

Foreign key, check, and exclusion constraints

EDB DMS can apply change events in parallel against destination database clusters. However, migrating some constraint types can negatively affect migration performance. These type of constraints lead to unnecessary CPU and memory utilization in the context of an in-flight data migration from a consistent and referentially integral source database.

Therefore, we recommend applying foreign key, check, and exclusion constraints on the destination database only after the migration is complete.

Migrating Postgres schemas with Migration Toolkit

Schema handling for Postgres databases

You can use several different methods to prepare your schema, exclude unsupported constraints, and import it to the destination database. The following are guidelines for how to export and import your schemas for Postgres migrations using Migration Toolkit.

Other valid routes for migrating DDL to import Postgres schemas include:

- Manually creating the schemas in the destination database
- Performing a pg_dump to obtain the schemas and

pg_restore to
restore them

- Using

pgAdmin or psql

Export

For data migrations from Postgres, we recommend using EDB Migration Toolkit to manage the schema. MTK's offline migration capability provides an easy way to extract a database's schema and separate constraints. Here are some guidelines:

- Use the `-offlineMigration` option, so Migration Toolkit can generate SQL scripts that you can apply on destination without having to configure connectivity.
- Use the `-schemaOnly <schema_scope>` option, so Migration Toolkit can generate scripts that focus only on the export of schemas.
- After you run Migration Toolkit, exclude (comment out) the following constraints from the generated SQL script before applying it on the destination database:
- Foreign key constraints
- Check constraints
- Exclusion constraints

- After you run Migration Toolkit, and if you plan to map objects to different names in the destination, modify the schema SQL script to reflect the new names. After the SQL script is generated, edit the object names directly in the script (DDL) before applying it to the destination. Also make sure that the structure and data types of the renamed object in the script exactly match the source. If you manually update object names in the DDL scripts, also update the names in other objects that reference the updated objects. For example, if you rename a table, also update any views or constraints that reference the original table name.

Later, during data migration, select the newly assigned object names. See Mapping schema, tables and columns during a migration for more information.

Import

After you prepare the DDL and exclude foreign key, check, and exclusion constraints, connect to the destination database and import the SQL-formatted DDL file.

To start the import, execute the generated offline migration script .

> **Note**
>
> You will have to apply the excluded foreign key, check, and exclusion constraints on the destination database later in the migration process, after the migration is performed successfully. Therefore, keep track of the excluded constraints, so you can apply them when the time comes.

Migrate Data

Creating Hybrid Manager project credentials for the migration

Credentials are required so you can perform data migrations. Before you can add a source or destination for a migration to Hybrid Manager (HM), you must create migration credentials. You can then use them to allow the EDB DMS agent to connect to HM.

You need one set of credentials per HM project . For example, if you're using a single HM project for all migrations, you need only one set of credentials.

1. On the HM console, select the project you're using for your migration activities.
2. In the project, select **Migrate > Credentials**.

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.

3. Select **Create Migration Credential**, enter a name for your credentials, and select **Create Migration Credential**.
4. After the credentials are generated, select **Download Credential**.

> **Note**
>
> This is the only time you can download the credentials. If you don't download them now and you need your credentials, you'll have to revoke this set and create new ones.

5. Unzip the downloaded zip file and copy the generated folder to the host where the EDB DMS agent is installed.

The decompressed credentials folder contains these files:

```
- `int.crt` - `int.truststore.p12` - `client.keystore.p12` - `client-key.
↪pem` - `client-cert.pem`
```

Configuring and running the EDB DMS agent in reader mode

Configure and run the DMS agent in reader mode to perform a data migration.

Prerequisites

1. Ensure the machine where you're running the DMS agent has Java/OpenJDK 17 or later installed.
2. Install the EDB DMS agent .
3. Open the EDB DMS agent located in `/opt/cdcagent/run-cdcagent.sh` with write permissions.

Configuring the EDB DMS Agent for migrations from Postgres

Set the variables in the script according to your environment and uncomment the edited lines. See parameters for more guidance. The following shows the unedited script:

```
# !/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
```

(continues on next page)

(continued from previous page)

```
#  Consult your system administrators

export RW_SERVICE_HOST=https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter

########################################
#  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>

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_0__RESOURCEID=1234567890abcdef

#  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=localhost
# export DBCONFIG_DATABASES_1__PORT=1521
```

(continues on next page)

(continued from previous page)

```
#  The CATALOG is the database name

# export DBCONFIG_DATABASES_1__CATALOG=ORCLCDB/ORCLPDB1
# export DBCONFIG_DATABASES_1__USERNAME=oracle

#  The password env can be set without specifying it here

#  but the env structure looks like this

# export DBCONFIG_DATABASES_1__PASSWORD=password

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_1__RESOURCEID=1234567890abcdef

########################################
#  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)
```

(continues on next page)

(continued from previous page)

```
    java -jar ./writer/quarkus-run.jar
    ;;
  *)
    echo "Unknown RUN_MODE: $RUN_MODE"
    exit 1
    ;;
esac
```

Running the EDB DMS agent

Start the migration:

```
cdcagent
```

Verifying your source appears in the HM console

- Sources* are databases you connect to Hybrid Manager (HM) using the EDB DMS agent in reader mode so you can use them for migrations.

To verify a source was recognized by the EDB DMS agent and is available for a migration:

1. In the HM console, select the project you're using for the current migration.
2. From the left navigation menu, select **Migrate > Sources**.

Each source's view shows the database name, database type (Postgres or Oracle), `cdcagent` ID, the current status, the source type, and more.

More information

For a description of the available data on this page of the HM console, see Sources .

Creating a data migration with Hybrid Manager

After you configured the EDB DMS agent (in reader mode) to establish connections to Hybrid Manager (HM), create a migration in the HM console. A migration establishes a sync between the source database and a destination database with the help of EDB DMS.

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 EDB DMS agent is listed in the **Source** menu.
6. Select the migration type: **Snapshot** or **Snapshot + Streaming** .
7. Under **Destination Type**, select an HM-managed destination database cluster and database for the migration. Then, select **Next**.
8. Select the tables and columns to migrate. Modify the table and column names if needed.
9. Select **Create Migration**.

The HM console displays a new migration.

If you chose the **Snapshot + Streaming** option, the EDB DMS agent is constantly streaming data when the migration displays the **Running** state. Changes to data are replicated from the source to the destination database as long as the migration is running.

> **Note**
>
> The EDB DMS agent in reader mode streams data changes. It doesn't stream changes in the DDL objects.

Finalizing a streaming data migration

If you selected the **Snapshot + Streaming** option when creating a migration, the EDB DMS agent continues to stream any data updates performed on the source database to the destination database until you mark the migration as completed (for example, to perform a cutover).

If you selected the **Snapshot** option, you can skip this page, as the DMS marks the migration as completed after it finishes copying the entire snapshot to the destination.

Stop the streaming process

This process applies only to **Snapshot + Streaming** migrations.

It's important that you carefully select the best time to stop the data migration stream. Ensure no new inserts are executed in the source and verify that the last known inserts were synchronized in the destination.

After checking that your data is present in the destination database, you can stop the stream by marking the migration as completed.

1. In the Hybrid Manager (HM) console, in your project, select **Migrate > Migrations**.
2. Select the checkmark next to your migration to mark the migration as completed.

Alternatively, select the migration from the list to open the Migration details page . Then select **Mark as Completed**.

This stops the streaming procedure.

Applying constraints after the migration is finalized

At the beginning of your data migration journey with EDB Data Migration Service (EDB DMS), you prepared and imported the schema of your source database. Now, connect to the destination database and reapply the constraints that were excluded from the schema and data migration.

Primary key and unique constraints

For primary key and unique constraints, you already created the tables and constraints in the destination Postgres database. This allowed EDB DMS to map them to the source objects and migrate data successfully. You don't need do to anything else.

The same applies to not null constraints if you included them in your schema import.

Foreign key, check, and exclusion constraints

You can now reapply any foreign key, check, or exclusion constraints you excluded during the schema preparation and import.

Ensuring data integrity

Rows in tables that don't have primary key or unique constraints were migrated with at-least-once delivery, Therefore, it's possible that these rows are duplicates.

Deduplication can be performed as part of the verification.

Verification

To verify that all data was migrated, compare the source and destination databases.

You can use LiveCompare .

Migrate your self-managed Postgres database to a different self-managed Postgres database

These are all the steps you need to perform a migration from a self-managed Postgres database to a different self-managed Postgres database using the data migration service (DMS) of Hybrid Manager (HM).

Prepare your source database

Prepare your source database for data migration. This involves creating a dedicated user for the migration and altering the database configuration to enable replication features.

1. *Preparing your AWS RDS/Aurora PostgreSQL source database* .

Assess the database (optional)

Install and configure the HM agent with your database to obtain observability and monitoring data for your Postgres database. This optional step also provides high-level migration metrics, like schema count and names, table count, and object count.

1. *Configure the HM agent to assess your database* .

Prepare your destination database

Create a server to host the destination database. Then, alter the destination database's configuration to to enable replication features.

1. *Creating your HM-managed destination database* .
2. *Preparing your Postgres self-managed destination database* .

Migrate your schema

Handle incompatibilities and transfer your schemas.

1. *Understanding schema handling* .
2. *Migrating Postgres schemas with Migration Toolkit* .

Migrate your data

Execute the data transfer and complete the final cleanup steps.

1. *Creating Hybrid Manager project credentials for the migration* .
2. *Configuring and running the EDB DMS agent in reader mode* .
3. *Verifying your source appears in the HM console* .

4. *Configuring and running the EDB DMS agent in writer mode* .
5. *Verifying your destination appears in the HM console* .
6. *Creating a data migration with Hybrid Manager* .
7. *Finalizing a streaming data migration* (for streaming migrations only).
8. *Applying constraints after the migration is finalized* .

Prepare Source

Preparing your Postgres source database

Connect to the source database

To prepare your Postgres source for the migration, you need administrative privileges. Create a change data capture (CDC) migration role with limited privileges for data migration.

Execute SQL statements with psql or a similar client.

To connect to the source database using `psql` :

```
psql -h <PG_HOST> -p <PG_PORT> -U <PG_USERNAME> -d <PG_DB_NAME>
```

Where:

- `<PG_DB_NAME>` is the name of the Postgres database source to connect to.
- `<PG_HOST>` is the Postgres database host.
- `<PG_PORT>` is the Postgres database port.
- `<PG_USERNAME>` is an administrative user who can create and grant roles, alter ownership of tables to migrate, and create a replication slot.

This command prompts you for the password associated with `<PG_USERNAME>` .

Set the Postgres configuration

Ensure the following configuration entries for Postgres are set in the `postgresql.conf` file as follows:

1. Configure `wal_level` as `logical` .

The CDC migration process leverages Postgres logical decoding. Setting `wal_level` to `logical` enables logical decoding of the Postgres write-ahead log (WAL).

1. Configure `max_wal_senders` appropriately.

If EDB Data Migration Service migration is the first streaming client for your database, set `max_wal_senders` to at least `1` . Other streaming clients might be present. Consult your DBA for the appropriate value for streaming client connectivity.

1. Configure `max_replication_slots` appropriately.

`max_replication_slots` must be at least `1` for the CDC migration process. This value can be higher if your organization uses Postgres replication.

1. Configure `max_wal_size` for adequate WAL LSN lifetime.

Set the `max_wal_size` value large enough that production traffic is generating mostly timed checkpoints and not requested checkpoints based on WAL size.

The streaming migration process also requires changes to be available in the WAL until they can be streamed to durable message storage in the cloud infrastructure of EDB DMS. Setting `max_wal_size` too small can affect performance. It can also interfere with the migration process by allowing Postgres LSNs to be dropped from the WAL before they can be streamed. We recommend setting `max_wal_size` to at least 8 GB for

improved migration performance . Consult your DBA to determine the appropriate size.

1. Restart the database server to apply these changes.

Create new roles and grant access for CDC migration

Create a new role for CDC migration with `LOGIN` and `REPLICATION` abilities granted:

```
CREATE ROLE <MIGRATION_ROLE> WITH REPLICATION LOGIN PASSWORD <MIGRATION_ROLE_PASSWORD>
↪;
```

`<MIGRATION_ROLE>` needs to own the source tables to autocreate Postgres publications. Because the source tables are already owned by another role, you create a role/user that can act as the new owner and grant the specified replication group role to both the current table owner and to `<MIGRATION_ROLE>` :

```
CREATE ROLE <REPLICATION_GROUP>;
GRANT <REPLICATION_GROUP> TO <MIGRATION_ROLE>;
GRANT <REPLICATION_GROUP> TO <ORIGINAL_OWNER>;
ALTER TABLE <TABLE_NAME> OWNER TO <REPLICATION_GROUP>
```

Where:

- `<MIGRATION_ROLE>` is the name of the Postgres role or user to use for CDC migration database access.
- `<ORIGINAL_OWNER>` is the original production owner of the table.
- `<REPLICATION_GROUP>` is the name of a role used to own the source tables to migrate for publication autocreation.
- `<TABLE_NAME>` is the name of a table to migrate and is case sensitive.

Grant usage of the source schemas to the CDC migration role

To select tables in a schema, you must grant usage on the schema in addition to granting `SELECT` on the tables of the schema. This usage is necessary for all source schemas containing tables to be migrated.

You can grant usage on a schema using the following command:

```
GRANT USAGE ON SCHEMA <DB_SCHEMA> TO <MIGRATION_ROLE>;
```

Grant `SELECT` on source tables to the CDC migration role

The new `<MIGRATION_ROLE>` now has schema access but still needs `SELECT` access to the source tables of those schemas. You can grant access across a schema or for each table.

For an entire schema's tables, use:

```
GRANT SELECT ON ALL TABLES IN SCHEMA <DB_SCHEMA> TO <MIGRATION_ROLE>;
```

For each table, use:

```
GRANT SELECT ON <TABLE_NAME> TO <MIGRATION_ROLE>
```

Where:

- <DB_SCHEMA> is the database schema name for the tables to migrate.
- <MIGRATION_ROLE> is the name of the Postgres role or user to use for CDC migration database access.
- <TABLE_NAME> is the name of a table to migrate and is case sensitive.

Grant CREATE privileges on the source database to the CDC migration role

Since the pgoutput plugin is being used, the migration user needs to be granted CREATE privileges on the source database for Debezium to be able to create publications. To grant the CREATE privilege:

```
GRANT CREATE ON DATABASE <PG_DB_NAME> to <MIGRATION_ROLE>
```

SSL configuration

Ensure you configure your source database server to accept SSL connections to allow the EDB DMS Agent in reader mode to connect to it. You must create a server certificate and a server private key to enable this configuration. You can create the private key with a tool like OpenSSL.

Validate your configuration

After you configure the source database, validate your Postgres configuration. For this, you need the configuration validation script packaged with the DMS Agent.

Prerequisites

1. Ensure the machine where you're running the DMS Agent has Java/OpenJDK 17 or later installed.
2. Install the EDB DMS Agent .
3. Navigate to the EDB DMS Agent folder in /opt/cdcagent/ .

Run the config validation script

This example shows how to run the validation script:

1. 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. DBZ_USER here refers to the user for cdc (the value of MIGRATION_ROLE above); DBZ_PASSWORD here refers to the password you set for the user of the cdc (the value of MIGRATION_ROLE_PASSWORD above) For example:

```
PG_USERNAME=postgres PG_PASSWORD=password PG_HOST=localhost PG_PORT=5432 DB_
↪NAME=postgres DBZ_USERNAME=debezium DBZ_PASSWORD=dbz ./postgresConfigValidation.sh "
↪${arr[@]}"
```

You get a response similar to the following:

```
 *** [Transporter] - Get PostgreSQL Major Version
PostgreSQL major version: 16
```

(continues on next page)

(continued from previous page)

```
 *** [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: 1 GB
[Pass] max_wal_size is at least 8 GB.

 *** [Transporter] - Validate checkpoints
checkpoints_timed: 424
checkpoints_req: 1
[Pass] Timed checkpoints are more frequent than requested checkpoints.

 *** [Transporter] - Check debezium user role
[Pass] User debezium is present
[Pass] User debezium has replication permission

 *** [Transporter] - Check SELECT privilege on the tables to be migrated
[Pass] User debezium has select privilege on all tables to be migrated.

 *** [Transporter] - Check CREATE privilege on database
[Pass] Role debezium has CREATE privilege on database postgres.
```

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 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

Debezium Postgres Connector for information on CDC configuration.

- See the

Postgres replication documentation for more information on
`max_replication_slots` configuration.

- See the

*EDB blog post on tuning `max_wal_size* <https://www.enterprisedb.com/blog/tuning-maxwalsize-postgresql>`_ and Postgres WAL documentation .

Assess Database

Configuring the HM agent to assess your database

You can install and configure the Hybrid Manager (HM) agent to obtain observability and monitoring data for your Postgres database. This optional step provides high-level metrics like schema count and names, table count, and object count, which can aid in migration planning.

> **Note**
>
> Configuring the HM agent isn't required to perform the migration, and it currently doesn't ingest the actual schema objects (DDL).

To register a database with the HM agent, see:

- Monitoring self-managed deployments
-

Monitoring Cloud Service Provider managed Postgres deployments

Prepare Destination

Creating your self-managed destination database

You can use existing databases as destinations for migrations. To use an existing instance as a destination for the migration, ensure the tables you migrate and the load generated on the destination don't interfere with existing workloads.

If you don't have a self-managed destination database yet, create one that the EDB DMS agent can connect to for Hybrid Manager (HM) migrations. This example assumes you 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 destination database to use as a destination for the migration.

In this example, the destination database is `tr_target` :

```
CREATE DATABASE tr_target;
```

Preparing your Postgres self-managed destination database

Prepare your destination database before starting a migration.

Prerequisites

- Create a database in your destination Postgres database server.

SSL configuration

You need to configure SSL only if you're migrating to a Postgres instance created outside of Hybrid Manager (HM). HM-managed destinations don't require this configuration.

Configure your source database server to accept SSL connections to allow the EDB DMS agent to connect to it. To enable this configuration, you must create a server certificate and a server private key (for example, with OpenSSL).

For testing purposes, 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. In your terminal, 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.

2. Move the created files to the Postgres data directory, so the server has access to them:

```
sudo cp server.crt server.key /var/lib/postgresql/17/<data_directory>
```

3. Set the required permissions for the files:

```
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
```

4. Modify the `postgresql.conf` file to enable SSL:

```
ssl = on
ssl_cert_file = server.crt
ssl_key_file = server.key
```

5. Update the `pg_hba.conf` file to allow local SSL connections. Add the following line:

```
hostssl all             all             0.0.0.0/0               md5
```

6. Restart your server to apply the changes:

```
/usr/lib/postgresql/17/bin/pg_ctl -D /var/lib/postgresql/17/<data_directory> restart
```

Create a role for CDC migration

You need to configure a role for migration only if you're migrating to a Postgres instance created outside of HM. HM-managed destinations don't require this configuration.

The EDB DMS agent requires certain permissions to access and be able to write data during the migration. For this, you can either use an existing superuser or create a new user with the required permissions. This example creates a new user:

1. Create a role for CDC migration with `LOGIN` abilities granted:

```
CREATE ROLE <MIGRATION_ROLE_WRITER> WITH LOGIN PASSWORD <ROLE_PASSWORD>;
```

Where `<MIGRATION_ROLE_WRITER>` is the name of the Postgres role or user to use for destination database access.

2. Grant permissions so the user can control the behavior of 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 versions. If your destination database is on Postgres 14 or earlier, use a superuser for the migration.

Provide this user to the EDB DMS agent during configuration via the `DBCONFIG_DATABASES_0__USERNAME` and `DBCONFIG_DATABASES_0__PASSWORD` values.

Migrate Schema

Understanding schema handling

Before you use EDB Data Migration Service (EDB DMS) to configure a data migration that uses the `cdcagent` , you must prepare and import your schema to the destination database.

Importing the schema to the destination database before the migration takes place is important as EDB DMS migrates only data. The destination database must have the schemas in place for the migration to populate them with data.

Some of your schema's constraints must be included before the data migration takes place. Others must be applied after the data migration is completed. This approach ensures you can migrate without performance degradation.

Schema integrity and performance considerations

The presence of destination database constraints, triggers, and WAL logging can impact the data migration performance. When this approach is possible, we recommend a two-step import of schema constraints.

Primary key and unique constraints

EDB DMS leverages primary key/unique constraints to provide an exactly-once delivery when migrating data to the destination database. Therefore, you must include primary key/unique constraints in the schema import that you perform before the data migration begins. Exclude other types of constraints from the schema import.

For rows in tables that don't have primary key/unique constraints you can achieve only at-least-once delivery. You can perform deduplication during the data migration verification.

> **Note**
>
> Not null constraints don't represent a significant performance impact for destination servers and can also be included in the schema import.

Foreign key, check, and exclusion constraints

EDB DMS can apply change events in parallel against destination database clusters. However, migrating some constraint types can negatively affect migration performance. These type of constraints lead to unnecessary CPU and memory utilization in the context of an in-flight data migration from a consistent and referentially integral source database.

Therefore, we recommend applying foreign key, check, and exclusion constraints on the destination database only after the migration is complete.

Migrating Postgres schemas with Migration Toolkit

Schema handling for Postgres databases

You can use several different methods to prepare your schema, exclude unsupported constraints, and import it to the destination database. The following are guidelines for how to export and import your schemas for Postgres migrations using Migration Toolkit.

Other valid routes for migrating DDL to import Postgres schemas include:

- Manually creating the schemas in the destination database
- Performing a pg_dump to obtain the schemas and

pg_restore to
restore them

- Using

pgAdmin or psql

Export

For data migrations from Postgres, we recommend using EDB Migration Toolkit to manage the schema. MTK's offline migration capability provides an easy way to extract a database's schema and separate constraints. Here are some guidelines:

- Use the `-offlineMigration` option, so Migration Toolkit can generate SQL scripts that you can apply on destination without having to configure connectivity.
- Use the `-schemaOnly <schema_scope>` option, so Migration Toolkit can generate scripts that focus only on the export of schemas.
- After you run Migration Toolkit, exclude (comment out) the following constraints from the generated SQL script before applying it on the destination database:
- Foreign key constraints
- Check constraints
- Exclusion constraints
- After you run Migration Toolkit, and if you plan to map objects to different names in the destination, modify the schema SQL script to reflect the new names. After the SQL script is generated, edit the object names directly in the script (DDL) before applying it to the destination. Also make sure that the structure and data types of the renamed object in the script exactly match the source. If you manually update object names in the DDL scripts, also update the names in other objects that reference the updated objects. For example, if you rename a table, also update any views or constraints that reference the original table name.

Later, during data migration, select the newly assigned object names. See Mapping schema, tables and columns during a migration for more information.

Import

After you prepare the DDL and exclude foreign key, check, and exclusion constraints, connect to the destination database and import the SQL-formatted DDL file.

To start the import, execute the generated offline migration script .

> **Note**
>
> You will have to apply the excluded foreign key, check, and exclusion constraints on the destination database later in the migration process, after the migration is performed successfully. Therefore, keep track of the excluded constraints, so you can apply them when the time comes.

Migrate Data

Creating Hybrid Manager project credentials for the migration

Credentials are required so you can perform data migrations. Before you can add a source or destination for a migration to Hybrid Manager (HM), you must create migration credentials. You can then use them to allow the EDB DMS agent to connect to HM.

You need one set of credentials per HM project . For example, if you're using a single HM project for all migrations, you need only one set of credentials.

1. On the HM console, select the project you're using for your migration activities.
2. In the project, select **Migrate > Credentials**.

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.

3. Select **Create Migration Credential**, enter a name for your credentials, and select **Create Migration Credential**.
4. After the credentials are generated, select **Download Credential**.

> **Note**
>
> This is the only time you can download the credentials. If you don't download them now and you need your credentials, you'll have to revoke this set and create new ones.

5. Unzip the downloaded zip file and copy the generated folder to the host where the EDB DMS agent is installed.

The decompressed credentials folder contains these files:

```
- `int.crt` - `int.truststore.p12` - `client.keystore.p12` - `client-key.
↪pem` - `client-cert.pem`
```

Configuring and running the EDB DMS agent in reader mode

Configure and run the DMS agent in reader mode to perform a data migration.

Prerequisites

1. Ensure the machine where you're running the DMS agent has Java/OpenJDK 17 or later installed.
2. Install the EDB DMS agent .
3. Open the EDB DMS agent located in `/opt/cdcagent/run-cdcagent.sh` with write permissions.

Configuring the EDB DMS Agent for migrations from Postgres

Set the variables in the script according to your environment and uncomment the edited lines. See parameters for more guidance. The following shows the unedited script:

```
# !/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

#  Consult your system administrators

export RW_SERVICE_HOST=https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter

########################################
#  DMS Agent TLS Configuration          #

########################################

#  You need to create migration credentials in EDB Postgres AI platform

#  and set the path of the credentials directory
```

(continues on next page)

(continued from previous page)

```
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>

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_0__RESOURCEID=1234567890abcdef

#  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=localhost
# export DBCONFIG_DATABASES_1__PORT=1521

#  The CATALOG is the database name

# export DBCONFIG_DATABASES_1__CATALOG=ORCLCDB/ORCLPDB1
# export DBCONFIG_DATABASES_1__USERNAME=oracle

#  The password env can be set without specifying it here

#  but the env structure looks like this

# export DBCONFIG_DATABASES_1__PASSWORD=password
```

(continues on next page)

(continued from previous page)

```
#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_1__RESOURCEID=1234567890abcdef

########################################
#  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 EDB DMS agent

Start the migration:

```
cdcagent
```

Verifying your source appears in the HM console

- Sources* are databases you connect to Hybrid Manager (HM) using the EDB DMS agent in reader mode so you can use them for migrations.

To verify a source was recognized by the EDB DMS agent and is available for a migration:

1. In the HM console, select the project you're using for the current migration.
2. From the left navigation menu, select **Migrate > Sources**.

Each source's view shows the database name, database type (Postgres or Oracle), `cdcagent` ID, the current status, the source type, and more.

More information

For a description of the available data on this page of the HM console, see Sources .

Configuring and running the EDB DMS agent in writer mode

Configure and run the EDB DMS agent in writer mode to perform a data migration only if the destination for your migration is a self-managed Postgres database. For migrations into a Hybrid Manager (HM)-managed database, you don't need an additional DMS agent in writer mode.

Prerequisites

1. Ensure your machine has Java/OpenJDK 17 or later installed.

2. Install the EDB DMS agent .

3. Open the EDB DMS agent located in `/opt/cdcagent/run-cdcagent.sh` with write permissions.

Configuring the EDB DMS agent for migrations into self-managed Postgres

1. Set the variables in the script according to your environment and uncomment the edited lines. See parameters for more guidance. The following shows the unedited script:

```
# !/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.

#
```

(continues on next page)

(continued from previous page)

```
########################################
#  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

#  Consult your system administrators

export RW_SERVICE_HOST=https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter

########################################
#  DMS Agent TLS Configuration           #

########################################

#  You need to create migration credentials in EDB Postgres AI platform

#  and set 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=<5332>

#  The CATALOG is the database name
```

(continues on next page)

(continued from previous page)

```
export DBCONFIG_DATABASES_0__CATALOG=<destination_db_name>
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>

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_0__RESOURCEID=1234567890abcdef

#  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=localhost
# export DBCONFIG_DATABASES_1__PORT=1521

#  The CATALOG is the database name

# export DBCONFIG_DATABASES_1__CATALOG=ORCLCDB/ORCLPDB1
# export DBCONFIG_DATABASES_1__USERNAME=oracle

#  The password env can be set without specifying it here

#  but the env structure looks like this

# export DBCONFIG_DATABASES_1__PASSWORD=password

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_1__RESOURCEID=1234567890abcdef

########################################
#  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

#  Generic loglevel

# export QUARKUS_LOG_LEVEL=DEBUG

#  Loglevel for a single package

# 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 EDB DMS agent

1. Start the migration:

```
cdcagent
```

Verifying your destination appears in the HM console

- Destinations* are databases you connected to Hybrid Manager (HM) using the EDB DMS agent in writer mode so you can use them for migrations.

To verify a destination was recognized by the EDB DMS agent and is available for a migration:

1. In the HM console, select the project you're using for the current migration.
2. From the left navigation menu, select **Migrate > Destinations**.

Each destinations's view shows the database name, database type, `cdcagent` ID, the current status, the destination type, and more.

More information

For a description of the available data on this page of the HM console, see Destinations .

Creating a data migration with Hybrid Manager

After you have configured a EDB DMS agent in reader mode and a EDB DMS agent in writer mode to establish connections to Hybrid Manager (HM), create a migration in the HM console. A migration establishes a sync between the source database and a destination database with the help of EDB DMS.

1. In the HM console, select the project from which you want to manage the migration.
2. In your 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 EDB DMS agent is listed in the **Source** menu.

6. Select the migration type: **Snapshot** or **Snapshot + Streaming** .

7. Under **Destination Type**, select a self-managed instance as destination for the migration. Then, select **Next**.
8. Select the tables and columns to migrate. Modify the table and column names if needed.
9. Select **Create Migration**.

The HM console displays a new migration.

If you chose the **Snapshot + Streaming** option, the EDB DMS agents are constantly streaming data when the migration displays the **Running** state. Changes to data are replicated from the source to the destination database as long as the migration is running.

> **Note**
>
> The EDB DMS agent in reader mode streams data changes. It doesn't stream changes in the DDL objects.

Finalizing a streaming data migration

If you selected the **Snapshot + Streaming** option when creating a migration, the EDB DMS agent continues to stream any data updates performed on the source database to the destination database until you mark the migration as completed (for example, to perform a cutover).

If you selected the **Snapshot** option, you can skip this page, as the DMS marks the migration as completed after it finishes copying the entire snapshot to the destination.

Stop the streaming process

This process applies only to **Snapshot + Streaming** migrations.

It's important that you carefully select the best time to stop the data migration stream. Ensure no new inserts are executed in the source and verify that the last known inserts were synchronized in the destination.

After checking that your data is present in the destination database, you can stop the stream by marking the migration as completed.

1. In the Hybrid Manager (HM) console, in your project, select **Migrate > Migrations**.
2. Select the checkmark next to your migration to mark the migration as completed.

Alternatively, select the migration from the list to open the Migration details page . Then select **Mark as Completed**.

This stops the streaming procedure.

Applying constraints after the migration is finalized

At the beginning of your data migration journey with EDB Data Migration Service (EDB DMS), you prepared and imported the schema of your source database. Now, connect to the destination database and reapply the constraints that were excluded from the schema and data migration.

Primary key and unique constraints

For primary key and unique constraints, you already created the tables and constraints in the destination Postgres database. This allowed EDB DMS to map them to the source objects and migrate data successfully. You don't need do to anything else.

The same applies to not null constraints if you included them in your schema import.

Foreign key, check, and exclusion constraints

You can now reapply any foreign key, check, or exclusion constraints you excluded during the schema preparation and import.

Ensuring data integrity

Rows in tables that don't have primary key or unique constraints were migrated with at-least-once delivery, Therefore, it's possible that these rows are duplicates.

Deduplication can be performed as part of the verification.

Verification

To verify that all data was migrated, compare the source and destination databases.

You can use LiveCompare .

Reference

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. Because the `RW_SERVICE_HOST` leverages the DMS (transporter) service for migrations, you can derive it from `TRANSPORTER_RW_SERVICE_DOMAIN_NAME` , a domain name established by HM administrators or installers for migration component communication.

To obtain the `RW_SERVICE_HOST` :

1. Ask the HM installer or administrator to provide the value for `TRANSPORTER_RW_SERVICE_DOMAIN_NAME` .
2. Add the `https://` protocol prefix and append `/transporter` to the admin-provided URL.

This modified URL is the correct value to set for `RW_SERVICE_HOST` .

More information

The `RW_SERVICE_HOST` value must always be derived directly from the `TRANSPORTER_RW_SERVICE_DOMAIN_NAME` . The following table provides examples of how the `RW_SERVICE_HOST` is derived, showing both a recommended and a custom naming convention.

Recommended:

TRANSPORTER_RW_SERVICE_DOMAIN_NAME	RW_SERVICE_HOST
dms.examplecompany.com	https://dms.examplecompany.com/transporter
dms-dev.examplecompany.c	https://dms-dev.examplecompany.com/transporter
dms.hm001.examplecompany.com	https://dms.hm001.examplecompany.com/transporter
dms-hm-us-east-001.examp	https://dms-hm-us-east-001.examplecompany.com/transporter

Custom:

TRANSPORTER_RW_SERVICE_DOMAIN_NAME	RW_SERVICE_HOST
987635.examplecompany.com	https://987635.examplecompany.com/transporter
migration_service.examplecompany.com	https://migration_service.examplecompany.com/transporter

continues on next page

Table 10 – continued from previous page

TRANSPORTER_RW_SERVICE_DOMAIN_NAME	RW_SERVICE_HOST
location001.examplecompany.com	https://location001.examplecompany.com/transporter
us_east.examplecompany.com	https://us_east.examplecompany.com/transporter

CREDENTIAL_DIRECTORY_PATH

The directory path of the credentials folder you downloaded from the HM console .

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.

DBCONFIG_DATABASES_0__USERNAME

The database username of the source/destination database.

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 for resource management. If you previously registered this database with the HM agent for monitoring and schema ingestion, use the same `resource_id` here. This consistency is essential for the DMS to correctly correlate the resource across both the HM agent and DMS agent components.

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` .

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_CATEGORY__COM_ENTERPRISEDB__LEVEL

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.

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 HM agent (for example, register an Oracle database with the agent). 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 *Oracle XE compatibility restrictions* 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 HM agent's *EDB DMS agent parameter reference* for more information.

Version: Database version. See *Oracle XE compatibility restrictions* 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 HM agent.

> **Note**
>
> - 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 HM agent detects schema changes to tables and columns while running. However, changes to views or procedures aren't detected until you restart the HM 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 *Detailed view* 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 HM agent. It's the project you assign to an agent in `agent.project_id`. See the HM agent *EDB DMS agent parameter reference* for more information.

Tags: Labels assigned to the migration of a specific database in `databases.tags`. See the HM 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 HM 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 *Oracle XE compatibility restrictions* to understand which databases are supported for migration assessment.
- See *Data security and sovereignty* to understand how your data is stored by the HM agent.

Detailed view

When you select a database from the **Migrations** tab of the **Estate** page in Hybrid Manager (HM), a detailed view opens. This view includes functionality that you can leverage to perform migrations from external databases.

> **Note**
>
> Currently, Postgres databases offer only a high-level assessment. A more comprehensive analysis is available for Oracle databases.

Status

Use the status selector next to the database name to manage and track your migration workflow. This selector allows you to update the current state of a database migration, 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. Also, for Oracle databases, the **Schemas** tab allows you to *create new Migration Portal projects* directly from the extracted schemas.

> **Important**
>
> The HM agent detects schema changes to tables and columns while running. However, changes to views or procedures aren't detected until you restart the HM agent.

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). Available for Oracle databases only.
- Migration level of effort, color-coded by day ranges (0-5, 6-15, 16-25, 26-40, 40+ days), defined by 8-hours working days. Available for Oracle databases only.

Schema table

- **Name:** Name of the schema.
- **Latest Assessment:** Available for Oracle databases only. Status of the latest assessment. Possible values are Not started, In progress, Complete and Error.
- **Last Assessed:** Available for Oracle databases only. Timestamp corresponding to the last time the schema was assessed. The assessment is regenerated or updated if the HM agent discovers that a new schema is available or if an existing one was updated.
- **# of Tables:** Number of tables in the assessed schema.
- **# of Objects:** Number of objects in the assessed schema.
- **Complexity:** Available for Oracle databases only. Indicates how challenging it will be to convert a specific 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:** Available for Oracle databases only. Indicates how many days it will take to migrate a specific schema, in 8-hour working days, based on a detailed, object-level analysis.
- **Dependencies:** Available for Oracle databases only. The name of the schema that the current schema depends on. We recommend migrating schemas with their dependencies in the same batch (using the same Migration Portal project) to ensure that all objects referenced by the schema exist in the destination database.

- **MP Project:** Available for Oracle databases only. Use the integrated Migration Portal to convert an Oracle schema to EDB Postgres Advanced Server. Select the plus sign to open a window where you can provide details to start a new schema migration project with the Migration Portal Project-scoped workspace vs. user's workspace . After you provide the details, HM creates a Migration Portal project in the background. You can access this project later from the ***Migration*** tab.

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: Summarizes key analysis results and assessment metadata.

Compatibility Summary: Available for Oracle databases only. Summarizes database features and schema objects as well as other factors that contribute to the overall migration complexity and level of effort. The complexity is assigned depending on a scoring system that analyses several factors. Some of the factors include database size, large object usage, used partition types, Oracle database features, procedural database objects, and whether they're EDB Postgres Advanced Server compatible. Each of the analyzed object gets a 1-5 score (very easy to very hard). The max complexity value is displayed here.

Oracle PL/SQL features used: Available for Oracle databases only. Lists all PL/SQL objects that contain Oracle built-in features. To understand the migration impact of the Oracle PL/SQL feature, see the object count, the occurrence count, whether they're EDB Postgres Advanced Server-compatible, and the results of the migration complexity assessment.

If all calls of a specific feature have equivalent EDB Postgres Advanced Server implementations, the feature is marked as **EPAS Compatible**. If no equivalent is detected, the feature is marked as incompatible.

Oracle database features used: Available for Oracle databases only. Lists all Oracle database features in use in the database.

Incompatible PL/SQL features in objects: Available for Oracle databases only. Lists all Oracle and non-Oracle incompatible PL/SQL features while displaying the schema they are in, the object type (procedure, function, and so on), the object name, the incompatible feature name, how often the feature appears, and the migration complexity.

Migration

Helps you perform migration-related tasks.

Create a Destination: Creates a new cluster to serve as a destination for your migration. See *Creating a database cluster* for more guidance.

Migrate Schema(s): - For Oracle databases, this option lets you create a project in the built-in Migration Portal (Project-scoped workspace vs. user's workspace) that you can use to convert the Oracle schema to be compatible with EDB Postgres Advanced Server. After you resolve any incompatibilities, the Migration Portal service also lets you connect to the destination cluster to import the converted schema.

- For Postgres databases, this option provides information on how to use external tooling like Migration Toolkit or pg_dump, among other tools, to migrate Postgres schemas.

Migrate Data: Leverages HM's Data Migration Service to perform data migrations with CDC technology. Before you perform data migrations with this function, prepare your database, migrate your schema, and install and configure the EDB DMS agent. See Choose your migration for end-to-end examples of how to perform a complete database migration.

Migration assessment support

After registering a self-managed database with the Hybrid Manager (HM) 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 13-17
- EDB Postgres Advanced Server 13-17
- EDB Postgres Extended Server 13-17

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.

Postgres clusters pending

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

project-scoped access to migrate schemas loaded by the HM agent.

- Use the

user's workspace access 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 HM agent .

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 HM 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 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 HM 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

You can create Migration Portal projects from the HM console.

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 HM 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 Schema integrity and performance considerations 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.

> **Tip**
>
> 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.

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 HM 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.

Using tools to migrate Postgres schemas

import MigratingSchemasPostgres from '../../migration_guides/snippet_sources/4_migrating_schemas/preparing_postgres_schemas.mdx'

Using the HM console to perform data migration

To perform a data migration, you can leverage the data migration features in the Hybrid Manager (HM) console. This reference material explains the options available under the **Migrate** section of the left navigation in a project.

- *Additional Resources* : view and manage migration database sources that you added with the DMS agent in reader mode.

- *Diverse migration destinations* : view and manage migration database [destinations] that you added with the DMS agent in writer mode.
- *Migrations* : view existing data migrations or start new data migrations from existing sources to existing destinations.
- *image_discovery_credentials_secret_name* : Obtain project-scoped credentials required to connect sources and destinations with the DMS agent.

Sources

Sources are databases you connected to Hybrid Manager (HM) 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.
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. Look at *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 Hybrid Manager (HM) 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. Look at the 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 Hybrid Manager (HM) project 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 you create one or more 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 on the **Sources** page.
- **Destination** — Name of the destination, either a database in an HM-managed cluster or a destination configured through the **Destinations** page.
- **Status** — Current status of the migration.

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).
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. Look at 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 displays a list of all tables selected for the migration. For each table, you can view the source and destination table names, schemas, the snapshot status, and the count of columns selected. The status reflects the overall migration state for snapshot-only jobs or the snapshot phase status for snapshot + streaming jobs. You can expand each row to view the detailed column-level mapping from the source to the destination.

Creating a migration

To create migrations, select **Create Migration**. See Creating migration for an example.

Mapping schema, tables, and columns during a migration

When you create a migration with the HM console, the Data Migration Service (DMS) allows you to map your source schema and table and column names to different destination schemas, tables, and columns. This is particularly useful for restructuring your database during a migration, so you can align table names with new naming conventions or improve clarity.

Prerequisites

The DMS requires that all destination objects exist in the destination database before the data migration begins. Suppose 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`). Manually create or rename the destination object (`location`) in the destination database before starting 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 a 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 objects with different names, the structure of the destination object must match the source data.

Object mapping

1. In the HM console, select **Migrate > Migrations > Create Migration**. See Creating migration for more details.
2. In the **Create Migrations** window, select the **Tables** tab for your migration.
3. To map a schema to a renamed destination schema:
4. Find a table in 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 in the list (`<schema>.<table>`) that 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>`).
13. Find the column you want 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

You need credentials to perform migrations. Before you can add a source or destination for a migration to Hybrid Manager (HM), you must create migration credentials. You can then use them to allow the EDB DMS agent to connect to HM.

The **Credentials** view shows all the credentials in the project in a table format. The entry for each credential includes its name, status, when it was created, and expiry date.

Creating migration credentials

See Getting credentials to create and download migration credentials.

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,
    "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.

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
```

5.3.7 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.

Managing HM Postgres and model assets

- *Asset library*

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.

- *How do I create knowledge bases for RAG?*

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 *How do I create knowledge bases for RAG?* 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* 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.

5.3.8 Backups and restores

The fundamental tasks of backing up and restoring are:

- *HM backups and DR*
- *Backups and restores*

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).

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 HA/DR 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.

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.

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:

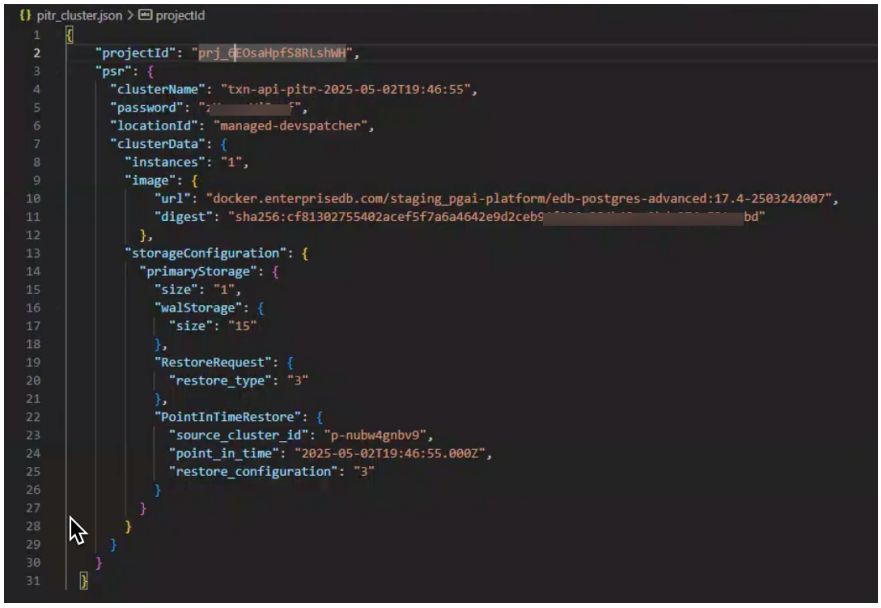
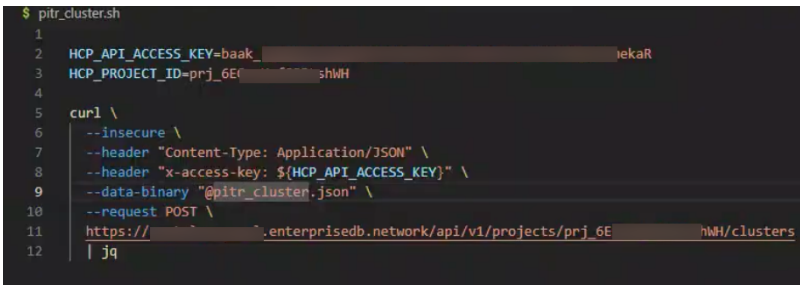


Fig. 8: PGAIHM Restore via API JSON file

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:

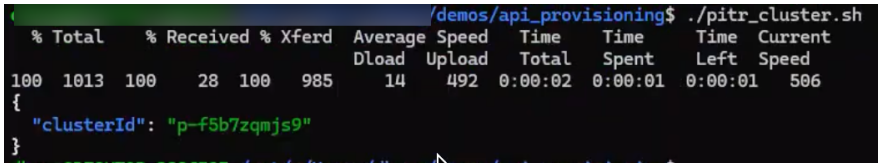


Fig. 9: PGAIHM Restore via API shell file conf

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:

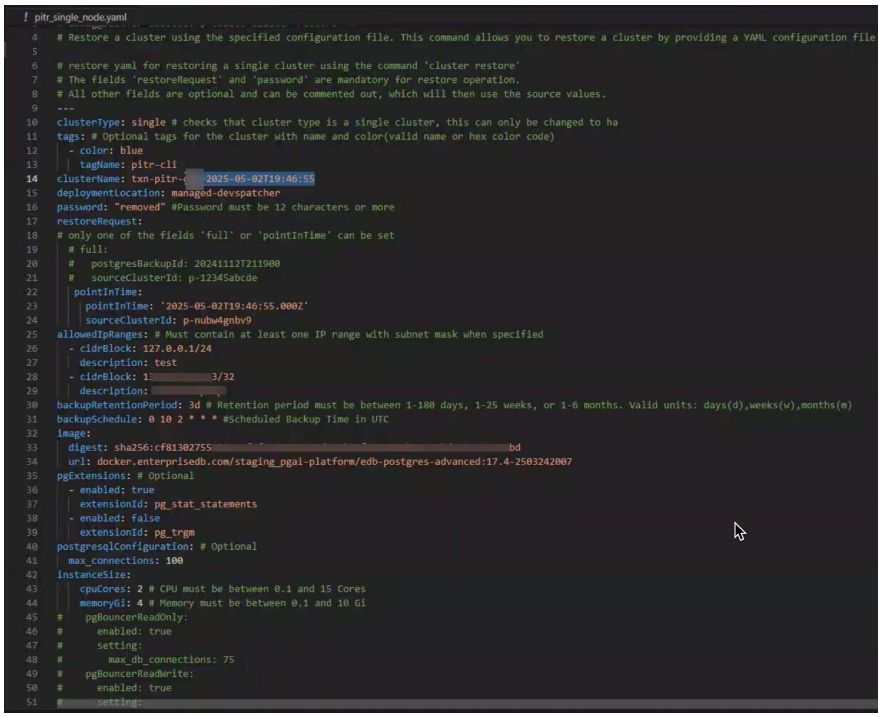


Fig. 10: HM Restore via CLI YAML file

> **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:



Fig. 11: PGAIHM Restore via CLI YAML file

5.3.9 Hybrid Manager CLI - edbctl

edbctl is a command line tool (CLI) to manage Hybrid Manager (HM) resources and perform database cluster operations.

Prerequisites

Hybrid Manager CLI is available for most Linux distributions. To use edbctl, you must install the package from the `pgai-platform` repository.

Obtain your EDB subscription token

The `pgai-platform` repository requires a token to enable downloads. To obtain your token, log in to EDB Repos 2.0 . If this is your first time visiting the EDB Repos 2.0 page, you must select **Request Access** to generate your token. Once a generated token is available, select the **Copy** icon to copy it to your clipboard or the eye icon to view it.

Set the `EDB_SUBSCRIPTION_TOKEN` environment variable

Once you have the token, execute the command shown for your operating system, substituting your token for `<your-token>` .

```
export EDB_SUBSCRIPTION_TOKEN=<your-token>
```

Install from the pgai-platform repository

Run the appropriate commands for your operating system.

Debian or Ubuntu

On Debian or Ubuntu, you can install HM CLI using the `apt` package manager.

```
curl -1sSLf "https://downloads.enterprisedb.com/$EDB_SUBSCRIPTION_TOKEN/pgai-platform/
↪setup.deb.sh" | sudo -E bash
```

If this command returns an error like `curl: (22) The requested URL returned error: 404`, check that you entered the correct token.

When the command is successful, you'll see output like this:

```
Executing the  setup script for the enterprisedb/pgai-platform repository ...
...
```

You can now install the `edbctl` package using the command:

```
sudo apt-get install edbctl
```

RHEL, Rocky, AlmaLinux, or Oracle Linux

On RHEL, Rocky, AlmaLinux, or Oracle Linux, you can install HM CLI using the `yum` package manager. You can also use the `dnf` package manager, which is the default package manager for RHEL 8 and later.

```
curl -1sSLf "https://downloads.enterprisedb.com/$EDB_SUBSCRIPTION_TOKEN/pgai-platform/
↪setup.rpm.sh" | sudo -E bash
```

If this command returns an error like `curl: (22) The requested URL returned error: 404`, check that you entered the correct token.

When the command is successful, you'll see output like this:

```
Executing the  setup script for the enterprisedb/pgai-platform repository ...
...
```

You can now install the `edbctl` package using the command:

```
sudo dnf install edbctl
```

```
sudo yum install edbctl
```

Configuring the HM CLI

After installing the package, configure an *Access key* using the `credential` command. These credentials allow you to access the supported HM resources and manage them using the CLI. The CLI supports various commands for managing projects, clusters, backups, credentials, and more.

Configuration example:

```
edbctl credential import-access-key --name user1 --access-key your_HM_access_key --
↪address https://YOUR_HM_INSTANCE_URL:443
edbctl config set context_project "Your Project Name"
```

Options

Use the `edbctl --help or -h` command to return the complete list of commands:

```
edbctl --help
__OUTPUT__
Usage:
  edbctl [options]
```

(continues on next page)

(continued from previous page)

```
  edbctl [command]

Management Commands:
  credential        Manage authentication credentials.
  project           Manage projects associated with the database clusters.
  cluster           Manage database cluster resources.
  backup            Manage cluster backups.

Configuration Commands:
  config            Manages configuration settings for edbctl tool
  help              Help about any command
  completion        Generate the autocompletion script for the specified shell

Setup Commands:
  image             Asset management commands
  operand           Operand image management commands
  image-pull-secret Pull Secret management commands

Options:
  -h, --help      help for edbctl
  -v, --version   version for edbctl

Use "edbctl [command] --help" for more information about a command.
```

More information

- *edbctl credential* — Manage authentication credentials.
- *edbctl image* — Asset management commands
- *edbctl image-pull-secret* — Pull secret management commands.
- *edbctl operand* — Operand image management commands.
- *edbctl project* — Manage projects associated with the database clusters.
- *edbctl cluster* — Manage database cluster resources.
- *edbctl backup* — Manage cluster backups.
- *edbctl config* — Manage configuration settings for *edbctl* tool.

edbctl backup

This utility manages cluster backups. Use the commands to create, delete, and list backups for your clusters. Use the subcommands to perform specific backup operations.

Commands

- *edbctl backup list* — Displays a list of all available backups.
- *edbctl backup create* — Creates a backup of a specified cluster.
- *edbctl backup delete* — Deletes a backup of a specified cluster.

edbctl backup list

Lists all available backups for a specified cluster. Displays detailed information, including backup ID, name, start time, location, status, and tags.

```
edbctl backup list [options]
```

Where `options` are:

```
    --cluster-id string   Cluster ID (required). The immutable unique identifier for␣
↪a cluster.
    --location string     Location ID.
- c, --credential string   The credential which you created via the `credential␣
↪create` command. The default is fetched from context_credential.
- P, --project string      The project that groups your clusters and other resources.␣
↪The default is taken from context_project.
- o, --output string       [table json yaml xml] (default "table")
- h, --help                Help for list.
```

This example shows how to list the backup of the clusters:

```
edbctl backup list
```

edbctl backup create

Creates a backup for a specified cluster. Provide the cluster ID, backup name, location ID, and optional tags to start the backup.

```
edbctl backup create [options]
```

Where `options` are:

```
    --cluster-id string   Cluster ID (required). The unique identifier for the␣
↪cluster to back up.
    --method string       Backup method (required). Specify the backup method: BARMAN_
↪CLOUD or VOLUME_SNAPSHOT.
    --name string         Backup name (required). The name of the backup to create.
    --location string     Location ID (required).
    --group-id string     PGD cluster group ID to back up. Required for PGD clusters.
    --tags strings        Assign tags to the backup in the format name=color (color␣
↪is optional).
- c, --credential string   The credential that you created via `credential create`␣
↪command. The default is fetched from `context_credential`.
- P, --project string      The project that groups your clusters and other resources,␣
↪the default is taken from `context_project`
- y, --yes                 auto-confirm all confirmations
- h, --help                help for create
```

These examples show how to create a backup of the cluster:

```
edbctl backup create --cluster-id p-8nllnskncq --name "My Backup" --method "BARMAN_
↪CLOUD" --location "managed-devspatcher"
edbctl backup create --cluster-id p-8nllnskncq --name "My Backup" --method "BARMAN_
↪CLOUD" --location "managed-devspatcher"  --tags name2=blue
```

(continues on next page)

(continued from previous page)

```
edbctl backup create --cluster-id p-8nllnskncq --name "My Backup" --method "VOLUME_
↪SNAPSHOT" --location "managed-devspatcher"  --tags name2=#FF0000
```

edbctl backup delete

Deletes a backup from a specified cluster using its backup ID. This action is irreversible, so ensure you have the correct backup ID before proceeding.

```
edbctl backup delete {--id} [options]
```

This example shows how to delete the backup:

```
edbctl backup delete --id my_backup_id --cluster-id p-8nllnskncq
```

Where `options` are:

```
- i, --id string           Backup ID (required). The immutable unique identifier for␣
↪the clusters backup.
    --cluster-id string   Cluster ID (required). The immutable unique identifier for␣
↪a cluster.
- c, --credential string   The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- P, --project string      The project that groups your clusters and other resources.␣
↪The default is taken from `context_project`.
- y, --yes                 auto-confirm all confirmations.
- h, --help                Help for delete.
```

edbctl catalog

The `edbctl catalog` command is the entry point for all operations related to managing database catalog resources within a Hybrid Manager cluster. It allows you to perform various tasks such as listing, adding, updating, and removing database catalogs.

Commands

The following commands fall under the `edbctl catalog` namespace and provide specific catalog management functionality:

- *edbctl catalog create* : Used to provision and register a new database catalog resource.
- *edbctl catalog delete* : Used to permanently remove a specified catalog.
- *edbctl catalog edit* : Used to modify the configuration or metadata of an existing catalog.
- *edbctl catalog list* : Used to display a list of all catalogs currently deployed or available in the project.

edbctl catalog create

The `edbctl catalog create` command provisions and registers a new database catalog resource within the Hybrid Manager cluster. This command supports creating both project catalogs for resource grouping and cluster catalogs specifically for EDB Postgres Distributed (PGD) and Advanced High Availability (AHA) clusters, with ability to assign custom tags for organization and metadata.

```
edbctl catalog create [options]
```

Where `options` are:

```
    --type string               Type of catalog to create. The valid values are␣
↪`project`, `pgd`, `aha`. The default is `project`  (required).
    --tags strings              Comma-separated list of tags to assign to the catalog␣
↪in `name=value` format, where `value` is optional.
    --cluster-id string         The unique ID of the cluster associated with the␣
↪catalog (required for `pgd` and `aha` cluster types).
    --datagroup-id string       The unique ID of the Data Group associated with the␣
↪catalog (required for `pgd` and `aha` cluster types).
    --database-name string      The name of the database associated with the catalog␣
↪(required for `pgd` and `aha` cluster types).
-P, --project string            The name of the project associated with the catalog␣
↪(required for `pgd` and `aha` cluster types).
-y, --yes                       Auto-confirms all prompts and confirmations, enabling␣
↪non-interactive operation.
-h, --help                      Displays help information for the `create` command.
```

This example shows how to create a default project catalog and assign a tag to it:

```
edbctl catalog create --tags env=dev
```

This example shows how to create a project catalog within a specific project:

```
edbctl catalog create --project "prj_123" --tags name1=blue --type project
```

This example shows how to create a PGD cluster catalog associated with a specific project, cluster, Data Group, and database:

```
edbctl catalog create --tags env=prod --type pgd --cluster-id "p-123" --datagroup-id
↪"p-123-a" --database-name "mydb"
```

This example shows how to create a PGD type catalog within a specified project:

```
edbctl catalog create --project "prj_123" --tags env=prod --type pgd --cluster-id "p-
↪456" --datagroup-id "p-456-b" --database-name "bdrdb"
```

edbctl catalog delete

The `edbctl catalog delete` command is used to permanently remove a specified catalog from a registered catalog resource by providing its unique identifier. This action is irreversible and should be used with caution.

This command requires the unique catalog ID to ensure the correct resource is deleted.

```
edbctl catalog delete --id [options]
```

Where `options` are:

```
    --id string                 The unique ID of the catalog to delete (required).
-P, --project string            The name of the project that groups your clusters and␣
↪other resources. The default is taken from the current configuration context␣
↪(required).
```

(continues on next page)

(continued from previous page)

```
-c, --credential string         The name of the credential to use for authentication.␣
↪The default is taken from the current configuration context (required).
-y, --yes                       Auto-confirms all prompts and confirmations, enabling␣
↪non-interactive operation.
-h, --help                      Displays help information for the `delete` command.
```

This example shows how to delete a catalog by providing its unique ID:

```
edbctl catalog delete --id 11ee9cd2-7696-11f0-800c-b7b1f62bfc07
```

This example shows how to delete a catalog by providing its unique ID using non-interactive operation:

```
edbctl catalog delete --id 11ee9cd2-7696-11f0-800c-b7b1f62bfc07 -y
```

edbctl catalog edit

The `edbctl catalog edit` command is used to modify the configuration of an existing catalog, primarily by updating its tags and other metadata.

This command requires the unique catalog ID to ensure the correct resource is edited and the mandatory `--tags` option to specify the new tags.

```
edbctl catalog edit [options]
```

Where `options` are:

```
-i, --id string                 The unique ID of the catalog to edit (required).
    --tags strings              Comma-separated list of tags to assign to the catalog␣
↪in `name=value` format, where `value` is optional.
-P, --project string            The name of the project that groups your clusters and␣
↪other resources. The default is taken from the current configuration context␣
↪(required).
-c, --credential string         The name of the credential to use for authentication.␣
↪The default is taken from the current configuration context (required).
-I, --interactive               Enables interactive mode for editing.
-y, --yes                       Auto-confirms all prompts and confirmations, enabling␣
↪non-interactive operation.
-h, --help                      Displays help information for the `edit` command.
```

Important

The *--tags* option is replacement based. Any existing tags on the catalog that are not included in the command will be removed. To maitain existing tags, you must explicitly list them along with any new tags you wish to add. To remove all tags, use the empty string *--tags ""* .

This example shows how to replace all exsiting tags with a new list (`env=blue` and `team`) for a catalog by providing its unique ID:

```
edbctl catalog edit --id 12c4-7c4a-11f0-932a-cd765vg --tags env=blue,team
```

This example shows how to remove all tags from a catalog by providing its unique ID:

```
edbctl catalog edit --id 12c4-7c4a-11f0-932a-cd765vg --tags ""
```

edbctl catalog list

The `edbctl catalog list` command is used to display a list of all registered catalogs. This command provides filtering options based on catalog type, ID, cluster information, and the scope (current project or entire estate).

By default, this command lists all catalogs within the current project context.

```
edbctl catalog list [options]
```

Where `options` are:

```
    --type string               Filter the list by catalog type. Valid values:␣
↪`project`, `pgd`.
-i, --id string                 The unique ID of a specific catalog to display.
    --cluster-id string         Filter PGD-type catalogs by a comma-separated list of␣
↪Cluster IDs.
    --datagroup-id string       Filter PGD-type catalogs by a comma-separated list of␣
↪Data Group IDs.
    --estate                    If present, lists catalogs across all projects in the␣
↪estate, rather than just the current project. You will see catalogs for projects␣
↪where they have a role.
-P, --project string            Specifies the project to query. The default project␣
↪is taken from the current configuration context.
-o, --output string             Sets the output format. The valid values are `table`␣
↪(default), `json`, `yaml`, `xml`, and `json-raw`.
-h, --help                      Displays help information for the `list` command.
```

These examples show how to use different options of `edbctl catalog list` command.

```
#  List all catalogs in the current project context:

edbctl catalog list

#  Filter the list to show only catalogs of type `project`:

edbctl catalog list --type project

#  Filter the list to show only catalogs of type `pgd`:

edbctl catalog list --type pgd

#  List the details of a single catalog by its unique ID:

edbctl catalog list --id "12c4-7c4a-11f0-932a-cd765vg"

#  List catalogs specifically in the project with ID `prj_123`:

edbctl catalog list --project "prj_123"

#  List catalogs across all projects in the entire estate:
```

(continues on next page)

(continued from previous page)

```
edbctl catalog list --estate

#  List only PGD catalogs associated with the cluster ID `p-123`:

edbctl catalog list --type pgd --cluster-id "p-123"

#  List catalogs in JSON format instead of the default table view:

edbctl catalog list --output json
```

edbctl cluster template

The `edbctl cluster-template` command group provides the necessary tools to manage pre-defined cluster templates. These templates serve as blueprints, allowing you to standardize configurations and rapidly deploy database clusters with consistent settings across the environment.

Commands

- *edbctl cluster-template create* : Used to define and register a new database cluster template.
- *edbctl cluster-template list* : Used to view all available cluster templates within your project or estate.

edbctl cluster-template create

The `edbctl cluster-template create` command allows you to define and save a database cluster configuration as a reusable template. By providing a YAML configuration file, you can standardize complex architectures—including Single, HA, PGD, and Analytics clusters—ensuring consistent deployments across your organization.

Use this command to register a new template by pointing to a local YAML file that specifies the cluster's technical requirements (CPU, memory, storage, backups, and Postgres configurations).

```
edbctl cluster-template create [options]
```

Where `options` are:

```
-F, --config-file string        Path to the cluster configuration YAML file␣
↪(required). For more details, see [Configuration file structure](#configuration-
↪file-structure).
-n, --name string               Name of the cluster template (required).
-d, --description string        A brief summary of the templates purpose or␣
↪configuration.
-P, --project string            The project context. Defaults to `context_project`.
-c, --credential string         The name of the credential to use for authentication.␣
↪The default is taken from the current configuration context (required).
-y, --yes                       Auto-confirms all prompts and confirmations, enabling␣
↪non-interactive operation.
-h, --help                      Displays help information for the `create` command.
```

This example shows how to create a cluster template using a sample YAML configuration file:

Sample YAML configuration file for single node:

```
#  cluster details

clusterType: single # Required. Type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-single-name # Required.
password: password@1234 # Required. Password must be 12 characters or more

#  primary details

primaryPgName: Primary-Replica
imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
deploymentLocation: managed-devspatcher # Required.
networking: private # Required. Network access type for the cluster. Valid values:␣
↪private, public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days, 1-
↪25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
# image:  # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize: # Required.
  cpuCores: 1 # CPU must be between 0.1 and 15 Cores
  memoryGi: 1 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage: # Required.
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
  walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
# transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
```

(continues on next page)

(continued from previous page)

```
#   passphrase: mypassphrase

#   keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

#  replica details

replicaClusters:
  - replicaPgName: replica-1
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #      url: docker.example.com/platform/example/postgres:latest
    #      digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
  - replicaPgName: replica-2 # Required.
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
```

(continues on next page)

(continued from previous page)

```
    #       - cidrBlock: 127.0.0.1/24
    #         description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #     url: docker.example.com/platform/example/postgres:latest
    #     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
```

This example shows how to create High Availability cluster template using sample YAML files:

Sample YAML configuration file for HA cluster:

```
#  cluster details

clusterType: ha # Required. Type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-ha-name # Required.
password: password@1234 # Required. Password must be 12 characters or more

#  primary details

primaryPgName: Primary-Replica
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
standbyReplicas: 2 # Required for HA cluster
deploymentLocation: managed-devspatcher # Required.
networking: private # Required. Network access type for the cluster. Valid values:␣
↪private, public
```

(continues on next page)

(continued from previous page)

```
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days, 1-
↪25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize: # Required.
  cpuCores: 1 # CPU must be between 0.1 and 15 Cores
  memoryGi: 1 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage: # Required.
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
  walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
# transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.

#   passphrase: mypassphrase

#   keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

#  replica details

replicaClusters:
  - replicaPgName: replica-1
    standbyReplicas: 2 # Required for HA cluster
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
```

(continues on next page)

(continued from previous page)

```
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #      url: docker.example.com/platform/example/postgres:latest
    #      digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
  - replicaPgName: replica-2 # Required.
    standbyReplicas: 2 # Required for HA cluster
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #     url: docker.example.com/platform/example/postgres:latest
    #     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
```

(continues on next page)

(continued from previous page)

```
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
```

This example shows how to create an analytics cluster template using a sample YAML configuration file:

Sample YAML configuration file for Analytics:

```
clusterType: analytics # required, type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
clusterName: create-analytics-name # required
deploymentLocation: managed-devspatcher # required
password: password@1234 #Password must be 12 characters or more
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days, 1-
↪25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize: # Required
    cpuCores: 1 # CPU must be between 0.1 and 15 Cores
    memoryGi: 1 # Memory must be between 0.1 and 10 Gi
```

This example shows how to create a PGD cluster template using a sample YAML configuration file:

Sample YAML configuration file for PGD:

```
clusterType: pgd # Type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-pgd-name # required
password: Password@1234 #Password must be 12 characters or more
dataGroups:
  - groupName: Data Group A # required
    deploymentLocation: managed-devspatcher # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#     allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#       - cidrBlock: 9.9.9.9/28

#         description: Allow traffic from App A

#       - cidrBlock: 10.10.10.10/27

#         description: Allow traffic from App B

    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 3 # Required. Only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration:
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 27 26 0   # Required. Scheduled Backup Cron
    instanceSize: # Required
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage: # Required
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For PGD␣
↪clusters, the recommended size is 10 Gi or more.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
```

(continues on next page)

(continued from previous page)

```
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional, WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field

#     transparentDataEncryption: # Optional. Only one of passphrase or KeyId should␣
↪be set, adds a layer of security by encrypting supported data stored at rest.

#       passphrase: mypassphrase

#       keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

  - groupName: Data Group B
    deploymentLocation: managed-devspatcher # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#     allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#       - cidrBlock: 9.9.9.9/28

#         description: Allow traffic from App A

#       - cidrBlock: 10.10.10.10/27

#         description: Allow traffic from App B

    backupRetentionPeriod: 30d  # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 3 # only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration:
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 6 27 0   # Required. Scheduled Backup Cron
    instanceSize: # Required
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
```

(continues on next page)

(continued from previous page)

```
      databaseStorage: # Required
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For PGD␣
↪clusters, the recommended size is 10 Gi or more.
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage: # Optional field, but default value is 1 Gi
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field

#     transparentDataEncryption: # Optional. Only one of passphrase or KeyId should␣
↪be set, adds a layer of security by encrypting supported data stored at rest.

#       passphrase: mypassphrase

#       keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

witnessGroups:
  - groupName: Witness Group  # required
    deploymentLocation: managed-devspatcher # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    instanceSize:
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
```

This example shows how to create an advanced high availability (AHA) cluster template using a sample YAML configuration file:

Sample YAML configuration file for AHA cluster:

```
clusterType: aha # required, type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-aha-name # required
password: Password@1234 # Password must be 12 characters or more
dataGroup:
  deploymentLocation: managed-devspatcher # required
  groupName: Data Group A # required
  networking: private # Network access type for the cluster. Valid values: private,␣
```

(continues on next page)

(continued from previous page)

```
↪public

#   allowedIpRanges: # Optional, must contain at least one IP range with subnet mask␣
↪when specified

#     - cidrBlock: 9.9.9.9/28

#       description: Allow traffic from App A

#     - cidrBlock: 10.10.10.10/27

#       description: Allow traffic from App B

  backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days,␣
↪1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
  nodes: 3 # Required. Only can be set to 1, 2 or 3
  imageId: e791cfd57486281653d3d6135872125c # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set

#   image: # only one of `imageId` or `image` should be set

#     url: docker.example.com/platform/example/postgres:latest

#     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

  postgresqlConfiguration:
    max_connections: 100
    max_locks_per_transaction: 64
  backupSchedule: 27 26 0   # Required. Scheduled Backup Cron
  instanceSize: # Required
      cpuCores: 2 # CPU must be between 0.1 and 47.51 Cores
      memoryGi: 2 # Memory must be between 0.1 and 178.51 Gi. For Aha clusters, the␣
↪recommended memory is 2 Gi or more.
  storage:
    databaseStorage: # Required
      sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For Aha␣
↪clusters, the recommended size is 10 Gi or more.
      storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    walStorage:
      sizeGi: 20 # Size must be between 0 and Unlimited (0-Unlimited Gi)
      storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field

#   transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.

#     passphrase: mypassphrase

#     keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
```

Configuration file structure

The configuration file supports several cluster types. Below are the core parameters for each type:

Cluster types and architectures

- **Single** : A single-node database cluster.
- **HA** : A high-availability cluster with standby replicas.
- **PGD** : EDB Postgres Distributed cluster for multi-master or multi-region setups.
- **Analytics** : A cluster optimized for analytics workloads.

Key configuration parameters

- `clusterType` : Specifies the type of cluster - `Single` , `HA` , `PGD` , or `Analytics` (required).
- `instanceSize` : CPU Cores (0.1-15) and Memory in Gi (0.1-10)(required).
- `storage` : Database storage size and `storageClass` like `ssd-csi` (required).
- `backupSchedule` : A cron expression for automated backups (required).
- `imageId` : The image UUID for the database cluster.
- `postgresqlConfiguration` : Key-value pairs for `postgresql.conf` .

edbctl cluster-template list

The `edbctl cluster-template list` command provides a way to view and inspect your saved database cluster blueprints. You can use this command for high-level discovery of available templates, deep inspection of a template's specifications, or to export a template's configuration back into YAML format.

```
edbctl cluster-template list [--id [--config-yaml]] [options]
```

Where `options` are:

```
- i, --id string              The unique ID of the cluster template to inspect.
- n, --name string          Filter the list by a specific template name.
- y, --config-yaml          Prints the template's configuration in a raw YAML format␣
↪(used with `--id`).
- o, --output string       Format options: table (default), json, yaml, xml, json-raw.
- P, --project  -P string     The project context. Defaults to `context_project`.
- -credential   -c  string  Authentication credential name. Defaults to `context_
↪credential`.
- -help -h                  Displays help for the `list` command.
```

These examples show how to list all cluster templates in the current project context:

```
#  List all templates:

edbctl cluster-template list

#  List templates filtered by name:

edbctl cluster-template list --name "prod-standard"

#  List templates using ID:

```

(continues on next page)

(continued from previous page)

```
edbctl cluster-template list --id "p-c5fh47nfjcg4ci71uv32"

#  Export the list of templates in YAML format:

edbctl cluster-template list --id "p-c5fh47nfjcg4ci71uv32" --config-yaml
```

edbctl cluster

Manages database cluster resources. Use this command to create, edit, view, delete, and restore all types of clusters.

Commands

The `edbctl cluster` command has following options:

- *edbctl cluster list* — Displays active or deleted clusters.
- *edbctl cluster list-connection-info* — Lists the connection string for a specified cluster.
- *edbctl cluster create* — Creates a new cluster.
- *edbctl cluster edit* — Edits an existing cluster.
- *edbctl cluster delete* — Deletes a specified cluster.
- *edbctl cluster restore* — Restores a specified cluster
- *edbctl cluster switchover-primary* — Switches over a specified replica to primary.

edbctl cluster list

Lists all active or deleted clusters. Use the `[--deleted]` flag to display deleted clusters. You can also filter results by cluster ID or architecture. When you provide the `[--id]` flag, detailed information about the cluster is displayed, including instance size, storage, and node count.

```
edbctl cluster list [--id] [--deleted] [options]
```

Where `options` are:

```
- i, --id string                    Cluster ID. Filter the list by a specific␣
↪cluster ID.
- n, --name string                  Cluster name. Filter the list by a specific␣
↪cluster name.
- d, --deleted                      The cluster is a deleted cluster.
- A, --cluster-architecture string  Cluster architecture ID. Filter the list by␣
↪architecture (for example, single/ha).
- o, --output string                table json yaml xml json-raw (default "table").
- c, --credential string            The credential that you created via `credential␣
↪create` command. The default is fetched from `context_credential`
- P, --project string               The project that groups your clusters and other␣
↪resources. The default is taken from `context_project`.
- h, --help                         Help for list.
```

Examples

```
edbctl cluster list
edbctl cluster list --id p-c5fh47nfjcg4ci71uv32
edbctl cluster list --deleted
```

edbctl cluster list-connection-info

Retrieves and displays the connection string for a specified Postgres cluster using its cluster ID. The output includes the read/write URI, database name, host, port, and user.

```
edbctl cluster list-connection-info {--id} [options]
```

Where `options` are:

```
-i, --id string           Cluster ID (required). The unique identifier for the␣
↪cluster to list connection string.
-o, --output string       [table json yaml xml json-raw] (default "table")
-c, --credential string   The credential that you created via `credential create`␣
↪command. The default is fetched from `context_credential`.
-P, --project string      The project that groups your clusters and other resources.␣
↪the default is taken from `context_project`.
-h, --help                Help for list-connection-info.
```

These examples show how to retrieve and display the connection strings for specific clusters:

```
edbctl cluster list-connection-info --id <pg_id>
edbctl cluster list-connection-info -i <pg_id>
```

edbctl cluster create

Creates a cluster using a specified configuration file.

This command provisions a new cluster based on the settings defined in a YAML configuration file. The file specifies key details such as:

- **Cluster Type**: single, ha, pgd, aha, or analytics
- **PostgreSQL Configuration**: pgConfig
- **Resource Allocation**: storage, memory
- **Backup Settings**
- **Image**

```
edbctl cluster create [options]
```

Where `options` are:

```
- F, --config-file string   Path to the cluster config YAML file.
- c, --credential string    The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- h, --help                 Help for create.
- P, --project string       The project that groups your clusters and other resources.
↪ The default is taken from `context_project`.
- y, --yes                  Auto-confirm all confirmations.
```

This example shows how to create a PSR - Single Node cluster using sample YAML files:

```
edbctl cluster create -F <Sample YAML Configuration file for single node>
```

Sample YAML configuration file for single node:

```
#  cluster details

clusterType: single # Required. Type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-single-name # Required.
password: password@1234 # Required. Password must be 12 characters or more

#  primary details

primaryPgName: Primary-Replica
imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
deploymentLocation: managed-devspatcher # Required.
networking: private # Required. Network access type for the cluster. Valid values:␣
↪private, public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days, 1-
↪25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
# image:  # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize: # Required.
  cpuCores: 1 # CPU must be between 0.1 and 15 Cores
  memoryGi: 1 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage: # Required.
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
  walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
```

(continues on next page)

(continued from previous page)

```
    storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
# transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.

#   passphrase: mypassphrase

#   keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

#  replica details

replicaClusters:
  - replicaPgName: replica-1
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #      url: docker.example.com/platform/example/postgres:latest
    #      digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
  - replicaPgName: replica-2 # Required.
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
```

(continues on next page)

(continued from previous page)

```
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #     url: docker.example.com/platform/example/postgres:latest
    #     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
```

This example shows how to create a PSR - High Availability cluster using a YAML configuration file:

```
edbctl cluster create -F <Sample YAML configuration file for High Availability>
```

Sample YAML Configuration file for High Availability:

```
#  cluster details

clusterType: ha # Required. Type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-ha-name # Required.
password: password@1234 # Required. Password must be 12 characters or more

#  primary details
```

(continues on next page)

(continued from previous page)

```
primaryPgName: Primary-Replica
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
standbyReplicas: 2 # Required for HA cluster
deploymentLocation: managed-devspatcher # Required.
networking: private # Required. Network access type for the cluster. Valid values:␣
↪private, public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days, 1-
↪25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize: # Required.
  cpuCores: 1 # CPU must be between 0.1 and 15 Cores
  memoryGi: 1 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage: # Required.
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
  walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
# transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.

#   passphrase: mypassphrase

#   keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

#  replica details

replicaClusters:
  - replicaPgName: replica-1
```

(continues on next page)

(continued from previous page)

```
    standbyReplicas: 2 # Required for HA cluster
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #      url: docker.example.com/platform/example/postgres:latest
    #      digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
  - replicaPgName: replica-2 # Required.
    standbyReplicas: 2 # Required for HA cluster
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
```

(continues on next page)

(continued from previous page)

```
    #     url: docker.example.com/platform/example/postgres:latest
    #     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
```

This example shows how to create an analytics cluster using a sample YAML configuration file:

```
edbctl cluster create -F <Sample YAML configuration file for Analytics>
```

Sample YAML configuration file for Analytics:

```
clusterType: analytics # required, type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
clusterName: create-analytics-name # required
deploymentLocation: managed-devspatcher # required
password: password@1234 #Password must be 12 characters or more
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days, 1-
↪25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
```

(continues on next page)

(continued from previous page)

```
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize: # Required
    cpuCores: 1 # CPU must be between 0.1 and 15 Cores
    memoryGi: 1 # Memory must be between 0.1 and 10 Gi
```

This example shows how to create a PGD cluster using a sample YAML configuration file:

```
edbctl cluster create -F <Sample YAML configuration file for PGD>
```

Sample YAML configuration file for PGD:

```
clusterType: pgd # Type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-pgd-name # required
password: Password@1234 #Password must be 12 characters or more
dataGroups:
  - groupName: Data Group A # required
    deploymentLocation: managed-devspatcher # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#     allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#       - cidrBlock: 9.9.9.9/28

#         description: Allow traffic from App A

#       - cidrBlock: 10.10.10.10/27

#         description: Allow traffic from App B

    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 3 # Required. Only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest
```

(continues on next page)

(continued from previous page)

```
#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration:
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 27 26 0   # Required. Scheduled Backup Cron
    instanceSize: # Required
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage: # Required
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For PGD␣
↪clusters, the recommended size is 10 Gi or more.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional, WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field

#     transparentDataEncryption: # Optional. Only one of passphrase or KeyId should␣
↪be set, adds a layer of security by encrypting supported data stored at rest.

#       passphrase: mypassphrase

#       keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

  - groupName: Data Group B
    deploymentLocation: managed-devspatcher # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#     allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#       - cidrBlock: 9.9.9.9/28

#         description: Allow traffic from App A

#       - cidrBlock: 10.10.10.10/27

#         description: Allow traffic from App B

    backupRetentionPeriod: 30d  # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 3 # only can be set to 2 or 3
```

(continues on next page)

(continued from previous page)

```
#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration:
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 6 27 0   # Required. Scheduled Backup Cron
    instanceSize: # Required
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage: # Required
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For PGD␣
↪clusters, the recommended size is 10 Gi or more.
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage: # Optional field, but default value is 1 Gi
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field

#     transparentDataEncryption: # Optional. Only one of passphrase or KeyId should␣
↪be set, adds a layer of security by encrypting supported data stored at rest.

#       passphrase: mypassphrase

#       keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

witnessGroups:
  - groupName: Witness Group  # required
    deploymentLocation: managed-devspatcher # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    instanceSize:
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
```

This example shows how to create an advanced high availability (AHA) cluster using a sample YAML configuration file:

```
edbctl cluster create -F <Sample YAML configuration file for AHA>
```

Sample YAML configuration file for AHA cluster:

```
clusterType: aha # required, type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-aha-name # required
password: Password@1234 # Password must be 12 characters or more
dataGroup:
  deploymentLocation: managed-devspatcher # required
  groupName: Data Group A # required
  networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#   allowedIpRanges: # Optional, must contain at least one IP range with subnet mask␣
↪when specified

#     - cidrBlock: 9.9.9.9/28

#       description: Allow traffic from App A

#     - cidrBlock: 10.10.10.10/27

#       description: Allow traffic from App B

  backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days,␣
↪1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
  nodes: 3 # Required. Only can be set to 1, 2 or 3
  imageId: e791cfd57486281653d3d6135872125c # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set

#   image: # only one of `imageId` or `image` should be set

#     url: docker.example.com/platform/example/postgres:latest

#     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

  postgresqlConfiguration:
    max_connections: 100
    max_locks_per_transaction: 64
  backupSchedule: 27 26 0   # Required. Scheduled Backup Cron
  instanceSize: # Required
      cpuCores: 2 # CPU must be between 0.1 and 47.51 Cores
      memoryGi: 2 # Memory must be between 0.1 and 178.51 Gi. For Aha clusters, the␣
↪recommended memory is 2 Gi or more.
  storage:
    databaseStorage: # Required
```

(continues on next page)

(continued from previous page)

```
      sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For Aha␣
↪clusters, the recommended size is 10 Gi or more.
      storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    walStorage:
      sizeGi: 20 # Size must be between 0 and Unlimited (0-Unlimited Gi)
      storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field

#   transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.

#     passphrase: mypassphrase

#     keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
```

edbctl cluster edit

Edits an existing cluster using a YAML configuration file. This command allows you to modify cluster settings, database configurations, backup policies, extensions, and resource allocations.

```
edbctl cluster edit [options]
```

Where `options` are:

```
- F, --config-file string   Path to the cluster config YAML file.
- c, --credential string    The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- h, --help                 Help for edit.
- P, --project string       The project that groups your clusters and other resources.
↪ The default is taken from `context_project`.
- y, --yes                  Auto-confirm all confirmations.
```

This example shows how to edit a cluster using a YAML configuration file:

> **Note**
>
> The following fields are mandatory: - *clusterID* - *clusterType* All the other fields are optional. If not provided, the values will not be changed.

```
edbctl cluster edit -F <Example YAML configuration file>
```

Example YAML configuration file:

```
clusterType: single # checks that cluster type is a single cluster, this can only be␣
↪changed to ha
clusterId: p-abcdefg # cluster id of the cluster to be edited
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to preserve existing tags. tags values will overwrite existing␣
↪tags. set tags: [] to remove all tags.
```

(continues on next page)

(continued from previous page)

```
  - color: blue
    tagName: production
  - color: green
    tagName: staging
  - color: red
    tagName: development
name: edit-single-name
primaryPgName: edited-primary-replica-name
imageId: 6535c8bc9e358f0afaf5a706ea1bc722 # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
password: password@1234 #Password must be 12 characters or more
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: "192.168.1.0/24"

#     description: "Office network"

#   - cidrBlock: "10.0.0.0/16"

#     description: "VPN network"

backupRetentionPeriod: 40d # Retention period must be between 1-180 days, 1-25 weeks,␣
↪or 1-6 months. Valid units: days(d),weeks(w),months(m)
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 500
  shared_buffers: 256
  application_name: beaconator
backupSchedule: 0 6     # Scheduled Backup Time in Cron format (UTC)
instanceSize:
  cpuCores: 4 # CPU must be between 0.1 and 15 Cores
  memoryGi: 16 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage:  # Primary Storage size in Gi
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size needs to␣
↪be the same or greater than the existing databaseStorage size.
    storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
  walStorage:   # Optional field. WAL Storage size in Gi, a separate storage volume␣
↪for Write-Ahead Logs (Recommended for high write workloads) (Optional)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size needs to␣
↪be the same or greater than the existing walStorage size.
    storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
```

(continues on next page)

(continued from previous page)

```
#  to edit: if you provide a replica with a replicaPgId then that existing replica␣
↪will be preserved/edited

#  to add a replica: if you do provide a replica without a replicaPgId then a new␣
↪replica will be created

#  to remove a replica: if you do not provide a replica with a replicaPgId that␣
↪exists then that existing replica will be removed

replicaClusters:
  # this is editing a replica cluster because it provides a replicaPgId
  - replicaPgId: p-abcdefg-b
    replicaPgName: test replica d
    imageId: 6535c8bc9e358f0afaf5a706ea1bc722 # Image ID can be fetched with command␣
↪`edbctl image list-image-tags --location-id <location-id>`, only one of `imageId`␣
↪or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public
    #allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #  - cidrBlock: "192.168.1.0/24"
    #    description: "Office network"
    #  - cidrBlock: "10.0.0.0/16"
    #    description: "VPN network"
    backupRetentionPeriod: 40d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    #image: # only one of `imageId` or `image` should be set
    #  url: docker.example.com/platform/example/postgres:latest
    #  digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 500
      shared_buffers: 256
      application_name: beaconator
    backupSchedule: 0 6     # Scheduled Backup Time in Cron format (UTC)
    instanceSize:
      cpuCores: 4 # CPU must be between 0.1 and 15 Cores
      memoryGi: 16 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage:  # Primary Storage size in Gi
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing databaseStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional field. WAL Storage size in Gi, a separate storage␣
↪volume for Write-Ahead Logs (Recommended for high write workloads) (Optional)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing walStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
  # this is adding a new replica because it does not provide a replicaPgId
```

(continues on next page)

(continued from previous page)

```
  - replicaPgName: test replica e
    locationId: managed-devspatcher
    imageId: 6535c8bc9e358f0afaf5a706ea1bc722 # Image ID can be fetched with command␣
↪`edbctl image list-image-tags --location-id <location-id>`, only one of `imageId`␣
↪or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public
    #allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #  - cidrBlock: "192.168.1.0/24"
    #    description: "Office network"
    #  - cidrBlock: "10.0.0.0/16"
    #    description: "VPN network"
    backupRetentionPeriod: 40d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    #image: # only one of `imageId` or `image` should be set
    #  url: docker.example.com/platform/example/postgres:latest
    #  digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 500
      shared_buffers: 256
      application_name: beaconator
    backupSchedule: 0 6      # Scheduled Backup Time in Cron format (UTC)
    instanceSize:
      cpuCores: 4 # CPU must be between 0.1 and 15 Cores
      memoryGi: 16 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage:  # Primary Storage size in Gi
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing databaseStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:    # Optional field. WAL Storage size in Gi, a separate storage␣
↪volume for Write-Ahead Logs (Recommended for high write workloads) (Optional)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing walStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
  # any replicas that are not provided with a replicaPgId will be removed
```

This example shows how to edit High Availability cluster using sample YAML files:

> **Note**
>
> The following fields are mandatory:

- `clusterID`
- `clusterType`

All the other fields are optional. If not provided, the values will not be changed.

```
clusterType: ha # checks that cluster type is a ha cluster, this can only be changed
↪to single
clusterId: p-abcdefg # cluster id of the cluster to be edited
tags: # Optional tags for the cluster with name and color(valid name or hex color
↪code). Remove tags to preserve existing tags. tags values will overwrite existing
↪tags. set tags: [] to remove all tags.
  - color: blue
    tagName: production
  - color: green
    tagName: staging
  - color: red
    tagName: development
name: edit-ha-name
imageId: 6535c8bc9e358f0afaf5a706ea1bc722 # Image ID can be fetched with command
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
password: password@1234 #Password must be 12 characters or more
networking: private # Network access type for the cluster. Valid values: private,
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when
↪specified

#   - cidrBlock: "192.168.1.0/24"

#     description: "Office network"

#   - cidrBlock: "10.0.0.0/16"

#     description: "VPN network"

backupRetentionPeriod: 40d # Retention period must be between 1-180 days, 1-25 weeks,
↪or 1-6 months. Valid units: days(d),weeks(w),months(m)
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 500
  shared_buffers: 256
  application_name: beaconator
backupSchedule: 0 6     # Scheduled Backup Time in Cron format (UTC)
instanceSize:
  cpuCores: 4 # CPU must be between 0.1 and 15 Cores
  memoryGi: 16 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage:  # Primary Storage size in Gi
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size needs to
↪be the same or greater than the existing databaseStorage size.
    storageClass: ssd-csi # Supported storage classes can be fetched with command
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
  walStorage:   # Optional field. WAL Storage size in Gi, a separate storage volume
↪for Write-Ahead Logs (Recommended for high write workloads) (Optional)
```

(continues on next page)

(continued from previous page)

```
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size needs to
↪be the same or greater than the existing walStorage size.
    storageClass: ssd-csi # Supported storage classes can be fetched with command
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field

#  to edit: if you provide a replica with a replicaPgId then that existing replica
↪will be preserved/edited

#  to add a replica: if you do provide a replica without a replicaPgId then a new
↪replica will be created

#  to remove a replica: if you do not provide a replica with a replicaPgId that
↪exists then that existing replica will be removed

standbyReplicas: 3          # for HA cluster
replicaClusters:
  # this is editing a replica cluster because it provides a replicaPgId
  - replicaPgId: p-abcdefg-b
    replicaPgName: test replica d
    imageId: 6535c8bc9e358f0afaf5a706ea1bc722 # Image ID can be fetched with command
↪`edbctl image list-image-tags --location-id <location-id>`, only one of `imageId`
↪or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,
↪public
    #allowedIpRanges: # Must contain at least one IP range with subnet mask when
↪specified
    #  - cidrBlock: "192.168.1.0/24"
    #    description: "Office network"
    #  - cidrBlock: "10.0.0.0/16"
    #    description: "VPN network"
    backupRetentionPeriod: 40d # Retention period must be between 1-180 days, 1-25
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    #image: # only one of `imageId` or `image` should be set
    #  url: docker.example.com/platform/example/postgres:latest
    #  digest:
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 500
      shared_buffers: 256
      application_name: beaconator
    backupSchedule: 0 6     # Scheduled Backup Time in Cron format (UTC)
    instanceSize:
      cpuCores: 4 # CPU must be between 0.1 and 15 Cores
      memoryGi: 16 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage:  # Primary Storage size in Gi
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size
↪needs to be the same or greater than the existing databaseStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional field. WAL Storage size in Gi, a separate storage
↪volume for Write-Ahead Logs (Recommended for high write workloads) (Optional)
```

(continues on next page)

(continued from previous page)

```
      sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size
↪needs to be the same or greater than the existing walStorage size.
      storageClass: ssd-csi # Supported storage classes can be fetched with command
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
   standbyReplicas: 3          # for HA cluster
 # this is adding a new replica because it does not provide a replicaPgId
 - replicaPgName: test replica e
   locationId: managed-devspatcher
   imageId: 6535c8bc9e358f0afaf5a706ea1bc722 # Image ID can be fetched with command
↪`edbctl image list-image-tags --location-id <location-id>`, only one of `imageId`
↪or `image` should be set
   networking: private # Network access type for the cluster. Valid values: private,
↪public
   #allowedIpRanges: # Must contain at least one IP range with subnet mask when
↪specified
   #  - cidrBlock: "192.168.1.0/24"
   #    description: "Office network"
   #  - cidrBlock: "10.0.0.0/16"
   #    description: "VPN network"
   backupRetentionPeriod: 40d # Retention period must be between 1-180 days, 1-25
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
   #image: # only one of `imageId` or `image` should be set
   #  url: docker.example.com/platform/example/postgres:latest
   #  digest:
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
   postgresqlConfiguration: # Optional
     max_connections: 500
     shared_buffers: 256
     application_name: beaconator
   backupSchedule: 0 6     # Scheduled Backup Time in Cron format (UTC)
   instanceSize:
     cpuCores: 4 # CPU must be between 0.1 and 15 Cores
     memoryGi: 16 # Memory must be between 0.1 and 10 Gi
   storage:
     databaseStorage:  # Primary Storage size in Gi
       sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size
↪needs to be the same or greater than the existing databaseStorage size.
       storageClass: ssd-csi # Supported storage classes can be fetched with command
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
     walStorage:   # Optional field. WAL Storage size in Gi, a separate storage
↪volume for Write-Ahead Logs (Recommended for high write workloads) (Optional)
       sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size
↪needs to be the same or greater than the existing walStorage size.
       storageClass: ssd-csi # Supported storage classes can be fetched with command
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
   standbyReplicas: 3          # for HA cluster
 # any replicas that are not provided with a replicaPgId will be removed
```

This example shows how to edit an analytics cluster using a sample YAML configuration file:

Note

The following fields are mandatory:

- `clusterID`
- `clusterType`

All the other fields are optional. If not provided, the values will not be changed.

```
clusterType: analytics # checks that cluster type is an analytics cluster, cannot␣
↪change this field
clusterId: p-pd6kf9unmi # required, cluster ID of the cluster to be edited
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to preserve existing tags. tags values will overwrite existing␣
↪tags. set tags: [] to remove all tags.
  - color: blue
    tagName: production
  - color: green
    tagName: staging
  - color: red
    tagName: development
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
name: edit-analytics-name
password: password@1234 #Password must be 12 characters or more
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 192.168.1.0/24

#     description: Office network

#   - cidrBlock: 10.0.0.0/16

#     description: VPN network

backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25 weeks,␣
↪or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 0 6     # Scheduled Backup Time in Cron format (UTC)
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
  max_locks_per_transaction: 64
instanceSize:
    cpuCores: 4 # CPU must be between 0.1 and 15 Cores
    memoryGi: 16 # Memory must be between 0.1 and 10 Gi
```

This example shows how to edit a PGD cluster using a sample YAML configuration file:

> **Note**
>
> The fields *clusterType* and *clusterId* are mandatory The logic for forming the YAML file is as follows: - If you mention just *groupId* , then the group will be retained. - If you mention the *groupId* along with some fields, then the group will be retained and updated with the given values. - If you don't give a *groupId* and add other details, such as *groupName* , *image* , etc., then that group will be added. - If you don't mention any *groupId* , then the group will be deleted. !!!

Sample YAML configuration file for PGD:

```
clusterType: pgd # checks that cluster type is a pgd cluster, cannot change this field
clusterId: p-12345abcde
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to preserve existing tags. tags values will overwrite existing␣
↪tags. set tags: [] to remove all tags.
  - color: blue
    tagName: tag-1
name: edit-pgd-name
password: Password@1234
dataGroups:
  # add new data group A
  - groupName: Data Group A # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#     allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#       - cidrBlock: 9.9.9.9/28

#         description: Allow traffic from App A

#       - cidrBlock: 10.10.10.10/27

#         description: Allow traffic from App B

    backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 2 # only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration: # Optional
```

(continues on next page)

(continued from previous page)

```
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 0 4 8   # Scheduled Backup Cron
    instanceSize:
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage: # Size must be between 0 and Unlimited (0-Unlimited Gi)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For PGD␣
↪clusters, the recommended size is 10 Gi or more. Size needs to be the same or␣
↪greater than the existing databaseStorage size.
      walStorage: # Optional, WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing walStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
  - groupId: p-12345abcde-x
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public
    groupName: Data Group B # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set

#     allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#       - cidrBlock: 9.9.9.9/28

#         description: Allow traffic from App A

#       - cidrBlock: 10.10.10.10/27

#         description: Allow traffic from App B

    backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 2 # only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration: # Optional
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 0 4 8   # Scheduled Backup Cron
    instanceSize:
```

(continues on next page)

(continued from previous page)

```
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage:
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing databaseStorage size.
      walStorage: # Optional, WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing walStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
witnessGroups:
  # add new witness group
  - groupName: Witness Group A # required when adding a new witness group
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    instanceSize:
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
```

This example that shows how to edit a PGD cluster while retaining group using a sample YAML configuration file:

Second sample YAML configuration file for PGD:

```
clusterType: pgd # checks that cluster type is a pgd cluster, cannot change this field
clusterId: p-12345abcde # this will ensure the cluster will be retained
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to preserve existing tags. tags values will overwrite existing␣
↪tags. set tags: [] to remove all tags.
  - color: blue
    tagName: tag-1
name: edit-pgd-name
password: Password@1234 #Password must be 12 characters or more
dataGroups:
  - groupId: p-12345abcde-a # this will ensure the group will be retained
  - groupId: p-12345abcde-b
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
witnessGroups: # this will ensure the witness group will be retained
  - groupId: p-12345abcde-c
```

(continues on next page)

(continued from previous page)

```
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    groupName: Witness Group JFK

#     image: # only one of `imageId` or `image` should be set, should match the image␣
↪of the data group

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    instanceSize:
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
```

edbctl cluster delete

Deletes a specified cluster using its cluster ID. This action permanently removes the cluster from your project and can't be undone, so ensure you have the correct cluster ID before proceeding.

```
edbctl cluster delete [options]
```

Where `options` are:

```
- i, --id string           Cluster ID (required). The unique identifier for the␣
↪cluster to delete.
- c, --credential string   The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- y, --yes                 Auto-confirm all confirmations.
- P, --project string      The project that groups your clusters and other resources,␣
↪the default is taken from `context_project`.
- h, --help                Help for delete
```

These examples show how to delete the cluster:

```
edbctl cluster delete
edbctl cluster delete --id p-8nllnskncq
edbctl cluster delete -i p-8nllnskncq
```

edbctl cluster restore

You can restore a cluster by providing a YAML configuration file with the required specifications. Use the `cluster restore` command and pass the file using the `--file` or `-f` option. While restoring, you must specify one of the following restore modes:

- `full`
- `pointInTime`

Make sure you uncomment only one mode at a time.

> **Note**
>
> The *restoreRequest* and *password* fields are mandatory for all clusters except PGD and AHA. All other fields are optional and default to the values from the source cluster if not specified.

```
edbctl cluster restore [options]
```

Where `options` are:

```
- F, --config-file string   Path to the cluster config YAML file (required).
- c, --credential string    The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- y, --yes                  Auto-confirm all confirmations.
- P, --project string       The project that groups your clusters and other resources.
↪ The default is taken from `context_project`.
- h, --help                 Help for restore.
```

This example shows how to restore a single cluster using a YAML configuration file:

> **Note**
>
> The following fields are mandatory:

- `clusterType`
- `restoreRequest`
- `password`

All the other primary cluster fields are optional. If not provided, then will use the source values. Replica details can be added as needed, but are not mandatory.

restore-single.yaml

```
clusterType: single # checks that cluster type is a single cluster, this can only be␣
↪changed to ha
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to use source cluster tags. tags values will use exactly these␣
↪tags only. set tags: [] to use no tags.
  - color: blue
    tagName: tag-1
clusterName: restore-single-name
restoreRequest:
  # only one of the fields full or pointInTime can be set
  full:
    postgresBackupId: 20241112T211900
    sourceClusterId: p-12345abcde
  #  pointInTime:
  #    pointInTime: 2025-01-01T13:40:04.000Z
  #    sourceClusterId: p-12345abcde

#  primary replica details

primaryPgName:  Primary Cluster # Required.
```

(continues on next page)

(continued from previous page)

```
deploymentLocation: managed-devspatcher
password: password@1234 #Password must be 12 characters or more
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25 weeks,␣
↪or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23   # Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize:
    cpuCores: 1 # CPU must be between 0.1 and 15 Cores
    memoryGi: 1 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage:
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
  walStorage:   # Optional field. WAL Storage size in Gi, a separate storage volume␣
↪for Write-Ahead Logs (Recommended for high write workloads) (Optional)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
# transparentDataEncryption: # Optional. Only one of passphrase or keyId should be␣
↪set. If TDE is enabled on source cluster, configuration must match source or␣
↪provide passphrase.

#   passphrase: mypassphrase # Required if source cluster uses TDE with passphrase

#   #keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4 # Uses KMS key from source if source␣
↪cluster uses KMS

#  replica details

replicaClusters:
  - replicaPgName: replica-1
```

(continues on next page)

(continued from previous page)

```
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #      url: docker.example.com/platform/example/postgres:latest
    #      digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 1 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    transparentDataEncryption: # Optional. Only one of passphrase or keyId should be␣
↪set. If TDE is enabled on source cluster, configuration must match source or␣
↪provide passphrase.
      passphrase: mypassphrase # Required if source cluster uses TDE with passphrase
      #keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4 # Uses KMS key from source if␣
↪source cluster uses KMS
```

This example shows how to restore a high availability cluster using a YAML configuration file:

> **Note**
>
> The following fields are mandatory:

- `clusterType`
- `restoreRequest`
- `password`

All the other primary cluster fields are optional. If not provided, then will use the source values. Replica details can be added as needed, but are not mandatory.

restore-ha.yaml

```
clusterType: ha # checks that cluster type is a ha cluster, this can only be changed
↪to single
tags: # Optional tags for the cluster with name and color(valid name or hex color
↪code). Remove tags to use source cluster tags. tags values will use exactly these
↪tags only. set tags: [] to use no tags.
  - color: blue
    tagName: tag-1
clusterName: restore-ha-name
restoreRequest:
  # only one of the fields full or pointInTime can be set
  full:
    postgresBackupId: 20241112T211900
    sourceClusterId: p-12345abcde
  #  pointInTime:
  #    pointInTime: 2025-01-01T13:40:04.000Z
  #    sourceClusterId: p-12345abcde

#  primary replica details

primaryPgName:  Primary Cluster # Required.
deploymentLocation: managed-devspatcher
password: password@1234 #Password must be 12 characters or more
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
networking: private # Network access type for the cluster. Valid values: private,
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25 weeks,
↪or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23   # Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize:
    cpuCores: 1 # CPU must be between 0.1 and 15 Cores
    memoryGi: 1 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage:
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with
```

(continues on next page)

(continued from previous page)

```
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
  walStorage:   # Optional field. WAL Storage size in Gi, a separate storage volume␣
↪for Write-Ahead Logs (Recommended for high write workloads) (Optional)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
# transparentDataEncryption: # Optional. Only one of passphrase or keyId should be␣
↪set. If TDE is enabled on source cluster, configuration must match source or␣
↪provide passphrase.

#   passphrase: mypassphrase # Required if source cluster uses TDE with passphrase

#   #keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4 # Uses KMS key from source if source␣
↪cluster uses KMS

standbyReplicas: 2 # for HA cluster

#  replica details

replicaClusters:
  - replicaPgName: replica-1
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23   # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #      url: docker.example.com/platform/example/postgres:latest
    #      digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
```

(continues on next page)

(continued from previous page)

```
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field

#     transparentDataEncryption: # Optional. Only one of passphrase or keyId should␣
↪be set. If TDE is enabled on source cluster, configuration must match source or␣
↪provide passphrase.

#       passphrase: mypassphrase # Required if source cluster uses TDE with passphrase

#       #keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4 # Uses KMS key from source if␣
↪source cluster uses KMS

    standbyReplicas: 2 # Required for HA cluster
```

This example shows how to restore an analytics cluster using a YAML configuration file:

Note

The following fields are mandatory:

- `clusterType`
- `restoreRequest`
- `password`

All the other cluster fields are optional. If not provided, then will use the source values.

restore-analytics.yaml

```
clusterType: analytics # checks that cluster type is a analytics cluster, cannot␣
↪change this field
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to use source cluster tags. tags values will use exactly these␣
↪tags only. set tags: [] to use no tags.
  - color: blue
    tagName: tag-1
clusterName: restore-analytics-name
deploymentLocation: managed-devspatcher
password: password@1234 #Password must be 12 characters or more
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
restoreRequest:

#  only one of the fields full or pointInTime can be set

  full:
    postgresBackupId: 20241112T211900
    sourceClusterId: p-12345abcde
  #  pointInTime:
  #    pointInTime: 2025-01-01T13:40:04.000Z
  #    sourceClusterId: p-12345abcde
```

(continues on next page)

(continued from previous page)

```
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25 weeks,␣
↪or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23   # Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize:
    cpuCores: 1 # CPU must be between 0.1 and 15 Cores
    memoryGi: 1 # Memory must be between 0.1 and 10 Gi
```

This example shows how to define the configuration for a PGD cluster in a restore operation using a YAML configuration file:

Note

The following fields are mandatory:

- `sourceClusterID`
- `clusterType`
- `password`
- `dataGroups`
- `restorePgdGroupRequest`

All the other fields are optional. If omitted, the values from the source cluster are used.

restore-pgd.yaml

```
clusterType: pgd # checks that cluster type is a pgd cluster, cannot change this field
sourceClusterId: p-12345abcde # required
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to use source cluster tags. tags values will use exactly these␣
↪tags only. set tags: [] to use no tags.
  - color: blue
    tagName: tag-1
name: restore-pgd-name
password: Password@1234 #Password must be 12 characters or more
```

(continues on next page)

(continued from previous page)

```
dataGroups: # Currently, only one data group can be restored at a time
  - groupName: Data Group A
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    deploymentLocation: managed-devspatcher
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public
    backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 2 # only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration: # Optional
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 0 4 8   # Scheduled Backup Cron
    restorePgdGroupRequest:
      full: # for pointInTime restore mode use the pointInTime configuration as below␣
↪and comment out the full configuration
        postgresBackupId: 20241228T055553
        sourceGroupId: p-12345abcde-b
      # pointInTime:
      #   pointInTime: 2025-01-01T13:40:04.000Z
      #   sourceGroupId: p-12345abcde-b
    instanceSize:
        cpuCores: 1 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage:
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For PGD␣
↪clusters, the recommended size is 10 Gi or more.
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
      walStorage:   # Optional, WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    transparentDataEncryption: # Optional. Only one of passphrase or keyId should be␣
↪set. If TDE is enabled on source cluster, configuration must match source or␣
↪provide passphrase.
      passphrase: mypassphrase # Required if source cluster uses TDE with passphrase

#       keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4 # Uses KMS key from source if␣
```

(continues on next page)

(continued from previous page)

```
↪source cluster uses KMS
```

This example shows how to define the configuration for AHA cluster in restore operation using a YAML configuration file:

> **Note**
>
> The following fields are mandatory:

- `sourceClusterID`
- `clusterType`
- `password`
- `dataGroups`
- `restoreAhagroupRequest`

All the other fields are optional. If omitted, the values from the source cluster are used.

restore-aha.yaml

```
clusterType: aha # checks that cluster type is a aha cluster, cannot change this field
sourceClusterId: p-12345abcde # required
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to use source cluster tags. tags values will use exactly these␣
↪tags only. set tags: [] to use no tags.
  - color: blue
    tagName: tag-1
name: restore-aha-name
password: Password@1234 #Password must be 12 characters or more
dataGroup:
  groupName: Data Group A
  imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
  deploymentLocation: managed-devspatcher
  networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#   allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#     - cidrBlock: 9.9.9.9/28

#       description: Allow traffic from App A

#     - cidrBlock: 10.10.10.10/27

#       description: Allow traffic from App B

  backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
  nodes: 2 # only can be set to 2 or 3
```

(continues on next page)

(continued from previous page)

```
#   image: # only one of `imageId` or `image` should be set

#     url: docker.example.com/platform/example/postgres:latest

#     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

  postgresqlConfiguration: # Optional
    max_connections: 100
    max_locks_per_transaction: 64
  backupSchedule: 0 4 8   # Scheduled Backup Cron
  restoreAhaGroupRequest:
    #full: # for pointInTime restore mode use the pointInTime configuration as below␣
↪and comment out the full configuration
    #  postgresBackupId: 20241228T055553
     pointInTime:
       pointInTime: 2025-01-01T13:40:04.000Z
  instanceSize:
      cpuCores: 2 # CPU must be between 0.1 and 47.51 Cores
      memoryGi: 2 # Memory must be between 0.1 and 178.51 Gi. For Aha clusters, the␣
↪recommended memory is 2 Gi or more.
  storage:
    databaseStorage:
      sizeGi: 20 # Size must be between 0 and Unlimited (0-Unlimited Gi). For Aha␣
↪clusters, the recommended size is 10 Gi or more.
      storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    walStorage:
      sizeGi: 20 # Size must be between 0 and Unlimited (0-Unlimited Gi)
      storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes`␣
↪field
  transparentDataEncryption: # Optional. Only one of passphrase or keyId should be␣
↪set. If TDE is enabled on source cluster, configuration must match source or␣
↪provide passphrase.
    passphrase: mypassphrase # Required if source cluster uses TDE with passphrase
    #keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4 # Uses KMS key from source if source␣
↪cluster uses KMS
```

edbctl cluster switchover-primary

The `edbctl cluster switchover-primary` command performs a controlled role swap within a database cluster. It promotes a designated Replica to become the new Primary node while simultaneously demoting the current Primary to a Replica role. This is typically used for planned maintenance, performance optimization, or data center migration.

```
edbctl cluster switchover-primary [options]
```

Where `options` are:

```
- i, --id string                The unique ID of the cluster to switchover␣
↪(required).
```

(continues on next page)

(continued from previous page)

```
    --replica-id string         The unique ID of the replica node to promote to␣
↪Primary (required).
- P, --project string            The project context for the operation. Defaults to␣
↪`context_project`.
    --credential string         The credential to use for authentication. Defaults to␣
↪`context_credential`.
- y, --yes                       Auto-confirms all prompts and confirmations,␣
↪enabling non-interactive operation.
- h, --help                      Displays help information for the switchover-primary␣
↪command.
```

edbctl config

Displays or modifies edbctl configuration settings, including the default project, credentials, and output format.

Commands

The `edbctl config` command has following options:

- *edbctl config get* — Prints the value of a given configuration property.
- *edbctl config list* — Lists all configuration properties and values.
- *edbctl config set* — Updates the configuration with a value for the given property.

edbctl config get

Displays the value of a specified configuration property.

```
edbctl config get <key> [options]
```

Where `options` are:

```
-h, --help          Help for get.
-I, --interactive   Execute command interactively.
```

These examples show how to obtain the value of a given configuration property:

```
$ edbctl config set output_mode json
$ edbctl config get output_mode
json
$ edbctl config get output_mode
table
$ edbctl config get confirm_enabled
true
```

edbctl config list

Lists all configuration properties and their values.

```
edbctl config list [options]
```

Where `options` are:

```
-h, --help            Help for list.
-o, --output string   table json yaml xml json-raw (default "table")
```

edbctl config set

Updates a configuration property with a specified value.

```
edbctl config set <key> <value> [options]
```

Where `options` are:

```
-h, --help          Help for set.
-I, --interactive   Execute command interactively.
```

These examples show how to set a particular configuration property:

```
$ edbctl config set output_mode json
$ edbctl config set confirm_enabled true
$ edbctl config set prompt_enabled true
```

edbctl credential

This command creates and manages authentication credentials that allow users to access the resources. It allows you to import an Access API key as a login credential or set the environment variable `HCP_ACCESS_KEY` .

Commands

The `edbctl credential` command has following options:

- *edbctl credential add-tde-key* - Add a TDE key.
- *edbctl credential list* - Shows the credentials.
- *edbctl credential list-tde-key* - List TDE keys in the project.
- *edbctl credential import-access-key* - Imports an Access API key for authentication.
- *edbctl credential delete* - Deletes the credentials.

edbctl credential add-tde-key

The `edbctl credential add-tde-key` command registers a master key from an external Key Management Service (KMS) to be used for Transparent Data Encryption (TDE). TDE provides an additional layer of security by encrypting database files at rest, and is supported for EDB Postgres Advanced Server and EDB Postgres Extended Server (version 15 and later).

```
edbctl credential add-tde-key [options]
```

Where `options` are:

```
    --location-id string        Location ID (required). The location where the TDE␣
↪key will be stored.
    --kms-provider string       Key provider (required). The provider of the TDE key.␣
↪Valid values: `aws`, `hashicorp-vault`.
    --hc-vault-address string   HashiCorp Vault address (required if key provider is␣
```

(continues on next page)

(continued from previous page)

```
↪`hashicorp-vault`).
    --hc-vault-role string      HashiCorp Vault role (required if key provider is␣
↪`hashicorp-vault`).
    --hc-vault-key string       HashiCorp Vault key (required if key provider is␣
↪`hashicorp-vault`).
    --aws-kms-arn string        AWS KMS ARN (required if key provider is aws). The␣
↪Amazon Resource Name (ARN) of the AWS KMS key.
    --friendly-name string      Friendly name for the TDE key (optional). A human-
↪readable name for the TDE key.
- c, --credential string         The credential which you created via `credential␣
↪create` command, the default is fetched from `context_credential`.
- y, --yes                       Auto-confirm all confirmations.
- P, --project string            The project that groups your clusters and other␣
↪resources, the default is taken from `context_project`.
- h, --help                      help for `add-tde-key.`
```

This example shows how to add a TDE key from AWS KMS:

```
edbctl credential add-tde-key \
  --location-id "loc_123" \
  --kms-provider aws \
  --aws-kms-arn "arn:aws:kms:us-east-1:123456789012:key/abcd1234-56ef-78gh-90ij" \
  --friendly-name "AWS-Prod-Key"
```

This example shows how to add a TDE key from HashiCorp Vault:

```
edbctl credential add-tde-key \
  --location-id "loc_456" \
  --kms-provider hashicorp-vault \
  --hc-vault-address "https://vault.example.com" \
  --hc-vault-role "db-encryption-role" \
  --hc-vault-key "customer-data-key" \
  --friendly-name "Vault-Security-Key"
```

edbctl credential list

This command shows all the credentials. Optionally, use the `--name` option to list a specific credential.

```
edbctl credential list [options]
```

Where `options` are:

```
- h, --help            Help for list.
- n, --name string     Credential name (optional). Specify the name of the credential␣
↪to list.
- o, --output string   [table json yaml xml] \(default "table")
```

These examples show how to list the credentials:

```
#  List all the credentials

edbctl credential list
```

(continues on next page)

(continued from previous page)

```
#  List a specific credential

edbctl credential list -n <credential_name>
```

edbctl credential list-tde-key

The `edbctl credential list-tde-key` command allows you to view and manage the Transparent Data Encryption (TDE) keys registered within your project. It provides a way to verify key registration, retrieve specific Key IDs for cluster configurations, and audit which keys are available at specific deployment locations.

This command displays a list of TDE keys associated with a specific location. You can further narrow down the results by providing a specific Key ID or changing the output format for programmatic use.

```
edbctl credential list-tde-key [options]
```

Where `options` are:

```
    --location-id string    Location ID (required). The location to filter TDE keys.
    --id string             Filter the list to display details for a specific TDE key.
- P, --project string        The project that groups your clusters and other␣
↪resources, the default is taken from `context_project`.
- o, --output string         Output format: `table` (default), `json`, `yaml`, `xml`,␣
↪`json-raw`.
- h, --help                  Displays help information for the list-tde-key command.
```

These examples show how to use the `list-tde-key` command:

```
#  List all TDE keys for a specific location:

edbctl credential list-tde-key --location-id "loc_123"

#  List a specific TDE key by ID:

edbctl credential list-tde-key --location-id "loc_123" --id "key_456"

#  Export list to JSON:

edbctl credential list-tde-key --location-id "loc_123" --output json
```

edbctl credential import-access-key

This command is used to import an Access API key. You can get the Access API key from the server portal to authenticate. You can also use this credential for further operations.

```
edbctl credential import-access-key [--name --access-key --address] [options]
```

Where, `options` are:

```
- n, --name string         Credential name (required). The unique name for the␣
↪credential.
- k, --access-key string   Access key (required). The API key used for authentication.
```

(continues on next page)

(continued from previous page)

```
- a, --address string      Authenticator host address (required).
- I, --interactive         Execute command interactively.
- h, --help                Help for import-access-key.
```

These examples show how to use the `import-access-key` command:

```
#  Import the Access API key:

edbctl credential import-access-key

#  Import the specific Access API key using the name, access-key, and address:

edbctl credential import-access-key --name cred1 --access-key <your-key-here> --
↪address <authenticator-host-address-here format: [http|https]://host:port example:␣
↪http://localhost:8080 and https://secure.portal.net:443>
>

edbctl credential import-access-key -n cred1 -k <your-key-here> -a <authenticator-
↪host-address-here format: [http|https]://host:port example: http://localhost:8080␣
↪and https://secure.portal.net:443>
```

edbctl credential delete

This command deletes the specific credential with name.

```
edbctl credential delete <credential_name> [options]
```

Where `options` are:

```
- h, --help          Help for delete.
- I, --interactive   Execute command interactively.
- y, --yes           Auto-confirm all confirmations
```

This example shows how to delete the credential:

```
#  Delete the credential with <credential_name> name:

edbctl credential delete <credential_name>
```

edbctl image-pull-secret

This command helps to manage pull secrets used in the bootstrapping process of Hybrid Manager (HM).

Commands

`edbctl image-pull-secret` command has the following options available:

- *edbctl image-pull-secret create* — Create the Kubernetes namespaces and AssetPullSecrets
- *edbctl image-pull-secret list* — Lists the Kubernetes namespaces and AssetPullSecrets

edbctl image-pull-secret create

Sets up the necessary Kubernetes namespaces and AssetPullSecrets using the provided credentials, preparing for HM bootstrapping.

```
edbctl image-pull-secret create --username <username> --password <password> --
↪registry <registry> [options]
```

Where `options` are:

```
    --username string   Registry username (required).
    --password string   Registry password (required).
    --registry string   Container Registry (required). The URL of the container␣
↪registry.
- h, --help              Help for create.
```

This example shows how to create a Kubernetes namespace and AssetPullSecret:

```
edbctl image-pull-secret create --username <username> --password <password> --
↪registry <registry>
```

edbctl image-pull-secret list

Retrieves and lists the Kubernetes namespaces and AssetPullSecrets used in HM bootstrapping.

```
edbctl image-pull-secret list [options]
```

Where `options` are:

```
- h, --help help for list command
```

edbctl image

This command helps managing the images for Hybrid Manager (HM). With this command, you can sync container images from EDB's repository to a local registry and manage artifacts for internet-disconnected OpenShift environments.

Commands

`edbctl image` command has following options:

- *edbctl image create-image-set-config-for-oc-mirror* — Creates an AssetSetConfiguration YAML file for internet-disconnected OpenShift environments.
- *edbctl image list-image-tags* — Displays a list of all the available images for provisioning.
- *edbctl image sync-to-local-registry* — Syncs container images from EDB Repos 2.0 to a local registry.

edbctl image create-image-set-config-for-oc-mirror

Generates an AssetSetConfiguration YAML file for OpenShift environments without internet access. The file allows oc mirror to create a local tarball containing all container images for a specified Hybrid Manager release.

```
edbctl image create-image-set-config-for-oc-mirror [options]
```

Where `options` are:

```
    --source-registry-username string   Source registry username (required). The␣
↪username for authenticating with the source registry.
    --source-registry-password string   Source registry password (required). The␣
↪password for authenticating with the source registry.
    --version string                    Hybrid Manager release version (required).␣
↪Specify the version of the release to sync.
    --output-file-path string           Output YAML file path (optional). Defaults to␣
↪ImageSetConfiguration.yaml in the current directory.
- h, --help                              Help for create-image-set-config-for-oc-
↪mirror.
```

This example shows how to create an AssetSetConfiguration YAML file for an OpenShift environment.

```
edbctl image create-image-set-config-for-oc-mirror --source-registry-username
↪<username> --source-registry-password <password> --version <version> [--output-file-
↪path <output-file-path>]
```

edbctl image list-image-tags

Retrieves and displays a list of all available image tags used for provisioning. The output includes detailed metadata for each image tag, such as:

- Image tag ID
- Name
- SHA
- Version
- Compressed size
- Architecture
- Creation time
- Ingestion time
- Location
- Associated tags

```
edbctl image list-image-tags [options]
```

Where `options` are:

```
    --location-id string        Location ID (required). Specify the location ID to␣
↪filter images.
    --pg-type string            Filter images by Postgres type. Examples: epas,␣
↪postgres, pgextended.
    --major-version string      Filter images by major version. Examples: 17, 16.
    --pg-architectures string   Filter images by cluster architectures. Examples: pgd,
↪ aha, psr, analytics.
-c, --credential string         The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
-o, --output string             [table json yaml xml] (default "table")
-h, --help                      Help for list-image-tags.
```

This example shows how to list the image:

```
edbctl image list-image-tags --location-id "managed-devspatcher"
```

edbctl image sync-to-local-registry

Syncs container images from EDB Repos 2.0 to a local registry for managing a local HM installation.

edbctl image sync-to-local-registry [options]

Where `options` are:

```
    --source-registry-username string        Source registry username (required).
    --source-registry-password string        Source registry password (required).
    --destination-registry-uri string        Destination registry URI (required).
    --destination-registry-username string   Destination registry username (required).
    --destination-registry-password string   Destination registry password (required).
    --version string                         Hybrid Manager release version␣
↪(required).
    --sign-by-sigstore-private-key string    Path to the Sigstore private key.
    --sign-passphrase-file string            Path to the file containing the␣
↪passphrase for the Sigstore private key.
- y, --yes                                    Auto-confirm all confirmations.
- h, --help                                   Help for sync-to-local-registry.
```

This example shows how to sync the EDB Repo 2.0 to the local registry:

```
edbctl image sync-to-local-registry --source-registry-username <username> --source-
↪registry-password <password> --destination-registry-uri <registry> --destination-
↪registry-username <username> --destination-registry-password <password> --version
↪<version>
```

edbctl operand

Manages EDB Postgres AI Operands, particularly in the context of OpenShift environments, including those are internet-disconnected. The `edbctl operand` is the entry point for commands that handle operand management such as:

- Syncing operand images
- Managing artifacts for internet-disconnected (air-gapped) OpenShift environments

Commands

- *edbctl operand sync-to-local-registry* : Sync operand images from EDB repository to a local registry.
- *edbctl operand list-images* : Displays a complete list of all operand images managed by EDB.
- *edbctl operand download-image-list* : Used to download the file containing the list of required operand images.

edbctl operand sync-to-local-registry

Syncs the operand images from EDB repository to a local registry to manage a local install of HM.

```
edbctl operand sync-to-local-registry [options]
```

Where `options` are:

```
    --source-registry-username string         Source registry username (required).
    --source-registry-password string         Source registry password (required).
    --destination-registry-uri string         Destination registry URI (required).
    --destination-registry-username string    Destination registry username (required).
    --destination-registry-password string    Destination registry password (required).
    --version string                          Hybrid Manager release version␣
↪(required).
    --sign-by-sigstore-private-key string     Path to the Sigstore private key.
    --sign-passphrase-file string             Path to the file containing the␣
↪passphrase for the Sigstore private key.
-w, --workers integer                         Number of parallel workers to use for␣
↪syncing images (default is 3).
-y, --yes                                     Auto-confirm all confirmations.
-h, --help                                    Help for sync-to-local-registry.
```

This example shows how to sync operand images from EDB repository to a local registry:

```
edbctl image sync-to-local-registry --source-registry-username <username> --source-
↪registry-password <password> --destination-registry-uri <registry> --destination-
↪registry-username <username> --destination-registry-password <password> --version
↪<version> --workers 3 --cpu 3
```

edbctl operand list-images

This command is used to retrieve and display a comprehensive list of all container images associated with a specific version of the Hybrid Manager. It provides crucial information including the image tab, for each component.

```
edbctl operand list-images [options]
```

Where `options` are:

```
    --source-registry-username string          Source registry username (required).␣
↪The username for accessing the repository where the images are stored.
    --source-registry-password string          Source registry password (required).␣
↪The password for accessing the repository where the images are stored.
    --version string                           Hybrid Manager release version␣
↪(required). Specifies which version of the operand to list images for.
-y, --yes                                      Auto-confirm all confirmations.
-h, --help                                     Displays help information for the list-
↪images command.
```

edbctl operand download-image-list

This command allows you to download a file that contains a list of all required operand images for a specified version of the Hybrid Manager. This is particularly useful for preparing for installations in air-gapped or internet-disconnected environments.

```
edbctl operand download-image-list [options]
```

Where `options` are:

```
    --source-registry-username string       Source registry username (required). The␣
↪username for accessing the repository where the images are stored.
```

(continues on next page)

(continued from previous page)

```
    --source-registry-password string       Source registry password (required). The␣
↪password for accessing the repository where the images are stored.
    --version string                        Hybrid Manager release version (required).
↪ Specifies which version of the operand to download the image list for.
    --output-filename string                Output file path (required). The path␣
↪where the downloaded image list file will be saved.
-y, --yes                                   Auto-confirm all confirmations.
-h, --help                                  Displays help information for the␣
↪download-image-list command.
```

This example shows how to download the operand image list for a specific Hybrid Manager version:

```
edbctl operand download-image-list --version "v1.0.0" --source-registry-username
↪"myuser" --source-registry-password "mypassword" --output-filename "/tmp/edb-images.
↪txt"
```

edbctl operator

Manages EDB Postgres AI Operator components and artifacts. The `edbctl operator` command provides tools for managing the Postgres AI Operator, including:

- Syncing operator images
- Managing artifacts for internet-disconnected (air-gapped) OpenShift environments

Commands

- *edbctl operator sync-to-local-registry* : Sync operator images from EDB repository to a local registry.

edbctl operator sync-to-local-registry

Syncs the operator images from EDB repository to a local registry to manage a local install of Hybrid Manager.

```
edbctl operator sync-to-local-registry [options]
```

Where `options` are:

```
    --source-registry-username string        Source registry username (required).
    --source-registry-password string        Source registry password (required).
    --destination-registry-uri string        Destination registry URI (required).
    --destination-registry-username string   Destination registry username (required).
    --destination-registry-password string   Destination registry password (required).
    --version string                         Hybrid Manager release version␣
↪(required).
    --sign-by-sigstore-private-key string    Path to the Sigstore private key.
    --sign-passphrase-file string            Path to the file containing the␣
↪passphrase for the Sigstore private key.
-y, --yes                                    Auto-confirm all confirmations.
-h, --help                                   Help for sync-to-local-registry.
```

This example shows how to sync the EDB repository to a local registry:

```
edbctl operator sync-to-local-registry --source-registry-username <username> --source-
↪registry-password <password> --destination-registry-uri <registry> --destination-
↪registry-username <username> --destination-registry-password <password> --version
↪<version>
```

edbctl project

Manages projects for organizing clusters and resources. Use this command to create, edit, view, and delete projects, making it easier to manage your database clusters efficiently.

Commands

- *edbctl project list* — Displays all available projects.
- *edbctl project create* — Creates a new project.
- *edbctl project edit* — Edits a project.
- *edbctl project delete* — Deletes a specified project.

edbctl project list

Lists all available projects and displays detailed information, including project ID, name, user count, cluster count, and tags. You can also filter results by project name or ID.

```
edbctl project list [options]
```

Where `options` are:

```
- n, --name string         Project name (optional). Filter the list by a specific␣
↪project name.
- i, --id string           Project ID (optional). Filter the list by a specific␣
↪project ID.
- c, --credential string   The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- o, --output string       [table json yaml xml] \(default "table")
- h, --help                Help for list.
```

This example shows how to list all the projects:

```
edbctl project list
```

edbctl project create

Creates a new project with a specified name and optional tags. Use this command to define and organize projects efficiently.

```
edbctl project create [options]
```

Where `options` are:

```
- n, --name string         Project name (required). The unique name for the project.
    --tags strings        Assign tags to the project in the format name=color (color␣
↪is optional).
```

(continues on next page)

(continued from previous page)

```
- c, --credential string   The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- y, --yes                 Auto-confirm all confirmations.
- h, --help                Help for create.
```

These examples show how to create the projects using different options:

```
edbctl project create "My Project"
edbctl project create --name "My Project" --tags name2=blue
edbctl project create --name "My Project" --tags name2=#FF0000
```

edbctl project edit

Updates a project using its name or ID. You can rename the project and assign new tags. Make sure to provide the correct project name or ID before updating.

```
edbctl project edit {--id|--name} --new-name [options]
```

Where `options` are:

```
- i, --id string           Project ID (required if --name isnt provided). The␣
↪immutable unique identifier for a project.
- n, --name string         Project name (required if --id isnt provided). The mutable␣
↪unique identifier for a project.
- N, --new-name string     New name for the specified project.
    --tags strings        Assign tags to the project in the format name=color (color␣
↪is optional).
- c, --credential string   The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- y, --yes                 Auto-confirm all confirmations.
- I, --interactive         Execute command interactively.
- h, --help                Help for edit.
```

These examples show how to update the projects:

```
edbctl project edit --id prj_MFq4M4EWEoalBAL3 --new-name "New Name"
edbctl project edit --name "Old Name" --new-name "New Name"
edbctl project edit --name "Old Name" --tags name2=blue
```

edbctl project delete

Deletes a specified project using its name or ID. Before deleting a project, you must permanently remove all its clusters. This action is irreversible, so ensure you have the correct project name or ID before proceeding.

```
edbctl project delete {--id|--name} [options]
```

Where `options` are:

```
- i, --id string           Project ID (required if --name isnt provided). The␣
↪immutable unique identifier for a project.
- n, --name string         Project name (required if --id isnt provided). The mutable␣
↪unique identifier for a project.
```

(continues on next page)

(continued from previous page)

```
- c, --credential string   The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- y, --yes                 Auto-confirm all confirmations.
- h, --help                Help for delete
```

Examples

```
edbctl project delete --id prj_MFq4M4EWEoalBAL3
edbctl project delete --name "My Project"
```

5.3.10 Using the API

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:

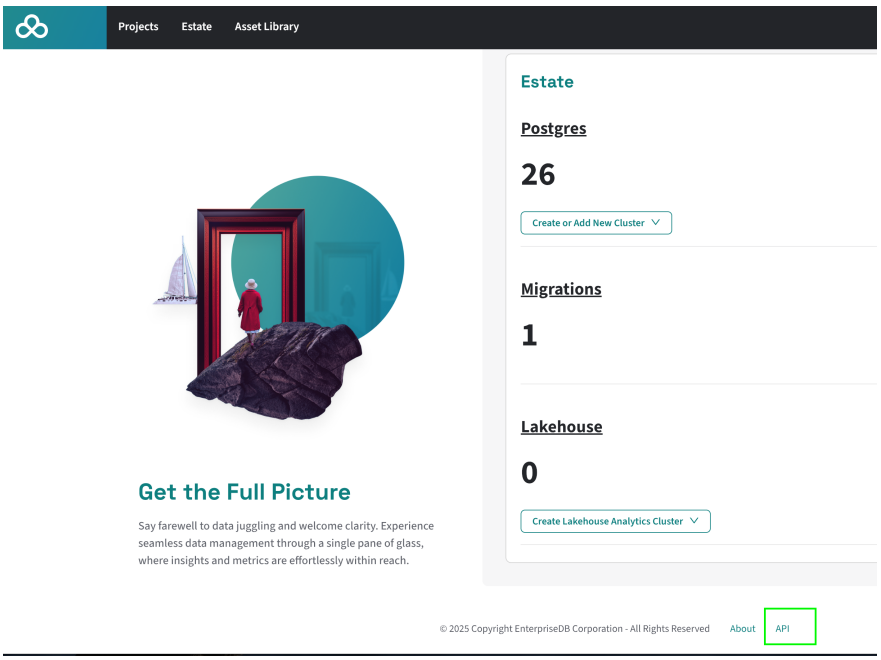


Fig. 12: API Docs

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.

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, *1. Generate a Fernet key (if not already created)* 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": []
    }
  ]
}
```

5.3.11 For Kubernetes Admin

Managing users in Hybrid Manager

Hybrid Manager (HM) uses Dex for user authentication in the HM console, acting as a proxy between HM and any native users or external user management systems (identity providers or IdPs) like LDAP and SAML.

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. This needs to be done at install time. After configuring your chosen IdP and adding new administrators for the organization, disable the native user login.

- See Configuring your own identity provider for more information.

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 configure an external IdP . After configuring an IdP, you can:

- *Disabling native user login* .

Changing password for native users

To avoid unauthorized access to your environment, change the password of the default native user during deployment or immediately after deployment. You can change the password for native users you manually added in the same way.

1. Create a new bcrypt-hashed password. To generate this value, use:

```
echo ${password} | htpasswd -BinC 10 admin | cut -d: -f2
```

Where the new password is stored behind the `${password}` variable.

The output prints the new hashed password, for example, `$2y$10$YDHMEnc8VZKnhvze538ePOLKol7dFH.S7IgpqC3.X3R/3P2Q9zRcG`.

1. Replace the hash for the existing user: `owner@mycompany.com` in this example. Replace this value either in your `values.yaml` file or `HybridControlPlane` custom resource, depending on the installation method you use.

```
portal:
  authentication:
    staticPasswords:
      - email: "owner@mycompany.com"
        hash: "$2y$10$YDHMEnc8VZKnhvze538ePOLKol7dFH.S7IgpqC3.X3R/    3P2Q9zRcG"
        username: "owner@mycompany.com"
        userID: "c5998173-a605-449a-a9a5-4a9c33e26df7"
```

1. If you're modifying the parameter in an already deployed installation, the `dex` service will restart automatically to apply the changes after saving them.

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 an external IdP .

Currently, you can't create human users with the Hybrid Manager (HM) console. You can do so only by tweaking the HM configuration file.

Adding a native user

Append new users to the `portal.authentication.staticPasswords` section during or after installation. Replace this value either in your `values.yaml` file or `HybridControlPlane` custom resource, depending on the installation method you use.

```
portal:
  authentication:
    staticPasswords:
      - email: "owner@mycompany.com"
        hash: "$2y$10$STTzuGJa3PydqHvlzi2z6OgDU1G/JLTqiuGblH.RemIutWxkztN5m"
        username: "owner@mycompany.com"
        userID: "c5998173-a605-449a-a9a5-4a9c33e26df7"
     - email: "<userA@example.com>"
       hash: "<example_hashed_password>"
       username: "<userA@example.com>"
       userID: "<unique_user_id>"
```

The dex service will restart to apply the changes automatically after saving them.

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 ${password} I htpasswd -BinC 10 userA I cut -d: -f2*, where the actual password is stored behind the *${password}* variable. *userA* represents the username used during the password hashing process. It can be any arbitrary text, as it's not used elsewhere in the configuration. Only the resulting hash is used.
userID	Each new user configured with a static password must have a distinct, unique identifier for the user (uuid). You can choose to generate this value with a UUID generator tool or assign a random sequence of characters manually.
username	Unique username for the user. This is the primary identifier for logging into 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.

Assign 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.

Disable native users

1. To disable native user login, unset the `staticPasswords` configuration by setting `portal.authentication.staticPasswords` to `[]`:

```
portal:
  authentication:
    staticPasswords: []
```

1. The dex service will restart automatically to apply the changes after saving them.

User roles and authorization

Several user roles can be assigned to Hybrid Manager (HM) users. See:

- *User role permissions by use case*
- *User role descriptions*

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 (admin)

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 (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 (platform admin)

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.

GenAI Builder user (GenAI Builder user)

This role is for users who work with the GenAI Builder feature of HM, granting them full access to its functions.

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 (owner)

In a specific project, the project owner 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 (editor)

This role is for users who need to actively work with the data in a project. Project editors have data read and write access.

Project viewer (viewer)

For users who only need to observe the data in a project, the project viewer role provides data read-only access.

Estate ingester (estate ingester)

This role is for machine users and provides access to perform estate ingests in a project.

Catalog data reader (catalog data reader)

Users with this role can read the Iceberg namespace/table/view of project-scoped catalogs.

Catalog data writer (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.

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 11 predefined roles are available.

Permissions	Org admin	Org owner	Platform admin	GenAI Builder user	Project owner	Project editor	Project viewer	Estate ingester	Catalog data reader	Catalog data writer
Access GenAI Builder (launchpad)				X						
Configure GenAI Builder			X							
Access Ops apps (launchpad)			X							
View projects within the org	X	X								
Update and delete projects		X								
View roles assigned at the project level	X	X			X	X				
View activity log for the org	X	X								
View and download usage report for the project		X			X	X				
View and download usage report the the org	X	X								
Create projects within the org		X								
Assign project roles		X			X					
Create, edit, and delete clusters					X	X				
View clusters, backups, estates, and migrations					X	X	X			
Assign org roles		X								

continues on next page

Table 17 – continued from previous page

Permissions	Org admin	Org owner	Platform admin	GenAI Builder user	Project owner	Project editor	Project viewer	Estate ingester	Catalog data reader	Catalog data writer
View activity log for the project		X			X	X				
View, edit, and delete owned projects					X					
Ingest self-managed Postgres cluster data								X*		
Create, update, and delete catalog					X	X				
Read catalog							X			
Read Iceberg data									X	X
Write and delete Iceberg data										X

- Only machine-users can be assigned to ingest self-managed cluster data.

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.

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 .

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 HM 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

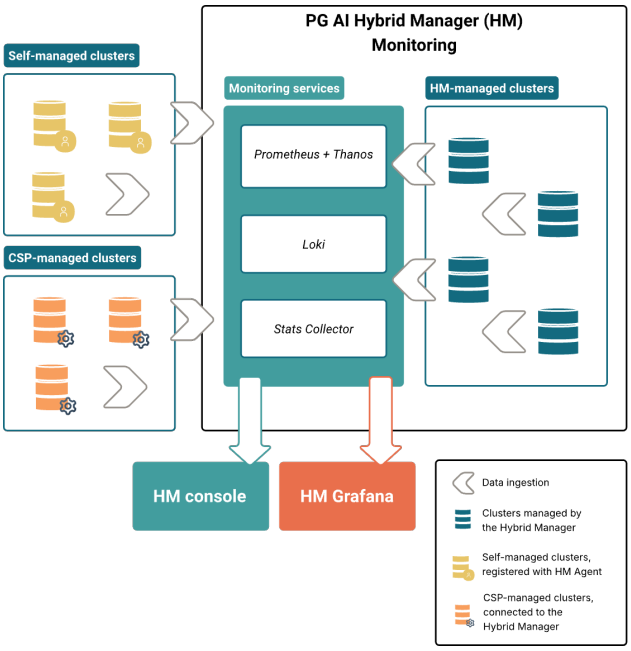


Fig. 13: Monitoring infrastructure in Hybrid Manager

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 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.

Connecting data into external tools

Currently, EDB doesn't support plugging metrics and log data collected by HM into external tools or enterprise systems.

Related topics

- *Enabling monitoring* ► Install the agent or connect your database to HM to enable monitoring on external databases.
- *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.

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

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 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 Beacon 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* 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.

Installing the Agent

See *Installing Vault* for instructions on how to install the Agent.

Configuring the HM agent

Configure the Hybrid Manager (HM) 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 HM agent can access it later.

Obtain Hybrid Manager connection parameters

The HM 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 HM agent.
- **Root certificate path (if required)**:The HM 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 HM agent. You will need to provide the directory path to this certificate later to configure the HM 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 HM agent uses to connect to your source database. Set the `databases.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.
```

2. Inside this directory, create a new file `beacon_agent.yaml` :

```
touch .beacon/beacon_agent.yaml
```

3. 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 HM agent`

<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.
<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*.
N/A	Set to - *"onprem"*.
N/A	Set to - *"onprem-settings"* to enable recommendations for Postgres configurations.
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.
N/A	Set to *""* if your organization has configured the machine where HM 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.
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.
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*.
N/A	Enables storing usage data (and reporting usage data, *Troubleshooting AI Factory on Hybrid Manager*). Your *company_code* can be found on the EDB Support Portal .
<location_id>	A unique identifier used by the Hybrid Manager 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.

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"*, *"other"*. For self-managed instances, set to *"other"*.
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 HM 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.

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 20 – 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.

Settings related to schema ingestion:

YAML database schema settings	Placeholder/Value	Guidelines
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 1m).
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.
- *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 HM agent to asses Oracle databases and extract schema for conversion with the Migration Portal.

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.databases` 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 becomes blank.
- `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: "<cluster_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: "<cluster_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.

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. 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. 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>"
```

(continues on next page)

(continued from previous page)

```
          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

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>
```

(continues on next page)

(continued from previous page)

```
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 agent or are getting errors when trying to run it, you can troubleshoot. Test the agent's functionalities in an isolated way to narrow down the source of the issue.

Testing connectivity from the agent to Hybrid Manager

To ensure there's a connection between the agent and Hybrid Manager (HM), run the agent:

```
beacon-agent --config
```

See Agent CLI for other 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 Agent to use an unencrypted connection. To do this, append the setting `?sslmode=disable` to the DSN used in the Agent's configuration , for example:

```
export DSN="postgresql://postgres:password@localhost:5432/postgres?sslmode=disable"
```

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 *HM console and observability* and *Chat model cluster metrics are missing from the Grafana model overview dashboard* .

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.

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 *Unable to query Knowledge Bases for self-managed clusters* 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 HM 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 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.

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*	*Chat model cluster metrics are missing from the Grafana model overview dashboard*	*Query metrics*	*GPU Recommendations for Default NIM Models*	*Accessing metrics and logs*
HM-managed cluster	Yes	Yes	Yes	Yes	Yes	Yes
External: **Self-managed cluster**	Yes	Yes	No	Yes	Yes	Yes
External: **CNPG database cluster**	Yes	Yes	Yes	Yes	Yes	Yes
External: **AWS RDS database**	Yes	partial support	No	No	No	No

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 *Chat model cluster metrics are missing from the Grafana model overview dashboard* . 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	No	Yes	No

continues on next page

Table 27 – continued from previous page

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Clock skew	Yes	No	Yes	No
Node status	Yes	No	Yes	No
Transaction rate	Yes	No	Yes	No
Connections (total)	Yes	No	Yes	No
Connections (per node)	Yes	No	Yes	No
Memory (per node)	Yes	No	Yes	No
Storage (per node)	Yes	No	Yes	No
WAL size (per node)	Yes	No	Yes	No
Replication status	Yes	No	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	No	No	No	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 HM 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
Cluster view ▶ Logs tab	10s

Note

Data is updated only when you explicitly reload the browser page.

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 HM 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 HM agent.
- CNP clusters : Partial support after the cluster is connected via HM 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 *Loki pods crash* 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

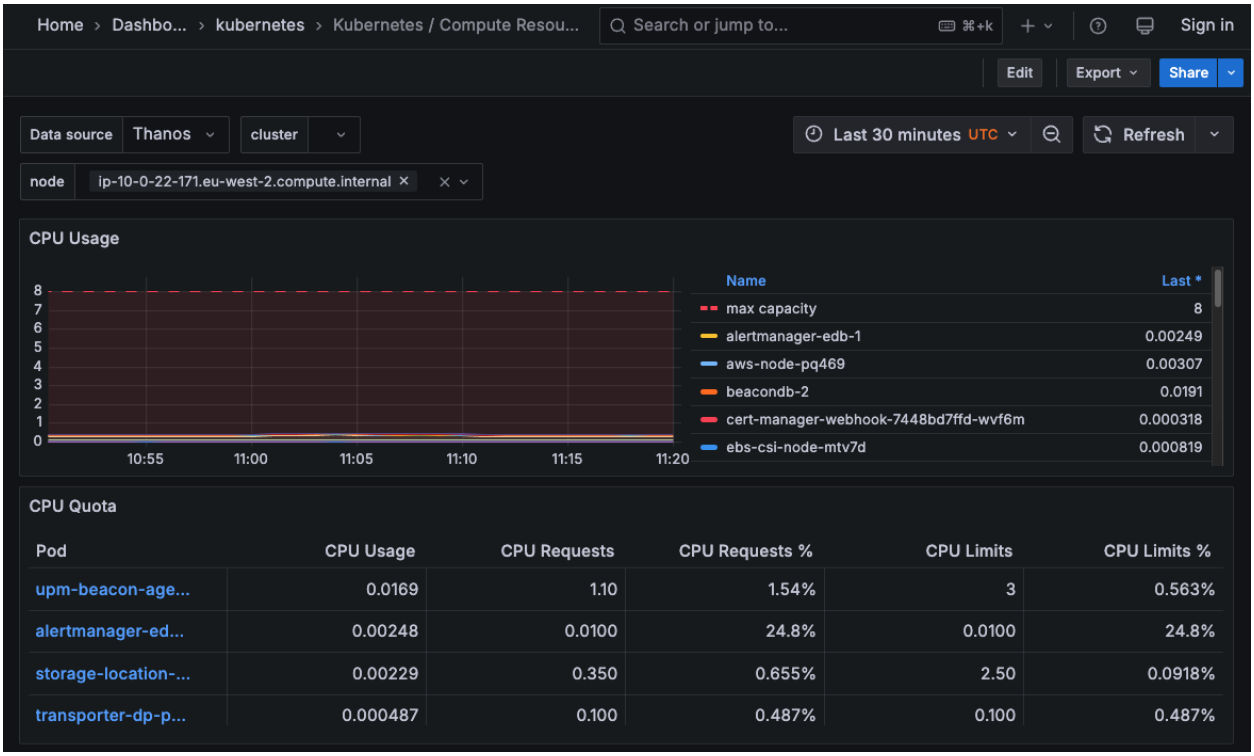


Fig. 14: Compute Resources by Node and Pods

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

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).

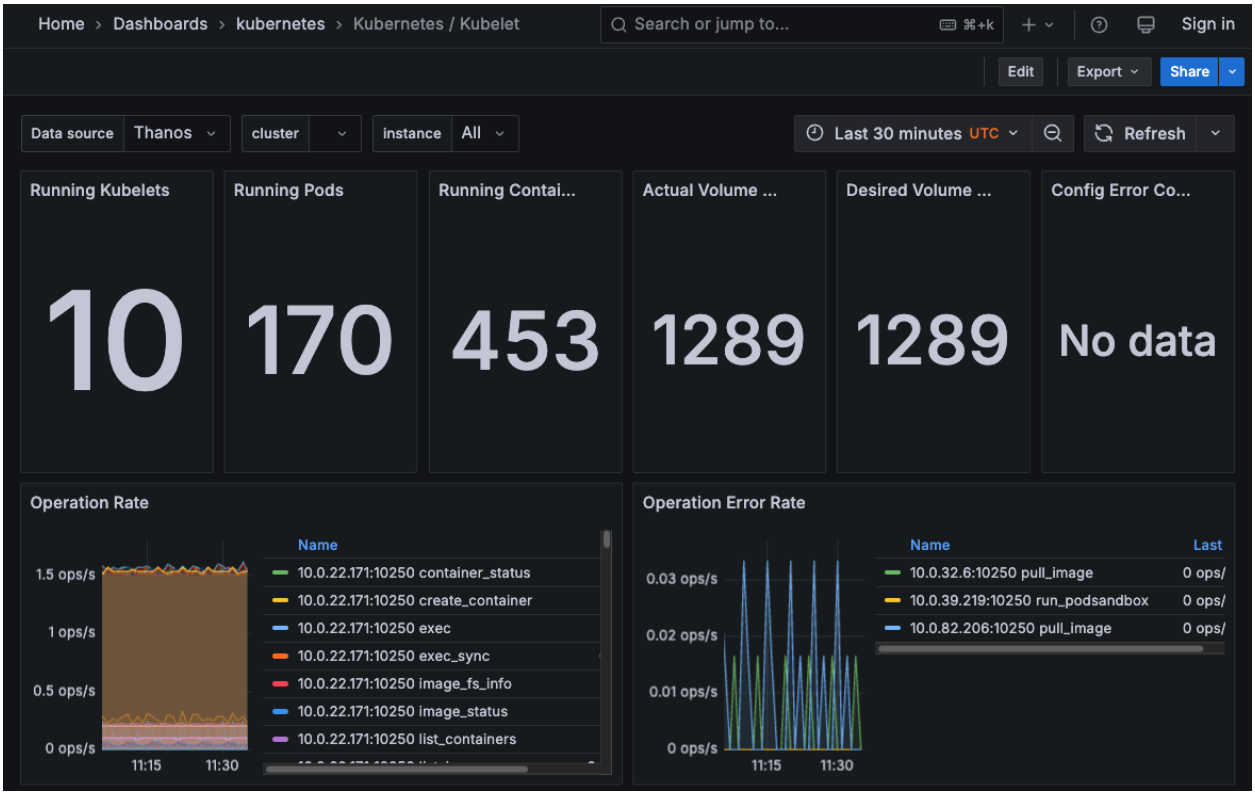


Fig. 15: Kubelet dashboard

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
- 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.

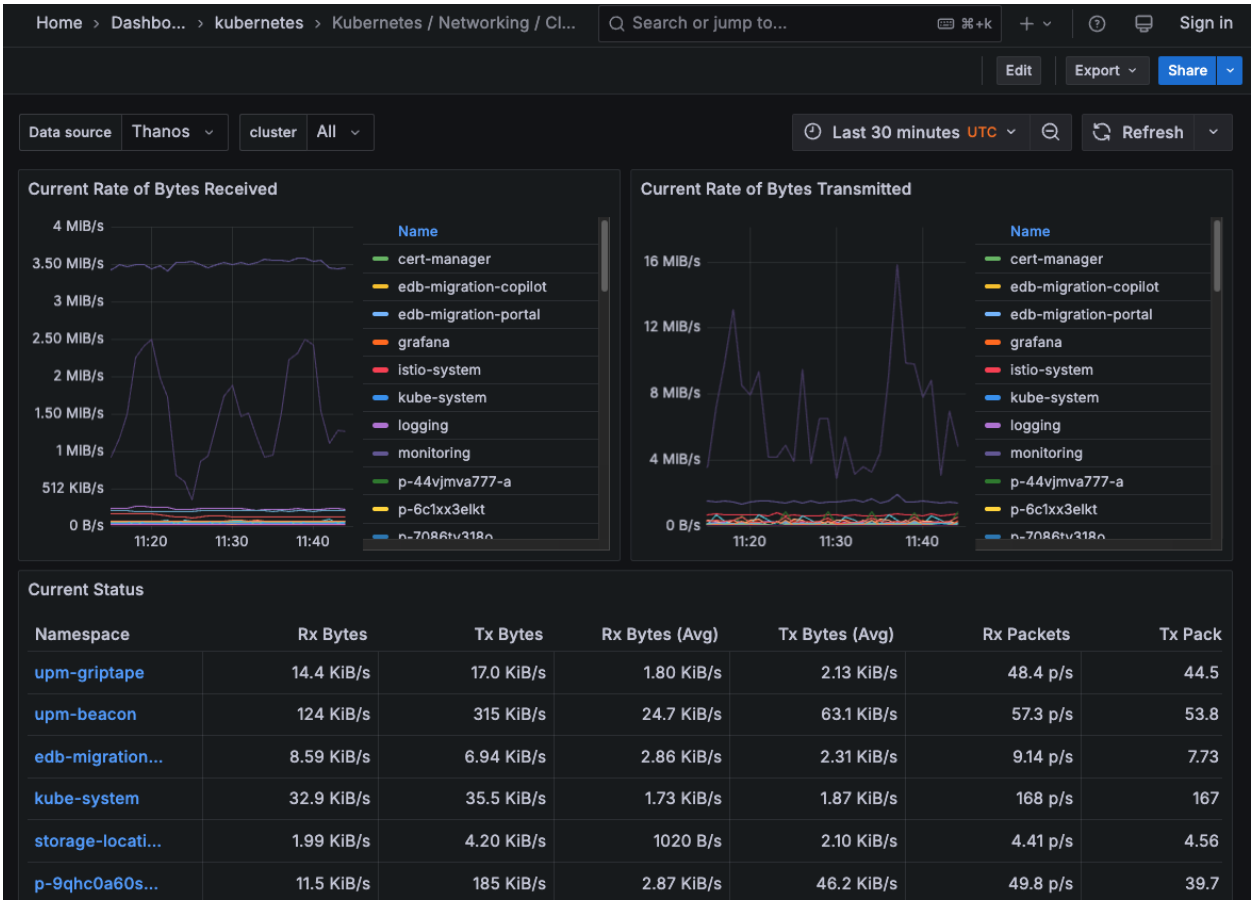


Fig. 16: Networking overall cluster dashboard

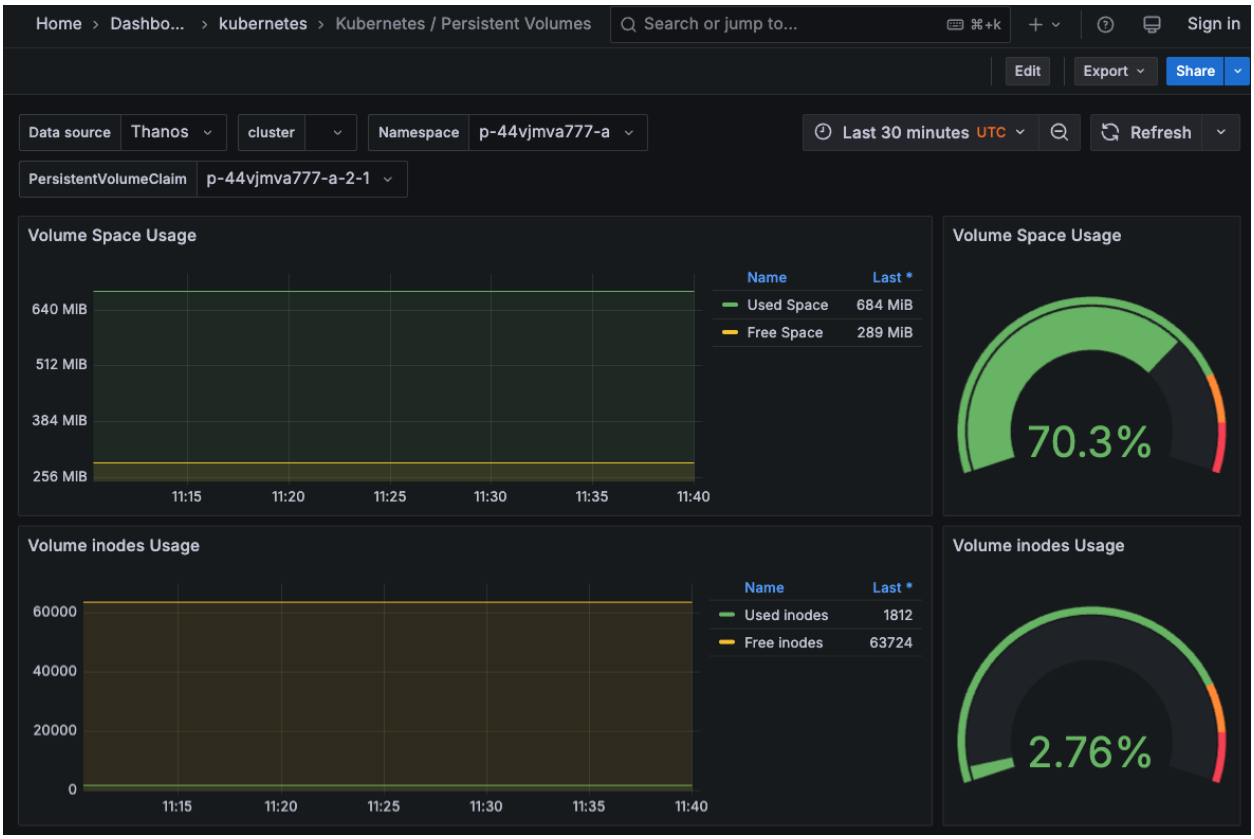


Fig. 17: Persistent Volumes dashboard

Kubernetes / StatefulSets

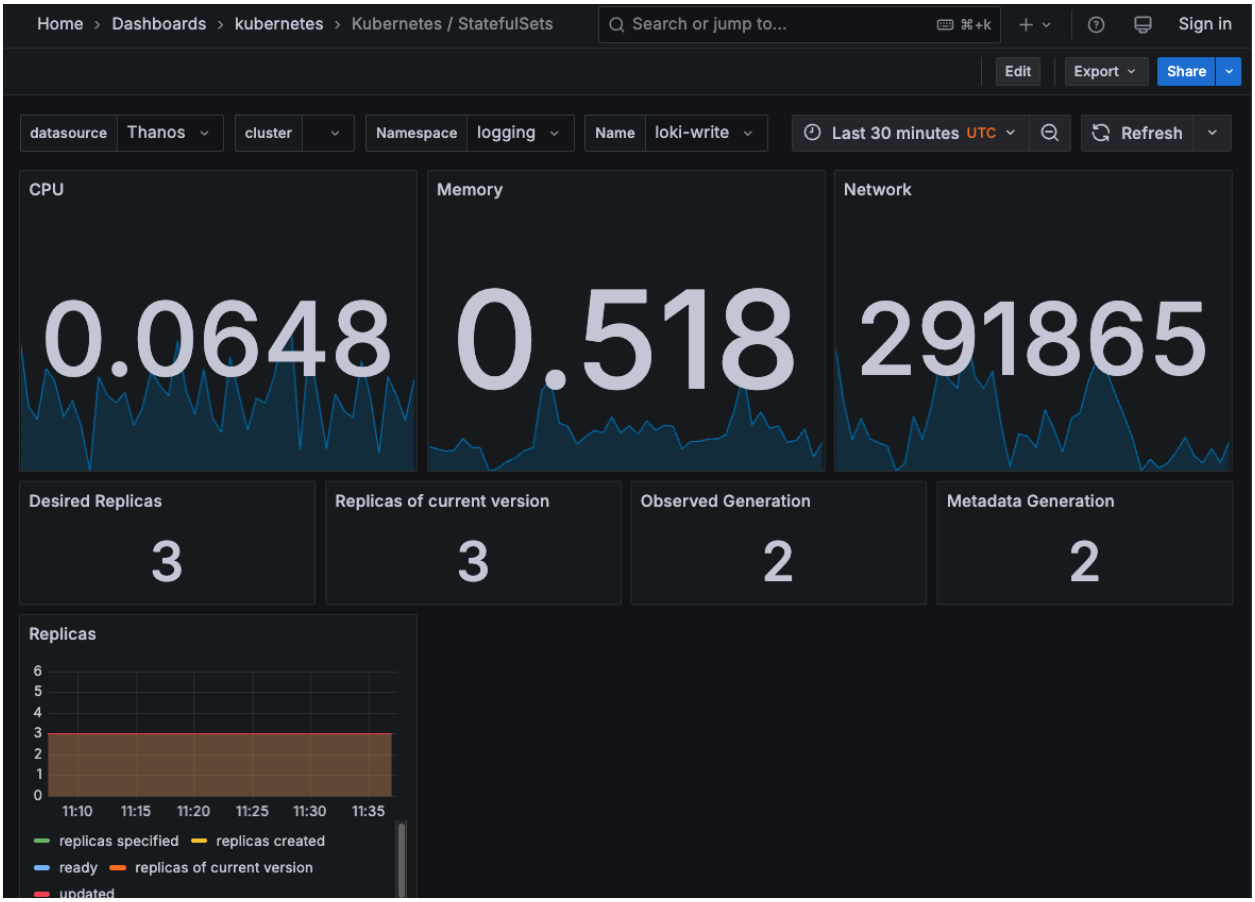


Fig. 18: StatefulSets dashboard

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.

Usage Use this to monitor the reliability and efficiency of the CoreDNS component and identify any discovery and communication issues among services.

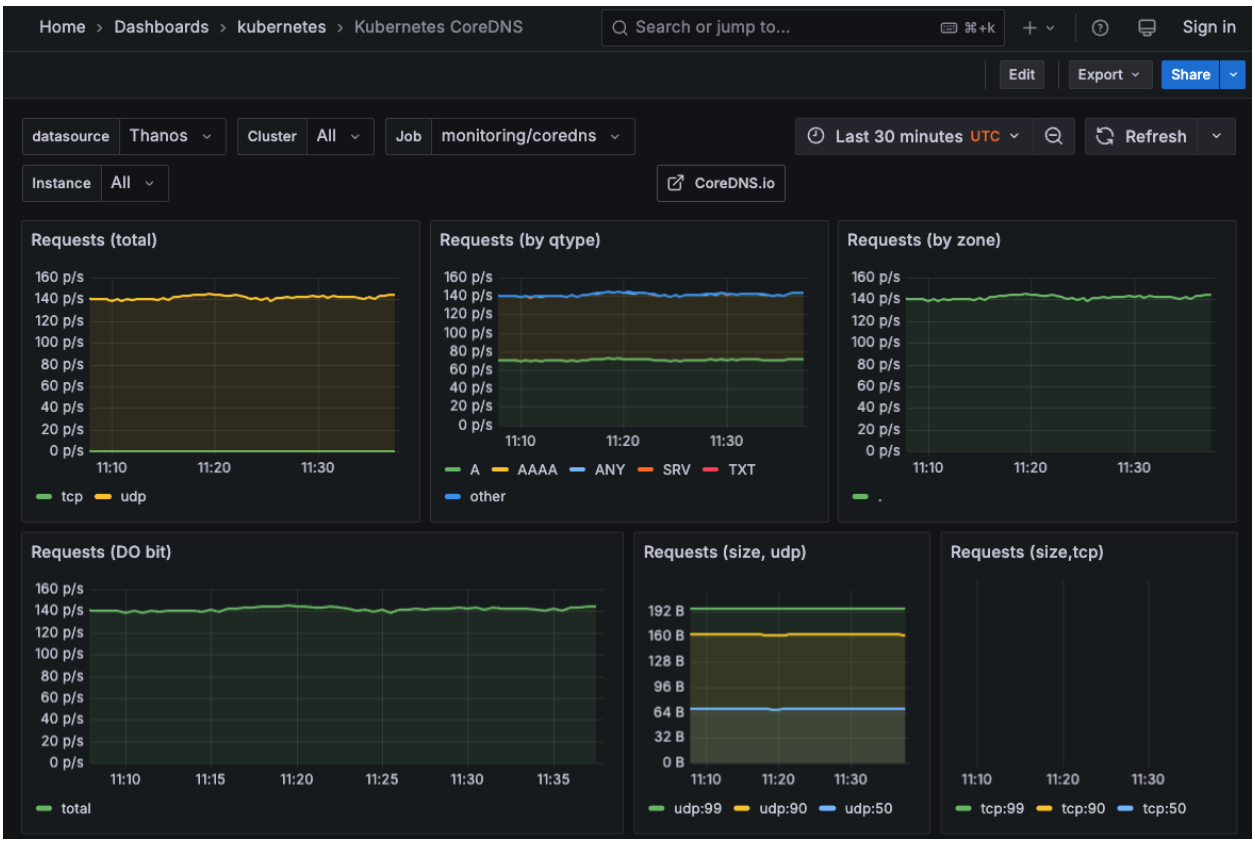


Fig. 19: CoreDNS dashboard

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



Fig. 20: Loki Chunks dashboard

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.

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

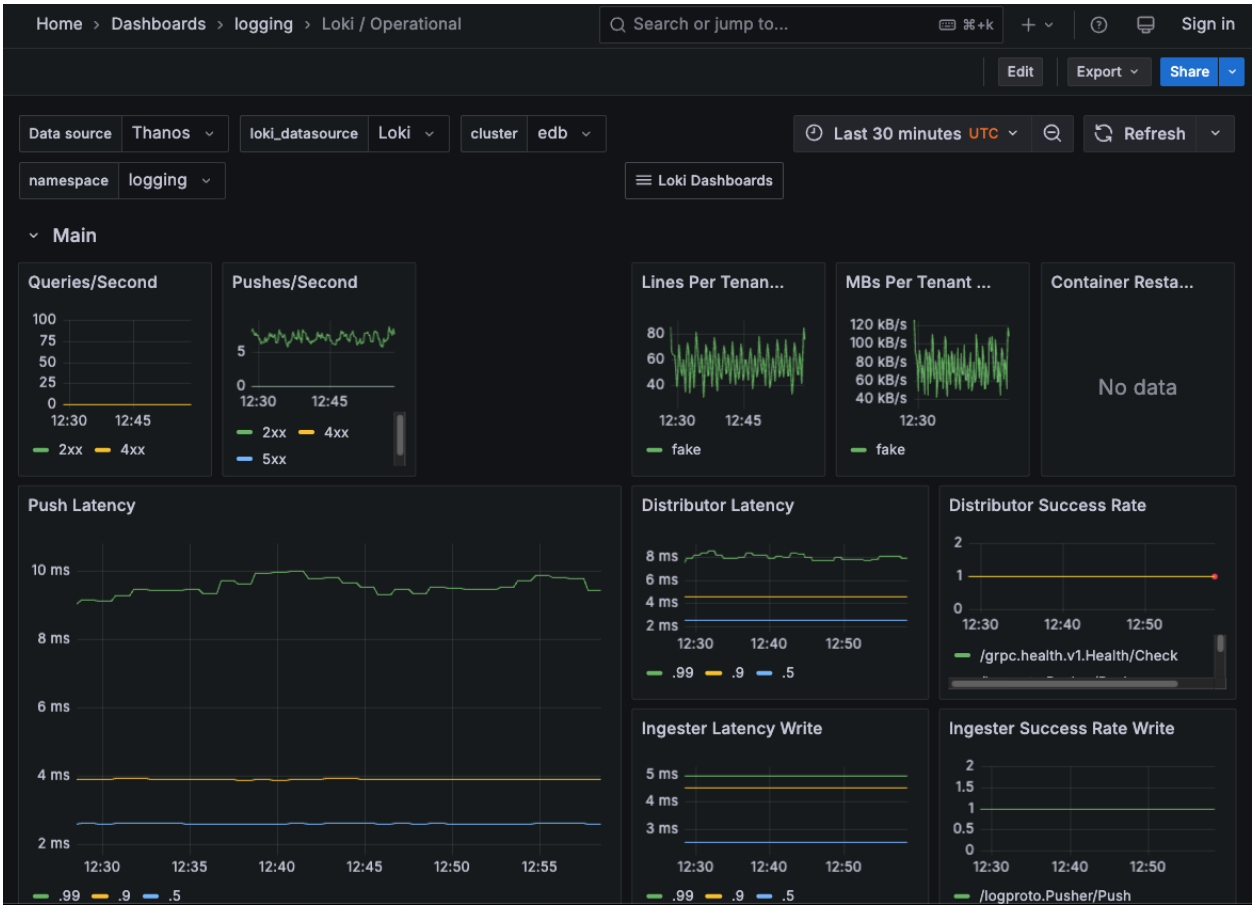


Fig. 21: Loki Operational dashboard

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

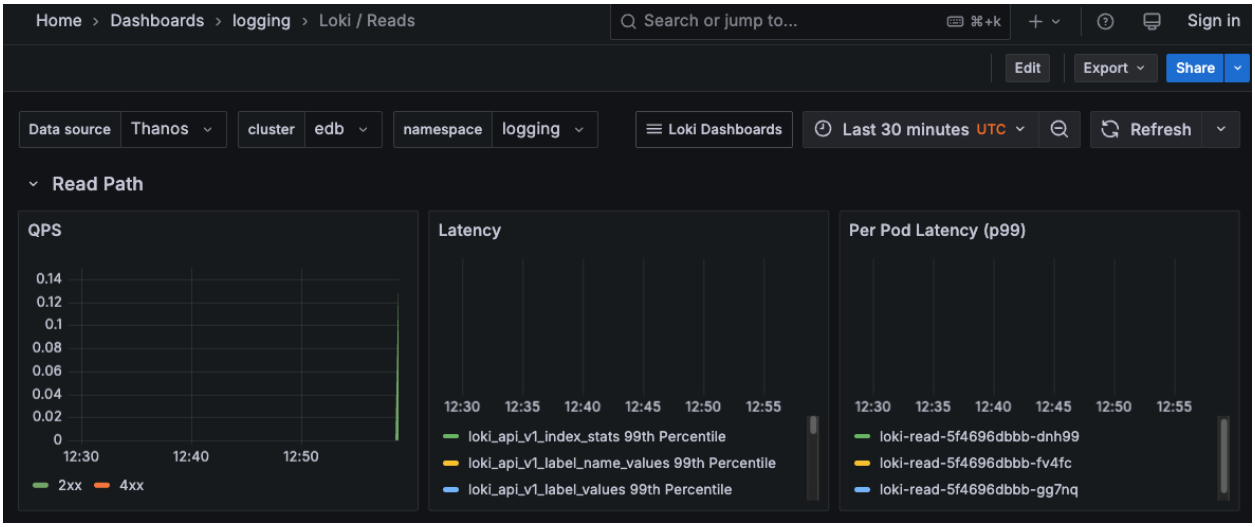


Fig. 22: Loki Reads dashboard

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.

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.

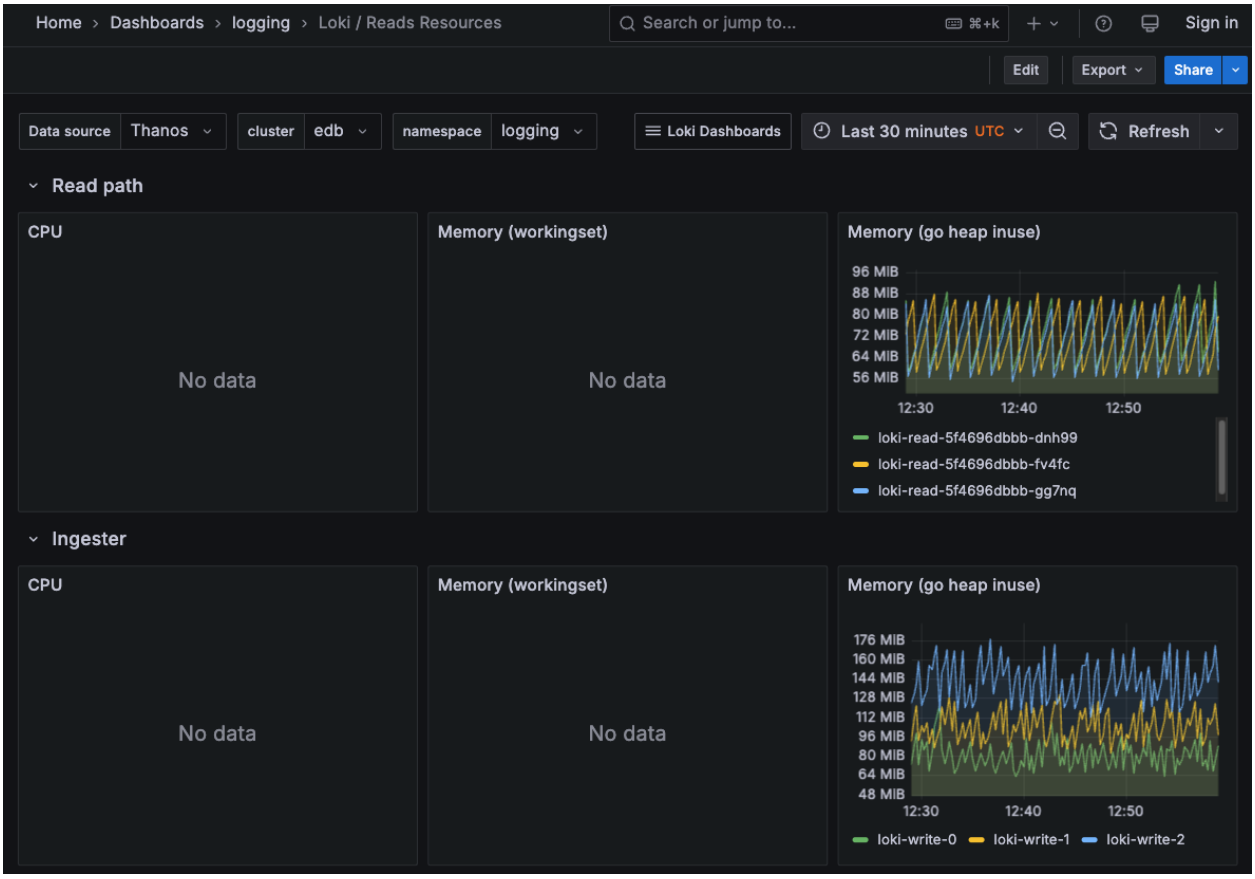


Fig. 23: Loki Reads Resources dashboard

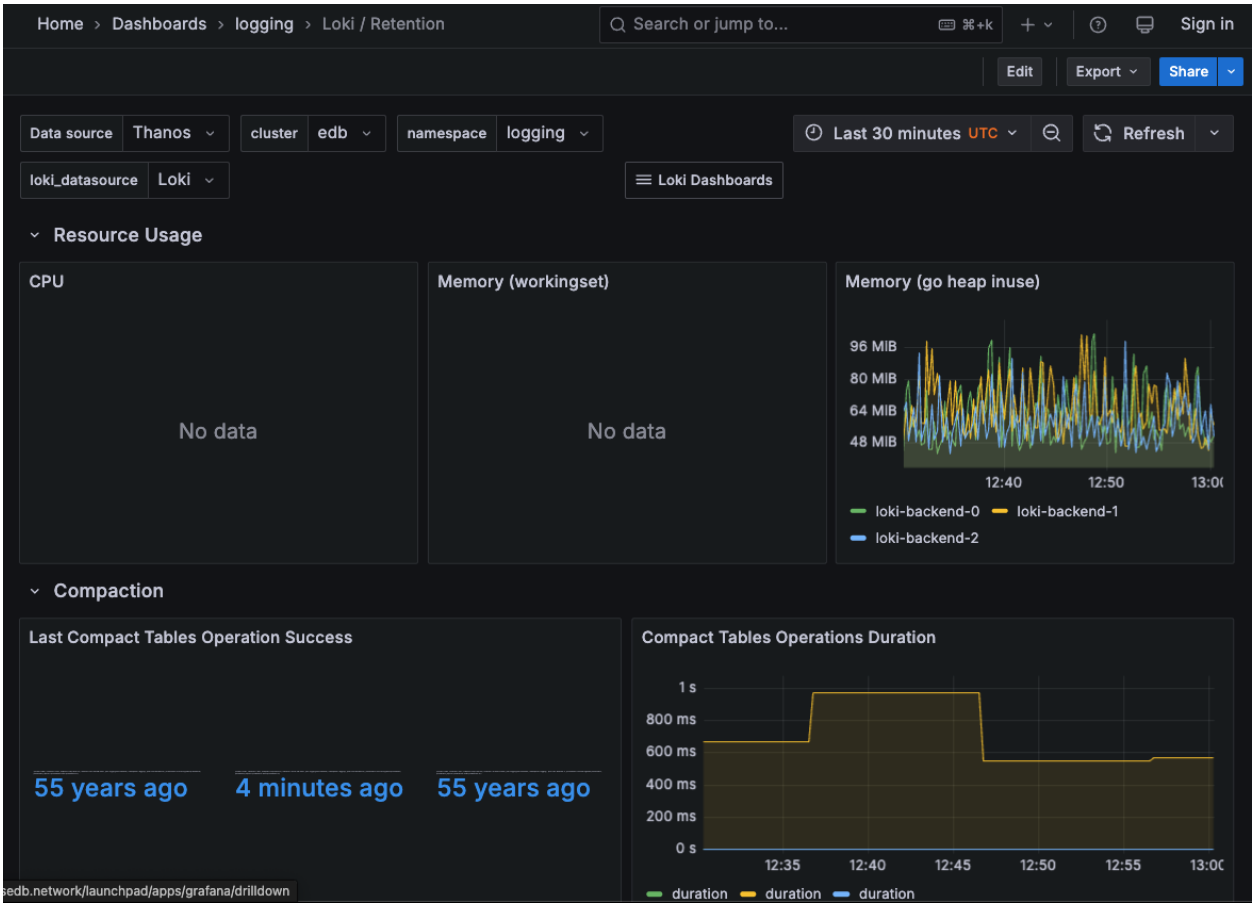


Fig. 24: Loki Retention dashboard

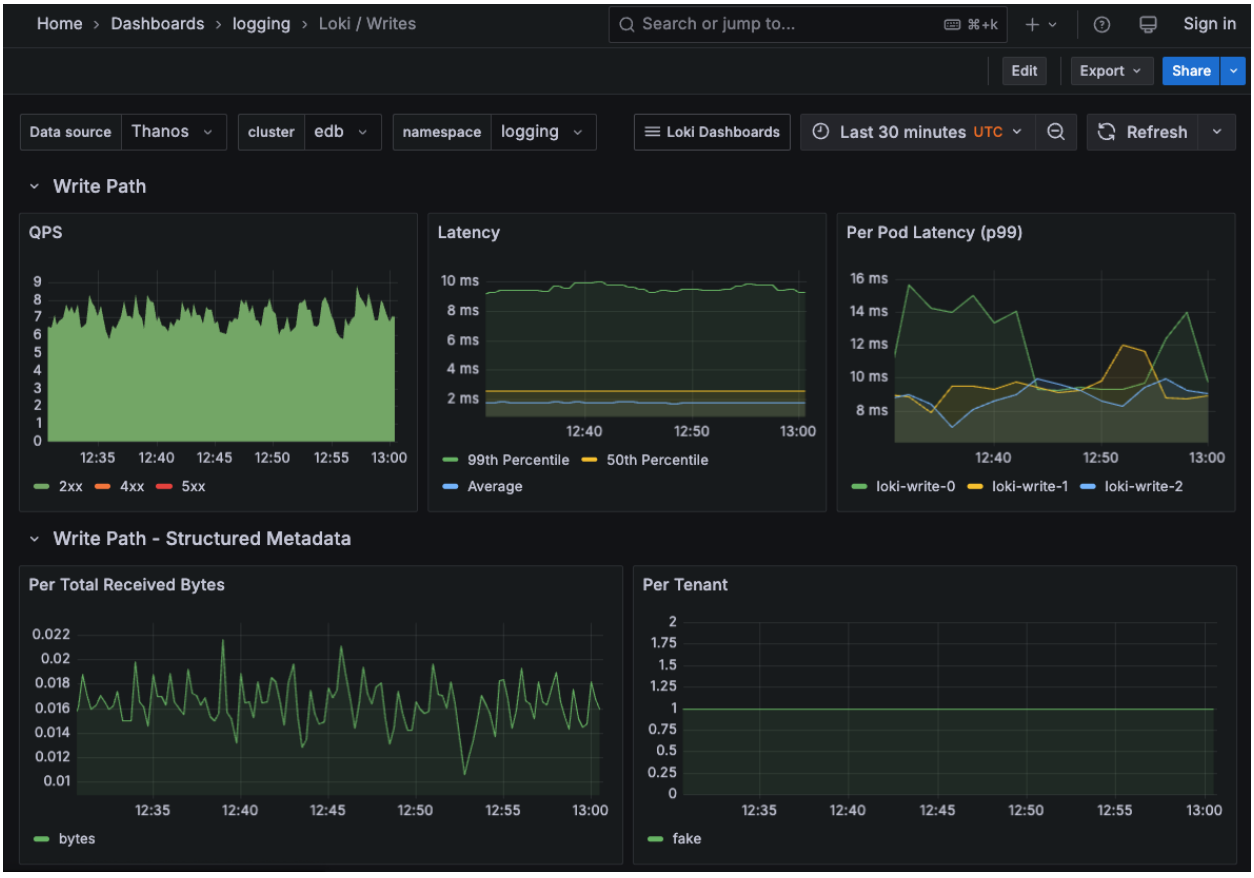


Fig. 25: Loki Writes dashboard

Loki / Writes Resources

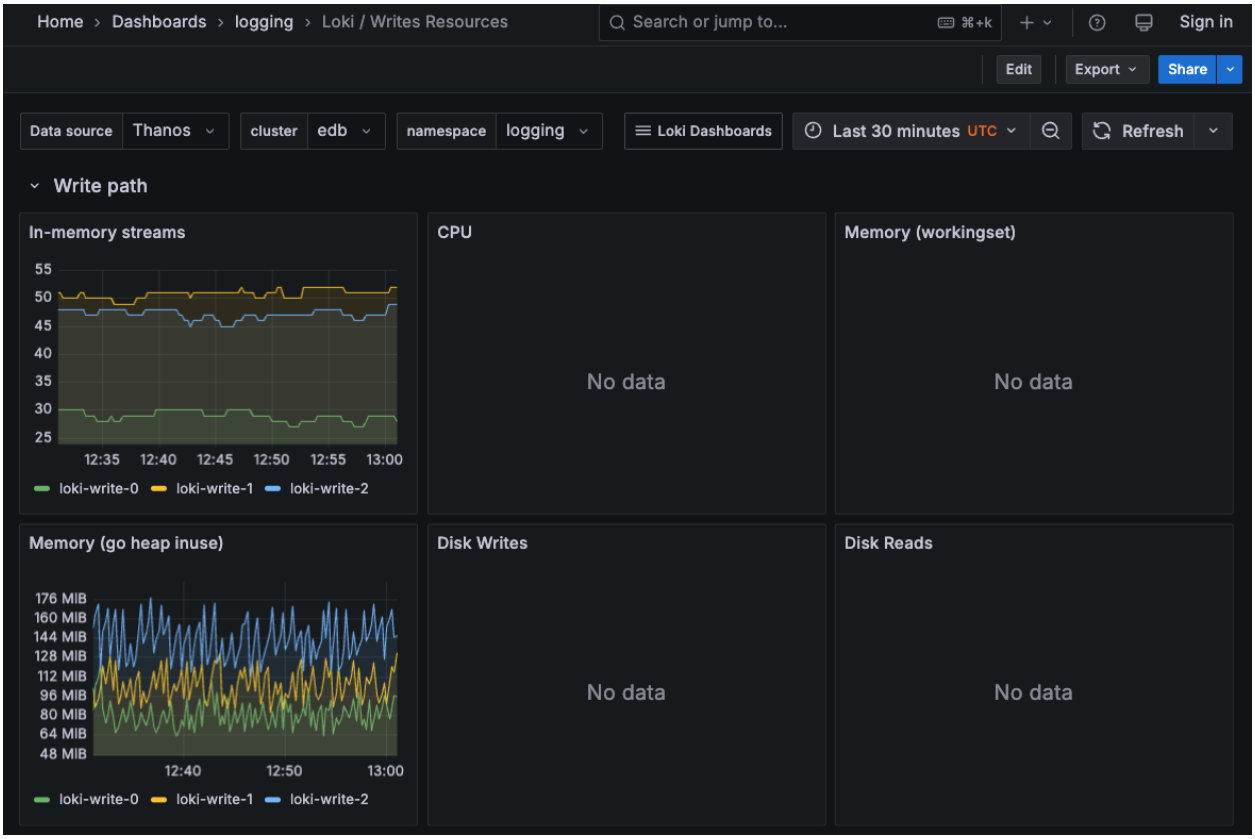


Fig. 26: Loki Writes Resources dashboard

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.

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).



Fig. 27: Loki Data dashboard

Hardware Utilisation

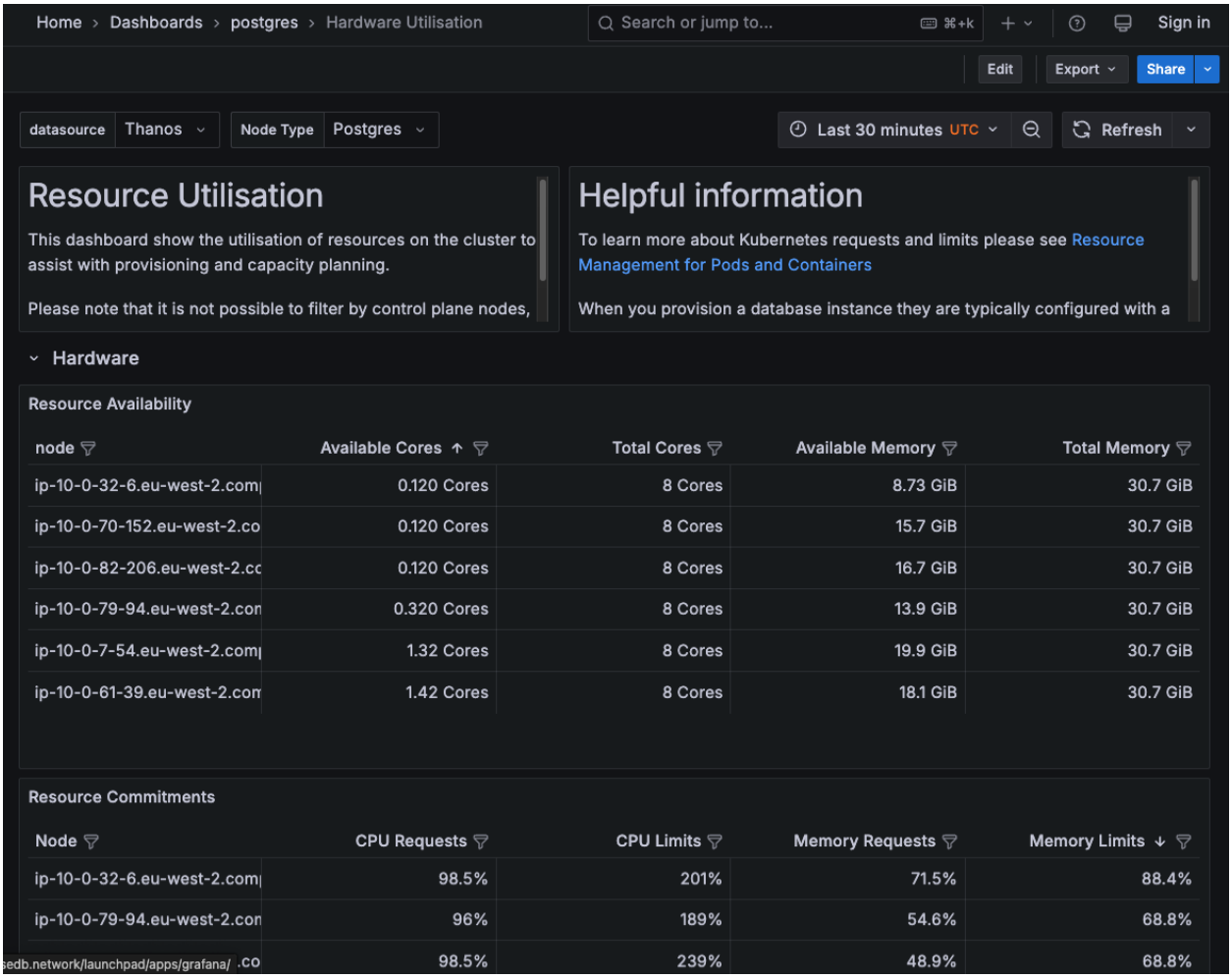


Fig. 28: Postgres Hardware Utilisation dashboard

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.

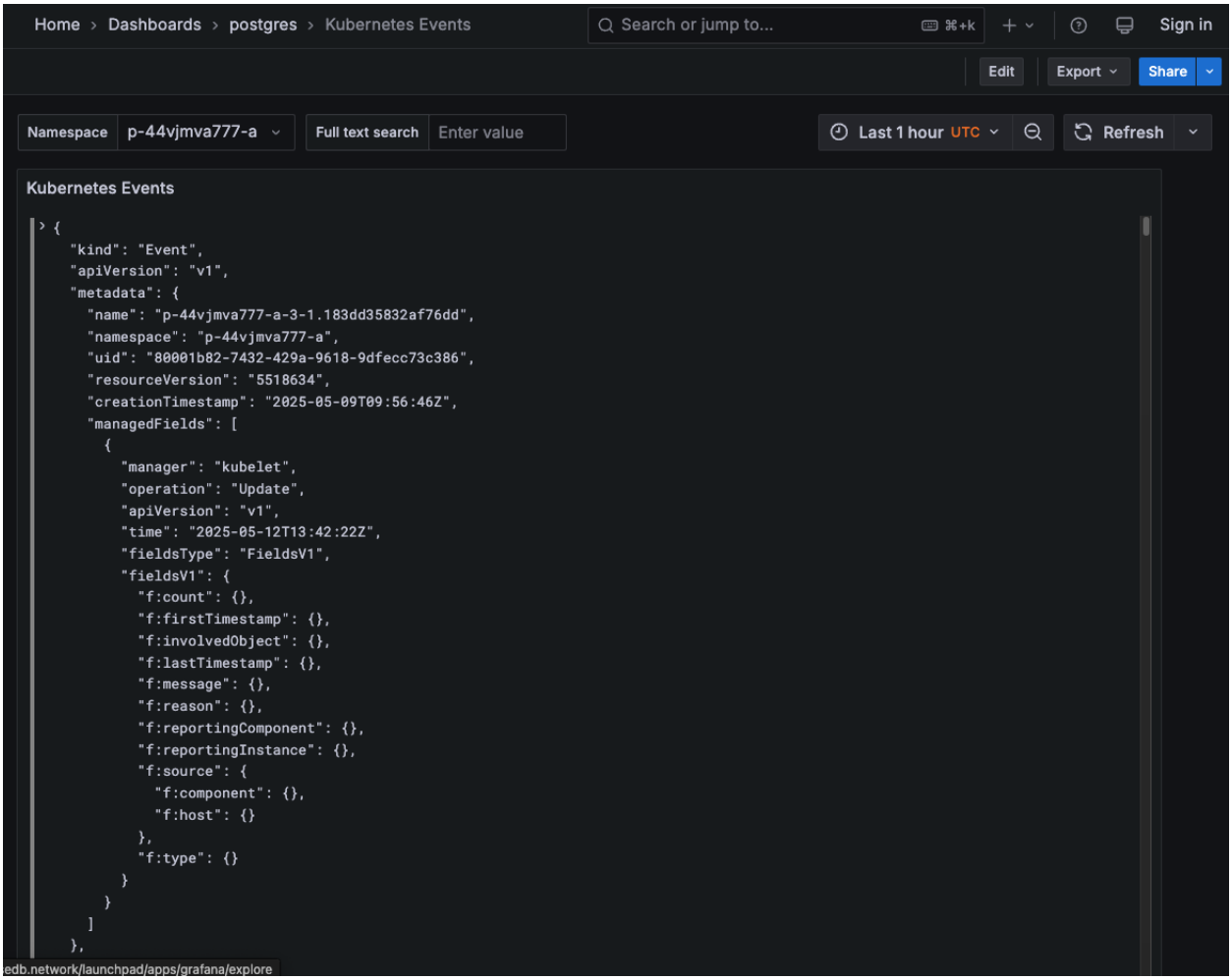


Fig. 29: Postgres Kubernetes Events dashboard

PGD Overview

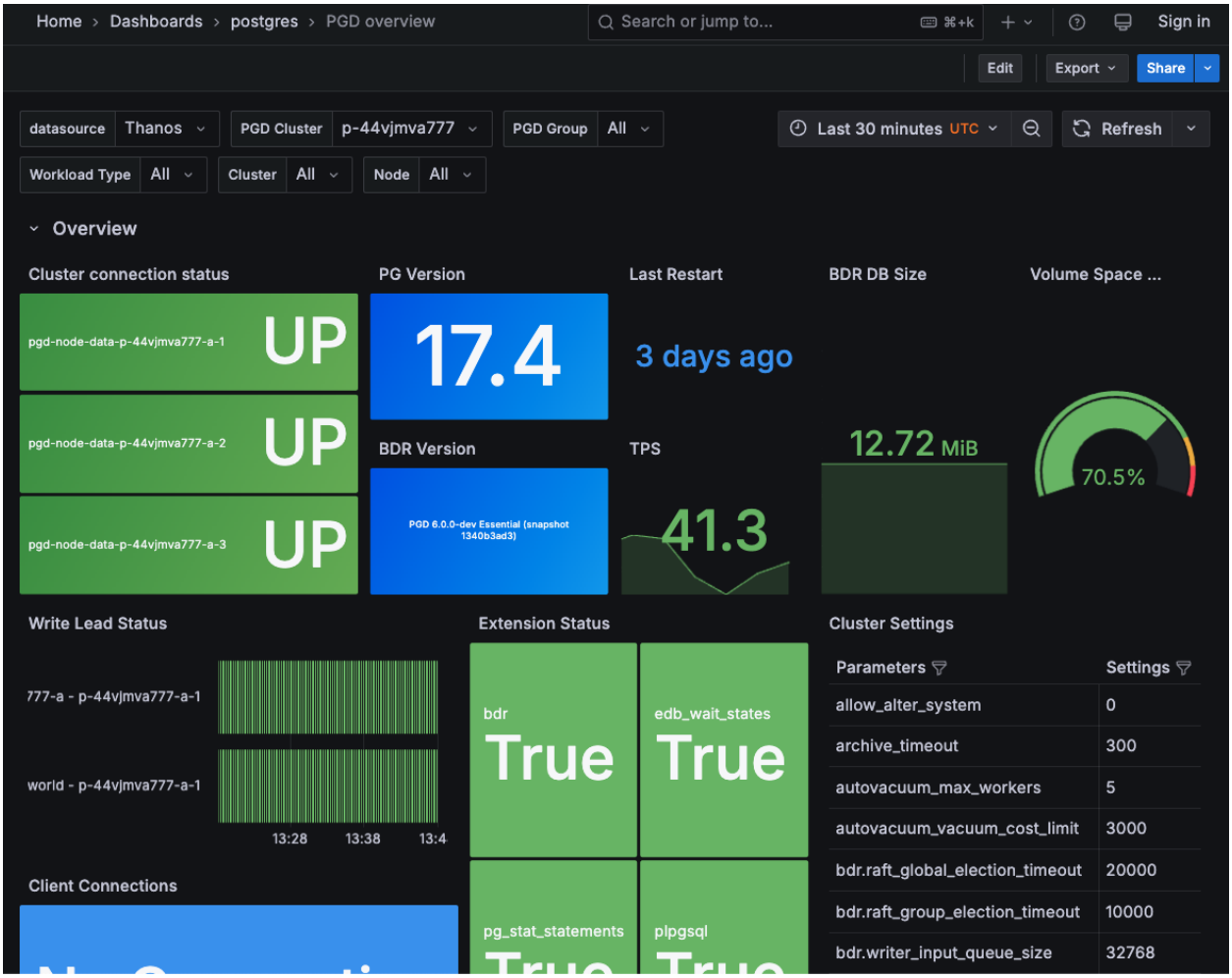


Fig. 30: Postgres PGD Overview dashboard

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.

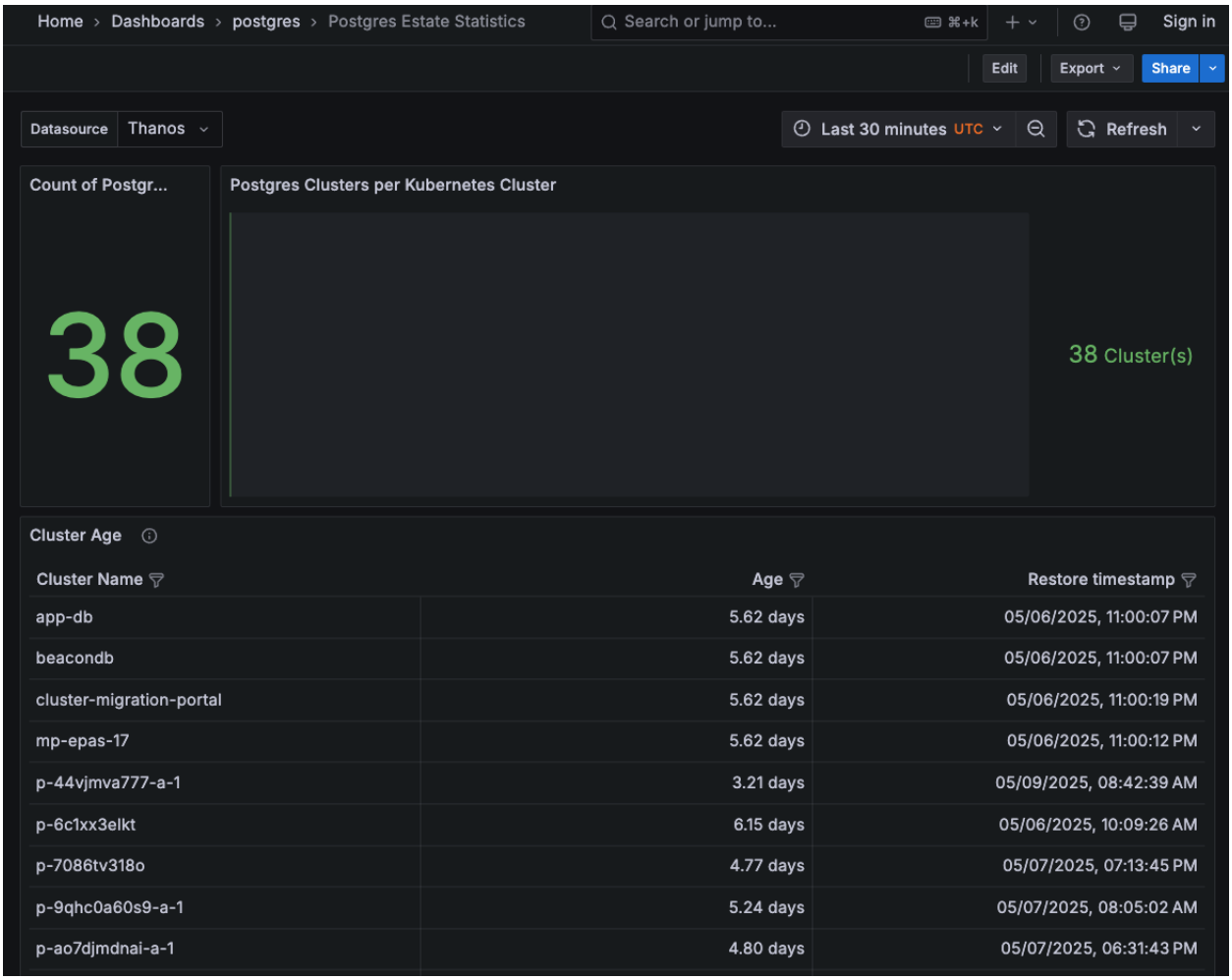


Fig. 31: Postgres Estate Statistics dashboard

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.

Postgres Status

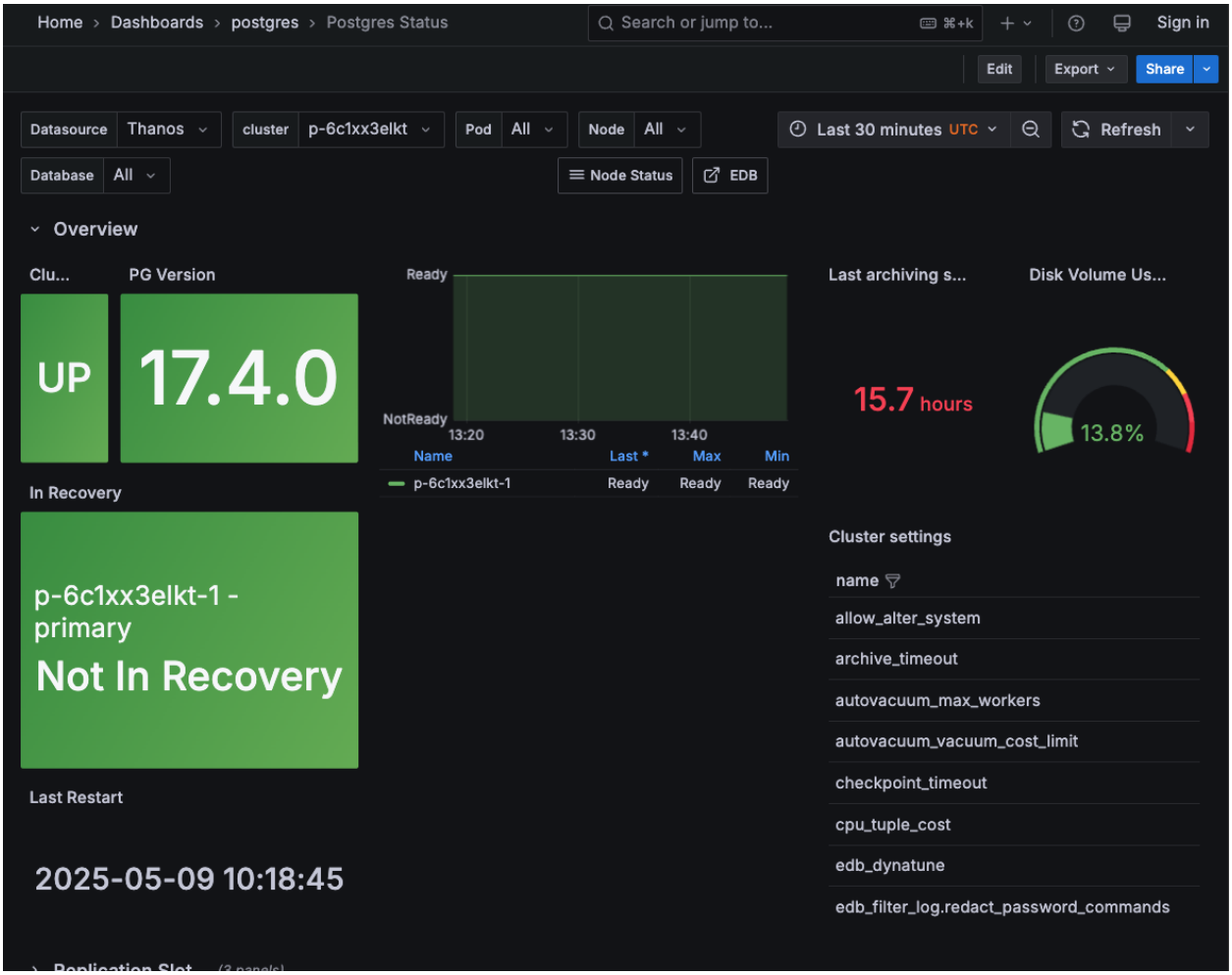


Fig. 32: Postgres Status dashboard

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

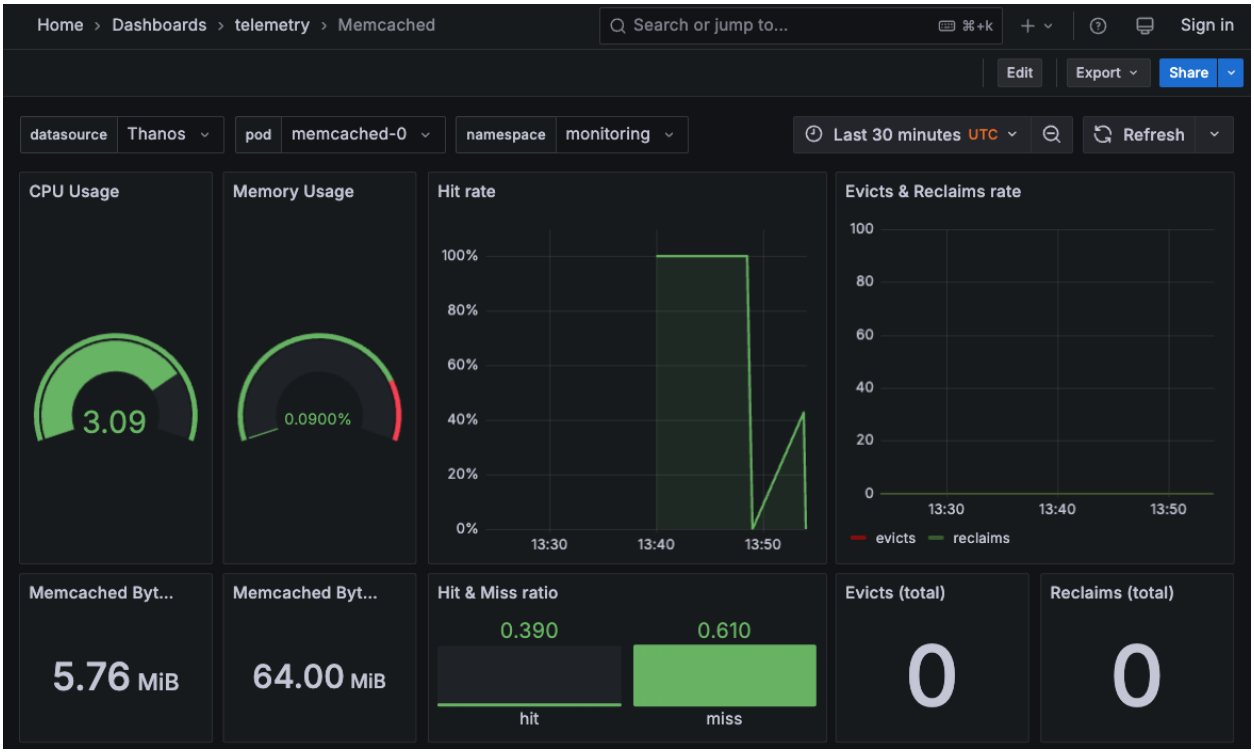


Fig. 33: Telemetry Memcached dashboard

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

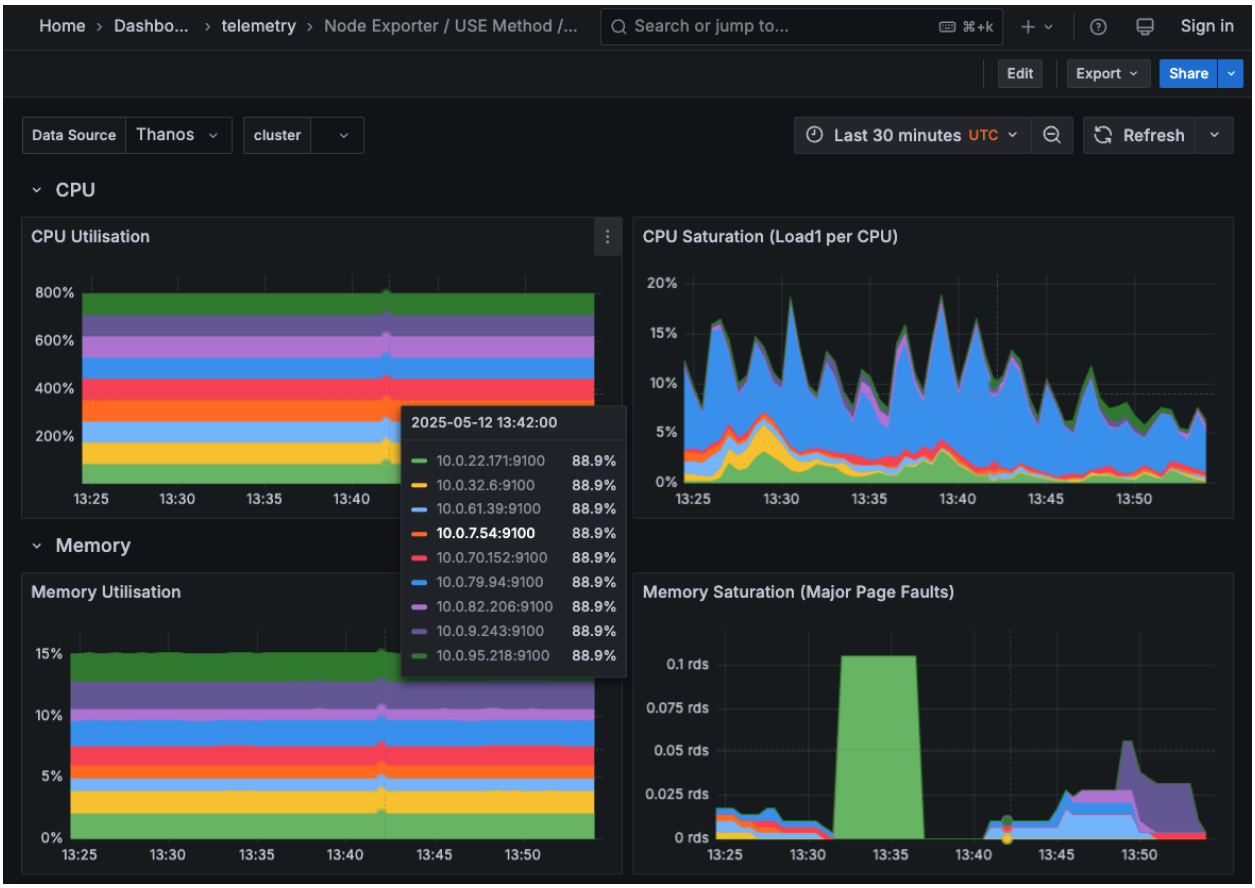


Fig. 34: Telemetry Node Exporter dashboards

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.

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

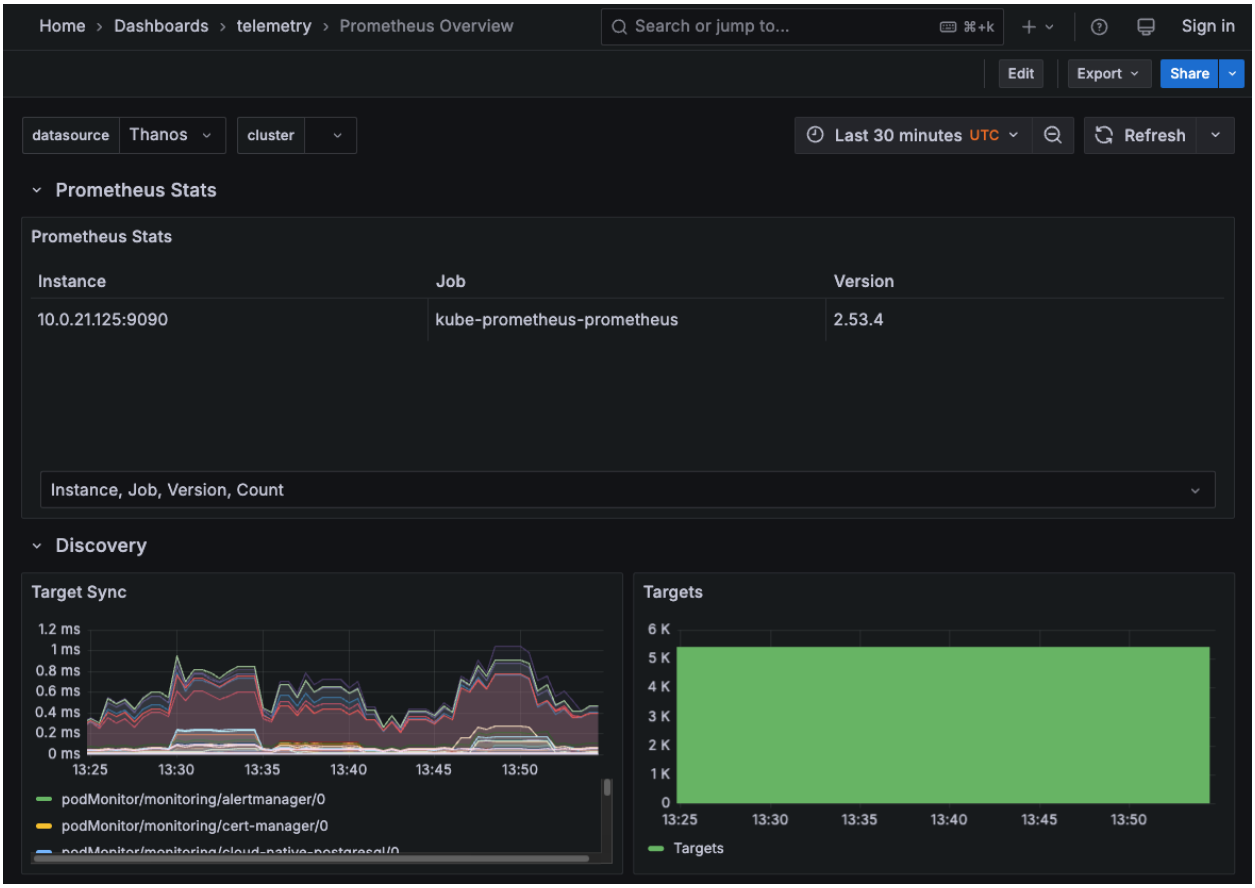


Fig. 35: Telemetry Prometheus dashboard

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.

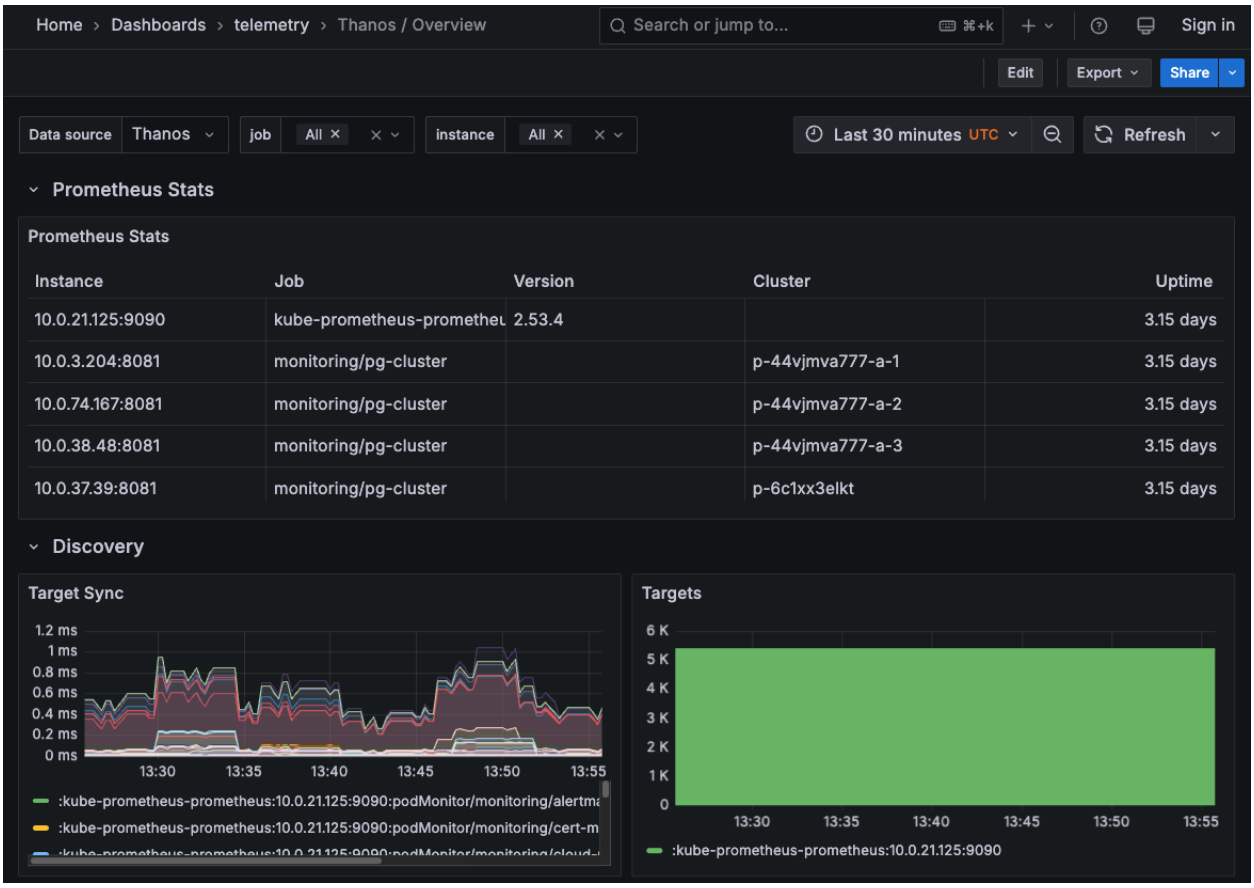


Fig. 36: Telemetry Thanos dashboards

- **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.

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

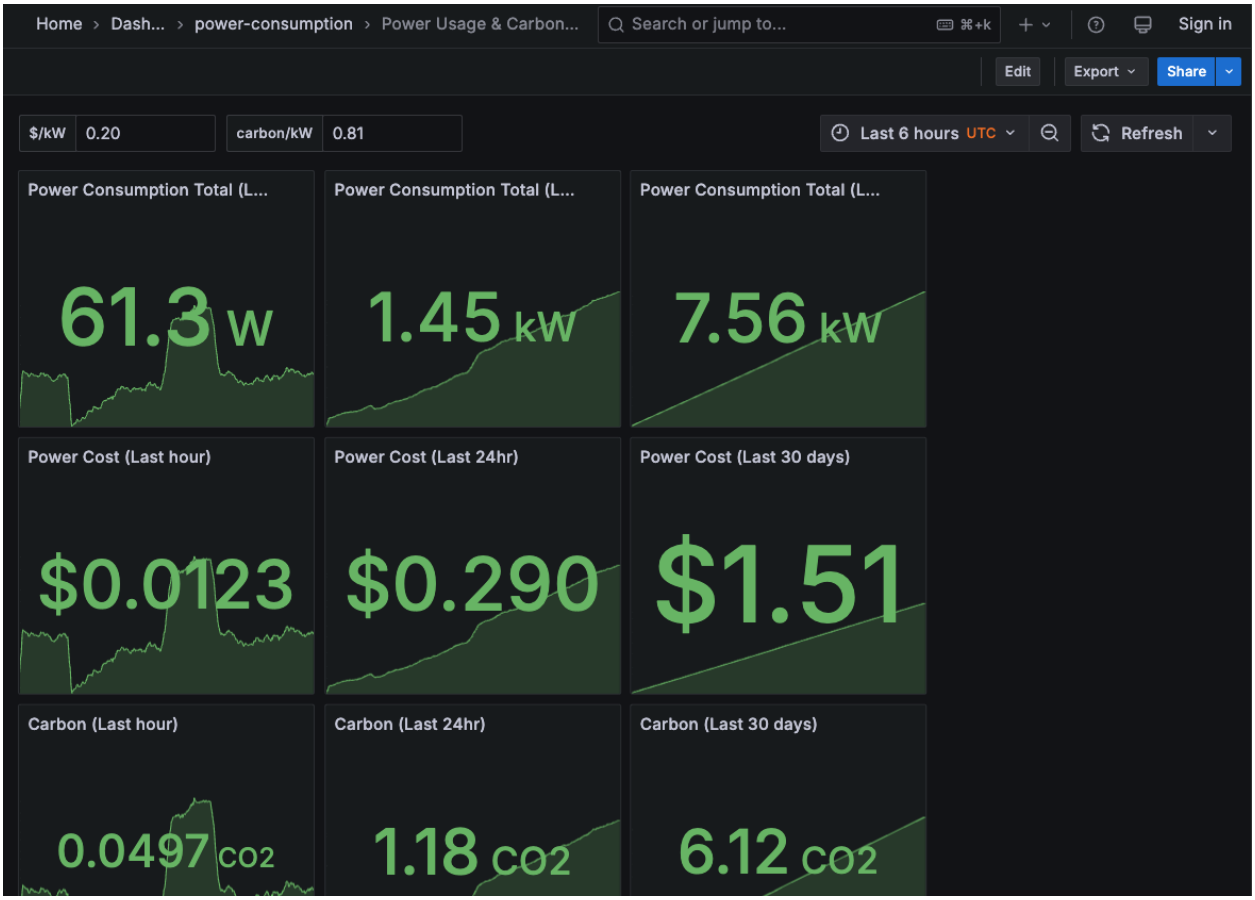


Fig. 37: Power usage and carbon footprint dashboard

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

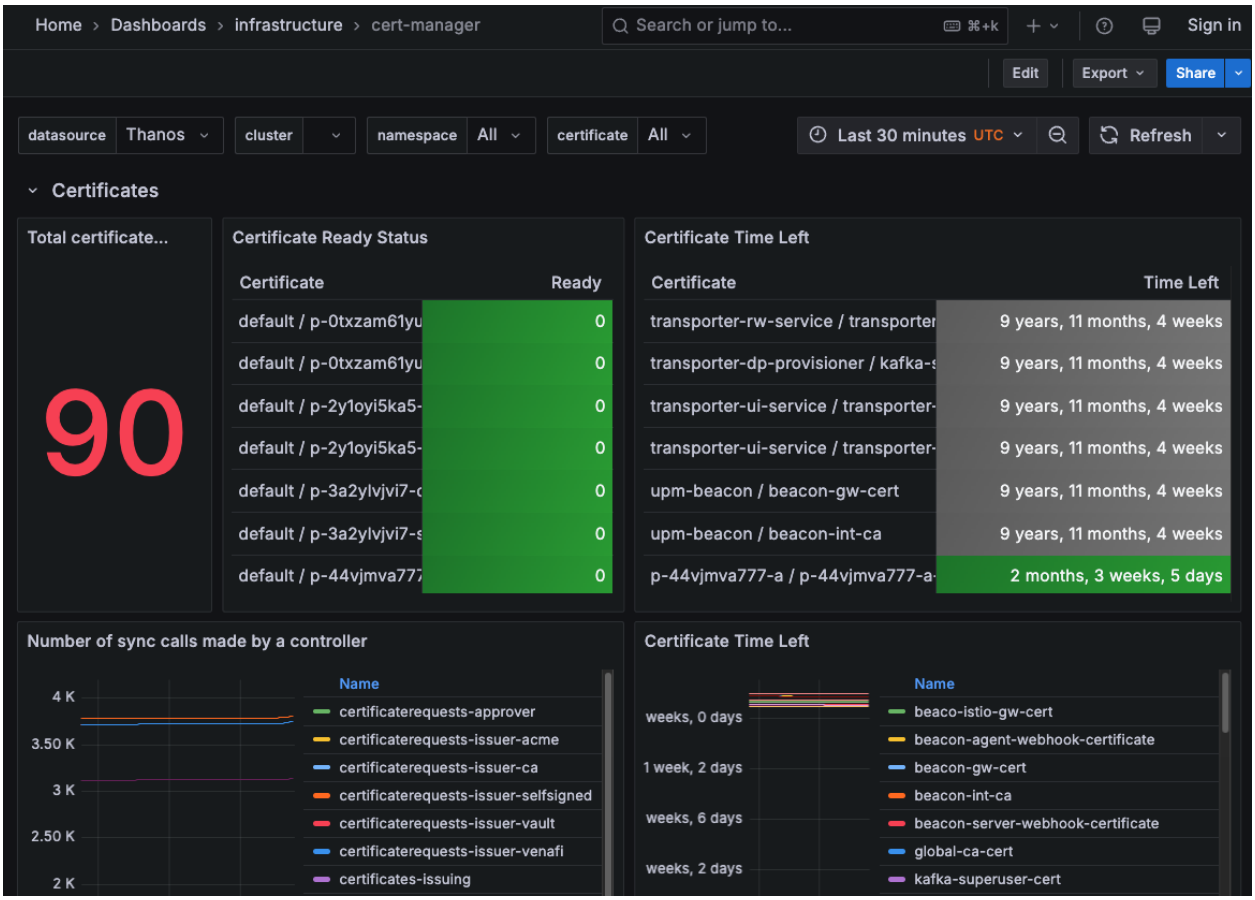


Fig. 38: Cert-manager dashboard

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.

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

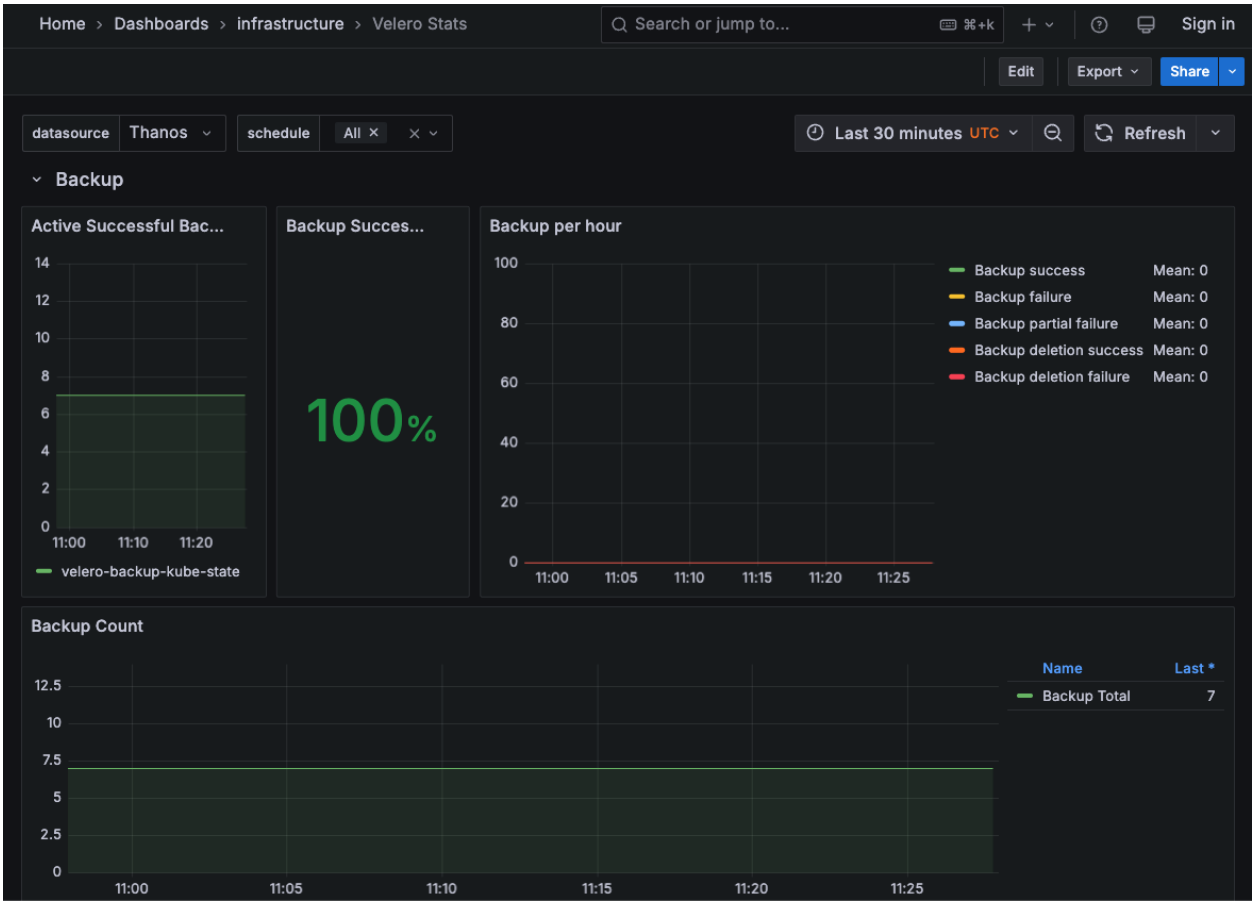


Fig. 39: Velero dashboard

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.

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 :ref:``upm-beacon-agent` memory limits are insufficient in complex environments <*upm-beacon-agent* memory limits are insufficient in complex environments>` 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* .

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, primarily to automate tasks in your organization.

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

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.

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-level notifications.

Webhook

1. In the HM console, from your profile menu, select **Settings**.
2. Select the **Notifications** tab.
3. Select **Webhook** and provide the details:
4. Host URL of the Webhook
5. Method of the Webhook: POST or PUT
6. Optionally, provide one of the following authentication parameters:
    - Basic auth username and password
    - Bearer token
7. Select **Save**.

Pager Duty

Before configuring the Pager Duty notification channel:

- You need access to a Pager Duty account with elevated privileges. - 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 **Pager Duty** and set the integration key (routing key).
4. Select **Save**.

Email

1. In the HM console, from your profile menu, select **Settings**.
2. Select the **Notifications** tab.
3. Select **Email** and enter the details:
4. Email To : Optionally, email group for system email notifications.
5. Email From : Email that's sending the notification.
6. Host : URL of the email service.
7. Port : Port of the email service.
8. Username : Username of the email service.
9. Password : Password of the email service
10. 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 *Webhook* , *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.

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, global, 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 in front of the event name to opt in or out of receiving notifications for a particular event. - Customize the email and HM console (in-app) notifications for the event by using the check box for Email and In-app.

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

These alerts can be configured at host level:

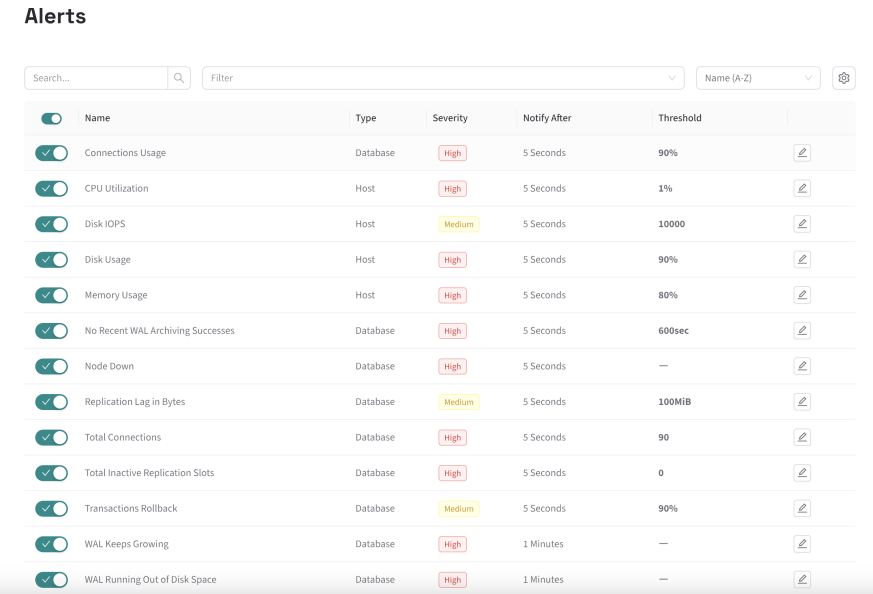


Fig. 40: Alerts

- CPU Utilization
- Disk IOPS
- Disk Usage
- Memory Usage

Upgrading Hybrid Manager

Hybrid Manager (HM) offers a *dual release strategy* : **LTS (Long-Term Support)** and **Innovation Releases (IR)**. Because these streams follow different versioning logic, the supported paths for moving between 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* .

Service availability during upgrades

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.

Supported LTS upgrade paths

LTS releases follow standard semantic versioning (e.g., **1.3.x**). These versions focus on stability and are intended for production environments requiring a long support lifecycle.

Minor version upgrades

Following `major.minor.patch` versioning, you can upgrade from one minor version to the immediate next minor version. You must ensure you are on the **latest patch version** of your current minor release before moving forward.

Supported: `1.3.x` to `1.4.y` (where `x` is the latest available patch at the time of upgrade).

Not supported: `1.3.x` to `1.5.y` (You must upgrade to `1.4.y` first).

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.

Supported IR upgrade paths

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.

Sequential upgrades

Innovation Releases require a **strict sequential upgrade pattern**. You cannot skip a monthly release.

Supported: `2025.11` to `2025.12` .

Not supported: `2025.11` to `2026.01` (You must upgrade to `2025.12` first).

Supported cross-stream upgrade paths

Moving between LTS and Innovation Releases is possible, but subject to specific constraints regarding version numbering and platform limitations.

Innovation Release to LTS

Innovation Releases (IR) eventually converge into a new Long-Term Support (LTS) version. This is known as a **consolidation point**. 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.

Supported: You can only upgrade to a new LTS version from the specific Innovation Release that immediately precedes it.

Unsupported: You cannot move to an LTS release from an older or mid-cycle Innovation Release. If you are on an earlier IR, you must first upgrade sequentially through the IR stream until you reach the consolidation point.

Because consolidation points (that take place twice a year) are scheduled based on feature readiness, they don't occur on fixed dates. 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.

> **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.

Minor upgrade to Hybrid Manager 1.3

Prerequisites:

- Administrative access to the Kubernetes cluster.
- Access to the `values.yaml` file used for the current installation.
- **Required tools:** * `helm` (Latest version recommended)

Upgrade from 1.2.3 → 1.3.0

This release introduces major changes to the underlying storage for monitoring (Thanos) and adds the Spire identity service.

Perform system level changes (EKS Only)

If you are running on **Amazon EKS**, you must update your IAM roles to allow NIM models to access S3.

- **Action:** Edit the IAM role associated with your Service Account.
- **Update:** Add the condition `"system:serviceaccount:m-*:m-*"` to the trust relationship policy.

Update your `values.yaml`

Modify your configuration file to accommodate new parameters.

1. **Update image tags.**

```
bootstrapImageTag: v1.3.0
```

1. **Increase storage for monitoring.**

The Thanos Compactor requires more storage in this release.

```
parameters:
  upm-thanos:
    compactor_pvc_storage: 100Gi
```

1. **Set new mandatory defaults.**

The following values can no longer be empty and must be defined:

```
parameters:
  upm-beaco-ff-base:
    aes_key_rotation_interval: "24h" # Example value
  upm-istio-gateway:
```

(continues on next page)

(continued from previous page)

```
    enable_server_session: true
    session_duration_seconds: 3600 # Example value
```

1. **Configure Spire trust domain.**

Spire is now included in the platform. You must set a unique trust domain for this installation.

```
parameters:
  global:
    trust_domain: <your-unique-trust-domain>
```

Perform the upgrade

```
helm repo update

helm upgrade --install edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap \
   -n edbpgai-bootstrap \
   -f ./values.yaml \
   --version 1.3.0
```

Post-upgrade expectations

- **Portal downtime:** The internal beacon database may take up to **30 minutes** to come online. The portal may be unavailable during this time.
- **Database restarts:** All customer databases will restart during this upgrade as they migrate from `barman-cloud` in-tree to the plugin architecture.
- **Model hosting:** Models previously created via `kubectl` must be re-installed using the UI.
- **Migration Portal:** Databases created prior to 1.3.0 cannot be used as destination targets for migration.

PGAIHM High Availability & Disaster Recovery (HA/DR)

Learn about HA/DR for your Hybrid Manager (HM) clusters, best practices, and more.

- *HA/DR planning and best practices*
- *Hybrid Manager backup and recovery*

HA/DR planning and best practices

Overview

You can configure a multi-data center topology using Hybrid Manager (HM) using EDB PostgreSQL Distributed (PGD) deployments to meet your high availability and disaster recovery (HA/DR) requirements. This topology includes a primary HM and one or more data-only HMs across multiple Kubernetes clusters.

In this HA environment, the primary HM is a standard deployment. The data-only HM is essentially the same HM deployment, with backend adjustments to connect it to the primary HM and to prevent accidental local provisioning. A key component facilitating this connection is the Agent and Beacon server, which handle communication and certificate passing between HMs.

In the event that one cluster goes down, your applications can be pointed to another cluster with synchronized data, ensuring continuous operation. A multi-data center setup provides redundancy in case of a data center outage. A central HM can manage databases across multiple remote Kubernetes clusters, simplifying database administration.

Topologies

You can deploy the multi-HM 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 PGD deployments across two or more Kubernetes clusters.
- **Multiple data-only HMs:** The system can support multiple data-only HMs, allowing a single primary HM to manage databases across many Kubernetes clusters. This is useful for organizations with numerous application teams, each with its own cluster.
- **Three HMs for PGD:** For optimal PGD deployments, especially with two data groups and a witness group, we recommend a three-HM topology. This configuration ensures an odd quorum for proper distributed consensus.

Multi-data center deployments include a primary data center containing the main HM instance and a secondary data center potentially with a data-only HM or other Kubernetes clusters, along with a global object store accessible from multiple data centers.

> **Note**
>
> HM version 1.3 supports deploying multiple HMs in different locations, with a limit of two locations. Other advanced topologies are planned for future versions.

> **Note**
>
> Cross data center HA/DR: High availability and disaster recovery for the control plane across data centers or regions isn't currently supported. Disaster recovery via backup and restore is the only option.

Requirements

The following requirements apply when configuring an HA topology:

- Two or more existing Kubernetes clusters, for example, using Amazon EKS or RedHat OpenShift.
- A fully deployed primary HM on one of the Kubernetes clusters.
- A data-only HM installed on the other Kubernetes clusters, using a provided script for configuration.
- When spanning databases across the multiple clusters, you must use PGD.
- Witness nodes in PGD require a minimum of 10GB of disk space.
- A high-performance networking layer in cloud and multi-region deployments. EKS simplifies this due to its integration with the AWS ecosystem. OpenShift may require more complex networking configurations. Complex networking configurations, especially in OpenShift, can lead to connectivity problems between clusters. Ensure a proper network setup to allow communication between PGD nodes. High network latency can cause PGD clusters to become unstable if Raft settings aren't properly tuned. Symptoms may include slow failure detection and recovery. Adjust Raft-related settings to accommodate higher network latency.

Network latency

There's an inherent trade-off between how quickly 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.

Kubernetes and failovers

Kubernetes handles failovers and redundancy within HM. The following configurations are supported:

- **Single cluster, single zone:** A basic Kubernetes setup with one cluster and one zone, containing multiple nodes and pods.
- **Single cluster, multiple zones:** A Kubernetes cluster divided into multiple zones, where a zone could represent a separate rack or data center. This is referred to as a *stretch cluster*
- **Multi-cluster, multi-location:** Multiple Kubernetes clusters located in different data centers, potentially geographically distant.

These configurations achieve an HA posture by ensuring continuous service availability within a cluster or across zones through redundancy and failover mechanisms. DR is achieved by recovering services and data in the event of a larger failure, such as a data center outage, using backups and data replication across different locations.

Recommendations

The following recommendation apply when using Kubernetes:

- While HM aims to simplify HA/DR, a basic understanding of Kubernetes concepts is beneficial. Understanding Kubernetes concepts like nodes, pods, zones, and clusters is essential for HA/DR planning.
- Adequate resources within the Kubernetes cluster are necessary for pods to be rescheduled in case of node failures.
- Separate HM and PostgreSQL workloads onto different worker nodes.
- Careful planning for storage is crucial, including the type of storage (local, shared, object), its location, redundancy, and backup/restore procedures. We highly recommend Object storage for disaster recovery scenarios, especially for data center failures.
- For stretch clusters, network latency must be within acceptable limits and account for witness node placement. For example, OpenShift recommends less than 5 milliseconds.

Considerations

Keep in mind the following considerations when using Kubernetes:

- If worker nodes use local storage, data isn't automatically failed over when a node fails. Recovery requires restoring from backup.
- Utilizing appropriate storage solutions like shared storage (such as Ceph and SAN) or object storage is critical for data persistence and recovery.
- HM services might not automatically fail over in a multi-zone scenario if their data is tied to a specific zone.
- Data availability in failover scenarios depends on the location and replication of the storage.
- Kubernetes can handle pod rescheduling in the event of node failures, but data recovery depends on the storage configuration.
- Failover to another zone or data center requires careful planning for data replication and application redirection.
- Lack of shared or replicated storage can hinder recovery.
- HM leverages Kubernetes capabilities for HA, but it's important to consider HM-specific configurations.

HM backups and DR

HM multi-data center environments use the following to achieve a robust data protection strategy:

- **Volume snapshots:** Disk-based backups, typically local to the data center.
- **Barman:** Object storage-based backups, which can be stored in a global object store.

Recommendations

Keep in mind the following recommendations regarding HM backups:

- We recommend a global object storage solution for storing Barman backups. This approach can ensure availability of backups across data centers.
- Follow the 3-2-1 backup strategy (three copies of data, two different media, one offsite copy) by using both volume snapshots and Barman backups.
- Consider replicating backup metadata across Kubernetes clusters to simplify restores in disaster recovery scenarios.
- Implement a feature to automate both Barman and volume snapshot backups.
- Develop a disaster recovery plan for HM in addition to your PostgreSQL databases.
- When working with on-premises deployments, consider network configurations for connecting multiple Red Hat OpenShift instances.

Considerations

Keep in mind the following considerations regarding HM backups:

- Restoring backups when HM is unavailable in a secondary data center isn't trivial.
- Backup metadata is stored within the Kubernetes cluster where the backup is initiated. If that cluster is lost, access to the metadata is also lost, complicating restores.
- Restoring backups in a secondary data center without the primary HM may require manual steps.
- A primary HM instance is required for managing backups and restores.
- Automated backups can be sent to only one location. The system doesn't automatically replicate backup definitions across locations.
- You can set up Barman or volume snapshots for automatic backups but not both concurrently through the HM console.
- If the primary HM is unavailable, restoring backups may require CLI interaction and manual configuration.
- Recovering data in a secondary data center requires access to the backup data and the ability to initiate a restore, even without the primary HM.
- Backup metadata is local to each Kubernetes cluster, so backups taken in one cluster aren't automatically visible in another.

Using

The HM console provides tools for configuring and managing backups and restores. You can also use the API to initiate backups to specific locations. You can use kubectl to view backups in a Kubernetes cluster.

For information on how to configure backups and restores of your HM cluster, see *HM backup and recovery* .

Hybrid Manager HA/DR recovery

Disaster can affect Hybrid Manager (HM) and make it unusable. For example, unavailability of the CSP region for an EKS-based appliance or an outage in a datacenter that makes a hardware appliance unusable can occur. A disaster recovery (DR) option allows you to restore your databases at the point-in-time from your available backups.

HM backups are handled with Velero .

There are two possible scenarios for recovering HM:

- **Restore HM to 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 alternative location**: You have two data centers (DC1, DC2), and HM runs in DC1. You need to restore HM from object storage to DC2.

DR scope

The DR procedures address the following:

- The Postgres clusters that you created in the appliance.
- The custom managed storage locations as defined internally in the appliance in the associated s3-compatible storage area.

> **Note**
>
> The DR procedures don't cover the migration components, although you can use them to restore the original appliance *transporter-db* and *migration-db* databases.

RTO and RPO

The ability to do any restore, the associated recovery time objective (RTO), and recovery point objective (RPO) depend on the frequency and size of the backups.

As those factors have significant variation depending on the criticality assigned to the environment and the nature of your data, you don't know RTO and RPO values in advance. We recommend that you properly prepare the environment and perform periodic disaster recovery exercises to ensure your RTO and RPO requirements can be met.

Backup readiness

Each appliance has a linked s3-compatible storage that stores:

- Internal backups (HM appliance data)
- Postgres backups (Postgres database backups)

You can also define custom storage locations in the same bucket to be used in the platform.

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 datacenter where the appliance resides.

> **Tip**
>
> When using an AWS S3 bucket, you can achieve replication by using cross-region replication .

Postgres databases use continuous backup by default, so you can restore them at any point in time. They are limited only by backup lifecycle policies.

Critical appliance data, such as the definition of the Postgres clusters, is stored as Kubernetes objects and included in the Velero backup. By default, this backup happens daily at 23:00, as defined by the default schedule `velero-backup-kube-state`.

If your RPO requires more frequent backups, you can define a new backup schedule.

> **Danger**
>
> Do not modify the default schedule, as it may be overwritten by an appliance software update.

The following example shows a custom schedule to back up the needed resources each hour:

```
apiVersion: velero.io/v1
kind: Schedule
metadata:
  name: custom-velero-backup-kube-state
  namespace: velero
spec:
  schedule: 0
  skipImmediately: false
  template:
    includedNamespaces:
    \-
    includedResources:
    \- storagelocations.biganimal.enterprisedb.com
    \- clusterwrappers.beacon.enterprisedb.com
    \- backupwrappers.beacon.enterprisedb.com
    snapshotVolumes: false
    ttl: 168h
```

DR procedure

The DR procedure is defined as the series of manual steps that you need to take from the deployment of a new appliance to the moment that it's possible to restore your Postgres clusters using the normal restore procedure.

> **Warning**
>
> The procedure is based on the 1.0 release of the appliance and is subject to constant change as the feature set changes. You must constantly test and update it for it to remain valid.

1. Confirm availability of backups

The first step ensures the backups of the unavailable appliance (aka "old backups") are reachable from the new appliance.

You can achieve this in multiple ways:

- Using a replicated bucket as the s3-compatible linked bucket for the new appliance, so the old backups are directly available to the new appliance.

- Copying the backups of the damaged appliance to the linked storage of the new appliance. You must copy the following items:
- Internal EDB backups folder, with the format `edb-internal-backups/\<random-string\>`
- The Postgres clusters backups folder `customer-pg-backups`
- Any folder corresponding to a defined custom storage location

> **Note**
>
> The internal backups folder defined for the new appliance will be different from the older one, as it will have a different <*random-string*> .

2. Preparation steps

Define a recovery backup storage location for Velero

Once you have backups available, you can define a new storage location for Velero so you can restore resources from the damaged appliance backups. This is a read-only location to prevent overwriting or removing those backups.

To define a new storage location, use the following Kubernetes manifest:

```
apiVersion: velero.io/v1
kind: BackupStorageLocation
metadata:
  annotations:
    appliance.enterprisedb.com/s3-prefixes: edb-internal-backups/\<old-backups-ramdom-
↪string\>/velero
  labels:
    appliance.enterprisedb.com/s3-credentials: bound
  name: recovery
  namespace: velero
spec:
  accessMode: ReadOnly
  config:
    insecureSkipTLSVerify: "false"
    region: \<region-of-attached-bucket\>
    s3ForcePathStyle: "true"
  default: false
  objectStorage:
    bucket: \<linked-bucket-name\>
    prefix: edb-internal-backups/\<old-backups-random-string\>/velero
  provider: aws
```

Confirm it using the `velero get backup-locations` command. It must show as `Available` . If the status is not `Available` , check the Velero pod logs for permission errors on the s3 bucket.

Choosing a Velero backup for recovery

Once the old internal Velero backups are available in the recovery storage location, you can list them with the following command:

```
velero get backups \--selector velero.io/storage-location=recovery
```

Typically, you choose the latest available completed backup to recover from. Note the Velero backup name, as well as the date and time (UTC), as both are required for a restore.

Example:

```
NAME                                          STATUS      ERRORS   WARNINGS   CREATED     ␣
↪                    EXPIRES   STORAGE LOCATION   SELECTOR
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.

Additional requirements

The following requirements apply to the recovery procedure:

- The new appliance must be running the same version of the Postgres AI software deployment as the old one.
- The same locations (`locations.beacon.enterprisedb.com` custom resource) used in the old appliance are available in the new one. `Locations` is currently an internal resource created during install and isn't available in the console. `managed-devspatcher` is the default value.
- Container images used to build the clusters in the old appliance are available to the new one.

3. Recovery steps

Restore EDB internal databases (`app-db` and `beacon-db`)

Once the old backups are available, you can restore the EDB internal databases. For each internal database:

1. Save the cluster manifest to a yaml file: `kubectl get \<cluster-name\> \-o yaml \>\ <cluster-name\>.yaml` .
2. Edit the cluster spec in the yaml file so the cluster is created from the backups:

- Replace the **init** section in bootstrap with a **recovery** section:

```
recovery:
  database: \<database name as in the init section\>
  owner: \<owner name as in the init section\>
  source: \<pg-cluster-name\>
  secret:
    name: \<secret name as in the init section\>
  recoveryTarget:
    targetTime: "\<recovery date in YYYY-MM-DD HH:MM:SS+00 format\>"
```

- Add the following section:

```
externalClusters:
\- barmanObjectStore:
    destinationPath:  S3://\<linked-bucket-name\>/edb-internal-backups/\<old-backups-
↪random-string\>/databases
    s3Credentials:
      inheritFromIAMRole: true
```

(continues on next page)

(continued from previous page)

```
    wal:
      maxParallel: 8
  name: \<pg-cluster-name\>
```

- Add the following prefix to the `appliance.enterprisedb.com/s3-prefixes` annotation of the `inheritedMetdata` section (the list is comma separated):

```
edb-internal-backups/\<old-backups-random-string\>/databases/\<db-name\>
```

3. Delete the cluster:

```
kubectl delete cluster \<cluster-name\>)
```

4. Clean the backup area for the cluster:

```
aws s3 rm s3://\<linked-bucket-name\>/edb-internal-backups/\<new-backups-random-
↪string\>/databases/\<pg-cluster-name\>  \--recursive
```

5. Apply the yaml file for the cluster to be re-created: `kubectl apply \-f \<cluster-name\>.yaml`
6. After the cluster is successfully restored and in a healthy state, restart the `accm-server` in the namespace `upm-beaco-ff-base` .

At this point, the portal on the new cluster is available again.

Configure the Velero plugin

The plugin helps restore the Kubernetes resources in a correct state, so only the custom managed storage locations are restored. The Postgres clusters resources are restored as deleted, so you can later restore data as desired.

The plugin configuration is made through a `ConfigMap` , so you must apply this manifest:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: velero-plugin-for-edbpgai
  namespace: velero
  labels:
    velero.io/plugin-config: ""
    enterprisedb.io/edbpgai-plugin: RestoreItemAction
data:
  Number of configure disaster recovery mode, so restored items are transformed as␣
↪needed
  drMode: "true"
  Number of configure a date corresponding to the velero backup date. Note the format\
↪!
  drDate: "\<recovery date in YYYY-MM-DDTHH:MM:SSZ format\>"
  Number of old and new buckets for internal custom storage locations
  oldBucket: \<old-appliance-bucket-name\>
  newBucket: \<new-appliance-bucket-name\>
```

Restore the custom managed storage locations

Configure and apply the following Velero restore resource manifest:

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-1-storagelocations
  namespace: velero
spec:
  Number of Change the backup name to a custom backup name as required
  backupName: \<velero-backup-name\>
  includedResources:
  \- storagelocations.biganimal.enterprisedb.com
  includeClusterResources: true
  labelSelector:
    matchLabels:
      biganimal.enterprisedb.io/reserved-by-biganimal: "false"
```

Restore the cluster wrappers

Configure and apply the following Velero restore resource manifest:

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-2-clusterwrappers
  namespace: velero
spec:
  Number of Change the backup name to a custom backup name as required
  backupName: \<velero-backup-name\>
  includedResources:
  \- clusterwrappers.beacon.enterprisedb.com
  restoreStatus:
    includedResources:
    \- clusterwrappers.beacon.enterprisedb.com
```

Restore the backup wrappers

Configure and apply the following Velero restore resource manifest:

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-3-backupwrappers
  namespace: velero
spec:
  Number of Change the backup name to a custom backup name as required
  backupName: \<velero-backup-name\>
  includedResources:
  \- backupwrappers.beacon.enterprisedb.com
  restoreStatus:
    includedResources:
```

(continues on next page)

(continued from previous page)

```
    \- backupwrappers.beacon.enterprisedb.com
```

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 AI 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. 41: Model options for the Migration Portal AI Copilot

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.

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 AI Factory capabilities.) The self-hosted models process the prompt and generate an answer.

Models: You create and manage these models using the EDB AI 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 \
  --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 AI 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 *AI Factory Architecture 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}" \
```

(continues on next page)

(continued from previous page)

```
    --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 \
    [...]
```

(continues on next page)

(continued from previous page)

```
    --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 \
    --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 (experimental)

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.

Troubleshooting Hybrid Manager

You can troubleshoot common problems in Hybrid Manager (HM).

Error codes

Error Code	Category	Error Message	Explanation
na	Installation	[upm-istio] - failed to install components: [upm-istio]	An error from bootstrap: the I
na	Installation	upm-beaco-ff-base install failed	
na	Installation	upm-thanos installation failed	
na	Installation	Failed to pull image	
na	Installation	Loki pods crash	The loki pods crashed and the
na	Portal Login	No healthy upstream	"No healthy upstream" is retu
401	Portal Login	401 Access Denied	401 error returned when loggi
500	Portal Login	HTTP Error 500	The back ingress gateway can
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 cl

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,␣
↪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:
```

(continues on next page)

(continued from previous page)

```
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:␣
↪dHlwZTogUzMKY29uZmlnOgogIGJ1Y2tldDogYXBwbGlhbmNlLXNoYW50ZXN0LWVrcy1jbHVzdGVyLWVkYi1wb3N0Z3JlcwogIHA

~ ⍰  echo␣
↪dHlwZTogUzMKY29uZmlnOgogIGJ1Y2tldDogYXBwbGlhbmNlLXNoYW50ZXN0LWVrcy1jbHVzdGVyLWVkYi1wb3N0Z3JlcwogIHA
↪| 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.
↪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␣
↪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
```

(continues on next page)

(continued from previous page)

```
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␣
↪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.
```

(continues on next page)

(continued from previous page)

```
↪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="${PORTAL_DOMAIN_NAME}" \
    --set parameters.transporter-rw-service.domain_name="${TRANSPORTER_RW_SERVICE_
↪DOMAIN_NAME}" \
    --set parameters.transporter-dp-agent.rw_service_url="https://${TRANSPORTER_RW_
↪SERVICE_DOMAIN_NAME}/transporter" \
    --set parameters.upm-beacon.server_host="${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" \
```

(continues on next page)

(continued from previous page)

```
    --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:␣
↪aXNzdWVyOiBodHRwczovL3BvcnRhbC5iYXN1cHBvcnQub3JnL2F1dGgKc3RvcmFnZToKICB0eXBlOiBwb3N0Z3JlcwogIGNvbmZ

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           ␣
```

(continues on next page)

(continued from previous page)

```
↪                                                                    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
      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
```

(continues on next page)

(continued from previous page)

```
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(
↪"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.
```

(continues on next page)

(continued from previous page)

```
↪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",
```

(continues on next page)

(continued from previous page)

```
  "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. !!!

Using query recommendations in Hybrid Manager

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Related information

- *Cluster Query Diagnostics*
- *Cluster recommendations*

CHAPTER 6

AI Factory in Hybrid Manager

The **AI Factory workload** in Hybrid Manager brings scalable AI, machine learning, and Gen AI 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 assistants.

With AI Factory in Hybrid Manager, you can:

- Deploy Gen AI assistants and agents 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

Hub quick links: *AI Factory Hub* — *Gen AI Builder on Hybrid Manager* — *Model Serving*

For in-depth architecture and core concepts, see the *AI Factory Hub* .

6.1 Example AI solutions you can build

AI 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 AI Factory architecture.

6.2 Learning paths

Follow structured learning paths to master AI Factory in Hybrid Manager:

- *AI Factory 101* — Introductory concepts and usage
- *AI Factory 201* — Building and managing Gen AI and AI Factory workloads
- *AI Factory 301* — Advanced integration, scaling, governance, and optimization

6.3 Use cases and solutions

We provide detailed guidance and patterns to help you build solutions with AI Factory:

- *Common workloads and use cases* — Proven patterns for AI Factory–powered applications
- *Industry Solutions* — Industry-specific examples and best practices

6.4 AI Factory in Hybrid Manager workloads

Hybrid Manager supports a full set of AI Factory capabilities, tightly integrated into its control plane:

6.4.1 Gen AI workloads

- *Gen AI in Hybrid Manager* — Gen AI capabilities in Hybrid Manager
- *Agent Studio & Builder (consolidated)* — Create assistants, tools, structures, rulesets; manage Knowledge Bases

6.4.2 Model management and serving

- *Model Library Architecture* — 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

6.4.3 Vector Engine

- *Vector Engine* — Integrated vector search and similarity capabilities with Postgres

6.5 Hybrid Manager learn content

In addition to AI Factory content in the hub, Hybrid Manager provides additional Learn content for Hybrid Manager–specific usage:

- *Concepts and architecture*
- *How-To Guides*
- *Learning paths*

6.6 How AI Factory fits within Hybrid Manager

Hybrid Manager is the control plane for your databases, analytics, and AI workloads. AI Factory complements other HM capabilities:

- Databases: Provision and manage Postgres clusters; AI Factory connects to these for vector search, RAG, and application endpoints.
- Analytics: Offload data to Iceberg/Delta and query via SQL; AI Factory's Pipelines and Vector Engine integrate with these datasets for retrieval.
- Operations: Use HM observability and governance to monitor Gen AI, Pipelines, and model serving alongside databases and analytics.

Prerequisites for AI Factory in HM:

- A Hybrid Manager project with AI Factory enabled and required entitlements for Gen AI, Pipelines, Vector Engine, and Model Serving.
- Access to GPU resources where needed for serving. See *GPU resource management* .

6.7 Get started

To begin building with AI Factory in Hybrid Manager:

- Follow the *AI Factory learning paths*
- Explore *Common workloads and use cases* and *industry solutions*
- Read *How-To Guides*
- Deploy your first Gen AI assistant or model using *Gen AI in HM* or *Model Serving*

AI Factory in Hybrid Manager gives you a scalable, secure platform to operationalize AI across your hybrid data estate.

6.7.1 Getting Started

Getting started (AI Factory on HM)

Hub quick links: *AI Factory Hub* — *Gen AI Builder on Hybrid Manager* — *Model Serving*

Hybrid Manager (HM) runs AI Factory workloads in a governed, Kubernetes-native project. Use this page to jump into the canonical hub docs with HM context.

What you can do in HM:

- Build assistants and agents: see *Gen AI Builder on Hybrid Manager*
- Serve models with KServe: see *Model Serving (hub)*
- Create Knowledge Bases: see *Knowledge Base (hub)*

Next steps in HM:

- Enable AI Factory in your project — TODO: link to HM enablement runbook
- Provision GPU node groups if needed — see *Setup GPU*
- Create your first Assistant — see *Create Assistant*

TODOs:

- Add HM UI screenshots for project creation and workload enablement
- Confirm internal enablement guide link

AI Factory Architecture on Hybrid Manager

Architectural Overview

AI Factory deploys as a collection of containerized services within Hybrid Manager's Kubernetes infrastructure, delivering *Sovereign AI on Hybrid Manager* capabilities through integrated *Model Governance* , *inference serving* , and *Gen AI Builder on Hybrid Manager* components. 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 application execution, and a storage layer for model artifacts and *How do I create knowledge bases for RAG?* . These layers integrate through Kubernetes APIs and custom resources, providing unified management while maintaining isolation between projects and workloads.

Core Components

Model Library Architecture

The *Model Library Architecture* 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* .

Gen AI Builder Runtime

Gen AI Builder on Hybrid Manager deploys as a multi-tier application providing visual development and runtime execution for AI applications. The architecture separates concerns between user interface, orchestration logic, and execution environments.

The builder runtime encompasses:

- Web interface pods serving the visual development environment
- Orchestration service pods coordinating agent and tool execution
- Agent executor pods running isolated AI workflows with LLM connections
- Tool service pods providing reusable functions for data access and integration (see *Connecting data into external tools*)
- State management through PostgreSQL with *pgvector and Vector Engine concepts* for embeddings

Each component runs with specific resource allocations and security contexts, ensuring isolation between user workloads while enabling controlled communication through service mesh policies. The orchestration layer manages workflow execution, maintaining conversation state and coordinating between LLM calls, tool invocations, and data retrievals. See: *Gen AI Builder (Assistants & threads)* , *Rulesets* .

Infrastructure Integration

Kubernetes Resource Management

AI Factory leverages Kubernetes resource primitives to ensure predictable performance and fair resource allocation across workloads. Resource management occurs at multiple levels through namespace quotas, pod specifications, and priority classes.

Resource allocation strategies include:

- Namespace-level quotas limiting total GPU and memory consumption per project
- Pod resource requests ensuring minimum guaranteed resources for critical workloads
- Resource limits preventing individual workloads from monopolizing cluster resources
- Priority classes ensuring production inference receives preferential scheduling
- Pod disruption budgets maintaining service availability during cluster operations

The scheduler considers GPU requirements when placing pods, using node selectors and affinity rules to ensure pods land on appropriately equipped nodes. Taints and tolerations prevent non-GPU workloads from consuming GPU-enabled nodes, maximizing availability for AI workloads. For a full setup guide, see *Setup GPU* .

GPU Infrastructure

GPU resources integrate through NVIDIA device plugins and container runtimes, enabling native GPU access from containerized workloads. The infrastructure supports various GPU configurations from single-GPU development instances to multi-GPU production deployments.

GPU management capabilities include:

- Device plugin discovery and advertisement of available GPU resources
- Container runtime configuration enabling CUDA access from pods
- Multi-Instance GPU support for partitioning large GPUs across smaller models
- Time-slicing configurations for development and testing workloads

- GPU feature discovery for automatic node labeling based on capabilities

Resource allocation considers GPU memory requirements, CUDA compute capabilities, and interconnect topology when scheduling workloads. Production deployments typically receive dedicated GPU allocations while development workloads may share GPUs through time-slicing or MIG partitions.

Storage Architecture

AI Factory utilizes object storage for model artifacts, datasets, and knowledge bases, with MinIO or cloud provider services (S3, Azure Blob, GCS) serving as primary storage backends. This architecture separates compute from storage, enabling independent scaling and cost optimization.

Storage integration patterns include:

- Model artifact storage using compressed formats optimized for loading performance
- Dataset storage with partitioning strategies for efficient parallel processing
- Vector embedding storage optimized for similarity search operations
- Checkpoint storage enabling training resumption and model versioning
- Cache layers reducing repeated downloads of frequently accessed models

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.

Network Architecture

Service Communication

Internal service communication occurs through Kubernetes service discovery with DNS resolution providing stable endpoints for inter-service calls. The service mesh adds security and observability layers without requiring application changes.

Communication patterns include:

- Service-to-service calls using cluster-local DNS names
- Load balancing across multiple pod replicas using service endpoints
- Circuit breaking preventing cascade failures during service degradation
- Retry mechanisms with exponential backoff for transient failures
- Timeout configurations preventing indefinite request blocking

Network policies enforce communication boundaries, restricting traffic flow between namespaces and preventing unauthorized service access. These policies implement zero-trust networking principles, requiring explicit authorization for all inter-service communication.

External Access

External access to AI services occurs through controlled ingress points with authentication and rate limiting. Multiple access patterns support different client requirements while maintaining security boundaries.

Access mechanisms include:

- Ingress controllers terminating TLS and routing to backend services
- API gateways providing authentication, authorization, and rate limiting

- Service mesh gateways enabling fine-grained traffic management
- Load balancers distributing traffic across available instances
- WebSocket support for streaming inference responses

Authentication integrates with enterprise identity providers through OAuth2/OIDC protocols, while API keys provide programmatic access for service accounts. Rate limiting prevents resource exhaustion while ensuring fair access across clients.

High Availability Considerations

Component Redundancy

Critical services deploy with redundancy to ensure availability during failures or maintenance operations. Redundancy strategies vary based on component statefulness and performance requirements.

Availability patterns include:

- Multi-replica deployments for stateless inference servers
- Active-passive configurations for stateful orchestration services
- Leader election mechanisms for components requiring single-writer semantics
- Geographic distribution across availability zones where applicable
- Rolling update strategies maintaining service availability during upgrades

Health monitoring detects component failures triggering automatic recovery procedures. Liveness probes restart unhealthy containers while readiness probes prevent traffic routing to pods still initializing. The scheduler automatically replaces pods on failed nodes, maintaining desired replica counts.

Data Durability

Data durability relies on underlying storage system guarantees with additional application-level protections for critical data. Object storage provides high durability for model artifacts and datasets while PostgreSQL replication ensures metadata availability.

Durability mechanisms include:

- Object storage replication across multiple availability zones
- PostgreSQL streaming replication for metadata databases
- Backup procedures for configuration and state data
- Version control for model artifacts and application code
- Disaster recovery procedures for catastrophic failures

Operational Monitoring

Observability Stack

The platform provides comprehensive observability through metrics, logs, and traces collected from all AI components. This observability enables proactive issue detection and performance optimization.

Monitoring capabilities include:

- Prometheus metrics collection from inference servers and application pods (see *Monitor InferenceService*)
- Grafana dashboards visualizing system health and performance trends

- Centralized logging aggregating container logs and application output
- Distributed tracing capturing request flows across services
- Alert rules triggering notifications for anomalous conditions

Custom metrics track AI-specific indicators including inference latency, token generation rates, GPU memory usage, and model accuracy drift. These metrics support both operational monitoring and capacity planning decisions.

Performance Analysis

Performance monitoring focuses on key indicators affecting user experience and resource efficiency. Metrics collection occurs at multiple granularities from system-wide aggregates to individual request traces.

Performance indicators include:

- End-to-end request latency from client to response
- Model inference time excluding network and queuing delays
- GPU utilization indicating resource efficiency
- Memory consumption patterns identifying optimization opportunities
- Queue depths revealing capacity constraints

Analysis tools correlate metrics across layers, identifying bottlenecks and optimization opportunities. This analysis informs scaling decisions, resource allocation adjustments, and architecture improvements.

Sovereign AI on Hybrid Manager

Hub quick links: *Sovereign AI (hub) — AI Factory Hub*

What Sovereign AI means in Hybrid Manager

With **AI Factory on Hybrid Manager**, AI workloads are **co-located with your data**:

- **Your environment.** Assistants, model serving, and vector search run directly inside your Hybrid Manager project and Kubernetes cluster.
- **Your data.** Source documents, embeddings, and conversation history 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.** Conversation threads and model logs are observable through the same platform you use for Postgres and analytics.

Connectivity options

- **Standard (connected).** Most environments run with internet connectivity to pull container images, updates, or libraries. Even when connected, **data stays inside your project** — unlike SaaS AI, it is not shared with the provider.
- **Air-gapped.** For high-security deployments, Hybrid Manager can be operated without internet access. You preload model images and registries, and all workloads stay entirely within your controlled cluster.

Contrast with external AI services

- **External LLM services** (e.g., OpenAI, Anthropic, Cohere): your prompts, documents, and embeddings leave your network and are processed in a third-party environment.
- **Hybrid Manager:** all retrieval, generation, and storage happen inside your project and infrastructure. Models are served next to your data, ensuring that sensitive content never leaves your control. Air-gapped deployments further guarantee that no external connectivity is required.

Learn more

- *Sovereign AI (hub)* — principles and patterns.
- *AI Factory Hub* — deeper guides on assistants, knowledge bases, and model serving.

Use Cases and Personas for AI Factory on Hybrid Manager

Overview

AI 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.

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 *Knowledge Base Creation* 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 60-70% 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

- Thread tracking for conversation history and escalation workflows

See *Customer Service Agent 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 *RAG Pipeline Documentation* 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.

User Personas

Platform Engineers

Platform engineers manage AI 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 (AI Factory on HM)*

Data Scientists

Data scientists leverage AI 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 bases following *Knowledge Base Management*
2. Configure retrievers using *Retriever Creation Guide*
3. Deploy assistants through *Assistant Configuration*

Application Developers

Developers integrate AI capabilities into business applications through APIs and SDKs.

Primary Responsibilities

- Integrate model endpoints into applications
- Implement conversation management using threads
- Build custom tools for assistant capabilities
- Handle error conditions and fallback strategies

Key Workflows

1. Access model endpoints via *KServe Integration*
2. Implement conversation tracking using *Thread Management*
3. Create custom tools following *Tool Development Guide*

Business Analysts

Business analysts use Gen AI Builder's visual interfaces to create AI applications without coding expertise.

Primary Responsibilities

- Define assistant behaviors and knowledge domains
- Configure data source connections
- Test and validate assistant responses
- Monitor usage patterns and effectiveness

Key Workflows

1. Design assistants using visual builder interface

2. Connect data sources through configuration wizards
3. Test interactions and refine behaviors iteratively

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

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

Getting Started by Use Case

For Knowledge Management

1. Review *Knowledge bases in the asset library*
2. Follow *Data Source Integration*
3. Implement *Hybrid Search*

For Customer Service

1. Start with *Assistant Concepts*
2. Configure *Rulesets* for behavior control
3. Deploy using *Quickstart UI*

For API Development

1. Deploy models via *InferenceService Creation*
2. Access endpoints following *Python Client Guide*
3. Implement monitoring per *Observability Documentation*

Additional Resources

Architecture References

- *AI Factory Architecture on Hybrid Manager*

Implementation Guides

Related Documentation

- *Hybrid Manager Overview*
- *Analytics Personas*

Frequently Asked Questions - AI Factory on Hybrid Manager

Table of Contents

- *Platform Capabilities*
- *Installation and Setup*
- *Model Management*
- *Gen AI Builder Runtime*
- *Operations and Maintenance*

- *Troubleshooting*
- *Security and Compliance*
- *Performance and Scaling*

Platform Capabilities

What types of models does HM 1.3 support?

Hybrid Manager 1.3 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 HM 1.3. 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 AI Factory from cloud AI services?

AI 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

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 (Llama 3.3 70B): 4 x L40S GPUs
- Text embeddings: 1 x L40S GPU

- Text reranking: 1 x L40S GPU
- Vision models: 1 x L40S GPU

Consult *Prerequisites for AI 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 AI 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-Gap Configuration* .

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 in HM 1.3:

- **llama-3.3-nemotron-super-49b**: Advanced reasoning and chat
- **llama-3.2-nemoretriever-300m-embed**: Text embeddings
- **llama-3.2-nv-rerankqa-1b**: 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

Gen AI Builder

Where do I find Gen AI documentation?

Primary Gen AI resources:

- *Gen AI Builder on Hybrid Manager* - Comprehensive reference
- *Concepts and architecture* - Component overview
- *AI Factory Architecture on Hybrid Manager* - System design

How do I create knowledge bases for RAG?

Knowledge base creation process:

1. Configure data sources (databases, documents, APIs)
2. Process content through NIM embedding models
3. Store vectors in PostgreSQL with pgvector
4. Configure retrieval strategies for search

Implementation guide: *Knowledge Base Creation* .

What are assistants and how do they work?

Assistants orchestrate interactions between users, knowledge bases, and external systems. They leverage NIM models for generation while maintaining conversation context through threads. Assistants differ from simple chatbots by incorporating retrieval, tool use, and structured reasoning capabilities.

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 (AI 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

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?

Retrieval troubleshooting:

1. **Embedding quality**: Verify appropriate NIM embedding model selection
2. **Document chunking**: Adjust chunk size and overlap parameters
3. **Search parameters**: Tune top-k and similarity thresholds
4. **Index completeness**: Confirm all documents processed successfully

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
- Assistant conversations via thread tracking
- Administrative operations on AI resources

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 HM 1.3 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.

Additional Resources

Getting Started

- *Prerequisites for AI Factory on Hybrid Manager*
- *AI Factory Architecture on Hybrid Manager*

Implementation Guides

- *Model Serving*
- *Gen AI Builder on Hybrid Manager*
- *Knowledge bases in the asset library*

Troubleshooting Resources

- *Model Verification*
-

For issues not addressed here, contact EDB support or consult the *AI Factory Hub* .

GPU Recommendations for Default NIM Models

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.
- PG.AI GenAI Builder (containerized Griptape) for building agentic AI assistants.

Default NIM Models

Model type	NIM model	NVIDIA NIM documented resource requirements
Text completion	llama-3.3-70b-instruct	4 × L40S
Text embeddings	arctic-embed-l	1 × L40S
Image embeddings	nvclip	1 × L40S
OCR	paddleocr	1 × L40S
Text reranking	llama-3.2-nv-rerankqa-1b-v2	1 × L40S

Minimum GPU Requirement

Based on the default models above, the minimum to run them concurrently is 8 × L40S GPUs.

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.

AI Factory Quickstart

Hub quick links: *Gen AI Builder on Hybrid Manager* — *Knowledge Bases and pipelines* — *Model Serving*

What you'll build

This quickstart gets you from zero to a working Gen AI assistant that answers questions from a Knowledge Base, plus a deployed model endpoint you can call from applications.

Prerequisites

- Prepare your deployment's GPU: *Setup GPU*
- Familiarity with the Gen AI concepts and Pipelines: *Gen AI Builder on Hybrid Manager* , *Knowledge Bases and pipelines*

Step 1 — Create a Knowledge Base

Create a Knowledge Base and ingest your content so the assistant can retrieve context:

- *Step 1 — Create a Knowledge Base*
- (Optional) Review Knowledge Base concepts: *Knowledge bases in the asset library*

Step 2 — Create an Assistant

Build a conversational assistant and attach your Knowledge Base:

- *Step 2 — Create an Assistant*
- (Optional) Review assistants: *What are assistants and how do they work?*

Step 3 — Deploy a model endpoint

Deploy a model using KServe so your applications can call it directly:

- *Deploy NIM containers*
- *Create an InferenceService*
- (Optional) Customize runtime: *Configure ServingRuntime*

Step 4 — Call the endpoint from an app

Test your deployed model with a simple client and learn how to reach endpoints:

- *Quickstart (Python)*
- *Access KServe endpoints*

Step 5 — Monitor and iterate

Keep an eye on model health and performance:

- *Monitor InferenceService*
- (Optional) Tune Knowledge Base performance: *Knowledge Base performance*

Next steps

- Learn progressively: *Learning paths*
- Explore patterns: *Common workloads and use cases* and *Common Resolutions*
- Deep dive into the APIs and reference material via the Gen AI and Model Serving hubs

6.7.2 Manual

Enabling AI Factory

Before you can use **GenAI Builder** in Hybrid Manager, you need to enable **AI Factory**. This involves two main steps:

1. Setting up a **Fernet key** for encryption (if not already created).
2. Connecting GenAI Builder to an **S3-compatible bucket** for its datalake.

1. Generate a Fernet key (if not already created)

AI Factory uses a Fernet key to encrypt sensitive data. If you ran one of the `-install-secrets.sh` scripts during Hybrid Manager setup, you may already have this key.

If not, generate it manually with the following commands (requires `kubectl` configured for your HM cluster):

```
FERNET_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64)

kubectl create namespace upm-griptape

kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: fernet-secret
  namespace: upm-griptape
stringData:
  FERNET_KEY: ${FERNET_KEY}
EOF
```

This creates a Kubernetes secret in the `upm-griptape` namespace for GenAI Builder to use.

2. Create an S3-compatible bucket

GenAI Builder requires an **S3 bucket** (the datalake) to store structures, tools, and indexed data.

Gather the following information from your bucket provider:

Field	Example
Bucket name	*mycompany-genai-datalake*
Access key	AWS access key ID or equivalent
Secret access key	AWS secret access key or equivalent
URL	*https://s3.us-west-2.amazonaws.com/*
Region	*us-west-2*

> **Note**
>
> The bucket name must be globally unique if using AWS.

Optional: CORS configuration

If you plan to use the **GenAI Builder web UI**, configure CORS rules on the bucket so the UI can access it. For AWS S3, a typical configuration is:

```
[
  {
    "AllowedHeaders": ["*"],
    "AllowedMethods": ["PUT", "POST", "DELETE", "GET", "HEAD"],
    "AllowedOrigins": ["https://${PORTAL_DOMAIN_NAME}"],
    "ExposeHeaders": []
  }
]
```

Replace `${PORTAL_DOMAIN_NAME}` with your HM portal domain.

Google Cloud Storage (GCS)

GCS can be used via its **S3 interoperability layer**:

1. Enable S3 interoperability in GCS settings.
2. Create or update a service account with roles:
   - *Storage Admin*
   - *Service Account Token Creator*
3. Generate an HMAC key pair.
4. Configure bucket CORS (save as `cors.json`):

```
[
  {
    "origin": ["https://<portal-url>"],
    "method": ["GET", "PUT", "POST", "DELETE", "HEAD"],
    "responseHeader": ["*"],
```

(continues on next page)

(continued from previous page)

```
    "maxAgeSeconds": 3600
  }
]
```

Apply with:

```
gsutil cors set cors.json gs://<bucket-name>
```

3. Connect the bucket to GenAI Builder

Once your bucket is ready:

1. In Hybrid Manager, select **Projects** from the dropdown menu on the upper-right corner.
2. Navigate to your project and select **Sovereign AI** from the left navigation bar.
3. Select **GenAI Builder**.
4. Log in with your administrator account:

Username: This is the same username you used to configure HM. To obtain the username:

```
kubectl exec -it resourcedb-1 -n upm-griptape -- psql resources -c "select * from␣
↪griptape_cloud_user"
```

Password: Your first-time password is `pgai_admin` by default.

3. In the **Administrator Console**, open the AI Factory configuration form.
4. Enter your bucket details:

Field	Example
S3 URL	*https://s3.us-west-2.amazonaws.com/*
S3 Bucket	*mycompany-genai-datalake*
S3 Access Key	*AKIA…*
S3 Secret Key	*wJalr…*
S3 Region	*us-west-2*

5. Click **Submit**.

4. Success: AI Factory enabled

Once saved:

- GenAI Builder can store and retrieve content in your datalake.
- You can now create **Libraries, Knowledge Bases, and Assistants**.
- Configuration details (like the Fernet key and bucket settings) are handled for you inside Hybrid Manager.

Gen AI Builder on Hybrid Manager

Overview

Gen AI Builder within Hybrid Manager enables organizations to create intelligent applications that combine Large Language Models with organizational data, all running within your controlled Kubernetes infrastructure. This platform

provides visual and programmatic interfaces for building retrieval-augmented generation (RAG) systems, conversational assistants, and automated workflows without external dependencies. See the *Gen AI Builder on Hybrid Manager* .

The integration ensures complete data sovereignty—your knowledge bases, model interactions, and conversation histories remain within your infrastructure while leveraging state-of-the-art AI capabilities through locally deployed models.

Core Architecture

LLM Integration

Gen AI Builder connects to models deployed through Hybrid Manager's *Model Serving* infrastructure, providing seamless access to language models for text generation, embeddings, and reranking operations.

Model Access Patterns

- Internal endpoints via cluster DNS for low-latency inference
- External endpoints through secured gateways for application integration
- OpenAI-compatible APIs enabling standard client libraries (see *OpenAI API compatibility*)

Supported Operations

- Chat completions for conversational interactions
- Embeddings for semantic search and knowledge base creation
- Reranking for result optimization in retrieval workflows

Knowledge Base Infrastructure

Knowledge bases transform organizational data into searchable, AI-ready formats through embedding and indexing within PostgreSQL databases enhanced with *pgvector and Vector Engine concepts* . See *Knowledge bases in the asset library* and *Knowledge Base Pipelines* .

Data Processing Pipeline

1. **Ingestion**: Connect to *data sources* including databases, documents, and APIs
2. **Chunking**: Split content into semantically meaningful segments
3. **Embedding**: Generate vector representations using deployed models
4. **Indexing**: Store vectors in PostgreSQL with metadata for filtering

Storage Architecture

- Vector embeddings stored in PostgreSQL with pgvector extension
- Document metadata maintained for filtering and attribution
- Original content preserved for reference and regeneration
- Incremental updates supported for dynamic knowledge bases

Building Intelligent Applications

Assistant Development

Assistants orchestrate the interaction between users, knowledge bases, and external systems through configurable workflows that combine retrieval, reasoning, and action. See *What are assistants and how do they work?* .

Configuration Components

- **Directive**: Define the assistant's purpose and behavior
- **Knowledge Integration**: Connect relevant *How do I create knowledge bases for RAG?* for context
- **Tool Binding**: Enable external system interactions via *Connecting data into external tools*
- **Model Selection**: Choose appropriate LLM for generation tasks

Example: Customer Service Assistant

```
assistant:
  name: customer-support
  directive: "Help customers with product inquiries and order status"
  knowledge_bases:
    - product-documentation
    - order-database
  tools:
    - order-tracking-api
    - ticket-creation
  model: llama-3-8b-instruct
```

Retrieval-Augmented Generation

RAG patterns ground LLM responses in organizational knowledge, reducing hallucination while maintaining conversational capabilities. See *Retrievers* .

Retrieval Flow

1. User query triggers semantic search across knowledge bases
2. Relevant documents retrieved based on vector similarity
3. Context provided to LLM along with user query
4. Response generated using retrieved information

Optimization Strategies

- Hybrid search combining vector and keyword matching
- Metadata filtering for domain-specific retrieval
- Reranking to improve result relevance (see *Using NVIDIA NIM*)
- Context window management for large documents

Tool Integration

Tools extend assistant capabilities by enabling interactions with external systems and APIs while maintaining security boundaries. See *Connecting data into external tools* .

Tool Types

- **REST APIs**: Connect to external services with authentication
- **Database Queries**: Execute SQL against operational databases
- **Internal Services**: Call Kubernetes services within cluster
- **Custom Functions**: Deploy specialized logic as containerized services

Security Controls

- Credential management through Kubernetes secrets

- Rate limiting to prevent abuse
- Audit logging for compliance tracking
- Scope restrictions per assistant configuration

Common Use Cases

Enterprise Knowledge Management

Organizations consolidate disparate information sources into unified, searchable knowledge bases accessible through natural language interfaces.

Implementation Pattern

- Ingest documentation from multiple repositories (Confluence, SharePoint, Google Drive)
- Create semantic indices for cross-source search
- Deploy assistants for different departments with filtered access
- Maintain audit trails for compliance requirements

Example: Technical Documentation Assistant A software company indexes all technical documentation, API references, and support tickets. Engineers query the system using natural language, receiving contextual answers with source citations.

Customer Service Automation

Intelligent assistants handle customer inquiries by combining product knowledge with real-time system access.

Capabilities

- Answer product questions using documentation knowledge base
- Check order status through API integration
- Create support tickets for complex issues
- Escalate to human agents when necessary

Example: E-commerce Support Bot An online retailer deploys an assistant that handles 70% of customer inquiries automatically, accessing order systems, product catalogs, and return policies through unified interfaces.

Compliance and Audit Support

Organizations leverage Gen AI to navigate complex regulatory requirements and maintain compliance documentation.

Features

- Query regulatory knowledge bases for specific requirements
- Generate compliance reports from operational data
- Track decision rationale through conversation threads
- Maintain immutable audit logs for review

Example: Financial Compliance Assistant A bank creates an assistant that helps compliance officers interpret regulations, generate required reports, and document decision-making processes with full traceability.

Operational Considerations

Performance Optimization

System performance depends on multiple factors requiring careful tuning for production deployments.

Key Metrics

- **Retrieval Latency**: Time to fetch relevant documents
- **Generation Latency**: LLM inference time for responses
- **End-to-End Latency**: Total time from query to response
- **Throughput**: Concurrent requests handled

Optimization Techniques

- Knowledge base partitioning for parallel search
- Response caching for frequently asked questions
- Model quantization for improved inference speed
- Connection pooling for database access

Resource Management

Gen AI applications require careful resource allocation across compute, memory, and storage dimensions.

Resource Requirements

- **Knowledge Base Storage**: PostgreSQL capacity for vectors and metadata
- **Model Serving**: GPU resources for LLM inference
- **Application Pods**: CPU and memory for orchestration logic
- **Network Bandwidth**: Data transfer between components

Scaling Strategies

- Horizontal scaling of retrieval services
- GPU sharing for development workloads
- Dedicated resources for production assistants
- Auto-scaling based on request patterns

Security Framework

Comprehensive security controls protect sensitive data throughout the application lifecycle.

Access Control

- Role-based permissions for assistant access
- Knowledge base filtering based on user context
- Tool authorization per assistant configuration
- API key management for external access

Data Protection

- Encryption at rest for knowledge base content

- TLS for all inter-service communication
- Sanitization of sensitive information in logs
- Secure credential storage in Kubernetes secrets

Monitoring and Observability

Conversation Tracking

Thread management provides comprehensive visibility into assistant interactions for debugging and compliance. See *Gen AI Builder (Assistants & threads)* .

Thread Components

- User queries and assistant responses
- Retrieved documents with relevance scores
- Tool invocations with parameters and results
- Token usage and latency metrics

Analysis Capabilities

- Response quality evaluation
- Knowledge gap identification
- Performance bottleneck detection
- User satisfaction tracking

System Monitoring

Operational metrics ensure reliable service delivery and capacity planning.

Monitoring Areas

- Knowledge base query patterns and performance
- Model endpoint utilization and latency
- Tool API success rates and response times
- Resource consumption trends

Alerting Scenarios

- High latency exceeding SLA thresholds
- Failed tool invocations affecting functionality
- Resource exhaustion risks
- Unusual usage patterns indicating issues

Implementation Guide

Getting Started

Organizations should begin with focused use cases that demonstrate value before expanding deployments.

Initial Setup

1. Deploy embedding model for knowledge base creation

2. Configure data sources for initial knowledge base
3. Deploy LLM for generation tasks
4. Create simple assistant with basic retrieval

Validation Steps

- Test retrieval quality with sample queries
- Verify response accuracy against source documents
- Measure performance under expected load
- Review security controls and access patterns

Best Practices

Knowledge Base Design

- Maintain clear document boundaries for attribution
- Implement regular update cycles for fresh content
- Use metadata effectively for filtering
- Monitor retrieval quality metrics

Assistant Configuration

- Start with restrictive directives and expand gradually
- Test edge cases and failure scenarios
- Implement fallback strategies for tool failures
- Version control assistant configurations

Production Deployment

- Establish baseline performance metrics
- Implement gradual rollout strategies
- Monitor user feedback and satisfaction
- Maintain configuration backups

Integration with Hybrid Manager

Project Organization

Gen AI resources deploy within Hybrid Manager projects, inheriting namespace isolation and resource controls.

Project Resources

- Knowledge bases as PostgreSQL databases
- Assistants as Kubernetes deployments
- Tools as service integrations
- Threads as persistent storage

UI Workflows

Hybrid Manager provides visual interfaces for Gen AI development without requiring deep technical expertise.

Visual Capabilities

- Drag-and-drop assistant configuration
- Knowledge base creation wizards
- Tool registration interfaces
- Conversation testing environments

Programmatic Access

SDKs and APIs enable integration with existing applications and automation workflows.

Integration Options

- Python SDK for application development
- REST APIs for service integration
- Kubernetes operators for GitOps workflows
- CLI tools for automation scripts

Next Steps

Begin exploring Gen AI Builder capabilities:

1. **Review Concepts**: Understand *Concepts and architecture* and *AI Factory Architecture on Hybrid Manager*
2. **Create Knowledge Base**: Follow the *knowledge base guide*
3. **Build Assistant**: Use the *assistant creation guide*
4. **Deploy Application**: Implement using the *quickstart UI*

For detailed documentation:

- *What are assistants and how do they work?*
- *Knowledge Bases Manual*
- *Tools Documentation*
- *SDK Reference*

Model Management on Hybrid Manager

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.

Core Concepts

Model Library

The *Model Library Architecture* 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 AI Factory capabilities including *Gen AI Builder on Hybrid Manager* assistants, *How do I create knowledge bases for RAG?* pipelines, 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 *Setup GPU* and *Update GPU resources*)
- 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

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

GPU Infrastructure

Resource Types

Hybrid Manager supports various GPU configurations to match workload requirements:

Development GPUs: Lower-tier GPUs (T4, RTX series) for testing and experimentation with time-slicing support for resource sharing **Production GPUs**: High-performance GPUs (A100, H100) with dedicated allocation for latency-sensitive inference workloads **Multi-Instance GPUs**: Partitioned GPUs enabling multiple smaller models to share hardware resources efficiently

Allocation Strategies

GPU scheduling occurs through Kubernetes device plugins that advertise available resources and enforce allocation policies. The system supports:

- Exclusive GPU allocation for production workloads
- Time-sliced sharing for development and testing
- MIG partitions for efficient multi-model serving
- Node affinity rules for workload placement

Resource quotas at the project level prevent GPU monopolization while ensuring fair access across teams. Automatic scaling adjusts GPU allocation based on inference demand, optimizing costs while meeting service requirements.

Integration Patterns

Gen AI Builder Applications

Gen AI Builder leverages model endpoints for assistants and agents. LLMs deployed through Model Serving provide text generation capabilities while embedding models support RAG pipelines.

Applications access models through cluster-local DNS:

```
http://model-name.namespace.svc.cluster.local/v1/chat/completions
```

Knowledge Base Pipelines

Knowledge bases utilize embedding models for document vectorization and semantic search. The Model Library ensures consistent model versions across ingestion and retrieval operations.

Pipeline integration benefits from:

- Validated embedding models with known performance characteristics
- Automatic model updates through library synchronization
- Consistent vector dimensions across knowledge base operations

Custom Applications

Organizations deploy specialized models for business-specific requirements. The serving infrastructure supports various model types including vision models, reranking models, and custom fine-tuned variants.

External applications access models through secured endpoints with API key authentication, enabling integration with existing business systems while maintaining security boundaries.

Security and Governance

Access Control

Role-based access control governs model operations at multiple levels:

- **Library Access**: Controls who can register and approve models
- **Deployment Permissions**: Manages who can deploy models to serving
- **Endpoint Access**: Determines who can invoke model endpoints

Integration with enterprise identity providers enables single sign-on while maintaining detailed audit trails for compliance requirements.

Model Governance

The governance framework ensures only validated models reach production:

- Security scanning identifies vulnerabilities before deployment
- Digital signatures verify model authenticity and integrity
- Approval workflows enforce multi-stakeholder validation
- Retention policies manage model lifecycle and deprecation

Data Protection

All model operations maintain data sovereignty within your infrastructure:

- Models run exclusively on your Kubernetes clusters
- Inference data never leaves your controlled environment
- Network policies enforce traffic isolation
- Encryption protects data in transit and at rest

Monitoring and Optimization

Performance Metrics

Comprehensive monitoring tracks model performance across multiple dimensions:

- **Inference Latency**: End-to-end request processing time
- **Token Throughput**: Generation rate for language models
- **Batch Efficiency**: Utilization of batching capabilities
- **Cache Hit Rates**: Effectiveness of response caching

These metrics inform optimization decisions including batch size tuning, resource allocation adjustments, and caching strategy refinements.

Resource Utilization

GPU monitoring provides visibility into hardware efficiency:

- Memory usage patterns identifying optimization opportunities
- Compute utilization indicating scaling requirements
- Power consumption for cost analysis
- Temperature monitoring for hardware health

Cost Management

The system tracks resource consumption for cost attribution:

- GPU hours per model and project
- Storage costs for model artifacts
- Network transfer for inference traffic
- Scaling patterns affecting resource costs

Best Practices

Model Selection

Choose models based on clear requirements analysis:

- Match model capabilities to use case requirements
- Consider resource constraints and cost implications
- Validate performance characteristics through testing

- Document model selection rationale for future reference

Deployment Strategy

Implement systematic deployment approaches:

- Start with conservative resource allocation
- Monitor actual usage before optimization
- Use canary deployments for model updates
- Maintain rollback capabilities for problematic deployments

Operational Excellence

Maintain high operational standards:

- Establish baseline performance metrics
- Implement comprehensive monitoring before production
- Document configuration decisions and procedures
- Regular review of resource utilization and costs

Getting Started

Begin your model management journey with these steps:

1. **Enable Model Capabilities** in your Hybrid Manager project through Settings → Features
2. **Configure Repository Rules** to connect trusted model sources
3. **Deploy Your First Model** using the guided deployment workflow
4. **Monitor Performance** through integrated dashboards

For detailed implementation guidance:

- *Model Library Configuration*
- *Create InferenceService*
- *GPU Setup Guide*
- *Monitoring InferenceServices*

Model management within Hybrid Manager transforms AI deployment from experimental prototypes to production-grade services, maintaining sovereignty while delivering enterprise capabilities.

6.7.3 How To Builders

Using models in Gen AI Builder

Hub quick links: *Gen AI how-tos* — *Model Serving* — *Model Library Architecture*

How models are consumed

In Hybrid Manager, **Gen AI Builder** lets you build assistants that use model endpoints served from the *Model Library Architecture* .

- When creating or editing an **Assistant**, you select which model to use (see *Step 2 — Create an Assistant*).

- Models available here are **governed by Hybrid Manager**: pulled from the Model Library, deployed to your project, and exposed via internal endpoints.
- This ensures that model calls stay within your environment — **no external API calls** are made by default.

Knowledge Bases and pipelines

Knowledge Bases in Gen AI Builder can be populated with embeddings generated by **Pipelines**:

- Pipelines ingest and prepare documents into **vector indexes** (see *Vector Engine concepts*).
- Knowledge Bases reference these pipelines, making them queryable for Retrieval-Augmented Generation (RAG).
- Assistants then combine model responses with Knowledge Base retrievals to ground outputs in your organization's data.

See: *Knowledge bases in the asset library* and *Pipelines (hub)* .

Environment and service discovery

When Gen AI Builder runs inside Hybrid Manager:

- **Service discovery** and **endpoints** are automatically managed by the platform.
- You do not need to configure external URLs; models appear directly in the Assistant creation UI.
- Any required **environment variables** (for service routing or authentication) are injected by HM.

This means you focus on building assistants — **Hybrid Manager takes care of wiring models, data, and observability together.**

Common tasks

Use these how-tos from the hub to start building:

- *Step 2 — Create an Assistant*
- *Step 1 — Create a Knowledge Base* and *Manage Knowledge Bases*
- *View threads*

Key takeaway

In Hybrid Manager, models and data are **co-located**:

- **Models**: deployed from the Model Library into your HM project.
- **Data**: ingested through pipelines, stored in Knowledge Bases.
- **Assistants**: combine both, with no traffic leaving your cluster.

6.7.4 Setup

Deploy AI Factory with Hybrid Manager UI

Overview

This guide provides deployment procedures for AI 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.

Prerequisites

Infrastructure Requirements

Before deploying AI 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 NVIDIA NIM requirements:

Model Type	NIM Model	GPU Require-ments	Container Image
Text Completion	llama-3.3-nemotron-su	4 x L40S	nvcr.io/nim/nvidia/llama-3.3-nemotron-super-49b-v1:1.8
Text Embeddings	llama-3.2-nemoretriev	1 x L40S	nvcr.io/nim/nvidia/llama-3.2-nemoretriever-300m-embe
Image Embeddings	nvclip	1 x L40S	nvcr.io/nim/nvidia/nvclip:latest
OCR	paddleocr	1 x L40S	nvcr.io/nim/baidu/paddleocr:latest
Text Reranking	llama-3.2-nv-rerankqa	1 x L40S	nvcr.io/nim/nvidia/llama-3.2-nv-rerankqa-1b-v2:latest

Ensure GPU nodes include:

- Label: `nvidia.com/gpu=true`
- Taint: `nvidia.com/gpu`

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 \
```

(continues on next page)

(continued from previous page)

```
    replicator.v1.mittwald.de/replicate-to=m-.*

#  Create image pull secret

kubectl -n default create secret docker-registry ngc-cred \
    --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.

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 **AI 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

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

Configure 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

Step 4: Monitor Deployment Status

View Model Server Clusters

Navigate to **AI 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

Step 5: Configure API Access

Generate API Tokens

For external access to deployed models:

1. Navigate to **AI 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
```

Step 6: Build Gen AI Applications

Create Knowledge Base

1. Navigate to **AI Factory → Gen AI Builder → Knowledge Bases**
2. Configure data sources:
   - PostgreSQL databases
   - Document repositories
   - API connections
3. Select embedding model from deployed services

4. Process content to generate embeddings

Configure Assistant

1. Navigate to **AI Factory → Assistants**
2. Create new assistant with:
   - Name and directive
   - Model endpoint selection
   - Knowledge base integration
   - Tool configuration
3. Deploy assistant application

See *Create Assistant Guide* for detailed procedures.

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
4. Apply changes (triggers rolling update)

Rolling Updates

Updates maintain availability through:

- Connection draining before restart
- Health verification between instances
- Automatic rollback on failure

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 AI Factory on Hybrid Manager* .

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

Prerequisites for AI Factory on Hybrid Manager

Infrastructure Requirements

Kubernetes Cluster Foundation

AI 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

Model Resource Requirements

Model Type	Example Model	GPU Require-ments	Container Image
Text Completion	llama-3.3-nemotron-su	4 x L40S	nvcr.io/nim/nvidia/llama-3.3-nemotron-super-49b-v1:1.8
Text Embeddings	llama-3.2-nemoretriev	1 x L40S	nvcr.io/nim/nvidia/llama-3.2-nemoretriever-300m-embe
Image Embeddings	nvclip	1 x L40S	nvcr.io/nim/nvidia/nvclip:latest
OCR	paddleocr	1 x L40S	nvcr.io/nim/baidu/paddleocr:latest
Text Reranking	llama-3.2-nv-rerankqa	1 x L40S	nvcr.io/nim/nvidia/llama-3.2-nv-rerankqa-1b-v2:latest

Reference the NVIDIA NIM documentation for detailed resource specifications per model.

Registry Configuration

Internet-Connected Deployments

For clusters with internet access, configure NVIDIA NGC registry authentication to enable model image pulls.

NGC API Key Configuration

Obtain an NGC API key following the NVIDIA NGC documentation .

Create the required secrets:

```
#  Set your NGC API key

NGC_API_KEY=<your-ngc-api-key>

#  Create model runtime secret

kubectl -n default create secret generic nvidia-nim-secrets \
    --from-literal=NGC_API_KEY=${NGC_API_KEY}

#  Enable secret replication across namespaces

kubectl -n default annotate secret nvidia-nim-secrets \
    replicator.v1.mittwald.de/replicate-to=m-.*
```

Image Pull Secret Configuration

Configure Docker registry authentication for image pulls:

```
#  Create image pull secret

kubectl -n default create secret docker-registry ngc-cred \
    --docker-server=nvcr.io \
    --docker-username=$oauthtoken \
    --docker-password=${NGC_API_KEY}
```

(continues on next page)

(continued from previous page)

```
#  Enable secret replication

kubectl -n default annotate secret ngc-cred \
    replicator.v1.mittwald.de/replicate-to=m-.*
```

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: *Model Library Architecture*
- 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*

Network Requirements

Service Communication

AI Factory components require specific network configurations for inter-service communication.

Network Policies

- Allow traffic between model serving pods and application namespaces
- Enable ingress for external model endpoint access
- Support service mesh communication for observability

DNS Resolution

- Cluster-local DNS for internal service discovery
- External DNS for public endpoint access if required

Storage Access

Ensure network connectivity to required storage systems.

Object Storage

- S3-compatible storage for model artifacts and profiles
- Sufficient bandwidth for model download operations
- IAM roles or credentials for storage access

Security Prerequisites

RBAC Configuration

Configure role-based access control for AI Factory operations.

Required Roles

- Namespace administrator for project-level management
- Model operator for deployment operations
- Read-only access for monitoring and observability

Certificate Management

Prepare SSL certificates for secure communication.

Certificate Requirements

- TLS certificates for model endpoint exposure
- Internal CA for service mesh communication
- Certificate rotation procedures for compliance

Next Steps

With prerequisites satisfied, proceed to:

1. *GPU Setup Guide* for detailed GPU configuration
2. *Model Library Configuration* for registry integration
3. *Create First InferenceService* to deploy initial model

For planning assistance, consult:

- *AI Factory Architecture on Hybrid Manager*
- *Model Serving Concepts*

6.7.5 How To Models

Using models in Model Clusters & AIDB

Hub quick links: *Model Serving* — *Access KServe endpoints*

From applications

Applications can call deployed model endpoints directly via **KServe InferenceServices**.

- **Quickstart (Python):** *Quickstart using inference endpoint*

These endpoints provide a standard REST API for sending requests and retrieving predictions or generations from your deployed models.

From AIDB (SQL patterns)

AIDB lets you call models from SQL, making them available directly inside Postgres. This can be useful for embedding models in pipelines or enabling in-database inference.

- **OpenAI API compatibility:** *Using models via OpenAI-compatible SQL calls*
- **NVIDIA NIM:** *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. If you connect from outside, configure authentication using the HM ingress and project-scoped credentials.
- **Observability.** Requests and logs from these endpoints flow into HM observability, giving you usage metrics, latency, and error tracking.

Key takeaway

- **Applications** → call KServe inference endpoints over REST.
- **AIDB** → use SQL patterns to call models inline, including OpenAI-style calls and NVIDIA NIM.
- **Hybrid Manager** handles service discovery, auth, and observability so you can focus on building.

6.7.6 Operate & Observe

Observability (AI Factory on HM)

Hub quick links: *Model Serving observability* — *Gen AI Builder on Hybrid Manager*

Observability in Hybrid Manager

When you deploy AI Factory components on Hybrid Manager, the platform gives you integrated observability for both **Model Serving** (KServe) and **Gen AI Builder** 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*

Gen AI Builder (Assistants & threads)

- Each **assistant thread** is recorded in the system so you can trace inputs, retrieval context, and outputs.
- Thread data can be viewed in the Gen AI Builder UI inside Hybrid Manager.
- Usage metrics (number of runs, latency, errors) are integrated into the platform's observability stack.
- See: *View threads (hub)*

Accessing metrics and logs

- **Dashboards.** Hybrid Manager surfaces AI Factory metrics through the built-in observability dashboards (Grafana).
- **Logs.** Use `kubectl logs` or the HM UI log viewer to inspect Gen AI Builder pods, KServe InferenceServices, and retriever jobs.
- **Tracing.** OpenTelemetry drivers are included in Gen AI Builder for deeper tracing; see *Gen AI observability drivers (hub)* .

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

kubectl port-forward -n observability svc/grafana 3000:3000
```

Key takeaway

- **Model Serving** → monitored as KServe services with full metrics.
- **Gen AI Builder** → assistants and threads tracked for usage and auditing.
- **Hybrid Manager** provides a unified observability layer — no separate monitoring stack to configure.

Troubleshooting AI Factory on Hybrid Manager

Diagnostic Overview

This guide provides systematic troubleshooting procedures for AI Factory components within Hybrid Manager environments. Issues typically fall into three categories: infrastructure problems, model serving failures, and application-level errors.

Quick Reference Links

- *Frequently Asked Questions - AI Factory on Hybrid Manager*
- *GPU Setup Guide*
- *InferenceService Creation*
- *Gen AI Builder on Hybrid Manager*

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
```

(continues on next page)

(continued from previous page)

```
#  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.

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>

#  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.

Gen AI Application Issues

Knowledge Base Retrieval Failures

Symptoms

- Empty or irrelevant search results
- Embedding generation errors
- Vector database connection failures

Diagnostic Process

```
#  Check embedding model status

kubectl get inferenceservice <embedding-model> -n <namespace>

#  Verify PostgreSQL connectivity

kubectl exec -it <app-pod> -n <namespace> -- pg_isready -h <postgres-host>

#  Review application logs

kubectl logs <gen-ai-app-pod> -n <namespace>
```

Resolution Steps

Embedding Model Issues Verify embedding model deployment and endpoint accessibility. Ensure consistent embedding dimensions between indexing and retrieval.

Database Connectivity Check PostgreSQL service availability and pgvector extension installation. Verify connection credentials in application configuration.

Assistant Response Errors

Symptoms

- Assistant fails to generate responses
- Tool invocation failures
- Thread state inconsistencies

Investigation Methods

Access thread logs for detailed execution traces:

```
#  Query thread history (application-specific)

kubectl exec <app-pod> -n <namespace> -- cat /app/logs/threads.log
```

Common Resolutions

Model Endpoint Unavailable Verify LLM InferenceService status and endpoint configuration in assistant settings.

Tool Integration Failures Check external API connectivity and authentication. Verify tool definitions match API specifications.

Context Window Exceeded Reduce retrieval chunk size or implement context management strategies for large documents.

Log Analysis

Log Locations

AI 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
- Gen AI applications: 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
```

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.

Escalation Procedures

Support Resources

When internal troubleshooting proves insufficient:

1. **Documentation Review**
   - Consult *AI 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 AI 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
```

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

Welcome to the central resource for understanding and leveraging the analytics capabilities embedded within EDB Hybrid Manager (HM).

Hub quick links: *Analytics Hub* — *Core Concepts* — *Related How-Tos*

Hybrid Manager provides integrated support for:

- Apache Iceberg® and catalog services
- Delta Lake querying
- Tiered Tables with EDB Postgres Distributed (PGD)
- Lakehouse Clusters for high-speed analytics on object storage
- Unified management and observability of your analytics stack

By using these capabilities, you can build modern, cost-effective analytics architectures on Postgres with reduced operational overhead—fully managed through Hybrid Manager.

7.1 Why Analytics in Hybrid Manager

EDB Hybrid Manager integrates analytics to help you:

- **Gain unified visibility:** View key metrics and operational trends across your entire EDB Postgres fleet managed by HM.
- **Optimize performance:** Identify performance bottlenecks and understand query behavior within HM-managed databases and Lakehouse Clusters.
- **Manage resources efficiently:** Monitor and tune resource consumption for HM-managed instances, including analytical workloads.
- **Enhance operational health:** Track the availability and status of analytical components and underlying database infrastructure.
- **Simplify data architectures:** Provision and manage EDB Postgres Lakehouse Clusters, implement Tiered Tables with PGD, and integrate with object storage—all from HM.

7.2 Core Concepts and Capabilities

7.2.1 Apache Iceberg®

In Hybrid Manager, Iceberg is used to:

- Manage large analytical tables in Iceberg format, queried by Lakehouse Clusters.
- Support PGD offloading of cold partitions to object storage in Iceberg format.
- Enable interoperability with external tools (Spark, Trino, etc.) via catalog-managed tables.
- Facilitate catalog management via HM-managed services (Lakekeeper) or external catalogs.

Apache Iceberg® in Hybrid Manager

Next step: *Configure an Iceberg catalog connection*

7.2.2 Delta Lake

Hybrid Manager supports querying Delta Lake tables:

- Lakehouse Clusters can query existing Delta Lake tables stored in object storage.
- Integrate HM-managed Lakehouse with Delta-based pipelines (Spark, Databricks, etc.).
- Enable fast analytics on existing Delta data lakes via Lakehouse Clusters.

Delta Lake in Hybrid Manager

Next step: *Read Iceberg/Delta with or without a catalog*

7.2.3 Tiered Tables with PGD

Tiered Tables enable efficient management of large time-series datasets:

- PGD's AutoPartition automates time-based partitioning.
- Cold partitions are offloaded to Iceberg format in object storage.
- Lakehouse Clusters provide query access across hot and cold data.
- Hybrid Manager helps configure and monitor Tiered Tables end-to-end.

Tiered Tables in Hybrid Manager

Next step: *Tiered Tables in Hybrid Manager*

7.2.4 Lakehouse Clusters

Lakehouse Clusters are managed compute nodes in HM:

- Provide scalable, vectorized query execution on object storage.
- Support both Iceberg and Delta Lake formats.
- Integrate with catalogs (HM-managed or external).
- Unified with the broader HM ecosystem (PGD offloading, Tiered Tables, BI integrations).

EDB Postgres Lakehouse Clusters on Hybrid Manager

Next step: *Create a Lakehouse cluster*

7.3 How These Capabilities Work Together

- **Data offloading:** PGD clusters offload cold data to object storage in Iceberg format.
- **Catalog services:** HM integrates with catalogs to manage table metadata; Lakehouse and PGD query through these catalogs.
- **Lakehouse Clusters:** Managed compute nodes run scalable analytical queries on offloaded data.
- **Delta Lake integration:** Lakehouse Clusters also query existing Delta Lake tables in place.
- **Unified management:** Hybrid Manager provides a single control plane to provision, configure, monitor, and optimize your analytics architecture.

7.4 Getting Started: Quickstart Checklist

Follow this checklist to begin applying Hybrid Manager analytics features:

- *Analytics concepts (hub)* — Understand core patterns and capabilities.
- *Create a Lakehouse cluster* — Deploy Lakehouse Clusters to enable scalable querying on object storage.
- *Configure an Iceberg catalog connection* — Connect HM-managed or external catalogs to manage Iceberg tables.
- *Read with or without a catalog* — Query Iceberg and Delta Lake tables from your Lakehouse Clusters.
- *Read/write without a catalog* — Direct file-based access for development and testing.
- *Explore Persona-Based Guidance* — Tailored recommendations for DBAs, DevOps/SREs, Data Scientists, and Developers.
- *Explore architectures and examples* — Concepts and patterns to solve common analytics problems.
- *Analytics How-To Index* — Step-by-step guides for configurations and operations.

7.5 Explore Further

- *Analytics concepts (hub)*
- *Apache Iceberg® in Hybrid Manager*
- *Delta Lake in Hybrid Manager*
- *Tiered Tables in Hybrid Manager*
- *EDB Postgres Lakehouse Clusters on Hybrid Manager*
- *Persona-Based Guide*
- *Architectures and examples*
- *Analytics How-To Index*

By leveraging these capabilities, Hybrid Manager enables you to build a modern, flexible, and performant analytics architecture across your EDB Postgres deployments—all through a unified and operationally efficient experience.

7.5.1 Analytics Concepts in Hybrid Manager

Hybrid Manager enables you to build modern data lakehouse and analytics solutions for EDB Postgres.

Hub quick links: *Analytics Hub* — *Core Concepts* — *Related How-Tos*

This page introduces key concepts as they are applied in Hybrid Manager — with links to detailed explanations in the *Analytics Hub* for background.

Architecture overview

Hybrid Manager integrates the following layers:

- **Lakehouse Clusters** for scalable analytical compute
- **Object Storage** for cost-efficient analytical data storage
- **PGD Clusters** for transactional workloads and automated Tiered Tables offload
- **Catalog Services** (HM-managed or external) for metadata and interoperability
- **PGAA and PGFS** to bridge Postgres with the data lake

This architecture enables both Postgres-first and multi-engine data lakehouse designs.

For a general introduction, see *EDB Postgres Lakehouse Clusters on Hybrid Manager* .

Lakehouse Clusters in Hybrid Manager

Lakehouse Clusters provide scalable analytical compute in Hybrid Manager:

- Provisioned and managed through HM
- Equipped with PGAA extensions for vectorized query execution
- Designed to query data in object storage (Iceberg, Delta Lake)
- Interoperable with external tools (Spark, Trino)

Lakehouse Clusters are central to analytics in Hybrid Manager.

Learn more: *EDB Postgres Lakehouse Clusters on Hybrid Manager*

Apache Iceberg® in Hybrid Manager

Hybrid Manager integrates Iceberg as the primary table format for offloaded and analytical data:

- PGD clusters offload data to Iceberg format for Tiered Tables
- Lakehouse Clusters query Iceberg tables efficiently
- Catalog services (Lakekeeper or external) manage Iceberg table metadata

Iceberg provides an open and reliable foundation for lakehouse architectures in HM.

Conceptual overview: *Apache Iceberg® in Hybrid Manager*

Delta Lake in Hybrid Manager

Delta Lake provides additional flexibility:

- Lakehouse Clusters can query existing Delta Lake tables in object storage
- Common in environments with Spark/Databricks pipelines
- Delta Lake support in HM is currently read-only

Hybrid Manager enables Postgres-based analytics on Delta Lake data without ETL.

Conceptual overview: *Delta Lake in Hybrid Manager*

Tiered Tables in Hybrid Manager

Tiered Tables provide automated lifecycle management of large time-series or historical datasets:

- PGD clusters use BDR AutoPartition to manage partitions
- Older partitions are offloaded automatically to object storage in Iceberg format
- Lakehouse Clusters and PGD parent tables can query both hot and cold data

Tiered Tables are a key optimization pattern for combining transactional and analytical workloads in Hybrid Manager.

Conceptual overview: *Tiered Tables in Hybrid Manager*

PGAA and PGFS in Hybrid Manager

PGAA (Analytics Accelerator)

PGAA extensions provide:

- Vectorized query execution
- Integration with object storage via PGFS
- Support for querying Iceberg and Delta Lake tables
- Catalog integration

PGAA powers both Lakehouse Clusters and PGD offload pipelines.

PGFS (Postgres File System)

PGFS defines object storage locations:

- Used by PGAA to read/write data in object storage
- Supports S3-compatible storage, GCS, and others
- Configured in both PGD clusters and Lakehouse Clusters

PGFS is a simple but critical building block for data lake integration.

How these concepts fit together in Hybrid Manager

Layer	Role
PGD Clusters	Transactional layer, source for Tiered Tables
Lakehouse Clusters	Analytical compute, queries across Iceberg/Delta and Postgres
Object Storage	Cost-efficient analytical data storage
Catalog Services	Centralized metadata for Iceberg tables
PGAA + PGFS	Bridge between Postgres and object storage

Next topic

Solving Analytics Problems in Hybrid Manager

7.5.2 Delta Lake in Hybrid Manager

Delta Lake is an open-source table format that brings ACID transactions and reliability to data lakes.

Hub quick links: *Analytics Hub* — *Core Concepts* — *Related How-Tos*

Hybrid Manager (HM) integrates Delta Lake capabilities into EDB Postgres deployments, enabling Lakehouse clusters to query Delta Lake tables stored in object storage.

For a general overview of Delta Lake, see *Delta Lake in Hybrid Manager* .

Why use Delta Lake with Hybrid Manager

- **Query existing Delta Lakes:** Leverage data lakes already built on Delta Lake format with fast Postgres SQL.
- **Simplify analytics pipelines:** Avoid unnecessary ETL—query Delta Lake tables in place from Postgres.
- **Broader ecosystem integration:** Connect HM-managed Postgres to Delta Lake data produced by Spark, Trino, Flink, and other tools.
- **Cost-effective lakehouse architecture:** Store large datasets in object storage and query via Lakehouse clusters.

Key terms and architecture overview

For definitions of analytics terms used in Hybrid Manager—such as PGFS, PGAA, Lakehouse, and Analytics Offload—see *Analytics concepts (hub)* .

When should I use Delta Lake in Hybrid Manager?

Use Delta Lake with Hybrid Manager when you want to:

- **Query existing data lakes** built on Delta Lake format, without ETL or data duplication.
- **Integrate Postgres and data lake ecosystems**—query Delta Lake tables from Postgres SQL clients.
- **Enable unified analytics** across operational Postgres data and data lake data.
- **Support BI tools and ad-hoc queries** on Delta Lake content using familiar Postgres tools.
- **Leverage Lakehouse Clusters** for scalable, fast SQL on large Delta datasets.

Important: PGAA currently supports **read-only queries** on Delta Lake tables. Writing or updating Delta tables via PGAA is not supported.

Key capabilities of Delta Lake in Hybrid Manager

Querying existing Delta Lake tables

What: Run SQL queries on Delta Lake tables stored in object storage.

Why: Enable BI tools and Postgres users to query existing Delta Lake data without duplicating or moving it.

How: Define PGFS storage locations and PGAA external tables in Lakehouse clusters.

Where: S3-compatible object storage with Delta Lake format (`_delta_log` + Parquet files).

How-To: *Read Iceberg/Delta with or without a catalog*

Simplifying Postgres + Delta Lake integration

What: Connect HM Lakehouse clusters to Delta Lake tables created by Spark or other tools.

Why: Build unified reporting and analytics across your operational and data lake systems.

How: Create PGFS storage locations and PGAA reader tables pointing to Delta Lake paths.

Where: Shared object storage locations used by Delta Lake pipelines.

How-To: *Configure PGFS*

Supporting unified SQL-based analytics

What: Enable Postgres SQL queries over both operational data and Delta Lake data.

Why: Empower application developers, data scientists, and BI users to query data lake content without complex tooling.

How: Use PGAA reader tables in Lakehouse clusters; optionally join with Postgres data.

Where: Delta Lake tables in object storage + Postgres tables in Lakehouse cluster or via FDW/dblink.

How-To: *Read Iceberg/Delta with or without a catalog*

Getting started with Delta Lake in Hybrid Manager

To begin using Delta Lake with Hybrid Manager:

1. *Provision a Lakehouse cluster* .
2. *Configure PGFS* pointing to your Delta Lake object storage.
3. Enable the `pgaa` extension on the Lakehouse Cluster.
4. Create PGAA reader tables for Delta Lake paths.
5. Query Delta Lake tables using standard Postgres clients.

Related How-Tos

- *Read Iceberg/Delta with or without a catalog*
- *Configure PGFS*

Next topic

Tiered Tables in Hybrid Manager

7.5.3 Apache Iceberg® in Hybrid Manager

Apache Iceberg® is an open table format for large analytical datasets stored in object storage.

Hub quick links: *Analytics Hub* — *Core Concepts* — *Related How-Tos*

Hybrid Manager (HM) integrates Iceberg capabilities into EDB Postgres deployments, enabling:

- Efficient querying of object storage via Lakehouse clusters
- Structured data offloading from PGD clusters to Iceberg format
- Centralized catalog management for Iceberg tables
- Interoperability with external analytics engines (Spark, Trino, Flink, etc.)

For a general overview of Iceberg, see *Apache Iceberg® in Hybrid Manager* .

Why use Iceberg with Hybrid Manager

Apache Iceberg® provides an open, reliable, and performant foundation for analytics on large datasets. Using it through Hybrid Manager allows you to:

- Query Iceberg tables with high performance from Lakehouse clusters using vectorized execution.
- Offload transactional PGD data into Iceberg format for cost-efficient tiered storage and long-term analytics.
- Manage Iceberg catalogs centrally through HM or connect to external catalogs.

- Share data seamlessly between Postgres and other tools (Spark, Trino, Flink) using the Iceberg format.

Key terms and architecture overview

For definitions of analytics terms used in Hybrid Manager—such as PGFS, PGAA, Lakehouse, Catalog, AutoPartition, and Analytics Offload—see *Analytics concepts (hub)* .

When should I use Iceberg with Hybrid Manager?

Use Iceberg with Hybrid Manager when you want to:

- **Archive PGD data cost-effectively** and still query it with Postgres or other tools.
- **Unify a data lakehouse architecture** with HM-managed or external Iceberg catalog.
- **Enable ad-hoc analytics** on large object storage datasets without ETL.
- **Implement Tiered Tables** to manage large time-series datasets and storage lifecycle in PGD.
- **Integrate Postgres and external data processing** tools using the Iceberg format.

Key capabilities of Iceberg in Hybrid Manager

Querying existing Iceberg tables

What: Run SQL queries on Iceberg tables already stored in object storage.

Why: Reuse data created by other tools (Spark, Trino, Flink, PGD offload) without ETL or duplication.

How: Use Lakehouse clusters to define PGAA external tables pointing to Iceberg data.

Where: Iceberg tables in S3-compatible object storage, either file-based or catalog-managed.

How-To: *Read Iceberg/Delta with or without a catalog*

Iceberg catalog integration

What: Connect Lakehouse and PGD nodes to Iceberg catalogs (HM-managed or external).

Why: Manage Iceberg table metadata centrally and query with consistent semantics.

How: Use PGAA functions (`pgaa.add_catalog` , `pgaa.attach_catalog`) to connect and import catalogs.

Where: HM-managed Lakekeeper catalog, or third-party catalogs (Glue, Nessie, Polaris).

How-To: *Configure an Iceberg catalog connection*

Note: When using **Tiered Tables** or **PGD offload**, it is strongly recommended to use a catalog-managed Iceberg table for interoperability and lifecycle management.

Offloading PGD data to Iceberg

What: Offload PGD transactional data into Iceberg format for analytical storage.

Why: Optimize storage costs and reduce PGD operational load while keeping data queryable.

How: Configure PGFS locations and/or Iceberg catalog, then enable analytics replication on PGD tables.

Where: Offloaded Iceberg tables in object storage, queryable by Lakehouse or other tools.

Learn concepts for *Tiered Tables in Hybrid Manager*

Use cases for Iceberg in Hybrid Manager

- Archiving PGD data in Iceberg format while maintaining query access
- Centralized data lakehouse architecture with HM-managed catalog
- Ad-hoc analytics on large data volumes through Lakehouse clusters
- Implementing tiered storage and lifecycle management for PGD datasets
- Exposing PGD offloaded data to Spark, Trino, Flink pipelines using Iceberg format

Related: *Analytics concepts (hub)*

Getting started with Iceberg in Hybrid Manager

To begin using Iceberg with Hybrid Manager:

1. *Provision a Lakehouse cluster* .
2. *Configure an Iceberg catalog connection* (HM-managed or external).
3. Learn *Tiered Tables in Hybrid Manager* if integrating with PGD.
4. *Read Iceberg/Delta with or without a catalog* .
5. Use standard Postgres clients to run analytical queries.

Related How-Tos

- *Configure an Iceberg catalog connection*
- *Read Iceberg/Delta with or without a catalog*

Next topic

Delta Lake in Hybrid Manager

7.5.4 EDB Postgres Lakehouse Clusters on Hybrid Manager

EDB Postgres Lakehouse Clusters provide high-speed analytical compute for querying open lakehouse table formats (Apache Iceberg®, Delta Lake) in object storage.

Hub quick links: *Analytics Hub — Core Concepts — Related How-Tos*

Hybrid Manager (HM) enables you to provision and manage Lakehouse Clusters through its console and API, integrating them with your existing PGD clusters and broader analytics ecosystem.

For foundational concepts about EDB Postgres Lakehouse architecture, see *EDB Postgres Lakehouse Clusters on Hybrid Manager* .

Why use Lakehouse Clusters in Hybrid Manager

- **Managed analytical compute:** Provision Lakehouse Clusters in HM to run fast queries on object storage.
- **Separation of compute and storage:** Keep data in object storage, scale analytical compute as needed.
- **Integration with PGD:** Use Lakehouse Clusters with PGD Tiered Tables and offloaded data.
- **Interoperability:** Lakehouse Clusters can query Iceberg and Delta Lake tables shared with other analytics engines.
- **Familiar Postgres interface:** Use standard SQL clients and tools with EDB Postgres Lakehouse Clusters.

Key terms and architecture overview

For definitions of core analytics terms used in Hybrid Manager—such as PGFS, PGAA, Lakehouse Cluster, and Analytics Offload—see *Analytics concepts (hub)* .

When should I use Lakehouse Clusters in Hybrid Manager?

Use Lakehouse Clusters in Hybrid Manager when you want to:

- **Run fast, scalable analytics** on large datasets stored in object storage.
- **Query data from PGD Tiered Tables** or offloaded PGD transactional data.
- **Integrate Postgres with external data lakes** built on Iceberg or Delta Lake.
- **Support BI tools and ad-hoc users** with Postgres SQL access to data lake content.
- **Manage analytical compute separately** from transactional databases—scale only when needed.

Key capabilities of Lakehouse Clusters in Hybrid Manager

Running fast queries on object storage

What: Run analytical queries on Apache Iceberg® and Delta Lake tables stored in object storage.

Why: Enable fast, scalable analytics on large datasets without moving them into Postgres storage.

How: Provision Lakehouse Clusters in HM and query data using PGAA with vectorized query execution.

Where: S3-compatible object storage buckets connected via PGFS.

How-To: *Read Iceberg/Delta with or without a catalog*

Supporting PGD Tiered Tables and offloading

What: Use Lakehouse Clusters as the query target for PGD offloaded and tiered data.

Why: Optimize PGD operational storage while keeping historical data queryable.

How: Configure PGD node groups for analytics replication to object storage in Iceberg format.

Where: Offloaded PGD data in Iceberg, queried through Lakehouse Clusters.

Learn concepts for *Tiered Tables in Hybrid Manager*

Querying external data lakes

What: Query Iceberg or Delta Lake tables created by other tools (Spark, Trino, Flink).

Why: Avoid data duplication and ETL by querying external data lakes from Postgres.

How: Connect Lakehouse Clusters to external Iceberg catalogs or directly to Delta Lake storage.

Where: S3-compatible object storage, with optional catalog integration.

How-To: *Configure an Iceberg catalog connection*

How-To: *Read Iceberg/Delta with or without a catalog*

Centralized management of analytical compute

What: Provision, monitor, and manage analytical clusters in HM alongside your PGD clusters.

Why: Manage analytical and transactional compute in one control plane.

How: Use the HM console or API to create and configure Lakehouse Clusters.

Where: Lakehouse Clusters run in cloud environments managed by HM.

How-To: *Create a Lakehouse cluster*

Getting started with Lakehouse Clusters in Hybrid Manager

To begin using Lakehouse Clusters with Hybrid Manager:

1. *Provision a Lakehouse cluster* .
2. *Configure PGFS storage locations* pointing to object storage.
3. (Optional) *Configure an Iceberg catalog connection* .
4. Learn *Tiered Tables in Hybrid Manager* if integrating with PGD.
5. Create Lakehouse tables or external readers for Iceberg/Delta data.
6. Query your data using Postgres clients.

Related How-Tos

- *Create a Lakehouse cluster*
- *Configure an Iceberg catalog connection*
- *Read Iceberg/Delta with or without a catalog*

Next topic

Apache Iceberg® in Hybrid Manager

7.5.5 Tiered Tables in Hybrid Manager

Tiered Tables in Hybrid Manager (HM) provide an automated solution for managing large time-series or historical datasets by moving older data from PGD clusters to cost-effective object storage in Apache Iceberg® format.

Hub quick links: *Analytics Hub* — *Core Concepts* — *Related How-Tos*

Hybrid Manager integrates this capability across:

- PGD clusters with BDR AutoPartition
- PGAA extensions for offloading
- Lakehouse Clusters for querying
- Centralized catalog services (Lakekeeper or external Iceberg catalogs)

For a conceptual overview of Tiered Tables, see *Tiered Tables in Hybrid Manager* .

Why use Tiered Tables with Hybrid Manager

- **Lower storage costs:** Offload “cold” data to object storage (Iceberg) and shrink primary PGD transactional tables.
- **Faster transactional performance:** Keep “hot” data partitions small for efficient PGD operations.

- **Automated lifecycle management:** Move data across tiers automatically based on age.
- **Transparent analytics:** Query both hot and cold data via PGD parent table or Lakehouse Cluster.
- **Unified management:** Configure and monitor all components through Hybrid Manager.

Key terms and architecture overview

For definitions of analytics terms used in Hybrid Manager—such as PGFS, PGAA, Analytics Offload, and Lakehouse—see *Analytics concepts (hub)* .

When should I use Tiered Tables in Hybrid Manager?

Use Tiered Tables in Hybrid Manager when you want to:

- **Manage large time-series datasets** in a cost-efficient way.
- **Keep PGD operational tables lean** for better performance.
- **Meet compliance needs** by keeping older data available but outside of PGD storage.
- **Enable BI users to run historical trend queries** without impacting production databases.
- **Automate your data lifecycle** with minimal manual intervention.

Use cases for Tiered Tables

- **Time-series data:** Logging, IoT sensor readings, application telemetry.
- **Archival:** Long-term retention of cold data for compliance.
- **Historical trend analysis:** BI tools querying years of data without impacting PGD performance.
- **Large, append-mostly tables:** Keep transactional footprint small while retaining full analytical access.

How Tiered Tables work in your HM architecture

- **PGD clusters:** Manage partitioning and automatic offload of old partitions to Iceberg.
- **PGFS storage locations:** Define object storage targets for offload.
- **Iceberg catalogs:** Optionally manage offloaded tables in a catalog (Lakekeeper or external).
- **Lakehouse Clusters:** Provide scalable analytical compute to query offloaded Iceberg data.
- **Monitoring:** Use HM monitoring tools and observability queries to track offload status and storage savings.

Prerequisites within EDB Hybrid Manager

Before implementing Tiered Tables in HM:

- **Active Hybrid Manager instance**
- **Provisioned PGD cluster:** Version 6.0+ with PGAA and PGFS extensions enabled
- **Lakehouse Cluster (recommended):** For querying offloaded data
- **Catalog service:** Optional, but recommended — HM-managed Lakekeeper or external REST-compatible catalog
- **Machine user for catalog (if using catalog):** With appropriate catalog data writer/reader permissions
- **Object storage:** S3-compatible, with credentials if private
- **User permissions:** Database user must have create/alter/execute privileges for BDR and PGAA functions

Main capabilities

- **Automated partitioning:** Define BDR AutoPartition strategy and `analytics_offload_period`.
- **Storage tiering:** Use PGFS or Iceberg catalog targets for offloaded data.
- **Query transparently:** PGD parent table queries can hit both local and Iceberg tiers. Lakehouse Clusters can query Iceberg tables directly.
- **Monitor status:** Track offload progress, validate Iceberg content, and observe space savings.

Getting started with Tiered Tables in Hybrid Manager

To begin using Tiered Tables with Hybrid Manager:

1. *Configure PGFS* for object storage access.
2. Learn *Tiered Tables in Hybrid Manager* and apply policies (AutoPartition, offload period) in PGD.
3. (Optional) Use a catalog (Lakekeeper or external) when interoperability across engines is needed.
4. Query offloaded Iceberg data via Lakehouse clusters.

Related resources

- *PGFS reference*
- *Tiered Tables in Hybrid Manager*

Observability tips

- Use HM dashboards for PGD cluster health and offload progress.
- Run analytics queries on `bdr.analytics_table` and partition views.
- Use `pg_total_relation_size()` to observe space reclaimed on PGD nodes.
- Use cloud storage console or analytics to track Iceberg object size growth.

Next topic

EDB Postgres Lakehouse Clusters on Hybrid Manager

CHAPTER 8

Sovereign Data and AI Factory

Sovereign Data and AI 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 AI 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*
- *Common workloads and use cases*
- *Frequently Asked Questions - AI Factory on Hybrid Manager*

To learn more about lifecycle operations, see Using Hybrid Manager .

8.5.1 Concepts and architecture

Sovereign Data and AI 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* .

Node types and roles

The system includes several node roles, each with distinct responsibilities and configurations.

Administration node

The administration node provides additional software packages, licensing, and support tooling for your Sovereign Data and AI Factory. You need one administration node per installation.

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.

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

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.

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

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 AI Factory is suitable for:

- Government or national data infrastructure
- Financial services subject to local data regulations
- Enterprises with strict cloud restrictions or sensitive IP

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 | Check | Check Embeddings | Convert internal data into vector form | No | Check Retrieval-augmented generation (RAG) | Combine Postgres + LLMs | No | Check Inference | Serve models locally using GPUs | No | Check Agentic workflows | Build AI agents that act autonomously | No | Check

For configuration details, see *System specifications* .

8.5.2 Frequently asked questions

Why use this system instead of a cloud service?

Sovereign Data and AI 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

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.

What software does it run?

The system runs:

- EDB Postgres Advanced Server or Extended Server
- Hybrid Manager for orchestration and observability
- AI 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

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.

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.

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 AI Factory nodes can be added to the internal cluster and managed through lifecycle workflows.

How do I scale the system?

You can order and rack additional compute nodes or GPU-enabled AI nodes.

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 AI Factory documentation.

Administration node

The central control node of the system. Hosts the Hybrid Manager interface, authentication services, and orchestration logic.

AI node

A GPU-enabled server used for model inference, vector processing, and agentic execution. Part of the optional AI workload configuration.

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.

Control plane

The set of three Hybrid Manager nodes responsible for cluster scheduling, monitoring, lifecycle operations, and access control. Operates independently of data workloads.

Compute node

A general-purpose high-performance node used for running Postgres clusters, analytics, or vector pipelines. Can be scaled out as needed.

Embedding

The process of converting structured or unstructured data into vector representations suitable for AI workflows such as retrieval and classification.

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.

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.

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.

Lifecycle services

Ongoing support and management for both hardware and software, including security patches, OS updates, Postgres upgrades, and firmware management.

EDB Postgres Distributed (PGD)

EDB's multi-node, high-availability Postgres solution. Optional for customers who require globally distributed or synchronous multi-region clusters.

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.

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.

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 AI 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.

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

Configuration #1: Core Compute

Includes:

- 1x Administration node
- 3x Hybrid Manager / Control plane nodes
- 3x Hyper 2U compute nodes

Administration node

Category	Product / Model	Description	Qty
System	SYS-111C-NR	1U X13SEDW-F Server	1
Processor	P4X-EMR4514Y-SRN	Intel Xeon Gold 4514Y 16-Core @ 2.0GHz	1
Memory	MEM-DR516L-SL01-	16GB DDR5-5600 1RX8 LP ECC RDIMM (Total 128GB)	8
Storage	HDS-MUN-MTFDKC	Micron 7450 MAX 3.2TB NVMe PCIe 4.0 U.3	2
Storage Controller	AOC-VROCSTNMOI	Intel VROC (RSTe) Standard HW Key	1
Networking	AOC-AG-I2M-O	AIOM Dual-Port 1GbE RJ45	1
Networking	AOC-S25G-I2S-O	Std Low Profile 25G Dual Port SFP28	1

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-MC(	Intel Xeon Platinum 6538N 32-Core @ 2.1GHz	2
Cooling	SNK-P0087V	Passive CPU Heatsink for X13 Platform	2
Memory	MEM-DR532L-SL01-	32GB DDR5-5600 2Rx8 LP ECC RDIMM (Total 1024GB)	32
Storage	HDS-SMN0-MZ1L29	Samsung PM9A3 960GB NVMe PCIe 4.0 M.2	2
Storage	HDS-TUN-KCD8XPU	Kioxia CD8-P 15.3TB NVMe PCIe 5.0 2.5"	4
Storage	HDS-TUN-KCD8XPU	Kioxia CD8-P 1.92TB NVMe PCIe 5.0 2.5"	2
Storage Controller	AOC-VROCPREMOI	Intel VROC Premium Controller (RAID 0, 1, 5, 10)	1
Networking	AOC-A100G-M2CM-	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

Configuration #2: Advanced AI

Includes all Core Compute components, plus:

- 2x 2-GPU Hyper 2U Servers
- 2x 8-GPU B200 10U Servers

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-MC(	Intel Xeon Platinum 6538N 32-Core @ 2.1GHz	2
Cooling	SNK-P0087V	1U Passive CPU Heatsink for X13 Platform	2
Memory	MEM-DR532L-SL01-	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-MZ1L29	Samsung PM9A3 960GB NVMe PCIe 4.0 M.2	2

continues on next page

Table 4 – continued from previous page

Category	Product / Model	Description	Qty
Storage	HDS-TUN-KCD8XPU	Kioxia CD8-P 15.3TB NVMe PCIe 5.0 2.5"	4
Storage	HDS-TUN-KCD8XPU	Kioxia CD8-P 1.92TB NVMe PCIe 5.0 2.5"	2
Storage Controller	AOC-VROCPREMOI	Intel VROC Premium Controller (RAID 0, 1, 5, 10)	1
Networking	AOC-A100G-M2CM-	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-GPU B200 10U Server (per server x2)

Category	Product / Model	Description	Qty
System	SYS-A21GE-NBRT-G	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
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 AI 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.

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 AI Factory exists to meet the needs of the following.

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.

Regulated environments

- Operates with limited internet dependency.
- Supports deployment lifecycles.
- Enables LLM and AI use cases in environments previously closed to them.

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.

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.

Lifecycle-managed infrastructure

- Joint EDB + Supermicro support agreements.
- Hardware and software patches integrated and coordinated.
- No reliance on customer-led infrastructure management or DevOps.

Supported workload patterns

The following workloads are supported end to end on the Sovereign Data and AI Factory system.

High-availability Postgres

- Run mission-critical systems of record with automatic failover.
- Observability and lifecycle are baked in.

Hybrid DBaaS

- Serve internal teams with multiple clusters from one interface.
- Role-based access control and cluster-level visibility.

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.

Retrieval-augmented generation (RAG)

- Query structured internal data and pass it to an LLM.
- Host both retrieval and inference within your own boundary.

On-premises inference

- Serve generative and embedding models on dedicated GPU hardware.
- Use standard containers (for example, NVIDIA NIM) to abstract model serving.

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 AI Factory system during the first 90 days—from delivery through active use and support.

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

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.

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.

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.

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.

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.