

EDB Postgres AI Hybrid Manager

Release 1.2

EnterpriseDB Corporation

Jun 25, 2025

Contents

1 EDB Postgres AI Hybrid Manager **3**
1.1 Hybrid Manager Overview . . . 3

2 Hybrid Manager Release Notes **5**
2.1 Hybrid Manager 1.2 release notes . . . 5
2.1.1 Release Highlights . . . 5
2.1.2 Deployment Options . . . 6
2.1.3 New Features & Enhancements . . . 6
2.1.4 Observability, Monitoring & Insights . . . 7
2.1.5 Sustainability: ESG Reporting (Preview) . . . 7
2.1.6 Platform Improvements . . . 7
2.1.7 Resources . . . 8

3 EDB Postgres AI Hybrid Manager **9**
3.1 Key capabilities . . . 9
3.2 Integrated solutions . . . 9
3.3 Next steps . . . 10
3.3.1 Hybrid Manager Architecture . . . 10
3.3.2 Supported Database Types . . . 12
3.3.3 Supported Postgres Distributions . . . 19

4 Installing the PG AI Hybrid Manager **21**
4.1 Installing the PG AI Hybrid Manager . . . 21
4.2 Planning . . . 21
4.2.1 Kubernetes (Generic) . . . 21
4.3 Considering a preinstalled hardware solution? . . . 21
4.3.1 Kubernetes planning and minimum recommended requirements . . . 22
4.3.2 Install Hybrid Manager on Red Hat OpenShift . . . 28
4.3.3 Install Hybrid Manager on AWS EKS . . . 42
4.3.4 Install Hybrid Manager on GCP GKE . . . 71
4.3.5 Customizations . . . 85

5 Using Hybrid Manager **95**
5.1 Tasks . . . 95
5.1.1 Hybrid Manager Terminology . . . 95
5.1.2 Hybrid Manager Console . . . 100
5.1.3 For Users . . . 107

5.1.4 For Kubernetes Administrators . . . 190

6 AI Factory in Hybrid Manager 283
6.1 Learning Paths . . . 283
6.2 AI Factory in Hybrid Manager Workloads . . . 283
6.2.1 Gen AI Workloads . . . 284
6.2.2 Model Management and Serving . . . 284
6.2.3 Vector Engine . . . 284
6.3 Get Started . . . 284
6.3.1 Enabling AI Factory . . . 284
6.3.2 Gen AI in Hybrid Manager . . . 288
6.3.3 Learn AI Factory in Hybrid Manager . . . 292
6.3.4 Model capabilities in Hybrid Manager . . . 302

7 Analytics in Hybrid Manager 307
7.1 Why Analytics in Hybrid Manager . . . 307
7.2 Core Concepts and Capabilities . . . 308
7.2.1 Apache Iceberg . . . 308
7.2.2 Delta Lake . . . 308
7.2.3 Tiered Tables with PGD . . . 308
7.2.4 Lakehouse Clusters . . . 308
7.3 How These Capabilities Work Together . . . 309
7.4 Getting Started: Quickstart Checklist . . . 309
7.5 Explore Further . . . 309
7.5.1 Delta Lake in Hybrid Manager . . . 310
7.5.2 Apache Iceberg in Hybrid Manager . . . 311
7.5.3 EDB Postgres Lakehouse Clusters on Hybrid Manager . . . 313
7.5.4 Learning Analytics in Hybrid Manager . . . 316
7.5.5 Tiered Tables in Hybrid Manager . . . 347

8 Sovereign AI and Data Factory 351
8.1 Why choose a hardware-based system? . . . 351
8.2 What's inside . . . 351
8.3 Supported workloads . . . 352
8.4 Hardware configurations . . . 352
8.5 Explore next . . . 352
8.5.1 Concepts and architecture . . . 352
8.5.2 Frequently asked questions . . . 354
8.5.3 Glossary and key terms . . . 356
8.5.4 System specifications . . . 357
8.5.5 Common workloads and use cases . . . 359
8.5.6 What to Expect . . . 361

EDB Postgres AI Hybrid Manager 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-prem**, or **hybrid environments**.

It works on the follow Kubernetes distributions: **Amazon EKS**, **Google GKE**, and **Red Hat OpenShift (RHOS)**.

CHAPTER 1

EDB Postgres AI Hybrid Manager

1.1 Hybrid Manager Overview

Area	Capabilities
Database Management	Provision, upgrade, and operate PostgreSQL and Distributed Postgres clusters across your Kubernetes environments.
AI Integration	Serve models using NVIDIA NIM, build knowledge bases, and connect AI pipelines — all governed and observable.
Analytics Tiering	Push cold data to object storage, query via Iceberg or Delta Lake, and combine transactional and analytical use cases.
Kubernetes-Native Monitoring	Track resource usage, query diagnostics, alerts, and cluster health using built-in dashboards (Prometheus, Grafana, Loki).
Security and Access	Connect with your identity provider, enforce least-privilege roles, and manage project-level access across teams.

CHAPTER 2

Hybrid Manager Release Notes

2.1 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.

2.1.1 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 AI and Data 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.

2.1.2 Deployment Options

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

Sovereign AI and Data Factory (Hardware Appliance)

For organizations requiring maximum control and zero external dependencies, the **Sovereign AI and Data 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)**.

2.1.3 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.

2.1.4 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.

2.1.5 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_2 emissions using configurable emission factors.
- **Region-Aware Reporting**: Customize dashboards with local energy cost ($/kWh) and emission rate (kg CO_2/kWh) based on the deployment region.
- **Integrated Grafana Dashboards**: Access the new **Power Usage & Carbon Footprint** dashboards directly from the Hybrid Manager UI.
- **Preview Phase Goals**: Validate demand for ESG observability and collect feedback from interested users.

2.1.6 Platform Improvements

Area	Highlights
Deployment options	New Sovereign AI and Data 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 UI, 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.

2.1.7 Resources

- *Installing Hybrid Manager on AWS EKS*
- *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 Key capabilities

Area	Description
Database provisioning	Deploy highly available PostgreSQL or Distributed Postgres clusters using UI, 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	Integrated dashboards using Prometheus, Grafana, and Loki provide 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.

3.2 Integrated solutions

Hybrid Manager unlocks core functionality across the EDB Postgres AI ecosystem:

- **AI Factory** – Build and run GenAI workloads using private data
- **Analytics** – Enable open-format analytics with Iceberg, Delta Lake, and tiered offloading
- **Migration** – Accelerate adoption of Postgres with in-platform migration tools

- **Monitoring** – Built-in observability stack with Kubernetes and database insights
- **Cluster Lifecycle** – Templates, upgrades, HA settings, and restore workflows

3.3 Next steps

- *How it fits in the Hybrid Manager architecture*
- *Supported Postgres Distributions*
- *Supported Database Types*
- *Install Hybrid Manager on AWS EKS*
- Use Hybrid Manager

3.3.1 Hybrid Manager Architecture

The outer-most 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, there are three main logical groupings to note:

1. **The Kubernetes control plane**: The Kubernetes control plane is comprised of the core Kubernetes components, such as `kube-apiserver` , `etcd` , `kube-scheduler` , `kube-controller-manager` , and, if on a cloud setup, `cloud-controller-manager` and implemented by two or three Kubernetes control nodes.
2. **The Control Compute Grouping**: The Control Compute Grouping of HM is a logical grouping of 70+ pods running on 2 or 3 Kubernetes worker nodes that implement the 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 is not as simply as pictured in the diagram. If needed, more worker nodes can be added to support more database-as-a-service resources.
3. **The Data Compute Grouping**: The Data Compute Grouping of HM is a logical grouping of Postgres clusters organized by namespaces and running 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. The number of worker nodes can be increased from three as more databases are added using HM, but maybe require either adding the worker nodes manually or configuring auto scaling alongside your infrastructure to do this automatically.

Both the Control Compute Grouping's worker nodes and the 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 manged service Auto Scaling groups . However, the HM's Kubernetes auto-scaling capability can be disabled to control costs. In addition, it's important to note that enabling auto-scaling functionality for on-premises deployments requires additional infrastructure configuration since there is no out-of-the-box auto-scaling feature like AWS's Auto Scaling groups for on-premises deployments. Monitoring the distribution and resource usage of pods and worker nodes implementing the database clusters can be achieved through the *included Grafana dashboards* or by using Kubernetes `kubectl commands <Tracing clusters back to their Kubernetes resources>` .

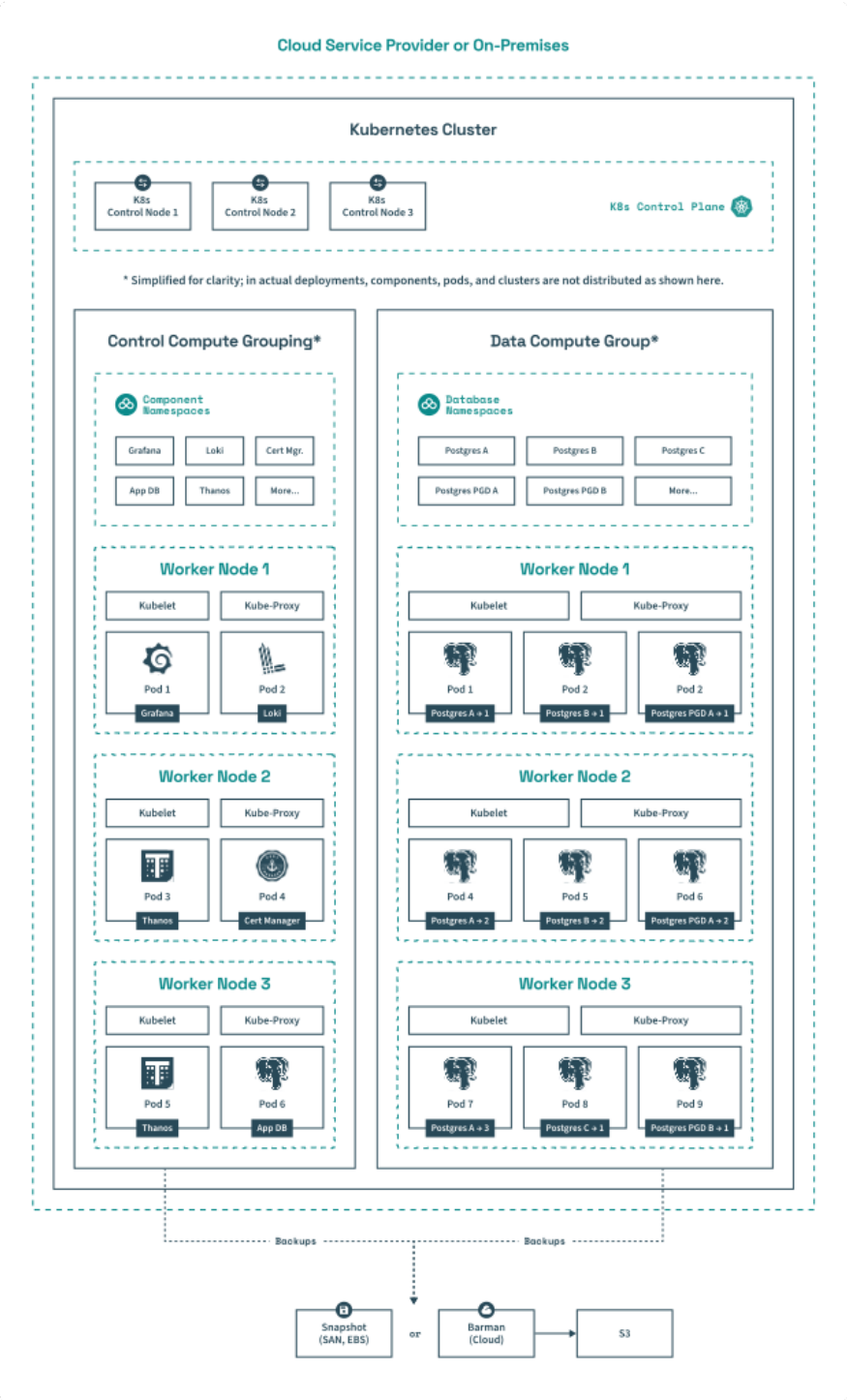


Fig. 1: High-level diagram of Hybrid Manager's architecture

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 SANs (Storage Area Networks) 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.

3.3.2 Supported Database Types

Hybrid Manager supports four cluster types:

- ref:*Single node <Single-node cluster>*
- ref:*High Availability and resilience <High Availability and resilience>*
- ref:*Advanced high availability <Advanced high availability>*
- ref:*Distributed high availability <Distributed high availability>*

You choose the type of cluster you want on the *Create Cluster* page.

Single-node cluster

For nonproduction use cases where high availability isn't a primary concern, a cluster deployment with high availability not 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.

High availability

The Primary/Standby High Availability option is provided to minimize 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.

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 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.



Fig. 2: Single node cluster

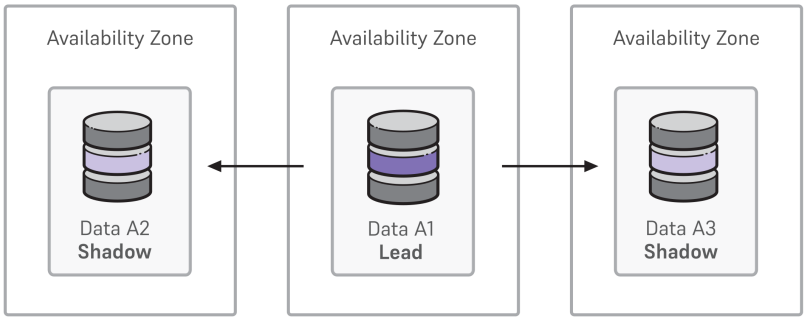


Fig. 3: Primary/standby high availability

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.

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 within that region.

They are designed for users who want a simpler setup and fewer configuration options than distributed high-availability clusters, while still benefiting 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 the PGD Connection Manager (embedded in every node) to ensure connections go to the current write leader node. This eliminates the need to manage conflicts across the cluster making application development easier.

width

70% :alt: Advanced high-availability cluster architecture

Advanced high-availability cluster architecture

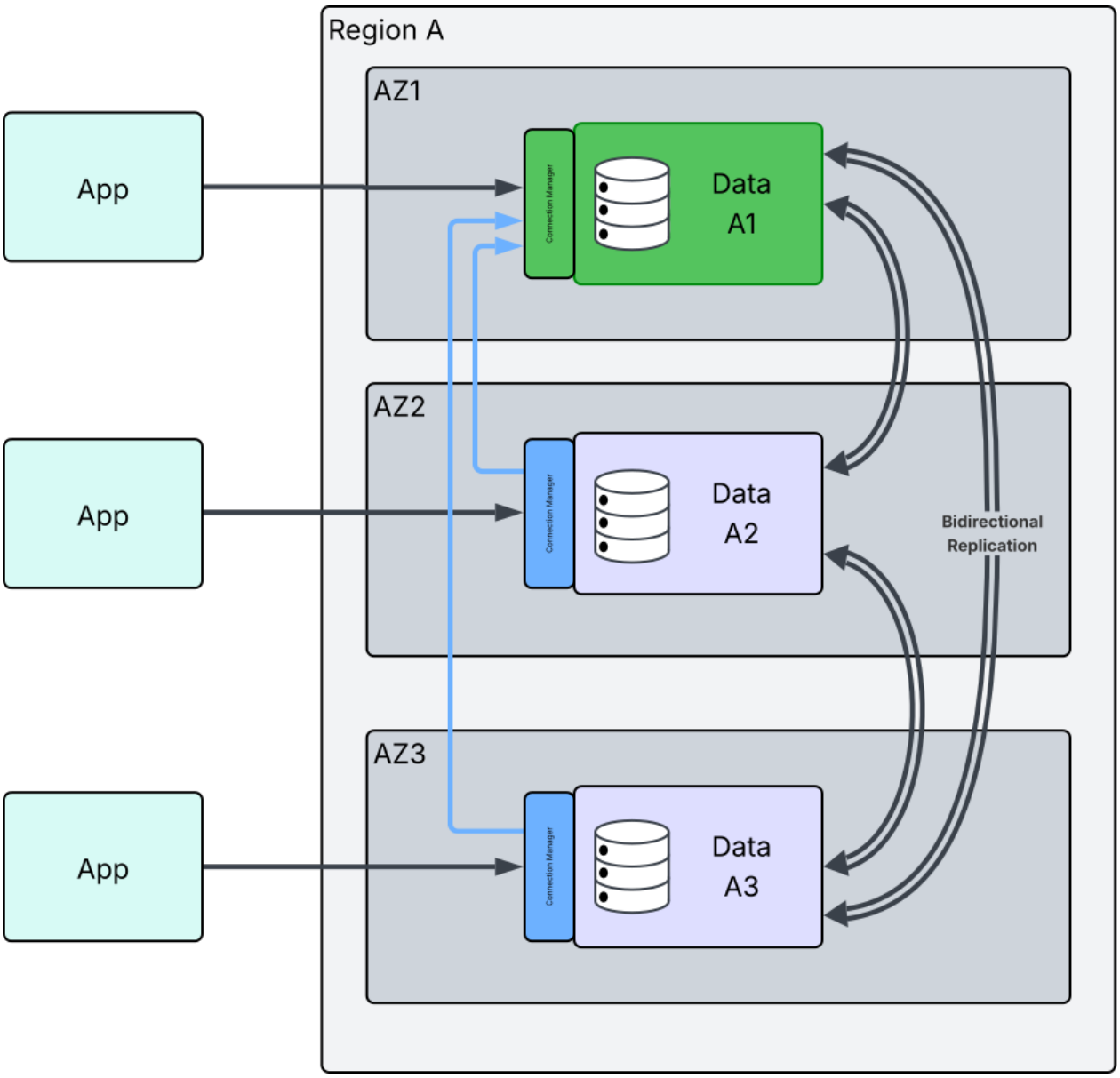
In this diagram, you can see the applications connecting to the PGD cluster through the Connection Manager ports. The Connection Manager is responsible for routing the read and write transactions to the appropriate nodes in the group. The write leader is responsible for handling all write transactions 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 are in sync.

AHA clusters support both EDB Postgres Advanced Server and EDB Postgres Extended Server database distributions. They do not support Community PostgreSQL distributions.

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 compared to a physical replication-based system. Distributed high-availability clusters let you deploy a cluster across multiple regions or a single region. For use cases where high availability across regions is a major concern, a cluster deployment with distributed high availability enabled can provide two data groups with a witness group in a third region.


Region A
AZ1
App
Connection Manager
Data
A1
AZ2
App
Connection Manager
Data
A2
Bidirectional
Replication
AZ3
App
Connection Manager
Data
A3

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

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 routes 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 an 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

  width
  : 70% :alt: region(2 data + 1 witness)

    region(2 data + 1 witness)

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

  width
  : 70% :alt: region(3 data)

    region(3 data)

Multiple groups with a witness node

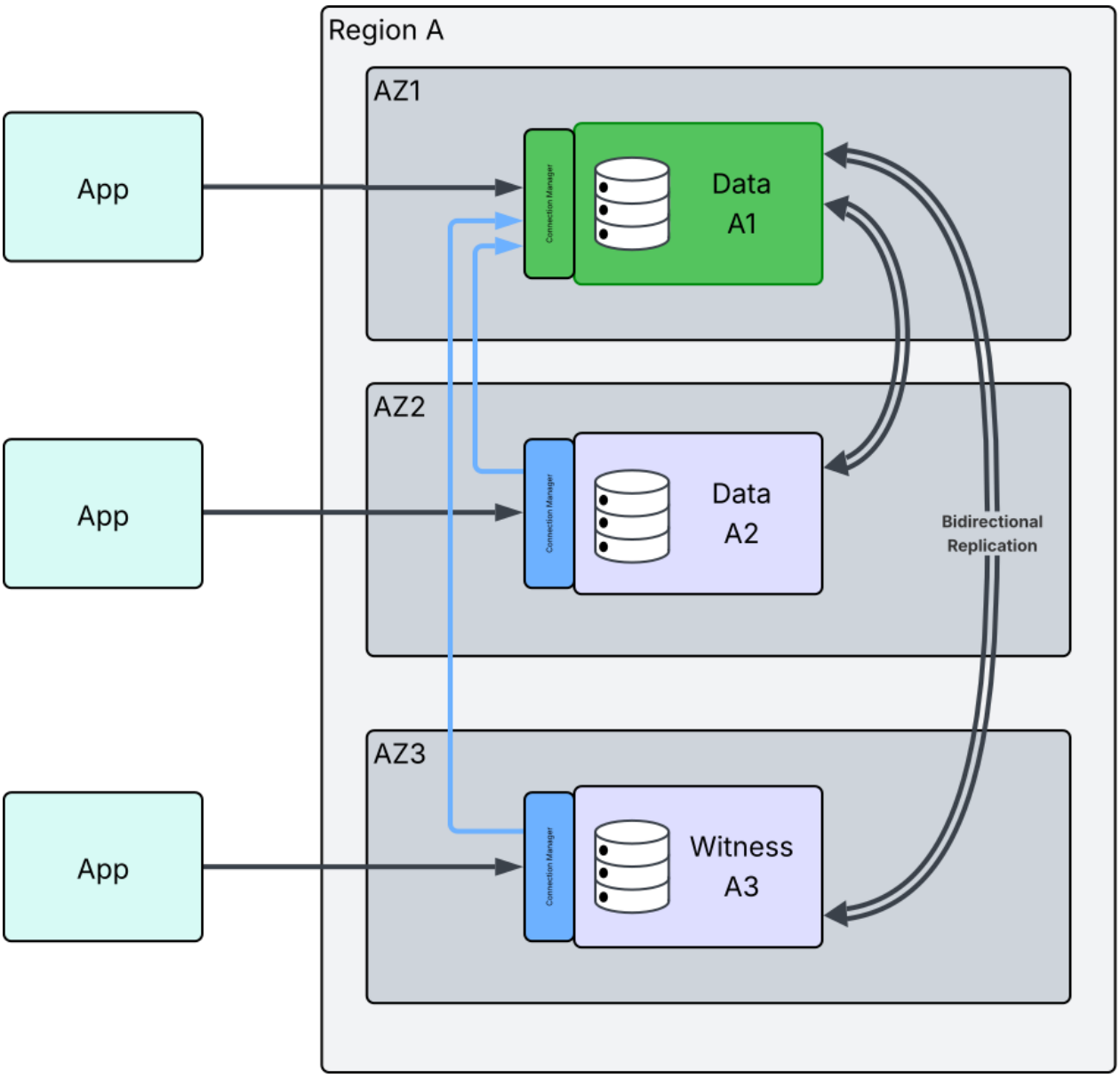
A configuration with multiple data locations has two data groups that contain either:

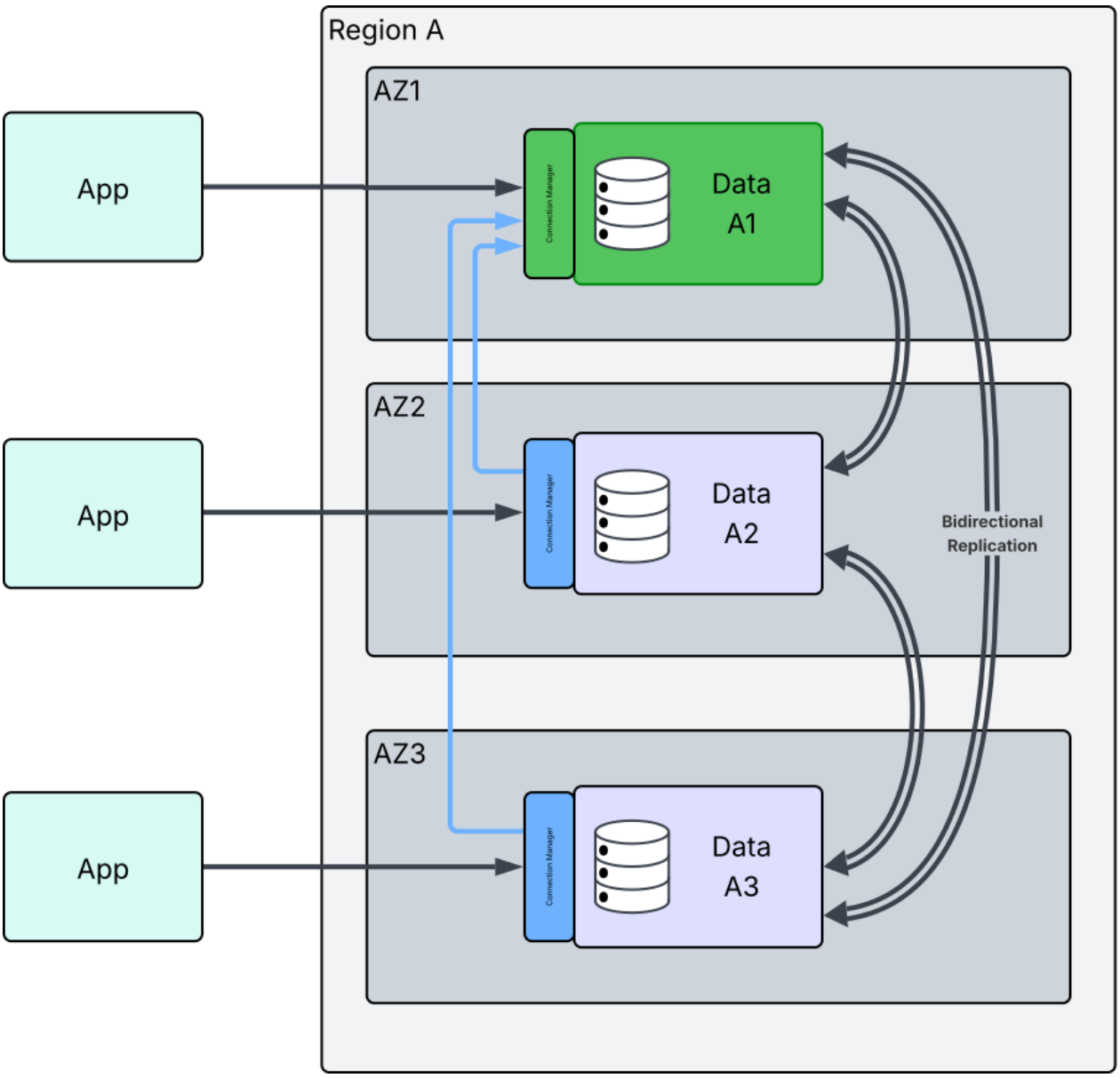
- Three data nodes in each group: * A data node and two shadow nodes in one group * The same configuration in another group * A witness node in a third group

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

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 lead. We recommend that applications that rely on advisory locking avoid using read-only workloads for those transactions.


Region A
AZ1
App
Connection Manager
Data
A1
AZ2
App
Connection Manager
Data
A2
Bidirectional
Replication
AZ3
App
Connection Manager
Witness
A3


Region A
AZ1
App
Connection Manager
Data
A1
AZ2
App
Connection Manager
Data
A2
Bidirectional
Replication
AZ3
App
Connection Manager
Data
A3

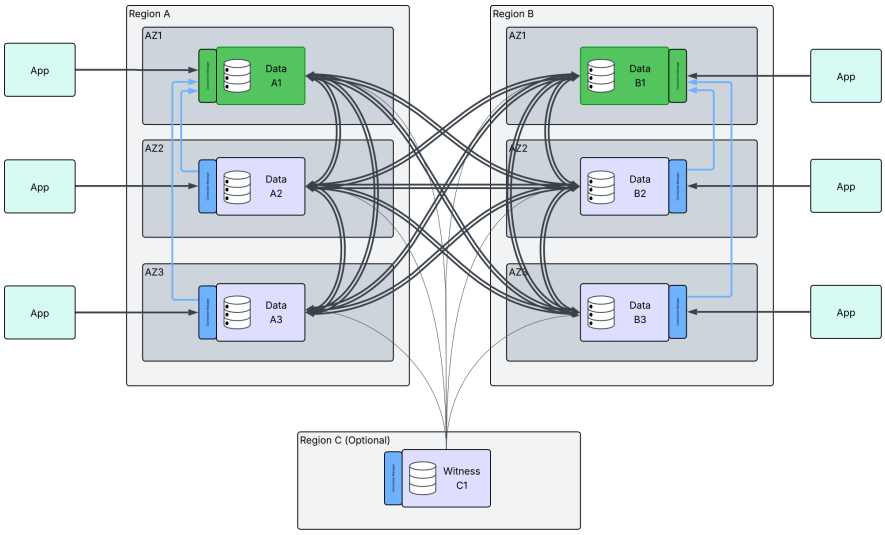


Fig. 4: groups(2 x 3 data + 1 witness)

3.3.3 Supported Postgres Distributions

PostgreSQL

PostgreSQL offers all the features gathered in the *PostgreSQL documentation* .

EDB Postgres Extended Server

EDB Postgres Extended Server adds a number of features to PostgreSQL to support transparent data encryption .

EDB Postgres Extended Server can also be combined with and fully utilize EDB' s powerful high-availability and multi-master replication solution, EDB Postgres Distributed .

EDB Postgres Advanced Server

EDB Postgres Advanced Server builds on the features of EDB Postgres Extended Server and supports features such as advanced security , advanced SQL and, optional, Oracle compatibility .

CHAPTER 4

Installing the PG AI Hybrid Manager

4.1 Installing the PG AI Hybrid Manager

PG AI Hybrid Manager (HM) provides orchestration, observability, and lifecycle operations for Postgres, analytics, and AI workloads across cloud and on-premises deployments.

This section provides installation guidance for users deploying Hybrid Manager on their own infrastructure, across supported Kubernetes platforms. If you're using a turnkey hardware system like *Sovereign AI and Data Factory (Hardware Appliance)* , Hybrid Manager is preinstalled and requires no setup.

4.2 Planning

4.2.1 Kubernetes (Generic)

Use this guide to prepare a custom Kubernetes cluster for Hybrid Manager.

- Planning your Kubernetes cluster for HM installation

This includes sizing recommendations, storage class expectations, and node group setup.

4.3 Considering a preinstalled hardware solution?

If you're looking for a fully managed deployment of Hybrid Manager and its supporting infrastructure, consider the *Sovereign AI and Data Factory (Hardware Appliance)* — a turnkey hardware solution delivered by EDB.

This system includes:

- Preinstalled, end-to-end support of Hybrid Manager and Redhat Openshift
- Full lifecycle management of hardware and software
- Built-in support for Postgres, vector workloads, and GPU-powered inference

Learn more at the Engineered Systems overview . —

4.3.1 Kubernetes planning and minimum recommended requirements

This document provides both general guidance for planning your Kubernetes cluster for to install Hybrid Manager and the minimum recommended specifications for the cluster.

- *General guidance for planning your Kubernetes cluster*
- *Minimum recommended specifications for Kubernetes clusters*

Planning Your Kubernetes Cluster for Hybrid Manager Installation

When preparing to deploy Hybrid Manager (HM), make certain to first consider the following preliminary aspects of the design:

1. Use Case

- What workloads need to be managed?
- Are you deploying in production, testing, or a hybrid environment?

2. Base Infrastructure

- Is the cluster to be hosted by a Cloud Service Provider (CSP) or on-premises? - For on-premises, consider bare-metal, VM, or private cloud options.

3. Networking

- Availability zones and subnets (public/private)
- NAT Gateway/Egress
- Network security rules
- Resource tagging and latency considerations
- **Network security**
- Cluster network policies - Implement Kubernetes network policies to control traffic flow between pods, namespaces, and external resources.
- Encrypted communication - Use TLS for all communication between HM components and external clients. - Ensure in-cluster service communication is encrypted
- Firewalls and security groups - For on-premises deployments, configure firewalls to restrict access to Kubernetes API servers and control-plane nodes. - For cloud-based deployments, utilize security groups to restrict inbound and outbound traffic to the cluster

4. Kubernetes Cluster Settings

- Kubernetes version.
- Identity management (e.g., LDAP, SAML, OpenID Connect).
- Resource tags, secrets/config maps.
- Taints and tolerations for node groups.

5. Node Group Configuration

- Instance types and compute groupings.

6. Permission Policies

- PGAIHM needs permissions to the whole Kubernetes cluster
- Node role permissions and RBAC settings.
- Quotas and limit ranges.
- Network policies.

7. Storage Integration

- CSI drivers and object storage (e.g., S3-compatible).
- Volume types, persistent volume claims, and snapshot support.

8. Load Balancer/Ingress Integration

- Cloud provider or on-premises specifics.
- Permissions for load balancer integration.

9. Data security

```
- Encryption at rest
    - Ensure persistent volumes used by HM have encryption enabled
    - Use a storage backend that supports encryption (e.g. AWS S3 for backups in EKS,␣
↪NVMe with encryption for local storage on-premises)

- Encryption in transit
    - Enable encryption for all data transfer between HM components and the database␣
↪clusters they manage.

- Secrets management
    - Store all sensitive information (e.g. database passwords, API keys) in Kubernetes␣
↪Secrets.
    - Use a secure external secrets manager (e.g. HashiCorp Vault, AWS Secrets Manager)␣
↪when possible.
```

10. Platform-Specific Custom Configurations

- Differences between Red Hat OpenShift, AWS EKS, and other distributions.

By addressing these considerations, you can ensure your Kubernetes cluster is properly configured to support HM in any deployment scenario.

Minimum recommended specifications for the Hybrid Manager

General recommended resource requirements

When configuring a Kubernetes cluster to run PGAIHM, the minimum recommended general resource requirements are:

New Kubernetes cluster

Running PGAIHM on an existing Kubernetes cluster is not supported at this time. Therefore, you must create a new Kubernetes cluster for the purposes of hosing PGAIHM.

Compute

When considering compute requirements for the Hybrid Manager (HM), it is important to understand that PGAIHM itself is relatively lightweight out of the box and it' s only the workloads from different use cases which may require scaling.

The minimum requirements for PGAIHM are 16GB of RAM, 4 vCPUs, and 50GB of disk on one machine; these are the resources PGAIHM consumes upon installation. This is only recommended for testing PGAIHM, not for production use.

Handling workloads

For general production use, 1-20 Postgres clusters (depending on size of the workload), the following compute recommendations are suggested:

- **6 Kubernetes worker nodes**, 3 for HM control components, 3 for HM database nodes, each with:
- 8+ cores
- 32 GB of memory
- 64 GB of storage

For larger data workloads, more worker nodes are likely required.

AI

If planning to run **AI workloads**, add at least:

- 1 GPU node, 16-32 cores, 30 Gi storage space

However, if your AI workloads are very large or you have many of them, you likely need more GPU resources. To see exact specification you may need for your particular model needs, consult the NVIDIA documentation .

Object store

- **S3 compatible object store**
- `Ceph <https://github.com/ceph>`_ , `MinIO <https://github.com/minio/minio>`_ ,
- `AWS S3 <https://aws.amazon.com/s3/>`_ for EKS deployments

Storage Layer

- A **Container Storage Interface** (CSI) driver compatible with Kubernetes for dynamic provisioning of persistent volumes
- OpenShift Container Storage (OCS) or Rook-Ceph for on-premises deployments
- AWS EBS CSI driver for EKS
- **Storage classes**
- At least one default storage class configured with dynamic volume provisioning

- For on-premises deployments, NVMe is recommended for optimal performance. For example, systems like the Supermicro SYS-221H-TNR are suitable for deployments requiring high-speed storage.
- SAN - Must use a Storage Area Network (SAN) for centralized and high-performance needs of HM. - Ensure SAN supports encryption at rest and integrates with Kubernetes via a CSI driver. - Provide a dedicated and secure network for SAN traffic.
- Snapshot support - Ensure the CSI driver supports volume snapshots if using snapshot backups in HM

Configuration settings

The following configuration settings are recommended for running HM on Kubernetes:

- Kubernetes version 1.20 or later.

Network settings

- **Pod network**
- Minimum CIDR range for pod IPs depends on the Cloud Service Provider or your on-premises needs. - Example: /22 or more for AWS EKS is advised
- **Service network** - Minimum CIDR range for Kubernetes services depends on the CSP or your on-premises needs.
- **External access** - If accessing HM using the internet, these ingress/egress requirements apply
- **DNS** - Cluster-wide DNS service must be operational.

Security settings

- **Access control**

```
- Ensure that Kubernetes RBAC is configured with the principle of least privelege. Users,
services, and pods should only have access to the resources they need.
```

- **Authentication**

```
- Use secure authentication mechanisms such as integration with an identity provider␣
↪(IDP)
    - LDAP, SAML, or OpenID Connect
- Multi-factor Authentication (MFA)
    - Enforce MFA for all administrator accounts accessing the HM.
```

- **Network security**
- Cluster network policies - Implement Kubernetes network policies to control traffic flow between pods, namespaces, and external resources.
- Encrypted communication - Use TLS for all communication between HM components and external clients. - Ensure in-cluster service communication is encrypted
- Firewalls and security groups - For on-premises deployments, configure firewalls to restrict access to Kubernetes API servers and control-plane nodes. - For cloud-based deployments, utilize security groups to restrict inbound and outbound traffic to the cluster
- **Data security**
- Encryption at rest - Ensure persistent volumes used by HM have encryption enabled - Use a storage backend that supports encryption (e.g. AWS S3 for backups in EKS, NVMe with encryption for local storage on-premises)

- Encryption in transit - Enable encryption for all data transfer between HM components and the database clusters they manage.
- Secrets management - Store all sensitive information (e.g. database passwords, API keys) in Kubernetes Secrets. - Use a secure external secrets manager (e.g. HashiCorp Vault, AWS Secrets Manager) when possible.
- **Backup security**
- Ensure backup data stored in Barman Cloud or snapshots is encrypted.
- Use RBAC policies for backup storage to limit access to authorized personnel and services.
- Regularly test backup restoration processes to ensure they are secure and effective.
- **Monitoring and auditing**
- Audit logs - Enable and monitor Kubernetes audit logs for administrative actions and unusual activity
- Log aggregation - Use the intergrated Grafan and Loki stack to centralize and analyze logs for security events.
- Alerting - Configure alerts for critical events such as failed authentications, unauthorized access attempts, or resource policy violations.
- **Container Security**
- Use container images from trusted sources and scan them regularly for vulnerabilities (e.g with tools like Trivy or Clair)
- Apply Kubernetes Pod Security Standards (PSS) to enforce baseline security configurations for pods.
- Restrict container priveleges - Disable privilege escalation. - Set `runAsNonRoot` to ture in pod security contexts
- **Kubernetes cluster security**
- Disable anonymous access to the Kubernetes API server.
- Regularly patch and update Kubernetes and all associated components.
- **Physical security** (for on-premises deployments)
- SAN Security - Secure physical access to SAN hardware, ensuring it is located in a restricted, access-controlled data center. - Implement redundancy and failover configurations for critical SAN components to ensure high availability. - Isolate SAN network traffic to prevent unauthorized access and reduce the risk of interception.
- **Compliance**
- Ensure HM deployment adheres to organizational or industry-specific security standards (e.g. SOC 2, HIPAA, GDPR).
- Regularly review compliance and update configurations as standards evolve.

Please see capcacity planning document for guidance on resources based on HM functionality or use cases.

2. Kubernetes Platform and Version Requirements

HM is designed to run on various conformant Kubernetes distributions. Supported platforms include:

- Amazon Elastic Kubernetes Service (EKS)
- Google Kubernetes Engine (GKE)
- Red Hat OpenShift (RHOS)

(Note: Specific version compatibility should be checked against the HM release notes.)

3. Kubernetes Nodes and Processor Requirements

3.1. Node Characteristics

- **Operating System:** Nodes must be Linux-compatible.
- **Processor Architecture:** HM components primarily target AMD64/x86-64 architecture. While PostgreSQL itself can run on ARM64, mixed-architecture Kubernetes clusters for HM deployments are not standard and may present support complexities.

3.2. Minimum Environmental Layout

A typical minimum production-viable environment consists of:

- **Location:** Single city.
- **Infrastructure:** Single physical data center (for on-premises) OR a single region in a Cloud Service Provider (CSP).
- **Kubernetes Cluster:** One Kubernetes cluster.
- **Node Allocation for HM:** - A minimum of 3 Kubernetes nodes are required to host the HM control plane components, ensuring high availability. These can be Kubernetes control plane nodes or worker nodes, labeled appropriately.
- **Node Allocation for PostgreSQL (Optional but Recommended):** - While not a strict HM requirement (PostgreSQL can be co-located if resources and labels permit), dedicating at least 3 additional Kubernetes worker nodes for PostgreSQL instances is recommended for production workloads to ensure resource isolation and performance.

4. External Services Requirements

HM, running within Kubernetes, relies on several external services that must be provisioned and accessible to the Kubernetes cluster:

- **Networking:**
- Local network infrastructure (on-premises) or a Virtual Private Cloud (VPC) in a CSP.
- **Load Balancer:** Required for exposing HM and PostgreSQL services. This is typically a managed service in CSPs. For on-premises RHOS, a solution like MetalLB or an enterprise load balancer must be integrated.
- **DNS:** Reliable DNS resolution for service discovery within and outside the cluster. Control over DNS records for service exposure is necessary.
- **Identity Provider (IDP):** Optional, for *integrating with existing authentication systems* .
- **Object Storage:** An S3-compatible object storage solution is mandatory for backups and other platform features. This is a managed service in CSPs; for on-premises, a compatible solution must be provisioned and accessible to RHOS.
- **Key Management System (KMS):** Optional, for managing encryption keys.
- **Block Storage:** Persistent, high-performance block storage is required for PostgreSQL data volumes and potentially other stateful components of HM.
- In CSPs, this is typically a service like AWS EBS, GCP Persistent Disk, or Azure Disk Storage.
- For on-premises deployments, a Storage Area Network (SAN) or equivalent high-performance storage solution integrated with Kubernetes (via CSI drivers) is necessary. A SAN can be considered functionally equivalent to CSP block storage services in this context.

5. Dependent External Services (Connectivity)

The Kubernetes cluster nodes running HM components and EDB Postgres instances require outbound internet access to the following EDB services:

- **EDB Container Registry:** For pulling container images. - Endpoint: docker.enterprisedb.com
- **EDB Entitlements Service:** For license validation and service entitlements. - Endpoint: biganimal.com

4.3.2 Install Hybrid Manager on Red Hat OpenShift

You can install Hybrid Manager (HM) on Red Hat OpenShift (RHOS).

- Perform the *Prerequisites* .
- Understand the difference between the two installation methods: *Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator*
- Perform the installation using EDB Postgres AI Operator:
- *Installing Hybrid Manager on RHOS using EDB Postgres AI Operator*
- *Update your DNS zone records*
- Validate your install by *Connecting to Hybrid Manager*

Red Hat Open Shift installation prerequisites

Red Hat Open SHift has prerequisites to meet before you can install EDB Postgres AI Hybrid Manager (HM). Fulfill the prerequisites to ensure that you have everything you need before you start the installation process.

Prerequisites

- *A system from which you can run the installation process*
- *A RHOS cluster configured for use with HM*
- *Necessary environment variables set*

Next steps

Once you have fulfilled these requirements, you can move on to *Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator* .

Install system

You need a system from which you can run the installation process. This system needs the following software installed:

- `OpenShift CLI (oc) <https://docs.redhat.com/en/documentation/openshift_container_platform/4.17/html/cli_tools/openshift-cli-oc#cli-getting-started>`_
- `kubectl <https://kubernetes.io/docs/reference/kubectl/>`_ to manage your Kubernetes cluster
- `yq <https://github.com/mikefarah/yq>`_ to prepare the Hybrid Manager installation files

You can install these tools using the linked instructions or by using the instructions for your operating system that follow.

macOS

```
brew install kubectl yq
```

Linux

```
sudo apt install kubectl
sudo snap install yq
```

```
sudo yum install kubectl
sudo wget https://github.com/mikefarah/yq/releases/latest/download/yq_linux_amd64 -O /
↪usr/bin/yq && sudo chmod +x /usr/bin/yq
```

```
sudo apt-get install kubectl
sudo wget https://github.com/mikefarah/yq/releases/latest/download/yq_linux_amd64 -O /
↪usr/bin/yq && sudo chmod +x /usr/bin/yq
```

Windows

```
winget install kubectl --source winget
winget install yq --source winget
```

1. Install kubectl .
2. Install yq .

Next steps

After you've installed the necessary software, you can move on to *setting up aa RHOS cluster* .

Creating an Red Hat OpenShift cluster for use with Hybrid Manager

After you've *installed your system* , you're ready to create a Red Hat OpenShift (RHOS) cluster for use with Hybrid Manager (HM).

Preparing an OpenShift cluster for HM installation

To successfully install HM on a Red Hat OpenShift (RHOS) on-premises cluster, certain prerequisites and configurations must be in place.

Preconfiguration cluster requirements

- HM currently supports:
  - OpenShift 4.17 with Kubernetes 1.30

or:

```
* OpenShift 4.18 with Kubernetes 1.31
```

Step 1: Configure networking

- Define network policies. Ensure appropriate network policies are set up to manage traffic between Kubernetes control plane, worker nodes, and external systems.
- Define ingress/egress rules.
- Ensure communication between cluster components and the HM stack is secure. - Validate ingress/egress network policies to allow cluster components to communicate while restricting unnecessary external access.
- Set up load balancer: If using an external load balancer, configure it to handle OpenShift API traffic and worker node communication.
- Configure DNS: Ensure DNS records are set up for the cluster, including internal service discovery for HM.
- Ensure the cluster has adequate private network isolation for worker nodes. OpenShift typically handles this via the software-defined network (SDN) configuration.

Step 2: Configure node size and resources

To support HM and its associated components, the nodes in your OpenShift cluster must meet the resource requirements.

Node sizing

- **Master nodes**: OpenShift control plane nodes must meet the standard requirements for resource allocation (for example, CPU, memory, and disk size).
- General recommendations: - 3 master nodes each with:
  - CPU: - Minimum: 4 vCPUs (for up to 10 worker nodes) - Recommended: 8 vCPUs for medium-sized clusters (10-50 worker nodes) - For >50 worker nodes: 16+ vCPUs
  - Memory: - Minimum: 32 GB RAM (for up to 10 worker nodes) - Recommended: 64 GB RAM or more for larger workloads (10-50 worker nodes) - For >50 worker nodes: 64+ GB RAM
  - Disk - Minimum: 100 GB SSD - Recommended: 200 GB SSD for larger workloads - For >50 worker nodes: >200 GB SSD
- **Worker nodes**: Allocate sufficient resources to worker nodes to handle HM control components and Postgres clusters: 3 worker nodes for the HM control components and 3 worker nodes for the Postgres workloads are recommended for smaller workloads (5-20 Postgres clusters). As you scale up Postgres workloads from this, you may require more worker nodes for the Postgres workloads and eventually even more control worker nodes for supporting the increasing number of Postgres workload worker nodes.
- General recommendations:

```
* 6 worker nodes each with: * CPU: * Minimum: 4 vCPUs (this could be higher depending on␣
↪the type of workload) * Memory:  * Minimum: 32 GB of RAM per node * Disk:  * Minimum:␣
↪100 GB of persistent storage per node (adjust based on database and logging␣
↪requirements). Use fast disks (SSD) for optimal performance.
```

To (optionally) use different types of nodes for the HM control nodes and the HM Postgres data nodes hosting the Postgres workloads, RHOS requires additional machine sets:

- Control nodes machine set:

```
spec:
    replicas: 3
    template:
```

(continues on next page)

(continued from previous page)

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

```
-   Data nodes hosting Postgres workloads machine set:
```

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

Resource reservations

- To prevent contention with HM workloads, reserve CPU and memory for system components.
- Allocate additional resources for HM monitoring and observability tools (for example, Grafana, Loki, and Prometheus).
- Node pool configuration:
- If desired, use taints and tolerations to isolate HM workloads on specific nodes.
- Label nodes for specific tasks (for example, “monitoring”, “database”) to optimize resource allocation.
- Scaling:

```
-   Set up autoscaling to ensure the cluster can dynamically handle increased demand.␣
↪OpenShift's Horizontal Pod Autoscaler (HPA) can be used to scale pods as needed.
```

Networking bandwidth

Ensure each node has adequate networking capacity to handle HM communication and external data transfer (for example, S3 backups).

Baseline recommendations:

Component	Recommended bandwidth	Justification
K8s control plane nodes	1 Gbps+ per node	Handles internal Kubernetes traffic, API server requests, logs/metrics, and orchestration tasks.

continues on next page

Table 1 – continued from previous page

Component	Recommended bandwidth	Justification
Worker nodes (HM)	1 Gbps+ per node	For HM control components, PostgreSQL replication, internode communication, and metrics/logs.
External bandwidth	1-20 Gbps (aggregated)	S3 backups and inter-cluster replication may require high throughput.

Step 3: Install the required CSI drivers

HM uses storage and snapshot features that depend on CSI drivers. Verify the following drivers are installed and properly configured in your OpenShift cluster:

- Persistent storage driver: Install a storage class driver compatible with your storage backend (for example, OpenShift Container Storage, Ceph, or other on-prem solutions).
- Snapshot controller: Configure the CSI snapshot controller for volume snapshot management.

Step 4: Configure storage classes

Default storage class: Set up a default storage class optimized for HM workloads.

Custom storage classes: Create additional storage classes if specific workloads require custom configurations.

Step 5: Prepare namespace and secrets

Namespace for HM: Create a dedicated namespace for HM components to ensure isolation and manageability.

```
oc create ns edbpgai-bootstrap
```

Secrets: Create Kubernetes secrets for any required credentials, such as object storage credentials, database access tokens, or other sensitive information. The following are HM-specific secrets and configurations. The first ImagePullSecret is necessary for HM to function properly. Griptape and DeltaLake object storage and secret for Catalog are optional, as they depend on whether you're setting up your cluster for GenAI Builder or Catalog.

- **Configure ImagePullSecret and namespace** (necessary for HM)

1. Set environmental variables.

So that you can more easily log in to OC Registry, there are a few container registry variables to set:

```
export CONTAINER_REGISTRY_URI=<local-registry-uri>
export CONTAINER_REGISTRY_USERNAME=<container-registry-username>
export CONTAINER_REGISTRY_PASSWORD=<container-registry-password>
```

2. Prepare the pull secret file.

Prepare the directory for the image pull secret:

```
tmpdir=$(mktemp -d /tmp/pull-secret-<your-pull-secret-name)
chmod 0700 "$(tmpdir)"
```

3. Log in to the registry and update the secret.

Log in to OC Registry:

```
oc registry login \
--registry=$${CONTANER_REGISTRY_URI}" \
--auth-basic="${CONTAINER_REGISTRY_USERNAME}:${CONTAINER_REGISTRY_PASSWORD}"
    --to="$tmpdir/pull-secret"
```

Then update the secret:

```
oc set data secret/pull-secret -n openshift-config \
    --from-file=.dockerconfigjson=" $tmpdir/pull-secret"
```

4. Clean up.

Clean up the temporary directory you made for the secret:

```
rm -rf $tmpdir
```

- ** *Enabling GenAI Builder on Hybrid Manager* (optional for GenAI Builder)**
- ** *Enabling Data Catalogs on Hybrid Manager* (optional for using Catalog)**
- ** *Customizing Migration Portal secrets for secure internal communication* (optional to secure internal communications with Migration Portal)**

Step 6: Configure authentication and security keys

HM and its underlying components require secure authentication mechanisms to ensure proper communication between components and to protect sensitive data.

1. **Generate AES-256 encryption key.**

HM uses an AES-256 encryption to secure sensitive data during communication or at rest (for example, database credentials, tokens). To generate a random AES-256 encryption key:

```
export AES_256_KEY=$(openssl rand -base64 32)
```

2. **Store the key in Kubernetes.**

To make the key accessible to HM and associated services, create a Kubernetes secret in the appropriate namespace:

1. Run the following command to create the secret:

```
kubectl create secret generic hm-auth-key \
    --namespace <hm-namespace> \
    --from-literal=aes-256-key=$AES_256_KEY
```

2. Verify the secret:

```
kubectl get secret hm-auth-key --namespace <hm-namespace>
```

Step 7: Configure object storage

Object storage backend: To support HM backups, connect the cluster to a dedicated object storage solution (for example, MinIO, Ceph Object Gateway, or AWS S3-compatible storage).

Among others, Hybrid Manager uses this bucket to store:

- Managed storage locations

- Postgres WALs + backups
- Logs (both Postgres and internal services)
- Metrics (both Postgres and internal services)

Create a bucket

Create a MinIO or Ceph bucket .

Create a bucket user

After creating a dedicated bucket for Hybrid Manager with your provider, create an identity user with full access to the bucket, as well as AWS static credentials for the identity user to grant the Hybrid Manager access to the bucket.

Download the MinIO Client .

1. Retrieve the MinIO deployment name and create a username and password:

```
export MINIO_DEPLOYMENT_NAME=<minio_deployment_name> # deployment name to minio
export BUCKET_NAME=<minio_bucket_name> # bucket name
export AWS_ACCESS_KEY_ID=<minio_user_name> # user name
export AWS_SECRET_ACCESS_KEY=<minio_user_password> # password to the user
export MINIO_POLICY_NAME=<minio_policy_name> # policy to minio
```

1. Create a new user:

```
mc admin user add ${MINIO_DEPLOYMENT_NAME} ${AWS_ACCESS_KEY_ID} ${AWS_SECRET_ACCESS_KEY}
```

1. Create one MinIO policy:

```
cat << EOF > policy.json
{
    "Version" : "2012-10-17",
    "Statement": [
        {
            "Effect" : "Allow",
            "Action" : [
                "s3:*"
            ],
            "Resource" : [
                "arn:aws:s3:::${BUCKET_NAME}",
                "arn:aws:s3:::${BUCKET_NAME}/*"
            ]
        }
    ]
}
EOF
```

1. Apply the policy:

```
mc admin policy create ${MINIO_DEPLOYMENT_NAME} ${MINIO_POLICY_NAME} ./policy.json
```

1. Attach the policy to the user:

```
mc admin policy attach ${MINIO_DEPLOYMENT_NAME} ${MINIO_POLICY_NAME} --user ${AWS_ACCESS_
↪KEY_ID}
```

Download Ceph's command line interface Radosgw-admin utility .

1. Retrieve the bucket and some environment variables:

```
export BUCKET_NAME=<bucket_name> # bucket name
export USER_NAME=<user_name> # user name
```

1. Create the user:

```
radosgw-admin user create --uid ${USER_NAME} --display-name= ${USER_NAME} > user.json

export AWS_ACCESS_KEY_ID=$(cat user.json | jq -r .keys[0].access_key)
export AWS_SECRET_ACCESS_KEY=$(cat user.json | jq -r .keys[0].secret_key)
```

1. Grant the user access to the bucket:

```
radosgw-admin bucket link --bucket ${BUCKET_NAME} --uid ${USER_NAME}
```

Create a secret for bucket access

After preparing the dedicated user for HM access to the bucket, create a secret that associates the created user with the Hybrid Manager to provide access for object storage.

```
apiVersion: v1
kind: Secret
metadata:
  name: edb-object-storage # name cannot be changed
  namespace: default # namespace cannot be changed
stringData:
    auth_type: credentials

    # Optional: Used only when the object storage servers certificate is not issued by a␣
↪well-known CA
    #
    # Base64 string of the CA bundle for the certificate used by the object storage␣
↪server
    aws_ca_bundle_base64: xxx

    # Required: Endpoint URL to the object storage
    aws_endpoint_url_s3: xxx

    # Required: AWS Static Credentials - AWS_ACCESS_KEY_ID
    aws_access_key_id: xxx

    # Required: AWS Static Credentials - AWS_SECRET_ACCESS_KEY
    aws_secret_access_key: xxx

    # Required: Bucket name
    bucket_name: xxx
```

(continues on next page)

(continued from previous page)

```
    # Required: Region of the bucket
    aws_region: xxx

    # Optional: true or false
    #
    # When server-side encryption is disabled, set this to true. By default, its value␣
↪is false, indicating that server-side encryption is enabled.
    server_side_encryption_disabled: bool
```

Step 8: Validate network security

- Firewall rules: Ensure firewall rules allow necessary communication between the OpenShift cluster, external object storage, and any external dependencies of HM.
- Ingress/egress rules: Restrict access to only the required endpoints and IP ranges to minimize exposure.

Step 9: Adjust cluster configurations

Some default OpenShift configurations may need adjustments:

- Node pool tuning: Optimize node configurations for HM workloads, including resource allocation and taints/tolerations.
- Storage class adjustments: Replace or modify default storage classes as needed.
- Networking policies: Refine networking configurations to ensure compatibility with HM requirements.

Step 10: Install required add-ons

- Install and configure any required OpenShift operators or add-ons:
- Service mesh: HM relies on istio service mesh.
- Monitoring and logging: Set up OpenShift monitoring and logging solutions to integrate with HM observability components.

Proceed to setting necessary environmental variables

With the cluster configured, proceed to the next phase of the pre-installation: *setting the necessary environmental variables for installation* .

Set the necessary environmental variables for install

The HM cluster needs access to certain user domains during the installation.

Set domain variables

```
export DEFAULT_INGRESS_DOMAIN=<rhos default ingress domain name> # rhos default ingress␣
↪domain
```

Important

DEFAULT_INGRESS_DOMAIN is the default ingress domain name of the cluster. You can query it using the following command:

```
oc get ingresses.config/cluster -o jsonpath={.spec.domain}
```

The ingresses.config/cluster object is created during the installation.

See the OpenShift documentation for more.

```
export OPENSHIFT_CONSOLE_DOMAIN_NAME=<openshift console domain name> # Domain name to␣
↪OpenShift console
export PORTAL_DOMAIN_NAME=<portal domain name> # Customer domain name for the EDB␣
↪Postgres AI Platform Portal
export TRANSPORTER_RW_SERVICE_DOMAIN_NAME=<transporter rw service domain name> #␣
↪Customer domain name for the EDB Postgres AI Platform Transporter RW Service
```

Important

PORTAL_DOMAIN_NAME and *TRANSPORTER_RW_SERVICE_DOMAIN_NAME* must be:

```
- distinct from each other—do not use the same name for both - distinct from␣
↪`OPENSHIFT_CONSOLE_DOMAIN_NAME` - formatted as `<app-name>.<cluster-domain>`
```

```
export BEACON_SERVICE_DOMAIN_NAME=<beacon service domain name> # Customer domain name␣
↪for the EDB Postgres AI Beacon Service
export CERT_MANAGER_NAMESPACE=<cert-manager namespace> # Customer namespace in which␣
↪cert-manager is installed as pre-requirements
```

Set storage class variable

The HM cluster requires a predefined storage class:

```
export STORAGE_CLASS=<storage class> # Customer storage class on OpenShift Container␣
↪Platform
```

Set authentication variables

Some authentication values are needed for installing HM:

```
export AUTHENTICATION_EMAIL=<email> # The email address of the user.
export AUTHENTICATION_HASH=<password hash>  # The hashed password of the user obtained␣
↪for example as: echo "mypassword" | htpasswd -BinC 10 admin | cut -d: -f
export AUTHENTICATION_USERNAME=<Owner MyCompany> # The username of the user
export AUTHENTICATION_USER_ID="c5998173-a605-449a-a9a5-4a9c33e26df7" # A unique␣
↪identifier for the user must not be changed since the first time installation.
```

Optional: set FIPs mode Transporter Data Operator

Optionally, you can set FIPS mode for Transporter Data Operator:

```
export TRANSPORTER_FIPS_ENABLED=true # Enable FIPS mode for Transporter Data Operator
```

Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator

Install Operator dependencies

EDB Postgres AI Operator has an explicit dependency cert-manager operator for Red Hat OpenShift (RHOS), which manages all namespaces. It's installed automatically before EDB Postgres AI Platform Operator.

The operator for Red Hat also has implicit dependencies including:

- RHOS Service Mesh 2, which manages all namespaces
- Node Feature Disovery Operator, which manages the namespace `openshift-nfd`
- NVIDIA GPU Operator, which manages the namespace `nvidia-gpu-operator`

If you've already installed these operators, you don't need to reinstall them.

Install the operator for RHOS

1. To install EDB Postgres AI Operator on RHOS, first ensure you've prepared the image pull secret for accessing the EDB Cloudsmith repository. Also verify that you have the necessary permissions to pull images from the repository `docker.enterprisedb.com/pgai-platform/edb-hcp-operator/manager`.

```
export DOCKER_SERVER="docker.enterprisedb.com"
export DOCKER_USERNAME="<your-docker-user-name>"
export DOCKER_PASSWORD="<your-docker-password>"
```

1. Create a new OC project for EDB Postgres AI Operator:

```
oc new-project edb-hcp-operator-system
```

1. Set up the image pull secret to access the EDB Cloudsmith repository:

```
oc create secret docker-registry edb-hcp-operator-pull-secret \
  --docker-server="$DOCKER_SERVER" \
  --docker-username="$DOCKER_USERNAME" \
  --docker-password="$DOCKER_PASSWORD" \
  --namespace-edb-hcp-operator-system
```

1. Use OpenShift Container Platform (OCP) to install EDB Postgres AI Operator for Red Hat OpenShift:
2. Log in to the OCP web console.
3. Select **Operators > OperatorHub**.
4. In the filter box, enter `EDB Postgres AI`.
5. Select **EDB Postgres AI Operator** and select **Install**.
6. On the Install Operator page: a. For the operator, select **Installed Namespace** using the `edb-hcp-operator-system` name you created earlier. b. Select **Install**.

> **Tip**
>
> On RHOS, cert-manager is a required dependency for Hybrid Manager. When installing cert-manager, so that it works correctly, ensure that **Installation Mode** is configured as **All namespaces on the cluster**.

Install Hybrid Manager on RHOS

After EDB Postgres AI Operator is installed on your RHOS cluster, you need to use the Operator to install Hybrid Manager. The following YAML file initiates the Hybrid Manager installation:

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  source:
    useLocalKustomizations: true
  flavour: rhos
  version: v1.2.0+appl
  imageRegistry: ${CONTAINER_REGISTRY_URI}
  imagePullSecrets:
    - name: edb-cred
      namespace: edbpgai-bootstrap
  globalParameters:
    internal_backup_folder: 4befab698c8d
    portal_domain_name: ${PORTAL_DOMAIN_NAME}
    default_ingress_domain: ${DEFAULT_INGRESS_DOMAIN}
    storage_class: ${STORAGE_CLASS}
  componentsParameters:
    edb-migration-portal:
      enable_pdb: "false"
    transporter-data-operator:
      fips_enabled: "${TRANSPORTER_FIPS_ENABLED}"
    transporter-dp-agent:
      rw_service_url: https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter
    transporter-rw-service:
      domain_name: ${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}
    upm-api-pnxt-metric:
      log_level: info
      tsdb_host: thanos-query.monitoring.svc.cluster.local
      tsdb_port: "9090"
    upm-api-pnxt-pd:
      log_level: info
    upm-beaco-ff-base:
      usage_generator_interval: 24h
    upm-beaco-usages:
      usage_generator_interval: 24h
    upm-beacon:
      server_host: ${BEACON_SERVICE_DOMAIN_NAME}
      beacon_server_replicas: "1"
      external_metrics_replicas: "1"
      beacon_db_hibernation: "\"off\""
```

(continues on next page)

(continued from previous page)

```
    upm-istio-gateway:
      openshift_console_domain_name: ${OPENSHIFT_CONSOLE_DOMAIN_NAME}
      cookie_aeskey: ${AES_256_KEY}
      ingress_http_node_port: "32542"
      ingress_https_node_port: "30288"
      ingress_grpc_tls_node_port: "30290"
    upm-prometheus:
      memory: 4Gi
    upm-trust-manager:
      cert_manager_namespace: ${CERT_MANAGER_NAMESPACE}
  pgai:
    portal:
      authentication:
        clientSecret: edb-postgres-ai-secret
        idpConnectors: []
        staticPasswords:
          - email: ${AUTHENTICATION_EMAIL}
            hash: ${AUTHENTICATION_HASH}
            username: ${AUTHENTICATION_USERNAME}
            userID: ${AUTHENTICATION_USER_ID}
  beaconAgent:
    applianceProviderEnabled: true
    plaintext: true
    beaconServerAddress: "upm-beacon-server-ingestion.upm-beacon.svc.cluster.local:8080"
    tlsInsecure: false
    location: "devspatcher"
    logLevel: "debug"
    beaconDBConnected: true
    inCluster: true
    provisioning:
      loadBalancersEnabled: true
      nodePortDomain: "nodeportdomain.com"
      imageDiscovery: true
      imagesetDiscoveryAuthenticationType: "basic"
      imagesetDiscoveryContainerRegistryURL: "us-east1-docker.pkg.dev"
    providers:
      - appliance
```

After the YAML has been applied, you can use the following command to monitor the installation status:

```
oc get hybridcontrolplane edbpgai -o yaml
```

Three different statuses can be returned, depending on the phase the installation is currently in:

```
- deployed # the installation was successful

- deploying # the installation is still in progress

- failed: # the install was unsuccessful
```

After the install is successful, *updating DNS zone records and verify the install* **.

Uninstalling Hybrid Manager installed via Operator

> **Warning**
>
> The following procedure uninstalls Hybrid Manager. Use this procedure only if you're sure you want to uninstall Hybrid Manager.

There may be cases where you want to delete Hybrid Manager. Because Hybrid Manager is protected from deletion by default, you must first disable that protection:

```
kubectl annotate hybridcontrolplane edbpgai biganimal.enterprisedb.io/deletion-
↪protect=disabled --overwrite
```

You can then uninstall:

```
kubectl delete hybridcontrolplane edbpgai --wait=false
```

To monitor the process, run:

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

Connecting to Hybrid Manager

To verify that your installation of the EDB Postgres AI Hybrid Manager platform was successful, connect to Hybrid Manager Console using your browser.

```
https://portal.foo.network/
```

In this configuration, the https doesn't validate, as the hostname doesn't exist on the wider internet. Accept the certificate warning to proceed to the portal. Accept the certificate warning to proceed to the portal.

At the login screen, you can enter the username and password generated with the bootstrap installation. See *Managing users in Hybrid Manager* for more information.

After you log in, you can *Create a database cluster template* .

Updating DNS zone records

After you've completed the Helm or Operator install, update your DNS zone records and verify that Hybrid Manager is working as expected.

Update DNS zone records

1. Retrieve the hostname of the portal service. This hostname represents the ingress endpoint for your Hybrid Manager.

```
kubectl get svc -n istio-system istio-ingressgateway -o yaml | yq .status.loadBalancer.
↪ingress[0].hostname
```

1. Create CNAME records in your DNS zone for the following domain names and point them to the retrieved hostname.

- `PORTAL_DOMAIN_NAME`
- `TRANSPORTER_RW_SERVICE_DOMAIN_NAME`

- BEACON_SERVICE_DOMAIN_NAME

Each CNAME record maps a user-facing domain name to the ingress hostname.

Now you can *connect to the portal* and verify the installation.

4.3.3 Install Hybrid Manager on AWS EKS

You can install the Hybrid Manager (HM) on Amazon EKS using either a Helm chart to bootstrap or with the Hybrid Manager Operator.

- *Prerequisites*
- *Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator*
- *Uninstalling using the Hybrid Manager Operator*
- *Installing using the Helm chart*
- *Connecting to the portal*
- *Uninstalling Hybrid Manager*

EKS installation prerequisites

AWS EKS has prerequisites to meet before you can install the EDB Postgres AI Software Deployment. Fulfill the prerequisites to ensure that you have everything you need before you start the installation process.

Prerequisites

- *A system from which you can run the installation process*
- *An AWS account with the necessary permissions*
- *An EKS cluster configured to host HM*

Next steps

After you complete the prerequisites, you can move on to *installing the EDB Postgres AI Software Deployment on AWS EKS* .

Install system

You need a system from which you can run the installation process. This system needs the following software installed:

- `AWS CLI <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html#getting-started-install-instructions>`_ — The AWS Command Line Interface (CLI) to manage your AWS services.
- `kubectl <https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html>`_ to manage your Kubernetes cluster.
- `yq <https://github.com/mikefarah/yq>`_ to prepare the HM installation files.
- `Helm <https://helm.sh/docs/intro/install/>`_ to render and apply the Kubernetes manifest.

You can install these tools using the linked instructions or by using the instructions for your particular operating system that follow.

macOS

Install the AWS CLI, kubectl, yq, and Helm:

```
brew install awscli kubectl yq helm
```

Linux

> **Note**
>
> These instructions are for Ubuntu 16.04 or newer, as they assume Snap support is available by default.

```
sudo snap install aws-cli --classic
sudo snap install helm --classic
sudo snap install kubectl --classic
sudo snap install yq
```

Install the AWS CLI:

```
sudo yum install awscli
```

Install kubectl:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/
↪bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
kubectl version --client
```

Install Helm:

```
curl -LO https://get.helm.sh/helm-v3.12.3-linux-amd64.tar.gz
tar -xvf helm-v3.12.3-linux-amd64.tar.gz
sudo mv linux-amd64/helm /usr/local/bin/helm
helm version
```

Install yq:

```
sudo curl -L "https://github.com/mikefarah/yq/releases/latest/download/yq_linux_amd64" -
↪o /usr/bin/yq
sudo chmod +x /usr/bin/yq
yq --version
```

```
sudo apt update
sudo apt install snapd
sudo snap install snapd
sudo snap install aws-cli --classic
sudo snap install kubectl --classic
sudo snap install yq
```

Windows

```
winget install Amazon.awscli --source winget
winget install kubectl --source winget
winget search helm
winget install helm
winget install yq --source winget
```

1. Install the AWS CLI .
2. Install kubectl .
3. Install yq .
4. Install *Apply the Helm chart* .

Installing Helm

1. Download the Helm binary for Windows. On the Helm GitHub Releases page , locate the latest release and download the helm-v3.x.x-windows-amd64.zip file.
2. Extract the Zip file. Use a tool like 7-Zip, WinRAR, or the built-in Windows extraction utility to unzip the downloaded file.
3. Copy `helm.exe` from the extracted folder to a directory in your path. Common directories include `C:\Program Files\` , `C:\Program Files (x86)\` , or you can create a directory, such as `C:\Helm\` , and add it to your path.
4. Verify the installation:
5. Open a new Command Prompt or PowerShell window.
6. Run the following command to verify that Helm is installed correctly:

````` bash helm version

```

``````

3. The output displays Helm version information.

Next steps

Once you have installed the necessary software, you can move on to *setting up an AWS account with the necessary permissions* .

Setting up an AWS account

To install a Hybrid Manager cluster, you need an AWS account with privileges to create the necessary resources.

Required permissions

To create an EKS cluster, you need an AWS account with the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

(continues on next page)
```

(continued from previous page)

```
            "Effect": "Allow",
            "Action": [
                "eks:*",
                "sts:AssumeRole",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:PutRolePolicy",
                "iam:PassRole",
                "iam:ListPolicies",
                "iam:ListRolePolicies",
                "iam:ListAttachedRolePolicies",
                "iam:ListRoleTags",
                "iam:TagRole",
                "iam:GenerateServiceLastAccessedDetails",
                "iam:ListInstanceProfilesForRole",
                "iam:ListOpenIDConnectProviders",
                "iam:GetOpenIDConnectProvider",
                "iam:CreateOpenIDConnectProvider",
                "iam:TagOpenIDConnectProvider",
                "iam:GetServiceLastAccessedDetails",
                "iam:AttachRolePolicy",
                "ec2:TerminateInstances",
                "iam:deleteOpenIDConnectProvider",
                "iam:detachRolePolicy",
                "iam:DeleteRolePolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

We recommend that you create a role with the above permissions and add it to your existing permissions by attaching it to your user. We also recommend that you review these permissions with your security team to ensure they're appropriate for your use case.

Create an AWS account

If you don't already have an AWS account, you can create one on the AWS website . Select **Create an AWS Account** and follow the instructions.

Create an IAM user

1. Log in to the AWS Management Console .
2. Navigate to the IAM service by searching for **IAM** in the search bar.
3. In the IAM dashboard, in the left menu, select **Users**, then select **Add user**.
4. Enter a username for the new user and select the **Programmatic access** check box.
5. Select **Next: Permissions**.

6. Attach a policy. In the **Set permissions** step, select **Attach existing policies directly** and search for the AdministratorAccess policy. Select the policy and select **Next: Tags**.
7. If you want, add tags to the user and select **Next: Review**.
8. Review the user details. To create the IAM user, select **Create user**.

> **Note**
>
> Record the access key ID and secret access key. You will need them to configure the AWS CLI.

Configure the AWS CLI

To configure the AWS CLI, you first need the AWS CLI installed on your local machine. (You perform this step as part of the *Install system* prerequisite).

To configure the CLI, run:

```
aws configure
```

Next steps

After you configure the AWS CLI, you can *create an EKS cluster* .

Creating an EKS cluster

After you have *installed your system* and *configured your AWS account* , you can create an EKS cluster using the AWS Management Console.

Step 1: Log in to the AWS Management Console

Log in with your account credentials at https://aws.amazon.com/console/ to get to the management console.

Enter `EKS` in the search bar and select **Elastic Kubernetes Service** from the search results.

Step 2: Obtain the bootstrap kit

You install Hybrid Manager (HM) using a bootstrap kit. This kit is a collection of scripts and Helm charts that are used to install the EDB Software Deployment platform.

You need the following files:

- *eks-fix-auto.sh*

Copy the file to a directory on your local machine and `cd` into that directory.

Then, using the AWS Management Console:

1. Select the AWS region you want to create the cluster in.
2. Make a note of the region you select. You will need this later in the process.
3. Go to the EKS page. Enter `EKS` in the search bar and select **Elastic Kubernetes Service** from the search results.
4. On the EKS page, select **Create Cluster**.

Step 3: Configure the cluster

In the configuration options:

- Select **Quick configuration (with EKS Auto Mode) - new**.
- In the **Cluster name** field, enter a name for your cluster and make a note of it.
- In the **Kubernetes version** field, select the version you want to use. (The default is 1.31 for EKS Auto Mode.)
- In the **Cluster IAM Role** field, leave the value `AmazonEKSAutoClusterRole` .
- In the **Node IAM Role** field, leave the value `AmazonEKSAutoNodeRole` .

Select or create a VPC for your cluster

You need to provide a VPC for your cluster. You can either create a VPC or use an existing one.

1. Select **Create VPC**.
2. On the **Create VPC** page, under **Resources to create**, select **VPC and more**.
3. Enter the following information:

- Under **Name tag auto-generation**, enter a name for your VPC.
- Under **Number of Availability Zones**, select 3.
- Under **Number of public subnets**, ensure that 3 is selected.
- Under **Number of private subnets**, ensure that 3 is selected.
- Under NAT gateways, select **In 1 AZ**

4. Select **Create VPC**. The **Create VPC workflow** page appears and steps through creating the VPC. When complete, select **View VPC**.

In the view of the VPC, for each public subnet in the resource map:

1. Hover over the subnet and select the square box with arrow icon that appears. A page opens where you can edit the subnet.
2. Select **Actions > Edit subnet settings**.
3. Select **Auto-assign public IPv4 address > Enable auto-assign public IPv4 address**.
4. Select **Save**.

Add tags for each public subnet:

1. Select the **Tags** tab.
2. Select **Manage Tags**
3. Select **Add Tag**
4. Enter `kubernetes.io/role/elb` in the **Key** field, and `1` in the **Value** field.
5. Select **Save**
6. Close the tab/window and return to the VPC view.

Add tags for each private subnet in the resource map:

1. Hover over the subnet and select the square box with arrow icon that appears.
2. Select the **Tags** tab.
3. Select **Manage Tags**

4. Select **Add Tag**
5. Enter `kubernetes.io/role/internal-elb` in the **Key** field, and `1` in the **Value** field.
6. Select **Save**
7. Close the tab/window and return to the VPC view.

If you have an existing VPC you want to use, select it from the dropdown.

The VPC must have at least two public subnets and two private subnets in different availability zones and must have the following tags:

Public subnets:

- `kubernetes.io/role/elb` : `1`

Private subnets:

- `kubernetes.io/role/internal-elb` : `1`

Return to the EKS cluster creation page

If you created a new VPC, you need to refresh the EKS cluster creation page. To refresh the dropdown so you can select the VPC you just created, select the circular arrow icon next to **Create VPC**.

1. Select the subnets for your cluster. You must select at least two private subnets in different availability zones.

> **Note**
>
> These subnets must be private, not public, as their purpose is to facilitate communication between the Kubernetes control plane and the Kubernetes worker nodes. Public subnets could leave the cluster internals open to unnecessary security threats. The subnet tagging informs EKS to create the ingress load balancers on the correct subnets.

2. Select **Create**.

The cluster is now being created. This can take a few minutes. You can monitor the creation on the EKS console by selecting **Clusters** in the breadcrumb at the top of the page.

Step 4: Configure kubectl

After you create the cluster, you need to configure kubectl to interact with it.

Open your terminal and run:

```
aws eks --region <REGION> update-kubeconfig --name <CLUSTER-NAME>
```

Where:

- `<REGION>` is the region in which you created the cluster.
- `<CLUSTER-NAME>` is the name you gave the cluster.

> **Note**
>
> If you haven't set the *AWS_PROFILE* environment variable, include the *--profile <profile-name>* option in the command.

You can now interact with your EKS cluster. To test, run:

```
kubectl get pods -A
```

Step 5: Create an S3 bucket and connect it to the cluster

HM requires an S3 bucket to store backups and other data. Create an S3 bucket and connect it to the cluster.

Create an S3 bucket

1. Open the S3 service in the AWS Management Console. Enter `S3` in the search bar and select **S3** from the search results.
2. Select **Create bucket**.
3. Enter a name for the bucket name field. This must be globally unique.
4. Select **Create Bucket**.

Create the role for access

1. Return to the EKS dashboard. Enter `EKS` in the search bar and select **Elastic Kubernetes Service** from the search results.
2. Select the cluster you created.
3. In the **Details** section of the overview, copy the OpenID Connect provider URL.
4. Open the IAM service in the AWS Management Console. enter `IAM` in the search bar and select **IAM** from the search results.
5. In the left menu, select **Identity Providers**.
6. Select **Add Provider**.
7. On the **Configure Provider** page, select **OpenID Connect**, then paste the URL you copied into the **Provider URL** field.
8. In the audience field, enter `sts.amazonaws.com` .
9. Select **Add Provider**.

A green banner displays the message `Provider created successfully` and an **Assign Role** button appears.

10. Select **Assign Role**.
11. Under **Role Options**, select **Create a New Role**. Select **Next**.

Select a trusted entity:

1. On the **Select trusted entity** page, select **Web identity**.

In the **Web identity** section, the **Identity Provider** field shows the provider you just created.

2. Select `sts.amazonaws.com` in the **Audience** field and select **Next**.

Grant permissions to the new role:

1. On the **Add Permissions** page, enter `s3full` to the search field. Select the `AmazonS3FullAccess` policy and select **Next**.
2. Enter a role name, such as `<clustername>-s3access` , and select **Create Role**.

A green banner displays with the message `Role created successfully` and a **View Role** button appears.

3. Select **View Role**.

4. In the view of the role, copy the role's **ARN** value.

Create the secret

You must create the secret in the default namespace (`-n default`) for it to function correctly.

In your terminal, run:

```
kubectl create secret generic -n default edb-object-storage \
    --from-literal auth_type=workloadIdentity \
    --from-literal aws_region=<REGION> \
    --from-literal aws_role_arn=<ROLE-ARN> \
    --from-literal bucket_name=<BUCKET-NAME>
```

Where:

- <REGION> is the region where you are creating the cluster.
- <ROLE-ARN> is the ARN of the role you created.
- <BUCKET-NAME> is the name of the bucket you created.

Step 6: Add the EBS CSI driver and CSI snapshot controller

The PGAIHM requires two AWS CSI-related drivers to manage snapshots correctly: the EBS CSI driver and CSI snapshot controller.

AWS EBS CSI driver

The EBS CSI diver allows Kubernetes clusters to dynamically provision and manage storage resources using AWS services like Elastic Block Store (EBS) and Elastic File System (EFS). The integration helps Kubernetes workloads seamlessly use AWS storage services as persistent storage.

To install the EBS CSI driver:

1. In the EKS cluster page of the EKS cluster you're installing PGAIHM on, select the **Add-ons** tab.
2. Select **Get More Addons**.
3. On the **Add-ons** page, on the tile for EBS CSI Driver, select the check box, then select **Next**.
4. On the **Configure** page, select **Next**.
5. On the **Review** page, select **Create**.

The EBS CSI add-on driver is added to your cluster.

AWS CSI snapshot controller

The CSI snapshot controller is a Kubernetes controller that manages volume snapshots.

To install the CSI snapshot controller:

1. In the EKS cluster page of the EKS cluster you are installing PGAIHM on, select the **Add-ons** tab.
2. Select **Get More Addons**.
3. On the **Add-ons** page, on the tile for CSI Snapshot Controller, select the check box, then select **Next**.
4. On the **Configure** page, select **Next**
5. On the **Review** page, select **Create**.

The CSI snapshot controller add-on is added to your cluster.

Step 7: Update the cluster for compatibility

The EKS cluster created by the AWS Management Console isn't fully compatible with HM, so you need to make some changes to the cluster. A script is available to make those changes for you.

Download the *Helm eks-fix-auto.sh* script or *Operator eks-fix-auto.sh* script and run it in your terminal.

```
$SHELL eks-fix-auto.sh
```

The script replaces the default nodepool and storage classes with ones more appropriate for the workload and configures a few other settings.

Step 8: Install Hybrid Manager

Next, *Install the Hybrid Manager on EKS* .

Install Hybrid Manager on EKS

You can install the Hybrid Manager (HM) on Amazon EKS using either a Helm chart to bootstrap or with the Hybrid Manager Operator.

You need to meet the *Prerequisites* before you can install Hybrid Manager on EKS.

You can then install Hybrid Manager using one of the following methods:

- *Installing Hybrid Manager on AWS EKS using the Hybrid Manager Operator*
- *Installing Hybrid Manager on AWS EKS*

After installing Hybrid Manager, you can connect to the EDB Postgres AI Hybrid Manger Portal using the *Connecting to the Hybrid Manager Console* guide.

Installing Hybrid Manager on AWS EKS using the Hybrid Manager Operator

You can install the Hybrid Manager on Amazon EKS using the Hybrid Manager Operator. This is a Helm chart that deploys the Hybrid Manager platform on your EKS cluster. The operator is responsible for managing the lifecycle of the Hybrid Manager platform, including installing, upgrading, and uninstalling the platform.

Important

Complete the *Prerequisites* before you start this part of the installation process. It will step through the process of creating an EKS cluster, installing the necessary components, and configuring the environment ready for the Hybrid Manager Operator.

Set the version to install

The version of the EDB Hybrid Manager platform is set by the environment variable EDB_PLATFORM_VERSION. This should be set to the version of the platform you wish to install. For example, to install version v1.2.0, you would set the environment variable as follows:

```
export EDB_PLATFORM_VERSION="v1.2.0"
```

Set your token

This installation process requires that you save your EDB subscription token as an environment variable. You can obtain it by going to your EDB Account Profile . (Log in if prompted to.) There you will find an entry for Repos 2.0 token:

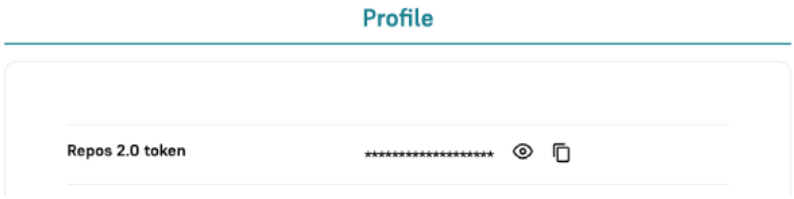


Fig. 1: Repo Token

Take this value and set it as an environment variable:

```
export EDB_SUBSCRIPTION_TOKEN="your-token"
```

Obtain the bootstrap kit

Hybrid Manager is installed using a bootstrap kit. This kit is a collection of scripts and charts that are used to install the EDB Hybrid Manager platform.

- eks-install-operator.sh - This script installs the Hybrid Manager Operator on your EKS cluster.

```
registry="docker.enterprisedb.com"
username="pgai-platform"
password=""

echo "Enter the password for ${username}@${registry}"
read -s password

if [ -z "$password" ]; then
    echo 1>&2 "Error: password is required" >&2
    echo 1>&2
    usage
    exit 1
fi

kubectl create namespace edb-hcp-operator-system

helm repo add enterprisedb-edbpgai "https://downloads.enterprisedb.com/${password}/$
↪{RELEASE}pgai-platform/helm/charts/"
helm repo update

kubectl create secret docker-registry edb-hcp-operator-pull-secret \
    --docker-server="${registry}" \
    --docker-username="${RELEASE}${username}" \
    --docker-password="${password}" \
    --namespace edb-hcp-operator-system

helm upgrade \
    -n edb-hcp-operator-system \
```

(continues on next page)

(continued from previous page)

```
    --install \
    --set "imagePullSecrets[0].name=edb-hcp-operator-pull-secret" \
    --set controllerManager.manager.image.tag=$OPERATOR_VERSION \
    --set controllerManager.manager.image.repository=docker.enterprisedb.com/${RELEASE}
↪pgai-platform/edb-hcp-operator/manager \
    --version $OPERATOR_VERSION \
    edb-hcp-operator enterprisedb-edbpgai/edb-hcp-operator
```

- eks-install-secrets.sh - This script installs the secrets required for the installation process.

```
registry="docker.enterprisedb.com"
username="pgai-platform"
password=""

echo "Enter the password for ${username}@${registry}"
read -s password

if [ -z "$password" ]; then
    echo 1>&2 "Error: password is required" >&2
    echo 1>&2
    usage
    exit 1
fi

kubectl create ns edbpgai-bootstrap
kubectl create ns upm-replicator

kubectl create secret docker-registry edb-cred \
    -n edbpgai-bootstrap \
    --docker-server="${registry}" \
    --docker-username="${username}" \
    --docker-password="${password}"

kubectl create secret docker-registry edb-cred \
    -n upm-replicator \
    --docker-server="${registry}" \
    --docker-username="${username}" \
    --docker-password="${password}"

kubectl annotate secret -n upm-replicator edb-cred replicator.v1.mittwald.de/replicate-
↪to="*" --overwrite

kubectl create namespace upm-griptape

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

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

kubectl create namespace upm-lakekeeper

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

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

echo "PG_CONFOUNDING_KEY is ${PG_CONFOUNDING_KEY} - store safely"
```

- prepare-op.sh - This script prepares the values file for the installation process.

```
# !/bin/bash
echo "HM Operator values.yaml preparation\n"

#  Import the eks-default-env file

source ./default-env.sh

#  If EDB_PLATFORM_VERSION is not set, error and exit

if [ -z "$EDB_PLATFORM_VERSION" ]; then
  echo "EDB_PLATFORM_VERSION is not set, please set it to the version of EDB Platform␣
↪you are deploying"
  exit
fi

#  Test for the presence of yq

if ! command -v yq &> /dev/null
then
    echo "yq could not be found, please install it"
    exit
fi

cp hm-default.yaml hm-values.yaml

#  Prepare values.yaml for EKS

export AES_256_KEY=$(openssl rand -base64 32)

#  if AUTHENTICATION_PASSWORD_HASH is not set, generate it
```

(continues on next page)

(continued from previous page)

```
if [ -z "$AUTHENTICATION_PASSWORD_HASH" ]; then
  echo "Generating password hash - please ensure you remove the password from your␣
↪history"
  export AUTHENTICATION_PASSWORD_HASH=$(echo -n $AUTHENTICATION_PASSWORD | htpasswd -
↪BinC 10 admin | cut -d: -f2)
else
  echo "Using prehashed password"
fi

if [[ "$EDB_PLATFORM_VERSION" == v1.2.* ]]; then
    echo "EDB Platform version is v1.2.*"
    yq -i .spec.imageRegistry = env(CONTAINER_REGISTRY_URI) |
    .spec.componentsParameters.upm-istio-gateway.cookie_aeskey = env(AES_256_KEY) |
    .spec.globalParameters.portal_domain_name = env(PORTAL_DOMAIN_NAME) |
    .spec.globalParameters.storage_class = env(STORAGE_CLASS) |
    .spec.componentsParameters.transporter-data-operator.fips_enabled = (env(TRANSPORTER_
↪FIPS_ENABLED)|tostring) |
    .spec.componentsParameters.transporter-rw-service.domain_name = env(TRANSPORTER_RW_
↪SERVICE_DOMAIN_NAME) |
    .spec.componentsParameters.transporter-dp-agent.rw_service_url= "https://
↪"+env(TRANSPORTER_RW_SERVICE_DOMAIN_NAME)+"/transporter" |
    .spec.componentsParameters.upm-beacon.server_host= env(BEACON_SERVICE_DOMAIN_NAME) |
    .spec.pgai.portal.authentication.staticPasswords[0].email=env(AUTHENTICATION_EMAIL) |
    .spec.pgai.portal.authentication.staticPasswords[0].hash=env(AUTHENTICATION_PASSWORD_
↪HASH) |
    .spec.pgai.portal.authentication.staticPasswords[0].username=env(AUTHENTICATION_
↪USER) |
    .spec.pgai.portal.authentication.staticPasswords[0].userID="c5998173-a605-449a-a9a5-
↪4a9c33e26df" |
    .spec.beaconAgent.provisioning.imagesetDiscoveryContainerRegistryURL= env(IMAGESET_
↪REGISTRY_URI) |
    .spec.beaconAgent.provisioning.imagesetDiscoveryAuthenticationType= env(IMAGESET_
↪AUTHTYPE) |
    .spec.beaconAgent.location=env(LOCATION_NAME) hm-values.yaml
fi

echo "Your hm-values.yaml file is ready"
```

- default-env.sh - This file contains the default environment variables for the installation process.

```
#  exports for EKS

export EDB_PLATFORM_VERSION="v1.2.0"
export CONTAINER_REGISTRY_URI="docker.enterprisedb.com/pgai-platform"
export IMAGESET_REGISTRY_URI=$CONTAINER_REGISTRY_URI
export IMAGESET_AUTHTYPE="token"
export PORTAL_DOMAIN_NAME="portal.foo.network"
export TRANSPORTER_RW_SERVICE_DOMAIN_NAME="transporter.foo.network"
export BEACON_SERVICE_DOMAIN_NAME="beacon.foo.network"
export AUTHENTICATION_EMAIL="owner@mycompany.com"
export AUTHENTICATION_USER="owner@mycompany.com"
```

(continues on next page)

(continued from previous page)

```
export LOCATION_NAME="default-location"
export STORAGE_CLASS="gp2"
export TRANSPORTER_FIPS_ENABLED=false

#  Set a password hash for the user or pass a password to have it hashed for you.

#  If you pass a password, you will need to remove it from your history.

#  You can hash a password using the following command:

#  echo -n "password" | htpasswd -BinC 10 admin | cut -d: -f2

#

#  export AUTHENTICATION_PASSWORD="password"

#  If setting AUTHENTICATION_PASSWORD_HASH, ensure it is single quoted to prevent␣
↪variable expansion (e.g. $2y$10$...).

export AUTHENTICATION_PASSWORD_HASH=$2y$10$vKOAXfLHbeV1OQxMpxlLdOIwnX.JAN.ZrD9ZU//
↪ocrNQwhQIMtXhy
```

- hm-default.yaml - This file contains the default values for a configuration.

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  source:
    useLocalKustomizations: true
  flavour: eks
  version: v1.2.0+appl
  imageRegistry: ${CONTAINER_REGISTRY_URI}
  imagePullSecrets:
    - name: edb-cred
      namespace: edbpgai-bootstrap
  globalParameters:
    internal_backup_folder: 4befab698c8d
    portal_domain_name: ${PORTAL_DOMAIN_NAME}
    storage_class: ${STORAGE_CLASS}
  componentsParameters:
    edb-migration-portal:
      enable_pdb: "false"
    transporter-data-operator:
      fips_enabled: "${TRANSPORTER_FIPS_ENABLED}"
    transporter-dp-agent:
      rw_service_url: https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter
    transporter-rw-service:
      domain_name: ${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}
    upm-api-pnxt-metric:
      log_level: info
```

(continues on next page)

(continued from previous page)

```
    tsdb_host: thanos-query.monitoring.svc.cluster.local
    tsdb_port: "9090"
  upm-api-pnxt-pd:
    log_level: info
  upm-beaco-ff-base:
    usage_generator_interval: 24h
  upm-beaco-usages:
    usage_generator_interval: 24h
  upm-beacon:
    server_host: ${BEACON_SERVICE_DOMAIN_NAME}
    beacon_server_replicas: "1"
    external_metrics_replicas: "1"
    beacon_db_hibernation: "\"off\""
  upm-istio-gateway:
    cookie_aeskey: ${AES_256_KEY}
  upm-prometheus:
    memory: 4Gi
pgai:
  portal:
    authentication:
      clientSecret: edb-postgres-ai-secret
      idpConnectors: []
      staticPasswords:
        - email: ${AUTHENTICATION_EMAIL}
          hash: ${AUTHENTICATION_HASH}
          username: ${AUTHENTICATION_USERNAME}
          userID: ${AUTHENTICATION_USER_ID}
beaconAgent:
  applianceProviderEnabled: true
  plaintext: true
  beaconServerAddress: "upm-beacon-server-ingestion.upm-beacon.svc.cluster.local:8080"
  tlsInsecure: false
  location: "devspatcher"
  logLevel: "debug"
  beaconDBConnected: true
  inCluster: true
  provisioning:
    loadBalancersEnabled: true
    nodePortDomain: "nodeportdomain.com"
    imageDiscovery: true
    imagesetDiscoveryAuthenticationType: "eks_managed_identity"
    imagesetDiscoveryContainerRegistryURL: 123456079902.dkr.ecr.us-east-1.amazonaws.com
  providers:
    - appliance
```

Copy all files to a directory on your local machine and `cd` into that directory.

Install the Hybrid Manager Operator

To install the Hybrid Manager Operator, run the `eks-install-operator.sh` script. (As you already set the token as an environment variable, you can pipe it into the script). You will also need to set the environment variable `OPERATOR_VERSION` to the version of the operator you wish to install, for example `v1.2.0` . This will typically be the same version as the EDB Hybrid Manager platform you are installing, so you can set it as follows:

```
export OPERATOR_VERSION=$EDB_PLATFORM_VERSION
```

Then you can run the script to install the operator:

```
echo $EDB_SUBSCRIPTION_TOKEN | $SHELL eks-install-operator.sh
```

Install secrets

The install process requires a number of secrets to be installed in the Kubernetes cluster. These secrets are used to authenticate with the EDB Download Repository and to store the credentials for the EDB Software Deployment platform.

to install the secrets required for the bootstrap process, run the `install-secrets.sh` script. (As you already set the token as an environment variable, you can pipe it into the script.)

```
echo $EDB_SUBSCRIPTION_TOKEN | $SHELL eks-install-secrets.sh
```

```
Enter the password for pgai-platform@docker.enterprisedb.com
namespace/edbpgai-bootstrap created
namespace/upm-replicator created
secret/edb-cred created
secret/edb-cred created
secret/edb-cred annotated
namespace/upm-griptape created
secret/fernet-secret created
namespace/upm-lakekeeper created
secret/pg-confounding-key created
PG_CONFOUNDING_KEY is 01234567890123456789ABCDEFG - store safely
```

This process creates secrets needed for Hybrid Manager to run. You can find out more about these secrets in:

- ** *Enabling GenAI Builder on Hybrid Manager* (optional for GenAI Builder)**
- ** *Enabling Data Catalogs on Hybrid Manager* (optional for using Catalog)**
- See also ** *Customizing Migration Portal secrets for secure internal communication* (optional to secure internal communications with Migration Portal)**

Prepare the values file

The bootstrap process requires a number of values to be set in the Helm chart. These values are stored in a file called `values.yaml` . You can create this file by running the `prepare.sh` script, which uses variables from the `default-env.sh` file. Edit the `default-env.sh` file to set the values you require.

This is the default `default-env.sh` file:

```
#  exports for EKS

export EDB_PLATFORM_VERSION="v1.2.0"
```

(continues on next page)

(continued from previous page)

```
export CONTAINER_REGISTRY_URI="docker.enterprisedb.com/pgai-platform"
export IMAGESET_REGISTRY_URI=$CONTAINER_REGISTRY_URI
export IMAGESET_AUTHTYPE="token"
export PORTAL_DOMAIN_NAME="portal.foo.network"
export TRANSPORTER_RW_SERVICE_DOMAIN_NAME="transporter.foo.network"
export BEACON_SERVICE_DOMAIN_NAME="beacon.foo.network"
export AUTHENTICATION_EMAIL="owner@mycompany.com"
export AUTHENTICATION_USER="owner@mycompany.com"
export LOCATION_NAME="default-location"
export STORAGE_CLASS="gp2"
export TRANSPORTER_FIPS_ENABLED=false

#  Set a password hash for the user or pass a password to have it hashed for you.

#  If you pass a password, you will need to remove it from your history.

#  You can hash a password using the following command:

#  echo -n "password" | htpasswd -BinC 10 admin | cut -d: -f2

#

#  export AUTHENTICATION_PASSWORD="password"

#  If setting AUTHENTICATION_PASSWORD_HASH, ensure it is single quoted to prevent␣
↪variable expansion (e.g. $2y$10$...).

export AUTHENTICATION_PASSWORD_HASH=$2y$10$vKOAXfLHbeV1OQxMpxlLdOIwnX.JAN.ZrD9ZU//
↪ocrNQwhQIMtXhy
```

Here is a brief guide to the variables in the `default-env.sh` file.

Variable	Description
CONTAINER_REGISTRY_URI	The URI of the container registry to use. This is set to the EDB Download Repository.
IMAGESET_REGISTRY_URI	The URI of the image set registry. This is set to the EDB Download Repository.
IMAGESET_AUTHTYPE	The authentication type for the image set registry. Set to token as we are using a token to authenticate with the EDB Download Repository.
TRANSPORTER_RW_SERVICE_I	The domain name for the Transporter service.
PORTAL_DOMAIN_NAME	The domain name for the portal.
BEACON_SERVICE_DOMAIN	The domain name for the beacon service.
AUTHENTICATION_EMAIL	The email address of the user to create.
AUTHENTICATION_USER	The username of the user to create.
AUTHENTICATION_PASSWORD	The password of the user to create. This can be set as plain text or as a hash with AUTHENTICATION_PASSWORD_HASH.

continues on next page

Table 2 – continued from previous page

Variable	Description
AUTHENTICATION_PASSWORD_HASH	The hash of the password to create.
LOCATION_NAME	The location name for the beacon agent, also known as Agent.
STORAGE_CLASS	The storage class to use for the installation. This is set to gp2 for EBS volumes on AWS.
TRANSPORTER_FIPS_ENABLED	Whether to enable FIPS mode for the Transporter service.

Run the `prepare.sh` script to create the `values.yaml` file.

```
$SHELL prepare.sh
```

This create a `values.yaml` file with the values set in the `default-env.sh` file.

You must start with the `hcp- You need to create two values for the bootstrap process. These are the AES_256_KEY and the AUTHENTICATION_PASSWORD_HASH. You can create these values with the following commands:

```
export AES_256_KEY=$(openssl rand -base64 32)
```

If you have a password to hash, you can do so with the following command:

```
export AUTHENTICATION_PASSWORD_HASH=$(echo -n $AUTHENTICATION_PASSWORD | htpasswd -BinC␣
↪10 admin | cut -d: -f2)
```

You now need to edit `values.yaml` file and set the values required for your installation. Find the appropriate key in the `values.yaml` file and set the values as follows. Replace `$<name>` entries with the value you have set the environment variables of the same name to.

1. Set `spec.componentsParameters.upm-istio-gateway.cookie_aeskey` to $AES_256_KEY.
2. Set `spec.imageRegistry` to $CONTAINER_REGISTRY_URI.
3. Set `spec.globalParameters.portal_domain_name` to $PORTAL_DOMAIN_NAME.
4. Set `spec.componentsParameters.transporter-rw-service.domain_name` to $TRANSPORTER_RW_SERVICE_DOMAIN_NAME.
5. Set `spec.componentsParameters.transporter-dp-agent.rw_service_url` to `https://` followed by $TRANSPORTER_RW_SERVICE_DOMAIN_NAME followed by `/transporter` . eg, given the values above, this would be `https://transporter.foo.network/transporter` .
6. Set `spec.componentsParameters.upm-beacon.server_host` to $BEACON_SERVICE_DOMAIN_NAME.
7. Set `spec.pgai.portal.authentication.staticPasswords[0].email` to $AUTHENTICATION_EMAIL.
8. Set `spec.pgai.portal.authentication.staticPasswords[0].hash` to $AUTHENTICATION_PASSWORD_HASH.
9. Set `spec.pgai.portal.authentication.staticPasswords[0].username` to $AUTHENTICATION_USER.
10. Set `spec.pgai.portal.authentication.staticPasswords[0].userID` “c5998173-a605-449a-a9a5-4a9c33e26df” .
11. Set `spec.beaconAgent.location` to $LOCATION_NAME

12. Set `spec.globalParameters.storage_class` to $STORAGE_CLASS.
13. Set `spec.componentsParameters.transporter-data-operator.fips_enabled` to $TRANSPORTER_FIPS_ENABLED. (Ensure the boolean value is quoted as it is handled as a string.)

The `hm-values.yaml` file should now contain the values required for the installation process.

Install the Hybrid Manager platform

The next step is to apply the values file which you created in the previous step to the Hybrid Manager Operator. This will trigger the installation of the Hybrid Manager platform on your EKS cluster.

```
kubectl apply -f ./hm-values.yaml
```

You can check the status of the installation by running:

```
kubectl get hybridcontrolplanes.edbpgai.edb.com
```

When complete, the output should look similar to this:

```
NAME      PHASE
edbpgai   deployed
```

If not deploying, run:

```
kubectl get hybridcontrolplanes.edbpgai.edb.com -o yaml
```

For more information on the status of the installation, you can check the logs of the Hybrid Manager Operator:

Configure the IP address in /etc/hosts

Obtain the IP address of your ingress gateway to configure in your `/etc/hosts` file. Run:

```
kubectl -n istio-system get service istio-ingressgateway -o jsonpath={.status.
↪loadBalancer.ingress[0].hostname} | strings | nslookup
```

This command returns output like:

```
Server:         192.168.111.1
Address:        192.168.111.1#53

Non-authoritative answer:
Name:   k8s-istiosys-istioing-b3dc9b4f3c-81dc5c45b1d5f6d2.elb.us-east-1.amazonaws.com
Address: 52.70.180.102
```

At this point your have the IP address of the portal. You can add this IP address to your DNS service and map it to the domain name you set in the `default-env.sh` file, which is `portal.foo.network` in this example.

If you do not have a DNS service configured, consider adding the IP address to your local `/etc/hosts` file. This will allow you to access the Hybrid Manager portal using the domain name you set in the `default-env.sh` file.

You can then add IP address to your `/etc/hosts` file like this:

```
echo "52.70.180.102 portal.foo.network" | sudo tee -a /etc/hosts
```

Substitute the IP address with the value you obtained earlier.

Next steps

You can now verify the installation by *connecting to the HM Console* .

Installing Hybrid Manager on AWS EKS

You can install the Hybrid Manager (HM) on Amazon EKS .

Important

Completed the *Prerequisites* before you start this part of the installation process.

Set the version to install

The version of the EDB Software Deployment platform is set by the environment variable EDB_PLATFORM_VERSION. This should be set to the version of the platform you wish to install. For example, to install version v1.2.0, you would set the environment variable as follows:

```
export EDB_PLATFORM_VERSION="v1.2.0"
```

Set your token

This installation process requires that you save your EDB subscription token as an environment variable. You can obtain it by going to your EDB Account Profile . (Log in if prompted to.) There you will find an entry for Repos 2.0 token:

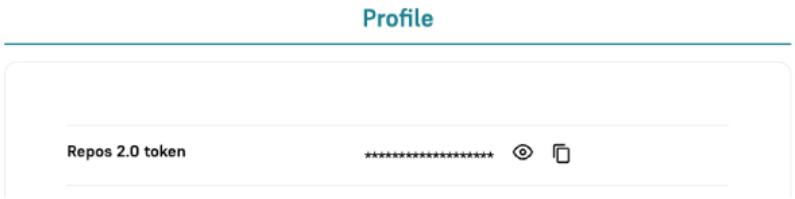


Fig. 2: Repo Token

Take this value and set it as an environment variable:

```
export EDB_SUBSCRIPTION_TOKEN="your-token"
```

Obtain the bootstrap kit

Hybrid Manager is installed using a bootstrap kit. This kit is a collection of scripts and Helm charts that are used to install the EDB Software Deployment platform.

- eks-install-secrets.sh

```
registry="docker.enterprisedb.com"
username="pgai-platform"
password=""

echo "Enter the password for ${username}@${registry}"
read -s password
```

(continues on next page)

(continued from previous page)

```
if [ -z "$password" ]; then
    echo 1>&2 "Error: password is required" >&2
    echo 1>&2
    usage
    exit 1
fi

kubectl create ns edbpgai-bootstrap
kubectl create ns upm-replicator

kubectl create secret docker-registry edb-cred \
    -n edbpgai-bootstrap \
    --docker-server="${registry}" \
    --docker-username="${username}" \
    --docker-password="${password}"

kubectl create secret docker-registry edb-cred \
    -n upm-replicator \
    --docker-server="${registry}" \
    --docker-username="${username}" \
    --docker-password="${password}"

kubectl annotate secret -n upm-replicator edb-cred replicator.v1.mittwald.de/replicate-
↪to="*" --overwrite

kubectl create namespace upm-griptape

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

kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
    name: fernet-secret
    namespace: upm-griptape
stringData:
    FERNET_KEY: ${FERNET_KEY}
EOF

kubectl create namespace upm-lakekeeper

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

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

(continues on next page)

(continued from previous page)

```
EOF

echo "PG_CONFOUNDING_KEY is ${PG_CONFOUNDING_KEY} - store safely"
```

- prepare.sh

```
# !/bin/bash
echo "HM Helm values.yaml preparation\n"

##  If EDB_PLATFORM_VERSION is not set, error and exit

if [ -z "$EDB_PLATFORM_VERSION" ]; then
  echo "EDB_PLATFORM_VERSION is not set, please set it to the version of EDB Platform␣
↪you are deploying"
  exit
fi

##  Test for the presence of yq

if ! command -v yq &> /dev/null
then
  echo "yq could not be found, please install it"
  exit
fi

##  Import the eks-default-env file

source ./default-env.sh

##  Test for presence of helm repo

value=`helm repo list -o yaml | yq eval contains([{ "name":"edbpgai"}]) -`
if [[ "$value" == "false" ]]; then
  echo "EDB Platform Helm repository not found, adding in now"
  helm repo add edbpgai "https://downloads.enterprisedb.com/${EDB_SUBSCRIPTION_TOKEN}/
↪pgai-platform/helm/charts/"
fi

##  Always update the helm repo

helm repo update
helm show values edbpgai/edbpgai-bootstrap --version $EDB_PLATFORM_VERSION > values.yaml
echo "\n\n"

##  Prepare version dependent values

export AES_256_KEY=$(openssl rand -base64 32)

if [[ "$EDB_PLATFORM_VERSION" == v1.2.* ]]; then
    echo "EDB Platform version is v1.2.*"
    yq -i .bootstrapImageName = env(CONTAINER_REGISTRY_URI)+"/edbpgai-bootstrap/
↪bootstrap-"+env(EDB_TARGET_PLATFORM) |
```

(continues on next page)

(continued from previous page)

```
    .bootstrapImageTag=env(EDB_PLATFORM_VERSION) |
    del(.bootstrapImage) |
    .parameters.upm-istio-gateway.cookie_aeskey = env(AES_256_KEY) |
    del(.parameters.upm-beacon-ff-base) |
    .beaconAgent.provisioning.imagesetDiscoveryContainerRegistryURL= env(IMAGESET_
↪REGISTRY_URI) |
    .beaconAgent.provisioning.imagesetDiscoveryContainerRegistryAuthType= env(IMAGESET_
↪AUTHTYPE) |
    .containerRegistryURL = env(CONTAINER_REGISTRY_URI) values.yaml
fi

##  Prepare values.yaml for EKS

##  if AUTHENTICATION_PASSWORD_HASH is not set, generate it

if [ -z "$AUTHENTICATION_PASSWORD_HASH" ]; then
  echo "Generating password hash - please ensure you remove the password from your␣
↪history"
  export AUTHENTICATION_PASSWORD_HASH=$(echo -n $AUTHENTICATION_PASSWORD | htpasswd -
↪BinC 10 admin | cut -d: -f2)
else
  echo "Using prehashed password"
fi

yq -i .system = env(EDB_TARGET_PLATFORM) |
.parameters.global.portal_domain_name = env(PORTAL_DOMAIN_NAME) |
.parameters.transporter-rw-service.domain_name = env(TRANSPORTER_RW_SERVICE_DOMAIN_NAME)␣
↪|
.parameters.transporter-dp-agent.rw_service_url= "https://"+env(TRANSPORTER_RW_SERVICE_
↪DOMAIN_NAME)+"/transporter" |
.parameters.upm-beacon.server_host= env(BEACON_SERVICE_DOMAIN_NAME) |
.pgai.portal.authentication.staticPasswords[0].email=env(AUTHENTICATION_EMAIL) |
.pgai.portal.authentication.staticPasswords[0].hash=env(AUTHENTICATION_PASSWORD_HASH) |
.pgai.portal.authentication.staticPasswords[0].username=env(AUTHENTICATION_USER) |
.pgai.portal.authentication.staticPasswords[0].userID="c5998173-a605-449a-a9a5-
↪4a9c33e26df" |
.beaconAgent.location=env(LOCATION_NAME) values.yaml

echo "Your values.yaml file is ready"
```

- default-env.sh

```
##  exports for EKS

export EDB_PLATFORM_VERSION="v1.2.0"
export EDB_TARGET_PLATFORM="eks"
export CONTAINER_REGISTRY_URI="docker.enterprisedb.com/pgai-platform"
export IMAGESET_REGISTRY_URI=$CONTAINER_REGISTRY_URI
export IMAGESET_AUTHTYPE="token"
export PORTAL_DOMAIN_NAME="portal.foo.network"
export TRANSPORTER_RW_SERVICE_DOMAIN_NAME="transporter.foo.network"
export BEACON_SERVICE_DOMAIN_NAME="beacon.foo.network"
```

(continues on next page)

(continued from previous page)

```
export AUTHENTICATION_EMAIL="owner@mycompany.com"
export AUTHENTICATION_USER="owner@mycompany.com"
export LOCATION_NAME="default-location"

#  Set a password hash for the user or pass a password to have it hashed for you.

#  If you pass a password, you will need to remove it from your history.

#  You can hash a password using the following command:

#  echo -n "password" | htpasswd -BinC 10 admin | cut -d: -f2

#

#  export AUTHENTICATION_PASSWORD="password"

#  If setting AUTHENTICATION_PASSWORD_HASH, ensure it is single quoted to prevent␣
↪variable expansion (e.g. $2y$10$...).

export AUTHENTICATION_PASSWORD_HASH=$2y$10$vKOAXfLHbeV1OQxMpxlLdOIwnX.JAN.ZrD9ZU//
↪ocrNQwhQIMtXhy
```

Copy all files to a directory on your local machine and `cd` into that directory.

Install secrets

The bootstrap process requires a number of secrets to be installed in the Kubernetes cluster. These secrets are used to authenticate with the EDB Download Repository and to store the credentials for the EDB Software Deployment platform.

to install the secrets required for the bootstrap process, run the `install-secrets.sh` script. (As you already set the token as an environment variable, you can pipe it into the script.)

```
echo $EDB_SUBSCRIPTION_TOKEN | $SHELL eks-install-secrets.sh
```

```
Enter the password for pgai-platform@docker.enterprisedb.com
Creating secret edb-cred
namespace/upm-replicator configured
secret/edb-cred configured
namespace/edbpgai-bootstrap configured
secret/edb-cred configured
secret/edb-cred annotated
PG_CONFOUNDING_KEY is 012345678901234567890ABCDEFG - store safely
Installation completed
```

This process creates secrets needed for Hybrid Manager to run. You can find out more about these secrets in:

- ** *Enabling GenAI Builder on Hybrid Manager* (optional for GenAI Builder)**
- ** *Enabling Data Catalogs on Hybrid Manager* (optional for using Catalog)**
- See also ** *Customizing Migration Portal secrets for secure internal communication* (optional to secure internal communications with Migration Portal)**

Prepare the values file

The bootstrap process requires a number of values to be set in the Helm chart. These values are stored in a file called `values.yaml` . You can create this file by running the `prepare.sh` script, which uses variables from the `default-env.sh` file. Edit the `default-env.sh` file to set the values you require.

This is the default `default-env.sh` file:

```
#  exports for EKS

export EDB_PLATFORM_VERSION="v1.2.0"
export EDB_TARGET_PLATFORM="eks"
export CONTAINER_REGISTRY_URI="docker.enterprisedb.com/pgai-platform"
export IMAGESET_REGISTRY_URI=$CONTAINER_REGISTRY_URI
export IMAGESET_AUTHTYPE="token"
export PORTAL_DOMAIN_NAME="portal.foo.network"
export TRANSPORTER_RW_SERVICE_DOMAIN_NAME="transporter.foo.network"
export BEACON_SERVICE_DOMAIN_NAME="beacon.foo.network"
export AUTHENTICATION_EMAIL="owner@mycompany.com"
export AUTHENTICATION_USER="owner@mycompany.com"
export LOCATION_NAME="default-location"

#  Set a password hash for the user or pass a password to have it hashed for you.

#  If you pass a password, you will need to remove it from your history.

#  You can hash a password using the following command:

#  echo -n "password" | htpasswd -BinC 10 admin | cut -d: -f2

#

# export AUTHENTICATION_PASSWORD="password"

export AUTHENTICATION_PASSWORD_HASH="$2y$10$vKOAXfLHbeV1OQxMpxlLdOIwnX.JAN.ZrD9ZU//
↪ocrNQwhQIMtXhy"
```

Here is a brief guide to the variables in the `default-env.sh` file.

Variable	Description
EDB_TARGET_PLATFOR	The target platform for the installation. Set to eks as we are installing on Amazon EKS.
CON-TAINER_REGISTRY_URI	The URI of the container registry to use. This is set to the EDB Download Repository.
IMAGE-SET_REGISTRY_URI	The URI of the image set registry. This is set to the EDB Download Repository.
IMAGE-SET_AUTHTYPE	The authentication type for the image set registry. Set to token as we are using a token to authenticate with the EDB Download Repository.
TRANS-PORTER_RW_SERVICE_I	The domain name for the Transporter service.
POR-TAL_DOMAIN_NAME	The domain name for the portal.
BEA-CON_SERVICE_DOMAIN	The domain name for the beacon service.

continues on next page

Table 3 – continued from previous page

Variable	Description
AUTHENTICATION_EMAIL	The email address of the user to create.
AUTHENTICATION_USER	The username of the user to create.
AUTHENTICATION_PASSWORD	The password of the user to create. This can be set as plain text or as a hash with AUTHENTICATION_PASSWORD_HASH.
AUTHENTICATION_PASSWORD_HASH	The hash of the password to create.
LOCATION_NAME	The location name for the beacon agent, also known as Agent.

Run the `prepare.sh` script to create the `values.yaml` file.

```
$SHELL prepare.sh
```

This create a `values.yaml` file with the values set in the `default-env.sh` file.

Install the Helm chart repo with the following command:

```
helm repo add edbpgai "https://downloads.enterprisedb.com/${EDB_SUBSCRIPTION_TOKEN}/pgai-
↪platform/helm/charts"
```

Once the charts repository is added, ensure it is up to date:

```
helm repo update
```

Now you can create the `values.yaml` file from the helm repo using the following command:

```
helm show values edbpgai/edbpgai-bootstrap > values.yaml
```

You now have a `values.yaml` file with the chart's default values for the bootstrap process.

You need to create two values for the bootstrap process. These are the AES_256_KEY and the AUTHENTICATION_PASSWORD_HASH. You can create these values with the following commands:

```
export AES_256_KEY=$(openssl rand -base64 32)
```

If you have a password to hash, you can do so with the following command:

```
export AUTHENTICATION_PASSWORD_HASH=$(echo -n $AUTHENTICATION_PASSWORD | htpasswd -BinC␣
↪10 admin | cut -d: -f2)
```

You now need to edit `values.yaml` file and set the values required for your installation. Find the appropriate key in the `values.yaml` file and set the values as follows. Replace $<name> entries with the value you have set the environment variables of the same name to.

1. Sets `bootstrapImageName` to $CONTAINER_REGISTRY_URI followed by "/edbpgai-bootstrap/bootstrap-" and then $EDB_TARGET_PLATFORM. eg, given the values above, this would be `docker.enterprisedb.com/pgai-platform/edbpgai-bootstrap/bootstrap-eks` .
2. Set `bootstrapImageTag` to $EDB_PLATFORM_VERSION.
3. Set `parameters.upm-istio.cookie_aeskey` to $AES_256_KEY.
4. Set `containerRegistryURL` to $CONTAINER_REGISTRY_URI.

5. Set `system` to $EDB_TARGET_PLATFORM.
6. Set `parameters.global.portal_domain_name` to $PORTAL_DOMAIN_NAME.
7. Set `parameters.transporter-rw-service.domain_name` to $TRANSPORTER_RW_SERVICE_DOMAIN_NAME.
8. Set `parameters.transporter-dp-agent.rw_service_url` to `https://` followed by $TRANSPORTER_RW_SERVICE_DOMAIN_NAME followed by `/transporter` . eg, given the values above, this would be `https://transporter.foo.network/transporter` .
9. Set `parameters.upm-beacon.server_host` to $BEACON_SERVICE_DOMAIN_NAME. 10. Set `pgai.portal.authentication.staticPasswords[0].email` to $AUTHENTICATION_EMAIL. 11. Set `pgai.portal.authentication.staticPasswords[0].hash` to $AUTHENTICATION_PASSWORD_HASH. 12. Set `pgai.portal.authentication.staticPasswords[0].username` to $AUTHENTICATION_USER. 13. Set `pgai.portal.authentication.staticPasswords[0].userID` "c5998173-a605-449a-a9a5-4a9c33e26df" . 14. Set `beaconAgent.location` to $LOCATION_NAME

Apply the Helm chart

With the `values.yaml` file prepared, you can apply the Helm chart to start the bootstrap process. This create the necessary resources in the Kubernetes cluster to start the bootstrap process.

```
helm upgrade -n edbpgai-bootstrap \
   --install -f ./values.yaml \
   --version "${EDB_PLATFORM_VERSION/-appl/+appl}" \
   edbpgai-bootstrap edbpgai/edbpgai-bootstrap
```

This command installs the bootstrap container in the Kubernetes cluster. The bootstrap container then installs the rest of the EDB Software Deployment platform.

Follow the bootstrap process

The bootstrap process takes time to complete. You can follow the progress by looking at the logs of the bootstrap container:

```
kubectl logs -f -l app=edbpgai-bootstrap -n edbpgai-bootstrap
```

Configure the IP address in /etc/hosts

Obtain the IP address of your ingress gateway to configure in your `/etc/hosts` file. Run:

```
kubectl -n istio-system get service istio-ingressgateway -o jsonpath={.status.
↪loadBalancer.ingress[0].hostname} | strings | nslookup
```

This command returns output like:

```
Server:         192.168.111.1
Address:        192.168.111.1#53

Non-authoritative answer:
Name:   k8s-istiosys-istioing-b3dc9b4f3c-81dc5c45b1d5f6d2.elb.us-east-1.amazonaws.com
Address: 52.70.180.102
```

At this point your have the IP address of the portal. You can add this IP address to your DNS service and map it to the domain name you set in the `default-env.sh` file, which is `portal.foo.network` in this example.

If you do not have a DNS service configured, consider adding the IP address to your local `/etc/hosts` file. This will allow you to access the Hybrid Manager portal using the domain name you set in the `default-env.sh` file.

You can then add IP address to your `/etc/hosts` file like this:

```
echo "52.70.180.102 portal.foo.network" | sudo tee -a /etc/hosts
```

Substitute the IP address with the value you obtained earlier.

Next steps

You can now verify the installation by *connecting to the HM Console* .

Connecting to the Hybrid Manager Console

To verify that your *Apply the secrets to your installation* of the EDB Software Deployment platform was successful, connect to the HM Console using your browser.

Connect to the portal by entering the URL in your browser:

```
https://portal.foo.network/
```

In this configuration, the https doesn;t validate, as the hostname doesn' t exist on the wider internet. Accept the certificate warning to proceed to the portal.

At the login screen, you can enter the username and password generated with the bootstrap installation either via *Helm values file* or *Operator values file* . See *Managing users in Hybrid Manager* for more information.

After you' ve logged in, you can *Create a database cluster template* .

Uninstall Hybrid Manager from EKS

Uninstalling Hybrid manager from an EKS cluster is dependent on how you installed it.

Uninstalling using the bootstrap installed Hybrid Manager

If you installed Hybrid Manager using the bootstrap installation, you can uninstall it by running the following command:

```
helm uninstall -n edbpgai--n edbpgai-bootstrap edbpgai-bootstrap edbpgai/edbpgai-
↪bootstrap
```

This command will remove the Hybrid Manager components that were installed using the bootstrap method.

Uninstalling using the Hybrid Manager Operator

To uninstall the Hybrid Manager Operator, you can run the following commands:

```
##  by default, EDB Postgres AI is protected from the deletion, disable the protection␣
↪first

kubectl annotate hybridcontrolplane edbpgai biganimal.enterprisedb.io/deletion-
↪protect=disabled --overwrite

##  uninstall the EDB Postgres AI

kubectl delete hybridcontrolplane edbpgai --wait=false
```

(continues on next page)

(continued from previous page)

```
##  monitor the uninstallation process

kubectl get hybridcontrolplane edbpgai -o yaml
```

This will remove the Hybrid Manager Operator and all associated resources from your EKS cluster.

Post uninstallation

Note that after uninstalling Hybrid Manager, your EKS cluster still retains resources created as a side effect of the installation. You can remove these resources manually if you no longer need them, keep them for future use, or delete the EKS cluster itself, which will remove all resources associated with it.

4.3.4 Install Hybrid Manager on GCP GKE

You can install the Hybrid Manager (HM) on Google Cloud GKE .

- *Prerequisites*
- *Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator*
- *Connecting to the portal*

Hybrid Manager installation methods for Google Cloud GKE

There are two methods you can use to install Hybrid Manager on GCP GKE. Select the method that fits your use case:

- Using *Apply the Helm chart* . Use a Helm install for testing and experimenting with Hybrid Manager. A Helm install is suitable for nonproduction use cases.
- Using *Install Hybrid Manager on Google Cloud GKE using EDB Postgres AI Operator* . Use the EDB Postgres AI Operator install when you are ready to use Hybrid Manager in production.

Install Hybrid Manager on GCP GKE using Helm

Set the environmental variables for Helm install

1. To prepare for the install, specify the container registry URI:

```
export CONTAINER_REGISTRY_URI=<local-registry-URI>
```

2. Set the edb-bootstrap image version:

```
export EDBPGAI_BOOTSTRAP_IMAGE_VERSION="<edb-bootstrap-image-version"
```

Make sure to change the value of <edb-bootstrap-image-version> to the image version you're targeting for this install.

3. Set the edb-bootstrap Helm chart version:

```
export EDBPGAI_BOOTSTRAP_HELM_CHART_VERSION="<edbpgai-bootstrap-helm-version>"
```

Be sure to change the value of <edbpgai-boostrap-helm-version> to the edb-pgai Helm chart version you're targeting for this install.

4. Set your EDB Cloudsmith token:

```
export EDB_CLOUDSMITH_TOKEN=<edb-cloudsmith-token>
```

5. Change `edb-cloudsmith-token` to your EDB Cloudsmith token.

> **Note**
>
> For Hybrid Manager version v1.1.0 and later, use the variables *upm-istio.cooke_aeskey* and *upm-istio.openshift_console_domain_name* . Ensure you use the correct variable for you HM version and remove the other parameter in the [Helm upgrade command](Install Hybrid Manager using the Helm chart).

After your environment variables are set, you can prepare for the installation.

Prepare for the installation

1. Add the Helm chart repository from EDB Cloudsmith:

```
helm repo add enterprisedb-edbpgai "https://downloads.enterprisedb.com/${EDB_CLOUDSMITH_
↪TOKEN}/pgai-platform/helm/charts/"
```

2. Update the repository you just added:

```
helm repo update
```

3. Retrieve the default edbpgai-bootstrap Helm `values.yaml` file:

```
helm show values enterprisedb-edbpgai/edbpgai-bootstrap > values.yaml
```

4. Edit the `values.yaml` file to the correct values, replacing the environmental variables you set earlier where appropriate:

```
system: "gke"
containerRegistryURL: "${CONTAINER_REGISTRY_URI}"
bootstrapImageName: "${CONTAINER_REGISTRY_URI}/edbpgai-bootstrap/bootstrap-gke"
bootstrapImageTag: "${EDBPGAI_BOOTSTRAP_IMAGE_VERSION}"
  global:
    default_ingress_domain: "${DEFAULT_INGRESS_DOMAIN}"
    portal_domain_name: "${PORTAL_DOMAIN_NAME}"
  transporter-rw-service:
    domain_name: "${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}"
  transporter-dp-agent:
    rw_service_url: "https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter"
  transporter-data-operator:
    fips_enabled: ${TRANSPORTER_FIPS_ENABLED}
  upm-beacon:
    server_host: "${BEACON_SERVICE_DOMAIN_NAME}"
  upm-istio-gateway:
    cookie_aeskey: "${AES_256_KEY}"
pgai:
  portal:
    authentication:
      staticPasswords:
        - email: "${AUTHENTICATION_EMAIL}"
```

(continues on next page)

(continued from previous page)

```
        hash: "${AUTHENTICATION_HASH}"
        username: "${AUTHENTICATION_USERNAME}"
        userID: "${AUTHENTICATION_USER_ID}"
```

Enable dynamic image discovery

To enable the dynamic image discovery from the registry, you need to configure it with the needed options. See Image discovery configuration for detailed instructions on how to do this.

After identifying the required image discovery configuration, update `values.yaml` with the appropriate settings.

For example, the following is a sample configuration that uses a Google Artifact Registry in the `us-east1` region with `"basic"` authentication:

```
beaconAgent:
  location: "devspatcher"
  beaconDBConnected: true
  plaintext: true
  logLevel: "debug"
  inCluster: true
  provisioning:
    provider: "gcp"
    imageDiscovery: true
    loadBalancersEnabled: true
    imagesetDiscoveryAuthenticationType: "basic"
    imagesetDiscoveryContainerRegistryURL: "us-east1-docker.pkg.dev"
  providers:
    - appliance
beaconServer:
  logLevel: "debug"
```

Install Hybrid Manager using the Helm chart

With everything prepared, you can install Hybrid Manager using the edbpgai-bootstrap Helm chart:

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    -f ./values.yaml \
    --version "${EDBPGAI_BOOTSTRAP_HELM_CHART_VERSION}" \
    edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap
```

You can then monitor the install process using the following command. It takes about 20-30 minutes.

```
kubectl logs -f -l app=edbpgai-bootstrap -n edbpgai-bootstrap
```

After the install completes successfully, *update DNS zone records and verify the install* .

Install Hybrid Manager on Google Cloud GKE using EDB Postgres AI Operator

Install preparation

1. The following environmental variable is required for authentication to obtain the operator:

```
export EDB_CLOUDSMITH_TOKEN=<edb-cloudsmith-token>
```

1. Create a new namespace for the operator:

```
kubectl create namespace edb-hcp-operator-system
```

1. Add the Helm chart repository for the operator and update it:

```
helm repo add enterprisedb-edbpgai "https://downloads.enterprisedb.com/${EDB_CLOUDSMITH_
↪TOKEN}/pgai-platform/helm/charts/"
helm repo update
```

1. Proceed with the installation using the appropriate version, in this case 1.2:

```
export OPERATOR_VERSION=1.2.0
```

Install the operator on GKE

Install the operator using the instructions for your network use case:

1. To install EDB Postgres AI Operator on GKE, first ensure you've prepared the image pull secret for accessing the EDB Cloudsmith repository. Also verify that you have the necessary permissions to pull images from the repository `docker.enterprisedb.com/pgai-platform/edb-hcp-operator/manager` .

```
export DOCKER_SERVER="docker.enterprisedb.com"
export DOCKER_USERNAME="<your-docker-user-name>"
export DOCKER_PASSWORD="<your-docker-password>"
```

1. Create the image pull secret for the operator:

```
kubectl create secret docker-registry edb-hcp-operator-pull-secret \
    --docker-server="${DOCKER_SERVER}" \
    --docker-username="${DOCKER_USERNAME}" \
    --docker-password="${DOCKER_PASSWORD}" \
    --namespace edb-hcp-operator-system
```

1. Install the operator:

```
helm upgrade \
    -n edb-hcp-operator-system \
    --install \
    --set "imagePullSecrets[0].name=edb-hcp-operator-pull-secret" \
    --set controllerManager.manager.image.tag=$OPERATOR_VERSION \
    --set controllerManager.manager.image.repository=docker.enterprisedb.com/pgai-
↪platform/edb-hcp-operator/manager \
    --version $OPERATOR_VERSION \
    edb-hcp-operator enterprisedb-edbpgai/edb-hcp-operator
```

Install Hybrid Manager using the operator

Apply the YAML and install

After EDB Postgres AI Operator is installed on your GKE cluster, you need to use the operator to install Hybrid Manager. The following YAML file initiates the Hybrid Manager installation:

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  source:
    useLocalKustomizations: true
  flavour: gke
  version: v1.2.0+appl
  imageRegistry: ${CONTAINER_REGISTRY_URI}
  imagePullSecrets:
    - name: edb-cred
      namespace: edbpgai-bootstrap
  globalParameters:
    internal_backup_folder: 4befab698c8d
    portal_domain_name: ${PORTAL_DOMAIN_NAME}
    storage_class: ${STORAGE_CLASS}
  componentsParameters:
    edb-migration-portal:
      enable_pdb: "false"
    transporter-data-operator:
      fips_enabled: "${TRANSPORTER_FIPS_ENABLED}"
    transporter-dp-agent:
      rw_service_url: https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter
    transporter-rw-service:
      domain_name: ${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}
    upm-api-pnxt-metric:
      log_level: info
      tsdb_host: thanos-query.monitoring.svc.cluster.local
      tsdb_port: "9090"
    upm-api-pnxt-pd:
      log_level: info
    upm-beaco-ff-base:
      usage_generator_interval: 24h
    upm-beaco-usages:
      usage_generator_interval: 24h
    upm-beacon:
      server_host: ${BEACON_SERVICE_DOMAIN_NAME}
      beacon_server_replicas: "1"
      external_metrics_replicas: "1"
      beacon_db_hibernation: "\"off\""
    upm-istio-gateway:
      cookie_aeskey: ${AES_256_KEY}
    upm-prometheus:
      memory: 4Gi
  pgai:
```

(continues on next page)

(continued from previous page)

```
    portal:
      authentication:
        clientSecret: edb-postgres-ai-secret
        idpConnectors: []
        staticPasswords:
          - email: ${AUTHENTICATION_EMAIL}
            hash: ${AUTHENTICATION_HASH}
            username: ${AUTHENTICATION_USERNAME}
            userID: ${AUTHENTICATION_USER_ID}
  beaconAgent:
    applianceProviderEnabled: true
    plaintext: true
    beaconServerAddress: "upm-beacon-server-ingestion.upm-beacon.svc.cluster.local:8080"
    tlsInsecure: false
    location: "devspatcher"
    logLevel: "debug"
    beaconDBConnected: true
    inCluster: true
    provisioning:
      loadBalancersEnabled: true
      nodePortDomain: "nodeportdomain.com"
      imageDiscovery: true
      imagesetDiscoveryAuthenticationType: "basic"
      imagesetDiscoveryContainerRegistryURL: "us-east1-docker.pkg.dev"
    providers:
      - appliance
```

After the YAML has been applied, you can use the following command to monitor the installation status:

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

Three different statuses can be returned, depending on the phase the installation is currently in:

```
- deployed # the installation was successful

- deploying # the installation is still in progress

- failed: # the install was unsuccessful
```

After the install is successful, *updating DNS zone records and verify the install* **.

Uninstalling Hybrid Manager

If you installed Hybrid Manager using EDB Postgres AI Operator (versus *Apply the Helm chart*), you can uninstall it.

> **Warning**
>
> The following procedure uninstalls Hybrid Manager. Use this procedure only if you're sure you want to uninstall Hybrid Manager.

There may be cases where you want to delete Hybrid Manager. Because Hybrid Manager is protected from deletion by default, you must first disable that protection:

```
kubectl annotate hybridcontrolplane edbpgai biganimal.enterprisedb.io/deletion-
↪protect=disabled --overwrite
```

You can then uninstall:

```
kubectl delete hybridcontrolplane edbpgai --wait=false
```

To monitor the process, run:

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

Updating DNS zone records

After you've completed the Helm or Operator install, update your DNS zone records and verify that Hybrid Manager is working as expected.

Update DNS zone records

1. Retrieve the hostname of the portal service. This hostname represents the ingress endpoint for your Hybrid Manager.

```
kubectl get svc -n istio-system istio-ingressgateway -o yaml | yq .status.loadBalancer.
↪ingress[0].hostname
```

1. Create CNAME records in your DNS zone for the following domain names and point them to the retrieved hostname.

- `PORTAL_DOMAIN_NAME`
- `TRANSPORTER_RW_SERVICE_DOMAIN_NAME`
- `BEACON_SERVICE_DOMAIN_NAME`

Each CNAME record maps a user-facing domain name to the ingress hostname.

Now you can *connect to the portal* and verify the installation.

Prerequisites

GCP GKE has prerequisites to meet before you can install the EDB Postgres AI Software Deployment. Fulfill the prerequisites to ensure that you have everything you need before you start the installation process.

Prerequisites

- *A system from which you can run the installation process*
- An GCP account with the necessary permissions
- *A GCP cluster configured to host HM*

Next steps

After you complete the prerequisites, you can move on to *Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator* .

Install system

You need a system from which you can run the installation process. This system needs the following software installed:

- kubectl to manage your Kubernetes cluster.
- Helm to render and apply the Kubernetes manifest.

You can install these tools using the linked instructions or by using the instructions for your particular operating system that follow.

macOS

Install kubectl and Helm:

```
brew install kubectl helm
```

Linux

=== "Ubuntu"

> **Note**
>
> These instructions are for Ubuntu 16.04 or newer, as they assume Snap support is available by default.

```
sudo snap install helm --classic
sudo snap install kubectl --classic
```

=== "Red Hat"

Install kubectl:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/
↪bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
kubectl version --client
```

Install Helm:

```
curl -LO https://get.helm.sh/helm-v3.12.3-linux-amd64.tar.gz
tar -xvf helm-v3.12.3-linux-amd64.tar.gz
sudo mv linux-amd64/helm /usr/local/bin/helm
helm version
```

=== "Debian"

```
sudo apt update
sudo apt install snapd
sudo snap install snapd
sudo snap install kubectl --classic
```

Windows

=== "Winget"

```
winget install kubectl --source winget
winget search helm
winget install helm
```

=== "Manually"

1. Install kubectl .
2. Install *Installing Helm* .

Installing Helm

1. Download the Helm binary for Windows. On the Helm GitHub Releases page , locate the latest release and download the helm-v3.x.x-windows-amd64.zip file.
2. Extract the Zip file. Use a tool like 7-Zip, WinRAR, or the built-in Windows extraction utility to unzip the downloaded file.
3. Copy `helm.exe` from the extracted folder to a directory in your path. Common directories include `C:\Program Files\` , `C:\Program Files (x86)\` , or you can create a directory, such as `C:\Helm\` , and add it to your path.
4. Verify the installation:
5. Open a new Command Prompt or PowerShell window.
6. Run the following command to verify that Helm is installed correctly:

```
helm version
```

3. The output displays Helm version information.

Next steps

Once you have installed the necessary software, you can move on to setting up an GCP account with the necessary permissions.

Creating or configuring a GKE cluster for Hybrid Manager

GKE cluster

You need a GKE cluster to install Hybrid Manager (HM) onto. Follow the documentation on Creating a regional cluster in the GKE docs to set up a new one, if needed.

Permissions

Ensure you have the permission `Kubernetes Engine Admin` in `roles/container.admin` .

Different node types

HM supports running management services (HM control plane) and PostgreSQL workloads on different sets of nodes on GKE, requiring additional node pools.

To implement the different node sets, follow these steps:

HM control plane nodes

To make the HM control plane nodes a logical grouping that can be implemented with their own node type:

1. In your labels, set the following:

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

1. In your taints, set the following:

```
key: edbaiplatform.io/control-plane
value: "true"
effect: NoSchedule
```

PostgreSQL workload nodes

To make the Postgres workload nodes a logical grouping that can be implemented with their own node type:

1. In your labels, set the following:

```
edbaiplatform.io/postgres: "true"
```

1. In your taints, set the following:

```
key: edbaiplatform.io/postgres
value: "true"
effect: NoSchedule
```

For more information on how to append labels and taints for GKE managed node groups, see the documentation here .

Configure object storage

Object storage backend: To support HM backups, connect the cluster to a dedicated object storage solution (`STORAGE_CLASS`), for example, to a GCP bucket.

Among others, Hybrid Manager uses this bucket to store:

- Managed storage locations
- Postgres WALs + backups
- Logs (both Postgres and internal services)
- Metrics (both Postgres and internal services)

Create a GCP bucket

Download and install the gcloud CLI .

1. Set environment variables for the creation of your GCP bucket:

```
export GCP_PROJECT_ID="appliance-dev-444915" # change me to your own gcp project id
export GCP_BUCKET_NAME="tq-upm45906" # change me to your preferred bucket name
export GCP_BUCKET_REGION="us" # change me to your preferred location id
```

1. Set the project that will host the GCP bucket you' ll use for HM:

```
gcloud config set project $GCP_PROJECT_ID
```

1. Create the bucket:

```
gcloud storage buckets create gs://$GCP_BUCKET_NAME --location=$GCP_BUCKET_REGION
```

Ensure the bucket has hierarchical namespaces disabled.

Create a bucket user

After creating a dedicated bucket, create a service account for the Hybrid Manager with the necessary permissions (`roles/storage.admin`) to access the bucket.

1. Set the necessary environment variables:

```
export SERVICE_ACCOUNT_NAME="tq-upm45906" # change this to your preferred service␣
↪account name
export SERVICE_ACCOUNT_EMAIL=$SERVICE_ACCOUNT_NAME@$GCP_PROJECT_ID.iam.gserviceaccount.
↪com
```

1. Create the service account:

```
gcloud iam service-accounts create $SERVICE_ACCOUNT_NAME
```

1. Assign the storage admin role to the service account for the bucket:

```
gcloud storage buckets add-iam-policy-binding gs://$GCP_BUCKET_NAME \
  --member="serviceAccount:$SERVICE_ACCOUNT_EMAIL" \
  --role="roles/storage.admin"
```

1. Create a json key with the user permissions:

```
gcloud iam service-accounts keys create key-file.json \
  --key-file-type=json \
  --iam-account=$SERVICE_ACCOUNT_EMAIL
```

1. Get the base64 encoded credentials:

```
export GCP_CREDENTIAL_BASE64=$(base64 -w0 < key-file.json)
```

```
export GCP_CREDENTIAL_BASE64=$(base64 < key-file.json)
```

Create a secret for bucket access

After preparing the dedicated user for HM access to the bucket, create a secret that associates the created user with the Hybrid Manager to provide access for object storage.

```
apiVersion: v1
kind: Secret
metadata:
  # The name must be edb-object-storage
  name: edb-object-storage
  # The namespace must be default
```

(continues on next page)

(continued from previous page)

```
  namespace: default
stringData:
  # Specifies the cloud provider. In this case, it is set to gcp.
  provider: gcp
  # The region where the GCP bucket is located.
  location_id: ${GCP_BUCKET_REGION}
  # The ID of the GCP project.
  project_id: ${GCP_PROJECT_ID}
  # The name of the GCP bucket.
  bucket_name: ${GCP_BUCKET_NAME}
  # The base64 encoded JSON credentials for accessing the GCP bucket.
  credentials_json_base64: ${GCP_CREDENTIAL_BASE64}
```

Create a AES-256 Key

Generate an AES-256 key for authentication.

```
export AES_256_KEY=$(openssl rand -base64 32)
```

Sync HM container images to your private registry

A requirement to use HM is that you must host your own secure and approved internal registry. You also need to know what version of HM you want to install. The sync process is basically to take all the artifacts from Cloudsmith and sync them internally to the internal registry before installing or upgrading HM with the relevant Helm chart.

The sync process preserves the container images SHA256 to ensure the images security and immutability across different environments.

To perform the process:

1. Install skopeo , as you use this tool to do the sync.
2. Configure the HM release to be synced:

```
export EDBPGAI_RELEASE=<your-release-version>
```

1. Configure the EDB Cloudsmith access token:

```
export CS_EDB_TOKEN=<your-cloudsmith_edb_token>
```

1. Download the image list artifact:

```
curl -sLO "https://downloads.enterprisedb.com/${CS_EDB_TOKEN}/pgai-platform/raw/names/$
↪{EDBPGAI_RELEASE}-images.txt/versions/${EDBPGAI_RELEASE}/images.txt"
```

1. Configure the EDB Cloudsmith registry source:

```
export LOCAL_REGISTRY_URI=<your-local-registry-address>
```

1. Use skopeo to log in to the source registry and provide any necessary credentials as requested:

```
skopeo login docker.enterprisedb.com
```

1. Use skopeo to log in to the desitination registry and provide any necessary credentials as requested:

```
skopeo login <your-local-registry-address>
```

1. Parse the image list and sync each image:

```
while read -r image; do skopeo --override-os linux copy --multi-arch all docker://$EDB_
↪SOURCE_REGISTRY/${image/:*@/@} docker://$LOCAL_REGISTRY_URI/${image/:*@/@} --retry-
↪times 3; done < images.txt
```

Create the namespaces required for HM and necessary secrets

In this phase of the preinstall, you create the namespaces necessary for HM and then its required secrets. Both namespaces and secrets are used in the install process.

1. Set the necessary container registry environmental variables:

```
export CONTAINER_REGISTRY_URI=<local-registry-uri>
export CONTAINER_REGISTRY_USERNAME=<container-registry-username>
export CONTAINER_REGISTRY_PASSWORD=<container-registry-password>
```

1. Generate and set the Postgres confounding key for LakeKeeper and fernet key for Griptape:

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

1. Create the required namespaces:

```
kubectl create ns edbpgai-bootstrap
kubectl create ns upm-replicator
kubectl create ns upm-lakekeeper
kubectl create namespace upm-griptape
```

1. Create the secrets for HM, UPM-replicator, LakeKeepr, and Griptape:

```
kubectl create secret docker-registry edb-cred \
    -n edbpgai-bootstrap \
    --docker-server="${CONTAINER_REGISTRY_URI}" \
    --docker-username="${CONTAINER_REGISTRY_USERNAME}" \
    --docker-password="${CONTAINER_REGISTRY_PASSWORD}"
```

```
kubectl create secret docker-registry edb-cred \
    -n upm-replicator \
    --docker-server="${CONTAINER_REGISTRY_URI}" \
    --docker-username="${CONTAINER_REGISTRY_USERNAME}" \
    --docker-password="${CONTAINER_REGISTRY_PASSWORD}"
```

```
kubectl create secret generic pg-confounding-key \
--from-literal=PG_CONFOUNDING_KEY=${PG_CONFOUNDING_KEY} \
--namespace=upm-lakekeeper
```

```
kubectl create secret generic fernet-secret \
--from-literal=FERNET_KEY=${FERNET_KEY} \
--namespace=upm-griptape
```

1. Annotate the secrets:

```
kubectl annotate secret -n upm-replicator edb-cred replicator.v1.mittwald.de/replicate-
↪to="*" --overwrite
```

Griptape secret

See the following documentation for setting up the Griptape secret.

Catalog secret

See the following documentation for setting up the Catalog secret.

With the secrets all set, you can now move on to *setting up the environmental variables* needed for installation.

Set the necessary environmental variables for install

Set domain variables

The Hybrid Manager (HM) cluster needs access to certain user domains during the installation.

1. Set the variable for the HM's Portal domain name:

```
export PORTAL_DOMAIN_NAME=<your-hm-portal-domain-name>
```

1. Set the variable for Transporter RW service domain name:

```
export TRANSPORTER_RW_SERVICE_DOMAIN_NAME=<transporter rw service domain name> #␣
↪Customer domain name for the EDB Postgres AI Platform Transporter RW Service
```

1. Set the variable for the EDB Postgres AI Agent domain name:

```
export BEACON_SERVICE_DOMAIN_NAME=<beacon service domain name> # Customer domain name␣
↪for the EDB Postgres AI Beacon Service
```

Set storage class variable

The HM cluster requires a predefined storage class. :

```
export STORAGE_CLASS=<storage class>
```

One option is to use the default GKE storage class, `standard-rwo` .

Set authentication variables

Some authentication values are needed for installing HM.

1. Set the variable for your email address:

```
export AUTHENTICATION_EMAIL=<your-email>
```

1. Set the variable for your password hash—the hashed password you obtained. For example: `echo "mypassword" | htpasswd -BinC 10 admin | cut -d: -f2`:

```
export AUTHENTICATION_HASH=<your-password-hash>
```

1. Set the variable for your username. Owner MyCompany is the default user name.

```
export AUTHENTICATION_USERNAME=<Owner MyCompany>
```

1. Set your user id:

```
export AUTHENTICATION_USER_ID="your-unique-user-identifier"
```

Optional: set FIPs mode Transporter Data Operator

Optionally, you can set FIPS mode for the Transporter Data Operator:

```
export TRANSPORTER_FIPS_ENABLED=true # Enable FIPS mode for Transporter Data Operator
```

Connecting to Hybrid Manager

To verify that your installation of the EDB Postgres AI Hybrid Manager platform was successful, connect to Hybrid Manager Console using your browser.

```
https://portal.foo.network/
```

In this configuration, the https doesn't validate, as the hostname doesn't exist on the wider internet. Accept the certificate warning to proceed to the portal. Accept the certificate warning to proceed to the portal.

At the login screen, you can enter the default username and password generated with the bootstrap installation: `owner@mycompany.com` and `password` .

After you log in, you can *Create a database cluster template* .

4.3.5 Customizations

You can enable additional features for your Hybrid Manager by customizing your installations.

- Enable *Enabling GenAI Builder on Hybrid Manager* to build AI agents.
- Enable *Enabling Data Catalogs on Hybrid Manager* to manage data in a single table for analytics workloads.
- Customize *Customizing Migration Portal secrets for secure internal communication* for secure communication between internal components.

Enabling Data Catalogs on Hybrid Manager

The Hybrid Manager's 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 can skip key and secret creation and go straight 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 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 is at least 32 bytes long. - Create a new confounding key per Hybrid Manager deployment. .. ::

1. Create a namespace for the service:

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

1. Create a secret that references the created 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 have configured the secret, continue with the *Hybrid Manager installation methods for Google Cloud GKE* .

Backing up the confounding key

The Hybrid Manager administrator must keep the confounding key safe and back it up. The loss of this key in a disaster scenario leads to a situation in which there is no mechanism for accessing the Lakehouse data managed by the Hybrid Manager data catalog. This would require the administrator 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. Such a procedure is full of risks and would require support from EDB PG AI Professional Services team.

Fetch the key so you can store it safely:

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

Enabling GenAI Builder on Hybrid Manager

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 straight 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 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. .. ::

1. Create a namespace for the integration:

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

1. Create a secret that references the created fernet key and is stored in the dedicated namespace:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
   name: fernet-secret
   namespace: upm-griptape
stringData:
   FERNET_KEY: ${FERNET_KEY}
EOF
```

Configuring DataLake object storage for GenAI builder

1. Create a DataLake bucket in the object storage you are using for your Hybrid Manager deployment. GenAI Builder will use 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 new bucket with a unique name for your Griptape DataLake bucket.

1. Note the following information for your bucket. You will need it later when first using the GenAI launchpad application. The UI will prompt you for your DataLake bucket configuration, which requires the following:

- `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": [
```

(continues on next page)

(continued from previous page)

```
            "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 utilize 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 thay 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.

1. 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 the Hybrid Manager.

After you have configured the secret and created a dedicated bucket, continue with the *Hybrid Manager installation methods for Google Cloud GKE* .

Backing up the Fernet key

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

Fetch the key so you can store it safely:

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

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, we strongly recommend overriding them with custom secrets for production environments.

This ensures the secrets are not easily discoverable, reducing risk of breaches.

Understanding Migration Portal secrets

The **description** of each secret explains why it is needed, the **helm chart** explains where in the configuration file it must be referenced, and the **parameters** sections provide guidance on how to set non-default values.

Whereas this section provides an overview of the secrets and their functions, *Overriding the default secrets* provides an example on 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. This user must also be entered 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 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 the 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 that Migration Portal will 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 will ensure 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 that the copilot will use when accessing the Migration Portal database.
- `stringData.password` : Assign a generated password for this user.

Monitoring and Migration Copilot

Description: Allows the 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
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 different username than 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 are using, override the values by referencing the custom secrets in either the `values.yaml` config file, or the `HybridControlPlane` custom resource.

This example demonstrates how to create custom secrets and configure the 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 will generate 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="${PORTAL_DB_USERNAME}" \
  --from-literal=password="${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 case of using the secret overrides.

If you are 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.

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 Tasks

These sections cover day-to-day tasks in Hybrid Manager:

- *Assessing schemas for external projects* — 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.
- API Access — automate platform functions with the HM API.
- *Upgrading on Hybrid Manager* — follow procedures for major and minor upgrades of Postgres and PGD.
- *Customizing Migration Portal secrets for secure internal communication* — run schema and data migrations from Oracle, RDS, or other Postgres installations.
- Image Management — manage container images, repositories, and AI knowledge base assets.
- *Enabling the Migration Portal AI Copilot* — unlock guided migration workflows with Copilot.

5.1.1 Hybrid Manager Terminology

Review these terms to understand and leverage PG AI Hybrid Manager capabilities.

PG AI Hybrid Manager

PG AI 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 PG AI 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 PG AI 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 PG AI 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 PG AI 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 *Assessing schemas for external projects* for more information.

Estate

In the PG AI 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 PG AI Hybrid Manager, or connected an external database for monitoring, it becomes part of your Estate.

See *Postgres Estate Statistics* 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 PG AI 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 PG AI Hybrid Manager, a Kubernetes cluster represents the foundational infrastructure upon which the PG AI Hybrid Manager software itself is deployed as a self-contained instance. PG AI 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 PG AI 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 PG AI 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 PG AI 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 PG AI Hybrid Manager or created externally but then connected to the PG AI Hybrid Manager for observability. See *Database cluster types in the PG AI Hybrid Manager* for an overview of all cluster types you can create or connect to the PG AI Hybrid Manager.

Not to be confused with **Kubernetes* cluster* or **Postgres* cluster* .

Database cluster types in the PG AI Hybrid Manager

The PG AI 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 PG AI Hybrid Manager, while others are created externally but connected to the PG AI Hybrid Manager to provide monitoring and migration capabilities.

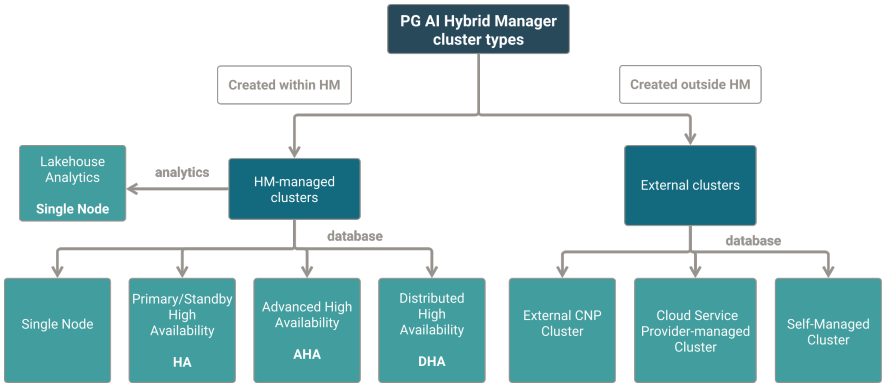


Fig. 1: Cluster Types in PG AI Hybrid Manager

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 the PG AI 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 the PG AI 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.

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 PG AI 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 the PG AI Hybrid Manager's unified control plane, users can easily deploy and manage these specialized clusters optimized for fast analytical queries.

The PG AI 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 the PG AI Hybrid Manager). They are deployed using other provisioning capabilities than the PG AI Hybrid Manager (e.g. manually), and are often deployed on a different machine than the PG AI 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 the PG AI Hybrid Manager for observability with the PG AI Agent and perform migrations with the Data Migration Service .

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 the PG AI Hybrid Manager don't appear under **Estate > Postgres**. Switch to **Estate > Migrations** to see information.

Cloud Service Provider-managed cluster

Cloud Service Provider-managed clusters or CSP-managed clusters are external clusters (created outside of the PG AI 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 the PG AI Hybrid Manager for observability and perform migrations with the Data Migration Service .

Examples: Postgres on AWS RDS.

External CNPG cluster

External CNPG database clusters are external database clusters created on a Kubernetes environment outside of the PG AI 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 the PG AI 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.

5.1.2 Hybrid Manager Console

What is Hybrid Manager Console?

Hybrid Manager (HM) Console is a user interface for provisioning and managing Postgres clusters. It's a centralized location—a console—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 Hybrid Manager 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

Hybrid Manager 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 a variety of 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.
- *The Asset Library* , where you can view container images, manage repositories, and view knowledge bases.

Launching third-party services

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

Exploring Hybrid Manager Console

Accessing the console

Once you have your account, to access Hybrid Manager 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, it opens on the overview page for your account and the *single pane of glass* summary of your EDB Postgres AI Hybrid Manager services. To return to this view from anywhere in the portal, select the EDB logo.

In this view, you can see summaries of your *Assessing schemas for external projects* 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 *Assessing schemas for external projects* .

Monitoring resources in your estate

In Hybrid Manager 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***
- *Configure an Iceberg REST catalog connection*
- **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 the Hybrid Manager, by selecting the database cluster in the table to open a detailed view.
- View an *assessment of the databases* that you have connected to the Hybrid Manager 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* .
- ref:*Create a destination clusters for a migration and perform a database migration <Migrations>* .

In the **Lakehouse** tab of the Estate, you can:

- Monitor your *analytics clusters* , Hybrid Manager clusters specialized for analytics workloads.

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

- View all *Configure an Iceberg REST catalog connection* tables attached to or created with the Hybrid Manager.

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**.

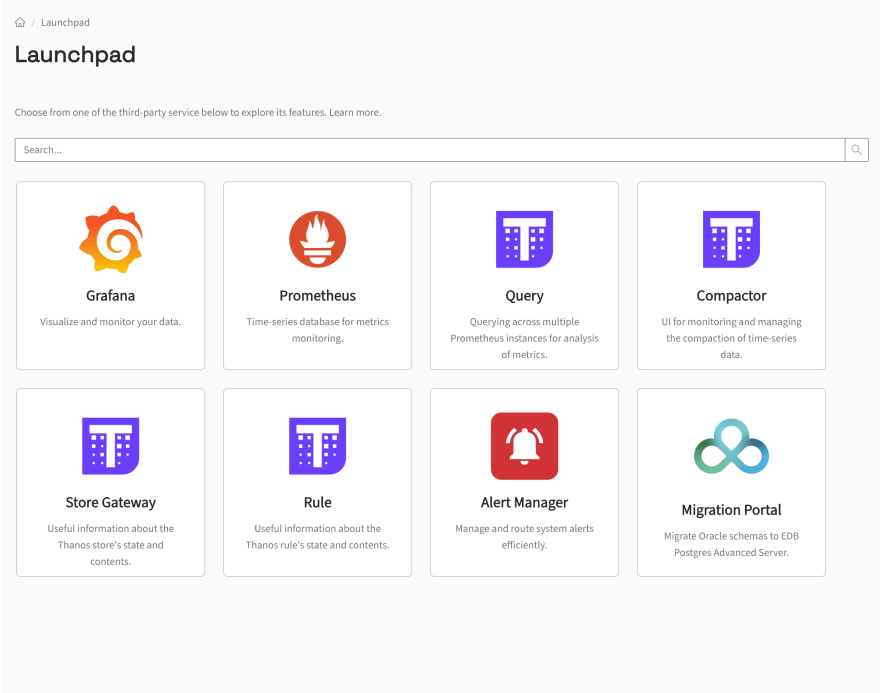


Fig. 2: Launchpad

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:

- *Using Grafana dashboards*
- 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 creation, deletion, and updating of 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:

- # of Alerts — Total number of alerts for the cluster.
- # of Cores — Total number of cores used by the cluster.
- Conn. Usage — Connection usage as a percentage.
- IO Wait % — Percentage of the wait events of type Input/Output.
- IOPS — Number of input/output operations per second for the cluster.
- Kernel Version — Platform type of a self-managed cluster.
- Last Backup — ISO 8601 timestamp of the last time the cluster was backed up.
- OS Version — Operating system version of a self-managed cluster.
- Provider — ID of the CSP provider.
- Region — Locationid for an appliance cluster.

- Rep. Lag — Replication lag in milliseconds.
- TDS — Total database size in gibibytes.

> **Note**
>
> 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 are interested in using two techniques:

- You can use the table's **Search** box to search by name for the clusters you want to display.
- You can use the **Filter** control to display clusters that match the following properties. Each property has a sub-menu 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 the Hybrid Manager.
    - 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

From the **Estate** page, select **Lakehouse** to view your Lakehouse clusters. 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, see Analytics/Lakehouse .

Lakehouse clusters table

The clusters table is a list of all of the available Lakehouse clusters across your projects, which helps you monitor their performance, health, status, and other metrics. You can modify the view to see more or fewer columns, and you can apply filters to see only clusters of interest.

Selecting columns to display

By default, the table shows these columns:

- Project Name — Project that the cluster belongs to.
- Cluster — Cluster name.
- Management — How the cluster is managed: by appliance or self-managed.
- Cluster Type — The type of cluster: PGD, single-node, or HA.
- Status — Current status of the cluster: OK, Error, Caution, or Working.
- Engine — Current Postgres distribution type this cluster is running: Postgres, EDB Postgres Advanced Server, or EDB Extended Server.
- Version — Major version of Postgres distribution the cluster is running: 13, 14, 15, 16, or 17.
- Alerts — Count of high-, medium-, and low-severity alerts for the cluster.
- Health Scores
- 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:

- # of Alerts — Total number of alerts for the cluster.
- # of Cores — Total number of cores used by the cluster.

- Conn. Usage — Connection usage as a percentage.
- IO Wait % — Percentage of the wait events of type Input/Output.
- IOPS — Number of input/output operations per second for the cluster.
- Kernel Version — Platform type of a self-managed cluster.
- 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.

> **Note**
>
> 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 are interested in using two techniques:

- You can use the table's **Search** box to search by name for the clusters you want to display.
- You can 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.

5.1.3 For Users

Projects

In Hybrid Manager, 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.

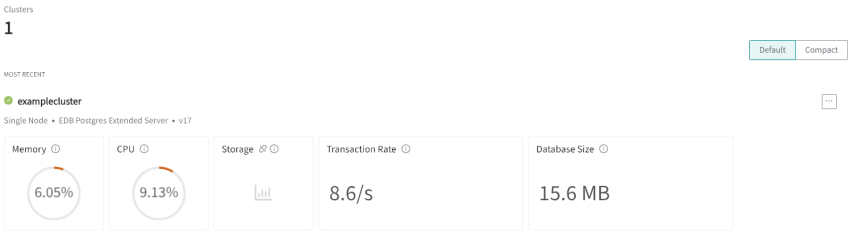


Fig. 3: Project Overview

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 *Credentials* .

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.

- Profile — Select **Profile** to change your project's name at any time. Project names must be unique in a Hybrid Manager instance.
- ref:*Notifications <Notifications>* — Select **Notifications** to configure how the project's notifications are send.
- Security — Select **Security** to add externally-managed keys for transparent data encryption (TDE) to this project. You can use these keys later during cluster creation to add data encryption. Refer to *Enabling Key Management Systems for TDE* for KMS-specific instructions.
- ref:*Alerts <Alerts>* — Select **Alerts** to see the alert settings of the project and control how and when they're sent.

Notifications

With EDB Postgres AI, you can customize the types of notifications you want to receive. There are two types of notification channels:

- **User notifications channels** — you can receive notifications via email, in-app inbox, or both. These notifications are for all users who have enabled them and if their role allows it.
- **System notifications channels** — you can receive notifications via email (mail group), webhook, pager duty, or any combination. These notifications are designed to be triggered once per event, primarily to automate tasks within your organization.

The user's role in the Hybrid Manager (HM) determines the extent of their access to notification preferences. The notifications are categorized into the following preference sections:

- Account — only for user notification channels
- Global — for both user and system notification channels
- Projects — for both 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 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 organisation owner or at the project level by the project owner. By default, each organisation's project inherits the global-level system notifications channel. However, you can enable/disable or re-configure the system notifications channel at the project level to override the default settings.

These steps show how to configure the system notification channels at Organization-level:

Configure Notification at the Global level

The Global level notifications can be set by Organization owner only.

Webhook:

1. Log into the EDB Postgres AI console.
2. From the menu under your name in the top-right panel, select Settings.
3. Select the Notifications tab. Options available are Webhook, Email, and Pager Duty.
4. Select Webhook and provide the details on the pop-up window:

i. Host URL of the Webhook

ii. Method of the Webhook - POST or PUT

iii. Optionally, provide one of the following authentication parameters

- For Basic Auth:
    1. Username
    2. Password
- For Bearer Token:
    1. Bearer Token

iv. Select the Save button.

Pager Duty:

1. Log into the EDB Postgres AI console.
2. From the menu under your name in the top-right panel, select Settings.
3. Select the Notifications tab. Options available are Webhook, Email, and Pager Duty.
4. Select Pager Duty and provide the details on the pop-up window:
    1. Set Routing Key(integration key).
    2. Select the Save button. !!!note Before configuring the Pager Duty notification channel:
    - You need to have access to a Pager Duty account with elevated privileges.
    - Pager Duty Service was created and configured with Events API v2 integration.
    - In your Pager Duty account, navigate to the desired Service and access its “Integrations” settings.
    - Add a new “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.

Email:

1. Log in to the EDB Postgres AI console.
2. From the menu under your name in the top-right panel, select Settings.
3. Select the Notifications tab. Options available are Webhook, Email, and Pager Duty.
4. Select Email and provide the details on the pop-up window:
    1. Email To - Optionally, for the system email notifications (Email group)
    2. Email From - for the email which is sending the email
    3. Host - URL of the email service

4. Port - port of the email service
5. Username - username of the email service
6. Password - of the email service
7. Select the Save button.

Configure Notification at Project level

Project-level notification settings can override global settings. Each project can have its unique configuration.

1. Log in to the EDB Postgres AI console
2. Select the Projects tab on the top
3. Select any Project
4. Go to the Project Overview page
5. Select Settings on the left-side menu
6. Select Notifications
7. Follow the steps mentioned in the global notification setting 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 EDB Postgres AI.

Select the bell icon at the top of your EDB Postgres AI 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

To manage the notification preferences:

1. Log into the EDB Postgres AI console.
2. From the menu under your name in the top-right panel, select My Account.
3. Select the Notifications tab. Notifications are grouped by account, global, and projects available to you.
4. Scroll to the Account, Global or Project section to view the events you wish to manage.
5. Select the right arrow next to the entity to expand the list of events.
6. For each notification, you can: - You can use the toggle button in front of the event name to opt out of or into receiving notifications for a particular event. - Customize the email and in-app notifications for the event by using the checkbox 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 the 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.

1. For HM clusters:
    - Go to Console, and on the top select Projects
    - Select any project from the list
    - In the left navigation, go to Clusters
    - Select any cluster with a ready state
    - On the Cluster detail page, select the Monitoring tab
2. For external HM clusters:

```
-  Go to Console, and on the top select Estate
```

    - Go to the table listing all the clusters
    - Select a ready cluster with Management as Self-managed
    - 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
- the Date Range Selector for a specific date range-based search
- the Filter for a severity level-based search
- the Sort utility, you can sort the alerts list based on severity, alert start time, and alert name
- the Refresh button, to refresh the alerts list
- the Column Selector dropdown, you can select the columns to be displayed in the table according to your preferences
- the Download button, to download the alerts list in a CSV format

You can use the Refresh button 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 automatically 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

Alerts can be configured 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. Sign in to the EDB Postgres AI Console .
2. On top of the console, Select Projects
3. Select any project from the list
4. In the left navigation, go to Setting
5. 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 are exceeded.

The list of the alerts with the default configuration:

The alerts that 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

The alerts that can be configured at host level:

- CPU Utilization

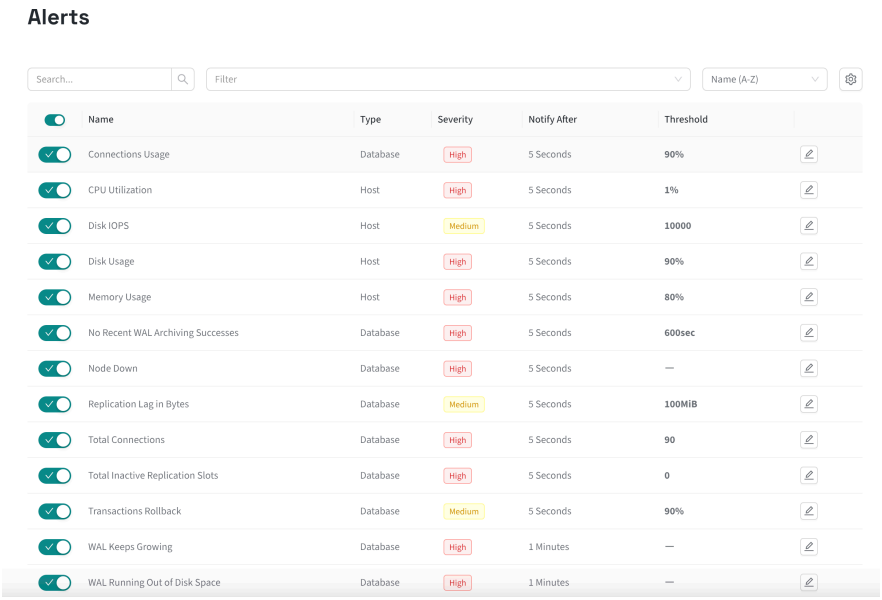


Fig. 4: Alerts

- Disk IOPS
- Disk Usage
- Memory Usage

!!!note - Any changes made to the alert configuration are saved automatically. - All the alerts are not available for external HM clusters due to the technical limitations of making them available.

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 UI, but currently, no Hybrid Manager 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

continues on next page

Table 1 – continued from previous page

Term	Definition	Measure
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

Working with clusters

You can use HM to create database clusters. The database can use any of these Postgres types: EDB Postgres Advanced Server, EDB Postgres Extended Server, or PostgreSQL.

To learn about creating clusters, see *Creating a database cluster* .

Either as part of cluster creation or after your cluster is creating, you might need to perform some related tasks. For example, you might need to manage your clusters (editing, restoring, and deleting) or understand how they' re performing. You might also want to create a template to use as the basis for future clusters you create.

To learn about these cluster management tasks, see *Managing clusters* .

Creating a database cluster

You can create a database cluster from a template (if you have one) or create the new 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** *Step 2: Choose your path* 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 **Create Cluster** *Step 2: Choose your path* page 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.

- ref:***Cluster Info** <Cluster types to create>*
- ref:***Cluster Settings** <Cluster setup>*
- ref:***DB Configuration** <Database configuration parameters>* (optional)
- ref:***Additional Settings** <Backup and security settings>* (optional)
- ref:***Data Groups** <Creating 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 provision, 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.

- ref:**Single Node** <*Single-node cluster*> 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.
- ref:**High Availability** <*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.
- ref:**Advanced High Availability** <*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.
- ref:**Distributed High Availability** <*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 on the 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 this tab only for single-node and high-availability clusters. You set these options in the *Creating 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 the 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 three 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 in Hybrid Manager* for more information. Generally, with no `-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 logs (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 Write-Ahead Logs (WAL) files. This default gives best performance and resilience. If you deselect it, 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 (gp2 or gp3) and size of the database storage volume in GiB (default is 1).

Write-Ahead Logs Storage — Under **Write-Ahead Logs Storage**, you can select the WAL storage class (gp2 or gp3) and size of the WAL storage volume in GiB (default is 1).

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 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* for PSR and PGD cluster deployment is set to 64MB and 128MB respectively.

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 select 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.

Danger

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 secures the data encryption key created by the Hybrid Manager with a passphrase when you create a TDE-enabled cluster.
- If the administrator of your Hybrid Manager instance previously *enabled a key from a key management system* , 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 Group** 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 only 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 the *Distribution high availability* reference.

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 three 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 in Hybrid Manager* 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 logs (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 (gp2 or gp3) and size of the database storage volume in GiB (default is 1).

Write-Ahead Logs Storage — Under **Write-Ahead Logs Storage**, you can select the WAL storage class (gp2 or gp3) and size of the WAL storage volume in GiB (default is 1).

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 once you have selected **EDB Postgres Advanced Server** or **EDB Postgres Extended Server** as the database type and selected 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 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 select 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.

Danger

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 duplicated, 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 predefined template, if your project has any. Using a template saves time and ensures consistency across environments.

To create a template, see *Create cluster tempaltes* .

Starting points

You can start to create a cluster from a template from the **Choose Your Path** page or from **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:

- ref:*Cluster types to create <Cluster types to create>*
- ref:*Cluster setup <Cluster setup>*
- ref:*Database configuration parameters <Database configuration parameters>*
- ref:*Backup and security settings <Backup and security settings>*
- ref:*Creating data groups <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 you can select.

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.

Managing clusters

Several management tasks are associated with clusters in the Hybrid Manager.

Cluster creation tasks

As part of the cluster creation process, you might want to:

- ref:*Create a cluster template <Create a database cluster template>* that you can use as the basis for new templates.
- ref:*Create a formula <Formulas for parameter values>* 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:

- ref:*Edit the cluster <Editing a cluster>* to make changes to most of its configuration.
- Change a cluster password.
- ref:*Delete a cluster <Deleting clusters>* .
- Upgrade a cluster. You can perform a minor upgrade or a major upgrade.
- ref:*Restore a deleted cluster <Restoring clusters>* .

The menu includes only the options the 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:

- ref:*Cluster types to create <Cluster types to create>*
- ref:*Cluster setup <Cluster setup>*
- ref:*Database configuration parameters <Database configuration parameters>*
- ref:*Backup and security settings <Backup and security settings>*
- ref:*Creating data groups <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 button.
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**:

- ref:**Cluster Info** <*Cluster types to create*>
- ref:**Cluster Settings** <*Cluster setup*>
- ref:**DB Configuration** <*Database configuration parameters*> (optional)
- ref:**Additional Settings** <*Backup and security settings*> (optional)
- ref:**Data Groups** <*Creating 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 the 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 resourcs:

- 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:

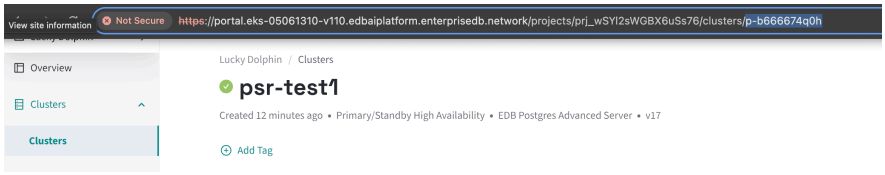


Fig. 5: Browse cluster

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, enter `Hardware Utilization` in the **Search** field. 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.

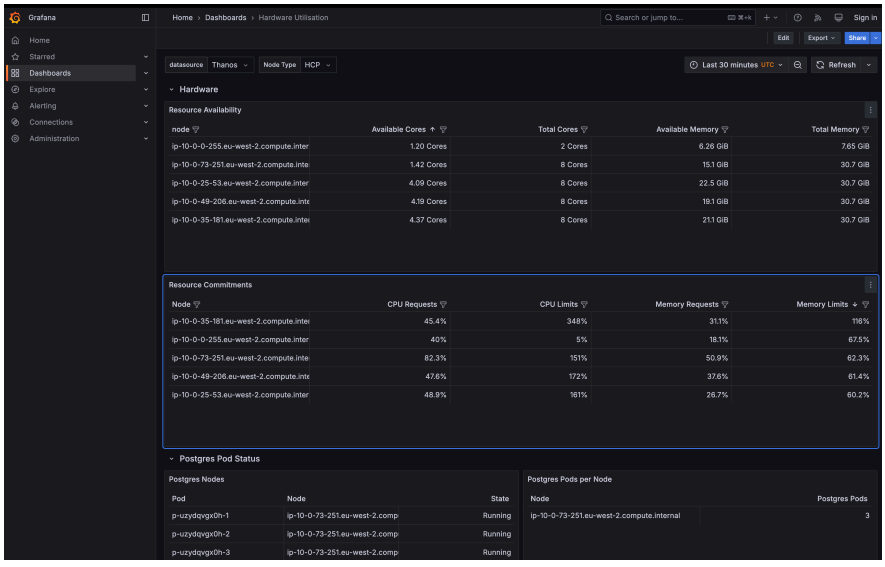


Fig. 6: Hardware Utilization dashboard

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

Requests, both CPU and memory, 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.

It does, however, 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

Limits, both CPU and memory, 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 UI, 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, *Grafana Hardware Utilization dashboard* the **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, and 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, and 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. Again, 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 `Pending` , 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, at the very end of the output, you should see something giving 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 us 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 may 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 and 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 deploying a cluster with LDAP configurations using the API, an LDAP server must be present in your environment.
- Before authentication 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. This will configure LDAP authentication.

Use the following steps to create or modify a cluster to use LDAP.

Creating a new cluster to use LDAP

Before using the REST API, make sure you have 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"
         }
       }
```

(continues on next page)

(continued from previous page)

```
    }
   }
 }}
```

Modify an existing cluster to use LDAP

For an existing cluster, use `PATCH` to configure it for LDAP authentication:

```
curl -X PATCH -H "x-access-key: ${KEY}" -H "Content-Type: application/json" --insecure -
↪s "https://portal.eks-05220612-main.edbaiplatform.enterprisedb.network/api/v1/projects/
↪prj_sXNNRhwj1YddHslR/clusters/p-lfdghq5gxe" -d {"projectId": "prj_sXNNRhwj1YddHslR",
↪"clusterId": "p-lfdghq5gxe",
"psr": {
  "ldapConfig": {
   "enabled": true,
   "config": {
    "hostname": "ldap-service.default.svc.cluster.local",
    "scheme": "ldap",
    "bindAuth": {
     "prefix": "uid=",
     "suffix": ",dc=example,dc=org"
    }
   }
  }
 }
}
```

Verification

Verify the `pg_hba.conf` file has been 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
- Example*:

```
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
- Example*:

```
postgresql:
  ldap:
    server: "openldap.default.svc.cluster.local"
    bindAuth:
      prefix: "uid="
      suffix: ",dc=example,dc=org"
```

LDAP over TLS

TLS can be used 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 will remain unencrypted unless SSL is configured.

References

https://www.postgresql.org/docs/current/auth-ldap.html

https://cloudnative-pg.io/documentation/preview/postgresql_conf/#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:

- ref:*Key terms and architecture overview <Key terms and architecture overview>* — A brief overview of the cluster.
- ref:*Connect the bucket to GenAI Builder <Connect the bucket to GenAI Builder>* — Connection information for the cluster.
- ref:*Monitoring and Migration Copilot <Monitoring and Migration Copilot>* — Monitoring information for the cluster.
- ref:*Diagnostics ("Query Diagnostics" and "Recommendations" tabs) <Diagnostics ("Query Diagnostics" and "Recommendations" tabs)>* — Query diagnostics information for the cluster.
- ref:*PGAIHM Backups and DR <PGAIHM Backups and DR>* — Backup information for the cluster.
- ref:*Enabling Data Catalogs on Hybrid Manager <Enabling Data Catalogs on Hybrid Manager>* — Logs for the cluster.
- ref:*Cluster Properties <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 `0` 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 for a cluster which can help to improve the Postgres cluster performance.

The Scorecard and Recommendation are directly related. A cluster will only score lower than Excellent if one or more recommendations are available. Once these recommendations are applied or dismissed the score will return to 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 two (Indexes and Statistics) are available at the moment.

Generally, grades are assigned on an A-F scale.

For Indexes:

- Excellent (A), if no recommendations were generated to tune indexes,
- Good (B), Fair (C) and Poor (D), depending on how many recommendations were generated.

For Statistics:

- Excellent (A), if no recommendations were generated to add statistics,
- Good (B), if at least one recommendation was generated.

Please note that:

- "Fail" (F) is not used for the time being since it is reserved for unhealthy cluster states that require immediate action.
- "Unknown" (U) is used when information to evaluate a cluster is unavailable.

For Indexes, the scores are computed by considering only primary nodes in the case of PSR and write leaders in the case of PGD/AHA. Furthermore, the score is related to the database in the PG cluster that is performing the worst (not an average over all databases created in such a cluster).

When selecting a specific category, text is displayed to help the user understand the relevance of these recommendations, beyond providing more 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”, the status of a recommendation (New, Ignored, Applied),
- “Database Name”,
- “Node ID”,
- “Size Increase” (for indexes), estimated index size,
- “Absolute Benefit” (for indexes), the absolute cost reduction that applying this index could produce,
- “Total Cost” (for indexes), the execution time plus plan time for running this query,
- “Cost Reduction” (for indexes), the 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), a rough estimation of the potential benefit of applying such statistic,
- “Recommended Time”, creation time for this recommendation,
- “Query”, text of the query to create an index or a statistic.

With the cog icon, it’s possible 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 Recommendations cannot be applied from the web interface; the user must manually copy and execute them in the terminal or an admin GUI.

The filters allow users to restrict data by:

- Database,
- Node,
- Status.

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 Recommendations

What do the “mark the recommendation as applied/ignored” buttons do?

You should use the ‘applied’ option to tell Hybrid Manager that you have applied the recommended change. If you mark a recommendation as applied, it will disappear from the list and when the page is next reloaded your score will increase commensurately. If the same recommendation is received again (suggesting the change was not actually applied or has been undone), the recommendation will reappear and the score will drop.

You should use the ‘ignored’ option to tell Hybrid Manager that you have reviewed the recommended change and decided it is not appropriate to apply it. If you mark a recommendation as ignored, it will disappear from the list and when the page is next reloaded your score will increase commensurately. If the same recommendation is received again it will be ignored and will not reappear.

Where does this data come from?

The Index and Statistics recommendations are sourced from the Query Advisor extension, as such this extension must be 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.

Upgrading on Hybrid Manager

Upgrades are an important part of database administration, whether it is a major or minor upgrade of your database.

- ref:*Postgres major version upgrades <Postgres major version upgrades>*
- ref:*Postgres minor version upgrades on HM <Postgres minor version upgrades on HM>*

Postgres major version upgrades

- ref:*Postgres major version upgrades on PGD with HM <Postgres major version upgrades on Hybrid Manager>*

Postgres major version upgrades on Hybrid Manager

Major version upgrades allow you to access new features, enhancements, and performance improvements introduced in the latest version of PostgreSQL. However, they may include changes that are not 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), users can manage major version upgrades at their 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**
>
> PG major version upgrades require a data restart. Downtime is expected.

While major upgrades offer significant benefits, they require careful preparation.

It is strongly recommended to perform a full backup of your cluster and validate a comprehensive recovery plan before proceeding. This ensures you can restore your data if any compatibility or operational issues arise during the upgrade process.

Prerequisites

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. Now check the cluster and database metrics and make sure everything is operating and all metrics are within the defined acceptable parameters.

Upgrade the PG major version of an existing instance/cluster created with HM

1. First, from the cluster's main page, select **Quick Action>Upgrade Cluster** and then choose **Major Upgrade**.
2. The **Upgrade Cluster — Major** dialogue window opens.
3. Select your preferred new major version image.
4. Review the upgrade path and confirm by selecting the **Continue** button.
5. The dialogue window now shows the specific package changes to be expected with the upgrade. Review these and select **Upgrade Cluster** if everything looks as correct.
6. The upgrade is applied automatically. You can then use the **Health Status**, **Monitoring**, and **Logs** tabs to monitor the process and/or 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 for the product in question to see if this is necessary in your case.

Rollback considerations

There is no automatic rollback available with this approach and performing a downgrade is not recommended. To undo the minor upgrade, restore from the latest backup in a new cluster running the preferred minor version and switching application workloads over to the new cluster. You can then delete the PG minor upgraded cluster or troubleshoot it further.

Postgres minor version upgrades on HM

Minor version upgrades use the same procedure regardless of the cluster type (Single Node, Primary/Standby, Distributed) and Postgres database type (PostgreSQL community, EDB Postgres Extended Server, 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 PG don't introduce new features or breaking changes, so it's recommended to adopt them as soon as they are released. Even if your PG cluster is running multiple minor versions behind, the recommendation is to adopt the latest minor version available.

Users have full control over when new versions are available with Hybrid Control Plane (HM). Once validated and published, new version images are available in HM's image library and available for upgrades on any existing cluster.

> **Note**
>
> PG minor version upgrades require a data restart. Minimal downtime is expected.

Although minor upgrades are ideally frequent and low risk, the recommended 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. Now check the cluster and database metrics and make sure everything is operating and all metrics are within the defined acceptable parameters.

Upgrade the PG minor version of an existing instance/cluster created with HM

1. First, from the cluster's main page, select **Quick Action>Upgrade Cluster** and then choose **Minor Upgrade**.
2. The **Upgrade Cluster — Minor** dialogue window opens.
3. Select your preferred new minor version image.
4. Review the upgrade path and confirm by selecting the **Continue** button.
5. The dialogue window now shows the specific package changes to be expected with the upgrade. Review these and select **Upgrade Cluster** if everything looks as correct.
6. The upgrade is applied automatically. You can then use the **Health Status**, **Monitoring**, and **Logs** tabs to monitor the process and/or 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 for the product in question to see if this is necessary in your case.

Rollback considerations

There is no automatic rollback available with this approach and performing a downgrade is not recommended. To undo the minor upgrade, restore from the latest backup in a new cluster running the preferred minor version and switching application workloads over to the new cluster. You can then delete the PG minor upgraded cluster or troubleshoot it further.

Migration with Hybrid Manager

How to migrate your database clusters.

- For migrations from Oracle, use the Migration Portal instance built into the Hybrid Manager to *manage and convert schemas for HM migrations* . For migrations from Postgres, use Migration Toolkit, an external tool, to manage the *schema migration* .
- Use the Data Migration Service to perform either a snapshot or streaming replication of a self-managed database, and migrate it to an HM-managed database.

See Migration Guides for end-to-end migration examples from Oracle, self-managed Postgres, AWS RDS and more.

Migration Portal

Migration Portal is a service that helps you assess, analyze, convert, and migrate Oracle database schemas to EDB Postgres Advanced Server. It is available as a web-based service or as an application in the Hybrid Manager. This section focusses on its usage as part of the Hybrid Manager.

Features

Some features within the Hybrid Manager deployment of Migration Portal are unique or function differently compared to the web-based version.

AI Copilot

!!!note

You must manually enable this feature in the Hybrid Manager Migration Portal, whereas in the web-based version it' s included out-of-the-box.

AI Copilot is an AI-driven chatbot tool, designed to assist you with issues that can occur during the schema assessment and conversion in the 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.

Unlike the Migration Portal, the AI Copilot is not enabled by default in the Hybrid Manager.

- To use the AI Copilot for Migration Portal queries, request the Hybrid Manager administrator to *enable and configure it with the AI/LLM of your organization* .
- See AI Copilot for more information.

SPL Check

!!!note

This function is included in the Hybrid Manager Migration Portal. The web-based Migration Portal will include it in a future release as well.

SPL Check is a code analysis tool that identifies potential runtime errors in your Oracle procedural code. It' s an important part of the assessment process when you' re migrating to EDB Postgres Advanced Server (EPAS).

All schemas that are assessed with the Hybrid Manager version of Migration Portal are analyzed by SPL Check. This function, which is based on the EDB SPL Check extension, detects execution time errors within `FUNCTION` , `PROCEDURE` , `PACKAGE BODY` ,`TYPE BODY` , and `TRIGGER` object types.

After the schemas are assessed, any objects that don' t pass the check are marked as *failed* in the Migration Portal project, alongside other compatibility issues. This lets you resolve these problems before importing your objects into the destination database, preventing runtime errors later on.

You can distinguish SPL Check errors from other incompatibilities because they are prefixed with `[spl_check]` . After you modify an affected object to fix an issue, select **Reassess** to rerun the check and confirm your edits have resolved the problem.

Project collaboration

!!!note

This function is included in the Hybrid Manager Migration Portal. The web-based Migration Portal will include it in a future release as well.

Migration Portal supports collaborative DDL editing, enabling multiple users in the same Hybrid Manager project to work on the same schema file simultaneously. If another user (user B) modifies a DDL you' re also editing, your changes (user A) will trigger a conflict resolution view. This view displays user B' s update in an **Update** view and your own edit in a **Target** view. You can then choose to **Use Updated** to adopt user B' s fix or **Reassess** to apply and save your own changes, overwriting the previous update. This ensures efficient, concurrent schema compatibility adjustments.

Accessing Migration Portal

The Migration Portal application of the Hybrid Manager offers two different workspaces, that are accessed differently depending on how you want to use it.

Project-scoped workspace vs. user' s workspace

When you deploy the Hybrid Manager, it automatically includes a built-in instance of Migration Portal. This instance, however, has two type of workspaces that vary depending on how you want to use it. They are also accessed differently.

- The *project-scoped workspace* in the Migration Portal is where you manage Migration Portal (MP) projects created with the Hybrid Manager Console for schemas loaded by the *Agent Studio in Hybrid Manager* .

This shared portal environment aligns your MP projects directly with your Hybrid Manager (HM) projects. Here, you' ll see all MP projects created by both yourself and other users who are part of the same HM project. You won' t, however, see MP 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 is not a shared environment, only your user will have access to the Migration Portal projects created in this workspace.

This separation allows the Migration Portal to cater to different types and scales of migration efforts: organized, large-scale, enterprise-driven projects (project-scoped) versus more flexible, smaller-scale, or individual-driven migrations (user' s workspace).

Accessing

A Migration Portal has two type of workspaces. How you access them depends on how you want to use the Migration Portal.

- To create and manage projects for schemas loaded into the Hybrid Manager by the Agent, see *Assessing schemas for HM projects* . This uses the project-scoped access to MP.

- To manually upload externally-provided DDL files and assess schemas with the Migration Portal, see *Assessing schemas for external projects* . This employs the user' s workspace access to MP.

Assessing schemas for HM projects

Use the collaborative Project-scoped workspace vs. user's workspace of the Migration Portal to convert and migrate schemas in the context of a database migration. The Hybrid Manager provides an integration that lets you use the HM Console to create a Migration Portal project where you can easily add source database schemas pre-loaded by the *Agent Studio in Hybrid Manager* while you are performing a database migration.

Only Oracle databases include an integration of the Migration Portal into a migration workflow, as their schemas require compatibility assessment and oftentimes conversion before they are migrated to the destination database.

To upload and assess externally-provided DDL schema files, *use the user's workspace* .

Prerequisites

- You *registered an Oracle database* with the Hybrid Manager.
- **Optional:** The administrator of Hybrid Manager has *enabled and configured the AI Copilot with the AI/LLM of your organization* to facilitate resolving schema incompatibilities in the Migration Portal.

Accessing Migration Portal

Use the Migration Portal integration to assess and migrate schemas:

1. In the **Estate** page, select the **Migrations** tab.
2. Select the **Database Name** of the database you want to migrate to open the database' s detailed view.

!!!Note

Only available for Oracle databases.

If you already created a schema migration project for one, several or all schemas, the Console displays a hyperlink with the name of the project in the **MP Project** column. This opens 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 MP Project** button or the schema-scoped **+** sign. This starts a new workflow to *create a Migration Portal* project.

Creating a Migration Portal project

You can create Migration Portal projects from the Hybrid Manager.

1. In the **Estate** page, select the **Migrations** tab.
2. Select the **Database Name** of the database you want to migrate to open the database' s detailed view.
3. In the detailed view of the database, select the **Schemas** tab.
4. Select **+ New MP Project** or the the plus sign next to one of the schemas you want to migrate.
5. In the pop-up window, enter the details for the Migration Portal project that will host the assessment, conversion and migration of the schemas and select **Continue**.
6. Select all the schemas you want to add to this Migration Portal project. You can only select schemas that the Agent has ingested successfully and schemas that are not already part of a different MP project.

Note: You can' t add new schemas to the existing project later, but you can use a different project to migrate other schemas to the same destination database.

7. Select **Create New Project**.
8. In the background, this creates a Migration Portal project and loads the source schemas into it for analysis.

After the Hybrid Manager finishes loading the schemas and creating a project in the Migration Portal, a hyperlink is added with the name of the project. Select the hyperlink to open the Migration Portal.

Working in the Migration Portal

The selected schemas have been uploaded and assessed by the Migration Portal. Use the AI Copilot or QuickHelp to help you resolve incompatibilities.

1. After you have resolved 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

Review Schema integrity and performance considerations to learn more about why constraints must be disabled and to review other considerations.

4. In the **Connect to your cluster** page, enter the host name of the database cluster you want to use as a destination.

!!!note

If you are migrating to an HM-managed database cluster: You can find the Read/Write Host and other connection data at **Clusters** > select the cluster > **Connect**.

5. Select **Next** and wait until the schemas have been imported successfully.
6. Connect to the target database cluster and ensure the schemas have been migrated. You can now close the Migration Portal.

Switching to user's workspace

While you are working on schemas in the Migration Portal, you can switch to the Project-scoped workspace vs. user's workspace for manual DDL upload.

For this, select **Switch to user's workspace**.

However, this workspace won't display the schemas you selected during project creation, and should only serve 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 <https://www.enterprisedb.com/docs/migration_portal/latest/04_mp_migrating_database/03_mp_schema_migration/>`_ of the converted schemas to a destination database.
- `Migration Portal documentation <https://www.enterprisedb.com/docs/migration_portal/latest/03_mp_using_portal/>`_ 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 the Migration Portal to assess, convert and migrate externaly-provided DDL files for Oracle database migrations.

To access the project-scoped workspace of the Migration Portal in the scope of Hybrid Manager migrations that leverage the Agent, *access the Migration Portal from the Oracle databases' detailed view* .

Prerequisites

- **Optional:** The administrator of Hybrid Manager has *enabled and configured the AI Copilot with the AI/LLM of your organization* to facilitate resolving schema incompatibilities in the Migration Portal.

Accessing Migration Portal

Use the Launchpad app to access the Migration Portal

1. *Access the Launchpad* .
2. Hover over the **Migration Portal** tile to reveal **View Service** and select the revealed button. This opens a new tab that takes you to the Migration Portal.

Next Steps

Use Migration Portal to:

- Resolve any incompatibilities and complete the schema conversion.
- ```
  `Perform an online or offline migration <https://www.enterprisedb.com/docs/migration_
  ↪portal/latest/04_mp_migrating_database/03_mp_schema_migration/>`_  of the␣
  ↪converted schemas to a destination database.
  ```

See the Migration Portal documentation for recommendations and guideline on how to use the portal.

Migration Walkthroughs

This section collects step-by-step guides that will help you perform migrations with the Hybrid Control Plane.

- ref:*Migrating a self-managed Oracle database to an HM-managed database <Migrate your Oracle database to the Hybrid Manager>*
- ref:*Migrating a self-managed Postgres database to an HM-managed database <Migrate your self-managed Postgres database to the PG AI Hybrid Manager>*
- ref:*Migrating a cloud service provider-managed AWS RDS database to an HM-managed database <Monitoring AWS RDS deployments on AWS>*
- ref:*Using the PG AI Agent for Oracle Migration Readiness <Using the Agent for Oracle Migration Readiness>*

Migrate your Oracle database to the Hybrid Manager

Migrating a self-managed Oracle database to a HM-managed cluster involves several procedures. This guide collects all procedures in a single place, with links to the instructions for each procedure.
  ```

Register your Oracle database

Registering the Oracle database with the Agent provides a migration assessment, as well as DDL schema extraction to be used by the Migration Portal.

Refer to *Using the Agent for Oracle Migration Readiness* to register your Oracle database.

Prepare the Oracle source databases

You prepared your Oracle database to allow the Agent to fetch monitoring metrics, migration statistics and schemas. Now you'll have to prepare it to allow the DMS Reader to fetch database data and perform migrations. See Preparing Oracle source databases for more information.

Create an HM-managed destination

When migrating a database to an HM-managed database cluster, we recommend choosing EDB Postgres Advanced Server due to its superior Oracle compatibility. If your schema has minimal compatibility issues, you can also opt for PostgreSQL or EDB Postgres Extended Servers. You'll choose destination this database later when performing the schema migration, as this cluster will serve as a migration destination.

Creating a database cluster walks you through creating an HM-managed cluster.

Prepare and import the source schema

Prepare the source schema and import it to the destination database. This step is not dependent on the previous steps, so it can be performed at any time before creating a migration. You must ensure the schema is converted and migrated to the destination database before you *create a data migration* .

There are many tools EDB offers to prepare the schema, but for migrations from Oracle, the Hybrid Manager includes a Migration Portal integration. Refer to *Assessing schemas for HM projects* to use the integration to import the schema.

Deploy the EDB DMS Reader

Deploying the EDB DMS Reader on an a machine that has access to your source database server allows the Reader to perform a connection between the Oracle database and the Hybrid Manager so it's recognized as a source for migration.

See *Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator* for more information.

Configure the EDB DMS Reader

To configure the EDB DMS Reader so the Hybrid Manager has access to source database data, you must configure the reader script to allow it to connect to both the source and destination databases for migration.

Get credentials from the Hybrid Manager **Console** and configure the reader as explained in Configuring and running Reader .

Run the EDB DMS Reader

Run the EDB DMS Reader to start reading the source database. This doesn't perform a migration yet, only ensures the DMS has access and can read the source database.

Create a migration

After you have configured the EDB DMS Reader to establish connections to the Hybrid Manager, create a new migration in the Console. A migration establishes a sync between the source database and a destination database with the help of EDB DMS.

See *Creating a migration* for instructions.

Complete the migration

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

If you selected the **Snapshot + Streaming** option during Migration creation, the EDB DMS Reader will continue to stream any data updates performed on the source database to the destination database until you mark the migration as completed.

See Stop the streaming process for instructions.

Apply constraints

Before migrating data with the EDB Data Migration Service, you prepared and imported the schema of your source database while excluding constraints.

Now, connect to the destination database and re-apply the constraints that were excluded from the schema and data migration.

Verify migration

Compare the source and destination databases to verify that all data was migrated.

You can use LiveCompare .

Migrate your self-managed Postgres database to the PG AI Hybrid Manager

To migrate a self-managed Postgres database to a database cluster created with the PG AI Hybrid Manager:

1. *Deploy the EDB DMS Reader*
2. *Create a destination database cluster managed by the PG AI Hybrid Manager*
3. *Prepare the Postgres source databases so the EDB DMS Reader can access data*
4. *Configure the EDB DMS Reader*
5. *Run the EDB DMS Reader*
6. *Prepare the source schema and import it to the target database*
7. *Create a migration*

Deploy the EDB DMS Reader

Deploying the EDB DMS Reader on an a machine that has access to your database server allows the Reader to perform a connection between the Postgres database and the PG AI Hybrid Manager so it is recognized as a source for migration.

See *Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator* for more information.

Create a destination database cluster managed by the PG AI Hybrid Manager

When you migrate a database to the PG AI Hybrid Manager, you need to create an EDB Postgres Advanced Server database cluster. You' ll target this database later when performing the *schema migration* , as this cluster will serve as a migration destination.

For more information, see *Creating a database cluster* .

Prepare the Postgres source databases so the EDB DMS Reader can access data

See *Preparing Postgres source databases* for more information.

Configure the EDB DMS Reader

To configure the EDB DMS Reader so the PG AI Hybrid Manager has access to source database data, you need to fill in some values in the reader script that give it access to both the source and target databases for migration.

1. Obtain credentials from the **PG AI Hybrid Manager Console** as explained in Getting credentials .
2. Perform the steps in the Configuring the reader section, but ensure you set the following variables set in the reader script:

```
export CLOUD_PROVIDER=appliance
export RW_SERVICE_HOST=<transporter-rw-service-host>
```

!!!note

Set `RW_SERVICE_HOST` to the domain name or host associated with the PG AI Hybrid Manager ingress.

Derive `RW_SERVICE_HOST` from the `TRANSPORTER_RW_SERVICE_DOMAIN_NAME` you assigned to your instance via the `values.yaml` file during installation and add `/transporter` .

Alternatively, derive `RW_SERVICE_HOST` from the URL you use to access the PG AI Console. For example, if the URL is `https://portal-transporter.foo.network` , get the domain name, and add `/transporter` at the end.

In this example, you' d have to set:

`export RW_SERVICE_HOST=transporter.foo.network/transporter`

Run the EDB DMS Reader

Execute the EDB DMS Reader to start reading the source database. This doesn' t perform a migration yet, only ensures the DMS has access and can read the source database.

Prepare the source schema and import it to the target database

This step is not dependent on the previous steps, so it can be performed at any time before creating a migration. You must ensure the schema is converted and migrated to the target database before you *create a data migration* .

There are many tools EDB offers to prepare the schema. For migrations from self-managed Postgres instances, we recommend using EDB Migration Toolkit (MTK). MTK's offline migration capability provides an easy way to extract a database' s schema and separate constraints.

!!!note

Ensure you exclude foreign key, check, and exclusion constraints from the DDL before importing the schema to the target database. Review Schema integrity and performance considerations to learn more about why constraints must be disabled and to review other considerations.

Create a migration

In the PG AI Console:

1. *Create a migration* .
2. Verify that the migration was successful. You can perform spot checks of tables, rows, schemas, etc., with psql, or compare both databases with LiveCompare .

3. If you selected a **Snapshot + Replication** migration, mark the migration as completed . Otherwise, skip this step.
4. Re-apply any excluded constraints .

Migrate your AWS RDS Postgres database to the Hybrid Control Plane

AWS RDS database clusters are CSP-managed databases.

To migrate data from an AWS RDS Postgres database to an HM-managed database:

1. *Deploy the Beacon Agent to the AWS RDS database*
2. *Prepare the AWS RDS source databases*
3. *Create a destination database cluster managed by the Hybrid Control Plane*
4. *Import the schema to the destination database*
5. *Deploy the EDB DMS Reader to an auxiliary machine*
6. *Configure the EDB DMS Reader, so it can access source data*
7. *Create a migration*
8. *Complete the migration*
9. *Apply constraints*
10. *Verify the migration*

Deploy the Beacon Agent to the AWS RDS database

Deploying the Beacon agent on your database server allows the Hybrid Control Plane to generate observability and monitoring data. While this step isn't strictly necessary for migrating your databases, it offers the significant benefit of providing a single, unified view of all your database instances, including those external to the Hybrid Control Plane.

For configuration instructions, see *Monitoring AWS RDS deployments on AWS* . Access the **Estate** to see an overview of the RDS instance.

Decide which database or databases you want to migrate to the Hybrid Control Plane.

Prepare the AWS RDS source databases

Before migrating data from an AWS RDS database using the EDB Data Migration Service, you must prepare the source database to allow the EDB DMS Reader to connect to the source and perfom replication.

See *Preparing AWS RDS source databases* for more information on how to enable logical replication and create a user for migration.

Create a destination database cluster managed by the Hybrid Control Plane

You can either use an existing HM-managed cluster or create a new cluster to use it as destination for the migration.

For more information, see *Creating a database cluster* .

Import the schema to the destination database

For migrations from RDS databases, we recommend using Migration Toolkit in offline mode for the export of schemas and handling constraints.

See Preparing schema for instructions on how to export your source schema, handle constraints and import the schema to your destination database.

Deploy the EDB DMS Reader to an auxiliary machine

Deploy the EDB DMS Reader on a machine that has access to your RDS database. We will call this the auxiliary machine.

!!!note

One way to ensure your auxiliary machine has access to the RDS instance is to add an inbound rule to the RDS instance. On the VPC security group assigned to the RDS instance, add an **Inbound rule** with the **Type** PostgreSQL and the IP address of the auxiliary machine.

See Installing EDB Data Migration Service Reader for more information.

Configure and start the EDB DMS Reader

To configure and run the EDB DMS Reader on your auxiliary machine, you need to obtain HM migration credentials, and then provide the EDB DMS Reader with access to this credentials and other settings.

This will allow the EDB DMS Reader to have access the source database, and connect it to the HM migration service.

See Configuring and running the EDB DMS Reader for more information.

!!!note

For RDS databases, ensure the `DBZ_DATABASES_0__HOSTNAME` value points to the RDS endpoint you can find under the **Connectivity & security** in the AWS Console. A redacted example is `foo_database.cb7asw1xxxxx.eu-west-0.rds.amazonaws.com` .

Create a migration

After you have configured the EDB DMS Reader to establish connections to the HM, create a new migration in the HM Portal. A migration establishes a sync between the source database and a destination database with the help of EDB DMS.

See *Creating migration credentials* for instructions.

Complete the migration

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

If you selected the **Snapshot + Streaming** option during Migration creation, the EDB DMS Reader will continue to stream any data updates performed on the source database to the target database until you mark the migration as completed.

See Stop the streaming process for instructions.

Apply constraints

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 target database and re-apply the constraints that were excluded from the schema and data migration.

Verify migration

Compare the source and target databases to verify that the all data was migrated.

You can use LiveCompare .

Using the Agent for Oracle Migration Readiness

Before starting a migration from an Oracle database source, connect it to the Hybrid Manager with the Agent. Registering the Oracle database with the Agent gives you 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 is available on the *Migrations* 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 Console that will help you perform migrations. These resources are available when you select a specific Oracle database from the **Estate** page > **Migrations** tab.
- Automatic schema DDL extraction that you can leverage to perform schema migrations with the built-in Migration Portal.

Prepare the Oracle database

The Agent needs an Oracle user with access and necessary permissions to the source database to assess an Oracle database, extract schemas and other information needed for the migration.

See *Prepare your Oracle instance* for instructions.

Create a machine user

The Agent requires a machine user to ingest data into the Hybrid Manager.

See *Creating a machine user* for instructions.

Install the Agent

Install the Agent package on a machine that has access to the Oracle database.

Configure the Agent

Configure the Agent to allow your Oracle database to connect to your Hybrid Manager and fetch migration data from the source database, as well as schema DDL.

Run the Agent

Once the Agent has been installed and configured, you can proceed to *Running the Agent* and *running it as a service* .

To maintain an uninterrupted connection to Hybrid Manager, you should run the Agent as a service. While manual execution can sustain a connection as long as the host machine is operational, running the Agent as a service ensures automatic system startup and restarts after unforeseen shutdowns.

If you don't set up the agent as a service, it will only perform assessments and stream schema data when you manually run it. Furthermore, by default, your database will disappear from the Hybrid Manager's **Estate** 72 hours after the agent stops streaming.

!!!important

The Agent detects schema changes to tables and columns while running. However, changes to views or procedures won't be detected until you restart the Agent.

Next steps

Once your Agent is running, refer to *Migrate your Oracle database to the Hybrid Manager* for next steps.

Migrating databases

Use **Migrate** options to:

- Add and manage migration database *Key links and learning resources* .
- Manage migration database *Destinations* .
- Perform database *Migrations* .
- Get project-scoped *Credentials* for your database migrations.

Sources

Sources are databases you connected to the Hybrid Manager using the Data Migration Service or DMS so you can use them as sources for migrations. For a database to appear in this view, you must install, configure, and run the *Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator* .

This view displays all the configured sources in the project.

Each source's view shows the database name, database type (Postgres or Oracle), `cdcreader` ID, the current *Verify InferenceServices 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 Reaader 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 Reader 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 Reader can't establish a connection to the externally managed database. This can be due to incorrect connection details (host, port, database name, username, password) or network issues.
Error	Unspecified problem. Investigate *cdcreader* logs to understand the specific cause.

!!!note

You can't use a source or destination that's part of an active migration (in any state, including running or paused) for another migration. The existing migration must be completed or deleted before you can reuse the source or destination.

Destinations

Destinations are externally managed databases you connected to the Hybrid Manager using the Data Migration Service or DMS so you can use them as destinations for migrations. For a database to appear in this view, you must install and configure the *EDB DMS Writer* .

This view displays all the configured destinations in the project.

Each destination's view shows the database name, database type, `cdcwriter` ID, the current *Verify InferenceServices 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 Writer 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 Writer 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 Writer can't establish a connection to the externally managed database. This can be due to incorrect connection details (host, port, database name, username, password) or network issues.
Error	Unspecified problem. Investigate HM Console and *cdcwriter* logs to understand the specific cause.

!!!note

You can't use a source or destination that's part of an active migration (in any state, including running or paused) for another migration. The existing migration must be completed or deleted before you can reuse the source or destination.

Migrations

Migrations are processes you can start in a project of the Hybrid Manager to move data from one database to another. These processes include migrating an external Oracle database to an HM-managed cluster and to an externally managed Postgres instance, among others.

Viewing existing migrations

After creating one or several migrations, this view displays all the migrations in the project in a table format. The table contains the columns:

- **Name** — Name you assigned to a migration.
- **Source** — Name of the source, configured through the **Sources** page.
- **Destination** — Name of the destination, either a database in an HM-managed cluster or a destination configured through the **Destinations** page.
- **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 Reader/Writer connects to the source and/or target databases.
Running	The migration is active and data is being transferred.
Pausing	The migration is being temporarily stopped.
Paused	The migration was temporarily stopped and can be resumed.
Error	Unspecified problem. Under **Migrations** , select the name of the migration to open the details page. Look at the **Error log** section for more information about the cause.
Completed	The migration finished successfully.
Failed	The migration encountered an error and could not be completed. Investigate HM Console logs to understand the specific cause.
Deleting	The migration is being terminated.
Deleted	The migration was successfully terminated and removed.

!!!note

You can't use a source or destination that's part of an active migration (in any state, including running or paused) for another migration. The existing migration must be completed or deleted before you can reuse the source or destination.

Creating a migration

Select **Create Migration** to create migrations. See *Creating migration credentials* for details.

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. This page contains additional information on the status of the migration, type of migration, source and destination database information, and error logs that can help you to troubleshoot.

If you selected the **Snapshot + Streaming** option when *Creating a migration* , you can use **Mark as Completed** to stop the streaming procedure.

Credentials

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

The **Credentials** view shows all the credentials in the project in a table format. The entry for each credential displays its name, status, when it was created, and expiry date.

Creating migration credentials

See Getting credentials for instructions on how 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`

Assessing Migrations

One of the main features of the Hybrid Manager is that it facilitates database migrations.

The **Migrations** tab that is accesible through the **Estate** page displays a list of external databases you have registered with the *Agent Studio in Hybrid Manager* .

This is your first stop when migrating external databases to the Hybrid Manager.

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 *Step 7: Update the cluster for compatibility* 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 Agent's *Parameter reference* for more information.

Version: Database version. See *Step 7: Update the cluster for compatibility* for currently supported versions.

of Schemas: Number of schemas in the assessed database that the DSN user has access to and, contains at least one table on which the DSN user has `SELECT` privileges. The DSN user is the source database user you specified during the configuration of the Agent.

!!!note s

- 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, but a known issue can cause it to be included when the DDL extractor creates temporary tables during extraction.
- The Agent detects schema changes to tables and columns while running. However, changes to views or procedures won't be detected until you restart the Agent.

or Cores: Number of cores allocated to the Oracle database.

Complexity: Level of complexity to migrate the database to the Hybrid Manager. Possible values are Very Easy, Easy, Medium, Hard, and Very Hard.

- **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 will need to be addressed 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: Level of effort, in 8-hour working days, required to convert the schema and migrate data to the Hybrid Manager. 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.

Other columns

You can use the gear icon to display additional columns:

Last assessed: Period of time that has transcurred since the database was last assessed to determine if the analysis is up-to-date.

Management: Shows whether the database is managed by yourself (self-managed), by a clud service provider or by the Hybrid Manager.

Project Name: Project that was used for the configuration of the agent. It is the project you assign to an agent in `agent.project_id`. See the Agent's *Parameter reference* for more information.

Tags: Labels assigned to the migration of a specific database in `databases.tags`. See the Agent's *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, etc.

Viewing and filtering the migrations table view

The **Estate** > **Migration** tab displays a table-based analysis of all databases that you can migrate with the Hybrid Manager after registering it with the 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 also alphabetically.
- Choose how often to refresh the table with the lightning icon. If disabled, the table won't be updated unless you manually refresh it. If **Auto refresh** is enabled, it will update the table 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 *Step 7: Update the cluster for compatibility* to understand which databases are supported for migration assessment.
- See *Data security and sovereignity* to understand how your data is stored by the Agent.

Detailed view

When you select a database from the **Estate** page > **Migrations** tab, a detailed view opens. This detailed view includes functionality that you can leverage to perform migrations from external databases.

Schemas

The **Schemas** tab offers an overview of the schemas found in your connected database. Here, you'll find summary metrics for the database, along with a list of schemas where you have 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. Additionally, the **Schemas** tab allows you to *create new Migration Portal projects* directly from the extracted schemas.

!!!important

The Agent detects schema changes to tables and columns while running. However, changes to views or procedures won't be detected until you restart the Agent.

Summary cards

- Total schemas for the selected database.
- Total tables for the selected database.
- Complexity overview for all schemas, color-coded by complexity (very easy, easy, medium, hard, very hard).
- Level of effort, color-coded by day ranges (0-5, 6-15, 16-25, 26-40, 40+ days), defined by 8-hours working days.

Schema table

- **Name:** Name of the schema.
- **Latest Assessment:** Status of the latest assessment. Possible values are Not started, In progress, Complete and Error.
- **Last Assessed:** Timestamp corresponding to the last time the schema was assessed. The assessment will be automatically regenerated or updated if the Agent discovers that a new schema is available or if an existing one has been updated.
- **# of Tables:** Number of tables in the assessed schema.
- **# of Objects:** Number of objects in the assessed schema.
- **Complexity:** 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:** Indicates how many days it will take to migrate a specific schema, in 8-hour working days, based on a detailed, object-level analysis.
- **MP Project:** 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, the Hybrid Manager creates a Migration Portal project in the background. You can access this project at a later time through the *Customizing Migration Portal secrets for secure internal communication* 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: 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 are EPAS 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: 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 are EPAS-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: lists all Oracle database features in use within the database.

Incompatible PL/SQL features in objects: lists all Oracle and non-Oracle incompatible PL/SQL features while displaying the schema they are in, the object type (procedure, function, etc.), the object name, the incompatible feature name, how often the feature appear and the migration complexity.

Migration

Helps you perform migration-related tasks.

Create a Destination: allows you to create a new cluster that will serve as a destination for your migration. See *Creating a database cluster* for more guidance.

Migrate Schema(s): 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 let's you connect to the destination cluster to import the converted schema.

Migrate Data: leverages the Hybrid Manager's Data Migration Service to perform data migrations with CDC technology. Before you perform migrations with this function, ensure you have prepared your database, exported your schema, installed and configured the EDB DMS Reader (and Writer). See *Migrate your Oracle database to the PG AI Hybrid Manager* for a chronological overview of the steps you must take to migrate from an Oracle database.

Compatibility

Currently, the Hybrid Manager supports the assessment of these **Oracle** databases:

- Oracle 11g
- Oracle 12c
- Oracle 18c
- Oracle 19c

Later versions (e.g., 21c, 23c) might also be assessed, but these versions have not been certified yet.

Postgres databases are also displayed in the **Estate > Migrations** view, however, assessments aren't supported yet.

Asset Management

The Asset Management section of the Hybrid Manager 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 Hybrid Manager Postgres and Model Assets

- ref:*Asset Library in Hybrid Manager <Asset Library in Hybrid Manager>*

The main page for managing assets in the Hybrid Manager Console, where you can view and manage container images and knowledge bases.

- ref:*Sync HM container images to your private registry <Sync HM container images to your private registry>*

A detailed guide on exploring and managing container images in the Asset Library, including searching, filtering, and viewing image details.

- ref:*Manage Repositories <Manage Repositories>*

How to control the visibility of repositories and images in the Asset Library by managing repository registrations and rules.

- ref:*Knowledge Bases <Knowledge Bases>*

A guide on managing AI models in the Asset Library, including creating and managing knowledge bases that contain AI models for use in clusters.

The Asset Library

The Asset Library is where you manage the assets, Container images and models, that are available for use in your Hybrid Manager clusters.

Accessing the Asset Library

To access the Asset Library, click the **Asset Library** tab at the top of the Hybrid Manager Console.

The 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 **Knowledge Base**. Each section allows you to view and manage the assets available for use in your Hybrid Manager 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 Hybrid Manager 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. Read the *Sync HM container images to your private registry* guide for more information on how to use this tab.

Knowledge Bases

A model in the Knowledge Bases is an AI model that can be used in Hybrid Manager. These models can be trained on specific datasets and can be used to perform various tasks, such as natural language processing, image recognition, and more. Read the *Knowledge Bases* guide for more information on how to use this tab.

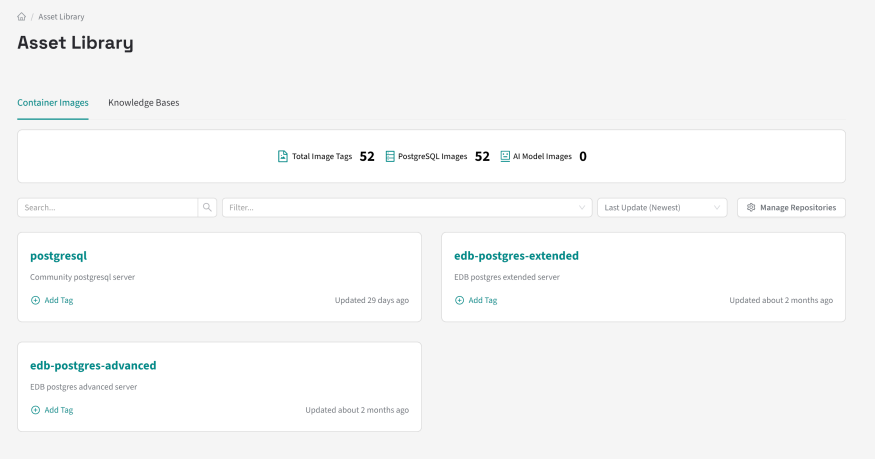


Fig. 7: Asset Library Page

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 clusters.

Container Images

A Container Image is a configuration of the database that includes various extensions and each image is identified by it's 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 Hybrid Manager images would have the image name `postgresql` and image tag `17.4-2503191025` 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). A further postfix of `-full` indicates that this image includes all the extensions available in the Hybrid Manager Postgres distribution.

Note that this tag format is specific to the official Hybrid Manager 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, while the image tag is used to identify the specific version of the image.

Image Statistics

The first part of the Container Images page is a summary of the number of images available in the library, broken down 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.

The page then moves on to the main content of the Container Images tab.

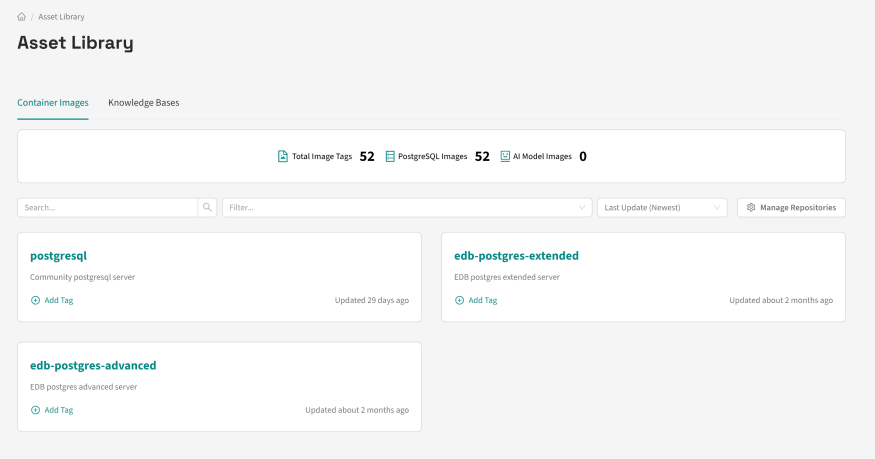


Fig. 8: Container Images Page

Container Images : Search and Filter Bar

In this view, the **Search** field allows you to enter a search term that is contained within the name or description of image repository. The **Filter** field allows you to filter the repositories by the type of repository (Postgres or AI Model).

The **Sort** field allows the display repositories to be sorted by the name of the repository (ascending or descending), by description (ascending or descending), by repository type (ascending or descending) or by the date the repository was updated (ascending or descending).

Finally on the search and filter bar, the **Manage Repositories** button allows you to manage the repositories that are available in the library. This will open a dialog that allows you to add, edit or delete repositories. This takes you to the **Manage Repositories** page, where you can manage the repositories and images that are available in the Asset Library. For more information on how to manage repositories, see the *Manage Repositories* guide.

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**: Hybrid Manager tags associated with this image repository. If no tags are associated with the image repository, this field will be show **Add Tag**.
- **Updated**: The number of days since the image repository was last updated.

Selecting the name of an image repository will take you 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**. There is also an **Add Tag** button that allows you to add a Hybrid Manager tag to the image repository. Any existing tags will be displayed before the button, and you can click on a tag to view the details of that tag.

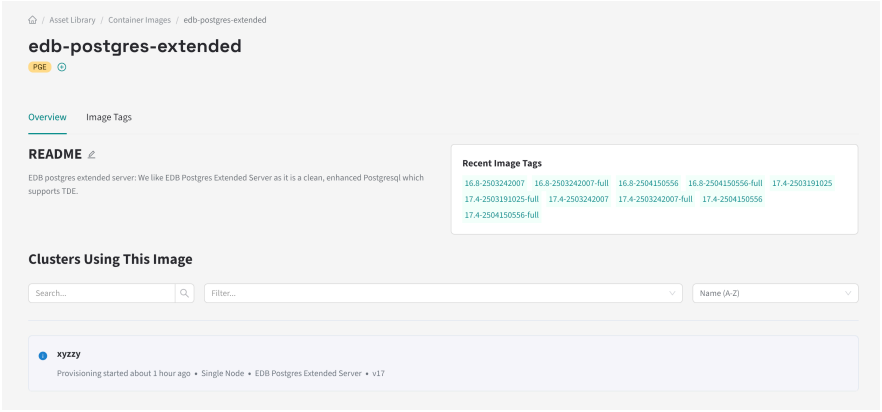


Fig. 9: Image Details Page - Overview Tab

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 any other details that may be useful.
- **Recent Image Tags**: The ten most recently created image tags for this image repository. Each tag is selectable and will take you to the Image Tag Details page for that tag.

Below the README and Recent Image Tags, there is a section for **Clusters Using This Image**. This 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" will be displayed.

To aid in searching for clusters, a search and filter bar is provided. The **Search** field allows you to enter a search term that is contained within the name or description of the cluster. The **Filter** field allows you to filter the clusters by their attributes (Cluster type, Cluster architecture, Postgres flavour and Postgres Version). The **Sort** field allows the display of clusters to be sorted by name (ascending or descending), by description (ascending or descending), or by the date the cluster was created (ascending or descending).

Selecting a cluster name will take you to the **Cluster Details** Overview page for that cluster.

Image Tags Tab

The **Image Tags** tab displays a list of all the image tags associated with this image repository.

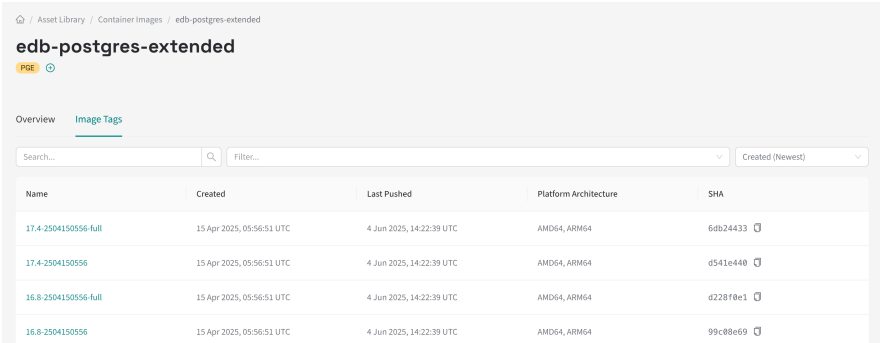


Fig. 10: Image Details Page - Image Tags Tab

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, such as X86, AMD64, ARM64, PPC64LE or S390X.
- **SHA**: The SHA256 hash of the image, which is a unique identifier for the image. It can be used to verify the image against other repositories. The last 8 digits of the hash are displayed and the full hash is available on hovering over those digits. The full hash can also be copied to the clipboard by clicking the copy icon next to the displayed hash.

To navigate through the list of image tags, you can use the search and filter bar at the top of the table. The **Search** field allows you to enter a search term that is contained within the name of the image tag. The **Filter** field allows you to filter the image tags by their attributes (EDB Official Release, Location Available, Platform Architecture). The **Sort** field allows the display of image tags to be sorted by version (ascending or descending), by created date (ascending or descending), by last pushed date (ascending or descending).

Selecting the name of an image tag will take you to the **Image Tag Details** page for that tag.

Image Tag Details

The **Image Tag Details** page provides the most detailed information about a specific image tag.

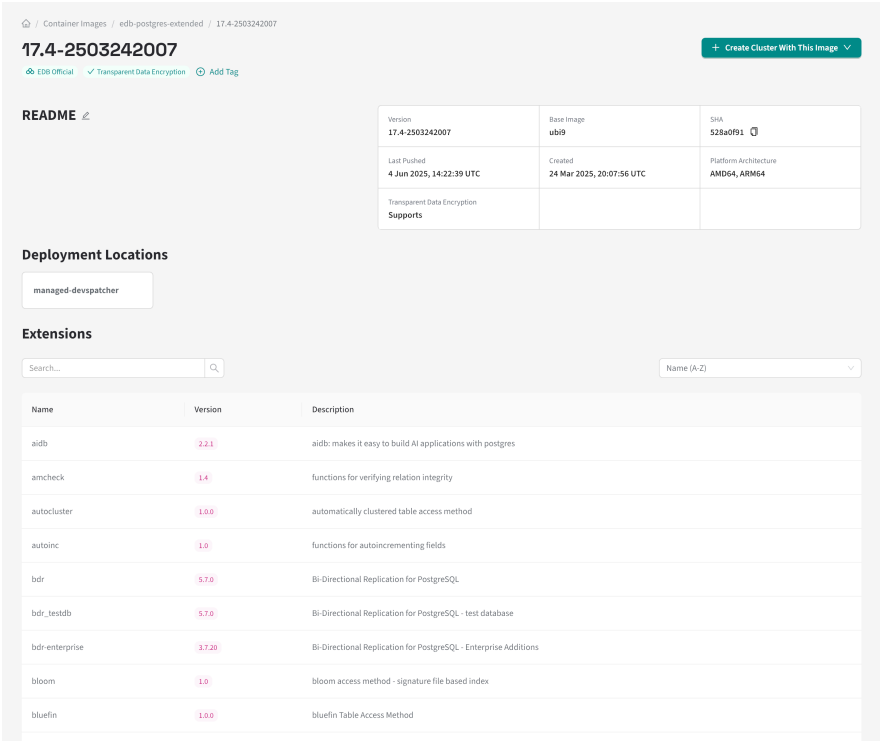


Fig. 11: Image Tag Details Page

At the top of the page you will see the name of the image tag. Below that are the Hybrid Manager tags associated with this image tag, if any. This can include the EDB Official tag and the Transparent Data Encryption tags, which are

generated by the platform, along with any user created tags. Next to those tags is an **Add Tag** which allows you to add (or remove) user created tags to the image tag, and create new tags if needed.

Below includes the following sections:

- **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, such as X86, AMD64, ARM64, PPC64LE or S390X.
- **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, such as X86, AMD64, ARM64, PPC64LE or S390X.
- **Transparent Data Encryption**: Indicates whether the image tag supports Transparent Data Encryption (TDE).

Below this information is a field labelled **Deployment Locations** which lists the locations where the image tag is available.

Below that is a table of **Extensions** that are included in this image tag. Each extension is displayed with its name, version, and description. Above the table is a search field that allows you to search for a specific extension by name or description. The table can also be sorted by name (ascending or descending).

Finally, there is a section for **Clusters Using This Image Tag**. This lists any clusters in the estate that are using this image tag. If no clusters are using this image tag, the message "No Data" will be displayed.

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.

This is achieved by allowing the system to include repositories and tagged images that match a specific pattern or patterns, with each pattern defined as a rule. This then allows you to control which repositories are included in the Asset Library and which images are available for use in your Hybrid Manager clusters.

By default, the asset library includes all the official Hybrid Manager repositories and all the tagged images within 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 as needed. Whichever route you choose, the Asset Library's repository inclusion rules still operate to control what is available in the Asset Library.

Manage Repositories

The **Manage Repositories** page allows you to add, edit or delete repositories registrations that are available in the Asset Library. This page is accessed by clicking the **Manage Repositories** button in the **Container Images** tab of the Asset Library.

The screen displays a table of the repositories that are currently registered in the Asset Library, with 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.

On each row, when populated, there is also an **Edit** button that allows you to edit the repository registration and a **Delete** button that allows you to delete the repository registration.

There is also a search field that allows you to search for a specific repository registration by name and a sort control that allows you to sort the table by the repository name or created date.

In the top right corner of the screen, there is a button to **Register New Repository**. Clicking this button opens a dialog that allows you to add a new repository registration.

Register New Repository

When you click the **Register New Repository** button, a dialog opens that allows you to enter the details of the new repository registration.

The dialog contains the following fields:

- **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 needs to match the name of the repository (or repositories) that you want to include in the Asset Library. For example, if you want to include all repositories that start with `pgaihm-` , you would enter `^pgaihm-.*$` as the regular expression. You can enter plain text but it will be treated as a regular expression, but it is recommended to use 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 is 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` , you would enter `^latest$` as the regular expression. You can enter plain text but it will be treated as a regular expression, but it is recommended to use a regular expression to ensure that the tag eactly matches the pattern you want.

You can add multiple rules by clicking the **Add Rule** button, which will add a new row to the list of rules. You can also delete a rule by clicking the **Delete** button next to the rule.

Once you have entered the details of the new repository registration, click the **Register Repository** button to save the registration. The new repository registration will then be added to the list of registered repositories and will be available in the Asset Library. This may take a few minutes to complete as the system will need to scan the repository for images that match the rules you have defined.

You can also cancel the registration by clicking the **Cancel** button, which will close the dialog without saving any changes.

You will be returned to the **Manage Repositories** page, where you can see the new repository registration in the list of registered repositories.

Return to the *Asset Library in Hybrid Manager* to see the effect of new repository rules and the images that are now available in the Asset Library.

Knowledge Bases in the Asset Library

Knowledge Bases in the Asset Library enables you to manage your collections of AI models that can be used in your Hybrid Manager 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:

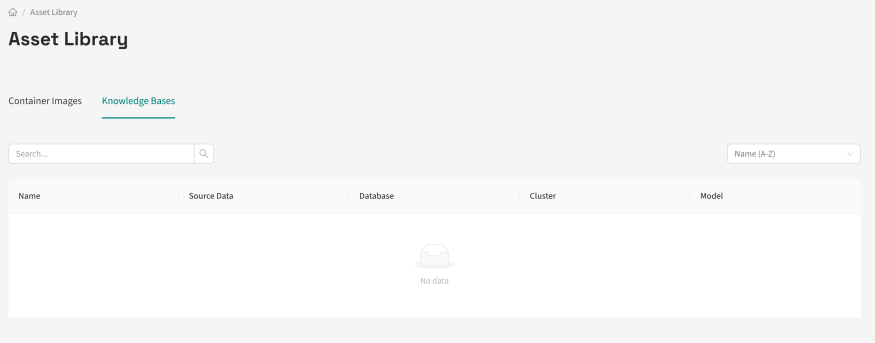


Fig. 12: Knowledge Bases Page

- **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.

About Backups and Restores

Basics

The fundamental tasks of backing up and restoring are:

- ref:*PGAIHM Backups and DR <PGAIHM Backups and DR>*
- ref:*About Backups and Restores <About Backups and Restores>*

Backups

The EDB PG AI Hybrid Manager (PGAIHM) features backup and recovery capabilities for various components, using various technologies.

PGAIHM offers two distinct backup strategies:

- **Postgres backups**: Backs up your data.
- **PGAIHM backups**: Backs up the PGAIHM system' s core functionality.

Postgres Backups (Data Plane)

Postgres backups focus on protecting your user data stored within the Postgres database. The following options are available:

- **Base Backups**: Base backups are available in all PGAIHM deployments.
- **Snapshot Backups**: 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 Logging (WAL) archiving, to provide granular point-in-time recovery (PITR). This allows for a very fine-grained recovery point objective (RPO).

PGAIHM Backups (Control Plane)

PGAIHM backups protect the PGAIHM system itself, including its metadata and system databases. They operate on a separate schedule and produce different backup files than Postgres backups. PGAIHM backups are file-system based, with hourly backups of metadata and backups of the PGAIHM's system Postgres databases by default.

Unlike Postgres backups, PGAIHM backups don't offer continuous backup through WAL archiving. This means recovery is limited to the hourly (or user-defined) backup intervals.

Key Differences and Recovery Implications

Because of these differences, the Recovery Point Objective (RPO) and Recovery Time Objective (RTO) will vary between your user data (Postgres) and the PGAIHM system.

- **Postgres (Data):**Offers strong RPO capabilities due to PITR and WAL archiving, potentially allowing for recovery to a very recent point in time.
- **PGAIHM (System):**RPO is limited to the user-modifiable (hourly by default) backup schedule.

Component	RPO	RTO	Method	Location
PGAIHM	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, possibly up 1 TB depending on hardware performance, is expected to be the effective size limit. A Postgres cluster with more than 1 TB of data may encounter backup durations and performance loads that negatively impact 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 impacting service quality.

Backup Transportability and Disaster Recovery

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 cannot be moved to another appliance, data center, or location. This effectively rules out DR with the most basic implementation of snapshots.

When the PGAIHM is configured with **enterprise storage and CSI drivers**that support snapshots, the PGAIHM 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 is dependent on the specific CSI driver implementation.

AWS Specifics:

- By default, AWS volume snapshots are **not transportable**.
- While restoration to a *new host* within the same AWS region is possible, cross-region restoration requires additional manual steps.

TopoLVM Specifics:

- TopoLVM snapshots are **not transportable**and cannot be restored to a different host.

Configuration

Use the following section to configure storage, object storage, and backups for the following:

- Single-Node Cluster
- Primary/Standby Cluster
- Postgres Distributed (PGD) Cluster

Single-Node Cluster Backups of a single-node cluster can have the following characteristics:

- A single-node cluster deployed using the PGAIHM.
- The Retention Period and Backup Time can be modified (or use the default values).
- Automated backups occur after cluster deployment and daily thereafter.
- Manual backups can be initiated.

Backup Primary/Standby Cluster Backups of a Primary/Standby cluster can have the following characteristics:

```
- A Primary/Standby cluster deployed using the PGAIHM.
```

- The Retention Period and Backup Time can be modified (or use the default values).
- Automated backups occur after cluster deployment and daily thereafter.
- Manual backups can be initiated.
- Backups are taken from a replica (primary / secondary).

Backup PGD Cluster Backups of a PGD cluster can have the following characteristics:

```
- A PGD cluster deployed using the PGAIHM.
```

- The Retention Period and Backup Time can be modified (or use the default values).
- Automated backups occur after cluster deployment and daily thereafter.
- Manual backups can be initiated.
- Backups are taken from a replica (primary / secondary).

PGAIHM Backups

PGAIHM backups include metadata needed to restore the PGAIHM. They do not include backups of user databases. PGAIHM backups use a tool called Velero to backup to object storage. The default backup schedule is hourly and can be adjusted.

When using the default schedule, the RPO is one hour. This means you can expect to lose up to one hour of PGAIHM data in the event of a crash. This doesn't refer to data loss at the user database level since user databases are protected with continuous backups of Postgres.

Taking a backup

Use the following steps to take a Postgres or PGAIHM backup:

Taking a Postgres backup

You can use the PGAIHM web 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 within the PGAIHM web console.

Use the following steps to take a Postgres backup in the PGAIHM web console:

Select the **Backups**tab to view existing backups:

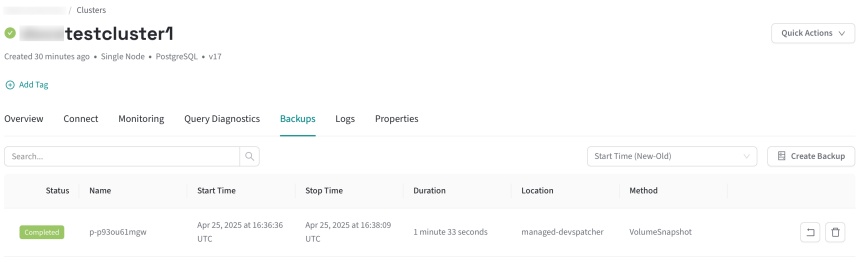


Fig. 13: Viewing Backups

The following fields are available:

Name	Description
Status	The current status of the backup.
Name	The name of the backup.
Start Time	The time the backup started.
Stop Time	The time the backup stopped.
Duration	The amount of time it took to complete the backup.
Location	The location of the backup.
Method	The method by which the backup was taken.

Select the **Create Backup**tab to create a new backup:

The following fields are available:

- Location
- Backup Name
- Backup Method

After entering data into those fields, select **OK**.

Taking an PGAIHM backup

Refer to the HA/DR section of the site to learn how to configure backups of the PGAIHM.

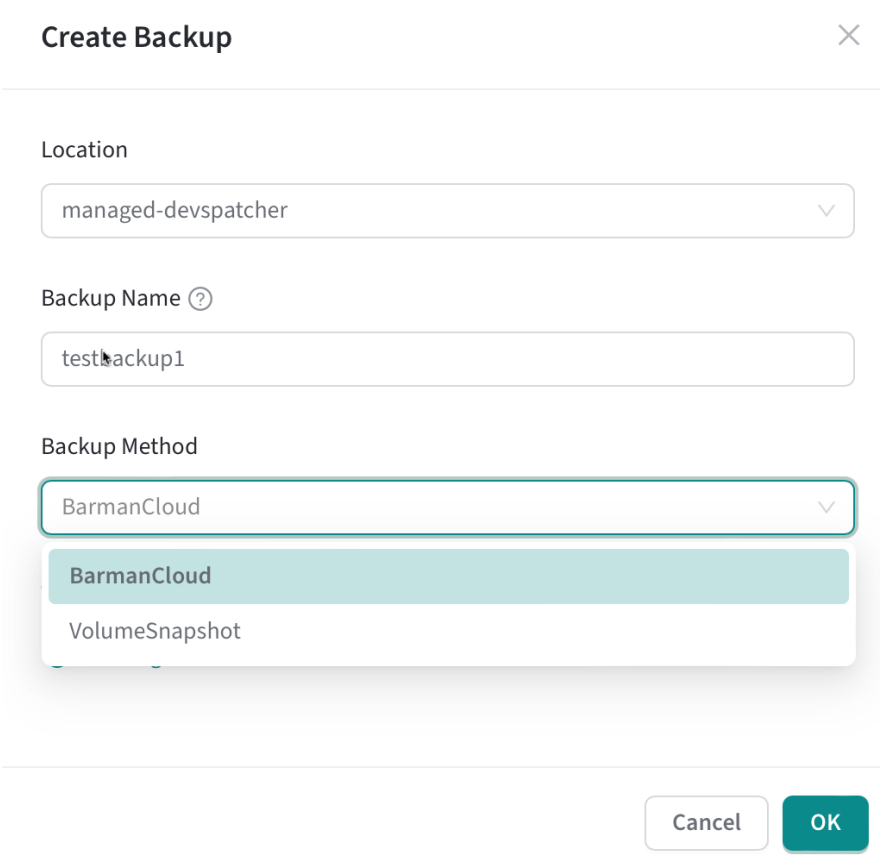


Fig. 14: Create Backup

Terminology

The following terminology is used for backup and recovery with Postgres and PGAIHM:

Backup

A backup is a copy of one or more files from the original location to a secondary location, like object storage. Backups vary by component type (e.g., control plane vs. data plane) and deployment type (e.g., CSP, engineered system, or RHOS).

Backup Retention Period

The backup retention period is the length of time that backup copies of data are stored before being deleted or overwritten. It defines how long an organization keeps its backups available for recovery purposes. It's how long you keep your backups.

Barman

Barman is an open-source Postgres backup tool that is primarily maintained and supported by EDB. When used alone, the term Barman refers to the "Barman server" or core backup tool, as opposed to the Barman Cloud Utility Scripts described below. Barman automates many of the Postgres backup and recovery primitives that are required for enterprise-class B\&R. Barman provides many innovative features like incremental backups (before Postgres offered block-level incremental backups), WAL streaming to achieve RPO=0 (or near zero), backup reporting, automated backup file retention management, and other useful capabilities.

Barman Cloud Utility Scripts

The Barman Cloud Utility Scripts , or "Barman Cloud" for short, are a collection of scripts that can be used separately from the Barman server and their purpose is to copy Postgres backups to S3-compatible object storage. Barman Cloud is used by the CNP operator and EDB Cloud Service to copy Postgres backups to object storage.

Base Backup

The term "base backup" refers to the Postgres backup utility *pg_basebackup* . This utility automates the backup, or file copy, of the required files needed to restore a Postgres instance.

Cloud

Cloud refers to a public cloud provider, namely AWS, Microsoft Azure, and Google Compute Platform (GCP).

Cluster

The term cluster is used to refer to a collection of databases in a single Postgres instance (aka, the Postgres cluster). In PGAIHM, it is also used to refer to a deployment type (i.e., single-node, PSR, PGD cluster).

Cloud Native Postgres

Cloud Native Postgres (CNP) refers to the open-source Postgres operator developed and maintained by EDB. It provides the foundation for Postgres on Kubernetes in the PGAIHM.

Dump

The term "dump" refers to the Postgres utility *pg_dump* . Please see pg_dump below.

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

This is a data export and import utility. Although it offers many capabilities to export databases in part or in whole, it is not an enterprise-class, scalable backup tool. Having said that, pg_dump is widely used to export data for a variety of purposes, including the creation of test databases, data archiving, migrating from one database to another (or one

DBaaS provider to another). PGAIHM users will expect to be able to 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

Recovery is 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

The recovery point objective (RPO) refers to the point, or time (e.g., 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, measured in time, that an organization can withstand after a disruption. It defines the point in time to which systems and data must be recovered.

Recovery Time Objective

The recovery time objective (RTO) is 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 should not exceed 30 minutes.

Restore

Restore refers to the process of copying files from a backup location to the original or an alternate location to prepare for recovery. An example is copying backup files from object storage to the database server.

Snapshot

The term "snapshot" is used here to refer to 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 PGAIHM deployment type and the availability of storage features like transportability.

Transportable Volume Snapshot

A transportable volume snapshot is a point-in-time copy of data that can be moved or copied to a different storage system, location, or cloud region. This allows for disaster recovery, data migration, and offsite backups.

Non-Transportable Volume Snapshot

A non-transportable volume snapshot is a point-in-time copy of data that is confined to the specific storage system or location where it was created. It is primarily used for local data recovery within that system.

Restores

Restoring a Postgres backup

You can use any of the following options to perform a restore:

- The PGAIHM web UI
- The API
- The CLI

Use the following steps to restore a Postgres cluster:

Restoring from the PGAIHM web UI cluster restore feature

1. Select the **Backups** tab to view existing backup that you can restore from. The following fields are available:

Name	Description
Status	The current status of the backup.
Name	The name of the backup.
Start Time	The time the backup started.
Stop Time	The time the backup stopped.
Duration	The amount of time it took to complete the backup.
Location	The location of the backup.
Method	The method by which the backup was taken.

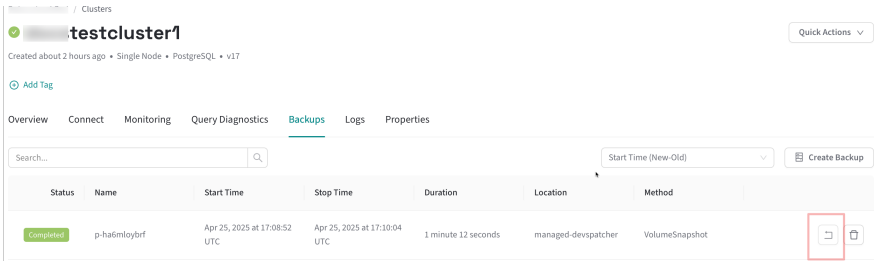


Fig. 15: Viewing Backups

2. Choose a backup to restore from.
3. Choose a cluster type to restore to.
4. [Optional] Select **Restore Cluster** for a full restore without changing any configuration.
5. In the **Cluster Settings**, provide a unique name for the restored cluster.
6. 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.
7. For **DB Configuration**, and **Additional Settings**, select your preferred database configuration.
8. Select **Restore Cluster** once you're ready to restore the cluster from a backup. The restored cluster will appear in the **Clusters** list in the console.

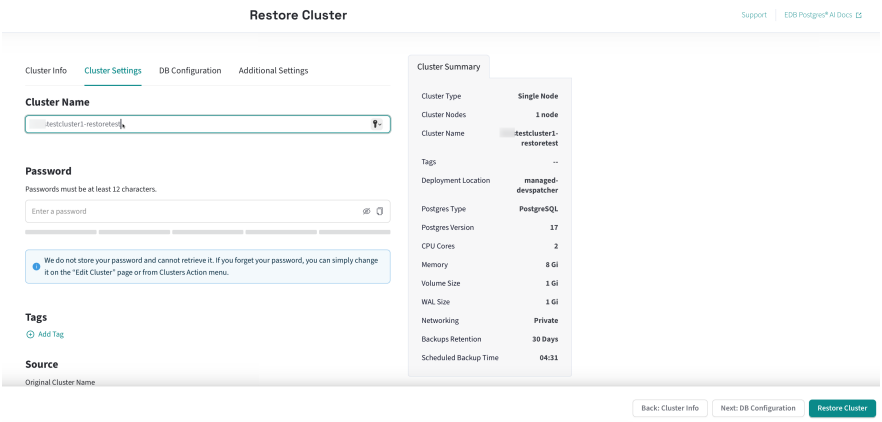


Fig. 16: Restore Cluster

Restoring from the API

To restore with the API, the following are required:

- A JSON file to specify the restore configuration.
- A shell script that uses curl to call the JSON file.

> **Tip**
>
> You can obtain the file templates for point in time restores in the EDB GitHub repository . The templates for restores are named *pitr_* .

Use the following steps 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**`` : Obtain 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:

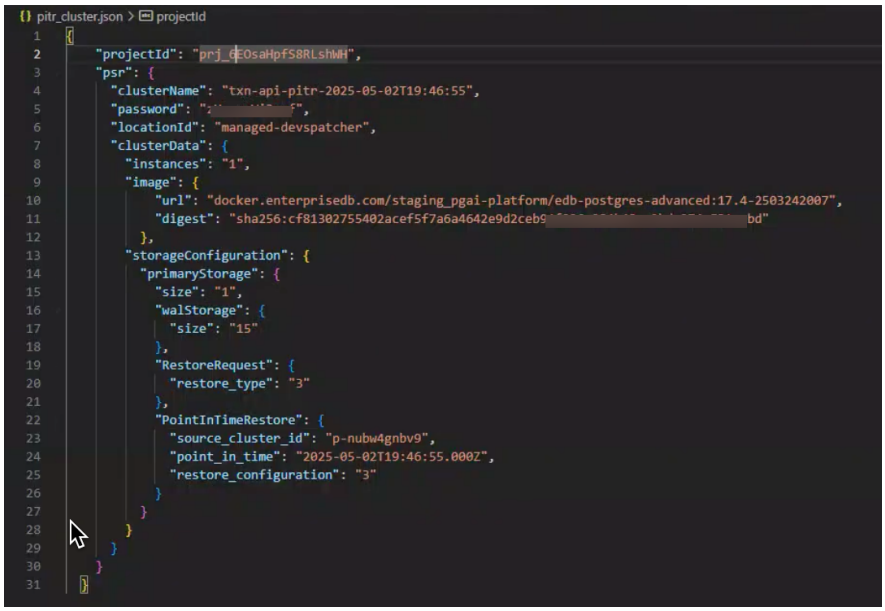


Fig. 17: PGAIHM Restore via API JSON file

2. Create a shell script that will call the JSON file. For example:

```
$ pitr_cluster.sh
1
2  HCP_API_ACCESS_KEY=baak_                                    ekaR
3  HCP_PROJECT_ID=prj_6E                                       shWH
4
5  curl \
6    --insecure \
7    --header "Content-Type: Application/JSON" \
8    --header "x-access-key: ${HCP_API_ACCESS_KEY}" \
9    --data-binary "@pitr_cluster.json" \
10   --request POST \
11   https://                    .enterprisedb.network/api/v1/projects/prj_6E                    hWH/clusters \
12   | jq
```

3. 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:

```
                                                    /demos/api_provisioning$ ./pitr_cluster.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1013  100    28  100   985     14    492  0:00:02  0:00:01  0:00:01   506
{
  "clusterId": "p-f5b7zqmjs9"
}
```

Fig. 18: PGAIHM Restore via API shell file conf

Restoring from the CLI

Use the following steps to restore to a point in time using the CLI.

To restore with the CLI, the following are required:

- A YAML file to specify the restore configuration.
- Use of `edbctl` at the CLI to call the YAML file.

1. In your YAML file, supply the following parameters:

> **Note**
>
> For more information on the expected parameters when using the restore command, use *edbctl cluster restore -h* in the CLI to view the help file.

1. Execute the CLI command by pointing it to your YAML file. A message will be returned to indicate the cluster restore has started. For example:

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 include activities such as provisioning, deprovisioning, and scaling clusters, as well as automating administrative operations.

For more information, see API docs from the EDB Postgres AI Portal:

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.

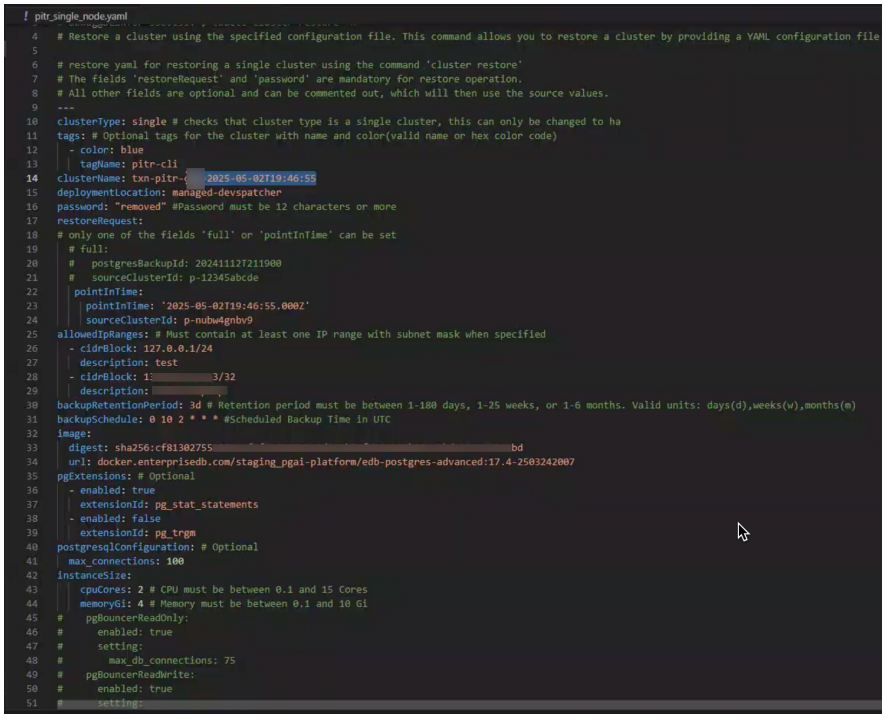


Fig. 19: PGAIHM Restore via CLI YAML file



Fig. 20: PGAIHM Restore via CLI YAML file

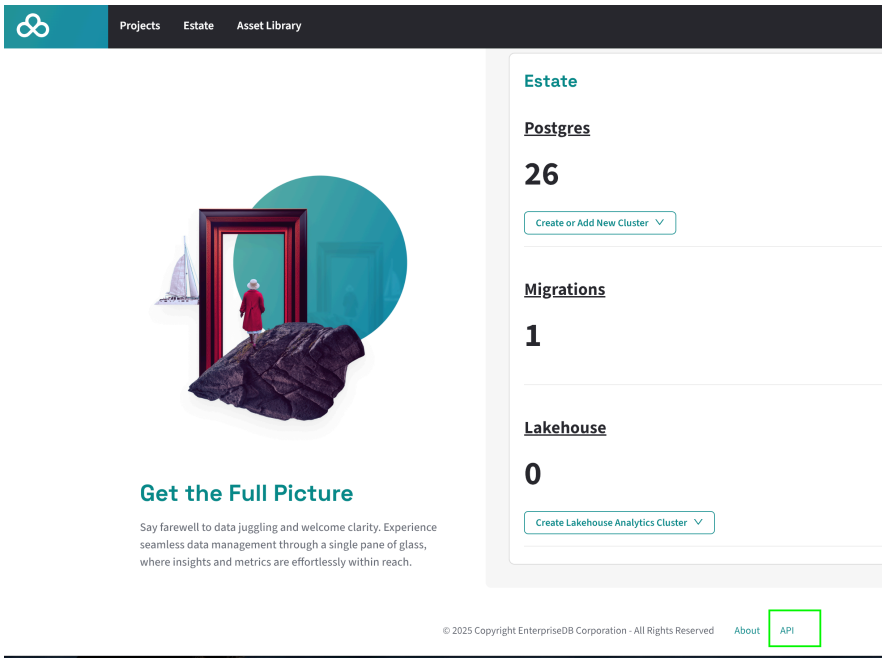


Fig. 21: API Docs

- 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 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 and normal user can manage his own access key.

Each key must have an expiration period between 1 and 365 days, set at creation time. Once an access key expires or if it is 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 the *Creating a machine user* . Optionally, can be individually added from **Access keys** tab on machine user's details page using **Create New Key** option.
Normal user	1	One access key, *Step 3: Validate that the PGFS storage location was created* and managed by user from their home page.

An access key can be revoked from 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 drop down menu, next to your organization name in the top right of the portal, select **My Account**.
3. From the My Account page, select the **Access Keys** tab.
4. From the Access Keys tab, select **Create New Key**.
5. Provide the **Access Key Name**.
6. Provide the **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 from the **Users** home page.

To view the details of a access key:

1. From the drop down menu, next to your organization name in the top right of the portal, select **Users Management**.
2. Select the **Edit** button for the user in the list.
3. View the **Access Keys** tab on the user's home page. Each row provides the access key details:
4. **Name** — Access key name.
5. **Access Key** — Access key in half encrypted format.
6. **Expiry at** — The expiry date for the access key.
7. **Created at** — The creation date for the access key.

To delete a specific 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 corner 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 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 HM UI.

Example response:

```
{
"data":
[
{
"projectId": "string",

"projectName": "string",
"userCount": 0,
"clusterCount": 0,
"tags":
[]
}
]
}
```

5.1.4 For Kubernetes Administrators

Managing users in Hybrid Manager

The Hybrid Manager uses Dex for user authentication in the Console, acting as a proxy between the Hybrid Manager and any native users or external user management systems (Identity Providers or IdPs) like LDAP and SAML.

During the initial bootstraping of the Hybrid Manager, an administrator sets up a native super-user for Console access. Use this native super-user 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 production deployments, we strongly recommend configuring an external IdP like LDAP, SAML, or another supported connector . This ensures robust and secure user management. After configuring your chosen IdP and adding new administrators for the organization, you should disable the native user login.

- See Configuring an IdP 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 *configuring an external IdP* . After configuring an IdP, you can:

- *Disabling native user login*

Changing password for native users

Change the password of the default native user during deployment or immediately after deployment to avoid unauthorized access to your environment. You can change the password for native users you have 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, e.g. `$2y$10$YDHMEnc8VZKnhvze538ePOLKol7dFH.S7IgpqC3.X3R/3P2Q9zRcG`

1. Replace the hash for the existing user, in this example, `owner@mycompany.com` . You will have to 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 are installing the Hybrid Manager, continue performing the installation. If you are modifying the parameter in an already deployed installation, *Applying modifications to the user configuration* to populate the changes.

Creating native users for testing

For testing or demo environments, you can simplify operations by adding more native users. For production deployments, we strongly recommend *configuring an external IdP* .

Currently, it' s not possible to create human users with the Console, but only by tweaking the Hybrid Manager configuration file.

Adding a native user

Append new users to the `portal.authentication.staticPasswords` section during or after installation. You will have to 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>"
```

Applying modifications to the user configuration to populate the changes.

Parameters

Settings 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 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 utilized.
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 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 to the Hybrid Manager Console as the IdP user that will become the organization owner at least once. This ensures the Console indexes the new user. Afterward, log out.
2. Log in as the current native owner of the Hybrid Manager (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. *Applying modifications to the user configuration* again to populate the changes.

Configuring your own identity provider

For production deployments, we strongly recommend configuring an external IdP like LDAP, SAML, or another supported Dex connector . This 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 native users for testing* .

Configuring LDAP

Configure LDAP to work with Dex authentication to enable end users to log in to the 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, refer to the *Okta LDAP Interface documentation* .

Configuring Dex to connect to Okta via LDAP connector

Add an entry for the Okta LDAP configuration where 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:
    clientSecret: "edb-postgres-ai-secret"
    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

This section details the necessary identity provider settings within the `portal.authentication` configuration. It focuses exclusively on parameters that require customization based on your environment. Any other parameters not explicitly mentioned here can be configured as shown in the provided example.

Parameter	Description
clientSecret	Client secret for the Hybrid Manager authentication, use the default *edb-postgres-ai-secret*.
idpConnectors	Enter an array of data per IdP connector you want to use for authentication. Refer to the official *Enabling the Migration Portal AI Copilot* 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 Dex should 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.base	Specify where Dex should look for users in your LDAP server. The administrator of your Okta LDAP instance will know the baseDN for users.
idpConnectors.config.groupSearch.bas	Specify where Dex should look for groups in your LDAP server. The administrator of your Okta LDAP instance will know the baseDN for users.
idpConnectors.config.groupSearch.bas	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'll need to reconcile the Dex configuration and restart the Dex services for your changes to take effect. See *Applying modifications to the user configuration* for instructions.

Assign new owner and disable native user

After assigning a new organization owner in the Hybrid Manager Console, disable the native user. For this, refer to *Disabling native user login* .

Configuring SAML

Configure SAML to work with Dex authentication to enable end users to log in to the 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 and 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 the Hybrid Manager 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:
    clientSecret: "edb-postgres-ai-secret"
    idpConnectors:
      - type: saml
        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

This section covers the essential identity provider settings within the `portal.authentication` configuration. It focuses only on 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
clientSecret	Client secret for the Hybrid Manager authentication, use the default *edb-postgres-ai-secret*.
idpConnectors	Enter an array of data per IdP connector you want to use for authentication. Refer to the official *Enabling the Migration Portal AI Copilot* 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.

continues on next page

Table 15 – continued from previous page

Parameter	Description
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 to which groups a user's belongs.
idpConnectors.config.caData	Certificate from your Okta application's SAML configuration. It allows the Hybrid Manager 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'll need to reconcile the Dex configuration and restart the Dex services for your changes to take effect. See *Applying modifications to the user configuration* for instructions.

Assigning new owner and disable native user

After assigning a new organization owner in the Hybrid Manager Console, disable the native user. For this, refer to *Disabling native user login* .

Applying modifications to the user configuration

After modifying the user authentication settings, you'll need to reconcile the Dex configuration and restart its services for your changes to take effect. The method for reconciling these updates depends on how you installed the Hybrid Manager: either via a Helm chart or using the EDB Operator.

Helm chart installations

On Hybrid Manager instances that were installed using the Helm chart method, apply the Dex modifications as follows:

1. Reapply the helm chart with the updated `values.yaml` . See *Install Hybrid Manager using the Helm chart* for an example on how to do this on a GCP installation.
2. Restart Dex to reload and add the new users:

```
kubectl rollout restart deploy upm-dex -n upm-dex
```

Operator installations

On Hybrid Manager instances that were installed using the operator method, apply the DEX modifications as follows:

1. Reapply the `HybridControlPlane` Custom Resource (CR). See *Install Hybrid Manager using Operator* for an example on how to do this on a RHOS installation.
2. Restart Dex to reload and add the new users:

```
kubectl rollout restart deploy upm-dex -n upm-dex
```

Monitoring with Hybrid Manager

With the Hybrid Manager you can collect a wide range of metrics (host metrics, Kubernetes metrics, postgres database data, query data, as well as logging data). You can visualize the collected data with the console or Grafana instance to analyze trends, configure alerting, improve the efficiency of databases, perform troubleshooting and much more.

Understanding the monitoring architecture

The Hybrid Manager 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 within the Hybrid Manager. These services and stores are deployed by default.

These collected metrics become available to you through two interfaces:

- The **Grafana** dashboard.
- The **Console** , for example, in the Estate tab or *Cluster view* .

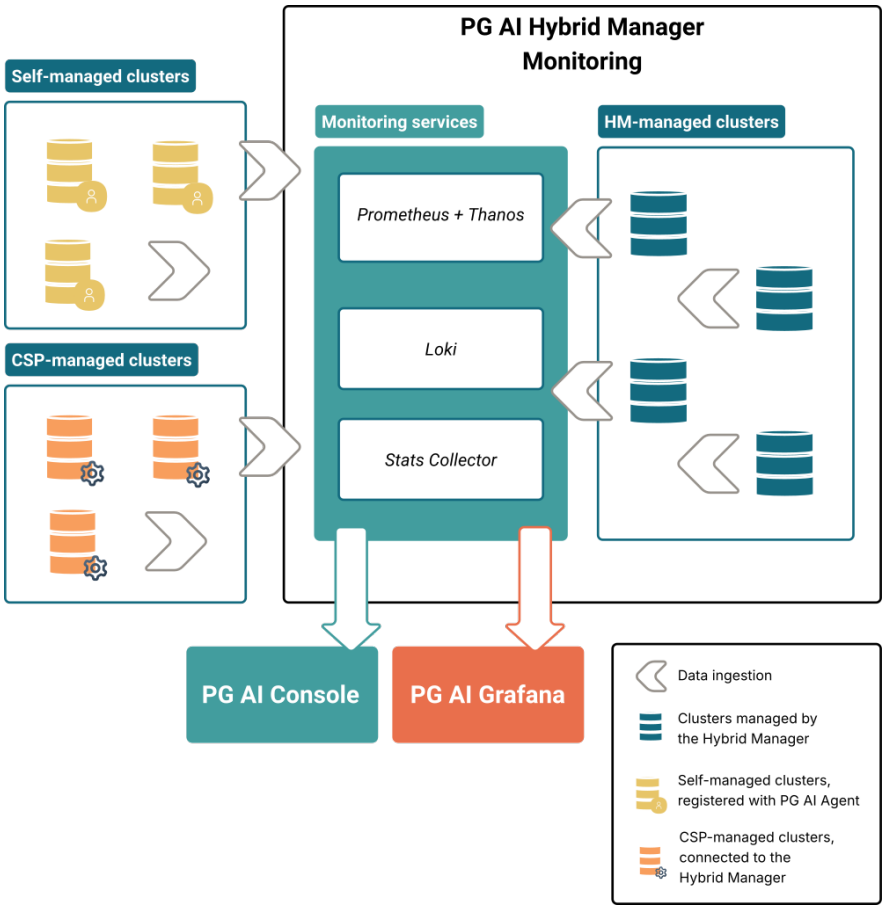


Fig. 22: Monitoring infrastructure in the Hybrid Manager

Enabling database monitoring

The monitoring services that come with the Hybrid Manager require access to the databases you want to monitor, as well as some extensions to enable the Stats Collector.

- For **HM-managed** clusters (created with the **Hybrid Manager**), the default setup already provides access to the monitoring stack.

- For external **self-managed** clusters, each of the databases must be registered with the **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 the Stats Collector (collection of query data), namely `edb_wait_states_data` , `edb_stat_monitor` , `edb_query_advisor` , and `pg_stat_statements` .
- For external **CSP-managed** databases (cloud service provider-managed), such as **AWS RDS**, the Hybrid Manager needs to get access to the database through a Kubernetes `Secret` resource to enable data collection. See Monitoring CSP-managed databases for more information. Query data with the Stats Collector is not available for CSP-managed databases at this moment.

Monitoring data ingestion and scope

Because of the different levels of access the Hybrid Manager has to the variety of cluster types, the way that data is ingested and the scope of collected data is varied.

- For **HM-managed** clusters (created with the **Hybrid Manager**), the databases automatically push host metrics, Postgres metrics, cluster metrics, query metrics, data required for recommendations, as well as 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, as well as logs to the corresponding Prometheus, Thanos, Stats Collector stores at configurable intervals.
- For external cloud service provider-managed databases (CSP-managed databases), such as **AWS RDS**, the Hybrid Manager leverages the Manager-to-database connection to pull host, and Postgres metrics to the central stores. Query data with the Stats Collector is not available for CSP-managed databases at this moment.

For more information, see *Metrics support scope* .

Monitoring refresh frequency

The refresh intervals for the metrics that are collected by the Hybrid Manager 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 the Hybrid Manager into external tools or enterprise systems.

Related topics

- ref:*Enabling monitoring <Enabling monitoring>* ☒ Install the agent or connect your database to the Hybrid Manager to enable monitoring on external databases.
- ref:*Metrics refresh intervals <Metrics latency>* ☒ Understand how often metrics are updated and how to customize the frequency.
- ref:*Monitoring with Grafana <monitoring_grafana>* ☒ Use Grafana to monitor your databases.
- ref:*Monitoring console <monitoring_console>* ☒ Use the console to monitor your databases.
- ref:*Notifications <Notifications>* ☒ Configure notifications.

Enabling monitoring

On HM-managed database clusters

Since HM-managed database clusters are created in the same environment as the Hybrid Manager, their resources are already accessible. Therefore, these clusters are monitored by default, and monitoring starts as soon as their deployment is finished.

On external databases

External databases are either managed by a cloud service provider or by yourself. For this reason, additional steps are required to create a connection between your database cluster and your Hybrid Manager instance.

There are two types of *Can I monitor 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 the Hybrid Manager.

Examples: Amazon RDS.

!!!note

You can also register Oracle databases with the Agent to *get your Oracle database ready for migration* .

Monitoring self-managed deployments

To monitor self-managed external Postgres instances from the **PG AI Hybrid Manager** that are deployed on bare metal or virtual machines, you must first install the PG AI Agent. The agent provides monitoring and observability metrics to your PG AI Hybrid Manager.

On completing this guide you will 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 PG AI Agent*
5. *Configure the PG AI Agent*
6. *Run the PG AI Agent*
7. *Configure the PG AI Agent to run as a service*

For extra help with the necessary administrative tasks needed to configure the agent, see *Administrative tasks* .

See *Troubleshooting* for help with general troubleshooting and *Data security and sovereignity* 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 are not necessary for its function.

To create a new user:

```
sudo useradd <agent_os_user>
```

We recommend that the agent OS user is added to the Postgres user's group. This facilitates granting the agent access to the Postgres log files later in this guide.

```
usermod -aG postgres <agent_os_user>
```

Preparing your database for monitoring

Prepare your Postgres instance

Monitoring user

When configuring the PG AI Agent, you must specify a user that the agent will use to connect to the database.

This can be any Postgres user with the `pg_monitor` role assigned. For this example, we will create a new user `agent_user`.

```
CREATE USER agent_user WITH PASSWORD <password>;
GRANT pg_monitor TO agent_user;
```

If you are using Postgres 16 or earlier, you must also grant the following permission to allow the agent to automatically 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 enable the full monitoring and optimization features of the PG AI Hybrid Manager, you must create a schema called `edb_internals` and install the following extensions there. These extensions add instrumentation to Postgres without which the **Query Diagnostics** and **Recommendations** tabs, as well as some metrics, will not be available.

- ref:*EDB Wait States <Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator>*
- ref:*EDB Query Advisor <Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator>*
- ref:*EDB Stat Monitor <Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator>*

In addition, *pg_stat_statements* must be enabled in the public schema.

To install the extensions:

1. Refer to the installation instructions linked above for the correct commands for your platform and Postgres distribution.

Note that 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
```

2. 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";
```

3. 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;
```

4. Add the edb_internals schema to the search path. For example, as a superuser:

```
ALTER SYSTEM SET
search_path="$user", public, edb_internals;
```

5. 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 advise you run the agent using a *dedicated OS user* .

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` will cause logs to be written to `/var/log/postgres-lc` and be readable by any user who is 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.

Prepare your Oracle instance

Instead of providing monitoring capabilities, registering an Oracle database with the PG AI Agent provides migration assessment and schema extraction capabilities.

To assess an Oracle database, the PG AI 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 Agent will still function if these roles aren't assigned, however, the agent will function in reduced feature capacity.

1. Ensure the Oracle user you'll use to perform the assessment has access to these resources

```
V$VERSION
V$INSTANCE
V$LICENSE
sys.all_tables
sys.all_tab_privs
```

2. It must also have at least one of the following permissions:

```
SELECT_CATALOG_ROLE
SELECT ANY DICTIONARY
```

You will later specify this user in the `provider.onprem.databases.dsn` section of the PG AI Agent configuration file.

Creating a machine user

In the EDB Postgres ® AI Hybrid Control Plane Console, create a machine user that Beacon Agent can use to ingest data into your Estate.

Creating a machine user

1. Select your profile's 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. Optional: Enter an e-mail address to be associated with the user. This is not required.
5. Check the **Create Access Key** checkbox. The form expands to request an **Access Key Name** and an **Expiry** time.
6. Select **Add User**.
7. Ensure you 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-side 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** pop-up window, check **estate ingester**.
7. Select **Submit**.

Your new machine user is ready to ingest data from the Beacon Agent.

Installing the Agent

Install and configure the Agent to monitor databases in your **Estate** page in the Hybrid Manager.

Prerequisites

- You created a machine user and access key with the `estate ingester` role assigned to it. See *Creating a machine user* .
- You need the project ID for the project where you want to monitor the database. To obtain the ID, navigate to your project and fetch the ID from the URL displayed in the browser.

Enable your system to download packages

Enable the system where you are planning on running the agent to download packages from the EDB repositories for EDB Postgres AI.

Locate your EDB subscription token on the EnterpriseDB repos download page .

Using the retrieved subscription token, set an environmental variable (`EDB_SUBSCRIPTION_TOKEN`) to the token's value. Also set an environmental variable for your EDB subscription type (`EDB_SUBSCRIPTION_TYPE`) to one of `standard` or `enterprise` depending on which plan you are signed up to:

```
export EDB_SUBSCRIPTION_TOKEN=<your-repos-token>
export EDB_SUBSCRIPTION_TYPE=<your-subscription-type>
```

Depending on your operating system, run the following command to give your system the capacity to download packages from the EDB repositories:

For RHEL-like or SLES:

```
curl -1sSLf "https://downloads.enterprisedb.com/$EDB_SUBSCRIPTION_TOKEN/$EDB_
↪SUBSCRIPTION_TYPE/setup.rpm.sh" | sudo -E bash
```

For Debian or Ubuntu:

```
curl -1sSLf "https://downloads.enterprisedb.com/$EDB_SUBSCRIPTION_TOKEN/$EDB_
↪SUBSCRIPTION_TYPE/setup.deb.sh" | sudo -E bash
```

Install the Agent package

Install the `beacon-agent` package from EDB's repositories:

For RHEL-like:

```
dnf install beacon-agent
```

Or if dnf isn't available:

```
yum install beacon-agent
```

For SLES:

```
zypper install beacon-agent
```

For Debian or Ubuntu:

```
apt install beacon-agent
```

Configuring the Agent

Configure the agent (deployed as `beacon-agent`) to allow it to connect to your Hybrid Manager and fetch monitoring data from the source database.

You can also use the agent to *get your Oracle database ready for migration* for migration and extract schemas for conversion.

Change to the *OS user you previously created* to perform the agent configuration.

Set environment variables

Ensure that following environment variables are available in your environment. These environment variables must be permanently available as they are required each time the agent starts.

1. Set `BEACON_AGENT_ACCESS_KEY` to the access key you obtained from the console when you *created a machine user* .
2. Set `$DSN` to the Data Source Name (DSN) to which you wish to connect. For example, to connect as user `postgres` authenticated with password `password` to a database `hr` on port 5432 you would specify:

```
postgresql://postgres:password@localhost:5432/hr
```

If you plan to connect to more than one database with this agent, you can specify multiple such variables, for example $DSN1, $DSN2. You can use any name you wish for these variables because they will be referenced by name in the configuration file.

Prepare a configuration file

1. Create a Beacon configuration directory in your home directory:

```
mkdir ${HOME}/.beacon
```

!!!Note If the location specified above is not convenient, you 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.

```
---
agent:
  access_key: $BEACON_AGENT_ACCESS_KEY
  beacon_server: <beacon-server>:9443
  project_id: <project-id>
  providers:
    - onprem
  root_ca_path: ""
general:
  logging:
    level: "info"
  metrics:
    push:
      enabled: true
      include_logs: true
      push_endpoint: <beacon-server>:9443
      root_ca_path: ""
    usage:
      company_code: <company-code>
      output:
        file:
          enabled: true
          path: beacon_usage.json
        http:
          enabled: false
          url: https://pg-usage.enterprisedb.com
provider:
  onprem:
    runner:
      enabled: true
    databases:
      - resource_id: <resource_id>
        disable_host_association: false
        dsn: $DSN
        logs:
          disabled: false
        metrics:
          disabled: false
          stats:
            enabled: true
            wait_states:
              enabled: true
              timeout: 10m
              buckets_count: 5
              bucket_duration: 1m
              bucket_offset: 1s
```

(continues on next page)

(continued from previous page)

```
            recommendations:
              enabled: true
              timeout: 10m
              interval: 1m
            query_texts:
              enabled: true
              scraping_timeout: 10m
              scraping_interval: 2m
              scraping_offset: 1s
              eviction_interval: 30m
              cache_items_max: 200
            query_stats:
              enabled: true
              buckets_count: 2
              bucket_duration: 1m0s
    host:
      metrics:
        disabled: false
      resource_id: <host-id>
dispatcher:
  enabled: true
  mode: standalone
  location_id: <location-id>
```

The placeholders denoted by <> should be replaced with appropriate values. Refer to *Parameter reference* for explanations of each parameter and how to obtain the appropriate value.

Validate the configuration file

Run the following command. This will validate the credentials provided in the previous steps by making a test connection to the Hybrid Manager server and print a summary of your configuration.

```
beacon-agent config
```

If the connection succeeds a summary of the connection details will be printed. If any error are shown in the output, it must be resolved before proceeding.

> **Note**
>
> If the **Connectivity** check fails with a message like *failed to verify certificate x509...* this likely means that the system on which you are running the agent does not trust the certificate used to sign the server certificate. This is likely symptomatic of an issue with root certificate distribution and should be raised with the team that manages your infrastructure. If you have a copy of the root certificate, you can provide the path to the root certificate by setting *agent.root_ca_path* and *general.metrics.push.root_ca_path* . Re-run *beacon-agent config* after setting these parameters to check the the issue is resolved.

If the the **Connectivity** check fails with a message like `connection reset by peer` this generally means the agent cannot reach the specified <`beacon-server`> because the URL is incorrect or there is a firewall preventing connection.

The agent is now configured, you may proceed to :ref:`Create an OS user to run the Agent <Create an OS user to run the Agent>` or :ref:`configure the agent to run as a service <Running the Agent as a service>` . Once your agent is running, consider *configuring usage reporting* .

Usage reporting

The agent collects basic usage data to enable EDB to better understand our customers. It doesn't send any data to EDB until you enable it.

Once the agent is running, the `general.metrics.usage.output.file.enabled` setting (set to `true` in the template above) allows the agent to save a local copy of the usage data in a file, `beacon_usage.json` .

Review this file to verify that the information is accurate and does not contain any data you are not happy to share with EDB. Then, enable reporting to EDB by setting `general.metrics.usage.output.http.enabled` to `true` .

Parameter reference

This section contains additional information to help you understand and edit the agent's configuration file.

YAML agent settings	Placeholder	Guidelines
agent.access_key	N/A	Access key you obtained from the console when you *Enabling the Migration Portal AI Copilot* .
agent.root_ca_path	N/A	If required, 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.
general.metrics.push.root_ca	N/A	Set to the same value as *agent.root_ca_path*.
agent.beacon_server	*<beacon-server>:9443*	Obtain the host URL of your beacon server from an administrator, the port is fixed as 9443. See *Administrative tasks* for more information.
*agent.beacon_dispatcher_*	*<beacon-server>:9443*	The same as *agent.beacon_server*.
general.metrics.push.push_er	*<beacon-server>:9443*	The same as *agent.beacon_server*.
agent.project_id	*<project-id>*	To obtain your project's ID, go to the console, select **Projects** , and select your project. The ID is the identifier starting with "prj_" in the URL of that page.
general.metrics.usage.compa	*<company-code>*	Your company code can be found on the EDB Support Portal .
dispatcher.location_id	*<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.

Settings relevant to the database configuration:

YAML database settings	Placeholder	Guidelines
databases.resource_id	*<resource-id>*	Assign an identifier to your database. This will link your database to an identifier. Encase in double quotes. Must be unique within the project.

continues on next page

Table 17 – continued from previous page

YAML database settings	Placeholder	Guidelines
databases.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_n* * The *database_type* can be postgresql or oracle.
databases.poll_interval	N/A	Specifies how often the Hybrid Manager checks polls the database for basic metadata. Provide the value in minutes (m) or seconds (s).
databases.tags	N/A	Assign tags to label the migration of that specific database. Encase in double quotes and introduce each tag with a dash.
databases.disable_host_a	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.
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.
host.metrics.disabled	N/A	Set to *true* to disable the collection of host metrics from the host on which the agent is located.

Monitoring multiple databases

You can specify more than one instance by adding additional items to the list.

```
databases:
      - resource_id: selfmanaged_postgres_1
        disable_host_association: false
        dsn: "$DSN1"
        ...
      - resource_id: selfmanaged_postgres_2
        disable_host_association: true
        dsn: "$DSN2"
        ...
```

Note

Currently monitoring data is only collected from the database specified in the DSN. For example if you specify the *postgres* database then the database size will only reflect the size of this database. To monitor other databases within the same Postgres instance you must specify them individually in the same way as specifying multiple instances.

Fine-tuning monitoring data collection

The template provided here enables all monitoring features. You can fine-tune which features are enabled.

Time-series metrics (the "Monitoring" tab)

The `general.metrics.push` section of the configuration file controls the pushing of time series metrics back to the Prometheus instance in the Hybrid Manager. These metrics populate the **Monitoring** tab. You can disable this by setting `general.metrics.push.enabled` to `false` .

Postgres logs (the "Logs" tab)

!!!Warning Known issue: viewing logs in Hybrid Manager Logs collected using the agent do not currently display correctly on the "Logs" tab. However, they are stored correctly, and they can be accessed through Grafana or downloaded from the "Logs" tab.

A partial workaround for this issue is to configure `log_destination = 'stderr'` in `postgresql.conf. This will cause log content to appear in the UI, but log entries containing line breaks will appear as multiple log entries in the UI.

By default, the agent attempts to discover the location of the log files by calling the Postgres function `pg_current_logfile()` . If you wish to explicitely 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 are 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`

```
databases:
  - resource_id: selfmanaged_postgres_1
  ...
  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.databases` controls the collection of 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 individual element by setting the appropriate `enabled` parameter.

- `metrics.stats.wait_states` controls the collect 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 options above are disabled, the **Query Diagnostics** tab will be disabled.

- `metrics.stats.recommendations` controls the collection of index and statistics recommendations from EDB Query Advisor. If disabled the **Recommendations** tab will be blank.
- `metrics.stats.query_text` controls the collection of query texts from both EDB Wait States and EDB Stat Monitor. If disabled, no query texts will be collected and you will only see query ID in **Query Diagnostics**.

Running the Agent

Run the agent using the following command:

```
beacon-agent
```

The agent will start sending data to the EDB Postgres AI Hybrid Manager. Follow the logs to monitor the data ingestion process.

After a few minutes, you should see the data from the agent in the EDB Postgres AI Platform with Hybrid Control Plane. Navigate to the **Estate** page to see the data from the agent.

!!!important

The Agent detects schema changes to tables and columns while running. However, changes to views or procedures won't be detected until you restart the Agent.

For production usage you should *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 estate, configure the agent to run as a service. This 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 will only stream metrics when you manually run it. Furthermore, your database will disappear from your Hybrid Manager's **Estate** 72 hours after the agent stops streaming.

Ensure you've completed agent installation and configuration before setting it up as a service.

!!!note

Future versions of the agent package may set this up automatically.

This example configures 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 new 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

2. Paste the following template into the created `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/>

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.
ExecStart=	*</usr/local/bin/beacon-agent>*	Provide the path to the directory where the Agent was installed. The path depends on the operating system.

3. 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
```

4. Your agent should now be running as a service. Check on the logs by running the following command:

```
journalctl -u beacon-agent.service
```

Troubleshooting

If you are having issues configuring the agent or are getting errors when trying to run it, this page will help you troubleshoot. Test the agent's functionalities in an isolated way to narrow down the source of the issue.

Testing connectivity from the Agent to HM

To ensure there's a connection between the Agent and the Hybrid Manager, run the Agent:

```
beacon-agent setup
```

If the output displays no errors, your connection to the Hybrid Manager is working properly, and the issue could have to do with the connection to your databases.

If the output displays errors, your connection to the Hybrid Manager is not working properly. Check the syntax of the

YAML file, the access key, the CA path and contents, DSN string, project ID one after the other to ensure the settings are correct.

Administrative tasks

To configure the Agent, the user requires 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 during or after the installation of the Hybrid Manager.

What is the `beacon_server` ?

To configure the Agent, the user 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 configured in your `values.yaml` file under `upm-beacon.server_host` . The port is fixed as 9443.

What is the root certificate (CA)?

The Agent makes a secure connection to the `beacon_server` URL using TLS. It is therefore necessary that the machine on which the Beacon Agent is running trusts 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 should 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 for some reason this is not the case, the user can set the `root_ca_path` value in the `beacon_agent.yaml` file. This is a directory where the user should have stored the root certificate, a certificate capable of validating the EDB Postgres AI Platform server certificate. However, non-admin users don't have access to the certificate, so you will 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 edbpgai-bootstrap beaconator-ca-bundle -o json
```

For a more readable output that only prints the root certificate, ensure you have `jq` and `base64` installed and run:

```
#  If `jq` and `base64` are available

kubectl get secret -n edbpgai-bootstrap beaconator-ca-bundle -o json | jq -r .data.
↪"public.crt" | base64 --decode
```

This command outputs the contents of the certificate. Send them to the person that requires them through a secure channel. The user will have to store it locally, in crt format.

Data security and sovereignty

When you deploy the Agent to monitor your external databases with the Hybrid Manager, 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 *Connecting to the Hybrid Manager Console* and *Using Grafana dashboards* .

For Oracle databases:

- database data to generate the automatic assessment of migration complexity and level of effort displayed in **Estate > Migrations**.
- 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. Subsequent releases will also gather migration assessment and schema DDL data for external Postgres and cloud service provider databases.

We don't collect data beyond what is strictly necessary for these purposes.

Data storage location

Your collected data stays within your chosen environment. The Hybrid Manager respects your data sovereignty. For example, if you have deployed the Hybrid Manager on your own premises, all collected data from your monitored databases remains within your on-premise infrastructure. If your Hybrid Manager instance is deployed on a cloud platform like EKS, the collected data will reside securely within your specific cloud environment.

Monitoring Cloud Service Provider managed Postgres deployments

PostgreSQL databases running outside the Hybrid Manager in a cloud service provider-managed (CSP-managed) environment can be easily monitored, with full estate representation support.

Currently, we support monitoring the following external CSPs:

- ref:*AWS RDS Setup <Monitoring AWS RDS deployments on AWS>*

Monitoring AWS RDS deployments on AWS

You can monitor your Amazon Web Service (AWS) Relational Database Service deployments with your EDB AI Hybrid Manager (HM).

Prerequisites

You must have access to the Kubernetes environment where the HM resides, and also have sufficient administrator privileges to change the settings on the entire HM cluster.

This is different than other external database monitoring situations with HM, such as *Monitoring self-managed deployments* deployments of Postgres, where less privileges are required.

You must also ensure that the user credentials stored in the Kubernetes secret (beacon-csp-credentials) have sufficient permissions for the AWS RDS environment being monitored. Without these permissions, the system cannot 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",
            "Action": [
                "servicequotas:GetServiceQuota"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowInstanceTypeDescriptions",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstanceTypes"
            ],
```

(continues on next page)

(continued from previous page)

```
            "Resource": "*"
        },
        {
            "Sid": "AllowInstanceFilterByTags",
            "Effect": "Allow",
            "Action": [
                "tag:GetResources"
            ],
            "Resource": "*"
        }
    ]
}
```

Setting up the Agent to monitor AWS RDS deployments on Hybrid Manager

The process of setting up the Agent to monitor all your AWS RDS deployments on the Hybrid Manager (HM) involves the following steps.

> **Note**
>
> Once you register an RDS region, the system adds every database you have in that RDS region. This could 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 not.

1. Add your `beacon-csp-credentials` secret.

Add your beacon-csp-credentials secret in your HM's Kubernetes cluster's `upm-beacon` namespace:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
name: beacon-csp-credentials
namespace: upm-beacon
stringData:
AWS_ACCESS_KEY_ID: replace_with_your_access_key_id
AWS_SECRET_ACCESS_KEY: replace_with_your_secret_access_key
AWS_REGION: example-us-east-1
BEACON_PROVIDER_AWS_CONSTANT_REGIONS: example-us-east-1
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 are presented with a total of five metrics:

- CPU
- Network Throughput
- Connections
- Disk IOPS
- Disk throughput

Disabling AWS RDS Monitoring in Hybrid Manager

You can turn off the monitoring of your AWS RDS deployments using the following steps.

1. Delete your `beacon-csp-credentials` secret:

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

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 are now no longer seeing your AWS RDS deployments on your Estate and are no longer monitoring them with HM.

Monitoring with the PG AI Console

You can access a comprehensive set of database metrics through the PG AI Console, with availability varying by metric and cluster type.

- ref:*Support scope* <*Support scope*> — Learn more about which type of metrics are supported for your clusters.
- ref:*Metrics latency* <*Metrics latency*> — Learn more about how often metrics are updated.

Support scope

The PG AI Hybrid Manager 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 PG AI Console depends on the metric and cluster type. This page provides an overview of the support scope and tells you where to find each grouping of metrics in the PG AI Console.

Summarized overview

Here is an overview of the available metrics per database cluster type.

	Host metrics	*Postgres metrics*	*Cluster metrics*	*Query metrics*	*Enabling the Migration Portal AI Copilot*	*Enabling Data Catalogs on Hybrid Manager*
HM-managed cluster	yes	yes	yes	yes	yes	yes
External: **Self-managed cluster**	yes	partial support	no	yes	yes	yes

continues on next page

Table 20 – continued from previous page

	Host metrics	*Postgres metrics*	*Cluster metrics*	*Query metrics*	*Enabling the Migration Portal AI Copilot*	*Enabling Data Catalogs on Hybrid Manager*
External: **CNPG database cluster**	no	partial support	partial support	yes	yes	yes
External: **AWS RDS database**	yes	partial support	no	no	no	no

!!!note s

- Self-managed multi-node instances are not displayed as groups at the moment. Each data node instance and its metrics are displayed separately. Clustered views for multi-node instances will be added in a coming release.
- Currently, the Hybrid Manager supports monitoring for EDB CloudNativePG clusters. Monitoring for open-source CloudNativePG clusters will be supported in a coming release.

Detailed overview

Host metrics

Host metrics display resource utilization for the underlying server infrastructure hosting your database instances. They are visible through the **Estate** view, and also through the cluster view ☒ ***Monitoring*** tab ☒ **Host** section.

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Memory	yes	yes	no	yes
CPU	yes	yes	no	yes
Storage	yes	no	no	yes
Disk IOPS	yes	yes	no	yes
Disk throughput	yes	yes	no	yes
Network activity	yes	yes	no	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 *Using Grafana dashboards* . 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 itself. They are visible through the **Estate** view, and also through the cluster view ☒ ***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	no	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	no	no	no
WAL size	yes	no	no	yes
WAL usage	yes	no	no	no
Live/Dead tuples	yes	no	no	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). They are visible through the **Estate** view, and also through the cluster view ☒ ***Monitoring*** tab ☒ **Topology** and **Replication** sections. Cluster metrics are not 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
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

continues on next page

Table 23 – continued from previous page

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Memory (per node)	yes	no	yes	no
Storage (per node)	yes	no	yes	no
WAL size (per node)	yes	no	no	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. They are visible through the cluster view ☒ ***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 (e.g. improve indexes). They are visible through the cluster view ☒ **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	no	no	no	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. They are visible through the cluster view ☒ ***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

The PG AI Hybrid Manager 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 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 the Hybrid Manager's internal services actively collect updated metrics and data from your database clusters. Whether your cluster is HM-managed or an external database, the backend systems are constantly gathering the latest information. This data is then stored in the Hybrid Manager's central data stores.

PG AI Console refresh: This is the second step, where the PG AI 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 Console is a combination of these two refresh intervals. The data on the screen will reflect information that is not older than the sum of these two intervals. In many cases, it will be much fresher, sometimes just seconds old.

Refresh intervals

Backend refresh: The Hybrid Manager'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	2m	2m	na	2m
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 *PG AI Agent configuration YAML* . In the case of CSP-managed clusters, define the intervals for the supported metrics through the *beacon-csp-credentials* .

PG AI Console refresh: The PG AI 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 PG AI Hybrid Manager deployments, HM-managed database clusters and any other external clusters connected to the Hybrid Manager. 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 the PG AI Agent as explained in Monitoring external databases .

Grafana support

Grafana dashboards are supported as follows:

- HM-managed clusters — full support out of the box.
- Self-managed clusters — partial support after the cluster is connected via the PG AI Agent.
- CNP clusters — partial support after the cluster is connected via the PG AI Agent.
- RDS clusters — partial support after the cluster is connected via Kubernetes secret.

Accessing Grafana

You can access the built-in Grafana instance via the *Launchpad* .

Filtering results

Every cluster managed by the Hybrid Manager 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

- ref:*Using Grafana dashboards <Using Grafana dashboards>*
- ref:*Grafana's Explore <Grafana's Explore>*

Using Grafana dashboards

The Grafana service shipped with the PG AI Hybrid Manager comes with a number of default dashboards designed to monitor different Hybrid Manager 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 the PG AI Hybrid Manager comes with a number of default dashboards that are grouped into the following categories:

- ref:*Kubernetes dashboards <Kubernetes dashboards>*
- ref:*Loki dashboards for logging monitoring <Loki dashboards for logging monitoring>*
- ref:*Postgres dashboards <Postgres dashboards>*
- ref:*Telemetry dashboards <Telemetry dashboards>*
- ref:*Other dashboards <Other dashboards>*

!!!note

The dashboard descriptions offer a generalized and exemplified overview of the default dashboards included with Hybrid Manager'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/or 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 a Hybrid Manager upgrade, node Kubernetes version upgrade or general node failure.

Create, use and manage dashboards

Consult the official Grafana documentation for more information on how to create, use and manage dashboards, here are a few useful links:

- ref:*Create a dashboard <Monitoring AWS RDS deployments on AWS>*
- ref:*Import dashboards <Monitoring AWS RDS deployments on AWS>*
- ref:*Modify dashboard settings <Monitoring AWS RDS deployments on AWS>*
- `Manage dashboards <https://grafana.com/docs/grafana/latest/dashboards/manage-dashboards/>`_
- ref:*Share dashboards <Monitoring AWS RDS deployments on AWS>*
- `Use Explore <https://grafana.com/docs/grafana/latest/explore/>`_ to query, collect and analyze data.
- `Use Alerting <https://grafana.com/docs/grafana/latest/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 the PG AI Hybrid Manager.

These are prebuilt dashboards called Kubernetes mixins from Grafana' s official library.

Kubernetes / API server

!!!note

This dashboard only displays data for on-premise PG AI Hybrid Manager 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 / *

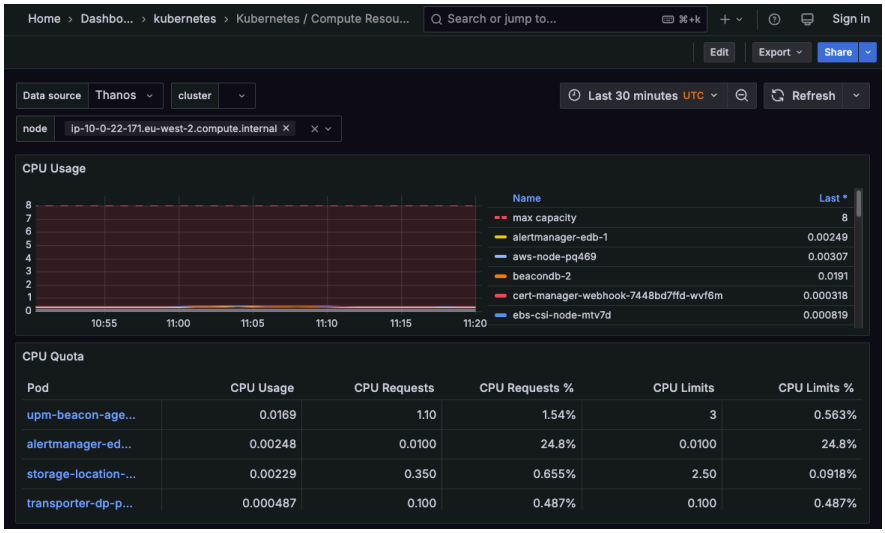


Fig. 23: Compute Resources by Node and Pods

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, etc.), 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 (e.g., 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, focussing on different Kubernetes levels, while offering different filtering options.

Metrics: Metrics include CPU, memory, bandwith, 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 aplication, of specific pods in a node, troubleshoot performance issues with specific application instances, monitor the performance of a particular application component.

Kubernetes / Kubelet

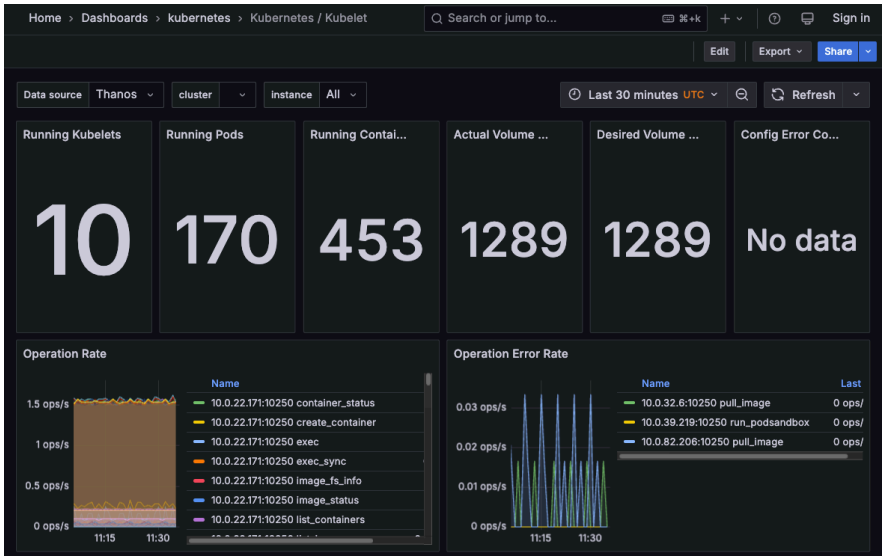


Fig. 24: Kubelet dashboard

Description: Monitors the health and performance of the Kubelet service running on each Kubernetes node, responsible for managing containers. You can see the aggregated performance of all kubelets, filter by kubelet instance and drill down into specific kubelet operations.

Metrics: Metrics include kubelet count, pods count, countainers 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, 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 within that namespace.
- **Namespace (Workloads)** shows network traffic metrics that you can filter by Kubernetes namespaces and by workload type (deployments, daemonsets, StatefulSets, etc.), focusing on individual workload traffic.

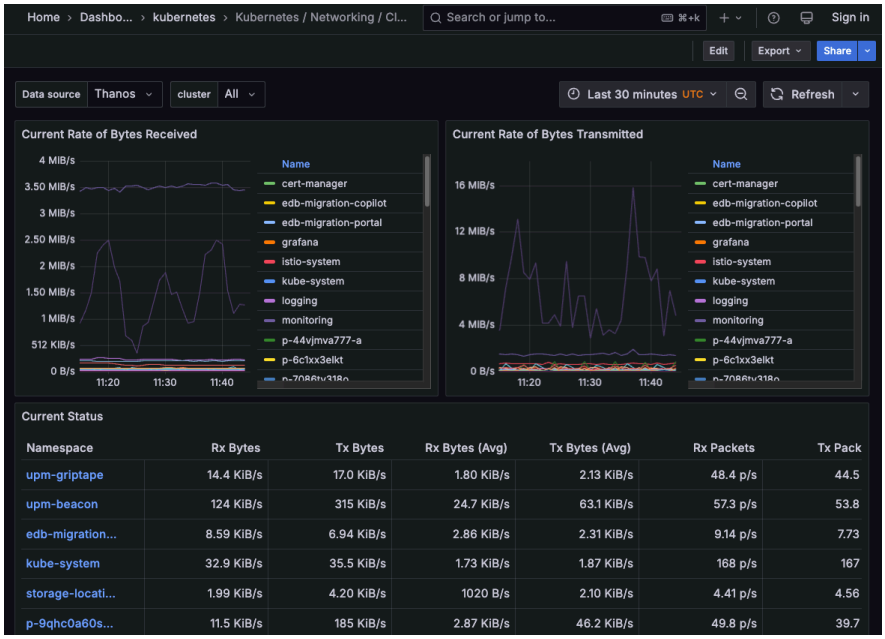


Fig. 25: Networking overall cluster dashboard

- **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 (e.g., a deployment workload).

Description: Shows the network usage at the Kubernetes cluster level. Each of the subdashboards presents the data differently, focussing on different Kubernetes levels, while offering different filtering options.

Metrics: Metrics include rates of bytes received and transmitted, rates of packets received and transmitted, receive and transmit bandwith, 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 aplication, troubleshoot network connectivity issues with specific pods in an application, or 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, capacity issues, plan for storage expansions and monitor storage costs.

Kubernetes / StatefulSets

Description: Provides specific monitoring for Kubernetes StatefulSets , which manage stateful applications. You can filter results by namespace (=application) and StatefulSet name.

Metrics: Metrics include CPU, memory, and network consumption, as well as replicas.

Usage: Use this to monitor the stability, performance and resource consumption of your stateful applications, as well as track whether your stateful applications have met their desired replicas.

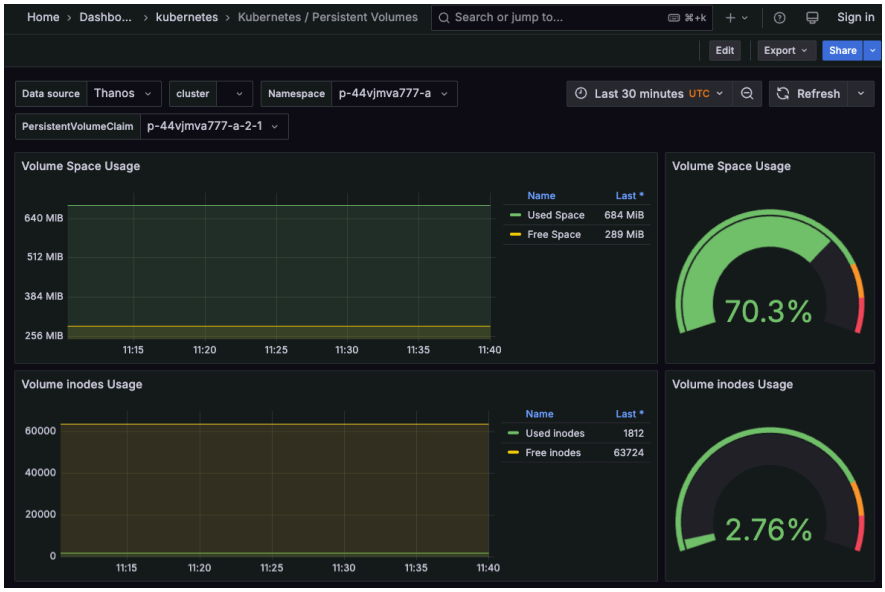


Fig. 26: Persistent Volumes dashboard

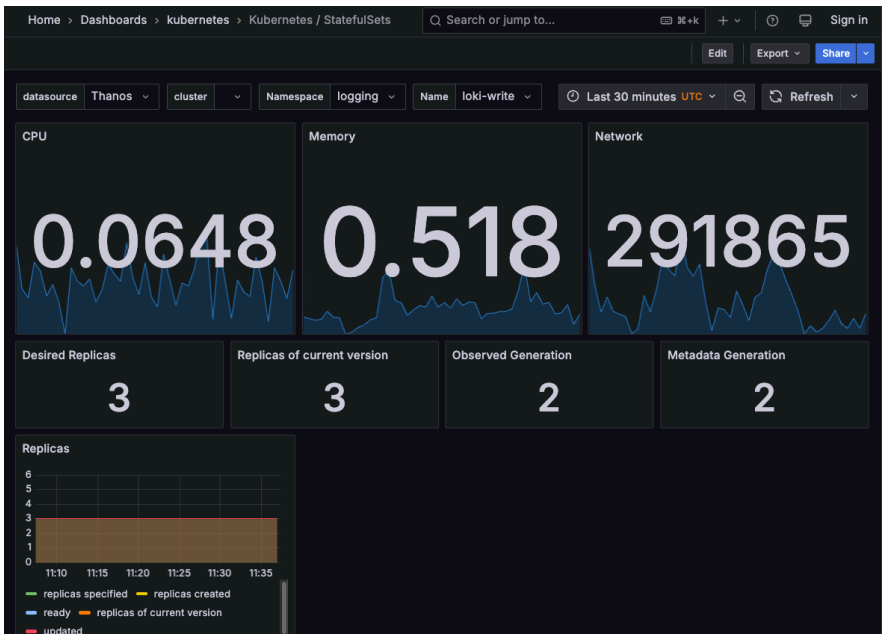


Fig. 27: StatefulSets dashboard

Kubernetes CoreDNS



Fig. 28: CoreDNS dashboard

Description: Monitors the performance and health of the CoreDNS service, which provides DNS resolution within the Kubernetes cluster, enabling communication within the pods and services.

Metrics: The metrics include DNS request rates by protocol, by query type, by DNS zone, by size, by response code, by response duration, by response size, among others. **Usage** Use this to monitor the reliability and efficiency of the CoreDNS component and identify any discovery and communication issues among services.

Loki dashboards for logging monitoring

These dashboards are related to the integrated Loki logging system, allowing you to visualize and analyze your logs. Visualization is limited to the `logging` namespace only, as it is the place where Loki performs its operations.

Loki dashboards are prebuilt dashboards called Loki mixins from Grafana's official library.

Loki / Chunks

Description: Monitors the storage and management of log data chunks (a grouping of log entries that belong to a single log stream) within 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 set to 30 days.

Metrics: Deletion duration, deletion errors, number of chunks deleted, number of series affected by deletion, deletion queue length.

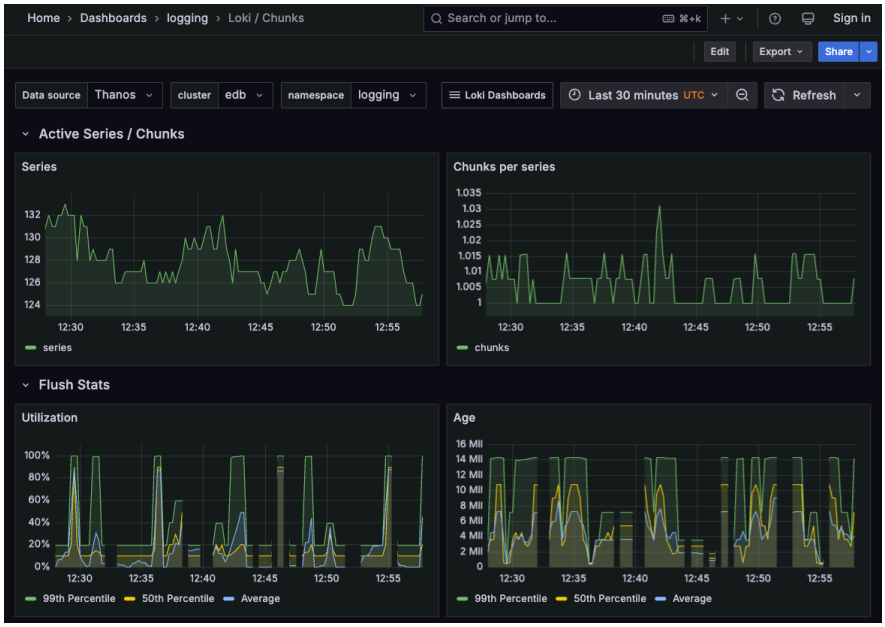


Fig. 29: Loki Chunks dashboard

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, errors. Use this for real-time log analysis and troubleshooting.

Loki / Operational

Description: Provides insights into the operational health and performance of the Loki system itself.

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 within Loki.

Loki / Reads

Description: Monitors the performance of log read queries against Loki.

Metrics: Loki read queries per second, latency per read operation, and per pod.

Usage: Allows you to assess the responsiveness of Loki to log queries.



Fig. 30: Loki Operational dashboard

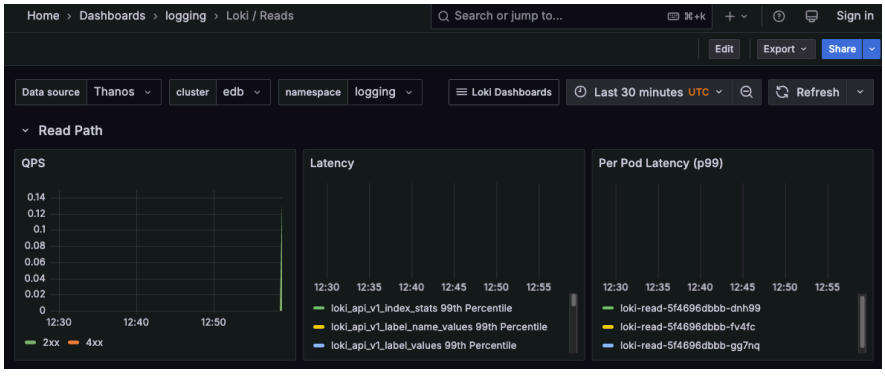


Fig. 31: Loki Reads dashboard

Loki / Reads Resources

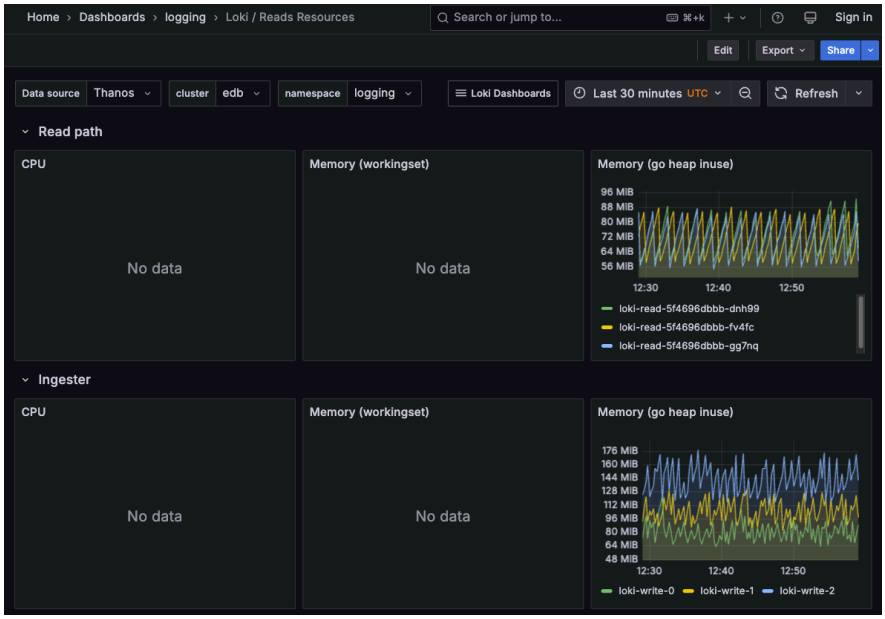


Fig. 32: Loki Reads Resources dashboard

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 within Loki.

Loki / Writes

Description: Monitors the performance of log ingestion (writes) queries into Loki.

Metrics: Loki write queries per second, latency per write operation, and per pod.

Usage: Helps you identify slow ingestion rates or dropped logs in the log forwarding pipeline from your applications to Loki.

Loki / Writes Resources

Description: Shows resource consumption (CPU, memory) by Loki write operations.

Metrics: Memory in use by individual Loki write pods and by Loki ingesting pods.

Usage: Helps you understand the resource overhead of log write querying.

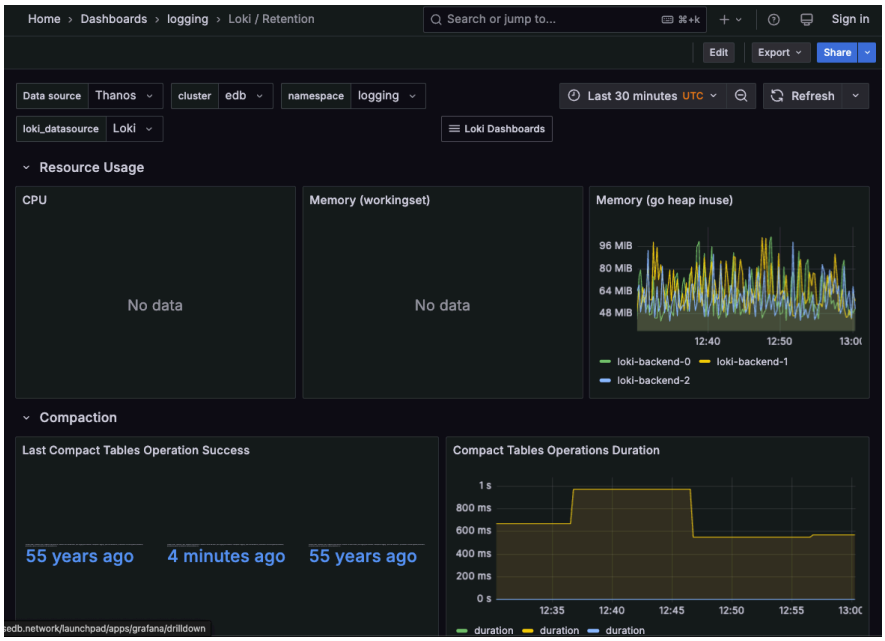


Fig. 33: Loki Retention dashboard



Fig. 34: Loki Writes dashboard

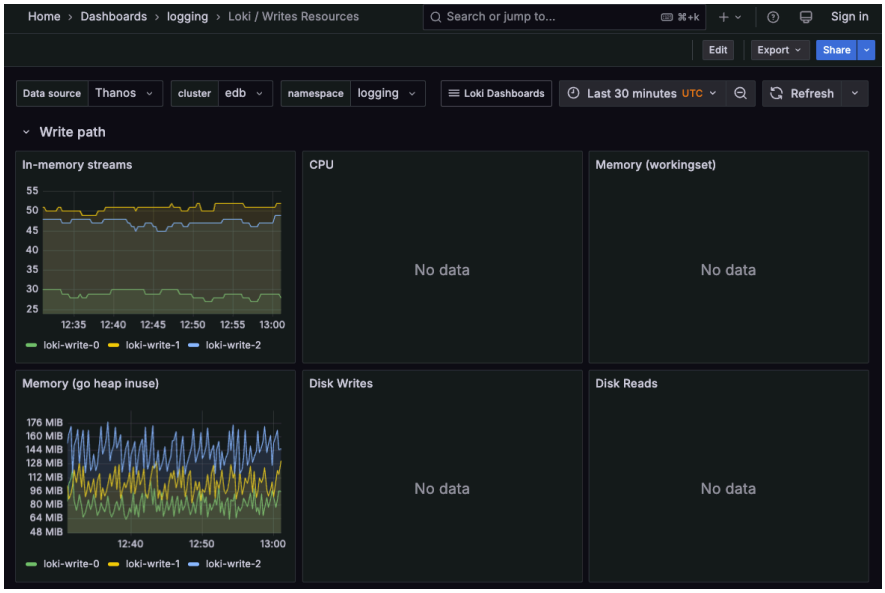


Fig. 35: Loki Writes Resources dashboard

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 the PG AI Hybrid Manager.

Hardware Utilisation

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, see if you have oversubscribed, 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 is 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 (e.g., 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. 36: Loki Data dashboard

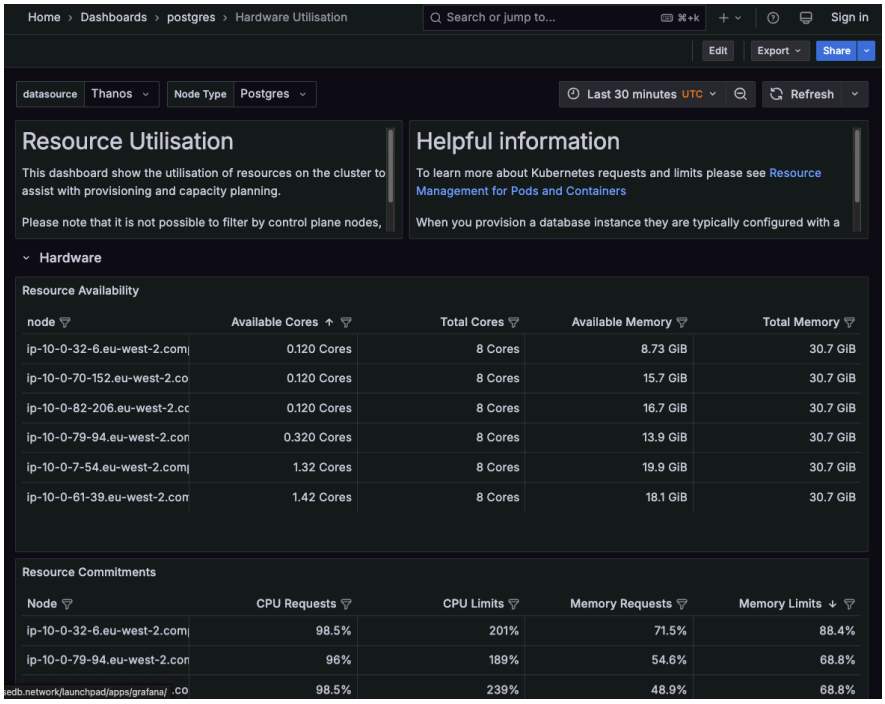


Fig. 37: Postgres Hardware Utilisation dashboard

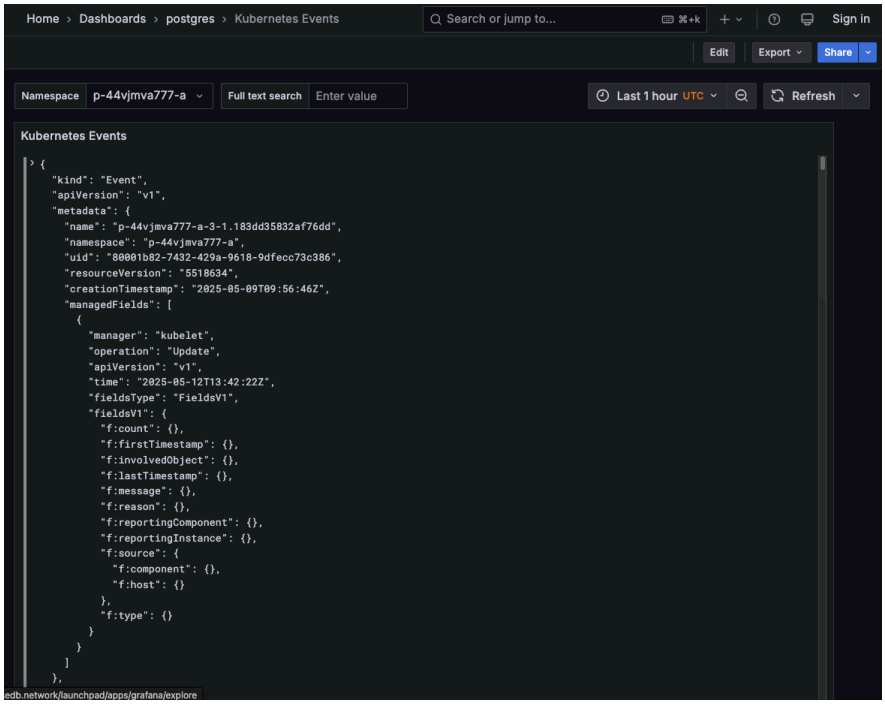


Fig. 38: Postgres Kubernetes Events dashboard

PGD overview

For Distributed Hidh Availability (DHA or PGD) clusters only.

Description: Provides an overview of how your Distributed High Availability (PGD) cluster is performing focussing 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 the clusters you are running.

Metrics: List and count of all running clusters, theur age and latest restore stamp.

Usage: Provides a high-level overview of the clusters under your **Estate**.

Postgres Logs

Description: You can view Postgres server logs.

Metrics: Logs in text form.

Usage: Provides a visual overview of Postgres log data, enabling quick identification of trends, errors. Use this for real-time log analysis and troubleshooting.

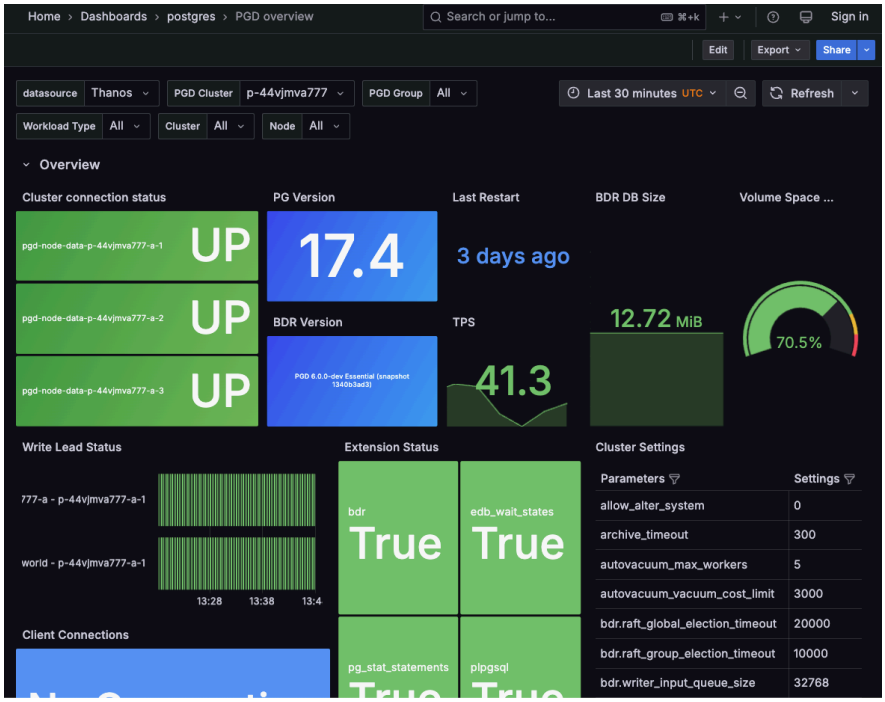


Fig. 39: Postgres PGD Overview dashboard

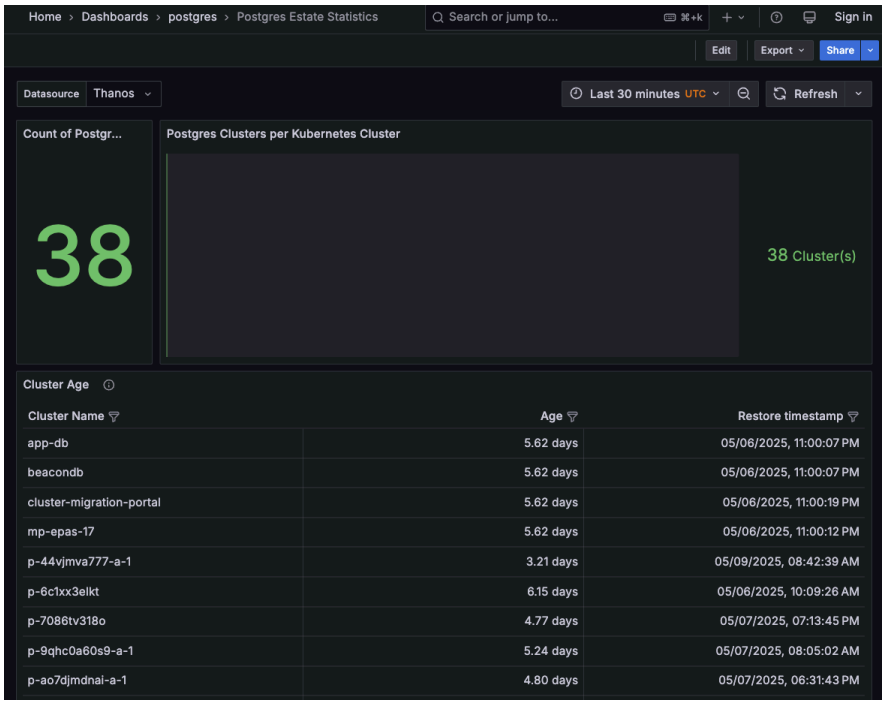


Fig. 40: Postgres Estate Statistics dashboard

Postgres Status

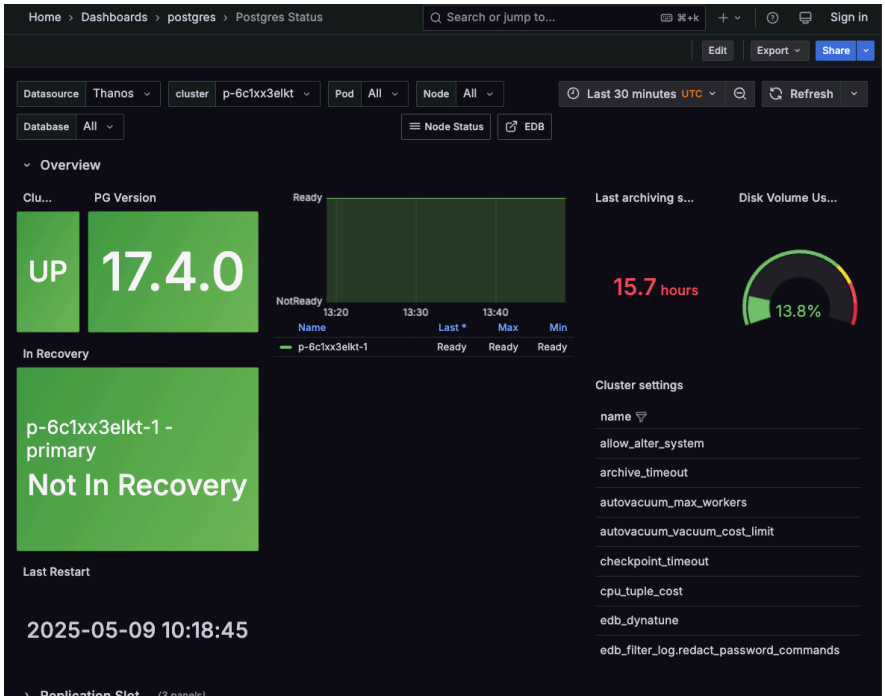


Fig. 41: Postgres Status dashboard

For all cluster types (sinle-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, WALfiling 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 PG AI Hybrid Manager platform itself and its underlying components.

FluentBit

This is a prebuilt *Fluent Bit mixin* from Grafana's official library.

Description: Provides metrics about the FluentBit instance itself, which is used to collect logs from the nodes, and ship them to Loki, which in turn informs the PG AI 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 the PG AI Hybrid Manager components.

Grafana Overview

Description: Provides metrics about the Grafana instance itself, 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 are using to view these dashboards. Identify if Grafana itself is overloaded or experiencing issues.

Memcached

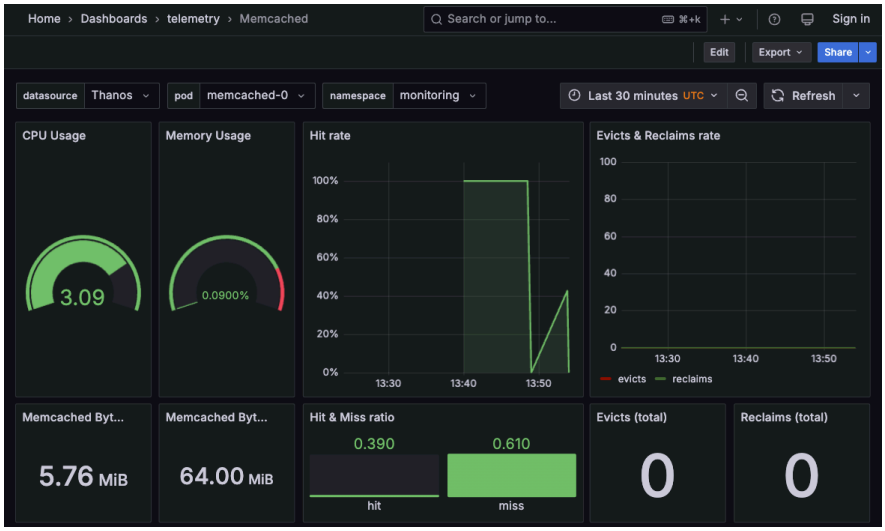


Fig. 42: Telemetry Memcached dashboard

Description: Monitors the health and performance of Memcached monitoring services within the PG AI Hybrid Manager infrastructure, providing insights into resource utilization, cache efficiency, and eviction rates.

Metrics: CPU and memory usage, hit and miss ratios, evicts & reclaims rate, memcached Bytes, and more.

Usage: Use this dashboard to understand the resource consumption of your Memcached instances, evaluate the effectiveness of the cache (hit rate), and monitor how frequently items are being evicted or reclaimed due to memory pressure.

Node Exporter / *

These are prebuilt Node mixins from Grafana's official library.

Node Exporter metrics are displayed in multiple dashboards that provide different views: **Nodes**, **USE Method / Cluster**,and **USE Method / Node**.

- **Nodes** provides the aggregated resource utilization for a specific node, broken down into types of CPU usage, loads, type of memory, and more.
- **USE Method / Cluster** displays the aggregated resource utilization for the entire Kubernetes cluster, with per-node colorization.
- **USE Method / Node** displays the total resource utilization per node and some metrics for non-volatile memory devices.

Description: Shows hardware metrics for nodes running any of the PG AI Hybrid Manager components.

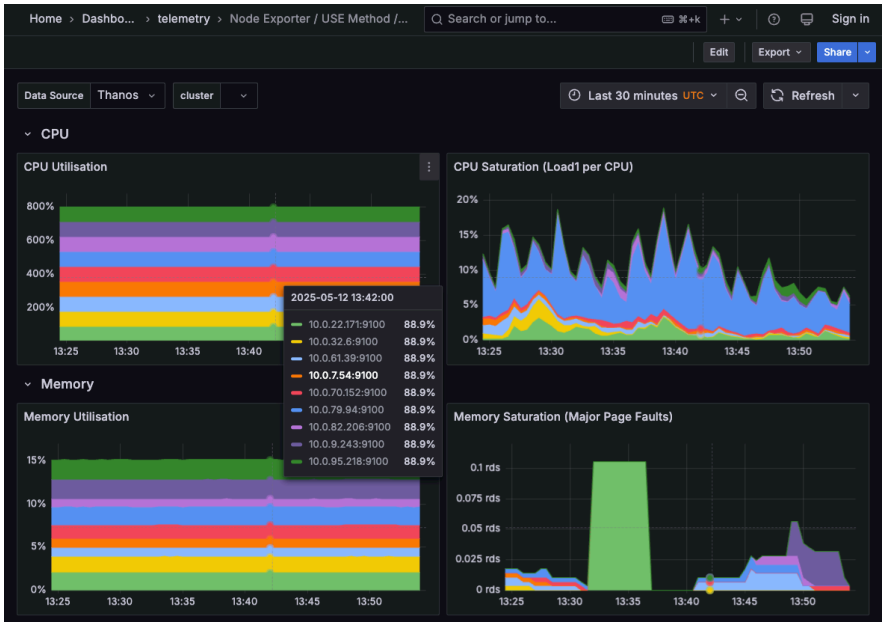


Fig. 43: Telemetry Node Exporter dashboards

Metrics: CPU utilization, load average, memory usage, disk usage, and more.

Usage: Provides several views of the hardware resources of the servers hosting the PG AI Hybrid Manager. Useful for identifying resource bottlenecks at the node level.

Prometheus Overview

This is a prebuilt Prometheus mixin from Grafana's official library.

Description: Monitors the health and performance of the Prometheus monitoring system, which is used to collect and ship metrics within the PG AI Hybrid Manager.

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 within the PG AI Hybrid Manager.
- **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, and cache operations, as well as any related errors.

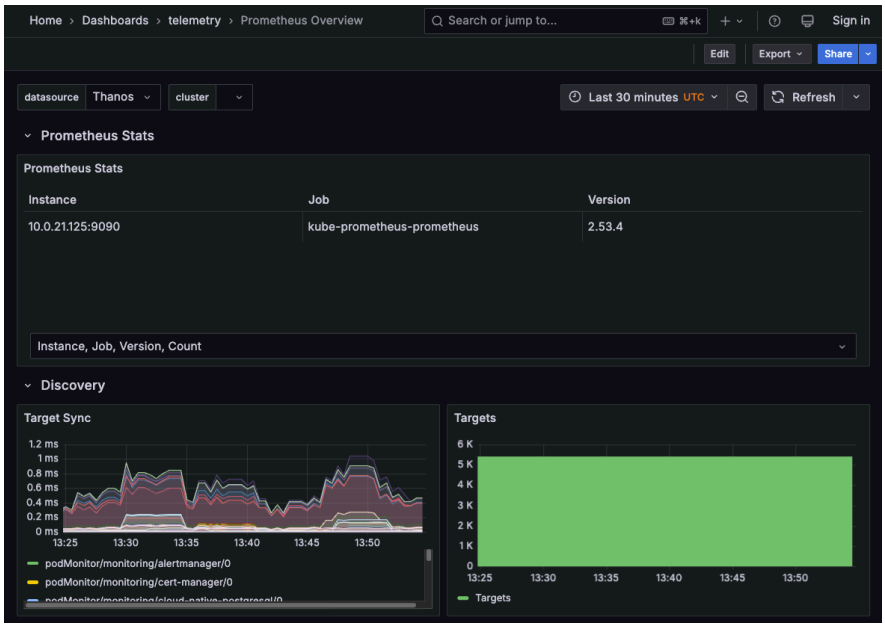


Fig. 44: Telemetry Prometheus dashboard

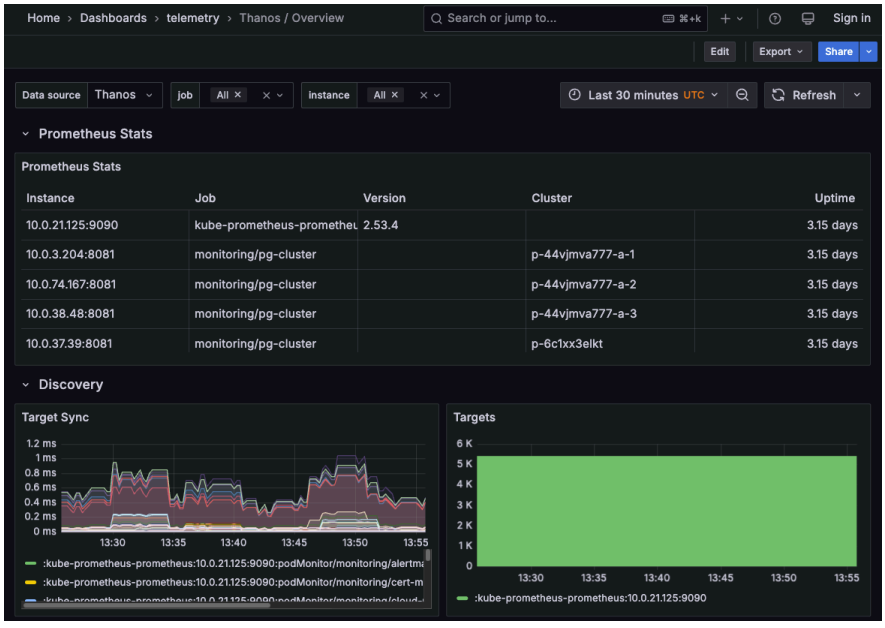


Fig. 45: Telemetry Thanos dashboards

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, 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 impact the responsiveness of Grafana dashboards, etc.

Other dashboards

Power Consumption: Power Usage & Carbon Footprint

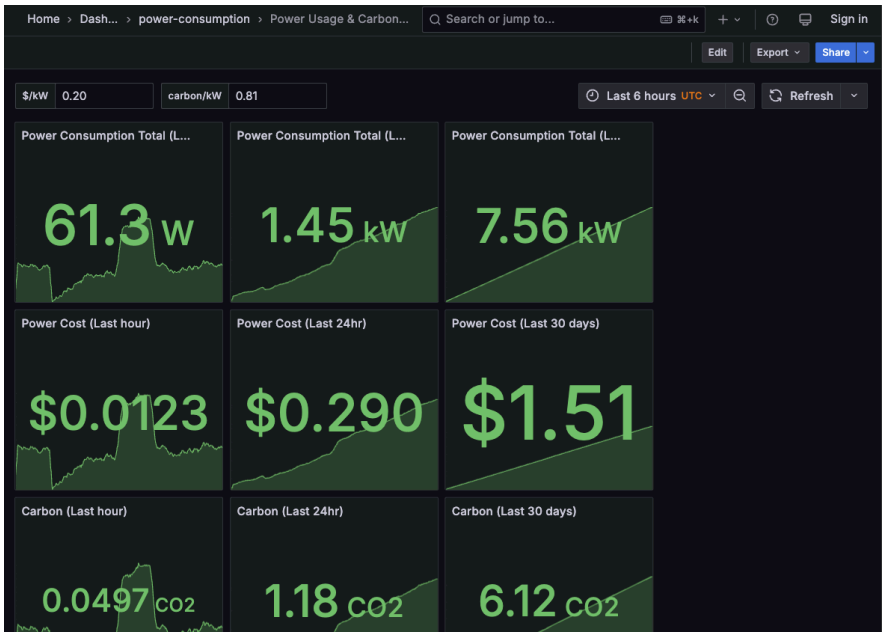


Fig. 46: Power usage and carbon footprint dashboard

Description: Monitors the energy consumption and estimated carbon footprint of the PG AI Hybrid Manager 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 methology and uses the freely available power measurements from TEADS dataset as source data.

Metrics: Power consumption (in Watts or Kilowatts), estimated carbon footprint and estimated power cost, broken down into graphs for the last hour, 24 hours, or 30 days.

Usage: Allows you to visualize the environmental impact of the HM infrastructure, track trends in energy usage, and potentially identify areas for optimization to reduce costs and environmental impact.

Infrastructure: cert-manager

Description: Monitors the health and performance of cert-manager , a Kubernetes add-on to automate the management and issuance of TLS certificates. Use this to ensure the proper functioning of certificate management within the PG AI Hybrid Manager.

Metrics: Number of issued certificates, certificate request status (pending, ready, failed), expiry times of certificates, and more.

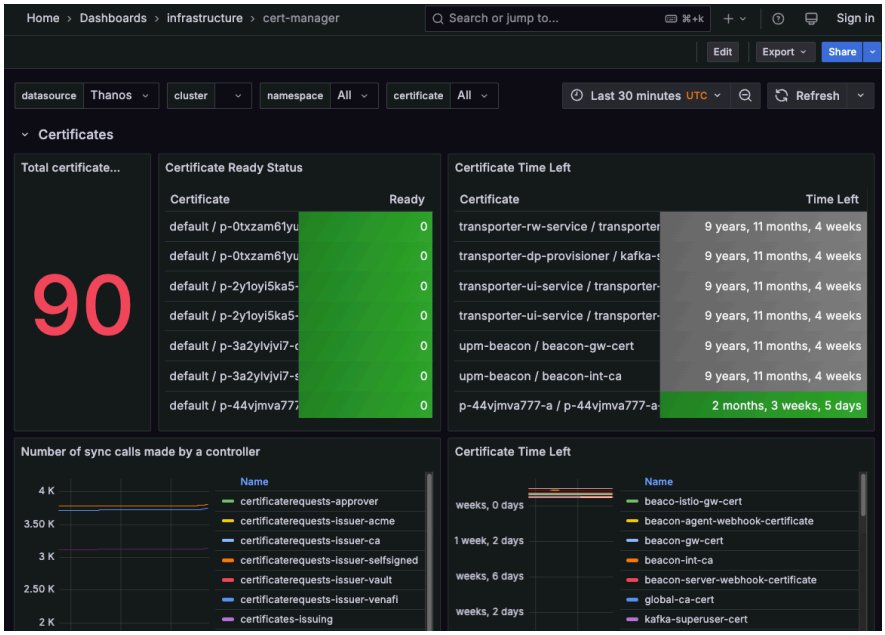


Fig. 47: Cert-manager dashboard

Usage: Helps ensure that TLS certificates within the PG AI Hybrid Manager 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 within the HM.

Metrics: Number of backups created, number of restores performed, status of backup and restore operations (completed, failed, in progress), backup size, restore duration, errors encountered during backups or restores, schedule status.

Usage: Provides visibility into the backup and restore operations within the PG AI Hybrid Manager. Allows you to verify the success of backups, monitor restore processes, and identify any issues with your disaster recovery strategy.

Grafana's Explore

Grafana offers an **Explore** option which 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** dropdown, select **cnp_instance_id**.
3. In the **Select value** dropdown, select the pods ID.
4. Craft your query to look for certain key words in that instance's logs, for example "error" or "exception.

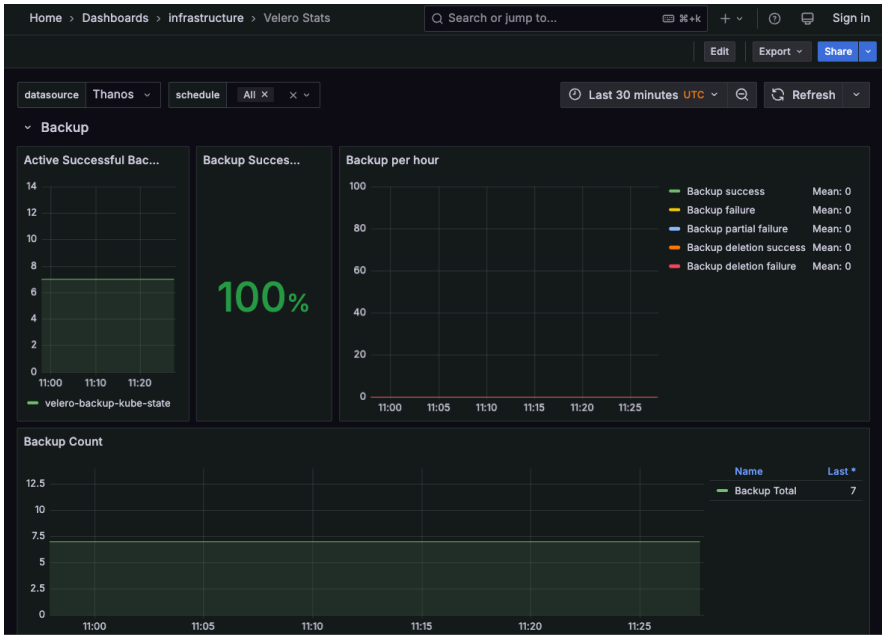


Fig. 48: Velero dashboard

Notifications

With EDB Postgres AI, you can customize the types of notifications you want to receive. There are two types of notification channels:

User notifications channels — you can receive notifications via email, in-app inbox, or both. These notifications are for all users who have enabled them and if their role allows it.

System notifications channels — you can receive notifications via email (mail group), webhook, pager duty, or any combination. These notifications are designed to be triggered once per event, primarily to automate tasks within your organization.

The user's role in the Hybrid Manager determines the extent of their access to notification preferences. The notifications are categorized into the following preference sections:

- Account — only for user notification channels
- Global — for both user and system notification channels
- Projects — for both 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 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 organisation owner or at the project level by the project owner. By default, each organisation's project inherits the global-level system notifications channel. However, you can enable/disable or re-configure the system notifications channel at the project level to override the default settings.

These steps show how to configure the system notification channels at Organization-level:

Configure Notification at the Global level

The Global level notifications can be set by Organization owner only.

Webhook:

1. Log into the EDB Postgres AI console.
2. From the menu under your name in the top-right panel, select Settings.
3. Select the Notifications tab. Options available are Webhook, Email, and Pager Duty.
4. Select Webhook and provide the details on the pop-up window:

i. Host URL of the Webhook

ii. Method of the Webhook — POST or PUT

iii. Optionally, provide one of the following authentication parameters

- For Basic Auth: 1. Username 2. Password
- For Bearer Token: 1. Bearer Token

iv. Select the Save button.

Pager Duty:

1. Log into the EDB Postgres AI console.
2. From the menu under your name in the top-right panel, select Settings.
3. Select the Notifications tab. Options available are Webhook, Email, and Pager Duty.
4. Select Pager Duty and provide the details on the pop-up window:
5. Set Routing Key(integration key).
6. Select the Save button.

!!!note

Before configuring the Pager Duty notification channel:

- You need to have access to a Pager Duty account with elevated privileges. - Pager Duty Service was created and configured with Events API v2 integration. - In your Pager Duty account, navigate to the desired Service and access its “Integrations” settings. - Add a new “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.

Email:

1. Log in to the EDB Postgres AI console.
2. From the menu under your name in the top-right panel, select Settings.
3. Select the Notifications tab. Options available are Webhook, Email, and Pager Duty.
4. Select Email and provide the details on the pop-up window:
5. Email To — Optionally, for the system email notifications (Email group)
6. Email From — for the email which is sending the email
7. Host — URL of the email service
8. Port — port of the email service
9. Username — username of the email service
10. Password — of the email service
11. Select the Save button.

Configure Notification at Project level

Project-level notification settings can override global settings. Each project can have its unique configuration.

1. Log in to the EDB Postgres AI console
2. Select the Projects tab on the top
3. Select any Project
4. Go to the Project Overview page
5. Select Settings on the left-side menu
6. Select Notifications
7. Follow the steps mentioned in the global notification setting 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 EDB Postgres AI.

Select the bell icon at the top of your EDB Postgres AI 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

To manage the notification preferences:

1. Log into the EDB Postgres AI console.
2. From the menu under your name in the top-right panel, select My Account.
3. Select the Notifications tab. Notifications are grouped by account, global, and projects available to you.
4. Scroll to the Account, Global or Project section to view the events you wish to manage.
5. Select the right arrow next to the entity to expand the list of events.
6. For each notification, you can: - You can use the toggle button in front of the event name to opt out of or into receiving notifications for a particular event. - Customize the email and in-app notifications for the event by using the checkbox 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 the 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.

1. For HM clusters:

- Go to Console, and on the top select Projects
- Select any project from the list
- In the left navigation, go to Clusters
- Select any cluster with a ready state
- On the Cluster detail page, select the Monitoring tab

2. For external HM clusters:

- Go to Console, and on the top select Estate
- Go to the table listing all the clusters
- Select a ready cluster with Management as Self-managed
- 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
- the Date Range Selector for a specific date range-based search

- the Filter for a severity level-based search
- the Sort utility, you can sort the alerts list based on severity, alert start time, and alert name
- the Refresh button, to refresh the alerts list
- the Column Selector dropdown, you can select the columns to be displayed in the table according to your preferences
- the Download button, to download the alerts list in a CSV format

You can use the Refresh button 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 automatically 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

Alerts can be configured 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. Sign in to the EDB Postgres AI Console .
2. On top of the console, Select Projects
3. Select any project from the list
4. In the left navigation, go to Setting
5. 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 are exceeded.

The list of the alerts with the default configuration:

The alerts that can be configured at database level:

- Connections Usage
- No Recent WAL Archiving Successes

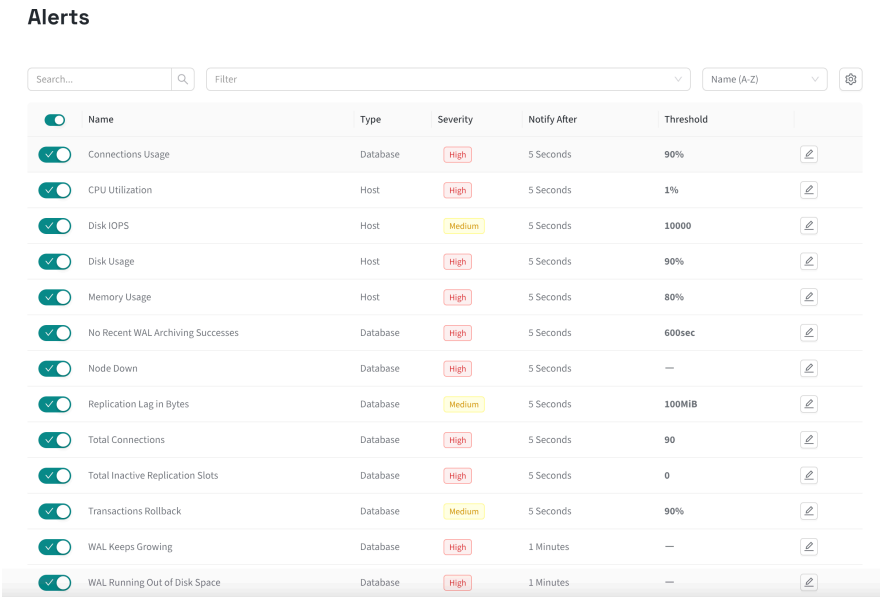


Fig. 49: Alerts

- Node Down
- Replication Lag in Bytes
- Total Connections
- Total Inactive Replication Slots
- Transactions Rollback
- WAL Keeps Growing
- WAL Running Out of Disk Space

The alerts that can be configured at host level:

- CPU Utilization
- Disk IOPS
- Disk Usage
- Memory Usage

!!!note

- Any changes made to the alert configuration are saved automatically. - All the alerts are not available for external HM clusters due to the technical limitations of making them available.

PGAIHM High Availability & Disaster Recovery (HA/DR)

Learn about HA/DR for your PGAIHM clusters, best practices, and more.

- ref:*HA/DR planning and best practices <HA/DR planning and best practices>*
- ref:*PGAIHM Backup and Recovery <HA/DR planning and best practices>*

HA/DR planning and best practices

Overview

You can configure a multi-data center topology using Hybrid Manager (PGAIHM) using PostgreSQL Distributed (PGD) deployments to meet your high availability and disaster recovery (HA/DR) requirements. This topology includes a primary PGAIHM and one or more data-only PGAIHMs across multiple Kubernetes clusters.

In this HA environment, the primary PGAIHM is a standard PGAIHM deployment. The data-only PGAIHM is essentially the same PGAIHM deployment, with backend adjustments to connect it to the primary PGAIHM, and to prevent accidental local provisioning. A key component facilitating this connection is the beacon agent and beacon server, which handle communication and certificate passing between the PGAIHMs.

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 PGAIHM can manage databases across multiple remote Kubernetes clusters, simplifying database administration.

Topologies

The multi-PGAIHM setup can be deployed with the following topologies:

- **Primary and Data-Only PGAIHM:** This is the basic setup with one primary PGAIHM and at least one data-only PGAIHM. It allows for PGD deployments across two or more Kubernetes clusters.
- **Multiple Data-Only PGAIHMs:** The system can support multiple data-only PGAIHMs, allowing a single primary PGAIHM to manage databases across many Kubernetes clusters. This is useful for organizations with numerous application teams, each with their own cluster.
- **Three PGAIHMs for PGD:** For optimal PGD deployments, especially with two data groups and a witness group, a three-PGAIHM topology is recommended. This ensures an odd quorum for proper distributed consensus.

Multi-data center deployments include a primary data center containing the main PGAIHM instance and a secondary data center potentially with a data-only PGAIHM or other Kubernetes clusters, along with a global object store accessible from multiple data centers.

> **Note**
>
> PGAIHM version 1.2 supports only a single PGAIHM. Advanced topologies with multiple PGAIHMs are planned for future versions.

> **Note**
>
> Cross data center HA: High availability across data centers or regions is not 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 PGAIHM on one of the Kubernetes clusters.
- A data-only PGAIHM installed on the other Kubernetes cluster(s), using a provided script for configuration.
- When spanning databases across the multiple clusters, PGD must be used.

- 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 are not properly tuned. Symptoms may include slow failure detection and recovery. Adjust raft-related settings to accommodate higher network latency.

Note

For single standalone or replica clusters (non-PGD), clusters can only be deployed entirely within one location. Spanning of replicas across locations isn't supported.

Network Latency

There is 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 PGAIHM user interface (UI).

Latency is less of a concern in on-premise, 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, particularly raft settings, that are not exposed in the UI. Use the `bdr run on nodes` function to execute commands on other nodes.

Kubernetes and failovers

Kubernetes handles failovers and redundancy within PGAIHM. 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 configuration 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 PGAIHM 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 PGAIHM 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. Object storage is highly recommended for disaster recovery scenarios, especially for data center failures.

- For stretch clusters, network latency should 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 is not automatically failed over when a node fails. Recovery requires restoring from backup.
- Utilizing appropriate storage solutions like shared storage (e.g., Ceph, SAN) or object storage is critical for data persistence and recovery.
- PGAIHM services themselves may 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.
- PGAIHM leverages Kubernetes capabilities for HA, but it's important to consider PGAIHM-specific configurations.

PGAIHM Backups and DR

PGAIHM 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 PGAIHM backups:

- A global object storage solution is recommended for storing Barman backups. This 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 PGAIHM itself, 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 PGAIHM backups:

- Restoring backups when PGAIHM 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 PGAIHM may require manual steps.
- A primary PGAIHM instance is required for managing backups and restores.
- Automated backups can only be sent to one location. The system does not automatically replicate backup definitions across locations.
- You can set up Barman or volume snapshots for automatic backups, but not both concurrently through the UI.
- If the primary PGAIHM 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 PGAIHM.
- Backup metadata is local to each Kubernetes cluster, so backups taken in one cluster are not automatically visible in another.

Using

The PGAIHM user interface 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 within a Kubernetes cluster.

For information on how to configure backups and restores of your PGAIHM cluster, see *PGAIHM Backup and Recovery* .

PGAIHM HA/DR Recovery

In the event of a disaster that impacts the appliance and makes it unusable, such as unavailability of the CSP region for an EKS-based appliance, or an outage in a datacenter that makes a hardware appliance unusable, a disaster recovery (DR) option allows you to have your databases restored at the point-in-time from your available backups.

PGAIHM backups are handled with Velero .

There are two possible scenarios for recovering your PGAIHM:

- **Restore PGAIHM to Original Location**: You have two data centers (DC1, DC2), and the PGAIHM runs in DC1. You need to restore PGAIHM from object storage to DC1.
- **Restore PGAIHM to Alternate Location**: You have two data centers (DC1, DC2), and the PGAIHM runs in DC1. You need to restore PGAIHM from object storage to DC2.

DR scope

The DR procedures address the following:

- The Postgres clusters that you' ve 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 do not 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, and the associated recovery time objective (RTO), and recovery point objective (RPO), depends 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, RTO and RPO values are not known in advance. It's recommended 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* (PGAIHM 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, replication can be achieved by using Cross-Region Replication

Postgres databases use continuous backup by default, so they can be restored at any point-in-time, and are only limited by back-up 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 backup 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
```

(continues on next page)

(continued from previous page)

```
    \- 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 need to be taken 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 feature set changes. It must be constantly tested and updated 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.

This can be achieved 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, called `customer-pg-backups`
- Any folder corresponding to a defined *custom storage location*

> **Note**
>
> 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 will be a read-only location to prevent an overwrite or removal of those backups.

To define a new storage location, use the following Kubernetes manifest:

```
apiVersion: velero.io/v1
kind: BackupStorageLocation
metadata:
  annotations:
```

(continues on next page)

(continued from previous page)

```
    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, the latest available completed backup would be chosen 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 will be referred as the **recovery date** in the rest of the document

Additional requirements

The following requirements apply to the recovery procedure:

- The new appliance will 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 not available in the UI, with `managed-devspatcher` being 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, use the following procedure:

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
    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\>
```

- Delete the cluster:

```
kubectl delete cluster \<cluster-name\>)
```

- 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
```

- Apply the yaml file for the cluster to be recreated: `kubectl apply \-f \<cluster-name\>.yaml`
- 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 should be 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 and the Postgres clusters resources are restored as deleted, so data can be later restored as desired.

The plugin configuration is made through a `ConfigMap` , so this manifest must be applied:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: velero-plugin-for-edbpgai
  namespace: velero
  labels:
    velero.io/plugin-config: ""
    enterprisedb.io/edbpgai-plugin: RestoreItemAction
data:
  \# configure disaster recovery mode, so restored items are transformed as needed
  drMode: "true"
  \# configure a date corresponding to the velero backup date. Note the format\!
  drDate: "\<recovery date in YYYY-MM-DDTHH:MM:SSZ format\>"
  \# 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:
  \# 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:
```

(continues on next page)

(continued from previous page)

```
  \# 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:
  \# Change the backup name to a custom backup name as required
  backupName: \<velero-backup-name\>
  includedResources:
  \- backupwrappers.beacon.enterprisedb.com
  restoreStatus:
    includedResources:
    \- backupwrappers.beacon.enterprisedb.com
```

Enabling the Migration Portal AI Copilot

After deploying the EDB Postgres AI Platform with Hybrid Manager (HM), Migration Portal , a Launchpad tool to help you migrate external databases, works out of the box.

However, the AI Copilot , a chat-bot tool that helps you resolve migration compatibility issues within the Migration Portal, is not 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.

Prerequisites

Prepare the resources your environment will require 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 are using OpenAI, you need your organization' s ID and API key.
- If you are using Azure OpenAI, you need your organization' s endpoint URL and an API key. Additionally, 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 the HM.

1. Check if the `edb-migration-copilot` exists:

```
kubectl get namespaces edb-migration-copilot
```

The namespace is created during the installation of the Hybrid Manager. If you are enabling the AI Copilot before installing the 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:

For OpenAI:

```
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
```

For Azure OpenAI:

```
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:

For OpenAI:

```
kubectl create secret generic ai-vendor-secrets \
  --namespace=edb-migration-copilot \
  --type=opaque \
  --from-literal=AI_VENDOR=OPENAI \
  --from-literal=OPENAI_ENDPOINT="${OPENAI_ORG_ID}" \
  --from-literal=OPENAI_API_KEY="${OPENAI_API_KEY}"
```

For Azure OpenAI:

```
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 new 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.

For OpenAI:

```
parameters:
  edb-migration-copilot:
    ai_vendor_secrets: ai-vendor-secrets
```

For Azure OpenAI:

```
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 is the recommended version for the copilot.
```

6. Reference the created file with when running the `helm upgrade` command during the *Apply the secrets to your installation* , so it uses the customized values for the Migration Portal. Use this option each subsequent time you invoke `helm upgrade` . Otherwise, your secrets will be overridden with the vendor's default secrets.

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

Enabling Key Management Systems 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` :

- ref:*Adding KMS support <Adding KMS support>*

Then, configure your Key Management System provider:

- ref:*HashiCorp Vault <HashiCorp Vault>*
- ref:*AWS KMS <AWS KMS>*
- ref:*Creating the Google Cloud KMS key <Creating the Google Cloud KMS key>*

Adding KMS support

When you deploy the Hybrid Manager, 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

- ref:*Configure HashiCorp Vault as the KMS for TDE <HashiCorp Vault>*
- ref:*Configure AWS Key Management Service as the KMS for TDE <AWS Key Management Service (KMS)>*
- ref:*Configure Google Cloud Key Management Service as the KMS for TDE <Google Cloud Key Management Service (Cloud KMS)>*

HashiCorp Vault

Before you can create TDE-enabled clusters with a HashiCorp Vault key, an administrator must install and configure Vault for the Hybrid Manager instance.

This example installs a customized version of Vault on the same Kubernetes environment as the Hybrid Manager, configures the Vault provider, and then uses the key to enable TDE for provisioning a new database cluster with encryption enabled.

If you're aiming for a vendor-neutral key management solution, it can be practical to use HashiCorp Vault, as it provides consistent, centralized key management across diverse cloud providers and on-premises environments, preventing vendor lock-in.

Installing Vault

Install HashiCorp Vault on the same Kubernetes environment where you have deployed the Hybrid Manager. Ensure you install it on its own `vault` namespace.

1. Set up the Helm repository:

```
helm repo add hashicorp https://helm.releases.hashicorp.com
```

2. Create a customization override for the installation to ensure Vault is installed on the Hybrid Manager's control plane, port 8200:

```
cat > helm-vault-raft-values.yml <<EOF
injector:
 tolerations: |
   - key: "edbaiplatform.io/control-plane"
     effect: NoSchedule
     operator: Equal
     value: "true"

 affinity: |
   nodeAffinity:
     preferredDuringSchedulingIgnoredDuringExecution:
     - preference:
         matchExpressions:
         - key: edbaiplatform.io/control-plane
           operator: Exists
       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
```

(continues on next page)

(continued from previous page)

```
      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 the PG AI Hybrid Manager using the PG AI 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"
    ],
```

(continues on next page)

(continued from previous page)

```
    "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.

Troubleshooting PGAIHM

This guide provides instructions for troubleshooting the Hybrid Manager (PGAIHM).

Logfiles

The following log files can be used when troubleshooting the PGAIHM:

Log file name	Category	Location	Explanation

Error codes

Error Code	Category	Error Message	Explanation	Potential Solution
na	Installation	[upm-istio] - failed to install components: [upm-istio]	An error from boot-strap: the LoadBalancer service is pending.	Fix the loadbalancer IAM role first, then delete the LB pod to trigger recreation.
na	Installation	upm-beaco-ff-base install failed		kubectl edit sc gp2
na	Installation	upm-thanos installation failed		
na	Installation	Failed to pull image		
na	Installation	Loki pods crash	The loki pods crashed and the installation failed.	
na	Portal Login	No healthy upstream	“No healthy upstream” is returned after the login.	
401	Portal Login	401 Access Denied	401 error returned when logging into the portal after the installation	
500	Portal Login	HTTP Error 500	The back ingress gateway can't access incoming connections.	
na	Database Provisioning	Postgres Cluster Provisioning Stuck at X% Complete		
na	Database Provisioning	CPU and Memory range is much smaller than the node-group		
na	Database Update	Disk scale up	The new value is shown on cluster yaml but not applied on pvc/pv.	Edit the storage-class

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:
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 or not:

```
kubectl get secret -n monitoring
```

The secret should match what you defined in `edb-object-storage` :

```
kubectl get secret -n monitoring  thanos-objstore-secret -o yaml
```

```
 apiVersion: v1
data:
  objstore.yml:␣
↪dHlwZTogUzMKY29uZmlnOgogIGJ1Y2tldDogYXBwbGlhbmNlLXNoYW50ZXN0LWVrcy1jbHVzdGVyLWVkYi1wb3N0Z3JlcwogIHJlZ

~ ⨯  echo␣
↪dHlwZTogUzMKY29uZmlnOgogIGJ1Y2tldDogYXBwbGlhbmNlLXNoYW50ZXN0LWVrcy1jbHVzdGVyLWVkYi1wb3N0Z3JlcwogIHJlZ
↪| 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
```

Re-run 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-
```

(continues on next page)

(continued from previous page)

```
↪appl"
  Warning  Failed     22s                kubelet            Error: AssetPullBackOff
  Normal   Pulling    7s (x2 over 52s)   kubelet            Pulling image "docker.
↪enterprisedb.com/staging_pgai-platform/edbpgai-bootstrap/bootstrap-eks:v1.0.6-appl"
```

Resolution:

1. In your AWS console, edit your related subnet and enable **Auto-assign public IPv4 address**.
2. Delete nodes to let EKS pick up the change:

```
k get node
```

```
NAME                  STATUS   ROLES    AGE   VERSION
i-001f23a316e13e905   Ready    <none>   19h   v1.31.4-eks-0f56d01
i-012fd44b2f575a514   Ready    <none>   20h   v1.31.4-eks-0f56d01
i-076d6a447c6551a21   Ready    <none>   20h   v1.31.4-eks-0f56d01
```

```
k delete node i-001f23a316e13e905 i-012fd44b2f575a514 i-076d6a447c6551a21
```

```
node "i-001f23a316e13e905" deleted
node "i-012fd44b2f575a514" deleted
node "i-076d6a447c6551a21" deleted
```

```
kgp -n edbpgai-bootstrap
```

```
NAME                                              READY   STATUS    RESTARTS   AGE
edbpgai-bootstrap-job-v1.0.6-appl-5ee7l7-xzfqt    1/1     Running   0          118s
```

• • •

Loki pods crash

Detailed Error:

```
k get pods -n logging
```

```
NAME                          READY   STATUS             RESTARTS         AGE
loki-backend-0                1/2     CrashLoopBackOff   16 (2m1s ago)    59m
loki-read-5b8555fb64-9srmq    0/1     CrashLoopBackOff   16 (2m40s ago)   59m
loki-read-5b8555fb64-fw6ls    0/1     CrashLoopBackOff   16 (2m7s ago)    59m
loki-read-5b8555fb64-nw2vq    0/1     CrashLoopBackOff   16 (2m17s ago)   59m
loki-write-0                  0/1     CrashLoopBackOff   16 (2m37s ago)   59m
loki-write-1                  0/1     CrashLoopBackOff   16 (2m5s ago)    59m
loki-write-2                  0/1     CrashLoopBackOff   16 (2m16s ago)   59m
```

```
k logs loki-write-0 -n logging
```

```
level=info ts=2025-02-26T09:32:35.173785968Z caller=main.go:126 msg="Starting Loki"␣
↪version="(version=release-3.1.x-89fe788, branch=release-3.1.x, revision=89fe788d)"
level=info ts=2025-02-26T09:32:35.173821129Z caller=main.go:127 msg="Loading␣
```

(continues on next page)

(continued from previous page)

```
↪configuration file" filename=/etc/loki/config/config.yaml
level=info ts=2025-02-26T09:32:35.174421237Z caller=server.go:352 msg="server listening␣
↪on addresses" http=[::]:3100 grpc=[::]:9095
level=info ts=2025-02-26T09:32:35.175391511Z caller=memberlist_client.go:435 msg="Using␣
↪memberlist cluster label and node name" cluster_label= node=loki-write-0-a0cfbc5d
level=info ts=2025-02-26T09:32:35.17599149Z caller=shipper.go:160 index-store=tsdb-2020-
↪05-15 msg="starting index shipper in WO mode"
level=info ts=2025-02-26T09:32:35.176072171Z caller=table_manager.go:136 index-
↪store=tsdb-2020-05-15 msg="uploading tables"
level=info ts=2025-02-26T09:32:35.176269663Z caller=head_manager.go:308 index-store=tsdb-
↪2020-05-15 component=tsdb-head-manager msg="loaded wals by period" groups=0
level=info ts=2025-02-26T09:32:35.176305024Z caller=manager.go:86 index-store=tsdb-2020-
↪05-15 component=tsdb-manager msg="loaded leftover local indices" err=null␣
↪successful=true buckets=0 indices=0 failures=0
level=info ts=2025-02-26T09:32:35.176325994Z caller=head_manager.go:308 index-store=tsdb-
↪2020-05-15 component=tsdb-head-manager msg="loaded wals by period" groups=1
level=info ts=2025-02-26T09:32:35.180393332Z caller=module_service.go:82 msg=starting␣
↪module=server
level=info ts=2025-02-26T09:32:35.180477643Z caller=module_service.go:82 msg=starting␣
↪module=memberlist-kv
level=error ts=2025-02-26T09:32:35.180526964Z caller=loki.go:524 msg="module failed"␣
↪module=memberlist-kv error="starting module memberlist-kv: invalid service state:␣
↪Failed, expected: Running, failure: service memberlist_kv failed: failed to create␣
↪memberlist: Failed to get final advertise address: no private IP address found, and␣
↪explicit IP not provided"
level=error ts=2025-02-26T09:32:35.180549544Z caller=loki.go:524 msg="module failed"␣
↪module=ring error="failed to start ring, because it depends on module memberlist-kv,␣
↪which has failed: invalid service state: Failed, expected: Running, failure: starting␣
↪module memberlist-kv: invalid service state: Failed, expected: Running, failure:␣
↪service memberlist_kv failed: failed to create memberlist: Failed to get final␣
↪advertise address: no private IP address found, and explicit IP not provided"
level=error ts=2025-02-26T09:32:35.180582375Z caller=loki.go:524 msg="module failed"␣
↪module=store error="failed to start store, because it depends on module memberlist-kv,␣
↪which has failed: invalid service state: Failed, expected: Running, failure: starting␣
↪module memberlist-kv: invalid service state: Failed, expected: Running, failure:␣
↪service memberlist_kv failed: failed to create memberlist: Failed to get final␣
↪advertise address: no private IP address found, and explicit IP not provided"
level=error ts=2025-02-26T09:32:35.180588655Z caller=loki.go:524 msg="module failed"
 module=ingester error="failed to start ingester, because it depends on module analytics,
 which has failed: context canceled"
level=error ts=2025-02-26T09:32:35.180593435Z caller=loki.go:524 msg="module failed"␣
↪module=distributor error="failed to start distributor, because it depends on module␣
↪analytics, which has failed: context canceled"
level=error ts=2025-02-26T09:32:35.180597415Z caller=loki.go:524 msg="module failed"␣
↪module=analytics error="failed to start analytics, because it depends on module␣
↪memberlist-kv, which has failed: invalid service state: Failed, expected: Running,␣
↪failure: starting module memberlist-kv: invalid service state: Failed, expected:␣
↪Running, failure: service memberlist_kv failed: failed to create memberlist: Failed to␣
↪get final advertise address: no private IP address found, and explicit IP not 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
```

(continues on next page)

(continued from previous page)

```
level=info ts=2025-02-26T09:32:35.180708417Z caller=loki.go:508 msg="Loki stopped"␣
↪running_time=85.855531ms
failed services
github.com/grafana/loki/v3/pkg/loki.(*Loki).Run
    /src/loki/pkg/loki/loki.go:566
main.main
    /src/loki/cmd/loki/main.go:129
runtime.main
    /usr/local/go/src/runtime/proc.go:271
runtime.goexit
    /usr/local/go/src/runtime/asm_amd64.s:1695
level=error ts=2025-02-26T09:32:35.180772078Z caller=log.go:216 msg="error running loki"␣
↪err="failed services\ngithub.com/grafana/loki/v3/pkg/loki.(*Loki).Run\n\t/src/loki/pkg/
↪loki/loki.go:566\nmain.main\n\t/src/loki/cmd/loki/main.go:129\nruntime.main\n\t/usr/
↪local/go/src/runtime/proc.go:271\nruntime.goexit\n\t/usr/local/go/src/runtime/asm_
↪amd64.s:1695"
```

Resolution:

There is a known issue from Loki https://github.com/grafana/loki/issues/8634 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, it should be 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` is not configured correctly, so update the correct configuration and re-run 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}" \
```

(continues on next page)

(continued from previous page)

```
    --set system="eks" \
    --set remoteContainerRegistryURL="${REGISTRY_PACKAGE_URL}" \
    --set internalContainerRegistryURL="${REGISTRY_PACKAGE_URL}" \
    --set bootstrapAsset="${REGISTRY_PACKAGE_URL}/edbpgai-bootstrap/bootstrap-eks:$
↪{EDBPGAI_BOOTSTRAP_IMAGE_VERSION}" \
    --set pgai.portal.authentication.staticPasswords[0].email="owner@mycompany.com" \
    --set pgai.portal.authentication.staticPasswords[0].hash=$2y$10
↪$STTzuGJa3PydqHvlzi2z6OgDU1G/JLTqiuGblH.RemIutWxkztN5m \
    --set pgai.portal.authentication.staticPasswords[0].username="owner@mycompany.com" \
    --set pgai.portal.authentication.staticPasswords[0].userID="c5998173-a605-449a-a9a5-
↪4a9c33e26df7" \
    --version "${EDBPGAI_BOOTSTRAP_HELM_CHART_VERSION}" \
    edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap
```

• • •

401 Access Denied

Detailed Error:

Resolution:

Check the `config.yaml` file. The `staticPasswords` section is not set properly at the case. Re-run `helm upgrade` with the proper values.

```
k get secret upm-dex -n upm-dex -o yaml
```

```
apiVersion: v1
data:
  config.yaml:␣
↪aXNzdWVyOiBodHRwczovL3BvcnRhbC5iYXN1cHBvcnQub3JnL2F1dGgKc3RvcmFnZToKICB0eXBlOiBwb3N0Z3JlcwogIGNvbmZpZz
…

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` . Our `ingressgateway` is an `istio-ingressgateway` .

```
k get all -n istio-system
```

```
NAME                                        READY   STATUS    RESTARTS   AGE
pod/istio-ingressgateway-7956f6d57c-jd8jm   1/1     Running   0          14m
pod/istio-ingressgateway-7956f6d57c-jmmv8   1/1     Running   0          14m
pod/istio-ingressgateway-7956f6d57c-r65tk   1/1     Running   0          14m
pod/istiod-7f7bbcdcbb-fzggv                 1/1     Running   0          3h26m
pod/istiod-7f7bbcdcbb-mxjpz                 1/1     Running   0          3h26m

NAME                           TYPE           CLUSTER-IP       EXTERNAL-IP
                                                       PORT(S)
         AGE
service/istio-ingressgateway   LoadBalancer   172.20.92.132    k8s-istiosys-istioing-
↪987da680df-8452f790b4602f2c.elb.us-east-1.amazonaws.com   80:30950/TCP,443:32604/TCP,
↪9443:31744/TCP   3h26m
service/istiod                 ClusterIP      172.20.112.158   <none>
                                                       15010/TCP,15012/TCP,443/TCP,15014/
↪TCP       3h26m

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/istio-ingressgateway    3/3     3            3           3h26m
deployment.apps/istiod                  2/2     2            2           3h26m

NAME                                               DESIRED   CURRENT   READY   AGE
replicaset.apps/istio-ingressgateway-68fcd46fff    0         0         0       84m
replicaset.apps/istio-ingressgateway-7956f6d57c    3         3         3       14m
replicaset.apps/istio-ingressgateway-866d9bd74b    0         0         0       3h26m
replicaset.apps/istiod-7f7bbcdcbb                  2         2         2       3h26m
```

Check the gateway configuration for configuration errors. The gateway rule is already set to listen on 443 ports:

```
k get gateway/upm-portal -n istio-system -oyaml
```

```
apiVersion: networking.istio.io/v1
kind: Gateway
….
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - portal.basupport.org
    port:
      name: http
      number: 80
      protocol: HTTP
    tls:
      httpsRedirect: true
  - hosts:
    - portal.basupport.org
    port:
```

(continues on next page)

(continued from previous page)

```
    name: https-443
    number: 443
    protocol: HTTPS
  ...
```

Then check the deployment logs:

```
2025-02-20T03:42:32.203112Z     warning envoy main external/envoy/source/server/server.
↪cc:835   Usage of the deprecated runtime key overload.global_downstream_max_
↪connections, consider switching to `envoy.resource_monitors.downstream_connections`␣
↪instead.This runtime key will be removed in future.      thread=15
2025-02-20T03:42:32.203389Z     warning envoy main external/envoy/source/server/server.
↪cc:928   There is no configured limit to the number of allowed active downstream␣
↪connections. Configure a limit in `envoy.resource_monitors.downstream_connections`␣
↪resource monitor.   thread=15
2025-02-20T03:42:32.211729Z     info    xdsproxy        connected to delta upstream XDS␣
↪server: istiod.istio-system.svc:15012   id=1
2025-02-20T03:42:32.268369Z     info    ads     ADS: new connection for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system-1
2025-02-20T03:42:32.268437Z     info    cache   returned workload certificate from cache
       ttl=23h59m59.731565472s
2025-02-20T03:42:32.268890Z     info    ads     SDS: PUSH request for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system resources:1 size:4.0kB resource:default
2025-02-20T03:42:32.279500Z     info    ads     ADS: new connection for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system-2
2025-02-20T03:42:32.279654Z     info    cache   returned workload certificate from cache
       ttl=23h59m59.72034871s
2025-02-20T03:42:32.279711Z     info    ads     SDS: PUSH request for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system resources:1 size:4.0kB resource:default
2025-02-20T03:42:32.280287Z     info    ads     ADS: new connection for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system-3
2025-02-20T03:42:32.280372Z     info    cache   returned workload trust anchor from␣
↪cache       ttl=23h59m59.719629366s
2025-02-20T03:42:32.280503Z     info    ads     SDS: PUSH request for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system resources:1 size:1.1kB resource:ROOTCA
2025-02-20T03:42:32.320657Z     info    wasm    fetching image staging_pgai-platform/upm-
↪beaco-filters/upm-oidc from registry docker.enterprisedb.com with tag v1.1.44
2025-02-20T03:42:32.323034Z     info    wasm    fetching image staging_pgai-platform/upm-
↪beaco-filters/upm-error-transformer from registry docker.enterprisedb.com with tag v1.
↪1.44
2025-02-20T03:42:32.323073Z     info    wasm    fetching image staging_pgai-platform/upm-
↪beaco-filters/upm-authz-checker from registry docker.enterprisedb.com with tag v1.1.44
2025-02-20T03:42:35.074684Z     warning envoy wasm external/envoy/source/extensions/
↪common/wasm/context.cc:1198 wasm log: error parsing plugin configuration: Error("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/
```

(continues on next page)

(continued from previous page)

```
↪config_subscription/grpc/delta_subscription_state.cc:269  delta config for type.
↪googleapis.com/envoy.config.core.v3.TypedExtensionConfig rejected: Unable to create␣
↪Wasm HTTP filter istio-system.upm-oidc        thread=15
2025-02-20T03:42:35.076572Z     warning envoy config external/envoy/source/extensions/
↪config_subscription/grpc/grpc_subscription_impl.cc:138    gRPC config for type.
↪googleapis.com/envoy.config.core.v3.TypedExtensionConfig rejected: Unable to create␣
↪Wasm HTTP filter istio-system.upm-oidc thread=15
2025-02-20T03:42:35.076577Z     warning envoy config external/envoy/source/extensions/
↪config_subscription/grpc/grpc_subscription_impl.cc:138    gRPC config for type.
↪googleapis.com/envoy.config.core.v3.TypedExtensionConfig rejected: Unable to create␣
↪Wasm HTTP filter istio-system.upm-oidc thread=15
2025-02-20T03:42:35.076584Z     warning envoy config external/envoy/source/extensions/
↪config_subscription/grpc/grpc_subscription_impl.cc:138    gRPC config for type.
↪googleapis.com/envoy.config.core.v3.TypedExtensionConfig rejected: Unable to create␣
↪Wasm HTTP filter istio-system.upm-oidc thread=15
2025-02-20T03:42:35.189419Z     info    Readiness succeeded in 3.054667763s
2025-02-20T03:42:35.189690Z     info    Envoy proxy is ready
```

The error here was because the `wasm plugin` was not 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 we 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 the `wasm plugin`, the `aes_key` is not 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 at UI 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, then I create the node, it doesn' t work, UI still show the same value after the node created.

Pending: https://enterprisedb.atlassian.net/browse/UPM-45883

- • •

Disk scale up

Detailed Error:

Scale disk on UI, 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",
```

(continues on next page)

(continued from previous page)

```
  "Cluster": {
    "name": "p-2hgee2782y",
    "namespace": "p-2hgee2782y"
  },
  "namespace": "p-2hgee2782y",
  "name": "p-2hgee2782y",
  "reconcileID": "2de2ade7-2bf7-45f4-a2af-3f135a4f7ad9",
  "error": "persistentvolumeclaims \"p-2hgee2782y-1\" is forbidden: only dynamically␣
↪provisioned pvc can be resized and the storageclass that provisions the pvc must␣
↪support resize",
  "stacktrace": "sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller[...
↪]).reconcileHandler\n\tcloud-native-postgres/vendor/sigs.k8s.io/controller-runtime/pkg/
↪internal/controller/controller.go:316\nsigs.k8s.io/controller-runtime/pkg/internal/
↪controller.(*Controller[...]).processNextWorkItem\n\tcloud-native-postgres/vendor/sigs.
↪k8s.io/controller-runtime/pkg/internal/controller/controller.go:263\nsigs.k8s.io/
↪controller-runtime/pkg/internal/controller.(*Controller[...]).Start.func2.2\n\tcloud-
↪native-postgres/vendor/sigs.k8s.io/controller-runtime/pkg/internal/controller/
↪controller.go:224"
}
```

Check and edit `storageclass` to `allowVolumeExpansion:  true`

```
kubecl edit storageclass gp2
```

```
k get storageclass gp2 -o yaml
```

```
allowVolumeExpansion: true
```

Delete the CNP operator pod to apply the new 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
```

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 **beacon 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 overview*

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

For in-depth architecture and core concepts, see the AI Factory Hub .

6.1 Learning Paths

Follow structured learning paths to master AI Factory in Hybrid Manager:

- *AI Factory 101 (Hybrid Manager)* — Introductory concepts and usage
- *AI Factory 201 (Hybrid Manager)* — Building and managing Gen AI and AI Factory workloads
- *AI Factory 301 (Hybrid Manager)* — Advanced integration, scaling, governance, and optimization

6.2 AI Factory in Hybrid Manager Workloads

Hybrid Manager supports a full set of AI Factory capabilities, tightly integrated into its control plane:

6.2.1 Gen AI Workloads

- Gen AI Overview in Hybrid Manager — Gen AI capabilities in Hybrid Manager
- *Agent Studio in Hybrid Manager* — Create and manage assistants, tools, structures, and rulesets
- *Gen AI Builder in Hybrid Manager* — Manage Knowledge Bases and Data Lakes

6.2.2 Model Management and Serving

- *Image and Model Library Explained* — Manage and govern your model assets
- *GPUs in Model Serving* — Deploy and scale model inference services on Kubernetes
- *GPU Resource Management* — Configure and allocate GPU capacity for serving

6.2.3 Vector Engine

- *Vector Engine* — Integrated vector search and similarity capabilities with Postgres

6.3 Get Started

To begin building with AI Factory in Hybrid Manager:

- Follow the *AI Factory Learning Paths (Hybrid Manager)*
- Explore *Typical use cases* and industry solutions
- Read Hybrid Manager-specific *AI Factory How-To Guides (Hybrid Manager)*
- Deploy your first Gen AI assistant or model using *Agent Studio in Hybrid Manager* or *GPUs in Model Serving*

AI Factory in Hybrid Manager gives you a scalable, secure platform to operationalize AI across your hybrid data estate. Start building today.

6.3.1 Enabling AI Factory

Before you can use AI Factory's Gen AI Builder capabilities, you need to enable it with some essential configuration steps:

Generate a Fernet Key (optional)

If you haven't already generated a Fernet key, you will need to do this to enable the cryptographic library that Gen AI Builder uses internally. This key is used to encrypt sensitive data in AI Factory.

If you used any of the EDB Postgres AI Hybrid Manager `-install-secrets.sh` scripts to set up your environment, you may already have a Fernet key generated. If not, you can generate one using the following command in your terminal. Ensure you have `kubectl` configured to point to your EDB Postgres AI Hybrid Manager 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:
```

(continues on next page)

(continued from previous page)

```
    name: fernet-secret
    namespace: upm-griptape
stringData:
    FERNET_KEY: ${FERNET_KEY}
EOF
```

This command generates a random Fernet key and creates a Kubernetes secret in the `upm-griptape` namespace. The key will be used by AI Factory to encrypt sensitive data.

Create an S3 Bucket

GenAI Builder requires an S3 bucket to store its data. This bucket is called the datalake. This datalake hosts GenAI Builder structures, tools, and indexed data. Ideally, ensure that the S3 bucket is created in the same region as your AI Factory deployment. If using AWS, remember that the bucket name must be globally unique. When creating the S3 bucket, you will need to note down the following information: - S3 Bucket: The name of the bucket to use for the Datalake. - S3 Access Key: The access key used to connect to the Datalake. - S3 Secret Access Key: The secret access key used to connect to the Datalake. - S3 URL: The endpoint URL used to connect to the Datalake. - S3 Region: The region where the S3 bucket is located, such as `us-west-2` or `eu-central-1` .

The Access Key and Secret Access Key are used to authenticate AI Factory's access to the S3 bucket. They may come from your AWS account or be specific keys created for the S3 bucket.

CORS settings for S3 Bucket

If you are using a web-based GenAI Builder UI, you may need to configure CORS (Cross-Origin Resource Sharing) settings for the S3 bucket. This allows the GenAI Builder UI to access resources in the S3 bucket from different origins. You can set up CORS rules on AWS, for example, in the S3 bucket settings under **Permissions**. A CORS configuration might look like this:

```
[
{
    "AllowedHeaders": [
        "*"
    ],
    "AllowedMethods": [
        "PUT",
        "POST",
        "DELETE",
        "GET",
        "HEAD"
    ],
    "AllowedOrigins": [
        "https://${PORTAL_DOMAIN_NAME}"
    ],
    "ExposeHeaders": []
}
]
```

Replace `${PORTAL_DOMAIN_NAME}` with the domain name of your GenAI Builder UI. This configuration allows the specified methods from the allowed origin to access the S3 bucket.

```
[
 {
```

(continues on next page)

(continued from previous page)

```
"origin": ["https://<portal url>"],
"method": ["GET", "PUT", "POST", "DELETE", "HEAD"],
"responseHeader": ["*"],
"maxAgeSeconds": 3600
 }
]
```

Then set that using gsutil:

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

This will allow the GenAI Builder UI to access resources in the GCS bucket from different origins, similar to how it would with an S3 bucket.

Connect the bucket to GenAI Builder

Once you have your S3 bucket created and the necessary credentials, you can connect it to GenAI Builder in the EDB Postgres AI Hybrid Manager. This process involves logging into the GenAI UI and configuring the AI Factory settings.

To log in to the GenAI UI, you will need to have the EDB Postgres AI Hybrid Manager set up and running. Once you have that, follow these steps:

1. **Access the EDB Postgres AI Hybrid Manager**: Open your web browser and navigate to the URL of your EDB Postgres AI Hybrid Manager instance.
2. **Navigate to Launchpad**: In the EDB Postgres AI Hybrid Manager, go to the "Launchpad" section.

   width
   70% :alt: Launchpad Menu

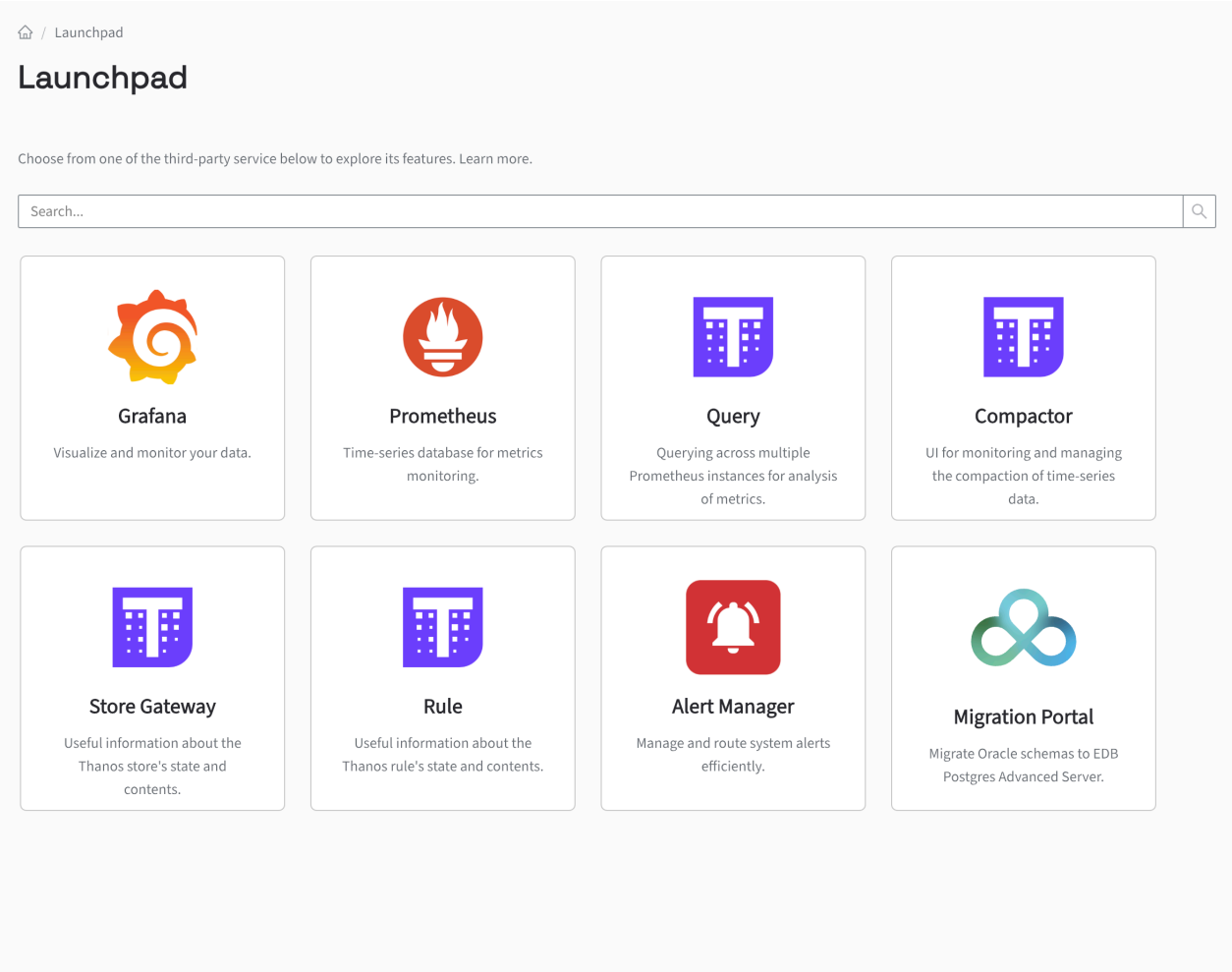
   Launchpad Menu

1. **Select GenAI Builder**: In the Launchpad, find and select "GenAI Builder" to access the GenAI UI.
2. **Log In**: Use your email address to log in to the GenAI UI. If you are using the account your created during the EDB Postgres AI Hybrid Manager setup, `owner@mycompany.com` use that email address.
3. **Enter the Password**: When prompted, enter the password for your account. If you are using the default account created during the setup, the password is `pgai_admin` .
4. **Access the GenAI UI**: After logging in, you will be presented with the GenAI UI, and a dialog will appear prompting you to configure AI Factory.

1. **Select Administrator Console**: Select the "Administrator Console" option to proceed with the configuration of AI Factory. It will display a configuration form for AI Factory.

1. **Fill in the Change Service Configuration Form**: In the configuration form, you will need to provide the following information:

- **S3 URL**: Enter the URL of your S3 bucket. This is typically in the format `https://s3.<region-name>.amazonaws.com/` - with the region where the s3 bucket was created in replacing `<region-name>` .
- **S3 Access Key**: Enter the access key for your S3 bucket; this may be your AWS access key or a specific access key created for the S3 bucket.
- **S3 Secret Access Key**: Enter the secret access key for your S3 bucket; this may be your AWS secret access key or a specific secret access key created for the S3 bucket.
- **S3 Bucket**: Enter the name of your S3 bucket that you created earlier.
- **S3 Region**: Specify the region where your S3 bucket is located, such as `us-west-2` or `eu-central-1` .



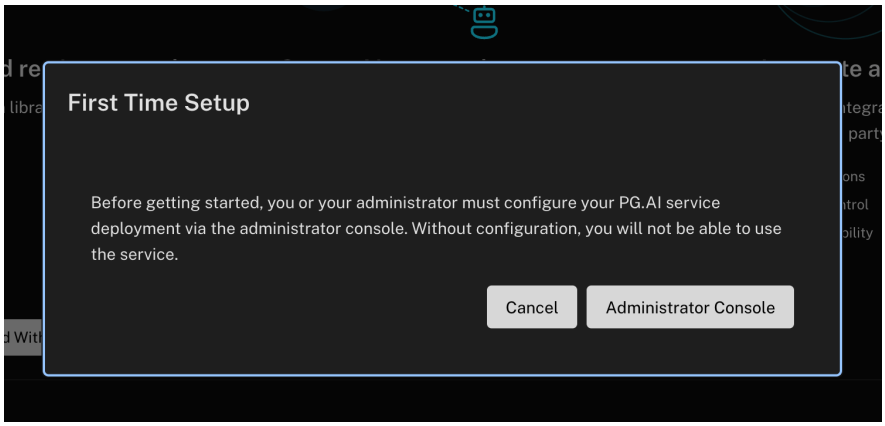


Fig. 1: GenAI UI Configuration

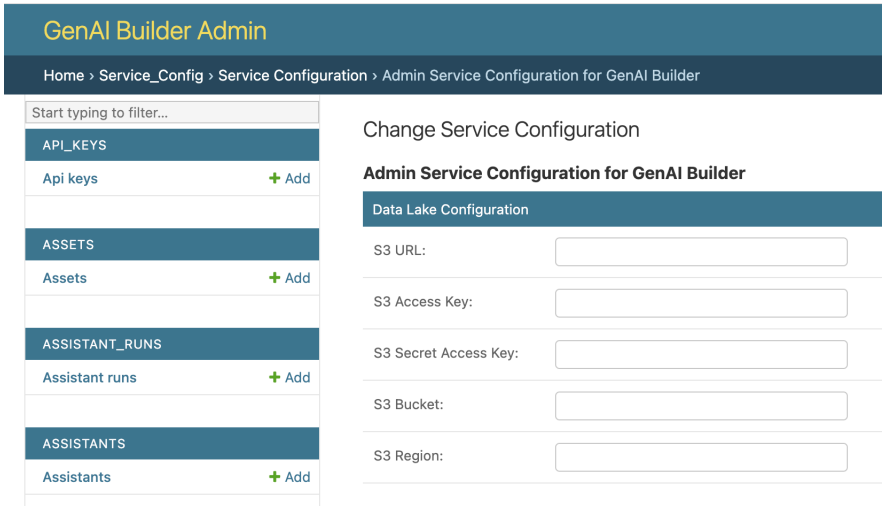


Fig. 2: GenAI Builder first time form

1. **Submit the Configuration**: After filling in the required fields, click the "Submit" button to save your configuration.

This will enable AI Factory and allow you to start using GenAI Builder capabilities.

6.3.2 Gen AI in Hybrid Manager

Hybrid Manager enables production-grade deployment of **Gen AI capabilities** through its AI Factory workload — delivering **Sovereign AI** solutions that integrate deeply with your **Postgres data**, **AI pipelines**, and **enterprise architecture**.

Hybrid Manager's **Kubernetes-native** foundation, combined with **AI Factory**, allows you to build advanced **agentic AI applications** that operate on trusted data, run within governed infrastructure, and connect seamlessly to Postgres.

This page explains how Gen AI workloads operate in Hybrid Manager and how they fit within the broader **EDB Postgres AI (PG AI)** ecosystem.

For an architectural deep dive and full feature reference, start with the *Gen AI Hub* .

Core components

Agent Studio

- A **low-code / no-code interface** for building and managing AI Assistants.
- Enables business teams, content owners, and developers to collaborate.
- Supports configuration of:
- LLM models (served via Model Serving)
- Knowledge Bases
- Rulesets and governance policies
- Tools for system integration
- Memory settings for context persistence

See: *Agent Studio in Hybrid Manager* .

Gen AI Builder

- A **code-driven agent development environment**, ideal for advanced AI developers.
- Supports creation of:
- Complex multi-agent workflows
- Tool-chaining and API orchestration
- Custom agent logic beyond basic chat
- Builders can leverage:
- Model Serving endpoints (KServe InferenceService)
- Knowledge Bases for Retrieval-Augmented Generation (RAG)
- Tools and Structures defined within the project
- Postgres-based data sources via AIDB and Vector Engine

See: *Gen AI Builder in Hybrid Manager* .

Hybrid Manager-specific considerations

- All Gen AI workloads run inside the Hybrid Manager project' s **Kubernetes cluster**.
- Models served via **Model Serving** are secured with project-level controls.
- Knowledge Bases and Pipelines operate within the project' s storage and data governance boundaries.
- Observability for all Gen AI activity is integrated with Hybrid Manager monitoring:
- Inference metrics
- Assistant usage
- Pipeline health
- GPU resource utilization
- Hybrid Manager enables **Sovereign AI**:
- Your data stays in your project.
- Your models run on your infrastructure.
- All AI pipelines and outputs are auditable and observable.

See: *Sovereign AI explained* .

Agent Studio in Hybrid Manager

Agent Studio in **Hybrid Manager** gives you an integrated way to create, manage, and govern **AI Assistants** running entirely within your Hybrid Manager project environment — fully aligned with your **Sovereign AI** architecture.

You can use Agent Studio to build Assistants that:

- Interact with **models deployed through Model Serving**.
- Retrieve content from **Knowledge Bases**.
- Apply **Rulesets** for governance and compliance.
- Execute **Tools** that perform actions or integrate with external systems.
- Maintain **Threads** for stateful, auditable conversations.

For a deeper background, see the *Agent Studio concepts page* in the AI Factory hub.

How Agent Studio works in Hybrid Manager

Agent Studio components run entirely in your **Hybrid Manager project's Kubernetes cluster**, with full observability and governance:

- **Assistants** are defined in Agent Studio and run within your project infrastructure.
- Assistants can call **models deployed via Model Serving** — see *Model Serving in Hybrid Manager* .
- Assistants retrieve content from **Knowledge Bases** built within the same project — see *Knowledge Bases explained* and *AI Accelerator Knowledge Base page* .
- Assistants apply **Rulesets** — defined in your Agent Studio configuration — see *Rulesets explained* .
- Assistants can invoke **Tools** implemented within your project — see *Tools explained* .
- **Threads** are captured automatically to support auditability and review — see *Threads explained* .

All these components operate under your control:

- Data stays in your infrastructure
- Models run on your GPUs, in your Kubernetes cluster
- Knowledge Bases and embeddings stay in your Postgres and object storage
- Assistant behavior is fully observable and governed through Rulesets

This supports **Sovereign AI** principles — see *Sovereign AI explained* .

Typical Agent Studio flow in Hybrid Manager

1. Define a **Knowledge Base** — *How-To: Create a Knowledge Base* .
2. Create and apply **Rulesets** — *How-To: Create a Ruleset* .
3. Define a **Retriever** to control Knowledge Base search —*How-To: Create a Retriever* .
4. Deploy **Tools** for API calls or business logic — *How-To: Create a Tool* .
5. Deploy supporting **models** via Model Serving — *How-To: Deploy a NIM container* .
6. Create your **Assistant** in Agent Studio — *How-To: Create an Assistant* .
7. Test and validate using **Threads** — *How-To: View Threads* .

Key links and learning resources

- *Agent Studio in Hybrid Manager*
- *Create an Assistant — How-To Guide*
- *Assistants explained*
- *Rulesets explained*
- *AI Accelerator Knowledge Bases*
- *Tools explained*
- *Threads explained*
- *Model Serving in Hybrid Manager*

- *Sovereign AI explained*

Next steps: After building your first Assistant, explore advanced topics in the AI Factory *Learning Paths* .

Gen AI Builder in Hybrid Manager

Gen AI Builder in **Hybrid Manager** enables you to build advanced **agentic AI applications** — fully aligned with your **Sovereign AI** architecture — with complete control over data, models, and application behavior.

Builder applications run as containers in your project's Kubernetes cluster and interact with:

- **KServe-powered Model Serving** endpoints
- **Knowledge Bases** built in your Hybrid Manager project
- **Tools** and **Structures** for executing business logic and accessing external systems

  For a deeper background, see the *Gen AI Builder concepts page* in the AI Factory hub and *Gen AI Builder in Hybrid Manager* .

How Gen AI Builder works in Hybrid Manager

- Gen AI Builder applications run in your project's **Kubernetes cluster** — no external compute dependencies required.
- Builders can call **models deployed via Model Serving** — see *Model Serving in Hybrid Manager* .
- Builders retrieve content from **Knowledge Bases** built in your Hybrid Manager project — see *Knowledge Bases explained* .
- Builders invoke **Tools** implemented within your project — see *Tools explained* .
- Builders manage **Threads** to enable stateful, auditable conversations — see *Threads explained* .
- Builders can leverage **Rulesets** to enforce governance and behavioral consistency — see *Rulesets explained* .

Key Hybrid Manager considerations

- Builder applications run on **GPU-enabled infrastructure** when required for high-performance inference — *Update GPU Resources* .
- Builders access **KServe-based Model Serving** endpoints deployed within your Hybrid Manager project — *Deploy AI Models* .
- Builders can access **Knowledge Bases** that are created within the project — see *AI Accelerator Knowledge Base page*
- Builders operate entirely within your Hybrid Manager environment —supporting **Sovereign AI** principles:
- Your data
- Your models
- Your AI pipelines
- Full observability and control

See *Sovereign AI explained* .

Typical Gen AI Builder flow in Hybrid Manager

1. Build or deploy **Tools** — *How-To: Create a Tool* .
2. Create and configure **Knowledge Bases** — *How-To: Create a Knowledge Base* .
3. Deploy supporting **models** via Model Serving — *How-To: Deploy a NIM container* .
4. Configure **Rulesets** as needed — *How-To: Create a Ruleset* .
5. Build **Assistants** and/or **agent workflows** using Gen AI Builder — *Gen AI Builder in Hybrid Manager* .
6. Monitor **Threads** and tune behavior — *How-To: View Threads* .

Learn more

- *AI Factory Concepts*
- *Sovereign AI explained*
- *Gen AI Builder in Hybrid Manager*
- *Create an AI Tool — How-To Guide*
- *Model Serving in Hybrid Manager*
- *AI Accelerator Knowledge Bases*

6.3.3 Learn AI Factory in Hybrid Manager

Learn with AI Factory

This section helps you build foundational knowledge and develop practical skills for using the AI Factory in EDB Postgres AI Platform. Whether you're enabling GPU support, deploying models, or designing AI-powered applications, the resources here are structured to support you at every stage.

The content is organized by intent, using the Diátaxis framework :

AI Factory How-To Guides (Hybrid Manager)

Step-by-step instructions for performing tasks in the AI Factory:

- *Setup GPU resources in Hybrid Manager* — provision GPU resources and prep your cluster.
- *Verify InferenceServices and GPU Usage in Hybrid Manager* — check model status and GPU usage after deployment.

Learning Paths

Follow a curated path through AI Factory topics, tailored by complexity:

- *AI Factory Learning Paths (Hybrid Manager)* — start with enabling the platform and progress to model deployment and advanced usage.

AI Factory in Hybrid Manager

This section explains how key **AI Factory** capabilities work in the context of **Hybrid Manager**.

Hybrid Manager delivers AI Factory as a managed **AI workload** within your Hybrid Manager (HCP) project. This allows you to run **agentic AI applications**, deploy and serve **AI models**, and integrate **vector and knowledge-based search** — all with Kubernetes-native control and security.

Topics

- *Gen AI in Hybrid Manager*
- *Agent Studio in Hybrid Manager*
- *Gen AI Builder in Hybrid Manager*
- *Model Serving in Hybrid Manager*
- *Asset Library in Hybrid Manager*
- *AI Accelerator Pipelines Knowledge Bases*

By combining **AI Factory** with **Hybrid Manager's Kubernetes-native controls and GPU resources**, you can deliver highly integrated and production-ready AI applications.

Container Registry Integration

Container Registry Integration

The Image and Model Library in Hybrid Control Plane (HCP) is powered by container registries. Registries provide the actual image content for:

- **Database images** (PostgreSQL, EPAS, PGE)
- **AI model images** (NVIDIA NIM, future custom models)

Understanding how container registry integration works is key to managing and extending your Image Library.

How does it work?

HCP relies on the **Beacon Agent** to pull image metadata from connected registries.

The Beacon Agent uses **Repository Rules** to know:

- Which registries to connect to
- Which repositories to scan
- Which image tags to import

→ The Image Library UI in HCP is a reflection of what the agent has pulled.

Key Concepts

Concept	Meaning
Registry	A container registry (ex: Azure ACR, AWS ECR, GCP GCR, private registry)
Repository	A collection of images for a given software or model (ex: *postgresql*, *meta/llama-3.3-49b*)
Tag	A version of an image (ex: *17.4-2503310734*, *1.8.5*, *latest*)
Repository Rule	A rule that tells Beacon Agent which repositories/tags to discover
HCP Tags	Organizational tags within HCP → these are separate from registry tags

Default registries

In ephemeral HCP environments:

- Default registry is `upmdev.azurecr.io` (Azure ACR).
- Repositories:
- `postgresql` → PostgreSQL community images.
- `edb-postgres-advanced` → EPAS images.
- `edb-postgres-extended` → PGE images.

In customer environments:

- Customer provides a **private registry**.
- Image Library is configured to pull from that registry.

Repository Rules

Repository Rules drive what Beacon Agent discovers:

- You can add **custom repositories** via Image Library UI → "Manage Repositories."
- You can edit the Beacon Agent config for advanced control.

Example agent config section:

```
dispatcher:
  capabilities:
    provisioning:
      images:
        discovery:
          registry:
            postgres_repositories:
              - name: postgresql
                description: Community PostgreSQL
              - name: edb-postgres-advanced
                description: EPAS
              - name: edb-postgres-extended
                description: PGE
              - name: postgresql-torsten
                description: Custom PG Build with AIDB
```

This tells agent which repositories to pull and display.

→ You can edit this config to add more repositories (advanced).

Private Registry Integration Flow Add repository via Image Library UI → simplest flow for most users.

Agent pulls image list periodically.

Image tags and metadata appear in Image Library.

You can select these images for:

Database provisioning

AI model serving (via Model Library in AI Factory).

Relationship to AI Factory Model Library Model Library in AI Factory is powered by Image Library.

All AI model images flow through this same registry → agent pull → Image Library → available for Model Serving.

This ensures consistent governance of all model images.

→ See: AI Factory Model Serving Explained Summary

- Registries provide the source of truth for images.
- Repository Rules tell HCP which repos/tags to scan.
- Beacon Agent pulls the images and updates Image Library UI.
- Private registries can be fully integrated and governed.
- AI Factory Model Library leverages this same system.

Related How-Tos Integrate Private Registry

Define Repository Rules

Related Explained Pages Image and Model Library Explained

AI Factory Model Serving Explained

Image and Model Library Explained

Image and Model Library Explained

The **Image and Model Library** is a core capability of the Hybrid Control Plane (HCP). It provides a unified, governed repository for managing container images across your entire HCP environment.

This includes:

- **Database images** → for PostgreSQL cluster provisioning.
- **AI model images** → for Model Serving (via AI Factory Model Library).

Why a single library?

Using a unified Image and Model Library ensures:

- **Governance** → all images (DB or AI model) pass through the same approval flow.
- **Consistency** → the same agent-driven sync pulls images from container registries.
- **Security** → private registry integration applies equally to all image types.
- **Flexibility** → both database and AI teams can manage images within a common framework.

What is the difference between Image Library and Model Library?

Capability	Image and Model Library (Hub)	Model Library (AI Factory)
Scope	All container images used in HCP	Subset: AI model images used for Model Serving
Primary Users	DB admins, platform teams, AI teams	AI Factory users, AI engineers
Backed by Container Registry?	Yes	Yes (via Image Library)
Supports Private Registries?	Yes	Yes (via Image Library)
Source of Truth	Container Registry → Image Library	Filtered view of Image Library

continues on next page

Table 2 – continued from previous page

Capability	Image and Model Library (Hub)	Model Library (AI Factory)
Usage	Postgres cluster provisioning, AI model serving	AI model serving only
UI Location	Image Library (HCP Portal)	Model Library (AI Factory UI)

Summary: → The Model Library is powered by Image Library. → All images pass through Image Library first → Model Library presents a user-friendly view of AI model images.

Architecture flow

```
Container Registry → Image and Model Library →
→ Postgres Provisioning / AI Model Serving
→ Model Library UI → Model Serving
```

- Postgres provisioning consumes database images from Image Library.
- AI Factory Model Library consumes model images → curated from Image Library.

Key Concepts Concept Meaning Repository A collection of images (ex: postgresql, meta/llama-3.3-49b) Tag A version of an image (ex: 17.4-2503310734, 1.8.5, latest) HCP Tags Organizational tags applied to image repos in HCP Repository Rules Define which repos/tags to sync from connected registries Private Registry Any customer-provided container registry integrated into HCP Beacon Agent Agent that pulls image metadata into Image Library

Use Cases Database images Provision new Postgres clusters using curated database images.

Support for:

PostgreSQL Community.

EDB Postgres Advanced Server (EPAS).

EDB Postgres Extended Server (PGE).

Manage extensions via OCI annotations.

AI model images Deploy AI models using AI Factory → Model Serving (powered by KServe).

Support for:

NVIDIA NIM model images.

Future: customer-provided model images.

Model Library surfaces a curated view of AI model images from Image Library.

Governance flow Connect container registries → private or public.

Define Repository Rules → which repos/tags to sync.

Beacon Agent pulls images → Image Library populated.

Image Library available to:

Postgres Provisioning flows.

Model Library (for AI model serving).

Model Library filters Image Library → only AI models shown.

Benefits

- Single governance model for all images.
- No bypass of security or compliance policies.

- Private registry integration → works equally for database and AI model images.
- Teams can manage their own repositories but operate within platform-wide visibility.

Related Explained Pages Container Registry Integration Explained

AI Factory Model Serving Explained (Model Library)

Related How-Tos

- Integrate Private Registry
- Define Repository Rules
- Find and Use Images
- Deploy AI Models
- Provision a Postgres Cluster with Custom Image

AI Factory How-To Guides (Hybrid Manager)

Use these **How-To Guides** to deploy and operate **AI Factory** capabilities inside your **Hybrid Manager (HCP)** environment.

Hybrid Manager runs AI Factory as an integrated AI workload inside your HCP Kubernetes project. While many core concepts are shared with the *AI Factory Hub* , there are important **Hybrid Manager-specific patterns and steps** to follow when:

- Setting up Model Serving infrastructure
- Deploying and verifying AI models (InferenceServices)
- Managing GPU resources for model serving
- Using AI models in Gen AI applications and database extensions
- Managing project- and user-scope permissions for AI Factory

Learn more

For conceptual background, see:

- *AI Factory in Hybrid Manager*
- *AI Factory Hub — Full Concepts*

Setup GPU resources in Hybrid Manager

Use this guide to prepare GPU resources in your Hybrid Manager (HCP) Kubernetes cluster to support Model Serving with KServe.

You provision GPU-enabled nodes, configure them for KServe, and store required secrets for deploying NVIDIA NIM models.

Goal

Prepare your HCP cluster to run GPU-based Model Serving workloads using KServe.

Estimated time

20–40 minutes (provisioning might take longer depending on your cloud provider).

What you accomplish

- Provision GPU node groups or node pools in your HCP cluster.
- Label and taint GPU nodes correctly.
- Deploy the NVIDIA device plugin DaemonSet.
- Store your NVIDIA API key as a Kubernetes secret.
- Enable your cluster to run NIM model containers in KServe.

What this unlocks

After you complete this procedure, you can deploy supported GPU-accelerated models through Model Serving:

- For AIDB Knowledge Bases.
- For GenAI Builder assistants.
- For custom model-based applications.

Prerequisites

- Access to an HCP Kubernetes cluster with appropriate permissions.
- Administrative access to provision node groups (AWS EKS / GCP GKE / RHOS).
- NVIDIA API key for accessing NIM models.
- Familiarity with basic `kubectl` usage.

Provision GPU nodes

Provision GPU node groups (EKS) or node pools (GKE / RHOS) in your HCP cluster:

- Use instances with L40S or A100 GPUs (for example, `g6e.12xlarge` on AWS or `a2-highgpu-4g` on GCP).
- Recommended: Provision at least one node with four GPUs to support large models such as Llama 70B.

Label and taint GPU nodes

Apply the following Kubernetes label and taint to GPU nodes.

Label:

```
kubectl label node <gpu-node-name> nvidia.com/gpu=true
```

Taint:

```
kubectl taint nodes <gpu-node-name> nvidia.com/gpu=true:NoSchedule
```

This ensures that KServe model pods are scheduled correctly and that Postgres clusters do not land on GPU nodes.

Deploy the NVIDIA device plugin

Deploy the NVIDIA device plugin DaemonSet.

```
kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.14.1/
↪nvidia-device-plugin.yml
```

Verify that the plugin is running.

```
kubectl get ds -n kube-system nvidia-device-plugin-daemonset
```

The plugin exposes GPU resources to Kubernetes and KServe.

Store NVIDIA API key as Kubernetes secret

Generate an NVIDIA API key from the NGC Catalog portal.

Create a Kubernetes secret.

```
kubectl create secret generic nvidia-nim-secrets --from-literal=NGC_API_KEY=<your_NVIDIA_
↪API_KEY>
```

This secret is required when deploying `ClusterServingRuntime` resources for NIM models.

Related concepts

- *GPUs in Model Serving*
- *KServe in AI Factory concepts*
- *AI Factory Learning Paths (Hybrid Manager)*

Next steps

- *Deploy NIM containers*
- *Verify InferenceServices and GPU Usage in Hybrid Manager*

Verify InferenceServices and GPU Usage in Hybrid Manager

Verify InferenceServices and GPU Usage in Hybrid Manager

Use this guide to confirm the correct deployment and operational status of InferenceServices and GPU resource usage within your Hybrid Manager (HCP) Kubernetes cluster.

Goal

Ensure your deployed InferenceServices are correctly utilizing GPU resources.

Estimated time

15–20 minutes.

What you accomplish

- Verify the status of deployed InferenceServices.
- Confirm GPU resource allocation and utilization.
- Troubleshoot common deployment and GPU-related issues.

Prerequisites

- Completed setup and deployment of GPU resources and NIM containers in Hybrid Manager.
- Access to HCP Kubernetes cluster with appropriate permissions.
- Familiarity with basic `kubectl` usage.

Verify InferenceServices Status

Check the status of your deployed InferenceServices to confirm they are operational.

```
kubectl get inferenceservice -n <namespace>
```

Look for the `READY` status to ensure your service is successfully running.

Confirm GPU Resource Usage

Check GPU resource allocation and usage on nodes.

```
kubectl describe nodes | grep nvidia.com/gpu
```

Review the output to verify GPU availability and allocation.

Use `nvidia-smi` from within your GPU-enabled pods to check real-time GPU utilization.

```
kubectl exec -n <namespace> -it <pod-name> -- nvidia-smi
```

Troubleshoot Common Issues

If the InferenceService is not ready or GPU resources are not properly allocated:

- Confirm the NVIDIA device plugin DaemonSet is running:

```
kubectl get ds -n kube-system nvidia-device-plugin-daemonset
```

- Check for resource constraints or scheduling issues using:

```
kubectl describe pods -n <namespace>
```

Address any errors or issues reported by these commands.

Related concepts

- *Setup GPU resources in Hybrid Manager*
- *Deploy NIM containers*
- *KServe in AI Factory concepts*

Next steps

- Optimize GPU resource utilization and scaling.
- Monitor model performance and health.

AI Factory Learning Paths (Hybrid Manager)

Follow these **Learning Paths** to gain practical skills for running **AI Factory** capabilities inside **Hybrid Manager** projects.

- • •

Getting Started

AI Factory 101 (Hybrid Manager)

- Understand AI Factory architecture in HCP
- Understand how AI Factory capabilities are deployed in Kubernetes
- Explore Model Library and Model Serving in Hybrid Manager
- Explore Gen AI Builder in Hybrid Manager

Start with:

- AI Factory in Hybrid Manager Explained
- *Model Library How-To Guides*
- *Model Serving How-To Guides*

- • •

Intermediate

AI Factory 201 (Hybrid Manager)

- Deploy custom AI models via Model Library
- Manage GPU resources for AI model serving
- Deploy InferenceServices with advanced configuration
- Monitor and troubleshoot model serving
- Connect Gen AI applications to served models

Suggested guides:

- *Integrate Private Container Registry*
- *Deploy NVIDIA NIM Containers*
- *Monitor InferenceServices*

- • •

Advanced

AI Factory 301 (Hybrid Manager)

- Optimize GPU usage across AI workloads
- Implement multi-model serving patterns
- Use model explainability tools (future)
- Integrate model endpoints with Hybrid Manager Gen AI pipelines
- Manage project- and user-scope for AI Factory features

Continue with:

- *Update GPU Resources*
- *Verify InferenceServices and GPU Usage in Hybrid Manager*
- • •

Learn more

For full conceptual background:

- *AI Factory in Hybrid Manager*
- *AI Factory Hub — Concepts*

For **Training** options:

- Instructor-led and self-paced AI Factory courses *(see EDB Training Catalog)*.

6.3.4 Model capabilities in Hybrid Manager

Hybrid Manager provides full support for **Model Serving** and **Model Library**, delivered through the AI Factory workload.

Models are deployed as scalable **Inference Services** using KServe on Hybrid Manager' s Kubernetes infrastructure.

The **Model Library** provides discovery and management of supported models and container images.

Where to learn more

- *Model Serving in Hybrid Manager*
- *Asset Library in Hybrid Manager*
- Full feature details: *Model Serving in Hybrid Manager* , *Asset Library in Hybrid Manager*

GPUs in Model Serving

GPU acceleration is essential for running modern deep learning models in production. Many Large Language Models (LLMs), embedding models, and vision models require GPUs to deliver acceptable inference performance.

Model Serving in AI Factory relies on GPU-enabled Kubernetes nodes for hosting KServe InferenceServices that run these models.

Why GPUs matter for Model Serving

- Many **NVIDIA NIM containers** are designed to run on GPUs, with optimized inference serving.
- Model Serving in AI Factory supports **GPU scheduling and resource control** through Kubernetes.
- GPU nodes enable serving models that would otherwise be too slow or expensive to run on CPUs.
- GPU-based serving supports **AIDB Knowledge Bases** and **GenAI Builder assistants** at scale.

GPU usage in Hybrid Manager

Hybrid Manager (HCP) manages the Kubernetes infrastructure where Model Serving runs. In this context:

- GPU-enabled node groups (AWS EKS) or node pools (GCP GKE, RHOS) must be provisioned.
- These nodes must be labeled and tainted to allow **KServe model pods** to schedule properly.
- The NVIDIA Kubernetes device plugin must be installed to expose GPU resources to Kubernetes.
- Kubernetes secrets must be created to store **NVIDIA API keys** required by NIM models.

Actions you can take

To enable GPU-based Model Serving:

1. Provision GPU node groups in your HCP Kubernetes cluster.
2. Label and taint GPU nodes correctly.
3. Deploy the NVIDIA device plugin DaemonSet.
4. Create a Kubernetes secret with your NVIDIA API key.
5. Deploy ClusterServingRuntime and InferenceService manifests targeting GPU nodes.

Related concepts

- Model Serving overview
- *KServe in AI Factory concepts*
- *AI Factory Learning Paths (Hybrid Manager)*
- *Model Serving How-To Guides*

Next steps

- Follow the *Setup GPU resources in Hybrid Manager* .

Asset Library in Hybrid Manager

The **Asset Library** in Hybrid Manager provides a central location for managing AI-related assets in your project. It replaces the prior **Model Library** and expands capabilities to include:

- AI model images used in Model Serving.
- Future support for additional AI artifacts (retrievers, embedding indexes, datasets).

You use the Asset Library to:

- Discover and manage NVIDIA NIM models.
- Integrate and manage private registries.

- Manage metadata and governance for model images.
- Support project-scoped model deployment and versioning.

How it fits in the Hybrid Manager architecture

- Asset Library overlays the **Image Library** provided by Hybrid Manager.
- All model images in Asset Library are backed by the project's image repository and governance flow.
- Images can be deployed to **KServe-based Model Serving** endpoints within the same project.
- Asset Library is the central governance point for models used by:
- Gen AI Builder Assistants.
- AIDB Knowledge Bases (embedding models).
- Custom AI pipelines and APIs.

Asset Library ensures that all models used in your AI Factory workloads are controlled, auditable, and deployed securely within your project's Kubernetes infrastructure.

Hybrid Manager considerations

- Repository configuration is scoped to each project. You can configure different private registries per project.
- The governance model follows Hybrid Manager controls, including image signing, provenance checks, and metadata tracking.

Typical use cases

- Deploy pre-integrated NVIDIA NIM models (LLMs, embeddings, OCR, rerankers).
- Add and manage organization-specific models from private registries.
- Control which model versions are available for different AI Factory workloads.
- Apply metadata and governance rules to track model usage and compliance.

Learn more

- *How to deploy AI models in Hybrid Manager*
- *Manage repository metadata*
- Full reference on the underlying concepts: *Asset Library in Hybrid Manager*
- *AI Factory Concepts*

Model Serving in Hybrid Manager

Model Serving in Hybrid Manager provides a scalable, Kubernetes-native way to serve AI models as production-grade inference services.

It is implemented using **KServe** and runs on GPU-enabled nodes in your Hybrid Manager project's Kubernetes cluster. Model Serving enables Gen AI applications, Knowledge Bases, and custom pipelines to use high-performance models under your control.

How Model Serving fits in the Hybrid Manager architecture

Model Serving is a core capability of Hybrid Manager's **AI Factory** workload:

- Models are deployed as KServe **InferenceServices** within the project's Kubernetes cluster.
- Model Serving is powered by GPU-enabled infrastructure that you provision and manage.
- Model images come from the **Asset Library** (formerly Model Library), backed by Hybrid Manager's image governance.
- Model endpoints (HTTP/gRPC) are available to:
- Gen AI Builder Assistants.
- AIDB Knowledge Bases.
- External applications and APIs.

Model Serving in Hybrid Manager ensures that all model serving is governed, auditable, and runs securely within your infrastructure —enabling **Sovereign AI** patterns.

How it works in Hybrid Manager

- KServe is installed and managed by Hybrid Manager within your project's Kubernetes cluster.
- You must provision GPU node groups or node pools to support high-performance model serving.
- GPU nodes must be correctly labeled and configured to support KServe workloads.
- Models are deployed from the **Asset Library** via ClusterServingRuntime and InferenceService definitions.
- Your applications and AI Factory workloads can invoke model endpoints via REST or gRPC.

Key Hybrid Manager considerations

- GPU infrastructure is required for most advanced models, such as LLMs, embeddings, and vision models.
- Hybrid Manager enables full observability of model serving, including Prometheus metrics and Kubernetes-native monitoring.
- Model serving endpoints are secured and managed within your Hybrid Manager project scope.
- Governance for model images and deployment comes from Hybrid Manager's integrated **Asset Library** and image controls.

Typical use cases

- Power Gen AI Builder Assistants with LLM or embedding models.
- Enable AIDB Knowledge Bases with GPU-accelerated embedding pipelines.
- Serve image models (OCR, vision) as part of multi-modal retrieval systems.
- Expose enterprise-grade model APIs to downstream applications.

Links to learn more

- *Deploying NVIDIA NIM models in Hybrid Manager*
- *Verify InferenceServices and GPU Usage in Hybrid Manager*
- *Model Serving Concepts Hub*

- *Frequently asked questions*
- *AI Factory Concepts*
- *Setup GPU resources in Hybrid Manager*

CHAPTER 7

Analytics in Hybrid Manager

Welcome to the central resource for understanding and leveraging the analytics capabilities embedded within EDB Hybrid Manager (HM).

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 topic: *Configure an Iceberg REST 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 topic: *Query Delta Lake tables*

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 topic: *Configure BDR AutoPartition with analytics offload*

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 topic: *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 in Hybrid Manager* — Understand key patterns and capabilities.
- *Create a Lakehouse cluster* — Deploy Lakehouse Clusters to enable scalable querying on object storage.
- *Configure an Iceberg REST catalog connection* — Connect HM-managed or external catalogs to manage Iceberg tables.
- *Query existing Apache Iceberg tables* — Query Iceberg tables from your Lakehouse Clusters.
- *Query Delta Lake tables* — Connect Lakehouse Clusters to existing Delta Lake tables.
- *Configure BDR AutoPartition with analytics offload* — Enable PGD AutoPartition and offload cold partitions to Iceberg.
- *Monitor Tiered Tables status and storage savings* — Track the health and performance of your Tiered Table setup.
- *Leveraging Analytics in Hybrid Manager* — Tailored recommendations for DBAs, DevOps/SREs, Data Scientists, and Developers.
- *Solving Business Problems with Analytics in Hybrid Manager* — Common analytics use cases and how Hybrid Manager solves them.
- Full How-To Index — Step-by-step guides for advanced configurations and operations.

7.5 Explore Further

- *Analytics Concepts in Hybrid Manager*
- *Apache Iceberg in Hybrid Manager*
- *Delta Lake in Hybrid Manager*
- *Tiered Tables in Hybrid Manager*
- *EDB Postgres Lakehouse Clusters on Hybrid Manager*
- *Leveraging Analytics in Hybrid Manager*
- *Solving Business Problems with Analytics in Hybrid Manager*
- *AI Factory How-To Guides (Hybrid Manager)*
- Analytics Tutorials

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 Delta Lake in Hybrid Manager

Delta Lake is an open-source table format that brings ACID transactions and reliability to data lakes.

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 *Understanding Delta Lake with EDB Solutions* .

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 core analytics terms used in Hybrid Manager—such as PGFS, PGAA, Lakehouse Cluster, and Analytics Offload—see *Analytics Concepts in Hybrid Manager* .

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).

Query Delta Lake tables

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.

Configure PGFS for Delta Lake

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.

Query Delta Lake tables

Getting started with Delta Lake in Hybrid Manager

To begin using Delta Lake with Hybrid Manager:

1. *Create a Lakehouse cluster* .
2. *Configure PGFS for Delta Lake* 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

- *Query Delta Lake tables*
- *Configure PGFS for Delta Lake*
- *Performance tuning for Delta Lake queries*

Next topic

Tiered Tables in Hybrid Manager

7.5.2 Apache Iceberg in Hybrid Manager

Apache Iceberg is an open table format for large analytical datasets stored in object storage.

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 with EDB Solutions* .

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 core analytics terms used in Hybrid Manager—such as PGFS, PGAA, Lakehouse Cluster, Catalog, AutoPartition, and Analytics Offload—see *Analytics Concepts in Hybrid Manager* .

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.

Query existing Apache Iceberg tables

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 REST 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.

Offload PGD data to Apache Iceberg

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

How-To: Iceberg use cases in Hybrid Manager

Getting started with Iceberg in Hybrid Manager

To begin using Iceberg with Hybrid Manager:

1. *Create a Lakehouse cluster* .
2. *Configure an Iceberg REST catalog connection* (HM-managed or external).
3. If offloading PGD data, *Offload PGD data to Apache Iceberg* .
4. If querying existing Iceberg tables, *Query existing Apache Iceberg tables* .
5. Use standard Postgres clients to run analytical queries.

Related How-Tos

- *Configure an Iceberg REST catalog connection*
- *Query existing Iceberg tables*
- *Offload PGD data to Apache Iceberg*

Next topic

Delta Lake in Hybrid Manager

7.5.3 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.

Hybrid Manager (HM) enables you to provision and manage Lakehouse Clusters through its UI 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: An Overview* .

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 in Hybrid Manager* .

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.

Query existing Apache Iceberg tables

Query Delta Lake tables

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.

Offload PGD data to Apache Iceberg

Query Tiered Tables from PGD and Lakehouse

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.

Configure an Iceberg REST catalog connection

Query existing Apache Iceberg tables

Query Delta Lake tables

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 HM UI or API to create and configure Lakehouse Clusters.

Where: Lakehouse Clusters run in cloud environments managed by HM.

Create a Lakehouse cluster

Getting started with Lakehouse Clusters in Hybrid Manager

To begin using Lakehouse Clusters with Hybrid Manager:

1. *Create a Lakehouse cluster* .
2. *Configure PGFS for Delta Lake* pointing to object storage.
3. (Optional) *Configure an Iceberg REST catalog connection* .
4. (Optional) *Offload PGD data to Apache Iceberg* using Tiered Tables.
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 REST catalog connection*
- *Query existing Apache Iceberg tables*
- *Query Delta Lake tables*
- *Offload PGD data to Apache Iceberg*
- *Query Tiered Tables from PGD and Lakehouse*

Next topic

Apache Iceberg in Hybrid Manager

7.5.4 Learning Analytics in Hybrid Manager

Use this section to learn how to apply EDB Hybrid Manager (HM) analytics features to your real-world workloads.

Hybrid Manager enables you to manage and operate advanced analytics capabilities across your Postgres deployments, including:

- **Lakehouse Clusters** for high-speed analytical queries on object storage
- **Apache Iceberg and Delta Lake** integration for modern data lake architectures
- **Tiered Tables** for cost-effective management of large time-series data
- Unified management of both operational and analytical data through a single control plane

Solving Business Problems

See common analytics use cases and how to solve them using Hybrid Manager capabilities.

- *Solving Analytics Problems in Hybrid Manager*

Next topic: *Persona-Based Guide to Using Analytics in Hybrid Manager*

How-To Guides

Step-by-step instructions for configuring and using Hybrid Manager analytics features.

- *How-To Guides for Analytics in Hybrid Manager*

Next topic: Analytics Tutorials

Next Steps

- Start by reviewing ** *Analytics Concepts in Hybrid Manager* ** to understand the core architecture and patterns.
- Explore ** *Solving Problems* ** to match features to your business needs.
- Use the ** *AI Factory How-To Guides (Hybrid Manager)* ** to configure and operate your system.
- Continue with the ** Analytics Tutorials ** for deeper hands-on learning.

Explore Further

- *Analytics in Hybrid Manager*
- *Working with Apache Iceberg in Hybrid Manager*
- *Working with Delta Lake in Hybrid Manager*
- *Implementing Tiered Tables in Hybrid Manager*
- *Why use Lakehouse Clusters in Hybrid Manager*

Analytics Concepts in Hybrid Manager

Hybrid Manager enables you to build modern data lakehouse and analytics solutions for EDB Postgres.

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 *What is a Postgres Lakehouse* .

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: An Overview

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 with EDB Solutions*

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: *Understanding Delta Lake with EDB Solutions*

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: *Understanding Tiered Tables with EDB Postgres*

PGAA and PGFS in Hybrid Manager

PGAA (Postgres Analytical Appliance)

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 Business Problems with Analytics in Hybrid Manager

How-To Guides for Analytics in Hybrid Manager

This section provides step-by-step guides for configuring and using core analytics features within your Hybrid Manager (HM) environment:

- Deploying and managing **Lakehouse Clusters** for advanced analytical queries
- Configuring and querying **Apache Iceberg** tables
- Configuring and querying **Delta Lake** tables
- Implementing and managing **Tiered Tables** with EDB Postgres Distributed (PGD)

For conceptual background, see *Analytics Concepts in Hybrid Manager* .

Working with Apache Iceberg

- *Configure an Iceberg REST catalog connection*
- *Query existing Apache Iceberg tables*
- *Offload PGD data to Apache Iceberg*

Next: Working with Delta Lake

Implementing Tiered Tables

Configure Tiered Tables

- *Configure PGFS storage for Tiered Tables*
- *Configure PGD node group for analytics offload*
- *Configure BDR AutoPartition with analytics offload*

Monitor and Operate Tiered Tables

- *Monitor Tiered Tables status and storage savings*
- *Query Tiered Tables from PGD and Lakehouse*

Next: *Solving Business Problems with Analytics in Hybrid Manager*

Use these How-To Guides to configure and operate advanced analytics features in Hybrid Manager. To match features to your business needs, see *Solving Business Problems with Analytics in Hybrid Manager* . For role-specific guidance, visit the *Leveraging Analytics in Hybrid Manager* .

Configure PGFS for Delta Lake

This guide explains how to configure a PGFS (EDB Postgres File System) storage location in a Hybrid Manager (HM) Lakehouse cluster, to enable querying Delta Lake tables stored in object storage.

You must configure a PGFS location before creating PGAA external tables that point to Delta Lake tables.

For background on Delta Lake integration in HM, see: *Delta Lake in Hybrid Manager*

Prerequisites

Before configuring PGFS for Delta Lake:

- You have provisioned a Lakehouse cluster in Hybrid Manager. If not yet done, see: *Create a Lakehouse cluster*

- You have existing Delta Lake tables stored in an S3-compatible object storage bucket. The bucket must contain both:
- Parquet data files
- `_delta_log` directory with Delta Lake transaction logs
- You have access credentials for the bucket (if private).
- You have a SQL client connected to your Lakehouse cluster:
- psql
- DBeaver
- pgAdmin
- Any Postgres-compatible client

Steps

Step 1: Create PGFS storage location

Run `pgfs.create_storage_location()` to register the object storage location.

You can create as many PGFS locations as needed, one per bucket or per logical group of Delta tables.

Example: Public bucket

SELECT pgfs.create_storage_location(name => 'my_public_delta_lake_store', url => 's3://my-public-delta-data/', options => '{"aws_skip_signature": "true"}', credentials => '{}');

Example: Private bucket (AWS S3)

SELECT pgfs.create_storage_location(name => 'my_private_delta_lake_assets', url => 's3://my-private-delta-data/', options => '{}', credentials => '{"access_key_id": "YOUR_AWS_ACCESS_KEY_ID", "secret_access_key": "YOUR_AWS_SECRET_ACCESS_KEY"}');

Step 2: Validate PGFS location

Run the following to list all PGFS storage locations:

SELECT * FROM pgfs.list_storage_locations();

Verify that your new location (such as `my_private_delta_lake_assets`) appears in the list.

Notes

- PGFS storage locations can be used by any PGAA external table definition.
- You can define multiple PGFS locations — useful if Delta Lake tables are distributed across buckets.
- Storage locations can point to any S3-compatible object store:
- AWS S3
- Google Cloud Storage
- Azure Storage
- MinIO

- Other compatible systems
- If you are configuring PGFS for Tiered Tables, see: *Configure PGFS storage for Tiered Tables*

Next steps

Now that you have configured a PGFS storage location:

- You can **query existing Delta Lake tables** from your Lakehouse cluster using PGAA reader tables. *Query Delta Lake tables*
- You can **tune query performance** on large Delta Lake tables to maximize vectorized query efficiency. *Performance tuning for Delta Lake queries*
- You can **review Delta Lake concepts and architecture in Hybrid Manager** to understand how this fits in your analytics stack. *Delta Lake in Hybrid Manager*

Configure an Iceberg REST catalog connection

This guide explains how to configure an Iceberg REST catalog connection in Hybrid Manager (HM).

You can connect Lakehouse clusters and PGD nodes to an Iceberg catalog to manage and query Iceberg tables.

Goals

After completing this guide, you will be able to:

- Connect a Lakehouse cluster or PGD node to an Iceberg REST catalog
- Use catalog-managed Iceberg tables for querying or PGD offloading
- Create new Iceberg tables in a catalog
- Refresh catalog metadata into Postgres

Before you start

Review these topics first:

- *Apache Iceberg in Hybrid Manager*
- *EDB Postgres Lakehouse Clusters on Hybrid Manager*
- *Offload PGD data to Apache Iceberg*

Prerequisites

- A Lakehouse cluster or PGD node provisioned in Hybrid Manager
- An Iceberg REST catalog available:
- HM-managed Lakekeeper catalog, or
- External catalog (AWS Glue, Project Nessie, Snowflake Polaris, etc.)
- Credentials for the catalog (REST endpoint URL, token if required, warehouse ID)
- PGAA extension enabled on the target node (Lakehouse cluster or PGD node)

Steps

1. Add catalog connection

Run pgaa.add_catalog() to register the catalog.

Example (general external catalog):

```
SELECT pgaa.add_catalog(
your_catalog_alias,
iceberg-rest,
{
"url": "https://your_catalog_rest_endpoint",
"token": "your_catalog_access_token",
"warehouse": "your_catalog_warehouse_id",
"danger_accept_invalid_certs": "true"
}
);
```

Example (HM-managed Lakekeeper catalog):

```
SELECT pgaa.add_catalog(
hm_lakekeeper_main,
iceberg-rest,
{
"url": "https://hm.example.com/catalog/v1",
"token": "your_hm_api_key",
"warehouse": "lakehouse_warehouse_1"
}
);
```

2. Attach catalog

After adding the catalog, run pgaa.attach_catalog():

```
SELECT pgaa.attach_catalog(your_catalog_alias);
```

This makes catalog tables visible and queryable in Postgres.

3. Import catalog (optional)

To refresh table definitions:

```
SELECT pgaa.import_catalog(your_catalog_alias);
```

Run this after new tables are created externally or if catalog content changes.

4. Creating catalog-managed Iceberg tables

Once a catalog is added and attached, you can create new Iceberg tables managed by the catalog.

Example:

```
CREATE TABLE public.catalog_managed_sensor_data (
    device_id TEXT,
    event_time TIMESTAMP WITH TIME ZONE,
    temperature FLOAT,
    humidity FLOAT
)
USING PGAA
WITH (
    pgaa.format = iceberg,
    pgaa.managed_by = your_catalog_alias,
    pgaa.catalog_namespace = iot_data,
    pgaa.catalog_table = hourly_sensor_readings
);
pgaa.catalog_namespace = iot_data,
pgaa.catalog_table = hourly_sensor_readings
);
```

- pgaa.managed_by: Catalog alias used in pgaa.add_catalog().
- pgaa.catalog_namespace: Target namespace in the catalog.
- pgaa.catalog_table: Target table name in the catalog.

Notes

- You can run SELECT * FROM information_schema.tables to list catalog-managed tables once attached.
- If you are using PGD offloading to Iceberg, the catalog must be attached first — see *Offload PGD data to Apache Iceberg* .
- Catalogs added via pgaa.add_catalog() can be shared between Lakehouse clusters and PGD nodes for consistent metadata access.

What this enables

After configuring an Iceberg catalog connection:

- You can query Iceberg tables managed by the catalog from Lakehouse clusters or PGD nodes
- You can offload PGD data into catalog-managed Iceberg tables
- You can create new catalog-managed Iceberg tables from Postgres
- You can refresh and synchronize catalog metadata into your Postgres environment

Next steps

Now that you have configured an Iceberg catalog connection:

- You can **query existing Iceberg tables** to make external data accessible in Postgres. *Query existing Apache Iceberg tables*
- You can **offload PGD data into this catalog** to enable cost-efficient storage and fast analytics on operational data. *Offload PGD data to Apache Iceberg*
- You can **create new catalog-managed Iceberg tables** to structure analytical data for both Postgres and external engines. See *4. Creating catalog-managed Iceberg tables*

- You can **review core concepts** about how Iceberg and catalogs fit into Hybrid Manager architecture. *Apache Iceberg in Hybrid Manager*

Configure BDR AutoPartition with analytics offload

To implement Tiered Tables, you must configure **BDR AutoPartition** on your PGD table.

This enables:

- Automatic time-based partitioning of your table
- Automatic offloading of old partitions to object storage in Apache Iceberg format

You can control:

- How often new partitions are created
- How old a partition must be to be offloaded (analytics_offload_period)

For background on how Tiered Tables work, see: *Tiered Tables in Hybrid Manager*

Prerequisites

Before configuring AutoPartition:

- You have configured the PGD node group for analytics offload. *Configure PGD node group for analytics offload*
- You have created a PGFS storage location and/or configured an Iceberg catalog. *Configure PGFS storage for Tiered Tables*
- You have created your target table and partitioned it by a time-based column (for example `event_timestamp`).

Steps

1. Create your partitioned table (if not already created).

Example:

```
CREATE TABLE public.application_logs (
log_id BIGSERIAL,
event_timestamp TIMESTAMP WITH TIME ZONE NOT NULL,
application_name TEXT,
log_level VARCHAR(10),
message TEXT,
PRIMARY KEY (log_id, event_timestamp)
) PARTITION BY RANGE (event_timestamp);
```

2. Configure AutoPartition for the table.

Example:

```
SELECT bdr.autopartition (
relation := public.application_logs,
partition_increment := 1 month,
partition_initial_lowerbound := 2023-01-01 00:00:00,
managed_locally := TRUE,
analytics_offload_period := 6 months
);
```

- `partition_increment` — How often to create new partitions.

- `partition_initial_lowerbound` — Start date for partition creation.
- `analytics_offload_period` — Age threshold for offloading partitions (e.g. 6 months old).

3. Verify AutoPartition configuration.

```
SELECT * FROM bdr.partition_config WHERE relid = public.application_logs::regclass;
```

You should see the partitioning policy and the `analytics_offload_period` .

4. Verify partitions are being created.

```
SELECT relname, relkind FROM pg_class WHERE relname LIKE application_logs_part_%;
```

5. Monitor AutoPartition tasks.

```
wqs.workid, wis.ap_wi_started_at, wis.ap_wi_finished_at,
wis.ap_wi_status, "partition", sql
FROM bdr.taskmgr_local_workitem_status wis
RIGHT JOIN bdr.taskmgr_local_work_queue_status wqs
ON wis.ap_wi_workid = wqs.workid
WHERE relname = public.application_logs AND sql NOT LIKE %bdr.autopartition_create_
↪partition%
ORDER BY ap_wi_finished_at DESC;
```

Notes

- AutoPartition will automatically manage new partitions and trigger offload as partitions become eligible.
- You can **adjust the analytics_offload_period** at any time by running bdr.autopartition() again with a new value.
- You can **pause/resume AutoPartition** on a table if needed.

Next steps

Now that you have configured AutoPartition:

- You can **monitor Tiered Table offload and validate storage savings**: *Monitor Tiered Tables status and storage savings*
- You can **query Tiered Tables** — your PGD parent table will transparently include hot (local) and cold (offloaded) partitions: *Query Tiered Tables from PGD and Lakehouse*
- You can **adjust PGD offload policies** if needed: *Configure PGD node group for analytics offload*
- You can **optimize Iceberg storage locations** if needed: *Configure PGFS storage for Tiered Tables*

Related concepts

For architecture overview and concepts behind Tiered Tables, see: *Tiered Tables in Hybrid Manager*

Configure PGD node group for analytics offload

After configuring a PGFS storage location, you must configure your **PGD node group** to offload analytics data for Tiered Tables.

You can choose to:

- **Offload without a catalog** (simpler setup — filesystem-based Iceberg tables), or

- **Offload to an Iceberg catalog** (recommended — enables interoperability with external tools)

Why configure PGD node group for offload?

By configuring this:

- You enable **automatic offloading of cold partitions** created by BDR AutoPartition.
- You can control where your offloaded data is stored — either in raw filesystem paths or as catalog-managed Iceberg tables.
- This step is required to implement full **Tiered Tables** behavior.

For background, see: *Tiered Tables in Hybrid Manager*

Prerequisites

Before configuring PGD node group for offload:

- You have a **PGD cluster** provisioned in Hybrid Manager.
- You have created a PGFS storage location: *Configure PGFS storage for Tiered Tables*
- (Optional) You have configured an Iceberg catalog connection (recommended): *Configure an Iceberg REST catalog connection*

Offloading without a catalog

If you want PGD to offload directly to raw Iceberg filesystem paths:

Step 1: Identify your PGD node group

SELECT node_group_name FROM bdr.node_group;

Example result: `edb_pgdx_node_group_1`

Step 2: Set the node group option `analytics_storage_location`

```
SELECT bdr.alter_node_group_option(
edb_pgdx_node_group_1,
analytics_storage_location,
hm_tiered_analytics_store
);
```

- Replace `'hm_tiered_analytics_store'` with the name of your PGFS storage location.

Result:

- PGD will offload eligible partitions into filesystem-based Iceberg tables at this location.

Offloading with an Iceberg catalog (recommended)

If you want PGD to offload into **Iceberg catalog-managed tables**:

Step 1: Ensure your catalog is configured and attached

```
SELECT pgaa.add_catalog(
hm_main_lakekeeper,
iceberg-rest,
{
"url": "https://hm.example.com/catalog/v1",
"token": "your_hm_api_key",
"warehouse": "lakehouse_warehouse_1"
}
);

SELECT pgaa.attach_catalog(hm_main_lakekeeper);
```

Step 2: Set the node group option `analytics_write_catalog`

```
SELECT bdr.alter_node_group_option(
edb_pgdx_node_group_1,
analytics_write_catalog,
hm_main_lakekeeper
);
```

- Replace `'hm_main_lakekeeper'` with the name of your attached catalog.

Result:

- PGD will offload eligible partitions as **Iceberg tables managed by this catalog**.

Validate the configuration

Run this to verify:

```
SELECT * FROM bdr.node_group_option WHERE node_group_name = edb_pgdx_node_group_1;
```

You should see **either**:

- `analytics_storage_location` set (no catalog), or
- `analytics_write_catalog` set (using a catalog).

Notes

- You must set **either** `analytics_storage_location` **or** `analytics_write_catalog` , not both.
- Using a catalog is strongly recommended if you want offloaded data to be queryable by:
- **Lakehouse clusters**
- **External analytics engines** (Spark, Trino, etc.)
- When using no catalog, offloaded tables will be written into filesystem paths in your PGFS location — suitable for internal use, but harder to interoperate.

Next steps

Now that you have configured PGD node group offload:

- You can **configure AutoPartition to begin partitioning and triggering offload**: *Configure BDR AutoPartition with analytics offload*
- You can **monitor Tiered Table offload and storage savings**: *Monitor Tiered Tables status and storage savings*
- You can **query Tiered Tables from PGD parent table or Lakehouse**: *Query Tiered Tables from PGD and Lakehouse*
- If you need to adjust storage location or catalog later, simply run `bdr.alter_node_group_option()` again.

Related concepts

For an architecture view of how this fits into the Tiered Tables flow, see: *Tiered Tables in Hybrid Manager*

Configure PGFS storage for Tiered Tables

To enable Tiered Tables, you must configure a **PGFS storage location** on your PGD cluster. This defines where PGD will offload "cold" data (as Iceberg tables) to object storage.

You can use an **S3-compatible object store** (AWS S3, Google Cloud Storage, or compatible private storage).

Why configure PGFS for Tiered Tables?

By configuring this:

- You give PGD a destination for **analytics offload of partitions** managed by Tiered Tables.
- You unlock the ability to transparently query cold data using **Lakehouse clusters** or PGD parent table.
- This is the first required step before configuring the PGD node group for offload and enabling AutoPartition.

For background, see: *Tiered Tables in Hybrid Manager*

Goals

After completing this How-To, you will be able to:

- Create a valid PGFS storage location pointing to your object storage bucket.
- Verify that your PGD cluster can use this location for offloading.
- Proceed to configuring PGD node group offload settings.

Prerequisites

Before you begin:

- You have a **PGD cluster** provisioned in Hybrid Manager.
- You have an **object storage bucket** created.
- You have **access credentials** for the bucket (if private).
- You have connected to your PGD node using `psql` or a SQL client.

Steps

Step 1: Connect to your PGD node

Connect using:

- psql
- DBeaver
- pgAdmin
- Any SQL client that supports your PGD node.

Step 2: Run pgfs.create_storage_location()

Public S3 bucket example:

```
SELECT pgfs.create_storage_location(
name => hm_tiered_analytics_store,
url => s3://your-public-bucket/path/to/tiered-data,
options => {"aws_skip_signature": "true"},
credentials => {}
);
```

Private S3 bucket example:

```
SELECT pgfs.create_storage_location(
name => hm_tiered_analytics_store,
url => s3://your-private-bucket/path/to/tiered-data,
options => {"region": "us-west-2"},
credentials => {"access_key_id": "YOUR_AWS_ACCESS_KEY_ID", "secret_access_key": "YOUR_
↪AWS_SECRET_ACCESS_KEY"}
);
```

Guidance:

- Replace `url` with your target bucket path.
- Adjust `options` and `credentials` for your cloud provider as needed.

Step 3: Validate that the PGFS storage location was created

Run:

SELECT * FROM pgfs.storage_location;

You should see your newly created PGFS location listed.

Notes

- You can create **multiple PGFS storage locations** and assign them to different PGD node groups.
- If multiple node groups offload to the same bucket, use **distinct path prefixes** per node group to avoid data overlap.

Next steps

Now that you have configured your PGFS storage location:

- You can configure **PGD node group offload settings** to target this location: *Configure PGD node group for analytics offload*
- You can configure **BDR AutoPartition** to manage partitioning and trigger offload: *Configure BDR AutoPartition with analytics offload*
- Later, you can monitor **offload status and storage savings**: *Monitor Tiered Tables status and storage savings*
- You can query Tiered Table data from PGD or Lakehouse: *Query Tiered Tables from PGD and Lakehouse*

Related concepts

For an architecture view of how PGFS fits into Tiered Tables, see: *Tiered Tables in Hybrid Manager*

Create a Lakehouse cluster

This guide explains how to create a **Lakehouse cluster** in Hybrid Manager (HM). Lakehouse clusters provide **scalable analytical compute** for fast queries on **Apache Iceberg** or **Delta Lake** tables stored in object storage.

For architectural background, see *Why use Lakehouse Clusters in Hybrid Manager* .

Why create a Lakehouse cluster?

By creating a Lakehouse cluster:

- You unlock the ability to **run fast Postgres SQL queries on large datasets** in object storage.
- You can query **Tiered Tables offloaded from PGD** or **external Iceberg/Delta Lake tables**.
- You provide compute for **BI tools and ad-hoc analytics** on your lakehouse data.

This is a foundational step for building your analytics architecture in Hybrid Manager.

Goals

After completing this How-To, you will be able to:

- Deploy a Lakehouse cluster.
- Connect to it using standard Postgres clients.
- Configure storage and catalogs for querying Iceberg and Delta Lake tables.
- Use the cluster to query existing or newly defined lakehouse tables.

Prerequisites

Before you begin:

- You have access to a Hybrid Manager environment with permissions to create Lakehouse clusters.
- You have object storage available (S3, GCS, MinIO, etc.).
- You have Iceberg or Delta Lake tables available — or plan to define them after cluster creation.
- You have an Iceberg catalog prepared if working with Iceberg tables:
- *Configure an Iceberg REST catalog connection*

Steps

Step 1: Initiate cluster creation

- In the HM dashboard, click **Create New**.
- Select **Lakehouse Analytics** or **Analytical Cluster**.
- Confirm that you are creating an Analytics Cluster for querying open lakehouse formats.

Step 2: Choose your path

You will be prompted to choose:

- **Templates** — Use a pre-configured template (if available).
- **Custom build** — Click **Start from Scratch** for full configuration control (recommended for most cases).

Step 3: Configure cluster settings

Cluster Settings

- **Cluster name** — Enter a unique name (e.g. `sales_analytics_lakehouse`).
- **Password** — Enter a strong superuser password.
- **Tags** — Optionally add tags (e.g. `environment:production`, `project:q3-reporting`).
- **Deployment location** — Select cloud provider and region.
- **Database type** — Choose EDB Postgres Extended Server or Advanced Server.
- **Postgres version** — Select supported version with Lakehouse extensions.
- **Instance size** — Select compute size appropriate for your workload.

Additional Settings

- **Networking** — Configure VPC, subnet, security groups, IP whitelisting.
- **Backups** — Configure backup settings if applicable.

Step 4: Review and create

- Review the **Cluster Summary**.
- Confirm all settings.
- Click **Create Cluster**.
- Monitor progress in the HM dashboard.

What you can do next

Now that your Lakehouse cluster is provisioned, you can:

- **Connect to the cluster** to run SQL queries: Use standard Postgres clients (psql, DBeaver, pgAdmin) with HM-provided connection details.
- **Configure storage and catalogs to access data:**
- *Configure PGFS for Delta Lake* — if querying Delta Lake or filesystem-based Iceberg.

- *Configure an Iceberg REST catalog connection* — if querying catalog-managed Iceberg tables or using PGD Tiered Tables.
- **Define and query Lakehouse tables:**
- *Query existing Apache Iceberg tables*
- *Query Delta Lake tables*
- **Query Tiered Tables offloaded from PGD:**
- *Query Tiered Tables from PGD and Lakehouse*

Related topics

- *Why use Lakehouse Clusters in Hybrid Manager*
- *Analytics in Hybrid Manager*

Monitor Tiered Tables status and storage savings

After configuring Tiered Tables, it's important to monitor:

- Which partitions have been offloaded
- Whether offloading is progressing as expected
- Storage savings on PGD vs. object storage
- Offload-related activity on your PGD cluster

You can monitor Tiered Tables from both:

- **PGD cluster** — using SQL observability queries
- **Hybrid Manager** — via monitoring dashboards (if available in your environment)

Why monitor Tiered Tables?

Monitoring Tiered Tables allows you to:

- Validate that offload is working as configured.
- Confirm that hot/cold partitioning is behaving as expected.
- Track storage savings and partition lifecycle.
- Troubleshoot issues with offload or partition management.

Goals

After completing this How-To, you will be able to:

- Identify which tables are configured for offload and their current status.
- Monitor AutoPartition offload activity and job success.
- Measure storage savings on PGD primary storage.
- Use observability queries to monitor partition states.

Prerequisites

Before you begin:

- Tiered Tables must be configured. See:
- *Configure PGFS storage for Tiered Tables*
- *Configure PGD node group for analytics offload*
- *Configure BDR AutoPartition with analytics offload*

Monitor analytics table status

To view which PGD tables are marked for analytics offload:

```
SELECT * FROM bdr.analytics_table;
```

Useful columns:

- `relation` — table name
- `state` — current offload state
- `enabled` — whether analytics offload is enabled
- `disabled` — whether offload is disabled
- `in_progress` — whether offload is currently running

Monitor AutoPartition offload activity

Check the AutoPartition work queue for offload-related tasks:

```
SELECT
wqs.workid, wis.ap_wi_started_at, wis.ap_wi_finished_at,
wis.ap_wi_status, "partition", sql
FROM bdr.taskmgr_local_workitem_status wis
RIGHT JOIN bdr.taskmgr_local_work_queue_status wqs
ON wis.ap_wi_workid = wqs.workid
WHERE sql NOT LIKE %bdr.autopartition_create_partition%
ORDER BY ap_wi_finished_at DESC;
```

This helps you track:

- When offload jobs ran
- Their status (success/failure)
- Which partition was affected

Monitor storage savings on PGD

To observe reduction in transactional storage used by the parent table:

```
SELECT pg_size_pretty(pg_total_relation_size(public.application_logs));
```

As partitions are offloaded and truncated, this size should decrease.

Advanced observability — Heap vs. Offloaded size

To compare local heap vs. offloaded size for partitions: You can run an advanced observability query from EDB's recommended runbooks.

- See the observability query from *EDB PGAA-Tiered-Tables-Observability-Queries* (EDB public runbooks repository).

This allows you to see:

- Per-partition storage split across heap and Iceberg/Delta
- Remaining heap size vs. offloaded size

What you can do next

Now that you can monitor Tiered Tables status and storage savings, you can:

- **Tune your partitioning and offload policies** — adjust `analytics_offload_period` if needed based on observed hot/cold access patterns.
- **Query Tiered Tables for analytics and BI workloads:**
- *Query Tiered Tables from PGD and Lakehouse*
- **Scale Lakehouse clusters appropriately** to handle offloaded data queries.
- **Validate compliance and retention goals** — confirm that old data is offloaded and retained per your policy.

Related How-Tos

- *Configure PGFS storage for Tiered Tables*
- *Configure PGD node group for analytics offload*
- *Configure BDR AutoPartition with analytics offload*
- *Query Tiered Tables from PGD and Lakehouse*

Offload PGD data to Apache Iceberg

This guide explains **how and when to offload PGD data to Apache Iceberg** in Hybrid Manager (HM), and what this unlocks for your analytics architecture.

Why offload PGD data to Iceberg?

- **Cost-efficient tiered storage:** Move older PGD data ("cold data") to object storage to lower primary database storage costs.
- **Faster PGD performance:** Keep PGD operational tables smaller and faster for transactional workloads.
- **Enable Lakehouse analytics:** Query offloaded data in Iceberg format using scalable Lakehouse clusters (vectorized queries).
- **Broader data integration:** Make PGD data available to other tools (Spark, Trino, Flink) via open Iceberg format.

When to use this:

- When implementing **Tiered Tables** (primary use case)
- When building an enterprise **lakehouse architecture** with unified PGD + external analytics
- When archiving historical PGD data with ongoing query access needs

For foundational concepts, see *Tiered Tables in Hybrid Manager* .

How PGD offload works in Hybrid Manager

Offloading PGD data to Iceberg involves 3 layers:

Layer	Action
Storage	Define a PGFS storage location and/or Iceberg catalog (where offloaded data goes).
Node group	Configure your PGD node group to use this storage target.
Table level	Enable analytics replication for each PGD table you want to offload.

Prerequisites

- An HM-managed PGD cluster.
- *Configure PGFS storage for Tiered Tables* defined in PGD (S3-compatible object store).
- (Optional, recommended) *Configure an Iceberg REST catalog connection* configured and attached.
- PGD node group configured for analytics offload:
- *Configure PGD node group for analytics offload*

Steps to offload PGD data

Step 1: Ensure node group is configured for offload (required)

- If using PGFS only: set `analytics_storage_location` .
- If using catalog: set `analytics_write_catalog` .

Reference: *Configure PGD node group for analytics offload*

Step 2: Enable analytics replication on your table

To start offloading a specific PGD table:

```
ALTER TABLE public.operational_logs SET (pgd.replicate_to_analytics = TRUE);
```

Result:

- PGD will automatically replicate this table' s data to Iceberg:
- **Immediately** if full table offload is used.
- **Partitioned** if combined with Tiered Tables and BDR AutoPartition.

Accessing offloaded data

If using Iceberg catalog

- Attach catalog on Lakehouse cluster:

```
SELECT pgaa.attach_catalog(your_catalog_alias);
```

- Query offloaded table directly by catalog name.

If using PGFS only (filesystem offload)

- Define PGAA reader table pointing to Iceberg path in object storage:

```
CREATE TABLE public.operational_logs_offloaded (...) USING PGAA WITH (pgaa.path = s3://
↪your/path, pgaa.format = iceberg);
```

Notes

- This mechanism powers **Tiered Tables**: older partitions offload automatically when using *Configure BDR AutoPartition with analytics offload* .
- You can enable/disable offload per table dynamically.
- Offloaded data remains queryable indefinitely in Iceberg.

What this unlocks

☒ Seamless tiered storage for large PGD tables ☒ Fast lakehouse analytics on historical PGD data ☒ Open format (Iceberg) interoperability ☒ Reduced PGD operational cost and footprint

Next steps

Now that your PGD data is offloading to Iceberg, you can:

- *Monitor Tiered Tables status and storage savings*
- *Query Tiered Tables from PGD and Lakehouse*
- *Query existing Apache Iceberg tables*

For broader architecture view, see *Analytics in Hybrid Manager* .

Performance tuning for Delta Lake queries

This guide provides tips for tuning and optimizing the performance of Delta Lake queries executed from a Hybrid Manager (HM) Lakehouse cluster.

Lakehouse clusters use a vectorized query engine (via PGAA) to efficiently query Delta Lake tables stored in object storage.

For background on Delta Lake integration, see *Delta Lake in Hybrid Manager* .

Goals

After completing this guide, you will:

Check: Understand common tuning patterns for Delta Lake queries from Lakehouse clusters Check: Be able to write queries that take advantage of partition pruning and data skipping Check: Know how to monitor query performance and adjust Lakehouse setup if needed Check: Unlock faster, more cost-efficient analytics over Delta Lake data from Postgres tools

When to use this

You should apply these practices when:

- You are querying existing Delta Lake tables from an HM Lakehouse cluster
- You want to improve query speed and resource efficiency

- You are preparing Lakehouse clusters for BI tool integration (Tableau, Power BI, etc.)
- You are troubleshooting long-running Delta Lake queries

General best practices

Use partitioning effectively

- Delta Lake supports partitioning — design your Delta Lake tables with partition columns that match common query filters.
- Queries that filter on partition columns can prune unnecessary Parquet files for faster scans.

Example:

```
SELECT order_id, sale_amount
FROM public.sales_delta_table
WHERE sale_year = 2024 AND sale_month = 5;
```

Leverage data skipping

- Delta Lake supports data skipping through statistics stored in the Delta transaction log.
- If your Delta Lake table is written by Spark or another tool that collects statistics, PGAA may leverage this for more efficient query planning.

Use LIMIT and targeted projections

- For exploratory queries, always use LIMIT and only SELECT necessary columns to minimize I/O.

Example:

```
SELECT order_id, sale_amount
FROM public.sales_delta_table
LIMIT 100;
```

Filter early and specifically

Push filters into your WHERE clause to enable maximum pruning.

Good:

```
SELECT * FROM public.sales_delta_table
WHERE sale_date >= 2023-01-01 AND product_category = Electronics;
```

Avoid:

```
SELECT * FROM public.sales_delta_table;
```

(full table scan)

Validate PGFS location proximity

- Ensure that your Lakehouse cluster is provisioned in a cloud region close to your object storage to minimize data transfer latency.

Monitor query performance

- Use HM monitoring dashboards to observe query times and resource usage on Lakehouse clusters.
- Identify and tune queries with long runtimes or large I/O.

Known limitations

- PGAA currently supports read-only access to Delta Lake tables.
- Schema evolution is supported — PGAA will read the current schema from the Delta transaction log.
- Complex ACID features (MERGE, DELETE) are not supported for writes via PGAA — use Spark or supported ETL tools for Delta table modifications.

Next steps

Now that you have optimized your Delta Lake queries:

- You can *Query Delta Lake tables* with better performance.
- You can *Configure PGFS for Delta Lake* to ensure fast, reliable storage access.
- You can *Integrate Lakehouse queries with BI tools* to unlock new analytics use cases.

For architecture overview, see *Analytics in Hybrid Manager* .

Query Delta Lake tables

This guide explains how to query existing Delta Lake tables from a Hybrid Manager (HM) Lakehouse cluster.

You can connect Lakehouse clusters to Delta Lake data stored in object storage and run SQL queries using PGAA.

For background on Delta Lake integration, see *Delta Lake in Hybrid Manager* . ## Goals

After completing this guide, you will:

Check: Be able to define PGFS storage locations pointing to Delta Lake data Check: Be able to create PGAA reader tables that expose Delta Lake data to Postgres SQL Check: Unlock SQL-based analytics and BI access to existing Delta Lake tables Check: Enable Postgres tools and users to query Delta data without moving or ETL ## When to use this

You should follow this guide when:

- You already have Delta Lake tables in an object storage location
- You want to query this data using Lakehouse clusters in Hybrid Manager
- You want to connect BI tools or Postgres clients to your Delta Lake data
- You are testing performance or validating Lakehouse integration with Delta ## Prerequisites
- An active Lakehouse cluster provisioned via Hybrid Manager
- Delta Lake tables in an S3-compatible object storage location
- _delta_log directory and Parquet files must be present in the Delta Lake path
- Credentials to access object storage (if needed) ## Steps

1. Define a PGFS storage location

Connect to your Lakehouse cluster and define a PGFS storage location.

Public bucket example:

```
SELECT pgfs.create_storage_location(
name => my_public_delta_lake_store,
url => s3://my-public-delta-data/,
options => {"aws_skip_signature": "true"},
credentials => {}
);
```

Private bucket example:

edb_notranlate_1 ### 2. Enable PGAA extension

edb_notranlate_2 ### 3. Create PGAA external table for Delta Lake

This exposes your Delta Lake table to Postgres SQL:

edb_notranlate_3 ### 4. Query Delta Lake data

Example: get record count.

```
SELECT COUNT(*) FROM public.sales_delta_table;
```

Example: filter and project columns.

`sql SELECT order_id, customer_name, sale_amount FROM public.sales_delta_table WHERE sale_date >= 2023-01-01 AND product_category = Electronics LIMIT 100; ## Notes`

- Delta Lake schema is inferred from the _delta_log
- Lakehouse cluster uses a vectorized query engine for fast queries
- No catalog integration required — Delta Lake is filesystem-based ## Next steps

Now that you have created a Delta Lake external table and run queries:

- You can *Configure PGFS for Delta Lake* to add more locations as needed
- You can *Performance tuning for Delta Lake queries* to optimize large queries
- You can *Integrate Lakehouse queries with BI tools* to enable business intelligence access to Delta data For an architecture overview, see *Analytics in Hybrid Manager* .

Query existing Apache Iceberg tables

This guide explains how to query existing Apache Iceberg tables from a Hybrid Manager (HM) Lakehouse cluster.

You can connect Lakehouse clusters to Iceberg tables stored in object storage and run SQL queries using PGAA.

For background on Iceberg integration, see *Apache Iceberg in Hybrid Manager* .

Goals

After completing this guide, you will:

Check: Be able to define PGFS storage locations pointing to Iceberg data Check: Be able to create PGAA reader tables that expose Iceberg tables to Postgres SQL Check: Unlock SQL-based analytics and BI access to existing Iceberg tables Check: Enable Postgres tools and users to query Iceberg data created by other systems

When to use this

You should follow this guide when:

- You already have Iceberg tables written by Spark, Trino, Flink, PGD offload, or other systems
- You want to query this data using Lakehouse clusters in Hybrid Manager
- You want to connect BI tools or Postgres clients to your Iceberg data
- You are testing performance or validating Lakehouse integration with existing Iceberg datasets

Prerequisites

- An HM-provisioned EDB Postgres Lakehouse cluster
- Existing Apache Iceberg tables in S3-compatible object storage
- Appropriate storage permissions (if required)
- Path to Iceberg table root, containing either version-hint.text or vX.metadata.json

Steps

1. Define a PGFS storage location

Connect to your Lakehouse cluster and define a PGFS storage location.

Example:

```
SELECT pgfs.create_storage_location(
name => my_s3_iceberg_data,
url => s3://your-bucket/path/to/iceberg,
options => {},
credentials => {"access_key_id": "...", "secret_access_key": "..."}
);
```

2. Create PGAA external table for Iceberg

```
CREATE TABLE public.my_sales_iceberg_data (
    sales_id INT,
    sales_date DATE,
    sales_amount NUMERIC
)
USING PGAA
WITH (
    pgaa.storage_location = my_s3_iceberg_data,
    pgaa.path = sales_records/iceberg_table_root,
    pgaa.format = iceberg
);
```

This exposes your Iceberg table to Postgres SQL:

```
CREATE TABLE public.my_sales_iceberg_data ()
USING PGAA
WITH (
pgaa.storage_location = my_s3_iceberg_data,
```

(continues on next page)

(continued from previous page)

```
pgaa.path = sales_records/iceberg_table_root,
pgaa.format = iceberg
);
```

Note: For Iceberg tables not managed by a catalog, pgaa.path typically points to a directory with a version-hint.text file or directly to a vX.metadata.json file.

3. Query the Iceberg table

Run queries using standard SQL.

Example:

```
SELECT * FROM public.my_sales_iceberg_data
WHERE sales_region = North America
LIMIT 100;
```

Notes

- Lakehouse cluster uses a vectorized query engine for fast Iceberg queries
- You do not need a catalog connection to query file-based Iceberg tables
- If your Iceberg tables are catalog-managed, see *Configure an Iceberg REST catalog connection* for an alternative flow

Next steps

Now that you have created an Iceberg external table and run queries:

- You can *Configure an Iceberg REST catalog connection* to query catalog-managed tables
- You can *Offload PGD data to Apache Iceberg* to add more data to your Iceberg lakehouse
- You can *Query Tiered Tables from PGD and Lakehouse* to integrate with operational data

For an architecture overview, see *Analytics in Hybrid Manager* .

Query Tiered Tables from PGD and Lakehouse

Once Tiered Tables are implemented, you can query both **hot data** (local PGD partitions) and **cold data** (offloaded Iceberg partitions) transparently.

This guide explains:

- How to query Tiered Tables from PGD
- How to query Tiered Tables from a Lakehouse cluster
- How the query planner routes queries
- How to query offloaded-only data

For background, see *Tiered Tables in Hybrid Manager* .

Goals

After completing this guide, you will:

- Be able to query Tiered Tables transparently from PGD and Lakehouse
- Understand how the query planner accesses hot and cold data
- Know how to query **offloaded-only** partitions directly
- Know when to use the offloaded view for reports and analysis

When to use this

You should follow this guide when:

- You have implemented Tiered Tables in Hybrid Manager
- You want to validate that hot + cold data is accessible in PGD queries
- You want to query only offloaded data from PGD or Lakehouse
- You want to optimize reports against historical (cold) data

Prerequisites

- BDR AutoPartition with `analytics_offload_period` configured See *Configure BDR AutoPartition with analytics offload*
- PGD node group offload configured See *Configure PGD node group for analytics offload*
- (Optional) Lakehouse cluster connected to your Iceberg catalog See *Query existing Apache Iceberg tables*

Querying the parent table in PGD

The **parent partitioned table** (e.g. `public.application_logs`) is the unified interface for hot + cold data.

Example query:

```
SELECT log_level, COUNT(*)
FROM public.application_logs
WHERE event_timestamp >= 2023-06-01 AND event_timestamp < 2024-06-01
GROUP BY log_level;
```

What happens under the hood:

- Planner accesses **local PGD partitions** for hot data
- Planner accesses **Iceberg offloaded partitions** via PGAA for cold data
- Transparent to your query — no special syntax required

Tip: Run EXPLAIN ANALYZE to observe planner behavior.

Querying offloaded data only (PGD)

PGAA creates an **offloaded view** automatically:

```
SELECT * FROM public.application_logs_offloaded;
```

This view queries **only offloaded partitions**.

Example:

```
SELECT log_level, COUNT(*)
FROM public.application_logs_offloaded;
```

When to use the offloaded view:

- Reports targeting **cold / historical** data only
- Queries where you want to avoid scanning hot partitions
- Testing that offload is working as expected

Querying offloaded data from Lakehouse

If PGD offload uses a catalog (recommended), attach the same catalog in your Lakehouse cluster:

```
SELECT pgaa.add_catalog(...);

SELECT pgaa.attach_catalog(...);
```

Then query Iceberg tables directly:

```
SELECT * FROM iot_data.application_logs_part_2023_jan;
```

Alternatively, you can replicate the `application_logs_offloaded` view logic from PGD into Lakehouse:

- Create a view that selects all catalog-managed offloaded partitions
- Useful for BI tools to match the PGD experience

Notes on planner behavior

- The **parent table** is always the default recommended interface
- Planner automatically optimizes access to PGD vs Iceberg partitions
- Queries with **wide time windows** may benefit from targeting `application_logs_offloaded`
- The offloaded view is only created if AutoPartition was configured with `analytics_offload_period`

Next steps

Now that you have queried Tiered Tables:

- You can *Monitor Tiered Tables status and storage savings* to track partition movement and space savings
- You can *Offload PGD data to Apache Iceberg* for broader historical coverage
- You can *Query existing Apache Iceberg tables* to integrate with external tools and BI platforms

For a broader architecture view, see *Analytics in Hybrid Manager* .

Leveraging Analytics in Hybrid Manager

This page provides role-based guidance on how to leverage analytics features within Hybrid Manager (HM).

Whether you are a Database Administrator, DevOps Engineer, Data Scientist, or Application Developer, Hybrid Manager provides tools to help you manage, analyze, and optimize your EDB Postgres deployments.

For key concepts, see *Analytics Concepts in Hybrid Manager* .

Database Administrator (DBA)

Primary focus: Performance, reliability, and cost-efficiency of HM-managed Postgres environments.

Key objectives and HM features

- Monitor PGD clusters and Lakehouse nodes using HM dashboards.
- Implement Tiered Tables on PGD to offload cold data to object storage.
- Analyze trends and scale Lakehouse clusters as needed.
- Manage backup and recovery for PGD and Lakehouse metadata (including catalogs).
- Control access to Lakehouse clusters through HM and Postgres security features.

Key links

- *Why use Lakehouse Clusters in Hybrid Manager*
- *Implementing Tiered Tables in Hybrid Manager*
- *Working with Apache Iceberg in Hybrid Manager*
- *Leveraging Analytics in Hybrid Manager*

DevOps Engineer / SRE

Primary focus: Automating deployment and observability of analytics infrastructure.

Key objectives and HM features

- Use HM UI, API, or CLI to deploy PGD clusters and Lakehouse Clusters.
- Adjust Lakehouse Cluster size as needed via HM.
- Monitor Lakehouse and PGD performance in HM dashboards.
- Automate provisioning and configuration through HM API/CLI.
- Optimize cost using separate compute/storage architecture.

Key links

- *Why use Lakehouse Clusters in Hybrid Manager*
- *Implementing Tiered Tables in Hybrid Manager*
- *Working with Apache Iceberg in Hybrid Manager*
- *Leveraging Analytics in Hybrid Manager*

Data Scientist / Data Analyst

Primary focus: Efficiently accessing and analyzing data managed by Hybrid Manager.

Key objectives and HM features

- Connect BI tools or analytical tools (Jupyter, R, SQL clients) to Lakehouse nodes.
- Query Iceberg and Delta Lake tables via Lakehouse.
- Use vectorized Lakehouse nodes for fast analytics on large datasets.
- Prepare and clean data using Postgres SQL on Lakehouse or PGD.

Key links

- *Why use Lakehouse Clusters in Hybrid Manager*
- *Working with Apache Iceberg in Hybrid Manager*
- *Working with Delta Lake in Hybrid Manager*
- *Implementing Tiered Tables in Hybrid Manager*

Application Developer

Primary focus: Building applications that interact with data managed by Hybrid Manager.

Key objectives and HM features

- Connect applications to Lakehouse nodes or PGD read replicas for embedded reporting.
- Route complex analytical queries to Lakehouse nodes.
- Manage time-series data across hot (PGD) and cold (Lakehouse) tiers using Tiered Tables.
- (Optional) If using Gen AI Builder with HM, integrate Griptape Structures/Tools with applications.

Key links

- *Why use Lakehouse Clusters in Hybrid Manager*
- *Implementing Tiered Tables in Hybrid Manager*
- *Leveraging Analytics in Hybrid Manager*

For an overview of core analytics concepts, see *Analytics Concepts in Hybrid Manager* .

Solving Business Problems with Analytics in Hybrid Manager

This page explores common analytics use cases and explains how Hybrid Manager (HM) features can help you address them.

By using EDB Postgres Lakehouse Clusters, Tiered Tables with PGD, and integrations with open table formats (Apache Iceberg, Delta Lake), you can build flexible analytics solutions within your Postgres environment.

For key concepts, see *Analytics Concepts in Hybrid Manager* .

For persona-based guidance, see *Persona-Based Guide to Using Analytics in Hybrid Manager* .

Common use cases and HM solutions

Ad-hoc business intelligence and reporting on large datasets

Challenge: Business users and data analysts need to run complex queries on large historical datasets without affecting operational PGD clusters.

HM solution:

- Store large datasets in object storage as Iceberg or Delta Lake tables.
- Provision Lakehouse Clusters for fast, vectorized query execution.
- Connect BI tools (Tableau, Power BI, Looker) to Lakehouse Clusters using standard Postgres connectors.
- Use an Iceberg catalog (HM-managed or external) for table discovery and management.

Learn more: *Why use Lakehouse Clusters in Hybrid Manager*

Working with Apache Iceberg in Hybrid Manager

Cost-effective archival and analysis of time-series data

Challenge: Manage growing time-series data cost-effectively while maintaining access for historical analysis.

HM solution:

- Configure Tiered Tables with PGD AutoPartition and `analytics_offload_period` for automatic offload of cold partitions to Iceberg.
- Query cold data in object storage using Lakehouse Clusters.
- Support transparent access to both hot and cold data through PGD parent tables.

Learn more: *Implementing Tiered Tables in Hybrid Manager*

Integrating operational data with an enterprise data lake

Challenge: Combine data from PGD clusters with other enterprise data sources for unified analytics.

HM solution:

- Offload PGD data to object storage in Iceberg or Delta formats.
- Use a shared Iceberg catalog (HM-managed or external) to support interoperability.
- Query shared Iceberg or Delta tables using Lakehouse Clusters or external engines (Spark, Trino).

Learn more: *Working with Apache Iceberg in Hybrid Manager*

Why use Lakehouse Clusters in Hybrid Manager

Accelerating analytical components of applications

Challenge: Applications require analytical queries that could impact primary PGD cluster performance.

HM solution:

- Route read-heavy analytical queries to dedicated Lakehouse Clusters.
- Precompute aggregates in Lakehouse tables or materialized views for faster application queries.
- Use Tiered Tables to manage time-series data efficiently across hot and cold storage.

Benefits:

- Protect PGD performance.
- Provide scalable compute for application analytics.
- Leverage cost-effective storage.

Industry examples and tutorials

Financial services

- Building a daily risk reporting system using Lakehouse and offloaded PGD data
- Implementing Tiered Tables for long-term trade data archival and analysis

Retail and e-commerce

- Analyzing customer purchase history with Lakehouse and BI tool integration

Telecommunications

- Call Detail Record (CDR) analysis using Tiered Tables and Lakehouse Clusters

Next topic

Leveraging Analytics 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.

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 *Understanding Tiered Tables with EDB Postgres* .

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 core analytics terms used in Hybrid Manager—such as PGFS, PGAA, Analytics Offload, and Lakehouse Cluster—see *Analytics Concepts in Hybrid Manager* .

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 storage for Tiered Tables* .
2. *Configure PGD node group for analytics offload* .
3. *Configure BDR AutoPartition with analytics offload* .
4. (Optional) *Query Tiered Tables from PGD and Lakehouse* .
5. *Monitor Tiered Tables status and storage savings* .

Related How-Tos

- *Configure PGFS storage for Tiered Tables*
- *Configure PGD node group for analytics offload*
- *Configure BDR AutoPartition with analytics offload*
- *Query Tiered Tables from PGD and Lakehouse*
- *Monitor Tiered Tables status and storage savings*

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 AI and Data Factory

Sovereign AI and Data 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 —with no external dependencies.

This engineered system is optimized for sovereignty, security, and scale. It is preassembled, lifecycle-managed, and ready to run advanced Postgres, vector, and AI pipelines the moment it's installed.

8.1 Why choose a hardware-based system?

Cloud-first isn't always an option. Sovereign AI and Data 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 *Solving Business Problems with Analytics in Hybrid Manager* for examples.

8.4 Hardware configurations

EDB offers 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 next

- *Concepts and architecture*
- *Glossary and key terms*
- *What to Expect*
- *Solving Business Problems with Analytics in Hybrid Manager*
- *Frequently asked questions*

To learn more about lifecycle operations, visit *Agent Studio in Hybrid Manager* .

8.5.1 Concepts and architecture

Sovereign AI and Data 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.

Node types and roles

As a pre-requisite, please see configuration details in *System specifications* . 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 AI and Data Factory. You need 1 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 are not 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. Customers do not 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 AI and Data 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 | OK | OK Embeddings | Convert internal data into vector form | NO| OK Retrieval-augmented generation (RAG) | Combine Postgres + LLMs | NO| OK Inference | Serve models locally using GPUs | NO| OK Agentic workflows | Build AI agents that act autonomously | NO| OK

For configuration details, see *System specifications* .

8.5.2 Frequently asked questions

Frequently asked questions

Why use this system instead of a cloud service?

Sovereign AI and Data Factory gives you complete physical and operational control over your infrastructure. It is designed for:

- Environments where data must remain on-premises
- Workloads that require consistent, predictable performance
- Organizations that need AI capabilities but cannot 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—alongside 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 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. This includes:

- Embedding models like `bge-base` , `instructor-xl` , `e5`
- Language models like `LLaMA2` , `Mistral` , `Mixtral`
- Task-specific models (summarization, classification, etc.)

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 AI and Data 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

This glossary defines the key terms and architectural components used throughout the Sovereign AI and Data 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.

Postgres Distributed (PGD)

EDB's multi-node, high-availability Postgres solution. Optional for customers who require globally distributed or synchronous multi-region clusters.

RAG (Retrieval-Augmented Generation)

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—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

System specifications

The Sovereign AI and Data 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-SRN6J-MC	Intel Xeon Gold 4514Y 16-Core @ 2.0GHz	1
Memory	MEM-DR516L-SL01-ER56	16GB DDR5-5600 1RX8 LP ECC RDIMM (Total 128GB)	8

continues on next page

Table 2 – continued from previous page

Category	Product / Model	Description	Qty
Storage	HDS-MUN-MTFDKCC3T2TFS1BC	Micron 7450 MAX 3.2TB NVMe PCIe 4.0 U.3	2
Storage Controller	AOC-VROCSTNMOD	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-MCC	Intel Xeon Platinum 6538N 32-Core @ 2.1GHz	2
Cooling	SNK-P0087V	Passive CPU Heatsink for X13 Platform	2
Memory	MEM-DR532L-SL01-ER56	32GB DDR5-5600 2Rx8 LP ECC RDIMM (Total 1024GB)	32
Storage	HDS-SMN0-MZ1L2960HCJR	Samsung PM9A3 960GB NVMe PCIe 4.0 M.2	2
Storage	HDS-TUN-KCD8XPUG15T3	Kioxia CD8-P 15.3TB NVMe PCIe 5.0 2.5"	4
Storage	HDS-TUN-KCD8XPUG1T92	Kioxia CD8-P 1.92TB NVMe PCIe 5.0 2.5"	2
Storage Controller	AOC-VROCPREMOD	Intel VROC Premium Controller (RAID 0, 1, 5, 10)	1
Networking	AOC-A100G-M2CM-O	AIOM 2-port 100GbE QSFP28, Mellanox CX-6 DX	2
Power	PWS-2K63A-1R	1U 2600W Redundant Power Supply	2
Cooling	FAN-0239L4	80x80x38 mm, 16.8K RPM, Middle Cooling Fan	4
Expansion	RSC-H-68G5 / RSC-H-6G5L	1U LHS Hyper Riser cards (PCIe 5.0 x16/x8, PCIe 5.0 x16)	2
Security	AOM-TPM-9672V-O	TPM 2.0 Module	1

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-MCC	Intel Xeon Platinum 6538N 32-Core @ 2.1GHz	2
Cooling	SNK-P0087V	1U Passive CPU Heatsink for X13 Platform	2

continues on next page

Table 4 – continued from previous page

Category	Product / Model	Description	Qty
Memory	MEM-DR532L-SL01-ER56	32GB DDR5-5600 2Rx8 LP ECC RDIMM (Total 1024GB)	32
GPU	GPU-NVL40S	NVIDIA Ada L40S 48GB GDDR6 PCIe Gen 4	2
Storage	HDS-SMN0-MZ1L2960HCJR	Samsung PM9A3 960GB NVMe PCIe 4.0 M.2	2
Storage	HDS-TUN-KCD8XPUG15T3	Kioxia CD8-P 15.3TB NVMe PCIe 5.0 2.5"	4
Storage	HDS-TUN-KCD8XPUG1T92	Kioxia CD8-P 1.92TB NVMe PCIe 5.0 2.5"	2
Storage Controller	AOC-VROCPREMOD	Intel VROC Premium Controller (RAID 0, 1, 5, 10)	1
Networking	AOC-A100G-M2CM-O	AIOM 2-port 100GbE QSFP28, Mellanox CX-6 DX	2
Power	PWS-2K63A-1R	1U 2600W Redundant Power Supply	2
Cooling	FAN-0239L4	80x80x38 mm, 16.8K RPM, Middle Cooling Fan	4
Cooling	MCP-310-8294...	GPU Plastic Air Shrouds	2
Expansion	RSC-H-68G5 / RSC-H-6G5L	1U LHS Hyper Riser cards (PCIe 5.0 x16/x8, PCIe 5.0 x16)	2
Security	AOM-TPM-9672V-O	TPM 2.0 Module	1

8-GPU B200 10U Server (per server x2)

Category	Product / Model	Description	Qty
System	SYS-A21GE-NBRT-G1	Supermicro 10U B200 Gold Series GPU Server	1
Processor	Intel ® Xeon ® Platinum 8570	56-Core @ 2.10GHz (350W)	1
Memory	DDR5 ECC RDIMM	96GB DDR5 5600MHz ECC RDIMM (Total 3072 GB)	32
GPU	NVIDIA HGX B200	Baseboard with 8x B200 180GB GPUs	1
DPU	NVIDIA BlueField-3	BF3220 DPU 200GbE Dual Port, Crypto Enabled	1
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

Common workloads and use cases

Sovereign AI and Data 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.

This section outlines why customers choose the system over cloud-based platforms, and which use cases it's optimized for.

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 AI and Data Factory exists to meet the needs of:

1. Data sovereignty and compliance

- All data, models, and workloads remain inside your physical infrastructure.
- No external APIs or third-party storage 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.

3. Unified platform for Postgres + AI

- Combines HA Postgres, vector search, and GPU inference in one system, controlled by your organization.
- Simplifies orchestration, monitoring, and support.
- Eliminates fragmented solutions and duplicate infrastructure.

4. Predictable performance and cost

- No egress fees, throttling, or unpredictable GPU queues.
- Optimized I/O and compute across known hardware.
- Total cost of ownership is clear and controlled.

5. Lifecycle-managed infrastructure

- Joint EDB + Supermicro support agreements.
- Hardware and software patches integrated and coordinated.
- No reliance on customer-led infrastructure management or DevOps.

Supported workload patterns

The following workloads are supported end-to-end on the Sovereign AI and Data Factory system:

High-availability Postgres

- Run mission-critical systems of record with automatic failover.
- Observability and lifecycle 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 (e.g., 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

This page outlines the experience of receiving, setting up, and operating your Sovereign AI and Data 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 interface.

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 containers.
- Create RAG-style applications and internal chatbots.

You don't need to access Kubernetes directly—everything is managed through the UI.

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.
