EDB Postgres® AI for CloudNativePG™ Global Cluster

Version 1.2.2

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Built at 2026-10-29T03:38:43

1 EDB Postgres® AI for CloudNativePG™ Global Cluster 3
2 EDB Postgres® AI for CloudNativePG™ Global Cluster release notes 5
2.1 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.2.2 release notes 6
2.2 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.2.1 release notes 7
2.3 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.2.0 release notes 9
2.4 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.1.3 release notes 11
2.5 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.1.2 release notes 13
2.6 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.1.1 release notes 15
2.7 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.1.0 release notes 16
2.8 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.0.1 release notes 18
2.9 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.0.0 release notes 19
3 Before You Start 20
4 Use cases 23
5 Architecture 25
6 Installation and upgrade 30
7 Quick start 39
8 Operator configuration 41
9 Examples of configuration 44
11 Managing EDB Postgres Distributed (PGD) databases 49
12 Image catalog 52
13 Backup on object stores 54
14 Recovery 58
15 Security 64
16 Connectivity 68
17 Certificates 73
18 Client TLS/SSL connections 78
19 Declarative pausing and resuming 79
20 EDB"s private container registry 80
21 Predefined labels 85
22 PGD Node configuration 87
23 Monitoring 90
24 PGDGroup parting 94
25 SQLMutations 96
26 Red Hat OpenShift 98
27 Join method 103
28 Transparent data encryption (TDE) 107
29 LDAP authentication 110
30 Logging 111
31 API Reference 118
32 Supported versions 158
33 Troubleshooting 161

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

1 EDB Postgres® AI for CloudNativePG™ Global Cluster

EDB Postgres® AI for CloudNativePG™ Global Cluster (also CNPG-GC or PGD4K) is an operator designed to manage EDB Postgres Distributed (PGD) workloads on Kubernetes, with traffic routed by PGD Proxy. The v1.x release of PGD4K is designed to manage PGD v5 workloads.

The main custom resource that the operator provides is called `PGDGroup`.

Architectures can also be deployed across different Kubernetes clusters.

Before you start

CNPG-GC provides you with a way to deploy EDB Postgres Distributed in a Kubernetes environment. Therefore, we recommend reading the EDB Postgres Distributed documentation.

To start working with CNPG-GC, read the following in the PGD documentation:

- Terminology
- PGD overview
- Choosing your architecture
- Choosing a Postgres distribution

For advanced usage and maximum customization, it's also important to be familiar with the EDB Postgres for Kubernetes documentation, as described in Architecture.

Supported Kubernetes distributions

CNPG-GC is available for:

- Kubernetes version 1.34 or later through a Helm chart
- Red Hat OpenShift version 4.18 or later only through the Red Hat OpenShift certified operator

Requirements

CNPG-GC requires that the Kubernetes/OpenShift clusters hosting the distributed PGD cluster were prepared by you to cater for:

- The public key infrastructure (PKI) encompassing all the Kubernetes clusters the PGD global group is spread across. mTLS is required to authenticate and authorize all nodes in the mesh topology and guarantee encrypted communication.
- Networking infrastructure across all Kubernetes clusters involved in the PGD global group to ensure that each node can communicate with each other

CNPG-GC also requires Cert Manager 1.10 or later.

> **About connectivity**
>
> See Connectivity for more information.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

API reference

For a list of resources provided by CNPG-GC, see the API reference.

Trademarks

Postgres, PostgreSQL, and the Slonik logo are trademarks or registered trademarks of the PostgreSQL Community Association of Canada, and used with their permission.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

2 EDB Postgres® AI for CloudNativePG™ Global Cluster release notes

The EDB Postgres Distributed for Kubernetes documentation describes the major version of EDB Postgres Distributed for Kubernetes, including minor releases and patches. The release notes provide information on what is new in each release. For new functionality introduced in a minor or patch release, the content also indicates the release that introduced the feature.

Version	Release date
1.2.2	16 Sep 2026
1.2.1	09 Jun 2026
1.2.0	05 Jan 2026
1.1.3	24 Sep 2025
1.1.2	25 Jul 2025
1.1.1	18 Mar 2025
1.1.0	25 Dec 2024
1.0.1	14 Oct 2024
1.0.0	24 Apr 2024

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

2.1 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.2.2 release notes

Released: 16 September 2026

Highlights

- With the fix of CVE-2026-55769 in CloudNativePG `1.28.4` or later, `search_path` is now pinned to `pg_catalog, public, pg_temp` on every operator-issued connection. The PGD operator's mutation reconciler now scopes `search_path` to `public` for the duration of each `PGDGroup` mutation transaction, so unqualified `CREATE` statements in user-authored mutations continue to land in the intended schema instead of failing with a permission error.

Supported versions

- Kubernetes: 1.34, 1.35, 1.36
- PostgreSQL: 17, 16, 15, and 14
- The tested PG4K version is v1.28.4
- Update the default PGD image to PostgreSQL 17.11 and PGD 5.9.5 expanded (image `postgresql-pgd:17.11-pgd595-ubi9`)
- Update the default PGD proxy image to PGD proxy 5.9.5 (image `edb-pgd-proxy:5.9.5-ubi9`)

This release of EDB Postgres® AI for CloudNativePG™ Global Cluster includes the following:

Features

Description
The operator log level can now be set from the operator's ConfigMap or Secret with the `OPERATOR_LOG_LEVEL` key, which overrides the log level configured in the operator deployment. Accepted values are `trace`, `debug`, `info`, `warn`, and `error`.
The physical join job now retries transient failures on its own, with a `BackoffLimit` of `3`, and is reported as `Failed` only once those retries are exhausted. See Physical join for what to do with a job that has failed.
The labels and annotations of a `PGDGroup` are now inherited by the physical join job it creates, so the job is selected by the same label-based tooling — network policies, cost allocation, monitoring — as the rest of the group.

Security Fixes

Description
Bump the go version to v1.27.1.
Bump grpc go module version to v1.83.2.

Bug Fixes

Description
The `manager group` subcommands — barman restore, join, remote join and group bootstrap — now emit logs and honor `--log-level` and `--log-destination`.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

2.2 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.2.1 release notes

Released: 9 June 2026

Highlights

With the fix of CVE-2026-44477 / GHSA-423p-g724-fr39 in PG4K operator `1.28.3` or later, the metrics exporter no longer authenticates as the `postgres` superuser. It now uses a dedicated `cnp_metrics_exporter` role with `pg_monitor` and `bdr_monitor` privileges only, closing a chain that let a low-privilege database user gain PostgreSQL superuser. Custom monitoring queries that read user-owned tables, or use target_databases: '*' against databases where `PUBLIC CONNECT` has been revoked, need explicit `GRANT` statements to `cnp_metrics_exporter`.

Supported versions

- Kubernetes: 1.33, 1.34, and 1.35
- PostgreSQL: 17, 16, 15, and 14
- The tested PG4K version is v1.28.3
- Update the default PGD image to PostgreSQL 17.10 and PGD 5.9.4 (image `postgresql-pgd:17.10-pgd594-ubi9`)
- Update the default PGD proxy image to PGD proxy 5.9.4 (image `edb-pgd-proxy:5.9.4-ubi9`)

This release of EDB Postgres® AI for CloudNativePG™ Global Cluster includes the following:

Features

Description	Addresses
Add parent group uuid in status and label	#2164
Track pgdgroup phases updates in logs	#2172
Support default metrics in ConfigMap and Secret	#2132
Add retry support for restore from volume snapshots	#2198
Disable WAL archive and remove EDB audit log in physical join job	#2245

Security Fixes

Description	Addresses
Updated operator base image to ubi-micro v9.8	#1514
Bump the go version to v1.26.4	

Bug Fixes

Description	Addresses
Fix the issue that physical join can not start if CNP cluster is hibernated and failed to set phase in status	#1906
Cleanup stale recovery settings in postgresql.auto.conf after physical join	#2115
Recognize the connection refused message as PostgreSQL restarting signal	#2182
Enhance the pgdgroupcleanup phase to ensure the `NoExecutorFound` is a final phase	#2177

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Description	Addresses
Propagate PgIdent from PGDGroup to CNP Cluster	#2277

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

2.3 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.2.0 release notes

Released: 5 January 2026

Highlights

- *New PG4K operator LTS Support:* PG4K operator v1.28.0 is now supported

Supported versions

- Kubernetes: 1.34, 1.33, and 1.32
- PostgreSQL: 17, 16, 15, 14

  PostgreSQL 17.7 is the default image
- PGD Extension: v5.9
- PG4K: v1.28.0, v1.25.5

Default images

- PGD: `docker.enterprisedb.com/k8s/postgresql-pgd:17.7-pgd591-ubi9`
- PGD proxy: `docker.enterprisedb.com/k8s/edb-pgd-proxy:5.9.1-ubi9`

This release of EDB Postgres® AI for CloudNativePG™ Global Cluster includes the following:

Features

Description	Addresses
Physical remote join discovery When performing a physical join, discovery can use the group service and will use the DSN from the remote database catalog to perform pg_basebackup and the physical join.	#1659
PGD group cleanup The pgdgroupcleanup reconciler now uses phases to represent progress.	#1617, #1643

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Security Fixes

Description	Addresses
Updated operator base image to ubi-micro v9.7.	#1514

Bug Fixes

Description	Addresses
Prevent operator reconcile when pgd nodes are parted.	#1615
Explicitly set the image name when a pgdgroup uses the default image.	#1592

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

2.4 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.1.3 release notes

Released: 24 September 2025

Highlights

- *New PGD Version Support:* PGD version 5.9 is now supported
- *New PG4K operator Version Support:* PG4K operator v1.26.1 is now supported

Supported versions

- Kubernetes: 1.33, 1.32, 1.31, and 1.30
- Latest PGD with PostgreSQL: 17, 16, 15, 14, and 13

  PostgreSQL 17.6 is the default image

  PostgreSQL 13 support ends on November 12, 2025

- PG4K: v1.25.0, v1.25.1, v1.25.2, v1.26.0,v1.26.1

Default images

- PGD: `docker.enterprisedb.com/k8s_enterprise_pgd/postgresql-pgd:17.6-pgd590-ubi9`
- PGD proxy: `docker.enterprisedb.com/k8s_enterprise_pgd/edb-pgd-proxy:5.9.0-ubi9`

This release of EDB Postgres® AI for CloudNativePG™ Global Cluster includes the following:

Features

Description	Addresses
Barman Cloud Plugin Update the default Barman cloud plugin to v0.6.0. Support customization of the plugin image from the operator configmap	#1068, #1421
PGD Group Configuration Add support to configure instance pod env through pgdgroup YAML	#1509

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Security Fixes

Description	Addresses
Update the operator base image to latest version of ubi-micro v9.6.	#1514

Bug Fixes

Description	Addresses
PGD 5.9 compatibility enable_raft for subgroup could not be enabled with PGD 5.9	#1516
VolumeSnapshot restore restore from VolumeSnapshot sometimes hangs during cleanup metadata phase	#1547
Physical join logging Pipe the physical join (bdr_init_physical command) log with the PostgreSQL log into the physical join job	#1515
TLS and certificates Mount the correct serverTLSSecret in the physical join job, fix physical join job failure when a self-signed certificate is used and only the private key is shared across regions	#1505
Operator configmap name to use pgd-operator-controller-manager-config in all cases	#1493
Show major version upgrade in PGDGroup phase	#1333

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

2.5 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.1.2 release notes

Released: 25 July 2025

Highlights

- *New PGD Version Support:* PGD version 5.8 is now supported
- *New PG4K operator Version Support:* PG4K operator v1.26.0 is now supported

Supported versions

- Kubernetes: 1.33, 1.32, 1.31, and 1.30
- Latest PGD with PostgreSQL: 17, 16, 15, 14, and 13

  PostgreSQL 17.5 is the default image

  PostgreSQL 13 support ends on November 12, 2025
- PG4K: v1.25.0, v1.25.1, v1.25.2, v1.26.0

Default images

- PGD: `docker.enterprisedb.com/k8s/postgresql-pgd:17.5-pgd581-ubi9`
- PGD proxy: `docker.enterprisedb.com/k8s/edb-pgd-proxy:5.8.1-ubi9`

This release of EDB Postgres® AI for CloudNativePG™ Global Cluster includes the following:

Features

Description	Addresses
initDB Options for PGDGroup Supports defining initDB options to pass specific parameters to initdb during PGDGroup setup.	#1302
Major version In-place Upgrade for PGDGroup Enables cross-branch major version upgrades for PGDGroup. Major version In-place Upgrade requires operator v1.1.2, working in conjunction with PG4K v1.26.0, PGD v5.8.1, and above.	
DiscoveryJob Retry Settings Allows configuration of retry, delay, and timeout parameters for the discoveryJob.	#1377
Reconciliation Loop Control Adds an annotation to disable reconciliation for PGDGroup, providing more control over group management.	#1241

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Description	Addresses
Physical Join Enhancement Enables physical joining to nodes with TDE enabled, enhancing security options.	#1258
Barman Cloud Plugin Integration Supports using the barman-cloud plugin for backup, restore, and WAL archiving. The plugin must be installed in the pg4k operator namespace as a prerequisite.	#1068
Flexible PGDGroup Creation Allows PGDGroup creation with only a single data node.	#1200
Enhanced Operator Image Build Improves the build process with baking. Now, images are signed with cosign, and OCI attestations—including the Software Bill of Materials (SBOM) and provenance data—are generated.	#1115

Bug Fixes

Description	Addresses
Inherited Metadata for Scheduled Backups Ensures that scheduled backups inherit metadata from their associated PGDGroup.	#1346
LoadBalancer Class Rendering Correctly renders LoadBalancerClass for LoadBalancer services.	#1290
Cleanup of Completed Jobs Automatically cleans up jobs once the PGDGroup is healthy.	#1368
Scheduling Backups Post-Health Check Delays scheduled backups until the PGDGroup is fully healthy.	#1276
Raft Status Check Checks the global Raft status instead of subgroup-specific status before performing a physical join.	#1274
Backup Node Selection Filters out replica clusters when assigning backup nodes, avoiding misassignments.	#1263
First Recoverability Points Ensures that firstRecoverabilityPoints reflect values from backups taken by any method.	#1250

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

2.6 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.1.1 release notes

Released: 18 March 2025

Highlights

- *New PGD Version Support:* PGD version 5.7 is now supported
- *New PG4K operator Version Support:* PG4K operator v1.25.1 is now supported

Supported versions

- Kubernetes: 1.32, 1.31, 1.30, and 1.29
- PostgreSQL: 17, 16, 15, 14, and 13
- PG4K: v1.22.8, v1.22.8, v1.25.0, v1.25.1

  Default PGD image: postgresql-pgd 17.4-5.7.0-1

  Default PGD proxy image: edb-pgd-proxy 5.7.0-1

This release of EDB Postgres® AI for CloudNativePG™ Global Cluster includes the following:

Features

Description	Addresses
Support PGD node physical join to local group	#745
Support PGD node physical join to remote group	#805
ImageCatalog support for PGD operators	#1026
Improved detection to decide whether `writeLead` transfer is needed before patching the `writelead` node	#1040

Bug Fixes

Description	Addresses
Ensure 'bdr' can be added only once into PG additional libraries	#1085
Share configuration file for pgd-proxy and pgd-cli in proxy pod	#1084

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

2.7 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.1.0 release notes

Released: 25 December 2024

Highlights

- *Support for New PG4K Release:* Starting with version 1.1.0, the PGD4K operator now supports the new PG4K Long-Term Support (LTS) release, version 1.25.
- *New PGD Version Support:* PGD version 5.6.1 is now supported.

Supported versions

- Kubernetes: 1.32, 1.31, 1.30, and 1.29
- PostgreSQL: 17, 16, 15, 14, and 13
- PG4K: v1.22.8, v1.25.0

  Default PGD image: postgresql-pgd 17.2-5.6.1

  Default PGD proxy image: edb-pgd-proxy 5.6.0

  Officially dropped support for PostgreSQL 12

  PostgreSQL 13 support ends on November 12, 2025

This release of EDB Postgres® AI for CloudNativePG™ Global Cluster includes the following:

Features

Description	Addresses
New Operator Architecture Support Added support for linux/arm64 and linux/ppc64le architectures.	
Configuration Support for cnpg-i Plugin Configuration settings for the cnpg-i plugin have been implemented	#929
Avoid unnecessary failure over Avoid unnecessary write lead transfer when `reconcilePodSpec` is used in PG4K cluster	#885
Scale Down Write Leader Last The write leader will be the last PGD node to scale down	#920

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Description	Addresses
Enhanced Restore Process The restore process has been improved by splitting it into components, and using separate jobs for different types of restore and bootstrap operations	#817, #852
Webhook Validation enhancement validity of `restore.serverNames` will be checked via the webhook	#932
Include clusterStatus as part of the PGDGroup status for information on the phase of the managed clusters.	#875

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

2.8 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.0.1 release notes

Released: 14 October 2024

This release of EDB Postgres® AI for CloudNativePG™ Global Cluster includes the following:

Features

Description	Addresses
Operator base image upgraded to UBI9.	#787
Support LDAP authentication configuration in PGDGroup.	#793
Support tablespace configuration in PGDGroup.	#801
Support projectedVolumeTemplate configuration in PGDGroup.	#791
Support read node routing on pgd with creation of a proxy service.	#686
Support backup and restore PGDGroup using VolumeSnapshot.	#549
Support multiple schedulers in PGDGroup for different backup methods.	#731
Support new mutation type which is only executed against write lead node.	
Avoid unnecessary write leader transfer when PGDGroup configuration is changed.	#772
Handle nodeSelector and toleration changes for paused PGDGroup.	#581
Reuse nodeSelector and tolerations in discovery job.	#774
Support application database configuration to be applied before join.	#838
Support topologySpreadConstraints configuration.	#839
Kubernetes v1.28 - v1.31	
Openshift v4.12 - v4.14	
EDB Postgres Distributed v5.5.1	
PG4K v1.22.2 - v1.22.6	

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

2.9 EDB Postgres® AI for CloudNativePG™ Global Cluster 1.0.0 release notes

Released: 24 April 2024

This is the first major stable release of CNPG-GC, a Kubernetes operator to deploy and manage EDB Postgres Distributed clusters.

The operator implements the `PGDGroup` custom resource in the API group `pgd.k8s.enterprisedb.io`. This resource can be used to create and manage EDB Postgres Distributed clusters inside Kubernetes.

The CNPG-GC operator leverages EDB Postgres for Kubernetes (PG4K) and inherits many of that project's capabilities. CNPG-GC version 1.0 is based, specifically, on release 1.22 of PG4K. Please refer to the PG4K release notes for more details.

This release of EDB Postgres® AI for CloudNativePG™ Global Cluster includes the following:

Features

Description
Deployment of EDB Postgres Distributed clusters with versions 5 and later
Additional self-healing capability on top of that of Postgres Distributed, such as recovery and restart of failed PGD nodes
Definition of the services to connect applications to the write leader of each PGD group
Implementation of Raft subgroups
TLS connections and client certificate authentication
Continuous backup to an S3 compatible object store
Pause and resume a PGD cluster, saving computational resources by temporarily removing database pods while keeping the database PVCs.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

3 Before You Start

Before you get started, it's essential that you become familiar with some terminology that's specific to Kubernetes and PGD.

Kubernetes terminology

- **Node**

  A *node* is a worker machine in Kubernetes, either virtual or physical, where all services necessary to run pods are managed by the control plane nodes.

- **Pod**

  A *pod* is the smallest computing unit that can be deployed in a Kubernetes cluster and is composed of one or more containers that share network and storage.

- **Service**

  A *service* is an abstraction that exposes as a network service an application that runs on a group of pods and standardizes important features, such as service discovery across applications, load balancing, and failover.

- **Secret**

  A *secret* is an object that's designed to store small amounts of sensitive data, such as passwords, access keys, or tokens, for use within pods.

- **Storage class**

  A *storage class* allows an administrator to define the classes of storage in a cluster, including provisioner (such as AWS EBS), reclaim policies, mount options, volume expansion, and so on.

- **Persistent volume**

  A *persistent volume* (PV) is a resource in a Kubernetes cluster that represents storage that was either manually provisioned by an administrator or dynamically provisioned by a *storage class* controller. A PV is associated with a pod using a *persistent volume claim*, and its lifecycle is independent of any pod that uses it. Normally, a PV is a network volume, especially in the public cloud. A *local persistent volume* (LPV) is a persistent volume that exists only on the particular node where the pod that uses it is running.

- **Persistent volume claim**

  A *persistent volume claim* (PVC) represents a request for storage, which might include size, access mode, or a particular storage class. Similar to how a pod consumes node resources, a PVC consumes the resources of a PV.

- **Namespace**

  A *namespace* is a logical and isolated subset of a Kubernetes cluster and can be seen as a *virtual cluster* within the wider physical cluster. Namespaces allow administrators to create separated environments based on projects, departments, teams, and so on.

- **RBAC**

  Role-based access control (RBAC), also known as *role-based security*, is a method used in computer systems security to restrict access to the network and resources of a system to authorized users only. Kubernetes has a native API to control roles at the namespace and cluster level and associate them with specific resources and individuals.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- CRD

  A *custom resource definition* (CRD) is an extension of the Kubernetes API and allows developers to create new data types and objects, *called custom resources*.

- Operator

  An *operator* is a Kubernetes software extension that automates those steps that are normally performed by a human operator when managing one or more applications or given services. An operator assists Kubernetes in making sure that the resource's defined state always matches the observed one.

- `kubectl`

  `kubectl` is the command-line tool used to manage a Kubernetes cluster.

EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) requires a Kubernetes version supported by the community. See Supported releases for details.

PGD terminology

For more information, see Terminology in the PGD documentation.

- Data node

  A PGD database instance.

- Failover

  The automated process that recognizes a failure in a highly available database cluster and takes action to connect the application to another active database.

- Switchover

  A planned change in connection between the application and the active database node in a cluster, typically done for maintenance.

- Write leader

  In always-on architectures, a node is selected as the correct connection endpoint for applications. This node is called the write leader. The write leader is selected by consensus of a quorum of data nodes.

Cloud terminology

- Region

  A *region* in the cloud is an isolated and independent geographic area organized in *availability zones*. Zones within a region have very little round-trip network latency.

- Zone

  An *availability zone* in the cloud (also known as a *zone*) is an area in a region where resources can be deployed. Usually, an availability zone corresponds to a data center or an isolated building of the same data center.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

What to do next

Now that you are familiar with the terminology, you can test CNPG-GC on your laptop using a local cluster before deploying the operator in your selected cloud environment.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

4 Use cases

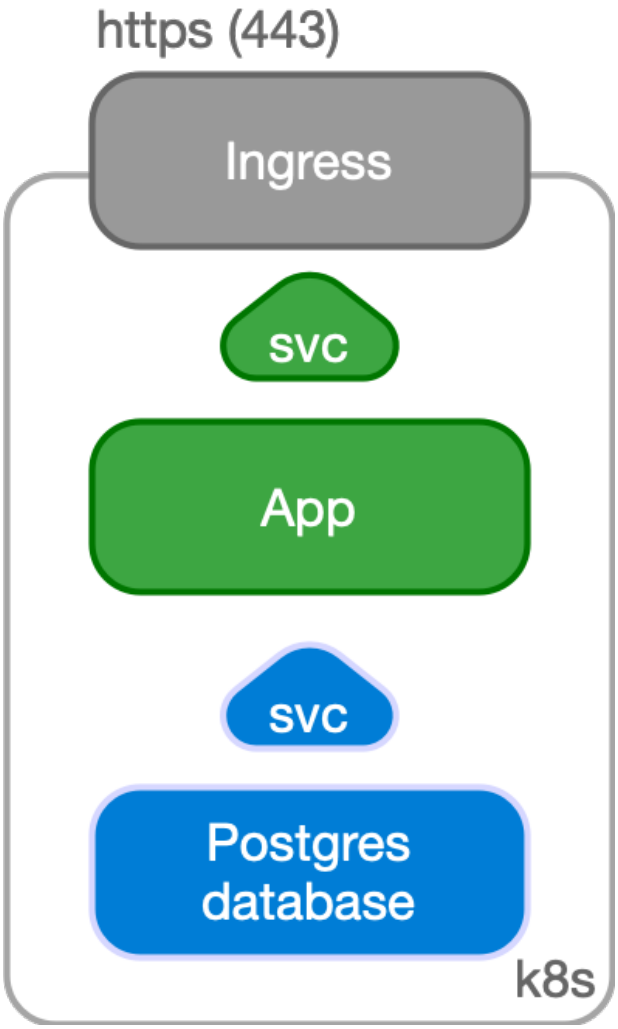
EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) was designed to work with applications that reside in the same Kubernetes cluster for a full cloud native experience.

However, it might happen that, while the database can be hosted inside a Kubernetes cluster, applications can't be containerized at the same time and need to run in a traditional environment such as a VM.

The following is a summary of the basic considerations. See the EDB Postgres for Kubernetes documentation for more detail.

Case 1: Applications inside Kubernetes

In a typical situation, the application and the database run in the same namespace inside a Kubernetes cluster.



The application, normally stateless, is managed as a standard deployment, with multiple replicas spread over different Kubernetes nodes and internally exposed through a ClusterIP service.

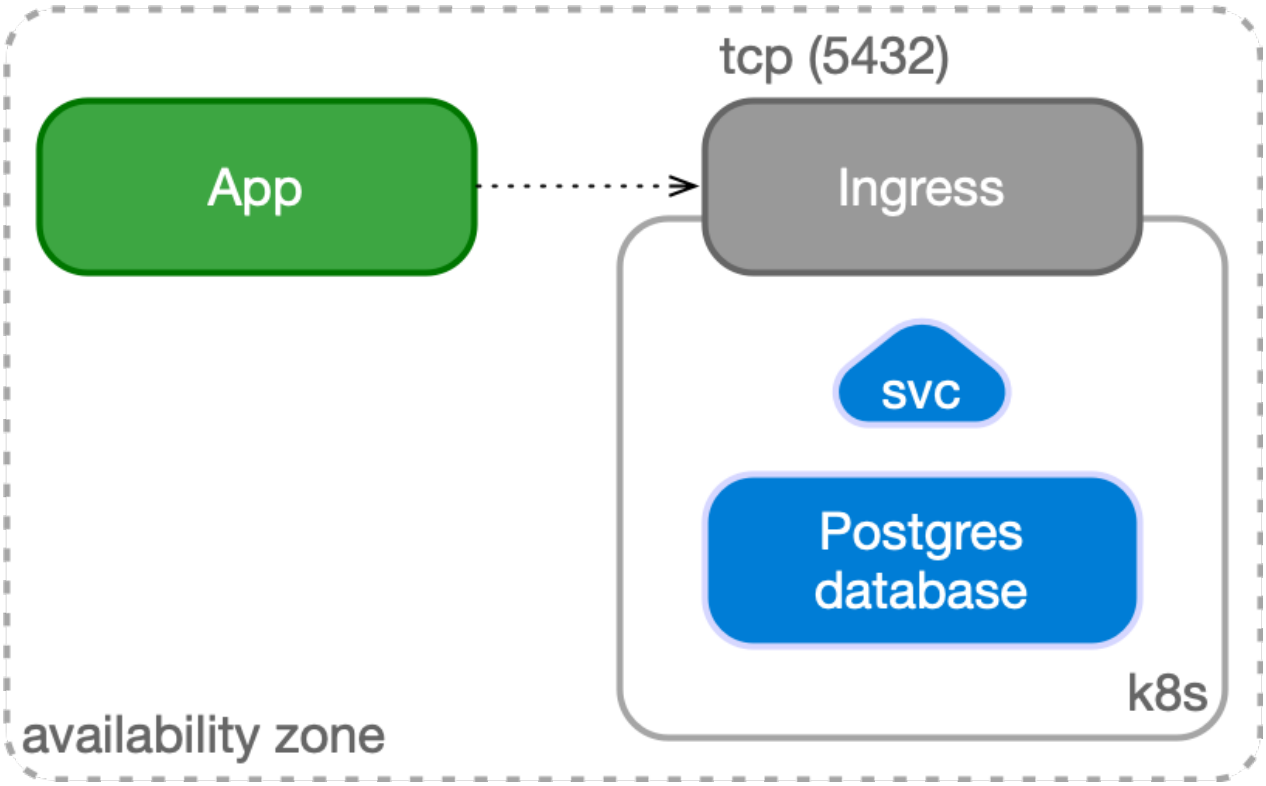
The service is exposed externally to the end user through an Ingress and the provider's load balancer facility by way of HTTPS.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Case 2: Applications outside Kubernetes

Another possible use case is to manage your PGD database inside Kubernetes while having your applications outside of it, for example, in a virtualized environment. In this case, PGD is represented by an IP address or host name and a TCP port, corresponding to the defined Ingress resource in Kubernetes.

The application can still benefit from a TLS connection to PGD.



Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

5 Architecture

Consider these main architectural aspects when deploying EDB Postgres Distributed in Kubernetes.

EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) is a Kubernetes operator designed to deploy and manage EDB Postgres Distributed clusters running in private, public, hybrid, or multi-cloud environments.

Relationship with EDB Postgres Distributed

EDB Postgres Distributed (PGD) is a multi-master implementation of Postgres designed for high performance and availability. PGD generally requires deployment using Trusted Postgres Architect (TPA), a tool that uses Ansible to provision and deploy PGD clusters.

CNPG-GC offers a different way of deploying PGD clusters, leveraging containers and Kubernetes. The advantages are that the resulting architecture:

- Is self-healing and robust.
- Is managed through declarative configuration.
- Takes advantage of the vast and growing Kubernetes ecosystem.

Relationship with EDB Postgres for Kubernetes

A PGD cluster consists of one or more *PGD groups*, each having one or more *PGD nodes*. A PGD node is a Postgres database. CNPG-GC internally manages each PGD node using the `Cluster` resource as defined by EDB Postgres for Kubernetes, specifically a cluster with a single instance (that is, no replicas).

You can configure the single PostgreSQL instance created by each `Cluster` in the `.spec.cnp` section of the PGD Group spec.

In CNPG-GC, as in EDB Postgres for Kubernetes, the underlying database implementation is responsible for data replication. However, it's important to note that failover and switchover work differently, entailing Raft election and nominating new write leaders. EDB Postgres for Kubernetes handles only the deployment and healing of data nodes.

Managing PGD using CNPG-GC

The CNPG-GC operator can manage the complete lifecycle of PGD clusters. As such, in addition to PGD nodes (represented as single-instance `Clusters`), it needs to manage other objects associated with PGD.

PGD relies on the Raft algorithm for distributed consensus to manage node metadata, specifically agreement on a *write leader*. Consensus among data nodes is also required for operations such as generating new global sequences or performing distributed DDL.

These considerations force additional actors in PGD above database nodes.

CNPG-GC manages the following:

- Data nodes. A node is a database and is managed by PG4K, creating a `Cluster` with a single instance.
- Witness nodes are basic database instances that don't participate in data replication. Their function is to guarantee that consensus is possible in groups with an even number of data nodes or after network partitions. Witness nodes are also managed using a single-instance `Cluster` resource.
- PGD proxies act as Postgres proxies with knowledge of the write leader. PGD proxies need information from Raft to route writes to the current write leader.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Proxies and routing

PGD groups assume full mesh connectivity of PGD nodes. Each node must be able to connect to every other node using the appropriate connection string (a `libpq` -style DSN). Write operations don't need to be sent to every node. PGD takes care of replicating data after it's committed to one node.

For performance, we often recommend sending write operations mostly to a single node, the *write leader*. Raft is used to identify which node is the write leader and to hold metadata about the PGD nodes. PGD proxies are used to transparently route writes to write leaders and to quickly pivot to the new write leader in case of switchover or failover.

It's possible to configure *Raft subgroups*, each of which can maintain a separate write leader. In CNPG-GC, a PGD group containing a PGD proxy comprises a Raft subgroup.

Two kinds of routing are available with PGD proxies:

- Global routing uses the top-level Raft group and maintains one global write leader.
- Local routing uses subgroups to maintain separate write leaders. Local routing is often used to achieve geographical separation of writes.

In CNPG-GC, local routing is used by default, and a configuration option is available to select global routing.

For more information on routing with Raft, see Proxies, Raft, and Raft subgroups in the PGD documentation.

PGD architectures and high availability

EDB proposes several recommended architectures to make good use of PGD's distributed multi-master capabilities and to offer high availability.

The PGD deployment patterns are built from either one group in a single location or two groups in two separate locations. See Choosing your deployment pattern in the PGD documentation for more information.

Deploying PGD on Kubernetes

CNPG-GC leverages Kubernetes to deploy and manage PGD clusters. As such, some adaptations are necessary to translate PGD into the Kubernetes ecosystem.

Images and operands

You can configure PGD to run one of three Postgres distributions. See the PGD documentation to understand the features of each distribution.

To function in Kubernetes, containers are provided for each Postgres distribution. These are the *operands*. In addition, the operator images are kept in those same repositories.

See EDB private image registries for details on accessing the images.

Kubernetes architecture

Some of the points of the PG4K document on Kubernetes architecture are reproduced here. See the PG4K documentation for details.

Kubernetes natively provides the possibility to span separate physical locations. These physical locations are also known as data centers, failure zones, or, more frequently, *availability zones*. They are connected to each other by way of redundant, low-latency, private network connectivity.

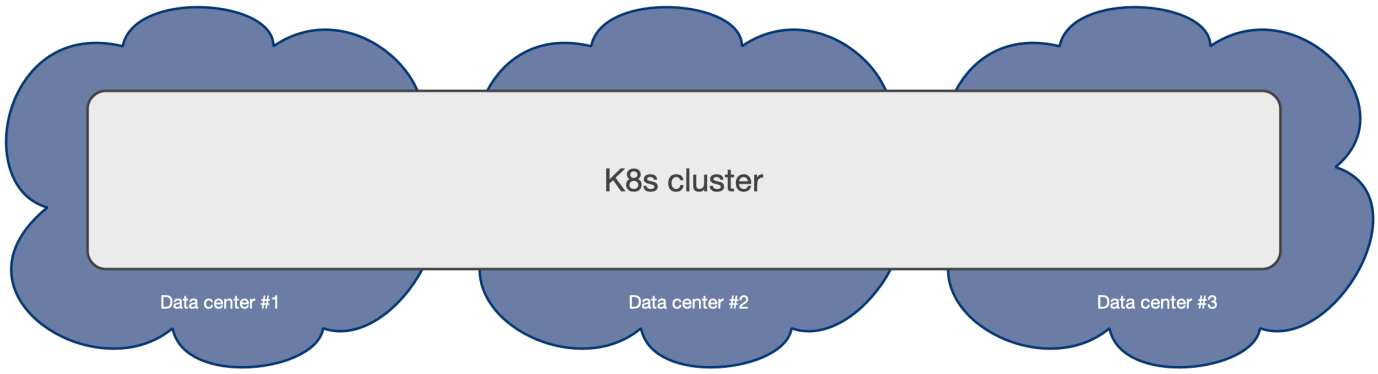
Being a distributed system, the recommended minimum number of availability zones for a *Kubernetes cluster* is three. This minimum makes the control plane resilient to the failure of a single zone. This means that each data center is active at any time and can run workloads simultaneously.

You can install CNPG-GC in a single Kubernetes cluster or across multiple Kubernetes clusters.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Single Kubernetes cluster

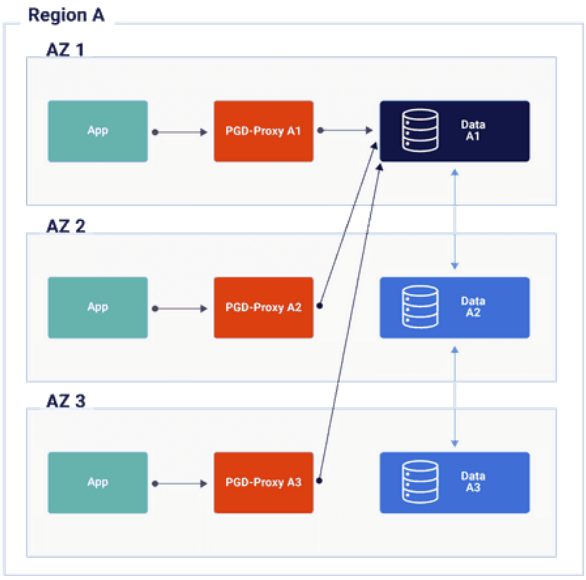
A multi-availability-zone Kubernetes architecture is typical of Kubernetes services managed by cloud providers. Such an architecture enables the CNPG-GC and the EDB Postgres for Kubernetes operators to schedule workloads and nodes across availability zones, considering all zones active.



PGD clusters can be deployed in a single Kubernetes cluster and take advantage of Kubernetes availability zones to enable high-availability architectures, including the Always On recommended architectures.

PGD clusters can be deployed in a single Kubernetes cluster that spans multiple availability zones to enable high-availability architectures with a single location.

To learn more about the pros and cons of a single location deployment pattern, refer to the PGD deployment patterns documentation.



The CNPG-GC operator can control the scheduling of pods (that is, which pods go to which data center) using affinity, tolerations, and node selectors, as is the case with EDB Postgres for Kubernetes. Individual scheduling controls are available for proxies as well as nodes.

See the Kubernetes documentation on scheduling, and Scheduling in the EDB Postgres for Kubernetes documentation for more information.

Effective Implementation of PGBouncer

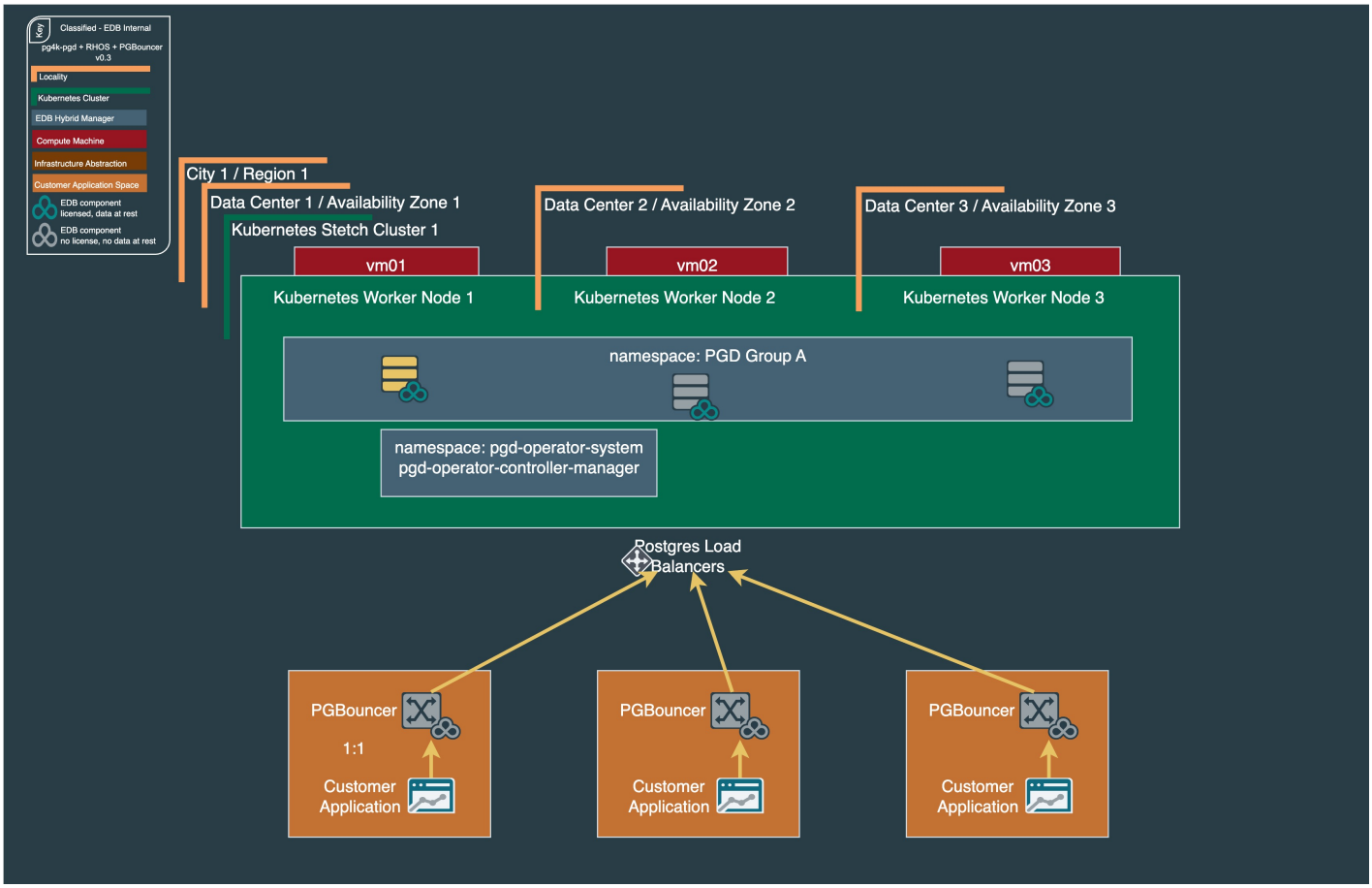
Effective application-side connection pooling is critical for optimizing application thread efficiency, limiting the overhead of creating new connections for servicing atomic requests. While application stacks with robust, integrated pooling (e.g., the JDBC Pool in a JVM) are ideal, many applications require an external solution. In these cases, PgBouncer serves as an excellent application-side connection pooler.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

However, the deployment strategy for PgBouncer is crucial. A conventional approach often places it as a centralized, database-side tier fronting the Postgres cluster. In a highly-available architecture like PGD, this centralized model is an anti-pattern because it creates a single point of failure (SPOF) and a potential performance bottleneck. A more resilient and scalable approach is to deploy PgBouncer in a sidecar pattern, application-side, creating a one-to-one association with each application instance. This strategy offers several advantages:

- Fault Isolation: The failure of a single PgBouncer instance only impacts its associated application instance, not the entire system.
- Horizontal Scalability: Pooling resources scale naturally with the application, eliminating the need to manage a separate, auto-scaling pooling tier.
- Simplified Configuration: The application connects to a stable local endpoint (e.g., localhost:6432), abstracting away the database topology.

Within the CNPG-GC ecosystem, it's vital to distinguish the roles of different components. PGD Proxy is the mandatory, intelligent entry point for routing and load balancing; it understands the PGD cluster's state and directs connections appropriately. The newer PGD Connection Manager is also PGD-aware and adds important functions, including some pooling capability. By contrast, PgBouncer is not embedded with PGD logic; it is not clustered or cluster-aware. This lack of integration reinforces its value as an application-side component, where it can manage connection efficiency without interfering with the PGD-aware routing and high-availability managed by PGD Proxy.

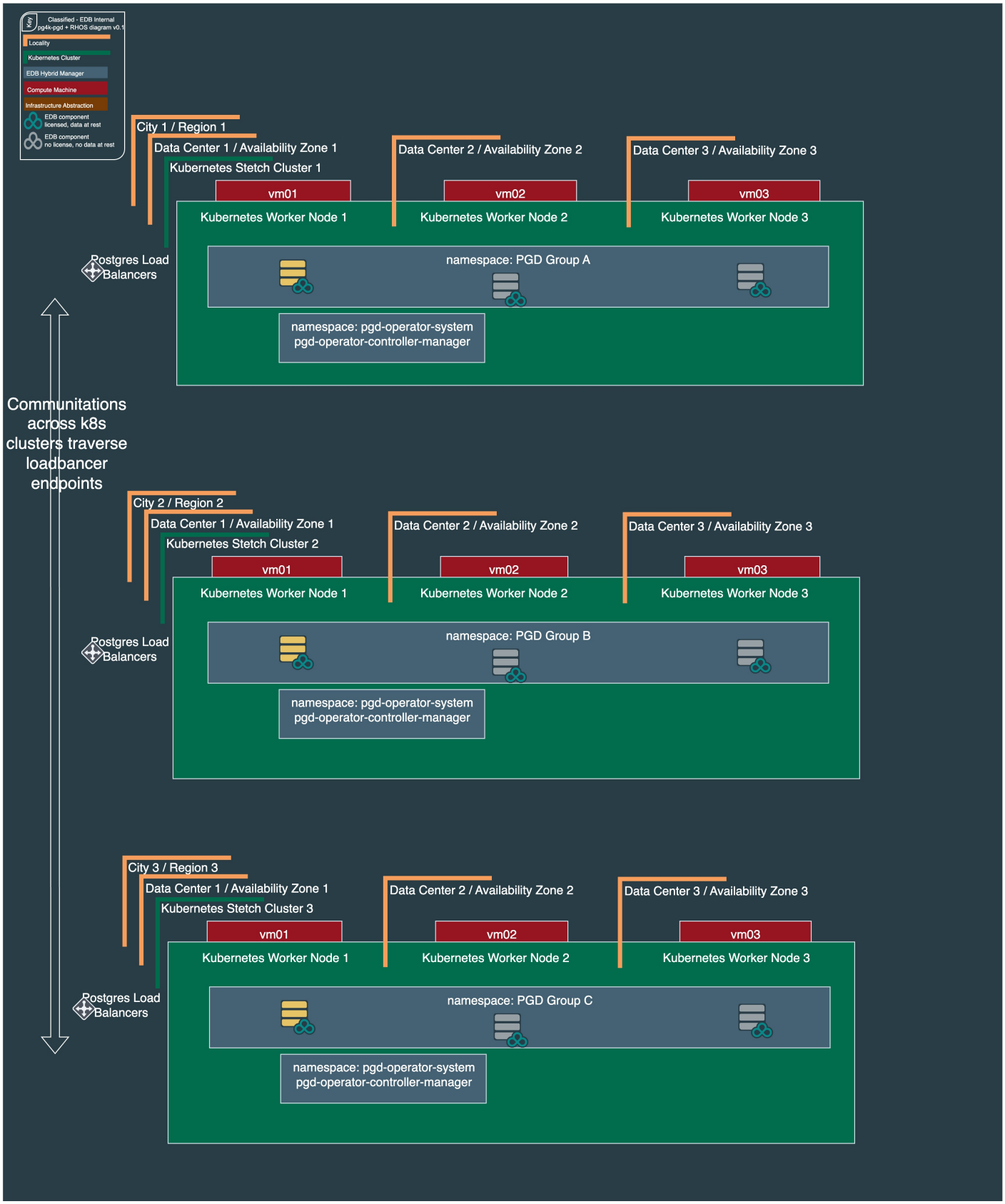


Multiple Kubernetes clusters

EDB Postgres Distributed (PGD) can be deployed in a multi-cluster topology, spanning distinct Kubernetes clusters across different regions or availability zones. This architecture necessitates reliable, underlying network connectivity between all participating clusters. The fundamental requirement is that every PGD service, identified by its -node or -group suffix, must be discoverable and routable from all other clusters in the distributed system. While PGD for Kubernetes supports the full range of deployment patterns, addressing the cross-cluster networking challenge can be achieved by exposing services through network load balancers.

Depicted here is a reference architecture with three PGD Data Groups (A, B, and C), each containing three PGD Nodes, distributed across nine data centers in three cities to maximize availability. The diagram is simplified to highlight the crucial detail that internode communication across the Kubernetes cluster boundaries is facilitated by standard TCP network load balancers, with each PGD service endpoint being resolved via a unique FQDN. This service exposure strategy effectively decouples the networking domains of each Kubernetes cluster, preserving their operational independence without requiring a complex network fabric or service mesh.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.



Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

6 Installation and upgrade

> **OpenShift**
>
> For instructions on how to install EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) on Red Hat OpenShift Container Platform, see "OpenShift".

Installing the operator on Kubernetes

Obtaining an EDB subscription token

> **Important**
>
> You must obtain an EDB subscription token to install CNPG-GC. The token grants access to the EDB private software repositories.

Installing CNPG-GC requires an EDB Repos 2.0 token to gain access to the EDB private software repositories. For instructions on obtaining this token, see: Get your token.

Then set the Repos 2.0 token as an environment variable `EDB_SUBSCRIPTION_TOKEN`:

```
EDB_SUBSCRIPTION_TOKEN=<your-token>
```

> **Warning**
>
> The token is sensitive information. Ensure that you don't expose it to unauthorized users.

You can now proceed with the installation.

Using the Helm chart

You can install the operator using the provided Helm chart.

Directly using the operator manifest

You can deploy the CNPG-GC operator directly using the manifest. This manifest installs both CNPG-GC operator and the latest supported EDB Postgres for Kubernetes operator in the same namespace. To deploy the operators using the manifest, follow the steps below:

Install the cert-manager

CNPG-GC requires Cert Manager 1.10 or higher. You can follow the installation guide or use this command to deploy cert-manager:

```
kubectl apply -f \
  https://github.com/cert-manager/cert-manager/releases/download/v1.16.2/cert-manager.yaml
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Install the EDB pull secret

Before installing CNPG-GC, you need to create a *pull secret* for the EDB container registry.

The pull secret needs to be saved in the namespace where the operator will reside (`pgd-operator-system` by default). Create the `pgd-operator-system` namespace using this command:

```
kubectl create namespace pgd-operator-system
```

To create the pull secret, run the following command:

```
kubectl create secret -n pgd-operator-system docker-registry edb-pull-secret \
  --docker-server=docker.enterprisedb.com \
  --docker-username=k8s \
  --docker-password=${EDB_SUBSCRIPTION_TOKEN}
```

Install the operator manifest

After the pull-secret is added to the namespace, you can install the operator like any other resource in Kubernetes: through a YAML manifest applied via `kubectl`.

To install the manifest for the latest version of the operator:

```
kubectl apply --server-side -f \
  https://get.enterprisedb.io/pg4k-pgd/pg4k-pgd-1.2.0-rc1.yaml
```

Check the operator deployment:

```
kubectl get deployment -n pgd-operator-system pgd-operator-controller-
manager
```

> **Note**
>
> As CNPG-GC internally manages each PGD node using the `Cluster` resource defined by EDB Postgres for Kubernetes, you also need to have the EDB Postgres for Kubernetes operator installed as a dependency. The manifest used above contains a well-tested version of EDB Postgres for Kubernetes operator, which will be installed into the same namespace as the CNPG-GC operator.

Details about the deployment

In Kubernetes, the operator is by default installed in the `pgd-operator-system` namespace as a Kubernetes `Deployment`. The name of this deployment depends on the installation method. When installed through the manifest, by default it's named `pgd-operator-controller-manager`. When installed via Helm, by default the deployment name is derived from the Helm release name, appended with the suffix `-edb-postgres-distributed-for-kubernetes` (that is, `<name>-edb-postgres-distributed-for-kubernetes`).

> **Note**
>
> With Helm, you can customize the name of the deployment via the `fullnameOverride` field in the *"values.yaml"* file.

You can get more information using the `describe` command in `kubectl`:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
$ kubectl get deployments -n pgd-operator-
system
NAME                READY   UP-TO-DATE   AVAILABLE
AGE
<deployment-name>   1/1     1            1
18m
```

```
kubectl describe deploy
\
  -n pgd-operator-system \
  <deployment-name>
```

As with any deployment, it sits on top of a ReplicaSet and supports rolling upgrades. The default configuration of the CNPG-GC operator is a deployment of a single replica, which is suitable for most installations. If the node where the pod is running isn't reachable anymore, the pod will be rescheduled on another node.

If you require high availability at the operator level, it's possible to specify multiple replicas in the deployment configuration, given that the operator supports leader election. In addition, you can take advantage of taints and tolerations to make sure that the operator does not run on the same nodes where the actual PostgreSQL clusters are running. (This might even include the control plane for self-managed Kubernetes installations.)

> **Operator configuration**
>
> You can change the default behavior of the operator by overriding some default options. For more information, see"Operator configuration".

Deploy PGD clusters

Be sure to create a cert issuer before you start deploying PGD clusters. The Helm chart prompts you to do this, but in case you miss it, you can run, for example:

```
kubectl apply -f \
  https://raw.githubusercontent.com/EnterpriseDB/edb-postgres-for-kubernetes-
charts/main/hack/samples/issuer-selfsigned.yaml
```

With the operators and a self-signed cert issuer deployed, you can start creating PGD clusters. SeeQuick start for an example.

Default operand images

By default, each operator release binds a default version and flavor for PGD and PGD Proxy images. If the image names aren't specified in the `spec.pgd.imageName` and `spec.pgdproxy.imageName` fields of the PGDGroup YAML file, the default images are used.

You can overwrite default images using the `pgd-operator-controller-manager-config` operator configuration map. For more details, see CNPG-GC operator configuration.

You can find the images the PGD cluster is using by checking the PGDGroup status:

```
kubectl get pgdgroup <pgdgroup name> -o yaml | yq
".status.image"
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Specifying operand images

This example shows a PGD cluster using explicit image names:

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: group-example-
customized
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  imageName: docker.enterprisedb.com/k8s/edb-postgres-extended-pgd:17.6-pgd590-ubi9
  pgdProxy:
    imageName: docker.enterprisedb.com/k8s/edb-pgd-proxy:5.9.0-ubi9
  imagePullSecrets:
  - name: registry-pullsecret
  pgd:
    parentGroup:
      name: world
      create: true
  cnp:
    storage:
      size:
1Gi
```

Specifying operand images using ImageCatalog

Since the release of version 1.1.1, the PGD4K-PGD operator supports using `ImageCatalog` to specify operand images.

Different `ImageCatalogs` are available based on PGD versions for each PostgreSQL flavor. Note that the image included in the `ImageCatalog` are ubi-9 based.

- EDB Postgres Advanced PGD: `https://get.enterprisedb.io/pgd-k8s-image-catalogs/epas-k8s-pgd<PGD_VERSION>-ubi9.yaml`
- EDB Postgres Extended PGD: `https://get.enterprisedb.io/pgd-k8s-image-catalogs/pgextended-k8s-pgd<PGD_VERSION>-ubi9.yaml`
- Postgres Community PGD: `https://get.enterprisedb.io/pgd-k8s-image-catalogs/postgresql-k8s-pgd<PGD_VERSION>-ubi9.yaml`

You can create an `ImageCatalog` in the PGD cluster namespace and reference it in your YAML file.

This command creates an EDB Postgres Extended PGD 5.9 `ImageCatalog`:

```
kubectl create -f
\
  https://get.enterprisedb.io/pgd-k8s-image-catalogs/epas-k8s-pgd5.9-ubi9.yaml
```

This example shows how to use the EDB Postgres Extended PGD 5.9 `ImageCatalog` to specify a PostgreSQL major version of 17 for the PGD operand image. The PGD Proxy image is also sourced from the `ImageCatalog`.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: group-example-
catalog
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  imageCatalogRef:
    apiGroup: pgd.k8s.enterprisedb.io
    kind: ImageCatalog
    major: 17
    name: epas-k8s-pgd59-ubi9
...
```

Operator upgrade

> **CRITICAL WARNING: UPGRADING OPERATORS**
>
> OpenShift users, or any customer attempting an operator upgrade, MUST configure the new unified repository pull secret (docker.enterprisedb.com/k8s) before running the upgrade. If the old, deprecated repository path is still in use during the upgrade process, image pull failure will occur, leading to deployment failure and potential downtime. Follow the Central Migration Guide first.

> **Important**
>
> Carefully read the release notes before performing an upgrade, as some versions might require extra steps.

The CNPG-GC (PGD4K) operator relies on the EDB Postgres for Kubernetes (PG4K) operator to manage clusters. To upgrade the EDB PGD4K operator, the PG4K operator must also be upgraded to a supported version. We recommend keeping the EDB PG4K operator on the long-term support (LTS) version, as this is the tested version compatible with the PGD4K operator.

To upgrade the EDB Postgres Distributed (PGD) for Kubernetes operator:

1. Upgrade the EDB Postgres for Kubernetes operator as a dependency.
2. Upgrade the PGD4K controller and the related Kubernetes resources.

When you upgrade the EDB Postgres for Kubernetes operator, the instance manager on each PGD node is also upgraded. For more details, seeEDB Postgres for Kubernetes upgrades.

Unless differently stated in the release notes, those steps are normally done by applying the manifest of the newer version for plain Kubernetes installations.

Compatibility among versions

CNPG-GC (PGD4K) follows semantic versioning. Every release of the operator within the same API version is compatible with the previous one. The current API version is v1, corresponding to versions 1.x.y of the operator.

The minor version of PGD4K operator is tracking a PG4K LTS release change. For example:

- PGD4K operator 1.0.x is fully tested against PG4K LTS 1.22.
- PGD4K operator 1.1.x is fully tested against PG4K LTS 1.25.
- PGD4K operator 1.2.x is fully tested against PG4K LTS 1.28.

A PGD4K operator release has the same support scope as the PG4K LTS release it's tracking.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

In addition to new features, new versions of the operator contain bug fixes and stability enhancements.

> **Important**
>
> Each version is released to maintain the most secure and stable Postgres environment. Because of this, we strongly encourage you to upgrade to the latest version of the operator.

The release notes contains a detailed list of the changes introduced in every released version of CNPG-GC. Read them before upgrading to a newer version of the software.

Most versions are directly upgradable. In that case, applying the newer manifest for plain Kubernetes installations will complete the upgrade.

When versions aren't directly upgradable, you must remove the old version (of both PGD4K and PG4K) before installing the new one. This won't affect user data, only the operator.

Upgrading to 1.0.1 on Red Hat OpenShift

On the OpenShift platform, starting from version 1.0.1, the CNPG-GC operator is required to reference the LTS releases of the PG4K operator. For example, PGD4K v1.0.1 specifically references the PG4K LTS version 1.22.x. (Any patch release of the 1.22 LTS branch is valid.)

To upgrade PGD4K operator, ensure that the PG4K operator is upgraded to a supported version first. Only then can you upgrade the PGD4K operator.

> **Important**
>
> As PGD4K v1.0.0 doesn't require referencing an LTS release of the PG4K operator, it may have been installed with any PG4K version. If the installed PG4K version is less than 1.22, you can upgrade to PG4K version 1.22 by changing the subscription channel. For more information, see Upgrading the operator. However, if the installed PG4K version is greater than 1.22, you must reinstall the operator from the stable-1.22 channel to upgrade PGD4K to v1.0.1.

Upgrading to 1.2.0 on Red Hat OpenShift

Starting with release 1.2.0, operator and operand images use a single EDB registry. Before upgrading the operator and cluster, follow the EDB Registry Migration Guide to update your image pull secrets with the new credentials.

Server-side apply of manifests

To ensure compatibility with Kubernetes 1.29 and upcoming versions, CNPG-GC now mandates the use of server-side apply when deploying the operator manifest.

While employing this installation method poses no challenges for new deployments, updating existing operator manifests using the `--server-side` option may result in errors like the following:

```
Apply failed with 1 conflict: conflict with "kubectl-client-side-apply" using..
```

If such errors arise, you can resolve them by explicitly specifying the `--force-conflicts` option to enforce conflict resolution:

```
kubectl apply --server-side --force-conflicts -f <OPERATOR_MANIFEST>
```

From then on, `kube-apiserver` is acknowledged as a recognized manager for the CRDs, eliminating the need for any further manual intervention on this matter.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Operand upgrade

Operand upgrades fall into two categories based on PostgreSQL and PGD versions:

- Minor version upgrades (for example, PostgreSQL from 17.4 to 17.5 and PGD from 5.7 to 5.8)
- Major version upgrades (for example, PostgreSQL from 16.x to 17.5, no PGD major version upgrade as only PGD 5.x is supported now)

> **Note**
>
> The PGD operand upgrade proceeds sequentially on each node. The node upgrade process is managed by the PG4K operator. For detailed information, see PostgreSQL upgrades

Checking current PGD and proxy versions

Before upgrading, you can check the current PGD and PGD proxy versions:

```
kubectl get pgdgroup <pgdgroup name> -o yaml | yq
".status.image"
```

Minor version upgrade

The PGD cluster supports in-place upgrades of the operand image's minor version, though the PostgreSQL service is temporarily unavailable during the upgrade.

Upgrade procedure

- Using default or customized image name: To upgrade the operand to a new minor version, replace the imageName in the `spec.imageName` and `spec.pgdproxy.imageName` sections of the PGD group YAML file with the new imageName. The images on each node will be upgraded sequentially and restarted accordingly.

- Using image catalog: If the PGD cluster manages image versions using an `ImageCatalog`, upgrade the image version specified in the referenced `ImageCatalog`. The PGD cluster applies the new image version.

Major version upgrade

Since release v1.1.2, the PGD4K operator supports in-place upgrades for major PostgreSQL versions. During the process, each PGD node is upgraded sequentially, with the write leader transferred to an available node before the upgrade.

Prerequisites for major version in-place upgrade

- PGD4K operator v1.1.2 or higher
- PG4K operator v1.26.0 or higher
- PGD operand 5.8 or greater

Upgrade procedure

Like minor version upgrades, initiating a major version upgrade involves updating the `spec.imageName` in the PGDGroup to point to the new operand image.

Example scenario:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Suppose you have a PGDGroup named `pgd-sample`, currently running with PostgreSQL 16.9 plus PGD 5.8.1. You plan to upgrade to PostgreSQL 17.5 plus PGD 5.8.1.

Step-by-step guidance

1. Check the current operand version you're using:

```
kubectl -n pgd get pgdgroup pgd-sample -o yaml | yq ".status.image"
```

Output:

```
pgd: docker.enterprisedb.com/k8s/edb-postgres-advanced-pgd:16.10-pgd581-ubi9
proxy: docker.enterprisedb.com/k8s/edb-pgd-proxy:5.8.1-ubi9
```

2. Update the operand image

Edit the PGDGroup and update the `spec.imageName` or patch it directly to `docker.enterprisedb.com/k8s/edb-postgres-advanced-pgd:17.5-pgd581-ubi9`

```
kubectl patch pgdgroup pgd-sample -n pgd --patch \
'{"spec": {"imageName": "docker.enterprisedb.com/k8s/edb-postgres-advanced-pgd:17.5-pgd581-ubi9"}}' \
--type=merge
```

3. Monitor the node-by-node major version upgrade

The cluster begins upgrading nodes sequentially:

Observe the upgrading node, for example, pgd-sample-3, shows "Upgrading Postgres major version".

```
> kubectl -n pgd get cluster
NAME           AGE    INSTANCES   READY   STATUS                              PRIMARY
pgd-sample-1   121m   1           1       Cluster in healthy state            pgd-sample-1-1
pgd-sample-2   118m   1           1       Cluster in healthy state            pgd-sample-2-1
pgd-sample-3   115m   1                   Upgrading Postgres major version    pgd-sample-3-1
```

During the process, the PGDGroup status is:

```
> kubectl -n pgd get pgdgroup
NAME         DATA INSTANCES   WITNESS INSTANCES   PHASE
AGE
pgd-sample   2                1                   PGDGroup - Waiting for nodes to be ready   123m
```

The upgrade of individual nodes is managed via dedicated jobs:

```
> kubectl -n pgd get job
NAME                           STATUS    COMPLETIONS   DURATION   AGE
pgd-sample-3-1-major-upgrade   Running   0/1           3m22s      3m22s
```

Check logs in the upgrade job pod for detailed upgrade status

```
kubectl -n pgd logs -f pgd-sample-3-1-major-upgrade-ldtnj
```

Once a node's major version upgrade completes, the process moves to the next node:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
NAME           AGE    INSTANCES   READY   STATUS                              PRIMARY
pgd-sample-1   128m   1           1       Cluster in healthy state            pgd-sample-1-1
pgd-sample-2   125m   1                   Upgrading Postgres major version    pgd-sample-2-1
pgd-sample-3   122m   1           1       Cluster in healthy state            pgd-sample-3-1
```

4. Confirm completion and health

Once all nodes are upgraded, the PGDGroup phase switches to Healthy.

```
NAME         DATA INSTANCES   WITNESS INSTANCES   PHASE                AGE
pgd-sample   2                1                   PGDGroup - Healthy   137m
```

Verify the overall image version:

```
kubectl -n pgd get pgdgroup  pgd-sample -o yaml | yq ".status.image"
```

Output:

```
pgd: docker.enterprisedb.com/k8s/edb-postgres-advanced-pgd:17.5-pgd581-ubi9
proxy: docker.enterprisedb.com/k8s/edb-pgd-proxy:5.8.1-ubi9
```

5. Confirm PostgreSQL version on each node:

```
kubectl -n pgd exec -it pgd-sample-1-1 -c postgres -- psql -c "select version()"
```

Output:

```
                                                
```

7 Quick start

You can test an EDB Postgres Distributed (PGD) cluster on your laptop or computer using EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) on a single local Kubernetes cluster built with Kind.

> **Warning**
>
> These instructions are only for demonstration, testing, and practice purposes and must not be used in production.

This quick start shows you how to start an EDB Postgres Distributed cluster on your local Kubernetes installation so you can experiment with it.

> **Important**
>
> To connect to the Kubernetes cluster, make sure that you have `kubectl` installed on your machine. See the Kubernetes documentation on installing `kubectl`.

Part 1 - Set up the local Kubernetes playground

Install Kind, a tool for running local Kubernetes clusters using Docker container nodes. (Kind stands for Kubernetes IN Docker.) If you already have access to a Kubernetes cluster, you can skip to Part 2.

Install Kind on your environment following the instructions in Kind Quick Start. Then, create a Kubernetes cluster:

```
kind create cluster --name
pgd
```

Part 2 - Install CNPG-GC

After you have a Kubernetes installation up and running on your laptop, you can install CNPG-GC.

See Installation for details.

Part 3 - Deploy a PGD cluster

As with any other deployment in Kubernetes, to deploy a PGD cluster you need to apply a configuration file that defines your desired `PGDGroup` resources that make up a PGD cluster.

Some sample files are included in the CNPG-GC repository. The flexible_3regions.yaml manifest contains the definition of a PGD cluster with two data groups and a global witness node spread across three regions. Each data group consists of two data nodes and a local witness node.

> **Regions and availability zones**
>
> When creating Kubernetes clusters in different regions or availability zones for cross-regional replication, ensure the clusters can communicate with each other by enabling network connectivity. Specifically, every service created with a `-node` or `-group` suffix must be discoverable by all other `-node` and `-group` services.

> **Further reading**
>
> For more details about the available options, see the "API Reference" section.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

You can deploy the `flexible-3-regions` example by saving it first and running:

```
kubectl apply -f flexible_3regions.yaml
```

You can check that the pods are being created using the `get pods` command:

```
kubectl get
pods
```

The pods are being created as part of PGD nodes. As described in Architecture, they're implemented on top of EDB Postgres for Kubernetes clusters.

You can list the clusters then, which shows the PGD nodes:

```
$ kubectl get
clusters
NAME         AGE     INSTANCES   READY   STATUS
PRIMARY
region-a-1   2m50s   1           1       Cluster in healthy state   region-a-1-
1
region-a-2   118s    1           1       Cluster in healthy state   region-a-2-
1
region-a-3   91s     1           1       Cluster in healthy state   region-a-3-
1
…
…
```

Ultimately, the PGD nodes are created as part of the PGD groups that make up your PGD cluster.

```
$ kubectl get
pgdgroups
NAME       DATA INSTANCES   WITNESS INSTANCES   PHASE
AGE
region-a   2                1                   PGDGroup - Healthy
4m50s
region-b   2                1                   PGDGroup - Healthy
4m50s
region-c   0                1                   PGDGroup - Healthy
4m50s
```

Notice how the region-c group is only a witness node.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

8 Operator configuration

EDB Postgres® AI for CloudNativePG™ Global Cluster Operator configuration

The operator for EDB Postgres® AI for CloudNativePG™ Global Cluster (PGD4K) is installed from a standard deployment manifest and follows the convention over configuration paradigm. While this is fine in most cases, there are some scenarios where you want to change the default behavior, such as:

- defining a different default image for PGD or PGD proxy
- defining an additional pull secret for operator and operand images

The behavior of the operator can be customized through a `ConfigMap` / `Secret` that is located in the same namespace of the operator deployment and with `pgd-operator-controller-manager-config` as the name.

> **Important**
>
> Any change to the config's `ConfigMap` / `Secret` will not be automatically detected by the operator, - and as such, it needs to be reloaded (see below). Moreover, changes only apply to the resources created after the configuration is reloaded.

> **Important**
>
> The operator first processes the ConfigMap values and then the Secret's, in this order. As a result, if a parameter is defined in both places, the one in the Secret will be used.

Available options

The operator looks for the following environment variables to be defined in the `ConfigMap` / `Secret`:

Name	Description
`PGD_IMAGE_NAME`	PGD operand image name to overwrite the default one in manifest.
`PGD_PROXY_IMAGE_NAME`	PGD proxy operand image name to overwrite the default one in manifest.
`PULL_SECRET_NAME`	Name of an additional pull secret to be defined in the operator's namespace and to be used to download images
`BOOTSTRAP_JOB_IMAGE_NAME`	Bootstrap job image name is the image name to overwrite the default bootstrap job image if barman-cloud plugin is used (`k8s.pgd.enterprisedb.io/bootstrapJobImage` annotation still takes precedence, if present in PGDGroup)
`WEBHOOK_CERT_DIR`	Webhook cert dir is the directory where the certificates for the webhooks need to be written. This is different between plain Kubernetes and OpenShift
`OPERATOR_IMAGE_NAME`	Operator image name is the name of the image of the operator, that is used to bootstrap Pods
`POSTGRES_TIMEOUT`	Postgres timeout contains the timeout in seconds used when communicating with a Postgres instance (Defaults to 15)
`MAX_CONCURRENT_RECONCILES`	Max concurrent reconciles is the maximum number of concurrent Reconciles which can be run (Defaults to 15)
`MONITORING_QUERIES_CONFIGMAP`	Name of the ConfigMap in the operator namespace containing default monitoring queries to distribute to all PGDGroup namespaces (see Default Monitoring Queries)
`MONITORING_QUERIES_SECRET`	Name of the Secret in the operator namespace containing default monitoring queries to distribute to all PGDGroup namespaces (see Default Monitoring Queries)
`OPERATOR_LOG_LEVEL`	Operator log level is the log level for the operator (Defaults to empty), available options are: trace, debug, info, warn, or error. This option will override the log level defined in deployment

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

When you specify an additional pull secret name using the `PULL_SECRET_NAME` parameter, the operator will use that secret to create a pull secret for every created PGD group. That secret will be named `<group-name>-pull`.

The namespace where the operator looks for the `PULL_SECRET_NAME` secret is where you installed the operator. If the operator is not able to find that secret, it will ignore the configuration parameter.

Defining an operator config map

The example below customizes the behavior of the operator, by defining an additional image pull secret.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: pgd-operator-controller-manager-config
  namespace: pgd-operator-system
data:
  PULL_SECRET_NAME: additional-pull-
secret
```

Defining an operator secret

The example below customizes the behavior of the operator, by defining the default operand image names.

```
apiVersion: v1
kind:
Secret
metadata:
  name: pgd-operator-controller-manager-config
  namespace: pgd-operator-system
type:
Opaque
stringData:
  PGD_IMAGE_NAME: docker.enterprisedb.com/k8s/postgresql-pgd:17.5-pgd581-
ubi9
  PGD_PROXY_IMAGE_NAME: docker.enterprisedb.com/k8s/edb-pgd-proxy:5.8.1-ubi9
```

Restarting the operator to reload configs

For the change to be effective, you need to recreate the operator pods to reload the config map. If you have installed the operator on Kubernetes using the manifest you can do that by issuing:

```
kubectl rollout restart deployment \
    -n pgd-operator-system \
    pgd-operator-controller-manager
```

Otherwise, If you have installed the operator using OLM, or you are running on Openshift, run the following command specifying the namespace the operator is installed in:

```
kubectl delete pods -n [NAMESPACE_NAME_HERE] \
  -l app.kubernetes.io/name=pgd-operator
```

> **Warning**
>
> Customizations will be applied only to `PGD groups` resources created after the reload of the operator deployment.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

pprof HTTP Server

The operator can expose a PPROF HTTP server with the following endpoints on `localhost:6060`:

- `/debug/pprof/`. Responds to a request for "/debug/pprof/" with an HTML page listing the available profiles
- `/debug/pprof/cmdline`. Responds with the running program's command line, with arguments separated by NULL bytes.
- `/debug/pprof/profile`. Responds with the pprof-formatted cpu profile. Profiling lasts for duration specified in seconds GET parameter, or for 30 seconds if not specified.
- `/debug/pprof/symbol`. Looks up the program counters listed in the request, responding with a table mapping program counters to function names.
- `/debug/pprof/trace`. Responds with the execution trace in binary form. Tracing lasts for duration specified in seconds GET parameter, or for 1 second if not specified.

To enable the operator you need to edit the operator deployment add the flag `--pprof-server=true`.

You can do this by executing these commands:

```
kubectl edit deployment -n pgd-operator-system pgd-operator-controller-manager
```

Then on the edit page scroll down the container args and add `--pprof-server=true`, as in this example:

```
      containers:
      - args:
        - controller
        - --enable-leader-
election
        - --config-map-name=pgd-operator-controller-manager-
config
        - --secret-name=pgd-operator-controller-manager-
config
        - --log-
level=info
        - --pprof-server=true # relevant
line
        command:
        -
/manager
```

Save the changes; the deployment now will execute a roll-out, and the new pod will have the PPROF server enabled.

EDB Postgres for Kubernetes Operator configuration

The CNPG-GC (PGD4K) operator manages each PGD node using the `Cluster` resource defined by the EDB Postgres for Kubernetes (PG4K) operator. Therefore, the configuration for the PG4K operator remains applicable. If you need to modify the default behavior of the PG4K operator, like change the default monitor query, setting the EDB_LICENSE_KEY when using EPAS, you can do so by defining a `Configmap` or `Secrets` within the PG4K operator's namespace.

For reference, there are available options supported by PG4K operator's `Configmap` or `Secrets`.

You can follow the steps defined operator configuration for more details.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

9 Examples of configuration

> **Important**
>
> The available examples are for demonstration and experimentation purposes only.

These examples are configuration files for setting up your EDB Postgres Distributed cluster in a Kubernetes environment.

Basics

- **Flexible 3 regions**

  `flexible_3regions.yaml` : provides a PGD cluster with two data groups and a global witness node spread across three regions, where each data group consists of two data nodes and a local witness node.

- **Flexible 3 regions using physical join**

  `flexible_3regions_physical_join.yaml` : provides a PGD cluster with two data groups and a global witness node just like `Flexible 3 regions` . The difference is the group join in this sample is using physical join. Group region-a is the first group to startup (`spec.pgd.parentGroup.create=true`), nodes in region-a join with each other using physical join. Group region-b define its `spec.pgd.groupJoinMethod=physical` , which means the first node in region-b will start and physical join a remote node, the join target is defined in `spec.pgd.discovery` section of group region-b, The remaining nodes in region-b will join with each other using physical join as `spec.cnp.joinMethod=physical` . region-c is a witness group which will always use logical join.

- **Flexible 3 regions with pre-provisioned client secrets**

  `flexible_3regions_provisioned_secrets.yaml`

  provides a PGD cluster with two data groups and a global witness node just like `Flexible 3 regions` . In each PGD group, the server TLS certificate is managed by the operator and cert-manager. The client replication certificate is pre-provisioned. Here are the steps to set up this sample:

1. Establish the self-signed issuer and certificate in target namespace. Since we are using a pre-provisioned client replication certificate, we can delete the client secrets, certificate, and issuer created by `issuer-selfsigned.yaml` .

   ```
   kubectl -n <namespace> apply -f issuer-selfsigned.yaml
   kubectl -n <namespace> delete secrets/client-ca-key-pair
   \
       certificate/client-ca
   \
       issuer/client-ca-
   issuer
   ```

2. Create the PGD group. The pre-provisioned client certificate and its CA certificate are included in the `flexible_3regions_provisioned_secrets.yaml` file as well.

   ```
   kubectl -n <namespace> apply -f
   flexible_3regions_provisioned_secrets.yaml
   ```

Flexible 3 regions in different namespaces

`flexible_3regions_3ns.yaml`

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

provides a sample that builds upon the `Flexible 3 regions` example. Each region is assigned a different namespace. The `discovery` and `connectivity` section are changed in accordance with the namespaces. To set up this sample, we need to pre-create the `client-ca-key-pair` and `server-ca-key-pair` secrets in each namespace utilizing private keys generated with the ECDSA algorithm. This ensures that the server TLS certificates and client replication certificates generated by `cert-manager` are signed from the same private key.

- Create the namespaces `region-a`, `region-b` and `region-c`.
- Create the CA secrets using issuer-ecdsa-key.yaml in each namespace for the certificates.

```
kubectl -n region-a apply -f issuer-ecdsa-key.yaml
kubectl -n region-b apply -f issuer-ecdsa-key.yaml
kubectl -n region-c apply -f issuer-ecdsa-key.yaml
```

- Create the self-signed issuer and corresponding certificate in each namespace. The CA secrets `client-ca-key-pair` and `server-ca-key-pair` will be refreshed with `ca.crt`.

```
kubectl -n region-a apply -f issuer-selfsigned.yaml
kubectl -n region-b apply -f issuer-selfsigned.yaml
kubectl -n region-c apply -f issuer-selfsigned.yaml
```

- Create the PGD group

```
kubectl apply -f
flexible_3regions_3ns.yaml
```

parted group cleanup

`flexible_cleanup.yaml`

provides a sample of using PGDGroupCleanup to clean up the metadata of `region-b`. The cleanup is run from `region-a`. All nodes belonging to `region-b` need to be in `PARTED` status before running this CR.

Backup and restore

- **sample group with two schedulers**

  `group_example_with_2schedulers.yaml`

  provides a PGD group sample with two scheduled backups configured: one using volumeSnapshot, the other using barmanObjectStore.

- **sample group with backup and restore using** `barmanObjectStore`

  `group_example_with_barman_backup.yaml`

  provides a backup sample for a three-node PGD group. This setup includes scheduled backups and continuous WAL archiving to barmanObjectStore.

: `group_example_with_barman_restore.yaml` : offers a restore sample for three regions PGD groups. In the first region, group `group-example-with-barman-restore-a` is restored from backups, and parent group `world` is created in this group. In the second and third region, group `group-example-with-barman-restore-b` and `group-example-with-barman-restore-c` are created from scratch, and respectively join the restored group.

sample group with backup and restore using `volumeSnapshot`

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

`group_example_with_vs_backup.yaml`

specifies a three-node PGD group configured with scheduled volume snapshot backup and continuous WAL archiving to barmanObjectStore.

: `group_example_with_vs_restore.yaml` : defines full restore from volume snapshot backup.

: `group_example_with_vs_pitr.yaml` : defines restore from volume snapshot backup followed by point-in-time recovery.

> **Note**
>
> The `volumeSnapshot` sample above utilizes the `csi-hostpath-sc` storage class. Please verify that your storage class supports volume snapshots. For more details, refer to Backup on volume snapshots.

Read node routing

sample group with read node routing enabled

`group_example_with_readnode.yaml`

offers a sample with read node routing enabled.

LDAP

- **sample group using LDAP with bind and search**

  `group_example_with_ldap_bind_search.yaml`

  provides an LDAP sample of PGD group configured to use `bind` and `search` for authentication.

- **sample group use ldap with simple bind**

  `group_example_with_ldap_simple_bind.yaml`

  provides an LDAP sample of PGD group to use simple bind for authentication.

Managed roles

sample group with managed roles

`group_example_with_managed.yaml`

provides a PGD group sample with managed roles and managed services.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

TDE

sample group using tde

`group_example_with_tde.yaml`

provides a PGD group sample with TDE enabled. Since TDE requires PostgreSQL to be a specific flavor and version, the YAML file includes configurations for both the PGD and PGD proxy images.

Configurations

- **sample group with mutations**

  `group_example_with_mutations.yaml`

  offers a PGD group sample with `always` mutation configured.

- **sample group with service template configured**

  `group_example_with_service_template.yaml`

  offers a sample with `groupSevriceTemplate` , `nodeServiceTemplate` `proxyServiceTemplate` and `proxyReadServiceTemplate` configured.

- **sample group with customized settings**

  `group_example_customized.yaml`

  offers a PGD group sample with:

- pgd and proxy image customized
- postgres parameters customized
- data and witness nodes storage configuration customized separately
- data instance pod environments variables customized
- data and witness nodes WAL segment size customized
- enable the `globalRouting` , proxy now points to the writeLead of the global group

- **sample group with operand image customized using ImageCatalog**

  `group_example_catalog.yaml`

  offers a PGD group sample with pgd and proxy image customized using ImageCatalog.

- **sample group with scale up using physical join**

  `group_example_with_physical_join.yaml`

  offers a PGD group sample with physical join configured.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- **sample group with witness configuration**

  `group_example_witness_config.yaml`

  offers a PGD group having witness node configuration. Once the witness node is configured using `spec.witness` section, it will not share the configuration from `spec.cnp` section.

For a list of available options, see the"API Reference" page.

> **Note**
>
> The PGD group sample above requires cert-manager with a self-signed issuer. You can use issuer-selfsigned.yaml to create the self-signed issuer before setting up the PGD group.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

11 Managing EDB Postgres Distributed (PGD) databases

As described in the architecture document, EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) is an operator created to deploy PGD databases. By leveraging the Kubernetes ecosystem, it can offer self-healing and declarative control. The operator is also responsible for the backup and restore operations. See Backup.

However, many of the operations and control of PGD clusters aren't managed by the operator. The pods created by CNPG-GC come with the PGD CLI installed. You can use this tool, for example, to execute a switchover.

PGD CLI

> **Warning**
>
> Don't use the PGD CLI to create and delete resources. For example, avoid the `create-proxy` and `delete-proxy` commands. Provisioning of resources is under the control of the operator, and manual creation and deletion isn't supported.

As an example, execute a switchover command.

We recommend that you use the PGD CLI from proxy pods.

> **Note**
>
> If you are running the PGD CLI inside a PGD instance pod, set the BDR database in the PGD DSN. Either add `--dsn "dbname=$APP_DATABASE"` to each command, or export `PGD_CLI_DSN=dbname=$APP_DATABASE` for the whole session. On the OpenShift pods (which run with an arbitrary user), include the PostgreSQL user as well:
>
> - Use `--dsn "dbname=$APP_DATABASE user=postgres"` with each command, or
> - Export `PGD_CLI_DSN="dbname=$APP_DATABASE user=postgres"` for the entire session.

To find the proxy pods, get a pod listing for your cluster:

```
kubectl get pods -n my-namespace

NAME                   READY   STATUS    RESTARTS      AGE
location-a-1-1         1/1     Running   0             2h
location-a-2-1         1/1     Running   0             2h
location-a-3-1         1/1     Running   0             2h
location-a-proxy-0     1/1     Running   0             2h
location-a-proxy-1     1/1     Running   0             2h
```

The proxy nodes have `proxy` in the name. Choose one, and get a command prompt in it:

```
kubectl exec -n my-namespace -ti location-a-proxy-0 -- bash
```

You now have a bash session open with the proxy pod. The `pgd` command is available:

```
pgd

Available Commands:
  check-health       Checks the health of the EDB Postgres Distributed cluster.
  <- snipped ->
  switchover         Switches over to new write leader.
  <- snipped ->
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

You can easily move your way through getting the information needed for the switchover:

```
pgd switchover --help

  $ pgd switchover --group-name group_a --node-name bdr-a1
  switchover is complete
```

```
pgd show-groups -f $PGD_CONFIG_FILE

Group      Group ID   Type   Parent Group Location Raft Routing Write Leader
-----      --------   ----   ------------ -------- ---- ------- ------------
world      3239291720 global                      true true    location-a-2
location-a 2135079751 data   world                 true true    location-a-1
```

```
pgd show-nodes -f $PGD_CONFIG_FILE
Node         Node ID    Group      Type Current State Target State Status Seq ID
----         -------    -----      ---- ------------- ------------ ------ ------
location-a-1 3165289849 location-a data ACTIVE        ACTIVE       Up     1
location-a-2 3266498453 location-a data ACTIVE        ACTIVE       Up     2
location-a-3 1403922770 location-a data ACTIVE        ACTIVE       Up     3
```

Accessing the database

In Use cases is a discussion on using the database within the Kubernetes cluster versus from outside. In Connectivity, you can find a discussion on services, which is relevant for accessing the database from applications.

However you implement your system, your applications must use the proxy service to connect to reap the benefits of PGD and of the increased self-healing capabilities added by the EDB Postgres Distributed for Kubernetes operator.

> **Important**
>
> As per the EDB Postgres for Kubernetes defaults, data nodes are created with a database called `app` and owned by a user named `app`, in contrast to the `bdrdb` database described in the EDB Postgres Distributed documentation. You can configure these values in the `cnp` section of the manifest. For reference, see Bootstrap in the EDB Postgres for Kubernetes documentation.

You might, however, want access to your PGD data nodes for administrative tasks, using the psql CLI.

You can get a pod listing for your PGD cluster and `kubectl exec` into a data node:

```
kubectl exec -n my-namespace -ti location-a-1-1 -- psql
```

In the familiar territory of psql, remember that the default created database is named `app` (see previous warning).

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
postgres=# \c app
You are now connected to database "app" as user "postgres".
app=# \x
Expanded display is on.
app=# select * from bdr.node_summary;
-[ RECORD 1 ]----------------------------------------
node_name              | location-a-1
node_group_name        | location-a
interface_connstr      | host=location-a-1-node user=streaming_replica sslmode=verify-ca port=5432
sslkey=/controller/certificates/streaming_replica.key
sslcert=/controller/certificates/streaming_replica.crt sslrootcert=/controller/certificates/server-ca.crt
application_name=location-a-1 dbname=app
peer_state_name        | ACTIVE
peer_target_state_name | ACTIVE

<- snipped ->
```

For your applications, use the non-privileged role (`app` by default).

You need the user credentials, which are stored in a Kubernetes secret:

```
kubectl get secrets

NAME                     TYPE                       DATA   AGE
<- snipped ->
location-a-app           kubernetes.io/basic-auth   2      2h
```

This secret contains the username and password needed for the Postgres DSN, encoded in base64:

```
kubectl get secrets location-a-app -o yaml

apiVersion: v1
data:
  password: <base64-encoded-password>
  username: <base64-encoded-username>
kind: Secret
metadata:
  creationTimestamp: <timestamp>
  labels:

<- snipped ->
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

12 Image catalog

A PGD group has its own `ImageCatalog` and `ClusterImageCatalog`. `ImageCatalog` and `ClusterImageCatalog` are essential resources that allow you to define PGD and PGD Proxy images for creating a `PGDGroup`.

The key distinction is in their scope: an `ImageCatalog` is namespaced, while a `ClusterImageCatalog` is cluster scoped.

Both share a common structure, comprising a list of PGD images and a Proxy image. Each PGD image contains a `major` field indicating the major Postgres version of the image.

> **Warning**
>
> The operator trusts the user-defined major version and doesn't conduct any PostgreSQL version detection. It's your responsibility to ensure alignment between the declared major version in the catalog and the PostgreSQL image.

The `major` field's value must remain unique in a catalog, preventing duplication across images. However, distinct catalogs can expose different images under the same `major` value.

Example of a namespaced `ImageCatalog`

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind: ImageCatalog
metadata:
  name: pgd-postgres-extended
  namespace: default
spec:
  images:
    - major: 15
      image: docker.enterprisedb.com/k8s/edb-postgres-extended-pgd:15.14-pgd590-ubi9
    - major: 16
      image: docker.enterprisedb.com/k8s/edb-postgres-extended-pgd:16.10-pgd590-ubi9
    - major: 17
      image:  docker.enterprisedb.com/k8s/edb-postgres-extended-pgd:17.6-pgd590-ubi9
  proxyImage: docker.enterprisedb.com/k8s/edb-pgd-proxy:5.9.0-ubi9
```

Example of a cluster-wide catalog using `ClusterImageCatalog` resource

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
ClusterImageCatalog
metadata:
  name: pgd-postgres-extended
spec:
  images:
    - major: 15
      image: docker.enterprisedb.com/k8s/edb-postgres-extended-pgd:15.14-pgd590-ubi9
    - major: 16
      image:  docker.enterprisedb.com/k8s/edb-postgres-extended-pgd:16.10-pgd590-ubi9
    - major: 17
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
      image:  docker.enterprisedb.com/k8s/edb-postgres-extended-pgd:17.6-pgd590-ubi9
  proxyImage: docker.enterprisedb.com/k8s/edb-pgd-proxy:5.9.0-ubi9
```

A `PGDGroup` resource has the flexibility to reference either an `ImageCatalog` or a `ClusterImageCatalog` to precisely specify the desired image:

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: group-
example
spec:
  [...]
  imageCatalogRef:
    apiGroup: pgd.k8s.enterprisedb.io
    kind: ImageCatalog
    name: pgd-postgres-extended
    major: 16
  [...]
```

PGD groups utilizing these catalogs maintain continuous monitoring. Any alterations to the images in a catalog trigger automatic updates for all associated PGD groups that reference that specific entry.

Image catalog with latest operand

The latest operand images for each PGD extension minor release are maintained in the PGD Operand images section. These images are the up-to-date images stored in the `k8s` repository. You can either reference the entire imageCatalog directly or use individual operand images as needed.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

13 Backup on object stores

EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) supports *online/hot backup* of PGD clusters through physical backup and WAL archiving on an object store. This means that the database is always up (no downtime required) and that point-in-time recovery (PITR) is available.

Common object stores

Multiple object stores are supported, such as AWS S3, Microsoft Azure Blob Storage, Google Cloud Storage, MinIO Gateway, or any S3-compatible provider. Given that CNPG-GC configures the connection with object stores by relying on EDB Postgres for Kubernetes, see the EDB Postgres for Kubernetes common object stores for backups documentation for more information.

> **Important**
>
> The EDB Postgres for Kubernetes documentation's Cloud Provider configuration section is available at `spec.backup.barmanObjectStore`. In CNPG-GC examples, the object store section is at a different path: `spec.backup.configuration.barmanObjectStore`.

WAL archive

WAL archiving is the process that sends WAL files to the object storage, and it's essential to execute online/hot backups or PITR. In CNPG-GC, each PGD node is set up to archive WAL files in the object store independently.

The WAL archive is defined in the PGD Group `spec.backup.configuration.barmanObjectStore` stanza, and is enabled as soon as a destination path and cloud credentials are set. You can choose to compress WAL files before they're uploaded and you can encrypt them. You can also enable parallel WAL archiving:

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
[...]
spec:
  backup:
    configuration:
      barmanObjectStore:
        [...]
        wal:
          compression: gzip
          encryption:
AES256
          maxParallel: 8
```

For more information, see the EDB Postgres for Kubernetes WAL archiving documentation.

Scheduled backups

Scheduled backups are the recommended way to configure your backup strategy in CNPG-GC. When the PGD group `spec.backup.configuration.barmanObjectStore` and `.spec.backup.schedulers[].schedule` stanzas are configured, the operator selects one of the PGD data nodes as the elected backup node in which it creates a `Scheduled Backup` resource.

The `.spec.backup.schedulers[].method` field allows you to define the scheduled backup method. Two backup methods are supported:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- `volumeSnapshot`
- `barmanObjectStore` (the default)

You can define more than one scheduler, but each method can be used by only one scheduler. That is, two schedulers aren't allowed to use the same method.

For object store backups, with the default `barmanObjectStore` method, use the stanza `spec.backup.configuration.barmanObjectStore` to define the object store information for both backup and WAL archiving. For more information, see Backup on object stores in the EDB Postgres for Kubernetes documentation.

To perform volumeSnapshot backups, you can select the `volumeSnapshot` method. Use the stanza `spec.backup.configuration.barmanObjectStore.volumeSnapshot` to define the volumeSnapshot configuration. For more information, see Backup on volume snapshots in the EDB Postgres for Kubernetes documentation.

This example shows how to use the `volumeSnapshot` method for backup. WAL archiving is still done onto the Barman object store.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
[...]
spec:
  backup:
    configuration:
      volumeSnapshot:
        className: csi-hostpath-snapclass
      barmanObjectStore:
        destinationPath: "<destination path
here>"
        s3Credentials:
          accessKeyId:
            name: backup-storage-creds
            key: ID
          secretAccessKey:
            name: backup-storage-creds
            key:
KEY
        wal:
          compression: gzip
          encryption:
AES256
          maxParallel: 8
    schedulers:
    - method:
volumeSnapshot
      schedule: "0 0 0 * *
*"
      backupOwnerReference: self
      suspend: false
      immediate: true
```

For a comparison of these two backup methods, see Object stores or volume snapshots in the EDB Postgres for Kubernetes documentation.

The `.spec.backup.schedulers[].schedule` field allows you to define a cron schedule, expressed in theGo `cron` package format:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
[...]
spec:
  backup:
    schedulers:
    - method: barmanObjectStore
      schedule: "0 0 0 * *
*"
      backupOwnerReference: self
      suspend: false
      immediate: true
```

If necessary, you can suspend scheduled backups by setting `.spec.backup.schedulers[].suspend` to `true`. This setting prevents new backups from being scheduled.

If you want to execute a backup as soon as the `ScheduledBackup` resource is created, set `.spec.backup.schedulers[].immediate` to `true`.

`.spec.backup.schedulers[].backupOwnerReference` indicates the `ownerReference` to use in the created backup resources. The options are:

- **none** — Doesn't set an owner reference for created backup objects.
- **self** — Sets the `ScheduledBackup` object as owner of the backup.
- **cluster** — Sets the cluster as owner of the backup.

> **Warning**
>
> The `.spec.backup.cron` field is deprecated. Use `.spec.backup.schedulers` instead. While you can still use `.spec.backup.cron`, you can't use it at the same time as `.spec.backup.schedulers`.

> **Note**
>
> The EDB Postgres for Kubernetes `ScheduledBackup` object contains the `cluster` option to specify the cluster to back up. This option currently isn't supported by CNPG-GC and is ignored if specified.

If an elected backup node is deleted, the operator transparently elects a new backup node and reconciles the `ScheduledBackup` resource accordingly.

Retention policies

CNPG-GC can manage the automated deletion of backup files from the backup object store using retention policies based on the recovery window. This process also takes care of removing unused WAL files and WALs associated with backups that are scheduled for deletion.

You can define your backups with a retention policy of 30 days:

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
[...]
spec:
  backup:
    configuration:
      retentionPolicy: "30d"
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

For more information, see the EDB Postgres for Kubernetes retention policies in the EDB Postgres for Kubernetes documentation.

> **Important**
>
> Currently, the retention policy is applied only for the elected `Backup Node` backups and WAL files. Given that each other PGD node also archives its own WALs independently, it's your responsibility to manage the lifecycle of those WAL files, for example by leveraging the object storage data retention policy. Also, if you have an object storage data retention policy set up on every PGD node directory, make sure it's not overlapping or interfering with the retention policy managed by the operator.

Compression algorithms

Backups and WAL files are uncompressed by default. However, multiple compression algorithms are supported. For more information, see the EDB Postgres for Kubernetes compression algorithms documentation.

Tagging of backup objects

It's possible to specify tags as key-value pairs for the backup objects, namely base backups, WAL files, and history files. For more information, see the EDB Postgres for Kubernetes documentation about tagging of backup objects.

On-demand backups of a PGD node

A PGD node is represented as a single-instance EDB Postgres for Kubernetes `Cluster` object. As such, if you need to, it's possible to request an on-demand backup of a specific PGD node by creating an EDB Postgres for Kubernetes `Backup` resource. To do that, see EDB Postgres for Kubernetes on-demand backups in the EDB Postgres for Kubernetes documentation.

> **Hint**
>
> You can retrieve the list of EDB Postgres for Kubernetes clusters that make up your PGD group by running `kubectl get cluster -l k8s.pgd.enterprisedb.io/group=my-pgd-group -n my-namespace`.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

14 Recovery

In EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC), recovery is available as a way to bootstrap a new PGD group starting from an available physical backup of a PGD node. Recovery can't be performed in place on an existing PGD group.

CNPG-GC also supports point-in-time recovery (PITR), which allows you to restore a PGD group up to any point in time, from the first available backup in your catalog to the last archived WAL. Having a WAL archive is mandatory for PITR.

Prerequisites

Before recovering from a backup:

- Make sure that the PostgreSQL configuration (`.spec.cnp.postgresql.parameters`) of the recovered cluster is compatible with the original one from a physical replication standpoint.
- When recovering in a newly created namespace, first set up a cert-manager CA issuer before deploying the recovered PGD group.

For more information, see EDB Postgres for Kubernetes recovery - Additional considerations in the EDB Postgres for Kubernetes documentation.

Recovery from an object store

You can recover from a PGD node backup created by Barman Cloud and stored on supported object storage.

For example, given a PGD group named `pgdgroup-example` with three instances with backups available, your object storage will contain a directory for each node:

`pgdgroup-example-1` , `pgdgroup-example-2` , `pgdgroup-example-3`

This example defines a full recovery from the object store. The operator transparently selects the latest backup among the defined `serverNames` and replays up to the last available WAL.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: pgdgroup-restore
spec:
  [...]
  restore:
    serverNames:
      - pgdgroup-backup-1
      - pgdgroup-backup-2
      - pgdgroup-backup-3
    barmanObjectStore:
      destinationPath: "<destination path
here>"
      s3Credentials:
        accessKeyId:
          name: backup-storage-creds
          key: ID
        secretAccessKey:
          name: backup-storage-creds
          key:
KEY
      wal:
        compression: gzip
        encryption:
AES256
        maxParallel: 8
```

> **Important**
>
> Make sure to correctly configure the WAL section according to the source cluster. In the example, since the `pgdgroup-example` PGD group uses `compression` and `encryption`, make sure to set the proper parameters also in the PGD group that's being created by the `restore`.

> **Note**
>
> The example takes advantage of the parallel WAL restore feature, dedicating up to eight jobs to concurrently fetch the required WAL files from the archive. This feature can appreciably reduce the recovery time. Make sure that you plan ahead for this scenario and tune the value of this parameter for your environment. It makes a difference when you need it.

PITR from an object store

Instead of replaying all the WALs up to the latest one, after extracting a base backup, you can ask PostgreSQL to stop replaying WALs at any point in time. PostgreSQL uses this technique to achieve PITR. (The presence of a WAL archive is mandatory.)

This example defines a time-based target for the recovery:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: pgdgroup-restore
spec:
  [...]
  restore:
    recoveryTarget:
      targetTime: "2023-08-11 11:14:21.00000+02"
    serverNames:
      - pgdgroup-backup-1
      - pgdgroup-backup-2
      - pgdgroup-backup-3
    barmanObjectStore:
      destinationPath: "<destination path
here>"
      s3Credentials:
        accessKeyId:
          name: backup-storage-creds
          key: ID
        secretAccessKey:
          name: backup-storage-creds
          key:
KEY
      wal:
        compression: gzip
        encryption:
AES256
        maxParallel: 8
```

> **Important**
>
> PITR requires you to specify a `targetTime` recovery target by using the options described in Recovery targets. When you use `targetTime` or `targetLSN`, the operator selects the closest backup that was completed before that target. Otherwise, it selects the last available backup in chronological order between the specified `serverNames`.

Recovery from an object store specifying a `backupID`

The `.spec.restore.recoveryTarget.backupID` option allows you to specify a base backup from which to start the recovery process. By default, this value is empty. If you assign a value to it, the operator uses that backup as the base for the recovery. The value must be in the form of a Barman backup ID.

This example recovers a new PGD group from a specific backupID of the `pgdgroup-backup-1` PGD node:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: pgdgroup-restore
spec:
  [...]
  restore:
    recoveryTarget:
      backupID: 20230824T133000
    serverNames:
      - pgdgroup-backup-1
    barmanObjectStore:
      destinationPath: "<destination path
here>"
      s3Credentials:
        accessKeyId:
          name: backup-storage-creds
          key: ID
        secretAccessKey:
          name: backup-storage-creds
          key:
KEY
      wal:
        compression: gzip
        encryption:
AES256
        maxParallel: 8
```

> **Important**
>
> When you specify a `backupID`, make sure to list only the related PGD node in the `serverNames` option, and avoid listing the other ones.

> **Note**
>
> Defining a specific `backupID` is especially needed when using one of the following recovery targets: `targetName`, `targetXID`, and `targetImmediate`. In such cases, it's important to specify `backupID`, unless the last available backup in the catalog is okay.

Recovery from volumeSnapshot

You can also recover a PGDGroup from a volumeSnapshot backup. The stanza `spec.restore.volumeSnapshots` is used to define the criteria for volumeSnapshots restore candidates. The operator transparently selects the latest volumeSnapshot among the candidates.

The operator requires the following annotations/labels in the volumeSnapshot. These annotations/labels are automatically added if volumeSnapshots are taken by the operator.

Annotations:

- `k8s.enterprisedb.io/backupEndTime` is used to compare and select the latest snapshot.
- `k8s.enterprisedb.io/pvcRole` represents the pvcRole of the volumeSnapshot. Supported roles include PG_WAL and PG_DATA.

Labels:

- `k8s.enterprisedb.io/cluster` indicates the node where the volumeSnapshot is taken, crucial for fetching the serverName in the object store for WAL replaying.
- `k8s.enterprisedb.io/backupName` is the backup name of the volumeSnapshot. Used to group volumeSnapshots when more volumes are defined in the backup.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- `k8s.enterprisedb.io/tablespaceName` represents the tablespace name of the volumeSnapshot when the volumeSnapshot role is `PG_TABLESPACE`.

This example shows a full recovery from volumeSnapshots. After the volumeSnapshot recovery, WAL replaying for full recovery will target server `pgdgroup-backup-vs-1`.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: pgdgroup-restore
spec:
  [...]
  restore:
    volumeSnapshots:
      selector:
        matchLabels:
          "k8s.enterprisedb.io/cluster": pgdgroup-backup-vs-1
          "k8s.pgd.enterprisedb.io/group": pgdgroup-backup-vs
    barmanObjectStore:
      destinationPath: "<destination path
here>"
      s3Credentials:
        accessKeyId:
          name: backup-storage-creds
          key: ID
        secretAccessKey:
          name: backup-storage-creds
          key:
KEY
      wal:
        compression: gzip
        encryption:
AES256
        maxParallel: 8
```

For more information, see Recovery from volumeSnapshot objects in the EDB Postgres for Kubernetes documentation.

PITR from volumeSnapshot

You can instruct PostgreSQL to halt the replay of write-ahead logs (WALs) at any specific moment during volumeSnapshot recovery. This is the same capability as when recovering from an object store.

This example shows setting a time-based target for recovery using volume snapshots:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: pgdgroup-restore
spec:
  [...]
  restore:
    recoveryTarget:
      targetTime: "2023-08-11 11:14:21.00000+02"
    volumeSnapshots:
      selector:
        matchLabels:
          "k8s.enterprisedb.io/cluster": pgdgroup-backup-vs-1
          "k8s.pgd.enterprisedb.io/group": pgdgroup-backup-vs
    barmanObjectStore:
      destinationPath: "<destination path
here>"
      s3Credentials:
        accessKeyId:
          name: backup-storage-creds
          key: ID
        secretAccessKey:
          name: backup-storage-creds
          key:
KEY
      wal:
        compression: gzip
        encryption:
AES256
        maxParallel: 8
```

Recovery targets

Beyond PITR are other recovery target criteria you can use. For more information on all the available recovery targets, seeEDB Postgres for Kubernetes recovery targets in the EDB Postgres for Kubernetes documentation.

Recovering and creating PGD Groups in multiple regions

To recover from a backup and create multiple PGD Groups joined across regions, you can only restore the first group (where `spec.pgd.parentGroup.create` is set to true) from the backup. The other groups need to be set up from scratch, and then joined to the first group. This follows recommendations when using `bdr.join_node_group`:

> We recommend that the newly joining database be empty except for the BDR extension.

Refer to the PGD documentation for more detail.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

15 Security

Security for EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) is analyzed at three layers: code, container, and cluster.

> **Warning**
>
> In addition to security practices described here, you must perform regular InfoSec duties on your Kubernetes cluster. Familiarize yourself with Overview of Cloud Native Security in the Kubernetes documentation.

> **About the 4C's Security Model**
>
> See The 4C's Security Model in Kubernetes blog article for a better understanding and context of the approach EDB takes with security in CNPG-GC.

Code

CNPG-GC' source code undergoes systematic static analysis, including checks for security vulnerabilities, using the popular open-source linter for Go, GolangCI-Lint, directly integrated into the CI/CD pipeline. GolangCI-Lint can run multiple linters on the same source code.

The following tools are used to identify security issues:

- **Golang Security Checker (`gosec`):** A linter that scans the abstract syntax tree of the source code against a set of rules designed to detect known vulnerabilities, threats, and weaknesses, such as hard-coded credentials, integer overflows, and SQL injections. GolangCI-Lint runs `gosec` as part of its suite.
- **govulncheck:** This tool runs in the CI/CD pipeline and reports known vulnerabilities affecting Go code or the compiler. If the operator is built with a version of the Go compiler containing a known vulnerability, `govulncheck` will detect it.
- **Snyk:** Runs as part of the CI workflow to scan for code, dependency, and container security and licensing issues.

> **Important**
>
> A failure in the static code analysis phase of the CI/CD pipeline is a blocker for the entire delivery of CNPG-GC, meaning that each commit is validated against all the linters defined by GolangCI-Lint.

Container

Every container image that's part of CNPG-GC is built by way of CI/CD pipelines following every commit. Such images include not only those of the operator but also of the operands, specifically every supported PostgreSQL version. In the pipelines, images are scanned with:

- Dockle: for best practices in terms of the container build process
- Black Duck: Checks for vulnerabilities in Open Source components and validates license consistency.

> **Important**
>
> All operand images are rebuilt weekly by our pipelines to incorporate security updates at the base image and package level, providing patch level updates for the container images that EDB distributes.

> **Warning**
>
> Running outdated images can expose your environment to security risks and performance issues. We highly recommend that you update to the latest image version and keep your images up to date. This will ensure that you take advantage of the latest updates and patches available.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

The following guidelines and frameworks were taken into account for container-level security:

- The Container Image Creation and Deployment Guide, developed by the Defense Information Systems Agency (DISA) of the United States Department of Defense (DoD)
- The CIS Benchmark for Docker, developed by the Center for Internet Security (CIS)

> **About the container-level security**
>
> See the Security and Containers in CNPG-GC blog article for more information about the approach that EDB takes on security at the container level in CNPG-GC.

Cluster

Security at the cluster level takes into account all Kubernetes components that form both the control plane and the nodes as well as the applications that run in the cluster, including PostgreSQL.

Role-based access control (RBAC)

The operator interacts with the Kubernetes API server with a dedicated service account called pgd-operator-controller-manager. In Kubernetes this account is installed by default in the `pgd-operator-system` namespace. A cluster role binds between this service account and the pgd-operator-controller-manager cluster role that defines the set of rules, resources, and verbs granted to the operator.

RedHat OpenShift directly manages the operator RBAC entities by way of Operator Lifecycle Manager (OLM). OLM allows you to grant permissions only where they're required, implementing the principle of least privilege.

> **Important**
>
> These permissions are exclusively reserved for the operator's service account to interact with the Kubernetes API server. They aren't directly accessible by the users of the operator that interact only with `PGDGroup` and `PGDGroupCleanup` resources.

The following are some examples and, most importantly, the reasons why CNPG-GC requires full or partial management of standard Kubernetes namespaced resources.

- `jobs`

  The operator needs to handle jobs to manage different `PGDGroup` phases.

- `poddisruptionbudgets`

  The operator uses pod disruption budgets to make sure enough PGD nodes are kept active during maintenance operations.

- `pods`

  The operator needs to manage PGD nodes as a `Cluster` resource.

- `secrets`

  Unless you provide certificates and passwords to your data nodes, the operator adopts the "convention over configuration" paradigm by self-provisioning random-generated passwords and TLS certificates and by storing them in secrets.

- `serviceaccounts`

  The operator needs to create a service account to enable the `PGDGroup` recovery job to retrieve the backup objects from the object store where they reside.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- `services`

  The operator needs to control network access to the PGD cluster from applications and properly manage failover/switchover operations in an automated way.

- `statefulsets`

  The operator needs to manage PGD proxies.

- `validatingwebhookconfigurations` **and** `mutatingwebhookconfigurations`

  The operator injects its self-signed webhook CA into both webhook configurations, which are needed to validate and mutate all the resources it manages. For more details, see the Kubernetes documentation.

To see all the permissions required by the operator, you can run `kubectl describe clusterrole pgd-operator-manager-role`.

CNPG-GC internally manages the PGD nodes using the `Cluster` resource as defined by EDB Postgres for Kubernetes. See the EDB Postgres for Kubernetes documentation for the list of permissions used by the EDB Postgres for Kubernetes operator service account.

Calls to the API server made by the instance manager

The instance manager, which is the entry point of the operand container, needs to make some calls to the Kubernetes API server to ensure that the status of some resources is correctly updated and to access the config maps and secrets that are associated with that Postgres cluster. Such calls are performed through a dedicated `ServiceAccount` created by the operator that shares the same PostgreSQL `Cluster` resource name.

> **Important**
>
> The operand can access only a specific and limited subset of resources through the API server. A service account is the recommended way to access the API server from within a pod. See the Kubernetes documentation for details.

See the EDB Postgres for Kubernetes documentation for more information on the instance manager.

Pod security policies

A pod security policy is the Kubernetes way to define security rules and specifications that a pod needs to meet to run in a cluster. For InfoSec reasons, every Kubernetes platform must implement them.

CNPG-GC doesn't require privileged mode for containers execution. The PostgreSQL containers run as the postgres system user. No component requires running as root.

Likewise, volumes access doesn't require privileged mode or root privileges. Proper permissions must be assigned by the Kubernetes platform or administrators. The PostgreSQL containers run with a read-only root filesystem, that is, no writable layer.

The operator explicitly sets the required security contexts.

On Red Hat OpenShift, CNPG-GC runs in the `restricted` security context constraint, the most restrictive one. The goal is to limit the execution of a pod to a namespace allocated UID and SELinux context.

> **Security Context Constraints in OpenShift**
>
> For more information on security context constraints (SCC) in OpenShift, see the Managing SCC in OpenShift article.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

> **Security context constraints and namespaces**
>
> As stated in the Openshift documentation, SCCs aren't applied in the default namespaces (`default` , `kube-system` , `kube-public` , `openshift-node` , `openshift-infra` , `openshift`). Don't use them to run pods. PGD clusters deployed in those namespaces will be unable to start due to missing SCCs.

Exposed ports

CNPG-GC exposes ports at operator, instance manager, and operand levels, as shown in the table.

System	Port number	Exposing	Name	Certificates	Authentication
operator	9443	webhook server	`webhook-server`	TLS	Yes
operator	8080	metrics	`metrics`	no TLS	No
instance manager	9187	metrics	`metrics`	no TLS	No
instance manager	8000	status	`status`	no TLS	No
operand	5432	PostgreSQL instance	`postgresql`	optional TLS	Yes

PGD

The current implementation of CNPG-GC creates passwords for the postgres superuser and the database owner.

As far as encryption of passwords is concerned, CNPG-GC follows the default behavior of PostgreSQL: starting with PostgreSQL 14, `password_encryption` is by default set to `scram-sha-256` . On earlier versions, it's set to `md5` .

> **Important**
>
> See Connection DSNs and SSL in the PGD documentation for details.

CNPG-GC uses the `postgres` role to maintain PGD nodes. Due to this, it is not allowed to disable `postgres` user access by setting `enableSuperuserAccess` to `false` in the `cnp` section of the spec.

Storage

CNPG-GC delegates encryption at rest to the underlying storage class. For data protection in production environments, we highly recommend that you choose a storage class that supports encryption at rest.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

16 Connectivity

Information about secure network communications in a PGD cluster includes:

- Services
- Domain names resolution using fully qualified domain names (FQDN)
- TLS configuration
- Connecting from an application

> **Notice**
>
> Although these topics might seem unrelated to each other, they all participate in the configuration of the PGD resources to make them universally identifiable and accessible over a secure network.

Services

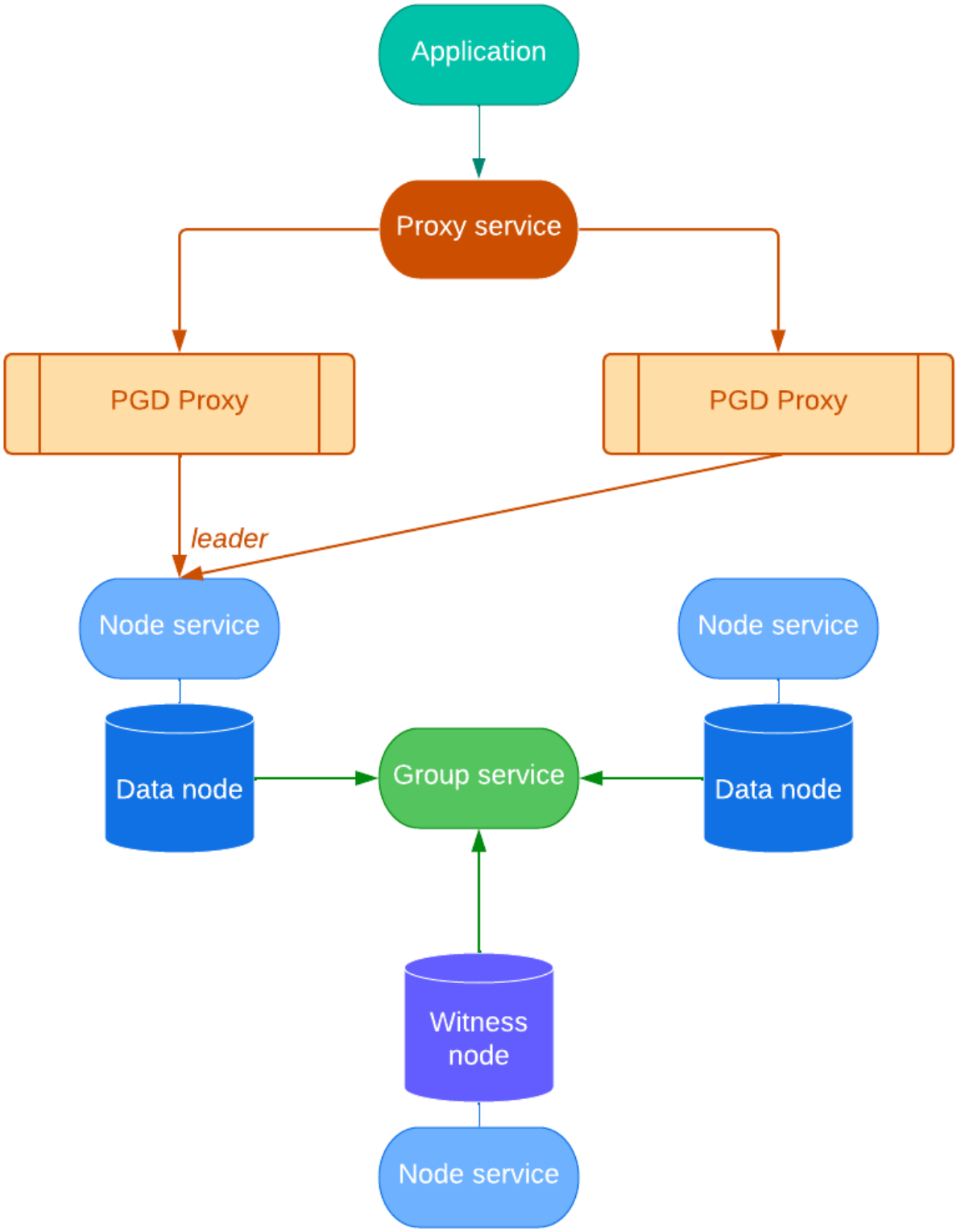
Resources in a PGD cluster are accessible through Kubernetes services. Every PGD group manages several of them, namely:

- One service per node, used for internal communications (*node service*).
- A *group service* to reach any node in the group, used primarily by EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) to discover a new group in the cluster.
- A *proxy service* to enable applications to reach the write leader of the group transparently using PGD Proxy.
- A *proxy-r service* to enable applications to reach the *read nodes* of the group, transparently using PGD Proxy. This service is disabled by default and controlled by the `.spec.proxySettings.enableReadNodeRouting` setting.

> **Note**
>
> The term *read nodes* refers to nodes that are not designated as the write leader. In the PGD group, each node is capable of handling writes. The Write leader is the elected node designated to process writes through the `proxy service`. Meanwhile, *read nodes* reached by the `proxy-r service` are also writable, but it is the user's responsibility to ensure that write SQL commands are not sent to the `proxy-r service`. For more details, please refer to Read-only routing with PGD Proxy

For an example that uses these services, see Connecting an application to a PGD cluster.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.



Each service is generated from a customizable template in the `.spec.connectivity` section of the manifest.

All services must be reachable using their FQDN from all the PGD nodes in all the Kubernetes clusters. SeeDomain names resolution.

CNPG-GC provides a service templating framework that gives you the availability to easily customize services at the following levels:

- **Node Service Template** Each PGD node is reachable using a service that can be configured in the `.spec.connectivity.nodeServiceTemplate` section. You can leverage the variables `{node_svc}` and `{node_fqdn}` to substitute for the node service name and the node service's FQDN.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- **Group Service Template** Each PGD group has a group service that's a single entry point for the whole group and that can be configured in the `.spec.connectivity.groupServiceTemplate` section. You can leverage the variables `{group_svc}` and `{group_fqdn}` to substitute for the group service name and the group service's FQDN.

- **Proxy Service Template** Each PGD group has a proxy service to reach the group write leader through the PGD proxy. It can be configured in the `.spec.connectivity.proxyServiceTemplate` section. This is the entry-point service for applications. You can leverage the variables `{proxy_svc}` and `{proxy_fqdn}` to substitute for the proxy service name and the proxy service's FQDN.

- **Proxy Read Service Template** Each PGD group has a proxy service to reach the group read nodes through the PGD proxy. It can be enabled with `.spec.proxySettings.enableReadNodeRouting`, and can be configured in the `.spec.connectivity.proxyReadServiceTemplate` section. This is the entry-point service for applications. You can leverage the variables `{proxy_svc}` and `{proxy_fqdn}` to substitute for the proxy service name and the proxy service's FQDN.

You can use templates to create a LoadBalancer service or to add arbitrary annotations and labels to a service to integrate with other components available in the Kubernetes system (that is, to create external DNS names or tweak the generated load balancer).

Here is an example of a proxy service template, which creates a LoadBalancer proxy service. Once the service is created, `{proxy_svc}` will be replaced with the proxy service name, and `{proxy_fqdn}` will be replaced with the proxy service's FQDN.

```
spec:
  connectivity:

...
    proxyServiceTemplate:
      metadata:
        annotations:
          proxy_service_name: '{proxy_svc}'
          proxy_service_fqdn: '{proxy_fqdn}'
      spec:
        loadBalancerSourceRanges:
        - 0.0.0.0/0
        ports:
        - name:
postgres
          port: 5432
          protocol:
TCP
          targetPort: 5432
        type: LoadBalancer
      updateStrategy: patch
```

Domain names resolution

CNPG-GC ensures that all resources in a PGD group have a FQDN, and by convention uses the PGD group name as a prefix for all of them.

As a result, it expects you to define the domain name of the PGD group. This can be done through the `.spec.connectivity.dns` section, which controls how the FQDNs for the resources are generated with two fields:

- `domain` — Domain name for all the objects in the PGD group to use (mandatory).
- `hostSuffix` — Suffix to add to each service in the PGD group (optional).

You can also define a list of additional `domain` and `hostSuffix` in `.spec.connectivity.dns.additional`. Make sure that these additional `domain` and `hostSuffix` entries match with the rendered nodeServiceTemplate and groupServiceTemplate. Node Services and Group Services with additional `domain` and `hostSuffix` entries will be included in the Subject Alternative Name of the service certificate.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

TLS configuration

CNPG-GC requires that resources in a PGD cluster communicate over a secure connection. It relies on PostgreSQL's native support for SSL connections to encrypt client/server communications using TLS protocols for increased security.

Currently, CNPG-GC requires that cert-manager is installed. Cert-manager was chosen as the tool to provision dynamic certificates given that it's widely recognized as the standard in a Kubernetes environment.

The `spec.connectivity.tls` section describes how the communication between the nodes happens. Please refer to the certificates section for more details on how TLS is configured.

Connecting from an application

Connecting to a PGD Group from an application running inside the same Kubernetes cluster or from outside the cluster is a simple procedure. In both cases, you connect to the proxy service of the PGD Group as the `app` user. The proxy service is a LoadBalancer service that routes the connection to the write leader or read nodes of the PGD Group, depending on which proxy service it's connecting to.

Connecting from inside the cluster

When connecting from inside the cluster, you can use the proxy service name to connect to the PGD group. The proxy service name is composed of the PGD group name plus `-proxy`, and the optional host suffix defined in the `.spec.connectivity.dns` section of the `PGDGroup` custom resource.

For example, if the PGD group name is `my-group`, and the host suffix is `.my-domain.com`, the proxy service name is `my-group-proxy.my-domain.com`.

Before connecting, you need to get the password for the app user from the app user secret. The app user name is defined by `spec.pgd.ownerName`, and the app user secret is defined by `spec.pgd.ownerCredentialsSecret`.

You can get the username and password from the secret using the following commands:

```
kubectl get secret <app user secret> -o jsonpath='{.data.username}' | base64 --
decode
kubectl get secret <app user secret> -o jsonpath='{.data.password}' | base64 --
decode
```

With this, you have all the pieces for a connection string to the PGD group:

```
postgresql://<app-user>:<app-password>@<proxy-service-name>:5432/<database>
```

Or, for a `psql` invocation:

```
psql -U <app-user> -h <proxy-service-name>
<database>
```

Where `app-user` and `app-password` are the values you got from the secret, and `database` is the name of the database you want to connect to, defined by `spec.pgd.databaseName`. (The default is `app` for the app user.)

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Connecting from outside the Kubernetes cluster

When connecting from outside the Kubernetes cluster, in the general case, the Ingress resource or a load balancer is necessary. Check your cloud provider or local installation for more information about their behavior in your environment.

Ingresses and load balancers require a pod selector to forward connection to the PGD proxies. When configuring them, we suggest using the following labels:

- `k8s.pgd.enterprisedb.io/group` — Set the PGD group name.
- `k8s.pgd.enterprisedb.io/workloadType` — Set to `pgd-proxy`.

If using Kind or other solutions for local development, the easiest way to access the PGD group from outside is to use port forwarding to the proxy service. You can use the following command to forward port 5432 on your local machine to the proxy service:

```
kubectl port-forward svc/my-group-proxy.my-domain.com 5432:5432
```

Where `my-group-proxy.my-domain.com` is the proxy service name from the previous example.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

17 Certificates

EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) was designed to natively support TLS certificates. To set up a PGD cluster, each PGD node requires:

- Server certificates configuration.
- Client certificates configuration.

> **Note**
>
> You can find all the secrets used by each PGD node and the expiry dates in the cluster (PGD node) status.

Server certificates configuration

The server certificate configuration is handled in the `spec.connectivity.tls.serverCert` section of the PGDGroup custom resource. This configuration requires a server CA secret and a cert-manager template to generate the TLS certificates needed for the underlying Postgres instance to terminate TLS connections.

The following assumptions must be met for this section to function correctly:

1. Cert-manager must be installed.
2. An issuer specified in `spec.connectivity.tls.serverCert.certManager.issuerRef` is available for the domain specified in `spec.connectivity.dns.domain` and any additional domains listed in `spec.connectivity.tls.serverCert.certManager.spec.dnsNames`.
3. A server CA secret containing the public certificate of the CA used by the issuer must be created.

> **Note**
>
> The server CA secret specified by `spec.connectivity.tls.serverCert.caCertSecret` will be referenced as `serverCASecret` in the underlying CNP nodes. The public part, `ca.crt`, validates the server certificate and is included as `sslrootcert` into client connection strings. The private part, `ca.key`, is used to automatically sign the server SSL certificate, if a self-signed certificate is employed.

> **Note**
>
> The server TLS secret generated by PGD group will be specified as the value of `serverTLSSecret` in the underlying CNP nodes. For more information, refer to server certificates.

DNS names

The operator will add the following `altDnsNames` to the server TLS certificate:

```
${nodeName}${hostSuffix}.${domain}
${groupName}${hostSuffix}.${domain}
${proxyName}${hostSuffix}.${domain}
${nodeName}${additionalHostSuffix}.${additionalDomain}
${groupName}${additionalHostSuffix}.${additionalDomain}
${proxyName}${additionalHostSuffix}.${additionalDomain}
```

Users are responsible for including any necessary names in `spec.connectivity.tls.serverCert.certManager.spec.dnsNames`, based on their underlying networking architecture (e.g. any load balancers to access the nodes).

For example, consider a PGD Group configured as follows:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: region-a
spec:
  instances: 3

...
  connectivity:
    dns:
      # we need to configure the domain for the group so it could be
resolved
      domain: enterprisedb.network
      additional:
        - domain: alternate.network
          hostSuffix: -
dc1
```

All of the following Subject Alternative Names (SANs) will be added to the server TLS certificate:

```
region-a-1-node.domain.enterprisedb.network
region-a-group.domain.enterprisedb.network
region-a-proxy.domain.enterprisedb.network
region-a-1-node-dc1.alternate.network
region-a-group-dc1.alternate.network
region-a-proxy-dc1.alternate.network
```

Client certificates configuration

The client certificates configuration is managed under the `spec.connectivity.tls.clientCert` section of the PGDGroup custom resource. This configuration requires a client CA secret and a client cert-manager template to generate the client streaming replication certificate for the `streaming_replica` Postgres user.

The following assumptions must be met for this section to function properly:

1. Cert-manager must be installed.
2. An issuer specified in `spec.connectivity.tls.clientCert.certManager.issuerRef` is available; this issuer will be used to sign a certificate with the common name `streaming_replica` .
3. A client CA secret must be present. It contains the public certificate of the CA used by the issuer.

The operator will use the configuration under `spec.connectivity.tls.clientCert.certManager` to create a certificate request for the `streaming_replica` Postgres user. The resulting certificate will be used to secure communication between the nodes.

> **Note**
>
> The client CA secret specified by `spec.connectivity.tls.clientCert.clientCA` will be used as the value of `clientCASecret` in the underlying CNP nodes. The public part, `ca.crt` , will be provided as `ssl_ca_file` to all the instances, allowing them to verify client certificates they have signed. The private part, `ca.key` , is optional and can be used to sign client certificate generated by the kubectl cnp plugin.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Client pre-provisioned replication certificate

Alternatively, you can specify a secret containing the pre-provisioned client certificate for the `streaming_replica` user using the `spec.connectivity.tls.clientCert.preProvisioned.streamingReplica.secretRef` option. In this case, the certificate lifecycle is managed entirely by a third party (manually or automatically), by simply updating the content of the secret.

> **Note**
>
> Regardless of how the client streaming replication certificate is provided, it will be used as the value of `replicationTLSSecret` in the underlying CNP nodes. For more information, refer to the section on Client certificate.

TLS mode

You can configure the TLS mode, which determines how the server certificates are verified during communication between nodes, using `spec.connectivity.tls.mode`. Its default value is `verify-ca`. Note that the TLS mode cannot be changed once the PGD Group is set up. The `mode` accepts the following values, as documented in SSL Support in the PostgreSQL documentation:

- `verify-full`
- `verify-ca`
- `required`

Self-signed example

This example demonstrates how to use cert-manager to set up a self-signed CA and generate required certificates.

First, we need to generate the server and client CA certificates. We will create two self-signed issuers, `server-ca-issuer` and `client-ca-issuer`, in the target namespace.

```
apiVersion: cert-manager.io/v1
kind:
Issuer
metadata:
  name: selfsigned-
issuer
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind:
Issuer
metadata:
  name: server-ca-issuer
spec:
  ca:
    secretName: server-ca-key-pair
---
apiVersion: cert-manager.io/v1
kind:
Issuer
metadata:
  name: client-ca-issuer
spec:
  ca:
    secretName: client-ca-key-pair
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

With the following Certificate object, we can generate the private key and a signed certificate from the issuer. The private key and signed certificate will be stored in the secrets named `server-ca-key-pair` and `client-ca-key-pair`.

```
apiVersion: cert-manager.io/v1
kind:
Certificate
metadata:
  name: server-ca
spec:
  isCA: true
  commonName: my-selfsigned-server-
ca
  secretName: server-ca-key-pair
  privateKey:
    algorithm: ECDSA
    size: 256
  issuerRef:
    name: selfsigned-
issuer
    kind:
Issuer
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1
kind:
Certificate
metadata:
  name: client-ca
spec:
  isCA: true
  commonName: my-selfsigned-client-
ca
  secretName: client-ca-key-pair
  privateKey:
    algorithm: ECDSA
    size: 256
  issuerRef:
    name: selfsigned-
issuer
    kind:
Issuer
    group: cert-manager.io
```

We can now configure the PGD group. The `server-ca-key-pair` and `client-ca-key-pair` will be used as the server CA secret and client CA secret, respectively. The cert-manager template can be set up to reference the corresponding issuer to automatically generate the server TLS certificate and client replication certificate.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: region-a
spec:

...
  connectivity:
    tls:
      mode: verify-ca
      clientCert:
        caCertSecret: client-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: client-ca-issuer
              kind:
Issuer
              group: cert-manager.io
      serverCert:
        caCertSecret: server-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: server-ca-issuer
              kind:
Issuer
              group: cert-manager.io
```

For more information about how certificates work, see theEDB Postgres for Kubernetes documentation.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

18 Client TLS/SSL connections

> **Certificates**
>
> See Certificates for more details on how EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) supports TLS certificates.

The CNPG-GC operator was designed to work with TLS/SSL for both encryption in transit and authentication on the server and client sides. PGD nodes are created as cluster resources using the EDB Postgres for Kubernetes operator. This includes deploying a certificate authority (CA) to create and sign TLS client certificates.

See the EDB Postgres for Kubernetes documentation for more information on issuers and certificates.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

19 Declarative pausing and resuming

The *declarative pausing and resuming* feature enables saving CPU power by removing the database pods while keeping the database PVCs.

Declarative pausing and resuming leverages the hibernation functionality available for EDB Postgres for Kubernetes. For additional depth and an explanation of how hibernation works, see the Postgres for Kubernetes documentation on declarative hibernation.

Request a pause by adding the `k8s.pgd.enterprisedb.io/pause` annotation to the desired PGD group.

For example:

```
kubectl annotate pgdgroup region-a
k8s.pgd.enterprisedb.io/pause=on
```

After a few seconds, the requested PGD group will be in a paused state, with all the database pods removed:

```
kubectl get
pgdgroups

NAME       DATA INSTANCES   WITNESS INSTANCES   PHASE
AGE
region-a   2                1                   PGDGroup - Paused
25m
region-b   2                1                   PGDGroup - Healthy
25m
region-c   0                1                   PGDGroup - Healthy
25m
```

To resume a paused PGD group, set the annotation to `off`. Remember to add the `--overwrite` flag:

```
kubectl annotate pgdgroup region-a k8s.pgd.enterprisedb.io/pause=off --
overwrite
```

In a few seconds, you should see the nodes start resuming, and the pods being re-created.

```
kubectl get
pgdgroups

NAME       DATA INSTANCES   WITNESS INSTANCES   PHASE
AGE
region-a   2                1                   Pause - resume nodes
1m
region-b   2                1                   PGDGroup - Healthy
25m
region-c   0                1                   PGDGroup - Healthy
25m
```

There are some requirements before the pause annotation can put the PGD group in a pause state. Ideally, the PGD Group should be in Healthy state. Alternatively, if all the data nodes in the PGD Group are healthy at the individual level, Pause can also be initiated.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

20 EDB"s private container registry

The images for the EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) and EDB Postgres for Kubernetes operators, as well as various operands, are kept in a private container image registry under `docker.enterprisedb.com`.

> **Important**
>
> Access to the private registry requires an account with EDB and is reserved for EDB customers with a valid subscription plan. Credentials are run through your EDB account. These instructions are the same for trial subscriptions.

Repository information

Collect the following information:

1. Your EDB account token
2. The name of the repository, which will be `k8s`
3. The repository server, which will be `docker.enterprisedb.com`

For clarity, the following examples assume your token is in an environment variable named `EDB_SUBSCRIPTION_TOKEN`.

Example with `docker login`

You can log in via Docker from your terminal. In this context,

- The server is `docker.enterprisedb.com`
- The username is the repository, `k8s`
- The password is your EDB account token (stored in `$EDB_SUBSCRIPTION_TOKEN`)

```
docker login docker.enterprisedb.com  \
  --username k8s
\
  --password "$EDB_SUBSCRIPTION_TOKEN"
```

output

```
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Login Succeeded
```

Operand images

Operand distributions

CNPG-GC is an operator that supports running EDB Postgres Distributed (PGD) version 5 on three PostgreSQL distributions:

- PostgreSQL
- EDB Postgres Advanced Server (EPAS)
- EDB Postgres Extended (PGE)

> **Important**
>
> See Choosing a Postgres distribution in the PGD documentation for details and a comparison of PGD on the different supported PostgreSQL distributions.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Due to the immutable application container adoption in EDB operators, the operator expects the container images to include all the binaries required to run the requested version of PGD on top of the required distribution and version of Postgres.

These images follow the requirements and the conventions described in Container image requirements in the EDB Postgres for Kubernetes documentation, adding the `bdr5` extension.

The table shows the image name prefix for each Postgres distribution.

Postgres distribution	Versions	Image name
PostgreSQL	14 - 17	postgresql-pgd
EDB Postgres Extended	14 - 17	edb-postgres-extended-pgd
EDB Postgres Advanced	14 - 17	edb-postgres-advanced-pgd

Identifying image names

You can select a specific operand and proxy image version that's appropriate for your Postgres distribution.

PGD image name and tag

The operand image name is composed of the Postgres distribution and EDB Postgres Distributed keyword.

Postgres type: The first part of the image name is the Postgres distribution you're using. The table shows how Postgres types correspond to this part of the image name.

Postgres distribution	Image name
EDB Postgres Advanced Server	`edb-postgres-advanced-pgd`
EDB Postgres Extended Server	`edb-postgres-extended-pgd`
PostgreSQL	`postgresql-pgd`

The operand image tag is composed of the Version numbers, and Base image version. In the format of `<Postgres version>-pgd<PGD extension version>-ubi[8|9]`

Version numbers: The versions of the Postgres distribution and PGD extension, separated by a dash. **Base image version:** The Universal Base Image (UBI) version on which the operand is based.

These identifiers, together with their versions, form the operand image name. The table shows some examples.

Postgres version/Proxy image	EDB Postgres Distributed version	UBI	Operand image name and tag
EDB Postgres Advanced 17.7.0	PGD 5.9.1	8	edb-postgres-advanced-pgd:17.7-pgd591-ubi8
EDB Postgres Extended 15.15	PGD 5.9.1	9	edb-postgres-extended-pgd:15.15-pgd591-ubi9
PostgreSQL 16.11	PGD 5.9.1	9	postgresql-pgd:16.11-pgd591-ubi9

For an overview of Postgres compatibility, see Platform compatibility.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Proxy image name and tag

The proxy image name and version derive from EDB Postgres Distributed. See the EDB Postgres Distributed release notes for an overview of PGD versions and corresponding PGD proxy versions. The table shows an example.

Proxy version	Proxy image name and tag
PGD Proxy 5.9.1	edb-pgd-proxy:5.9.1-ubi9

> **Base image version**
>
> As with the operand images, specify the UBI version via the `-ubi8` or `-ubi9` prefixes.

Customize Operand images

You can customize the default operand images for the CNPG-GC operator, and these changes will apply to all future PGDGroup creations. Additionally, you have the option to customize the operand for each individual PGDGroup deployment.

Customize Default Operand Images in Helm Chart Deployment

Use the following Helm chart command to deploy the CNPG-GC operator with customized operand and proxy names. Please note that all operand and operator images will be pulled from the same private repository defined by `global.repository`. For more information, visit the Helm chart page.

```
helm upgrade --dependency-update \
  --install edb-pg4k-pgd \
  --namespace pgd-operator-system \
  --create-namespace \
  edb/edb-postgres-distributed-for-kubernetes
\
  --set global.repository=docker.enterprisedb.com/k8s \
  --set global.pgdImageName=${EDB_PGD_IMAGE_NAME} \
  --set global.proxyImageName=${EDB_PROXY_IMAGE_NAME} \
  --set image.imageCredentials.username=k8s \
  --set image.imageCredentials.password=${EDB_SUBSCRIPTION_TOKEN}
```

> **Note**
>
> With the above installation, the default operand and proxy image are configured in the operator's ConfigMap `pgd-operator-controller-manager-config`. For more information, please refer to the Operator Configuration.

Here's an example of the customization for:

- Using EDB Postgres Advanced Server 15.14.0 as the Postgres option.
- Using PGD 5.9.0 as the Postgres Distributed version.
- Using 5.9.0 as the PGD Proxy version.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
helm upgrade --dependency-update \
    --install edb-pg4k-pgd \
    --namespace pgd-operator-system \
    --create-namespace \
    edb/edb-postgres-distributed-for-kubernetes
\
    --set global.repository=docker.enterprisedb.com/k8s \
    --set global.pgdImageName=edb-postgres-advanced-pgd:15.14-pgd590-ubi8
\
    --set global.proxyImageName=edb-pgd-proxy:5.9.0-ubi8
\
    --set image.imageCredentials.username=k8s \
    --set image.imageCredentials.password=${EDB_SUBSCRIPTION_TOKEN}
```

Update Default Operand Images for an Existing Operator

You can modify the default operand images by updating the `pgd-operator-controller-manager-config` ConfigMap or Secret. For further details, please refer to the Operator Configuration documentation.

Customize Operand Images for Single PGDGroup Deployment

You can also customize operand and proxy image for a single PGDGroup deployment. In this case, it's necessary to create the image pull secrets in the target namespace. If you've already installed the CNPG-GC operator from the private registry, you should have set up an image pull secret.

```
kubectl create secret docker-registry registry-pullsecret
\
  -n <CLUSTER-NAMESPACE> \
  --docker-server=docker.enterprisedb.com \
  --docker-username=k8s \
  --docker-password=${EDB_SUBSCRIPTION_TOKEN}
```

As mentioned earlier, the `docker-username` corresponds to the private registry name, `k8s`, while the `docker-password` is the token obtained from the EDB portal.

After creating your pull secret, ensure you set the `imagePullSecrets` field in the PGD group manifest, along with the `imageName`. The manifest below will create a PGD group running PG Extended from the `k8s` repository.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: group-example-
pge
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  imageName: docker.enterprisedb.com/k8s/edb-postgres-extended-pgd:17.6-pgd590-ubi9
  imagePullSecrets:
  - name: registry-pullsecret
  pgdProxy:
    imageName: docker.enterprisedb.com/k8s/edb-pgd-proxy:5.9.0-ubi9
  pgd:
    parentGroup:
      name: world
      create: true
  cnp:
    storage:
      size:
1Gi
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

21 Predefined labels

These predefined labels are managed by the EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) operator.

- `k8s.pgd.enterprisedb.io/certificateType`

  Indicates the type of the certificates. `replication` indicates a certificate to be used to authenticate the replication client. `server` indicates a certificate to be used for server authentication.

- `k8s.pgd.enterprisedb.io/group`

  Name of the PGDGroup that the resource belongs to. Added to cluster or instance resources.

- `k8s.pgd.enterprisedb.io/isWitnessService`

  Indicates a service is for a witness node.

- `k8s.pgd.enterprisedb.io/type`

  Type of the resource added to cluster or instance resources, usually `node`.

- `k8s.pgd.enterprisedb.io/workloadType`

  Indicates the workload type of the resource added to cluster or instance resources. `pgd-node-data` indicates data node; `pgd-node-witness` a witness node; `pgd-proxy` for PGD Proxy node; `proxy-svc` for PGD Proxy service; `group-svc` for PGD group service to communicate with any node in the PGDGroup; `node-svc` is a service created from the CNP service template; `scheduled-backup` is added to `scheduledBackup` resources; `bootstrap-cross-location-pgd-group` is added to the pod that creates a cross-location PGD group; `pgd-node-restore` is added to the pod that starts the node restore process.

Predefined annotations

- `k8s.pgd.enterprisedb.io/dirtyMetadata`

  Set in a CNP cluster that has been generated from a backup and needs to have its metadata cleaned up before creating the PGD node. This is written by the restore job.

- `k8s.pgd.enterprisedb.io/hash`

  Holds the hash of a certain part of PGDGroup spec that is utilized in various entities like `Cluster`, `ScheduledBackup`, `StatefulSet`, and `Service (node, group and proxy service)` to determine if any updates are required for the corresponding resources.

- `k8s.pgd.enterprisedb.io/latestCleanupExecuted`

  Set in the PGDGroup to indicate that the cleanup was executed.

- `k8s.pgd.enterprisedb.io/node`

  Contains the name of the node for which a certain certificate was generated. Added to the certificate resources.

- `k8s.pgd.enterprisedb.io/nodeRestartHash`

  Stores the hash of the CNP configuration in PGDGroup. A restart is needed when the configuration is changed.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- `k8s.pgd.enterprisedb.io/noFinalizers`

  Set in the PGDGroup with value `true` to skip the finalizer execution. For internal use only.

- `k8s.pgd.enterprisedb.io/pause`

  Set in the PGDGroup to pause a PGDGroup.

- `k8s.pgd.enterprisedb.io/recoverabilityPointsByMethod`

  Set in the PGDGroup to store the CNP cluster's first recoverability points by method in a tamper-proof place.

- `k8s.pgd.enterprisedb.io/reconciliationLoop`

  Set in the PGDGroup with `disabled` to temporarily disable the reconciliation from the operator.

- `k8s.pgd.enterprisedb.io/remoteTarget`

  Set in the PGDGroup during remote join by the bootstrap job, to indicate the remote node to physically join.

- `k8s.pgd.enterprisedb.io/seedingServer`

  Set in the PGDGroup to indicate to the operator which server to restore. This is written by the restore job.

- `k8s.pgd.enterprisedb.io/seedingSnapshots`

  Set in the PGDGroup to indicate to the operator which snapshots to restore. This is written by the restore job.

- `k8s.pgd.enterprisedb.io/useBarmanCloudPlugin`

  Set in the PGDGroup with value `true` to promote the operator to enable the barman-cloud plugin. Once the plugin is enabled, a new sidecar container will be created for the instance pod. The sidecar container will manage the barman base backup and WAL archiving, rather than PostgreSQL container. Keep in mind that the barman-cloud plugin must be installed in the pg4k operator namespace beforehand, and a restart of instance pod is required for these changes to take effect.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

22 PGD Node configuration

The EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC or PGD4K) operator relies on the EDB Postgres for Kubernetes (PG4K) operator to manage individual nodes.

For each PGDGroup, a PGD node is represented as a single-instance PG4K cluster. The PGD4K operator reflects modifications made to the PGDGroup onto the CNP cluster and leverages the PG4K operator to handle these changes. These modifications are specific to the fields `spec.cnp` and `spec.witness` within the PGDGroup.

By default, `spec.cnp` governs the settings for all nodes within the group. If you wish for the witness node to have different configurations from the data nodes, you can define `spec.witness`. Please note that once `spec.witness` is defined, you must explicitly specify all configuration parameters; any configurations not explicitly defined will default to the standard settings.

InitDB option

You can specify the options passed to the initdb command within the `spec.[cnp|witness].initDBOptions` section. The following nodes will initialize from scratch using these options:

- All witness nodes
- Data nodes that use logical join
- The data node that is the first node in the initial group

The supported initdb options within PGDGroup are:

- dataChecksums
- encoding
- walSegmentSize
- localeCollate
- localeCType
- localeProvider
- locale
- icuLocale
- icuRules
- builtinLocale

PGDGroup passes these initdb options to the underlying PG4K cluster. For more details on supported initdb options, please refer to Passing Options to initdb in the PG4K documentation.

Managed configuration

The PGD operator allows configuring the `managed` section of a PG4K cluster. The `spec.cnp.managed` stanza is used for configuring the supported managed roles and services within the cluster.

In this example, a PGDGroup is defined with default `read only` and `read` services disabled. Additionally, it is configured to have a managed role named `foo` with the specified properties set up in postgres.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: group-example-with-
managed
spec:
  [...]
  cnp:
    [...]
    managed:
      roles:
        - name:
foo
          comment:
Foo
          ensure: present
          connectionLimit: 100
          login: true
          superuser: true
          createdb: true
          createrole: true
          replication: true
      services:
        disabledDefaultServices:
          - ro
          -
r
```

For more information about managed roles, see Database role management in the EDB Postgres for Kubernetes documentation.

> **Note**
>
> The PGD4K operator also leverages the PG4K operator to handle managed configurations. User and role definitions in the managed configuration are created or modified within the `postgres` database.

Node Environment Variables

The PGD operator allows configuring the `env` section of a PG4K cluster. The `spec.cnp.env` stanza is used for configuring the environment variables for the instance pod (node).

In the following example, the `WORK_LOAD_TYPE` variable is set for data and witness nodes. If you need to configure additional environment variables for each node type, add them under the respective env maps.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: group-example-with-
environment
spec:
  [...]
  cnp:
    [...]
    env:
      - name:
WORK_LOAD_TYPE
        value: data
  witness:
    [...]
    env:
      - name:
WORK_LOAD_TYPE
        value: witness
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

23 Monitoring

Monitoring Instances

The PGD4K operator leverages the monitoring capabilities of the PG4K operator to provide monitoring for PGD instances. Therefore, all the monitoring configuration options available for PG4K are also available for PGD instances.

The PGD4K operator offers a central location for those monitoring settings in the `spec.cnp.monitoring` section, which will be applied to all nodes in the PGD group.

By default, the PG4K operator provides an HTTP/HTTPS metrics exporter on port `9187` for Prometheus, and both PG4K and PGD4K come with predefined sets of metrics as well as a highly configurable and customizable system to define additional queries via one or more ConfigMap or Secret resources.

> **Note**
>
> Refer to Exposed ports for information about the ports exposed by PGD nodes for monitoring purposes and the related security implications.

> **Note**
>
> Refer to How to inspect the exported metrics for instructions on how to access and inspect the metrics exported by PGD nodes and the operator pod.

> **Note**
>
> Starting from PG4K operator v1.28.3 release, the PG4K operator now uses a dedicated `cnp_metrics_exporter` role with `pg_monitor` privileges to run the metric queries. This change enhances security by following the principle of least privilege. In PGD nodes, the `pg_monitor` role is automatically granted the `bdr_monitor` role to query BDR tables. Refer to release notes for more details about this change.

Monitoring with the Prometheus operator

Like monitoring PG4K instances, you can monitor PGD nodes using the Prometheus Operator's PodMonitor resource. The recommended approach is to manually create and manage a PodMonitor for each PG4K Cluster. This method provides you with full control over the monitoring configuration and lifecycle.

Refer to monitoring with the Prometheus operator for more details about monitoring PG4K instances with the Prometheus operator, which is also applicable to PGD nodes.

Predefined set of metrics

The PGD4K operator installation manifests come with a predefined ConfigMap, called `pgd-operator-default-monitoring`, to be used by all Clusters. `MONITORING_QUERIES_CONFIGMAP` is by default set to `pgd-operator-default-monitoring` in the operator configuration. This ConfigMap contains a predefined set of BDR-related monitoring queries that are automatically injected into each PGD node's monitoring configuration. These metrics use the `cnp_bdr_` prefix. You can customize this ConfigMap to add or modify the default monitoring queries for all PGD nodes.

> **Note**
>
> Refer to predefined set of metrics for more information about the predefined metrics provided by the PG4K operator, which are also applicable to PGD instances.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

User-defined metrics

Default monitoring queries work alongside any user-defined custom queries configured in `spec.cnp.monitoring`. If a user-defined query has the same metric name as a default one, the user-defined query takes precedence.

Here is an example of how to configure the user-defined queries in the PGDGroup manifest. The customized queries exist in the `example-monitoring-configmap` ConfigMap and `example-monitoring-secret` Secret, and they will be injected together with the default queries:

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: group-example-with-
monitoring
spec:
  [...]
  cnp:
    [...]
    monitoring:
      disableDefaultQueries: false
      customQueriesConfigMap:
        - name: example-monitoring-
configmap
          key: custom-queries
      customQueriesSecret:
        - name: example-monitoring-
secret
          key: custom-queries
```

> **Note**
>
> Refer to user-defined metrics for more information about the user-defined metrics provided by the PG4K operator, which are also applicable to PGD instances.

Default Monitoring Queries

The PGD4K operator also supports distributing a set of default monitoring queries to all PGD nodes. This allows you to define a common set of monitoring queries that will be automatically injected into each node's monitoring configuration, ensuring consistent monitoring across the entire PGD group. You can configure the default monitoring queries by providing a ConfigMap or Secret in the operator's namespace, as described in the following sections.

How it works

The operator automatically replicates the source ConfigMap and/or Secret to each PGDGroup namespace as `pgd-operator-default-monitoring`, and injects them into every node's monitoring configuration. Changes to the source are synced on the next reconciliation cycle.

Configuring default monitoring queries

To configure default monitoring queries, set the following environment variables in theoperator configuration:

Environment Variable	Description
`MONITORING_QUERIES_CONFIGMAP`	Name of the ConfigMap in the operator namespace containing the default monitoring queries
`MONITORING_QUERIES_SECRET`	Name of the Secret in the operator namespace containing the default monitoring queries

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Both the ConfigMap and Secret must store the monitoring queries under the `queries` data key. You can check the default queries by inspecting the `pgd-operator-default-monitoring` ConfigMap in the operator namespace:

```
kubectl get configmap pgd-operator-default-monitoring
\
    -n pgd-operator-system -o yaml
```

> **Note**
>
> In the OpenShift environment, the namespace is replaced with `openshift-operators` instead of `pgd-operator-system`.

> **Important**
>
> Changes to the operator configuration require an operator restart to take effect. However, once the operator is configured, any updates to the source ConfigMap or Secret content are automatically propagated to all PGDGroup namespaces without restarting.

Disabling default monitoring queries

You can disable default monitoring queries for a specific PGDGroup by setting `disableDefaultQueries` to `true` in the `spec.cnp.monitoring` section:

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: pgdgroup-example
spec:
  [...]
  cnp:
    monitoring:
      disableDefaultQueries: true
  [...]
```

When default queries are disabled for a PGDGroup, the operator skips creating the default monitoring resources in that namespace and does not inject them into nodes.

> **Important**
>
> The `disableDefaultQueries` field controls the default metrics from both the PGD4K operator and the PG4K operator. Setting it to `true` disables the default set of metrics from PG4K as well as the PGD4K operator's default monitoring queries. If you only want to disable the PGD4K operator's default monitoring queries while keeping the PG4K defaults, leave `disableDefaultQueries` as `false` (or unset) and instead set the `MONITORING_QUERIES_CONFIGMAP` and `MONITORING_QUERIES_SECRET` environment variables to empty values in the operator configuration.

> **Note**
>
> If all PGDGroups in a namespace have default queries disabled (or are deleted), the operator automatically cleans up the `pgd-operator-default-monitoring` ConfigMap and Secret from that namespace.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Monitoring PGD4K Operator instance

Like the PG4K operator, the PGD operator also internally exposes Prometheus metrics via HTTP on port 8080, named `metrics`.

Currently, the PGD operator exposes default Kubebuilder metrics. See kubebuilder documentation for more details.

Monitoring on OpenShift

Refer to the Monitoring on OpenShift section for details.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

24 PGDGroup parting

Deletion and finalizers

When deleting a PGD Group, the operator will start parting every node in the group first. It will connect to an active instance and part every node in the target group. Once a node is parted, the node will not participate in replication and consensus operations. To make sure the node is correctly parted before being deleted, the operator uses the `k8s.pgd.enterprisedb.io/partNodes` finalizer. Please refer to the kubernetes documentation on finalizers for context.

> **Note**
>
> If a namespace holding a PGD Group is deleted directly, the operator cannot ensure the deleting and parting sequence is carried out correctly. Before deleting a namespace, it is recommended to delete all the contained PGD groups.

Time limit

When parting a node, the operator needs to connect to an active instance to execute the `bdr.part_node` function. To avoid this operation hanging, a time limit for the finalizer is used; by default, it is 300 seconds. After the time limit expires, the finalizer will be removed, and the node will be deleted anyway, potentially leaving stale metadata in the global PGD catalog. This time limit can be configured through `spec.failingFinalizerTimeLimitSeconds`, which is specified in seconds.

Skip finalizer

For testing purposes only, the operator also provides an annotation to skip the finalizer: `k8s.pgd.enterprisedb.io/noFinalizers`. When this annotation is added to a PGDGroup, the finalizer will be skipped when the PGDGroup is being deleted, and the nodes will not be parted from the PGD cluster.

PGDGroup cleanup

Clean up parted nodes

Once the PGDGroup is deleted from Kubernetes and all the nodes belonging to this group are parted, the group information is still available in the pgd metadata like `bdr.node_group`. The PGD4K operator defines a CRD named `PGDGroupCleanup` to help drop the PGDGroup and clean up the `parted` nodes belonging to this group, if any.

In the example below, the `PGDGroupCleanup` executes locally from `region-a`, and will clean up all parted nodes of region-b, with the prerequisite that all the nodes must be in the `PARTED` state. It will then drop the PGDGroup `region-b`.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroupCleanup
metadata:
  name: region-b-cleanup
spec:
  executor: region-a
  target: region-b
```

Please note that if the target group (`region-b` in the example) contains nodes not in a `PARTED` state, the Group Cleanup will stop in phase `PGDGroupCleanup - Waiting for nodes in target PGDGroup to be parted`. In cases of extreme need, we can add the `force` option.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

> **Warning**
>
> Using `force` can leave the PGD cluster in an inconsistent state. Use it only to recover from failures in which you can't part the group nodes any other way.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroupCleanup
metadata:
  name: region-b-cleanup
spec:
  force: true
  executor: region-a
  target: region-b
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

25 SQLMutations

SQLMutations consist of a list of SQL queries to execute on the application database via the superuser role after a PGD node joins the PGDGroup. Each SQLMutation includes an `isApplied` list of queries and an `exec` list of queries. The `isApplied` SQL queries are used to check if the mutation was already applied. If any of the `isApplied` queries return false, the `exec` list of SQL queries is added to the execution queue.

Here's a sample of SQLMutations:

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
[...]
spec:
  pgd:
    mutations:
      - isApplied:
          -
|
             WITH node_route_config AS
(
               SELECT node_name,
route_dsn
               FROM
bdr.node_routing_config_summary

)
            SELECT NOT EXISTS(SELECT * FROM node_route_config WHERE regexp_count(route_dsn,
'application_name=') >= 1);
        exec:
          -
|
            WITH node_route_config AS
(
              SELECT
node_name,
                     regexp_replace(route_dsn, concat(' application_name=',  node_name), '') as
route_dsn
              FROM
bdr.node_routing_config_summary

)
            SELECT bdr.alter_node_option(node_name, 'route_dsn',
route_dsn)
            FROM node_route_config;
```

SQLMutation types

The operator offers three types of SQLMutations, which you specify with `spec.pgd.mutations[].type`. The default is `always`.

- `beforeSubgroupRaft`
- `always`
- `writeLeader`

The `beforeSubgroupRaft` and `always` mutations are evaluated in every reconcile loop. The difference between the two mutations lies in their execution phase:

- For `always` mutations, they're run in each reconcile loop without any restrictions on the PGDGroup.
- `beforeSubgroupRaft` mutations are executed only if the PGDGroup has defined data nodes, and specifically before the subgroup Raft becomes ready.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Both `beforeSubgroupRaft` and `always` mutations can run on any PGD node in the PGDGroup, including witness nodes. Therefore, don't use them for making data changes to the application database, as witness nodes don't contain application database data.

The `writeLeader` mutation is triggered and executed after the write leader is elected. The `exeI. Oh. I. c` operations are carried out exclusively on the write leader node.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

26 Red Hat OpenShift

EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) is a certified operator that can be installed on OpenShift using a web interface.

Ensuring access to the EDB private registry

> **Important**
>
> You need access to the private EDB repository where both the operator and operand images are stored. Access requires a valid EDB subscription plan. See Accessing EDB private image registries for details.

The OpenShift install uses pull secrets to access the operand and operator images, which are held in a private repository.

Once you have credentials to the private repo, you need to create two pull secrets in the `openshift-operators` namespace:

- `pgd-operator-pull-secret` for the EDB Postgres® AI for CloudNativePG™ Cluster (PG4K) operator images
- `postgresql-operator-pull-secret` for the PG4K operator images

You can create each secret using the `oc create` command by replacing `<TOKEN>` with the repository token for your EDB account, as explained in Get your token.

```
oc create secret docker-registry pgd-operator-pull-secret
\
  -n openshift-operators \
  --docker-server=docker.enterprisedb.com \
  --docker-username=k8s \
  --docker-password="<TOKEN>"

oc create secret docker-registry postgresql-operator-pull-secret
\
  -n openshift-operators \
  --docker-server=docker.enterprisedb.com \
  --docker-username=k8s \
  --docker-password="<TOKEN>"
```

> **Note**
>
> For pg4k-pgd operator v1.1.3 and earlier (certified against the legacy k8s_enterprise_pgd repository), you must create the `pgd-operator-pull-secret` image-pull secret using `k8s_enterprise_pgd` as the registry username.

Installing the operator

!!!

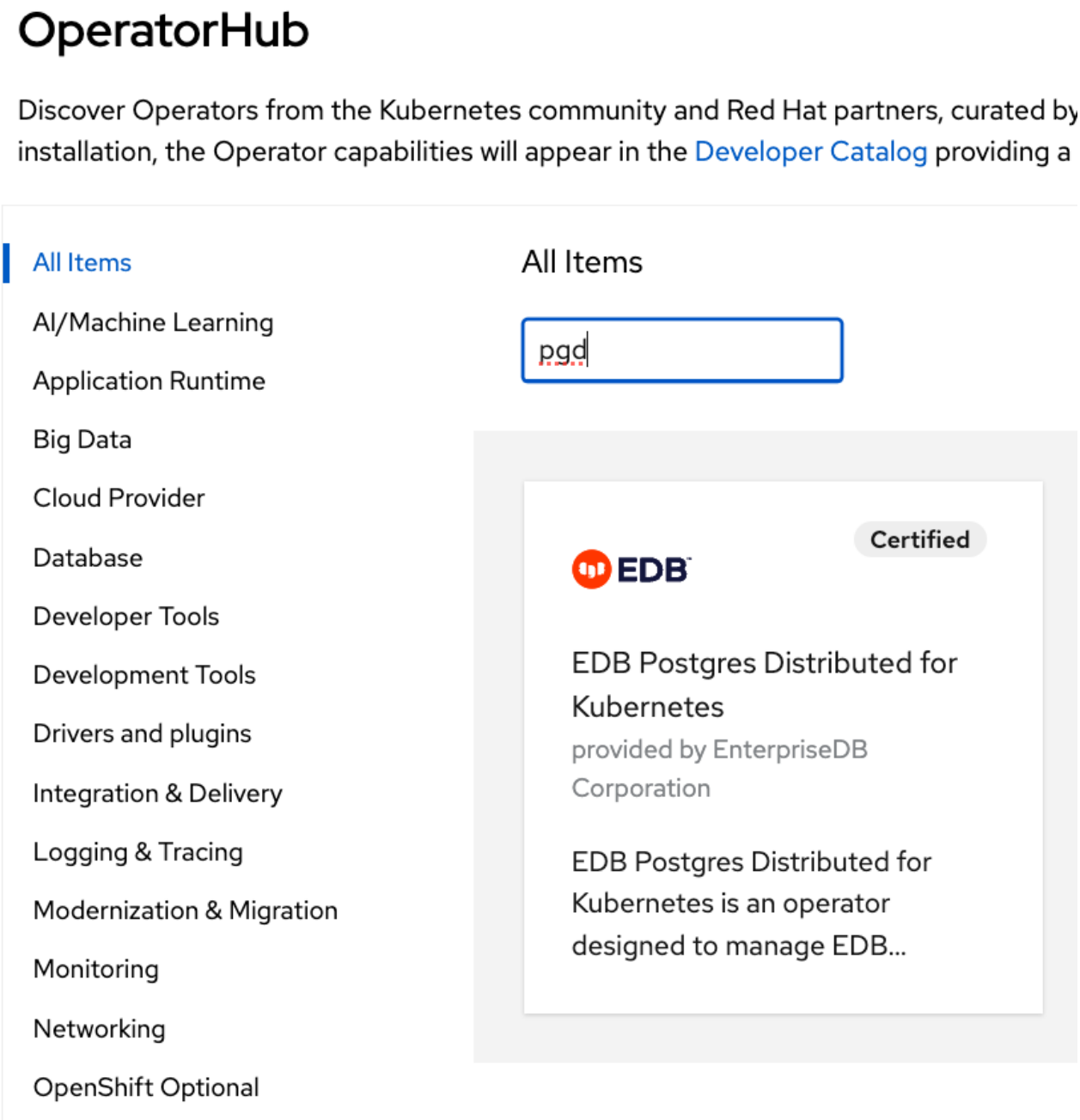
> **CRITICAL WARNING: UPGRADING OPERATORS**
>
> OpenShift users, or any customer attempting an operator upgrade, MUST configure the new unified repository pull secret (docker.enterprisedb.com/k8s) before running the upgrade. If the old, deprecated repository path is still in use during the upgrade process, image pull failure will occur, leading to deployment failure and potential downtime. Follow the Central Migration Guide first.

The CNPG-GC operator can be found in the Red Hat OperatorHub directly from your OpenShift dashboard.

- From the hamburger menu, select **Operators > OperatorHub**.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- In the web console, use the search box to filter the listing. For example, enter `EDB` or `pgd`:

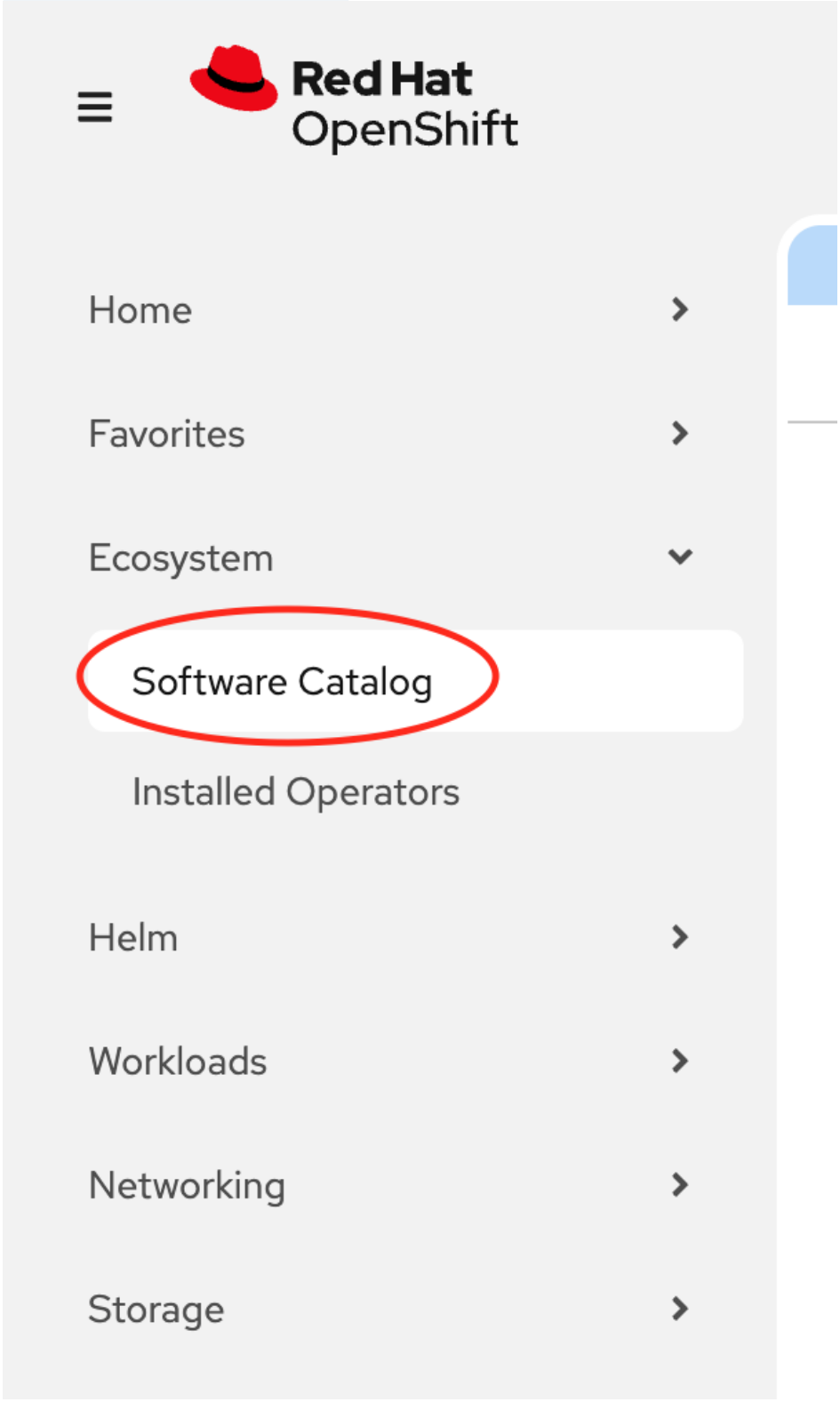


- Read the information about the operator and select **Install**.
- In the Operator Installation page, select:
  - The installation mode. Cluster-wide is currently the only mode.
  - The update channel (currently **preview**).
  - The approval strategy, following the availability on the marketplace of a new release of the operator, certified by Red Hat:
    - **Automatic**: OLM upgrades the running operator with the new version.
    - **Manual**: OpenShift waits for human intervention by requiring an approval in the **Installed Operators** section.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Note

If you are running with OpenShift 4.20 or later, OperatorHub has been integrated into the Software Catalog. In the web console, navigate to `Operators -> Software Catalog` and select a Project to view the software catalog.



Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Cluster-wide installation

With cluster-wide installation, you're asking OpenShift to install the operator in the default `openshift-operators` namespace and to make it available to all the projects in the cluster. This is the default and normally recommended approach to install EDB Postgres Distributed for Kubernetes.

From the web console, for **Installation mode**, select **All namespaces on the cluster (default)**.

On installation, the operator is visible in all namespaces. In case there were problems during installation, check the logs in any pods in the `openshift-operators` project on the **Workloads > Pods** page as you would with any other OpenShift operator.

> **Beware**
>
> By choosing the cluster-wide installation, you can't easily move to a single-project installation later.

Creating a PGD cluster

After the installation by OpenShift, the operator deployment is in the `openshift-operators` namespace. Notice the cert-manager operator was also installed, as was the EDB Postgres for Kubernetes operator (`postgresql-operator-controller-manager`).

```
$ oc get deployments -n openshift-
operators
NAME                                        READY   UP-TO-DATE   AVAILABLE
AGE
cert-manager-operator                       1/1     1            1
11m
pgd-operator-controller-manager             1/1     1            1
11m
postgresql-operator-controller-manager-1-20-0   1/1     1            1
23h
…
```

After checking that the `pgd-operator-controller-manager` deployment is READY, you can start creating PGD clusters. The CNPG-GC repository contains some useful sample files.

You must deploy your PGD clusters on a dedicated namespace/project. The default namespace is reserved.

First, create a new namespace, then deploy a self-signed certificate `Issuer` in it:

```
oc create ns my-
namespace
oc apply -n my-namespace -f
\
  https://raw.githubusercontent.com/EnterpriseDB/edb-postgres-for-kubernetes-
charts/main/hack/samples/issuer-selfsigned.yaml
```

Using PGD in a single OpenShift cluster in a single region

Now you can deploy a PGD cluster, for example a flexible 3-region, which contains two data groups and a witness group. You can find the YAML manifest in the file `flexible_3regions.yaml` .

```
oc apply -f flexible_3regions.yaml -n my-
namespace
```

Your PGD groups start to come up:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
$ oc get pgdgroups -n my-
namespace
NAME       DATA INSTANCES   WITNESS INSTANCES   PHASE                PHASE DETAILS
AGE
region-a   2                1                   PGDGroup - Healthy
23m
region-b   2                1                   PGDGroup - Healthy
23m
region-c   0                1                   PGDGroup - Healthy
23m
```

Using PGD in multiple OpenShift clusters in multiple regions

To deploy PGD in multiple OpenShift clusters in multiple regions, you must first establish a way for the PGD groups to communicate with each other.

Configuring the connectivity is outside the scope of this documentation. However, once you've established connectivity between the OpenShift clusters, you can deploy PGD groups synced with one another.

Channel

The CNPG-GC operator is available in a single OLM channel named `stable` since v1.0.0. All the stable releases will be available in this channel. The `candidate` channel is only used for RC and preview release.

Dependencies

The CNPG-GC operator (PGD4K) on OpenShift has the following dependencies:

- EDB Postgres® AI for CloudNativePG™ Cluster operator (PG4K): In release v1.0.1 and later, PGD4K is constrained to use an LTS release of PG4K.
- cert-manager: In release v1.1.1 and later, both cert-manager Operator for Red Hat OpenShift and cert-manager Operator for Community are supported.

For more details about the supported versions for each release, see Supported versions. Be aware that you can install or upgrade the PGD4K operator only when those dependencies are met.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

27 Join method

When scaling up a PGD group by adding more data nodes or creating cross-region groups, it's crucial for new nodes to join the existing group properly. There are two methods for joining:

- Logical join

  This method uses the bdr.join_node_group function to integrate the new node into the PGD group. It's important that the joining node doesn't contain any schemas or data present in the PGD group. It's recommended that the new database contain only the BDR extension, as data synchronization occurs during the join.

- Physical join

  This method uses the bdr_init_physical command to speed up the joining process. You can prepare data in advance before executing `bdr_init_physical`.

For more information about join methods, see Creating and joining PGD groups.

The initial group

The PGD4K operator allows you to configure the join methods for nodes joining PGD groups as well as for groups joining other groups.

When creating a single PGD group, you must configure it to create the *parent group* (`spec.pgd.parentGroup.create` =true). If you intend to create multiple PGD groups that will cross join, at least one of these groups must be designated to form the parent group. This group is referred to as the *initial* PGD group.

The first node in the `initial` PGD group can either be restored from a backup (as described in the recovery documentation) or created from scratch. This node will always use a logical join to connect to the parent group it created. Conversely, the first node in the *non-initial* PGD group is used for cross joining with other PGD groups.

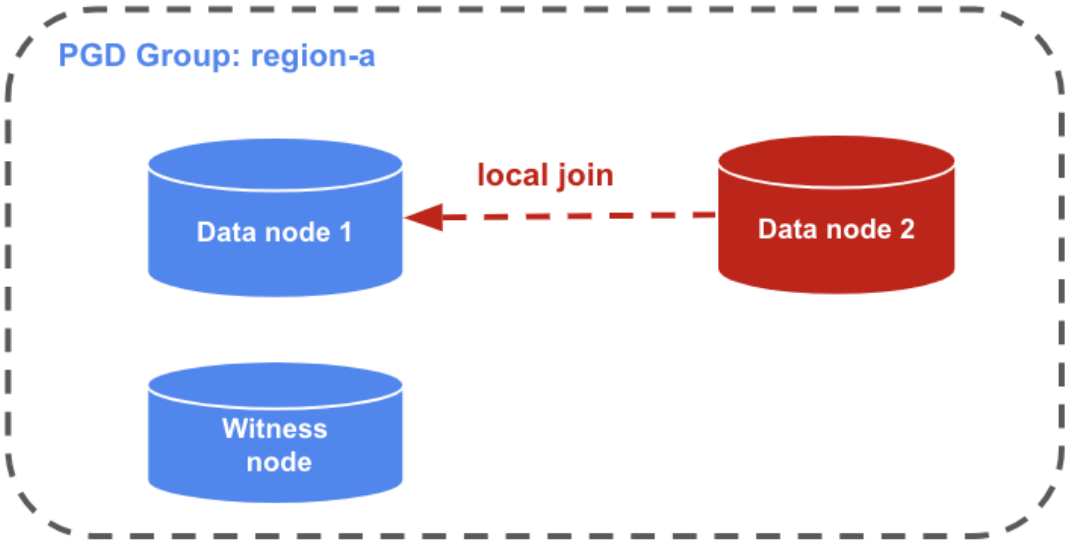
You can configure the join method for the first node (group join) and the subsequent nodes (node join) separately.

> **Note**
>
> In the scenario where a group joins another group, the first node of the joining group actually creates the group first before joining a different group (either physically or logically).

Configure the join method for node join

You can configure the join method for a node joining a group when scaling up the PGD group and adding new nodes to the local group. In this context, the joining group is the same as the node group.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.



The `spec.cnp.joinMethod` parameter configures how a node joins a local group. By default, a logical join is used. To enable a physical join for a new node, set `spec.cnp.joinMethod` = `physical`.

In the following example, the second node uses the `pg_basebackup` command to replicate data from the first node in the `region-a` group. Then it uses `bdr_init_physical` to join the `region-a` group.

This example shows a physical join:

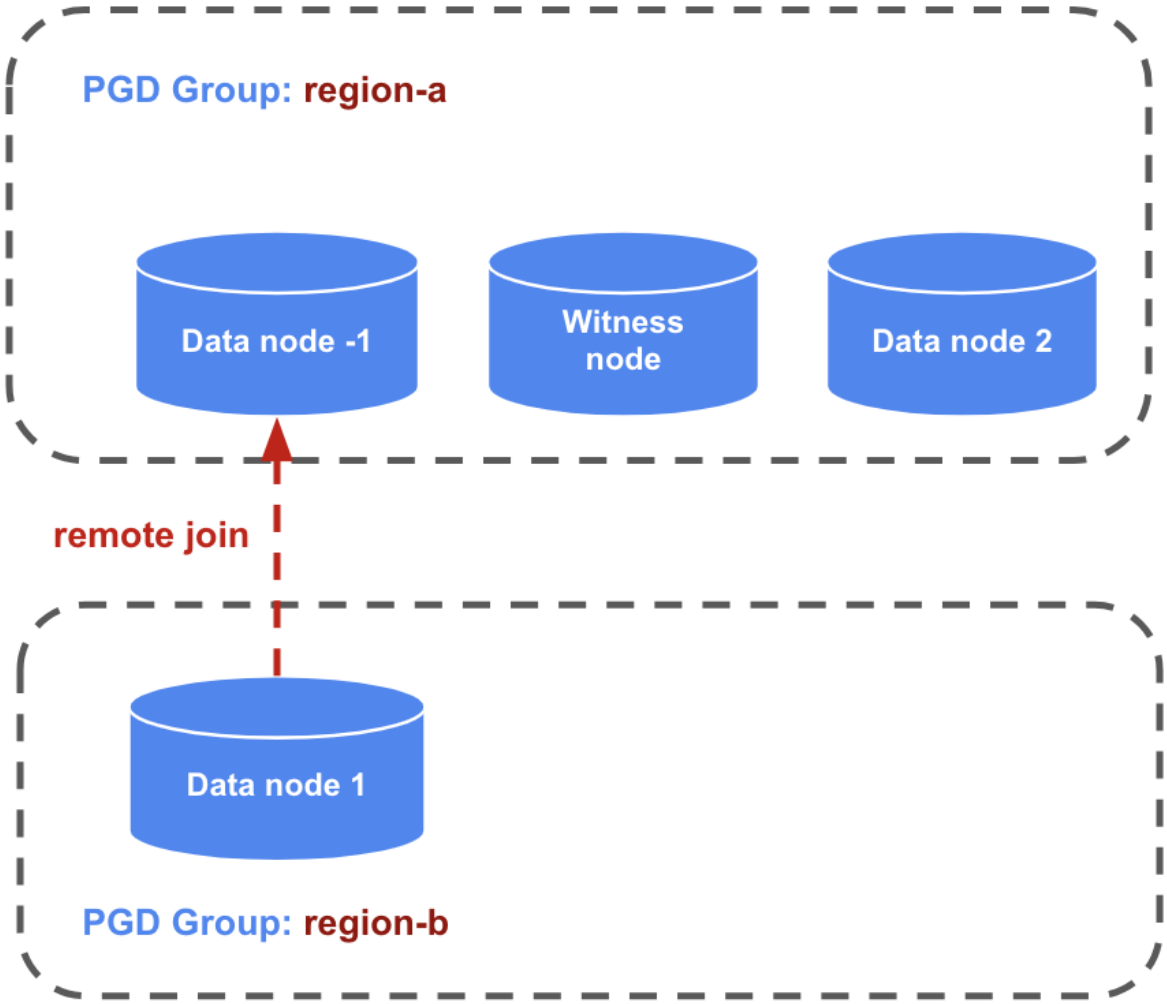
```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: region-a
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  cnp:
    joinMethod:
physical
    storage:
      size:
1Gi

...
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Configure the join method for group join

You can configure the join method for group joins when creating cross-region PGD groups. The first node in the new PGD group joins an existing PGD group. In this context, the joining group is distinct from the node group.



The PGD4K operator uses the `spec.pgd.joinMethod` section (which defaults to logical) to determine how PGD groups join with each other. It uses the `spec.pgd.discovery` section to specify the target to join.

This example shows a logical join, where `spec.pgd.groupJoinMethod=logical`. The first node in the region-b group waits for all hosts specified in `spec.pgd.discovery` to become available. It then identifies a suitable target to perform the logical join using the `bdr.join_node_group` function. The *group service* of all remote groups is typically defined in the discovery section for this type of join.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: region-b
  namespace: region-b
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    groupJoinMethod: logical
    discovery:
      - host: region-a-group
      - host: region-c-group
  cnp:
    storage:
      size:
1Gi
```

This example shows a physical join, configured with `spec.pgd.groupJoinMethod=physical`. The first node in the region-b group will traverse all the nodes defined in `spec.pgd.discovery`, waiting to find one node with a successful Raft consensus to perform the physical join using `bdr_init_physical`. Typically, the *node service* of the remote group is specified in the discovery section for this type of join.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: region-b
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    groupJoinMethod:
physical
    discovery:
      - host: region-a-3-node
      - host: region-a-1-node
      - host: region-a-2-node
  cnp:
    joinMethod:
physical
    storage:
      size:
1Gi
```

> **Note**
>
> A group join is essentially a type of node join, specifically when the first node in the group joins a node from a different group.

> **Note**
>
> Witness nodes always use a logical join to connect to a group.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

28 Transparent data encryption (TDE)

> **Important**
>
> TDE is available *only* for operands that support it: EDB Postgres Advanced Server versions 15 and newer and EDB Postgres Extended versions 15 and newer.

Transparent data encryption, or TDE, is a technology used by several database vendors to encrypt data at rest, that is, database files on disk. However, TDE doesn't encrypt data in use.

TDE is included in EDB Postgres Advanced Server or EDB Postgres Extended, starting with version 15, and is supported by EDB Postgres® AI for CloudNativePG™ (CNPG-GC).

> **Important**
>
> Before you proceed, please take some time to familiarize with theTDE feature in the EPAS documentation.

With TDE activated, both WAL files and files for tables are encrypted. Data encryption/decryption is entirely transparent to the user, as it's managed by the database without requiring any application changes or updated client drivers.

The support for TDE on CNPG-GC relies on the implementation from EDB Postgres for Kubernetes (PG4K). See the PG4K documentation for the full context.

You can use TDE with a passphrase stored in a Kubernetes secret, which is used to encrypt the EDB Postgres Advanced Server binary key.

> **EDB Postgres Advanced Server documentation**
>
> See the EDB Postgres Advanced Server documentation for details on this encryption key.

TDE on CNPG-GC relies on the PG4K implementation. Activating TDE on a cluster uses the `epas` section of the manifest, which is in the `cnp` section used for PG4K-level directives such as storage. Use the `tde` stanza to enable TDE, and set the name of the Kubernetes secret holding the TDE encryption key.

The following YAML portion contains both a secret holding a passphrase (base-64 encoded), and the `epas` section activating TDE with the passphrase.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
---
apiVersion: v1
kind:
Secret
metadata:
  name: tde-key
data:
  key:
bG9zcG9sbGl0b3NkaWNlbnBpb3Bpb3Bpb2N1YW5kb3RpZW5lbmhhbWJyZWN1YW5kb3RpZW5lbmZyaW8=

---
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
[…]
spec:
  instances: 3
[…]
  cnp:
    postgresql:
      epas:
        tde:
          enabled: true
          secretKeyRef:
            name: tde-key
            key:
key
    storage:
      size:
1Gi
```

Again, see the PG4K documentation for additional depth, including how to create the encryption secret and additional ways of using TDE.

As shown in the TDE feature documentation, the information is encrypted at rest.

For example, open a psql terminal into one of your data nodes.

```
kubectl exec -ti <DATA-NODE> -- psql
app
```

Create a new table including a text column:

```
create table foo(bar int, baz
varchar);
insert into foo(bar, baz) values (1, 'hello'), (2,
'goodbye');
```

Verify the location where the newly defined table is stored on disk:

```
select
pg_relation_filepath('foo');
 pg_relation_filepath
----------------------
 base/16385/16387
```

You can open a terminal on the same data node:

```
kubectl exec -ti <DATA-NODE> --
bash
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

There, you can verify the file was encrypted:

```
cd $PGDATA/base/16385
hexdump -C 16387 | grep hello
hexdump -C 16387 | grep goodbye
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

29 LDAP authentication

EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) supports LDAP authentication. LDAP configuration on CNPG-GC relies on the implementation from EDB Postgres for Kubernetes (PG4K). See the PG4K documentation for the full context.

> **Important**
>
> Before you proceed, familiarize yourself with the LDAP authentication feature in the Postgres documentation.

With LDAP support, only the user authentication is sent to LDAP, so the user must already exist in the PostgreSQL database.

This example shows an LDAP configuration using `simple bind` mode in PGDGroup. The Postgres server uses `prefix + username + suffix` and password to bind the LDAP server to achieve the authentication.

```
spec:
  [...]
  cnp:
    postgresql:
      ldap:
        server: 'ldap-service.ldap.svc.cluster.local'
        bindAsAuth:
          prefix: "uid="
          suffix: ",dc=example,dc=org"
```

This example shows how to configure LDAP using `search+bind` mode in PGDGroup. In this mode, the Postgres instance is first bound to the LDAP using `bindDN` with its password stored in the secret `bindPassword`. Then Postgres tries to perform a search under `baseDN` to find a username that matches the item specified by `searchAttribute`. If a match is found, Postgres finally verifies the entry and the password against the LDAP server.

```
spec:
  [...]
  cnp:
    postgresql:
    ldap:
        server: 'ldap-service.ldap.svc.cluster.local'
        bindSearchAuth:
        baseDN: "dc=example,dc=org"
        bindDN: "cn=admin,dc=example,dc=org"
        searchAttribute: "uid"
        bindPassword:
            name: ldap-bind-password
            key:
password
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

30 Logging

EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) outputs logs in JSON format directly to standard output, including PostgreSQL logs, without persisting them to storage for security reasons. This design facilitates seamless integration with most Kubernetes-compatible log management tools, including command line ones like stern.

As CNPG-GC leverages the EDB Postgres for Kubernetes clusters to manage nodes, logs for EDB Postgres for Kubernetes operator and instance are also applicable here. Check here for more information about logs for EDB Postgres for Kubernetes.

Each log entry includes the following fields:

- `level` – The log level (e.g., `info`, `notice`).
- `ts` – The timestamp.
- `logger` – The type of log (e.g., `postgres`, `pg_controldata`).
- `msg` – The log message, or the keyword `record` if the message is in JSON format.
- `record` – The actual record, with a structure that varies depending on the `logger` type.
- `logging_pod` – The name of the pod where the log was generated.

> **Info**
>
> If your log ingestion system requires custom field names, you can rename the `level` and `ts` fields using the `log-field-level` and `log-field-timestamp` flags in the operator controller. This can be configured by editing the `Deployment` definition of the `cloudnative-pg` operator.

Node Logs

Node logs are PostgreSQL instance pod logs. You can configure the log level for the PGD node in the PGDGroup specification using the `logLevel` option, PGDGroup will deliver the `logLevel` changes to all the PG4K clusters underlying. Available log levels are: `error`, `warning`, `info` (default), `debug`, and `trace`.

- change the logLevel for data node in `spec.cnp.logLevel`
- change the logLevel for witness node in `spec.witness.logLevel`

> **Important**
>
> Currently, the log level can only be set at the time the instance starts. Changes to the log level in the pgdgroup specification after the group has started will cause the nodes to restart.

Operator Logs

There are two operator logs you can check here as described above:

- Operator logs for CNPG-GC
- Operator logs for EDB Postgres for Kubernetes

The logs produced by the operator pod can be configured with log levels, same as instance pods: `error`, `warning`, `info` (default), `debug`, and `trace`.

The log level for the operator can be configured by editing the `Deployment` definition of the operator and setting the `--log-level` command line argument to the desired value.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

PostgreSQL Logs

Each PostgreSQL log entry is a JSON object with the `logger` key set to `postgres`. The structure of the log entries is as follows:


```
{
  "level": "info",
  "ts": 1619781249.7188137,
  "logger": "postgres",
  "msg": "record",
  "record": {
    "log_time": "2021-04-30 11:14:09.718 UTC",
    "user_name": "",
    "database_name": "",
    "process_id": "25",
    "connection_from": "",
    "session_id": "608be681.19",
    "session_line_num": "1",
    "command_tag": "",
    "session_start_time": "2021-04-30 11:14:09
UTC",
    "virtual_transaction_id": "",
    "transaction_id": "0",
    "error_severity": "LOG",
    "sql_state_code": "00000",
    "message": "database system was interrupted; last known up at 2021-04-30 11:14:07
UTC",
    "detail": "",
    "hint": "",
    "internal_query": "",
    "internal_query_pos": "",
    "context": "",
    "query": "",
    "query_pos": "",
    "location": "",
    "application_name": "",
    "backend_type": "startup"
  },
  "logging_pod": "region-a-1-1",
}
```


> **Info**
>
> Internally, the operator uses PostgreSQL's CSV log format. For more details, refer to the PostgreSQL documentation on CSV log format.

PGAudit Logs

CNPG-GC offers seamless and native support for PGAudit on PostgreSQL clusters.

To enable PGAudit, add the necessary `pgaudit` parameters in the `spec.[cnp|witness].postgresql` section of the PGDGroup configuration. PGDGroup will update the configuration to each EDB Postgres for Kubernetes underlying.

> **Important**
>
> The PGAudit library must be added to `shared_preload_libraries`. EDB Postgres for Kubernetes automatically manages this based on the presence of `pgaudit.*` parameters in the PostgreSQL configuration. The operator handles both the addition and removal of the library from `shared_preload_libraries`.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Additionally, the operator manages the creation and removal of the PGAudit extension across all databases within the cluster.

The following example demonstrates a PostgreSQL `PGDGroup` deployment with PGAudit enabled and configured:

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: pgdgroup-example
spec:
  instances: 3
  cnp:
    postgresql:
      parameters:
        "pgaudit.log": "all, -
misc"
        "pgaudit.log_catalog": "off"
        "pgaudit.log_parameter": "on"
        "pgaudit.log_relation": "on"

  storage:
    size:
1Gi
```

The audit CSV log entries generated by PGAudit are parsed and routed to standard output in JSON format, similar to all other logs:

- `.logger` is set to `pgaudit`.
- `.msg` is set to `record`.
- `.record` contains the entire parsed record as a JSON object. This structure resembles that of `logging_collector` logs, with the exception of `.record.audit`, which contains the PGAudit CSV message formatted as a JSON object.

This example shows sample log entries:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
{
  "level": "info",
  "ts": 1627394507.8814096,
  "logger": "pgaudit",
  "msg": "record",
  "record": {
    "log_time": "2021-07-27 14:01:47.881 UTC",
    "user_name": "postgres",
    "database_name": "postgres",
    "process_id": "203",
    "connection_from": "[local]",
    "session_id": "610011cb.cb",
    "session_line_num": "1",
    "command_tag": "SELECT",
    "session_start_time": "2021-07-27 14:01:47
UTC",
    "virtual_transaction_id": "3/336",
    "transaction_id": "0",
    "error_severity": "LOG",
    "sql_state_code": "00000",
    "backend_type": "client
backend",
    "audit": {
      "audit_type": "SESSION",
      "statement_id": "1",
      "substatement_id": "1",
      "class": "READ",
      "command": "SELECT FOR KEY
SHARE",
      "statement": "SELECT
pg_current_wal_lsn()",
      "parameter": "<none>"
    }
  },
  "logging_pod": "cluster-example-1",
}
```


See the PGAudit documentation for more details about each field in a record.

EDB Audit logs

Clusters that are running on EDB Postgres Advanced Server (EPAS) can enable EDB Audit as follows:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
apiVersion:
pgd.k8s.enterprisedb.io/v1beta1
kind:
PGDGroup
metadata:
  name: pgdgroup-example
spec:
  instances: 3
  imageName: docker.enterprisedb.com/k8s/edb-postgres-advanced-pgd:17-pgd5-ubi9
  licenseKey: <LICENSE>

  cnp:
    postgresql:
      epas:
        audit: true

  storage:
    size:
1Gi
```

Setting `.spec.cnp.postgresql.epas.audit: true` enforces the following parameters to all the nodes:

```
edb_audit = 'csv'
edb_audit_destination = 'file'
edb_audit_directory = '/controller/log'
edb_audit_filename = 'edb_audit'
edb_audit_rotation_day = 'none'
edb_audit_rotation_seconds = '0'
edb_audit_rotation_size = '0'
edb_audit_tag = ''
edb_log_every_bulk_value = 'false'
```

Other parameters can be passed via `.spec.cnp.postgresql.parameters` as usual.

The audit CSV logs are parsed and routed to stdout in JSON format, similarly to all the remaining logs:

- `.logger` set to `edb_audit`
- `.msg` set to `record`
- `.record` containing the whole parsed record as a JSON object

See the example below:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
{
  "level": "info",
  "ts": 1624629110.7641866,
  "logger": "edb_audit",
  "msg": "record",
  "record": {
    "log_time": "2021-06-25 13:51:50.763 UTC",
    "user_name": "postgres",
    "database_name": "postgres",
    "process_id": "68",
    "connection_from": "[local]",
    "session_id": "60d5df76.44",
    "session_line_num": "5",
    "process_status": "idle in transaction",
    "session_start_time": "2021-06-25 13:51:50
UTC",
    "virtual_transaction_id": "3/93",
    "transaction_id": "1183",
    "error_severity": "AUDIT",
    "sql_state_code": "00000",
    "message": "statement: GRANT EXECUTE ON function pg_catalog.pg_read_binary_file(text) TO
\"streaming_replica\"",
    "detail": "",
    "hint": "",
    "internal_query": "",
    "internal_query_pos": "",
    "context": "",
    "query": "",
    "query_pos": "",
    "location": "",
    "application_name": "",
    "backend_type": "client
backend",
    "command_tag": "GRANT",
    "audit_tag": "",
    "type": "grant"
  },
  "logging_pod": "pgdgroup-example-1-1",
}
```


See EDB Audit file for more details about the records' fields.

Other Logs

All logs generated by the operator and PG4K instances are in JSON format, with the `logger` field indicating the process that produced them. The possible `logger` values are as follows:

- `barman-cloud-wal-archive`: logs from `barman-cloud-wal-archive`
- `barman-cloud-wal-restore`: logs from `barman-cloud-wal-restore`
- `edb_audit`: from the EDB Audit extension
- `initdb`: logs from running `initdb`
- `pg_basebackup`: logs from running `pg_basebackup`
- `pg_controldata`: logs from running `pg_controldata`
- `pg_ctl`: logs from running any `pg_ctl` subcommand
- `pg_rewind`: logs from running `pg_rewind`
- `pgaudit`: logs from the PGAudit extension
- `postgres`: logs from the `postgres` instance (with `msg` distinct from `record`)
- `wal-archive`: logs from the `wal-archive` subcommand of the instance manager
- `wal-restore`: logs from the `wal-restore` subcommand of the instance manager

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- `instance-manager`: from the PostgreSQL instance manager in each node.

With the exception of `postgres` and `edb_audit`, which follow a specific structure, all other `logger` values contain the `msg` field with the escaped message that is logged.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

31 API Reference

Packages

- pgd.k8s.enterprisedb.io/v1beta1

Package v1beta1 contains API Schema definitions for the pgd v1beta1 API group

Resource Types

- ClusterImageCatalog
- ImageCatalog
- PGDGroup
- PGDGroupCleanup

Backup

Backup configures the backup of cnp-pgd nodes

Appears in:

- PGDGroupSpec

Field	Description	Required	Default	Validation
`configuration` *BackupConfiguration*	The CNP configuration to be used for backup. ServerName value is reserved by the operator.			
`cron` *ScheduledBackupSpec*	The scheduled backup for the data. Deprecated: This field is deprecated and will be removed in future versions.			
`schedulers` *ScheduledBackupSpec array*	Define schedulers for the backup. Each scheduler has a different backup method. Only one of either `backup.cron` or `backup.schedulers` can be defined.			

BackupStatus

BackupStatus contains the current status of the pgd backup

Appears in:

- PGDGroupStatus

Field	Description	Required	Default	Validation
`clusterName` *string*	ClusterName the elected cluster to take the backup, the backup could be scheduled with different methods, but only one cluster will take the backup			
`scheduledBackupName` *string*	ScheduledBackupName is the name of the scheduled backup. Deprecated: This field is deprecated and will be removed in future versions. Please use ScheduledBackupStatus instead			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`scheduledBackupHash` *string*	ScheduledBackupHash is the hash of the scheduled backup configuration. Deprecated: This field is deprecated and will be removed in future versions. Please use ScheduledBackupStatus instead			
`scheduledBackups` *ScheduledBackupStatus array*	ScheduledBackupStatus contains the status of all scheduled backups			

BarmanCloudPluginStatus

BarmanCloudPluginStatus contains the status of the barman-cloud plugin

Appears in:

- PluginStatus

Field	Description	Required	Default	Validation
`pluginEnabled` *boolean*	PluginEnabled is true when the barman cloud plugin is enabled			
`objectStore` *ObjectStoreStatus array*	ObjectStore contains the name and hashcode of the managed objectStore			

CNPStatus

CNPStatus contains any relevant status for the operator about CNP

Appears in:

- PGDGroupStatus

Field	Description	Required	Default	Validation
`dataInstances` *integer*			0	
`witnessInstances` *integer*			0	
`clusterStatus` *ClusterStatus array*	ClusterStatus contains the list of the status of the CNP clusters			
`firstRecoverabilityPointsByMethod` *object (keys:string, values:RecoverabilityPointsByMethod)*	The recoverability points by method, keyed per CNP clusterName nolint: lll Deprecated: the field is not set for backup plugins.			
`firstRecoverabilityPoints` *object (keys:string, values:string)*	The recoverability points, keyed per CNP clusterName, as a date in RFC3339 format Deprecated: the field is not set for backup plugins.			
`superUserSecretIsPresent` *boolean*		True		
`applicationUserSecretIsPresent` *boolean*		True		
`podDisruptionBudgetIsPresent` *boolean*		True		

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

CatalogImage

CatalogImage defines the image and major version

Appears in:

- ImageCatalogSpec

Field	Description	Required	Default	Validation
`image` *string*	The image reference	True		
`major` *integer*	The PostgreSQL major version of the image. Must be unique within the catalog.	True		Minimum: 10

CertManagerTemplate

CertManagerTemplate contains the data to generate a certificate request

Appears in:

- ClientCertConfiguration
- ServerCertConfiguration

Field	Description	Required	Default	Validation
`spec` *CertificateSpec*	The Certificate object specification			
`metadata` *Metadata*	Refer to Kubernetes API documentation for fields of `metadata`.			

ClientCertConfiguration

ClientCertConfiguration contains the information to generate the certificate for the streaming_replica user

Appears in:

- TLSConfiguration

Field	Description	Required	Default	Validation
`caCertSecret` *string*	CACertSecret is the secret of the CA to be injected into the CloudNativePG ClientCASecret configuration	True		
`certManager` *CertManagerTemplate*	The cert-manager template used to generate the certificates			
`preProvisioned` *ClientPreProvisionedCertificates*	PreProvisioned contains how to fetch the pre-generated client certificates			

ClientPreProvisionedCertificates

ClientPreProvisionedCertificates instruct how to fetch the pre-generated client certificates

Appears in:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- ClientCertConfiguration

Field	Description	Required	Default	Validation
`streamingReplica` *PreProvisionedCertificate*	StreamingReplica the pre-generated certificate for 'streaming_replica' user			

ClusterImageCatalog

ClusterImageCatalog is the Schema for the clusterimagecatalogs API

Field	Description	Required	Default	Validation
`apiVersion` *string*	`pgd.k8s.enterprisedb.io/v1beta1`	True		
`kind` *string*	`ClusterImageCatalog`	True		
`metadata` *ObjectMeta*	Refer to Kubernetes API documentation for fields of `metadata`.	True		
`spec` *ImageCatalogSpec*	Specification of the desired behavior of the ClusterImageCatalog. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status	True		

ClusterStatus

ClusterStatus contains the current status of the CNP cluster

Appears in:

- CNPStatus

Field	Description	Required	Default	Validation
`name` *string*	Name is the name of the CNP cluster			
`phase` *string*	Phase is the current phase of the CNP cluster			

CnpBaseConfiguration

CnpBaseConfiguration contains the configuration parameters that can be applied to both CNP Witness and Data nodes

Appears in:

- CnpConfiguration
- PGDGroupSpec

Field	Description	Required	Default	Validation
`startDelay` *integer*	The time in seconds that is allowed for a PostgreSQL instance to successfully start up (default 3600)		3600	

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`stopDelay` *integer*	The time in seconds that is allowed for a PostgreSQL instance node to gracefully shutdown (default 180)		180	
`smartShutdownTimeout` *integer*	The time in seconds that controls the window of time reserved for the smart shutdown of Postgres to complete. Make sure you reserve enough time for the operator to request a fast shutdown of Postgres (that is: `stopDelay` - `smartShutdownTimeout`).		15	
`storage` *StorageConfiguration*	Configuration of the storage of the instances	True		
`walStorage` *StorageConfiguration*	Configuration of the WAL storage for the instances			
`clusterMaxStartDelay` *integer*	The time in seconds that is allowed for a PostgreSQL instance to successfully start up (default 300)		300	
`affinity` *AffinityConfiguration*	Affinity/Anti-affinity rules for Pods			
`resources` *ResourceRequirements*	Resources requirements of every generated Pod. Please refer to https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ for more information.			
`postgresql` *PostgresConfiguration*	Configuration of the PostgreSQL server			
`monitoring` *MonitoringConfiguration*	The configuration of the monitoring infrastructure of this cluster			
`logLevel` *string*	The instances' log level, one of the following values: error, warning, info (default), debug, trace		info	Enum: [error warning info debug trace]
`serviceAccountTemplate` *ServiceAccountTemplate*	The service account template to be passed to CNP			
`otel` *OTELConfiguration*	OpenTelemetry Configuration Deprecated: This field is deprecated and will be removed in future versions.			
`postInitSQL` *string array*	List of SQL queries to be executed as a superuser immediately after a node has been created - to be used with extreme care (by default empty)			
`postInitTemplateSQL` *string array*	List of SQL queries to be executed as a superuser in the `template1` after a node has been created - to be used with extreme care (by default empty)			
`postInitApplicationSQL` *string array*	List of SQL queries to be executed as a superuser in the application database right after the cluster has been created - to be used with extreme care as any data created in application database before logical join will cause the join to fail (by default empty)			
`postInitApplicationSQLRefs` *SQLRefs*	List of references to ConfigMaps or Secrets containing SQL files to be executed as a superuser in the application database right after the cluster has been created. The references are processed in a specific order: first, all Secrets are processed, followed by all ConfigMaps. Within each group, the processing order follows the sequence specified in their respective arrays. - to be used with extreme care (by default empty)			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`postInitTemplateSQLRefs` *SQLRefs*	List of references to ConfigMaps or Secrets containing SQL files to be executed as a superuser in the `template1` database right after the cluster has been created. The references are processed in a specific order: first, all Secrets are processed, followed by all ConfigMaps. Within each group, the processing order follows the sequence specified in their respective arrays. - to be used with extreme care (by default empty)			
`postInitSQLRefs` *SQLRefs*	List of references to ConfigMaps or Secrets containing SQL files to be executed as a superuser in the `postgres` database right after the cluster has been created. The references are processed in a specific order: first, all Secrets are processed, followed by all ConfigMaps. Within each group, the processing order follows the sequence specified in their respective arrays. - to be used with extreme care (by default empty)			
`seccompProfile` *SeccompProfile*	The SeccompProfile applied to every Pod and Container. Defaults to: `RuntimeDefault`			
`metadata` *InheritedMetadata*	Refer to Kubernetes API documentation for fields of `metadata`.			
`managed` *ManagedConfiguration*	The configuration that is used by the portions of PostgreSQL that are managed by the CNP instance manager			
`projectedVolumeTemplate` *ProjectedVolumeSource*	Template to be used to define projected volumes, projected volumes will be mounted under `/projected` base folder			
`tablespaces` *TablespaceConfiguration array*	The tablespaces configuration			
`topologySpreadConstraints` *TopologySpreadConstraint array*	TopologySpreadConstraints specifies how to spread matching pods among the given topology. More info: https://kubernetes.io/docs/concepts/scheduling-eviction/topology-spread-constraints/			
`plugins` *PluginConfiguration array*	The plugins configuration, containing any plugin to be loaded with the corresponding configuration			
`initDBOptions` *InitDBOptions*	InitDBOptions specifies the options to be passed to the `initdb` command when the node is created from scratch			
`env` *EnvVar array*	Env follows the Env format to pass environment variables to the pods created in the PG4K cluster			
`envFrom` *EnvFromSource array*	EnvFrom follows the EnvFrom format to pass environment variables sources to the pods to be used by Env			

CnpConfiguration

CnpConfiguration contains the configurations of the data nodes that will be injected into the resulting clusters composing the PGD group

Appears in:

- PGDGroupSpec

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`startDelay` *integer*	The time in seconds that is allowed for a PostgreSQL instance to successfully start up (default 3600)		3600	
`stopDelay` *integer*	The time in seconds that is allowed for a PostgreSQL instance node to gracefully shutdown (default 180)		180	
`smartShutdownTimeout` *integer*	The time in seconds that controls the window of time reserved for the smart shutdown of Postgres to complete. Make sure you reserve enough time for the operator to request a fast shutdown of Postgres (that is: `stopDelay` - `smartShutdownTimeout`).		15	
`storage` *StorageConfiguration*	Configuration of the storage of the instances	True		
`walStorage` *StorageConfiguration*	Configuration of the WAL storage for the instances			
`clusterMaxStartDelay` *integer*	The time in seconds that is allowed for a PostgreSQL instance to successfully start up (default 300)		300	
`affinity` *AffinityConfiguration*	Affinity/Anti-affinity rules for Pods			
`resources` *ResourceRequirements*	Resources requirements of every generated Pod. Please refer to https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ for more information.			
`postgresql` *PostgresConfiguration*	Configuration of the PostgreSQL server			
`monitoring` *MonitoringConfiguration*	The configuration of the monitoring infrastructure of this cluster			
`logLevel` *string*	The instances' log level, one of the following values: error, warning, info (default), debug, trace		info	Enum: [error warning info debug trace]
`serviceAccountTemplate` *ServiceAccountTemplate*	The service account template to be passed to CNP			
`otel` *OTELConfiguration*	OpenTelemetry Configuration Deprecated: This field is deprecated and will be removed in future versions.			
`postInitSQL` *string array*	List of SQL queries to be executed as a superuser immediately after a node has been created - to be used with extreme care (by default empty)			
`postInitTemplateSQL` *string array*	List of SQL queries to be executed as a superuser in the `template1` after a node has been created - to be used with extreme care (by default empty)			
`postInitApplicationSQL` *string array*	List of SQL queries to be executed as a superuser in the application database right after the cluster has been created - to be used with extreme care as any data created in application database before logical join will cause the join to fail (by default empty)			
`postInitApplicationSQLRefs` *SQLRefs*	List of references to ConfigMaps or Secrets containing SQL files to be executed as a superuser in the application database right after the cluster has been created. The references are processed in a specific order: first, all Secrets are processed, followed by all ConfigMaps. Within each group, the processing order follows the sequence specified in their respective arrays. - to be used with extreme care (by default empty)			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`postInitTemplateSQLRefs` *SQLRefs*	List of references to ConfigMaps or Secrets containing SQL files to be executed as a superuser in the `template1` database right after the cluster has been created. The references are processed in a specific order: first, all Secrets are processed, followed by all ConfigMaps. Within each group, the processing order follows the sequence specified in their respective arrays. - to be used with extreme care (by default empty)			
`postInitSQLRefs` *SQLRefs*	List of references to ConfigMaps or Secrets containing SQL files to be executed as a superuser in the `postgres` database right after the cluster has been created. The references are processed in a specific order: first, all Secrets are processed, followed by all ConfigMaps. Within each group, the processing order follows the sequence specified in their respective arrays. - to be used with extreme care (by default empty)			
`seccompProfile` *SeccompProfile*	The SeccompProfile applied to every Pod and Container. Defaults to: `RuntimeDefault`			
`metadata` *InheritedMetadata*	Refer to Kubernetes API documentation for fields of `metadata`.			
`managed` *ManagedConfiguration*	The configuration that is used by the portions of PostgreSQL that are managed by the CNP instance manager			
`projectedVolumeTemplate` *ProjectedVolumeSource*	Template to be used to define projected volumes, projected volumes will be mounted under `/projected` base folder			
`tablespaces` *TablespaceConfiguration array*	The tablespaces configuration			
`topologySpreadConstraints` *TopologySpreadConstraint array*	TopologySpreadConstraints specifies how to spread matching pods among the given topology. More info: https://kubernetes.io/docs/concepts/scheduling-eviction/topology-spread-constraints/			
`plugins` *PluginConfiguration array*	The plugins configuration, containing any plugin to be loaded with the corresponding configuration			
`initDBOptions` *InitDBOptions*	InitDBOptions specifies the options to be passed to the `initdb` command when the node is created from scratch			
`env` *EnvVar array*	Env follows the Env format to pass environment variables to the pods created in the PG4K cluster			
`envFrom` *EnvFromSource array*	EnvFrom follows the EnvFrom format to pass environment variables sources to the pods to be used by Env			
`joinMethod` *JoinMethod*	JoinMethod specifies the method data nodes will use to join the local group. PGD supports two ways of joining a local group: logical and physical. In logical join, the node will join the group by using `bdr.join_node_group`, synchronizing data from scratch. In physical join the node will join the group by using `bdr_init_physical`, synchronizing data from a physical backup.	True	logical	Enum: [logical physical]

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`enableSuperuserAccess` *boolean*	When this option is enabled, the CNP operator will create or use the secret defined in the SuperuserSecret to allow superuser (postgres) access to the database. When this option is disabled on a running Group, the operator will ignore the content of the secret and set the password of the `postgres` user to `NULL`. Enabled by default.		true	
`superuserSecret` *LocalObjectReference*	The secret containing the superuser password. A new secret will be created with a randomly generated password if not defined. This field is only allowed in the CNP Instances configuration. A Witness Node will always use the same SuperuserSecret as the other instances.			

ConnectionString

Underlying type: object

ConnectionString represent the parameters to connect to a PostgreSQL cluster

Appears in:

- PgdConfiguration

ConnectivityConfiguration

ConnectivityConfiguration describes how to generate the services and certificates for the PGDGroup

Appears in:

- PGDGroupSpec

Field	Description	Required	Default	Validation
`dns` *RootDNSConfiguration*	Describes how the FQDN for the resources should be generated			
`tls` *TLSConfiguration*	The configuration of the TLS infrastructure	True		
`nodeServiceTemplate` *ServiceTemplate*	Instructs how to generate the service for each node			
`groupServiceTemplate` *ServiceTemplate*	Instructs how to generate the service for the PGDGroup			
`proxyServiceTemplate` *ServiceTemplate*	Instructs how to generate the service pointing to the PGD Proxy for write leader node routing			
`proxyReadServiceTemplate` *ServiceTemplate*	Instructs how to generate the service pointing to the PGD Proxy for read nodes routing			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

ConnectivityStatus

ConnectivityStatus contains any relevant status for the operator about Connectivity

Appears in:

- PGDGroupStatus

Field	Description	Required	Default	Validation
`replicationTLSCertificate` *ReplicationCertificateStatus*	ReplicationTLSCertificate is the name of the replication TLS certificate, if we have it			
`nodeTLSCertificates` *NodeCertificateStatus array*	NodeTLSCertificates are the names of the certificates that have been created for the PGD nodes			
`unusedCertificates` *string array*	UnusedCertificates are the names of the certificates that we don't use anymore for the PGD nodes			
`nodesWithoutCertificates` *string array*	NodesWithoutCertificates are the names of the nodes which have not a server certificate			
`nodesNeedingServiceReconciliation` *string array*	NodesNeedingServiceReconciliation are the names of the nodes which have not a server certificate			
`configurationHash` *string*	ConfigurationHash is the hash code of the connectivity configuration, used to check if we had a change in the configuration or not			

DNSConfiguration

DNSConfiguration describes how the FQDN for the resources should be generated

Appears in:

- RootDNSConfiguration

Field	Description	Required	Default	Validation
`domain` *string*	Contains the domain name of by all services in the PGDGroup. It is responsibility of the user to ensure that the value specified here matches with the rendered nodeServiceTemplate and groupServiceTemplate			
`hostSuffix` *string*	Contains an optional suffix to add to all the service names in the PGDGroup. The meaning of this setting it to allow the user to easily mark all the services created in a location for routing purpose (i.e., add a generic rule to CoreDNS to rewrite some service suffixes as local)			

DiscoveryJobConfig

DiscoveryJobConfig contains a series of fields that configure the discovery job

Appears in:

- PgdConfiguration

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`delay` *integer*	Delay amount of time to sleep between retries, measured in seconds		10	
`retries` *integer*	Retries how many times the operation should be retried		30	
`timeout` *integer*	Timeout amount of time given to the operation to succeed, measured in seconds		300	

ImageCatalog

ImageCatalog is the Schema for the imagecatalogs API

Field	Description	Required	Default	Validation
`apiVersion` *string*	`pgd.k8s.enterprisedb.io/v1beta1`	True		
`kind` *string*	`ImageCatalog`	True		
`metadata` *ObjectMeta*	Refer to Kubernetes API documentation for fields of `metadata`.	True		
`spec` *ImageCatalogSpec*	Specification of the desired behavior of the ImageCatalog. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status	True		

ImageCatalogRef

ImageCatalogRef defines the referenced ImageCatalog and the referenced major Postgres version

Appears in:

- PGDGroupSpec

Field	Description	Required	Default	Validation
`apiGroup` *string*	APIGroup is the group for the resource being referenced. If APIGroup is not specified, the specified Kind must be in the core API group. For any other third-party types, APIGroup is required.			
`kind` *string*	Kind is the type of resource being referenced	True		
`name` *string*	Name is the name of resource being referenced	True		
`major` *integer*	The major version of PostgreSQL we want to use from the ImageCatalog	True		

ImageCatalogSpec

ImageCatalogSpec defines the desired ImageCatalog

Appears in:

- ClusterImageCatalog
- ImageCatalog

Field	Description	Required	Default	Validation

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`pgdImages` *CatalogImage array*	List of CatalogImages available in the catalog	True		MaxItems: 8 MinItems: 1
`proxyImage` *string*	The proxy image available in the catalog			

ImageStatus

ImageStatus defines the current used images

Appears in:

- PGDGroupStatus

Field	Description	Required	Default	Validation
`pgd` *string*	PGD is the current used pgd image			
`proxy` *string*	PGDProxy is the current used pgd proxy image			

InheritedMetadata

InheritedMetadata contains metadata to be inherited by all resources related to a Cluster

Appears in:

- CnpBaseConfiguration
- CnpConfiguration
- PGDGroupSpec

Field	Description	Required	Default	Validation
`labels` *object (keys:string, values:string)*				
`annotations` *object (keys:string, values:string)*				

InitDBOptions

InitDBOptions contain options for nodes start with initDB

Appears in:

- CnpBaseConfiguration
- CnpConfiguration

Field	Description	Required	Default	Validation
`dataChecksums` *boolean*	Whether the `-k` option should be passed to initdb, enabling checksums on data pages (default: `false`)			
`encoding` *string*	The value to be passed as option `--encoding` for initdb (default: `UTF8`)			
`localeCollate` *string*	The value to be passed as option `--lc-collate` for initdb (default: `C`)			
`localeCType` *string*	The value to be passed as option `--lc-ctype` for initdb (default: `C`)			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`locale` *string*	Sets the default collation order and character classification in the new database.			
`localeProvider` *string*	This option sets the locale provider for databases created in the new cluster. Available from PostgreSQL 16.			
`icuLocale` *string*	Specifies the ICU locale when the ICU provider is used. This option requires `localeProvider` to be set to `icu`. Available from PostgreSQL 15.			
`icuRules` *string*	Specifies additional collation rules to customize the behavior of the default collation. This option requires `localeProvider` to be set to `icu`. Available from PostgreSQL 16.			
`builtinLocale` *string*	Specifies the locale name when the builtin provider is used. This option requires `localeProvider` to be set to `builtin`. Available from PostgreSQL 17.			
`walSegmentSize` *integer*	The value in megabytes (1 to 1024) to be passed to the `--wal-segsize` option for initdb (default: empty, resulting in PostgreSQL default: 16MB)			Maximum: 1024 Minimum: 1

JoinMethod

Underlying type: string

JoinMethod represents one of the supported methods of joining a node

Appears in:

- CnpConfiguration
- PgdConfiguration

Field	Description
`physical`	JoinMethodPhysical represents the use of physical join
`logical`	JoinMethodLogical represents the use of logical join

Metadata

Metadata is a structure similar to the metav1.ObjectMeta, but still parseable by controller-gen to create a suitable CRD for the user.

Appears in:

- CertManagerTemplate
- ServiceTemplate

Field	Description	Required	Default	Validation
`labels` *object (keys:string, values:string)*	Map of string keys and values that can be used to organize and categorize (scope and select) objects. May match selectors of replication controllers and services. More info: http://kubernetes.io/docs/user-guide/labels			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`annotations` *object (keys:string, values:string)*	Annotations is an unstructured key value map stored with a resource that may be set by external tools to store and retrieve arbitrary metadata. They are not queryable and should be preserved when modifying objects. More info: http://kubernetes.io/docs/user-guide/annotations			

NodeCertificateStatus

NodeCertificateStatus encapsulate the status of the server certificate of a CNP node

Appears in:

- ConnectivityStatus

Field	Description	Required	Default	Validation
`name` *string*	Name is the name of the certificate			
`hash` *string*	Hash is the hash of the configuration for which it has been generated			
`isReady` *boolean*	Ready is true when the certificate is ready			
`preProvisioned` *boolean*	PreProvisioned is true if the certificate is preProvisioned			
`nodeName` *string*	NodeName is the name of the CNP cluster using this certificate	True		

NodeKindName

Underlying type: string

NodeKindName is a type containing the potential values of node_kind_name from bdr.node_summary

Appears in:

- NodeSummary

Field	Description
`witness`	
`data`	

NodeSummary

NodeSummary shows relevant info from bdr.node_summary

Appears in:

- PGDGroupStatus

Field	Description	Required	Default	Validation
`node_name` *string*	Name of the node			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`node_group_name` *string*	NodeGroupName is the name of the joined group			
`peer_state_name` *string*	Consistent state of the node in human-readable form	True		
`peer_target_state_name` *string*	State which the node is trying to reach (during join or promotion)			
`node_kind_name` *NodeKindName*	The kind of node: witness or data			

OTELConfiguration

OTELConfiguration is the configuration for external openTelemetry

Deprecated: This field is deprecated and will be removed in future versions.

Appears in:

- CnpBaseConfiguration
- CnpConfiguration

Field	Description	Required	Default	Validation
`metricsURL` *string*	The OpenTelemetry HTTP endpoint URL to accept metrics data Deprecated: This field is deprecated and will be removed in future versions.			
`traceURL` *string*	The OpenTelemetry HTTP endpoint URL to accept trace data Deprecated: This field is deprecated and will be removed in future versions.			
`traceEnable` *boolean*	Whether to push trace data to OpenTelemetry traceUrl Deprecated: This field is deprecated and will be removed in future versions.			
`tls` *OTELTLSConfiguration*	TLSConfiguration provides the TLS certificate configuration when MetricsURL and TraceURL are using HTTPS Deprecated: This field is deprecated and will be removed in future versions.			

OTELTLSConfiguration

OTELTLSConfiguration contains the certificate configuration for TLS connections to openTelemetry

Deprecated: This field is deprecated and will be removed in future versions.

Appears in:

- OTELConfiguration

Field	Description	Required	Default	Validation
`caBundleSecretRef` *SecretKeySelector*	CABundleSecretRef is a reference to a secret field containing the CA bundle to verify the openTelemetry server certificate Deprecated: This field is deprecated and will be removed in future versions.			
`clientCertSecret` *LocalObjectReference*	ClientCertSecret is the name of the secret containing the client certificate used to connect to openTelemetry. It must contain both the standard "tls.crt" and "tls.key" files, encoded in PEM format. Deprecated: This field is deprecated and will be removed in future versions.			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

ObjectStoreStatus

ObjectStoreStatus contains the name and hash of the objectStore

Appears in:

- BarmanCloudPluginStatus

Field	Description	Required	Default	Validation
`name` *string*	Name is the name of the objectStore			
`hash` *string*	Hash stored the objectStore hashcode	True		

OperatorPhase

Underlying type: string

OperatorPhase it represents a phase of the PGDGroup controller

Appears in:

- PGDGroupStatus

Field	Description
`PGDGroup - Healthy`	PhaseHealthy triggers when no other phase condition is met
`PGDGroup - Paused`	PhasePaused triggers when the PGDGroup is paused and all instances and proxies pod are deleted
`PGDGroup - Waiting for nodes to be ready`	PhaseWaitingForNodesToBeReady triggers when a PGDGroup node is not in status ready
`PGDGroup - Waiting for nodes major version in-place upgrade`	PhaseWaitingForNodesInPlaceUpgrade is the phase where a in-place major version upgrade (postgres, bdr, etc) is happening
`PGDGroup - Unable to detect a phase for the PGD group`	PhaseUnableToDetectPhase triggers when a PGDGroup phase cannot be calculated. This is a programmatic error
`Default Metrics - Reconcile Resources`	PhaseDefaultMetricsReconcileResources triggers when resources need to be created or updated

OperatorPhaseCleanup

Underlying type: string

OperatorPhaseCleanup it represents a phase of the PGDGroupCleanup controller

Appears in:

- PGDGroupCleanupStatus

Field	Description

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description
`PGDGroupCleanup - Failed, no reliable node found in the executor PGDGroup`	NoExecutorFound the phase when the executor PGDGroup cannot be found or has no reliable node, which is required for the cleanup process, and the cleanup cannot proceed. This phase indicates a failure state for the PGDGroupCleanup operation. This is a terminal phase, and the PGDGroupCleanup will not be requeued anymore when it is in this phase.
`PGDGroupCleanup - Completed`	Completed the phase when the cleanup process has been completed successfully This is a terminal phase, and the PGDGroupCleanup will not be requeued anymore when it is in this phase.
`PGDGroupCleanup - Parting the nodes`	PartingNodes the phase when the cleanup process is parting the nodes in the target PGDGroup
`PGDGroupCleanup - Removing the PGDGroup`	RemovingGroup the phase when the cleanup process is removing the PGDGroup after all the nodes in the target PGDGroup are parted
`PGDGroupCleanup - Waiting for nodes in target PGDGroup to be parted`	TargetNotParted the phase when the cleanup process is waiting for the nodes in the target PGDGroup to be parted, which is required before removing the PGDGroup. This phase happens when force flag in PGDGroupCleanup is false and waiting for nodes to be parted.

PGDGroup

PGDGroup is the Schema for the pgdgroups API

Field	Description	Required	Default	Validation
`apiVersion` *string*	`pgd.k8s.enterprisedb.io/v1beta1`	True		
`kind` *string*	`PGDGroup`	True		
`metadata` *ObjectMeta*	Refer to Kubernetes API documentation for fields of `metadata`.	True		
`spec` *PGDGroupSpec*		True		
`status` *PGDGroupStatus*				

PGDGroupCleanup

PGDGroupCleanup is the Schema for the pgdgroupcleanups API

Field	Description	Required	Default	Validation
`apiVersion` *string*	`pgd.k8s.enterprisedb.io/v1beta1`	True		
`kind` *string*	`PGDGroupCleanup`	True		
`metadata` *ObjectMeta*	Refer to Kubernetes API documentation for fields of `metadata`.	True		
`spec` *PGDGroupCleanupSpec*		True		
`status` *PGDGroupCleanupStatus*		True		

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

PGDGroupCleanupSpec

PGDGroupCleanupSpec defines the desired state of PGDGroupCleanup

Appears in:

- PGDGroupCleanup

Field	Description	Required	Default	Validation
`executor` *string*		True		
`target` *string*		True		
`force` *boolean*	Force will force the removal of the PGDGroup even if the target PGDGroup nodes are not parted		false	

PGDGroupCleanupStatus

PGDGroupCleanupStatus defines the observed state of PGDGroupCleanup

Appears in:

- PGDGroupCleanup

Field	Description	Required	Default	Validation
`phase` *OperatorPhaseCleanup*	Phase the phase of current cleanup cr			
`nodesToPart` *string array*	NodesToPart shows nodes that are not parted in the target group			
`isNodeGroupExists` *boolean*	IsNodeGroupExists indicates if the target group is still not dropped yet			

PGDGroupSpec

PGDGroupSpec defines the desired state of PGDGroup

Appears in:

- PGDGroup

Field	Description	Required	Default	Validation
`imageName` *string*	Name of the container image, supporting both tags (`<image>:<tag>`) and digests for deterministic and repeatable deployments (`<image>:<tag>@sha256:<digestValue>`)			
`imageCatalogRef` *ImageCatalogRef*	Defines the referenced ImageCatalog and the referenced major Postgres version			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`imagePullPolicy` *PullPolicy*	Image pull policy. One of `Always`, `Never` or `IfNotPresent`. If not defined, it defaults to `IfNotPresent`. Cannot be updated. More info: https://kubernetes.io/docs/concepts/containers/images#updating-images		IfNotPresent	
`imagePullSecrets` *LocalObjectReference array*	The list of pull secrets to be used to pull operator and or the operand images			
`inheritedMetadata` *InheritedMetadata*	Metadata that will be inherited by all objects related to the pgdGroup	True		
`instances` *integer*	Number of instances required in the cluster	True		Minimum: 0
`proxyInstances` *integer*	Number of proxy instances required in the cluster			Minimum: 0
`witnessInstances` *integer*	Number of witness instances required in the cluster			Minimum: 0
`backup` *Backup*	The configuration to be used for backups in the CNP instances.			
`restore` *Restore*	The configuration to restore this PGD group from an Object Store service			
`cnp` *CnpConfiguration*	DataInstances configuration that will be injected into the CNP clusters that compose the PGD Group	True		
`witness` *CnpBaseConfiguration*	WitnessInstances configuration that will be injected into the WitnessInstances CNP clusters If not defined, it will default to the DataInstances configuration			
`pgd` *PgdConfiguration*	Pgd contains instructions to bootstrap this cluster	True		
`pgdProxy` *PGDProxyConfiguration*	PGDProxy contains instructions to configure PGD Proxy			
`connectivity` *ConnectivityConfiguration*	Configures the connectivity of the PGDGroup, like services and certificates that will be used.	True		
`failingFinalizerTimeLimitSeconds` *integer*	The amount of seconds that the operator will wait in case of a failing finalizer. A finalizer is considered failing when the operator cannot reach any nodes of the PGDGroup		300	

PGDGroupStatus

PGDGroupStatus defines the observed state of PGDGroup

Appears in:

- PGDGroup

Field	Description	Required	Default	Validation
`latestGeneratedNode` *integer*	ID of the latest generated node (used to avoid node name clashing)			
`phase` *OperatorPhase*	The initialization phase of this cluster			
`phaseDetails` *string*	The details of the current phase			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`phaseTroubleshootHints` *string*	PhaseTroubleshootHints general troubleshooting indications for the given phase			
`phaseType` *PhaseType*	PhaseType describes the phase category.			Enum: [action wait fail]
`conditions` *Condition array*	Conditions for PGDGroup object			
`nodes` *NodeSummary array*	The list of summaries for the nodes in the group			
`backup` *BackupStatus*	The node that is taking backups of this PGDGroup			
`restore` *RestoreStatus*	The status of the restore process			
`PGD` *PGDStatus*	Last known status of PGD			
`CNP` *CNPStatus*	Last known status of CNP			
`PGDProxy` *PGDProxyStatus*	Last known status of PGDProxy			
`connectivity` *ConnectivityStatus*	Last known status of Connectivity			
`pause` *PauseStatus*	Last known status of Pause			
`image` *ImageStatus*	Last known status of used image			
`plugins` *PluginStatus*	Plugin Last known status of the plugins			

PGDNodeGroupEntry

PGDNodeGroupEntry shows information about the node groups available in the PGD configuration

Appears in:

- PGDStatus

Field	Description	Required	Default	Validation
`name` *string*	Name is the name of the node group	True		
`uuid` *string*	UUID is the UUID of the group			
`enableProxyRouting` *boolean*	EnableProxyRouting is true is the node group allows running PGD Proxies			
`enableRaft` *boolean*	EnableRaft is true if the node group has a subgroup raft instance			
`routeWriterMaxLag` *integer*	RouteWriterMaxLag Maximum lag in bytes of the new write candidate to be selected as write leader, if no candidate passes this, there will be no writer selected automatically			
`routeReaderMaxLag` *integer*	RouteReaderMaxLag Maximum lag in bytes for node to be considered viable read-only node			
`routeWriterWaitFlush` *boolean*	RouteWriterWaitFlush Whether to wait for replication queue flush before switching to new leader when using `bdr.routing_leadership_transfer()`			

PGDNodeGroupSettings

PGDNodeGroupSettings contains the settings of the PGD Group

Appears in:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- PgdConfiguration

Field	Description	Required	Default	Validation
`routeWriterMaxLag` *integer*	RouteWriterMaxLag Maximum lag in bytes of the new write candidate to be selected as write leader, if no candidate passes this, there will be no writer selected automatically Defaults to -1		-1	
`routeReaderMaxLag` *integer*	RouteReaderMaxLag Maximum lag in bytes for node to be considered viable read-only node Defaults to -1		-1	
`routeWriterWaitFlush` *boolean*	RouteWriterWaitFlush Whether to wait for replication queue flush before switching to new leader when using `bdr.routing_leadership_transfer()` Defaults to false		false	

PGDProxyConfiguration

PGDProxyConfiguration defines the configuration of PGD Proxy

Appears in:

- PGDGroupSpec

Field	Description	Required	Default	Validation
`imageName` *string*	Name of the PGDProxy container image			
`logLevel` *string*	The PGD Proxy log level, one of the following values: error, warning, info (default), debug, trace		info	Enum: [error warning info debug trace]
`logEncoder` *string*	The format of the log output		json	Enum: [text json]
`proxyAffinity` *Affinity*	ProxyAffinity/Anti-affinity rules for pods			
`proxyNodeSelector` *object (keys:string, values:string)*	ProxyNodeSelector rules for pods			
`proxyTolerations` *Toleration array*	ProxyTolerations rules for pods			
`proxyResources` *ResourceRequirements*	Defines the resources assigned to the proxy. If not defined uses defaults requests and limits values.			

PGDProxyEntry

PGDProxyEntry shows information about the proxies available in the PGD configuration

Appears in:

- PGDStatus

Field	Description	Required	Default	Validation
`name` *string*	Name is the name of the proxy	True		

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`fallbackGroupNames` *string array*	FallbackGroupNames are the names of the fallback groups configured for this proxy			
`parentGroupName` *string*	ParentGroupName is the parent PGD group of this proxy			
`maxClientConn` *integer*	MaxClientConn maximum number of connections the proxy will accept			
`maxServerConn` *integer*	MaxServerConn maximum number of connections the proxy will make to the Postgres node			
`serverConnTimeout` *integer*	ServerConnTimeout connection timeout for server connections in seconds			
`serverConnKeepalive` *integer*	ServerConnKeepalive keepalive interval for server connections in seconds			
`fallbackGroupTimeout` *integer*	FallbackGroupTimeout the interval after which the routing falls back to one of the fallback_groups			
`consensusGracePeriod` *integer*	ConsensusGracePeriod the duration in seconds for which proxy continues to route even upon loss of a Raft leader.			
`readListenPort` *integer*	ReadListenPort is the port where the proxy will listen and route queries to read nodes			

PGDProxySettings

PGDProxySettings contains the settings of the proxy

Appears in:

- PgdConfiguration

Field	Description	Required	Default	Validation
`fallbackGroups` *string array*	FallbackGroups is the list of groups the proxy should forward connection to when all the data nodes of this PGD group are not available			
`maxClientConn` *integer*	MaxClientConn maximum number of connections the proxy will accept. Defaults to 32767		32767	
`maxServerConn` *integer*	MaxServerConn maximum number of connections the proxy will make to the Postgres node. Defaults to 32767		32767	
`serverConnTimeout` *integer*	ServerConnTimeout connection timeout for server connections in seconds. Defaults to 2		2	
`serverConnKeepalive` *integer*	ServerConnKeepalive keepalive interval for server connections in seconds. Defaults to 10		10	
`fallbackGroupTimeout` *integer*	FallbackGroupTimeout the interval after which the routing falls back to one of the fallback_groups. Defaults to 60		60	
`consensusGracePeriod` *integer*	ConsensusGracePeriod the duration in seconds for which proxy continues to route even upon loss of a Raft leader. If set to 0s, proxy stops routing immediately. Defaults to 6		6	
`enableReadNodeRouting` *boolean*	EnableReadNodeRouting is the switch to control whether the proxy will route queries to read nodes through read_listen_port or not. By default it is false		false	

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

PGDProxyStatus

PGDProxyStatus any relevant status for the operator about PGDProxy

Appears in:

- PGDGroupStatus

Field	Description	Required	Default	Validation
`proxyInstances` *integer*				
`writeLead` *string*	WriteLead is a reserved field for the operator, is not intended for external usage. Will be removed in future versions			
`proxyHash` *string*	ProxyHash contains the hash we use to detect if we need to reconcile the proxies			

PGDRaftStatus

Underlying type: string

PGDRaftStatus indicates a known status of the PGDRaft

Appears in:

- PGDStatus

Field	Description
`OK`	PGDRaftConsensusStatusOk indicates that the raft consensus is working correctly
`CRITICAL`	PGDRaftConsensusStatusCritical indicates that the raft consensus is not ready
`UNKNOWN`	PGDRaftConsensusStatusUnknown indicates that the raft consensus status is unknown
`WARNING`	PGDRaftConsensusStatusWarning is a warning that an election may be in progress

PGDStatus

PGDStatus any relevant status for the operator about PGD

Appears in:

- PGDGroupStatus

Field	Description	Required	Default	Validation
`raftConsensusLastChangedStatus` *PGDRaftStatus*	RaftConsensusLastChangedStatus indicates the latest reported status from bdr.monitor_group_raft			
`raftConsensusLastChangedMessage` *string*	RaftConsensusLastChangedMessage indicates the latest reported message from bdr.monitor_group_raft	True		
`raftConsensusLastChangedTimestamp` *string*	RaftConsensusLastChangedTimestamp indicates when the status and message were first reported			
`registeredProxies` *PGDProxyEntry array*	RegisteredProxies is the status of the registered proxies			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`nodeGroup` *PGDNodeGroupEntry*	NodeGroup is the status of the node group associated with the PGDGroup			
`globalNodeGroup` *PGDNodeGroupEntry*	GlobalNodeGroup is the status of the global node group associated with the PGDGroup			
`extensionVersion` *string*	ExtensionVersion is the pgd extension version detected on the raft lead node			

ParentGroupConfiguration

ParentGroupConfiguration contains the topology configuration of PGD

Appears in:

- PgdConfiguration

Field	Description	Required	Default	Validation
`name` *string*	Name of the parent group	True		MinLength: 1
`create` *boolean*	Create is true when the operator should create the parent group if it doesn't exist		false	

PauseStatus

PauseStatus contains the information of group hibernating

Appears in:

- PGDGroupStatus

Field	Description	Required	Default	Validation
`active` *boolean*	Active indicates the PGDGroup is either: - in process of pausing - already paused - in process of resuming			
`instances` *integer*	Instances is the number of paused PGD instances			
`lastStartedTime` *Time*	LastStartedTime is the last time the PGDGroup started pausing			
`lastCompletedTime` *Time*	LastCompletedTime is last time the PGDGroup completed pausing			
`lastResumeStartedTime` *Time*	LastResumeStartedTime is the last time the PGDGroup started resuming			
`lastResumeCompletedTime` *Time*	LastCompletedTime is last time the PGDGroup completed resuming			

PgdConfiguration

PgdConfiguration is the configuration of the PGD group structure

Appears in:

- PGDGroupSpec

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`parentGroup` *ParentGroupConfiguration*	ParentGroup configures the topology of the PGD group	True		
`groupJoinMethod` *JoinMethod*	GroupJoinMethod defines the method in case of cross region join		logical	Enum: [logical physical]
`discovery` *ConnectionString array*	The parameters we will use to connect to a node belonging to the parent PGD group. Even if provided, the following parameters will be overridden with default values: `application_name`, `sslmode`, `dbname` and `user`. The following parameters should not be provided nor used, as they are not even overridden with defaults: `sslkey`, `sslcert`, `sslrootcert`			
`discoveryJob` *DiscoveryJobConfig*	DiscoveryJob the configuration of the PGD Discovery job		{ delay:10 retries:30 timeout:300 }	
`databaseName` *string*	Name of the database used by the application. Default: `app`.			
`ownerName` *string*	Name of the owner of the database in the instance to be used by applications. Defaults to the value of the `database` key.			
`ownerCredentialsSecret` *LocalObjectReference*	Name of the secret containing the initial credentials for the owner of the user database. If empty a new secret will be created from scratch			
`proxySettings` *PGDProxySettings*	Configuration for the proxy		{ enableReadNodeRouting:false fallbackGroupTimeout:60 maxClientConn:32767 maxServerConn:32767 serverConnKeepalive:10 serverConnTimeout:2 }	
`nodeGroupSettings` *PGDNodeGroupSettings*	Configuration for the PGD Group			
`globalRouting` *boolean*	GlobalRouting is true when global routing is enabled, and in this case the proxies will be created in the parent group		false	
`mutations` *SQLMutations*	List of SQL mutations to apply to the node group			

PhaseType

Underlying type: string

PhaseType describes the type of the OperatorPhase

Appears in:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- PGDGroupStatus

Field	Description
`action`	PhaseTypeAction triggers a reconciliation for a pkg
`wait`	PhaseTypeWait the phase is waiting for an external resource to complete an action
`fail`	PhaseTypeFail the operator is failing, most likely requires a manual intervention

PluginStatus

PluginStatus contains the status of the plugins and managed plugins

Appears in:

- PGDGroupStatus

Field	Description	Required	Default	Validation
`barmanCloud` *BarmanCloudPluginStatus*	BarmanCloud stored the barman cloud plugin status			

PreProvisionedCertificate

PreProvisionedCertificate contains the data needed to supply a pre-generated certificate

Appears in:

- ClientPreProvisionedCertificates

Field	Description	Required	Default	Validation
`secretRef` *string*	SecretRef a name pointing to a secret that contains a tls.crt and tls.key			

RecoverabilityPointsByMethod

Underlying type: Time

RecoverabilityPointsByMethod contains the first recoverability points for a given backup method

Appears in:

- CNPStatus

ReplicationCertificateStatus

ReplicationCertificateStatus encapsulate the certificate status

Appears in:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- ConnectivityStatus
- NodeCertificateStatus

Field	Description	Required	Default	Validation
`name` *string*	Name is the name of the certificate			
`hash` *string*	Hash is the hash of the configuration for which it has been generated			
`isReady` *boolean*	Ready is true when the certificate is ready			
`preProvisioned` *boolean*	PreProvisioned is true if the certificate is preProvisioned			

Restore

Restore configures the restore of a PGD group from an object store

Appears in:

- PGDGroupSpec

Field	Description	Required	Default	Validation
`volumeSnapshots` *VolumeSnapshotsConfiguration*	The configuration for volumeSnapshot restore			
`barmanObjectStore` *BarmanObjectStoreConfiguration*	The configuration for the barman-cloud tool suite			
`recoveryTarget` *RecoveryTarget*	By default, the recovery process applies all the available WAL files in the archive (full recovery). However, you can also end the recovery as soon as a consistent state is reached or recover to a point-in-time (PITR) by specifying a `RecoveryTarget` object, as expected by PostgreSQL (i.e., timestamp, transaction Id, LSN, ...). More info: https://www.postgresql.org/docs/current/runtime-config-wal.html#RUNTIME-CONFIG-WAL-RECOVERY-TARGET			
`serverNames` *string array*	The list of server names to be used as a recovery origin. One of these servers will be elected as the seeding one when evaluating the recovery target, this option is only used when restore from barmanObjectStore.			

RestoreStatus

RestoreStatus contains the current status of the restore process

Appears in:

- PGDGroupStatus

Field	Description	Required	Default	Validation
`serverName` *string*	The name of the server to be restored	True		
`VolumeSnapshots` *VolumeSnapshotRestoreStatus array*	selected volumeSnapshots to restore			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

RootDNSConfiguration

RootDNSConfiguration describes how the FQDN for the resources should be generated

Appears in:

- ConnectivityConfiguration

Field	Description	Required	Default	Validation
`domain` *string*	Contains the domain name of by all services in the PGDGroup. It is responsibility of the user to ensure that the value specified here matches with the rendered nodeServiceTemplate and groupServiceTemplate			
`hostSuffix` *string*	Contains an optional suffix to add to all the service names in the PGDGroup. The meaning of this setting it to allow the user to easily mark all the services created in a location for routing purpose (i.e., add a generic rule to CoreDNS to rewrite some service suffixes as local)			
`additional` *DNSConfiguration array*	AdditionalDNSConfigurations adds more possible FQDNs for the resources			

SQLMutation

SQLMutation is a series of SQL statements to apply atomically

Appears in:

- SQLMutations

Field	Description	Required	Default	Validation
`isApplied` *string array*	List of boolean-returning SQL queries. If any of them returns false the mutation will be applied	True		MinItems: 1
`exec` *string array*	List of SQL queries to be executed to apply this mutation	True		MinItems: 1
`type` *SQLMutationType*	Type determines when the SQLMutation occurs. 'always': reconcile the mutation at each reconciliation cycle 'beforeSubgroupRaft': are executed only before the subgroupRaft is enabled If not specified, the Type defaults to 'always'.		always	Enum: [always beforeSubgroupRaft writeLeader]

SQLMutationType

Underlying type: string

SQLMutationType a supported type of SQL Mutation

Appears in:

- SQLMutation

Field	Description
`always`	SQLMutationTypeAlways tries to reconcile the mutation at each reconciliation cycle

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description
`beforeSubgroupRaft`	SQLMutationTypeBeforeSubgroupRaft are executed only before the subgroupRaft is enabled
`writeLeader`	SQLMutationTypeWriteLeader are executed only after raft Consensus and execute on write leader

SQLMutations

Underlying type: SQLMutation

SQLMutations A list of SQLMutation

Appears in:

- PgdConfiguration

Field	Description	Required	Default	Validation
`isApplied` *string array*	List of boolean-returning SQL queries. If any of them returns false the mutation will be applied	True		MinItems: 1
`exec` *string array*	List of SQL queries to be executed to apply this mutation	True		MinItems: 1
`type` *SQLMutationType*	Type determines when the SQLMutation occurs. 'always': reconcile the mutation at each reconciliation cycle 'beforeSubgroupRaft': are executed only before the subgroupRaft is enabled If not specified, the Type defaults to 'always'.		always	Enum: [always beforeSubgroupRaft writeLeader]

ScheduledBackupSpec

ScheduledBackupSpec defines the desired state of ScheduledBackup

Appears in:

- Backup

Field	Description	Required	Default	Validation
`suspend` *boolean*	If this backup is suspended or not			
`immediate` *boolean*	If the first backup has to be immediately start after creation or not			
`schedule` *string*	The schedule does not follow the same format used in Kubernetes CronJobs as it includes an additional second specifier, see https://pkg.go.dev/github.com/robfig/cron#hdr-CRON_Expression_Format	True		
`backupOwnerReference` *string*	Indicates which ownerReference should be put inside the created backup resources. - none: no owner reference for created backup objects (same behavior as before the field was introduced) - self: sets the Scheduled backup object as owner of the backup - cluster: set the cluster as owner of the backup		none	Enum: [none self cluster]

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`target` *BackupTarget*	The policy to decide which instance should perform this backup. If empty, it defaults to `cluster.spec.backup.target`. Available options are empty string, `primary` and `prefer-standby`. `primary` to have backups run always on primary instances, `prefer-standby` to have backups run preferably on the most updated standby, if available.			Enum: [primary prefer-standby]
`method` *BackupMethod*	The backup method to be used, possible options are `barmanObjectStore`, `volumeSnapshot` and `plugin`. Defaults to: `barmanObjectStore`.		barmanObjectStore	Enum: [barmanObjectStore volumeSnapshot plugin]
`pluginConfiguration` *BackupPluginConfiguration*	Configuration parameters passed to the plugin managing this backup			
`online` *boolean*	Whether the default type of backup with volume snapshots is online/hot (`true`, default) or offline/cold (`false`) Overrides the default setting specified in the cluster field '.spec.backup.volumeSnapshot.online'			
`onlineConfiguration` *OnlineConfiguration*	Configuration parameters to control the online/hot backup with volume snapshots Overrides the default settings specified in the cluster '.backup.volumeSnapshot.onlineConfiguration' stanza			

ScheduledBackupStatus

ScheduledBackupStatus contains the status of the scheduled backup

Appears in:

- BackupStatus

Field	Description	Required	Default	Validation
`method` *BackupMethod*	Method is the backup method used to take the backup			
`name` *string*	Name is the name of the scheduled backup			
`hash` *string*	Hash is the hash of the scheduled backup configuration			

ServerCertConfiguration

ServerCertConfiguration contains the information to generate the certificates for the nodes

Appears in:

- TLSConfiguration

Field	Description	Required	Default	Validation
`caCertSecret` *string*	CACertSecret is the secret of the CA to be injected into the CloudNativePG ServerCASecret configuration	True		

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`certManager` *CertManagerTemplate*	The cert-manager template used to generate the certificates	True		

ServiceTemplate

ServiceTemplate is a structure that allows the user to set a template for the Service generation.

Appears in:

- ConnectivityConfiguration

Field	Description	Required	Default	Validation
`metadata` *Metadata*	Refer to Kubernetes API documentation for fields of `metadata`.			
`spec` *ServiceSpec*	Specification of the desired behavior of the service. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status			
`updateStrategy` *ServiceUpdateStrategy*	UpdateStrategy indicates how to update the services generated by this template.		patch	Enum: [patch replace]

ServiceUpdateStrategy

Underlying type: string

ServiceUpdateStrategy defines the type for updating LoadBalancers. Allowed values are "patch" and "replace".

Appears in:

- ServiceTemplate

Field	Description
`patch`	ServiceUpdateStrategyPatch indicates to update existing services with a PATCH request
`replace`	ServiceUpdateStrategyReplace indicates to update existing LoadBalancer services by recreating the service ServiceUpdateStrategyReplace should be used for LoadBalancer services created in AWS.

TLSConfiguration

TLSConfiguration is the configuration of the TLS infrastructure used by PGD to connect to the nodes

Appears in:

- ConnectivityConfiguration

Field	Description	Required	Default	Validation
`mode` *TLSMode*		True	verify-ca	Enum: [verify-ca verify-full require]

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`serverCert` *ServerCertConfiguration*	The configuration for the server certificates	True		
`clientCert` *ClientCertConfiguration*	The configuration for the client certificates	True		

TLSMode

Underlying type: string

TLSMode describes which mode should be used for the node to node communications

Appears in:

- TLSConfiguration

Field	Description
`verify-full`	VerifyFull verifies that the certificate is signed by the passed CA and the server host name matches the certificate
`verify-ca`	VerifyCA verifies that the certificate is signed by the passed CA
`require`	Require requires TLS but do not perform any certificate validation

VolumeSnapshotRestoreStatus

VolumeSnapshotRestoreStatus the volumeSnapshot to restore

Appears in:

- RestoreStatus

Field	Description	Required	Default	Validation
`snapshotName` *string*	SnapshotName is the snapshot name to restore			
`pvcRole` *PVCRole*	PVCRole is the pvcRole snapshot to restore			
`tablespaceName` *string*	TablespaceName is the tablespace name the snapshot belongs to, if the pvcRole is `PG_TABLESPACE`			

VolumeSnapshotsConfiguration

VolumeSnapshotsConfiguration contains the configuration for the volumeSnapshots restore

Appears in:

- Restore

Field	Description	Required	Default	Validation
`selector` *LabelSelector*	Label selector used to select the volumeSnapshot to restore	True		

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

CertificateKeystores

CertificateKeystores configures additional keystore output formats to be created in the Certificate's output Secret.

Appears in:

- CertificateSpec

Field	Description	Required	Default	Validation
`jks` *JKSKeystore*	JKS configures options for storing a JKS keystore in the `spec.secretName` Secret resource.			
`pkcs12` *PKCS12Keystore*	PKCS12 configures options for storing a PKCS12 keystore in the `spec.secretName` Secret resource.			

CertificateList

CertificateList is a list of Certificates

Field	Description	Required	Default	Validation
`apiVersion` *string*	`cert-manager.io/v1`	True		
`kind` *string*	`CertificateList`	True		
`metadata` *ListMeta*	Refer to Kubernetes API documentation for fields of `metadata`.	True		

CertificatePrivateKey

CertificatePrivateKey contains configuration options for private keys used by the Certificate controller. This allows control of how private keys are rotated.

Appears in:

- CertificateSpec

Field	Description	Required	Default	Validation
`rotationPolicy` *PrivateKeyRotationPolicy*	RotationPolicy controls how private keys should be regenerated when a re-issuance is being processed. If set to Never, a private key will only be generated if one does not already exist in the target `spec.secretName`. If one does exists but it does not have the correct algorithm or size, a warning will be raised to await user intervention. If set to Always, a private key matching the specified requirements will be generated whenever a re-issuance occurs. Default is 'Never' for backward compatibility.			
`encoding` *PrivateKeyEncoding*	The private key cryptography standards (PKCS) encoding for this certificate's private key to be encoded in. If provided, allowed values are `PKCS1` and `PKCS8` standing for PKCS#1 and PKCS#8, respectively. Defaults to `PKCS1` if not specified.			Enum: [PKCS1 PKCS8]

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`algorithm` *PrivateKeyAlgorithm*	Algorithm is the private key algorithm of the corresponding private key for this certificate. If provided, allowed values are either `RSA`, `Ed25519` or `ECDSA` If `algorithm` is specified and `size` is not provided, key size of 256 will be used for `ECDSA` key algorithm and key size of 2048 will be used for `RSA` key algorithm. key size is ignored when using the `Ed25519` key algorithm.			Enum: [RSA ECDSA Ed25519]
`size` *integer*	Size is the key bit size of the corresponding private key for this certificate. If `algorithm` is set to `RSA`, valid values are `2048`, `4096` or `8192`, and will default to `2048` if not specified. If `algorithm` is set to `ECDSA`, valid values are `256`, `384` or `521`, and will default to `256` if not specified. If `algorithm` is set to `Ed25519`, Size is ignored. No other values are allowed.			

CertificateSpec

CertificateSpec defines the desired state of Certificate. A valid Certificate requires at least one of a CommonName, DNSName, or URISAN to be valid.

Appears in:

- CertManagerTemplate

Field	Description	Required	Default	Validation
`subject` *X509Subject*	Full X509 name specification (https://golang.org/pkg/crypto/x509/pkix/#Name).			
`commonName` *string*	CommonName is a common name to be used on the Certificate. The CommonName should have a length of 64 characters or fewer to avoid generating invalid CSRs. This value is ignored by TLS clients when any subject alt name is set. This is x509 behaviour: https://tools.ietf.org/html/rfc6125#section-6.4.4			
`duration` *Duration*	The requested 'duration' (i.e. lifetime) of the Certificate. This option may be ignored/overridden by some issuer types. If unset this defaults to 90 days. Certificate will be renewed either 2/3 through its duration or `renewBefore` period before its expiry, whichever is later. Minimum accepted duration is 1 hour. Value must be in units accepted by Go time.ParseDuration https://golang.org/pkg/time/#ParseDuration			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`renewBefore` *Duration*	How long before the currently issued certificate's expiry cert-manager should renew the certificate. The default is 2/3 of the issued certificate's duration. Minimum accepted value is 5 minutes. Value must be in units accepted by Go time.ParseDuration https://golang.org/pkg/time/#ParseDuration			
`dnsNames` *string array*	DNSNames is a list of DNS subjectAltNames to be set on the Certificate.			
`ipAddresses` *string array*	IPAddresses is a list of IP address subjectAltNames to be set on the Certificate.			
`uris` *string array*	URIs is a list of URI subjectAltNames to be set on the Certificate.			
`emailAddresses` *string array*	EmailAddresses is a list of email subjectAltNames to be set on the Certificate.			
`secretName` *string*	SecretName is the name of the secret resource that will be automatically created and managed by this Certificate resource. It will be populated with a private key and certificate, signed by the denoted issuer. IMPORTANT: this field was required in the original cert-manager API declaration	True		
`keystores` *CertificateKeystores*	Keystores configures additional keystore output formats stored in the `secretName` Secret resource.			
`issuerRef` *ObjectReference*	IssuerRef is a reference to the issuer for this certificate. If the `kind` field is not set, or set to `Issuer`, an Issuer resource with the given name in the same namespace as the Certificate will be used. If the `kind` field is set to `ClusterIssuer`, a ClusterIssuer with the provided name will be used. The `name` field in this stanza is required at all times.	True		
`isCA` *boolean*	IsCA will mark this Certificate as valid for certificate signing. This will automatically add the `cert sign` usage to the list of `usages`.			
`usages` *KeyUsage array*	Usages is the set of x509 usages that are requested for the certificate. Defaults to `digital signature` and `key encipherment` if not specified.			Enum: [signing digital signature content commitment key encipherment key agreement data encipherment cert sign crl sign encipher only decipher only any server auth client auth code signing email protection s/mime ipsec end system ipsec tunnel ipsec user timestamping ocsp signing microsoft sgc netscape sgc]
`privateKey` *CertificatePrivateKey*	Options to control private keys used for the Certificate.			
`encodeUsagesInRequest` *boolean*	EncodeUsagesInRequest controls whether key usages should be present in the CertificateRequest			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`revisionHistoryLimit` *integer*	revisionHistoryLimit is the maximum number of CertificateRequest revisions that are maintained in the Certificate's history. Each revision represents a single `CertificateRequest` created by this Certificate, either when it was created, renewed, or Spec was changed. Revisions will be removed by oldest first if the number of revisions exceeds this number. If set, revisionHistoryLimit must be a value of `1` or greater. If unset (`nil`), revisions will not be garbage collected. Default value is `nil` .			ExclusiveMaximum: false

JKSKeystore

JKSKeystore configures options for storing a JKS keystore in the `spec.secretName` Secret resource.

Appears in:

- CertificateKeystores

Field	Description	Required	Default	Validation
`create` *boolean*	Create enables JKS keystore creation for the Certificate. If true, a file named `keystore.jks` will be created in the target Secret resource, encrypted using the password stored in `passwordSecretRef` . The keystore file will only be updated upon re-issuance. A file named `truststore.jks` will also be created in the target Secret resource, encrypted using the password stored in `passwordSecretRef` containing the issuing Certificate Authority	True		
`passwordSecretRef` *SecretKeySelector*	PasswordSecretRef is a reference to a key in a Secret resource containing the password used to encrypt the JKS keystore.	True		

KeyUsage

Underlying type: string

KeyUsage specifies valid usage contexts for keys. See: https://tools.ietf.org/html/rfc5280#section-4.2.1.3

```
https://tools.ietf.org/html/rfc5280#section-4.2.1.12
```

Valid KeyUsage values are as follows: "signing", "digital signature", "content commitment", "key encipherment", "key agreement", "data encipherment", "cert sign", "crl sign", "encipher only", "decipher only", "any", "server auth", "client auth", "code signing", "email protection", "s/mime", "ipsec end system", "ipsec tunnel", "ipsec user", "timestamping", "ocsp signing", "microsoft sgc", "netscape sgc"

Validation:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

- Enum: [signing digital signature content commitment key encipherment key agreement data encipherment cert sign crl sign encipher only decipher only any server auth client auth code signing email protection s/mime ipsec end system ipsec tunnel ipsec user timestamping ocsp signing microsoft sgc netscape sgc]

Appears in:

- CertificateSpec

Field	Description
`signing`	UsageSigning is a key usage flag
`digital signature`	UsageDigitalSignature is a key usage flag
`content commitment`	UsageContentCommitment is a key usage flag
`key encipherment`	UsageKeyEncipherment is a key usage flag
`key agreement`	UsageKeyAgreement is a key usage flag
`data encipherment`	UsageDataEncipherment is a key usage flag
`cert sign`	UsageCertSign is a key usage flag
`crl sign`	UsageCRLSign is a key usage flag
`encipher only`	UsageEncipherOnly is a key usage flag
`decipher only`	UsageDecipherOnly is a key usage flag
`any`	UsageAny is a key usage flag
`server auth`	UsageServerAuth is a key usage flag
`client auth`	UsageClientAuth is a key usage flag
`code signing`	UsageCodeSigning is a key usage flag
`email protection`	UsageEmailProtection is a key usage flag
`s/mime`	UsageSMIME is a key usage flag
`ipsec end system`	UsageIPsecEndSystem is a key usage flag
`ipsec tunnel`	UsageIPsecTunnel is a key usage flag
`ipsec user`	UsageIPsecUser is a key usage flag
`timestamping`	UsageTimestamping is a key usage flag
`ocsp signing`	UsageOCSPSigning is a key usage flag
`microsoft sgc`	UsageMicrosoftSGC is a key usage flag
`netscape sgc`	UsageNetscapeSGC is a key usage flag

LocalObjectReference

LocalObjectReference is a reference to an object in the same namespace as the referent. If the referent is a cluster-scoped resource (e.g. a ClusterIssuer), the reference instead refers to the resource with the given name in the configured 'cluster resource namespace', which is set as a flag on the controller component (and defaults to the namespace that cert-manager runs in).

Appears in:

- SecretKeySelector

Field	Description	Required	Default	Validation
`name` *string*	Name of the resource being referred to. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names	True		

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

ObjectReference

ObjectReference is a reference to an object with a given name, kind and group.

Appears in:

- CertificateSpec

Field	Description	Required	Default	Validation
`name` *string*	Name of the resource being referred to.	True		
`kind` *string*	Kind of the resource being referred to.			
`group` *string*	Group of the resource being referred to.			

PKCS12Keystore

PKCS12Keystore configures options for storing a PKCS12 keystore in the `spec.secretName` Secret resource.

Appears in:

- CertificateKeystores

Field	Description	Required	Default	Validation
`create` *boolean*	Create enables PKCS12 keystore creation for the Certificate. If true, a file named `keystore.p12` will be created in the target Secret resource, encrypted using the password stored in `passwordSecretRef`. The keystore file will only be updated upon re-issuance. A file named `truststore.p12` will also be created in the target Secret resource, encrypted using the password stored in `passwordSecretRef` containing the issuing Certificate Authority	True		
`passwordSecretRef` *SecretKeySelector*	PasswordSecretRef is a reference to a key in a Secret resource containing the password used to encrypt the PKCS12 keystore.	True		

PrivateKeyAlgorithm

Underlying type: string

PrivateKeyAlgorithm represent a private key algorithm

Validation:

- Enum: [RSA ECDSA Ed25519]

Appears in:

- CertificatePrivateKey

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description
`RSA`	RSAKeyAlgorithm denotes the RSA private key type.
`ECDSA`	ECDSAKeyAlgorithm denotes the ECDSA private key type.
`Ed25519`	Ed25519KeyAlgorithm denotes the Ed25519 private key type.

PrivateKeyEncoding

Underlying type: string

PrivateKeyEncoding represent a private key encoding

Validation:

- Enum: [PKCS1 PKCS8]

Appears in:

- CertificatePrivateKey

Field	Description
`PKCS1`	PKCS1 key encoding will produce PEM files that include the type of private key as part of the PEM header, e.g. `BEGIN RSA PRIVATE KEY`. If the keyAlgorithm is set to 'ECDSA', this will produce private keys that use the `BEGIN EC PRIVATE KEY` header.
`PKCS8`	PKCS8 key encoding will produce PEM files with the `BEGIN PRIVATE KEY` header. It encodes the keyAlgorithm of the private key as part of the DER encoded PEM block.

PrivateKeyRotationPolicy

Underlying type: string

PrivateKeyRotationPolicy denotes how private keys should be generated or sourced when a Certificate is being issued.

Appears in:

- CertificatePrivateKey

SecretKeySelector

SecretKeySelector is a reference to a specific 'key' within a Secret resource. In some instances, `key` is a required field.

Appears in:

- JKSKeystore
- PKCS12Keystore

Field	Description	Required	Default	Validation

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Field	Description	Required	Default	Validation
`name` *string*	Name of the resource being referred to. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names	True		
`key` *string*	The key of the entry in the Secret resource's `data` field to be used. Some instances of this field may be defaulted, in others it may be required.			

X509Subject

X509Subject Full X509 name specification

Appears in:

- CertificateSpec

Field	Description	Required	Default	Validation
`organizations` *string array*	Organizations to be used on the Certificate.			
`countries` *string array*	Countries to be used on the Certificate.			
`organizationalUnits` *string array*	Organizational Units to be used on the Certificate.			
`localities` *string array*	Cities to be used on the Certificate.			
`provinces` *string array*	State/Provinces to be used on the Certificate.			
`streetAddresses` *string array*	Street addresses to be used on the Certificate.			
`postalCodes` *string array*	Postal codes to be used on the Certificate.			
`serialNumber` *string*	Serial number to be used on the Certificate.			

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

32 Supported versions

This page lists the status for currently supported releases of EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC)

Naming Scheme

The CNPG-GC (PGD4K) releases follows Semantic Versioning 2.0.0 and is structured as follows:

```
<major>.<minor>.<patch>
```

- `<major>` maps to the supported PGD version. Version 1.x corresponds to PGD version 5.x.
- `<minor>` is incremented for each supported PG4K Long-Term Support release dependency change.
- `<patch>` indexes the patches for the current `<minor>` release, representing small changes relative to the `<minor>` release.

Git tags for versions are prefixed with `v`.

General Support status

CNPG-GC version 1.x is used to manage Postgres Distributed version 5.x. The general support status is as follows:

> **Note**
>
> The following table summarizes the general support status for CNPG-GC. For detailed support status, including supported Kubernetes versions, PG4K versions, Postgres versions, and Postgres Distributed versions, please refer to the Detailed support status section below.

Version	Currently Supported	Release Date (first release)	Supported Kubernetes Versions	Supported PG4K Versions	Supported Postgres versions	Supported Postgres Distributed versions
1.2.x	Yes	Jan 5, 2026	v1.32 - v1.36	v1.28	v14 - v17	v5.9.x
1.1.x	Yes	Dec 25, 2024	v1.28 - v1.34	v1.25, v1.26	v13 - v17	v5.6.1 -
1.0.x	Yes	Apr 24, 2024	v1.28 - v1.34	v1.22	v13 - v17	v5.5.1 -

The Postgres (operand) versions are limited to those supported by EDB Postgres Distributed (PGD).

> **Important**
>
> Please be aware that this page is informative only. The "Platform Compatibility" page from the EDB website contains the official list of supported software and Kubernetes distributions.

Detailed support status

The CNPG-GC (PGD4K) operator uses the EDB Postgres for Kubernetes (PG4K) operator to manage each PGD node as a PG4K Cluster. Each minor release reaches its End of Life in alignment with the corresponding PG4K LTS release. For details on the support scope of each PG4K LTS release, please refer to the platform compatibility

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Version	Currently Supported	Release Date	Supported Kubernetes Versions	Supported PG4K Versions	Supported Postgres versions	Supported Postgres Distributed versions	Supported OpenShift Versions	Supported Cert-manager versions
1.2.2	Yes	Sept 10, 2026	1.34 - 1.36	v1.28.0-v1.28.4	v14 - v17	v5.9.4, v5.9.5	v4.18 - v4.22	>=v1.10.0
1.2.1	Yes	Jun 9, 2026	1.32 - 1.34	v1.28.0-v1.28.3	v14 - v17	v5.9.3, v5.9.4, v5.9.5	v4.16 - v4.21	>=v1.10.0
1.2.0	Yes	Jan 5, 2026	1.32 - 1.34	v1.28.0-v1.28.3	v14 - v17	v5.9.1, v5.9.2, v5.9.3, v5.9.4	v4.12 - v4.20	>=v1.10.0
1.1.3	Yes	Sept 25, 2025	1.30 - 1.33	v1.25.x, v1.26.x	v13 - v17	v5.9.0	v4.12 - v4.19	>=v1.10.0
1.1.2	Yes	Jul 25, 2025	1.30 - 1.33	v1.25.x, v1.26.0	v13 - v17	v5.8.1	v4.12 - v4.19	>=v1.10.0
1.1.1	Yes	Mar 18, 2025	1.29 - 1.32	v1.22.8, v1.22.9, v1.25.x	v13 - v17	v5.7.0	v4.12 - v4.18	>=v1.10.0

> **Note**
>
> The CNPG-GC (PGD4K) 1.2.1 operator doesn't support in-place minor version upgrades to PGD operand images based on open-source PostgreSQL versions 14.24, 15.19, 16.15, 17.11, and 18.6.
>
> This doesn't affect operand images based on EPAS or PGE distributions.
>
> Workarounds:
>
> - Instead of performing an in-place upgrade, recreate the cluster nodes with the target operand image.
> - Perform an in-place upgrade to a higher PostgreSQL major version.

> **Note**
>
> The supported OpenShift versions are those supported at the time the operator is released; an OpenShift version that reaches End of Support (EoS) is not removed from the list.

> **Note**
>
> CNPG-GC (PGD4K) version 1.0.0 didn't restrict supported PG4K version to LTS releases. When upgrading PGD4K from 1.0.0 to 1.0.1, if the installed PG4K version exceeds 1.22, uninstall the PG4K operator and reinstall it with a 1.22.x to satisfy the dependency requirements.

PGD Operator manifests

For each CNPG-GC release, we created an all-in-one manifests with CNPG-GC and the latest supported EDB Postgres for Kubernetes. To deploy those all-in-one manifests on Kubernetes, create the edb-pull-secret in advance and then apply the manifests.

```
kubectl apply --server-side --force-conflicts -f \
   https://get.enterprisedb.io/pg4k-pgd/pg4k-pgd-1.2.0.yaml
```

PGD4K release	Nested PG4K release	Manifests
v1.2.2	v1.28.4	pg4k-pgd-1.2.2.yaml

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

PGD4K release	Nested PG4K release	Manifests
v1.2.1	v1.28.3	pg4k-pgd-1.2.1.yaml
v1.2.0	v1.28.3	pg4k-pgd-1.2.0.yaml
v1.1.3	v1.26.1	pg4k-pgd-1.1.3.yaml
v1.1.2	v1.26.0	pg4k-pgd-1.1.2.yaml
v1.1.1	v1.25.1	pg4k-pgd-1.1.1.yaml
v1.1.0	v1.25.0	pg4k-pgd-1.1.0.yaml

PGD Operand images

For each minor release of PGD, EDB consistently builds the latest PostgreSQL version integrated with the latest PGD extension. Once a new minor version of PGD is released, EDB switches to build the latest PostgreSQL with the newly released PGD minor.

For example: If PGD 5.7 is the latest minor release, EDB will continue building PGD images with the latest PostgreSQL patches for each supported PostgreSQL version, along with the latest PGD 5.7 release (e.g., PGD 5.7.1). When PGD 5.8 is released, EDB will then update to build images with the latest PostgreSQL patches combined with PGD 5.8.x.

We maintain an imageCatalog for each PGD minor release. Each imageCatalog contains the operand image that includes the latest PostgreSQL patches and the latest PGD patch.

PGD Release	Flavor	ImageCatalog
PGD 5.9	EDB Postgres Advanced PGD	epas-k8s-pgd59-ubi8 epas-k8s-pgd59-ubi9
PGD 5.9	EDB Postgres Extended PGD	pgextended-k8s-pgd59-ubi8 pgextended-k8s-pgd59-ubi9
PGD 5.9	Postgres Community PGD	postgresql-k8s-pgd59-ubi8 postgresql-k8s-pgd59-ubi9
PGD 5.8	EDB Postgres Advanced PGD	epas-k8s-pgd58-ubi8 epas-k8s-pgd58-ubi9
PGD 5.8	EDB Postgres Extended PGD	pgextended-k8s-pgd58-ubi8 pgextended-k8s-pgd58-ubi9
PGD 5.8	Postgres Community PGD	postgresql-k8s-pgd58-ubi8 postgresql-k8s-pgd58-ubi9

> **Repository change for PGD 5.8 and later image catalogs**
>
> We're unifying our repositories, and thus operand image paths will change - please follow the Central Migration Guide to avoid deployment failure and downtime when upgrading.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

33 Troubleshooting

In this page, you can find some basic information on how to troubleshoot EDB Postgres® AI for CloudNativePG™ Global Cluster (CNPG-GC) in your Kubernetes cluster deployment.

> **Hint**
>
> As a Kubernetes administrator, you should have the `kubectl` Cheat Sheet page bookmarked!

Before you start

Kubernetes environment

What can make a difference in a troubleshooting activity is to provide clear information about the underlying Kubernetes system.

Make sure you know:

- the Kubernetes distribution and version you are using
- the specifications of the nodes where PostgreSQL is running

Useful utilities

On top of the mandatory `kubectl` utility, for troubleshooting, we recommend the following plugins/utilities to be available in your system:

- `cnp` plugin for `kubectl`, which could be used to talk with each individual node (PG4K cluster)
- `jq`, a lightweight and flexible command-line JSON processor
- `grep`, searches one or more input files for lines containing a match to a specified pattern. It is already available in most *nix distros. If you are on Windows OS, you can use `findstr` as an alternative to `grep` or directly use `wsl` and install your preferred *nix distro and use the tools mentioned above.

Logs

All resources created and managed by CNPG-GC log to standard output in accordance with Kubernetes conventions, using JSON format.

While logs are typically processed at the infrastructure level and include those from CNPG-GC and EDB Postgres for Kubernetes, accessing logs directly from the command line interface is critical during troubleshooting. You have three primary options for doing so:

- Use the `kubectl logs` command to retrieve logs from a specific resource, and apply `jq` for better readability.
- Use the `kubectl cnp logs` command for EDB Postgres for Kubernetes-specific logging, this is useful to collecting logs node by node.
- Leverage specialized open-source tools like stern, which can aggregate logs from multiple resources (e.g., all pods in a PGDGroup by selecting the `k8s.pgd.enterprisedb.io/group` label), filter log entries, customize output formats, and more.

> **Note**
>
> The following sections provide examples of how to retrieve logs for various resources when troubleshooting CNPG-GC.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Operator information

There are two operators for managing resources within a PGDGroup:

- CNPG-GC (PGD4K operator) Manages the PGDGroup and resources directly created by it.
- EDB Postgres for Kubernetes (PG4K operator) Manages PG4K clusters, which are used as nodes within a PGDGroup.

By default, the PGD4K operator is installed in the `pgd-operator-system` namespace as a `Deployment`. (Refer to the "Details about the deployment" section section for more information.)

To list the operator pods, run:

```
kubectl get pods -n pgd-operator-system
```

> **Note**
>
> Under normal circumstances, you should have one pod where the operator is running, identified by a name starting with `pgd-operator-controller-manager-`. In case you have set up your operator for high availability, you should have more entries. Those pods are managed by a deployment named `pgd-operator-controller-manager`.

Collect the relevant information about the operator that is running in pod `<POD>` with:

```
kubectl describe pod -n pgd-operator-system <POD>
```

Then get the logs from the same pod by running:

```
kubectl logs -n pgd-operator-system <POD>
```

Gather more information about the PGD4K operator

Get logs from all pods in CNPG-GC operator Deployment (in case you have a multi operator deployment) by running:

```
kubectl logs -n pgd-operator-system \
  deployment/pgd-operator-controller-manager --all-containers=true
```

> **Tip**
>
> You can add `-f` flag to above command to follow logs in real time.

Save logs to a JSON file by running:

```
kubectl logs -n pgd-operator-system \
  deployment/pgd-operator-controller-manager --all-containers=true | \
  jq -r . > pgd_logs.json
```

Gather more information about the PG4K operator

As PGD4K operator leverages the PG4K operator to manage each node, we also need to collect the PG4K operator logs. Please visitOperator Information to collect logs for the PG4K operator.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

PGDGroup information

You can check the status of the `pgd-sample` PGDGroup in the `NAMESPACE` namespace with:

```
kubectl get pgdgroup -n <NAMESPACE> pgd-sample
```

Output:

```
NAME         DATA INSTANCES   WITNESS INSTANCES   PHASE                AGE
pgd-sample   2                1                   PGDGroup - Healthy   3h1m
```

The above example describes a healthy PGDGroup cluster consisting of 2 data nodes and 1 witness node.

A PGDGroup is composed of multiple nodes, where each node is a single-instance PG4K cluster. To view all nodes, you can retrieve the cluster information and filter by the label `k8s.pgd.enterprisedb.io/group`. Each node is named following the format: `<group name>-<number>`.

```
kubectl -n pgd get cluster -l k8s.pgd.enterprisedb.io/group=pgd-sample -A
```

Output:

```
NAME           AGE    INSTANCES   READY   STATUS                     PRIMARY
pgd-sample-1   3h2m   1           1       Cluster in healthy state   pgd-sample-1-1
pgd-sample-2   179m   1           1       Cluster in healthy state   pgd-sample-2-1
pgd-sample-3   176m   1           1       Cluster in healthy state   pgd-sample-3-1
```

PGD Node pod information

Each PGD node is a single-instance PG4K cluster running on Kubernetes. To retrieve the list of instances belonging to a specific PGDGroup, use the following command:

with:

```
kubectl get pod -l k8s.pgd.enterprisedb.io/group=pgd-sample -A
```

Output:

```
NAMESPACE   NAME                 READY   STATUS    RESTARTS   AGE    ROLE
pgd         pgd-sample-1-1       1/1     Running   0          57m    primary
pgd         pgd-sample-2-1       1/1     Running   0          61m    primary
pgd         pgd-sample-3-1       1/1     Running   0          65m    primary
pgd         pgd-sample-proxy-0   1/1     Running   0          3h
pgd         pgd-sample-proxy-1   1/1     Running   0          179m
```

You can check if/how a pod is failing by running:

```
kubectl get pod -n <NAMESPACE> -o yaml <GROUP>-<N>-1
```

You can get all the logs for a given pgd with:

```
kubectl logs -n <NAMESPACE> <GROUP>-<N>-1
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

If you want to limit the search to the PostgreSQL process only, you can run:

```
kubectl logs -n <NAMESPACE> <GROUP>-<N>-1 | \
  jq 'select(.logger=="postgres") | .record.message'
```

The following example also adds the timestamp:

```
kubectl logs -n <NAMESPACE> <GROUP>-<N>-1 | \
  jq -r 'select(.logger=="postgres") | [.ts, .record.message] | @csv'
```

If the timestamp is displayed in Unix Epoch time, you can convert it to a user-friendly format:

```
kubectl logs -n <NAMESPACE> <GROUP>-<N>-1 | \
  jq -r 'select(.logger=="postgres") | [(.ts|strflocaltime("%Y-%m-%dT%H:%M:%S %Z")), .record.message] |
@csv'
```

Gather and filter extra information about PostgreSQL pods

Check logs from a specific pod that has crashed:

```
kubectl logs -n <NAMESPACE> --previous <GROUP>-<N>-1
```

Get FATAL errors from a specific PostgreSQL pod:

```
kubectl logs -n <NAMESPACE> <GROUP>-<N>-1 | \
  jq -r '.record | select(.error_severity == "FATAL")'
```

Output:

```
{
  "log_time": "2021-11-08 14:07:44.520 UTC",
  "user_name": "streaming_replica",
  "process_id": "68",
  "connection_from": "10.244.0.10:60616",
  "session_id": "61892f30.44",
  "session_line_num": "1",
  "command_tag": "startup",
  "session_start_time": "2021-11-08 14:07:44
UTC",
  "virtual_transaction_id": "3/75",
  "transaction_id": "0",
  "error_severity": "FATAL",
  "sql_state_code": "28000",
  "message": "role \"streaming_replica\" does not
exist",
  "backend_type": "walsender"
}
```

Filter PostgreSQL DB error messages in logs for a specific pod:

```
kubectl logs -n <NAMESPACE> <GROUP>-<N>-1 | jq -r '.err | select(. != null)'
```

Output:

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

```
dial unix /controller/run/.s.PGSQL.5432: connect: no such file or directory
```

Get messages matching `err` word from a specific pod:

```
kubectl logs -n <NAMESPACE> <GROUP>-<N>-1 | jq -r '.msg' | grep "err"
```

Output:

```
2021-11-08 14:07:39.610 UTC [15] LOG:  ending log output to stderr
```

Get all logs from PostgreSQL process from a specific pod:

```
kubectl logs -n <NAMESPACE> <GROUP>-<N>-1| \
  jq -r '. | select(.logger == "postgres") | select(.msg != "record") | .msg'
```

Output:

```
2021-11-08 14:07:52.591 UTC [16] LOG:  redirecting log output to logging collector process
2021-11-08 14:07:52.591 UTC [16] HINT:  Future log output will appear in directory "/controller/log".
2021-11-08 14:07:52.591 UTC [16] LOG:  ending log output to stderr
2021-11-08 14:07:52.591 UTC [16] HINT:  Future log output will go to log destination "csvlog".
```

Get pod logs filtered by fields with values and join them separated by `|` running:

```
kubectl logs -n <NAMESPACE> <GROUP>-<N>-1 | \
  jq -r '[.level, .ts, .logger, .msg] | join(" | ")'
```

Output:

```
info | 1636380469.5728037 | wal-archive | Backup not configured, skip WAL archiving
info | 1636383566.0664876 | postgres | record
```

ScheduledBackup and Backup information

You can list the scheduled backups for the pgdgroup-backup label filter using the following command:

```
kubectl get scheduledbackup -l k8s.pgd.enterprisedb.io/group=pgdgroup-backup -A
```

Output:

```
NAME                           AGE     CLUSTER             LAST BACKUP
pgdgroup-backup-1-pgd-barman   9m27s   pgdgroup-backup-1   9m27s
pgdgroup-backup-1-pgd-vol      9m26s   pgdgroup-backup-1   9m26s
```

The scheduled backup is named with the format `<cluster>-pgd-<backup method>`. The `<cluster>` is the node name chosen as the backup target. If the backup is properly configured, WAL archiving occurs on all nodes, but backup is only taken on the selected node.

You can also list the backups that have been created with:

```
kubectl get backup -l k8s.pgd.enterprisedb.io/group=pgdgroup-backup -A
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Output:

```
NAMESPACE   NAME                                          AGE   CLUSTER             METHOD              PHASE
ERROR
pgd         pgdgroup-backup-1-pgd-barman-20250731094444   18m   pgdgroup-backup-1   barmanObjectStore
completed
pgd         pgdgroup-backup-1-pgd-vol-20250731094445      18m   pgdgroup-backup-1   volumeSnapshot
completed
```

More troubleshooting knowledge

You can reference Before you start in CloudNativePG cluster for more information about Kubernetes troubleshooting knowledge.

Some known issues

These known issues and limitations are in the current release of CNPG-GC.

Switching between global Routing and local Routing

PGD supports the PGD Proxy routing to global write lead and local write lead. This can be achieved by setting the `spec.pgd.globalRouting` field in the `PGDGroup` object to `true` or `false`. The change of this setting will update the PGD Proxy configuration, you can see the `node_group_name` for the corresponding proxy from table `bdr.proxy_config_summary`, Or run the `pgd show-proxies` command in the PGD proxy pod.

There is a corner case once you switched the routing method, the PGD Proxy still routes to the old write lead (global or local), in this case, we can rolling restart the PGD Proxy pods to have PGD Proxy force refreshed.

```
kubectl rollout restart statefulset <PGD_GROUP_NAME>-proxy -n <NAMESPACE>
```

Postgres major version upgrades

This version of CNPG-GC release (v1.1.2) supports the major version upgrade with following restrictions:

- PGD4K Operator v1.1.2 or higher
- PG4K Operator v1.26.0 or higher
- PGD Operand 5.8 or greater

Physical join

Since release v1.1.1 and operand PGD 5.7, the operator supports node physical joins to other ready nodes.

If a physical join job is fails:

```
> kubectl get job -n <NAMESPACE>
NAME                                   STATUS   COMPLETIONS   DURATION   AGE
pgdgroup-backup-barman-2-physical-join   Failed   0/1           10m        10m
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

If the `BackoffLimit` is reached and the physical join job is failed before non-retriable step, you can delete the job to trigger the physical join again:

```
kubectl delete job pgdgroup-backup-barman-2-physical-join -n <NAMESPACE>
```

If the `BackoffLimit` is reached and the physical join job is failed in non-retriable step, the node data may be left in an inconsistent state, we need to delete the node to recreate the node again:

```
kubectl delete cluster pgdgroup-backup-barman-2 -n <NAMESPACE>
```

Physical join failed before the non-retriable step

> **Note**
>
> We can check the logs of the physical join job to determine whether the failure happened before or in the non-retriable step. If the failure happened in the non-retriable step, the physical join may already started the data copying, simply recreating the job alone is not enough, because the retry is intentionally blocked to avoid re-running the non-idempotent join step on partially initialized data. In that case recreate the data node so it starts from a fresh volume.

One key pre-condition for a physical join is the establishment of a global Raft consensus. If a physical join job is pending, you can use the PGD function `bdr.monitor_group_raft` to verify whether this pre-condition has been satisfied.

Data migration

This version of CNPG-GC doesn't support declarative import of data from other Postgres databases. To migrate schemas and data, you can use traditional Postgres migration tools such as EDB*Loader or Migration Toolkit/Replication Server. You can also use `pg_dump ...` on the source database and pipe the command's output to your target database with `psql -c`.

Connectivity with PgBouncer

CNPG-GC **doesn't support** using PgBouncer to pool client connection requests. This limitation applies to both the open-source and EDB versions of PgBouncer.

Backup operations

To configure an CNPG-GC environment, you must apply a `PGDGroup` YAML object to each Kubernetes cluster. Applying this object creates all necessary services for implementing a distributed architecture.

If you added a `spec.backup` section to this `PGDGroup` object with the goal of setting up a backup configuration, the backup will fail unless you also set the `spec.backup.schedulers` value.

Error output example:

```
The PGDGroup "region-a" is invalid: spec.backup.schedulers: Invalid value: "": Empty spec string
```

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.

Workaround

To work around this issue, add a `spec.backup.schedulers` section with a schedule that meets your requirements, for example:

```
spec:
  instances: 3
  pgd:
    parentGroup:
      create: true
      name: world
  backup:
    configuration:
      barmanObjectStore:

...
    schedulers:
      - method: barmanObjectStore
        immediate: true
        schedule: "0 */5 * * *
*"
```

Known issues and limitations in EDB Postgres Distributed

All issues and limitations known for the EDB Postgres Distributed version that you include in your deployment also affect your CNPG-GC instance.

For example, if the EDB Postgres Distributed version you're using is 5.x, your CNPG-GC instance will be affected by these5.x known issues and 5.x limitations.

Copyright © 2009 - 2026 EnterpriseDB Corporation. All rights reserved.