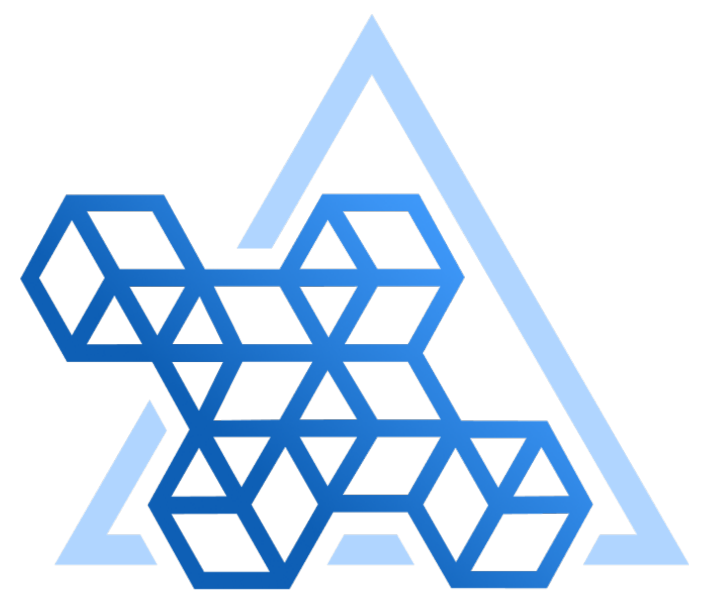


Percona Operator for PostgreSQL documentation

2.3.1 (January 23, 2024)

Percona Technical Documentation Team

Percona LLC and/or its affiliates, © 2009 - 2024

Table of contents

1. About 4
1.1 Percona Operator for PostgreSQL documentation 4
1.2 Compare various solutions to deploy PostgreSQL in Kubernetes 6
1.3 Design overview 9
2. Quickstart guide 12
2.1 Overview 12
2.2 1 Quick install 13
2.3 2 Connect to the PostgreSQL cluster 20
2.4 3 Insert sample data 22
2.5 4 Make a backup 24
2.6 5 Monitor the database 27
2.7 What’s next? 31
3. Installation 32
3.1 System requirements 32
3.2 Install Percona Distribution for PostgreSQL on Minikube 33
3.3 Install Percona Distribution for PostgreSQL on Google Kubernetes Engine (GKE) 37
3.4 Install Percona Distribution for PostgreSQL on Amazon Elastic Kubernetes Service (EKS) 42
3.5 Install Percona Distribution for PostgreSQL on OpenShift 48
3.6 Install Percona Distribution for PostgreSQL on Kubernetes 51
4. Configuration 55
4.1 Users 55
4.2 Exposing cluster 59
4.3 Changing PostgreSQL options 61
4.4 Binding Percona Distribution for PostgreSQL components to specific Kubernetes/OpenShift Nodes 63
4.5 Labels and annotations 65
4.6 Transport layer security (TLS) 67
4.7 Telemetry 71
5. Management 73
5.1 Upgrade Database and Operator 73
5.2 Upgrade from version 1 to version 2 76
5.3 Back up and restore 84
5.4 High availability and scaling 95
5.5 Using sidecar containers 98
5.6 Pause/resume PostgreSQL cluster 100
5.7 Monitor with Percona Monitoring and Management (PMM) 101

Percona LLC and/or its affiliates, © 2009 - 2024

6. HowTo 105
- 6.1 Install Percona Distribution for PostgreSQL with customized parameters 105
- 6.2 How to deploy a standby cluster for Disaster Recovery 106
- 6.3 Use Docker images from a custom registry 110
- 6.4 Add custom PostgreSQL extensions 112
- 6.5 Percona Operator for PostgreSQL single-namespace and multi-namespace deployment 117
- 6.6 Delete Percona Operator for PostgreSQL 122
- 6.7 Monitor Kubernetes 126
- 6.8 Use PostGIS extension with Percona Distribution for PostgreSQL 132

7. Troubleshooting 137
- 7.1 Initial troubleshooting 137
- 7.2 Exec into the containers 140
- 7.3 Check the logs 141

8. Reference 142
- 8.1 Custom Resource options 142
- 8.2 Percona certified images 168
- 8.3 Versions compatibility 171
- 8.4 Copyright and licensing information 173
- 8.5 Trademark policy 174

9. Release Notes 176
- 9.1 Percona Operator for PostgreSQL Release Notes 176
- 9.2 Percona Operator for PostgreSQL 2.3.1 177
- 9.3 Percona Operator for PostgreSQL 2.3.0 179
- 9.4 Percona Operator for PostgreSQL 2.2.0 182
- 9.5 Percona Operator for PostgreSQL 2.1.0 (Tech preview) 184
- 9.6 Percona Operator for PostgreSQL 2.0.0 (Tech preview) 186

Percona LLC and/or its affiliates, © 2009 - 2024

1. About

1.1 Percona Operator for PostgreSQL documentation

The Percona Operator for PostgreSQL automates the creation, modification, or deletion of items in your Percona Distribution for PostgreSQL environment. The Operator contains the necessary Kubernetes settings to maintain a consistent PostgreSQL cluster.

Percona Kubernetes Operator is based on best practices for configuration and setup of a Percona Distribution for PostgreSQL cluster. The benefits of the Operator are many, but saving time and delivering a consistent and vetted environment is key.

This is the documentation for the latest release, **2.3.1** (Release Notes).

Starting with Percona Kubernetes Operator is easy. Follow our documentation guides, and you'll be set up in a minute.

1.1.1 Installation guides

Want to see it for yourself? Get started quickly with our step-by-step installation instructions.

Quickstart guides →

1.1.2 Security and encryption

Rest assured! Learn more about our security features designed to protect your valuable data.

Security measures →

Backup management

Learn what you can do to maintain regular backups of your PostgrgeSQL cluster.

Backup management →

Troubleshooting

Our comprehensive resources will help you overcome challenges, from everyday issues to specific doubts.

1.1.3 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Percona LLC and/or its affiliates, © 2009 - 2024

Last update: 2024-01-16

Percona LLC and/or its affiliates, © 2009 - 2024

1.2 Compare various solutions to deploy PostgreSQL in Kubernetes

There are multiple ways to deploy and manage PostgreSQL in Kubernetes. Here we will focus on comparing the following open source solutions:

- Crunchy Data PostgreSQL Operator (PGO)
- CloudNative PG from Enterprise DB
- Stackgres from OnGres
- Zalando Postgres Operator
- Percona Operator for PostgreSQL

1.2.1 Generic

Feature/ Product	Percona Operator for PostgreSQL	Stackgres	CrunchyData	CloudNativePG (EDB)	Zalando
Open-source license	Apache 2.0	AGPL 3	Apache 2.0, but images are under Developer Program	Apache 2.0	MIT
PostgreSQL versions	12 - 16	12-15	12, 13, 14	11 - 15	11 - 14
Kubernetes conformance	Various versions are tested	Various versions are tested	Various versions are tested	Various versions are tested	AWS EKS

1.2.2 Maintenance

Feature/ Product	Percona Operator for PostgreSQL	Stackgres	CrunchyData	CloudNativePG (EDB)	Zalando
Operator upgrade	✅	✅	✅	✅	✅
Database upgrade	Automated and safe	Automated and safe	Manual	Manual	Manual
Compute scaling	Horizontal and vertical	Horizontal and vertical	Horizontal and vertical	Horizontal and vertical	Horizontal and vertical
Storage scaling	Manual	Manual	Manual	Manual	Manual, automated for AWS EBS

Percona LLC and/or its affiliates, © 2009 - 2024

1.2.3 PostgreSQL topologies

Feature/ Product	Percona Operator for PostgreSQL	Stackgres	CrunchyData	CloudNativePG (EDB)	Zalando
Warm standby	✅	✅	✅	✅	✅
Hot standby	✅	✅	✅	✅	✅
Connection pooling	✅	✅	✅	✅	✅
Delayed replica	🚫	🚫	🚫	🚫	🚫

1.2.4 Backups

Feature/ Product	Percona Operator for PostgreSQL	Stackgres	CrunchyData	CloudNativePG (EDB)	Zalando
Scheduled backups	✅	✅	✅	✅	✅
WAL archiving	✅	✅	✅	✅	✅
PITR	✅	✅	✅	✅	✅
GCS	✅	✅	✅	✅	✅
S3	✅	✅	✅	✅	✅
Azure	✅	✅	✅	✅	✅

1.2.5 Monitoring

Feature/ Product	Percona Operator for PostgreSQL	Stackgres	CrunchyData	CloudNativePG (EDB)	Zalando
Solution	Percona Monitoring and Management and sidecars	Exposing metrics in Prometheus format	Prometheus stack and pgMonitor	Exposing metrics in Prometheus format	Sidecars

Percona LLC and/or its affiliates, © 2009 - 2024

1.2.6 Miscellaneous

Feature/ Product	**Percona Operator for PostgreSQL**	**Stackgres**	**CrunchyData**	**CloudNativePG (EDB)**	**Zalando**
Customize PostgreSQL configuration	✅	✅	✅	✅	✅
Helm	✅	✅	✅	✅	✅
Transport encryption	✅	✅	✅	✅	✅
Data-at-rest encryption	Through storage class	Through storage class	Through storage class	Through storage class	Through storage class
Create users/ roles	✅	✅	✅	✅	limited

1.2.7 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

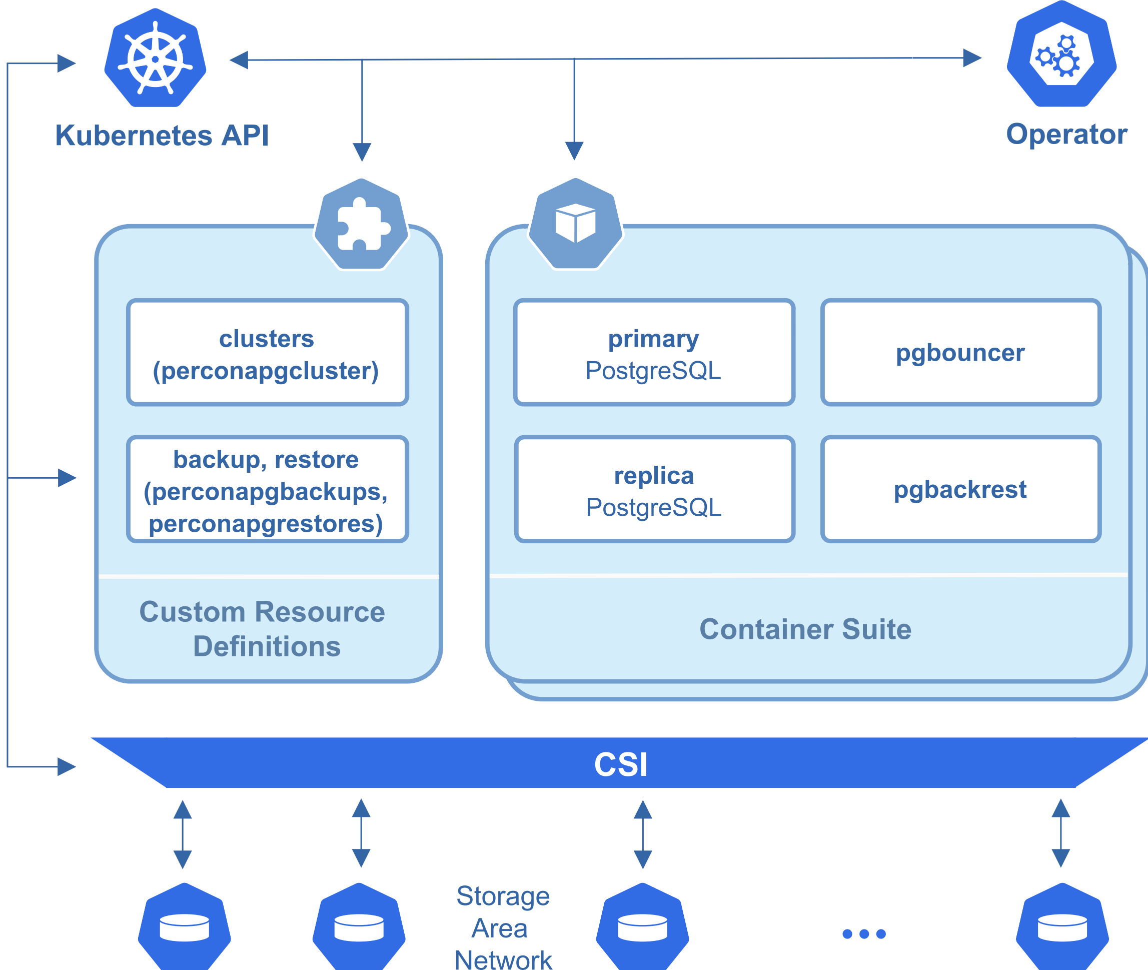
Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

1.3 Design overview

The Percona Operator for PostgreSQL automates and simplifies deploying and managing open source PostgreSQL clusters on Kubernetes. The Operator is based on CrunchyData's PostgreSQL Operator.

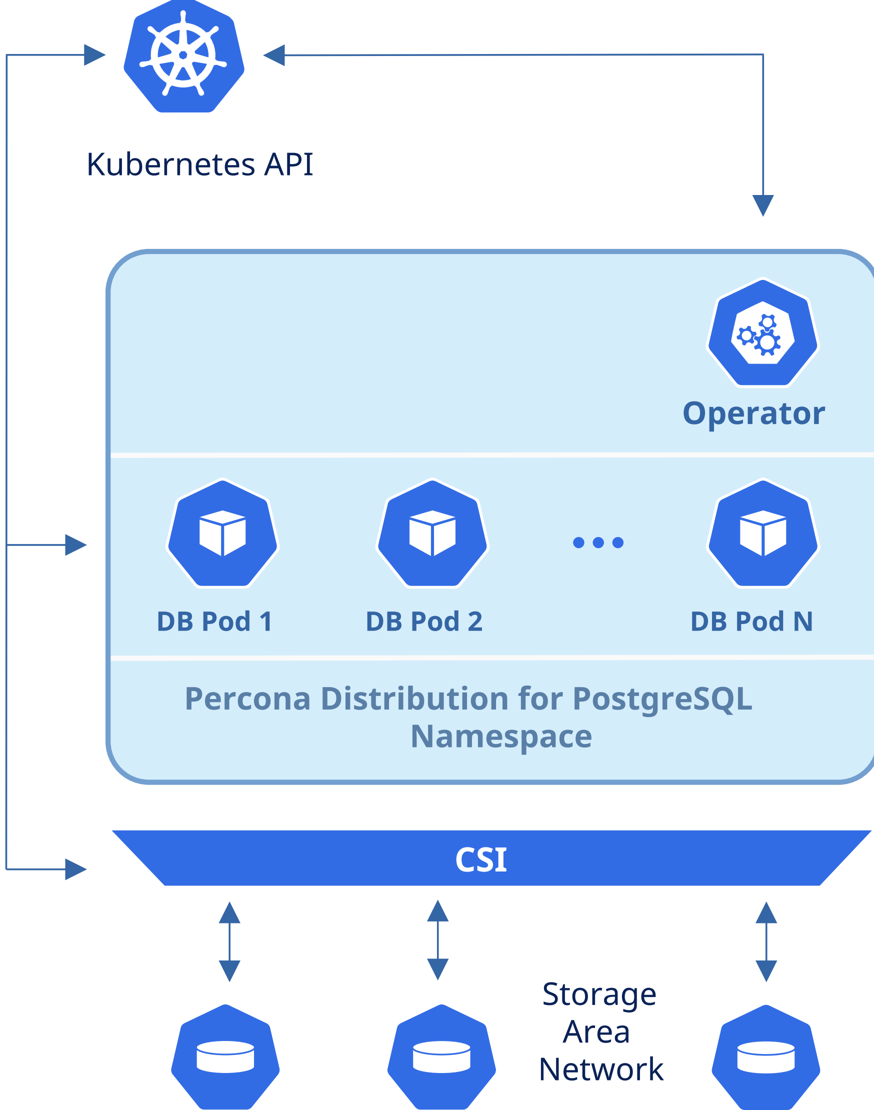


PostgreSQL containers deployed with the Operator include the following components:

- The PostgreSQL database management system, including:
- PostgreSQL Additional Supplied Modules,
- pgAudit PostgreSQL auditing extension,
- PostgreSQL set_user Extension Module,
- wal2json output plugin,
- The pgBackRest Backup & Restore utility,
- The pgBouncer connection pooler for PostgreSQL,
- The PostgreSQL high-availability implementation based on the Patroni template,
- the pg_stat_monitor PostgreSQL Query Performance Monitoring utility,
- LLVM (for JIT compilation).

To provide high availability the Operator involves node affinity to run PostgreSQL Cluster instances on separate worker nodes if possible. If some node fails, the Pod with it is automatically re-created on another node.

Percona LLC and/or its affiliates, © 2009 - 2024



To provide data storage for stateful applications, Kubernetes uses Persistent Volumes. A *PersistentVolumeClaim* (PVC) is used to implement the automatic storage provisioning to pods. If a failure occurs, the Container Storage Interface (CSI) should be able to re-mount storage on a different node.

The Operator functionality extends the Kubernetes API with Custom Resources Definitions. These CRDs provide extensions to the Kubernetes API, and, in the case of the Operator, allow you to perform actions such as creating a PostgreSQL Cluster, updating PostgreSQL Cluster resource allocations, adding additional utilities to a PostgreSQL cluster, e.g. pgBouncer for connection pooling and more.

When a new Custom Resource is created or an existing one undergoes some changes or deletion, the Operator automatically creates/changes/deletes all needed Kubernetes objects with the appropriate settings to provide a proper Percona PostgreSQL Cluster operation.

Following CRDs are created while the Operator installation:

- `perconapgclusters` stores information required to manage a PostgreSQL cluster. This includes things like the cluster name, what storage and resource classes to use, which version of PostgreSQL to run, information about how to maintain a high-availability cluster, etc.
- `perconapgbackups` and `perconapgrestores` are in charge for making backups and restore them.

Percona LLC and/or its affiliates, © 2009 - 2024

1.3.1 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-04-14

Percona LLC and/or its affiliates, © 2009 - 2024

2. Quickstart guide

2.1 Overview

Ready to get started with the Percona Operator for PostgreSQL? In this section, you will learn some basic operations, such as:

- Install and deploy an Operator
- Connect to PostgreSQL
- Insert sample data to the database
- Set up and make a manual backup
- Monitor the database health with PMM

2.1.1 Next steps

Install the Operator →

2.1.2 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-09-14

Percona LLC and/or its affiliates, © 2009 - 2024

2.2 1 Quick install

2.2.1 Install Percona Distribution for PostgreSQL using kubectl

A Kubernetes Operator is a special type of controller introduced to simplify complex deployments. The Operator extends the Kubernetes API with custom resources.

The Percona Operator for PostgreSQL is based on best practices for configuration and setup of a Percona Distribution for PostgreSQL cluster in a Kubernetes-based environment on-premises or in the cloud.

We recommend installing the Operator with the kubectl command line utility. It is the universal way to interact with Kubernetes. Alternatively, you can install it using the Helm package manager.

Prerequisites

To install Percona Distribution for PostgreSQL, you need the following:

1. The **kubectl** tool to manage and deploy applications on Kubernetes, included in most Kubernetes distributions. Install not already installed, follow its official installation instructions.
2. A Kubernetes environment. You can deploy it on Minikube for testing purposes or using any cloud provider of your choice. Check the list of our officially supported platforms.

See also

- Set up Minikube
- Create and configure the GKE cluster
- Set up Amazon Elastic Kubernetes Service

Percona LLC and/or its affiliates, © 2009 - 2024

Procedure

Here's a sequence of steps to follow:

Percona LLC and/or its affiliates, © 2009 - 2024

1. Create the Kubernetes namespace for your cluster. It is a good practice to isolate workloads in Kubernetes by installing the Operator in a custom namespace. For example, let's name it `postgres-operator`:

   ```
   $ kubectl create namespace postgres-operator
   ```

   Expected output

   ```
   namespace/postgres-operator was created
   ```

   We will use this namespace further on in this document. If you used another name, make sure to replace it in the following commands.

2. Deploy the Operator using the following command:

   ```
   $ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-postgresql-operator/v2.3.1/deploy/bundle.yaml -n postgres-operator
   ```

   Expected output

   ```
   customresourcedefinition.apiextensions.k8s.io/perconapgbackups.pgv2.percona.com serverside-applied
   customresourcedefinition.apiextensions.k8s.io/perconapgclusters.pgv2.percona.com serverside-applied
   customresourcedefinition.apiextensions.k8s.io/perconapgrestores.pgv2.percona.com serverside-applied
   customresourcedefinition.apiextensions.k8s.io/postgresclusters.postgres-operator.crunchydata.com serverside-applied
   serviceaccount/percona-postgresql-operator serverside-applied
   role.rbac.authorization.k8s.io/percona-postgresql-operator serverside-applied
   rolebinding.rbac.authorization.k8s.io/service-account-percona-postgresql-operator serverside-applied
   deployment.apps/percona-postgresql-operator serverside-applied
   ```

   At this point, the Operator Pod is up and running.

3. Deploy Percona Distribution for PostgreSQL cluster:

   ```
   $ kubectl apply -f https://raw.githubusercontent.com/percona/percona-postgresql-operator/v2.3.1/deploy/cr.yaml -n postgres-operator
   ```

   Expected output

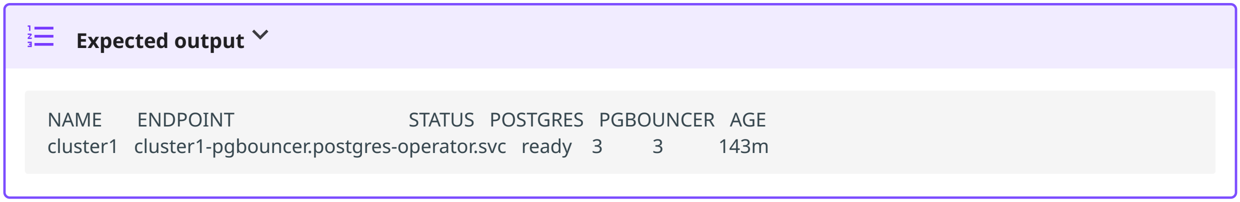
   ```
   perconapgcluster.pgv2.percona.com/cluster1 created
   ```

4. Check the Operator and replica set Pods status.

   ```
   $ kubectl get pg -n postgres-operator
   ```

   It may take some time to create the Operator. The creation process is over when both the Operator and replica set Pods report the `ready` status:

Percona LLC and/or its affiliates, © 2009 - 2024



Expected output

```
NAME      ENDPOINT                                STATUS   POSTGRES   PGBOUNCER   AGE
cluster1  cluster1-pgbouncer.postgres-operator.svc   ready    3          3           143m
```

You have successfully installed and deployed the Operator with default parameters. You can check them in the Custom Resource options reference.

Next steps

Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

2.2.2 Install Percona Distribution for PostgreSQL using Helm

Helm is the package manager for Kubernetes. A Helm chart is a package that contains all the necessary resources to deploy an application to a Kubernetes cluster.

You can find Percona Helm charts in percona/percona-helm-charts repository in Github.

Prerequisites

To install and deploy the Operator, you need the following:

1. Helm v3.
2. kubectl command line utility.
3. A Kubernetes environment. You can deploy it locally on Minikube for testing purposes or using any cloud provider of your choice. Check the list of our officially supported platforms.

See also

- Set up Minikube
- Create and configure the GKE cluster
- Set up Amazon Elastic Kubernetes Service

Percona LLC and/or its affiliates, © 2009 - 2024

Installation

Here's a sequence of steps to follow:

1. Add the Percona's Helm charts repository and make your Helm client up to date with it:

```
$ helm repo add percona https://percona.github.io/percona-helm-charts/
$ helm repo update
```

2. It is a good practice to isolate workloads in Kubernetes via namespaces. Create a namespace:

```
$ kubectl create namespace <my-namespace>
```

3. Install the Percona Operator for PostgreSQL:

```
$ helm install my-operator percona/pg-operator --namespace <my-namespace>
```

The `my-namespace` is the name of your namespace. The `my-operator` parameter is the name of a new release object which is created for the Operator when you install its Helm chart (use any name you like).

4. Install Percona Distribution for PostgreSQL:

```
$ helm install cluster1 percona/pg-db -n <my-namespace>
```

The `cluster1` parameter is the name of a new release object which is created for the Percona Distribution for PostgreSQL when you install its Helm chart (use any name you like).

5. Check the Operator and replica set Pods status.

```
$ kubectl get pg -n <my-namespace>
```

The creation process is over when both the Operator and replica set Pods report the `ready` status:

Expected output

```
NAME      ENDPOINT                               STATUS   POSTGRES   PGBOUNCER   AGE
cluster1  cluster1-pgbouncer.postgres-operator.svc   ready    3          3           143m
```

You have successfully installed and deployed the Operator with default parameters. You can check them in the Custom Resource options reference.

Next steps

Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Percona LLC and/or its affiliates, © 2009 - 2024

Last update: 2023-10-04

Percona LLC and/or its affiliates, © 2009 - 2024

2.3 2 Connect to the PostgreSQL cluster

When the installation is done, we can connect to the cluster.

The pgBouncer component of Percona Distribution for PostgreSQL provides the point of entry to the PostgreSQL cluster. We will use the pgBouncer URI to connect.

The pgBouncer URI is stored in the Secret object, which the Operator generates during the installation.

To connect to PostgreSQL, do the following:

1. List the Secrets objects

```
$ kubectl get secrets -n <namespace>
```

The Secrets object we target is named as <cluster_name>-pguser-<cluster_name>. The <cluster_name> value is the name of your Percona Distribution for PostgreSQL Cluster. The default variant is:

cluster1-pguser-cluster1

cluster1-pg-db-pguser-cluster1-pg-db

2. Retrieve the pgBouncer URI from your secret, decode and pass it as the PGBOUNCER_URI environment variable. Replace the <secret>, <namespace> placeholders with your Secret object and namespace accordingly:

```
$ PGBOUNCER_URI=$(kubectl get secret <secret> --namespace <namespace> -o jsonpath='{.data.pgbouncer-uri}' |
base64 --decode)
```

The following example shows how to pass the pgBouncer URI from the default Secret object cluster1-pguser-cluster1:

```
$ PGBOUNCER_URI=$(kubectl get secret cluster1-pguser-cluster1 --namespace <namespace> -o
jsonpath='{.data.pgbouncer-uri}' | base64 --decode)
```

3. Create a Pod where you start a container with Percona Distribution for PostgreSQL and connect to the database. The following command does it, naming the Pod pg-client and connects you to the cluster1 database:

```
$ kubectl run -i --rm --tty pg-client --image=perconalab/percona-distribution-postgresql:16 --restart=Never -- psql
$PGBOUNCER_URI
```

It may take some time to create the Pod and connect to the database. As the result, you should see the following sample output:

```
psql (16)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.
cluster1=>
```

Congratulations! You have connected to your PostgreSQL cluster.

Percona LLC and/or its affiliates, © 2009 - 2024

2.3.1 Next steps

2.3.2 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

2.4 3 Insert sample data

The next step after connecting to the cluster is to insert some sample data to PostgreSQL.

2.4.1 Create a schema

Every database in PostgreSQL has a default schema called `public`. A schema stores database objects like tables, views, indexes and allows organizing them into logical groups.

When you create a table, it ends up in the `public` schema by default. In recent PostgreSQL versions (starting from PostgreSQL 15), non-database owners cannot access the `public` schema. Therefore, you need to create a new schema to insert the data.

Use the following statement to create a schema

```
CREATE SCHEMA demo;
```

2.4.2 Create a table

After you created a schema, all tables you create end up in this schema if not specified otherwise.

At this step, we will create a sample table `Library` as follows:

```
CREATE TABLE LIBRARY(
  ID INTEGER NOT NULL,
  NAME TEXT,
  SHORT_DESCRIPTION TEXT,
  AUTHOR TEXT,
  DESCRIPTION TEXT,
  CONTENT TEXT,
  LAST_UPDATED DATE,
  CREATED DATE
);
```

2.4.3 Insert the data

PostgreSQL does not have the built-in support to generate random data. However, it provides the `random()` function which generates random numbers and `generate_series()` function which generates the series of rows and populates them with the numbers incremented by 1 (by default).

Combine these functions with a couple of others to populate the table with the data:

```
INSERT INTO LIBRARY(id, name, short_description, author,
                    description,content, last_updated, created)
SELECT id, 'name', md5(random()::text), 'name2'
     ,md5(random()::text),md5(random()::text)
     ,NOW() - '1 day'::INTERVAL * (RANDOM()::int * 100)
     ,NOW() - '1 day'::INTERVAL * (RANDOM()::int * 100 + 100)
FROM generate_series(1,100) id;
```


Percona LLC and/or its affiliates, © 2009 - 2024

This command does the following:

- Fills in the columns `id`, `name`, `author` with the values `id`, `name` and `name2` respectively;
- generates the random md5 hash sum as the values for the columns `short_description`, `description` and `content`;
- generates the random number of dates from the current date and time within the last 100 days, and
- inserts 100 rows of this data

Now your cluster has some data in it.

2.4.4 Next steps

2.4.5 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

2.5 4 Make a backup

Now your database contains some data, so it's a good time to learn how to manually make a full backup of your data with the Operator.

Note

If you are interested to learn more about backups, their types and retention policy, see the Backups section.

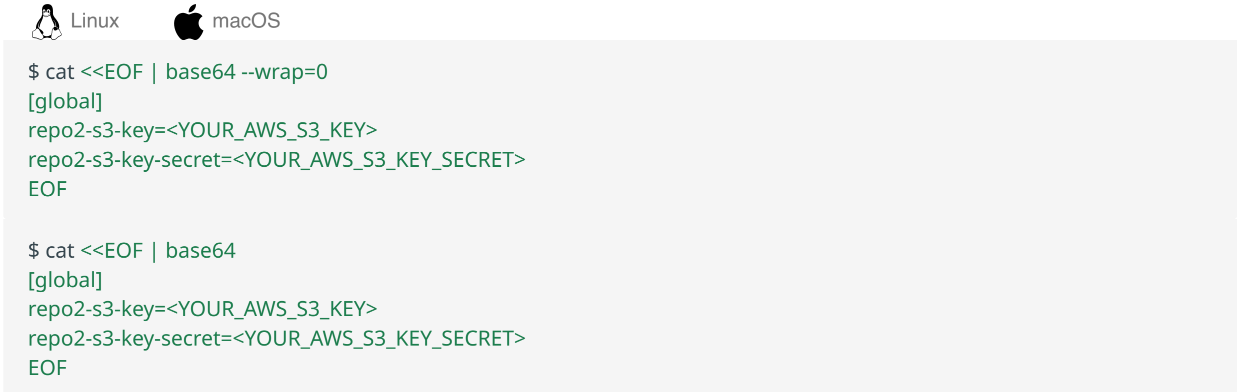
2.5.1 Considerations

- In this tutorial we use AWS S3 as the backup storage. You need the following S3-related information:
- The name of S3 bucket;
- The endpoint - the URL to access the bucket
- The region - the location of the bucket
- S3 credentials such as S3 key and secret to access the storage.

  If you don't have access to AWS, you can use any S3-compatible storage like MinIO. Also check the list of supported storages.
- The Operator uses the `pgBackRest` tool to make backups. `pgBackRest` stores the backups and archives WAL segments in repositories. The Operator has up to four `pgBackRest` repositories named `repo1`, `repo2`, `repo3` and `repo4`. In this tutorial we use `repo2` for backups.

2.5.2 Configure backup storage

1. Encode the S3 credentials and the pgBackRest repository name (`repo2` in our setup).




```
$ cat <<EOF | base64 --wrap=0
[global]
repo2-s3-key=<YOUR_AWS_S3_KEY>
repo2-s3-key-secret=<YOUR_AWS_S3_KEY_SECRET>
EOF

$ cat <<EOF | base64
[global]
repo2-s3-key=<YOUR_AWS_S3_KEY>
repo2-s3-key-secret=<YOUR_AWS_S3_KEY_SECRET>
EOF
```

2. Create the Secret configuration file and specify the base64-encoded string from the previous step. The following is the example of the `cluster1-pgbackrest-secrets.yaml` Secret file:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-pgbackrest-secrets
type: Opaque
data:
  s3.conf: <base64-encoded-configuration-contents>
```


Percona LLC and/or its affiliates, © 2009 - 2024

3. Create the Secrets object from this yaml file. Specify your namespace instead of the `<namespace>` placeholder:

```
$ kubectl apply -f cluster1-pgbackrest-secrets.yaml -n <namespace>
```

4. Update your `deploy/cr.yaml` configuration. Specify the Secret file you created in the `backups.pgbackrest.configuration` subsection, and put all other S3 related information in the `backups.pgbackrest.repos` subsection under the repository name that you intend to use for backups. This name must match the name you used when you encoded S3 credentials on step 1.

   For example, the S3 storage for the `repo2` repository looks as follows:

```
...
backups:
  pgbackrest:
    ...
    configuration:
      - secret:
          name: cluster1-pgbackrest-secrets
    ...
    repos:
    - name: repo2
      s3:
        bucket: "<YOUR_AWS_S3_BUCKET_NAME>"
        endpoint: "<YOUR_AWS_S3_ENDPOINT>"
        region: "<YOUR_AWS_S3_REGION>"
```

5. Create or update the cluster. Specify your namespace instead of the `<namespace>` placeholder:

```
$ kubectl apply -f deploy/cr.yaml
```

2.5.3 Make a backup

For manual backups, you need a backup configuration file.

1. Edit the example backup configuration file deploy/backup.yaml. Specify your cluster name and the `repo` name.

```
apiVersion: pgv2.percona.com/v2
kind: PerconaPGBackup
metadata:
  name: backup1
spec:
  pgCluster: cluster2
  repoName: repo1
#  options:
#  - --type=full
```

2. Apply the configuration. This instructs the Operator to start a backup.

```
$ kubectl apply -f deploy/backup.yaml -n <namespace>
```

3. List the backup

```
$ kubectl get pg-backup -n <namespace>
```


Percona LLC and/or its affiliates, © 2009 - 2024

Congratulations! You have made the first backup manually. Want to learn more about backups? See the Backup and restore section for details like types, retention and how to automatically make backups according to the schedule.

2.5.4 Next steps

2.5.5 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

2.6 5 Monitor the database

Finally, when we are done with backup, it's time for one more step. In this section you will learn how to monitor the health of Percona Distribution for PostgreSQL with Percona Monitoring and Management (PMM).

Note

Only PMM 2.x versions are supported by the Operator.

PMM is a client/server application. It includes the PMM Server and the number of PMM Clients running on each node with the database you wish to monitor.

A PMM Client collects needed metrics and sends gathered data to the PMM Server. As a user, you connect to the PMM Server to see database metrics on a number of dashboards.

PMM Server and PMM Client are installed separately.

2.6.1 Install PMM Server

You must have PMM server up and running. You can run PMM Server as a *Docker image*, a *virtual appliance*, or on an *AWS instance*. Please refer to the official PMM documentation for the installation instructions.

Percona LLC and/or its affiliates, © 2009 - 2024

2.6.2 Install PMM Client

To install PMM Client as a side-car container in your Kubernetes-based environment, do the following:

Percona LLC and/or its affiliates, © 2009 - 2024

1. Get the PMM API key from PMM Server. The API key must have the role "Admin". You need this key to authorize PMM Client within PMM Server.

Generate the PMM API key

You can query your PMM Server installation for the API Key using `curl` and `jq` utilities. Replace `<login>:<password>@<server_host>` placeholders with your real PMM Server login, password, and hostname in the following command:

```
$ API_KEY=$(curl --insecure -X POST -H "Content-Type: application/json" -d '{"name":"operator", "role": "Admin"}' "https://
<login>:<password>@<server_host>/graph/api/auth/keys" | jq .key)
```

Note

The API key is not rotated.

2. Specify the API key as the `PMM_SERVER_KEY` value in the deploy/secrets.yaml secrets file.

```
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-pmm-secret
type: Opaque
stringData:
  PMM_SERVER_KEY: ""
```

3. Create the Secrets object using the `deploy/secrets.yaml` file.

```
$ kubectl apply -f deploy/secrets.yaml -n postgres-operator
```

4. Update the `pmm` section in the deploy/cr.yaml file.

- Set `pmm.enabled = true` .
- Specify your PMM Server hostname / an IP address for the `pmm.serverHost` option. The PMM Server IP address should be resolvable and reachable from within your cluster.

```
  pmm:
    enabled: true
    image: percona/pmm-client:2.41.0
#   imagePullPolicy: IfNotPresent
    secret: cluster1-pmm-secret
    serverHost: monitoring-service
```

5. Update the cluster

```
$ kubectl apply -f deploy/cr.yaml -n postgres-operator
```

6. Check that corresponding Pods are not in a cycle of stopping and restarting. This cycle occurs if there are errors on the previous steps:

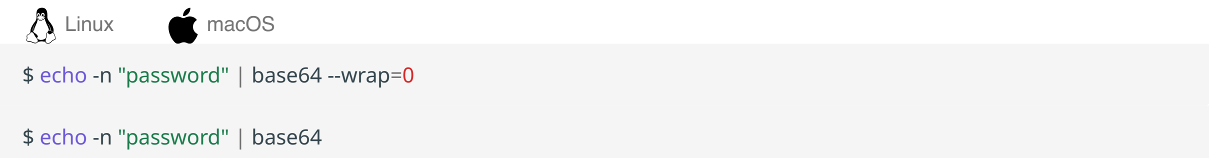
Percona LLC and/or its affiliates, © 2009 - 2024

```
$ kubectl get pods -n postgres-operator
$ kubectl logs <pod_name> -c pmm-client
```

2.6.3 Update the secrets file

The `deploy/secrets.yaml` file contains all values for each key/value pair in a convenient plain text format. But the resulting Secrets Objects contains passwords stored as base64-encoded strings. If you want to *update* the password field, you need to encode the new password into the base64 format and pass it to the Secrets Object.

To encode a password or any other parameter, run the following command:



Linux macOS

```
$ echo -n "password" | base64 --wrap=0
```

```
$ echo -n "password" | base64
```

For example, to set the new PMM API key in the `my-cluster-name-secrets` object, do the following:

Linux macOS

```
$ kubectl patch secret/cluster1-pmm-secret -p '{"data":{"PMM_SERVER_KEY": '$(echo -n new_key | base64 --wrap=0)'}}'
```

```
$ kubectl patch secret/cluster1-pmm-secret -p '{"data":{"PMM_SERVER_KEY": '$(echo -n new_key | base64)'}}'
```

2.6.4 Check the metrics

Let's see how the collected data is visualized in PMM.

1. Log in to PMM server.
2. Click **PostgreSQL** from the left-hand navigation menu. You land on the **Instances Overview** page.
3. Click **PostgreSQL** → **Other dashboards** to see the list of available dashboards that allow you to drill down to the metrics you are interested in.

2.6.5 Next steps

What's next →

2.6.6 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

2.7 What's next?

Congratulations! You have completed all the steps in the Get started guide.

You have the following options to move forward with the Operator:

- Deepen your monitoring insights by setting up Kubernetes monitoring with PMM
- Control Pods assignment on specific Kubernetes Nodes by setting up affinity / anti-affinity
- Ready to adopt the Operator for production use and need to delete the testing deployment? Use this guide to do it.

2.7.1 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Last update: 2023-09-14

Percona LLC and/or its affiliates, © 2009 - 2024

3. Installation

3.1 System requirements

The Operator is validated for deployment on Kubernetes, GKE and EKS clusters. The Operator is cloud native and storage agnostic, working with a wide variety of storage classes, hostPath, and NFS.

3.1.1 Supported versions

The Operator 2.3.1 is developed, tested and based on:

- PostgreSQL 12.17, 13.13, 14.10, 15.5, and 16.1 as the database. Other versions may also work but have not been tested.
- pgBouncer 1.21.0 for connection pooling
- Patroni 3.1.0 for high-availability.

3.1.2 Supported platforms

The following platforms were tested and are officially supported by the Operator 2.3.1:

- Google Kubernetes Engine (GKE) 1.24 - 1.28
- Amazon Elastic Container Service for Kubernetes (EKS) 1.24 - 1.28
- OpenShift 4.11.55 - 4.14.6
- Minikube 1.32

Other Kubernetes platforms may also work but have not been tested.

3.1.3 Installation guidelines

Choose how you wish to install Percona Operator for PostgreSQL:

- with Helm
- with `kubectl`
- on Minikube
- on Google Kubernetes Engine (GKE)
- on Amazon Elastic Kubernetes Service (AWS EKS)
- in a Kubernetes-based environment

3.1.4 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

3.2 Install Percona Distribution for PostgreSQL on Minikube

Installing the Percona Operator for PostgreSQL on Minikube is the easiest way to try it locally without a cloud provider.

Minikube runs Kubernetes on GNU/Linux, Windows, or macOS system using a system-wide hypervisor, such as VirtualBox, KVM/QEMU, VMware Fusion or Hyper-V. Using it is a popular way to test Kubernetes application locally prior to deploying it on a cloud.

This document describes how to deploy the Operator and Percona Distribution for PostgreSQL on Minikube.

3.2.1 Set up Minikube

1. Install Minikube, using a way recommended for your system. This includes the installation of the following three components:

a. kubectl tool,

b. a hypervisor, if it is not already installed,

c. actual minikube package

2. After the installation, initialize and start the Kubernetes cluster. The parameters we pass for the following command increase the virtual machine limits for the CPU cores, memory, and disk, to ensure stable work of the Operator:

```
$ minikube start --memory=5120 --cpus=4 --disk-size=30g
```

This command downloads needed virtualized images, then initializes and runs the cluster.

3. After Minikube is successfully started, you can optionally run the Kubernetes dashboard, which visually represents the state of your cluster. Executing `minikube dashboard` starts the dashboard and opens it in your default web browser.

3.2.2 Deploy the Percona Operator for PostgreSQL

1. Deploy the Operator using the following command:

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-postgresql-operator/v2.3.1/deploy/
bundle.yaml
```

Expected output

```
customresourcedefinition.apiextensions.k8s.io/perconapgbackups.pgv2.percona.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconapgclusters.pgv2.percona.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconapgrestores.pgv2.percona.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/postgresclusters.postgres-operator.crunchydata.com serverside-applied
serviceaccount/percona-postgresql-operator serverside-applied
role.rbac.authorization.k8s.io/percona-postgresql-operator serverside-applied
rolebinding.rbac.authorization.k8s.io/service-account-percona-postgresql-operator serverside-applied
deployment.apps/percona-postgresql-operator serverside-applied
```

As the result you have the Operator Pod up and running.

2. Deploy Percona Distribution for PostgreSQL:

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-postgresql-operator/v2.3.1/deploy/cr.yaml
```


Percona LLC and/or its affiliates, © 2009 - 2024

Expected output

```
perconapgcluster.pgv2.percona.com/cluster1 created
```

Note

This deploys default Percona Distribution for PostgreSQL configuration. Please see deploy/cr.yaml and Custom Resource Options for the configuration options. You can clone the repository with all manifests and source code by executing the following command:

```
$ git clone -b v2.3.1 https://github.com/percona/percona-postgresql-operator
```

After editing the needed options, apply your modified `deploy/cr.yaml` file as follows:

```
$ kubectl apply -f deploy/cr.yaml
```

3. The creation process may take some time. When the process is over your cluster will obtain the `ready` status. You can check it with the following command:

```
$ kubectl get pg -n postgres-operator
```

Expected output

```
NAME      ENDPOINT                        STATUS   POSTGRES   PGBOUNCER   AGE
cluster1  cluster1-pgbouncer.default.svc  ready    3          3           30m
```

3.2.3 Verify the Percona Distribution for PostgreSQL cluster operation

When creation process is over, you can try to connect to the cluster.

Percona LLC and/or its affiliates, © 2009 - 2024

During the installation, the Operator has generated several secrets, including the one with password for default PostgreSQL user. This default user has the same login name as the cluster name.

1. Use `kubectl get secrets` command to see the list of Secrets objects. The Secrets object you are interested in is named as `<cluster_name>-pguser-<cluster_name>` (substitute `<cluster_name>` with the name of your Percona Distribution for PostgreSQL Cluster). The default variant will be `cluster1-pguser-cluster1`.
2. Use the following command to get the password of this user:

```
$ kubectl get secret <cluster_name>-<user_name>-<cluster_name> -n postgres-operator --template='{{.data.password | base64decode}}{{"\n"}}'
```

3. Create a pod and start Percona Distribution for PostgreSQL inside. The following command will do this, naming the new Pod `pg-client`:

```
$ kubectl run -i --rm --tty pg-client --image=perconalab/percona-distribution-postgresql:16 --restart=Never -- bash -il
```

Executing it may require some time to deploy the corresponding Pod.

4. Run a container with `psql` tool and connect its console output to your terminal. This command will connect you as a `cluster1` user to a `cluster1` database via the PostgreSQL interactive terminal.

```
[postgres@pg-client /]$ PGPASSWORD='pguser_password' psql -h cluster1-pgbouncer.postgres-operator.svc -p 5432 -U cluster1 cluster1
```

Sample output

```
psql (16)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.
pgdb=>
```

3.2.4 Delete the cluster

If you need to delete the Operator and PostgreSQL cluster (for example, to clean up the testing deployment before adopting it for production use), check this HowTo.

If you no longer need the Kubernetes cluster in Minikube, the following are the steps to remove it.

1. Stop the Minikube cluster:

```
$ minikube stop
```

2. Delete the cluster

```
$ minikube delete
```

This command deletes the virtual machines, and removes all associated files.

Percona LLC and/or its affiliates, © 2009 - 2024

3.2.5 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

3.3 Install Percona Distribution for PostgreSQL on Google Kubernetes Engine (GKE)

Following steps help you install the Operator and use it to manage Percona Distribution for PostgreSQL with the Google Kubernetes Engine. The document assumes some experience with Google Kubernetes Engine (GKE). For more information on GKE, see the Kubernetes Engine Quickstart.

3.3.1 Prerequisites

All commands from this installation guide can be run either in the **Google Cloud shell** or in **your local shell**.

To use *Google Cloud shell*, you need nothing but a modern web browser.

If you would like to use *your local shell*, install the following:

1. gcloud. This tool is part of the Google Cloud SDK. To install it, select your operating system on the official Google Cloud SDK documentation page and then follow the instructions.
2. kubectl. This is the Kubernetes command-line tool you will use to manage and deploy applications. To install the tool, run the following command:

```
$ gcloud auth login
$ gcloud components install kubectl
```

3.3.2 Create and configure the GKE cluster

You can configure the settings using the `gcloud` tool. You can run it either in the Cloud Shell or in your local shell (if you have installed Google Cloud SDK locally on the previous step). The following command creates a cluster named `cluster-1`:

```
$ gcloud container clusters create cluster-1 --project <project name> --zone us-central1-a --cluster-version  --machine-type n1-standard-4 --num-nodes=3
```

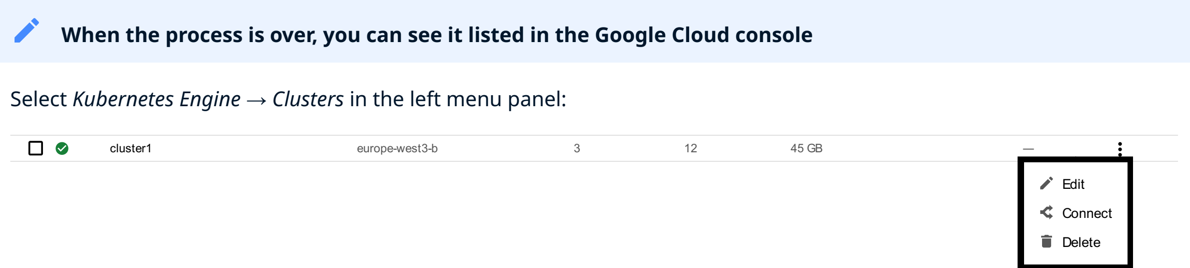
Note

You must edit the above command and other command-line statements to replace the `<project name>` placeholder with your project name. You may also be required to edit the *zone location*, which is set to `us-central1` in the above example. Other parameters specify that we are creating a cluster with 3 nodes and with machine type of 4 vCPUs and 45 GB memory.

You may wait a few minutes for the cluster to be generated.

When the process is over, you can see it listed in the Google Cloud console

Select *Kubernetes Engine* → *Clusters* in the left menu panel:



Percona LLC and/or its affiliates, © 2009 - 2024

Now you should configure the command-line access to your newly created cluster to make `kubectl` be able to use it.

In the Google Cloud Console, select your cluster and then click the *Connect* shown on the above image. You will see the connect statement which configures the command-line access. After you have edited the statement, you may run the command in your local shell:

```
$ gcloud container clusters get-credentials cluster-1 --zone us-central1-a --project <project name>
```

Finally, use your Cloud Identity and Access Management (Cloud IAM) to control access to the cluster. The following command will give you the ability to create Roles and RoleBindings:

```
$ kubectl create clusterrolebinding cluster-admin-binding --clusterrole cluster-admin --user $(gcloud config get-value core/account)
```

Expected output

```
clusterrolebinding.rbac.authorization.k8s.io/cluster-admin-binding created
```

3.3.3 Install the Operator and deploy your PostgreSQL cluster

1. First of all, use the following `git clone` command to download the correct branch of the percona-postgresql-operator repository:

```
$ git clone -b v2.3.1 https://github.com/percona/percona-postgresql-operator
$ cd percona-postgresql-operator
```

2. Create the Kubernetes namespace for your cluster if needed (for example, let's name it `postgres-operator`):

```
$ kubectl create namespace postgres-operator
```

Expected output

```
namespace/postgres-operator was created
```

Note

To use different namespace, specify other name instead of `postgres-operator` in the above command, and modify the `-n postgres-operator` parameter with it in the following steps. You can also omit this parameter completely to deploy everything in the `default` namespace.

3. Deploy the Operator using the following command:

```
$ kubectl apply --server-side -f deploy/bundle.yaml -n postgres-operator
```


Percona LLC and/or its affiliates, © 2009 - 2024

Expected output

```
customresourcedefinition.apiextensions.k8s.io/perconapgbackups.pgv2.percona.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconapgclusters.pgv2.percona.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconapgrestores.pgv2.percona.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/postgresclusters.postgres-operator.crunchydata.com serverside-applied
serviceaccount/percona-postgresql-operator serverside-applied
role.rbac.authorization.k8s.io/percona-postgresql-operator serverside-applied
rolebinding.rbac.authorization.k8s.io/service-account-percona-postgresql-operator serverside-applied
deployment.apps/percona-postgresql-operator serverside-applied
```

As the result you will have the Operator Pod up and running.

4. Deploy Percona Distribution for PostgreSQL:

```
$ kubectl apply -f deploy/cr.yaml -n postgres-operator
```

Expected output

```
perconapgcluster.pgv2.percona.com/cluster1 created
```

The creation process may take some time. When the process is over your cluster will obtain the `ready` status. You can check it with the following command:

```
$ kubectl get pg -n postgres-operator
```

Expected output

```
NAME      ENDPOINT                      STATUS   POSTGRES   PGBOUNCER   AGE
cluster1  cluster1-pgbouncer.default.svc   ready    3          3           30m
```

You can also track the creation process in Google Cloud console via the Object Browser

When the creation process is finished, it will look as follows:

Name	Status	Type	Pods	Namespace	Cluster
cluster1-backup-7hsq	OK	Job	0/1	pg-opertor	cluster1
cluster1-instance1-mntz	OK	Stateful Set	1/1	pg-opertor	cluster1
cluster1-pgbouncer	OK	Deployment	1/1	pg-opertor	cluster1
cluster1-repo-host	OK	Stateful Set	1/1	pg-opertor	cluster1
cluster1-repo1-full	OK	Cron Job	0/0	pg-opertor	cluster1
percona-postgresql-operator	OK	Deployment	1/1	pg-opertor	cluster1

Percona LLC and/or its affiliates, © 2009 - 2024

3.3.4 Verifying the cluster operation

When creation process is over, `kubectl get pg -n <namespace>` command will show you the cluster status as `ready`, and you can try to connect to the cluster.

During the installation, the Operator has generated several secrets, including the one with password for default PostgreSQL user. This default user has the same login name as the cluster name.

1. Use `kubectl get secrets` command to see the list of Secrets objects. The Secrets object you are interested in is named as `<cluster_name>-pguser-<cluster_name>` (substitute `<cluster_name>` with the name of your Percona Distribution for PostgreSQL Cluster). The default variant will be `cluster1-pguser-cluster1`.
2. Use the following command to get the password of this user:

```
$ kubectl get secret <cluster_name>-<user_name>-<cluster_name> -n postgres-operator --template='{{.data.password |
base64decode}}{{"\n"}}'
```

3. Create a pod and start Percona Distribution for PostgreSQL inside. The following command will do this, naming the new Pod `pg-client`:

```
$ kubectl run -i --rm --tty pg-client --image=perconalab/percona-distribution-postgresql:16 --restart=Never -- bash -il
```

   Executing it may require some time to deploy the corresponding Pod.

4. Run a container with `psql` tool and connect its console output to your terminal. This command will connect you as a `cluster1` user to a `cluster1` database via the PostgreSQL interactive terminal.

```
[postgres@pg-client /]$ PGPASSWORD='pguser_password' psql -h cluster1-pgbouncer.postgres-operator.svc -p 5432 -U
cluster1 cluster1
```

Sample output

```
psql (16)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.
pgdb=>
```

3.3.5 Removing the cluster

If you need to delete the Operator and PostgreSQL cluster (for example, to clean up the testing deployment before adopting it for production use), check this HowTo.

Also, there are several ways that you can delete your Kubernetes cluster in GKE.

You can clean up the cluster with the `gcloud` command as follows:

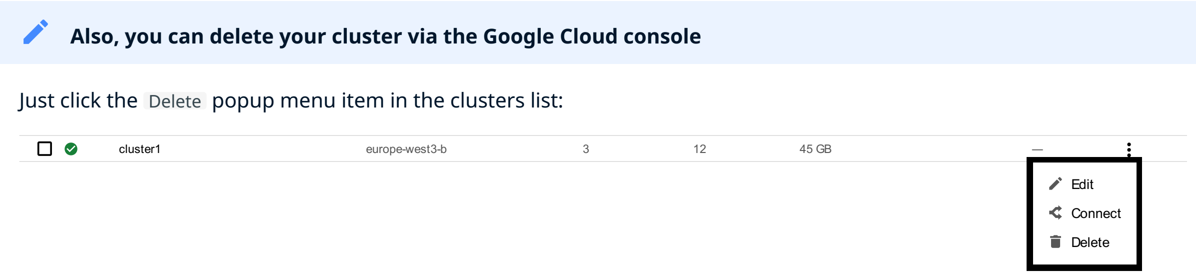
```
$ gcloud container clusters delete <cluster name>
```

The return statement requests your confirmation of the deletion. Type `y` to confirm.

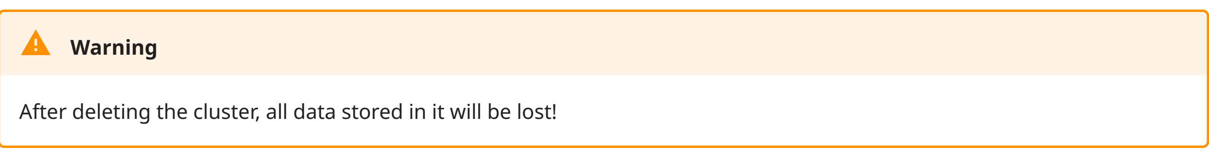
Percona LLC and/or its affiliates, © 2009 - 2024

Also, you can delete your cluster via the Google Cloud console

Just click the `Delete` popup menu item in the clusters list:



The cluster deletion may take time.



Warning

After deleting the cluster, all data stored in it will be lost!

3.3.6 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

3.4 Install Percona Distribution for PostgreSQL on Amazon Elastic Kubernetes Service (EKS)

This guide shows you how to deploy Percona Operator for PostgreSQL on Amazon Elastic Kubernetes Service (EKS). The document assumes some experience with the platform. For more information on the EKS, see the Amazon EKS official documentation.

3.4.1 Prerequisites

Software installation

The following tools are used in this guide and therefore should be preinstalled:

1. **AWS Command Line Interface (AWS CLI)** for interacting with the different parts of AWS. You can install it following the official installation instructions for your system.
2. **eksctl** to simplify cluster creation on EKS. It can be installed along its installation notes on GitHub.
3. **kubectl** to manage and deploy applications on Kubernetes. Install it following the official installation instructions.

Also, you need to configure AWS CLI with your credentials according to the official guide.

Creating the EKS cluster

1. To create your cluster, you will need the following data:
- name of your EKS cluster,
- AWS region in which you wish to deploy your cluster,
- the amount of nodes you would like tho have,
- the desired ratio between on-demand and spot instances in the total number of nodes.

Note

spot instances are not recommended for production environment, but may be useful e.g. for testing purposes.

After you have settled all the needed details, create your EKS cluster following the official cluster creation instructions.

2. After you have created the EKS cluster, you also need to install the Amazon EBS CSI driver on your cluster. See the official documentation on adding it as an Amazon EKS add-on.

Note

CSI driver is needed for the Operator to work propely, and is not included by default starting from the Amazon EKS version 1.22. Therefore sers with existing EKS cluster based on the version 1.22 or earlier need to install CSI driver before updating the EKS cluster to 1.23 or above.

Percona LLC and/or its affiliates, © 2009 - 2024

3.4.2 Install the Operator and Percona Distribution for PostgreSQL

The following steps are needed to deploy the Operator and Percona Distribution for PostgreSQL in your Kubernetes environment:

Percona LLC and/or its affiliates, © 2009 - 2024

1. Create the Kubernetes namespace for your cluster if needed (for example, let's name it `postgres-operator`):

```
$ kubectl create namespace postgres-operator
```

Expected output

```
namespace/postgres-operator was created
```

Note

To use different namespace, specify other name instead of `postgres-operator` in the above command, and modify the `-n postgres-operator` parameter with it in the following two steps. You can also omit this parameter completely to deploy everything in the `default` namespace.

2. Deploy the Operator using the following command:

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-postgresql-operator/v2.3.1/deploy/
bundle.yaml -n postgres-operator
```

Expected output

```
customresourcedefinition.apiextensions.k8s.io/perconapgbackups.pgv2.percona.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconapgclusters.pgv2.percona.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconapgrestores.pgv2.percona.com serverside-applied
customresourcedefinition.apiextensions.k8s.io/postgresclusters.postgres-operator.crunchydata.com serverside-applied
serviceaccount/percona-postgresql-operator serverside-applied
role.rbac.authorization.k8s.io/percona-postgresql-operator serverside-applied
rolebinding.rbac.authorization.k8s.io/service-account-percona-postgresql-operator serverside-applied
deployment.apps/percona-postgresql-operator serverside-applied
```

As the result you will have the Operator Pod up and running.

3. The operator has been started, and you can deploy your Percona Distribution for PostgreSQL cluster:

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-postgresql-operator/v2.3.1/deploy/cr.yaml -n
postgres-operator
```

Expected output

```
perconapgcluster.pgv2.percona.com/cluster1 created
```


Percona LLC and/or its affiliates, © 2009 - 2024

Note

This deploys default Percona Distribution for PostgreSQL configuration. Please see deploy/cr.yaml and Custom Resource Options for the configuration options. You can clone the repository with all manifests and source code by executing the following command:

```
$ git clone -b v2.3.1 https://github.com/percona/percona-postgresql-operator
```

After editing the needed options, apply your modified `deploy/cr.yaml` file as follows:

```
$ kubectl apply -f deploy/cr.yaml -n postgres-operator
```

The creation process may take some time. When the process is over your cluster will obtain the `ready` status. You can check it with the following command:

```
$ kubectl get pg
```

Expected output

```
NAME      ENDPOINT                       STATUS   POSTGRES   PGBOUNCER   AGE
cluster1  cluster1-pgbouncer.default.svc ready    3          3           30m
```

3.4.3 Verifying the cluster operation

When creation process is over, `kubectl get pg` command will show you the cluster status as `ready`, and you can try to connect to the cluster.

Percona LLC and/or its affiliates, © 2009 - 2024

During the installation, the Operator has generated several secrets, including the one with password for default PostgreSQL user. This default user has the same login name as the cluster name.

1. Use `kubectl get secrets` command to see the list of Secrets objects. The Secrets object you are interested in is named as `<cluster_name>-pguser-<cluster_name>` (substitute `<cluster_name>` with the name of your Percona Distribution for PostgreSQL Cluster). The default variant will be `cluster1-pguser-cluster1`.
2. Use the following command to get the password of this user:

```
$ kubectl get secret <cluster_name>-<user_name>-<cluster_name> -n postgres-operator --template='{{.data.password |
base64decode}}{{"\n"}}'
```

3. Create a pod and start Percona Distribution for PostgreSQL inside. The following command will do this, naming the new Pod `pg-client`:

```
$ kubectl run -i --rm --tty pg-client --image=perconalab/percona-distribution-postgresql:16 --restart=Never -- bash -il
```

Executing it may require some time to deploy the corresponding Pod.

4. Run a container with `psql` tool and connect its console output to your terminal. This command will connect you as a `cluster1` user to a `cluster1` database via the PostgreSQL interactive terminal.

```
[postgres@pg-client /]$ PGPASSWORD='pguser_password' psql -h cluster1-pgbouncer.postgres-operator.svc -p 5432 -U
cluster1 cluster1
```

Sample output

```
psql (16)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.
pgdb=>
```

3.4.4 Removing the cluster

If you need to delete the Operator and PostgreSQL cluster (for example, to clean up the testing deployment before adopting it for production use), check this HowTo.

To delete your Kubernetes cluster in EKS, you will need the following data:

- name of your EKS cluster,
- AWS region in which you have deployed your cluster.

You can clean up the cluster with the `eksctl` command as follows (with real names instead of `<region>` and `<cluster name>` placeholders):

```
$ eksctl delete cluster --region=<region> --name="<cluster name>"
```

The cluster deletion may take time.

Warning

After deleting the cluster, all data stored in it will be lost!

Percona LLC and/or its affiliates, © 2009 - 2024

3.4.5 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

3.5 Install Percona Distribution for PostgreSQL on OpenShift

Percona Operator for PostgreSQL is a Red Hat Certified Operator. This means that Percona Operator is portable across hybrid clouds and fully supports the Red Hat OpenShift lifecycle.

Installing Percona Distribution for PostgreSQL on OpenShift includes two steps:

- Installing the Percona Operator for PostgreSQL,
- Install Percona Distribution for PostgreSQL using the Operator.

3.5.1 Install the Operator

You can install Percona Operator for PostgreSQL on OpenShift using the Red Hat Marketplace web interface or using the command line interface.

Install the Operator via the command-line interface

1. First of all, clone the percona-postgresql-operator repository:

```
$ git clone -b v2.3.1 https://github.com/percona/percona-postgresql-operator
$ cd percona-postgresql-operator
```

Note

It is crucial to specify the right branch with `-b` option while cloning the code on this step. Please be careful.

2. The Custom Resource Definition for Percona Distribution for PostgreSQL should be created from the `deploy/crd.yaml` file. Custom Resource Definition extends the standard set of resources which OpenShift “knows” about with the new items (in our case ones which are the core of the Operator). Apply it as follows:

```
$ oc apply --server-side -f deploy/crd.yaml
```

This step should be done only once; it does not need to be repeated with any other Operator deployments.

3. Create the OpenShift namespace for your cluster if needed (for example, let’s name it `postgres-operator`):

```
$ oc create namespace postgres-operator
```

Note

To use different namespace, specify other name instead of `postgres-operator` in the above command, and modify the `-n postgres-operator` parameter with it in the following two steps. You can also omit this parameter completely to deploy everything in the `default` namespace.

4. The role-based access control (RBAC) for Percona Distribution for PostgreSQL is configured with the `deploy/rbac.yaml` file. Role-based access is based on defined roles and the available actions which correspond to each role. The role and actions are defined for Kubernetes resources in the yaml file. Further details about users and roles can be found in specific OpenShift documentation)

```
$ oc apply -f deploy/rbac.yaml -n postgres-operator
```


Percona LLC and/or its affiliates, © 2009 - 2024

Note

Setting RBAC requires your user to have cluster-admin role privileges. For example, those using Google OpenShift Engine can grant user needed privileges with the following command:

```
$ oc create clusterrolebinding cluster-admin-binding --clusterrole=cluster-admin --user=$(gcloud config get-value core/account)
```

5. If you are going to use the operator with anyuid https://docs.openshift.com/container-platform/4.9/authentication/managing-security-context-constraints.html security context constraint please execute the following command:

```
$ sed -i '/disable_auto_failover: "false"/a \ \ \ \ disable_fsgroup: "false"' deploy/operator.yaml
```

6. Start the Operator within OpenShift:

```
$ oc apply -f deploy/operator.yaml -n postgres-operator
```

Optionally, you can add PostgreSQL Users secrets and TLS certificates to OpenShift. If you don't, the Operator will create the needed users and certificates automatically, when you create the database cluster. You can see documentation on Users and TLS certificates if still want to create them yourself.

Note

You can simplify the Operator installation by applying a single `deploy/bundle.yaml` file instead of running commands from the steps 2 and 4:

```
$ oc apply -f deploy/bundle.yaml
```

This will automatically create Custom Resource Definition, set up role-based access control and install the Operator as one single action.

7. After the Operator is started Percona Distribution for PostgreSQL cluster can be created at any time with the following command:

```
$ oc apply -f deploy/cr.yaml -n postgres-operator
```

Creation process will take some time. The process is over when both Operator and replica set Pods have reached their Running status:

```
$ oc get pg -n postgres-operator
```

Expected output

```
NAME     ENDPOINT                              STATUS  POSTGRES  PGBOUNCER  AGE
cluster1  cluster1-pgbouncer.postgres-operator.svc  ready   3         3          143m
```

3.5.2 Verifying the cluster operation

When creation process is over, `oc get pg` command will show you the cluster status as `ready`, and you can try to connect to the cluster.

Percona LLC and/or its affiliates, © 2009 - 2024

During the installation, the Operator has generated several secrets, including the one with password for default PostgreSQL user. This default user has the same login name as the cluster name.

1. Use `oc get secrets` command to see the list of Secrets objects. The Secrets object you are interested in is named as `<cluster_name>-pguser-<cluster_name>` (substitute `<cluster_name>` with the name of your Percona Distribution for PostgreSQL Cluster). The default variant will be `cluster1-pguser-cluster1`.
2. Use the following command to get the password of this user:

```
$ oc get secret <cluster_name>-<user_name>-<cluster_name> -n postgres-operator --template='{{.data.password |
base64decode}}{{"\n"}}'
```

3. Create a pod and start Percona Distribution for PostgreSQL inside. The following command will do this, naming the new Pod `pg-client`:

```
$ oc run -i --rm --tty pg-client --image=perconalab/percona-distribution-postgresql:16 --restart=Never -- bash -il
```

Executing it may require some time to deploy the corresponding Pod.

4. Run a container with `psql` tool and connect its console output to your terminal. This command will connect you as a `cluster1` user to a `cluster1` database via the PostgreSQL interactive terminal.

```
[postgres@pg-client /]$ PGPASSWORD='pguser_password' psql -h cluster1-pgbouncer.postgres-operator.svc -p 5432 -U
cluster1 cluster1
```

Sample output

```
psql (16)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.
pgdb=>
```

3.5.3 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

3.6 Install Percona Distribution for PostgreSQL on Kubernetes

Following steps will allow you to install the Operator and use it to manage Percona Distribution for PostgreSQL in a Kubernetes-based environment.

Percona LLC and/or its affiliates, © 2009 - 2024

1. First of all, clone the percona-postgresql-operator repository:

```
$ git clone -b v2.3.1 https://github.com/percona/percona-postgresql-operator
$ cd percona-postgresql-operator
```

Note

It is crucial to specify the right branch with `-b` option while cloning the code on this step. Please be careful.

2. The Custom Resource Definition for Percona Distribution for PostgreSQL should be created from the `deploy/crd.yaml` file. Custom Resource Definition extends the standard set of resources which Kubernetes "knows" about with the new items (in our case ones which are the core of the Operator). Apply it as follows:

```
$ kubectl apply --server-side -f deploy/crd.yaml
```

This step should be done only once; it does not need to be repeated with any other Operator deployments.

3. Create the Kubernetes namespace for your cluster if needed (for example, let's name it `postgres-operator`):

```
$ kubectl create namespace postgres-operator
```

Note

To use different namespace, specify other name instead of `postgres-operator` in the above command, and modify the `-n postgres-operator` parameter with it in the following two steps. You can also omit this parameter completely to deploy everything in the `default` namespace.

4. The role-based access control (RBAC) for Percona Distribution for PostgreSQL is configured with the `deploy/rbac.yaml` file. Role-based access is based on defined roles and the available actions which correspond to each role. The role and actions are defined for Kubernetes resources in the yaml file. Further details about users and roles can be found in Kubernetes documentation.

```
$ kubectl apply -f deploy/rbac.yaml -n postgres-operator
```

Note

Setting RBAC requires your user to have cluster-admin role privileges. For example, those using Google Kubernetes Engine can grant user needed privileges with the following command:

```
$ kubectl create clusterrolebinding cluster-admin-binding --clusterrole=cluster-admin --user=$(gcloud config get-value core/account)
```

5. Start the Operator within Kubernetes:

```
$ kubectl apply -f deploy/operator.yaml -n postgres-operator
```

Optionally, you can add PostgreSQL Users secrets and TLS certificates to Kubernetes. If you don't, the Operator will create the needed users and certificates automatically, when you create the database cluster. You can see documentation on Users and TLS certificates if still want to create them yourself.

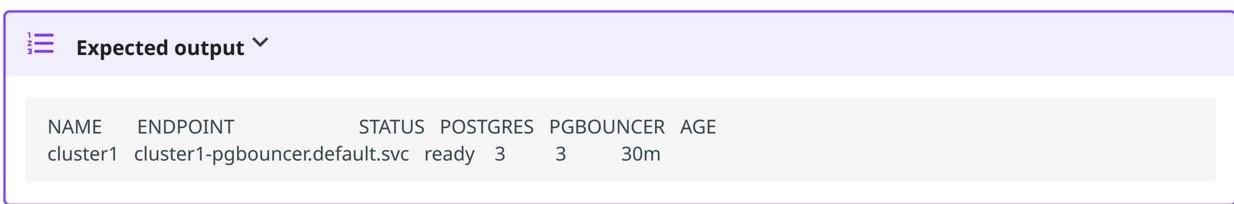
Percona LLC and/or its affiliates, © 2009 - 2024

6. After the Operator is started Percona Distribution for PostgreSQL cluster can be created at any time with the following command:

```
$ kubectl apply -f deploy/cr.yaml -n postgres-operator
```

The creation process may take some time. When the process is over your cluster will obtain the `ready` status. You can check it with the following command:

```
$ kubectl get pg -n postgres-operator
```



Expected output

```
NAME      ENDPOINT                       STATUS   POSTGRES   PGBOUNCER   AGE
cluster1  cluster1-pgbouncer.default.svc ready    3          3           30m
```

3.6.1 Verifying the cluster operation

When creation process is over, `kubectl get pg` command will show you the cluster status as `ready`, and you can try to connect to the cluster.

Percona LLC and/or its affiliates, © 2009 - 2024

During the installation, the Operator has generated several secrets, including the one with password for default PostgreSQL user. This default user has the same login name as the cluster name.

1. Use `kubectl get secrets` command to see the list of Secrets objects. The Secrets object you are interested in is named as `<cluster_name>-pguser-<cluster_name>` (substitute `<cluster_name>` with the name of your Percona Distribution for PostgreSQL Cluster). The default variant will be `cluster1-pguser-cluster1`.
2. Use the following command to get the password of this user:

```
$ kubectl get secret <cluster_name>-<user_name>-<cluster_name> -n postgres-operator --template='{{.data.password | base64decode}}{{"\n"}}'
```

3. Create a pod and start Percona Distribution for PostgreSQL inside. The following command will do this, naming the new Pod `pg-client`:

```
$ kubectl run -i --rm --tty pg-client --image=perconalab/percona-distribution-postgresql:16 --restart=Never -- bash -il
```

Executing it may require some time to deploy the corresponding Pod.

4. Run a container with `psql` tool and connect its console output to your terminal. This command will connect you as a `cluster1` user to a `cluster1` database via the PostgreSQL interactive terminal.

```
[postgres@pg-client /]$ PGPASSWORD='pguser_password' psql -h cluster1-pgbouncer.postgres-operator.svc -p 5432 -U cluster1 cluster1
```

Sample output

```
psql (16)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.
pgdb=>
```

3.6.2 Deleting the cluster

If you need to delete the cluster (for example, to clean up the testing deployment before adopting it for production use), check this HowTo.

3.6.3 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-13

Percona LLC and/or its affiliates, © 2009 - 2024

4. Configuration

4.1 Users

Operator provides a feature to manage users and databases in your PostgreSQL cluster. This document describes this feature, defaults and ways to fine tune your users.

4.1.1 Defaults

When you create a PostgreSQL cluster with the Operator and do not specify any additional users or databases, the Operator will do the following:

- Create a database that matches the name of your PostgreSQL cluster.
- Create an unprivileged PostgreSQL user with the name of the cluster. This user has access to the database created in the previous step.
- Create a Secret with the login credentials and connection details for the PostgreSQL user which is in relation to the database. This is stored in a Secret named `<clusterName>-pguser-<clusterName>`. These credentials include:
- `user`: The name of the user account.
- `password`: The password for the user account.
- `dbname`: The name of the database that the user has access to by default.
- `host`: The name of the host of the database. This references the Service of the primary PostgreSQL instance.
- `port`: The port that the database is listening on.
- `uri`: A PostgreSQL connection URI that provides all the information for logging into the PostgreSQL database via pgBouncer
- `jdbc-uri`: A PostgreSQL JDBC connection URI that provides all the information for logging into the PostgreSQL database via the JDBC driver.

As an example, using our `cluster1` PostgreSQL cluster, we would see the following created:

- A database named `cluster1`.
- A PostgreSQL user named `cluster1`.
- A Secret named `cluster1-pguser-cluster1` that contains the user credentials and connection information.

Percona LLC and/or its affiliates, © 2009 - 2024

4.1.2 Custom Users and Databases

Users and databases can be customized in `spec.users` section in the Custom Resource. Section can be changed at the cluster creation time and adjusted over time. Note the following:

- If `spec.users` is set during the cluster creation, the Operator will not create any default users or databases except for PostgreSQL. If you want additional databases, you will need to specify them.
- For each user added in `spec.users`, the Operator will create a Secret of the `<clusterName>-pguser-<userName>` format (such default Secret naming can be altered for the user with the `spec.users.secretName` option). This Secret will contain the user credentials.
- If no databases are specified, `dbname` and `uri` will not be present in the Secret.
- If at least one option under the `spec.users.databases` is specified, the first database in the list will be populated into the connection credentials.
- The Operator does not automatically drop users in case of removed Custom Resource options to prevent accidental data loss.
- Similarly, to prevent accidental data loss Operator does not automatically drop databases (see how to actually drop a database here).
- Role attributes are not automatically dropped if you remove them. You need to set the inverse attribute to actually drop them (e.g. `NOSUPERUSER`).
- The special `postgres` user can be added as one of the custom users; however, the privileges of this user cannot be adjusted.

Creating a New User

Change `PerconaPGCluster` Custom Resource (e.g. by editing your YAML manifest in the `deploy/cr.yaml` configuration file):

```
...
spec:
  users:
    - name: perconapg
```

Apply the changes (e.g. with the usual `kubctl apply -f deploy/cr.yaml’ command) will create the new user:

- The user will only be able to connect to the default `postgres` database.
- The credentials of this user are populated in the `<clusterName>-pguser-perconapg` secret. There are no connection credentials.
- The user is unprivileged.

The following example shows how to create a new `pgtest` database and let `perconapg` user access it. The appropriate Custom Resource fragment will look as follows:

```
...
spec:
  users:
    - name: perconapg
      databases:
        - pgtest
```

If you inspect the `<clusterName>-pguser-perconapg` Secret after applying the changes, you will see `dbname` and `uri` options populated there, and the database is created as well.

Percona LLC and/or its affiliates, © 2009 - 2024

Adjusting privileges

You can set role privileges by using the standard role attributes that PostgreSQL provides and adding them to the `spec.users.options` subsection in the Custom Resource. The following example will make the `perconapg` a superuser. You can add the following to the spec in your `deploy/cr.yaml` :

```yaml
...
spec:
  users:
    - name: perconapg
      databases:
        - pgtest
      options: "SUPERUSER"
```

Apply changes with the usual `kubctl apply -f deploy/cr.yaml' command.

To actually revoke the superuser privilege afterwards, you will need to do and apply the following change:

```yaml
...
spec:
  users:
    - name: perconapg
      databases:
        - pgtest
      options: "NOSUPERUSER"
```

If you want to add multiple privileges, you can use a space-separated list as follows:

```yaml
...
spec:
  users:
    - name: perconapg
      databases:
        - pgtest
      options: "CREATEDB CREATEROLE"
```

`postgres` User

By default, the Operator does not create the `postgres` user. You can create it by applying the following change to your Custom Resource:

```yaml
...
spec:
  users:
    - name: postgres
```

This will create a Secret named `<clusterName>-pguser-postgres` that contains the credentials of the `postgres` account.

Deleting users and databases

The Operator does not delete users and databases automatically. After you remove the user from the Custom Resource, it will continue to exist in your cluster. To remove a user and all of its objects, as a superuser you will need to run `DROP OWNED` in each database the user has objects in, and `DROP ROLE` in your PostgreSQL cluster.

```sql
DROP OWNED BY perconapg;
DROP ROLE perconapg;
```


Percona LLC and/or its affiliates, © 2009 - 2024

For databases, you should run the `DROP DATABASE` command as a superuser:

```
DROP DATABASE pgtest;
```

Managing user passwords

If you want to rotate user's password, just remove the old password in the correspondent Secret: the Operator will immediately generate a new password and save it to the appropriate Secret. You can remove the old password with the `kubectl patch secret` command:

```
$ kubectl patch secret <clusterName>-pguser-<userName> -p '{"data":{"password":""}}'
```

Also, you can set a custom password for the user. Do it as follows:

```
$ kubectl patch secret <clusterName>-pguser-<userName> -p '{"stringData":{"password":"<custom_password>",
"verifier":""}}'
```

Superuser and pgBouncer

For security reasons we do not allow superusers to connect to cluster through pgBouncer by default. You can connect through `primary` service (read more in exposure documentation).

Otherwise you can use the proxy.pgBouncer.exposeSuperusers Custom Resource option to enable superusers connection via pgBouncer.

4.1.3 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-06-29

Percona LLC and/or its affiliates, © 2009 - 2024

4.2 Exposing cluster

The Operator provides entry points for accessing the database by client applications. The database cluster is exposed with regular Kubernetes Service objects configured by the Operator.

This document describes the usage of Custom Resource manifest options to expose the clusters deployed with the Operator.

4.2.1 PgBouncer

We recommend exposing the cluster through PgBouncer, which is enabled by default. You can disable pgBouncer by setting `proxy.pgBouncer.replicas` to 0.

The following example deploys two pgBouncer nodes exposed through a LoadBalancer Service object:

```
proxy:
  pgBouncer:
    replicas: 2
    image: percona/percona-postgresql-operator:2.3.1-ppg14-pgbouncer
    expose:
      type: LoadBalancer
```

The Service will be called `<clusterName>-pgbouncer` :

```
$ kubectl get service
```

Expected output

```
NAME                TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)          AGE
cluster1-pgbouncer  LoadBalancer   10.88.8.48    34.133.38.186   5432:30601/TCP   20m
```

You can connect to the database using the External IP of the load balancer and port `5432` .

If your application runs inside the Kubernetes cluster as well, you might want to use the Cluster IP Service type in `proxy.pgBouncer.expose.type` , which is the default. In this case to connect to the database use the internal domain name - `cluster1-pgbouncer.<namespace>.svc.cluster.local` .

4.2.2 Exposing the cluster without PgBouncer

You can connect to the cluster without a proxy. For that use `<clusterName>-ha` Service object:

```
$ kubectl get service
```

Expected output

```
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
cluster1-ha   ClusterIP   10.88.8.121   <none>        5432/TCP   115s
```

This service points to the active primary. In case of failover to the replica node, will change the endpoint automatically.

Percona LLC and/or its affiliates, © 2009 - 2024

To change the Service type, use `expose.type` in the Custom Resource manifest. For example, the following manifest will expose this service through a load balancer:

```
spec:
...
  expose:
    type: LoadBalancer
```

4.2.3 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-05-04

Percona LLC and/or its affiliates, © 2009 - 2024

4.3 Changing PostgreSQL options

Despite the Operator's ability to configure PostgreSQL and the large number of Custom Resource options, there may be situations where you need to pass specific options directly to your cluster's PostgreSQL instances. For this purpose, you can use the PostgreSQL dynamic configuration method provided by Patroni. You can pass PostgreSQL options to Patroni through the Operator Custom Resource, updating it with `deploy/cr.yaml` configuration file).

Custom PostgreSQL configuration options should be included into the `patroni.dynamicConfiguration.postgresql.parameters` subsection as follows:

```
...
patroni:
  dynamicConfiguration:
    postgresql:
      parameters:
        max_parallel_workers: 2
        max_worker_processes: 2
        shared_buffers: 1GB
        work_mem: 2MB
```

Please note that configuration changes will be automatically applied to the running instances as soon as you apply Custom Resource changes in a usual way, running the `kubectl apply -f deploy/cr.yaml` command.

You can apply custom configuration in this way for both new and existing clusters.

Normally, options should be applied to PostgreSQL instances dynamically without restart, except the options with the postmaster context. Changing options which have `context=postmaster` will cause Patroni to initiate restart of all PostgreSQL instances, one by one. You can check the context of a specific option using the `SELECT name, context FROM pg_settings;` query to to see if the change should cause a restart or not.

Note

The Operator passes options to Patroni without validation, so there is a theoretical possibility of the cluster malfunction caused by wrongly configured PostgreSQL instances. Also, this configuration method is used for PostgreSQL options only and cannot be applied to change other Patroni dynamic configuration options. It means that options in the `parameters` subsection under `patroni.dynamicConfiguration.postgresql` will be applied, and everything else in `patroni.dynamicConfiguration.postgresql` will be ignored.

4.3.1 Using host-based authentication (pg_hba)

PostgreSQL Host-Based Authentication (pg_hba) allows controlling access to the PostgreSQL database based on the IP address or the host name of the connecting host. You can configure `pg_hba` through the Custom Resource `patroni.dynamicConfiguration.postgresql.pg_hba` subsection as follows:

```
...
patroni:
  dynamicConfiguration:
    postgresql:
      pg_hba:
      - host    all all 0.0.0.0/0 md5
```

As you may guess, this example allows all hosts to connect to any database with MD5 password-based authentication.

Percona LLC and/or its affiliates, © 2009 - 2024

Obviously, you can connect both `dynamicConfiguration.postgresql.parameters` and `dynamicConfiguration.postgresql.pg_hba` subsections:

```
...
patroni:
  dynamicConfiguration:
    postgresql:
      parameters:
        max_parallel_workers: 2
        max_worker_processes: 2
        shared_buffers: 1GB
        work_mem: 2MB
      pg_hba:
      - local   all all trust
      - host    all all 0.0.0.0/0 md5
      - host    all all ::1/128   md5
      - host    all mytest 123.123.123.123/32 reject
```

The changes will be applied after you update Custom Resource in a usual way:

```
$ kubectl apply -f deploy/cr.yaml
```

4.3.2 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

4.4 Binding Percona Distribution for PostgreSQL components to specific Kubernetes/OpenShift Nodes

The operator does good job automatically assigning new Pods to nodes with sufficient resources to achieve balanced distribution across the cluster. Still there are situations when it is worth to ensure that pods will land on specific nodes: for example, to get speed advantages of the SSD equipped machine, or to reduce network costs choosing nodes in a same availability zone.

Appropriate sections of the deploy/cr.yaml file (such as `proxy.pgBouncer`) contain keys which can be used to do this, depending on what is the best for a particular situation.

4.4.1 Affinity and anti-affinity

Affinity makes Pod eligible (or not eligible - so called "anti-affinity") to be scheduled on the node which already has Pods with specific labels, or has specific labels itself (so called "Node affinity"). Particularly, Pod anti-affinity is good to reduce costs making sure several Pods with intensive data exchange will occupy the same availability zone or even the same node - or, on the contrary, to make them land on different nodes or even different availability zones for the high availability and balancing purposes. Node affinity is useful to assign PostgreSQL instances to specific Kubernetes Nodes (ones with specific hardware, zone, etc.).

Pod anti-affinity is controlled by the `affinity.podAntiAffinity` subsection, which can be put into `proxy.pgBouncer` and `backups.pgbackrest.repoHost` sections of the `deploy/cr.yaml` configuration file.

`podAntiAffinity` allows you to use standard Kubernetes affinity constraints of any complexity:

```
affinity:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 1
      podAffinityTerm:
        labelSelector:
          matchLabels:
            postgres-operator.crunchydata.com/cluster: keycloakdb
            postgres-operator.crunchydata.com/role: pgbouncer
        topologyKey: kubernetes.io/hostname
```

You can see the explanation of these affinity options in Kubernetes documentation.

4.4.2 Topology Spread Constraints

Topology Spread Constraints allow you to control how Pods are distributed across the cluster based on regions, zones, nodes, and other topology specifics. This can be useful for both high availability and resource efficiency.

Pod topology spread constraints are controlled by the `topologySpreadConstraints` subsection, which can be put into `proxy.pgBouncer` and `backups.pgbackrest.repoHost` sections of the `deploy/cr.yaml` configuration file as follows:

```
topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: my-node-label
    whenUnsatisfiable: DoNotSchedule
    labelSelector:
      matchLabels:
        postgres-operator.crunchydata.com/instance-set: instance1
```

You can see the explanation of these affinity options in Kubernetes documentation.

Percona LLC and/or its affiliates, © 2009 - 2024

4.4.3 Tolerations

Tolerations allow Pods having them to be able to land onto nodes with matching *taints*. Toleration is expressed as a `key` with and `operator`, which is either `exists` or `equal` (the latter variant also requires a `value` the key is equal to). Moreover, toleration should have a specified `effect`, which may be a self-explanatory `NoSchedule`, less strict `PreferNoSchedule`, or `NoExecute`. The last variant means that if a *taint* with `NoExecute` is assigned to node, then any Pod not tolerating this *taint* will be removed from the node, immediately or after the `tolerationSeconds` interval, like in the following example.

You can use `instances.tolerations` and `backups.pgbackrest.jobs.tolerations` subsections in the `deploy/cr.yaml` configuration file as follows:

```
tolerations:
- effect: NoSchedule
  key: role
  operator: Equal
  value: connection-poolers
```

The Kubernetes Taints and Toleratins contains more examples on this topic.

4.4.4 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

4.5 Labels and annotations

Labels and annotations are used to attach additional metadata information to Kubernetes resources.

Labels and annotations are rather similar. The difference between them is that labels are used by Kubernetes to identify and select objects, while annotations are assigning additional *non-identifying* information to resources. Therefore, typical role of Annotations is facilitating integration with some external tools.

4.5.1 Setting labels and annotations in the Custom Resource

You can set labels and/or annotations as key/value string pairs in the Custom Resource metadata section of the `deploy/cr.yaml`. For PostgreSQL, pgBouncer and pgBackRest Pods, use `instances.metadata.annotations` / `instances.metadata.labels`, `proxy.pgbouncer.metadata.annotations` / `proxy.pgbouncer.metadata.labels`, or `backups.pgbackrest.metadata.annotations` / `backups.pgbackrest.metadata.labels` keys as follows:

```
apiVersion: pgv2.percona.com/v2
kind: PerconaPGCluster
...
spec:
...
 instances:
  - name: instance1
   replicas: 3
   metadata:
    annotations:
     my-annotation: value1
    labels:
     my-label: value2
   ...
```

For PostgreSQL and pgBouncer Services, use `expose.annotations` / `expose.labels` or `proxy.pgbouncer.expose.annotations` / `proxy.pgbouncer.expose.labels` keys as follows:

```
apiVersion: pgv2.percona.com/v2
kind: PerconaPGCluster
...
spec:
 ...
 expose:
  annotations:
   my-annotation: value1
  labels:
   my-label: value2
  ...
```

The easiest way to check which labels are attached to a specific object with is using the additional `--show-labels` option of the `kubectl get` command. Checking the annotations is not much more difficult: it can be done as in the following example:

```
$ kubectl get service cluster1-pgbouncer -o jsonpath='{.metadata.annotations}'
```

4.5.2 Settings labels and annotations to the Operator Pod

You can assign labels and/or annotations to the Pod of the Operator itself by editing the and the deploy/operator.yaml configuration file before applying it during the installation.

Percona LLC and/or its affiliates, © 2009 - 2024

```yaml
apiVersion: apps/v1
kind: Deployment
...
spec:
...
  template:
    metadata:
      labels:
        app.kubernetes.io/component: operator
        app.kubernetes.io/instance: percona-postgresql-operator
        app.kubernetes.io/name: percona-postgresql-operator
        app.kubernetes.io/part-of: percona-postgresql-operator
        pgv2.percona.com/control-plane: postgres-operator
        ...
```

4.5.3 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-11-30

Percona LLC and/or its affiliates, © 2009 - 2024

4.6 Transport layer security (TLS)

The Percona Operator for PostgreSQL uses Transport Layer Security (TLS) cryptographic protocol for the following types of communication:

- Internal - communication between PostgreSQL instances in the cluster
- External - communication between the client application and the cluster

The internal certificate is also used as an authorization method for PostgreSQL Replica instances.

TLS security can be configured in following ways:

- the Operator can generate long-term certificates automatically at cluster creation time,
- you can generate certificates manually.

The following subsections explain how to configure TLS security with the Operator yourself, as well as how to temporarily disable it if needed.

4.6.1 Allow the Operator to generate certificates automatically

The Operator is able to generate long-term certificates automatically and turn on encryption at cluster creation time, if there are no certificate secrets available. Just deploy your cluster as usual, with the `kubectl apply -f deploy/cr.yaml` command, and certificates will be generated.

4.6.2 Check connectivity to the cluster

You can check TLS communication with use of the `psql`, the standard interactive terminal-based frontend to PostgreSQL. The following command will spawn a new `pg-client` container, which includes needed command and can be used for the check (use your real cluster name instead of the `<cluster-name>` placeholder):

```
$ cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pg-client
spec:
  replicas: 1
  selector:
    matchLabels:
      name: pg-client
  template:
    metadata:
      labels:
        name: pg-client
    spec:
      containers:
        - name: pg-client
          image: perconalab/percona-distribution-postgresql:16
          imagePullPolicy: Always
          command:
          - sleep
          args:
          - "100500"
          volumeMounts:
            - name: ca
              mountPath: "/tmp/tls"
      volumes:
      - name: ca
```


Percona LLC and/or its affiliates, © 2009 - 2024

```
      secret:
        secretName: <cluster_name>-ssl-ca
        items:
        - key: ca.crt
          path: ca.crt
          mode: 0777
EOF
```

Now get shell access to the newly created container, and launch the PostgreSQL interactive terminal to check connectivity over the encrypted channel (please use real cluster-name, PostgreSQL user login and password):

```
$ kubectl exec -it deployment/pg-client -- bash -il
[postgres@pg-client /]$ PGSSLMODE=verify-ca PGSSLROOTCERT=/tmp/tls/ca.crt psql postgres://<postgresql-user>:<postgresql-password>@<cluster-name>-pgbouncer.<namespace>.svc.cluster.local
```

Now you should see the prompt of PostgreSQL interactive terminal:

```
$ psql (16)
Type "help" for help.
pgdb=>
```

4.6.3 Generate certificates manually

To use custom TLS certificates for a Postgres cluster, you will need to create a Secret in the Namespace of your cluster that contains the TLS key (`tls.key`), TLS certificate (`tls.crt`) and the CA certificate (`ca.crt`) to use. The Secret should contain the following values:

```
data:
  ca.crt: <value>
  tls.crt: <value>
  tls.key: <value>
```

You should generate certificates twice: one set is for external communications, and another set is for internal ones. A secret created for the external use must be added to the `secrets.customTLSSecret.name` field of your Custom Resource. A certificate generated for internal communications must be added to the `secrets.customReplicationTLSSecret.name` field.

For example, if you have files named `ca.crt` , `hippo.key` , and `hippo.crt` stored on your local machine, you could run the following command:

```
$ kubectl create secret generic -n postgres-operator hippo.tls \
  --from-file=ca.crt=ca.crt \
  --from-file=tls.key=hippo.key \
  --from-file=tls.crt=hippo.crt
```

Now you can add the custom TLS Secret name to the `secrets.customTLSSecret.name` field in your Rustom Resource:

```
secrets:
  customTLSSecret:
    name: hippo.tls
```

Don't forget to apply changes as usual:

```
$ kubectl apply -f deploy/cr.yaml
```


Percona LLC and/or its affiliates, © 2009 - 2024

4.6.4 Check your certificates for expiration

1. First, check the necessary secrets names (`cluster1-cluster-cert` and `cluster1-replication-cert` by default):

```
$ kubectl get secrets
```

You will have the following response:

```
NAME                    TYPE    DATA  AGE
cluster1-cluster-cert        Opaque  3     11m
...
cluster1-replication-cert     Opaque  3     11m
...
```

2. Now use the following command to find out the certificates validity dates, substituting Secrets names if necessary:

```
$ {
 kubectl get secret/cluster1-replication-cert -o jsonpath='{.data.tls\.crt}' | base64 --decode | openssl x509 -noout -dates
 kubectl get secret/cluster1-cluster-cert -o jsonpath='{.data.ca\.crt}' | base64 --decode | openssl x509 -noout -dates
 }
```

The resulting output will be self-explanatory:

```
notBefore=Jun 28 10:20:19 2023 GMT
notAfter=Jun 27 11:20:19 2024 GMT
notBefore=Jun 28 10:20:18 2023 GMT
notAfter=Jun 25 11:20:18 2033 GMT
```

4.6.5 Keep certificates after deleting the cluster

In case of cluster deletion, objects, created for SSL (Secret, certificate, and issuer) are not deleted by default.

If the user wants the cleanup of objects created for SSL, there is a finalizers.percona.com/delete-ssl Custom Resource option, which can be set in `deploy/cr.yaml` : if this finalizer is set, the Operator will delete Secret, certificate and issuer after the cluster deletion event.

4.6.6 Connect to the database cluster without TLS

Omitting TLS is also possible, but we recommend that you connect to your cluster with the TLS protocol enabled.

You can enable connections without TLS (e.g. for demonstration purposes) via the following line to the custom PostgreSQL configuration. Add the following line to the Operator Custom Resource via the `deploy/cr.yaml` configuration file:

```
...
patroni:
 dynamicConfiguration:
  postgresql:
   pg_hba:
   - host    all all 0.0.0.0/0 md5
```

See Using host-based authentication for more details.

Percona LLC and/or its affiliates, © 2009 - 2024

4.6.7 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-20

Percona LLC and/or its affiliates, © 2009 - 2024

4.7 Telemetry

The Telemetry function enables the Operator gathering and sending basic anonymous data to Percona, which helps us to determine where to focus the development and what is the uptake for each release of Operator.

The following information is gathered:

- ID of the Custom Resource (the `metadata.uid` field)
- Kubernetes version
- Platform (is it Kubernetes or Openshift)
- Is PMM enabled, and the PMM Version
- Operator version
- PostgreSQL version
- PgBackRest version
- Was the Operator deployed with Helm
- Are sidecar containers used
- Are backups used

We do not gather anything that identify a system, but the following thing should be mentioned: Custom Resource ID is a unique ID generated by Kubernetes for each Custom Resource.

Telemetry is enabled by default and is sent to the Version Service server when the Operator connects to it at scheduled times to obtain fresh information about version numbers and valid image paths needed for the upgrade.

The landing page for this service, check.percona.com, explains what this service is.

You can disable telemetry with a special option when installing the Operator:

- if you install the Operator with helm, use the following installation command:

```
$ helm install my-db percona/pg-db --version 2.3.1 --namespace my-namespace --set disable_telemetry="true"
```

- if you don't use helm for installation, you have to edit the `operator.yaml` before applying it with the `kubectl apply -f deploy/operator.yaml` command. Open the `operator.yaml` file with your text editor, find the `DISABLE_TELEMETRY` environment variable and set it to `"true"`

```
...
- name: DISABLE_TELEMETRY
  value: "true"
...
```

4.7.1 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Percona LLC and/or its affiliates, © 2009 - 2024

Last update: 2023-11-01

Percona LLC and/or its affiliates, © 2009 - 2024

5. Management

5.1 Upgrade Database and Operator

5.1.1 Upgrade from the Operator version 1.x to version 2.x

The Operator version 2.x has a lot of differences compared to the version 1.x. This makes upgrading from version 1.x to version 2.x quite different from a normal upgrade. In fact, you have to migrate the cluster from version 1.x to version 2.x.

There are several ways to do such version 1.x to version 2.x upgrade. Choose the method based on your downtime preference and roll back strategy:

	Pros	Cons
Data Volumes migration - re-use the volumes that were created by the Operator version 1.x	The simplest method	- Requires downtime - Impossible to roll back
Backup and restore - take the backup with the Operator version 1.x and restore it to the cluster deployed by the Operator version 2.x	Allows you to quickly test version 2.x	Provides significant downtime in case of migration
Replication - replicate the data from the Operator version 1.x cluster to the standby cluster deployed by the Operator version 2.x	- Quick test of v2 cluster - Minimal downtime during upgrade	Requires significant computing resources to run two clusters in parallel

5.1.2 Update Database and Operator version 2.x

Starting from the version 2.2.0 Percona Operator for PostgreSQL allows upgrades to newer 2.x versions. The upgradable components of the cluster are the following ones:

- the Operator;
- Custom Resource Definition (CRD),
- Database Management System (Percona Distribution for PostgreSQL).

The list of recommended upgrade scenarios includes two variants:

- Upgrade to the new versions of the Operator *and* Percona Distribution for PostgreSQL,
- Minor Percona Distribution for PostgreSQL version upgrade *without* the Operator upgrade.

Upgrading the Operator and CRD

Note

The Operator supports **last 3 versions of the CRD**, so it is technically possible to skip upgrading the CRD and just upgrade the Operator. If the CRD is older than the new Operator version *by no more than three releases*, you will be able to continue using the old CRD and even carry on Percona Distribution for PostgreSQL minor version upgrades with it. But the recommended way is to update the Operator *and* CRD.

Only the incremental update to a nearest version of the Operator is supported (for example, update from 2.2.0 to 2.3.0). To update to a newer version, which differs from the current version by more than one, make several incremental updates sequentially.

Percona LLC and/or its affiliates, © 2009 - 2024

Considering the Operator uses `postgres-operator` namespace, upgrade to the version 2.3.1 includes the following steps.

1. Update the Custom Resource Definition for the Operator, taking it from the official repository on Github, and do the same for the Role-based access control:

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-postgresql-operator/v2.3.1/deploy/crd.yaml
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-postgresql-operator/v2.3.1/deploy/rbac.yaml -n postgres-operator
```

Note

In case of cluster-wide installation, use `deploy/cw-rbac.yaml` instead of `deploy/rbac.yaml`.

2. Now you should apply a patch to your deployment, supplying necessary image name with a newer version tag. You can find the proper image name for the current Operator release in the list of certified images. updating to the `2.3.1` version should look as follows:

```
$ kubectl -n postgres-operator patch deployment percona-postgresql-operator \
  -p'{"spec":{"template":{"spec":{"containers":[{"name":"operator","image":"percona/percona-postgresql-operator:2.3.1"}]}}}}'
```

3. The deployment rollout will be automatically triggered by the applied patch. You can track the rollout process in real time with the `kubectl rollout status` command with the name of your cluster:

```
$ kubectl rollout status deployments percona-postgresql-operator
```


Percona LLC and/or its affiliates, © 2009 - 2024

5.1.3 Upgrading Percona Distribution for PostgreSQL

Upgrading Percona Distribution for PostgreSQL can be done as follows:

1. Apply a patch to your Custom Resource, setting necessary Custom Resource version and image names with a newer version tag.

Note

Check the version of the Operator you have in your Kubernetes environment. Please refer to the Operator upgrade guide to upgrade the Operator and CRD, if needed.

Patching Custom Resource is done with the `kubectl patch pg` command. Actual image names can be found in the list of certified images. For example, updating `cluster1` cluster to the `2.3.1` version should look as follows:

```
$ kubectl -n postgres-operator patch pg cluster1 --type=merge --patch '{
  "spec": {
    "crVersion":"2.3.1",
    "image": "percona/percona-postgresql-operator:2.3.1-ppg15-postgres",
    "proxy": { "pgBouncer": { "image": "percona/percona-postgresql-operator:2.3.1-ppg15-pgbouncer" } },
    "backups": { "pgbackrest":  { "image": "percona/percona-postgresql-operator:2.3.1-ppg15-pgbackrest" } },
    "pmm": { "image": "percona/pmm-client:2.41.0" }
  }}'
```

Warning

The above command upgrades various components of the cluster including PMM Client. It is highly recommended to upgrade PMM Server **before** upgrading PMM Client. If it wasn't done and you would like to avoid PMM Client upgrade, remove it from the list of images, reducing the last of two patch commands as follows:

```
$ kubectl -n postgres-operator patch pg cluster1 --type=merge --patch '{
  "spec": {
    "crVersion":"2.3.1",
    "image": "percona/percona-postgresql-operator:2.3.1-ppg15-postgres",
    "proxy": { "pgBouncer": { "image": "percona/percona-postgresql-operator:2.3.1-ppg15-pgbouncer" } },
    "backups": { "pgbackrest":  { "image": "percona/percona-postgresql-operator:2.3.1-ppg15-pgbackrest" } }
  }}'
```

The deployment rollout will be automatically triggered by the applied patch. The update process is successfully finished when all Pods have been restarted.

5.1.4 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

5.2 Upgrade from version 1 to version 2

5.2.1 Upgrade using data volumes

Prerequisites:

The following conditions should be met for the Volumes-based migration:

- You have a version 1.x cluster with `spec.keepData: true` in the Custom Resource
- You have both Operators deployed and allow them to control resources in the same namespace
- Old and new clusters must be of the same PostgreSQL major version

This migration method has two limitations. First of all, this migration method introduces a downtime. Also, you can only reverse such migration by restoring the old cluster from the backup. See other migration methods if you need lower downtime and a roll back plan.

Prepare version 1.x cluster for the migration

1. Remove all Replicas from the cluster, keeping only primary running. It is required to assure that Volume of the primary PVC does not change. The `deploy/cr.yaml` configuration file should have it as follows:

```
...
pgReplicas:
  hotStandby:
    size: 0
```

2. Apply the Custom Resource in a usual way:

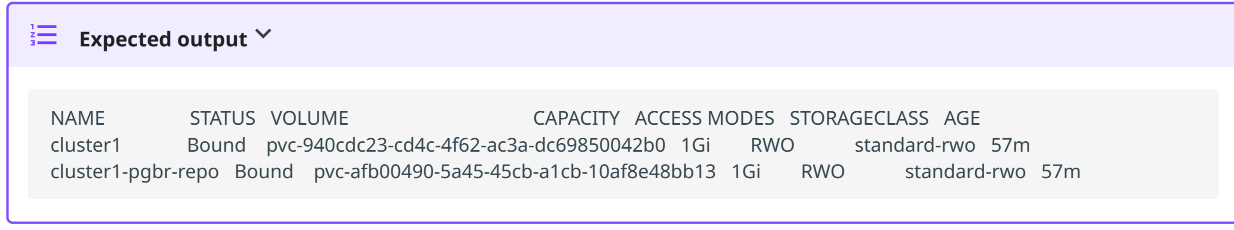
```
$ kubectl apply -f deploy/cr.yaml
```

3. When all Replicas are gone, proceed with removing the cluster. Double check that `spec.keepData` is in place, otherwise the Operator will delete the volumes!

```
$ kubectl delete perconapgcluster cluster1
```

4. Find PVC for the Primary and `pgBackRest`:

```
$ kubectl get pvc --selector=pg-cluster=cluster1 -n pgo
```



Expected output

```
NAME               STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
cluster1           Bound    pvc-940cdc23-cd4c-4f62-ac3a-dc69850042b0   1Gi        RWO            standard-rwo   57m
cluster1-pgbr-repo Bound    pvc-afb00490-5a45-45cb-a1cb-10af8e48bb13   1Gi        RWO            standard-rwo   57m
```

A third PVC used to store write-ahead logs (WAL) may also be present if external WAL volumes were enabled for the cluster.

Percona LLC and/or its affiliates, © 2009 - 2024

5. Permissions for `pgBackRest` repo folders are managed differently in version 1 and version 2. We need to change the ownership of the `backrest` folder on the Persistent Volume to avoid errors during migration. Running a `chown` command within a container fixes this problem. You can use the following manifest to execute it:

   chown-pod.yaml

   ```yaml
   apiVersion: v1
   kind: Pod
   metadata:
     name: chown-pod
   spec:
     volumes:
       - name: backrestrepo
         persistentVolumeClaim:
           claimName: cluster1-pgbr-repo
     containers:
       - name: task-pv-container
         image: ubuntu
         command:
         - chown
         - -R
         - 26:26
         - /backrestrepo/cluster1-backrest-shared-repo
         volumeMounts:
           - mountPath: "/backrestrepo"
             name: backrestrepo
   ```

   Apply it as follows:

   ```
   $ kubectl apply -f chown-pod.yaml -n pgo
   ```

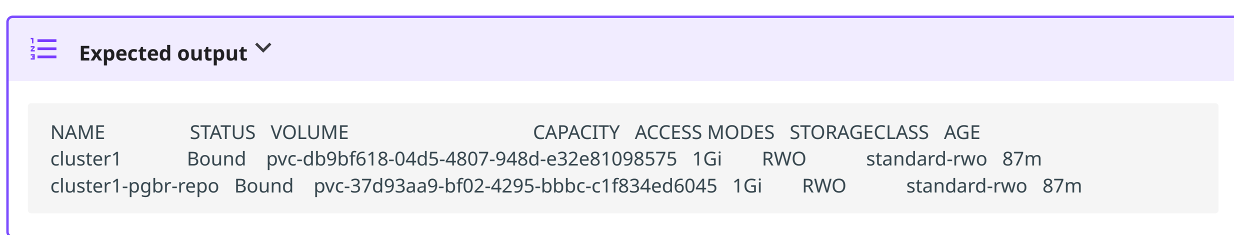

Percona LLC and/or its affiliates, © 2009 - 2024

Execute the migration to version 2.x

The old cluster is shut down, and Volumes are ready to be used to provision the new cluster managed by the Operator version 2.x.

1. **Install the Operator version 2** (if not done yet). Pick your favorite method from our documentaion.
2. Run the following command to show the names of PVC belonging to the old cluster:

```
$ kubectl get pvc --selector=pg-cluster=cluster1 -n pgo
```



Expected output

```
NAME              STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
cluster1          Bound    pvc-db9bf618-04d5-4807-948d-e32e81098575   1Gi        RWO            standard-rwo   87m
cluster1-pgbr-repo   Bound    pvc-37d93aa9-bf02-4295-bbbc-c1f834ed6045   1Gi        RWO            standard-rwo   87m
```

3. Now edit the Custom Resource manifest (`deploy/cr.yaml` configuration file) of the version 2.x cluster: add fields to the `dataSource.volumes` subsection, pointing to the PVCs of the version 1.x cluster:

```
...
dataSource:
  volumes:
    pgDataVolume:
      pvcName: cluster1
      directory: cluster1
    pgBackRestVolume:
      pvcName: cluster1-pgbr-repo
      directory: cluster1-backrest-shared-repo
```

4. Do not forget to set the proper PostgreSQL major version. It must be the same version that was used in version 1 cluster. You can set the version in the corresponding `image` sections and `postgresVersion` . The following example sets version 14:

```
spec:
  image: percona/percona-postgresql-operator:2.3.1-ppg14-postgres
  postgresVersion: 14
  proxy:
    pgBouncer:
      image: percona/percona-postgresql-operator:2.3.1-ppg14-pgbouncer
  backups:
    pgbackrest:
      image: percona/percona-postgresql-operator:2.3.1-ppg14-pgbackrest
```

5. Apply the manifest:

```
$ kubectl apply -f deploy/cr.yaml
```

The new cluster will be provisioned shortly using the volume of the version 1.x cluster. You should remove the `spec.datasource.volumes` section from your manifest.

Percona LLC and/or its affiliates, © 2009 - 2024

Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

5.2.2 Upgrade using backup and restore

This method allows you to migrate from the version 1.x to version 2.x cluster by restoring (actually creating) a new version 2.x PostgreSQL cluster using a backup from the version 1.x cluster.

Note

To make sure that all transactions are captured in the backup, you need to stop the old cluster. This brings downtime to the application.

Prepare the backup

1. Create the backup on the version 1.x cluster, following the official guide for manual (on-demand) backups. This involves preparing the manifest in YAML and applying it in the ususal way:

```
$ kubectl apply -f deploy/backup/backup.yaml
```

2. Pause or delete the version 1.x cluster to ensure that you have the latest data.

Warning

Before deleting the cluster, make sure that the spec.keepBackups Custom Resource option is set to `true`. When it's set, local backups will be kept after the cluster deletion, so you can proceed with deleting your cluster as follows:

```
$ kubectl delete perconapgcluster cluster1
```


Percona LLC and/or its affiliates, © 2009 - 2024

Restore the backup as a version 2.x cluster

Restore from S3 / Google Cloud Storage for backups repository

1. To restore from the S3 or Google Cloud Storage for backups (GCS) repository, you should first configure the `spec.backups.pgbackrest.repos` subsection in your version 2.x cluster Custom Resource to point to the backup storage system. Just follow the repository documentation instruction for S3 or GCS. For example, for GCS you can define the repository similar to the following:

```yaml
spec:
 backups:
  pgbackrest:
   repos:
   - name: repo1
     gcs:
       bucket: MY-BUCKET
       region: us-central1
```

2. Create and configure any required Secrets or desired custom pgBackrest configuration as described in the backup documentation for te Operator version 2.x.

3. Set the repository path in the `backups.pgbackrest.global` subsection. By default it is `/backrestrepo/<clusterName>-backrest-shared-repo`:

```yaml
 spec:
 backups:
  pgbackrest:
   global:
     repo1: /backrestrepo/cluster1-backrest-shared-repo
```

4. Set the `spec.dataSource` option to create the version 2.x cluster from the specific repository:

```yaml
spec:
 dataSource:
  postgresCluster:
   repoName: repo1
```

   You can also provide other pgBackRest restore options, e.g. if you wish to restore to a specific point-in-time (PITR).

5. Create the version 2.x cluster:

```
$ kubectl apply -f cr.yaml
```

Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

5.2.3 Migrate using Standby

This method allows you to migrate from version 1.x to version 2.x by creating a new version 2.x PostgreSQL cluster in a "standby" mode, mirroring the version 1.x cluster to it continuously. This method can provide minimal downtime, but requires additional computing resources to run two clusters in parallel.

This method only works if the version 1.x cluster uses Amazon S3 or S3-compatible storage, or Google Cloud storage (GCS) for backups. For more information on standby clusters, please refer to this article.

Migrate to version 2

There is no need to perform any additional configuration on version 1.x cluster, you will only need to configure the version 2.x one.

1. Configure `spec.backups.pgbackrest.repos` Custom Resource option to point to the backup storage system. For example, for GCS, the repository would be defined similar to the following:

```
spec:
 backups:
  pgbackrest:
   repos:
   - name: repo1
     gcs:
       bucket: MY-BUCKET
       region: us-central1
```

2. Create and configure any required secrets or desired custom pgBackrest configuration as described in the backup documentation for the version 2.x.
3. Set the repository path in `backups.pgbackrest.global` section of the Custom Resource configuration file. By default it will be `/backrestrepo/<clusterName>-backrest-shared-repo`:

```
spec:
backups:
 pgbackrest:
   global:
     repo1: /backrestrepo/cluster1-backrest-shared-repo
```

4. Enable the standby mode in `spec.standby` and point to the repository:

```
spec:
 standby:
  enabled: true
  repoName: repo1
```

5. Create the version 2.x cluster:

```
$ kubectl apply -f deploy/cr.yaml
```


Percona LLC and/or its affiliates, © 2009 - 2024

Promote version 2.x cluster

Once the standby cluster is up and running, you can promote it.

1. Delete version 1.x cluster, but ensure that `spec.keepBackups` is set to `true`.

```
$ kubectl delete perconapgcluster cluster1
```

2. Promote version 2.x cluster by disabling the standby mode:

```
spec:
 standby:
  enabled: false
```

You can use version 2.x cluster now. Also the 2.x version is now managing the object storage with backups, so you should not start your old cluster.

Create the replication user

Right after disabling standby, run the following SQL commands as a PostgreSQL superuser. For example, you can login as the `postgres` user, or exec into the Pod and use `psql`:

- add the managed replication user

```
CREATE ROLE _crunchyrepl WITH LOGIN REPLICATION;
```

- allow for the replication user to execute the functions required as part of "rewinding"

```
GRANT EXECUTE ON function pg_catalog.pg_ls_dir(text, boolean, boolean) TO _crunchyrepl;
GRANT EXECUTE ON function pg_catalog.pg_stat_file(text, boolean) TO _crunchyrepl;
GRANT EXECUTE ON function pg_catalog.pg_read_binary_file(text) TO _crunchyrepl;
GRANT EXECUTE ON function pg_catalog.pg_read_binary_file(text, bigint, bigint, boolean) TO _crunchyrepl;
```

The above step will be automated in upcoming releases.

Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

5.3 Back up and restore

5.3.1 About backups

In this section you will learn how to set up and manage backups of your data using the Operator.

You can make backups in two ways:

- *On-demand*. You can do them manually at any moment.
- *Schedule backups*. Configure backups and their schedule in the deploy/cr.yaml file. The Operator makes them automatically according to the schedule.

What you need to know

BACKUP REPOSITORIES

To make backups, the Operator uses the open source pgBackRest backup and restore utility.

When the Operator creates a new PostgreSQL cluster, it also creates a special *pgBackRest repository* to facilitate the usage of the pgBackRest features. You can notice an additional `repo-host` Pod after the cluster creation.

A pgBackRest repository consists of the following Kubernetes objects:

- A Deployment,
- A Secret that contains information specific to the PostgreSQL cluster (e.g. SSH keys, AWS S3 keys, etc.),
- A Pod with a number of supporting scripts,
- A Service.

In the `/deploy/cr.yml` file, pgBackRest repositories are listed in the `backups.pgbackrest.repos` subsection. You can have up to 4 repositories as `repo1`, `repo2`, `repo3`, and `repo4`.

BACKUP TYPES

You can make the following types of backups:

- `full`: A full backup of all the contents of the PostgreSQL cluster,
- `differential`: A backup of only the files that have changed since the last full backup,
- `incremental`: Default. A backup of only the files that have changed since the last full or differential backup.

BACKUP STORAGE

You have the following options to store PostgreSQL backups outside the Kubernetes cluster:

- Cloud storage:
- Amazon S3, or any S3-compatible storage,
- Google Cloud Storage,
- Azure Blob Storage
- A Persistent Volume attached to the pgBackRest Pod.

Next steps

Ready to move forward? Configure backup storage

Percona LLC and/or its affiliates, © 2009 - 2024

Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-09-14

Percona LLC and/or its affiliates, © 2009 - 2024

5.3.2 Configure backup storage

Configure backup storage for your backup repositories in the

Percona LLC and/or its affiliates, © 2009 - 2024

`backups.pgbackrest.repos` section of the `deploy/cr.yaml` configuration file.

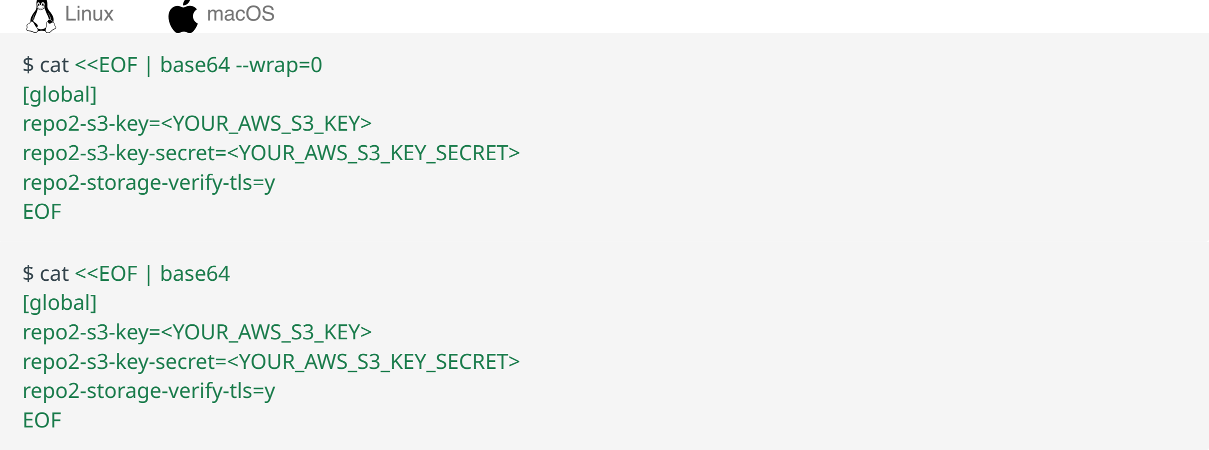
Percona LLC and/or its affiliates, © 2009 - 2024

To use S3-compatible storage for backups, you need to have the following S3-related information:

- The name of S3 bucket;
- The endpoint - the URL to access the bucket
- The region - the location of the bucket
- S3 credentials such as S3 key and secret to access the storage. These are stored in an encoded form in Kubernetes Secrets along with other sensitive information.

Configuration steps

1. Encode the S3 credentials and the pgBackRest repo name that you will use for backups. In this example, we use AWS S3 key and S3 key secret and `repo2`.




```
$ cat <<EOF | base64 --wrap=0
[global]
repo2-s3-key=<YOUR_AWS_S3_KEY>
repo2-s3-key-secret=<YOUR_AWS_S3_KEY_SECRET>
repo2-storage-verify-tls=y
EOF
```

```
$ cat <<EOF | base64
[global]
repo2-s3-key=<YOUR_AWS_S3_KEY>
repo2-s3-key-secret=<YOUR_AWS_S3_KEY_SECRET>
repo2-storage-verify-tls=y
EOF
```

The `repo2-storage-verify-tls` option in the above example enables TLS verification for pgBackRest (when set to `y` or simply omitted) or disables it, when set to `n`.

2. Create the Secret configuration file and specify the base64-encoded string from the previous step. The following is the example of the `cluster1-pgbackrest-secrets.yaml` Secret file:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-pgbackrest-secrets
type: Opaque
data:
  s3.conf: <base64-encoded-configuration-contents>
```

Note

This Secret can store credentials for several repositories presented as separate data keys.

3. Create the Secrets object from this YAML file. Replace the `<namespace>` placeholder with your value:

```
$ kubectl apply -f cluster1-pgbackrest-secrets.yaml -n <namespace>
```

4. Update your `deploy/cr.yaml` configuration. Specify the Secret file you created in the `backups.pgbackrest.configuration` subsection, and put all other S3 related information in the `backups.pgbackrest.repos` subsection under the repository name that you intend to use for backups. This name must match the name you used when you encoded S3 credentials on step 1.

For example, the S3 storage for the `repo2` repository looks as follows:

```yaml
...
backups:
  pgbackrest:
```


Percona LLC and/or its affiliates, © 2009 - 2024

Speed-up backups with pgBackRest asynchronous archiving

Backing up a database with high write-ahead logs (WAL) generation can be rather slow, because PostgreSQL archiving process is sequential, without any parallelism or batching. In extreme cases backup can be even considered unsuccessful by the Operator because of the timeout.

The pgBackRest tool used by the Operator can, if necessary, solve this problem by using the WAL asynchronous archiving feature.

You can set up asynchronous archiving in your storage configuration file for pgBackRest. Turn on the additional `archive-async` flag, and set the `process-max` value for `archive-push` and `archive-get` commands. Your storage configuration file may look as follows:

s3.conf

```
[global]
repo2-s3-key=REPLACE-WITH-AWS-ACCESS-KEY
repo2-s3-key-secret=REPLACE-WITH-AWS-SECRET-KEY
repo2-storage-verify-tls=n
repo2-s3-uri-style=path
archive-async=y
spool-path=/pgdata

[global:archive-get]
process-max=2

[global:archive-push]
process-max=4
```

No modifications are needed aside of setting these additional parameters. You can find more information about WAL asynchronous archiving in gpBackRest official documentation and in this blog pos.

Next steps

- Make an on-demand backup
- Make a scheduled backup

Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Last update: 2023-12-20

Percona LLC and/or its affiliates, © 2009 - 2024

5.3.3 Make scheduled backups

Backups schedule is defined on the per-repository basis in the `backups.pgbackrest.repos` subsection of the `deploy/cr.yaml` file.

You can supply each repository with a `schedules.<backup type>` key equal to an actual schedule that you specify in crontab format.

1. Before you start, make sure you have configured a backup storage.
2. Configure backup schedule in the `deploy/cr.yaml` file. The schedule is specified in crontab format as explained in Custom Resource options. The repository name must be the same as the one you defined in the backup storage configuration. The following example shows the schedule for `repo1` repository:

```
...
backups:
  pgbackrest:
  ...
      repos:
      - name: repo1
        schedules:
          full: "0 0 * * 6"
          differential: "0 1 * * 1-6"
        ...
```

1. Update the cluster:

```
$ kubectl apply -f deploy/cr.yaml
```

Next steps

Restore from a backup

Useful links

Backup retention

Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

5.3.4 Making on-demand backups

To make an on-demand backup manually, you need a backup configuration file. You can use the example of the backup configuration file deploy/backup.yaml:

```
apiVersion: pgv2.percona.com/v2
kind: PerconaPGBackup
metadata:
  name: backup1
spec:
  pgCluster: cluster1
  repoName: repo1
#  options:
#  - --type=full
```

Here's a sequence of steps to follow:

1. Before you start, make sure you have configured a backup storage.
2. In the `deploy/backup.yaml` configuration file, specify the cluster name and the repository name to be used for backups. The repository name must be the same as the one you defined in the backup storage configuration. It must also match the repository name specified in the `backups.pgbackrest.manual` subsection of the `deploy/cr.yaml` file.
3. If needed, you can add any pgBackRest command line options.
4. Make a backup with the following command:

```
$ kubectl apply -f deploy/backup.yaml
```

Tip

To list the backup, run:

```
$ kubectl get pg-backup
```

Next steps

Restore from a backup

Useful links

Backup retention

Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

5.3.5 Backup retention

The Operator supports setting pgBackRest retention policies for full and differential backups. When a full backup expires according to the retention policy, pgBackRest cleans up all the files related to this backup and to the write-ahead log. Thus, the expiration of a full backup with some incremental backups based on it results in expiring of all these incremental backups.

You can control backup retention by the following `pgBackRest` options:

- `--<repo name>-retention-full` how much full backups to retain,
- `--<repo name>-retention-diff` how much differential backups to retain.

Backup retention type can be either `count` (the number of backups to keep) or `time` (the number of days to keep a backup for).

You can set both backup type and retention policy for each of 4 repositories as follows.

```
backups:
  pgbackrest:
...
     global:
       repo1-retention-full: "14"
       repo1-retention-full-type: time
       ...
```

Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-09-14

Percona LLC and/or its affiliates, © 2009 - 2024

5.3.6 Restore the cluster from a previously saved backup

The Operator supports the ability to perform a full restore on a PostgreSQL cluster as well as a point-in-time-recovery. There are two ways to restore a cluster:

- restore to a new cluster using the dataSource.postgresCluster subsection,
- restore in-place to an existing cluster (note that this is destructive) using the backups.restore subsection.

Restore to a new PostgreSQL cluster

Restoring to a new PostgreSQL cluster allows you to take a backup and create a new PostgreSQL cluster that can run alongside an existing one. There are several scenarios where using this technique is helpful:

- Creating a copy of a PostgreSQL cluster that can be used for other purposes. Another way of putting this is *creating a clone*.
- Restore to a point-in-time and inspect the state of the data without affecting the current cluster.

To create a new PostgreSQL cluster from either the active one, or a former cluster whose pgBackRest repository still exists, use the dataSource.postgresCluster subsection options. The content of this subsection should copy the `backups` keys of the original cluster - ones needed to carry on the restore:

- `dataSource.postgresCluster.clusterName` should contain the new cluster name,
- `dataSource.postgresCluster.options` allow you to set the needed pgBackRest command line options,
- `dataSource.postgresCluster.repoName` should contain the name of the pgBackRest repository, while the actual storage configuration keys for this repository should be placed into `dataSource.pgbackrest.repo` subsection,
- `dataSource.pgbackrest.configuration.secret.name` should contain the name of a Kubernetes Secret with credentials needed to access cloud storage, if any.

Restore to an existing PostgreSQL cluster

To restore the previously saved backup, use a *backup restore* configuration file. The example of the backup configuration file is deploy/restore.yaml:

```
apiVersion: pgv2.percona.com/v2
kind: PerconaPGRestore
metadata:
  name: restore1
spec:
  pgCluster: cluster1
  repoName: repo1
  options:
  - --type=time
  - --target="2022-11-30 15:12:11+03"
```

The following keys are the most important ones:

- `pgCluster` specifies the name of your cluster,
- `repoName` specifies the name of one of the 4 pgBackRest repositories, already configured in the `backups.pgbackrest.repos` subsection,
- `options` passes through any pgBackRest command line options.

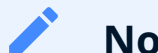
To start the restoration process, run the following command:

```
$ kubectl apply -f deploy/restore.yaml
```


Percona LLC and/or its affiliates, © 2009 - 2024

Restore the cluster with point-in-time recovery

Point-in-time recovery functionality allows users to revert the database back to a state before an unwanted change had occurred.



Note

For this feature to work, the Operator initiates a full backup immediately after the cluster creation, to use it as a basis for point-in-time recovery when needed (this backup is not listed in the output of the `kubectl get pg-backup` command).

You can set up a point-in-time recovery using the normal restore command of pgBackRest with few additional `spec.options` fields in `deploy/restore.yaml`:

- set `--type` option to `time`,
- set `--target` to a specific time you would like to restore to. You can use the typical string formatted as <YYYY-MM-DD HH:MM:DD>, optionally followed by a timezone offset: `"2021-04-16 15:13:32+00"` (`+00` in the above example means UTC),
- optional `--set` argument allows you to choose the backup which will be the starting point for point-in-time recovery. You can look through the available backups with the `kubectl get pg-backup` command to find out the proper backup name. This option must be specified if the target is one or more backups away from the current moment.

After setting these options in the *backup restore* configuration file, follow the standard restore instructions.

Note

Make sure you have a backup that is older than your desired point in time. You obviously can't restore from a time where you do not have a backup. All relevant write-ahead log files must be successfully pushed before you make the restore.

Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-22

Percona LLC and/or its affiliates, © 2009 - 2024

5.4 High availability and scaling

One of the great advantages brought by Kubernetes and the OpenShift platform is the ease of an application scaling. Scaling an application results in adding resources or Pods and scheduling them to available Kubernetes nodes.

Scaling can be vertical and horizontal. Vertical scaling adds more compute or storage resources to PostgreSQL nodes; horizontal scaling is about adding more nodes to the cluster. High availability looks technically similar, because it also involves additional nodes, but the reason is maintaining liveness of the system in case of server or network failures.

5.4.1 Vertical scaling

There are multiple components that Operator deploys and manages: PostgreSQL instances, pgBouncer connection pooler, etc. To add or reduce CPU or Memory you need to edit corresponding sections in the Custom Resource. We follow the structure for requests and limits that Kubernetes provides.

To add more resources to your PostgreSQL instances edit the following section in the Custom Resource:

```
spec:
...
  instances:
  - name: instance1
    replicas: 3
    resources:
      limits:
        cpu: 2.0
        memory: 4Gi
```

Use our reference documentation for the Custom Resource options for more details about other components.

5.4.2 High availability

Percona Operator allows you to deploy highly-available PostgreSQL clusters. There are two ways how to control replicas in your HA cluster:

1. Through changing `spec.instances.replicas` value
2. By adding new entry into `spec.instances`

5.4.3 Using `spec.instances.replicas`

For example, you have the following Custom Resource manifest:

```
spec:
...
  instances:
    - name: instance1
      replicas: 2
```

This will provision a cluster with two nodes - one Primary and one Replica. Add the node by changing the manifest...

```
spec:
...
```


Percona LLC and/or its affiliates, © 2009 - 2024

```yaml
  instances:
    - name: instance1
      replicas: 3
```

...and applying the Custom Resource:

```
$ kubectl apply -f deploy/cr.yaml
```

The Operator will provision a new replica node. It will be ready and available once data is synchronized from Primary.

5.4.4 Using `spec.instances`

Each instance's entry has its own set of parameters, like resources, storage configuration, sidecars, etc. When you add a new entry into instances, this creates replica PostgreSQL nodes, but with a new set of parameters. This can be useful in various cases:

- Test or migrate to new hardware
- Blue-green deployment of a new configuration
- Try out new versions of your sidecar containers

For example, you have the following Custom Resource manifest:

```yaml
spec:
...
  instances:
    - name: instance1
      replicas: 2
      dataVolumeClaimSpec:
        storageClassName: old-ssd
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 100Gi
```

Now you have a goal to migrate to new disks, which are coming with the `new-ssd` storage class. You can create a new instance entry. This will instruct the Operator to create additional nodes with the new configuration keeping your existing nodes intact.

```yaml
spec:
...
  instances:
    - name: instance1
      replicas: 2
      dataVolumeClaimSpec:
        storageClassName: old-ssd
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 100Gi
    - name: instance2
      replicas: 2
      dataVolumeClaimSpec:
        storageClassName: new-ssd
        accessModes:
        - ReadWriteOnce
```


Percona LLC and/or its affiliates, © 2009 - 2024

```yaml
      resources:
        requests:
          storage: 100Gi
```

5.4.5 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-06-22

Percona LLC and/or its affiliates, © 2009 - 2024

5.5 Using sidecar containers

The Operator allows you to deploy additional (so-called *sidecar*) containers to the Pod. You can use this feature to run debugging tools, some specific monitoring solutions, etc.

Note

Custom sidecar containers can easily access other components of your cluster.

Therefore they should be used carefully and by experienced users only.

5.5.1 Adding a sidecar container

You can add sidecar containers to PostgreSQL instance and pgBouncer Pods. Just use `sidecars` subsection in the `instances` or `proxy.pgBouncer` Custom Resource section in the `deploy/cr.yaml` configuration file. In this subsection, you should specify at least the name and image of your container, and possibly a command to run:

```
spec:
 instances:
  ....
  sidecars:
  - image: busybox
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo echo $(date -u) 'test' >> /dev/null; sleep 5; done"]
    name: my-sidecar-1
  ....
```

Apply your modifications as usual:

```
$ kubectl apply -f deploy/cr.yaml
```

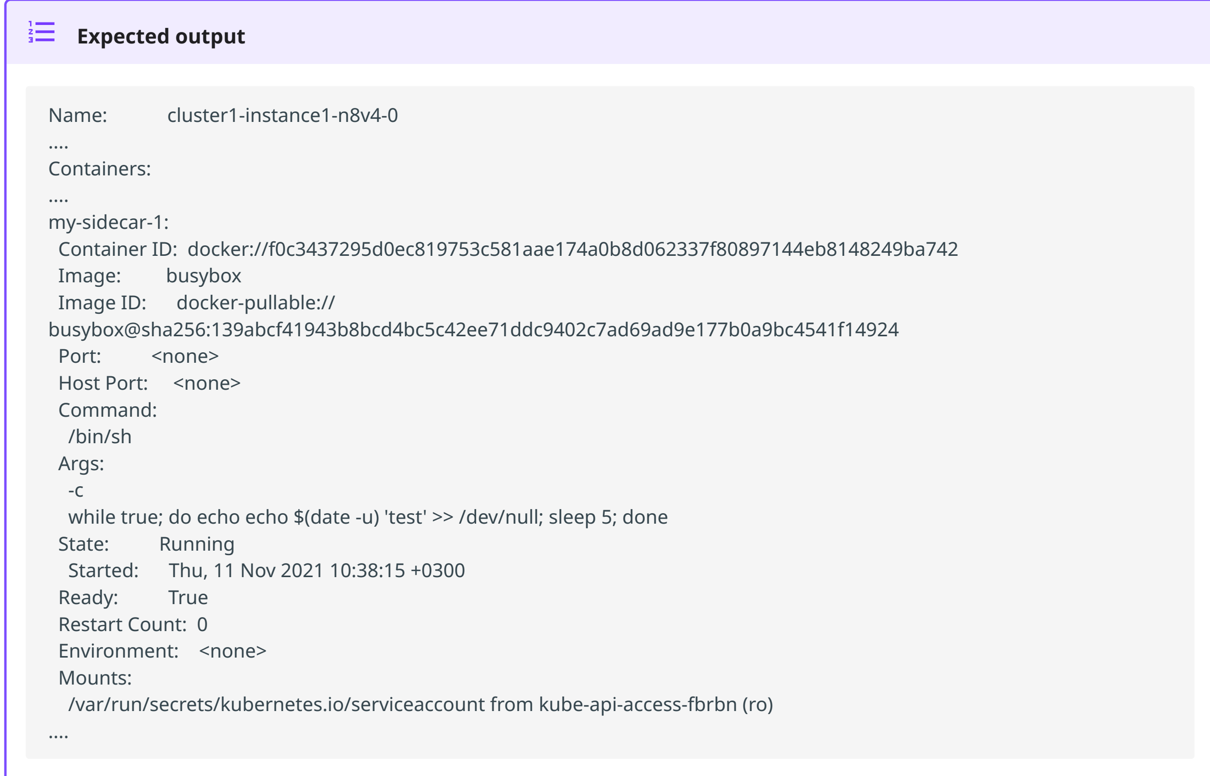
Note

More options suitable for the `sidecars` subsection can be found in the Custom Resource options reference.

Running `kubectl describe` command for the appropriate Pod can bring you the information about the newly created container:

```
$ kubectl describe pod cluster1-instance1
```


Percona LLC and/or its affiliates, © 2009 - 2024



Expected output

```
Name:         cluster1-instance1-n8v4-0
....
Containers:
....
my-sidecar-1:
  Container ID:  docker://f0c3437295d0ec819753c581aae174a0b8d062337f80897144eb8148249ba742
  Image:         busybox
  Image ID:      docker-pullable://
busybox@sha256:139abcf41943b8bcd4bc5c42ee71ddc9402c7ad69ad9e177b0a9bc4541f14924
  Port:          <none>
  Host Port:     <none>
  Command:
    /bin/sh
  Args:
    -c
    while true; do echo echo $(date -u) 'test' >> /dev/null; sleep 5; done
  State:         Running
    Started:     Thu, 11 Nov 2021 10:38:15 +0300
  Ready:         True
  Restart Count:  0
  Environment:    <none>
  Mounts:
    /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-fbrbn (ro)
....
```

5.5.2 Getting shell access to a sidecar container

You can login to your sidecar container as follows:

```
$ kubectl exec -it cluster1-instance1n8v4-0 -c my-sidecar-1 -- sh
/ #
```

5.5.3 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-05-03

Percona LLC and/or its affiliates, © 2009 - 2024

5.6 Pause/resume PostgreSQL cluster

There may be external situations when it is needed to pause your Cluster for a while and then start it back up (some works related to the maintenance of the enterprise infrastructure, etc.).

The `deploy/cr.yaml` file contains a special `spec.pause` key for this. Setting it to `true` gracefully stops the cluster:

```
spec:
  .......
  pause: true
```

To start the cluster after it was paused just revert the `spec.pause` key to `false`.

Note

There is an option also to put the cluster into a standby (read-only) mode instead of completely shutting it down. This is done by a special `spec.standby` key, which should be set to `true` for read-only state or should be set to `false` for normal cluster operation:

```
spec:
  .......
  standby: false
```

5.6.1 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

5.7 Monitor with Percona Monitoring and Management (PMM)

In this section you will learn how to monitor the health of Percona Distribution for PostgreSQL with Percona Monitoring and Management (PMM).

Note

Only PMM 2.x versions are supported by the Operator.

PMM is a client/server application. It includes the PMM Server and the number of PMM Clients running on each node with the database you wish to monitor.

A PMM Client collects needed metrics and sends gathered data to the PMM Server. As a user, you connect to the PMM Server to see database metrics on a number of dashboards.

PMM Server and PMM Client are installed separately.

5.7.1 Install PMM Server

You must have PMM server up and running. You can run PMM Server as a *Docker image*, a *virtual appliance*, or on an *AWS instance*. Please refer to the official PMM documentation for the installation instructions.

Percona LLC and/or its affiliates, © 2009 - 2024

5.7.2 Install PMM Client

To install PMM Client as a side-car container in your Kubernetes-based environment, do the following:

Percona LLC and/or its affiliates, © 2009 - 2024

1. Get the PMM API key from PMM Server. The API key must have the role "Admin". You need this key to authorize PMM Client within PMM Server.

Generate the PMM API key

You can query your PMM Server installation for the API Key using `curl` and `jq` utilities. Replace `<login>:<password>@<server_host>` placeholders with your real PMM Server login, password, and hostname in the following command:

```
$ API_KEY=$(curl --insecure -X POST -H "Content-Type: application/json" -d '{"name":"operator", "role": "Admin"}' "https://
<login>:<password>@<server_host>/graph/api/auth/keys" | jq .key)
```

Note

The API key is not rotated.

2. Specify the API key as the `PMM_SERVER_KEY` value in the deploy/secrets.yaml secrets file.

```
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-pmm-secret
type: Opaque
stringData:
  PMM_SERVER_KEY: ""
```

3. Create the Secrets object using the `deploy/secrets.yaml` file.

```
$ kubectl apply -f deploy/secrets.yaml -n postgres-operator
```

4. Update the `pmm` section in the deploy/cr.yaml file.

- Set `pmm.enabled = true`.
- Specify your PMM Server hostname / an IP address for the `pmm.serverHost` option. The PMM Server IP address should be resolvable and reachable from within your cluster.

```
  pmm:
    enabled: true
    image: percona/pmm-client:2.41.0
#    imagePullPolicy: IfNotPresent
    secret: cluster1-pmm-secret
    serverHost: monitoring-service
```

5. Update the cluster

```
$ kubectl apply -f deploy/cr.yaml -n postgres-operator
```

6. Check that corresponding Pods are not in a cycle of stopping and restarting. This cycle occurs if there are errors on the previous steps:

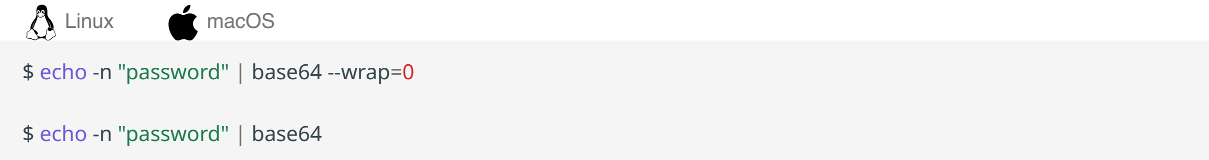
Percona LLC and/or its affiliates, © 2009 - 2024

```
$ kubectl get pods -n postgres-operator
$ kubectl logs <pod_name> -c pmm-client
```

5.7.3 Update the secrets file

The `deploy/secrets.yaml` file contains all values for each key/value pair in a convenient plain text format. But the resulting Secrets Objects contains passwords stored as base64-encoded strings. If you want to *update* the password field, you need to encode the new password into the base64 format and pass it to the Secrets Object.

To encode a password or any other parameter, run the following command:



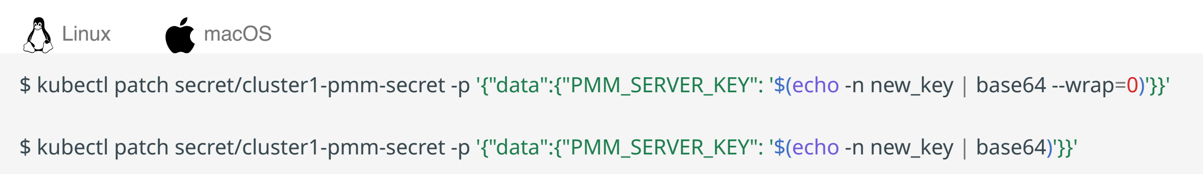
Linux

```
$ echo -n "password" | base64 --wrap=0
```

macOS

```
$ echo -n "password" | base64
```

For example, to set the new PMM API key in the `my-cluster-name-secrets` object, do the following:



Linux

```
$ kubectl patch secret/cluster1-pmm-secret -p '{"data":{"PMM_SERVER_KEY": '$(echo -n new_key | base64 --wrap=0)'}}'
```

macOS

```
$ kubectl patch secret/cluster1-pmm-secret -p '{"data":{"PMM_SERVER_KEY": '$(echo -n new_key | base64)'}}'
```

5.7.4 Check the metrics

Let's see how the collected data is visualized in PMM.

1. Log in to PMM server.
2. Click **PostgreSQL** from the left-hand navigation menu. You land on the **Instances Overview** page.
3. Click **PostgreSQL** → **Other dashboards** to see the list of available dashboards that allow you to drill down to the metrics you are interested in.

5.7.5 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

6. HowTo

6.1 Install Percona Distribution for PostgreSQL with customized parameters

You can customize the configuration of Percona Distribution for PostgreSQL and install it with customized parameters.

To check available configuration options, see deploy/cr.yaml and Custom Resource Options.

To customize the configuration when installing with `kubectl`, do the following:

1. Clone the repository with all manifests and source code by executing the following command:

```
$ git clone -b v2.3.1 https://github.com/percona/percona-postgresql-operator
```

2. Edit the required options and apply your modified `deploy/cr.yaml` file as follows:

```
$ kubectl apply -f deploy/cr.yaml -n postgres-operator
```

To install Percona Distribution for PostgreSQL with custom parameters using Helm, use the following command:

```
$ helm install --set key=value
```

You can pass any of the Operator's Custom Resource options as a `--set key=value[,key=value]` argument.

The following example deploys a PostgreSQL 16 based cluster in the `my-namespace` namespace, with enabled Percona Monitoring and Management (PMM):

```
$ helm install my-db percona/pg-db --version 2.3.1 --namespace my-namespace \
  --set postgresVersion=16 \
  --set pmm.enabled=true
```

6.1.1 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

6.2 How to deploy a standby cluster for Disaster Recovery

Disaster recovery is not optional for businesses operating in the digital age. With the ever-increasing reliance on data, system outages or data loss can be catastrophic, causing significant business disruptions and financial losses.

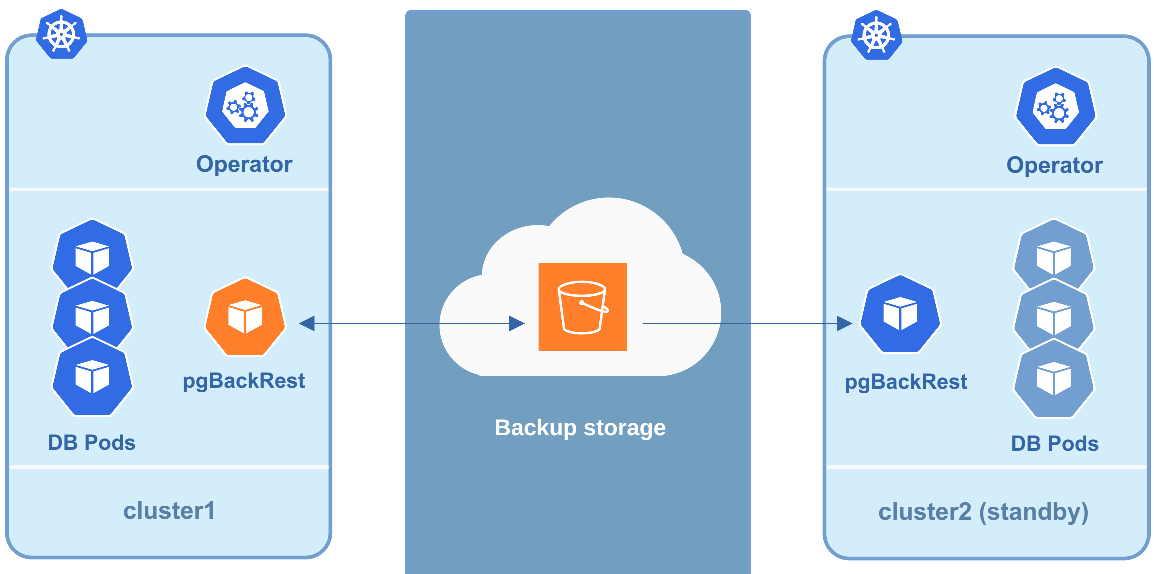
With multi-cloud or multi-regional PostgreSQL deployments, the complexity of managing disaster recovery only increases. This is where the Percona Operators come in, providing a solution to streamline disaster recovery for PostgreSQL clusters running on Kubernetes. With the Percona Operators, businesses can manage multi-cloud or hybrid-cloud PostgreSQL deployments with ease, ensuring that critical data is always available and secure, no matter what happens.

6.2.1 Solution overview

Operators automate routine tasks and remove toil. For standby, the Percona Operator for PostgreSQL version 2 provides the following options:

1. pgBackrest repo based standby
2. Streaming replication
3. Combination of (1) and (2)

This document describes the pgBackRest repo-based standby as the simplest one. The following is the architecture diagram:



1. This solution describes two Kubernetes clusters in different regions, clouds or running in hybrid mode (on-premises and cloud). One cluster is Main and the other is Disaster Recovery (DR)
2. Each cluster includes the following components:

a. Percona Operator

b. PostgreSQL cluster

c. pgBackrest

d. pgBouncer

3. pgBackrest on the Main site streams backups and Write Ahead Logs (WALs) to the object storage
4. pgBackrest on the DR site takes these backups and streams them to the standby cluster

Percona LLC and/or its affiliates, © 2009 - 2024

6.2.2 Deploy disaster recovery for PostgreSQL on Kubernetes

Configure Main site

1. Deploy the Operator using your favorite method. Once installed, configure the Custom Resource manifest, so that pgBackrest starts using the Object Storage of your choice. Skip this step if you already have it configured.
2. Configure the `backups.pgbackrest.repos` section by adding the necessary configuration. The below example is for Google Cloud Storage (GCS):

```
spec:
 backups:
  configuration:
   - secret:
      name: main-pgbackrest-secrets
  pgbackrest:
   repos:
   - name: repo1
     gcs:
      bucket: MY-BUCKET
```

   The `main-pgbackrest-secrets` value contains the keys for GCS. Read more about the configuration in the backup and restore tutorial.
3. Once configured, apply the custom resource:

```
$ kubectl apply -f deploy/cr.yaml
```

Expected output

```
perconapgcluster.pg.percona.com/standby created
```

The backups should appear in the object storage. By default pgBackrest puts them into the pgbackrest folder.

Configure DR site

The configuration of the disaster recovery site is similar to that of the Main site, with the only difference in standby settings.

The following manifest has `standby.enabled` set to `true` and points to the `repoName` where backups are (GCS in our case):

```
metadata:
 name: standby
spec:
...
 backups:
  configuration:
   - secret:
      name: standby-pgbackrest-secrets
  pgbackrest:
   repos:
   - name: repo1
     gcs:
      bucket: MY-BUCKET
 standby:
```


Percona LLC and/or its affiliates, © 2009 - 2024

```
    enabled: true
    repoName: repo1
```

Deploy the standby cluster by applying the manifest:

```
$ kubectl apply -f deploy/cr.yaml
```

Expected output

```
perconapgcluster.pg.percona.com/standby created
```

6.2.3 Failover

In case of the Main site failure or in other cases, you can promote the standby cluster. The promotion effectively allows writing to the cluster. This creates a net effect of pushing Write Ahead Logs (WALs) to the pgBackrest repository. It might create a split-brain situation where two primary instances attempt to write to the same repository. To avoid this, make sure the primary cluster is either deleted or shut down before trying to promote the standby cluster.

Once the primary is down or inactive, promote the standby through changing the corresponding section:

```
spec:
  standby:
    enabled: false
```

Now you can start writing to the cluster.

Split brain

There might be a case, where your old primary comes up and starts writing to the repository. To recover from this situation, do the following:

1. Keep only one primary with the latest data running
2. Stop the writes on the other one
3. Take the new full backup from the primary and upload it to the repo

Automate the failover

Automated failover consists of multiple steps and is outside of the Operator's scope. There are a few steps that you can take to reduce the Recovery Time Objective (RTO). To detect the failover we recommend having the 3rd site to monitor both DR and Main sites. In this case you can be sure that Main really failed and it is not a network split situation.

Another aspect of automation is to switch the traffic for the application from Main to Standby after promotion. It can be done through various Kubernetes configurations and heavily depends on how your networking and application are designed. The following options are quite common:

1. Global Load Balancer - various clouds and vendors provide their solutions
2. Multi Cluster Services or MCS - available on most of the public clouds
3. Federation or other multi-cluster solutions

Percona LLC and/or its affiliates, © 2009 - 2024

6.2.4 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-07-20

Percona LLC and/or its affiliates, © 2009 - 2024

6.3 Use Docker images from a custom registry

Using images from a private Docker registry may be required for privacy, security or other reasons. In these cases, Percona Operator for PostgreSQL allows the use of a custom registry. The following example illustrates how this can be done by the example of the Operator deployed in the OpenShift environment.

6.3.1 Prerequisites

1. First of all login to the OpenShift and create project.

```
$ oc login
Authentication required for https://192.168.1.100:8443 (openshift)
Username: admin
Password:
Login successful.
$ oc new-project pg
Now using project "pg" on server "https://192.168.1.100:8443".
```

2. There are two things you will need to configure your custom registry access:

- the token for your user,
- your registry IP address.

The token can be found with the following command:

```
$ oc whoami -t
ADO8CqCDappWR4hxjfDqwijEHei31yXAvWg61Jg210s
```

And the following one tells you the registry IP address:

```
$ kubectl get services/docker-registry -n default
NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
docker-registry   ClusterIP   172.30.162.173   <none>        5000/TCP   1d
```

3. Use the user token and the registry IP address to login to the registry:

```
$ docker login -u admin -p ADO8CqCDappWR4hxjfDqwijEHei31yXAvWg61Jg210s 172.30.162.173:5000
```

Expected output

```
Login Succeeded
```

4. Use the Docker commands to pull the needed image by its SHA digest:

```
$ docker pull docker.io/perconalab/percona-postgresql-
operator@sha256:991d6049059e5eb1a74981290d829a5fb4ab0554993748fde1e67b2f46f26bf0
```


Percona LLC and/or its affiliates, © 2009 - 2024

Expected output

```
Trying to pull repository docker.io/perconalab/percona-postgresql-operator ...
sha256:991d6049059e5eb1a74981290d829a5fb4ab0554993748fde1e67b2f46f26bf0: Pulling from docker.io/perconalab/percona-server-mongodb
Digest: sha256:991d6049059e5eb1a74981290d829a5fb4ab0554993748fde1e67b2f46f26bf0
Status: Image is up to date for docker.io/perconalab/percona-postgresql-operator@sha256:991d6049059e5eb1a74981290d829a5fb4ab0554993748fde1e67b2f46f26bf0
```

You can find correct names and SHA digests in the current list of the Operator-related images officially certified by Percona.

5. The following method can push an image to the custom registry for the example OpenShift `pg` project:

```
$ docker tag \
    docker.io/perconalab/percona-postgresql-operator@sha256:991d6049059e5eb1a74981290d829a5fb4ab0554993748fde1e67b2f46f26bf0 \
    172.30.162.173:5000/psmdb/percona-postgresql-operator:16
$ docker push 172.30.162.173:5000/pg/percona-postgresql-operator:16
```

6. Verify the image is available in the OpenShift registry with the following command:

```
$ oc get is
```

Expected output

```
NAME                          DOCKER REPO                                                TAGS    UPDATED
percona-postgresql-operator   docker-registry.default.svc:5000/pg/percona-postgresql-operator   16   2 hours ago
```

7. When the custom registry image is available, edit the the `image:` option in `deploy/operator.yaml` configuration file with a Docker Repo + Tag string (it should look like `docker-registry.default.svc:5000/pg/percona-postgresql-operator:16`)

Note

If the registry requires authentication, you can specify the `imagePullSecrets` option for all images.

8. Repeat steps 3-5 for other images, and update corresponding options in the `deploy/cr.yaml` file.
9. Now follow the standard Percona Operator for PostgreSQL installation instruction.

6.3.2 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

6.4 Add custom PostgreSQL extensions

One of the specific PostgreSQL features is the ability to provide it with additional functionality via Extensions. Percona Distribution for PostgreSQL supports a number of extensions, making this list available for the database cluster managed by the Operator as well.

Still there are cases when the needed extension is not in this list, or when it's a custom extension developed by the end-user. Adding more extensions is not an easy task in case of a containerized database in Kubernetes-based environment, as normally it would make the user to build a custom PostgreSQL image.

Still, starting from the Operator version 2.3 there is an alternative way to extend Percona Distribution for PostgreSQL by downloading prepackaged extensions from an external storage on the fly, as defined in the `extensions` section of the Operator Custom Resource.

6.4.1 Enabling or disabling built-in extensions

Percona Distribution for PostgreSQL built-in extensions can be easily enabled or disabled in the `extensions.builtin` subsection of the `deploy/cr.yaml` configuration file as follows:

```
extensions:
  ...
  builtin:
    pg_stat_monitor: true
    pg_audit: true
```

Apply changes after editing with `kubectl apply -f deploy/cr.yaml` command.

Note

Editing this section and applying it is causing Pods restart.

6.4.2 Adding custom extensions

Custom extensions are downloaded by the Operator from the cloud storage. User is in charge for properly packaging extension and uploading it to the storage.

Packaging custom extensions

Custom extension needs specific packaging to make the Operator able using it. The package must be a `.tar.gz` archive with all required files in a the correct directory structure.

1. Control file must be in `SHAREDIR/extension` directory
2. All required SQL script files must be in `SHAREDIR/extension` directory (there must be at least one SQL script)
3. Any shared library must be in `LIBDIR`

Note

In case of Percona Distribution for PostgreSQL images, `SHAREDIR` corresponds to `/usr/pgsql-${PG_MAJOR}/share` and `LIBDIR` to `/usr/pgsql-${PG_MAJOR}/lib`.

For example, the directory for `pg_cron` extension should look as follows:

Percona LLC and/or its affiliates, © 2009 - 2024

```
$ tree ~/pg_cron-1.6.1/
/home/user/pg_cron-1.6.1/
└── usr
    └── pgsql-15
        ├── lib
        │   └── pg_cron.so
        └── share
            └── extension
                ├── pg_cron--1.0--1.1.sql
                ├── pg_cron--1.0.sql
                ├── pg_cron--1.1--1.2.sql
                ├── pg_cron--1.2--1.3.sql
                ├── pg_cron--1.3--1.4.sql
                ├── pg_cron--1.4--1.4-1.sql
                ├── pg_cron--1.4-1--1.5.sql
                ├── pg_cron--1.5--1.6.sql
                └── pg_cron.control
```

The archive must be created with `usr` at the root and the name must conform `${EXTENSION}-pg${PG_MAJOR}-${EXTENSION_VERSION}`:

```
$ cd pg_cron-1.6.1/
$ tar -czf pg_cron-pg15-1.6.1.tar.gz usr/
```

Note

To understand which files are required for given extension could be not an easy task. One of the option to figure this out would be building and installing the extension from source on a virtual machine with Percona Distribution for PostgreSQL and copy all the installed files to the archive.

Percona LLC and/or its affiliates, © 2009 - 2024

6.4.3 Configuring custom extension loading

When the extension is packaged, it should be uploaded to the cloud storage (for now, Amazon S3 is the only supported storage type). When the upload is done, the storage and extension details should be specified in the Custom Resource to make the Operator download and install it.

Percona LLC and/or its affiliates, © 2009 - 2024

1. The Operator will need the following data to access extensions stored on the Amazon S3:

- the `metadata.name` key is the name which you wll further use to refer your Kubernetes Secret,
- the `data.AWS_ACCESS_KEY_ID` and `data.AWS_SECRET_ACCESS_KEY` keys are base64-encoded credentials used to access the storage (obviously these keys should contain proper values to make the access possible).

   Create the Secrets file with these base64-encoded keys as follows:

   extensions-secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-extensions-secret
type: Opaque
data:
  AWS_ACCESS_KEY_ID: <base64 encoded secret>
  AWS_SECRET_ACCESS_KEY: <base64 encoded secret>
```

   Note

   You can use the following command to get a base64-encoded string from a plain text one:

   in Linux in macOS

   For GNU/Linux:

```
$ echo -n 'plain-text-string' | base64 --wrap=0
```

   For Apple macOS:

```
$ echo -n 'plain-text-string' | base64
```

   Once the editing is over, create the Kubernetes Secret object as follows:

```
$ kubectl apply -f extensions-secret.yaml
```

2. Storage credentials are specified in the Custom Resource `extensions.storage` subsection. The appropriate fragment of the `deploy/cr.yaml` configuration file should look as follows:

```yaml
extensions:
  ...
  storage:
    type: s3
    bucket: pg-extensions
    region: eu-central-1
    secret:
      name: cluster1-extensions-secret
```

3. When the storage is configured, and the archive with the extension is already present in the appropriate bucket, the extension itself can be specified to the Operator in the Custom Resource via the `deploy/cr.yaml` configuration file as in the following example:

```yaml
extensions:
  ...
  custom:
```


Percona LLC and/or its affiliates, © 2009 - 2024

```yaml
- name: pg_cron
  version: 1.6.1
```

The installed extension will not be enabled by default. Enabling it in can be done for desired databases using the `CREATE EXTENSION` statement:

```sql
CREATE EXTENSION pg_cron;
```

Also, some extensions (such as `pg_cron`) can be used only if added to `shared_preload_libraries`. Users can do it via the `deploy/cr.yaml` configuration file as follows:

```
yaml
...
patroni:
  dynamicConfiguration:
    postgresql:
      parameters:
        shared_preload_libraries: pg_cron
        ...
```

6.4.4 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

6.5 Percona Operator for PostgreSQL single-namespace and multi-namespace deployment

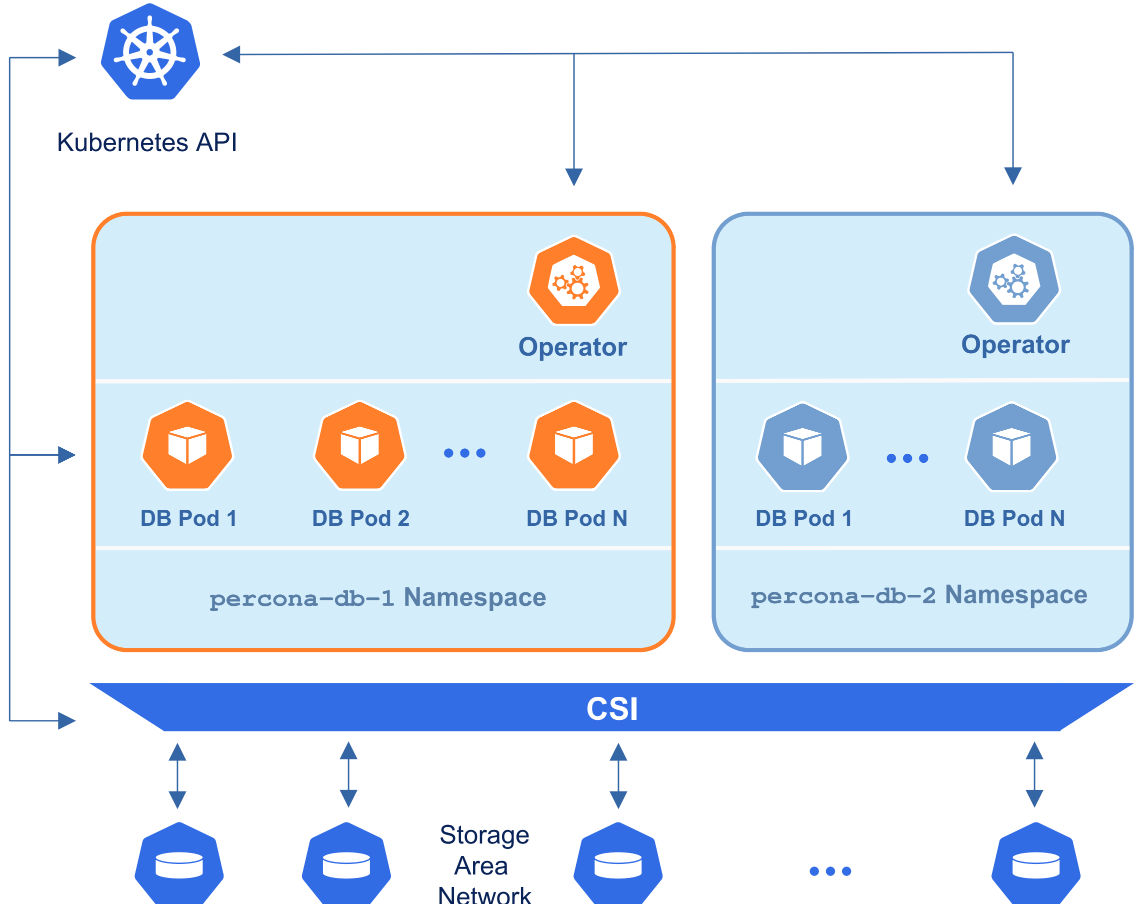
There are two design patterns that you can choose from when deploying Percona Operator for PostgreSQL and PostgreSQL clusters in Kubernetes:

- Namespace-scope - one Operator per Kubernetes namespace,
- Cluster-wide - one Operator can manage clusters in multiple namespaces.

This how-to explains how to configure Percona Operator for PostgreSQL for each scenario.

6.5.1 Namespace-scope

By default, Percona Operator for PostgreSQL functions in a specific Kubernetes namespace. You can create one during installation (like it is shown in the installation instructions) or just use the default namespace. This approach allows several Operators to co-exist in one Kubernetes-based environment, being separated in different namespaces:



Normally this is a recommended approach, as isolation minimizes impact in case of various failure scenarios. This is the default configuration of our Operator.

Percona LLC and/or its affiliates, © 2009 - 2024

Let's say you will use a Kubernetes Namespace called `percona-db-1`.

1. Clone `percona-postgresql-operator` repository:

```
$ git clone -b v2.3.1 https://github.com/percona/percona-postgresql-operator
$ cd percona-postgresql-operator
```

2. Create your `percona-db-1` Namespace (if it doesn't yet exist) as follows:

```
$ kubectl create namespace percona-db-1
```

3. Deploy the Operator using the following command:

```
$ kubectl apply --server-side -f deploy/bundle.yaml -n percona-db-1
```

4. Once Operator is up and running, deploy the database cluster itself:

```
$ kubectl apply -f deploy/cr.yaml -n percona-db-1
```

You can deploy multiple clusters in this namespace.

Add more namespaces

What if there is a need to deploy clusters in another namespace? The solution for namespace-scope deployment is to have more than one Operator. We will use the `percona-db-2` namespace as an example.

1. Create your `percona-db-2` namespace (if it doesn't yet exist) as follows:

```
$ kubectl create namespace percona-db-2
```

2. Deploy the Operator:

```
$ kubectl apply --server-side -f deploy/bundle.yaml -n percona-db-2
```

3. Once Operator is up and running deploy the database cluster itself:

```
$ kubectl apply -f deploy/cr.yaml -n percona-db-2
```

Note

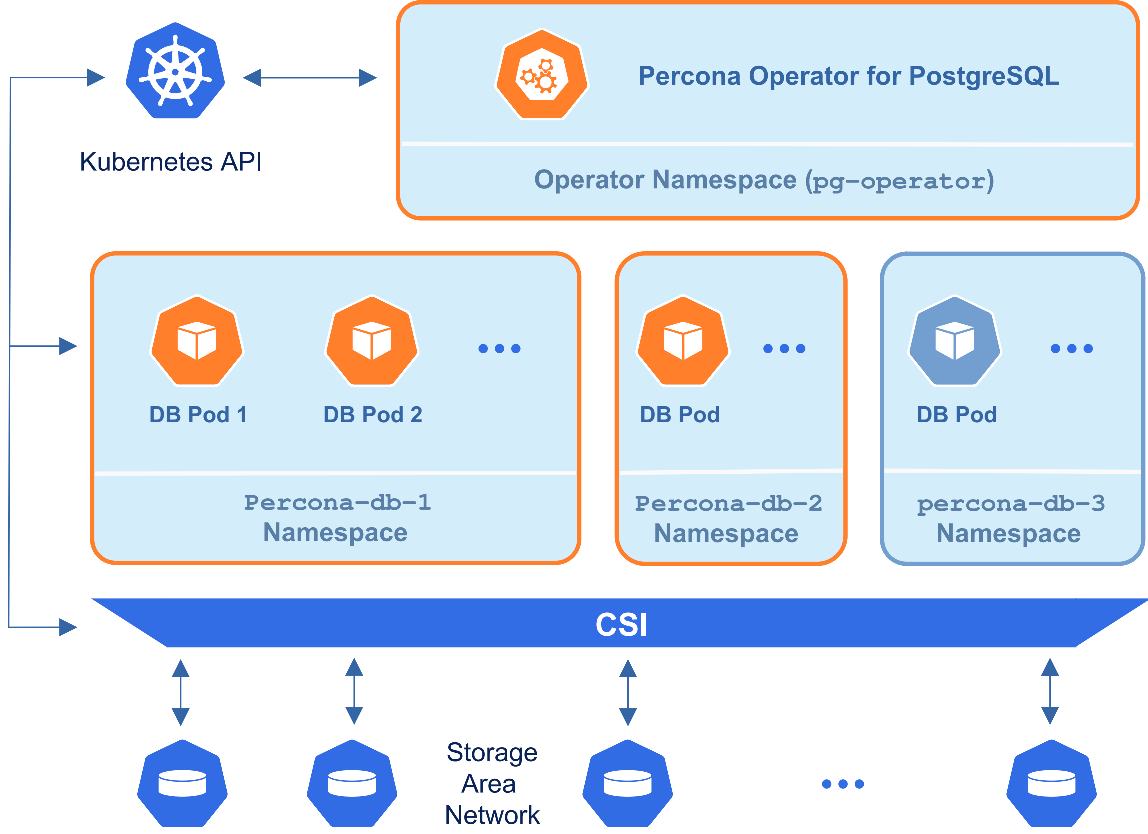
Cluster names may be the same in different namespaces.

6.5.2 Install the Operator cluster-wide

Sometimes it is more convenient to have one Operator watching for Percona Distribution for PostgreSQL custom resources in several namespaces.

We recommend running Percona Operator for PostgreSQL in a traditional way, limited to a specific namespace, to limit the blast radius. But it is possible to run it in so-called *cluster-wide* mode, one Operator watching several namespaces, if needed:

Percona LLC and/or its affiliates, © 2009 - 2024



To use the Operator in such cluster-wide mode, you should install it with a different set of configuration YAML files, which are available in the deploy folder and have filenames with a special `cw-` prefix: e.g. `deploy/cw-bundle.yaml`.

While using this cluster-wide versions of configuration files, you should set the following information there:

- `subjects.namespace` option should contain the namespace which will host the Operator,
- `WATCH_NAMESPACE` key-value pair in the `env` section should have `value` equal to a comma-separated list of the namespaces to be watched by the Operator, *and* the namespace in which the Operator resides. If this key is set to a blank string, the Operator will watch **only the namespace it runs in**, which would be the same as single-namespace deployment.

Percona LLC and/or its affiliates, © 2009 - 2024

The following simple example shows how to install Operator cluster-wide on Kubernetes.

1. Clone `percona-postgresql-operator` repository:

```
$ git clone -b v2.3.1 https://github.com/percona/percona-postgresql-operator
$ cd percona-postgresql-operator
```

2. Let's suppose that Operator's namespace should be the `pg-operator` one. Create it as follows:

```
$ kubectl create namespace pg-operator
```

3. Edit the `deploy/cw-bundle.yaml` configuration file to make sure it contains proper namespace name for the Operator:

```
...
subjects:
- kind: ServiceAccount
  name: percona-postgresql-operator
  namespace: pg-operator
...
```

4. Apply the `deploy/cw-bundle.yaml` file with the following command:

```
$ kubectl apply -f deploy/cw-bundle.yaml -n pg-operator
```

   Right now the operator deployed in cluster-wide mode will monitor all namespaces in the cluster, either already existing or newly created ones.

5. Create the namespace you have chosen for the cluster, if needed. let's call it `percona-db-1` for example:

```
$ kubectl create namespace percona-db-1
```

6. Deploy the cluster in the namespace of your choice:

```
$ kubectl apply -f deploy/cr.yaml -n percona-db-1
```

6.5.3 Verifying the cluster operation

When creation process is over, you can try to connect to the cluster.

Percona LLC and/or its affiliates, © 2009 - 2024

During the installation, the Operator has generated several secrets, including the one with password for default PostgreSQL user. This default user has the same login name as the cluster name.

1. Use kubectl get secrets command to see the list of Secrets objects. The Secrets object you are interested in is named as <cluster_name>-pguser-<cluster_name> (substitute <cluster_name> with the name of your Percona Distribution for PostgreSQL Cluster). The default variant will be cluster1-pguser-cluster1 .
2. Use the following command to get the password of this user:

```
$ kubectl get secret <cluster_name>-<user_name>-<cluster_name> -n postgres-operator --template='{{.data.password |
base64decode}}{{"\n"}}'
```

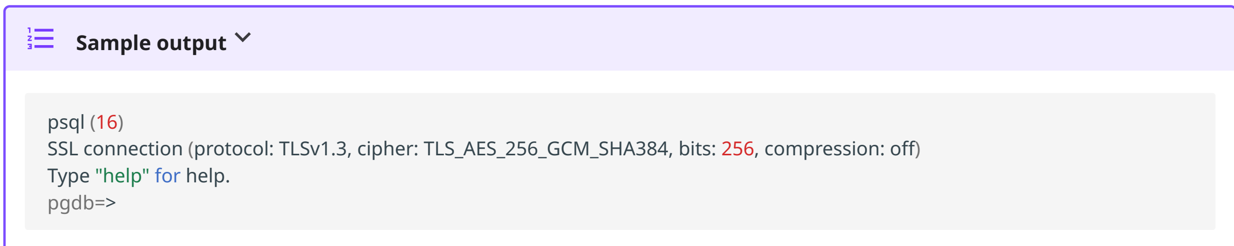
3. Create a pod and start Percona Distribution for PostgreSQL inside. The following command will do this, naming the new Pod pg-client :

```
$ kubectl run -i --rm --tty pg-client --image=perconalab/percona-distribution-postgresql:16 --restart=Never -- bash -il
```

   Executing it may require some time to deploy the corresponding Pod.

4. Run a container with psql tool and connect its console output to your terminal. This command will connect you as a cluster1 user to a cluster1 database via the PostgreSQL interactive terminal.

```
[postgres@pg-client /]$ PGPASSWORD='pguser_password' psql -h cluster1-pgbouncer.postgres-operator.svc -p 5432 -U
cluster1 cluster1
```



Sample output

```
psql (16)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.
pgdb=>
```

6.5.4 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

6.6 Delete Percona Operator for PostgreSQL

When cleaning up your Kubernetes environment (e.g., moving from a trial deployment to a production one, or testing experimental configurations), you may need to remove some (or all) of the following objects:

- Percona Distribution for PosgreSQL cluster managed by the Operator
- Percona Operator for PostgreSQL itself
- Custom Resource Definition deployed with the Operator

6.6.1 Delete a database cluster

You can delete the Percona Distribution for PosgreSQL cluster managed by the Operator by deleting the appropriate Custom Resource.

Note

There are two finalizers defined in the Custom Resource, which define whether TLS-related objects and data volumes should be deleted or preserved when the cluster is deleted.

- `finalizers.percona.com/delete-ssl` : if present, objects, created for SSL (Secret, certificate, and issuer) are deleted when the cluster deletion occurs.
- `finalizers.percona.com/delete-pvc` : if present, Persistent Volume Claims for the database cluster Pods are deleted when the cluster deletion occurs.

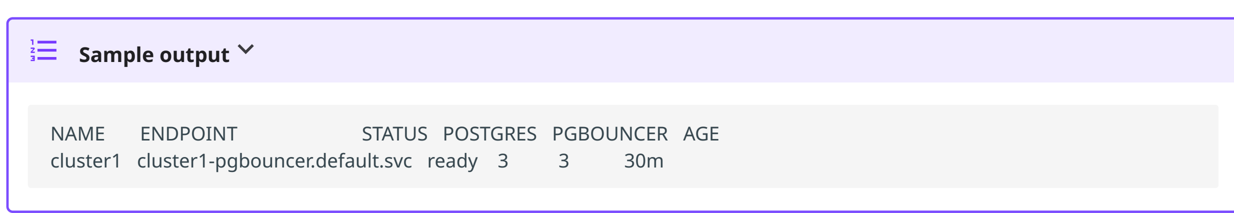
Both finalizers are off by default in the `deploy/cr.yaml` configuration file, and this allows you to recreate the cluster without losing data, credentials for the system users, etc.

Percona LLC and/or its affiliates, © 2009 - 2024

Here's a sequence of steps to follow:

1. List Custom Resources, replacing the `<namespace>` placeholder with your namespace.

```
$ kubectl get pg -n <namespace>
```

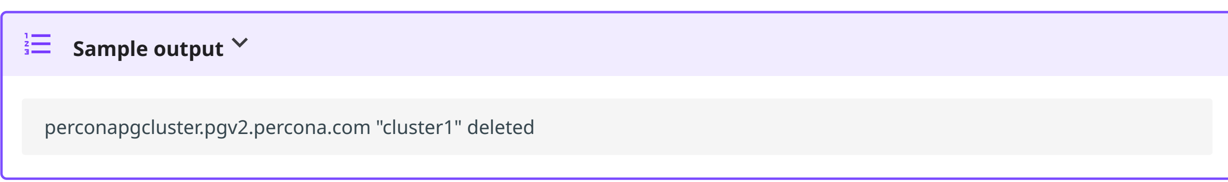


Sample output

```
NAME      ENDPOINT                      STATUS   POSTGRES   PGBOUNCER   AGE
cluster1  cluster1-pgbouncer.default.svc ready   3          3           30m
```

2. Delete the Custom Resource with the name of your cluster (for example, let's use the default `cluster1` name).

```
$ kubectl delete pg cluster1 -n <namespace>
```

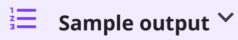


Sample output

```
perconapgcluster.pgv2.percona.com "cluster1" deleted
```

3. Check that the cluster is deleted by listing the available Custom Resources once again.

```
$ kubectl get pg -n <namespace>
```



Sample output

```
No resources found in <namespace> namespace.
```

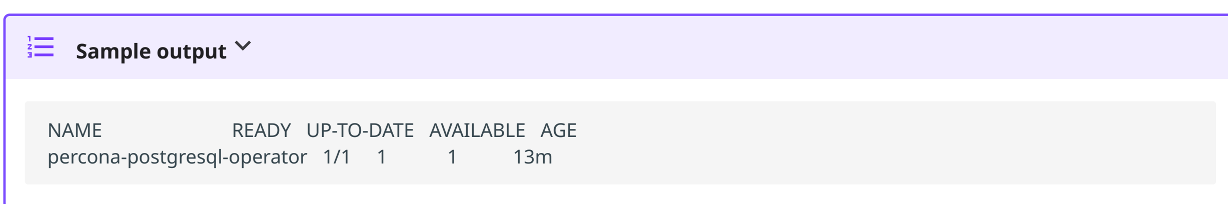

Percona LLC and/or its affiliates, © 2009 - 2024

6.6.2 Delete the Operator

You can uninstall the Operator by deleting the Deployments related to it.

1. List the deployments. Replace the `<namespace>` placeholder with your namespace.

```
$ kubectl get deploy -n <namespace>
```



Sample output

```
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
percona-postgresql-operator   1/1     1            1           13m
```

2. Delete the `percona-*` deployment

```
$ kubectl delete deploy percona-postgresql-operator -n <namespace>
```

3. Check that the Operator is deleted by listing the Pods. As a result you should have no Pods related to it.

```
$ kubectl get pods -n <namespace>
```

Sample output

```
No resources found in <namespace> namespace.
```

6.6.3 Delete Custom Resource Definition

If you are not just deleting the Operator and PostgreSQL cluster from a specific namespace, but want to clean up your entire Kubernetes environment, you can also delete the CustomRecourceDefinitions (CRDs).

Warning

CRDs in Kubernetes are non-namespaced but are available to the whole environment. This means that you shouldn't delete CRD if you still have the Operator and database cluster in some namespace.

Percona LLC and/or its affiliates, © 2009 - 2024

You can delete CRD as follows:

1. List the CRDs:

```
$ kubectl get crd
```

Sample output

```
allowlistedv2workloads.auto.gke.io              2023-09-07T14:15:30Z
allowlistedworkloads.auto.gke.io                2023-09-07T14:15:29Z
audits.warden.gke.io                            2023-09-07T14:15:32Z
backendconfigs.cloud.google.com                 2023-09-07T14:15:41Z
capacityrequests.internal.autoscaling.gke.io    2023-09-07T14:15:25Z
frontendconfigs.networking.gke.io               2023-09-07T14:15:41Z
managedcertificates.networking.gke.io           2023-09-07T14:15:41Z
memberships.hub.gke.io                          2023-09-07T14:15:30Z
perconapgbackups.pgv2.percona.com               2023-09-07T14:28:59Z
perconapgclusters.pgv2.percona.com              2023-09-07T14:29:02Z
perconapgrestores.pgv2.percona.com              2023-09-07T14:29:03Z
postgresclusters.postgres-operator.crunchydata.com  2023-09-07T14:29:06Z
serviceattachments.networking.gke.io            2023-09-07T14:15:44Z
servicenetworkendpointgroups.networking.gke.io  2023-09-07T14:15:43Z
storagestates.migration.k8s.io                  2023-09-07T14:15:53Z
storageversionmigrations.migration.k8s.io       2023-09-07T14:15:53Z
updateinfos.nodemanagement.gke.io               2023-09-07T14:15:55Z
volumesnapshotclasses.snapshot.storage.k8s.io   2023-09-07T14:15:52Z
volumesnapshotcontents.snapshot.storage.k8s.io  2023-09-07T14:15:52Z
volumesnapshots.snapshot.storage.k8s.io         2023-09-07T14:15:52Z
```

2. Now delete the `percona*.pgv2.percona.com` CRDs:

```
$ kubectl delete crd perconapgbackups.pgv2.percona.com perconapgclusters.pgv2.percona.com
perconapgrestores.pgv2.percona.com
```

Sample output

```
customresourcedefinition.apiextensions.k8s.io "perconapgbackups.pgv2.percona.com" deleted
customresourcedefinition.apiextensions.k8s.io "perconapgclusters.pgv2.percona.com" deleted
customresourcedefinition.apiextensions.k8s.io "perconapgrestores.pgv2.percona.com" deleted
```

6.6.4 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

6.7 Monitor Kubernetes

Monitoring the state of the database is crucial to timely identify and react to performance issues. Percona Monitoring and Management (PMM) solution enables you to do just that.

However, the database state also depends on the state of the Kubernetes cluster itself. Hence it’s important to have metrics that can depict the state of the Kubernetes cluster.

This document describes how to set up monitoring of the Kubernetes cluster health. This setup has been tested with the PMM Server as the centralized data storage and the Victoria Metrics Kubernetes monitoring stack as the metrics collector. These steps may also apply if you use another Prometheus-compatible storage.

6.7.1 Pre-requisites

To set up monitoring of Kubernetes, you need the following:

1. PMM Server up and running. You can run PMM Server as a Docker image, a virtual appliance, or on an AWS instance. Please refer to the official PMM documentation for the installation instructions.
2. Helm v3.
3. kubectl.
4. The PMM Server API key. The key must have the role “Admin”.

   Get the PMM API key:

   Generate the PMM API key

   You can query your PMM Server installation for the API Key using curl and jq utilities. Replace <login>:<password>@<server_host> placeholders with your real PMM Server login, password, and hostname in the following command:

```
$ API_KEY=$(curl --insecure -X POST -H "Content-Type: application/json" -d {"name":"operator", "role": "Admin"}' "https://<login>:<password>@<server_host>/graph/api/auth/keys" | jq .key)
```

Note

The API key is not rotated.

Percona LLC and/or its affiliates, © 2009 - 2024

6.7.2 Install the Victoria Metrics Kubernetes monitoring stack

Percona LLC and/or its affiliates, © 2009 - 2024

1. To install the Victoria Metrics Kubernetes monitoring stack with the default parameters, use the quick install command. Replace the following placeholders with your values:

- `API-KEY` - The API key of your PMM Server
- `PMM-SERVER-URL` - The URL to access the PMM Server
- `UNIQUE-K8s-CLUSTER-IDENTIFIER` - Identifier for the Kubernetes cluster. It can be the name you defined during the cluster creation.

  You should use a unique identifier for each Kubernetes cluster. The use of the same identifer for more than one Kubernetes cluster will result in the conflicts during the metrics collection.

- `NAMESPACE` - The namespace where the Victoria metrics Kubernetes stack will be installed. If you haven't created the namespace before, it will be created during the command execution.

  We recommend to use a separate namespace like `monitoring-system`.

```
$ curl -fsL  https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/main/vm-operator-k8s-stack/quick-install.sh | bash -s -- --api-key <API-KEY> --pmm-server-url <PMM-SERVER-URL> --k8s-cluster-id <UNIQUE-K8s-CLUSTER-IDENTIFIER> --namespace <NAMESPACE>
```

Note

The Prometheus node exporter is not installed by default since it requires privileged containers with the access to the host file system. If you need the metrics for Nodes, add the `--node-exporter-enabled` flag as follows:

```
$ curl -fsL  https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/main/vm-operator-k8s-stack/quick-install.sh | bash -s -- --api-key <API-KEY> --pmm-server-url <PMM-SERVER-URL> --k8s-cluster-id <UNIQUE-K8s-CLUSTER-IDENTIFIER> --namespace <NAMESPACE> --node-exporter-enabled
```

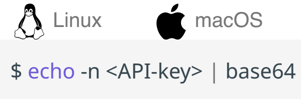
You may need to customize the default parameters of the Victoria metrics Kubernetes stack.

- Since we use the PMM Server for monitoring, there is no need to store the data in Victoria Metrics Operator. Therefore, the Victoria Metrics Helm chart is installed with the `vmsingle.enabled` and `vmcluster.enabled` parameters set to `false` in this setup.
- Check all the role-based access control (RBAC) rules of the `victoria-metrics-k8s-stack` chart and the dependencies chart, and modify them based on your requirements.

CONFIGURE AUTHENTICATION IN PMM

To access the PMM Server resources and perform actions on the server, configure authentication.

1. Encode the PMM Server API key with base64.




```
$ echo -n <API-key> | base64 --wrap=0
```

```
$ echo -n <API-key> | base64
```

2. Create the Namespace where you want to set up monitoring. The following command creates the Namespace `monitoring-system`. You can specify a different name. In the latter steps, specify your namespace instead of the <namespace> placeholder.

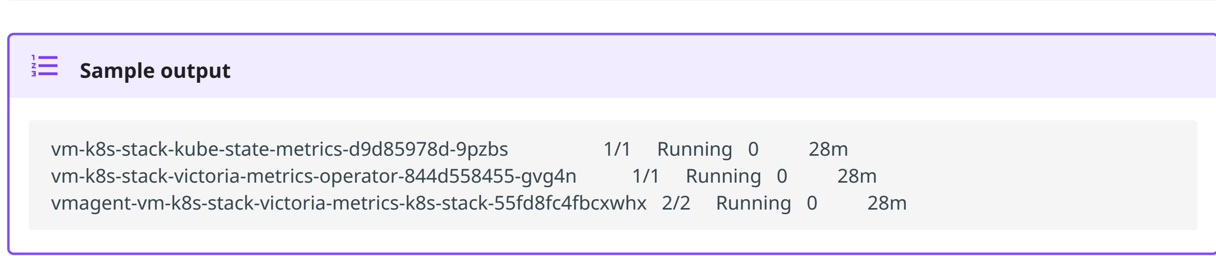
```
$ kubectl create namespace monitoring-system
```

3. Create the YAML file for the Kubernetes Secrets and specify the base64-encoded API key value within. Let's name this file `pmm-api-vmoperator.yaml`.

pmm-api-vmoperator.yaml

6.7.3 Validate the successful installation

```
$ kubectl get pods -n <namespace>
```



Sample output

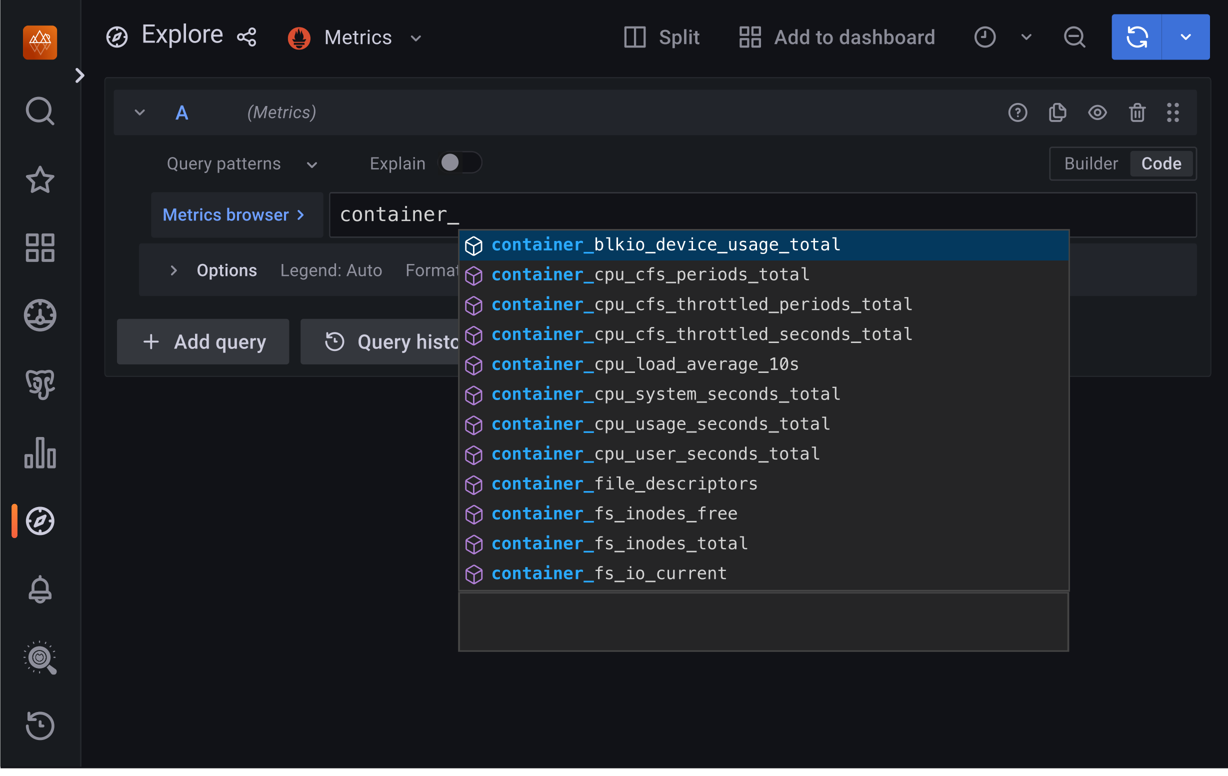
```
vm-k8s-stack-kube-state-metrics-d9d85978d-9pzbs           1/1    Running  0       28m
vm-k8s-stack-victoria-metrics-operator-844d558455-gvg4n        1/1    Running  0       28m
vmagent-vm-k8s-stack-victoria-metrics-k8s-stack-55fd8fc4fbcxwhx  2/2    Running  0       28m
```

What Pods are running depends on the configuration chosen in values used while installing `victoria-metrics-k8s-stack` chart.

6.7.4 Verify metrics capture

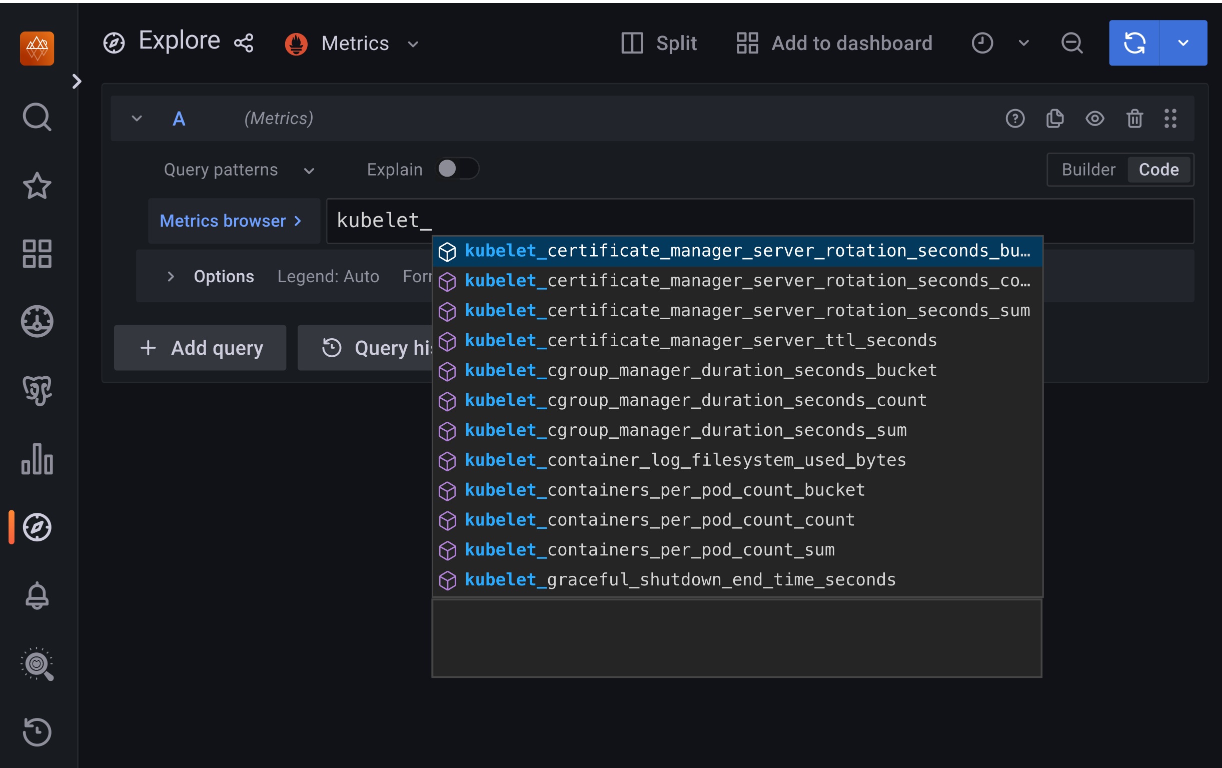
1. Connect to the PMM server.
2. Click **Explore** and switch to the **Code** mode.
3. Check that the required metrics are captured, type the following in the Metrics browser dropdown:

- cadvisor:

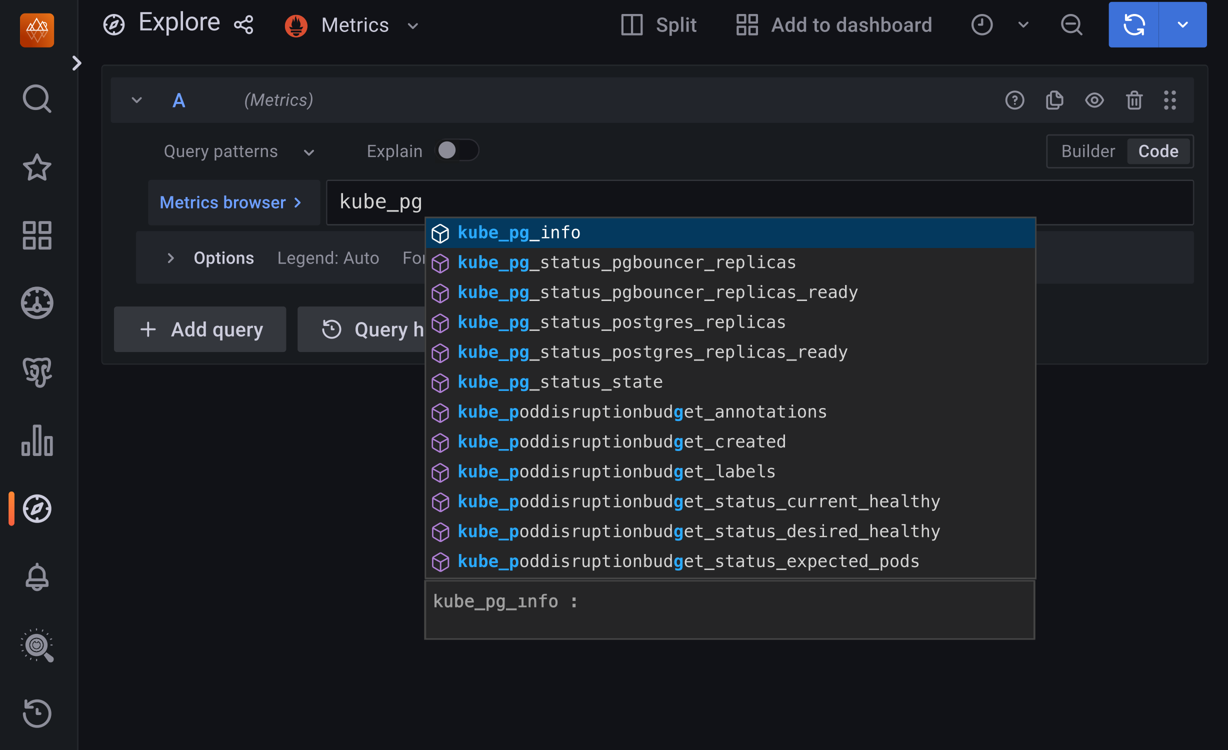


- kubelet:

Percona LLC and/or its affiliates, © 2009 - 2024



- kube-state-metrics metrics that also include Custom resource metrics for the Operator and database deployed in your Kubernetes cluster:



Percona LLC and/or its affiliates, © 2009 - 2024

6.7.5 Uninstall Victoria metrics Kubernetes stack

To remove Victoria metrics Kubernetes stack used for Kubernetes cluster monitoring, use the cleanup script. By default, the script removes all the Custom Resource Definitions(CRD) and Secrets associated with the Victoria metrics Kubernetes stack. To keep the CRDs, run the script with the `--keep-crd` flag.

Remove CRDs   Keep CRDs

Replace the `<NAMESPACE>` placeholder with the namespace you specified during the Victoria metrics Kubernetes stack installation:

```
$ bash <(curl -fsL https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/main/vm-operator-k8s-stack/
cleanup.sh) --namespace <NAMESPACE>
```

Replace the `<NAMESPACE>` placeholder with the namespace you specified during the Victoria metrics Kubernetes stack installation:

```
$ bash <(curl -fsL https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/main/vm-operator-k8s-stack/
cleanup.sh) --namespace <NAMESPACE> --keep-crd
```

Check that the Victoria metrics Kubernetes stack is deleted:

```
$ helm list -n <namespace>
```

The output should provide the empty list.

If you face any issues with the removal, uninstall the stack manually:

```
$ helm uninstall vm-k8s-stack -n < namespace>
```

6.7.6 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum   Get a Percona Expert   Join K8S Squad

Last update: 2024-01-12

Percona LLC and/or its affiliates, © 2009 - 2024

6.8 Use PostGIS extension with Percona Distribution for PostgreSQL

PostGIS is a PostgreSQL extension that adds GIS capabilities to this database.

Starting from the Operator version 2.3.0 it became possible to deploy and manage PostGIS-enabled PostgreSQL.

Due to the large size and domain specifics of this extension, Percona provides separate PostgreSQL Distribution images with it.

Percona LLC and/or its affiliates, © 2009 - 2024

6.8.1 Deploy the Operator with PostGIS-enabled database cluster

Following steps will allow you to deploy PostgreSQL cluster with these images.

Percona LLC and/or its affiliates, © 2009 - 2024

1. Clone the percona-postgresql-operator repository:

```
$ git clone -b v2.3.1 https://github.com/percona/percona-postgresql-operator
$ cd percona-postgresql-operator
```

Note

It is crucial to specify the right branch with `-b` option while cloning the code on this step. Please be careful.

2. The Custom Resource Definition for Percona Distribution for PostgreSQL should be created from the `deploy/crd.yaml` file. Custom Resource Definition extends the standard set of resources which Kubernetes "knows" about with the new items (in our case ones which are the core of the Operator). Apply it as follows:

```
$ kubectl apply --server-side -f deploy/crd.yaml
```

3. Create the Kubernetes namespace for your cluster if needed (for example, let's name it `postgres-operator`):

```
$ kubectl create namespace postgres-operator
```

4. The role-based access control (RBAC) for Percona Distribution for PostgreSQL is configured with the `deploy/rbac.yaml` file. Role-based access is based on defined roles and the available actions which correspond to each role. The role and actions are defined for Kubernetes resources in the yaml file. Further details about users and roles can be found in Kubernetes documentation.

```
$ kubectl apply -f deploy/rbac.yaml -n postgres-operator
```

Note

Setting RBAC requires your user to have cluster-admin role privileges. For example, those using Google Kubernetes Engine can grant user needed privileges with the following command:

```
$ kubectl create clusterrolebinding cluster-admin-binding --clusterrole=cluster-admin --user=$(gcloud config get-value core/
account)
```

5. Start the Operator within Kubernetes:

```
$ kubectl apply -f deploy/operator.yaml -n postgres-operator
```

6. After the Operator is started, modify the `deploy/cr.yaml` configuration file with PostGIS-enabled image - use `percona/percona-postgresql-operator:2.3.1-ppg16-postgres-gis` instead of `percona/percona-postgresql-operator:2.3.1-ppg16-postgres`

```
apiVersion: pgv2.percona.com/v2
kind: PerconaPGCluster
metadata:
  name: cluster1
spec:
  ...
  image: percona/percona-postgresql-operator:2.3.1-ppg16-postgres-gis
  ...
```

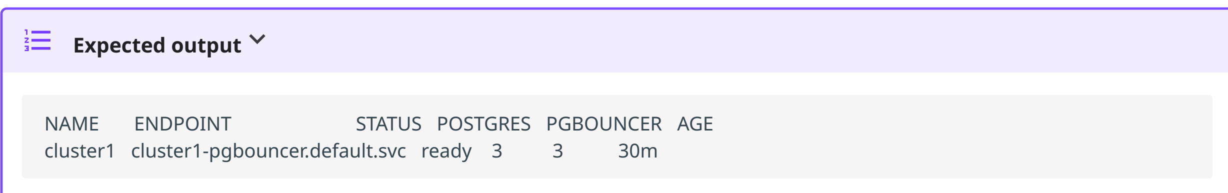
When done, Percona Distribution for PostgreSQL cluster can be created at any time with the following command:

Percona LLC and/or its affiliates, © 2009 - 2024

```
$ kubectl apply -f deploy/cr.yaml -n postgres-operator
```

The creation process may take some time. When the process is over your cluster will obtain the `ready` status. You can check it with the following command:

```
$ kubectl get pg -n postgres-operator
```



Expected output

```
NAME      ENDPOINT                      STATUS   POSTGRES   PGBOUNCER   AGE
cluster1  cluster1-pgbouncer.default.svc  ready  3          3           30m
```

6.8.2 Check PostGIS extension

To use PostGIS extension you should enable it for a specific database.

For example, you can create the new database named `mygisdata` with the `psql` tool as follows:

```
CREATE database mygisdata;
\c mygisdata;
CREATE SCHEMA gis;
```

Next, enable the `postgis` extension. Make sure you are connected to the database you created earlier and run the following command:

```
CREATE EXTENSION postgis;
```

Finally, check that the extension is enabled:

```
SELECT postgis_full_version();
```

The output should resemble the following:

```
postgis_full_version
    -----------------------------------------------------------------------------------------------------------------------------------------------------
 POSTGIS="3.3.3" [EXTENSION] PGSQL="140" GEOS="3.10.2-CAPI-1.16.0" PROJ="8.2.1" LIBXML="2.9.13"
LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU="0.5.0 (Internal)"
```

You can find more about using PostGIS in the official Percona Distribution for PostgreSQL documentation, as well as in this blogpost.

6.8.3 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Percona LLC and/or its affiliates, © 2009 - 2024

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

7. Troubleshooting

7.1 Initial troubleshooting

Percona Operator for PostgreSQL uses Custom Resources to manage options for the various components of the cluster.

- `PerconaPGCluster` Custom Resource with Percona PostgreSQL Cluster options (it has handy `pg` shortname also),
- `PerconaPGBackup` and `PerconaPGRestore` Custom Resources contain options for Percona XtraBackup used to backup Percona XtraDB Cluster and to restore it from backups (`pg-backup` and `pg-restore` shortnames are available for them).

The first thing you can check for the Custom Resource is to query it with `kubectl get` command:

```
$ kubectl get pg
```

Expected output

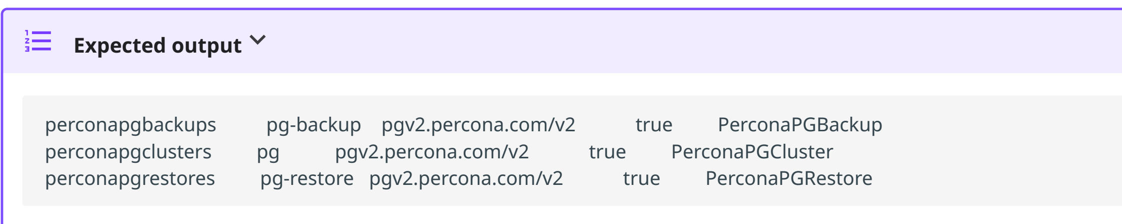
```
NAME      ENDPOINT                     STATUS   POSTGRES   PGBOUNCER   AGE
cluster1  cluster1-pgbouncer.default.svc  ready   3          3           30m
```

The Custom Resource should have `Ready` status.

Note

You can check which Percona's Custom Resources are present and get some information about them as follows:

```
$ kubectl api-resources | grep -i percona
```



Expected output

```
perconapgbackups      pg-backup   pgv2.percona.com/v2       true       PerconaPGBackup
perconapgclusters     pg          pgv2.percona.com/v2       true       PerconaPGCluster
perconapgrestores     pg-restore  pgv2.percona.com/v2       true       PerconaPGRestore
```

7.1.1 Check the Pods

If Custom Resource is not getting `Ready` status, it makes sense to check individual Pods. You can do it as follows:

```
$ kubectl get pods
```


Percona LLC and/or its affiliates, © 2009 - 2024

Expected output

```
NAME                                            READY   STATUS      RESTARTS   AGE
cluster1-backup-4vwt-p5d9j                      0/1     Completed   0          97m
cluster1-instance1-b5mr-0                       4/4     Running     0          99m
cluster1-instance1-b8p7-0                       4/4     Running     0          99m
cluster1-instance1-w7q2-0                       4/4     Running     0          99m
cluster1-pgbouncer-79bbf55c45-62xlk             2/2     Running     0          99m
cluster1-pgbouncer-79bbf55c45-9g4cb             2/2     Running     0          99m
cluster1-pgbouncer-79bbf55c45-9nrmd             2/2     Running     0          99m
cluster1-repo-host-0                            2/2     Running     0          99m
percona-postgresql-operator-79cd8586f5-2qzcs    1/1     Running     0          120m
```

The above command provides the following insights:

- `READY` indicates how many containers in the Pod are ready to serve the traffic. In the above example, `cluster1-repo-host-0` container has all two containers ready (2/2). For an application to work properly, all containers of the Pod should be ready.
- `STATUS` indicates the current status of the Pod. The Pod should be in a `Running` state to confirm that the application is working as expected. You can find out other possible states in the official Kubernetes documentation.
- `RESTARTS` indicates how many times containers of Pod were restarted. This is impacted by the Container Restart Policy. In an ideal world, the restart count would be zero, meaning no issues from the beginning. If the restart count exceeds zero, it may be reasonable to check why it happens.
- `AGE` : Indicates how long the Pod is running. Any abnormality in this value needs to be checked.

You can find more details about a specific Pod using the `kubectl describe pods <pod-name>` command.

```
$ $ kubectl describe pods cluster1-instance1-b5mr-0
```

Expected output

```
...
Name:         cluster1-instance1-b5mr-0
Namespace:    default
...
Controlled By:  StatefulSet/cluster1-instance1-b5mr
Init Containers:
 postgres-startup:
...
Containers:
 database:
...
 pgbackrest:
...
   Restart Count:  0
   Liveness:   http-get https://:8008/liveness delay=3s timeout=5s period=10s #success=1 #failure=3
   Readiness:  http-get https://:8008/readiness delay=3s timeout=5s period=10s #success=1 #failure=3
   Environment:
...
   Mounts:
...
Volumes:
...
Events:
...
```


Percona LLC and/or its affiliates, © 2009 - 2024

This gives a lot of information about containers, resources, container status and also events. So, describe output should be checked to see any abnormalities.

7.1.2 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-11-06

Percona LLC and/or its affiliates, © 2009 - 2024

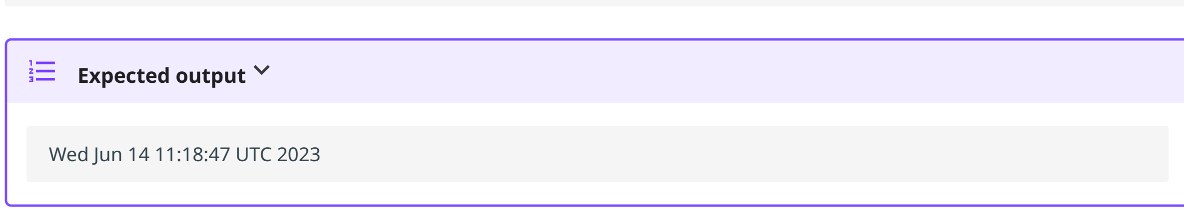
7.2 Exec into the containers

If you want to examine the contents of a container "in place" using remote access to it, you can use the `kubectl exec` command. It allows you to run any command or just open an interactive shell session in the container. Of course, you can have shell access to the container only if container supports it and has a "Running" state.

In the following examples we will access the container `database` of the `cluster1-instance1-b5mr-0` Pod.

- Run `date` command:

```
$ kubectl exec -ti cluster1-instance1-b5mr-0 -c database -- date
```



You will see an error if the command is not present in a container. For example, trying to run the `time` command, which is not present in the container, by executing `kubectl exec -ti cluster1-instance1-b5mr-0 -c database -- time` would show the following result:

```
OCI runtime exec failed: exec failed: unable to start container process: exec: "time": executable file not found in
$PATH: unknown command terminated with exit code 126
```

- Print log files to a terminal:

```
$ kubectl exec -ti cluster1-instance1-b5mr-0 -c database -- cat /pgdata/pg16/log/postgresql-*.log
```

- Similarly, opening an Interactive terminal, executing a pair of commands in the container, and exiting it may look as follows:

```
$ kubectl exec -ti cluster1-instance1-b5mr-0 -c database -- bash
bash-4.4$ hostname
cluster1-pxc-0
bash-4.4$ ls /pgdata/pg16/log/
postgresql-Wed.log
bash-4.4$ exit
exit
$
```

7.2.1 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

7.3 Check the logs

Logs provide valuable information. It makes sense to check the logs of the database Pods and the Operator Pod. Following flags are helpful for checking the logs with the `kubectl logs` command:

Flag	Description
`-c`, `--container=<container-name>`	Print log of a specific container in case of multiple containers in a Pod
`-f`, `--follow`	Follows the logs for a live output
`--since=<time>`	Print logs newer than the specified time, for example: `--since="10s"`
`--timestamps`	Print timestamp in the logs (timezone is taken from the container)
`-p`, `--previous`	Print previous instantiation of a container. This is extremely useful in case of container restart, where there is a need to check the logs on why the container restarted. Logs of previous instantiation might not be available in all the cases.

In the following examples we will access containers of the `cluster1-instance1-b5mr-0` Pod.

- Check logs of the `database` container:

```
$ kubectl logs cluster1-instance1-b5mr-0 --container database
```

- Check logs of the `pgbackrest` container:

```
$ kubectl logs cluster1-instance1-b5mr-0 --container pgbackrest
```

- Filter logs of the `database` container which are not older than 600 seconds:

```
$ kubectl logs cluster1-instance1-b5mr-0 --container database --since=600s
```

- Check logs of a previous instantiation of the `database` container, if any:

```
$ kubectl logs cluster1-instance1-b5mr-0 --container database --previous
```

7.3.1 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-08

Percona LLC and/or its affiliates, © 2009 - 2024

8. Reference

8.1 Custom Resource options

The Cluster is configured via the deploy/cr.yaml file.

The metadata part of this file contains the following keys:

- `name` (`cluster1` by default) sets the name of your Percona Distribution for PostgreSQL Cluster; it should include only URL-compatible characters, not exceed 22 characters, start with an alphabetic character, and end with an alphanumeric character;
- `finalizers.percona.com/delete-ssl` if present, activates the Finalizer which deletes objects, created for SSL (Secret, certificate, and issuer) after the cluster deletion event (off by default).
- `finalizers.percona.com/delete-pvc` if present, activates the Finalizer which deletes Persistent Volume Claims for the database cluster Pods after the deletion event (off by default).

Percona LLC and/or its affiliates, © 2009 - 2024

The spec part of the deploy/cr.yaml file contains the following:

Percona LLC and/or its affiliates, © 2009 - 2024

Key	crVersion
Value	string
Example	`2.3.1`
Description	Version of the Operator the Custom Resource belongs to

Key	standby.enabled
Value	boolean
Example	`false`
Description	Enables or disables running the cluster in a standby mode (read-only copy of an existing cluster, useful for disaster recovery, etc)

Key	standby.host
Value	string
Example	`"<primary-ip>"`
Description	Host address of the primary cluster this standby cluster connects to

Key	standby.port
Value	string
Example	`"<primary-port>"`
Description	Port number used by a standby copy to connect to the primary cluster
Key	openshift
Value	boolean
Example	`true`
Description	Set to `true` if the cluster is being deployed on OpenShift, set to `false` otherwise, or unset it for autodetection

Key	standby.repoName
Value	string
Example	`repo1`
Description	Name of the pgBackRest repository in the primary cluster this standby cluster connects to

Key	secrets.customTLSSecret.name
Value	string
Example	`cluster1-cert`
Description	A secret with TLS certificate generated for *external* communications, see Transport Layer Security (TLS) for details

Key	secrets.customReplicationTLSSecret.name
Value	string
Example	`replication1-cert`

Percona LLC and/or its affiliates, © 2009 - 2024

Description	A secret with TLS certificate generated for *internal* communications, see Transport Layer Security (TLS) for details

Key	users.name
Value	string
Example	`rhino`
Description	The name of the PostgreSQL user

Key	users.databases
Value	string
Example	`zoo`
Description	Databases accessible by a specific PostgreSQL user with rights to create objects in them (the option is ignored for `postgres` user; also, modifying it can't be used to revoke the already given access)

Key	users.password.type
Value	string
Example	`ASCII`
Description	The set of characters used for password generation: can be either `ASCII` (default) or `AlphaNumeric`

Key	users.options
Value	string
Example	`"SUPERUSER"`
Description	The `ALTER ROLE` options other than password (the option is ignored for `postgres` user)

Key	users.secretName
Value	string
Example	`"rhino-credentials"`
Description	The custom name of the user's Secret; if not specified, the default `<clusterName>-pguser-<userName>` variant will be used

Key	databaseInitSQL.key
Value	string
Example	`init.sql`
Description	Data key for the Custom configuration options ConfigMap with the init SQL file, which will be executed at cluster creation time

Key	databaseInitSQL.name
Value	string
Example	`cluster1-init-sql`
Description	Name of the ConfigMap with the init SQL file, which will be executed at cluster creation time

Percona LLC and/or its affiliates, © 2009 - 2024

Key	pause
Value	string
Example	`false`
Description	Setting it to `true` gracefully stops the cluster, scaling workloads to zero and suspending CronJobs; setting it to `false` after shut down starts the cluster back

Key	unmanaged
Value	string
Example	`false`
Description	Setting it to `true` stops the Operator's activity including the rollout and reconciliation of changes made in the Custom Resource; setting it to `false` starts the Operator's activity back

Key	dataSource.postgresCluster.clusterName
Value	string
Example	`cluster1`
Description	Name of an existing cluster to use as the data source when restoring backup to a new cluster

Key	dataSource.postgresCluster.repoName
Value	string
Example	`repo1`
Description	Name of the pgBackRest repository in the source cluster that contains the backup to be restored to a new cluster

Key	dataSource.postgresCluster.options
Value	string
Example	
Description	The pgBackRest command-line options for the pgBackRest restore command

Key	dataSource.pgbackrest.stanza
Value	string
Example	`db`
Description	Name of the pgBackRest stanza to use as the data source when restoring backup to a new cluster

Key	dataSource.pgbackrest.configuration.secret.name
Value	string
Example	`pgo-s3-creds`
Description	Name of the Kubernetes Secret object with custom pgBackRest configuration, which will be added to the pgBackRest configuration generated by the Operator

Key	dataSource.pgbackrest.global
Value	subdoc

Percona LLC and/or its affiliates, © 2009 - 2024

Example	`/pgbackrest/postgres-operator/hippo/repo1`
Description	Settings, which are to be included in the `global` section of the pgBackRest configuration generated by the Operator

Key	dataSource.pgbackrest.repo.name
Value	string
Example	`repo1`
Description	Name of the pgBackRest repository

Key	dataSource.pgbackrest.repo.s3.bucket
Value	string
Example	`"my-bucket"`
Description	The Amazon S3 bucket or Google Cloud Storage bucket
name used for backups	

Key	dataSource.pgbackrest.repo.s3.endpoint
Value	string
Example	`"s3.ca-central-1.amazonaws.com"`
Description	The endpoint URL of the S3-compatible storage to be used for backups (not needed for the original Amazon S3 cloud)

Key	dataSource.pgbackrest.repo.s3.region
Value	boolean
Example	`"ca-central-1"`
Description	The AWS region to use for Amazon and all S3-compatible storages

Key	image
Value	string
Example	`perconalab/percona-postgresql-operator:2.3.1-ppg16-postgres`
Description	The PostgreSQL Docker image to use

Key	imagePullPolicy
Value	string
Example	`Always`
Description	This option is used to set the policy for updating PostgreSQL images

Key	postgresVersion
Value	int
Example	14
Description	The major version of PostgreSQL to use

Percona LLC and/or its affiliates, © 2009 - 2024

Key	port
Value	int
Example	5432
Description	The port number for PostgreSQL

Key	expose.annotations
Value	label
Example	`my-annotation: value1`
Description	The Kubernetes annotations metadata for PostgreSQL

Key	expose.labels
Value	label
Example	`my-label: value2`
Description	Set labels for the PostgreSQL Service

Key	expose.type
Value	string
Example	`LoadBalancer`
Description	Specifies the type of Kubernetes Service for PostgreSQL

Key	expose.loadBalancerSourceRanges
Value	string
Example	`"10.0.0.0/8"`
Description	The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations)

Percona LLC and/or its affiliates, © 2009 - 2024

8.1.1 Instances section

The `instances` section in the deploy/cr.yaml file contains configuration options for PostgreSQL instances.

Percona LLC and/or its affiliates, © 2009 - 2024

Key	instances.metadata.labels
Value	label
Example	`pg-cluster-label: cluster1`
Description	Set labels for PostgreSQL Pods

Key	instances.name
Value	string
Example	`rs 0`
Description	The name of the PostgreSQL instance

Key	instances.replicas
Value	int
Example	3
Description	The number of Replicas to create for the PostgreSQL instance

Key	instances.resources.limits.cpu
Value	string
Example	`2.0`
Description	Kubernetes CPU limits for a PostgreSQL instance

Key	instances.resources.limits.memory
Value	string
Example	`4Gi`
Description	The Kubernetes memory limits for a PostgreSQL instance

Key	instances.topologySpreadConstraints.maxSkew
Value	int
Example	1
Description	The degree to which Pods may be unevenly distributed under the Kubernetes Pod Topology Spread Constraints

Key	instances.topologySpreadConstraints.topologyKey
Value	string
Example	`my-node-label`
Description	The key of node labels for the Kubernetes Pod Topology Spread Constraints

Key	instances.topologySpreadConstraints.whenUnsatisfiable
Value	string
Example	`DoNotSchedule`
Description	What to do with a Pod if it doesn’t satisfy the Kubernetes Pod Topology Spread Constraints

Percona LLC and/or its affiliates, © 2009 - 2024

Key	instances.topologySpreadConstraints.labelSelector.matchLabels
Value	label
Example	`postgres-operator.crunchydata.com/instance-set: instance1`
Description	The Label selector for the Kubernetes Pod Topology Spread Constraints

Key	instances.tolerations.effect
Value	string
Example	`NoSchedule`
Description	The Kubernetes Pod tolerations effect for the PostgreSQL instance

Key	instances.tolerations.key
Value	string
Example	`role`
Description	The Kubernetes Pod tolerations key for the PostgreSQL instance

Key	instances.tolerations.operator
Value	string
Example	`Equal`
Description	The Kubernetes Pod tolerations operator for the PostgreSQL instance

Key	instances.tolerations.value
Value	string
Example	`connection-poolers`
Description	The Kubernetes Pod tolerations value for the PostgreSQL instance

Key	instances.priorityClassName
Value	string
Example	`high-priority`
Description	The Kuberentes Pod priority class for PostgreSQL instance Pods

Key	instances.walVolumeClaimSpec.accessModes
Value	string
Example	`ReadWriteOnce`
Description	The Kubernetes PersistentVolumeClaim access modes for the PostgreSQL Write-ahead Log storage

Key	instances.walVolumeClaimSpec.resources.requests.storage
Value	string
Example	`1Gi`
Description	The Kubernetes storage requests for the storage the PostgreSQL instance will use

Percona LLC and/or its affiliates, © 2009 - 2024

Key	instances.dataVolumeClaimSpec.accessModes
Value	string
Example	`ReadWriteOnce`
Description	The Kubernetes PersistentVolumeClaim access modes for the PostgreSQL Write-ahead Log storage

Key	instances.dataVolumeClaimSpec.resources.requests.storage
Value	string
Example	`1Gi`
Description	The Kubernetes storage requests for the storage the PostgreSQL instance will use

Percona LLC and/or its affiliates, © 2009 - 2024

instances.sidecars subsection

The `instances.sidecars` subsection in the deploy/cr.yaml file contains configuration options for custom sidecar containers which can be added to PostgreSQL Pods.

Key	instances.sidecars.image
Value	string
Example	`mycontainer1:latest`
Description	Image for the custom sidecar container for PostgreSQL Pods

Key	instances.sidecars.name
Value	string
Example	`testcontainer`
Description	Name of the custom sidecar container for PostgreSQL Pods

Key	instances.sidecars.imagePullPolicy
Value	string
Example	`Always`
Description	This option is used to set the policy for the PostgreSQL Pod sidecar container

Key	instances.sidecars.env
Value	subdoc
Example	
Description	The environment variables set as key-value pairs for the custom sidecar container for PostgreSQL Pods

Key	instances.sidecars.envFrom
Value	subdoc
Example	
Description	The environment variables set as key-value pairs in ConfigMaps for the custom sidecar container for PostgreSQL Pods

Key	instances.sidecars.command
Value	array
Example	`["/bin/sh"]`
Description	Command for the custom sidecar container for PostgreSQL Pods

Key	instances.sidecars.args
Value	array
Example	`["-c", "while true; do trap 'exit 0' SIGINT SIGTERM SIGQUIT SIGKILL; done;"]`
Description	Command arguments for the custom sidecar container for PostgreSQL Pods

Percona LLC and/or its affiliates, © 2009 - 2024

8.1.2 Backup section

The `backup` section in the deploy/cr.yaml file contains the following configuration options for the regular Percona Distribution for PostgreSQL backups.

Percona LLC and/or its affiliates, © 2009 - 2024

Key	backups.pgbackrest.metadata.labels
Value	label
Example	pg-cluster-label: cluster1
Description	Set labels for pgBackRest Pods

Key	backups.pgbackrest.image
Value	string
Example	perconalab/percona-postgresql-operator:2.3.1-ppg16-pgbackrest
Description	The Docker image for pgBackRest

Key	backups.pgbackrest.configuration.secret.name
Value	string
Example	cluster1-pgbackrest-secrets
Description	Name of the Kubernetes Secret object with custom pgBackRest configuration, which will be added to the pgBackRest configuration generated by the Operator

Key	backups.pgbackrest.jobs.priorityClassName
Value	string
Example	high-priority
Description	The Kuberentes Pod priority class for pgBackRest jobs

Key	backups.pgbackrest.jobs.resources.limits.cpu
Value	int
Example	200
Description	Kubernetes CPU limits for a pgBackRest job

Key	backups.pgbackrest.jobs.resources.limits.memory
Value	int
Example	128Mi
Description	The Kubernetes memory limits for a pgBackRest job

Key	backups.pgbackrest.jobs.tolerations.effect
Value	string
Example	NoSchedule
Description	The Kubernetes Pod tolerations effect for a pgBackRest job

Key	backups.pgbackrest.jobs.tolerations.key
Value	string
Example	role
Description	The Kubernetes Pod tolerations key for a pgBackRest job

Percona LLC and/or its affiliates, © 2009 - 2024

Key	backups.pgbackrest.jobs.tolerations.operator
Value	string
Example	`Equal`
Description	The Kubernetes Pod tolerations operator for a pgBackRest job

Key	backups.pgbackrest.jobs.tolerations.value
Value	string
Example	`connection-poolers`
Description	The Kubernetes Pod tolerations value for a pgBackRest job

Key	backups.pgbackrest.global
Value	subdoc
Example	`/pgbackrest/postgres-operator/hippo/repo1`
Description	Settings, which are to be included in the `global` section of the pgBackRest configuration generated by the Operator

Key	backups.pgbackrest.repoHost.priorityClassName
Value	string
Example	`high-priority`
Description	The Kuberentes Pod priority class for pgBackRest repo

Key	backups.pgbackrest.repoHost.topologySpreadConstraints.maxSkew
Value	int
Example	1
Description	The degree to which Pods may be unevenly distributed under the Kubernetes Pod Topology Spread Constraints

Key	backups.pgbackrest.repoHost.topologySpreadConstraints.topologyKey
Value	string
Example	`my-node-label`
Description	The key of node labels for the Kubernetes Pod Topology Spread Constraints

Key	backups.pgbackrest.repoHost.topologySpreadConstraints.whenUnsatisfiable
Value	string
Example	`ScheduleAnyway`
Description	What to do with a Pod if it doesn't satisfy the Kubernetes Pod Topology Spread Constraints

Key	backups.pgbackrest.repoHost.topologySpreadConstraints.labelSelector.matchLabels
Value	label
Example	`postgres-operator.crunchydata.com/pgbackrest: ""`
Description	The Label selector for the Kubernetes Pod Topology Spread Constraints

Percona LLC and/or its affiliates, © 2009 - 2024

Key	backups.pgbackrest.repoHost.affinity.podAntiAffinity
Value	subdoc
Example	
Description	Pod anti-affinity, allows setting the standard Kubernetes affinity constraints of any complexity

Key	backups.pgbackrest.manual.repoName
Value	string
Example	`repo1`
Description	Name of the pgBackRest repository for on-demand backups

Key	backups.pgbackrest.manual.options
Value	string
Example	`--type=full`
Description	The on-demand backup command-line options which will be passed to pgBackRest for on-demand backups

Key	backups.pgbackrest.repos.name
Value	string
Example	`repo1`
Description	Name of the pgBackRest repository for backups

Key	backups.pgbackrest.repos.schedules.full
Value	string
Example	`0 0 * * 6`
Description	Scheduled time to make a full backup specified in the crontab format

Key	backups.pgbackrest.repos.schedules.differential
Value	string
Example	`0 0 * * 6`
Description	Scheduled time to make a differential backup specified in the crontab format

Key	backups.pgbackrest.repos.volume.volumeClaimSpec.accessModes
Value	string
Example	`ReadWriteOnce`
Description	The Kubernetes PersistentVolumeClaim access modes for the pgBackRest Storage

Key	backups.pgbackrest.repos.volume.volumeClaimSpec.resources.requests.storage
Value	string
Example	`1Gi`
Description	The Kubernetes storage requests for the pgBackRest storage

Percona LLC and/or its affiliates, © 2009 - 2024

Key	backups.pgbackrest.repos.s3.bucket
Value	string
Example	`"my-bucket"`
Description	The Amazon S3 bucket
name used for backups	

Key	backups.pgbackrest.repos.s3.endpoint
Value	string
Example	`"s3.ca-central-1.amazonaws.com"`
Description	The endpoint URL of the S3-compatible storage to be used for backups (not needed for the original Amazon S3 cloud)

Key	backups.pgbackrest.repos.s3.region
Value	boolean
Example	`"ca-central-1"`
Description	The AWS region to use for Amazon and all S3-compatible storages

Key	backups.pgbackrest.repos.gcs.bucket
Value	string
Example	`"my-bucket"`
Description	The Google Cloud Storage bucket
name used for backups	

Key	backups.pgbackrest.repos.azure.container
Value	string
Example	`my-container`
Description	Name of the Azure Blob Storage container for backups

Percona LLC and/or its affiliates, © 2009 - 2024

8.1.3 PMM section

The `pmm` section in the deploy/cr.yaml file contains configuration options for Percona Monitoring and Management.

Key	pmm.enabled
Value	boolean
Example	`false`
Description	Enables or disables monitoring Percona Distribution for PostgreSQL cluster with PMM

Key	pmm.image
Value	string
Example	`percona/pmm-client:2.41.0`
Description	Percona Monitoring and Management (PMM) Client Docker image

Key	pmm.imagePullPolicy
Value	string
Example	`IfNotPresent`
Description	This option is used to set the policy for updating PMM Client images

Key	pmm.pmmSecret
Value	string
Example	`cluster1-pmm-secret`
Description	Name of the Kubernetes Secret object for the PMM Server password

Key	pmm.serverHost
Value	string
Example	`monitoring-service`
Description	Address of the PMM Server to collect data from the cluster

Percona LLC and/or its affiliates, © 2009 - 2024

8.1.4 Proxy section

The `proxy` section in the deploy/cr.yaml file contains configuration options for the pgBouncer connection pooler for PostgreSQL.

Percona LLC and/or its affiliates, © 2009 - 2024

Key	proxy.pgBouncer.metadata.labels
Value	label
Example	pg-cluster-label: cluster1
Description	Set labels for pgBouncer Pods

Key	proxy.pgBouncer.replicas
Value	int
Example	3
Description	The number of the pgBouncer Pods to provide connection pooling

Key	proxy.pgBouncer.image
Value	string
Example	perconalab/percona-postgresql-operator:2.3.1-ppg16-pgbouncer
Description	Docker image for the pgBouncer connection pooler

Key	proxy.pgBouncer.exposeSuperusers
Value	boolean
Example	false
Description	Enables or disables exposing superuser user through pgBouncer

Key	proxy.pgBouncer.resources.limits.cpu
Value	int
Example	200m
Description	Kubernetes CPU limits for a pgBouncer container

Key	proxy.pgBouncer.resources.limits.memory
Value	int
Example	128Mi
Description	The Kubernetes memory limits for a pgBouncer container

Key	proxy.pgBouncer.expose.type
Value	string
Example	ClusterIP
Description	Specifies the type of Kubernetes Service for pgBouncer

Key	proxy.pgBouncer.expose.annotations
Value	label
Example	pg-cluster-annot: cluster1
Description	The Kubernetes annotations metadata for pgBouncer

Percona LLC and/or its affiliates, © 2009 - 2024

Key	proxy.pgBouncer.expose.labels
Value	label
Example	`pg-cluster-label: cluster1`
Description	Set labels for the pgBouncer Service

Key	proxy.pgBouncer.expose.loadBalancerSourceRanges
Value	string
Example	`"10.0.0.0/8"`
Description	The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations)

Value	string
Example	`preferred`
Description	Pod anti-affinity type, can be either `preferred` or `required`

Key	proxy.pgBouncer.config
Value	subdoc
Example	global: pool_mode: transaction
Description	Custom configuration options for pgBouncer. Please note that configuration changes are automatically applied to the running instances without validation, so having an invalid config can make the cluster unavailable

Percona LLC and/or its affiliates, © 2009 - 2024

proxy.pgBouncer.sidecars subsection

The `proxy.pgBouncer.sidecars` subsection in the deploy/cr.yaml file contains configuration options for custom sidecar containers which can be added to pgBouncer Pods.

Key	proxy.pgBouncer.sidecars.image
Value	string
Example	`mycontainer1:latest`
Description	Image for the custom sidecar container for pgBouncer Pods

Key	proxy.pgBouncer.sidecars.name
Value	string
Example	`testcontainer`
Description	Name of the custom sidecar container for pgBouncer Pods

Key	proxy.pgBouncer.sidecars.imagePullPolicy
Value	string
Example	`Always`
Description	This option is used to set the policy for the pgBouncer Pod sidecar container

Key	proxy.pgBouncer.sidecars.env
Value	subdoc
Example	
Description	The environment variables set as key-value pairs for the custom sidecar container for pgBouncer Pods

Key	proxy.pgBouncer.sidecars.envFrom
Value	subdoc
Example	
Description	The environment variables set as key-value pairs in ConfigMaps for the custom sidecar container for pgBouncer Pods

Key	proxy.pgBouncer.sidecars.command
Value	array
Example	`["/bin/sh"]`
Description	Command for the custom sidecar container for pgBouncer Pods

Key	proxy.pgBouncer.sidecars.args
Value	array
Example	`["-c", "while true; do trap 'exit 0' SIGINT SIGTERM SIGQUIT SIGKILL; done;"]`
Description	Command arguments for the custom sidecar container for pgBouncer Pods

Percona LLC and/or its affiliates, © 2009 - 2024

8.1.5 Patroni Section

The `patroni` section in the deploy/cr.yaml file contains configuration options to customize the PostgreSQL high-availability implementation based on Patroni.

Key	patroni.dynamicConfiguration
Value	subdoc

Example	postgresql:
	  parameters:
	    max_parallel_workers: 2
	    max_worker_processes: 2
	    shared_buffers: 1GB
	    work_mem: 2MB

Description	Custom PostgreSQL configuration options. Please note that configuration changes are automatically applied to the running instances without validation, so having an invalid config can make the cluster unavailable

Percona LLC and/or its affiliates, © 2009 - 2024

8.1.6 Custom extensions Section

The `extensions` section in the deploy/cr.yaml file contains configuration options to manage PostgreSQL extensions.

Percona LLC and/or its affiliates, © 2009 - 2024

Key	extensions.image
Value	string
Example	`percona/percona-postgresql-operator:2.3.1`
Description	Image for the custom PostgreSQL extension loader sidecar container

Key	extensions.imagePullPolicy
Value	string
Example	`Always`
Description	Policy for the custom extension sidecar container

Key	extensions.storage.type
Value	string
Example	`s3`
Description	The cloud storage type used for backups. Only `s3` type is currently supported

Key	extensions.storage.bucket
Value	string
Example	`pg-extensions`
Description	The Amazon S3 bucket name for prepackaged PostgreSQL custom extensions

Key	extensions.storage.region
Value	string
Example	`eu-central-1`
Description	The AWS region to use

Key	extensions.storage.secret.name
Value	string
Example	`cluster1-extensions-secret`
Description	The Kubernetes secret for the custom extensions storage. It should contain `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` keys.

Key	extensions.builtin
Value	label
Example	`pg_stat_monitor: true`
Description	The key-value pairs which enable or disable Percona Distribution for PostgreSQL builtin extensions

Key	extensions.custom.name
Value	string
Example	`pg_cron`
Description	Name of the PostgreSQL custom extension

Percona LLC and/or its affiliates, © 2009 - 2024

Key	extensions.custom.version
Value	string
Example	1.6.1
Description	Version of the PostgreSQL custom extension

8.1.7 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-12-22

Percona LLC and/or its affiliates, © 2009 - 2024

8.2 Percona certified images

Following table presents Percona's certified docker images to be used with the Percona Operator for PostgreSQL:

Percona LLC and/or its affiliates, © 2009 - 2024

Image	Digest
percona/percona-postgresql-operator:2.3.1	a6495c8e13d9fe3f50df12219e9d9cf64fa610fe5680a0a78d0e5c4fb3be2456
percona/percona-postgresql-operator:2.3.1-ppg12-postgres	f65fec42a82c937ea0ce1b25898a245d469375ed6ffddd2a396a032c86ccc2ee
percona/percona-postgresql-operator:2.3.1-ppg13-postgres	e6fd0b1f84d9ecf1baab04b477720ae46faf7f1df031b36dcf4eb94b4bdfc0d1
percona/percona-postgresql-operator:2.3.1-ppg14-postgres	8476acea85f323f8f22e96c1aef59d84ad3997b2ccc3b1ab9d3eb70b734d5f8c
percona/percona-postgresql-operator:2.3.1-ppg15-postgres	a8ef34191c6d29f93f4a9fdde3f97f89284d67726719c94f1f7f14bd2312677e
percona/percona-postgresql-operator:2.3.1-ppg16-postgres	2f329755dda215e233512c6e453a850bfc7505a7dfceb3a6ed64b8b84e532c15
percona/percona-postgresql-operator:2.3.1-ppg12-postgres-gis	876daa942c3f564b87d9af0bb2d17fca52b377f7f65d6fae8ac876f061ae3c9b
percona/percona-postgresql-operator:2.3.1-ppg13-postgres-gis	f35605252375afc6c9126f0bedc1f3cc6bfa79cbb6bf14dd85eb9f949c030201
percona/percona-postgresql-operator:2.3.1-ppg14-postgres-gis	5d307ea8925187413b77eb7767abe699b977cfa5e2448a7bc74ce150648e61c2
percona/percona-postgresql-operator:2.3.1-ppg15-postgres-gis	7bb8c9c5c077e376d45822e77924f8f1f44a06d4e239cfe2d360e91fbe71a25c
percona/percona-postgresql-operator:2.3.1-ppg16-postgres-gis	9277a9c3b3e69865c293e671a834299866357adb8237e787283569be1faec714
percona/percona-postgresql-operator:2.3.1-ppg12-pgbouncer	2613992361e9b6fa7aa037a139c069ce9c7b6a2282cdb586782511ec4213bfaf
percona/percona-postgresql-operator:2.3.1-ppg13-pgbouncer	7569a872d999803f53b795a18babd897bfac8ce5aa20e09c4c10be1c643a9399
percona/percona-postgresql-operator:2.3.1-ppg14-pgbouncer	d7b3756b42ded49defc1a5e3641fbeec18d4f3ac4e498c96999c55e764412240
percona/percona-postgresql-operator:2.3.1-ppg15-pgbouncer	8b3f138102f19c9f04c02d1eda409c44a7894ea66108bbc4b18b61ed4b002c60
percona/percona-postgresql-operator:2.3.1-ppg16-pgbouncer	6b6ceba33c3105a40e43bc8e47cd24d6379373dbd6fc25970d46b69b528fcd59
percona/percona-postgresql-operator:2.3.1-ppg12-pgbackrest	3c1f34a752238496d70ce08c765e24416a7d9fa13771ad7088a9440385deb262
	d659a6a6f2cd29425fb929c22226e5f48572bd83cca33a8cdb5d22846dd70299

Percona LLC and/or its affiliates, © 2009 - 2024

Image	Digest
percona/percona-postgresql-operator:2.3.1-ppg13-pgbackrest	
percona/percona-postgresql-operator:2.3.1-ppg14-pgbackrest	6033cbe86bf52b407196e5a5810b6323d5490fca68bb795d1e2192b27ba6dbc9
percona/percona-postgresql-operator:2.3.1-ppg15-pgbackrest	74679b3698b6b56a224280703c6423a13eb975f73f8cae6072ff6e523ba9db30
percona/percona-postgresql-operator:2.3.1-ppg16-pgbackrest	1c0271c06be6df3a02235f76364c2f7e92471f8b08385ed78219811bcce5338f
percona/pmm-client:2.41.0	60df62ef326075d42ad4eb30c037372ef1c20eff50e7d87c6da672be886e5ea7

8.2.1 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Last update: 2024-01-23

Percona LLC and/or its affiliates, © 2009 - 2024

8.3 Versions compatibility

Versions of the cluster components and platforms tested with different Operator releases are shown below. Other version combinations may also work but have not been tested.

Cluster components:

Operator	PostgreSQL	pgBackRest	pgBouncer
2.3.1	12 - 16	2.48	1.18.0
2.3.0	12 - 16	2.48	1.18.0
2.2.0	12 - 15	2.43	1.18.0
2.1.0	12 - 15	2.43	1.18.0
2.0.0	12 - 14	2.41	1.17.0
1.5.0	12 - 14	2.47	1.20.0
1.4.0	12 - 14	2.43	1.18.0
1.3.0	12 - 14	2.38	1.17.0
1.2.0	12 - 14	2.37	1.16.1
1.1.0	12 - 14	2.34	1.16.0 for PostgreSQL 12, 1.16.1 for other versions
1.0.0	12 - 13	2.33	1.13.0

Platforms:

Operator	GKE	EKS	Openshift	Minikube
2.3.1	1.24 - 1.28	1.24 - 1.28	4.11.55 - 4.14.6	1.32
2.3.0	1.24 - 1.28	1.24 - 1.28	4.11.55 - 4.14.6	1.32
2.2.0	1.23 - 1.26	1.23 - 1.27	-	1.30.1
2.1.0	1.23 - 1.25	1.23 - 1.25	-	-
2.0.0	1.22 - 1.25	-	-	-
1.5.0	1.24 - 1.28	1.24 - 1.28	4.11 - 4.14	1.32
1.4.0	1.22 - 1.25	1.22 - 1.25	4.10 - 4.12	1.28
1.3.0	1.21 - 1.24	1.20 - 1.22	4.7 - 4.10	-
1.2.0	1.19 - 1.22	1.19 - 1.21	4.7 - 4.10	-
1.1.0	1.19 - 1.22	1.18 - 1.21	4.7 - 4.9	-
1.0.0	1.17 - 1.21	1.21	4.6 - 4.8	-

8.3.1 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Percona LLC and/or its affiliates, © 2009 - 2024

Last update: 2024-01-23

Percona LLC and/or its affiliates, © 2009 - 2024

8.4 Copyright and licensing information

8.4.1 Documentation licensing

Percona Operator for PostgreSQL documentation is (C)2009-2023 Percona LLC and/or its affiliates and is distributed under the Creative Commons Attribution 4.0 International License.

8.4.2 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-06-27

Percona LLC and/or its affiliates, © 2009 - 2024

8.5 Trademark policy

This Trademark Policy is to ensure that users of Percona-branded products or services know that what they receive has really been developed, approved, tested and maintained by Percona. Trademarks help to prevent confusion in the marketplace, by distinguishing one company's or person's products and services from another's.

Percona owns a number of marks, including but not limited to Percona, XtraDB, Percona XtraDB, XtraBackup, Percona XtraBackup, Percona Server, and Percona Live, plus the distinctive visual icons and logos associated with these marks. Both the unregistered and registered marks of Percona are protected.

Use of any Percona trademark in the name, URL, or other identifying characteristic of any product, service, website, or other use is not permitted without Percona's written permission with the following three limited exceptions.

First, you may use the appropriate Percona mark when making a nominative fair use reference to a bona fide Percona product.

Second, when Percona has released a product under a version of the GNU General Public License ("GPL"), you may use the appropriate Percona mark when distributing a verbatim copy of that product in accordance with the terms and conditions of the GPL.

Third, you may use the appropriate Percona mark to refer to a distribution of GPL-released Percona software that has been modified with minor changes for the sole purpose of allowing the software to operate on an operating system or hardware platform for which Percona has not yet released the software, provided that those third party changes do not affect the behavior, functionality, features, design or performance of the software. Users who acquire this Percona-branded software receive substantially exact implementations of the Percona software.

Percona reserves the right to revoke this authorization at any time in its sole discretion. For example, if Percona believes that your modification is beyond the scope of the limited license granted in this Policy or that your use of the Percona mark is detrimental to Percona, Percona will revoke this authorization. Upon revocation, you must immediately cease using the applicable Percona mark. If you do not immediately cease using the Percona mark upon revocation, Percona may take action to protect its rights and interests in the Percona mark. Percona does not grant any license to use any Percona mark for any other modified versions of Percona software; such use will require our prior written permission.

Neither trademark law nor any of the exceptions set forth in this Trademark Policy permit you to truncate, modify or otherwise use any Percona mark as part of your own brand. For example, if XYZ creates a modified version of the Percona Server, XYZ may not brand that modification as "XYZ Percona Server" or "Percona XYZ Server", even if that modification otherwise complies with the third exception noted above.

In all cases, you must comply with applicable law, the underlying license, and this Trademark Policy, as amended from time to time. For instance, any mention of Percona trademarks should include the full trademarked name, with proper spelling and capitalization, along with attribution of ownership to Percona Inc. For example, the full proper name for XtraBackup is Percona XtraBackup. However, it is acceptable to omit the word "Percona" for brevity on the second and subsequent uses, where such omission does not cause confusion.

In the event of doubt as to any of the conditions or exceptions outlined in this Trademark Policy, please contact trademarks@percona.com for assistance and we will do our very best to be helpful.

8.5.1 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Percona LLC and/or its affiliates, © 2009 - 2024

Last update: 2023-06-27

Percona LLC and/or its affiliates, © 2009 - 2024

9. Release Notes

9.1 Percona Operator for PostgreSQL Release Notes

- *Percona Operator for PostgreSQL* 2.3.1 (2024-01-23)
- *Percona Operator for PostgreSQL* 2.3.0 (2023-12-21)
- *Percona Operator for PostgreSQL* 2.2.0 (2023-06-30)
- *Percona Operator for PostgreSQL* 2.1.0 Tech preview (2023-05-04)
- *Percona Operator for PostgreSQL* 2.0.0 Tech preview (2022-12-30)

9.1.1 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2024-01-23

Percona LLC and/or its affiliates, © 2009 - 2024

9.2 Percona Operator for PostgreSQL 2.3.1

- **Date**

  January 23, 2024

- **Installation**

  Installing Percona Operator for PostgreSQL

9.2.1 Release Highlights

This release provides a number of bug fixes, including fixes for the following vulnerabilities in PostgreSQL, pgBackRest, and pgBouncer images used by the Operator:

- OpenSSH could cause remote code execution by ssh-agent if a user establishes an SSH connection to a compromised or malicious SSH server and has agent forwarding enabled (CVE-2023-38408). This vulnerability affects pgBackRest and PostgreSQL images.
- The c-ares library could cause a Denial of Service with 0-byte UDP payload (CVE-2023-32067). This vulnerability affects pgBouncer image.

Both Operator 1.x (including version 1.5.0) and Operator 2.x (including version 2.3.0) are affected. The 2.x versions upgrade to 2.3.1 is recommended to resolve these issues.

9.2.2 Bugs Fixed

- K8SPG-493: Fix a regression due to which the Operator could run scheduled backup only one time
- K8SPG-494: Fix vulnerabilities in PostgreSQL, pgBackRest, and pgBouncer images
- K8SPG-496: Fix the bug where setting the `pause` Custom Resource option to `true` for the cluster with a backup running would not take effect even after the backup completed

9.2.3 Supported platforms

The Operator was developed and tested with PostgreSQL versions 12.17, 13.13, 14.10, 15.5, and 16.1. Other options may also work but have not been tested. The Operator 2.3.1 provides connection pooling based on pgBouncer 1.21.0 and high-availability implementation based on Patroni 3.1.0.

The following platforms were tested and are officially supported by the Operator 2.3.1:

- Google Kubernetes Engine (GKE) 1.24 - 1.28
- Amazon Elastic Container Service for Kubernetes (EKS) 1.24 - 1.28
- OpenShift 4.11.55 - 4.14.6
- Minikube 1.32

This list only includes the platforms that the Percona Operators are specifically tested on as part of the release process. Other Kubernetes flavors and versions depend on the backward compatibility offered by Kubernetes itself.

9.2.4 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Percona LLC and/or its affiliates, © 2009 - 2024

Last update: 2024-01-23

Percona LLC and/or its affiliates, © 2009 - 2024

9.3 Percona Operator for PostgreSQL 2.3.0

- **Date**

  December 21, 2023

- **Installation**

  Installing Percona Operator for PostgreSQL

9.3.1 Release Highlights

PostGIS support

Modern businesses heavily rely on location-based data to gain valuable insights and make data-driven decisions. However, integrating geospatial functionality into the existing database systems has often posed a challenge for enterprises. PostGIS, an open-source software extension for PostgreSQL, addresses this difficulty by equipping users with extensive geospatial operations for handling geographic data efficiently. Percona Operator now supports PostGIS, available through a separate container image. You can read more about PostGIS and how to use it with the Operator in our documentation.

9.3.2 OpenShift and PostgreSQL 16 support

The Operator is now compatible with the OpenShift platform empowering enterprise customers with seamless on-premise or cloud deployments on the platform of their choice. Also, PostgreSQL 16 was added to the range of supported database versions and is used by default starting with this release.

Experimental support for custom PostgreSQL extensions

One of great features of PostgreSQL is support for Extensions, which allow adding new functionality to the database on a plugin basis. Starting from this release, users can add custom PostgreSQL extensions dynamically, without the need to rebuild the container image (see this HowTo on how to create and connect yours).

9.3.3 New features

- K8SPG-311 and K8SPG-389: A new `loadBalancerSourceRanges` Custom Resource option allows to customize the range of IP addresses from which the load balancer should be reachable
- K8SPG-375: Experimental support for custom PostgreSQL extensions was added to the Operator
- K8SPG-391: The Operator is now compatible with the OpenShift platform
- K8SPG-434: The Operator now supports Percona Distribution for PostgreSQL version 16 and uses it as default database version

9.3.4 Improvements

- K8SPG-413: The Operator documentation now includes a comptibility matrix for each Operator version, specifying exact versions of all core components as well as supported versions of the database and platforms
- K8SPG-332: Creating backups and pausing the cluster do not interfere with each other: the Operator either postpones the pausing until the active backup ends, or postpones the scheduled backup on the paused cluster
- K8SPG-370: Logging management is now aligned with other Percona Operators, allowing to use structured logging and to control log level
- K8SPG-372: The multi-namespace (cluster-wide) mode of the Operator was improved, making it possible to customize the list of Kubernetes namespaces under the Operator's control

Percona LLC and/or its affiliates, © 2009 - 2024

- K8SPG-400: The documentation now explains how to allow application users to connect to a database cluster without TLS (for example, for testing or demonstration purposes)
- K8SPG-410: Scheduled backups now create `pg-backup` object to simplify backup management and tracking
- K8SPG-416: PostgreSQL custom configuration is now supported in the Helm chart
- K8SPG-422 and K8SPG-447: The user can now see backup type and status in the output of `kubectl get pg-backup` and `kubectl get pg-restore` commands
- K8SPG-458: Affinity configuration examples were added to the `default/cr.yaml` configuration file

9.3.5 Bugs Fixed

- K8SPG-435: Fix a bug with insufficient size of /tmp filesystem which caused PostgreSQL Pods to be recreated every few days due to running out of free space on it
- K8SPG-453: Bug in `pg_stat_monitor` PostgreSQL extensions could hang PostgreSQL
- K8SPG-279: Fix regression which made the Operator to crash after creating a backup if there was no backups.pgbackrest.manual section in the Custom Resource
- K8SPG-310: Documentation didn't explain how to apply pgBackRest `verifyTLS` option which can be used to explicitly enable or disable TLS verification for it
- K8SPG-432: Fix a bug due to which backup jobs and Pods were not deleted on deleting the backup object
- K8SPG-442: The Operator didn't allow to append custom items to the PostgreSQL `shared_preload_libraries` option
- K8SPG-443: Fix a bug due to which only English locale was installed in the PostgreSQL image, missing other languages support
- K8SPG-450: Fix a bug which prevented PostgreSQL to initialize the database on Kubernetes working nodes with enabled huge memory pages if Pod resource limits didn't allow using them
- K8SPG-401: Fix a bug which caused Operator crash if deployed with no `pmm` section in the `deploy/cr.yaml` configuration file

9.3.6 Supported platforms

The Operator was developed and tested with PostgreSQL versions 12.17, 13.13, 14.10, 15.5, and 16.1. Other options may also work but have not been tested. The Operator 2.3.0 provides connection pooling based on pgBouncer 1.21.0 and high-availability implementation based on Patroni 3.1.0.

The following platforms were tested and are officially supported by the Operator 2.3.0:

- Google Kubernetes Engine (GKE) 1.24 - 1.28
- Amazon Elastic Container Service for Kubernetes (EKS) 1.24 - 1.28
- OpenShift 4.11.55 - 4.14.6
- Minikube 1.32

This list only includes the platforms that the Percona Operators are specifically tested on as part of the release process. Other Kubernetes flavors and versions depend on the backward compatibility offered by Kubernetes itself.

9.3.7 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Percona LLC and/or its affiliates, © 2009 - 2024

Last update: 2023-12-21

Percona LLC and/or its affiliates, © 2009 - 2024

9.4 Percona Operator for PostgreSQL 2.2.0

- **Date**

  June 30, 2023

- **Installation**

  Installing Percona Operator for PostgreSQL

Percona announces the general availability of Percona Operator for PostgreSQL 2.2.0.

Starting with this release, Percona Operator for PostgreSQL version 2 is out of technical preview and can be used in production with all the improvements it brings over the version 1 in terms of architecture, backup and recovery features, and overall flexibility.

We prepared a detailed migration guide which allows existing Operator 1.x users to move their PostgreSQL clusters to the Operator 2.x. Also, see this blog post to find out more about the Operator 2.x features and benefits.

9.4.1 Improvements

- K8SPG-378: A new `crVersion` Custom Resource option was added to indicate the API version this Custom Resource corresponds to
- K8SPG-359: The new `users.secretName` option allows to define a custom Secret name for the users defined in the Custom Resource (thanks to Vishal Anarase for contributing)
- K8SPG-301: Amazon Elastic Container Service for Kubernetes (EKS) was added to the list of officially supported platforms
- K8SPG-302: Minikube is now officially supported by the Operator to enable ease of testing and developing
- K8SPG-326: Both the Operator and database can be now installed with the Helm package manager
- K8SPG-342: There is now no need in manual restart of PostgreSQL Pods after the monitor user password changed in Secrets
- K8SPG-345: The new `proxy.pgBouncer.exposeSuperusers` Custom Resource option makes it possible for administrative users to connect to PostgreSQL through PgBouncer
- K8SPG-355: The Operator can now be deployed in multi-namespace (“cluster-wide”) mode to track Custom Resources and manage database clusters in several namespaces

9.4.2 Bugs Fixed

- K8SPG-373: Fix the bug due to which the Operator did not not create Secrets for the `pguser` user if PMM was enabled in the Custom Resource
- K8SPG-362: It was impossible to install Custom Resource Definitions for both 1.x and 2.x Operators in one environment, preventing the migration of a cluster to the newer Operator version
- K8SPG-360: Fix a bug due to which manual password changing or resetting via Secret didn’t work

Known limitations

- Query analytics (QAN) will not be available in Percona Monitoring and Management (PMM) due to bugs PMM-12024 and PMM-11938. The fixes are included in the upcoming PMM 2.38, so QAN can be used as soon as it is released and both PMM Client and PMM Server are upgraded.

Percona LLC and/or its affiliates, © 2009 - 2024

9.4.3 Supported platforms

The Operator was developed and tested with PostgreSQL versions 12.14, 13.10, 14.7, and 15.2. Other options may also work but have not been tested. The Operator 2.2.0 provides connection pooling based on pgBouncer 1.18.0 and high-availability implementation based on Patroni 3.0.1.

The following platforms were tested and are officially supported by the Operator 2.2.0:

- Google Kubernetes Engine (GKE) 1.23 - 1.26
- Amazon Elastic Container Service for Kubernetes (EKS) 1.23 - 1.27
- Minikube 1.30.1 (based on Kubernetes 1.27)

This list only includes the platforms that the Percona Operators are specifically tested on as part of the release process. Other Kubernetes flavors and versions depend on the backward compatibility offered by Kubernetes itself.

9.4.4 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and “ask me anything” sessions with the Experts.

Last update: 2023-08-22

Percona LLC and/or its affiliates, © 2009 - 2024

9.5 Percona Operator for PostgreSQL 2.1.0 (Tech preview)

- **Date**

  May 4, 2023
- **Installation**

  Installing Percona Operator for PostgreSQL

The Percona Operator built with best practices of configuration and setup of Percona Distribution for PostgreSQL on Kubernetes.

Percona Operator for PostgreSQL helps create and manage highly available, enterprise-ready PostgreSQL clusters on Kubernetes. It is 100% open source, free from vendor lock-in, usage restrictions and expensive contracts, and includes enterprise-ready features: backup/restore, high availability, replication, logging, and more.

The benefits of using Percona Operator for PostgreSQL include saving time on database operations via automation of Day-1 and Day-2 operations and deployment of consistent and vetted environment on Kubernetes.

Note

Version 2.1.0 of the Percona Operator for PostgreSQL is a **tech preview release** and it is **not recommended for production environments.** As of today, we recommend using Percona Operator for PostgreSQL 1.x, which is production-ready and contains everything you need to quickly and consistently deploy and scale PostgreSQL clusters in a Kubernetes-based environment, on-premises or in the cloud.

9.5.1 Release Highlights

- PostgreSQL 15 is now officially supported by the Operator with the new exciting features it brings to developers
- UX improvements related to Custom Resource have been added in this release, including the handy `pg`, `pg-backup`, and `pg-restore` short names useful to quickly query the cluster state with the `kubectl get` command and additional information in the status fields, which now show `name`, `endpoint`, `status`, and `age`

9.5.2 New Features

- K8SPG-328: The new `delete-pvc` finalizer allows to either delete or preserve Persistent Volumes at Custom Resource deletion
- K8SPG-330: The new `delete-ssl` finalizer can now be used to automatically delete objects created for SSL (Secret, certificate, and issuer) in case of cluster deletion
- K8SPG-331: Starting from now, the Operator adds short names to its Custom Resources: `pg`, `pg-backup`, and `pg-restore`
- K8SPG-282: PostgreSQL 15 is now officially supported by the Operator

9.5.3 Improvements

- K8SPG-262: The Operator now does not attempt to start Percona Monitoring and Management (PMM) client if the corresponding secret does not contain the `pmmserver` or `pmmserverkey` key
- K8SPG-285: To improve the Operator we capture anonymous telemetry and usage data. In this release we add more data points to it

Percona LLC and/or its affiliates, © 2009 - 2024

- K8SPG-295: Additional information was added to the status of the Operator Custom Resource, which now shows `name`, `endpoint`, `status`, and `age` fields
- K8SPG-304: The Operator stops using trust authentication method in `pg_hba.conf` for better security
- K8SPG-325: Custom Resource options previously named `paused` and `shutdown` were renamed to `unmanaged` and `pause` for better alignment with other Percona Operators

9.5.4 Bugs Fixed

- K8SPG-272: Fix a bug due to which PMM agent related to the Pod wasn't deleted from the PMM Server inventory on Pod termination
- K8SPG-279: Fix a bug which made the Operator to crash after creating a backup if there was no `backups.pgbackrest.manual` section in the Custom Resource
- K8SPG-298: Fix a bug due to which the `shutdown` Custom Resource option didn't work making it impossible to pause the cluster
- K8SPG-334: Fix a bug which made it possible for the monitoring user to have special characters in the autogenerated password, making it incompatible with the PMM Client

9.5.5 Supported platforms

The following platforms were tested and are officially supported by the Operator 2.1.0:

- Google Kubernetes Engine (GKE) 1.23 - 1.25
- Amazon Elastic Container Service for Kubernetes (EKS) 1.23 - 1.25

This list only includes the platforms that the Percona Operators are specifically tested on as part of the release process. Other Kubernetes flavors and versions depend on the backward compatibility offered by Kubernetes itself.

9.5.6 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Community Forum Get a Percona Expert Join K8S Squad

Last update: 2023-08-22

Percona LLC and/or its affiliates, © 2009 - 2024

9.6 Percona Operator for PostgreSQL 2.0.0 (Tech preview)

- **Date**

  December 30, 2022

- **Installation**

  Installing Percona Operator for PostgreSQL

The Percona Operator is based on best practices for configuration and setup of a Percona Distribution for PostgreSQL on Kubernetes. The benefits of the Operator are many, but saving time and delivering a consistent and vetted environment is key.

Note

Version 2.0.0 of the Percona Operator for PostgreSQL is a **tech preview release** and it is **not recommended for production environments.** As of today, we recommend using Percona Operator for PostgreSQL 1.x, which is production-ready and contains everything you need to quickly and consistently deploy and scale PostgreSQL clusters in a Kubernetes-based environment, on-premises or in the cloud.

The *Percona Operator for PostgreSQL 2.x* is based on the 5.x branch of the Postgres Operator developed by Crunchy Data. Please see the main changes in this version below.

9.6.1 Architecture

Operator SDK is now used to build and package the Operator. It simplifies the development and brings more contribution friendliness to the code, resulting in better potential for growing the community. Users now have full control over Custom Resource Definitions that Operator relies on, which simplifies the deployment and management of the operator.

In version 1.x we relied on Deployment resources to run PostgreSQL clusters, whereas in 2.0 Statefulsets are used, which are the de-facto standard for running stateful workloads in Kubernetes. This change improves stability of the clusters and removes a lot of complexity from the Operator.

9.6.2 Backups

One of the biggest challenges in version 1.x is backups and restores. There are two main problems that our user faced:

- Not possible to change backup configuration for the existing cluster
- Restoration from backup to the newly deployed cluster required workarounds

In this version both these issues are fixed. In addition to that:

- Run up to 4 pgBackrest repositories
- Bootstrap the cluster from the existing backup through Custom Resource
- Azure Blob Storage support

9.6.3 Operations

Deploying complex topologies in Kubernetes is not possible without affinity and anti-affinity rules. In version 1.x there were various limitations and issues, whereas this version comes with substantial improvements that enables users to craft the topology of their choice.

Percona LLC and/or its affiliates, © 2009 - 2024

Within the same cluster users can deploy multiple instances. These instances are going to have the same data, but can have different configuration and resources. This can be useful if you plan to migrate to new hardware or need to test the new topology.

Each postgreSQL node can have sidecar containers now to provide integration with your existing tools or expand the capabilities of the cluster.

9.6.4 Try it out now

Excited with what you read above?

- We encourage you to install the Operator following our documentation.
- Feel free to share feedback with us on the forum or raise a bug or feature request in JIRA.
- See the source code in our Github repository.

9.6.5 Get expert help

If you need assistance, visit the community forum for comprehensive and free database knowledge, or contact our Percona Database Experts for professional support and services. Join K8S Squad to benefit from early access to features and "ask me anything" sessions with the Experts.

Last update: 2023-08-22

Percona LLC and/or its affiliates, © 2009 - 2024