PERCONA

Operator for MySQL based on Percona XtraDB Cluster

Documentation

1.19.0 (January 19, 2026)

Table of Contents

Percona Operator for MySQL Based on Percona XtraDB Cluster

The **Percona Operator for MySQL** is a Kubernetes-native solution designed to simplify the deployment, management, and scaling of MySQL clusters built on Percona XtraDB Cluster (PXC). The Operator leverages Kubernetes' orchestration capabilities to automate critical database management tasks, including cluster provisioning, backups, failover, and scaling.

Percona XtraDB Cluster (PXC) is an open-source, enterprise-grade MySQL solution designed for high availability and data consistency. It uses synchronous replication to ensure that data is consistent across all nodes in the cluster. PXC provides fault tolerance, automated failover, and scalability, making it ideal for running highly available MySQL databases in mission-critical environments.

This provides the foundation for the Percona Operator for MySQL, enabling simplified deployment and management of Percona XtraDB Cluster within Kubernetes environments.

What's new in version 1.19.0

Key Features and Benefits

1. Automated Deployment and Scaling

- Simplifies the creation of MySQL clusters with minimal configuration.
- Dynamically scales instances based on workload demands, optimizing resource usage.

2. High Availability

- Guarantees zero downtime with automated failover mechanisms.
- Utilizes synchronous replication to maintain data consistency across nodes.

3. Self-Healing

- Detects and recovers from node failures automatically to maintain cluster health.
- Ensures operational continuity with minimal manual intervention.

4. Backup and Restore

- Provides consistent, automated backups to cloud storage or local volumes.
- Enables quick recovery, ensuring data safety and business continuity.

5. Enhanced Security

- Supports encryption for data at rest and in transit.
- Integrates with Kubernetes Role-Based Access Control (RBAC) for secure database operations.

6. Operational Simplification

- Offers seamless integration with Kubernetes-native tools like `kubectl`.
- Streamlines database monitoring, management, and troubleshooting.

7. Flexibility for Cloud-Native Architectures

- Optimized for public, private, and hybrid cloud deployments.
- Allows unified management of databases across diverse environments.

Use Case

The **Percona Operator for MySQL** is ideal for various scenarios such as providing Database as a Service (DBaaS), ensuring high availability for mission-critical applications, scaling cloud-native applications, and implementing disaster recovery strategies. It is particularly useful for organizations with hybrid or multi-cloud infrastructures, where it simplifies the deployment and management of MySQL clusters across multiple environments. The Operator also benefits development and testing teams by enabling quick spin-up of MySQL clusters for testing and development purposes, helping to accelerate product development cycles and reduce operational overhead.

Being part of the open-source ecosystem, the Percona Operator benefits from community contributions and support, ensuring that it remains stable and robust over time.

If you're interested in contributing, feel free to: - Open an issue - Submit a pull request

For support or inquiries, contact Percona.

Get help from Percona

Our documentation guides are packed with information, but they can't cover everything you need to know about Percona Operator for MySQL Based on Percona XtraDB Cluster. They also won't cover every scenario you might come across. Don't be afraid to try things out and ask questions when you get stuck.

Percona's Community Forum

Be a part of a space where you can tap into a wealth of knowledge from other database enthusiasts and experts who work with Percona's software every day. While our service is entirely free, keep in mind that response times can vary depending on the complexity of the question. You are engaging with people who genuinely love solving database challenges.

We recommend visiting our Community Forum. It's an excellent place for discussions, technical insights, and support around Percona database software. If you're new and feeling a bit unsure, our FAQ and Guide for New Users ease you in.

If you have thoughts, feedback, or ideas, the community team would like to hear from you at Any ideas on how to make the forum better?. We're always excited to connect and improve everyone's experience.

Percona experts

Percona experts bring years of experience in tackling tough database performance issues and design challenges.

Talk to a Percona Expert

We understand your challenges when managing complex database environments. That's why we offer various services to help you simplify your operations and achieve your goals.

Service	Description
24/7 Expert Support	Our dedicated team of database experts is available 24/7 to assist you with any database issues. We provide flexible support plans tailored to your specific needs.
Hands-On Database Management	Our managed services team can take over the day-to-day management of your database infrastructure, freeing up your time to focus on other priorities.
Expert Consulting	Our experienced consultants provide guidance on database topics like architecture design, migration planning, performance optimization, and security best practices.
Comprehensive Training	Our training programs help your team develop skills to manage databases effectively, offering virtual and in-person courses.

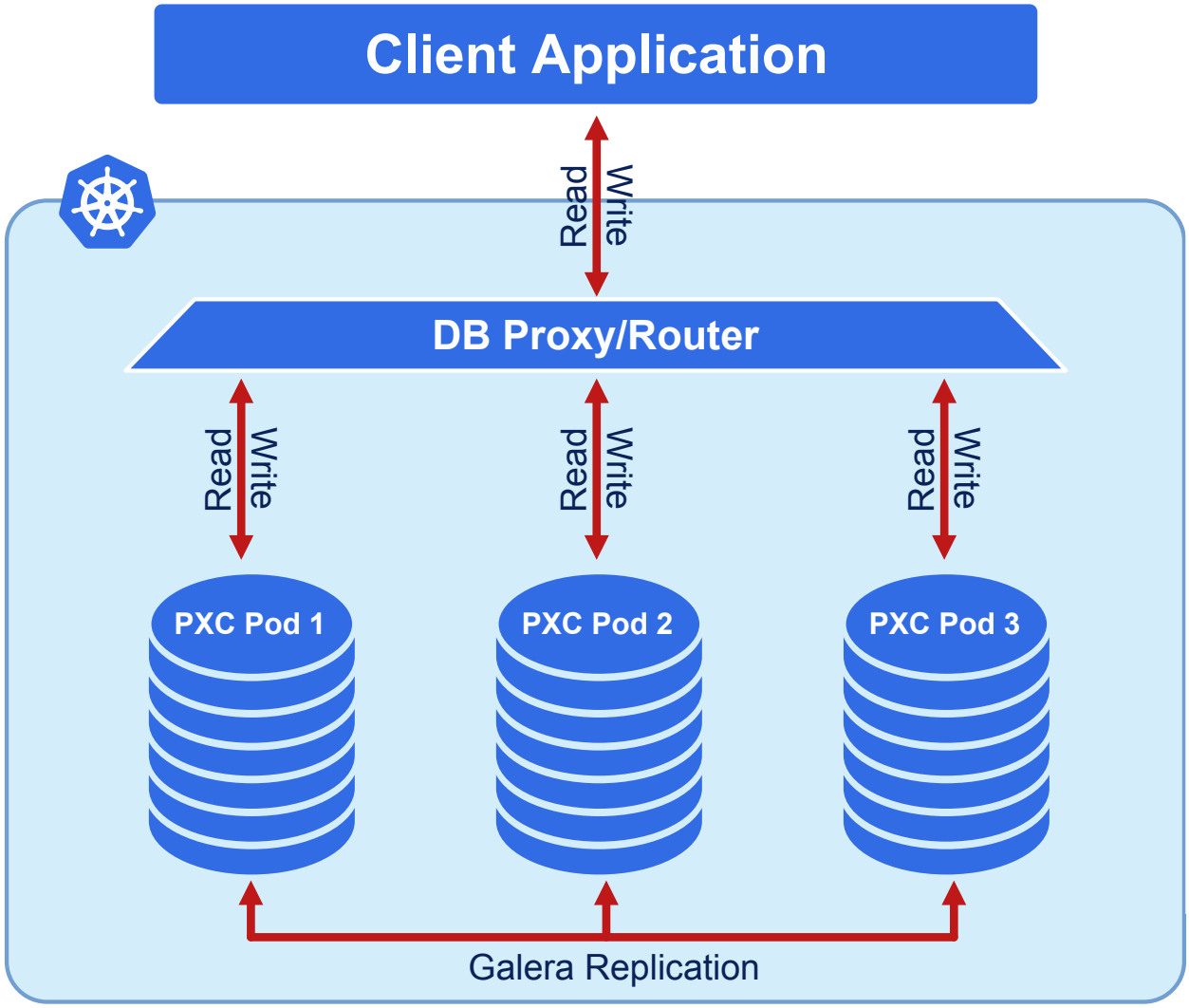
We're here to help you every step of the way. Whether you need a quick fix or a long-term partnership, we're ready to provide your expertise and support.

Features

Design overview

Percona XtraDB Cluster integrates *Percona Server for MySQL* running with the XtraDB storage engine, and *Percona XtraBackup* with the *Galera library* to enable synchronous multi-primary replication.

The design of the Operator is highly bound to the Percona XtraDB Cluster high availability implementation, which in its turn can be briefly described with the following diagram.



Being a regular MySQL Server instance, each node contains the same set of data synchronized across nodes. For production use, the recommended configuration is a minimum of 3 nodes and a maximum of 5 nodes.

Having a higher number of nodes (for example, 7) is not recommended because Percona XtraDB Cluster uses synchronous replication (Galera). Every write must be confirmed by all nodes. With 7 nodes, each transaction involves more network hops and acknowledgments, which increases latency and resource usage.

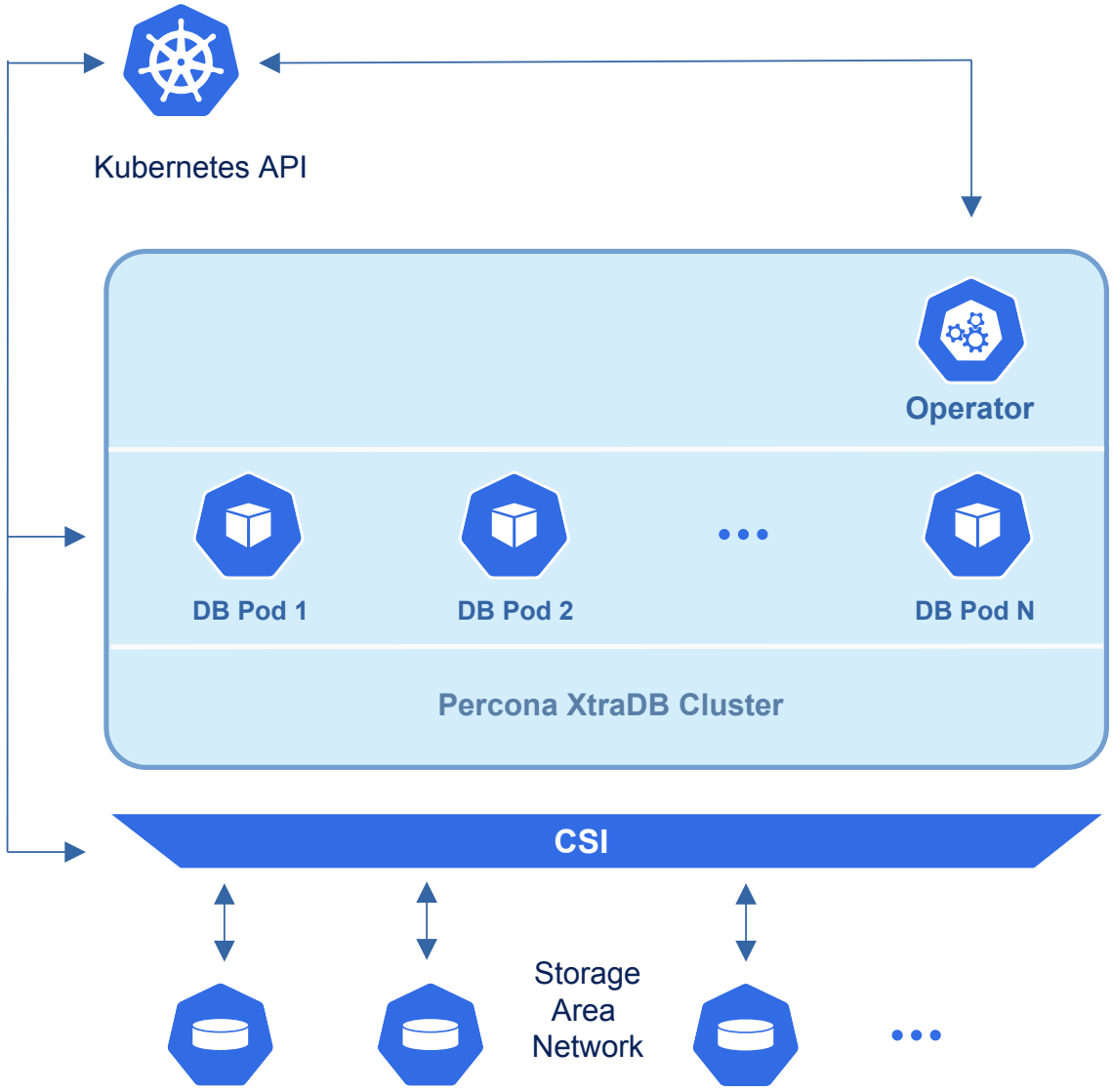
Even numbers of nodes are not recommended because Galera replication relies on quorum voting. With an even number of nodes, a network partition can split the cluster into two equal halves, leaving no majority. Without quorum, both sides stop serving queries to avoid data inconsistency, which reduces availability.

In a basic setup with 3 nodes, Percona XtraDB Cluster provides high availability, continuing to function if you take any of the nodes down. Additionally load balancing can be achieved with the HAProxy router, which accepts incoming traffic from MySQL clients and forwards it to backend MySQL servers.

Note

Optionally the Operator allows using ProxySQL daemon instead of HAProxy, which provides SQL-aware database workload management and can be more more efficient in comparison with other load balancers.

To provide high availability operator uses node affinity to run Percona XtraDB Cluster instances on separate worker nodes if possible. If some node fails, the pod with it is automatically re-created on another node.



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 PVC StorageClass must support this feature (Kubernetes and OpenShift support this in versions 1.9 and 3.9 respectively).

The Operator functionality extends the Kubernetes API with *PerconaXtraDBCluster* object, and it is implemented as a golang application. Each *PerconaXtraDBCluster* object maps to one separate Percona XtraDB Cluster setup. The Operator listens to all events on the created objects. When a new PerconaXtraDBCluster object 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 XtraDB Cluster operation.

Compare various solutions to deploy MySQL in Kubernetes

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

- KubeDB
- Bitpoke MySQL Operator (former Presslabs)
- Oracle MySQL Operator
- Moco by Cybozu
- Vitess Operator by PlanetScale
- Percona Operator for MySQL
  - based on Percona XtraDB Cluster
  - based on Percona Server for MySQL

Generic

The review of generic features, such as supported MySQL versions, open source models and more.

Feature/Product	Percona Operator for MySQL (based on PXC)	Percona Operator for MySQL (based on PS)	Bitpoke MySQL Operator	Moco	Oracle MySQL Operator	Vitess
Open source model	Apache 2.0	Apache 2.0	Apache 2.0	Apache 2.0	Apache 2.0	Apache 2.0
MySQL versions	5.7, 8.0	8.0	5.7	8.0	8.0	5.7, 8.0
Kubernetes conformance	Various versions are tested	Various versions are tested	Not guaranteed	Not guaranteed	Not guaranteed	Not guaranteed
Paid support	✅	✅	🚫	🚫	✅	🚫
Web-based GUI	Percona Everest	🚫	🚫	🚫	Oracle Enterprise Manager	🚫

MySQL Topologies

Focus on replication capabilities and proxies integrations.

Feature/Product	Percona Operator for MySQL (based on PXC)	Percona Operator for MySQL (based on PS)	Bitpoke MySQL Operator	Moco	Oracle MySQL Operator	Vitess
Replication	Sync with Galera	Async and Group Replication	Async	Semi-sync	Group Replication	Async
Proxy	HAProxy and ProxySQL	HAProxy and MySQL Router	None	None	MySQL Router	VTGate
Multi-cluster deployment	✅	🚫	🚫	🚫	🚫	🚫
Sharding	🚫	🚫	🚫	🚫	🚫	✅

Backups

Here are the backup and restore capabilities of each solution.

Feature/Product	Percona Operator for MySQL (based on PXC)	Percona Operator for MySQL (based on PS)	Bitpoke MySQL Operator	Moco	Oracle MySQL Operator	Vitess
Scheduled backups	✅	✅	✅	✅	✅	✅

Incremental backups	⛔	⛔	⛔	✅	⛔	⛔
PITR	✅	⛔	⛔	⛔	⛔	⛔
PVCs for backups	✅	⛔	⛔	⛔	✅	⛔

Monitoring

Monitoring is crucial for any operations team.

Feature/Product	Percona Operator for MySQL (based on PXC)	Percona Operator for MySQL (based on PS)	Bitpoke MySQL Operator	Moco	Oracle MySQL Operator	Vitess
Custom exporters	Through sidecars	Through sidecars	mysqld_exporter	mysqld_exporter	⛔	⛔
PMM	✅	✅	⛔	⛔	⛔	⛔

Miscellaneous

Compare various features that are not a good fit for other categories.

Feature/Product	Percona Operator for MySQL (based on PXC)	Percona Operator for MySQL (based on PS)	Bitpoke MySQL Operator	Moco	Oracle MySQL Operator	Vitess
Customize MySQL	ConfigMaps and Secrets	ConfigMaps and Secrets	ConfigMaps	ConfigMaps	ConfigMaps	⛔
Helm	✅	✅	✅	✅	✅	⛔
Transport encryption	✅	✅	⛔	⛔	✅	✅
Encryption-at-rest	✅	✅	⛔	⛔	⛔	⛔

Quickstart guides

Overview

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

- Install and deploy an Operator
- Connect to MySQL instance in Percona XtraDB Cluster
- Insert sample data to the database
- Set up and make a logical backup
- Monitor the database health with Percona Monitoring and Management (PMM)

Next steps

Install the Operator →

1. Quick install

Install Percona XtraDB Cluster using Helm

Helm is the package manager for Kubernetes. Percona Helm charts can be found in percona/percona-helm-charts repository on Github.

Pre-requisites

1. The **Helm** package manager. Install it following the official installation instructions.

   > **Note**
   >
   > Helm v3 is needed to run the following steps.

2. The **kubectl** tool to manage and deploy applications on Kubernetes. Install it following the official installation instructions.

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 <namespace>
   ```

3. Install the Percona Operator for MySQL based on Percona XtraDB Cluster:

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

   The `namespace` is the name of your namespace. The `my-op` parameter in the above example 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 XtraDB Cluster:

   ```
   $ helm install my-db percona/pxc-db --namespace <namespace>
   ```

   The `my-db` parameter in the above example is the name of a new release object which is created for the Percona XtraDB Cluster when you install its Helm chart (use any name you like).

5. Check the Operator and the Percona XtraDB Cluster Pods status.

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

   The creation process may take some time. When the process is over your cluster obtains the `ready` status.

   Expected output

   ```
   NAME       ENDPOINT                  STATUS   PXC   PROXYSQL   HAPROXY   AGE
   cluster1   cluster1-haproxy.default   ready    3               3         33d
   ```

You have successfully installed and deployed the Operator with default parameters.

This deploys default Percona XtraDB Cluster configuration with three HAProxy and three XtraDB Cluster instances.

You can find in the documentation for the charts, which Operator and database parameters can be customized during installation. Also you can check the rest of the Operator's parameters in the Custom Resource options reference.

Next steps

Connect to Percona XtraDB Cluster →

Useful links

Install Percona XtraDB Cluster with customized parameters

Install Percona XtraDB Cluster 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 MySQL based on XtraDB Cluster is based on best practices for configuration and setup of a Percona Server for MySQL 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.

Install with kubectl ↓ Install with Helm →

Prerequisites

To install Percona XtraDB Cluster, 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
- Create and configure the AKS cluster

Procedure

Here's a sequence of steps to follow:

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. Replace the `<namespace>` placeholder with your value.

```
$ kubectl create namespace <namespace>
```

Expected output

```
namespace/<namespace> was created
```

2. Deploy the Operator with the following command:

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-
operator/v1.19.0/deploy/bundle.yaml  -n <namespace>
```

Expected output

```
customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusters.pxc.percona.com created
customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusterbackups.pxc.percona.com created
customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusterrestores.pxc.percona.com created
customresourcedefinition.apiextensions.k8s.io/perconaxtradbbackups.pxc.percona.com created
role.rbac.authorization.k8s.io/percona-xtradb-cluster-operator created
serviceaccount/percona-xtradb-cluster-operator created
rolebinding.rbac.authorization.k8s.io/service-account-percona-xtradb-cluster-operator created
deployment.apps/percona-xtradb-cluster-operator created
```

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

3. Deploy Percona XtraDB Cluster:

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-operator/v1.19.0/deploy/cr.yaml -n
<namespace>
```

Expected output

```
perconaxtradbcluster.pxc.percona.com/ cluster1 created
```

4. Check the Operator and the Percona XtraDB Cluster Pods status.

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

The creation process may take some time. When the process is over your cluster obtains the `ready` status.

Expected output

```
NAME       ENDPOINT                  STATUS   PXC   PROXYSQL   HAPROXY   AGE
cluster1   cluster1-haproxy.default   ready    3                3         5m51s
```

You have successfully installed and deployed the Operator with default parameters.

The default Percona XtraDB Cluster configuration includes three HAProxy and three XtraDB Cluster instances.

You can check the rest of the Operator's parameters in the Custom Resource options reference.

Next steps

Connect to Percona XtraDB Cluster →

Useful links

Install Percona XtraDB Cluster with customized parameters

2. Connect to Percona XtraDB Cluster

In this tutorial, you will connect to the Percona XtraDB Cluster you deployed previously.

To connect to Percona XtraDB Cluster you will need the password for the `root` user. Passwords are stored in the Secrets object.

Here's how to get it:

1. List the Secrets objects

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

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

   via kubectl

   `cluster1-secrets`

   via Helm

   `cluster1-pxc-db-secrets`

2. Retrieve the password for the root user. Replace the `secret-name` and `namespace` with your values in the following commands:

```
$ kubectl get secret <secret-name> -n <namespace> --template='{{.data.root | base64decode}}{{"\n"}}'
```

1. Run a container with `mysql` tool and connect its console output to your terminal. The following command does this, naming the new Pod `percona-client`:

   ```
   $ kubectl run -n <namespace> -i --rm --tty percona-client --image=percona:8.0 --restart=Never -- bash -il
   ```

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

2. Connect to Percona XtraDB Cluster. To do this, run `mysql` tool in the percona-client command shell using your cluster name and the password obtained from the secret. The command will look different depending on whether your cluster provides load balancing with HAProxy (the default choice) or ProxySQL. If your password contains special characters, they may be interpreted by the shell, and you may get "Permission denied" messages,so put the password in single quotes (single quotes also avoid variable expansion in scripts):

   with HAProxy (default)

   ```
   $ mysql -h <cluster_name>-haproxy -uroot -p'<root_password>'
   ```

   with ProxySQL

   ```
   $ mysql -h <cluster_name>-proxysql -uroot -p'<root_password>'
   ```

Congratulations! You have connected to Percona XtraDB Cluster.

Next steps

Insert sample data →

3. Insert sample data

In this tutorial you will learn to insert sample data to Percona Server for MySQL.

We will enter SQL statements via the same MySQL shell we used to connect to the database.

1. Let's create a separate database for our experiments:

```
CREATE DATABASE mydb;
use mydb;
```

Output

```
Query OK, 1 row affected (0.01 sec)
Database changed
```

2. Now let's create a table which we will later fill with some sample data:

```
CREATE TABLE extraordinary_gentlemen (
    id int NOT NULL AUTO_INCREMENT,
    name varchar(255) NOT NULL,
    occupation varchar(255),
    PRIMARY KEY (id)
);
```

Output

```
Query OK, 0 rows affected (0.04 sec)
```

3. Adding data to the newly created table will look as follows:

```
INSERT INTO extraordinary_gentlemen (name, occupation)
  VALUES
  ("Allan Quartermain","hunter"),
  ("Nemo","fish"),
  ("Dorian Gray", NULL),
  ("Tom Sawyer", "secret service agent");
```

Output

```
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

4. Query the collection to verify the data insertion

```
SELECT *
FROM extraordinary_gentlemen;
```

Output

```
+----+------------------+----------------------+
| id | name             | occupation           |
+----+------------------+----------------------+
|  1 | Allan Quartermain | hunter               |
|  2 | Nemo             | fish                 |
|  3 | Dorian Gray      | NULL                 |
|  4 | Tom Sawyer       | secret service agent |
+----+------------------+----------------------+
```

5. Updating data in the database would be not much more difficult:

```
UPDATE  extraordinary_gentlemen
   SET occupation = "submariner"
WHERE name = "Nemo";
```

Output

```
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

6. Now if you repeat the SQL statement from step 4, you will see the changes take effect:

```
SELECT *
FROM extraordinary_gentlemen;
```

Output

```
+----+-------------------+----------------------+
| id | name              | occupation           |
+----+-------------------+----------------------+
|  1 | Allan Quartermain | hunter               |
|  2 | Nemo              | submariner           |
|  3 | Dorian Gray       | NULL                 |
|  4 | Tom Sawyer        | secret service agent |
+----+-------------------+----------------------+
```

Next steps

Make a backup →

4. Make a backup

In this tutorial, you will learn how to make a logical backup of your data manually. To learn more about backups, see the Backup and restore section.

Considerations and prerequisites

In this tutorial, we use the AWS S3 as the backup storage. You need the following S3-related information:

- the name of the S3 storage
- the name of the S3 bucket
- the region - the location of the bucket
- the S3 credentials to be used 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.

Also, we will use some files from the Operator repository for setting up backups. So, clone the percona-xtradb-cluster-operator repository:

```
git clone -b v1.19.0 git@github.com:percona/percona-xtradb-cluster-operator.git
$ cd percona-xtradb-cluster-operator
```

Note

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

Configure backup storage

1. Encode S3 credentials, substituting `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` with your real values:

   on Linux

   ```
   $ echo -n 'AWS_ACCESS_KEY_ID' | base64 --wrap=0
   $ echo -n 'AWS_SECRET_ACCESS_KEY' | base64 --wrap=0
   ```

   on MacOS

   ```
   $ echo -n 'AWS_ACCESS_KEY_ID' | base64
   $ echo -n 'AWS_SECRET_ACCESS_KEY' | base64
   ```

2. Edit the `deploy/backup-secret-s3.yaml` example Secrets configuration file and specify the following:

   - the `metadata.name` key is the name which you use to refer your Kubernetes Secret
   - the base64-encoded S3 credentials

   deploy/backup/backup-secret-s3.yaml

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: my-cluster-name-backup-s3
   type: Opaque
   data:
     AWS_ACCESS_KEY_ID: <YOUR_AWS_ACCESS_KEY_ID>
     AWS_SECRET_ACCESS_KEY: <YOUR_AWS_SECRET_ACCESS_KEY>
   ```

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

   ```
   $ kubectl apply -f deploy/backup/backup-secret-s3.yaml -n <namespace>
   ```

4. Update your `deploy/cr.yaml` configuration. Specify the following parameters in the `backup` section:

   - set the `storages.<NAME>.type` to `s3`. Substitute the `<NAME>` part with some arbitrary name that you will later use to refer this storage when making

backups and restores.

- set the `storages.<NAME>.s3.credentialsSecret` to the name you used to refer your Kubernetes Secret (`my-cluster-name-backup-s3` in the previous step).
- specify the S3 bucket name for the `storages.<NAME>.s3.bucket` option
- specify the region in the `storages.<NAME>.s3.region` option.

```
...
backup:
  ...
  storages:
    s3-us-west:
      type: s3
      s3:
        bucket: "S3-BACKUP-BUCKET-NAME-HERE"
        region: "<AWS_S3_REGION>"
        credentialsSecret: my-cluster-name-backup-s3
  ...
```

If you use a different S3-compatible storage instead of AWS S3, add the `endpointURL` key in the `s3` subsection, which should point to the actual cloud used for backups. This value is specific to the cloud provider. For example, using Google Cloud involves the following `endpointUrl`:

```
endpointUrl: https://storage.googleapis.com
```

5. Apply the configuration. Specify your namespace instead of the `<namespace>` placeholder:

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

Make a physical backup

Now when your have the configured storage in your Custom Resource, you can make your first backup.

1. To make a backup, you need the configuration file. Edit the sample `deploy/backup/backup.yaml` configuration file and specify the following:
   - `metadata.name` - specify the backup name. You will use this name to restore from this backup
   - `spec.pxcCluster` - specify the name of your cluster. This is the name you specified when deploying Percona XtraDB Cluster.
   - `spec.storageName` - specify the name of your already configured storage.

deploy/backup/backup.yaml

```
apiVersion: pxc.percona.com/v1
kind: PerconaXtraDBClusterBackup
metadata:
  finalizers:
    - delete-s3-backup
  name: backup1
spec:
  pxcCluster: cluster1
  clusterName: my-cluster-name
  storageName: s3-us-west
```

2. Apply the configuration. This instructs the Operator to start a backup. Specify your namespace instead of the `<namespace>` placeholder:

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

3. Track the backup progress.

```
kubectl get pxc-backup -n <namespace>
```

Output

```
NAME      CLUSTER       STORAGE      DESTINATION                                    STATUS    COMPLETED   AGE
backup1   cluster1      s3-us-west   s3://pxc-my-bucket/2023-10-10T16:36:46Z   Running               43s
```

When the status changes to `Succeeded`, backup is made.

Troubleshooting

You may face issues with the backup. To identify the issue, you can do the following:

- View the information about the backup with the following command:

```
$ kubectl get pxc-backup <backup-name> -n <namespace> -o yaml
```

- View the backup-agent logs. Use the previous command to find the name of the pod where the backup was made:

```
$ kubectl logs pod/<pod-name> -c xtrabackup -n <namespace>
```

Congratulations! You have made the first backup manually. Want to learn more about backups? See the Backup and restore section for how to configure point-in-time recovery, and how to automatically make backups according to the schedule.

Next steps

Monitor the database →

5. Monitor database with Percona Monitoring and Management (PMM)

The Operator integrates natively with Percona Monitoring and Management (PMM) for comprehensive database monitoring. While custom monitoring solutions solutions are also supported, they require manual setup and are not automated by the Operator.

The Operator is compatible with both PMM versions 2 and 3. We recommend using the latest PMM version 3 for optimal monitoring capabilities.

In this section, you'll learn how to monitor Percona XtraDB Cluster using PMM.

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.

Considerations

1. If you are using PMM server version 2, use a PMM client image compatible with PMM 2. If you are using PMM server version 3, use a PMM client image compatible with PMM 3. Check Percona certified images for the right one.
2. If you specified both authentication methods for PMM server configuration and they have non-empty values, priority goes to PMM 3.
3. For migration from PMM2 to PMM3, see PMM upgrade documentation. Also check the Automatic migration of API keys page.

Install PMM Server

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

Install PMM Client

PMM Client is installed as a side-car container in the database, HAProxy and ProxySQL Pods in your Kubernetes-based environment. To install PMM Client, do the following:

Configure authentication

PMM3

PMM3 uses Grafana service accounts to control access to PMM server components and resources. To authenticate in PMM server, you need a service account token. Generate a service account and token. Specify the Admin role for the service account.

Warning

When you create a service account token, you can select its lifetime: it can be either a permanent token that never expires or the one with the expiration date. PMM server cannot rotate service account tokens after they expire. So you must take care of reconfiguring PMM Client in this case.

PMM2

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.

From PMM UI

Generate the PMM API key

From command line

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)
```

Warning

The API key is not rotated.

Create a secret

Now you must pass the credentials to the Operator. To do so, create a Secret object.

Warning

PMM3 Client only works with PMM Server tokens. Make sure to use the `pmmservertoken` option for PMM3 Client. Otherwise you may face issues with the setup.

1. Create a Secret configuration file. You can use the deploy/secrets.yaml secrets file.

PMM 3

Specify the service account token as the `pmmservertoken` value in the secrets file:

```
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-secrets
type: Opaque
stringData:
  pmmservertoken: ""
```

PMM 2

Specify the API key as the `pmmserverkey` value in the secrets file:

```
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-secrets
type: Opaque
stringData:
  pmmserverkey: ""
```

2. Create the Secrets object using the `deploy/secrets.yaml` file. Replace the `<namespace>` placeholder with your value.

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

Expected output

```
secret/cluster1-secrets created
```

Deploy a PMM Client

1. Update the `pmm` section in the deploy/cr.yaml file.
   - Set `pmm.enabled` = `true`.
   - Specify the PMM Client image path. Check Percona certified images for the required one.
   - 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:2.44.1-1
    serverHost: monitoring-service
```

2. Update the cluster. Replace the `<namespace>` placeholder with your value.

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

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

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

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 Object 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 service account token in the `cluster1-secrets` object, use the following command replacing the placeholders in `<>` with your values:

Linux

```
$ kubectl patch secret/cluster1-secrets -p '{"data":{"pmmservertoken": '$(echo -n <new-token> | base64 --wrap=0)'}}'
```

macOS

```
$ kubectl patch secret/cluster1-secrets -p '{"data":{"pmmservertoken": '$(echo -n <new-token> | base64)'}}'
```

Check the metrics

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

Now you can access PMM via *https* in a web browser, with the login/password authentication, and the browser is configured to show Percona XtraDB Cluster metrics.

Next steps

What's next →

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
- You can also try operating the Operator and database clusters via the web interface with Percona Everest - an open-source web-based database provisioning tool based on Percona Operators. See Get started with Percona Everest on how to start using it

Installation

System requirements

The Operator was developed and tested with Percona XtraDB Cluster versions 8.4.7-7.1 (Tech preview), 8.0.44-35.1, and 5.7.44-31.65.

Other options may also work but have not been tested.

Supported platforms

The following platforms were tested and are officially supported by the Operator 1.19.0:

- Google Kubernetes Engine (GKE) 1.31 - 1.33
- Amazon Elastic Container Service for Kubernetes (EKS) 1.32 - 1.34
- Azure Kubernetes Service (AKS) 1.32 - 1.34
- OpenShift 4.17 - 4.20
- Minikube 1.37.0 based on Kubernetes 1.34.0

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

Resource limits

A cluster running an officially supported platform contains at least three Nodes, with the following resources:

- 2GB of RAM,
- 2 CPU threads per Node for Pods provisioning,
- at least 60GB of available storage for Persistent Volumes provisioning.

Installation guidelines

Choose how you wish to install the Operator:

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

Install Percona XtraDB Cluster on Minikube

Installing the Percona Operator for MySQL based on Percona XtraDB Cluster 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 the Kubernetes application locally prior to deploying it on a cloud.

The following steps are needed to run the Operator and Percona XtraDB Cluster on 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.

    After the installation, run `minikube start --memory=4096 --cpus=3` (parameters increase the virtual machine limits for the CPU cores and memory, to ensure stable work of the Operator). Being executed, this command will download needed virtualized images, then initialize and run the cluster.

2. Deploy the operator with the following command:

    ```
    $ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-operator/v1.19.0/deploy/bundle.yaml
    ```

3. Deploy Percona XtraDB Cluster:

    ```
    $ kubectl apply -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-operator/v1.19.0/deploy/cr-minimal.yaml
    ```

    Note

    This deploys one Percona XtraDB Cluster node and one HAProxy node. The deploy/cr-minimal.yaml is for minimal non-production deployment. For more configuration options please see deploy/cr.yaml and Custom Resource Options. You can clone the repository with all manifests and source code by executing the following command:

    ```
    $ git clone -b v1.19.0 https://github.com/percona/percona-xtradb-cluster-operator
    ```

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

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

    Creation process will 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 pxc
    ```

    Expected output

    ```
    NAME              ENDPOINT                          STATUS   PXC   PROXYSQL   HAPROXY   AGE
    minimal-cluster   minimal-cluster-haproxy.default   ready    3                3         5m51s
    ```

Verifying the cluster operation

It may take ten minutes to get the cluster started. When the `kubectl get pxc` command output shows you the cluster status as `ready`, you can try to connect to the cluster.

To connect to Percona XtraDB Cluster you will need the password for the root user. Passwords are stored in the Secrets object.

Here's how to get it:

1. List the Secrets objects.

    ```
    $ kubectl get secrets
    ```

    The Secrets object you are interested in has the `minimal-cluster-secrets` name by default.

2. Use the following command to get the password of the `root` user. Substitute the `<namespace>` placeholder with your value (and use the different Secrets object name instead of the `minimal-cluster-secrets`, if needed):

```
$ kubectl get secret minimal-cluster-secrets -n <namespace> --template='{{.data.root | base64decode}}{{"\n"}}'
```

3. Run a container with `mysql` tool and connect its console output to your terminal. The following command does this, naming the new Pod `percona-client`:

```
$ kubectl run -n <namespace> -i --rm --tty percona-client --image=percona:8.4 --restart=Never -- bash -il
```

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

4. Now run the `mysql` tool in the `percona-client` command shell using the password obtained from the Secret instead of the `<root_password>` placeholder. The command will look different depending on whether your cluster provides load balancing with HAProxy (the default choice) or ProxySQL:

with HAProxy (default)

```
$ mysql -h minimal-cluster-haproxy -uroot -p'<root_password>'
```

with ProxySQL

```
$ mysql -h minimal-cluster-proxysql -uroot -p'<root_password>'
```

This command will connect you to the MySQL server.

Install Percona XtraDB Cluster using Everest

Percona Everest is an open source cloud-native database platform that helps developers deploy code faster, scale deployments rapidly, and reduce database administration overhead while regaining control over their data, database configuration, and DBaaS costs.

It automates day-one and day-two database operations for open source databases on Kubernetes clusters. Percona Everest provides API and Web GUI to launch databases with just a few clicks and scale them, do routine maintenance tasks, such as software updates, patch management, backups, and monitoring.

You can try it in action by Installing Percona Everest and managing your first cluster.

Install Percona XtraDB Cluster on Google Kubernetes Engine (GKE)

This quickstart shows you how to configure the Percona Operator for MySQL based on Percona XtraDB Cluster with the Google Kubernetes Engine. The document assumes some experience with Google Kubernetes Engine (GKE). For more information on the GKE, see the Kubernetes Engine Quickstart.

Prerequisites

All commands from this quickstart 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. It 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
```

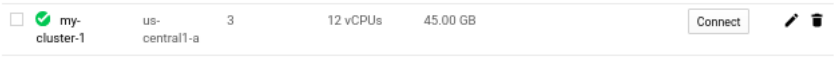
Configuring default settings for the cluster

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 will create a cluster named `my-cluster-1`:

```
$ gcloud container clusters create my-cluster-1 --project <project ID> --zone us-central1-a --cluster-version 1.33 --machine-type n1-standard-4 --num-nodes=3
```

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

You may wait a few minutes for the cluster to be generated, and then you will see it listed in the Google Cloud console (select *Kubernetes Engine* → *Clusters* in the left menu panel):



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 configures command-line access. After you have edited the statement, you may run the command in your local shell:

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

Installing the Operator

1. First of all, 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)
   ```

   The return statement confirms the creation:

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

2. Create a namespace and set the context for the namespace. The resource names must be unique within the namespace and provide a way to divide cluster resources between users spread across multiple projects.

   So, create the namespace and save it in the namespace context for subsequent commands as follows (replace the `<namespace name>` placeholder with some descriptive name):

   ```
   $ kubectl create namespace <namespace name>
   $ kubectl config set-context $(kubectl config current-context) --namespace=<namespace name>
   ```

   At success, you will see the message that namespace/ was created, and the context (gke_) was modified.

   Deploy the Operator using the following command:

   ```
   $ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-
   operator/v1.19.0/deploy/bundle.yaml
   ```

   Expected output

   ```
   customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusters.pxc.percona.com created
   customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusterbackups.pxc.percona.com created
   customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusterrestores.pxc.percona.com created
   customresourcedefinition.apiextensions.k8s.io/perconaxtradbbackups.pxc.percona.com created
   role.rbac.authorization.k8s.io/percona-xtradb-cluster-operator created
   serviceaccount/percona-xtradb-cluster-operator created
   rolebinding.rbac.authorization.k8s.io/service-account-percona-xtradb-cluster-operator created
   deployment.apps/percona-xtradb-cluster-operator created
   ```

3. The operator has been started, and you can deploy Percona XtraDB Cluster:

   ```
   $ kubectl apply -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-operator/v1.19.0/deploy/cr.yaml
   ```

   Expected output

   ```
   perconaxtradbcluster.pxc.percona.com/ cluster1 created
   ```

   Note

   This deploys default Percona XtraDB Cluster configuration with three HAProxy and three XtraDB Cluster instances. 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 v1.19.0 https://github.com/percona/percona-xtradb-cluster-operator
   ```

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

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

   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 pxc
   ```

   Expected output

   ```
   NAME       ENDPOINT                  STATUS   PXC   PROXYSQL   HAPROXY   AGE
   cluster1   cluster1-haproxy.default   ready    3               3         5m51s
   ```

Verifying the cluster operation

It may take ten minutes to get the cluster started. When `kubectl get pxc` command finally shows you the cluster status as `ready`, you can try to connect to the cluster.

To connect to Percona XtraDB Cluster you will need the password for the root user. Passwords are stored in the Secrets object.

Here's how to get it:

1. List the Secrets objects.

   ```
   $ kubectl get secrets
   ```

   The Secrets object you are interested in has the `cluster1-secrets` name by default.

2. Use the following command to get the password of the `root` user. Substitute the `<namespace>` placeholder with your value (and use the different Secrets object name instead of the `cluster1-secrets`, if needed):

   ```
   $ kubectl get secret cluster1-secrets -n <namespace> --template='{{.data.root | base64decode}}{{"\n"}}'
   ```

3. Run a container with `mysql` tool and connect its console output to your terminal. The following command does this, naming the new Pod `percona-client`:

   ```
   $ kubectl run -n <namespace> -i --rm --tty percona-client --image=percona:8.4 --restart=Never -- bash -il
   ```

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

4. Now run the `mysql` tool in the `percona-client` command shell using the password obtained from the Secret instead of the `<root_password>` placeholder. The command will look different depending on whether your cluster provides load balancing with HAProxy (the default choice) or ProxySQL:

   with HAProxy (default)

   ```
   $ mysql -h cluster1-haproxy -uroot -p'<root_password>'
   ```

   with ProxySQL

   ```
   $ mysql -h cluster1-proxysql -uroot -p'<root_password>'
   ```

   This command will connect you to the MySQL server.

Troubleshooting

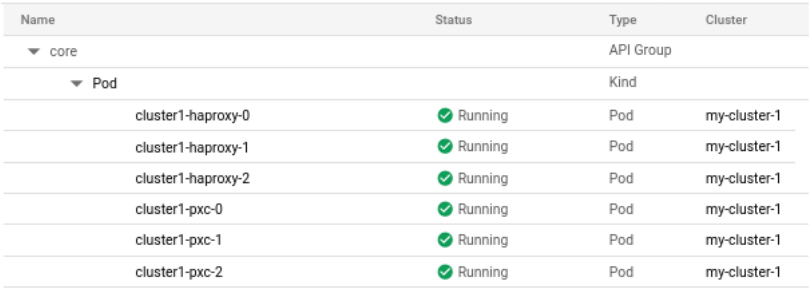
If `kubectl get pxc` command doesn't show `ready` status too long, you can check the creation process with the `kubectl get pods` command:

```
$ kubectl get pods
```

Expected output

```
NAME                                               READY   STATUS    RESTARTS   AGE
cluster1-haproxy-0                                 2/2     Running   0          6m17s
cluster1-haproxy-1                                 2/2     Running   0          4m59s
cluster1-haproxy-2                                 2/2     Running   0          4m36s
cluster1-pxc-0                                     3/3     Running   0          6m17s
cluster1-pxc-1                                     3/3     Running   0          5m3s
cluster1-pxc-2                                     3/3     Running   0          3m56s
percona-xtradb-cluster-operator-79966668bd-rswbk   1/1     Running   0          9m54s
```

Also, you can see the same information when browsing Pods of your cluster in Google Cloud console via the *Object Browser*:



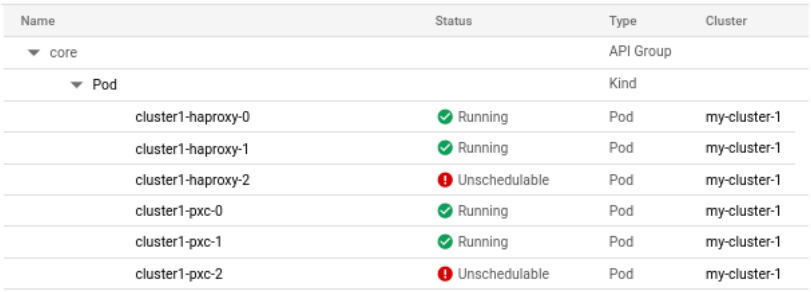
If the command output had shown some errors, you can examine the problematic Pod with the `kubectl describe <pod name>` command as follows:

```
$ kubectl describe pod  cluster1-pxc-2
```

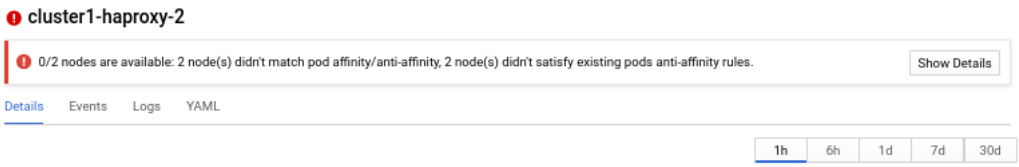
Review the detailed information for `Warning` statements and then correct the configuration. An example of a warning is as follows:

```
Warning  FailedScheduling  68s (x4 over 2m22s)  default-scheduler  0/1 nodes are available: 1 node(s) didn't match pod affinity/anti-affinity, 1 node(s) didn't satisfy existing pods anti-affinity rules.
```

Alternatively, you can examine your Pods via the *object browser*. Errors will look as follows:



Clicking the problematic Pod will bring you to the details page with the same warning:



Removing the GKE cluster

There are several ways that you can delete the cluster.

You can clean up the cluster with the `gcloud container clusters delete <cluster name> --zone <zone location>` command. The return statement requests your confirmation of the deletion. Type `y` to confirm.

```
$ gcloud container clusters delete my-cluster-1 --zone us-central1-a --project <project ID>
```

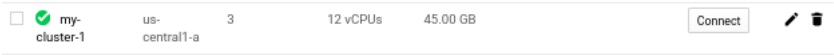
Expected output

```
The following clusters will be deleted.
 - [my-cluster-1] in [us-central1-a]

Do you want to continue (Y/n)?  y

Deleting cluster my-cluster-1...⠹
```

Also, you can delete your cluster via the GKE console. Just click the appropriate trashcan icon in the clusters list:



The cluster deletion may take time.

Install Percona XtraDB Cluster on Amazon Elastic Kubernetes Service (EKS)

This quickstart shows you how to deploy the Operator and Percona XtraDB Cluster on Amazon Elastic Kubernetes Service (EKS). The document assumes some experience with Amazon EKS. For more information on the EKS, see the Amazon EKS official documentation.

Prerequisites

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.

Create 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.

Install the Operator

1. Create a namespace and set the context for the namespace. The resource names must be unique within the namespace and provide a way to divide cluster resources between users spread across multiple projects.

   So, create the namespace and save it in the namespace context for subsequent commands as follows (replace the `<namespace name>` placeholder with some descriptive name):

   ```
   $ kubectl create namespace <namespace name>
   $ kubectl config set-context $(kubectl config current-context) --namespace=<namespace name>
   ```

   At success, you will see the message that namespace/ was created, and the context was modified.

   Deploy the Operator using the following command:

   ```
   $ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-
   operator/v1.19.0/deploy/bundle.yaml
   ```

Expected output

```
customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusters.pxc.percona.com created
customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusterbackups.pxc.percona.com created
customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusterrestores.pxc.percona.com created
customresourcedefinition.apiextensions.k8s.io/perconaxtradbbackups.pxc.percona.com created
role.rbac.authorization.k8s.io/percona-xtradb-cluster-operator created
serviceaccount/percona-xtradb-cluster-operator created
rolebinding.rbac.authorization.k8s.io/service-account-percona-xtradb-cluster-operator created
deployment.apps/percona-xtradb-cluster-operator created
```

2. The operator has been started, and you can deploy Percona XtraDB Cluster:

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-operator/v1.19.0/deploy/cr.yaml
```

Expected output

```
perconaxtradbcluster.pxc.percona.com/ cluster1 created
```

Note

This deploys default Percona XtraDB Cluster configuration with three HAProxy and three XtraDB Cluster instances. 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 v1.19.0 https://github.com/percona/percona-xtradb-cluster-operator
```

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

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

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 pxc
```

Expected output

```
NAME       ENDPOINT                  STATUS   PXC   PROXYSQL   HAPROXY   AGE
cluster1   cluster1-haproxy.default   ready    3               3         5m51s
```

Verifying the cluster operation

It may take ten minutes to get the cluster started. When `kubectl get pxc` command finally shows you the cluster status as `ready`, you can try to connect to the cluster.

To connect to Percona XtraDB Cluster you will need the password for the root user. Passwords are stored in the Secrets object.

Here's how to get it:

1. List the Secrets objects.

```
$ kubectl get secrets
```

The Secrets object you are interested in has the `cluster1-secrets` name by default.

2. Use the following command to get the password of the `root` user. Substitute the `<namespace>` placeholder with your value (and use the different Secrets object name instead of the `cluster1-secrets`, if needed):

```
$ kubectl get secret cluster1-secrets -n <namespace> --template='{{.data.root | base64decode}}{{"\n"}}'
```

3. Run a container with `mysql` tool and connect its console output to your terminal. The following command does this, naming the new Pod `percona-client`:

```
$ kubectl run -n <namespace> -i --rm --tty percona-client --image=percona:8.4 --restart=Never -- bash -il
```

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

4. Now run the `mysql` tool in the `percona-client` command shell using the password obtained from the Secret instead of the `<root_password>` placeholder. The command will look different depending on whether your cluster provides load balancing with HAProxy (the default choice) or ProxySQL:

   with HAProxy (default)

   ```
   $ mysql -h cluster1-haproxy -uroot -p'<root_password>'
   ```

   with ProxySQL

   ```
   $ mysql -h cluster1-proxysql -uroot -p'<root_password>'
   ```

   This command will connect you to the MySQL server.

Troubleshooting

If `kubectl get pxc` command doesn't show `ready` status too long, you can check the creation process with the `kubectl get pods` command:

```
$ kubectl get pods
```

Expected output

```
NAME                                               READY   STATUS    RESTARTS   AGE
cluster1-haproxy-0                                 2/2     Running   0          6m17s
cluster1-haproxy-1                                 2/2     Running   0          4m59s
cluster1-haproxy-2                                 2/2     Running   0          4m36s
cluster1-pxc-0                                     3/3     Running   0          6m17s
cluster1-pxc-1                                     3/3     Running   0          5m3s
cluster1-pxc-2                                     3/3     Running   0          3m56s
percona-xtradb-cluster-operator-79966668bd-rswbk   1/1     Running   0          9m54s
```

If the command output had shown some errors, you can examine the problematic Pod with the `kubectl describe <pod name>` command as follows:

```
$ kubectl describe pod  cluster1-pxc-2
```

Review the detailed information for `Warning` statements and then correct the configuration. An example of a warning is as follows:

```
Warning  FailedScheduling  68s (x4 over 2m22s)  default-scheduler  0/1 nodes are available: 1 node(s) didn't match pod affinity/anti-affinity, 1 node(s) didn't satisfy existing pods anti-affinity rules.
```

Install Percona XtraDB Cluster on Azure Kubernetes Service (AKS)

This guide shows you how to deploy Percona Operator for MySQL based on Percona XtraDB Cluster on Microsoft Azure Kubernetes Service (AKS). The document assumes some experience with the platform. For more information on the AKS, see the Microsoft AKS official documentation.

Prerequisites

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

1. **Azure Command Line Interface (Azure CLI)** for interacting with the different parts of AKS. You can install it following the official installation instructions for your system.
2. **kubectl** to manage and deploy applications on Kubernetes. Install it following the official installation instructions.

Also, you need to sign in with Azure CLI using your credentials according to the official guide.

Create and configure the AKS cluster

To create your cluster, you will need the following data:

- name of your AKS cluster,
- an Azure resource group, in which resources of your cluster will be deployed and managed.
- the amount of nodes you would like tho have.

You can create your cluster via command line using `az aks create` command. The following command will create a 3-node cluster named `cluster1` within some already existing resource group named `my-resource-group`:

```
$ az aks create --resource-group my-resource-group --name  cluster1 --enable-managed-identity --node-count 3 --node-vm-size
Standard_B4ms --node-osdisk-size 30 --network-plugin kubenet  --generate-ssh-keys --outbound-type loadbalancer
```

Other parameters in the above example specify that we are creating a cluster with machine type of Standard_B4ms and OS disk size reduced to 30 GiB. You can see detailed information about cluster creation options in the AKS official documentation.

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

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

```
az aks get-credentials --resource-group my-resource-group --name  cluster1
```

Install the Operator and deploy your Percona XtraDB Cluster

1. Deploy the Operator. By default deployment will be done in the `default` namespace. If that's not the desired one, you can create a new namespace and/or set the context for the namespace as follows (replace the `<namespace name>` placeholder with some descriptive name):

   ```
   $ kubectl create namespace <namespace name>
   $ kubectl config set-context $(kubectl config current-context) --namespace=<namespace name>
   ```

   At success, you will see the message that `namespace/<namespace name>` was created, and the context (`<cluster name>`) was modified.

   Deploy the Operator using the following command:

   ```
   $ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-
   operator/v1.19.0/deploy/bundle.yaml
   ```

Expected output

```
customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusters.pxc.percona.com created
customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusterbackups.pxc.percona.com created
customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusterrestores.pxc.percona.com created
customresourcedefinition.apiextensions.k8s.io/perconaxtradbbackups.pxc.percona.com created
role.rbac.authorization.k8s.io/percona-xtradb-cluster-operator created
serviceaccount/percona-xtradb-cluster-operator created
rolebinding.rbac.authorization.k8s.io/service-account-percona-xtradb-cluster-operator created
deployment.apps/percona-xtradb-cluster-operator created
```

2. The operator has been started, and you can deploy Percona XtraDB Cluster:

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-operator/v1.19.0/deploy/cr.yaml
```

Expected output

```
perconaxtradbcluster.pxc.percona.com/ cluster1 created
```

Note

This deploys default Percona XtraDB Cluster configuration with three HAProxy and three XtraDB Cluster instances. 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 v1.19.0 https://github.com/percona/percona-xtradb-cluster-operator
```

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

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

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 pxc
```

Expected output

```
NAME       ENDPOINT                  STATUS   PXC   PROXYSQL   HAPROXY   AGE
cluster1   cluster1-haproxy.default   ready    3               3         5m51s
```

Verifying the cluster operation

It may take ten minutes to get the cluster started. When `kubectl get pxc` command finally shows you the cluster status as `ready`, you can try to connect to the cluster.

To connect to Percona XtraDB Cluster you will need the password for the root user. Passwords are stored in the Secrets object.

Here's how to get it:

1. List the Secrets objects.

```
$ kubectl get secrets
```

The Secrets object you are interested in has the `cluster1-secrets` name by default.

2. Use the following command to get the password of the `root` user. Substitute the `<namespace>` placeholder with your value (and use the different Secrets object name instead of the `cluster1-secrets`, if needed):

```
$ kubectl get secret cluster1-secrets -n <namespace> --template='{{.data.root | base64decode}}{{"\n"}}'
```

3. Run a container with `mysql` tool and connect its console output to your terminal. The following command does this, naming the new Pod `percona-client`:

```
$ kubectl run -n <namespace> -i --rm --tty percona-client --image=percona:8.4 --restart=Never -- bash -il
```

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

4. Now run the `mysql` tool in the `percona-client` command shell using the password obtained from the Secret instead of the `<root_password>` placeholder. The command will look different depending on whether your cluster provides load balancing with HAProxy (the default choice) or ProxySQL:

   with HAProxy (default)

   ```
   $ mysql -h cluster1-haproxy -uroot -p'<root_password>'
   ```

   with ProxySQL

   ```
   $ mysql -h cluster1-proxysql -uroot -p'<root_password>'
   ```

   This command will connect you to the MySQL server.

Troubleshooting

If `kubectl get pxc` command doesn't show `ready` status too long, you can check the creation process with the `kubectl get pods` command:

```
$ kubectl get pods
```

Expected output

```
NAME                                               READY   STATUS    RESTARTS   AGE
cluster1-haproxy-0                                 2/2     Running   0          6m17s
cluster1-haproxy-1                                 2/2     Running   0          4m59s
cluster1-haproxy-2                                 2/2     Running   0          4m36s
cluster1-pxc-0                                     3/3     Running   0          6m17s
cluster1-pxc-1                                     3/3     Running   0          5m3s
cluster1-pxc-2                                     3/3     Running   0          3m56s
percona-xtradb-cluster-operator-79966668bd-rswbk   1/1     Running   0          9m54s
```

If the command output had shown some errors, you can examine the problematic Pod with the `kubectl describe <pod name>` command as follows:

```
$ kubectl describe pod  cluster1-pxc-2
```

Review the detailed information for `Warning` statements and then correct the configuration. An example of a warning is as follows:

```
Warning  FailedScheduling  68s (x4 over 2m22s)  default-scheduler  0/1 nodes are available: 1 node(s) didn't match pod affinity/anti-affinity, 1 node(s) didn't satisfy existing pods anti-affinity rules.
```

Removing the AKS cluster

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

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

You can clean up the cluster with the `az aks delete` command as follows (with real names instead of `<resource group>` and `<cluster name>` placeholders):

```
$ az aks delete --name <cluster name> --resource-group <resource group> --yes --no-wait
```

It may take ten minutes to get the cluster actually deleted after executing this command.

Warning

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

Install Percona XtraDB Cluster on OpenShift

Percona Operator for Percona XtrabDB Cluster 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 XtraDB Cluster on OpenShift includes two steps:

- Installing the Percona Operator for MySQL,
- Install Percona XtraDB Cluster using the Operator.

Install the Operator

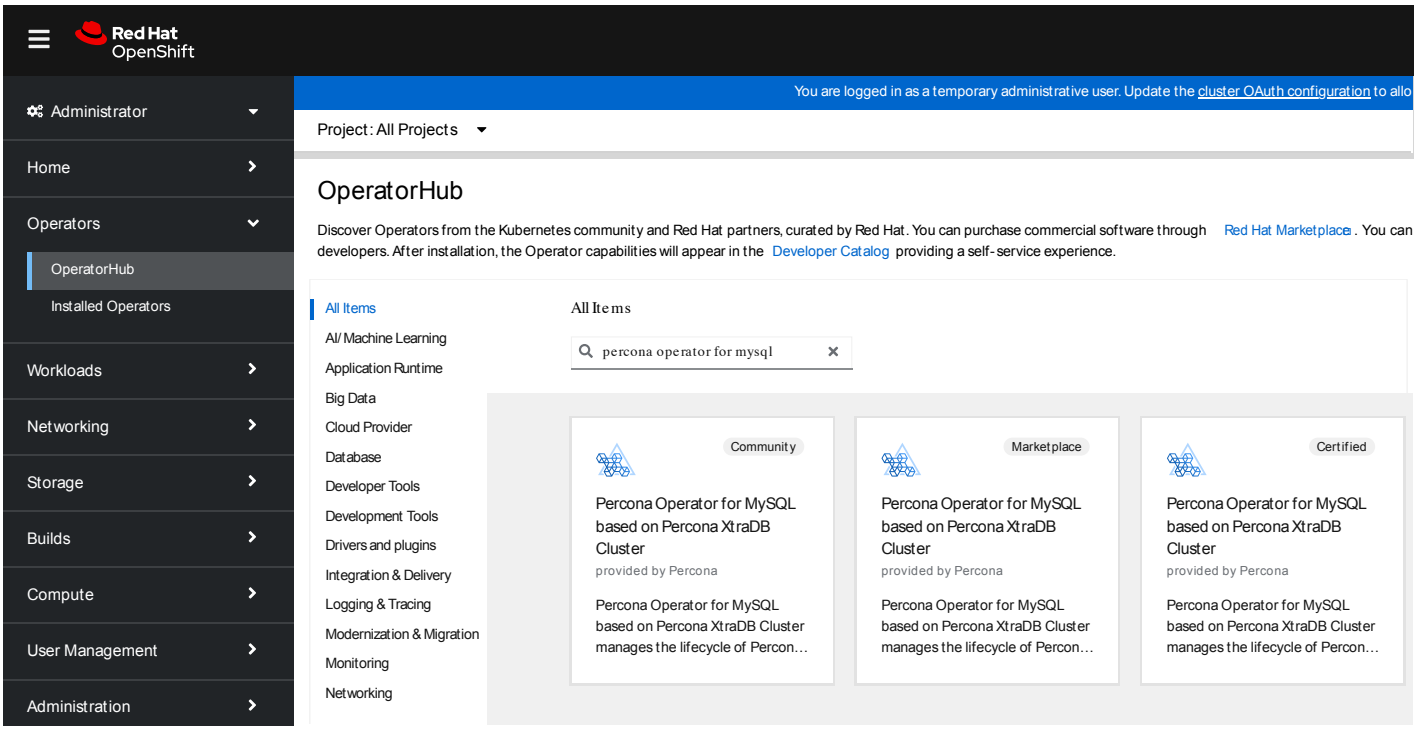
You can install Percona Operator for MySQL on OpenShift using the web interface (the Operator Lifecycle Manager), or using the command line interface.

Install the Operator via the Operator Lifecycle Manager (OLM)

Operator Lifecycle Manager (OLM) is a part of the Operator Framework that allows you to install, update, and manage the Operators lifecycle on the OpenShift platform.

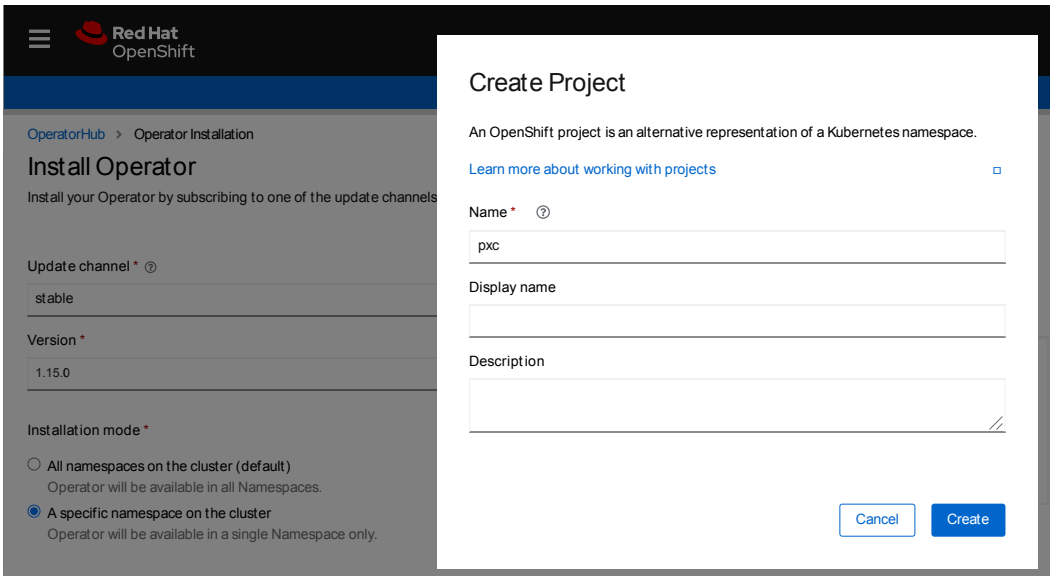
Following steps will allow you to deploy the Operator and Percona XtraDB Cluster on your OLM installation:

1. Login to the OLM and click the needed Operator on the OperatorHub page:



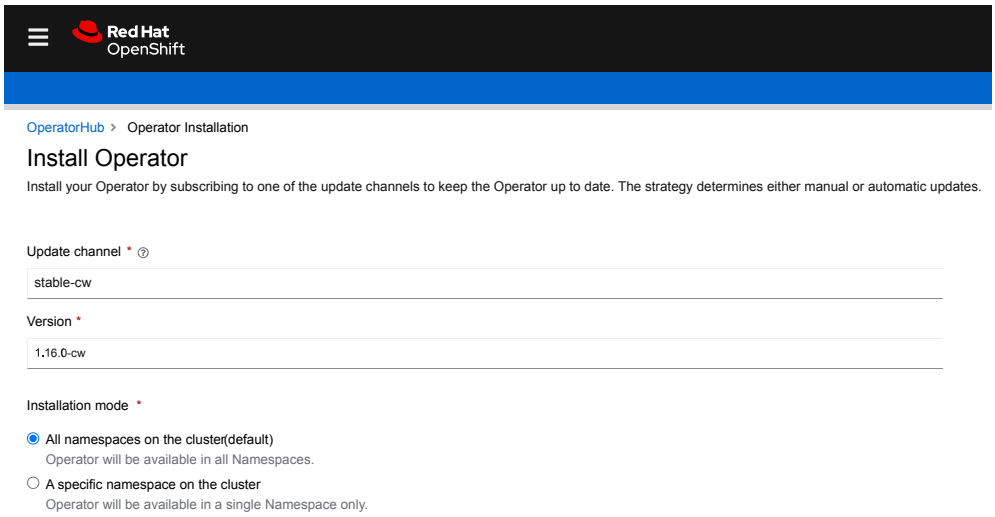
   Then click “Continue”, and “Install”.

2. A new page will allow you to choose the Operator version and the Namespace / OpenShift project you would like to install the Operator into.



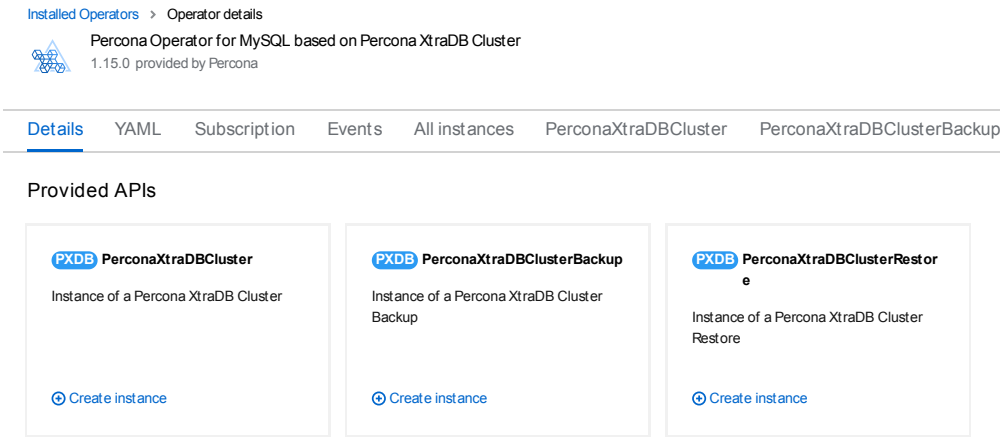
Note

To install the Operator in multi-namespace (cluster-wide) mode, use the one from the certified catalog. It has the Certified label. Choose values with `-cw` suffix for the update channel and version, and select the “All namespaces on the cluster” radio button for the installation mode instead of choosing a specific Namespace:



Click “Install” to install the Operator.

3. When the installation finishes, you can deploy Percona XtraDB Cluster. In the “Operator Details” you will see Provided APIs (Custom Resources, available for installation). Click “Create instance” for the `PerconaXtraDBCluster` Custom Resource.



You will be able to edit manifest to set needed Custom Resource options, and then click “Create” button to deploy your database cluster.

Install the Operator via the command-line interface

Single action installation

For a quick and streamlined installation, you can use the `deploy/bundle.yaml` file. Applying this single file will automatically create the Custom Resource Definition,

The steps are the following:

1. Clone the `percona-xtradb-cluster-operator` repository. Pay attention to specify the right branch with the -b option while cloning the code on this step:

```
$ git clone -b v1.19.0 https://github.com/percona/percona-xtradb-cluster-operator
$ cd percona-xtradb-cluster-operator
```

2. **For OpenShift 4.19**. Edit the `deploy/bundle.yaml` file.

   - Locate the Deployment custom resource for the Operator.
   - Update the `spec.image` field to

```
docker.io/percona/percona-xtradb-cluster-operator:1.19.0
```

3. Create the namespace

```
$ oc new-project pxc
```

4. Create the Custom Resource Definition, RBAC (role-based access control) and the Operator deployment.

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

Step-by-step installation

If you prefer to install each component manually, follow these steps:

1. Clone the repository. Pay attention to specify the right branch with the -b option while cloning the code on this step:

```
$ git clone -b v1.19.0 https://github.com/percona/percona-xtradb-cluster-operator
$ cd percona-xtradb-cluster-operator
```

2. Create the Custom Resource Definition (CRD)

   The CRD extends Kubernetes with new resource types required by the operator. This step only needs to be done once.

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

   Note

   Setting the Custom Resource Definition requires your user to have cluster-admin privileges.

   If you want to manage your Percona XtraDB Cluster with a non-privileged user, grant the necessary permissions by applying the following cluster role:

```
$ oc create clusterrole pxc-admin --verb="*" --
resource=perconaxtradbclusters.pxc.percona.com,perconaxtradbclusters.pxc.percona.com/status,perconaxtradbclusterbackups.pxc.perco
$ oc adm policy add-cluster-role-to-user pxc-admin <some-user>
```

   If you have cert-manager installed, run these commands to manage certificates with a non-privileged user:

```
$ oc create clusterrole cert-admin --verb="*" --resource=issuers.certmanager.k8s.io,certificates.certmanager.k8s.io
$ oc adm policy add-cluster-role-to-user cert-admin <some-user>
```

3. Create a new project for the cluster

```
$ oc new-project pxc
```

4. Set up RBAC (Role-Based Access Control)

   Apply the RBAC configuration to define roles and permissions for the operator:

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

5. **For OpenShift 4.19** Edit the `deploy/operator.yaml` and update the `spec.image` field to `docker.io/percona/percona-xtradb-cluster-operator:1.19.0`:

```
spec:
 containers:
    - command:
      ...
      image: docker.io/percona/percona-xtradb-cluster-operator:1.19.0
```

6. Deploy the Operator

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

For more details about users and roles, see the OpenShift documentation.

Install Percona XtraDB Cluster

1. Now that's time to add the Percona XtraDB Cluster users Secrets with logins and passwords to Kubernetes. By default, the Operator generates users Secrets automatically, and *no actions are required at this step*.

   Still, you can generate and apply your Secrets by your own. In this case, place logins and plaintext passwords for the user accounts in the data section of the `deploy/secrets.yaml` file; after editing is finished, create users Secrets with the following command:

```
$ oc create -f deploy/secrets.yaml
```

   More details about secrets can be found in Users.

2. Now certificates should be generated. By default, the Operator generates certificates automatically, and no actions are required at this step. Still, you can generate and apply your own certificates as secrets according to the TLS instructions.
3. After the operator is started and user secrets are added, Percona XtraDB Cluster can be created at any time with the following command:

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

   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:

```
$ oc get pxc
```

Expected output

```
NAME       ENDPOINT                  STATUS   PXC   PROXYSQL   HAPROXY   AGE
cluster1   cluster1-haproxy.default   ready    3               3         5m51s
```

Verify the cluster operation

It may take ten minutes to get the cluster started. When the `oc get pxc` command output shows you the cluster status as `ready`, you can try to connect to the cluster.

To connect to Percona XtraDB Cluster you will need the password for the root user. Passwords are stored in the Secrets object.

Here's how to get it:

1. List the Secrets objects.

```
$ oc get secrets
```

The Secrets object you are interested in has the `cluster1-secrets` name by default.

2. Use the following command to get the password of the `root` user. Substitute the `<namespace>` placeholder with your value (and use the different Secrets object name instead of the `cluster1-secrets`, if needed):

```
$ oc get secret cluster1-secrets -n <namespace> --template='{{.data.root | base64decode}}{{"\n"}}'
```

3. Run a container with `mysql` tool and connect its console output to your terminal. The following command does this, naming the new Pod `percona-client`:

```
$ oc run -n <namespace> -i --rm --tty percona-client --image=percona:8.4 --restart=Never -- bash -il
```

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

4. Now run the `mysql` tool in the `percona-client` command shell using the password obtained from the Secret instead of the `<root_password>` placeholder. The command will look different depending on whether your cluster provides load balancing with HAProxy (the default choice) or ProxySQL:

with HAProxy (default)

```
$ mysql -h cluster1-haproxy -uroot -p'<root_password>'
```

with ProxySQL

```
$ mysql -h cluster1-proxysql -uroot -p'<root_password>'
```

This command will connect you to the MySQL server.

Install Percona XtraDB Cluster on Kubernetes

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

```
$ git clone -b v1.19.0 https://github.com/percona/percona-xtradb-cluster-operator
$ cd percona-xtradb-cluster-operator
```

Note

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

1. Now Custom Resource Definition for Percona XtraDB Cluster 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).

   This step should be done only once; it does not need to be repeated with the next Operator deployments, etc.

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

2. The next thing to do is to add the `pxc` namespace to Kubernetes, not forgetting to set the correspondent context for further steps:

```
$ kubectl create namespace pxc
$ kubectl config set-context $(kubectl config current-context) --namespace=pxc
```

3. Now RBAC (role-based access control) for Percona XtraDB Cluster should be set up from the `deploy/rbac.yaml` file. Briefly speaking, role-based access is based on specifically defined roles and actions corresponding to them, allowed to be done on specific Kubernetes resources (details about users and roles can be found in Kubernetes documentation).

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

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)`

Finally it's time to start the operator within Kubernetes:

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

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:

```
$ kubectl apply --server-side -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.

4. Now that's time to add the Percona XtraDB Cluster users Secrets with logins and passwords to Kubernetes. By default, the Operator generates users Secrets automatically, and *no actions are required at this step*.

   Still, you can generate and apply your Secrets on your own. In this case, place logins and plaintext passwords for the user accounts in the data section of the `deploy/secrets.yaml` file; after editing is finished, create users Secrets with the following command:

```
$ kubectl create -f deploy/secrets.yaml
```

   More details about secrets can be found in Users.

5. Now certificates should be generated. By default, the Operator generates certificates automatically, and *no actions are required at this step*. Still, you can generate and apply your own certificates as secrets according to the TLS instructions.
6. After the operator is started and user secrets are added, Percona XtraDB Cluster can be created at any time with the following command:

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

Creation process will 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 pxc
```

Expected output

```
NAME       ENDPOINT                  STATUS   PXC   PROXYSQL   HAPROXY   AGE
cluster1   cluster1-haproxy.default   ready    3               3         5m51s
```

Verify the cluster operation

It may take ten minutes to get the cluster started. When `kubectl get pxc` command finally shows you the cluster status as `ready`, you can try to connect to the cluster.

To connect to Percona XtraDB Cluster you will need the password for the root user. Passwords are stored in the Secrets object.

Here's how to get it:

1. List the Secrets objects.

   ```
   $ kubectl get secrets
   ```

   The Secrets object you are interested in has the `cluster1-secrets` name by default.

2. Use the following command to get the password of the `root` user. Substitute the `<namespace>` placeholder with your value (and use the different Secrets object name instead of the `cluster1-secrets`, if needed):

   ```
   $ kubectl get secret cluster1-secrets -n <namespace> --template='{{.data.root | base64decode}}{{"\n"}}'
   ```

3. Run a container with `mysql` tool and connect its console output to your terminal. The following command does this, naming the new Pod `percona-client`:

   ```
   $ kubectl run -n <namespace> -i --rm --tty percona-client --image=percona:8.4 --restart=Never -- bash -il
   ```

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

4. Now run the `mysql` tool in the `percona-client` command shell using the password obtained from the Secret instead of the `<root_password>` placeholder. The command will look different depending on whether your cluster provides load balancing with HAProxy (the default choice) or ProxySQL:

   with HAProxy (default)

   ```
   $ mysql -h cluster1-haproxy -uroot -p'<root_password>'
   ```

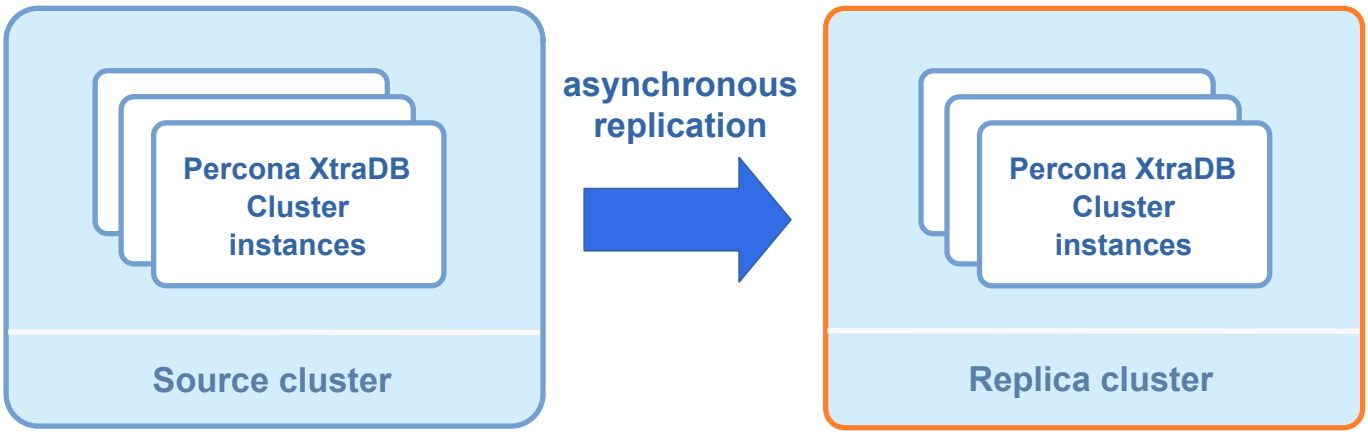
   with ProxySQL

   ```
   $ mysql -h cluster1-proxysql -uroot -p'<root_password>'
   ```

   This command will connect you to the MySQL server.

Set up Percona XtraDB Cluster cross-site replication

The cross-site replication involves configuring one Percona XtraDB Cluster as *Source*, and another Percona XtraDB Cluster as *Replica* to allow asynchronous replication between them:



The Operator automates configuration of *Source* and *Replica* Percona XtraDB Clusters, but the feature itself is not bound to Kubernetes. Either *Source* or *Replica* can run outside of Kubernetes, be regular MySQL instances and be out of the Operator's control.

This feature can be useful in several cases: for example, it can simplify migration from on-premises to the cloud with replication, and it can be really helpful in case of the disaster recovery too.

Note

Cross-site replication is based on Automatic Asynchronous Replication Connection Failover. Therefore it requires Percona XtraDB Cluster 8.0.22+ (MySQL 8.0.22+) to work.

Setting up MySQL for asynchronous replication without the Operator is out of the scope for this document, but it is described here and is also covered by this HowTo.

Configuring the cross-site replication for the cluster controlled by the Operator is explained in the following subsections.

Creating a Replica cluster

Cross-site replication can be configured on two sibling Percona XtraDB Clusters. That's why you should first make a fully operational clone of your main database cluster. After that your original cluster will be configured as *Source*, and a new one (the clone) will be configured as *Replica*.

The easiest way to achieve this is to use backups. You make a full backup of your main database cluster, and restore it to a new Kubernetes-based environment, following this HowTo.

Configuring cross-site replication on Source instances

You can configure *Source* instances for cross-site replication with `spec.pxc.replicationChannels` subsection in the `deploy/cr.yaml` configuration file. It is an array of channels, and you should provide the following keys for the channel in your *Source* Percona XtraDB Cluster:

- `pxc.replicationChannels.[].name` key is the name of the channel,
- `pxc.replicationChannels.[].isSource` key should be set to `true`.

Here is an example:

```
spec:
  pxc:
    replicationChannels:
    - name: pxc1_to_pxc2
      isSource: true
```

You will also need to expose every Percona XtraDB Cluster Pod of the *Source* cluster to make it possible for the *Replica* cluster to connect. This is done through the `pxc.expose` section in the `deploy/cr.yaml` configuration file as follows.

```
spec:
  pxc:
    expose:
      enabled: true
      type: LoadBalancer
```

> **Note**
>
> This will create a LoadBalancer per each Percona XtraDB Cluster Pod. In most cases, for cross-region replication to work this Load Balancer should be internet-facing.

The cluster will be ready for asynchronous replication when you apply changes as usual:

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

To list the endpoints assigned to PXC Pods list the Kubernetes Service objects by executing `kubectl get services -l "app.kubernetes.io/instance=cluster1"` command (don't forget to substitute `cluster1` with the real name of your cluster, if you don't use the default name).

Configuring cross-site replication on Replica instances

You can configure *Replica* instances for cross-site replication with `spec.pxc.replicationChannels` subsection in the `deploy/cr.yaml` configuration file. It is an array of channels, and you should provide the following keys for the channel in your *Replica* Percona XtraDB Cluster:

- `pxc.replicationChannels.[].name` key is the name of the channel,
- `pxc.replicationChannels.[].isSource` key should be set to `false`,
- `pxc.replicationChannels.[].sourcesList` is the list of *Source* cluster names from which Replica should get the data,
- `pxc.replicationChannels.[].sourcesList.[].host` is the host name or IP address of the Source,
- `pxc.replicationChannels.[].sourcesList.[].port` is the port of the source (`3306` port will be used if nothing specified),
- `pxc.replicationChannels.[].sourcesList.[].weight` is the *weight* of the source (in the event of a connection failure, a new source is selected from the list based on a weighted priority).

Here is the example:

```
spec:
  pxc:
    replicationChannels:
    - name: uspxc1_to_pxc2
      isSource: false
      sourcesList:
      - host: pxc1.source.percona.com
        port: 3306
        weight: 100
      - host: pxc2.source.percona.com
        weight: 100
      - host: pxc3.source.percona.com
        weight: 100
    - name: eu_to_pxc2
      isSource: false
      sourcesList:
      - host: pxc1.source.percona.com
        port: 3306
        weight: 100
      - host: pxc2.source.percona.com
        weight: 100
      - host: pxc3.source.percona.com
        weight: 100
```

The cluster will be ready for asynchronous replication when you apply changes as usual:

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

Note

You can also configure SSL channel for replication. Following options allow you using replication over an encrypted channel. Set the `replicationChannels.configuration.ssl` key to true, optionally enable host name identity verification with the `replicationChannels.configuration.sslSkipVerify` key, and set `replicationChannels.configuration.ca` key to the path name of the Certificate Authority (CA) certificate file:

```
replicationChannels:
- isSource: false
  name: uspxc1_to_pxc2
  configuration:
    ssl: true
    sslSkipVerify: true
    ca: '/etc/mysql/ssl/ca.crt'
    ...
```

SSL certificates on both sides should be signed by the same certificate authority for encrypted replication channels to work.

System user for replication

Replication channel demands a special system user with the same credentials on both *Source* and *Replica*.

The Operator creates a system-level Percona XtraDB Cluster user named `replication` for this purpose, with credentials stored in a Secret object along with other system users.

Note

If the Replica cluster is not a clone of the original one (for example, it's outside of Kubernetes and is not under the Operator's control) the appropriate user with necessary permissions should be created manually.

If you need you can change a password for this user as follows:

in Linux

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

in macOS

```
$ kubectl patch secret/cluster1-secrets -p '{"data":{"replication": "'$(echo -n new_password | base64)'"}}'
```

If you have changed the `replication` user's password on the Source cluster, you can have a *replication is not running* error message in log, similar to the following one:

```
{"level":"info","ts":1629715578.2569592,"caller":"zapr/zapr.go 69","msg":"Replication for channel is not running. Please, check
the replication status","channel":"pxc2_to_pxc1"}
```

To fix this error, do the following:

1. Find the Replica Pod which was chosen by the Operator for replication, using the following command:

   ```
   $ kubectl get pods --selector percona.com/replicationPod=true
   ```

2. Get the shell access to this Pod and login to the MySQL monitor as a root user:

   ```
   $ kubectl exec -c pxc --stdin --tty <pod_name> -- /bin/bash
   bash-4.4$ mysql -uroot -proot_password
   ```

3. Execute the following three SQL commands to propagate the `replication` user password from the Source cluster to Replica:

For Percona XtraDB Cluster 8.0.x and 8.4.x

```
STOP REPLICA IO_THREAD FOR CHANNEL '$REPLICATION_CHANNEL_NAME';
CHANGE REPLICATION SOURCE TO MASTER_PASSWORD='$NEW_REPLICATION_PASSWORD' FOR CHANNEL     '$REPLICATION_CHANNEL_NAME';
START REPLICA IO_THREAD FOR CHANNEL '$REPLICATION_CHANNEL_NAME';
```

For Percona XtraDB Cluster 5.7.x

```
STOP REPLICA IO_THREAD FOR CHANNEL '$REPLICATION_CHANNEL_NAME';
CHANGE MASTER TO MASTER_PASSWORD='$NEW_REPLICATION_PASSWORD' FOR CHANNEL     '$REPLICATION_CHANNEL_NAME';
START REPLICA IO_THREAD FOR CHANNEL '$REPLICATION_CHANNEL_NAME';
```

Upgrade

Update Percona Operator for MySQL based on Percona XtraDB Cluster

You can upgrade Percona Operator for MySQL based on Percona XtraDB Cluster to newer versions

The upgrade process consists of these steps:

- Upgrade the Operator
- Upgrade the database (Percona XtraDB Cluster).

Update scenarios

You can either upgrade both the Operator and the database, or you can upgrade only the database. To decide which scenario to choose, read on.

Full upgrade (CRD, Operator, and the database)

When to use this scenario:

- The new Operator version has changes that are required for new features of the database to work
- The Operator has new features or fixes that enhance automation and management.
- Compatibility improvements between the Operator and the database require synchronized updates.

When going on with this scenario, make sure to test it in a staging or testing environment first. Upgrading the Operator may cause performance degradation.

Upgrade only the database

When to use this scenario:

- The new version of the database has new features or fixes that are not related to the Operator or other components of your infrastructure
- You have updated the Operator earlier and now want to proceed with the database update.

When choosing this scenario, consider the following:

- Check that the current Operator version supports the new database version.
- Some features may require an Operator upgrade later for full functionality.

Update strategies

You can chose how you want to update your database cluster when you run an upgrade:

- *Smart Update* is the automated way to update the database cluster. The Operator controls how objects are updated. It restarts Pods in a specific order, with the primary instance updated last to avoid connection issues until the whole cluster is updated to the new settings.

  This update method applies during database upgrades and when making changes like updating a ConfigMap, rotating passwords, or changing resource values. It is the default and recommended way to update.
- *Rolling Update* is initiated manually and controlled by Kubernetes. The StatefulSet controller in Kubernetes deletes a Pod, updates it, waits till it reports the Ready status and proceeds to the next Pod. The order for Pod update is the same as for Pod termination. However, this order may not be optimal from the Percona Server for MongoDB point of view.
- *On Delete* strategy requires a user to manually delete a Pod to make Kubernetes StatefulSet controller recreate it with the updated configuration.

To select an update strategy, set the `updateStrategy` key in the Custom Resource manifest to one of the following:

- `SmartUpdate`
- `RollingUpdate`
- `OnDelete`

For a manual update of your database cluster using the `RollingUpdate` or `OnDelete` strategies, refer to the low-level Kubernetes way of database upgrades guide.

Update on OpenShift

If you run the Operator on Red Hat Marketplace or you run Red Hat certified Operators on OpenShift, you need to do additional steps during the upgrade. See this HOWTO for details.

Upgrade the Operator and CRD

To update the Operator, you need to update the Custom Resource Definition (CRD) and the Operator deployment. Also we recommend to update the Kubernetes database cluster configuration by updating the Custom Resource and the database components to the latest version. This step ensures that all new features that come with the Operator release work in your environment.

The database cluster upgrade process is similar for all installation methods, including Helm and OLM.

Considerations for Kubernetes Cluster versions and upgrades

1. Before upgrading the Kubernetes cluster, have a disaster recovery plan in place. Ensure that a backup is taken prior to the upgrade, and that point-in-time recovery is enabled to meet your Recovery Point Objective (RPO).
2. Plan your Kubernetes cluster or Operator upgrades with version compatibility in mind.

   The Operator is supported and tested on specific Kubernetes versions. Always refer to the Operator's release notes to verify the supported Kubernetes platforms.

   Note that while the Operator might run on unsupported or untested Kubernetes versions, this is not recommended. Doing so can cause various issues, and in some cases, the Operator may fail if deprecated API versions have been removed.
3. During a Kubernetes cluster upgrade, you must also upgrade the `kubelet`. It is advisable to drain the nodes hosting the database Pods during the upgrade process.
4. During the `kubelet` upgrade, nodes transition between `Ready` and `NotReady` states. Also in some scenarios, older nodes may be replaced entirely with new nodes. Ensure that nodes hosting database or proxy pods are functioning correctly and remain in a stable state after the upgrade.
5. Regardless of the upgrade approach, pods will be rescheduled or recycled. Plan your Kubernetes cluster upgrade accordingly to minimize downtime and service disruption.

Considerations for Operator upgrades

1. The Operator version has three digits separated by a dot (`.`) in the format `major.minor.patch`. Here's how you can understand the version `1.16.1`:
   - `1` is the major version
   - `16` is the minor version
   - `1` is the patch version.

   You can only upgrade the Operator to the nearest `major.minor` version (for example, from `1.15.1` to `1.16.1`).

   If the your current Operator version and the version you want to upgrade to differ by more than one minor version, you need to upgrade step by step. For example, if your current version is `1.14.x` and you want to move to `1.16.x`, first upgrade to `1.15.x`, then to `1.16.x`.

   Patch versions don't influence the upgrade, so you can safely move from `1.15.1` to `1.16.1`.

   Check the Release notes index for the list of the Operator versions.
2. CRD supports the **last 3 minor versions of the Operator**. This means it is compatible with the newest Operator version and the two older minor versions. If the Operator is older than the CRD *by no more than two versions*, you should be able to continue using the old Operator version. But updating the CRD *and* Operator is the **recommended path**.
3. Starting with version 1.12.0, the Operator no longer has a separate API version for each release in CRD. Instead, the CRD has the API version `v1`. Therefore, if you installed the CRD when the Operator version was **older than 1.12.0**, you must update the API version in the CRD manually to run the upgrade. To check your CRD version, use the following command:

   ```
   $ kubectl get crd perconaxtradbclusters.pxc.percona.com -o yaml | yq .status.storedVersions
   ```

   Sample output

   ```
   - v1-11-0
   - v1
   ```

   If the CRD version is other than `v1` or has multiple entries, run the manual update.
4. The Operator versions 1.14.0 and 1.15.0 **should be excluded** from the incremental upgrades sequence in favor of 1.14.1 and 1.15.1 releases.
   - The upgrade path from the version 1.14.1 should be 1.14.1 -> 1.15.1.
   - Direct upgrades from 1.13.0 to 1.14.1 and from 1.14.0 to 1.15.1 are supported.

5. To upgrade multiple single-namespace Operator deployments in one Kubernetes cluster, where each Operator controls a database cluster in its own namespace, do the following:
   - upgrade the CRD (not 3 minor versions far from the oldest Operator installation in the Kubernetes cluster) first
   - upgrade the Operators in each namespace incrementally to the latest minor version (e.g. from 1.15.1 to 1.16.1, then to 1.17.0)
6. Starting with version 1.18.0, the Operator supports PMM2 and PMM3. If you are using PMM server version 2, use a PMM client image compatible with PMM 2. If you are using PMM server version 3, use a PMM client image compatible with PMM 3. See PMM upgrade documentation for how to migrate from version 2 to version 3.

Upgrade manually

The upgrade includes the following steps.

1. **For Operators older than v1.12.0**: Update the API version in the Custom Resource Definition:

   Manually

   ```
   $ kubectl proxy &  \
   $ curl \
          --header "Content-Type: application/json-patch+json" \
          --request PATCH \
          --data '[{"op": "replace", "path": "/status/storedVersions", "value":["v1"]}]' \
          --url
   "http://localhost:8001/apis/apiextensions.k8s.io/v1/customresourcedefinitions/perconaxtradbclusters.pxc.percona.com/status"
   ```

   Expected output

   ```
   {
    {...},
     "status": {
       "storedVersions": [
         "v1"
       ]
     }
   }
   ```

   Via `kubectl patch`

   ```
   $ kubectl patch customresourcedefinitions perconaxtradbclusters.pxc.percona.com --subresource='status' --type='merge' -p
   '{"status":{"storedVersions":["v1"]}}'
   ```

   Expected output

   ```
   customresourcedefinition.apiextensions.k8s.io/perconaxtradbclusters.pxc.percona.com patched
   ```

2. Update the Custom Resource Definition for the Operator and the Role-based access control. Take the latest versions from the official repository on GitHub with the following commands:

   ```
   $ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-
   operator/v1.19.0/deploy/crd.yaml
   $ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-
   operator/v1.19.0/deploy/rbac.yaml
   ```

3. Next, update the Percona Server for MySQL Operator Deployment in Kubernetes by changing the container image of the Operator Pod to the latest version. Find the image name for the current Operator release in the list of certified images. Then apply a patch to the Operator Deployment and specify the image name and version. Use the following command to update the Operator to the `1.19.0` version:

   ```
   $ kubectl patch deployment percona-xtradb-cluster-operator \
     -p'{"spec":{"template":{"spec":{"containers":[{"name":"percona-xtradb-cluster-operator","image":"percona/percona-xtradb-
   cluster-operator:1.19.0"}]}}}}'
   ```

   For previous releases, please refer to the old releases documentation archive

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

Note

Labels set on the Operator Pod will not be updated during upgrade.

5. Update the Custom Resource, the database, backup, proxy and PMM Client image names with a newer version tag. This step ensures all new features and improvements of the latest release work well within your environment.

   Find the image names in the list of certified images.

   Check your custom HAProxy configuration **before** the upgrade to be compatible with the one available with the Operator version you're upgrading to. Find the `haproxy-global.cfg` for the Operator version 1.19.0 here. Adjust your configuration, if needed.

   We recommend to update the PMM Server **before** the upgrade of PMM Client. In order to use PMM3, upgrade your PMM Server to version 3.

   To keep using PMM2, specify the PMM Client version compatible with PMM Server 2.

   If you haven't updated your PMM Server yet, exclude PMM Client from the list of images to update.

   Since this is a working cluster, the way to update the Custom Resource is to apply a patch with the `kubectl patch pxc` command.

   With PMM Client

   For Percona XtraDB Cluster 8.4

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.4.7-7.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.4.0-5.1" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" },
       "pmm":      { "image": "percona/pmm-client:3.5.0" }
   }}'
```

   For Percona XtraDB Cluster 8.0

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.0.44-35.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.0.35-34.1" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" },
       "pmm":      { "image": "percona/pmm-client:3.5.0" }
   }}'
```

   For Percona XtraDB Cluster 5.7

```
```{.bash data-prompt="$"}
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:5.7.44-31.65" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:2.4.29" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" },
       "pmm":      { "image": "percona/pmm-client:3.5.0" }
   }}'
```
```

   Without PMM Client

   For Percona XtraDB Cluster 8.4

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.4.7-7.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.4.0-5.1" },
```

```
        "logcollector": { "image": "percona/fluentbit:4.0.1-1" }
    }}'
```

For Percona XtraDB Cluster 8.0

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.0.44-35.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.0.35-34.1" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" }
   }}'
```

For Percona XtraDB Cluster 5.7

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:5.7.44-31.65" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:2.4.29" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" }
   }}'
```

Upgrade via Helm

If you have installed the Operator using Helm, you can upgrade the Operator with the `helm upgrade` command.

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-xtradb-cluster-
   operator/v1.19.0/deploy/crd.yaml
   $ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-
   operator/v1.19.0/deploy/rbac.yaml
   ```

2. Next, update the Operator deployment.

   With default parameters

   If you installed the Operator with default parameters, the upgrade can be done as follows:

   ```
   $ helm upgrade my-op percona/pxc-operator --version 1.19.0
   ```

   With customized parameters

   If you installed the Operator with some customized parameters, you should list these options in the upgrade command.

   You can get the list of the used options in YAML format with the `helm get values my-op -a > my-values.yaml` command. Then pass this file directly to the upgrade command as follows:

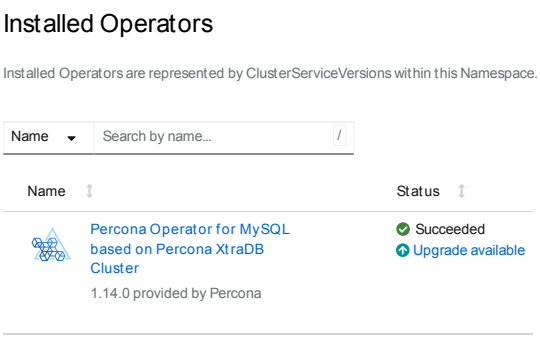
   ```
   $ helm upgrade my-op percona/pxc-operator --version 1.19.0 -f my-values.yaml
   ```

   The `my-op` parameter in the above example is the name of a release object which which you have chosen for the Operator when installing its Helm chart.

Upgrade via Operator Lifecycle Manager (OLM)

If you have installed the Operator on the OpenShift platform using OLM, you can upgrade the Operator within it.

1. List installed Operators for your Namespace to see if there are upgradable items.



2. Click the “Upgrade available” link to see upgrade details, then click “Preview InstallPlan” button, and finally “Approve” to upgrade the Operator.

Upgrade Percona XtraDB cluster

You can decide how to run the database upgrades:

- Automatically - the Operator periodically checks for new versions of the database images and for valid image paths and automatically updates your deployment with the latest, recommended or a specific version of the database and other components included. To do so, the Operator queries a special *Version Service* server at scheduled times. If the current version should be upgraded, the Operator updates the Custom Resource to reflect the new image paths and sequentially deletes Pods, allowing StatefulSet to redeploy the cluster Pods with the new image.
- Manually - you manually update the Custom Resource and specify the desired version of the database. Then, depending on the configured update strategy, either the Operator automatically updates the deployment to this version. Or you manually trigger the upgrade by deleting Pods.

The way to instruct the Operator how it should run the database upgrades is to set the `upgradeOptions.apply` Custom Resource option to one of the following:

- `Never` - the Operator never makes automatic upgrades. You must upgrade the Custom Resource and images manually.
- `Disabled` - the Operator doesn't not carry on upgrades automatically. You must upgrade the Custom Resource and images manually.
- `Recommended` - the Operator automatically updates the database and components to the version flagged as Recommended.
- `Latest` - the Operator automatically updates the database and components to the most recent available version
- `version` - specify the specific database version that you want to update to in the format `8.0.44-35.1`, `5.7.44-31.65`, etc.. The Operator updates the database to it automatically. Find available versions in the list of certified images.

For previous versions, refer to the old releases documentation archive).

Minor upgrade

To a specific version

Upgrading Percona XtraDB Cluster to a specific version

Assumptions

For the procedures in this tutorial, we assume that you have set up the `Smart Update` strategy to update the objects in your database cluster.

Read more about the Smart Update strategy and other available ones in the Upgrade strategies section.

Before you start

1. We recommend to update PMM Server **before** upgrading PMM Client.
2. If you are using PMM server version 2, use a PMM client image compatible with PMM 2. If you are using PMM server version 3, use a PMM client image compatible with PMM 3. See PMM upgrade documentation for how to migrate from version 2 to version 3.
3. If you are using custom configuration for HAProxy, check the HAProxy configuration file provided by the Operator **before the upgrade**. Find the `haproxy-global.cfg` for the Operator version 1.19.0 here.

   Make sure that your custom config is still compatible with the new variant, and make necessary additions, if needed.

Procedure

To update Percona XtraDB Cluster to a specific version, do the following:

1. Check the version of the Operator you have in your Kubernetes environment. If you need to update it, refer to the Operator upgrade guide.
2. Check the Custom Resource manifest configuration to be the following:

   - `spec.updateStrategy` option is set to `SmartUpdate`
   - `spec.upgradeOptions.apply` option is set to `Disabled` or `Never`.

   ```
   ...
   spec:
     updateStrategy: SmartUpdate
     upgradeOptions:
       apply: Disabled
       ...
   ```

3. Check the current version of the Custom Resource and what versions of the database and cluster components are compatible with it. Use the following command:

   ```
   $ curl https://check.percona.com/versions/v1/pxc-operator/1.19.0 |jq -r '.versions[].matrix'
   ```

   You can also find this information in the Versions compatibility matrix.

4. Update the Custom Resource, the database, backup, proxy and PMM Client image names with a newer version tag. Find the image names in the list of certified images.

   We recommend to update the PMM Server **before** the upgrade of PMM Client. In order to use PMM3, upgrade your PMM Server to version 3.

   If you haven't updated your PMM Server yet, exclude PMM Client from the list of images to update.

   To keep using PMM2, specify the PMM Client version compatible with PMM Server 2.

   Since this is a working cluster, the way to update the Custom Resource is to apply a patch with the `kubectl patch pxc` command.

   With PMM Client

   For Percona XtraDB Cluster 8.4

   ```
   $ kubectl patch pxc cluster1 --type=merge --patch '{
   ```

```
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.4.7-7.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.4.0-5.1" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" },
       "pmm":      { "image": "percona/pmm-client:3.5.0" }
   }}'
```

For Percona XtraDB Cluster 8.0

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.0.44-35.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.0.35-34.1" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" },
       "pmm":      { "image": "percona/pmm-client:3.5.0" }
   }}'
```

For Percona XtraDB Cluster 5.7

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:5.7.44-31.65" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:2.4.29" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" },
       "pmm":      { "image": "percona/pmm-client:3.5.0" }
   }}'
```

Without PMM Client

For Percona XtraDB Cluster 8.4

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.4.7-7.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.4.0-5.1" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" }

   }}'
```

For Percona XtraDB Cluster 8.0

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.0.44-35.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.0.35-34.1" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" }

   }}'
```

For Percona XtraDB Cluster 5.7

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:5.7.44-31.65" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:2.4.29" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" }
```

```
    }}'
```

5. 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 sts cluster1-pxc
```

Automated minor upgrade to the latest / recommended version

Assumptions

For the procedures in this tutorial, we assume that you have set up the `Smart Update` strategy to update the objects in your database cluster.

Read more about the Smart Update strategy and other available ones in the Upgrade strategies section.

Before you start

1. We recommend to update PMM Server **before** upgrading PMM Client.
2. If you are using custom configuration for HAProxy, check the HAProxy configuration file provided by the Operator **before the upgrade**. Find the `haproxy-global.cfg` for the Operator version {{ release }} here.

   Make sure that your custom config is still compatible with the new variant, and make necessary additions, if needed.

Procedure

You can configure the Operator to automatically upgrade Percona Server for MongoDB to the latest available, the recommended or to a specific version of your choice.

Learn more about automatic upgrades

The steps are the following:

1. Check the version of the Operator you have in your Kubernetes environment. If you need to update it, refer to the Operator upgrade guide.
2. Make sure that `spec.updateStrategy` option is set to `SmartUpdate`.
3. Change the `upgradeOptions.apply` option from `Disabled` to one of the following values:

   - `Recommended` - automatic upgrade will choose the most recent version of software flagged as “Recommended”. For newly created clusters, the Operator will always select Percona XtraDB Cluster 8.0 instead of Percona XtraDB Cluster 5.7, regardless of of the image path. For already existing clusters the Operator respects your choice of Percona XtraDB Cluster version (5.7 vs 8.0) and updates the selected version.
   - `8.0-recommended`, `5.7-recommended` - same as above, but preserves specific major Percona XtraDB Cluster version for newly provisioned clusters (e.g. 8.0 will not be automatically used instead of 5.7),
   - `Latest` - automatic upgrades will choose the most recent version of the software available
   - `8.0-latest`, `5.7-latest` - same as above, but preserves specific major Percona XtraDB Cluster version for newly provisioned clusters (e.g. 8.0 will not be automatically used instead of 5.7),
   - *version number* - specify the desired version explicitly (version numbers are specified as `8.0.44-35.1`, `5.7.44-31.65`, etc.). Actual versions can be found in the list of certified images. For older releases, please refer to the old releases documentation archive.

4. Make sure to set the valid Version Server URL for the `versionServiceEndpoint` key. The Operator checks the new software versions in the Version Server. If the Operator can’t reach the Version Server, the upgrades won’t happen.

   Percona’s Version Service (default)

   You can use the URL of the official Percona’s Version Service (default). Set `upgradeOptions.versionServiceEndpoint` to `https://check.percona.com`.

   Version Service inside your cluster

   Alternatively, you can run Version Service inside your cluster. This can be done with the `kubectl` command as follows:

   ```
   $ kubectl run version-service --image=perconalab/version-service --env="SERVE_HTTP=true" --port 11000 --expose
   ```

5. Specify the schedule to check for the new versions in in CRON format for the `upgradeOptions.schedule` option.

   The following example sets the midnight update checks with the official Percona’s Version Service:

```
spec:
  updateStrategy: SmartUpdate
  upgradeOptions:
    apply: Recommended
    versionServiceEndpoint: https://check.percona.com
    schedule: "0 0 * * *"
...
```

Note

You can force an immediate upgrade by changing the schedule to `* * * * *` (continuously check and upgrade) and changing it back to another more conservative schedule when the upgrade is complete.

6. Apply your changes to the Custom Resource:

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

How to carry on low-level manual upgrades of Percona XtraDB Cluster

The default and recommended way to upgrade the database cluster is using the Smart Update strategy. The Operator controls how objects are updated and restarts the Pods in a proper order during the database upgrade or for other events that require the cluster update. To these events belong ConfigMap updates, password rotation or changing resource values.

In some cases running an automatic upgrade of Percona XtraDB Cluster is not an option. For example, if the database upgrade impacts your application, you may want to have a full control over the upgrade process.

Running a manual database upgrade allows you to do just that. You can use one of the following *upgrade strategies*:

- *Rolling Update*, initiated manually and controlled by Kubernetes. Note that the order of Pods restart may not be optimal from the Percona Server for MongoDB point of view.
- *On Delete*, done by Kubernetes on per-Pod basis when Pods are manually deleted.

Before you start

1. We recommend to update PMM Server **before** upgrading PMM Client.
2. If you are using custom configuration for HAProxy, check the HAProxy configuration file provided by the Operator **before the upgrade**. Find the `haproxy-global.cfg` for the Operator version 1.19.0 here).

   Make sure that your custom config is still compatible with the new variant, and make necessary additions, if needed.

Rolling Update strategy and semi-automatic updates

To run a semi-automatic update of Percona XtraDB Cluster, do the following:

1. Check the version of the Operator you have in your Kubernetes environment. If you need to update it, refer to the Operator upgrade guide.
2. Edit the `deploy/cr.yaml` file and set the `updateStrategy` key to `RollingUpdate`.
3. Check the current version of the Custom Resource and what versions of the database and cluster components are compatible with it. Use the following command:

   ```
   $ curl https://check.percona.com/versions/v1/pxc-operator/1.19.0 |jq -r '.versions[].matrix'
   ```

   You can also find this information in the Versions compatibility matrix.
4. Update the Custom Resource, the database, backup, proxy and PMM Client image names with a newer version tag. Find the image names in the list of certified images.

   We recommend to update the PMM Server **before** the upgrade of PMM Client. In order to use PMM3, upgrade your PMM Server to version 3.

   If you haven't updated your PMM Server yet, exclude PMM Client from the list of images to update.

   To keep using PMM2, specify the PMM Client version compatible with PMM Server 2.

   Since this is a working cluster, the way to update the Custom Resource is to apply a patch with the `kubectl patch pxc` command.

   With PMM Client

   For Percona XtraDB Cluster 8.4

   ```
   $ kubectl patch pxc cluster1 --type=merge --patch '{
      "spec": {
          "crVersion":"1.19.0",
          "pxc":{ "image": "percona/percona-xtradb-cluster:8.4.7-7.1" },
          "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
          "haproxy":  { "image": "percona/haproxy:2.8.17" },
          "backup":   { "image": "percona/percona-xtrabackup:8.4.0-5.1" },
          "logcollector": { "image": "percona/fluentbit:4.0.1-1" },
          "pmm":      { "image": "percona/pmm-client:3.5.0" }
      }}'
   ```

   For Percona XtraDB Cluster 8.0

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.0.44-35.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.0.35-34.1" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" },
       "pmm":      { "image": "percona/pmm-client:3.5.0" }
   }}'
```

For Percona XtraDB Cluster 5.7

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:5.7.44-31.65" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:2.4.29" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" },
       "pmm":      { "image": "percona/pmm-client:3.5.0" }
   }}'
```

Without PMM Client

For Percona XtraDB Cluster 8.4

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.4.7-7.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.4.0-5.1" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" }

   }}'
```

For Percona XtraDB Cluster 8.0

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.0.44-35.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.0.35-34.1" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" }

   }}'
```

For Percona XtraDB Cluster 5.7

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:5.7.44-31.65" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:2.4.29" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" }

   }}'
```

5. After you applied the patch, the deployment rollout will be triggered automatically. You can track the rollout process in real time using the `kubectl rollout status` command with the name of your cluster:

```
$ kubectl rollout status sts cluster1-pxc
```

Manual upgrade (the On Delete strategy)

To update Percona XtraDB Cluster manually, do the following:

1. Check the version of the Operator you have in your Kubernetes environment. If you need to update it, refer to the Operator upgrade guide.

2. Edit the `deploy/cr.yaml` file and set the `updateStrategy` key to `OnDelete`.

3. Check the current version of the Custom Resource and what versions of the database and cluster components are compatible with it. Use the following command:

   ```
   $ curl https://check.percona.com/versions/v1/pxc-operator/1.19.0 |jq -r '.versions[].matrix'
   ```

   You can also find this information in the Versions compatibility matrix.

4. Update the Custom Resource, the database, backup, proxy and PMM Client image names with a newer version tag. Find the image names in the list of certified images.

   We recommend to update the PMM Server **before** the upgrade of PMM Client. In order to use PMM3, upgrade your PMM Server to version 3.

   If you haven't updated your PMM Server yet, exclude PMM Client from the list of images to update.

   To keep using PMM2, specify the PMM Client version compatible with PMM Server 2.

   Since this is a working cluster, the way to update the Custom Resource is to apply a patch with the `kubectl patch pxc` command.

   With PMM Client

   For Percona XtraDB Cluster 8.4

   ```
   $ kubectl patch pxc cluster1 --type=merge --patch '{
      "spec": {
          "crVersion":"1.19.0",
          "pxc":{ "image": "percona/percona-xtradb-cluster:8.4.7-7.1" },
          "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
          "haproxy":  { "image": "percona/haproxy:2.8.17" },
          "backup":   { "image": "percona/percona-xtrabackup:8.4.0-5.1" },
          "logcollector": { "image": "percona/fluentbit:4.0.1-1" },
          "pmm":      { "image": "percona/pmm-client:3.5.0" }
      }}'
   ```

   For Percona XtraDB Cluster 8.0

   ```
   $ kubectl patch pxc cluster1 --type=merge --patch '{
      "spec": {
          "crVersion":"1.19.0",
          "pxc":{ "image": "percona/percona-xtradb-cluster:8.0.44-35.1" },
          "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
          "haproxy":  { "image": "percona/haproxy:2.8.17" },
          "backup":   { "image": "percona/percona-xtrabackup:8.0.35-34.1" },
          "logcollector": { "image": "percona/fluentbit:4.0.1-1" },
          "pmm":      { "image": "percona/pmm-client:3.5.0" }
      }}'
   ```

   For Percona XtraDB Cluster 5.7

   ```
   $ kubectl patch pxc cluster1 --type=merge --patch '{
      "spec": {
          "crVersion":"1.19.0",
          "pxc":{ "image": "percona/percona-xtradb-cluster:5.7.44-31.65" },
          "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
          "haproxy":  { "image": "percona/haproxy:2.8.17" },
          "backup":   { "image": "percona/percona-xtrabackup:2.4.29" },
          "logcollector": { "image": "percona/fluentbit:4.0.1-1" },
          "pmm":      { "image": "percona/pmm-client:3.5.0" }
      }}'
   ```

   Without PMM Client

   For Percona XtraDB Cluster 8.4

   ```
   $ kubectl patch pxc cluster1 --type=merge --patch '{
   ```

```
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.4.7-7.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.4.0-5.1" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" }

   }}'
```

For Percona XtraDB Cluster 8.0

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:8.0.44-35.1" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:8.0.35-34.1" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" }

   }}'
```

For Percona XtraDB Cluster 5.7

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
   "spec": {
       "crVersion":"1.19.0",
       "pxc":{ "image": "percona/percona-xtradb-cluster:5.7.44-31.65" },
       "proxysql": { "image": "percona/proxysql2:2.7.3-1.2" },
       "haproxy":  { "image": "percona/haproxy:2.8.17" },
       "backup":   { "image": "percona/percona-xtrabackup:2.4.29" },
       "logcollector": { "image": "percona/fluentbit:4.0.1-1" }

   }}'
```

5. The Pod with the newer Percona XtraDB Cluster image will start after you delete it. Delete targeted Pods manually one by one to make them restart in desired order:

   1. Delete the Pod using its name with the command like the following one:

      ```
      $ kubectl delete pod cluster1-pxc-2
      ```

   2. Wait until Pod becomes ready:

      ```
      $ kubectl get pod cluster1-pxc-2
      ```

      The output should be like this:

      ```
      NAME             READY   STATUS    RESTARTS   AGE
      cluster1-pxc-2   1/1     Running   0          3m33s
      ```

The update process is successfully finished when all Pods have been restarted.

Upgrade Database and Operator on OpenShift

Upgrading database and Operator on Red Hat Marketplace or to upgrade Red Hat certified Operators on OpenShift generally follows the standard upgrade scenario, but includes a number of special steps specific for these platforms.

Considerations for using OpenShift 4.19

Starting with OpenShift 4.19, the way images with not fully qualified names are pulled has changed for repositories that share the same repository name on DockerHub and Red Hat Marketplace. By default the tags are pulled from Red Hat Marketplace. Specifying not fully qualified image names may result in the `ImagePullBackOff` error.

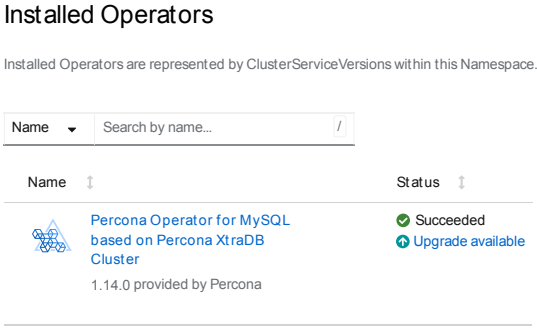
- **OLM installation:** Images are provided with the fully qualified names and are pulled from the Red Hat Marketplace/Dockerhub registry.
- **Manual install/update with default manifests:** Images must use the `docker.io` registry prefix to guarantee successful download from the Dockerhub `percona-xtradb-cluster` repository. See the Update via the command-line interface section for the exact steps.

Upgrading the Operator and CRD

Operator 1.15.0 and newer

You can actually update the Operator via the Operator Lifecycle Manager (OLM) web interface.

Login to your OLM installation and list installed Operators for your Namespace to see if there are upgradable items:



Click the “Upgrade available” link to see upgrade details, then click “Preview InstallPlan” button, and finally “Approve” to upgrade the Operator.

Operator 1.14.0

1. First of all you need to manually update `initContainer.image` Custom Resource option with the value of an alternative initial Operator installation image. You need doing this for all database clusters managed by the Operator. Without this step the cluster will go into error state after the Operator upgrade.

   a. Find the initial Operator installation image with `kubectl get deploy` command:

   ```
   $ kubectl get deploy percona-xtradb-cluster-operator -o yaml
   ```

   Expected output

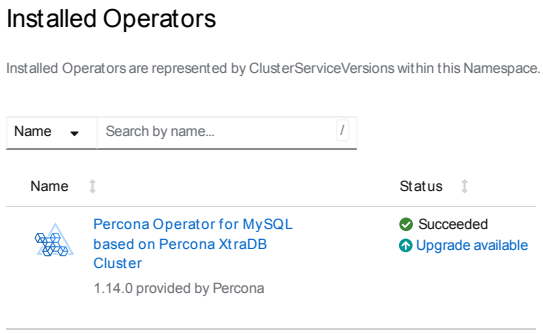
   ```
   ...
   "initContainer" : {
     "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-
   operator@sha256:4edb5a53230e023bbe54c8e9e1154579668423fc3466415d5b04b8304a8e01d7"
   },
   ...
   ```

   b. Apply a patch to update the `initContainer.image` option of your cluster Custom Resource with this value. Supposing that your cluster name is `cluster1`, the command should look as follows:

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
    "spec": {
       "initContainer": { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-
operator@sha256:4edb5a53230e023bbe54c8e9e1154579668423fc3466415d5b04b8304a8e01d7" }
    }}'
```

2. Now you can actually update the Operator via the Operator Lifecycle Manager (OLM) web interface.

   Login to your OLM installation and list installed Operators for your Namespace to see if there are upgradable items:



   Click the “Upgrade available” link to see upgrade details, then click “Preview InstallPlan” button, and finally “Approve” to upgrade the Operator.

Operator 1.13.0 and older

1. First of all you need to manually update `initImage` Custom Resource option with the value of an alternative initial Operator installation image. You need doing this for all database clusters managed by the Operator. Without this step the cluster will go into error state after the Operator upgrade.

   a. Find the initial Operator installation image with `kubectl get deploy` command:

   ```
   $ kubectl get deploy percona-xtradb-cluster-operator -o yaml
   ```

   Expected output

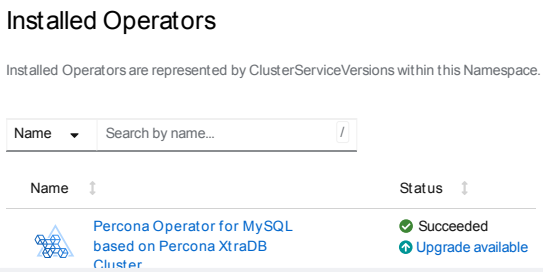
   ```
   ...
   "initContainer" : {
     "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-
   operator@sha256:4edb5a53230e023bbe54c8e9e1154579668423fc3466415d5b04b8304a8e01d7"
   },
   ...
   ```

   b. Apply a patch to update the `initImage` option of your cluster Custom Resource with this value. Supposing that your cluster name is `cluster1`, the command should look as follows:

   ```
   $ kubectl patch pxc cluster1 --type=merge --patch '{
       "spec": {
          "initImage":"registry.connect.redhat.com/percona/percona-xtradb-cluster-
   operator@sha256:4edb5a53230e023bbe54c8e9e1154579668423fc3466415d5b04b8304a8e01d7"
       }}'
   ```

2. Now you can actually update the Operator via the Operator Lifecycle Manager (OLM) web interface.

   Login to your OLM installation and list installed Operators for your Namespace to see if there are upgradable items:



Up

The ıe Considerations for using OpenShift 4.19.

1. ıtainer.image is set.

```
* If defined: skip the next step.
* If undefined: proceed to step 2.
```

Click the "Upgrade available" link to see upgrade details, then click "Preview InstallPlan" button, and finally "Approve" to upgrade the Operator.

a. Apply a patch to the clusters with undefined `initContainer.image` to define this image with the `docker.io` registry in the image path:

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
  "spec": {
    "initcontainer": {
      "image": "docker.io/percona/percona-xtradb-cluster-operator:1.17.0"
    }
  }
}'
```

Important! This command triggers the restart of your clusters. Wait till they restart and report the Ready status

a. Update the Operator deployment and specify the `docker.io` registry name in the image path:

```
$ kubectl patch deployment percona-xtradb-cluster-operator \
-p'{"spec":{"template":{"spec":{"containers":[{"name":"percona-xtradb-cluster-
operator","image":"docker.io/percona/percona-xtradb-cluster-operator:1.19.0"}]}}}}'
```

b. Update the Custom Resource version and the database cluster. Specify the `initContainer` image with the `docker.io` registry name in the path. Pay attention to the changed repositories for PXB and logcollector:

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
  "spec": {
    "crVersion": "1.19.0",
    "initContainer": "docker.io/percona/percona-xtradb-cluster-operator:1.19.0",
    "pxc":{ "image": "docker.io/percona/percona-xtradb-cluster:8.0.44-35.1" },
    "proxysql":{ "image": "docker.io/percona/proxysql2:2.7.3-1.2" },
    "haproxy":{ "image": "docker.io/percona/haproxy:2.8.17" },
    "backup":{ "image": "docker.io/percona/percona-xtrabackup:8.0.35-34.1" },
    "logcollector":{ "image": "docker.io/percona/fluentbit:4.0.1-1" },
    "pmm":{ "image": "docker.io/percona/pmm-client:2.44.1-1" }
  }
}'
```

Upgrading Percona XtraDB Cluster

1. Make sure that `spec.updateStrategy` option in the Custom Resource is set to `SmartUpdate`, `spec.upgradeOptions.apply` option is set to `Never` or `Disabled` (this means that the Operator will not carry on upgrades automatically).

```
...
spec:
  updateStrategy: SmartUpdate
  upgradeOptions:
    apply: Disabled
    ...
```

2. Find the **new** initial Operator installation image name (it had changed during the Operator upgrade) and other image names for the components of your cluster with the `kubectl get deploy` command:

```
$ kubectl get deploy percona-xtradb-cluster-operator -o yaml
```

Expected output


```
...
"initContainer" : {
  "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-
operator@sha256:e8c0237ace948653d8f3e297ec67276f23f4f7fb4f8018f97f246b65604d49e6"
},
...
"pxc": {
  "size": 3,
  "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:b526b83865ca26808aa1ef96f64319f65deba94b76c5b5b6aa181981ebd4282f"
},
...
"haproxy": {
  "enabled": true,
  "size": 3,
  "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:cbd4f1791941765eb6732f2dc88bad29bf23469898bd30f02d22a95c0f2aab9b"
},
...
"proxysql": {
  "enabled": false,
  "size": 3,
  "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:24f6d959efcf2083addf42f3b816220654133dc8a5a8a989ffd4caffe122e19c"
},
...
"logcollector": {
  "enabled": true,
  "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:cb6ccda7839b3205ffaf5cb8016d1f91ed3be4438334d2122beb38791a32c015"
},
...
"pmm": {
  "enabled": false,
  "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:165f97cdae2b6def546b0df7f50d88d83c150578bdb9c992953ed866615016f1"
},
...
"backup": {
  "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:483acaa57378ee5529479dbcabb3b8002751c1c43edd5553b52f001f323d4723"
},
...
```


3. Apply a patch to set the necessary `crVersion` value (equal to the Operator version) and update images in your cluster Custom Resource. Supposing that your cluster name is `cluster1`, the command should look as follows:

Operator 1.14.0 or newer

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
    "spec": {
       "crVersion":"1.19.0",
       "initContainer": { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-
operator@sha256:e8c0237ace948653d8f3e297ec67276f23f4f7fb4f8018f97f246b65604d49e6" },
       "pxc":{ "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:b526b83865ca26808aa1ef96f64319f65deba94b76c5b5b6aa181981ebd4282f" },
       "proxysql": { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:24f6d959efcf2083addf42f3b816220654133dc8a5a8a989ffd4caffe122e19c" },
       "haproxy":  { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:cbd4f1791941765eb6732f2dc88bad29bf23469898bd30f02d22a95c0f2aab9b" },
       "backup":   { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:483acaa57378ee5529479dbcabb3b8002751c1c43edd5553b52f001f323d4723" },
       "logcollector": { "image": "percona/percona-xtradb-cluster-operator:1.19.0-logcollector-fluentbit4.0.1-1" },
       "pmm":      { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:165f97cdae2b6def546b0df7f50d88d83c150578bdb9c992953ed866615016f1" }
    }}'
```

Operator 1.13.0 or older

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
    "spec": {
       "crVersion":"1.13.0",
       "initImage": "registry.connect.redhat.com/percona/percona-xtradb-cluster-
operator@sha256:e8c0237ace948653d8f3e297ec67276f23f4f7fb4f8018f97f246b65604d49e6",
       "pxc":{ "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:b526b83865ca26808aa1ef96f64319f65deba94b76c5b5b6aa181981ebd4282f" },
       "proxysql": { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:24f6d959efcf2083addf42f3b816220654133dc8a5a8a989ffd4caffe122e19c" },
       "haproxy":  { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:cbd4f1791941765eb6732f2dc88bad29bf23469898bd30f02d22a95c0f2aab9b" },
       "backup":   { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:483acaa57378ee5529479dbcabb3b8002751c1c43edd5553b52f001f323d4723" },
       "logcollector": { "image": "percona/percona-xtradb-cluster-operator:1.19.0-logcollector-fluentbit4.0.1-1" },
       "pmm":      { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:165f97cdae2b6def546b0df7f50d88d83c150578bdb9c992953ed866615016f1" }
    }}'
```

Warning

The above command upgrades various components of the cluster including PMM Client. If you didn't follow the official recommendation to upgrade PMM Server before upgrading PMM Client, you can avoid PMM Client upgrade by removing it from the list of images as follows:

Operator 1.14.0 or newer

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
    "spec": {
       "crVersion":"1.19.0",
       "initContainer": { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-
operator@sha256:e8c0237ace948653d8f3e297ec67276f23f4f7fb4f8018f97f246b65604d49e6" },
       "pxc":{ "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:b526b83865ca26808aa1ef96f64319f65deba94b76c5b5b6aa181981ebd4282f" },
       "proxysql": { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:24f6d959efcf2083addf42f3b816220654133dc8a5a8a989ffd4caffe122e19c" },
       "haproxy":  { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:cbd4f1791941765eb6732f2dc88bad29bf23469898bd30f02d22a95c0f2aab9b" },
       "backup":   { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:483acaa57378ee5529479dbcabb3b8002751c1c43edd5553b52f001f323d4723" },
       "logcollector": { "image": "percona/percona-xtradb-cluster-operator:1.19.0-logcollector-fluentbit4.0.1-1" }
    }}'
```

Operator 1.13.0 or older

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
    "spec": {
       "crVersion":"1.13.0",
       "initImage": "registry.connect.redhat.com/percona/percona-xtradb-cluster-
operator@sha256:e8c0237ace948653d8f3e297ec67276f23f4f7fb4f8018f97f246b65604d49e6",
       "pxc":{ "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:b526b83865ca26808aa1ef96f64319f65deba94b76c5b5b6aa181981ebd4282f" },
       "proxysql": { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:24f6d959efcf2083addf42f3b816220654133dc8a5a8a989ffd4caffe122e19c" },
       "haproxy":  { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:cbd4f1791941765eb6732f2dc88bad29bf23469898bd30f02d22a95c0f2aab9b" },
       "backup":   { "image": "registry.connect.redhat.com/percona/percona-xtradb-cluster-operator-
containers@sha256:483acaa57378ee5529479dbcabb3b8002751c1c43edd5553b52f001f323d4723" },
       "logcollector": { "image": "percona/percona-xtradb-cluster-operator:1.19.0-logcollector-fluentbit4.0.1-1" }
    }}'
```

4. 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 sts cluster1-pxc
```

Configuration

Users

MySQL user accounts within the Cluster can be divided into two different groups:

- *application-level users*: the unprivileged user accounts for applications,
- *system-level users*: the accounts needed to automate the cluster deployment and management tasks, such as Percona XtraDB Cluster Health checks or ProxySQL integration.

These two groups serve different purposes. Read the following sections to learn more.

Unprivileged users

The Operator doesn't create application-level (unprivileged) user accounts by default.

You can create these unprivileged users in the following ways:

- Automatically via Custom Resource. This ability is available with Operator versions 1.16.0 and newer
- Manually in MySQL

Create users in the Custom Resource

Starting from the Operator version 1.16.0, you can create users in Percona XtraDB Cluster via the `users` subsection in the Custom Resource. This is called declarative user management.

You can change the `users` section in the `deploy/cr.yaml` configuration file either at the cluster creation time or adjust it over time.

You can define users in the `users` section of your Custom Resource. For each user, you can specify:

- The user's login name
- Hosts the user is allowed to connect from
- Databases the user can access
- MySQL privilege grants
- A reference to a Secret resource containing the user's password

Here is an example configuration of the Custom Resource:

```
...
users:
- name: my-user
  dbs:
  - db1
  - db2
  hosts:
  - localhost
  grants:
  - SELECT
  - DELETE
  - INSERT
  withGrantOption: true
  passwordSecretRef:
    name: my-user-pwd
    key: my-user-pwd-key
...
```

For detailed information about all available options, see the Custom Resource reference.

Generate user passwords manually

You can create the Secret containing the user password manually. The Secret referenced in `users.passwordSecretRef.name` should follow this format:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-user-pwd
type: Opaque
stringData:
  password: my-user-pwd-key
```

The Operator tracks password changes in the Secret object and automatically updates the user password in the database when the Secret is modified.

Customize password generation by the Operator

By default, the Operator generates user passwords using alphanumeric characters plus a set of special symbols. The password length is randomly chosen in the range of 16 to 20 characters.

To ensure compatibility with tools that may not support certain special symbols or require a different password length, you can customize password generation using the `passwordGenerationOptions` subsection in the Custom Resource:

```
passwordGenerationOptions:
  symbols: "!#$%&()*+,-.<=>?@[]^_{}~"
  maxLength: 20
  minLength: 16
```

Important considerations

When creating users via Custom Resource, keep the following behavior in mind:

- **Minimal configuration**: The only required field is `users.name`. If you omit other fields:
- If no Secret is specified, the Operator generates a password and stores it in a Secret named `<cluster-name>-<custom-user-name>-secret`
- If `grants` or `dbs` are omitted, MySQL provides default grants
- **User deletion**: The Operator does not delete users when they are removed from the Custom Resource. This prevents accidental removal of pre-existing users.
- **Host changes**: When you update the `hosts` array (for example, changing `host1` to `host2`), the Operator creates a new user `user@host2` in addition to the existing `user@host1`. If you later change the host back to `host1`, the `user@host1` account will continue using its original password, which may differ from the password in the Secret.
- **Grant updates**: The Operator updates grants in an additive manner. It adds new grants but does not revoke existing ones that are not specified in the Custom Resource.
- **Duplicate users**: You cannot define two entries for the same user (for example, with different grants for different databases) in a single Custom Resource. However, you can achieve this by making sequential updates to the Custom Resource.
- **Invalid grants**: If you specify an invalid grant or set an administrative (global) grant while also specifying `dbs`, the Operator logs an error and creates the user with default grants (`GRANT USAGE`).

Create users manually

Instead of using the Custom Resource, you can create unprivileged users directly in MySQL using standard SQL commands. This approach gives you full control over user creation and is useful when you need to create users with specific configurations that may not be easily expressed in the Custom Resource format.

Create a user

To create a user manually, connect to your cluster and run the `GRANT` statement. Replace `cluster1` with your actual cluster name and `root_password` with your root password:

```
$ kubectl run -it --rm percona-client --image=percona:8.0 --restart=Never -- mysql -hcluster1-pxc -uroot -proot_password
mysql> GRANT ALL PRIVILEGES ON database1.* TO 'user1'@'%' IDENTIFIED BY 'password1';
```

Note

MySQL password for the user you create should not exceed 32 characters due to the replication-specific limit introduced in MySQL 5.7.5.

Verify user creation

After creating the user, verify that it was created successfully and can connect to the database. The following example connects to the cluster via ProxySQL and executes a simple query:

```
$ kubectl run -it --rm percona-client --image=percona:8.0 --restart=Never -- bash -il
percona-client:/$ mysql -h cluster1-proxysql -uuser1 -ppassword1
mysql> SELECT * FROM database1.table1 LIMIT 1;
```

If the connection succeeds and you can execute queries, the user has been created correctly with the appropriate permissions.

System Users

To automate the deployment and management of the cluster components, the Operator requires system-level Percona XtraDB Cluster users.

Credentials for these users are stored as a Kubernetes Secrets object. The Operator requires Kubernetes Secrets before Percona XtraDB Cluster is started. It uses an existing Secret, if it already exists. Otherwise, the Operator creates a new Secret with randomly generated passwords. The default Secret name is `cluster1-secrets`.

The name of the required Secret should be set in the `spec.secretsName` option of the `deploy/cr.yaml` configuration file.

In addition to `cluster1-secrets`, the Operator will also create an internal Secrets object named `internal-cluster1`, which exists for technical purposes and should not be edited by end users. Read more about internal Secret usage in Internal Secret and its usage section.

The following table shows system user names and purposes.

Warning

Don't use system users to run applications.

User Purpose	Username	Password Secret Key	Description
Admin	root	root	Database administrative user, can be used by the application if needed
ProxySQLAdmin	proxyadmin	proxyadmin	ProxySQL administrative user, can be used to add general-purpose ProxySQL users
Backup	xtrabackup	xtrabackup	The user to run backups, granted `all` privileges for the point-in-time recovery needs
Monitoring	monitor	monitor	User for internal monitoring purposes like liveness/readiness checks and PMM agent
PMM Server token	Should be set through the operator options	pmmservertoken	The service token used to access PMM server version 3. For PMM 2, use API key.
Operator Admin	operator	operator	Database administrative user, should be used only by the Operator
Replication	replication	replication	Administrative user needed for cross-site Percona XtraDB Cluster

Note

The administrative database user `operator` is always created in MySQL as `operator@'%'`. Using an `operator` user with a different host value (for example, `operator@'hostname'`) is not supported and such users should not exist in the database.

YAML object format

The default name of the Secrets object for the system users is `cluster1-secrets`. You can create your own Secret and reference it in the Custom Resource for your cluster in the `spec.secretsName` key.

When you create the Secrets object yourself, your YAML file should match the following simple format:

```
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-secrets
type: Opaque
stringData:
  root: root_password
  xtrabackup: backup_password
  monitor: monitory
  proxyadmin: admin_password
  operator: operatoradmin
  replication: repl_password
  pmmservertoken: my_pmm_server_token
```

The example above matches the default `deploy/secrets.yaml` file, which includes sample passwords. These are intended only for development or automated testing. **Don't use them in production**.

Update the Secret

When you create the Secrets object, you use the `stringData` type and specify all values for each key/value pair in plain text format. However, the resulting Secrets object contains passwords stored as base64-encoded strings in the `data` type.

To update any field, you'll need to encode the value into the base64 format.

Here's how to do it:

1. Run the following command in your local shell to encode the new value. Replace `new_password` with your value:

   in Linux

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

   in macOS

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

2. Update the Secrets object. For example, the following command updates the Admin user's password to `new_password` in the `cluster1-secrets` object:

   in Linux

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

   in macOS

   ```
   $ kubectl patch secret/cluster1-secrets -p '{"data":{"root": "'$(echo -n new_password | base64)'"}}'
   ```

Internal Secret and its usage

The Operator creates and updates an additional Secrets object which is named based on the cluster name, like `internal-cluster1`. This Secrets object is used only by the Operator. Users must not change it.

This object contains secrets with the same passwords as the one specified in `spec.secretsName` (e.g., `cluster1-secrets`). When you update the `cluster1-secrets` Secret, the Operator propagates these changes to the internal `internal-cluster1` Secrets object.

Password rotation policies and timing

When there is a change in user secrets, the Operator creates the necessary transaction to change passwords. This rotation happens almost instantly (the delay can be up to a few seconds), and you don't need to take any action beyond changing the password.

> **Note**
>
> Please don't change the `secretsName` option in the CR. Make changes inside the secrets object itself.

Starting from the Operator version 1.13.0, system users are created with the `PASSWORD EXPIRE NEVER` policy. This policy is automatically applied to system users on existing clusters when the Operator is upgraded to version 1.13.0.

Marking system users in MySQL

Starting with MySQL 8.0.16, a new feature called Account Categories has been implemented, which allows you to mark system users as such. See the official documentation on this feature for more details.

Development mode

To make development and testing easier, the `deploy/secrets.yaml` secrets file contains default passwords for Percona XtraDB Cluster system users.

These development-mode credentials from `deploy/secrets.yaml` are:

Secret Key	Secret Value

root	`root_password`
xtrabackup	`backup_password`
monitor	`monitory`
proxyadmin	`admin_password`
operator	`operatoradmin`
replication	`repl_password`

Warning

Do not use the default Percona XtraDB Cluster user passwords in production!

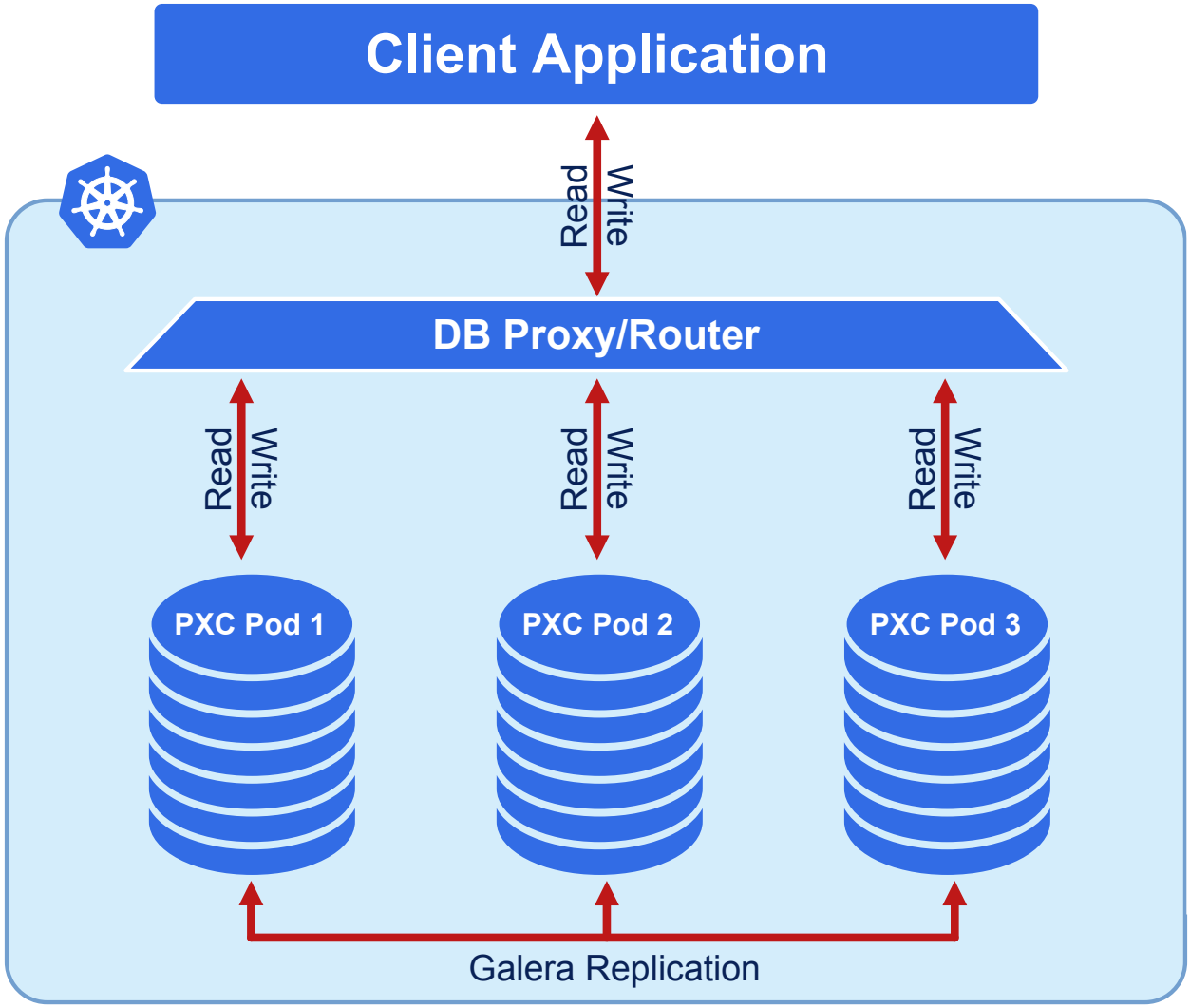
Exposing cluster

Percona Operator for MySQL based on Percona XtraDB Cluster provides entry points for accessing the database by client applications in several scenarios. In either way the 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.

Exposing cluster with HAProxy or ProxySQL

The Operator provides a choice of two cluster components to provide load balancing and proxy service: you can use either HAProxy or ProxySQL.



Load balancing and proxy service with HAProxy is the default choice.

- See how you can enable and use HAProxy and what are the limitations.
- See how you can enable and use ProxySQL and what are the limitations.

HAProxy

The default HAProxy based setup will contain the `cluster1-haproxy` Service listening on ports 3306 (MySQL primary) and 3309 (the proxy protocol useful for operations such as asynchronous calls), and also `cluster1-haproxy-replicas` Service for MySQL replicas, listening on port 3306 (this Service **should not be used for write requests**).

You can find the endpoint (the public IP address of the load balancer in our example) by getting the Service object with the `kubectl get service` command. The output will be as follows:

```
$ kubectl get service cluster1-haproxy
NAME                       TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)
AGE
cluster1-haproxy           LoadBalancer   10.12.23.173   <pending>
3306:32548/TCP,3309:30787/TCP,33062:32347/TCP,33060:31867/TCP   14s
cluster1-haproxy-replicas   LoadBalancer   10.12.25.208   <pending>     3306:32166/TCP
14s
```

You can control creation of these two Services with the following Custom Resource options:

- haproxy.exposePrimary.enabled enables or disables `cluster1-haproxy` Service,
- haproxy.exposeReplicas.enabled enables or disables `haproxy-replicas` Service.

By default `haproxy-replica` Service directs connections to all Pods of the database cluster in a round-robin manner, but `haproxy.exposeReplicas.onlyReaders` Custom Resource option allows to modify this behavior: setting it to `true` excludes current MySQL primary instance (writer) from the list, leaving only the reader instances. By default the option is set to `false`, which means that `haproxy-replicas` sends traffic to all Pods, including the active writer. The feature can be useful to simplify the application logic by splitting read and write MySQL traffic on the Kubernetes level.

Also, it should be noted that changing `haproxy.exposeReplicas.onlyReaders` value will cause HAProxy Pods to restart.

ProxySQL

If you configured your cluster with ProxySQL based setup, you will have `cluster1-proxysql` Service. You can find the endpoint (the public IP address of the load balancer in our example) by getting the Service object with the `kubectl get service` command. The output will be as follows:

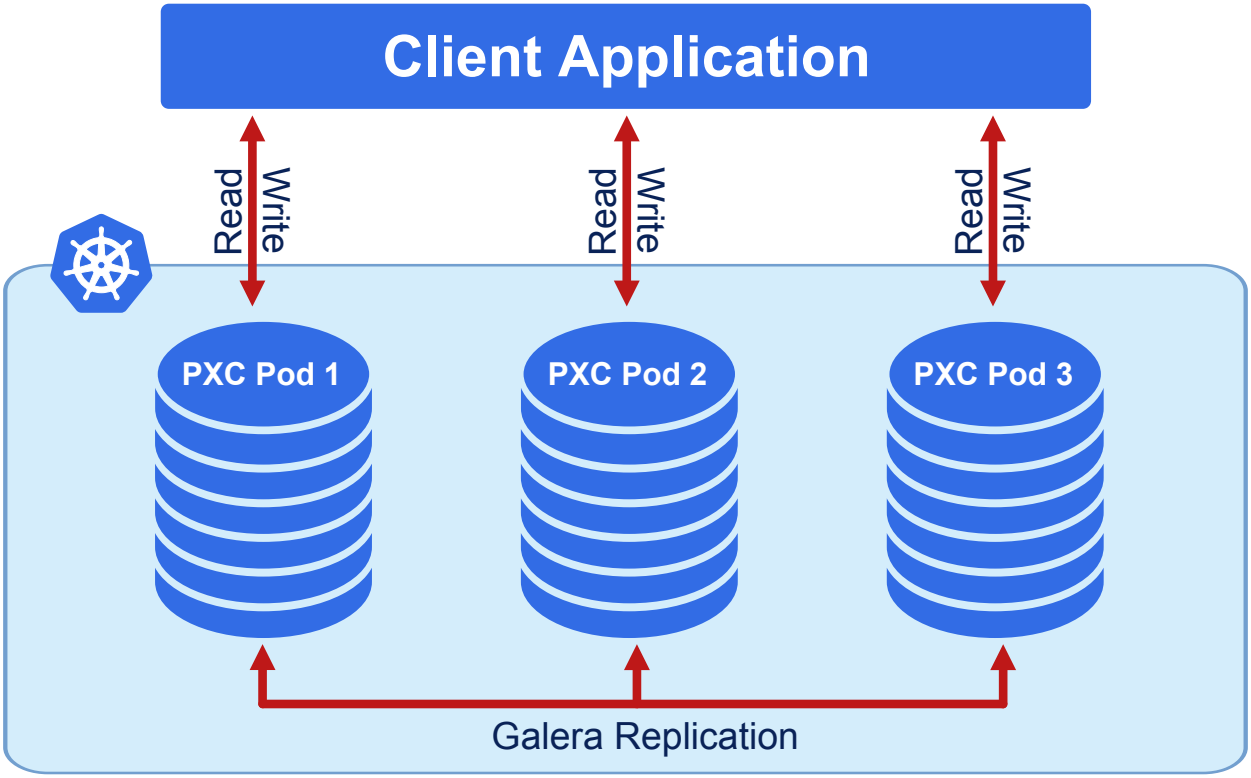
```
$ kubectl get service cluster1-proxysql
NAME                  TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                        AGE
cluster1-proxysql     LoadBalancer   10.0.238.36    35.192.172.85   3306:30408/TCP,33062:30217/TCP   115s
```

As you could notice, this command also shows mapped ports the application can use to communicate with MySQL primary instance (`3306` for the classic MySQL protocol).

You can enable or disable this Service with the proxysql.expose.enabled Custom Resource option.

Service per Pod

Still, sometimes it is required to expose all Percona XtraDB Cluster instances, where each of them gets its own IP address (e.g. in case of load balancing implemented on the application level).



This is possible by setting the following options in spec.pxc section.

- pxc.expose.enabled enables or disables exposure of Percona XtraDB Cluster instances,
- pxc.expose.type defines the Kubernetes Service object type.

The following example creates a dedicated LoadBalancer Service for each node of the MySQL cluster:

```
pxc:
  expose:
    enabled: true
    type: LoadBalancer
```

When the cluster instances are exposed in this way, you can find the corresponding Services with the `kubectl get services` command:

```
$ kubectl get services
NAME                            TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)                                 AGE
...
cluster1-pxc-0                  LoadBalancer   10.120.15.23    34.132.93.114   3306:30771/TCP                          111s
cluster1-pxc-1                  LoadBalancer   10.120.8.132    35.188.39.15    3306:30832/TCP                          111s
cluster1-pxc-2                  LoadBalancer   10.120.14.65    34.16.25.126    3306:32018/TCP                          111s
```

As you could notice, this command also shows mapped ports the application can use to communicate with MySQL instances (e.g. `3306` for the classic MySQL protocol, or `33060` for MySQL X Protocol useful for operations such as asynchronous calls).

Changing MySQL Options

You may need to change the MySQL configuration for your application. MySQL lets you customize its settings using a configuration file. You can include options from the my.cnf configuration file in one of these ways:

- Edit the `deploy/cr.yaml` file
- Use a ConfigMap
- Use a Secret object

In most cases, you don't need to add custom options because the Operator already provides sensible defaults for MySQL.

If you supply custom configuration in more than one way, the Operator will only use one method. It follows this order of preference:

1. First, it checks for a Secret object.
2. If it doesn't find a matching Secret, it looks for custom configuration in the Custom Resource (the `deploy/cr.yaml` file).
3. If neither of those exist, the Operator searches for a ConfigMap.

Edit the `deploy/cr.yaml` file

You can add options from the my.cnf configuration file by editing the configuration section of the `deploy/cr.yaml`. Here is an example:

```
spec:
  secretsName: cluster1-secrets
  pxc:
    ...
      configuration: |
        [mysqld]
        wsrep_debug=CLIENT
        [sst]
        wsrep_debug=CLIENT
```

See the Custom Resource options, PXC section for more details.

Use a ConfigMap

You can use a configmap and the cluster restart to reset configuration options. A configmap allows Kubernetes to pass or update configuration data inside a containerized application.

Use the `kubectl` command to create the configmap from external resources, for more information see Configure a Pod to use a ConfigMap.

For example, let's suppose that your application requires more connections. To increase your `max_connections` setting in MySQL, you define a `my.cnf` configuration file with the following setting:

```
[mysqld]
...
max_connections=250
```

You can create a configmap from the `my.cnf` file with the `kubectl create configmap` command.

You should use the combination of the cluster name with the `-pxc` suffix as the naming convention for the configmap. To find the cluster name, you can use the following command:

```
$ kubectl get pxc
```

The syntax for `kubectl create configmap` command is:

```
$ kubectl create configmap <cluster-name>-pxc <resource-type=resource-name>
```

The following example defines `cluster1-pxc` as the configmap name and the `my.cnf` file as the data source:

```
$ kubectl create configmap cluster1-pxc --from-file=my.cnf
```

To view the created configmap, use the following command:

```
$ kubectl describe configmaps cluster1-pxc
```

Use a Secret Object

The Operator can also store configuration options in Kubernetes Secrets. This can be useful if you need additional protection for some sensitive data.

You should create a Secret object with a specific name, composed of your cluster name and the `pxc` suffix.

Note

To find the cluster name, you can use the following command:

```
$ kubectl get pxc
```

Configuration options should be put inside a specific key inside of the `data` section. The name of this key is `my.cnf` for Percona XtraDB Cluster Pods.

Actual options should be encoded with Base64.

For example, let's define a `my.cnf` configuration file and put there a pair of MySQL options we used in the previous example:

```
[mysqld]
wsrep_debug=CLIENT
[sst]
wsrep_debug=CLIENT
```

You can get a Base64 encoded string from your options via the command line as follows:

in Linux

```
$ cat my.cnf | base64 --wrap=0
```

in macOS

```
$ cat my.cnf | base64
```

Note

Similarly, you can read the list of options from a Base64 encoded string:

```
$ echo "W215c3FsZF0Kd3NyZXBfZGVidWc9T04KW3NzdF0Kd3NyZXBfZGVidWc9T04K" | base64 --decode
```

Finally, use a yaml file to create the Secret object. For example, you can create a `deploy/my-pxc-secret.yaml` file with the following contents:

```
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-pxc
data:
  my.cnf: "W215c3FsZF0Kd3NyZXBfZGVidWc9T04KW3NzdF0Kd3NyZXBfZGVidWc9T04K"
```

When ready, apply it with the following command:

```
$ kubectl create -f deploy/my-pxc-secret.yaml
```

Note

Do not forget to restart Percona XtraDB Cluster to ensure the cluster has updated the configuration.

Make changed options visible to Percona XtraDB Cluster

Do not forget to restart Percona XtraDB Cluster to ensure the cluster has updated the configuration (see details on how to connect in the Install Percona XtraDB Cluster on Kubernetes page).

Auto-tuning MySQL options

Few configuration options for MySQL can be calculated and set by the Operator automatically based on the available Pod resource limits (memory and CPU) **if constant values for these options are not specified by user** (either in CR.yaml or in ConfigMap).

Options which can be set automatically are the following ones:

- `innodb_buffer_pool_size`
- `max_connections`

If Percona XtraDB Cluster Pod limits are defined, then limits values are used to calculate these options. If Percona XtraDB Cluster Pod limits are not defined, auto-tuning is not done.

Also, starting from the Operator 1.12.0, there is another way of auto-tuning. You can use `"{{ containerMemoryLimit }}"` as a value in `spec.pxc.configuration` as follows:

```
pxc:
    configuration: |
    [mysqld]
    innodb_buffer_pool_size={{containerMemoryLimit * 3 / 4}}
    ...
```

Control Pod scheduling on specific Kubernetes nodes with affinity, anti-affinity and tolerations

The Operator automatically assigns Pods to nodes with sufficient resources for balanced distribution across the cluster. You can configure Pods to be scheduled on specific nodes. For example, for improved performance on the SSD equipped machine or for cost optimization by choosing the nodes in the same availability zone.

Using the deploy/cr.yaml Custom Resource manifest, you can configure the following:

- Node selection rules to ensure that Pods are scheduled only on Kubernetes nodes that have specific labels.
- Affinity and anti-affinity rules to bind Pods to specific Kubernetes nodes
- Taints and tolerations to ensure that Pods are not scheduled onto inappropriate nodes

Node selector

The `nodeSelector` field lets you specify a list of key-value labels that a node must have in order for a Pod to be scheduled on it. For example, to ensure Pods land on nodes with SSD storage or in specific availability zones.

If a node doesn't have all the labels you specify in `nodeSelector`, your Pod won't run on that node. In other words, `nodeSelector` acts like a filter so that only nodes with the right labels are eligible to host your Pod.

The following example binds the Pod to any node having a `disktype: ssd` label:

```
nodeSelector:
  disktype: ssd
```

Affinity and Anti-affinity

Affinity controls Pod placement based on nodes which already have Pods with specific labels. Use affinity to:

- Reduce costs by placing Pods in the same availability zone
- Improve high availability by distributing Pods across different nodes or zones

The Operator provides two approaches:

- Simple - set anti-affinity for Pods using built-in options. You can set anti-affinity for `pxc`, `haproxy`, and `proxysql` Pods, and for the backup storage of the S3 and PVC
- Advanced - using Kubernetes constraints

Simple Anti-affinity

This approach does not require the knowledge of how Kubernetes assigns Pods to specific nodes.

Use the `antiAffinityTopologyKey` option with these values:

- `kubernetes.io/hostname` - Pods avoid the same host
- `topology.kubernetes.io/zone` - Pods avoid the same zone
- `topology.kubernetes.io/region` - Pods avoid the same region
- `none` - No constraints applied

Example

This configuration ensures that Percona XtraDB Cluster Pods are not placed on the same node:

```
affinity:
  antiAffinityTopologyKey: "kubernetes.io/hostname"
```

Advanced anti-affinity via Kubernetes constraints

For complex scheduling requirements, use the `advanced` option. This disables the `antiAffinityTopologyKey` effect and allows the use of standard Kubernetes affinity constraints:

```
affinity:
   advanced:
     podAffinity:
       requiredDuringSchedulingIgnoredDuringExecution:
       - labelSelector:
           matchExpressions:
           - key: security
             operator: In
             values:
             - S1
         topologyKey: topology.kubernetes.io/zone
     podAntiAffinity:
       preferredDuringSchedulingIgnoredDuringExecution:
       - weight: 100
         podAffinityTerm:
           labelSelector:
             matchExpressions:
             - key: security
               operator: In
               values:
               - S2
           topologyKey: kubernetes.io/hostname
     nodeAffinity:
       requiredDuringSchedulingIgnoredDuringExecution:
         nodeSelectorTerms:
         - matchExpressions:
           - key: kubernetes.io/e2e-az-name
             operator: In
             values:
             - e2e-az1
             - e2e-az2
       preferredDuringSchedulingIgnoredDuringExecution:
       - weight: 1
         preference:
           matchExpressions:
           - key: another-node-label-key
             operator: In
             values:
             - another-node-label-value
```

See Kubernetes affinity documentation for detailed explanations of these options.

Tolerations

Tolerations allow Pods to run on nodes with matching taints. A taint is a key-value pair associated with a node that marks the node to repel certain Pods.

Taints and tolerations work together to ensure Pods are not scheduled onto inappropriate nodes.

A toleration includes these fields:

- `key` - The taint key to match
- `operator` - Either `exists` (matches any value) or `equal` (requires exact value match)
- `value` - Required when `operator` is `equal`
- `effect` - The taint effect to tolerate:
- `NoSchedule` - Pods cannot be scheduled on the node
- `PreferNoSchedule` - Pods are discouraged from scheduling on the node
- `NoExecute` - Pods are evicted from the node (with optional `tolerationSeconds`)

This is the example configuration of a toleration:

```
tolerations:
- key: "node.alpha.kubernetes.io/unreachable"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 6000
```

Common use cases

- **Dedicated nodes**: Reserve nodes for specific workloads by tainting them and adding corresponding tolerations to authorized Pods.
- **Special hardware**: Keep Pods that don't need specialized hardware (like GPUs) off dedicated nodes by tainting those nodes.
- **Node problems**: Handle node failures gracefully with automatic taints and tolerations.

See Kubernetes Taints and Tolerations for detailed examples and use cases.

Priority classes

Pods may belong to some *priority classes*. Priority classes help the scheduler distinguish important Pods when eviction is needed. To use priority classes:

1. Create PriorityClasses in your Kubernetes cluster
2. Specify `PriorityClassName` in the deploy/cr.yaml file:

```
priorityClassName: high-priority
```

See Kubernetes Pod Priority documentation for more information on how to define and use priority classes in your cluster.

Pod Disruption Budgets

A Pod Disruption Budget (PDB) in Kubernetes helps keep your applications available during voluntary disruptions, such as deleting a deployment or draining a node for maintenance by a cluster administrator. A Pod Disruption Budget sets a limit on how many Pods can be unavailable at the same time due to these voluntary actions.

You can configure Pod disruption budget for Percona XtraDB Cluster, ProxySQL and HAProxy Pods using the `podDisruptionBudget` option in the Custom Resource.

This is the example configuration for Percona XtraDB Cluster Pods:

```
pxc:
  podDisruptionBudget:
      maxUnavailable: 1
      minAvailable: 0
```

Refer to the official Kubernetes documentation for more information about Pod disruption budgets and considerations how to protect your application.

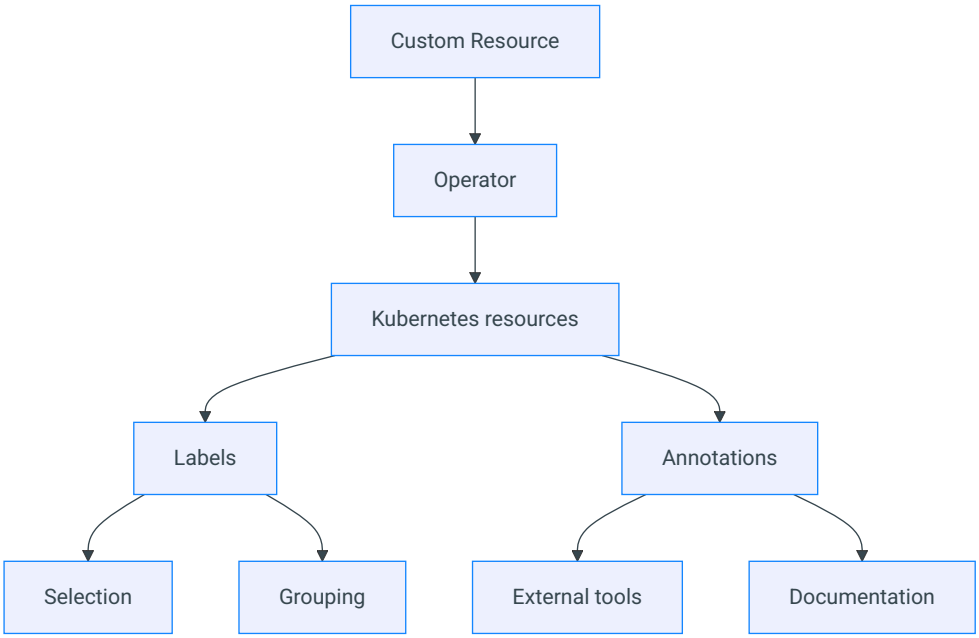
Labels and annotations

Labels and annotations are rather similar but differ in purpose.

Labels are used by Kubernetes to identify and select objects. They enable filtering and grouping, allowing users to apply selectors for operations like deployments or scaling.

Annotations are assigning additional *non-identifying* information that doesn't affect how Kubernetes processes resources. They store descriptive information like deployment history, monitoring configurations or external integrations.

The following diagram illustrates this difference:



Both Labels and Annotations are assigned to the following objects managed by Percona Operator for MySQL:

- Custom Resource Definitions
- Custom Resources
- Deployments
- Services
- StatefulSets
- PVCs
- Pods
- ConfigMaps and Secrets

When to use labels and annotations

Use **Labels** when:

- The information is used for object selection
- The data is used for grouping or filtering
- The information is used by Kubernetes controllers
- The data is used for operational purposes

Use **Annotations** when:

- The information is for external tools
- The information is used for debugging
- The data is used for monitoring configuration

Labels and annotations used by Percona Operator for MySQL

Labels

Name	Objects	Description	Example values
`app.kubernetes.io/name`	Services, StatefulSets, Deployments, etc.	Specifies the name of the application	percona-xtradb-cluster
`app.kubernetes.io/instance`	Pods, Services, StatefulSets, Deployments	Identifies a specific instance of the application	cluster1
`app.kubernetes.io/managed-by`	Services, StatefulSets	Indicates the controller managing the object	percona-xtradb-cluster-operator
`app.kubernetes.io/component`	Pods, Services, StatefulSets	Specifies the component within the application	pxc, proxysql, haproxy
`app.kubernetes.io/part-of`	Services, StatefulSets	Indicates the higher-level application the object belongs to	percona-xtradb-cluster
`app.kubernetes.io/version`	CustomResourceDefinition	Specifies the version of the Percona XtraDB Cluster Operator.	mysql.percona.com/1.19.0
`percona.com/cluster`	Custom Resource	Identifies the MySQL cluster instance	cluster1
`percona.com/backup-type`	Custom Resource	Specifies the type of backup being performed (e.g. cron for scheduled backups)	cron, xtrabackup
`percona.com/backup-name`	Custom Resource	Specifies the name of backup being performed	backup1
`percona.com/backup-job-name`	Job	Specifies the name of the backup job being performed	
`percona.com/backup-ancestor`	Custom Resource	Specifies the name of the backup that was used as a base for the current backup	cluster1-backup-2025-05-23
`percona.com/restore-svc-name`	Pods, PVC	Identifies resources associated with a specific restore operation	
`percona.com/restore-job-name`	Pods, Jobs	Specifies the name of a restore job being performed	
`rack`	Pods, Services, Deployments, StatefulSets	Identifies topology or rack awareness, often for scheduling or affinity	`rack-22`

Annotations

Name	Associated resources	Description	Example values
`iam.amazonaws.com/role`	Custom Resource	AWS IAM role for service account	`iam.amazonaws.com/role: role-arn`
`testName`	Backup jobs, Pods	Used for test identification in scheduled backups	`testName: scheduled-backup`
`percona.com/last-applied-tls`	Services	Stores the hash of the last applied TLS configuration for the service	
`percona.com/last-applied-secret`	Secrets	Stores the hash of the last applied user Secret configuration	
`percona.com/configuration-hash`	Services	Used to track and validate configuration changes in the MySQL cluster components	`percona.com/last-applied-secret: "hashvalue"`

`percona.com/last-config-hash`	Services	Stores the hash of the most recent configuration	
`percona.com/passwords-updated`	Secrets	Indicates when passwords were last updated in the Secret	
`percona.com/issue-vault-token`	Custom Resource	Signals the Operator to pause a cluster startup until a Vault token has been issued. Once the annotation is removed, the Operator restarts the cluster to apply the new Vault configuration and activate encryption	`percona.com/issue-vault-token: "true"`
`service.beta.kubernetes.io/aws-load-balancer-backend-protocol`	Services	Specifies the protocol for AWS load balancers	http, http-test
`service.beta.kubernetes.io/aws-load-balancer-backend`	Services	Specifies the backend type for AWS load balancers	test-type
`percona.com/headless-service`	Services	Exposes ProxySQL or HAProxy as a headless service	true
`percona.com/unsafe-pitr`	Restore object	Forces a restore from a backup marked as not appropriate for point-in-time recovery. This is unsafe configuration so you should use it when you are absolutely sure in your actions.	

Setting labels and annotations in the Custom Resource

You can define both Labels and Annotations as `key-value` pairs in the metadata section of a YAML manifest for a specific resource. For example, specifying labels and annotations in the `deploy/cr.yaml` Custom Resource looks as follows:

```
apiVersion: pxc.percona.com/v1
kind: PerconaXtraDBCluster
metadata:
  name: cluster1
  annotations:
    percona.com/issue-vault-token: "true"
  labels:
    ...
```

Note

The `percona.com/issue-vault-token: "true"` annotation is a special case. If you set this annotation and use HashiCorp Vault (for example, for data at rest encryption), the Operator will pause the cluster startup and repeatedly log `wait for token issuing` until you remove the annotation. This lets you automate Vault setup before the cluster starts.

Querying labels and annotations

To check which **labels** are attached to a specific object, use the additional `--show-labels` option of the `kubectl get` command.

For example, to see the Operator version associated with a Custom Resource Definition, use the following command:

```
$ kubectl get crd perconaxtradbclusters.pxc.percona.com --show-labels
```

Sample output

```
NAME                                  CREATED AT            LABELS
perconaxtradbclusters.pxc.percona.com   2025-07-29T09:02:34Z   app.kubernetes.io/component=crd,app.kubernetes.io/name=percona-xtradb-
cluster,app.kubernetes.io/part-of=percona-xtradb-cluster-operator,app.kubernetes.io/version=v1.19.0
```

To check **annotations** associated with an object, use the following command:

```
$ kubectl get <resource> <resource-name> -o jsonpath='{.metadata.annotations}'
```

For example:

```
$ kubectl get pod cluster1-pxc-0 -o jsonpath='{.metadata.annotations}'
```

Specifying labels and annotations ignored by the Operator

Sometimes various Kubernetes flavors can add their own annotations to the objects managed by the Operator.

The Operator keeps track of all changes to its objects and can remove annotations that it didn't create.

If there are no annotations or labels in the Custom Resource, the Operator does nothing if a new label or an annotation is added to the object.

If there is an annotation or a label specified in the Custom Resource, the Operator starts to manage annotations and labels. In this case it removes unknown annotations and labels.

A cloud provider can add own labels and annotations. Or you may have custom automation tools that add own labels or annotations and you need to keep them. To do this, you can specify which annotations and labels the Operator should ignore by listing them in the `spec.ignoreAnnotations` or `spec.ignoreLabels` keys of the `deploy/cr.yaml`, as follows:

```
spec:
  ignoreAnnotations:
    - some.custom.cloud.annotation/smth
  ignoreLabels:
    - some.custom.cloud.label/smth
...
```

The Operator will ignore any Service annotation or label, key of which **starts** with the mentioned above examples. For example, the following annotations and labels will be ignored after applying the above `cr.yaml` fragment:

```
annotations:
  some.custom.cloud.annotation/smth: somethinghere
labels:
  some.custom.cloud.label/smth: somethinghere
```

Local Storage support for the Percona Operator for MySQL

Among the wide rage of volume types, available in Kubernetes, there are some which allow Pod containers to access part of the local filesystem on the node. Two such options provided by Kubernetes itself are *emptyDir* and *hostPath* volumes. More comprehensive setups require additional components, such as OpenEBS Container Attached Storage solution

emptyDir

The name of this option is self-explanatory. When Pod having an emptyDir volume is assigned to a Node, a directory with the specified name is created on this node and exists until this Pod is removed from the node. When the Pod have been deleted, the directory is deleted too with all its content. All containers in the Pod which have mounted this volume will gain read and write access to the correspondent directory.

The `emptyDir` options in the deploy/cr.yaml file can be used to turn the emptyDir volume on by setting the directory name.

hostPath

A hostPath volume mounts some existing file or directory from the node's filesystem into the Pod.

The `volumeSpec.hostPath` subsection in the deploy/cr.yaml file may include `path` and `type` keys to set the node's filesystem object path and to specify whether it is a file, a directory, or something else (e.g. a socket):

```
volumeSpec:
  hostPath:
    path: /data
    type: Directory
```

Please note, that hostPath directory is not created automatically! It should be created manually on the node's filesystem. Also, it should have the attributives (access permissions, ownership, SELinux security context) which would allow Pod to access the correspondent filesystem objects according to pxc.containerSecurityContext and pxc.podSecurityContext.

`hostPath` is useful when you are able to perform manual actions during the first run and have strong need in improved disk performance. Also, please consider using tolerations to avoid cluster migration to different hardware in case of a reboot or a hardware failure.

More details can be found in the official hostPath Kubernetes documentation.

OpenEBS Local Persistent Volume Hostpath

Both *emptyDir* and *hostPath* volumes do not support Dynamic Volume Provisioning. Options that allow combining Dynamic Volume Provisioning with Local Persistent Volumes are provided by OpenEBS. Particularly, OpenEBS Local PV Hostpath allows creating Kubernetes Local Persistent Volumes using a directory (Hostpath) on the node. Such volume can be further accessed by applications via Storage Class and PersistentVolumeClaim.

Using it involves the following steps.

1. Install OpenEBS on your system along with the official installation guide.
2. Define a new Kubernetes Storage Class with OpenEBS with the YAML file (e. g. `local-hostpath.yaml`) as follows:

   ```
   apiVersion: storage.k8s.io/v1
   kind: StorageClass
   metadata:
     name: localpv
     annotations:
       openebs.io/cas-type: local
       cas.openebs.io/config: |
         - name: StorageType
           value: hostpath
         - name: BasePath
           value: /var/local-hostpath
   provisioner: openebs.io/local
   reclaimPolicy: Delete
   volumeBindingMode: WaitForFirstConsumer
   ```

   Two things to edit in this example are the `metadata.name` key (you will use it as a storage class name) and the `value` option under the `cas.openebs.io/config` (it should point to an already existing directory on the local filesystem of your node).

   When ready, apply the file with the `kubectl apply -f local-hostpath.yaml` command.

3. Now you can deploy the Operator and Percona XtraDB Cluster using this StorageClass in `deploy/cr.yaml`:

```
...
volumeSpec:
  persistentVolumeClaim:
    storageClassName: localpv
    accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 200Gi
```

Note

There are other storage options provided by the OpenEBS, which may be helpful within your cluster setup. Look at the OpenEBS for the Management of Kubernetes Storage Volumes blog post for more examples. Also, consider looking at the Measuring OpenEBS Local Volume Performance Overhead in Kubernetes post.

Define environment variables

Configure environment variables

You can configure environment variables in Percona Operator for XtraDB Cluster for the following purposes:

1. Operator environment variables - To control the Operator's behavior, such as logging, telemetry, and backup operations. These are set directly in the Operator Deployment.
2. Cluster component environment variables - To customize the behavior of cluster components (Percona XtraDB Cluster, HAProxy, ProxySQL). These are stored in Kubernetes Secrets and referenced in your cluster configuration.

When to use environment variables

Type	Use cases
Operator environment variables	- Control logging for better debugging and log aggregation - Manage telemetry - Optimize backup performance by setting the number of concurrent S3 workers - Configure the number of namespaces for the Operator to watch - Adjust concurrent reconciliation operations for better performance in multi-cluster environments
Cluster component environment variables	- Customize HAProxy - Optimize MySQL performance via alternative memory allocators like jemalloc - Configure ProxySQL - Configure monitoring and observability settings in PMM Client - Handle network policies

Configure Operator environment variables

You can configure the Percona Operator for XtraDB Cluster behavior by setting environment variables in the Operator Deployment. You can set environment variables in the following ways:

- For installations via `kubectl`, edit the Operator Deployment manifest (`deploy/operator.yaml`) before applying it, or modify the existing Deployment using `kubectl patch` or `kubectl edit`.
- For Helm installations you can set environment variables through Helm values.
- For installations on OpenShift, you can configure environment variables through the OLM subscription.

Available environment variables

`LOG_STRUCTURED`

Controls whether Operator logs are structured (JSON format) or plain text. Available since Operator version 1.12.0

Value type	Default	Example
string	`"false"`	`"true"`

When set to `"true"`, the Operator outputs logs in structured JSON format, which is useful for log aggregation systems. When set to `"false"` (default), logs are in plain text format.

Example configuration:

```
env:
 - name: LOG_STRUCTURED
   value: "true"
```

`LOG_LEVEL`

Sets the verbosity level of Operator logs. Available since Operator version 1.12.0

Value type	Default	Example
string	`"INFO"`	`"DEBUG"`

Valid values are:

- `"DEBUG"` - Most verbose, includes detailed debugging information
- `"INFO"` - Standard informational messages (default)
- `"WARN"` - Warning messages only
- `"ERROR"` - Error messages only

Example configuration:

```
env:
 - name: LOG_LEVEL
   value: "DEBUG"
```

`WATCH_NAMESPACE`

Specifies which namespaces the Operator should watch for Custom Resources.

Value type	Default	Example
string	Operator's namespace	`"pxc,pxc-dev"` or `""`

- If set to a comma-separated list of namespaces, the Operator watches only those namespaces (cluster-wide mode)

- If set to an empty string (`""`), the Operator watches all namespaces in the cluster
- If not set, the Operator watches only its own namespace

Example configuration for cluster-wide mode:

```
env:
 - name: WATCH_NAMESPACE
   value: "pxc,pxc-dev,pxc-prod"
```

See Cluster-wide installation for more details.

`DISABLE_TELEMETRY`

Disables the Operator's telemetry data collection.

Value type	Default	Example
string	`"false"`	`"true"`

When set to `"true"`, the Operator does not send anonymous telemetry data to Percona.

Example configuration:

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

See Telemetry for more information about what data is collected.

`S3_WORKERS_LIMIT`

This variable limits the number of parallel workers used for backup deletion from the S3-compatible storage (AWS S3, MinIO, etc.). Available since Operator version 1.8.0.

When to use:

- **High backup volume environments**: When you have many backups that need to be deleted, increasing this value can speed up cleanup operations
- **Backup deletion accumulation**: If you see repeated log messages like `"all workers are busy - skip backup deletion for now"`, consider increasing this value
- **Network saturation**: If you want to limit S3 operations to avoid saturating your network, decrease this value

Considerations:

- Lowering the value throttles all S3 operations (both uploads and deletions)
- Raising the value allows more concurrent operations but may increase network usage and Operator memory consumption
- Only positive integer values are accepted
- Setting this too high in environments with many backups may contribute to increased memory usage

Value type	Default	Example
string	`"10"`	`"20"`

Example configuration:

```
env:
 - name: S3_WORKERS_LIMIT
   value: "20"
```

`MAX_CONCURRENT_RECONCILES`

Controls the maximum number of concurrent reconciliation operations the Operator can perform.

Value type	Default	Example
string	`"1"`	`"3"`

This variable limits how many Custom Resources the Operator reconciles simultaneously. Increasing this value can improve performance in environments with many clusters, but may also increase resource usage.

Read more about concurrent reconciling in Configure concurrency for a cluster reconciliation chapter.

Example configuration:

```
env:
 - name: MAX_CONCURRENT_RECONCILES
   value: "3"
```

`PXCO_FEATURE_GATES`

Enables you to turn on specific features for the Operator.

Value type	Default	Example
string	`""` (empty)	`"XtrabackupSidecar=false"`

Supported values:

- `XtrabackupSidecar` - Enables the XtraBackup sidecar method for backups instead of the default SST (State Snapshot Transfer) method. Read more about backup methods the Operator uses. Disabled by default.

When to use:

Using the XtraBackup sidecar method is an alternative backup approach with different characteristics. You might want to use it if:

- You need better performance for large databases. The XtraBackup sidecar accesses data files directly without network overhead
- You require native encryption support for backups and/or incremental backups. These functionalities are not yet available in version 1.19.0 and will be added in future releases.

Example configuration:

Set the `PXCO_FEATURE_GATES` environment variable in the Operator Deployment:

```
env:
 - name: PXCO_FEATURE_GATES
   value: "XtrabackupSidecar=true"
```

Warning

If you set the environment variable during runtime, this causes the rolling restart of all database clusters managed by the Operator.

Important considerations for enabling the `XtrabackupSidecar`:

- PVC (Persistent Volume Claim) backups are not supported. Only cloud storage backups (S3, Azure, GCP) are available. PVC support will be added in future releases.
- This functionality affects all clusters managed by the Operator. You cannot enable it for specific clusters only.
- The Operator injects an XtraBackup sidecar container into each PXC Pod.
- The sidecar exposes a gRPC interface on port 6450 that handles backup requests.

Automatic environment variables

The following environment variables are automatically set by Kubernetes and should not be manually configured:

- `POD_NAME` - The name of the Operator Pod (set from `metadata.name`)
- `OPERATOR_NAME` - The name of the Operator (set to `percona-xtradb-cluster-operator`)

Update environment variables

Using kubectl patch

You can update environment variables in an existing Operator Deployment by applying a patch. To keep existing environment variables, you must specify the full list of them.

Here's how to do it:

1. Get the current environment variables:

```
kubectl get deployment percona-xtradb-cluster-operator -o jsonpath='{.spec.template.spec.containers[?(@.name=="percona-xtradb-cluster-operator")].env}'
```

2. Edit the output to add or update your variable (e.g., `S3_WORKERS_LIMIT`), then use the full list in your patch:

```
kubectl patch deployment percona-xtradb-cluster-operator \
  -p '{"spec":{"template":{"spec":{"containers":[{"name":"percona-xtradb-cluster-operator","env":[
    {"name":"POD_NAME","valueFrom":{"fieldRef":{"fieldPath":"metadata.name"}}},
    {"name":"OPERATOR_NAME","value":"percona-xtradb-cluster-operator"},
    {"name":"S3_WORKERS_LIMIT","value":"20"}
  ]}]}}}}'
```

Using kubectl edit

You can also edit the Deployment directly:

```
$ kubectl edit deployment percona-xtradb-cluster-operator
```

Then modify the `env` section in the container specification.

Using Helm

For Helm installations, you can set environment variables through Helm values. Refer to the Helm chart documentation for the specific syntax.

After the update

After modifying environment variables, the Operator Pod will be automatically restarted to apply the new configuration.

Define environment variables for cluster components

Cluster component environment variables let you customize the behavior of Percona XtraDB Cluster, HAProxy, and ProxySQL. These variables are stored in Kubernetes Secrets and referenced in your Custom Resource configuration. Below you'll find practical examples and steps for configuration.

Configure HAProxy environment variables

The following environment variables are available for HAProxy:

- `HA_CONNECTION_TIMEOUT`: Sets the timeout (in milliseconds) for HAProxy health checks on XtraDB Cluster nodes. The default is 10000 milliseconds (10 seconds), but you can increase this value for unstable Kubernetes networking or if you experience soft lockups on nodes.
- `OK_IF_DONOR`: Allows applications to connect to XtraDB Cluster donor nodes (nodes running backups). Enable if only one node is available and a second node is joining the cluster via SST.
- `HA_SERVER_OPTIONS`: Provides custom options for servers in the HAProxy configuration file. The default is `check inter 30000 rise 1 fall 5 weight 1`. You can add or adjust options as described in the HAProxy documentation.
- `PEER_LIST_SRV_PROTOCOL`: The protocol (TCP or UDP) for the Operator to use for peer-list SRV lookups. TCP is useful in large clusters with many nodes where peer-list SRV lookup returns large DNS responses or when DNS over UDP is blocked by network policies. You can configure the protocol for both HAProxy and ProxySQL.

Encode environment variable values

All environment variable values must be base64-encoded in the Secret's `data` section.

To encode a value, run this command:

On Linux

```
echo -n "1000" | base64 --wrap=0
```

On macOS

```
echo -n "1000" | base64
```

To verify or decode an encoded value:

```
$ echo "MTAwMA==" | base64 --decode
1000
```

Create a Secret

Create a Kubernetes Secret with your encoded environment variables. Here's an example Secret manifest:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-env-var-secrets
type: Opaque
data:
  HA_CONNECTION_TIMEOUT: MTAwMA==
  OK_IF_DONOR: MQ==
  HA_SERVER_OPTIONS: Y2hlY2sgaW50ZXIgMzAwMDAgcmlzZSAxIGZhbGwgNSB3ZWlnaHQgMQ==
  PEER_LIST_SRV_PROTOCOL: dGNw
```

Save this manifest to a file (for example, `deploy/my-env-secret.yaml`) and create the Secret:

```
$ kubectl create -f deploy/my-env-secret.yaml
```

Apply the configuration

1. Add the Secret name to your cluster configuration. Edit the `deploy/cr.yaml` file and add the `envVarsSecret` key to the `haproxy` section:

```
haproxy:
  envVarsSecret: my-env-var-secrets
```

2. Apply the updated configuration:

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

Configure alternative memory allocator

You can use an alternative memory allocator library for `mysqld` to optimize memory usage. This is often recommended when memory usage is higher than expected.

The Percona XtraDB Cluster Pods include the jemalloc allocator. You can enable it with the `LD_PRELOAD` environment variable.

Encode the LD_PRELOAD value

The value for `LD_PRELOAD` is `/usr/lib64/libjemalloc.so.1`. Encode it using base64:

On Linux

```
echo -n "/usr/lib64/libjemalloc.so.1" | base64 --wrap=0
```

On macOS

```
echo -n "/usr/lib64/libjemalloc.so.1" | base64
```

Create the memory allocator Secret

Create a Kubernetes Secret with the encoded value:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-new-env-var-secrets
type: Opaque
data:
  LD_PRELOAD: L3Vzci9saWI2NC9saWJqZW1hbGxvYy5zby4x
```

Save this manifest to a file (for example, `deploy/my-new-env-var-secret.yaml`) and create the Secret:

```
$ kubectl create -f deploy/my-new-env-var-secret.yaml
```

Apply the memory allocator configuration

1. Add the Secret to the PXC section in your `deploy/cr.yaml` file:

```
pxc:
  envVarsSecret: my-new-env-var-secrets
```

2. Apply the updated configuration:

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

Configure load balancing

Configure load balancing

Load balancing is distributing database connections and queries across multiple cluster nodes. This is crucial for ensuring high availability, optimal performance, and seamless scaling by preventing any single node from becoming a bottleneck or point of failure. Load balancing guarantees continued access to the database during node failures and maximizes the use of the cluster's combined resources.

You can use either HAProxy or ProxySQL for load balancing and proxy services in Percona Operator for MySQL. This guide helps you understand the differences between these proxies and choose the right one for your deployment.

You can control which proxy to use by enabling or disabling the `haproxy.enabled` and `proxysql.enabled` options in the `deploy/cr.yaml` configuration file.

HAProxy

HAProxy serves as a TCP-level load balancer in Percona Operator for MySQL. It sits in front of your Percona XtraDB Cluster, accepts incoming MySQL connections and distributes them evenly across available cluster nodes.

HAProxy is lightweight and efficient and introduces very little overhead. It has a minimal and straightforwatd configuration which makes it fast and stable.

HAProxy is not SQL-aware. This means it does not inspect or interpret SQL queries, so it cannot differentiate between reads (SELECT) and writes (INSERT, UPDATE, etc.). All traffic is routed based only on network and health status.

You can configure HAProxy to route write requests to the primary node and read requests - to the replica nodes. But you must also adjust your client applications to send read and read/write requests to different HAProxy ports.

The failover of cluster nodes is handled by the Operator. When a node fails, the Operator detects it and removes it from the HAProxy backend rotation.

ProxySQL

ProxySQL is an advanced, SQL-aware proxy included in Percona Operator for MySQL. It sits between applications and the database, providing more intelligent routing and management of queries.

ProxySQL examines each incoming SQL query and determines whether it is a read (e.g., SELECT) or a write (INSERT, UPDATE, DELETE, or SELECT FOR UPDATE). Then it routes queries as follows, achieving effective read/write splitting:

- read requests are routed either to all cluster nodes or only to replica nodes, depending on your configuration.
- write queries are routed only to the writer node.

Health and cluster topology awareness are built in, so ProxySQL can adapt to failover situations.

ProxySQL also offers features such as query rules, result caching, connection pooling, and multiplexing—helping to optimize database performance and reliability for complex workloads.

What load balancer to use?

Load balancer	Use cases
HAProxy	- You need simple, reliable load balancing without query-level logic - Your workload is write-heavy or doesn't benefit much from read scaling - You want minimal overhead and maximum throughput - You prefer straightforward deployments, smaller clusters, or environments where simplicity is key
ProxySQL	- You need read/write splitting to scale reads across replicas - Your workload is mixed (lots of SELECT queries alongside writes) - You want advanced features like query caching, multiplexing, or routing based on query rules - You're deploying larger clusters or complex workloads that need fine-grained traffic control

In a nutshell, HAProxy is ideal for scenarios where you need a lightweight, stable proxy that efficiently routes connections without analyzing SQL queries. According to benchmark comparisons, HAProxy excels in resource efficiency and connection throughput, making it the preferred choice for Kubernetes environments where resource optimization is crucial.

ProxySQL is ideal when you cannot modify your application code to separate read and write connections, but still want to benefit from read scaling. ProxySQL requires no changes to application code — you send all queries to ProxySQL and configure regex rules to route SELECT queries appropriately.

Next steps

- Configure HAProxy — Learn how to configure and customize HAProxy for your cluster
- Configure ProxySQL — Learn how to configure and customize ProxySQL

Configuring Load Balancing with HAProxy

This page describes how to configure and customize HAProxy for your Percona XtraDB Cluster. For information about choosing between HAProxy and ProxySQL, see Configure load balancing.

Use the following command to enable HAProxy:

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
  "spec": {
     "haproxy": {
        "enabled": true,
        "size": 3,
        "image": "percona/percona-xtradb-cluster-operator:1.19.0-haproxy" },
     "proxysql": { "enabled": false }
  }}'
```

Warning

Switching from ProxySQL to HAProxy will cause the downtime because the Operator needs to reconfigure the proxy Pods.

HAProxy services

The Operator creates two services for HAProxy:

`cluster1-haproxy` service

The `cluster1-haproxy` service listens on the following ports:

- `3306` is the default MySQL port. It is used by the mysql client, MySQL Connectors, and utilities such as mysqldump and mysqlpump
- `3309` is the proxy protocol port. Proxy protocol is used to store the client's IP address
- `33062` is the port to connect to the MySQL Administrative Interface
- `33060` is the port for the MySQLX protocol. It is supported by clients such as MySQL Shell, MySQL Connectors and MySQL Router
- `8404` is the port to connect to the HAProxy statistics page

The haproxy.enabled Custom Resource option enables or disables `cluster1-haproxy` service.

By default, the `cluster1-haproxy` service points to the first Percona XtraDB Cluster member (`cluster1-pxc-0`), when this member is available. If it is not available, members are selected in descending order of their numbers: `cluster1-pxc-2`, then `cluster1-pxc-1`. This service can be used for both read and write load, or it can also be used just for write load (single writer mode) in setups with split write and read loads.

The haproxy.exposePrimary.enabled Custom Resource option enables or disables the `cluster1-haproxy` service.

`cluster1-haproxy-replicas` service

The `cluster1-haproxy-replicas` service listens on port 3306 (MySQL).

This service selects Percona XtraDB Cluster members to serve queries following the Round Robin load balancing algorithm.

Don't use it for write requests.

The haproxy.exposeReplicas.enabled Custom Resource option enables or disables `cluster1-haproxy-replicas` service (on by default).

Expose HAProxy as a headless service

You may want to configure the HAProxy service as a headless Service. For example, if you have applications that need direct DNS access to individual HAProxy pods, such as when running in a multi-tenant setup or when handling advanced networking scenarios.

To enable HAProxy as a headless service, add the `percona.com/headless-service: true` annotation to the following options in the Custom Resource:

- `haproxy.exposePrimary.annotations` key to expose the primary HAProxy Pod
- `haproxy.exposeReplicas.annotations` key to expose the HAProxy replica Pods

```
spec:
  haproxy:
     exposePrimary:
       enabled: true
       annotations:
         percona.com/headless-service: true
       ....
     exposeReplicas:
       enabled: true
       annotations:
         percona.com/headless-service: true
```

This annotation works only at service creation time and can't be added later.

Upgrade behavior

When the cluster with HAProxy is upgraded, the following steps take place. First, reader members are upgraded one by one: the Operator waits until the upgraded Percona XtraDB Cluster member becomes synced, and then proceeds to upgrade the next member. When the upgrade is finished for all the readers, then the writer Percona XtraDB Cluster member is finally upgraded.

Exposing HAProxy

You can expose HAProxy, so that clients can connect to your database cluster from the outside. To do so, you need to set the service type `LoadBalancer` for the `haproxy-primary` service.

By default, the HAProxy is available for all clients. If you need to restrict the client IP addresses from which the load balancer should be reachable, list these IP addresses in the `loadBalancerSourceRanges` option.

Edit the `deploy/cr.yaml` Custom Resource manifest and specify the following configuration:

```
spec:
  haproxy:
    exposePrimary:
      type: LoadBalancer
      loadBalancerSourceRanges:
        - 10.0.0.0/8
```

Note that the `haproxy-replica` service inherits this setup. You can override it for the `haproxy-replica` service by setting the IP ranges to access the cluster for read requests. The configuration for the `haproxy-replica` service will be as follows:

```
spec:
  haproxy:
    enabled: true
    exposeReplicas:
      enabled: true
      type: LoadBalancer
```

Passing custom configuration options to HAProxy

You can pass custom configuration to HAProxy in one of the following ways:

- edit the `deploy/cr.yaml` file,
- use a ConfigMap,
- use a Secret object.

Note

If you specify a custom HAProxy configuration in this way, the Operator doesn't provide its own HAProxy configuration file except several hardcoded options (which therefore can't be overwritten). That's why you should specify either a full set of configuration options or nothing. Additionally, when upgrading Percona XtraDB Cluster it would be wise to check the HAProxy configuration file provided by the Operator and make sure that your custom config is still compatible with the new variant.

Edit the `deploy/cr.yaml` file

You can add options from the haproxy.cfg configuration file by editing `haproxy.configuration` key in the `deploy/cr.yaml` file. Here is an example:

```
...
haproxy:
    enabled: true
    size: 3
    image: percona/percona-xtradb-cluster-operator:1.19.0-haproxy
    configuration: |
      global
        maxconn 2048
        external-check
        stats socket /var/run/haproxy.sock mode 600 expose-fd listeners level user
      defaults
        log global
        mode tcp
        retries 10
        timeout client 10000
        timeout connect 100500
        timeout server 10000
      frontend galera-in
        bind *:3309 accept-proxy
        bind *:3306
        mode tcp
        option clitcpka
        default_backend galera-nodes

      frontend galera-admin-in
        bind *:33062
        mode tcp
        option clitcpka
        default_backend galera-admin-nodes

      frontend galera-replica-in
        bind *:3307
        mode tcp
        option clitcpka
        default_backend galera-replica-nodes

      frontend galera-mysqlx-in
        bind *:33060
        mode tcp
        option clitcpka
        default_backend galera-mysqlx-nodes

      frontend stats
        bind *:8404
        mode http
        http-request use-service prometheus-exporter if { path /metrics }
```

Use a ConfigMap

You can use a configmap and the cluster restart to reset configuration options. A configmap allows Kubernetes to pass or update configuration data inside a containerized application.

Use the `kubectl` command to create the configmap from external resources, for more information see Configure a Pod to use a ConfigMap.

For example, you define a `haproxy.cfg` configuration file with the following setting:

```
global
  maxconn 2048
  external-check
  stats socket /var/run/haproxy.sock mode 600 expose-fd listeners level user
defaults
  log global
  mode tcp
  retries 10
  timeout client 10000
  timeout connect 100500
  timeout server 10000
frontend galera-in
  bind *:3309 accept-proxy
  bind *:3306
  mode tcp
  option clitcpka
  default_backend galera-nodes
frontend galera-admin-in
  bind *:33062
  mode tcp
  option clitcpka
  default_backend galera-admin-nodes
frontend galera-replica-in
  bind *:3307
  mode tcp
  option clitcpka
  default_backend galera-replica-nodes
frontend galera-mysqlx-in
  bind *:33060
  mode tcp
  option clitcpka
  default_backend galera-mysqlx-nodes
frontend stats
  bind *:8404
  mode http
  http-request use-service prometheus-exporter if { path /metrics }
```

You can create a configmap from the `haproxy.cfg` file with the `kubectl create configmap` command.

You should use the combination of the cluster name with the `-haproxy` suffix as the naming convention for the configmap. To find the cluster name, you can use the following command:

```
$ kubectl get pxc
```

The syntax for `kubectl create configmap` command is:

```
kubectl create configmap <cluster-name>-haproxy <resource-type=resource-name>
```

The following example defines `cluster1-haproxy` as the configmap name and the `haproxy.cfg` file as the data source:

```
$ kubectl create configmap cluster1-haproxy --from-file=haproxy.cfg
```

To view the created configmap, use the following command:

```
$ kubectl describe configmaps cluster1-haproxy
```

Use a Secret Object

The Operator can also store configuration options in Kubernetes Secrets. This can be useful if you need additional protection for some sensitive data.

You should create a Secret object with a specific name, composed of your cluster name and the `haproxy` suffix.

Note

To find the cluster name, you can use the following command:

```
$ kubectl get pxc
```

Configuration options should be put inside a specific key inside of the `data` section. The name of this key is `haproxy.cfg` for ProxySQL Pods.

Actual options should be encoded with Base64.

For example, let's define a `haproxy.cfg` configuration file and put there options we used in the previous example:

```
global
  maxconn 2048
  external-check
  stats socket /var/run/haproxy.sock mode 600 expose-fd listeners level user
defaults
  log global
  mode tcp
  retries 10
  timeout client 10000
  timeout connect 100500
  timeout server 10000
frontend galera-in
  bind *:3309 accept-proxy
  bind *:3306
  mode tcp
  option clitcpka
  default_backend galera-nodes
frontend galera-admin-in
  bind *:33062
  mode tcp
  option clitcpka
  default_backend galera-admin-nodes
frontend galera-replica-in
  bind *:3307
  mode tcp
  option clitcpka
  default_backend galera-replica-nodes
frontend galera-mysqlx-in
  bind *:33060
  mode tcp
  option clitcpka
  default_backend galera-mysqlx-nodes
frontend stats
  bind *:8404
  mode http
  http-request use-service prometheus-exporter if { path /metrics }
```

You can get a Base64 encoded string from your options via the command line as follows:

in Linux

```
$ cat haproxy.cfg | base64 --wrap=0
```

in macOS

```
$ cat haproxy.cfg | base64
```

Note

Similarly, you can read the list of options from a Base64 encoded string:

```
$ echo "IGdsb2JhbAogICBtYXhjb25uIDIwNDgKICAgZXh0ZXJuYWwtY2hlY2sKICAgc3RhdHMgc29ja2V0\
  IC92YXIvcnVuL2hhcHJveHkuc29jayBtb2RlIDYwMCBleHBvc2UtZmQgbGlzdGVuZXJzIGxldmVs\
  IHVzZXIKIGRlZmF1bHRzCiAgIGxvZyBnbG9iYWwKICAgbW9kZSB0Y3AKICAgcmV0cmllcyAxMAog\
  ICB0aW1lb3V0IGNsaWVudCAxMDAwMAogICB0aW1lb3V0IGNvbm5lY3QgMTAwNTAwCiAgIHRpbWVv\
  dXQgc2VydmVyIDEwMDAwCiBmcm9udGVuZCBnYWxlcmEtaW4KICAgYmluZCAqOjMzMDkgYWNjZXB0\
  LXByb3h5CiAgIGJpbmQgKjozMzA2CiAgIG1vZGUgdGNwCiAgIG9wdGlvbiBjbGl0Y3BrYQogICBk\
  ZWZhdWx0X2JhY2tlbmQgZ2FsZXJhLW5vZGVzCiBmcm9udGVuZCBnYWxlcmEtcmVwbGljYS1pbgog\
  ICBiaW5kICo6MzMwOSBhY2NlcHQtcHJveHkKICAgYmluZCAqOjMzMDcKICAgbW9kZSB0Y3AKICAg\
  b3B0aW9uIGNsaXRjcGthCiAgIGRlZmF1bHRfYmFja2VuZCBnYWxlcmEtcmVwbGljYS1ub2Rlcwo=" | base64 --decode
```

Finally, use a yaml file to create the Secret object. For example, you can create a `deploy/my-haproxy-secret.yaml` file with the following contents:

```
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-haproxy
data:
  haproxy.cfg: "IGdsb2JhbAogICBtYXhjb25uIDIwNDgKICAgZXh0ZXJuYWwtY2hlY2sKICAgc3RhdHMgc29ja2V0\
     IC92YXIvcnVuL2hhcHJveHkuc29jayBtb2RlIDYwMCBleHBvc2UtZmQgbGlzdGVuZXJzIGxldmVs\
     IHVzZXIKIGRlZmF1bHRzCiAgIGxvZyBnbG9iYWwKICAgbW9kZSB0Y3AKICAgcmV0cmllcyAxMAog\
     ICB0aW1lb3V0IGNsaWVudCAxMDAwMAogICB0aW1lb3V0IGNvbm5lY3QgMTAwNTAwCiAgIHRpbWVv\
     dXQgc2VydmVyIDEwMDAwCiBmcm9udGVuZCBnYWxlcmEtaW4KICAgYmluZCAqOjMzMDkgYWNjZXB0\
     LXByb3h5CiAgIGJpbmQgKjozMzA2CiAgIG1vZGUgdGNwCiAgIG9wdGlvbiBjbGl0Y3BrYQogICBk\
     ZWZhdWx0X2JhY2tlbmQgZ2FsZXJhLW5vZGVzCiBmcm9udGVuZCBnYWxlcmEtcmVwbGljYS1pbgog\
     ICBiaW5kICo6MzMwOSBhY2NlcHQtcHJveHkKICAgYmluZCAqOjMzMDcKICAgbW9kZSB0Y3AKICAg\
     b3B0aW9uIGNsaXRjcGthCiAgIGRlZmF1bHRfYmFja2VuZCBnYWxlcmEtcmVwbGljYS1ub2Rlcwo="
```

When ready, apply it with the following command:

```
$ kubectl create -f deploy/my-haproxy-secret.yaml
```

Note

Do not forget to restart Percona XtraDB Cluster to ensure the cluster has updated the configuration.

Enabling the Proxy protocol

The Proxy protocol allows HAProxy to provide a real client address to Percona XtraDB Cluster.

Note

To use this feature, you should have a Percona XtraDB Cluster image version `8.0.21` or newer.

Normally Proxy protocol is disabled, and Percona XtraDB Cluster sees the IP address of the proxying server (HAProxy) instead of the real client address. But there are scenarios when making real client IP-address visible for Percona XtraDB Cluster is important: e.g. it allows to have privilege grants based on client/application address, and significantly enhance auditing.

You can enable Proxy protocol on Percona XtraDB Cluster by adding proxy_protocol_networks option to pxc.configuration key in the `deploy/cr.yaml` configuration file.

Note

Depending on the load balancer of your cloud provider, you may also need setting haproxy.externaltrafficpolicy option in `deploy/cr.yaml`.

More information about Proxy protocol can be found in the official HAProxy documentation.

Configuring load balancing with ProxySQL

This page describes how to configure and customize ProxySQL for your Percona XtraDB Cluster, including the advanced scheduler feature. For information about choosing between HAProxy and ProxySQL, see Configure load balancing.

`cluster1-proxysql` service

The `cluster1-proxysql` service listens on the following ports:

- `3306` is the default MySQL port. It is used by the mysql client, MySQL Connectors, and utilities such as `mysqldump` and `mysqlpump`
- `33062` is the port to connect to the MySQL Administrative Interface
- `6070` is the port to connect to the built-in Prometheus exporter to gather ProxySQL statistics and manage the ProxySQL observability stack

The `cluster1-proxysql` service uses the first Percona XtraDB Cluster member (`cluster1-pxc-0` by default) as the writer. Use the proxysql.expose.enabled Custom Resource option to enable or disable this service.

Headless ProxySQL service

You may want to configure the ProxySQL service as a headless Service. For example, if you have applications that need direct DNS access to individual ProxySQL pods, such as when running in a multi-tenant setup or when handling advanced networking scenarios.

To enable a headless ProxySQL service, add the `percona.com/headless-service: true` annotation in the `proxysql.expose.annotations` key of the `deploy/cr.yaml` file. Note that this annotation takes effect only at service creation time, so you need to set it when first creating the cluster.

```
spec:
  proxysql:
    expose:
      enabled: true
      annotations:
        percona.com/headless-service: true
  ...
```

Upgrade behavior

During cluster upgrades with ProxySQL, the Operator upgrades reader members one by one, waiting for each to show as online in ProxySQL before proceeding. After all readers are upgraded, the writer member is upgraded last. When both ProxySQL and Percona XtraDB Cluster are upgraded, they are upgraded in parallel.

Passing custom configuration options to ProxySQL

You can pass custom configuration to ProxySQL in these ways:

- by editing the `deploy/cr.yaml` file,
- by using a ConfigMap,
- by using a Secret object.

Regardless of which method you use, you must supply the full ProxySQL configuration, even if you are only modifying a single option.

Edit the `deploy/cr.yaml` file

Add options from the proxysql.cnf configuration file by editing the `proxysql.configuration` key in `deploy/cr.yaml`.

Here is an example:

```
...
proxysql:
  enabled: true
  size: 3
  image: percona/percona-xtradb-cluster-operator:1.19.0-proxysql
  configuration: |
    datadir="/var/lib/proxysql"

    admin_variables =
    {
        admin_credentials="admin:admin"
        mysql_ifaces="0.0.0.0:6032"
        refresh_interval=2000
        restapi_enabled=true
        restapi_port=6070

        cluster_username="admin"
        cluster_password="admin"
        checksum_admin_variables=false
        checksum_ldap_variables=false
        checksum_mysql_variables=false
        cluster_check_interval_ms=200
        cluster_check_status_frequency=100
        cluster_mysql_query_rules_save_to_disk=true
        cluster_mysql_servers_save_to_disk=true
        cluster_mysql_users_save_to_disk=true
        cluster_proxysql_servers_save_to_disk=true
        cluster_mysql_query_rules_diffs_before_sync=1
        cluster_mysql_servers_diffs_before_sync=1
        cluster_mysql_users_diffs_before_sync=1
        cluster_proxysql_servers_diffs_before_sync=1
    }

    mysql_variables=
    {
        monitor_password="monitor"
        monitor_galera_healthcheck_interval=1000
        threads=2
        max_connections=2048
        default_query_delay=0
        default_query_timeout=10000
        poll_timeout=2000
        interfaces="0.0.0.0:3306"
        default_schema="information_schema"
        stacksize=1048576
        connect_timeout_server=10000
        monitor_history=60000
        monitor_connect_interval=20000
        monitor_ping_interval=10000
        ping_timeout_server=200
        commands_stats=true
        sessions_sort=true
        have_ssl=false
        ssl_p2s_ca=""
        ssl_p2s_cert=""
        ssl_p2s_key=""
        ssl_p2s_cipher="ECDHE-RSA-AES128-GCM-SHA256"
        default_authentication_plugin="caching_sha2_password"
    }
```

Use a ConfigMap

A configmap allows Kubernetes to pass or update configuration data inside a containerized application. When you apply a ConfigMap, the cluster restarts.

See Configure a Pod to use a ConfigMap for information how to create a ConfigMap.

Here's the example configuration.

1. Create a `proxysql.cnf` configuration file:

```
datadir="/var/lib/proxysql"

admin_variables =
{
    admin_credentials="admin:admin"
    mysql_ifaces="0.0.0.0:6032"
    refresh_interval=2000
    restapi_enabled=true
    restapi_port=6070

    cluster_username="admin"
    cluster_password="admin"
    checksum_admin_variables=false
    checksum_ldap_variables=false
    checksum_mysql_variables=false
    cluster_check_interval_ms=200
    cluster_check_status_frequency=100
    cluster_mysql_query_rules_save_to_disk=true
    cluster_mysql_servers_save_to_disk=true
    cluster_mysql_users_save_to_disk=true
    cluster_proxysql_servers_save_to_disk=true
    cluster_mysql_query_rules_diffs_before_sync=1
    cluster_mysql_servers_diffs_before_sync=1
    cluster_mysql_users_diffs_before_sync=1
    cluster_proxysql_servers_diffs_before_sync=1
}

mysql_variables=
{
    monitor_password="monitor"
    monitor_galera_healthcheck_interval=1000
    threads=2
    max_connections=2048
    default_query_delay=0
    default_query_timeout=10000
    poll_timeout=2000
    interfaces="0.0.0.0:3306"
    default_schema="information_schema"
    stacksize=1048576
    connect_timeout_server=10000
    monitor_history=60000
    monitor_connect_interval=20000
    monitor_ping_interval=10000
    ping_timeout_server=200
    commands_stats=true
    sessions_sort=true
    have_ssl=false
    ssl_p2s_ca=""
    ssl_p2s_cert=""
    ssl_p2s_key=""
    ssl_p2s_cipher="ECDHE-RSA-AES128-GCM-SHA256"
    default_authentication_plugin="caching_sha2_password"
}
```

2. Find your cluster name:

```
$ kubectl get pxc
```

3. Create the ConfigMap using the cluster name with the `-proxysql` suffix:

```
$ kubectl create configmap cluster1-proxysql --from-file=proxysql.cnf
```

4. Verify the ConfigMap:

```
$ kubectl describe configmaps cluster1-proxysql
```

Use a Secret object

Store configuration options in Kubernetes Secrets for additional protection of sensitive data.

The Secret name must be composed of your cluster name and the `proxysql` suffix.

1. Find your cluster name:

```
$ kubectl get pxc
```

2. Create a `proxysql.cnf` configuration file with your options:

```
datadir="/var/lib/proxysql"

admin_variables =
{
    admin_credentials="admin:admin"
    mysql_ifaces="0.0.0.0:6032"
    refresh_interval=2000
    restapi_enabled=true
    restapi_port=6070

    cluster_username="admin"
    cluster_password="admin"
    checksum_admin_variables=false
    checksum_ldap_variables=false
    checksum_mysql_variables=false
    cluster_check_interval_ms=200
    cluster_check_status_frequency=100
    cluster_mysql_query_rules_save_to_disk=true
    cluster_mysql_servers_save_to_disk=true
    cluster_mysql_users_save_to_disk=true
    cluster_proxysql_servers_save_to_disk=true
    cluster_mysql_query_rules_diffs_before_sync=1
    cluster_mysql_servers_diffs_before_sync=1
    cluster_mysql_users_diffs_before_sync=1
    cluster_proxysql_servers_diffs_before_sync=1
}

mysql_variables=
{
    monitor_password="monitor"
    monitor_galera_healthcheck_interval=1000
    threads=2
    max_connections=2048
    default_query_delay=0
    default_query_timeout=10000
    poll_timeout=2000
    interfaces="0.0.0.0:3306"
    default_schema="information_schema"
    stacksize=1048576
    connect_timeout_server=10000
    monitor_history=60000
    monitor_connect_interval=20000
    monitor_ping_interval=10000
    ping_timeout_server=200
    commands_stats=true
    sessions_sort=true
    have_ssl=false
    ssl_p2s_ca=""
    ssl_p2s_cert=""
    ssl_p2s_key=""
    ssl_p2s_cipher="ECDHE-RSA-AES128-GCM-SHA256"
    default_authentication_plugin="caching_sha2_password"
}
```

3. Encode the configuration file with Base64:

in Linux

```
$ cat proxysql.cnf | base64 --wrap=0
```

in macOS

```
$ cat proxysql.cnf | base64
```

4. Create a Secret object with a name composed of your cluster name and the `proxysql` suffix. Put the Base64-encoded configuration in the `data` section under the `proxysql.cnf` key. Example `deploy/my-proxysql-secret.yaml`:

```
apiVersion: v1
kind: Secret
metadata:
    name: cluster1-proxysql
data:
    proxysql.cnf: "ZGF0YWRpcj0iL3Zhci9saWIvcHJveHlzcWwiCgphZG1pbl92YXJpYWJsZXMgPQp7CiBhZG1pbl9j\
     cmVkZW50aWFscz0icHJveHlhZG1pbjphZG1pbl9wYXNzd29yZCIKIG15c3FsX2lmYWNlcz0iMC4w\
     LjAuMDo2MDMyIgogcmVmcmVzaF9pbnRlcnZhbD0yMDAwCgogY2x1c3Rlcl91c2VybmFtZT0icHJv\
     eHlhZG1pbiIKIGNsdXN0ZXJfcGFzc3dvcmQ9ImFkbWluX3Bhc3N3b3JkIgogY2x1c3Rlcl9jaGVj\
     a19pbnRlcnZhbF9tcz0yMDAKIGNsdXN0ZXJfY2hlY2tfc3RhdHVzX2ZyZXF1ZW5jeT0xMDAsCiAg\
     dGVyX215c3FsX1ZXJ5X3J1bGVzX2RpZmZzX2J1Zm9yZV9zeW5jPTEKIGNsdXN0ZXJfcGFzc3dv\
     c2VydmVyc19pbl9wcm94eXNxbF9zZXJ2ZXJzX3NhdmVfdG9fZGlzaz10cnVlCiBjbHVz\
     c2VydmVyc19pbl9wcm94eXNxbF9zZXJfcHJveHlzcWxfc2VydmVyc19nZWZmcmVfc3luYz0x\
     Zm9yZV9zeW5jPTEKIGNsdXN0ZXJfcHJveHlzcWxfc2VydmVyc19kaWZmc19iZWZvcmVfc3luYz0x\
     Cn0KCm15c3FsX3ZhcmlhYmxlcz0KewogbW9uaXRvcl9wYXNzd29yZD0ibW9uaXRvciIKIG1vbml0\
     b3JfZ2FsZXJhX0aGNoZWNrX2ludGVydmFsPTEwMDAKIHRocmVhZHM9MgogbWF4X2Nvbm5l\
     Y3Rpb25zPTIwNDgKIGRlZmF1bHRfcXVlcnlfZGVsYXk9MAogZGVmYXVsdF9xdWVyeV90aW1lb3V0\
     PTEwMDAwCiBwb2xsX3RpbWVvdXQ9MjAwMAogaW50ZXJmYWNlcz0iMC4wLjAuMDozMzA2IgogZGVm\
     YXVsdF9zY2hlbWE9ImluZm9ybWF0aW9uX3NjaGVtYSIKIHN0YWNrc2l6ZT0xMDQ4NTc2CiBjb25u\
     ZWN0X3RpbWVvdXRfc2VydmVyPTEwMDAwCiBtb25pdG9yX2hpc3Rvcnk9NjAwMDAKIG1vbml0b3Jf\
     Y29ubmVjdF9pbnRlcnZhbD0yMDAwMAogbW9uaXRvcl9waW5nX2ludGVydmFsPTEwMDAwCiBwaW5n\
     X3RpbWVvdXRfc2VydmVyPTIwMAogY29tbWFuZHNfc3RhdHM9dHJ1ZQogc2hpc3Rvcnk9NjAwMDAK\
     cnVlCiBoYXZlX3NzbD10cnVlCiBzc2xfcDJzX2NhPSIvZXRjL3Byb3h5c3FsL3NzbC1pbnRlcm5h\
     bC9jYS5jcnQiCiBzc2xfcDJzX2NlcnQ9Ii9ldGMvcHJveHlzcWwvc3NsLWludGVybmFsL3Rscy5j\
     cnQiCiBzc2xfcDJzX2tleT0iL2V0Yy9wcm94eXNxbC9zc2wtaW50ZXJuYWwvdGxzLmtleSIKIHNz\
     bF9wMnNfY2lwaGVyPSJFQ0RIRS1SU0EtQUVTMTI4LUdDTS1TSEEyNTYiCn0K"
```

5. Apply the Secret:

```
$ kubectl create -f deploy/my-proxysql-secret.yaml
```

6. Restart Percona XtraDB Cluster to apply the configuration changes.

Accessing the ProxySQL Admin Interface

Use the ProxySQL admin interface to configure ProxySQL settings by connecting via the MySQL protocol.

1. Find the ProxySQL Pod name:

```
$ kubectl get pods
```

Sample output

```
NAME                                               READY   STATUS
RESTARTS   AGE
cluster1-pxc-node-0                                1/1     Running
0          5m
cluster1-pxc-node-1                                1/1     Running
0          4m
cluster1-pxc-node-2                                1/1     Running
0          2m
cluster1-proxysql-0                                1/1     Running
0          5m
percona-xtradb-cluster-operator-dc67778fd-qtspz    1/1     Running
0          6m
```

2. Get the admin password:

```
$ kubectl get secrets $(kubectl get pxc -o jsonpath='{.items[].spec.secretsName}') -o template='{{ .data.proxyadmin |
base64decode }}'
```

3. Connect to ProxySQL. Replace `cluster1-proxysql-0` with your Pod name and `admin_password` with the retrieved password:

```
$ kubectl exec -it cluster1-proxysql-0 -- mysql -h127.0.0.1 -P6032 -uproxyadmin -padmin_password
```

ProxySQL scheduler (tech preview)

By default, the Operator uses the internal ProxySQL scheduler for load balancing. In some cases, this scheduler may not fully recognize the cluster topology, directing both read and write traffic to the primary Pod. This can reduce scalability and efficiency and may increase the risk of overload and downtime.

To address this limitation, the Operator is integrated with the `pxc_scheduler_handler` tool starting with version 1.19.0. This external ProxySQL scheduler ensures the read/write splitting is distributed as follows:

- **SELECT queries** (without `FOR UPDATE`) are sent evenly to all PXC nodes or to all nodes except the primary, depending on your configuration
- **Non-SELECT queries** and **SELECT FOR UPDATE** queries are sent to the primary node
- The scheduler automatically manages the primary node, ensuring only one primary exists at a time

As a result, you achieve:

- Better performance through faster query processing and increased throughput
- Higher reliability by preventing single-node bottlenecks and points of failure
- Healthier cluster through early detection of replication lag and node issues
- Efficient resource utilization
- Improved user experience with consistent, predictable response times

Enable the scheduler

The scheduler is disabled by default to maintain backward compatibility. You can enable it by setting `proxysql.scheduler.enabled=true` in your Custom Resource.

1. Edit the `deploy/cr.yaml` file and add the scheduler configuration:

```
proxysql:
  enabled: true
  size: 3
  image: percona/percona-xtradb-cluster-operator:1.19.0-proxysql
  scheduler:
    enabled: true
```

2. Apply the configuration:

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

When the scheduler is enabled, you should see:

- **Hostgroup 10** (readers): All PXC nodes with weighted distribution
- **Hostgroup 11** (writer): Only the current writer node (typically `pod-0`) with a high weight (1000000)

To verify that ProxySQL is properly balancing read traffic across your Percona XtraDB Cluster nodes, you can run the following command. This script sends multiple `SELECT` queries through ProxySQL and then shows how the queries are distributed among cluster nodes.

Replace the `<user>` and `<password>` placeholders with the valid credentials of a MySQL user that has permissions to connect through ProxySQL and can execute simple `SELECT` statements on the cluster.

```
$ for i in $(seq 100); do
    kubectl exec -i cluster1-pxc-0 -c pxc -- mysql -u<user> -p<password> \
      --host cluster1-proxysql -Ne "SELECT VARIABLE_VALUE FROM \
      performance_schema.global_variables WHERE VARIABLE_NAME = 'wsrep_node_name' LIMIT 1" \
      2>/dev/null
  done | sort -n | uniq -c
```

You should see queries reported from multiple node names, indicating that read load is being balanced across different Percona XtraDB Cluster nodes (not just one node).

Scheduler behavior

After you enable the scheduler, it works as follows:

- **Writer node**: The scheduler sets `pod-0` (the first PXC Pod) as the writer node by default. The scheduler ensures only one writer exists at any time. As long as `pod-0` is available, it remains the writer.
- **Failover**: If `pod-0` becomes unavailable, the scheduler automatically promotes another Pod to be the writer. The scheduler uses weighted hostgroups to ensure all ProxySQL instances promote the same Pod during failover, preventing split-brain scenarios.
- **ProxySQL clustering**: When the scheduler is enabled, ProxySQL clustering is automatically disabled. This is because the scheduler and ProxySQL clustering do not work well together. The `proxysql-monit` sidecar container is removed from ProxySQL Pods, and each ProxySQL instance manages its own `mysql_servers` configuration independently.
- **Scaling considerations**: When scaling down the Percona XtraDB Cluster with the scheduler enabled and `writerIsAlsoReader=false`, you may encounter weight inconsistencies in the `runtime_mysql_servers` table. If weights do not update correctly after scaling (for example, a weight remains at 1000 instead of updating to 999), restart the ProxySQL pods to refresh the configuration. This limitation is planned to be addressed in future releases.

Warning

When the scheduler is enabled, ProxySQL clustering is disabled. Each ProxySQL instance manages its own server configuration independently. This ensures proper read/write splitting but means ProxySQL instances do not share configuration.

By default, the ProxySQL scheduler distributes read requests evenly across all your cluster nodes. You can exclude the primary from processing reads and reserve it only for accepting write requests by setting the `writerIsAlsoReader` option to `false`.

You can additionally fine-tune the scheduler's behavior for your workload and deployment scenario. See the Custom resource reference for a complete list of available options.

Disabling the scheduler

If you need to disable the scheduler after it has been enabled, be aware of the following limitations:

- **Hostgroup leftovers**: After disabling the scheduler, ProxySQL pod-0 may retain 80XX hostgroups in the `runtime_mysql_servers` table. Restart the Pod to clear these leftovers.

  ```
  kubectl delete pod <proxysql-pod-name> -n <namespace>
  ```

  The Operator will automatically recreate the pod with the correct configuration.
- **Incomplete configuration**: One or more ProxySQL Pods may not contain all Percona XtraDB Cluster nodes in their `mysql_servers` and `runtime_mysql_servers` table after disabling. Verify the configuration and restart affected Pods if needed.

To disable the scheduler, remove or set `proxysql.scheduler.enabled=false` in your Custom Resource. This causes the restart of ProxySQL pods.

Switching from one proxy to another

You can switch from one proxy to another. Find the points to consider below:

Switching from HAProxy to ProxySQL

For example, your application is growing and read traffic increases significantly. Switching from HAProxy to ProxySQL enables you to:

- Unlock read scaling by distributing SELECT queries across replicas
- Implement intelligent SQL-aware routing and caching
- Reduce overhead on the primary node by reserving it only for writes.

You can switch from HAProxy to ProxySQL starting with Operator version 1.19.0.

Important

If your MySQL client uses the `mysql_native_password` authentication plugin (for example, MySQL 5.7 client), but your database user is configured with the `caching_sha2_password` plugin, you must pass the `--default-auth=caching_sha2_password` option to the client for successful authentication as follows:

```
mysql -u<user> -p'<password>' -hcluster1-proxysql --default-auth=caching_sha2_password
```

Switching from ProxySQL to HAProxy

You may want to switch from ProxySQL to HAProxy if:

- You run smaller deployments where ProxySQL's advanced features are not needed. Another use case is when you prioritize simplicity and stability over fine-grained query routing.
- Your workload is predominantly write-oriented, and read/write splitting provides little benefit.
- You prefer a lightweight, efficient proxy with minimal configuration and fewer features to manage.

Switching to HAProxy enables you to simplify your deployment and reduce operational overhead.

Resource usage considerations

HAProxy and ProxySQL have different resource requirements and characteristics. You must use different resource specifications when switching from one proxy to another. Here's why:

- **Memory usage**: ProxySQL typically requires more memory than HAProxy due to its query caching, connection pooling, and SQL-aware features. HAProxy is more memory-efficient and focuses on connection-level load balancing.
- **CPU usage**: ProxySQL performs SQL parsing and routing and often needs at least 2–4x more CPU than HAProxy for the same workload.

Recommendations for adjusting resources

When switching proxies, adjust your resource requests and limits accordingly:

- **Switching from HAProxy to ProxySQL**: Increase memory and CPU allocations. Start with at least 1G memory and 600m CPU per ProxySQL pod, then monitor and adjust based on your workload. ProxySQL benefits from more memory for query caching and connection pooling.
- **Switching from ProxySQL to HAProxy**: You can reduce resource allocations since HAProxy is more lightweight. Start with 500Mi memory and 400m CPU per HAProxy pod, then adjust based on your connection load.
- **Monitor and adjust**: After switching, monitor resource usage using `kubectl top pods` and adjust requests and limits based on actual consumption patterns. Consider setting resource requests to match your typical usage and limits to handle peak loads.
- **Use separate resource configurations**: Define different resource specifications for each proxy type in your cluster configuration to avoid conflicts and ensure optimal performance.

How to switch

Warning

Switching from ProxySQL to HAProxy will cause the downtime because the Operator needs to reconfigure the proxy Pods.

You can switch from proxy to another on an existing cluster:

From ProxySQL to HAProxy

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
  "spec": {
     "haproxy": {
        "enabled": true,
        "size": 3,
        "image": "percona/percona-xtradb-cluster-operator:1.19.0-haproxy" },
     "proxysql": { "enabled": false }
  }}'
```

From HAProxy to ProxySQL

You must be running the Operator version 1.19.0 and higher.

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
  "spec": {
     "haproxy": { "enabled": false },
     "proxysql": {
        "enabled": true,
        "size": 3,
        "image": "percona/percona-xtradb-cluster-operator:1.19.0-proxysql"
     }
  }}'
```

Workload transfer and disaster recovery

Multi-data center setup for disaster recovery

Disaster can happen at any moment. To keep your services running smoothly, you can set up two Percona XtraDB Clusters in different locations (called “sites”). You then configure them to replicate data between each other. This makes sure both clusters have the same data and stay in sync. One site works as the primary site, and the other is a replica. It is usually in a standby mode.

If the primary site goes down, you need a way to move the workload to the backup site so that users won’t notice anything.

Once the primary site is fixed, you can move the services back to it.

This guide explains how to set up a disaster recovery system and transfer workloads between sites when something goes wrong.

Assumptions

- This guide is about two Percona XtraDB Clusters (PXC) set up with the Operator in Kubernetes. The clusters are in two separate sites which represent different Kubernetes environments.

  To differentiate the clusters, let’s name them:

  - `cluster1` is the PXC on the primary site
  - `cluster2` is the PXC on the replica site
- The primary and replica sites must be identical. The easiest way to achieve this is to make a backup on the primary site and restore it on the replica.
- We assume your applications are already set up to automatically switch to the replica site B if the primary site goes down. Setting this up is not covered in this guide.

Set up the primary site

Before you start

Clone the repository with all manifests and source code. You'll need it to edit configuration files for the database clusters, Secrets, backups and restores. Run the following command:

```
$ git clone -b v1.19.0 https://github.com/percona/percona-xtradb-cluster-operator
```

Make sure to clone the correct branch. The branch name is the same as the Operator release version.

Install the Operator and PXC

1. Create a namespace.

   ```
   $ kubectl create namespace <namespace>
   ```

2. Use the Quickstart guide to install the Operator and Percona XtraDB Cluster.

   You now have the `cluster1` database cluster up and running.

Export the database secrets (for Operator 1.17.0 and earlier)

While on the primary site, export the Secrets object with the user credentials. Both the primary and the replica sites must have the same user credentials. This enables the Operator to restore the backup from the primary on the replica site.

1. List the Secrets objects.

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

   Expected output

   ```
   cluster1-secrets        Opaque              6      5m43s
   cluster1-ssl            kubernetes.io/tls   3      5m42s
   cluster1-ssl-internal   kubernetes.io/tls   3      5m40s
   internal-cluster1       Opaque              6      5m43s
   ```

   The file we are interested in is called `cluster1-secrets` where `cluster1` is the name of your cluster.

2. Export the database cluster's Secret file. You'll need it later to set up the replica site. The replica must have the same users as the primary site to replicate data from it. The following command exports the `cluster1-secrets` Secret to a `pxcsecret.yaml` file. Feel free to use your name and namespace:

   ```
   $ kubectl get secret cluster1-secrets -n <namespace> -o yaml > pxcsecret.yaml
   ```

3. Edit the exported `pxcsecret.yaml` file: remove the `annotations`, `creationTimestamp`, `resourceVersion`, `selfLink`, and `uid` metadata fields.

Create a backup from the primary site

We will use this backup to deploy the replica site.

1. Configure the backup storage. Use either the Amazon S3 / S3-compatible storage, or the Azure Blob Storage. Persistent Volumes are specific to a namespace, meaning only Pods in the same namespace can access them.

   Use the Configure storage for backups tutorial for the steps.

2. Make an on-demand backup on the primary site.
3. View the information about a backup:

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

Expected output

```
NAME      CLUSTER    STORAGE      DESTINATION                                                STATUS      COMPLETED   AGE
backup1   cluster1   s3-us-west   s3://mybucket/cluster1-2025-03-18-10:55:43-full   Succeeded   3m25s       4m4s
```

Set up the replica site

The replica site must be the exact copy of the primary site and must have the same system user credentials. The easiest way to achieve this is to make a backup on the primary site and restore it on the replica.

Before you start

Clone the repository with all manifests and source code. You'll need it to edit configuration files for the database clusters, Secrets, backups and restores. Run the following command:

```
$ git clone -b v1.19.0 https://github.com/percona/percona-xtradb-cluster-operator
```

Make sure to clone the correct branch. The branch name is the same as the Operator release version.

Procedure

Let's create `cluster2` on the replica site.

1. Create a namespace.

   ```
   $ kubectl create namespace <namespace>
   ```

2. Create the Secrets object with the user credentials for the replica site. The Operator uses this Secret object when installing Percona XtraDB Cluster. As a result, the users in both sites have the same credentials. This is required to restore the backup from the main site on the replica.

   Edit the `pxcsecret.yaml` file that you exported from the primary site, if you haven't done it before. Remove the `annotations`, `creationTimestamp`, `resourceVersion`, `selfLink`, and `uid` metadata fields.

   You can create the replica site with the same name as the primary. In our setup we differentiate the clusters and must change the name in the Secret.

   The resulting Secret file must resemble the following:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: cluster2-secrets  # Change the name if needed
   type: Opaque
   stringData:
     monitor: <monitor-password>  # Decoded passwords here
     operator: <operator-password>
     proxyadmin: <proxyadmin-password>
     replication: <replication-password>
     root: <root-password>
     xtrabackup: <xtrabackup-password>
   ```

3. Create the Secret with the following command. Replace the `<namespace>` placeholder with your name:

   ```
   $ kubectl apply -f path/to/pxcsecret.yaml -n <namespace>
   ```

4. Install Percona XtraDB Cluster. Edit the `deploy/cr.yaml` file and specify the following configuration:

   - `metadata.name` - The name of the cluster if you want to change it. It must match the name you defined for the user Secret on step 2.

   ```
   metadata:
     name: cluster2  # The name of your cluster if you want to change it
   ```

5. Run the following command to install Percona XtraDB Cluster:

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

   It may take some time to install and initialize the cluster.

6. Check the status of the cluster:

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

Expected output

```
NAME       ENDPOINT                  STATUS   PXC   PROXYSQL   HAPROXY   AGE
cluster2   cluster2-haproxy.<namespace>   ready    3                3         36m
```

Restore the backup on the replica site

1. Create the Secret object with the credentials from the cloud storage where you made the backup to. The Operator uses the same Secret for backups and restores. For example, if you named the Secret `deploy/backup/backup-s3-secret.yaml`, run the following command to create the Secrets object on the replica site. Replace the `<namespace>` placeholder with your namespace.

   ```
   $ kubectl apply -f deploy/backup/backup-s3-secret.yaml -n <namespace>
   ```

2. To restore from a backup, create a special restore configuration file. Edit the sample `deploy/backup/restore.yaml` file.

   Specify the following information:

   - `spec.pxcCluster` - the name of the cluster on the replica site.
   - `spec.backupSource.destination` - the location of the backup on the backup storage. Run the `kubectl get pxc-backup -n <namespace>` on the main site to check the destination.

   Specify the storage information specific to the storage you used for the backup. For S3 storage, this will be the following:

   - `spec.backupSource.s3.bucket` - the name of the bucket where the backup is stored
   - `spec.backupSource.s3.credentialsSecret` - the name of the Secrets object with the credentials from the backup storage that you created in step 1.
   - `spec.backupSource.s3.region` - the region where the bucket is located. It must match the region that you defined in the `deploy/cr.yaml` file on when you made a backup.

   ```
   apiVersion: pxc.percona.com/v1
   kind: PerconaXtraDBClusterRestore
   metadata:
     name: restore1
   spec:
     pxcCluster: cluster2
     backupSource:
       destination: s3://mybucket/cluster1-2025-03-18-10:55:43-full
       s3:
         bucket: mybucket
         credentialsSecret: my-cluster-name-backup-s3
         region: us-west-2
   ```

3. Run the following command to start a restore:

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

4. Check the cluster status to see if the restore was successful:

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

Expected output

```
NAME       ENDPOINT                  STATUS   PXC   PROXYSQL   HAPROXY   AGE
cluster2   cluster2-haproxy.<namespace>   ready    3                3         36m
```

Configure replication between the sites

The sites must have the same copy of data. To do so, configure the replication between them so that sites are always in sync. The replication is defined via a replication channel where you specify which site is the source of data and which site receives it.

Prepare the primary site

Your replica site needs to connect to your primary site to replicate data from it. For this, each database Pod on the primary site must have an external IP addresses to be reached directly. This is done by exposing the database cluster Pods using the LoadBalancer service type. Read more about exposing a cluster.

1. Since the primary site is already running, we will patch its configuration with the following command. Replace the `<namespace>` placeholder with your namespace:

   ```
   $ kubectl patch pxc cluster1 -n <namespace> --type=merge --patch '{
     "spec": {
     "pxc": {
       "expose": {
         "enabled": true,
         "type": "LoadBalancer"
       }
     }}}'
   ```

2. Configure the replication channel on the primary site. Specify the following Custom Resource options in the `spec.pxc.replicationChannels` subsection in the `deploy/cr.yaml` file:

   - `pxc.replicationChannels.[].name` is the name of the channel,
   - `pxc.replicationChannels.[].isSource` defines what cluster the data is replicated from. Set the value to `true`.

   Run the following command to add this configuration:

   ```
   $ kubectl patch pxc cluster1 --type=merge --patch '{
     "spec": {
       "pxc": {
         "replicationChannels": [
           {
             "name": "pxc1_to_pxc2",
             "isSource": true
           }
         ]
       }
     }}'
   ```

3. Check that the Pods are exposed by listing the services:

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

   Expected output

   ```
   NAME                      TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)                                 7m5s
   cluster1-pxc-0            LoadBalancer   34.118.227.242  104.197.82.173   3306:32522/TCP                          7m5s
   cluster1-pxc-1            LoadBalancer   34.118.236.108  34.44.97.95      3306:32361/TCP
   cluster1-pxc-2            LoadBalancer   34.118.236.170  35.222.208.249   3306:31607/TCP
   ```

   Store the public IP addresses of your Pods. You will need them during the replica site setup.

Prepare the replica site

Configure the replication channel on the replica site. Specify the following Custom Resource options in the `spec.pxc.replicationChannels` subsection in the `deploy/cr.yaml` file:

- `spec.pxc.replicationChannels` - The replication channel configuration. The name of the channel must match the name on the primary site.
- `spec.pxc.replicationChannels[].isSource` - Set the value to `false` to indicate that the replica site is not the source of the data.
- `spec.pxc.replicationChannels[].sourcesList` - The list of sources. Specify the external IP addresses of the database Pods from the primary site.

Run the following command to apply a patch to the replica site's configuration with the required information. Don't forget to replace the `<placeholders>` with your values:

```
$ kubectl patch pxc cluster1 -n <namespace> --type=merge --patch '{
"spec": {
  "pxc": {
    "replicationChannels": [
      {
        "name": "pxc1_to_pxc2",
        "isSource": false,
        "sourcesList": [
          { "host": "34.118.227.242", "port": 3306, "weight": 100 },
          { "host": "34.118.227.242", "port": 3306, "weight": 100 },
          { "host": "34.118.227.242", "port": 3306, "weight": 100 }
        ]
      }
    ]
  }
}}'
```

Verify the replication

To verify that the replication is working, do the following:

1. Connect to Percona XtraDB Cluster on the primary site.
2. Create a database and a table.

```
mysql> CREATE DATABASE demo;
```

Expected output

```
Query OK, 1 row affected (0.02 sec)
```

```
mysql> CREATE TABLE demo.users(user_id INT PRIMARY KEY, user_name VARCHAR(30));
```

Expected output

```
Query OK, 0 rows affected (0.03 sec)
```

3. Insert some data into the database:

```
mysql> INSERT INTO demo.users VALUES (1, 'percona');
```

4. Connect to Percona XtraDB Cluster on the replica site.
5. Retrieve the data from the database:

```
mysql> SELECT * FROM demo.users;
```

Expected output

```
+---------+-----------+
| user_id | user_name |
+---------+-----------+
|       1 | percona   |
+---------+-----------+
1 row in set (0.00 sec)
```

Promote the replica site to a new primary

Let's say the primary site with `cluster1` is down. The client applications have automatically switched to the replica site. Now you need to reconfigure your setup to make `cluster2` on the replica site a new primary and have it handle the load.

Here's how to do it:

1. Modify the replication channel for `cluster2` within the `deploy/cr.yaml` file:

   - Set the `isSource` value to `true` to make the replica site the source of the data.
   - Remove the `sourcesList` configuration.

   Run the following command to apply a patch configuration to `cluster2`.

   ```
   $ kubectl patch pxc cluster2 -n <namespace> --type=merge --patch '{
     "spec": {
       "pxc": {
         "replicationChannels": [
           {
             "name": "pxc1_to_pxc2",
             "isSource": true
           }
         ]
       }
     }}'
   ```

   Now `cluster2` acts as the primary site.

2. While the old primary site is unavailable, `cluster1` no longer has up-to-date data. So you can delete it. Refer to the Delete the database cluster tutorial for the steps how to do it.

Restore the previous primary site

Let's say the root of the outage is no longer present. You can now install a new database cluster on this site. Let's use the previous name `cluster1` for it.

Install a new database cluster on the previous primary site

The new `cluster1` must be the exact copy of the current primary `cluster2`. We will use the same approach as we did when creating `cluster2`: make a backup from `cluster2` and restore it on `cluster1`.

The steps are the following:

1. Create the namespace
2. If you deleted the Operator, install it. Use the Quickstart for the steps.
3. Prepare the Secrets file with the user credentials for `cluster1`. The users on both sites must have the same credentials.

   You can reuse the `pxcsecret.yaml` secrets file or create a new one. Make sure that the passwords in this file match the passwords from the `cluster2-secrets` Secrets object. Check the Export the database secrets section to refresh your memory how to find the required Secrets object.

   Edit the `pxcsecret.yaml` file and change the name of the cluster to `cluster1`.

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: cluster1-secrets  # The name of the cluster to-be-installed
   type: Opaque
   stringData:
     monitor: <monitor-password>  # Decoded passwords here
     operator: <operator-password>
     proxyadmin: <proxyadmin-password>
     replication: <replication-password>
     root: <root-password>
     xtrabackup: <xtrabackup-password>
   ```

4. Create the Secrets object:

   ```
   $ kubectl apply -f path/to/pxcsecret.yaml -n <namespace>
   ```

5. Install Percona XtraDB Cluster with the `cluster1` name and the default parameters:

   ```
   $ kubectl apply -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-operator/v1.19.0/deploy/cr.yaml -n
   <namespace>
   ```

6. Check the status of the cluster:

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

   Expected output

   ```
   NAME       ENDPOINT                   STATUS   PXC   PROXYSQL   HAPROXY   AGE
   cluster1   cluster1-haproxy.<namespace>   ready    3               3         3m
   ```

Make a backup on the current primary site

1. Make a backup on the current primary `cluster2`. See the Create a backup from the primary site section for the steps.

   Expected output

   ```
   NAME      CLUSTER    STORAGE      DESTINATION                                                STATUS      COMPLETED   AGE
   backup1   cluster2   s3-us-west   s3://mybucket/cluster2-2025-03-21-12:05:37-full   Succeeded   2m55s       6m6s
   ```

Restore the backup on a new database cluster

1. Restore the backup from `cluster2` on `cluster1`. Change the `deploy/backup/restore.yaml` file as follows:
   - Change the `pxcCluster` name to `cluster1`. This is where you make the restore.
   - Change the `backupSource.destination` to the location of the backup on the backup storage. Run the `kubectl get pxc-backup -n <namespace>` on `cluster2` (the current primary) to check the destination.

   ```
   apiVersion: pxc.percona.com/v1
   kind: PerconaXtraDBClusterRestore
   metadata:
     name: restore1
   spec:
     pxcCluster: cluster1
     backupSource:
       destination: s3://mybucket/cluster2-2025-03-21-12:05:37-full
       s3:
         bucket: mybucket
         credentialsSecret: my-cluster-name-backup-s3
         region: us-west-2
   ```

2. Start the restore with the following command:

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

3. Check the status of the cluster:

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

   The cluster should report the Ready status.

Promote the new database cluster as the primary

The newly deployed site with `cluster1` is now the working copy of the current primary `cluster2`. It's time to configure it back as the primary site.

To do this, configure the replication channels on both sites. Refer to the Configure replication between the sites section for the steps.

Transport Encryption (TLS/SSL)

Transport Layer Security (TLS)

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

- External - communication between the client application and ProxySQL.
- Internal - communication between Percona XtraDB Cluster instances. The internal certificate is also used as an authorization method.

TLS Certificates

You can configure TLS security in several ways.

- By default, the Operator **generates long-term certificates** automatically during the cluster creation if there are no certificate secrets available. If you need new certificates, you must renew them manually.
- The Operator can use a *cert-manager*, which will automatically **generate and renew short-term TLS certificates**. You must explicitly install cert-manager for this scenario.

  The *cert-manager* acts as a self-signed issuer and generates certificates allowing you to deploy and use the Percona Operator without a separate certificate issuer.
- You can generate TLS certificates manually or obtain them from some other issuer and provide to the Operator.

For testing purposes, you can use pre-generated certificates available in the `deploy/ssl-secrets.yaml` file. But we strongly recommend **to not use them on any production system**!

TLS configuration

The following sections provide guidelines how to:

- Configure TLS security with the Operator using cert-manager
- Generate certificates manually
- Update certificates
- Disable TLS temporarily

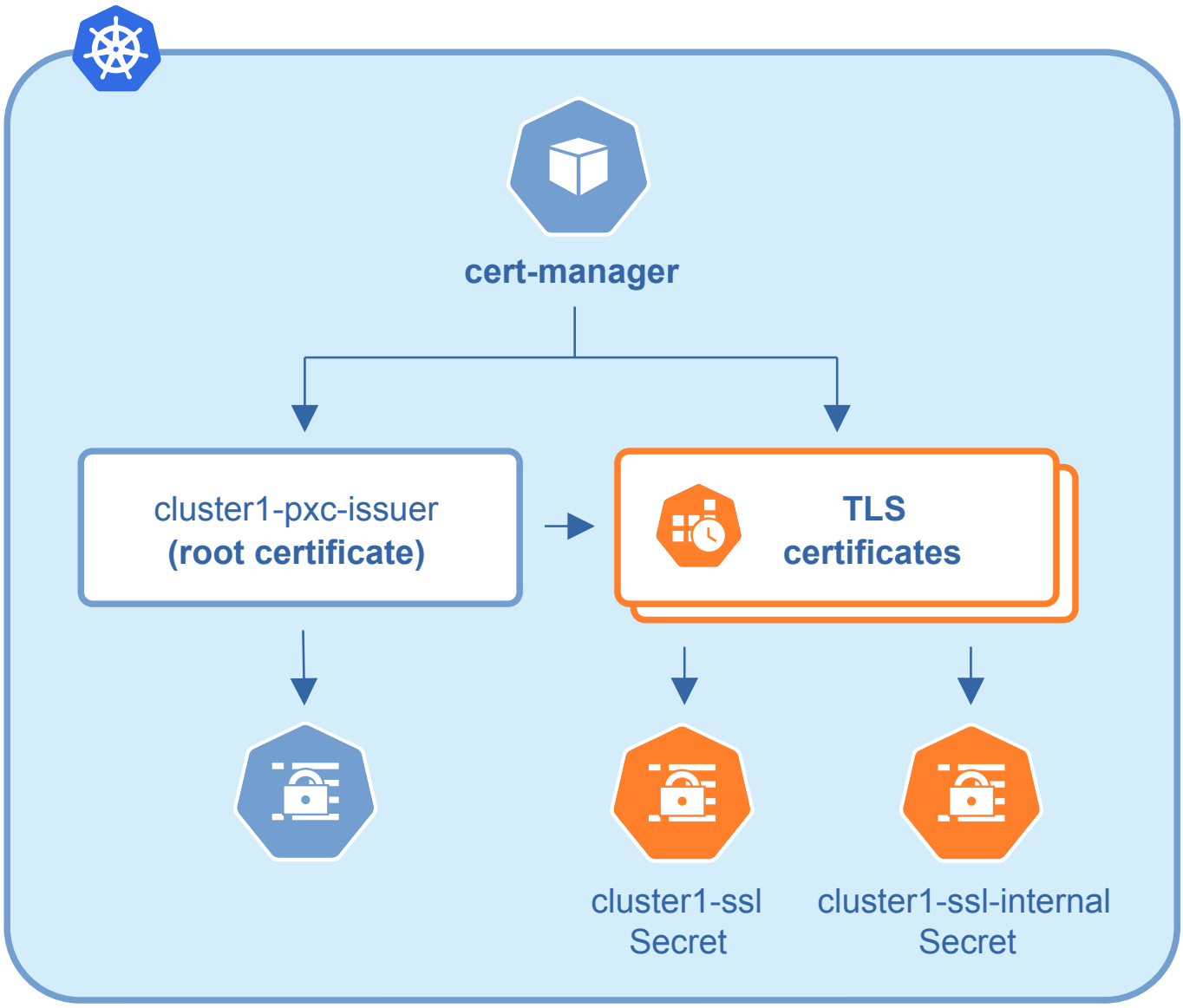
Install and use the cert-manager

About the cert-manager

A cert-manager is a Kubernetes certificate management controller which is widely used to automate the management and issuance of TLS certificates. It is community-driven, and open source.

When you have already installed cert-manager, nothing else is needed: just deploy the Operator, and the Operator will request a certificate from the cert-manager.

The *cert-manager* acts as a self-signed issuer and generates certificates. Certificates are valid for 3 months.



The Percona Operator self-signed issuer is local to the operator namespace. This self-signed issuer is created because Percona XtraDB Cluster requires all certificates issued by the same CA (Certificate authority).

Self-signed issuer allows you to deploy and use the Percona Operator without creating a cluster issuer separately.

Install the *cert-manager*

The cert-manager requires its own namespace

The steps to install the *cert-manager* are the following:

1. Create the `cert-manager` namespace:

```
kubectl create namespace cert-manager
```

2. Disable resource validations on the `cert-manager` namespace:

```
kubectl label namespace cert-manager certmanager.k8s.io/disable-validation=true
```

3. Install the cert-manager:

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.19.2/cert-manager.yaml
```

4. Verify the *cert-manager* by running the following command:

```
kubectl get pods -n cert-manager
```

The result should display the *cert-manager* and webhook active and running.

Expected output

```
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-69f748766f-6chvt              1/1     Running   0          65s
cert-manager-cainjector-7cf6557c49-l2cwt   1/1     Running   0          66s
cert-manager-webhook-58f4cff74d-th4pp      1/1     Running   0          65s
```

At this point, you can move on to deploying the Operator and your database cluster.

Customize certificate duration for cert-manager

When you deploy the cluster using the default configuration, the Operator triggers the cert-manager to create certificates with default duration of 90 days.

You can also customize the certificate duration. For example, to align certificate lifetimes with your organization's security and compliance policies.

Rules and limitations

Check the following rules and limitations for setting up the certificate duration:

1. You can set the duration **only when you create a new cluster**. Updating it in a running cluster is not supported.
2. The TLS certificate duration is subject to the following requirements:
   - The minimum accepted value is 1 hour. Durations below 1 hour are rejected.
   - Do **not** set the duration to exactly 1 hour; the Operator will fail to generate the correct certificate object if you do.
   - By default, cert-manager starts the renewal process when a certificate has one-third of its lifetime remaining, ensuring renewal before expiration. For example, if a certificate is valid for 1 hour, renewal will begin after approximately 40 minutes.
3. Minimum CA certificate duration is 730 hours (approximately 30 days). Do not set the duration to exactly 730 hours; the Operator will fail to generate the correct certificate object if you do.

Configuration

To set the custom duration, specify the following options in the Custom Resource:

- `.spec.tls.certValidityDuration` – validity period for TLS certificates
- `.spec.tls.caValidityDuration` – validity period for the Certificate Authority (CA)

Here's the example configuration:

```
tls:
  enabled: true
  certValidityDuration: 2160h
  caValidityDuration: 26280h
```

Create a new cluster with this configuration:

```
kubectl -f deploy/cr.yaml -n <namespace>
```

To verify the durations, you can check certificates for expiration at any time. This ensures your certificates are valid and helps you plan for renewals before they expire.

Generate certificates manually

You can generate TLS certificates manually instead of using the Operator's automatic certificate generation. This approach gives you full control over certificate properties and is useful for production environments with specific security requirements.

What you'll create

When you follow the steps from this guide, you'll generate these certificate files:

- `server.pem` - Server certificate for Percona XtraDB Cluster nodes
- `server-key.pem` - Private key for the server certificate
- `client.pem` - Client certificate for external connections
- `client-key.pem` - Private key for the client certificate
- `ca.pem` - Certificate Authority certificate
- `ca-key.pem` - Certificate Authority private key

Certificate requirements

You need to create **two sets** of certificates:

1. **External certificates** - for client connections from outside the cluster.
2. **Internal certificates** - for internal communication between Percona XtraDB Cluster nodes.

After creating the certificates, you'll create two Kubernetes Secrets and reference them in your cluster configuration.

Prerequisites

Before you start, make sure you have:

- `cfssl` and `cfssljson` tools installed on your system
- Your cluster name and namespace ready
- Access to your Kubernetes cluster

Procedure

Generate certificates

Replace `cluster1` and `my-namespace` with your actual cluster name and namespace in the commands below.

1. Set your cluster variables

   ```
   $ CLUSTER_NAME=my-cluster-name
   $ NAMESPACE=my-namespace
   ```

2. Create the Certificate Authority (CA)

   This command creates a root Certificate Authority that will sign all your certificates:

   ```
   $ cat <<EOF | cfssl gencert -initca - | cfssljson -bare ca
   {
     "CN": "Root CA",
     "key": {
       "algo": "rsa",
       "size": 2048
     }
   }
   EOF
   ```

3. Generate the server certificate for external communication. The command generates a server TLS certificate and a key for external connections, with SANs (Subject Alternative Names) for ProxySQL and Percona XtraDB Cluster endpoints.

```
$ cat <<EOF | cfssl gencert -ca=ca.pem -ca-key=ca-key.pem - | cfssljson -bare server
{
  "hosts": [
    "${CLUSTER_NAME}-proxysql",
    "*.${CLUSTER_NAME}-proxysql-unready",
    "*.${CLUSTER_NAME}-pxc"
  ],
  "CN": "${CLUSTER_NAME}-pxc",
  "key": {
    "algo": "rsa",
    "size": 2048
  }
}
EOF
```

The resulting files are `server.pem` (certificate) and `server-key.pem` (private key).

4. Create a Kubernetes Secret for the external certificate. This command creates a Kubernetes TLS secret named `cluster1-ssl`. The secret contains the server certificate (`server.pem`), its private key (`server-key.pem`), and the CA certificate (`ca.pem`). This secret should be referenced in your cluster's configuration for external TLS connections.

```
$ kubectl create secret generic cluster1-ssl \
  --from-file=tls.crt=server.pem \
  --from-file=tls.key=server-key.pem \
  --from-file=ca.crt=ca.pem \
  --type=Opague -n $NAMESPACE
```

5. Generate the server certificate for internal communication

   To secure communication between Percona XtraDB Cluster instances, you need a separate internal server certificate. Generate the internal TLS certificate and key with appropriate SANs:

```
$ cat <<EOF | cfssl gencert -ca=ca.pem -ca-key=ca-key.pem - | cfssljson -bare server-internal
{
  "hosts": [
    "*.${CLUSTER_NAME}-pxc"
  ],
  "CN": "${CLUSTER_NAME}-pxc-internal",
  "key": {
    "algo": "rsa",
    "size": 2048
  }
}
EOF
```

   The resulting files are `server-internal.pem` (certificate) and `server-internal-key.pem` (private key).

6. Create a Kubernetes Secret for the internal certificate

   This command creates a Kubernetes TLS secret named `cluster1-ssl-internal`. The secret contains the internal server certificate (`server-internal.pem`), its private key (`server-internal-key.pem`), and the CA certificate (`ca.pem`). This secret should be referenced in your cluster's configuration for internal TLS communications.

```
$ kubectl create secret generic cluster1-ssl-internal \
  --from-file=tls.crt=server-internal.pem \
  --from-file=tls.key=server-internal-key.pem \
  --from-file=ca.crt=ca.pem \
  --type=Opague -n $NAMESPACE
```

Configure your cluster

After creating the Secrets, add them to your cluster configuration in the `deploy/cr.yaml` file:

Add the secret for external use to the `spec.sslSecretName` option. Add the certificate for internal communications to the `spec.sslInternalSecretName` option.

```
spec:
  sslSecretName: cluster1-ssl
  sslInternalSecretName: cluster1-ssl-internal
```

Additional resources

Check the sample certificates in `deploy/ssl-secrets.yaml` for reference

Update certificates

How your TLS certificates are updated depends on how they were created:

- Certificates generated by the Operator are long-term. If you need to rotate them, you must do it manually.
- Certificates issued by the cert-manager are short-term. They are valid for 3 months. The cert-manager automatically reissues the certificates on schedule and without downtime.
- Certificates manually generated by you are not renewed automatically. It is your responsibility to timely update them. Use the steps in the following sections for how to do it.

This document describes how to update the internal certificate.

Check your certificates for expiration

If you use cert-manager:

1. Check the necessary secrets names (`cluster1-ssl` and `cluster1-ssl-internal` by default):

   ```
   kubectl get certificate
   ```

   You will have the following response:

   ```
   NAME                  READY   SECRET                AGE
   cluster1-ca-cert      True    cluster1-ca-cert      49m
   cluster1-ssl          True    cluster1-ssl          49m
   cluster1-ssl-internal True    cluster1-ssl-internal 49m
   ```

2. Optionally you can also check that the certificates issuer is up and running:

   ```
   kubectl get issuer
   ```

   The response should be as follows:

   ```
   NAME                    READY   AGE
   cluster1-pxc-ca-issuer  True    49m
   cluster1-pxc-issuer     True    49m
   ```

   Note

   If you don't use cert-manager, list your secrets:

   ```
   kubectl get secrets -n $NAMESPACE
   ```

   Then either use the default ones or the one you created

3. Use the following command to find out the certificates validity dates, substituting Secrets names if necessary:

   ```
   $ {
     kubectl get secret/cluster1-ssl-internal -o jsonpath='{.data.tls\.crt}' | base64 --decode | openssl x509 -inform pem -noout
   -text | grep "Not After"
     kubectl get secret/cluster1-ssl -o jsonpath='{.data.ca\.crt}' | base64 --decode | openssl x509 -inform pem -noout -text |
   grep "Not After"
     }
   ```

   Sample output

   ```
   notBefore=Nov  7 10:54:00 2025 GMT
   notAfter=Nov  7 10:54:00 2026 GMT
   ```

Update certificates without downtime

If you don't use cert-manager and have *created certificates manually*, you can follow the next steps to perform a no-downtime update of these certificates *if they are still valid*.

> **Note**
>
> For already expired certificates, follow the alternative way.

Having non-expired certificates, you can roll out new certificates (both CA and TLS) with the Operator as follows.

1. Generate a new CA certificate (`ca.pem`), a new TLS certificate (server.pem) and a key for it (server-key.pem).
2. Get the current CA (`ca.pem.old`) and TLS (`tls.pem.old`) certificates and the TLS certificate key (`tls.key.old`):

   ```
   kubectl get secret/cluster1-ssl -o jsonpath='{.data.ca\.crt}' | base64 --decode > ca.pem.old
   kubectl get secret/cluster1-ssl -o jsonpath='{.data.tls\.crt}' | base64 --decode > tls.pem.old
   kubectl get secret/cluster1-ssl -o jsonpath='{.data.tls\.key}' | base64 --decode > tls.key.old
   ```

3. Combine new and current `ca.pem` into a `ca.pem.combined` file:

   ```
   cat ca.pem ca.pem.old >> ca.pem.combined
   ```

4. Create a new Secrets object with the *old* TLS certificate (`tls.pem.old`) and key (`tls.key.old`), but a *new combined* `ca.pem` (`ca.pem.combined`):

   ```
   kubectl create secret generic cluster1-ssl \
   --from-file=tls.crt=server.pem.old \
   --from-file=tls.key=server-key.pem.old \
   --from-file=ca.crt=ca.pem.combined \
   --type=kubernetes.io/tls -o yaml --dry-run=client | kubectl apply -f -
   ```

5. The cluster will go through a rolling restart. This process will not cause issues, because every node has the old TLS certificate/key, and both new and old CA certificates.
6. Create a new Secrets object again. This time use a new TLS certificate (`server.pem` in the example) and a new TLS key (`server-key.pem`), and again the combined CA certificate (`ca.pem.combined`):

   ```
   kubectl create secret generic cluster1-ssl \
   --from-file=tls.crt=server.pem \
   --from-file=tls.key=server-key.pem \
   --from-file=ca.crt=ca.pem.combined \
   --type=Opague -o yaml --dry-run=client | kubectl apply -f -
   ```

7. The cluster will go through a rolling restart. This process will not cause issues, because every node already has a new CA certificate (as a part of the combined CA certificate), and can successfully allow joiners with new TLS certificate to join. A joiner node also has a combined CA certificate, so it can authenticate against older TLS certificate.
8. Create a final Secrets object: use the new TLS certificate (`server.pmm`) and its key (`server-key.pem`), and only the new CA certificate (`ca.pem`):

   ```
   kubectl create secret generic cluster1-ssl \
   --from-file=tls.crt=server.pem \
   --from-file=tls.key=server-key.pem \
   --from-file=ca.crt=ca.pem \
   --type=Opague -o yaml --dry-run=client | kubectl apply -f -
   ```

9. The cluster will go through a rolling restart, but it will do it without problems: the old CA certificate is removed, and every node is already using new TLS certificate and no nodes rely on the old CA certificate any more.

Update certificates with downtime

If your certificates have been already expired (or if you continue to use the Operator version prior to 1.9.0), you should move through the *pause - update Secrets - unpause* route as follows.

1. Pause the cluster in a standard way, and make sure it has reached its paused state.
2. If cert-manager is used, delete issuer and TLS certificates:

```
$ {
  kubectl delete issuer/cluster1-pxc-ca
  kubectl delete certificate/cluster1-ssl certificate/cluster1-ssl-internal
  }
```

3. Delete Secrets to force the SSL reconciliation:

```
kubectl delete secret/cluster1-ssl secret/cluster1-ssl-internal
```

4. Check certificates to make sure reconciliation have succeeded.
5. Unpause the cluster in a standard way, and make sure it has reached its running state.

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.delete-ssl option in `deploy/cr.yaml`: if this finalizer is set, the Operator will delete Secret, certificate and issuer after the cluster deletion event.

Deploy Percona XtraDB Cluster without TLS

You can deploy your Percona XtraDB Cluster without TLS, although we strongly recommend that you enable TLS for any production environment.

Disable TLS during initial deployment

To disable TLS at deployment (for example, for demonstration or testing), edit your `deploy/cr.yaml` file. Set the `unsafeFlags.tls` key to `true` and the `tls.enabled` key to `false`:

```
...
spec:
  ...
  unsafeFlags:
    tls: true
    ...
  tls:
    enabled: false
```

Enable or disable TLS on a running cluster

You can enable or disable TLS at any time by changing the `tls.enabled` Custom Resource option to `true` to enable TLS, or to `false` to disable TLS. Be aware that changing this setting on a running cluster causes downtime and can introduce disruptions.

- If you set `tls.enabled` to `false`, the Operator will pause the cluster, wait for all Pods to be deleted, set the `unsafeFlags.tls` option to `true`, delete the TLS secrets, and then unpause the cluster.
- If you set `tls.enabled` to `true`, the Operator will pause the cluster, wait for all Pods to be deleted, set the `unsafeFlags.tls` option to `false`, and then unpause the cluster.

Warning

Do not change the `tls.enabled` option while the cluster is in the process of enabling or disabling TLS. Changing this value mid-process will immediately unpause the cluster, even if the operation is not complete.

Data at rest encryption

Data at rest encryption

Data-at-rest encryption ensures that data stored on disk remains protected even if the underlying storage is compromised. This process is transparent to your applications, meaning you don't need to change your application code. If an unauthorized user gains access to the storage, they can't read the data files.

The Operator supports data at rest encryption in Percona XtraDB Cluster since version 1.4.0.

To encrypt tablespaces, schemas, backups and binlogs, the Operator uses the following tools:

- The `keyring_vault` **component** shipped with Percona XtraDB Cluster 8.4 and 5.7
- The `keyring_vault` **plugin** shipped with Percona XtraDB Cluster 8.0

For more information about keyring plugins and components, see Compare keyring components and keyring plugins chapter in Percona Server for MySQL documentation.

- HashiCorp Vault to securely store and manage master encryption keys, perform automatic key rotation, and enable audit logging.

This setup enhances the overall security posture of your Percona XtraDB cluster on Kubernetes.

Encryption flow

The encryption mechanism uses a two-tiered key architecture to secure your data:

- Each database instance has a master encryption key to encrypt tablespaces and binlogs. Master encryption key is stored separately from tablespace keys, in an external key management service like HashiCorp Vault.
- Each tablespace has a unique tablespace key to encrypt the data files (tables and indexes).

The data is encrypted before being written to disk. When you need to read the data, it's decrypted in memory for use and then re-encrypted before being written back to disk.

Key rotation

Key rotation is replacing the old master encryption key with the new one. When a new master encryption key is created, it is stored in Vault and tablespace keys are re-encrypted with it. The entire dataset is not re-encrypted and this makes the key rotation a fast and lightweight operation.

Read more about key rotation in the Rotate the master key.

Backups and encryption

Percona Operator for MySQL uses Percona XtraBackup; for backups and fully supports backing up encrypted data. The backups remain encrypted, ensuring your data is secure both on your live cluster and in your backup storage.

Keep your encryption keys safe

To restore from an encrypted backup, you **must have the original master encryption key**. If the encryption key is lost, your backups will be irrecoverable. Always ensure you have a secure and reliable process for managing and backing up your master encryption keys separately from your database backups.

Next steps

Choose a setup guide based on your security requirements:

- Configure data-at-rest encryption without TLS - For development and testing environments
- Configure data-at-rest encryption with TLS - For production environments requiring encrypted communication

Configure data at rest encryption without TLS

This guide walks you through deploying and configuring HashiCorp Vault to work with Percona Operator for MySQL based on Percona XtraDB Cluster to enable data-at-rest encryption using HTTP protocol.

Assumptions

1. This guide is provided as a best effort and builds upon procedures described in the official Vault documentation. Since Vault's setup steps may change in future releases, this document may become outdated; we cannot guarantee ongoing accuracy or responsibility for such changes. For the most up-to-date and reliable information, please always refer to the official Vault documentation.
2. For this setup we deploy the Vault server in High Availability (HA) mode on Kubernetes via Helm without TLS. The HA setup uses Raft storage backend and consists of 3 replicas for redundancy. Using Helm is not mandatory. Any supported Vault deployment (on-premises, in the cloud, or a managed Vault service) works as long as the Operator can reach it.
3. This guide uses Vault Helm chart version 0.30.0. You may want to change it to the required version by setting the `VAULT_HELM_VERSION` variable.

Prerequisites

Before you begin, ensure you have the following tools installed:

- `kubectl` - Kubernetes command-line interface
- `helm` - Helm package manager
- `jq` - JSON processor

Prepare your environment

1. Export the namespace and other variables as environment variables to simplify further configuration:

```
export NAMESPACE="vault"
export VAULT_HELM_VERSION="0.30.0"
export SERVICE="vault"
export SECRET_NAME_VAULT="vault-secret"
export POLICY_NAME="mysql-policy"
export WORKDIR="/tmp/vault"
```

2. Create a working directory for configuration files:

```
mkdir -p $WORKDIR
```

3. It is a good practice to isolate workloads in Kubernetes using namespaces. Create a namespace with the following command:

```
kubectl create namespace vault
```

Install Vault

For this setup, we install Vault in Kubernetes using the Helm 3 package manager in High Availability (HA) mode with Raft storage backend.

1. Add and update the Vault Helm repository:

```
helm repo add hashicorp https://helm.releases.hashicorp.com
helm repo update
```

2. Install Vault in HA mode:

```
helm upgrade --install ${SERVICE} hashicorp/vault \
  --disable-openapi-validation \
  --version ${VAULT_HELM_VERSION} \
  --namespace ${NAMESPACE} \
  --set "global.enabled=true" \
  --set "global.tlsDisable=true" \
  --set "global.platform=kubernetes" \
  --set "server.ha.enabled=true" \
  --set "server.ha.replicas=3" \
  --set "server.ha.raft.enabled=true" \
  --set "server.ha.raft.setNodeId=true" \
  --set-string "server.ha.raft.config=cluster_name = \"vault-integrated-storage\"
ui = true
listener \"tcp\" {
  address = \"[::]:8200\"
  cluster_address = \"[::]:8201\"
}
storage \"raft\" {
  path = \"/vault/data\"
}
disable_mlock = true
service_registration \"kubernetes\" {}"
```

This command does the following:

- Installs HashiCorp Vault in High Availability (HA) mode without TLS in your Kubernetes cluster
- Sets up Raft as the backend storage with three replicas for fault tolerance
- Configures the Vault TCP listener for HTTP communication (port 8200)

Sample output

```
NAME: vault
LAST DEPLOYED: Wed Aug 20 12:55:38 2025
NAMESPACE: vault
STATUS: deployed
REVISION: 1
NOTES:
Thank you for installing HashiCorp Vault!

Now that you have deployed Vault, you should look over the docs on using
Vault with Kubernetes available here:

https://developer.hashicorp.com/vault/docs
```

3. Wait for all Vault pods to be running:

```
kubectl wait --for=condition=Ready pod -l app.kubernetes.io/name=${SERVICE} -n $NAMESPACE --timeout=300s
```

4. Retrieve the Pod names where Vault is running:

```
kubectl -n $NAMESPACE get pod -l app.kubernetes.io/name=${SERVICE} -o jsonpath='{range .items[*]}{.metadata.name}{"\n"}{end}'
```

Sample output

```
vault-0
vault-1
vault-2
```

Initialize and unseal Vault

1. After Vault is installed, you need to initialize it. Run the following command to initialize the first pod:

```
kubectl exec -it pod/vault-0 -n $NAMESPACE -- vault operator init -key-shares=1 -key-threshold=1 -format=json >
${WORKDIR}/vault-init
```

The command does the following:

- Connects to the Vault Pod
- Initializes Vault server
- Creates 1 unseal key share which is required to unseal the server
- Outputs the init response to a local file. The file includes unseal keys and a root token.

2. Vault is started in a sealed state. In this state Vault can access the storage but it cannot decrypt data. In order to use Vault, you need to unseal it.

   Retrieve the unseal key from the file:

   ```
   unsealKey=$(jq -r ".unseal_keys_b64[]" < ${WORKDIR}/vault-init)
   ```

   Now, unseal the first Vault pod:

   ```
   kubectl exec -it pod/vault-0 -n $NAMESPACE -- vault operator unseal "$unsealKey"
   ```

   Sample output

   ```
   Key             Value
   ---             -----
   Seal Type       shamir
   Initialized     true
   Sealed          false
   Total Shares    1
   Threshold       1
   Version         1.20.1
   Build Date      2025-07-24T13:33:51Z
   Storage Type    raft
   Cluster Name    vault-cluster-55062a37
   Cluster ID      37d0c2e4-8f47-14f7-ca49-905b66a1804d
   HA Enabled      true
   ```

3. Add the remaining Pods to the Vault cluster. Run the following for loop:

   ```
   for POD in vault-1 vault-2; do
     kubectl -n "$NAMESPACE" exec $POD -- sh -c '
       vault operator raft join http://vault-0.vault-internal:8200
     '
   done
   ```

   The command connects to each Vault Pod (`vault-1` and `vault-2`) and issues the `vault operator raft join` command, which:

   - Joins the Pods to the Vault Raft cluster, enabling HA mode
   - Connects to the cluster leader (`vault-0`) over HTTP
   - Ensures all nodes participate in the Raft consensus and share storage responsibilities

   Sample output

   ```
   Key                    Value
   ---                    -----
   Joined Raft cluster    true
   Leader Address         http://vault-0.vault-internal:8200
   ```

4. Unseal the remaining Pods. Use this for loop:

   ```
   for POD in vault-1 vault-2; do
       kubectl -n "$NAMESPACE" exec $POD -- sh -c "
           vault operator unseal \"$unsealKey\"
       "
   done
   ```

Expected output

```
Key                Value
---                -----
Seal Type          shamir
Initialized        true
Sealed             false
Total Shares       1
Threshold          1
Version            1.20.1
Build Date         2025-07-24T13:33:51Z
Storage Type       raft
HA Enabled         true
```

Configure Vault

At this step you need to configure Vault and enable secrets within it. To do so you must first authenticate in Vault.

When you started Vault, it generates and starts with a root token that provides full access to Vault. Use this token to authenticate.

Run the following commands on a leader node. The remaining ones will synchronize from the leader.

1. Extract the Vault root token from the file where you saved the init response output:

   ```
   cat ${WORKDIR}/vault-init | jq -r ".root_token"
   ```

   Sample output

   ```
   hvs.*************Jg9r
   ```

2. Connect to Vault Pod:

   ```
   kubectl exec -it vault-0 -n $NAMESPACE -- /bin/sh
   ```

3. Authenticate in Vault with this token:

   ```
   vault login hvs.*************Jg9r
   ```

4. Enable the secrets engine at the mount path. The following command enables KV secrets engine v2 at the `pxc-secret` mount-path:

   ```
   vault secrets enable --version=2 -path=pxc-secret kv
   ```

   Sample output

   ```
   Success! Enabled the kv secrets engine at: pxc-secret/
   ```

5. Optionally, enable audit logs for vault:

   ```
   vault audit enable file file_path=/vault/vault-audit.log
   ```

6. Exit the Vault Pod:

   ```
   exit
   ```

Create a non-root token

Using the root token for authentication is not recommended, as it poses significant security risks. Instead, you should create a dedicated, non-root token for the Operator to use when accessing Vault. The permissions for this token are controlled by an access policy. Before you create a token you must first create the access policy.

1. Create a policy for accessing the kv engine path and define the required permissions in the `capabilities` parameter:

```
kubectl -n "$NAMESPACE" exec vault-0 -- sh -c "
  vault policy write $POLICY_NAME - <<EOF
path \"pxc-secret/data/*\" {
  capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\"]
}
path \"pxc-secret/metadata/*\" {
  capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\"]
}
path \"pxc-secret/*\" {
  capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\"]
}
EOF
"
```

2. Now create a token with a policy.

```
kubectl -n "${NAMESPACE}" exec pod/vault-0 -- vault token create -policy="${POLICY_NAME}" -format=json > "${WORKDIR}/vault-token.json"
```

3. Export the non-root token as an environment variable:

```
export NEW_TOKEN=$(jq -r '.auth.client_token' "${WORKDIR}/vault-token.json")
```

4. Verify the token:

```
echo "New Vault Token: $NEW_TOKEN"
```

Sample output

```
hvs.CAESINO**********************************************T2Y
```

Create a Secret for Vault

To enable Vault for the Operator, create a Secret object for it. To do so, create a YAML configuration file and specify the following information:

- A token to access Vault
- A Vault server URL
- The secrets mount path

Depending on Percona XtraDB Cluster version, you must specify the Vault configuration as follows:

- For Percona XtraDB Cluster 8.0 and 5.7 - as key=value pairs
- For Percona XtraDB Cluster 8.4 - as a JSON object

You can modify the example configuration file:

Percona XtraDB Cluster 8.0

```
apiVersion: v1
kind: Secret
metadata:
  name: keyring-secret-vault
type: Opaque
stringData:
  keyring_vault.conf: |-
    token = hvs.CAESINO****************************************T2Y
    vault_url = http://vault.vault.svc.cluster.local:8200
    secret_mount_point = pxc-secret
```

Percona XtraDB Cluster 8.4

```
apiVersion: v1
kind: Secret
metadata:
  name: keyring-secret-vault-84
type: Opaque
stringData:
  keyring_vault.conf: |-
    {
      "token": "hvs.CAESINO****************************************T2Y",
      "vault_url": "http://vault.vault.svc.cluster.local:8200",
      "secret_mount_point": "pxc-secret"
    }
```

Now create a Secret object. Replace the `<cluster-namespace>` placeholder with the namespace where your database cluster is deployed:

```
kubectl apply -f deploy/vault-secret.yaml -n <cluster-namespace>
```

Reference the Secret in your Custom Resource manifest

Now, reference the Vault Secret in the Operator Custom Resource manifest. Note that the Secret name is the one you specified in the `metadata.name` field when you created a Secret.

1. Export the namespace where the cluster is deployed and the secret name as environment variables:

   ```
   export CLUSTER_NAMESPACE=<cluster-namespace>
   export SECRET_NAME_PXC="keyring-secret-vault-84"
   ```

2. Update the cluster configuration. Since this is a running cluster, we will apply a patch referencing your Secret.

   Make sure to specify the correct Secret name. The default Secret name for MySQL 8.0 and 5.7 is `keyring-secret-vault`. In this setup we use `keyring-secret-vault-84` Secret name for MySQL 8.4. Use the following command as an example and specify the Secret name for the MySQL version you're using:

   ```
   kubectl patch pxc cluster1 \
     --namespace $CLUSTER_NAMESPACE \
     --type=merge \
     --patch "{\"spec\":{\"vaultSecretName\":\"${SECRET_NAME_PXC}\"}}"
   ```

This triggers cluster Pods to restart.

Use data-at-rest encryption

To use encryption, you can:

- turn it on for every table you create with the `ENCRYPTION='Y'` clause in your SQL statement. For example:

```
CREATE TABLE t1 (c1 INT, PRIMARY KEY pk(c1)) ENCRYPTION='Y';
CREATE TABLESPACE foo ADD DATAFILE 'foo.ibd' ENCRYPTION='Y';
```

Existing tables will not be encrypted unless you specifically enable it via the `ALTER TABLE .... ENCRYPTION='Y';` statement.

- turn on default encryption of a schema or a general tablespace. Then all tables you create will have encryption enabled. To turn on default encryption, use the following SQL statement:

```
SET default_table_encryption=ON;
```

See the following chapters of Percona Server for MySQL documentation in how to use encryption:

- Encrypt File-Per-Table Tablespace
- Encrypt schema or general tablespace
- Encrypt system tablespace
- Encrypt doublewrite file pages
- Encrypt temporary files

Verify encryption

Refer to the Percona Server for MySQL documentation for guidelines how to verify encryption in your database.

Configure data at rest encryption with TLS

This guide walks you through deploying and configuring HashiCorp Vault with TLS enabled to work with Percona Operator for MySQL based on Percona XtraDB Cluster to enable data-at-rest encryption using HTTPS protocol.

By default, Percona XtraDB Cluster (PXC) and Vault communicate over an unencrypted HTTP protocol. You can enable encrypted HTTPS protocol with TLS as an additional security layer to protect the data transmitted between Vault and your PXC nodes. HTTPS ensures that sensitive information, such as encryption keys and secrets, cannot be intercepted or tampered with on the network.

Assumptions

1. This guide is provided as a best effort and builds upon procedures described in the official Vault documentation. Since Vault's setup steps may change in future releases, this document may become outdated; we cannot guarantee ongoing accuracy or responsibility for such changes. For the most up-to-date and reliable information, please always refer to the official Vault documentation.
2. For this setup we deploy the Vault server in High Availability (HA) mode on Kubernetes via Helm with TLS enabled. The HA setup uses Raft storage backend and consists of 3 replicas for redundancy. Using Helm is not mandatory. Any supported Vault deployment (on-premises, in the cloud, or a managed Vault service) works as long as the Operator can reach it.
3. This guide uses Vault Helm chart version 0.30.0. You may want to change it to the required version by setting the `VAULT_HELM_VERSION` variable.

Prerequisites

Before you begin, ensure you have the following tools installed:

- `kubectl` - Kubernetes command-line interface
- `helm` - Helm package manager
- `openssl` - OpenSSL toolkit for generating certificates
- `jq` - JSON processor

Prepare your environment

1. Export the namespace and other variables as environment variables to simplify further configuration:

```
export NAMESPACE="vault"
export VAULT_HELM_VERSION="0.30.0"
export SERVICE="vault"
export CSR_NAME="vault-csr"
export SECRET_NAME_VAULT="vault-secret"
export POLICY_NAME="mysql-policy"
export WORKDIR="/tmp/vault"
```

2. Create a working directory where for certificate and configuration files:

```
mkdir -p /tmp/vault
```

3. It is a good practice to isolate workloads in Kubernetes using namespaces. Create a namespace with the following command:

```
kubectl create namespace vault
```

Generate certificates

To use TLS, you'll need the following certificates:

- A private key for the Vault server
- A certificate for the Vault server signed by the Kubernetes CA
- The Kubernetes CA certificate

These files store sensitive information. Make sure to keep them in a secure location.

Generate the private key

Generate a private key for the Vault server:

```
openssl genrsa -out ${WORKDIR}/vault.key 2048
```

Create the Certificate Signing Request (CSR)

A Certificate Signing Request (CSR) is a file that contains information about your server and the certificate you need. You create it using your private key, and then submit it to Kubernetes to get a certificate signed by the Kubernetes Certificate Authority (CA). The signed certificate proves your server's identity and enables secure TLS connections.

1. Create the Certificate Signing Request configuration file:

   Specify the certificate details that Kubernetes needs to sign your certificate:

   - **Request settings** (`[req]`): References the sections for certificate extensions and distinguished name. The distinguished name section is left empty as Kubernetes will populate it automatically.
   - **Certificate extensions** (`[v3_req]`): Defines how the certificate can be used. `serverAuth` allows the certificate for server authentication, while `keyUsage` specifies the cryptographic operations the certificate supports (non-repudiation, digital signature, and key encipherment).
   - **Subject Alternative Names** (`[alt_names]`): Lists all DNS names and IP addresses where your Vault service can be accessed. This includes the service name, fully qualified domain names (FQDNs) for different Kubernetes DNS contexts (namespace-scoped, cluster-scoped with `.svc`, and fully qualified with `.svc.cluster.local`), and the localhost IP address.

   ```
   cat > "${WORKDIR}/csr.conf" <<'EOF'
   [req]
   default_bits = 2048
   prompt = no
   encrypt_key = yes
   default_md = sha256
   distinguished_name = kubelet_serving
   req_extensions = v3_req
   [ kubelet_serving ]
   O = system:nodes
   CN = system:node:*.vault.svc.cluster.local
   [ v3_req ]
   basicConstraints = CA:FALSE
   keyUsage = nonRepudiation, digitalSignature, keyEncipherment, dataEncipherment
   extendedKeyUsage = serverAuth, clientAuth
   subjectAltName = @alt_names
   [alt_names]
   DNS.1 =*.vault-internal
   DNS.2 = *.vault-standby
   DNS.3 =*.vault-internal.vault.svc.cluster.local
   DNS.4 = *.vault-standby.vault.svc.cluster.local
   DNS.5 =*.vault
   DNS.6 = vault.vault.svc.cluster.local
   IP.1 = 127.0.0.1
   EOF
   ```

2. Generate the CSR. The following command creates the Certificate Signing Request file using your private key and the configuration file.

   The `-subj` parameter specifies the distinguished name directly: the Common Name (CN) identifies your Vault service using the Kubernetes node naming convention (`system:node:${SERVICE}.${NAMESPACE}.svc`), and the Organization (O) field is set to `system:nodes`, which Kubernetes requires to recognize and sign the certificate. The command combines these subject fields with the certificate extensions defined in the configuration file to produce the complete CSR.

   ```
   openssl req -new -key $WORKDIR/vault.key \
     -subj "/CN=system:node:${SERVICE}.${NAMESPACE}.svc;/O=system:nodes" \
     -out $WORKDIR/server.csr -config $WORKDIR/csr.conf
   ```

Issue the certificate

To get your certificate signed by Kubernetes, you need to submit the CSR through the Kubernetes API. The CSR file you generated with OpenSSL must be wrapped in a Kubernetes CertificateSigningRequest resource.

1. Create the CSR YAML file to send it to Kubernetes:

This YAML file creates a Kubernetes CertificateSigningRequest object that contains your CSR. The file embeds the base64-encoded CSR content and specifies:

- The signer name (`kubernetes.io/kubelet-serving`) that tells Kubernetes which CA should sign the certificate
- The groups field (`system:authenticated`) that identifies who can approve this CSR
- The certificate usages that define how the certificate can be used (digital signature, key encipherment, and server authentication)

```
cat > $WORKDIR/csr.yaml <<EOF
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: ${CSR_NAME}
spec:
  groups:
  - system:authenticated
  request: $(cat $WORKDIR/server.csr | base64 | tr -d '\n')
  signerName: kubernetes.io/kubelet-serving
  usages:
  - digital signature
  - key encipherment
  - server auth
EOF
```

2. Create the CertificateSigningRequest (CSR) object:

```
kubectl create -f ${WORKDIR}/csr.yaml
```

3. Approve the CSR in Kubernetes:

```
kubectl certificate approve ${CSR_NAME}
```

4. Confirm the certificate was issued:

```
kubectl get csr ${CSR_NAME}
```

Sample output

```
NAME        AGE   SIGNERNAME                     REQUESTOR       REQUESTEDDURATION   CONDITION
vault-csr   16s   kubernetes.io/kubelet-serving   minikube-user   <none>              Approved,Issued
```

Retrieve the certificates

After Kubernetes approves and signs your CSR, you need to retrieve the signed certificate and the Kubernetes CA certificate. These certificates are required to configure TLS for your Vault server.

1. Retrieve the signed certificate from the CertificateSigningRequest object. The certificate is base64-encoded in Kubernetes, so you decode it and save it to a file.

```
kubectl get csr ${CSR_NAME} -o jsonpath='{.status.certificate}' | base64 -d > $WORKDIR/vault.crt
```

2. Retrieve Kubernetes CA certificate:

   This command retrieves the Kubernetes cluster's Certificate Authority (CA) certificate from your `kubeconfig` file. The CA certificate is needed to verify that the signed certificate is valid and was issued by the Kubernetes CA. The command uses `kubectl config view` with flags to get the raw, flattened configuration and extract the CA certificate data, which is also base64-encoded.

```
kubectl config view \
  --raw \
  --minify \
  --flatten \
  -o jsonpath='{.clusters[].cluster.certificate-authority-data}' \
  | base64 -d > ${WORKDIR}/vault.ca
```

Store certificates in Kubernetes secrets

Create a TLS secret in Kubernetes to store the certificates and key:

```
kubectl create secret generic ${SECRET_NAME_VAULT} \
  --namespace ${NAMESPACE} \
  --from-file=vault.key=$WORKDIR/vault.key \
  --from-file=vault.crt=$WORKDIR/vault.crt \
  --from-file=vault.ca=$WORKDIR/vault.ca
```

Install Vault with TLS

For this setup, we install Vault in Kubernetes using the Helm 3 package manager in High Availability (HA) mode with Raft storage backend and with TLS enabled.

1. Add and update the Vault Helm repository:

   ```
   helm repo add hashicorp https://helm.releases.hashicorp.com
   helm repo update
   ```

2. Install Vault with TLS enabled:

   ```
   helm upgrade --install ${SERVICE} hashicorp/vault \
     --disable-openapi-validation \
     --version ${VAULT_HELM_VERSION} \
     --namespace ${NAMESPACE} \
     --set "global.enabled=true" \
     --set "global.tlsDisable=false" \
     --set "global.platform=kubernetes" \
     --set server.extraEnvironmentVars.VAULT_CACERT=/vault/userconfig/${SECRET_NAME_VAULT}/vault.ca \
     --set "server.extraEnvironmentVars.VAULT_TLSCERT=/vault/userconfig/${SECRET_NAME_VAULT}/vault.crt" \
     --set "server.extraEnvironmentVars.VAULT_TLSKEY=/vault/userconfig/${SECRET_NAME_VAULT}/vault.key" \
     --set "server.volumes[0].name=userconfig-${SECRET_NAME_VAULT}" \
     --set "server.volumes[0].secret.secretName=${SECRET_NAME_VAULT}" \
     --set "server.volumes[0].secret.defaultMode=420" \
     --set "server.volumeMounts[0].mountPath=/vault/userconfig/${SECRET_NAME_VAULT}" \
     --set "server.volumeMounts[0].name=userconfig-${SECRET_NAME_VAULT}" \
     --set "server.volumeMounts[0].readOnly=true" \
     --set "server.ha.enabled=true" \
     --set "server.ha.replicas=3" \
     --set "server.ha.raft.enabled=true" \
     --set "server.ha.raft.setNodeId=true" \
     --set-string "server.ha.raft.config=cluster_name = \"vault-integrated-storage\"
   ui = true
   listener \"tcp\" {
     tls_disable = 0
     address = \"[::]:8200\"
     cluster_address = \"[::]:8201\"
     tls_cert_file = \"/vault/userconfig/${SECRET_NAME_VAULT}/vault.crt\"
     tls_key_file  = \"/vault/userconfig/${SECRET_NAME_VAULT}/vault.key\"
     tls_client_ca_file = \"/vault/userconfig/${SECRET_NAME_VAULT}/vault.ca\"
   }
   storage \"raft\" {
     path = \"/vault/data\"
   }
   disable_mlock = true
   service_registration \"kubernetes\" {}"
   ```

   This command does the following:

   - Installs HashiCorp Vault in High Availability (HA) mode with secure TLS enabled in your Kubernetes cluster.
   - Configures Vault pods to use certificates from a Kubernetes Secret via volume mounts for secure HTTPS communication between Vault and clients.
   - Sets up Raft as the backend storage with three replicas for fault tolerance, and configures the Vault TCP listener to enforce TLS with your specified certificate files.

Sample output

```
NAME: vault
LAST DEPLOYED: Wed Aug 20 12:55:38 2025
NAMESPACE: vault
STATUS: deployed
REVISION: 1
NOTES:
Thank you for installing HashiCorp Vault!

Now that you have deployed Vault, you should look over the docs on using
Vault with Kubernetes available here:

https://developer.hashicorp.com/vault/docs
```

3. Retrieve the Pod name where Vault is running:

```
kubectl -n $NAMESPACE get pod -l app.kubernetes.io/name=${SERVICE} -o jsonpath='{range .items[*]}{.metadata.name}{"\n"}{end}'
```

Sample output

```
vault-0
vault-1
vault-2
```

Initialize and unseal Vault

1. After Vault is installed, you need to initialize it. Run the following command to initialize the first pod:

```
kubectl exec -it pod/vault-0 -n $NAMESPACE -- vault operator init -key-shares=1 -key-threshold=1 -format=json >
${WORKDIR}/vault-init
```

The command does the following:

- Connects to the Vault Pod
- Initializes Vault server with TLS enabled
- Creates 1 unseal key share which is required to unseal the server
- Outputs the init response to a local file. The file includes unseal keys and root token.

2. Vault is started in a sealed state. In this state Vault can access the storage but it cannot decrypt data. In order to use Vault, you need to unseal it.

Retrieve the unseal key from the file:

```
unsealKey=$(jq -r ".unseal_keys_b64[]" < ${WORKDIR}/vault-init)
```

Now, unseal Vault. Run the following command:

```
kubectl exec -it pod/vault-0 -n $NAMESPACE -- vault operator unseal "$unsealKey"
```

Sample output

```
Key                     Value
---                     -----
Seal Type               shamir
Initialized             true
Sealed                  false
Total Shares            1
Threshold               1
Version                 1.19.0
Build Date              2025-03-04T12:36:40Z
Storage Type            raft
Cluster Name            vault-integrated-storage
Cluster ID              ed275c91-e227-681b-5aaa-f7a9fc19e37e
Removed From Cluster    false
HA Enabled              true
HA Cluster              <https://vault-0.vault-internal:8201>
HA Mode                 active
Active Since            2025-12-15T13:36:42.542059059Z
Raft Committed Index    37
Raft Applied Index      37
```

3. Add the remaining Pods to the Vault cluster. If you have another secret name, replace the `vault-secret` with your value in the following for loop:

```
for POD in vault-1 vault-2; do
  kubectl -n "$NAMESPACE" exec $POD -- sh -c '
    vault operator raft join -address=https://${HOSTNAME}.vault-internal:8200 \
      -leader-ca-cert="$(cat /vault/userconfig/vault-secret/vault.ca)" \
      -leader-client-cert="$(cat /vault/userconfig/vault-secret/vault.crt)" \
      -leader-client-key="$(cat /vault/userconfig/vault-secret/vault.key)" \
      https://vault-0.vault-internal:8200;
  '
done
```

The command connects to each Vault Pod (`vault-1` and `vault-2`) and issues the `vault operator raft join` command, which: * Joins the Pods to the Vault Raft cluster, enabling HA mode. * Uses the necessary TLS certificates to securely connect to the cluster leader (`vault-0`). * Ensures all nodes participate in the Raft consensus and share storage responsibilities.

Sample output

```
Key       Value
---       -----
Joined    true
```

4. Unseal the remaining Pods. Use this for loop:

```
for POD in vault-1 vault-2; do
    kubectl -n "$NAMESPACE" exec $POD -- sh -c "
        vault operator unseal \"$unsealKey\"
    "
done
```

Expected output

```
Key                     Value
---                     -----
Seal Type               shamir
Initialized             true
Sealed                  true
Total Shares            1
Threshold               1
Unseal Progress         0/1
Unseal Nonce            n/a
Version                 1.19.0
Build Date              2025-03-04T12:36:40Z
Storage Type            raft
Removed From Cluster    false
HA Enabled              true
```

Configure Vault

At this step you need to configure Vault and enable secrets within it. To do so you must first authenticate in Vault.

When you started Vault, it generates and starts with a root token that provides full access to Vault. Use this token to authenticate.

Run the following command on a leader node. The remaining ones will synchronize from the leader.

1. Extract the Vault root token from the file where you saved the init response output:

   ```
   cat ${WORKDIR}/vault-init | jq -r ".root_token"
   ```

   Sample output

   ```
   hvs.*************Jg9r
   ```

2. Connect to Vault Pod:

   ```
   kubectl exec -it vault-0 -n $NAMESPACE -- /bin/sh
   ```

3. Authenticate in Vault with this token:

   ```
   vault login hvs.*************Jg9r
   ```

4. Enable the secrets engine at the mount path. The following command enables KV secrets engine v2 at the `secret` mount-path:

   ```
   vault secrets enable --version=2 -path=pxc-secret kv
   ```

   Sample output

   ```
   Success! Enabled the kv secrets engine at: secret/
   ```

5. Optionally, enable audit logs for vault:

   ```
   vault audit enable file file_path=/vault/vault-audit.log
   ```

6. Exit the Vault Pod:

   ```
   exit
   ```

Create a non-root token

Using the root token for authentication is not recommended, as it poses significant security risks. Instead, you should create a dedicated, non-root token for the Operator to use when accessing Vault. The permissions for this token are controlled by an access policy. Before you create a token you must first create the access policy.

1. Create a policy for accessing the kv engine path and define the required permissions in the `capabilities` parameter:

   ```
   kubectl -n "$NAMESPACE" exec vault-0 -- sh -c "
     vault policy write $POLICY_NAME - <<EOF
   path \"pxc-secret/data/*\" {
     capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\"]
   }
   path \"pxc-secret/metadata/*\" {
     capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\"]
   }
   path \"pxc-secret/*\" {
     capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\"]
   }
   EOF
   "
   ```

2. Now create a token with a policy.

```
kubectl -n "${NAMESPACE}" exec pod/vault-0 -- vault token create -policy="${POLICY_NAME}" -format=json > "${WORKDIR}/vault-token.json
```

3. Export the non-root token as an environment variable:

```
export NEW_TOKEN=$(jq -r '.auth.client_token' "${WORKDIR}/vault-token.json")
```

4. Verify the token:

```
echo "New Vault Token: $NEW_TOKEN"
```

Sample output

```
hvs.CAESINO****************************************T2Y
```

Create a Secret for Vault

To enable Vault for the Operator, create a Secret object for it. To do so, create a YAML configuration file and specify the following information:

- A token to access Vault
- A Vault server URL (using HTTPS)
- The secrets mount path
- Path to TLS certificates
- Contents of the `ca.cert` certificate file. This is the `vault.ca` file in our example.

Depending on Percona XtraDB Cluster version, you must specify the Vault configuration as follows:

- For Percona XtraDB Cluster 8.0 and 5.7 - as key=value pairs
- For Percona XtraDB Cluster 8.4 - as a JSON object

You can modify the example configuration file:

Percona XtraDB Cluster 8.0

```
apiVersion: v1
kind: Secret
metadata:
  name: keyring-secret-vault
type: Opaque
stringData:
  keyring_vault.conf: |-
    token = hvs.CAESINO****************************************T2Y
    vault_url = https://vault.vault.svc.cluster.local:8200
    secret_mount_point = secret
    vault_ca = /etc/mysql/vault-keyring-secret/ca.cert
  ca.cert: |-
    -----BEGIN CERTIFICATE-----
    MIIEczCCA1ugAwIBAgIBADANBgkqhkiG9w0BAQQFAD..AkGA1UEBhMCR0Ix
    EzARBgNVBAgTClNvbWUtU3RhdGUxFDASBgNVBAoTC0..0EgTHRkMTcwNQYD
    7vQMfXdGsRrXNGRGnX+vWDZ3/zWI0joDtCkNnqEpVn..HoX
    -----END CERTIFICATE-----
```

Replace the certificate content with the actual CA certificate from `${WORKDIR}/vault.ca`.

Percona XtraDB Cluster 8.4

```
apiVersion: v1
kind: Secret
metadata:
  name: keyring-secret-vault-84
type: Opaque
stringData:
  keyring_vault.conf: |-
    {
      "token": "hvs.CAESINO****************************************T2Y",
      "vault_url": "https://vault.vault.svc.cluster.local:8200",
      "secret_mount_point": "secret",
      "vault_ca": "/etc/mysql/vault-keyring-secret/ca.cert"
    }
  ca.cert: |-
    -----BEGIN CERTIFICATE-----
    MIIEczCCA1ugAwIBAgIBADANBgkqhkiG9w0BAQQFAD..AkGA1UEBhMCR0Ix
    EzARBgNVBAgTClNvbWUtU3RhdGUxFDASBgNVBAoTC0..0EgTHRkMTcwNQYD
    7vQMfXdGsRrXNGRGnX+vWDZ3/zWI0joDtCkNnqEpVn..HoX
    -----END CERTIFICATE-----
```

Replace the certificate content with the actual CA certificate from `${WORKDIR}/vault.ca`.

Note

You must either specify the certificate value or don't declare it at all. Having a commented `#ca.cert` field in the Secret configuration file is not allowed.

Now create a Secret object for your database cluster. Replace the `<cluster-namespace>` placeholder with the namespace where your database cluster is deployed:

```
kubectl apply -f deploy/vault-secret.yaml -n <cluster-namespace>
```

Reference the Secret in your Custom Resource manifest

Now, reference the Vault Secret in the Operator Custom Resource manifest. Note that the Secret name is the one you specified in the `metadata.name` field when you created a Secret.

1. Export the namespace where the cluster is deployed and the secret name as environment variables:

   ```
   export CLUSTER_NAMESPACE=<cluster-namespace>
   export SECRET_NAME_PXC="keyring-secret-vault-84"
   ```

2. Update the cluster configuration. Since this is a running cluster, we will apply a patch referencing your Secret.

   Make sure to specify the correct Secret name. The default Secret name for MySQL 8.0 and 5.7 is `keyring-secret-vault`. In this setup we use `keyring-secret-vault-84` Secret name for MySQL 8.4. Use the following command as an example and specify the Secret name for the MySQL version you're using:

```
kubectl patch pxc cluster1 \
  --namespace $CLUSTER_NAMESPACE \
  --type=merge \
  --patch "{\"spec\":{\"vaultSecretName\":\"${SECRET_NAME_PXC}\"}}"
```

This triggers cluster Pods to restart.

Use data-at-rest encryption

To use encryption, you can:

- turn it on for every table you create with the `ENCRYPTION='Y'` clause in your SQL statement. For example:

```
CREATE TABLE t1 (c1 INT, PRIMARY KEY pk(c1)) ENCRYPTION='Y';
CREATE TABLESPACE foo ADD DATAFILE 'foo.ibd' ENCRYPTION='Y';
```

Existing tables will not be encrypted unless you specifically enable it via the `ALTER TABLE .... ENCRYPTION='Y';` statement.

- turn on default encryption of a schema or a general tablespace. Then all tables you create will have encryption enabled. To turn on default encryption, use the following SQL statement:

```
SET default_table_encryption=ON;
```

See the following chapters of Percona Server for MySQL documentation in how to use encryption:

- Encrypt File-Per-Table Tablespace
- Encrypt schema or general tablespace
- Encrypt system tablespace
- Encrypt doublewrite file pages
- Encrypt temporary files

Verify encryption

Refer to the Percona Server for MySQL documentation for guidelines how to verify encryption in your database.

Troubleshooting

If you encounter issues during the setup, use the following troubleshooting tips:

1. **Certificate Signing Request (CSR) issues**: If you have problems with the CSR, manually delete it and recreate it:

   ```
   kubectl delete csr vault-csr || true
   ```

   Then recreate and re-approve it in Kubernetes following the steps in the Issue the certificate section.
2. **Vault policy issues**: Check the mount points and permissions. Ensure that:
   - The mount path in your policy matches the path where you enabled the secrets engine
   - The policy has the required capabilities (`create`, `read`, `update`, `delete`, `list`) for the paths your application needs
3. **Mount point conflicts**: If you encounter issues with a mount point in Vault, you cannot reuse it. You need to:
   - Provide a new mount path when enabling the secrets engine
   - Update your access policy to include the new mount path
   - Update the `secret_mount_point` value in your Secret configuration to match the new mount path
4. **Secret format mismatch**: Check the MySQL version in your cluster and ensure you apply the correct format of the secret:
   - For Percona XtraDB Cluster 8.0 and 5.7 - use key=value pairs format
   - For Percona XtraDB Cluster 8.4 - use JSON object format

   Verify that the secret is created in the same namespace as your database cluster.
5. Verify that you reference the correct secret name in your Custom Resource.

Clean up

After you finish working with Vault, you can clean up the temporary files:

```
rm -rf $WORKDIR
```

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)
- PMM Version
- Operator version
- Percona XtraDB Cluster version
- HAProxy version
- ProxySQL version
- Percona XtraBackup version
- Is Operator deployed in a cluster-wide mode

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 by setting the `DISABLE_TELEMETRY` environment variable in the Operator Deployment.

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

```
$ helm install my-db percona/pxc-db --version 1.19.0 --namespace my-namespace --set disable_telemetry="true"
```

- if you don't use Helm for installation, configure the `DISABLE_TELEMETRY` environment variable in the Operator Deployment.

See Operator environment variables for detailed instructions.

Configure concurrency for a cluster reconciliation

Reconciliation is the process by which the Operator continuously compares the desired state with the actual state of the cluster. The desired state is defined in a Kubernetes custom resource, like `PerconaXtraDBCluster`.

If the actual state does not match the desired state, the Operator takes actions to bring the system into alignment. This means creating, updating, or deleting Kubernetes resources (Pods, Services, ConfigMaps, etc.) or performing database-specific operations like scaling, backups, or failover.

Reconciliation is triggered by a variety of events, including:

- Changes to the cluster configuration
- Changes to the cluster state
- Changes to the cluster resources

By default, the Operator has one reconciliation worker. This means that if you deploy or update 2 clusters at the same time, the Operator will reconcile them sequentially.

The `MAX_CONCURRENT_RECONCILES` environment variable in the `percona-xtradb-cluster-operator` deployment controls the number of concurrent workers that can reconcile resources in Percona XtraDB Cluster clusters in parallel.

Thus, to extend the previous example, if you set the number of reconciliation workers to `2`, the Operator will reconcile both clusters in parallel. This also helps you with benchmarking the Operator performance.

The general recommendation is to set the number of concurrent workers equal to the number of Percona XtraDB Cluster clusters. When the number of workers is greater, the excessive workers will remain idle.

Set the number of reconciliation workers

1. Check the index of the `MAX_CONCURRENT_RECONCILES` environment variable using the following command:

   ```
   kubectl get deployment percona-xtradb-cluster-operator -o jsonpath='{.spec.template.spec.containers[0].env[?
   (@.name=="MAX_CONCURRENT_RECONCILES")].value}'
   ```

   Sample output

   ```
   1
   ```

2. To set a new value and verify it's been updated, run the following command:

   ```
   $ kubectl patch deployment percona-xtradb-cluster-operator \
   --type='strategic' \
   -o yaml \
   -p='{
       "spec": {
           "template": {
               "spec": {
                   "containers": [
                       {
                           "name": "percona-xtradb-cluster-operator",
                           "env": [
                               {
                                   "name": "MAX_CONCURRENT_RECONCILES",
                                   "value": "2"
                               }
                           ]
                       }
                   ]
               }
           }
       }
   }'\
   -o jsonpath='{.spec.template.spec.containers[0].env[?(@.name=="MAX_CONCURRENT_RECONCILES")].value}'
   ```

The command does the following:

- Patches the deployment to update the `MAX_CONCURRENT_RECONCILES` environment variable

- Sets the value to `2`.
- Outputs the result

You can set the value to any number greater than 0.

Sample output

```{.text .no-copy}
2
```

Management

Backup and restore

Providing Backups

It's important to back up your database to keep your data safe. Backups help protect your system against data loss and corruption and ensure business stability. They are also a quick way to recover the database if something happens with it.

A backup starts after you create a Backup object. You can create a Backup object in two ways:

- manually at any moment. This way you start an on-demand backup.
- instruct the Operator to create it automatically according to a schedule that you define for it. This is a scheduled backup.

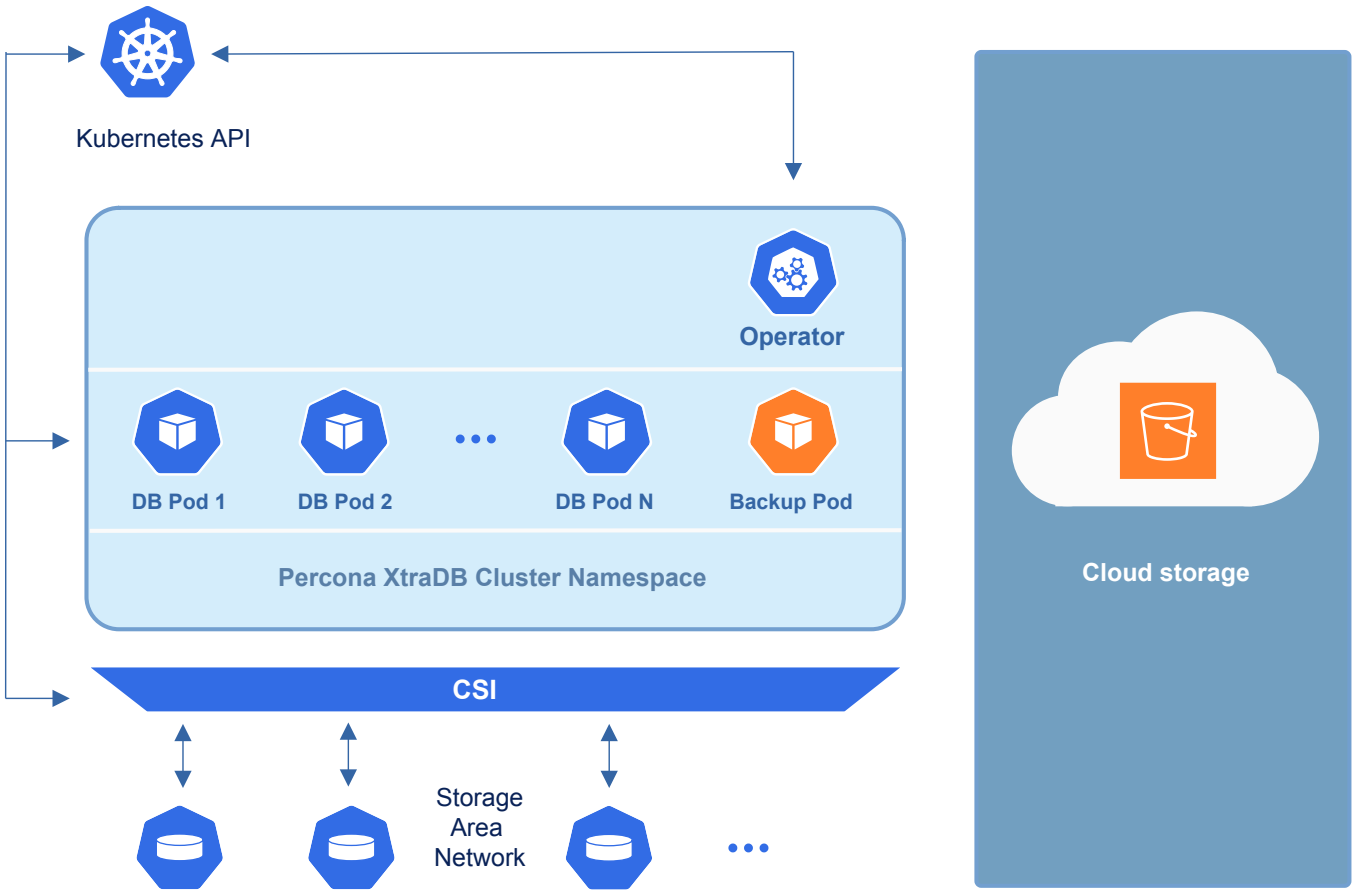
The Operator does physical backups using the Percona XtraBackup tool. By default, it uses the SST (State Snapshot Transfer) method, which creates a separate backup Pod to perform backups.

Alternatively, you can enable the XtraBackup sidecar method. This method uses a sidecar container running in each PXC Pod that provides better performance and native encryption support. See Backup methods for details.

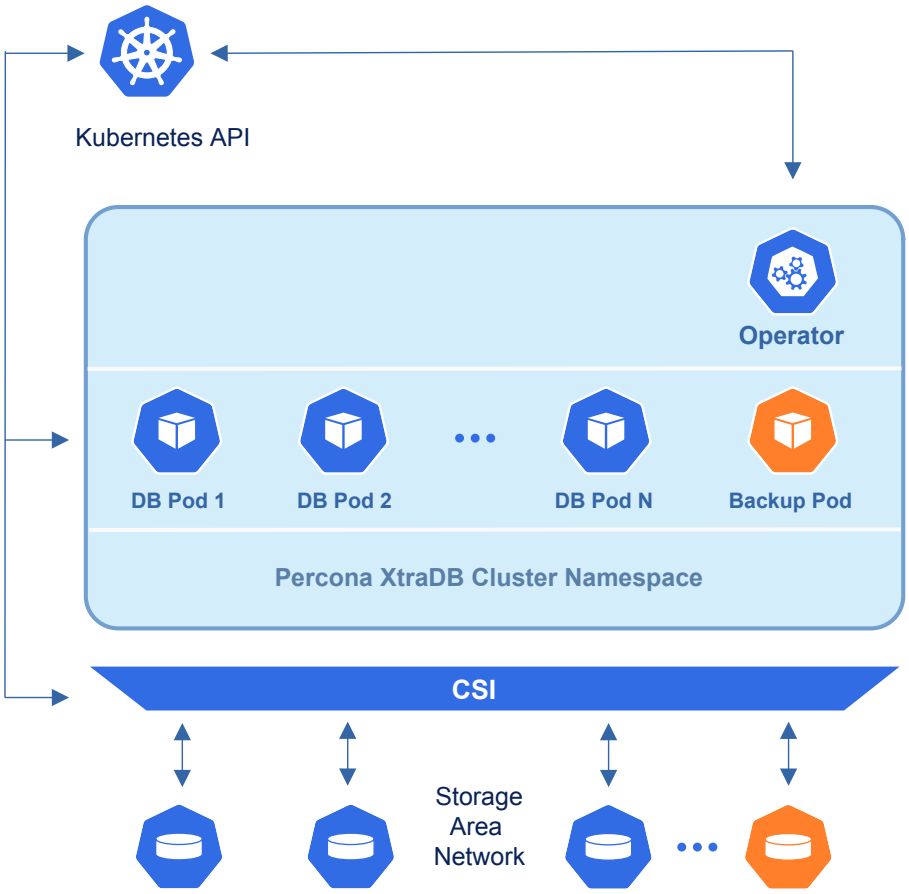
Backup storage

You can store backups outside of Kubernetes cluster in one of the supported cloud storages:

- Amazon S3 or S3-compatible storage,
- Azure Blob Storage:



If you're running a Kubernetes cluster on premises, you can store backups inside it using a Persistent Volume. For example, if you don't use a remote backup storage or if storage costs are high for you.



Backup methods

The Operator supports two backup methods:

SST method (default)

The default backup method uses State Snapshot Transfer (SST). When you create a Backup object, the Operator:

1. Sets up a backup Pod that runs Percona XtraBackup inside and creates a backup Job
2. Creates a path in the storage to save the backup data
3. Starts copying the data files from the Percona XtraDB Cluster to the backup storage
4. The Percona XtraDB Cluster Pod that serves the data enters the Donor state and stops receiving all requests

The backup task is resource-consuming and can affect performance. That's why the Operator uses one of the secondary Percona XtraDB Cluster Pods for backups. The exception is a one-pod deployment, where the same Pod is used for all tasks.

After the data files are copied and uploaded to the remote backup storage, the Operator marks the backup Pod as 'Completed' and deletes it. The Operator also updates the status of the Backup object.

XtraBackup sidecar method (tech preview)

When you enable the `XtrabackupSidecar` feature gate, the Operator uses a different backup approach:

1. An XtraBackup sidecar container runs in each Percona XtraDB Cluster Pod, providing a gRPC server interface for making backups.
2. When you create a Backup object, the Operator creates a Job that acts like a client and sends requests directly to the sidecar.
3. The sidecar performs the backup and uploads it to the cloud storage (S3, Azure, or GCP). The database Pod doesn't change its state to Donor and continues processing all requests.

As with the SST method, the Operator uses one of the secondary Percona XtraDB Cluster Pods for backups to not overload the primary Pod.

Benefits of the XtraBackup sidecar method:

- **Better performance**: Direct access to data files without network overhead
- **Easier troubleshooting**: The sidecar container runs continuously in the Percona XtraDB Cluster Pod, so you can check backup logs and status at any time. SST backups may fail with cryptic errors when a network issue occurs. This makes it difficult to diagnose the root cause or intervene to resolve problems.
- **Native encryption**: Built-in support for encrypted backups with proper key management. This functionality is not yet available in version 1.19.0 but will be added in future releases.
- **Incremental backups**: Make your backups more efficient by saving only the data that has changed since the last backup, rather than copying the entire database each time. This reduces the amount of backup storage required and allows you to take backups more frequently with less impact on performance. This functionality is not yet available in version 1.19.0 but will be added in future releases.

Limitations:

- PVC (Persistent Volume Claim) backups are not supported when this feature is enabled. This support is planned to be implemented in future releases.
- Only cloud storage backups (S3, Azure, GCP) are available

To enable this method, set `PXCO_FEATURE_GATES=XtrabackupSidecar=true` in the Operator Deployment. See Configure Operator environment variables for detailed instructions.

Multiple backups

You can run several backups. For example, schedule weekly backups on one storage and daily backups on another one. You can also run an on-demand backup to be on the safe side before you do some maintenance work.

Several backups run in parallel by default if they happen at the same time. If they overload your cluster, you can turn off parallel backups with the `backup.allowParallel` configuration option in the `cr.yaml` file. Then, the Operator queues the backups and runs them sequentially.

The Operator ensures the sequence by creating a lock for a running backup. It releases the lock after the backup either succeeds or fails and starts the next one from the queue. The lock is also released if you delete a running backup.

You can fine-tune the queue by assigning a waiting time for a backup to start. Use the `spec.startingDeadlineSeconds` option in the `deploy/cr.yaml` file to set this time for all backups. You can also override it for a specific on-demand backup by defining the `startingDeadlineSeconds` option within the backup configuration. This setting has a higher priority.

If the backup doesn't start within the defined time, the Operator automatically marks it as "failed".

Configure automatic cleanup of backup Jobs and Pods

You can specify the time to live for a backup Job after the backup operation finishes. When the TTL expires, the Operator automatically deletes the Job and its associated Pod.

Use the `backup.ttlSecondsAfterFinished` setting in the `deploy/cr.yaml` file to set this time for all backups, both on-demand and scheduled. This setting also applies for restores.

If it takes longer to finish a backup than the defined `backup.ttlSecondsAfterFinished` value, the Operator applies the `internal.percona.com/keep-job` finalizer to allow the operation to finish. After the operation completes with the Succeeded or Failed status, the finalizer is removed and the Job is cleaned up.

Backup suspension for an unhealthy database cluster

Your database cluster can become unhealthy. For example, when one of the Pods crashes and restarts. The Operator monitors the database cluster state while a backup is running and suspends it for an unhealthy cluster to reduce the load on the cluster.

To offload the database cluster even more, you can define how long a backup remains suspended. Use the `spec.backup.suspendedDeadlineSeconds` option in the `cr.yaml` file for all backups. Or set it in the `backup.yaml` configuration files for a specific backup. The setting in the `backup.yaml` file has a higher priority.

After this duration expires, the Operator automatically marks this backup as "failed".

Otherwise, after the cluster is recovered and reports the Ready status, the Operator resumes the backup and tries to finish it.

Note that if some files were already saved on the storage when a backup was suspended, the Operator deletes them and reruns the backup.

If you want to run backups in an unhealthy cluster, set the `spec.unsafeFlags.backupIfUnhealthy` option in the `deploy/cr.yaml` file to `true`. Use this option with caution because it can affect the cluster performance.

Configure storage for backups

You can configure storage for backups in the `backup.storages` subsection of the Custom Resource, using the deploy/cr.yaml configuration file.

Before configuring the storage, you need to create a Kubernetes Secret object that contains the credentials needed to access your storage.

Amazon S3 or S3-compatible storage

To use Amazon S3 or S3-compatible storage for backups, create a Secret object with your access credentials. Use the deploy/backup/backup-secret-s3.yaml file as an example. You must specify the following information:

- the `metadata.name` key is the name the Kubernetes secret which you will reference in the Custom Resource
- `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` are base64-encoded keys to access S3 storage

Use the following command to encode the keys:

in Linux

```
echo -n 'plain-text-string' | base64 --wrap=0
```

in macOS

```
echo -n 'plain-text-string' | base64
```

Here's the example configuration of the Secret file:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-cluster-name-backup-s3
type: Opaque
  data:
    AWS_ACCESS_KEY_ID: <YOUR-BASE64-ENCODED-KEY-HERE>
    AWS_SECRET_ACCESS_KEY: <YOUR-BASE64-ENCODED-SECRET-HERE>
```

1. Create the Kubernetes Secret object with this file:

   ```
   $ kubectl apply -f deploy/backup/backup-secret-s3.yaml -n <namespace>
   ```

   Note

   If the previous backup attempt fails (because of a temporary networking problem, etc.) the backup job tries to delete the unsuccessful backup leftovers first, and then makes a retry. Therefore there will be no backup retry without DELETE permissions to the objects in the bucket. Also, setting Google Cloud Storage Retention Period can cause a similar problem.

2. Configure the storage in the Custom Resource. Modify the `deploy/cr.yaml` file and define the following information:

   - `storages.<NAME>.type` - set to `s3`. Substitute the part with some name you will later use to refer this storage when making backups and restores.
   - `storages.<NAME>.s3.credentialsSecret` set to the name of the Secret you created previously
   - `storages.<NAME>.s3.bucket` - where the data will be stored
   - `storages.<NAME>.s3.region` - location of the bucket

   Here is an example of the deploy/cr.yaml configuration file which configures Amazon S3 storage for backups:

```
...
backup:
  ...
  storages:
    s3-us-west:
      type: s3
      s3:
        bucket: S3-BACKUP-BUCKET-NAME-HERE
        region: us-west-2
        credentialsSecret: my-cluster-name-backup-s3
        caBundle:
          name: minio-ca-bundle
          key: tls.cert
  ...
```

S3-compatible storage

If you use S3-compatible storage instead of Amazon S3, add the `endpointUrl` option in the `s3` subsection. This points to your storage service and is specific to your cloud provider. For example, for MinIO:

```
endpointUrl: https://minio-service:9000
```

For more configuration options, see the Operator Custom Resource options.

Microsoft Azure Blob storage

1. To store backups on the Azure Blob storage, you need to create a Secret with the following values:
   - the `metadata.name` key is the name which you will further use to refer your Kubernetes Secret,
   - the `data.AZURE_STORAGE_ACCOUNT_NAME` and `data.AZURE_STORAGE_ACCOUNT_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 following the deploy/backup/backup-secret-azure.yaml example:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: azure-secret
   type: Opaque
   data:
     AZURE_STORAGE_ACCOUNT_NAME: UkVQTEFDRS1XSVRILUFXUy1BQ0NFU1MtS0VZ
     AZURE_STORAGE_ACCOUNT_KEY: UkVQTEFDRS1XSVRILUFXUy1TRUNSRVQtS0VZ
   ```

   Note

   You can use the following command to get a base64-encoded string from a plain text one:

   in Linux

   ```
   $ echo -n 'plain-text-string' | base64 --wrap=0
   ```

   in macOS

   ```
   $ echo -n 'plain-text-string' | base64
   ```

   Once the editing is over, create the Kubernetes Secret object as follows:

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

2. Put the data needed to access the Azure Blob storage into the `backup.storages` subsection of the Custom Resource.
   - `storages.<NAME>.type` should be set to `azure` (substitute the `<NAME>` part with some arbitrary name you will later use to refer this storage when making backups and restores).
   - `storages.<NAME>.azure.credentialsSecret` key should be set to the name used to refer your Kubernetes Secret (`azure-secret` in the last example).
   - `storages.<NAME>.azure.container` option should contain the name of the Azure container.

   The options within the `storages.<NAME>.azure` subsection are further explained in the Operator Custom Resource options.

Here is an example of the deploy/cr.yaml configuration file which configures Azure Blob storage for backups:

```
...
backup:
  ...
  storages:
    azure-blob:
      type: azure
      azure:
        container: <your-container-name>
        credentialsSecret: azure-secret
      ...
```

Persistent Volume

Here is an example of the `deploy/cr.yaml` backup section fragment, which configures a private volume for filesystem-type storage:

```
...
backup:
  ...
  storages:
    fs-pvc:
      type: filesystem
      volume:
        persistentVolumeClaim:
          accessModes: [ "ReadWriteOnce" ]
          resources:
            requests:
              storage: 6G
  ...
```

Note

Please take into account that 6Gi storage size specified in this example may be insufficient for the real-life setups; consider using tens or hundreds of gigabytes. Also, you can edit this option later, and changes will take effect after applying the updated `deploy/cr.yaml` file with `kubectl`.

Note

Typically, Percona XtraBackup tools used by the Operator to perform the backup/restore process does not require any additional configuration beyond the standard parameters mentioned above. However, if access to a non-standard cloud requires some fine-tuning, you can pass additional options to the binary XtraBackup utilities using the following Custom Resource options: backup.storages.STORAGE_NAME.containerOptions.args.xtrabackup, backup.storages.STORAGE_NAME.containerOptions.args.xbcloud, and backup.storages.STORAGE_NAME.containerOptions.args.xbstream. Also, you can set environment variables for the XtraBackup container with backup.storages.STORAGE_NAME.containerOptions.env.

Configure TLS verification with custom certificates for S3 storage

Version added: 1.19.0

You can use your organization's custom TLS / SSL certificates and instruct the Operator to securely verify TLS communication with S3 storage.

To configure TLS verification with custom certificates, do the following:

1. Create the Secret object that contains the CA bundle needed to verify the S3 endpoint's TLS certificate.
2. Modify the S3 storage configuration in the Custom Resource and specify the following information:
   - `storages.<NAME>.s3.caBundle.name` is the name of the Secret object you created previously
   - `storages.<NAME>.s3.caBundle.key` is the name of the file in the Secret containing the CA bundle.

   Here's the example configuration:

```
...
backup:
  ...
  storages:
    s3-us-west:
      type: s3
      s3:
        bucket: S3-BACKUP-BUCKET-NAME-HERE
        region: us-west-2
        credentialsSecret: my-cluster-name-backup-s3
        caBundle:
          name: minio-ca-bundle
          key: ca.crt
  ...
```

The Operator will use this configuration to securely verify TLS communication with S3 storage during backups and restores.

Store binary logs for point-in-time recovery

Point-in-time recovery allows users to roll back the cluster to a specific transaction or time. You can even skip a transaction if you don't need it anymore. To make a point-in-time recovery, the Operator needs a backup and binary logs (binlogs) of the server to.

A binary log records all changes made to the database, such as updates, inserts, and deletes. It is used to synchronize data across servers for and point-in-time recovery.

Point-in-time recovery is off by default and is supported by the Operator with Percona XtraDB Cluster versions starting from 8.0.21-12.1.

After you enable point-in-time recovery, the Operator spins up a separate point-in-time recovery Pod, which starts saving binary log updates to the backup storage.

Considerations

1. You must use either s3-compatible or Azure-compatible storage for both binlog and full backup to make the point-in-time recovery work
2. The Operator saves binlogs without any cluster-based filtering. Therefore, either use a separate folder per cluster on the same bucket or use different buckets for binlogs.

   Also,we recommend to have an empty bucket or a folder on a bucket for binlogs when you enable point-in-time recovery. This bucket/folder should not contain no binlogs nor files from previous attempts or other clusters.
3. Don't purge binlogs before they are transferred to the backup storage. Doing so breaks point-in-time recovery.
4. Disable the retention policy as it is incompatible with the point-in-time recovery. To clean up the storage, configure the Bucket lifecycle on the storage

Enable point-in-time recovery

To use point-in-time recovery, set the following keys in the `pitr` subsection under the `backup` section of the deploy/cr.yaml manifest:

- `backup.pitr.enabled` - set it to `true`
- `backup.pitr.storageName` - specify the same storage name that you have defined in the `storages` subsection
- `timeBetweenUploads` - specify the number of seconds between running the binlog uploader

The following example shows how the `pitr` subsection looks like if you use the S3 storage:

```
backup:
  ...
  pitr:
    enabled: true
    storageName: s3-us-west
    timeBetweenUploads: 60
```

For how to restore a database to a specific point in time, see Restore the cluster with point-in-time recovery.

Binary logs statistics

The point-in-time recovery Pod has statistics metrics for binlogs. They provide insights into the success and failure rates of binlog operations, timeliness of processing and uploads and potential gaps or inconsistencies in binlog data.

The available metrics are:

- `pxc_binlog_collector_success_total` - The total number of successful binlog collection cycles. It helps monitor how often the binlog collector successfully processes and uploads binary logs.
- `pxc_binlog_collector_gap_detected_total` - Tracks the total number of gaps detected in the binlog sequence during collection. Highlights potential issues with missing or skipped binlogs, which could impact replication or recovery.
- `pxc_binlog_collector_last_processing_timestamp` - Records the timestamp of the last successful binlog collection operation.
- `pxc_binlog_collector_last_upload_timestamp` - Records the timestamp of the last successful binlog upload to the storage
- `pxc_binlog_collector_uploaded_total` - The total number of successfully uploaded binlogs

Gather metrics data

You can connect to the point-in-time recovery Pod using the `<pitr-pod-service>:8080/metrics` endpoint to gather these metrics and further analyze them.

List services to get the point-in-time-recovery service name:

```
$ kubectl get services | grep 'pitr'
```

Expected output

```
cluster1-pitr                       ClusterIP   34.118.225.138   <none>        8080/TCP
```

Access locally via port forwarding

Use this method to access the metrics from your local machine.

1. Forward the Kubernetes service's port:

   ```
   $ kubectl port-forward svc/cluster1-pitr 8080:8080
   ```

2. Open your browser and visit `<http://localhost:8080/metrics>`

   Expected output

   ```
   # HELP go_gc_duration_seconds A summary of the wall-time pause (stop-the-world) duration in garbage collection cycles.
   # TYPE go_gc_duration_seconds summary
   go_gc_duration_seconds{quantile="0"} 0.000109735
   go_gc_duration_seconds{quantile="0.25"} 0.000147529
   go_gc_duration_seconds{quantile="0.5"} 0.000176199
   go_gc_duration_seconds{quantile="0.75"} 0.000196962
   go_gc_duration_seconds{quantile="1"} 0.000570426
   go_gc_duration_seconds_sum 0.002970858
   go_gc_duration_seconds_count 14
   # HELP go_gc_gogc_percent Heap size target percentage configured by the user, otherwise 100. This value is set by the GOGC environment
   variable, and the runtime/debug.SetGCPercent function. Sourced from /gc/gogc:percent.
   # TYPE go_gc_gogc_percent gauge
   go_gc_gogc_percent 100
   # HELP go_gc_gomemlimit_bytes Go runtime memory limit configured by the user, otherwise math.MaxInt64. This value is set by the GOMEMLIMIT
   environment variable, and the runtime/debug.SetMemoryLimit function. Sourced from /gc/gomemlimit:bytes.
   # TYPE go_gc_gomemlimit_bytes gauge
   go_gc_gomemlimit_bytes 9.223372036854776e+18
   # HELP go_goroutines Number of goroutines that currently exist.
   # TYPE go_goroutines gauge
   go_goroutines 31
   # HELP go_info Information about the Go environment.
   # TYPE go_info gauge
   go_info{version="go1.24.3"} 1
   # HELP go_memstats_alloc_bytes Number of bytes allocated in heap and currently in use. Equals to /memory/classes/heap/objects:bytes.
   # TYPE go_memstats_alloc_bytes gauge
   go_memstats_alloc_bytes 2.83268e+06
   # HELP go_memstats_alloc_bytes_total Total number of bytes allocated in heap until now, even if released already. Equals to
   /gc/heap/allocs:bytes.
   # TYPE go_memstats_alloc_bytes_total counter
   go_memstats_alloc_bytes_total 5.80031448e+08
   # HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table. Equals to
   /memory/classes/profiling/buckets:bytes.
   # TYPE go_memstats_buck_hash_sys_bytes gauge
   go_memstats_buck_hash_sys_bytes 5696
   # HELP go_memstats_frees_total Total number of heap objects frees. Equals to /gc/heap/frees:objects + /gc/heap/tiny/allocs:objects.
   # TYPE go_memstats_frees_total counter
   go_memstats_frees_total 112652
   # HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata. Equals to /memory/classes/metadata/other:bytes.
   # TYPE go_memstats_gc_sys_bytes gauge
   go_memstats_gc_sys_bytes 3.83684e+06
   # HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and currently in use, same as go_memstats_alloc_bytes. Equals to
   /memory/classes/heap/objects:bytes.
   # TYPE go_memstats_heap_alloc_bytes gauge
   go_memstats_heap_alloc_bytes 2.83268e+06
   # HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used. Equals to /memory/classes/heap/released:bytes +
   /memory/classes/heap/free:bytes.
   # TYPE go_memstats_heap_idle_bytes gauge
   go_memstats_heap_idle_bytes 5.681152e+08
   # HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.  Equals to /memory/classes/heap/objects:bytes +
   /memory/classes/heap/unused:bytes
   # TYPE go_memstats_heap_inuse_bytes gauge
   go_memstats_heap_inuse_bytes 5.423104e+06
   # HELP go_memstats_heap_objects Number of currently allocated objects. Equals to /gc/heap/objects:objects.
   # TYPE go_memstats_heap_objects gauge
   go_memstats_heap_objects 11876
   # HELP go_memstats_heap_released_bytes Number of heap bytes released to OS. Equals to /memory/classes/heap/released:bytes.
   # TYPE go_memstats_heap_released_bytes gauge
   go_memstats_heap_released_bytes 5.66616064e+08
   # HELP go_memstats_heap_sys_bytes Number of heap bytes obtained from system. Equals to /memory/classes/heap/objects:bytes +
   /memory/classes/heap/unused:bytes + /memory/classes/heap/released:bytes + /memory/classes/heap/free:bytes.
   # TYPE go_memstats_heap_sys_bytes gauge
   ```

```
go_memstats_heap_sys_bytes 5.73538304e+08
# HELP go_memstats_last_gc_time_seconds Number of seconds since 1970 of last garbage collection.
# TYPE go_memstats_last_gc_time_seconds gauge
go_memstats_last_gc_time_seconds 1.7492150571437228e+09
# HELP go_memstats_mallocs_total Total number of heap objects allocated, both live and gc-ed. Semantically a counter version for
go_memstats_heap_objects     gauge. Equals to /gc/heap/allocs:objects + /gc/heap/tiny/allocs:objects.
# TYPE go_memstats_mallocs_total counter
go_memstats_mallocs_total 124528
# HELP go_memstats_mcache_inuse_bytes Number of bytes in use by mcache     structures. Equals to /memory/classes/metadata/mcache/inuse:bytes.
# TYPE go_memstats_mcache_inuse_bytes gauge
go_memstats_mcache_inuse_bytes 4832
# HELP go_memstats_mcache_sys_bytes Number of bytes used for mcache structures obtained from system. Equals to
/memory/classes/metadata/mcache/inuse:bytes + /memory/classes/metadata/mcache/free:bytes.
# TYPE go_memstats_mcache_sys_bytes gauge
go_memstats_mcache_sys_bytes 15704
# HELP go_memstats_mspan_inuse_bytes Number of bytes in use by mspan     structures. Equals to /memory/classes/metadata/mspan/inuse:bytes.
# TYPE go_memstats_mspan_inuse_bytes gauge
go_memstats_mspan_inuse_bytes 125920
# HELP go_memstats_mspan_sys_bytes Number of bytes used for mspan structures obtained from system. Equals to
/memory/classes/metadata/mspan/inuse:bytes + /memory/classes/metadata/mspan/free:bytes.
# TYPE go_memstats_mspan_sys_bytes gauge
go_memstats_mspan_sys_bytes 146880
# HELP go_memstats_next_gc_bytes Number of heap bytes when next garbage     collection will take place. Equals to /gc/heap/goal:bytes.
# TYPE go_memstats_next_gc_bytes gauge
go_memstats_next_gc_bytes 6.04629e+06
# HELP go_memstats_other_sys_bytes Number of bytes used for other system  allocations. Equals to /memory/classes/other:bytes.
# TYPE go_memstats_other_sys_bytes gauge
go_memstats_other_sys_bytes 764832
# HELP go_memstats_stack_inuse_bytes Number of bytes obtained from system for stack allocator in non-CGO environments. Equals to
/memory/classes/heap/stacks:bytes.
# TYPE go_memstats_stack_inuse_bytes gauge
go_memstats_stack_inuse_bytes 1.081344e+06
# HELP go_memstats_stack_sys_bytes Number of bytes obtained from system for stack allocator. Equals to /memory/classes/heap/stacks:bytes +
/memory/classes/os-stacks:bytes.
# TYPE go_memstats_stack_sys_bytes gauge
go_memstats_stack_sys_bytes 1.081344e+06
# HELP go_memstats_sys_bytes Number of bytes obtained from system. Equals to /memory/classes/total:byte.
# TYPE go_memstats_sys_bytes gauge
go_memstats_sys_bytes 5.793896e+08
# HELP go_sched_gomaxprocs_threads The current runtime.GOMAXPROCS setting, or the number of operating system threads that can execute user-
level Go code simultaneously. Sourced from /sched/gomaxprocs:threads.
# TYPE go_sched_gomaxprocs_threads gauge
go_sched_gomaxprocs_threads 4
# HELP go_threads Number of OS threads created.
# TYPE go_threads gauge
go_threads 10
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.55
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP process_network_receive_bytes_total Number of bytes received by the     process over the network.
# TYPE process_network_receive_bytes_total counter
process_network_receive_bytes_total 1.172862e+06
# HELP process_network_transmit_bytes_total Number of bytes sent by the process  over the network.
# TYPE process_network_transmit_bytes_total counter
process_network_transmit_bytes_total 632432
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 9
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 3.9350272e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.74921402754e+09
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.901723648e+09
# HELP process_virtual_memory_max_bytes Maximum amount of virtual memory   available in bytes.
# TYPE process_virtual_memory_max_bytes gauge
process_virtual_memory_max_bytes 1.8446744073709552e+19
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 3
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
# HELP pxc_binlog_collector_failure_total Total number of failed binlog     collection cycles
# TYPE pxc_binlog_collector_failure_total counter
pxc_binlog_collector_failure_total 0
# HELP pxc_binlog_collector_gap_detected_total Total number of times the gap was detected in binlog
# TYPE pxc_binlog_collector_gap_detected_total counter
pxc_binlog_collector_gap_detected_total 0
# HELP pxc_binlog_collector_last_processing_timestamp Timestamp of the last     successful binlog processing
# TYPE pxc_binlog_collector_last_processing_timestamp gauge
pxc_binlog_collector_last_processing_timestamp 1.7492150471803956e+09
# HELP pxc_binlog_collector_last_upload_timestamp Timestamp of the last     successful binlog upload
# TYPE pxc_binlog_collector_last_upload_timestamp gauge
pxc_binlog_collector_last_upload_timestamp 1.749214031447092e+09
```

```
# HELP pxc_binlog_collector_success_total Total number of successful binlog     collection cycles
# TYPE pxc_binlog_collector_success_total counter
pxc_binlog_collector_success_total 19
# HELP pxc_binlog_collector_uploaded_total Total number of successfully     uploaded binlogs
# TYPE pxc_binlog_collector_uploaded_total counter
pxc_binlog_collector_uploaded_total 1
```

Access directly from a Pod

You can gather the metrics from inside a database cluster.

1. Connect to the cluster as follows, replacing the `<namespace>` placeholder with your value:

```
$ kubectl run -n <namespace> -i --rm --tty percona-client --image=percona:8.0 --restart=Never -- bash -il
```

2. Connect to the point-in-time recovery port using `curl`:

```
$ curl cluster1-pitr:8080/metrics
```

Expected output

```
# HELP go_gc_duration_seconds A summary of the wall-time pause (stop-the-world) duration in garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0.000109735
go_gc_duration_seconds{quantile="0.25"} 0.000147529
go_gc_duration_seconds{quantile="0.5"} 0.000176199
go_gc_duration_seconds{quantile="0.75"} 0.000196962
go_gc_duration_seconds{quantile="1"} 0.000570426
go_gc_duration_seconds_sum 0.002970858
go_gc_duration_seconds_count 14
# HELP go_gc_gogc_percent Heap size target percentage configured by the user, otherwise 100. This value is set by the GOGC environment
variable, and the runtime/debug.SetGCPercent function. Sourced from /gc/gogc:percent.
# TYPE go_gc_gogc_percent gauge
go_gc_gogc_percent 100
# HELP go_gc_gomemlimit_bytes Go runtime memory limit configured by the user, otherwise math.MaxInt64. This value is set by the GOMEMLIMIT
environment variable, and the runtime/debug.SetMemoryLimit function. Sourced from /gc/gomemlimit:bytes.
# TYPE go_gc_gomemlimit_bytes gauge
go_gc_gomemlimit_bytes 9.223372036854776e+18
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 31
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.24.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated in heap and currently in use. Equals to /memory/classes/heap/objects:bytes.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.83268e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated in heap until now, even if released already. Equals to
/gc/heap/allocs:bytes.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 5.80031448e+08
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table. Equals to
/memory/classes/profiling/buckets:bytes.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 5696
# HELP go_memstats_frees_total Total number of heap objects frees. Equals to /gc/heap/frees:objects + /gc/heap/tiny/allocs:objects.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 112652
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata. Equals to /memory/classes/metadata/other:bytes.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 3.83684e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and currently in use, same as go_memstats_alloc_bytes. Equals to
/memory/classes/heap/objects:bytes.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 2.83268e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used. Equals to /memory/classes/heap/released:bytes +
/memory/classes/heap/free:bytes.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 5.681152e+08
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.  Equals to /memory/classes/heap/objects:bytes +
/memory/classes/heap/unused:bytes
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 5.423104e+06
# HELP go_memstats_heap_objects Number of currently allocated objects. Equals to /gc/heap/objects:objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 11876
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS. Equals to /memory/classes/heap/released:bytes.
# TYPE go_memstats_heap_released_bytes gauge
go_memstats_heap_released_bytes 5.66616064e+08
# HELP go_memstats_heap_sys_bytes Number of heap bytes obtained from system. Equals to /memory/classes/heap/objects:bytes +
/memory/classes/heap/unused:bytes + /memory/classes/heap/released:bytes + /memory/classes/heap/free:bytes.
# TYPE go_memstats_heap_sys_bytes gauge
go_memstats_heap_sys_bytes 5.73538304e+08
```

```
# HELP go_memstats_last_gc_time_seconds Number of seconds since 1970 of last garbage collection.
# TYPE go_memstats_last_gc_time_seconds gauge
go_memstats_last_gc_time_seconds 1.7492150571437228e+09
# HELP go_memstats_mallocs_total Total number of heap objects allocated, both live and gc-ed. Semantically a counter version for
go_memstats_heap_objects     gauge. Equals to /gc/heap/allocs:objects + /gc/heap/tiny/allocs:objects.
# TYPE go_memstats_mallocs_total counter
go_memstats_mallocs_total 124528
# HELP go_memstats_mcache_inuse_bytes Number of bytes in use by mcache     structures. Equals to /memory/classes/metadata/mcache/inuse:bytes.
# TYPE go_memstats_mcache_inuse_bytes gauge
go_memstats_mcache_inuse_bytes 4832
# HELP go_memstats_mcache_sys_bytes Number of bytes used for mcache structures obtained from system. Equals to
/memory/classes/metadata/mcache/inuse:bytes + /memory/classes/metadata/mcache/free:bytes.
# TYPE go_memstats_mcache_sys_bytes gauge
go_memstats_mcache_sys_bytes 15704
# HELP go_memstats_mspan_inuse_bytes Number of bytes in use by mspan     structures. Equals to /memory/classes/metadata/mspan/inuse:bytes.
# TYPE go_memstats_mspan_inuse_bytes gauge
go_memstats_mspan_inuse_bytes 125920
# HELP go_memstats_mspan_sys_bytes Number of bytes used for mspan structures obtained from system. Equals to
/memory/classes/metadata/mspan/inuse:bytes + /memory/classes/metadata/mspan/free:bytes.
# TYPE go_memstats_mspan_sys_bytes gauge
go_memstats_mspan_sys_bytes 146880
# HELP go_memstats_next_gc_bytes Number of heap bytes when next garbage     collection will take place. Equals to /gc/heap/goal:bytes.
# TYPE go_memstats_next_gc_bytes gauge
go_memstats_next_gc_bytes 6.04629e+06
# HELP go_memstats_other_sys_bytes Number of bytes used for other system  allocations. Equals to /memory/classes/other:bytes.
# TYPE go_memstats_other_sys_bytes gauge
go_memstats_other_sys_bytes 764832
# HELP go_memstats_stack_inuse_bytes Number of bytes obtained from system for stack allocator in non-CGO environments. Equals to
/memory/classes/heap/stacks:bytes.
# TYPE go_memstats_stack_inuse_bytes gauge
go_memstats_stack_inuse_bytes 1.081344e+06
# HELP go_memstats_stack_sys_bytes Number of bytes obtained from system for stack allocator. Equals to /memory/classes/heap/stacks:bytes +
/memory/classes/os-stacks:bytes.
# TYPE go_memstats_stack_sys_bytes gauge
go_memstats_stack_sys_bytes 1.081344e+06
# HELP go_memstats_sys_bytes Number of bytes obtained from system. Equals to /memory/classes/total:byte.
# TYPE go_memstats_sys_bytes gauge
go_memstats_sys_bytes 5.793896e+08
# HELP go_sched_gomaxprocs_threads The current runtime.GOMAXPROCS setting, or the number of operating system threads that can execute user-
level Go code simultaneously. Sourced from /sched/gomaxprocs:threads.
# TYPE go_sched_gomaxprocs_threads gauge
go_sched_gomaxprocs_threads 4
# HELP go_threads Number of OS threads created.
# TYPE go_threads gauge
go_threads 10
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.55
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP process_network_receive_bytes_total Number of bytes received by the     process over the network.
# TYPE process_network_receive_bytes_total counter
process_network_receive_bytes_total 1.172862e+06
# HELP process_network_transmit_bytes_total Number of bytes sent by the process  over the network.
# TYPE process_network_transmit_bytes_total counter
process_network_transmit_bytes_total 632432
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 9
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 3.9350272e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.74921402754e+09
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.901723648e+09
# HELP process_virtual_memory_max_bytes Maximum amount of virtual memory   available in bytes.
# TYPE process_virtual_memory_max_bytes gauge
process_virtual_memory_max_bytes 1.8446744073709552e+19
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 3
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
# HELP pxc_binlog_collector_failure_total Total number of failed binlog     collection cycles
# TYPE pxc_binlog_collector_failure_total counter
pxc_binlog_collector_failure_total 0
# HELP pxc_binlog_collector_gap_detected_total Total number of times the gap was detected in binlog
# TYPE pxc_binlog_collector_gap_detected_total counter
pxc_binlog_collector_gap_detected_total 0
# HELP pxc_binlog_collector_last_processing_timestamp Timestamp of the last     successful binlog processing
# TYPE pxc_binlog_collector_last_processing_timestamp gauge
pxc_binlog_collector_last_processing_timestamp 1.7492150471803956e+09
# HELP pxc_binlog_collector_last_upload_timestamp Timestamp of the last     successful binlog upload
# TYPE pxc_binlog_collector_last_upload_timestamp gauge
pxc_binlog_collector_last_upload_timestamp 1.749214031447092e+09
# HELP pxc_binlog_collector_success_total Total number of successful binlog     collection cycles
```

```
# TYPE pxc_binlog_collector_success_total counter
pxc_binlog_collector_success_total 19
# HELP pxc_binlog_collector_uploaded_total Total number of successfully     uploaded binlogs
# TYPE pxc_binlog_collector_uploaded_total counter
pxc_binlog_collector_uploaded_total 1
```

Note that the statistics data is not kept when the point-in-time recovery Pod restarts. This means that the counters like `pxc_binlog_collector_success_total` are reset.

Make a backup

Making scheduled backups

Backups schedule is defined in the `backup` section of the Custom Resource and can be configured via the deploy/cr.yaml file.

1. The `backup.storages` subsection should contain at least one configured storage.
2. The `backup.schedule` subsection allows to actually schedule backups:
   - set the `backup.schedule.name` key to some arbitrary backup name (this name will be needed later to restore the backup).
   - specify the `backup.schedule.schedule` option with the desired backup schedule in crontab format.
   - set the `backup.schedule.storageName` key to the name of your already configured storage.
   - you can optionally define the retention policy for backups: how many backups which should be kept in the storage.

Here is an example of the `deploy/cr.yaml` with a scheduled Saturday night backup kept on the Amazon S3 storage:

```
...
backup:
  storages:
    s3-us-west:
      type: s3
      s3:
        bucket: S3-BACKUP-BUCKET-NAME-HERE
        region: us-west-2
        credentialsSecret: my-cluster-name-backup-s3
  schedule:
   - name: "sat-night-backup"
     schedule: "0 0 * * 6"
     retention:
        count: 3
        type: count
        deleteFromStorage: true
     storageName: s3-us-west
  ...
```

Note

Before the Operator version 1.10 scheduled backups were based on Kubernetes CronJobs, while newer Operator versions take care about scheduled backups itself. Clusters upgraded from the Operator version 1.9 may need manual deletion of scheduled backups CronJobs, if any existed prior to the upgrade (otherwise backups will run twice).

You can check if there are any CronJobs in the namespace of your cluster related to scheduled backups as follows:

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

Expected output

```
NAME SCHEDULE SUSPEND ACTIVE LAST SCHEDULE AGE
sat-night-backup 0 0 * * 6 False 0 <none> 4m36s
```

Deleting CronJob is straightforward:

```
$ kubectl delete cronjob sat-night-backup -n <namespace>
```

Expected output

```
cronjob.batch "sat-night-backup" deleted
```

Make on-demand backup

Before you begin

1. Export the namespace as an environment variable. Replace the `<namespace>` placeholder with your value:

   ```
   export NAMESPACE=<namespace>
   ```

2. Check the configuration of the `PerconaXtraDBCluster` Custom Resource. Verify that you have configured backup storage and specified its configuration in the `backup.storages` subsection of the Custom Resource.

Backup steps

To make an on-demand backup, use *a special backup configuration YAML file*. The example of such file is deploy/backup/backup.yaml

1. Specify the following keys

   - Set the `metadata.name` key to assign a name to the backup. You will use it to make a restore
   - Set the `spec.pxcCluster` key to the name of your cluster
   - Set the `spec.storageName` key to a storage configuration defined in your `deploy/cr.yaml` file.
   - Optionally, add the `percona.com/delete-backup` entry under `metadata.finalizers` to enable deletion of backup files from a backup storage when the backup object is removed (manually or by schedule).

   Here's the example configuration:

   ```
   apiVersion: pxc.percona.com/v1
   kind: PerconaXtraDBClusterBackup
   metadata:
     finalizers:
       - percona.com/delete-backup
     name: backup1
   spec:
     pxcCluster: cluster1
     storageName: fs-pvc
   ```

2. Start the backup process:

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

3. Track the backup process by checking the status of the Backup object:

   ```
   $ kubectl get pxc-backup -n $NAMESPACE -w
   ```

   The `-w` flag instructs the Operator to provide real-time updates about the backup progress. The `Succeeded` status indicates that a backup is completed.

   Expected output

   ```
   NAME      CLUSTER    STORAGE   DESTINATION                              STATUS      COMPLETED   AGE
   backup1   cluster1   fs-pvc    pvc/xb-backup1-20251201102237-8f7b3390   Succeeded   3s          76s
   ```

4. View detailed information about the backup using the `kubectl describe` command:

   ```
   $ kubectl describe pxc-backup -n $NAMESPACE
   ```

   The `Status` section of the output provides useful details about the backup state, the error message in case of issues with the backup, and the storage details.

Sample output

```
Name:         backup1
Namespace:    <my-namespace>
Labels:       <none>
Annotations:  <none>
API Version:  pxc.percona.com/v1
Kind:         PerconaXtraDBClusterBackup
Metadata:
  Creation Timestamp:  2025-12-01T10:22:37Z
  Generation:          1
  Resource Version:    1764584633525183013
  UID:                 8f7b3390-fa7a-4c37-85f2-9037c093589f
Spec:
  Pxc Cluster:   cluster1
  Storage Name:  fs-pvc
Status:
  Completed:    2025-12-01T10:23:50Z
  Destination:  pvc/xb-backup1-20251201102237-8f7b3390
  Image:        perconalab/percona-xtradb-cluster-operator:main-pxc8.0-backup
  Pvc:
    Access Modes:
      ReadWriteOnce
    Resources:
      Requests:
        Storage:            6G
    Storage Class Name:     standard-rwo
    Volume Mode:            Filesystem
    Volume Name:            pvc-5238d6db-f40a-4608-8f8e-d0d74f328de9
```

Enable compression for backups

You can enable ZSTD compression for backups if you run Percona XtraDB Cluster 8.0.34 and higher.

To enable compression, use the pxc.configuration key in the `deploy/cr.yaml` configuration file. Specify the following options from the `my.cnf` configuration file in the `[sst]` and `[xtrabackup]` sections:

```
pxc:
  image: percona/percona-xtradb-cluster:8.4.0-5.1
  configuration: |
    ...
    [sst]
    xbstream-opts=--decompress
    [xtrabackup]
    compress=zstd
    ...
```

When enabled, compression will be used for both backups and SST.

Copy backup to a local machine

Make a local copy of a previously saved backup requires not more than the backup name. This name can be taken from the list of available backups returned by the following command:

```
$ kubectl get pxc-backup
```

When the name is known, backup can be downloaded to the local machine as follows:

```
$ ./deploy/backup/copy-backup.sh <backup-name> path/to/dir
```

For example, this downloaded backup can be restored to the local installation of Percona Server:

```
$ service mysqld stop
$ rm -rf /var/lib/mysql/*
$ cat xtrabackup.stream | xbstream -x -C /var/lib/mysql
$ xtrabackup --prepare --target-dir=/var/lib/mysql
$ chown -R mysql:mysql /var/lib/mysql
$ service mysqld start
```

If needed, you can also restore the backup to a Kubernetes cluster following the instructions in this howto.

Restore from a backup

Restore the cluster from a previously saved backup

You can restore from a backup as follows:

- On the same cluster where you made a backup
- On a new cluster deployed in a different Kubernetes-based environment.

This document focuses on the restore to the same cluster.

Restore scenarios

This document covers the following restore scenarios:

- Restore from a full backup - the restore from a backup without point-in-time
- Point-in-time recovery - restore to a specific time, a specific or latest transaction or skip a specific transaction during a restore. This ability requires that you configure storing binlogs for point-in-time recovery
- Restore when a backup has different passwords

To restore from a backup, you create a special Restore object using a special restore configuration file. The example of such file is deploy/backup/restore.yaml.

You can check available options in the restore options reference.

Note that you **cannot restore** to emptyDir and hostPath volumes, but you can make a backup from such storage (i. e., from emptyDir/hostPath to S3), and later restore it to a Persistent Volume.

Before you start

1. Make sure that the cluster is running.
2. List the cluster to find the correct cluster name. Replace the `<namespace>` with your value:

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

3. List backups to retrieve the desired backup name. Replace the `<namespace>` with your value:

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

4. For point-in-time recovery, disable storing binlogs point-in-time functionality on the existing cluster. You must do it regardless of whether you made the backup with point-in-time recovery or without it. Use the following command and replace the cluster name and the `<namespace>` with your values:

   ```
   $ kubectl patch pxc cluster1 \
     -n <namespace> \
     --type merge \
     -p '{"spec":{"backup":{"pitr":{"enabled":false}}}}'
   ```

Restore from a full backup

To restore your Percona XtraDB cluster from a backup, define a `PerconaXtraDBClusterRestore` custom resource. Set the following keys:

- `spec.pxcCluster`: the name of the target cluster
- `spec.backupName`: the name of your backup,

Pass this configuration to the Operator:

via the YAML manifest

1. Edit the deploy/backup/restore.yaml file and specify the following keys:

```
apiVersion: pxc.percona.com/v1
kind: PerconaXtraDBClusterRestore
metadata:
  name: restore1
spec:
  pxcCluster: cluster1
  backupName: backup1
```

2. Start the restore with this command:

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

via the command line

You can skip creating a separate file by passing YAML content directly:

```
$ cat <<EOF | kubectl apply -f-
apiVersion: "pxc.percona.com/v1"
kind: "PerconaXtraDBClusterRestore"
metadata:
  name: "restore1"
spec:
  pxcCluster: "cluster1"
  backupName: "backup1"
EOF
```

Restore with point-in-time recovery

1. Check a time to restore for a backup. Use the command below to find the latest restorable timestamp:

```
$ kubectl get pxc-backup <backup_name> -o jsonpath='{.status.latestRestorableTime}'
```

2. Set the following keys for the `PerconaXtraDBClusterRestore` custom resource:

- `spec.pxcCluster`: the name of the target cluster
- `spec.backupName`: the name of your backup
- for the `pitr` section:
- `type`: one of the following values:
  - `date` - roll back to specific date,
  - `transaction` - roll back to a specific transaction (available since Operator 1.8.0),
  - `latest` - recover most recent transaction,
  - `skip` - skip a specific transaction (available since Operator 1.7.0).
- `date`: is used with `type=date` option and contains the value in the datetime format,
- `gtid`: is used with `type=transaction` option and contains exact GTID of a transaction **which follows** the last transaction included into the recovery (available since the Operator 1.8.0)
- (optional) `storageName`: the exact name of the storage. Note that you must have already defined the storage in the `backup.storages` subsection of the `deploy/cr.yaml` file.

3. Pass this configuration to the Operator:

via the YAML manifest

a. Edit the deploy/backup/restore.yaml file.

The sample configuration may look as follows:

```
apiVersion: pxc.percona.com/v1
kind: PerconaXtraDBClusterRestore
metadata:
  name: restore1
spec:
  pxcCluster: cluster1
  backupName: backup1
  pitr:
    type: date
    date: "2020-12-31 09:37:13"
    backupSource:
      storageName: s3-us-west
```

b. Start the restore:

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

via the command line

You can skip editing the YAML file and pass its contents to the Operator via the command line. For example:

```
$ cat <<EOF | kubectl apply -f-
apiVersion: "pxc.percona.com/v1"
kind: "PerconaXtraDBClusterRestore"
metadata:
  name: "restore1"
spec:
  pxcCluster: "cluster1"
  backupName: "backup1"
  pitr:
    type: date
    date: "2020-12-31 09:37:13"
    backupSource:
      storageName: "s3-us-west"
EOF
```

4. Make a new full backup after the restore, because your restored database is now the new baseline for future recoveries

Binlog gaps

The Operator monitors the binlog gaps detected by binlog collector, if any. If a backup contains such gaps, the Operator will mark the status of the latest successful backup with a new condition field that indicates backup can't guarantee consistent point-in-time recovery. This condition looks as follows:

```
apiVersion: pxc.percona.com/v1
kind: PerconaXtraDBClusterBackup
metadata:
  name: backup1
spec:
  pxcCluster: pitr
  storageName: minio
status:
  completed: "2022-11-25T15:57:29Z"
  conditions:
  - lastTransitionTime: "2022-11-25T15:57:48Z"
    message: Binlog with GTID set e41eb219-6cd8-11ed-94c8-9ebf697d3d20:21-22 not found
    reason: BinlogGapDetected
    status: "False"
    type: PITRReady
  state: Succeeded
```

Trying a point-in-time restore from such backup (with the condition value "False") results in the following error:

```
Backup doesn't guarantee consistent recovery with PITR. Annotate PerconaXtraDBClusterRestore with percona.com/unsafe-pitr to
force it.
```

You can bypass this check and force the restore by annotating it with `pxc.percona.com/unsafe-pitr` as follows:

```
apiVersion: pxc.percona.com/v1
kind: PerconaXtraDBClusterRestore
metadata:
  annotations:
    percona.com/unsafe-pitr: "true"
  name: restore2
spec:
  pxcCluster: pitr
  backupName: backup1
  pitr:
    type: latest
    backupSource:
      storageName: "minio-binlogs"
```

Restore the cluster when backup has different passwords

User passwords on the target cluster may have changed and now differ from the ones in a backup.

Starting with version 1.18.0, the Operator no longer requires matching secrets between the backup and the target cluster. After the restore, it changes user passwords using the local Secret as a source. It also creates missing system users and adds missing grants. So you can restore from a full backup or run a point-in-time restore as usual.

For the Operator versions 1.17.0 and earlier, read on.

If the cluster is restored to a backup which has different user passwords, the Operator will be unable connect to database using the passwords in Secrets, and so will fail to reconcile the cluster.

Let's consider an example with four backups, first two of which were done before the password rotation and therefore have different passwords:

```
NAME      CLUSTER    STORAGE   DESTINATION      STATUS      COMPLETED   AGE
backup1   cluster1   fs-pvc    pvc/xb-backup1   Succeeded   23m         24m
backup2   cluster1   fs-pvc    pvc/xb-backup2   Succeeded   18m         19m
backup3   cluster1   fs-pvc    pvc/xb-backup3   Succeeded   13m         14m
backup3   cluster1   fs-pvc    pvc/xb-backup4   Succeeded   8m53s       9m29s
backup4   cluster1   fs-pvc    pvc/xb-backup5   Succeeded   3m11s       4m29s
```

In this case you will need some manual operations same as the Operator does to propagate password changes in Secrets to the database **before restoring a backup**.

When the user updates a password in the Secret, the Operator creates a temporary Secret called `<clusterName>-mysql-init` and puts (or appends) the required `ALTER USER` statement into it. Then MySQL Pods are mounting this init Secret if exist and running corresponding statements on startup. When a new backup is created and successfully finished, the Operator deletes the init Secret.

In the above example passwords are changed after backup2 was finished, and then three new backups were created, so the init Secret does not exist. If you want to restore to backup2, you need to create the init secret by your own with the latest passwords as follows.

1. Make a base64-encoded string with needed SQL statements (substitute each `<latestPass>` with the password of the appropriate user):

in Linux

```
$ cat <<EOF | base64 --wrap=0
ALTER USER 'root'@'%' IDENTIFIED BY '<latestPass>';
ALTER USER 'root'@'localhost' IDENTIFIED BY '<latestPass>';
ALTER USER 'operator'@'%' IDENTIFIED BY '<latestPass>';
ALTER USER 'monitor'@'%' IDENTIFIED BY '<latestPass>';
ALTER USER 'clustercheck'@'localhost' IDENTIFIED BY '<latestPass>';
ALTER USER 'xtrabackup'@'%' IDENTIFIED BY '<latestPass>';
ALTER USER 'xtrabackup'@'localhost' IDENTIFIED BY '<latestPass>';
ALTER USER 'replication'@'%' IDENTIFIED BY '<latestPass>';
EOF
```

in macOS

```
$ cat <<EOF | base64
ALTER USER 'root'@'%' IDENTIFIED BY '<latestPass>';
ALTER USER 'root'@'localhost' IDENTIFIED BY '<latestPass>';
ALTER USER 'operator'@'%' IDENTIFIED BY '<latestPass>';
ALTER USER 'monitor'@'%' IDENTIFIED BY '<latestPass>';
ALTER USER 'clustercheck'@'localhost' IDENTIFIED BY '<latestPass>';
ALTER USER 'xtrabackup'@'%' IDENTIFIED BY '<latestPass>';
ALTER USER 'xtrabackup'@'localhost' IDENTIFIED BY '<latestPass>';
ALTER USER 'replication'@'%' IDENTIFIED BY '<latestPass>';
EOF
```

2. After you obtained the needed base64-encoded string, create the appropriate Secret:

```
$ kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: cluster1-mysql-init
data:
  init.sql: <base64encodedstring>
EOF
```

3. Now you can restore the needed backup as usual.

How to restore backup to a new Kubernetes-based environment

You can restore from a backup as follows:

- On the same cluster where you made a backup
- On a new cluster deployed in a different Kubernetes-based environment.

This document focuses on the restore on a new cluster deployed in a different Kubernetes environment.

For Operator version 1.17.0 and earlier

When restoring to a new Kubernetes-based environment, make sure it has a Secrets object with the same **user passwords** as in the original cluster. More details about secrets can be found in System Users. The name of the required Secrets object can be found out from the `spec.secretsName` key in the `deploy/cr.yaml` (`cluster1-secrets` by default).

Restore scenarios

This document covers the following restore scenarios:

- Restore from a full backup - the restore from a backup without point-in-time
- Point-in-time recovery - restore to a specific time, a specific or latest transaction or skip a specific transaction during a restore. This ability requires that you configure storing binlogs for point-in-time recovery

To restore from a backup, you create a special Restore object using a special restore configuration file. The example of such file is deploy/backup/restore.yaml.

You can check available options in the restore options reference.

Before you start

1. Make sure that the cluster is running.
2. List the cluster to find the correct cluster name. Replace the `<namespace>` with your value:

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

3. List backups to retrieve the desired backup name. Replace the `<namespace>` with your value:

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

4. For point-in-time recovery, disable storing binlogs point-in-time functionality on the existing cluster. You must do it regardless of whether you made the backup with point-in-time recovery or without it. Use the following command and replace the cluster name and the `<namespace>` with your values:

   ```
   $ kubectl patch pxc cluster1 \
     -n <namespace> \
     --type merge \
     -p '{"spec":{"backup":{"pitr":{"enabled":false}}}}'
   ```

Restore from a full backup

Configure the `PerconaXtraDBClusterRestore` Custom Resource. Specify the following keys in the deploy/backup/restore.yaml file:

- set `spec.pxcCluster` key to the name of the target cluster to restore the backup on,
- configure the `spec.backupSource` subsection to point to the PVC or the cloud storage where the backup is stored.

PVC volume

The `spec.backupSource` subsection should include:

- `storageName` - the storage name, which should be configured in the main CR
- `destination` key should be equal to the PVC Name:

```
...
backupSource:
  destination: pvc/PVC_VOLUME_NAME
  storageName: pvc
  ...
```

Note

If you need a headless Service for the restore Pod (i.e. restoring from a Persistent Volume in a tenant network), mention this in the `metadata.annotations` as follows:

```
annotations:
  percona.com/headless-service: "true"
...
```

S3-compatible storage

The `spec.backupSource` subsection should include:

- a destination key. Take it from the output of the `kubectl get pxc-backup` command. The destination consists of the `s3://` prefix, the S3 bucket name and the backup name.
- the necessary storage configuration keys, just like in the `deploy/cr.yaml` file of the source cluster.
- `verifyTLS` to verify the storage server TLS certificate
- the custom TLS configuration if you use it for backups. Refer to the Configure TLS verification with custom certificates for S3 storage section for more information.

```
...
backupSource:
  verifyTLS: true
  destination: s3://S3-BUCKET-NAME/BACKUP-NAME
  s3:
    bucket: S3-BUCKET-NAME
    credentialsSecret: my-cluster-name-backup-s3
    region: us-west-2
    endpointUrl: https://URL-OF-THE-S3-COMPATIBLE-STORAGE
    caBundle: #If you use custom TLS certificates for S3 storage
      name: minio-ca-bundle
      key: ca.crt
```

Azure Blob storage

The `destination` key should have value composed of three parts: the `azure://` prefix, the Azure Blob container, and the backup name, which you have already found out using the `kubectl get pxc-backup` command. Also you should add necessary Azure configuration keys, same as those used to configure Azure Blob storage for backups in the `deploy/cr.yaml` file:

```
...
backupSource:
  destination: azure://AZURE-CONTAINER-NAME/BACKUP-NAME
  azure:
    container: AZURE-CONTAINER-NAME
    credentialsSecret: my-cluster-azure-secret
    ...
```

- After that, the actual restoration process can be started as follows:

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

Restore the cluster with point-in-time recovery

> **Note**
>
> Disable the point-in-time functionality on the existing cluster before restoring a backup on it, regardless of whether the backup was made with point-in-time recovery or without it.

1. Set appropriate keys in the deploy/backup/restore.yaml file.

   - set `spec.pxcCluster` key to the name of the target cluster to restore the backup on,
   - put additional restoration parameters to the `pitr` section:
     - `type` key can be equal to one of the following options,
       - `date` - roll back to specific date,
       - `transaction` - roll back to a specific transaction (available since Operator 1.8.0),
       - `latest` - recover to the latest possible transaction,
       - `skip` - skip a specific transaction (available since Operator 1.7.0).
     - `date` key is used with `type=date` option and contains value in datetime format,
     - `gtid` key (available since the Operator 1.8.0) is used with `type=transaction` option and contains exact GTID of a transaction **which follows** the last transaction included into the recovery,
   - set `spec.backupSource` subsection to point on the appropriate S3-compatible storage. This subsection should contain a `destination` key equal to the s3 bucket with a special `s3://` prefix, followed by necessary S3 configuration keys, same as in `deploy/cr.yaml` file.

   The resulting `restore.yaml` file may look as follows:

   ```
   apiVersion: pxc.percona.com/v1
   kind: PerconaXtraDBClusterRestore
   metadata:
     name: restore1
   spec:
     pxcCluster: cluster1
     backupName: backup1
     pitr:
       type: date
       date: "2020-12-31 09:37:13"
       backupSource:
             destination: s3://S3-BUCKET-NAME/BACKUP-NAME
             s3:
               bucket: S3-BUCKET-NAME
               credentialsSecret: my-cluster-name-backup-s3
               region: us-west-2
               endpointUrl: https://URL-OF-THE-S3-COMPATIBLE-STORAGE
   ```

   - you can also use a `storageName` key to specify the exact name of the storage (the actual storage should be already defined in the `backup.storages` subsection of the `deploy/cr.yaml` file):

     ```
     ...
     storageName: s3-us-west
     backupSource:
       destination: s3://S3-BUCKET-NAME/BACKUP-NAME
     ```

2. Run the actual restoration process:

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

3. Make a new full backup after the restore, because your restored database is now the new baseline for future recoveries.

Delete the unneeded backup

The maximum amount of stored backups is controlled by the backup.schedule.keep option (only successful backups are counted). Older backups are automatically deleted, so that amount of stored backups do not exceed this number. Setting `keep=0` or removing this option from `deploy/cr.yaml` disables automatic deletion of backups.

Manual deleting of a previously saved backup requires not more than the backup name. This name can be taken from the list of available backups returned by the following command:

```
$ kubectl get pxc-backup
```

When the name is known, backup can be deleted as follows:

```
$ kubectl delete pxc-backup/<backup-name>
```

Scale MySQL on Kubernetes and OpenShift

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 MySQL nodes; horizontal scaling is about adding more nodes to the cluster.

Vertical scaling

Scale compute resources

The Operator deploys and manages multiple components, such as Percona XtraDB Cluster (PXC), HAProxy or ProxySQL, etc. You can manage CPU or memory for every component separately by editing corresponding sections in the Custom Resource. We follow the structure for `requests` and `limits` that Kubernetes provides.

To add more resources to your MySQL nodes in PXC edit the following section in the Custom Resource:

```
spec:
...
  pxc:
    ...
    resources:
      requests:
        memory: 4G
        cpu: 2
      limits:
        memory: 4G
        cpu: 2
```

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

Scale storage

Kubernetes manages storage with a PersistentVolume (PV), a segment of storage supplied by the administrator, and a PersistentVolumeClaim (PVC), a request for storage from a user. Starting with Kubernetes v1.11, a user can increase the size of an existing PVC object (considered stable since Kubernetes v1.24). The user cannot shrink the size of an existing PVC object.

Starting from the version 1.14.0, you can scale Percona XtraDB Cluster storage automatically by configuring the Custom Resource manifest. Alternatively, you can scale the storage manually. For either way, the volume type must support PVCs expansion.

Find exact details about PVCs and the supported volume types in Kubernetes documentation.

Storage resizing with Volume Expansion capability

Certain volume types support PVCs expansion. You can run the following command to check if your storage supports the expansion capability:

```
$ kubectl describe sc <storage class name> | grep AllowVolumeExpansion
```

Expected output

```
AllowVolumeExpansion: true
```

To enable storage resizing via volume expansion, set the enableVolumeExpansion Custom Resource option to `true` (it is turned off by default). When enabled, the Operator will automatically expand such storage for you when you change the `pxc.volumeSpec.persistentVolumeClaim.resources.requests.storage` option in the Custom Resource.

For example, you can do it by editing and applying the `deploy/cr.yaml` file:

```
spec:
  ...
  enableVolumeExpansion: true
    ...
  pxc:
    ...
    volumeSpec:
      ...
      persistentVolumeClaim:
        resources:
          requests:
            storage: <NEW STORAGE SIZE>
```

Apply changes as usual:

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

The storage size change takes some time. When it starts, the Operator automatically adds the `pvc-resize-in-progress` annotation to the `PerconaXtraDBCluster` Custom Resource. The annotation contains the timestamp of the resize start and indicates that the resize operation is running.. After the resize finishes, the Operator deletes this annotation.

Warning

If the new storage size can't be reached because there is a resource quota in place and the PVC storage limits are reached, this will be detected, there will be no scaling attempts, and the Operator will revert the value in the Custom Resource option back. If resize isn't successful (for example, no quota is set, but the new storage size turns out to be just too large), the Operator will detect Kubernetes failure on scaling, and revert the Custom Resource option. Still, Kubernetes will continue attempts to fulfill the scaling request until the problem is fixed manually by the Kubernetes administrator.

Manual resizing without Volume Expansion capability

Manual resizing is the way to go if:

- your version of the Operator is older than 1.14.0,
- your volumes have a type that does not support Volume Expansion, or
- you do not rely on automated scaling.

You will need to delete Pods and their persistent volumes one by one to resync the data to the new volumes. **This way you can also shrink the storage.**

Here's how to resize the storage:

1. Update the Custom Resource with the new storage size by editing and applying the `deploy/cr.yaml` file:

   ```
   spec:
   ...
     pxc:
       ...
       volumeSpec:
         persistentVolumeClaim:
           resources:
             requests:
               storage: <NEW STORAGE SIZE>
   ```

2. Apply the Custom Resource update for the changes to come into effect:

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

3. Delete the StatefulSet with the `orphan` option

   ```
   $ kubectl delete sts <statefulset-name> --cascade=orphan
   ```

   The Pods will not go down and the Operator is going to recreate the StatefulSet:

   ```
   $ kubectl get sts <statefulset-name>
   ```

Expected output

```
cluster1-pxc      3/3     39s
```

4. Scale up the cluster (Optional)

   Changing the storage size would require us to terminate the Pods, which decreases the computational power of the cluster and might cause performance issues. To improve performance during the operation we are going to change the size of the cluster from 3 to 5 nodes:

   ```
   ...
   spec:
   ...
     pxc:
       ...
       size: 5
   ```

   Apply the change:

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

   New Pods will already have the new storage:

   ```
   $ kubectl get pvc
   ```

   Expected output

   ```
   NAME                     STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
   datadir-cluster1-pxc-0   Bound    pvc-90f0633b-0938-4b66-a695-556bb8a9e943   10Gi       RWO            standard       110m
   datadir-cluster1-pxc-1   Bound    pvc-7409ea83-15b6-448f-a6a0-12a139e2f5cc   10Gi       RWO            standard       109m
   datadir-cluster1-pxc-2   Bound    pvc-90f0b2f8-9bba-4262-904c-1740fdd5511b   10Gi       RWO            standard       108m
   datadir-cluster1-pxc-3   Bound    pvc-439bee13-3b57-4582-b342-98281aca50ba   19Gi       RWO            standard       49m
   datadir-cluster1-pxc-4   Bound    pvc-2d4f3a60-4ec4-48a0-96cd-5243e2f05234   19Gi       RWO            standard       47m
   ```

5. Delete PVCs and Pods with the old storage size one by one. Wait for data to sync before you proceed to the next node.

   ```
   $ kubectl delete pvc <PVC NAME>
   $ kubectl delete pod <POD NAME>
   ```

   The new PVC is going to be created along with the Pod.

The storage size change takes some time. When it starts, the Operator automatically adds the `pvc-resize-in-progress` annotation to the `PerconaXtraDBCluster` Custom Resource. The annotation contains the timestamp of the resize start and indicates that the resize operation is running.. After the resize finishes, the Operator deletes this annotation.

Horizontal scaling

Size of the cluster is controlled by a size key in the Custom Resource options configuration. That's why scaling the cluster needs nothing more but changing this option and applying the updated configuration file. This may be done in a specifically saved config:

```
spec:
...
  pxc:
    ...
    size: 5
```

Apply the change:

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

Alternatively, you cana do it on the fly, using the following command:

```
$ kubectl scale --replicas=5 pxc/<CLUSTER NAME>
```

In this example we have changed the size of the Percona XtraDB Cluster to `5` instances.

Automated scaling

To automate horizontal scaling it is possible to use Horizontal Pod Autoscaler (HPA). It will scale the Custom Resource itself, letting Operator to deal with everything else.

It is also possible to use Kubernetes Event-driven Autoscaling (KEDA), where you can apply more sophisticated logic for decision making on scaling.

For now it is not possible to use Vertical Pod Autoscaler (VPA) with the Operator due to the limitations it introduces for objects with owner references.

Monitor database with Percona Monitoring and Management (PMM)

The Operator integrates natively with Percona Monitoring and Management (PMM) for comprehensive database monitoring. While custom monitoring solutions solutions are also supported, they require manual setup and are not automated by the Operator.

The Operator is compatible with both PMM versions 2 and 3. We recommend using the latest PMM version 3 for optimal monitoring capabilities.

In this section, you'll learn how to monitor Percona XtraDB Cluster using PMM.

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.

Considerations

1. If you are using PMM server version 2, use a PMM client image compatible with PMM 2. If you are using PMM server version 3, use a PMM client image compatible with PMM 3. Check Percona certified images for the right one.
2. If you specified both authentication methods for PMM server configuration and they have non-empty values, priority goes to PMM 3.
3. For migration from PMM2 to PMM3, see PMM upgrade documentation. Also check the Automatic migration of API keys page.

Install PMM Server

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

Install PMM Client

PMM Client is installed as a side-car container in the database, HAProxy and ProxySQL Pods in your Kubernetes-based environment. To install PMM Client, do the following:

Configure authentication

PMM3

PMM3 uses Grafana service accounts to control access to PMM server components and resources. To authenticate in PMM server, you need a service account token. Generate a service account and token. Specify the Admin role for the service account.

Warning

When you create a service account token, you can select its lifetime: it can be either a permanent token that never expires or the one with the expiration date. PMM server cannot rotate service account tokens after they expire. So you must take care of reconfiguring PMM Client in this case.

PMM2

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.

From PMM UI

Generate the PMM API key

From command line

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)
```

Warning

The API key is not rotated.

Create a secret

Now you must pass the credentials to the Operator. To do so, create a Secret object.

Warning

PMM3 Client only works with PMM Server tokens. Make sure to use the `pmmservertoken` option for PMM3 Client. Otherwise you may face issues with the setup.

1. Create a Secret configuration file. You can use the deploy/secrets.yaml secrets file.

PMM 3

Specify the service account token as the `pmmservertoken` value in the secrets file:

```
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-secrets
type: Opaque
stringData:
  pmmservertoken: ""
```

PMM 2

Specify the API key as the `pmmserverkey` value in the secrets file:

```
apiVersion: v1
kind: Secret
metadata:
  name: cluster1-secrets
type: Opaque
stringData:
  pmmserverkey: ""
```

2. Create the Secrets object using the `deploy/secrets.yaml` file. Replace the `<namespace>` placeholder with your value.

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

Expected output

```
secret/cluster1-secrets created
```

Deploy a PMM Client

1. Update the `pmm` section in the deploy/cr.yaml file.
   - Set `pmm.enabled` = `true`.
   - Specify the PMM Client image path. Check Percona certified images for the required one.
   - 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:2.44.1-1
    serverHost: monitoring-service
```

2. Update the cluster. Replace the `<namespace>` placeholder with your value.

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

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

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

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 Object 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 service account token in the `cluster1-secrets` object, use the following command replacing the placeholders in `<>` with your values:

Linux

```
$ kubectl patch secret/cluster1-secrets -p '{"data":{"pmmservertoken": '$(echo -n <new-token> | base64 --wrap=0)'}}'
```

macOS

```
$ kubectl patch secret/cluster1-secrets -p '{"data":{"pmmservertoken": '$(echo -n <new-token> | base64)'}}'
```

Check the metrics

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

Now you can access PMM via *https* in a web browser, with the login/password authentication, and the browser is configured to show Percona XtraDB Cluster metrics.

Specify additional PMM parameters

You can specify additional parameters for pmm-admin add mysql and pmm-admin add proxysql commands, if needed. Use the `pmm.pxcParams` and `pmm.proxysqlParams` Custom Resource options for that.

The Operator automatically manages common Percona XtraDB Cluster Service Monitoring parameters mentioned in the official PMM documentation, such as username, password, service-name, host, etc. Assigning values to these parameters is not recommended and can negatively affect the functionality of the PMM setup carried out by the Operator.

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:

on Linux

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

on macOS

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

For example, to set the new PMM API key to `new_key` in the `cluster1-secrets` object, do the following:

in Linux

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

on macOS

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

Check PMM Client health and status

A probe is a diagnostic mechanism in Kubernetes which helps determine whether a container is functioning correctly and whether it should continue to run, accept traffic, or be restarted.

PMM Client has the following probes:

- **Readiness probe** determines when a PMM Client is available and ready to accept traffic
- **Liveness probe** determines when to restart a PMM Client

To configure probes, use the `spec.pmm.readinessProbes` and `spec.pmm.livenessProbes` Custom Resource options.

Add custom PMM prefix to the cluster name

When user has several clusters with the same namespace, cluster and Pod names, and a single PMM Server, it is possible to add only one of them to the PMM Server instance because of this names coincidence.

For such cases it is possible to specify a custom prefix to the cluster name, which will be visible within PMM, and so names will become unique.

You can do it by setting the `PMM_PREFIX` environment variable via the Secret, specified in the `pxc.envVarsSecret` Custom Resource option.

Here is an example of the YAML file used to create the Secret with the `my-unique-prefix-` prefix encoded in base64 format:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-env-var-secrets
type: Opaque
data:
  PMM_PREFIX: bXktdW5pcXVlLXByZWZpeC0=
```

Follow the instruction on all details needed to create a Secret for environment variables and adding them to the Custom Resource.

Implement custom monitoring solution without PMM

You can deploy your own monitoring solution instead of PMM, but since the Operator will know nothing about it, it will not gain the same level of deployment automation from the Operator side, and there will be no configuration via the Custom Resource. The approach to this is to deploy your monitoring agent as a sidecar container in Percona XtraDB Cluster Pods. See sidecar containers documentation for details.

Note

You can use the `monitor` system user for monitoring purposes as PMM Client does. The Operator tracks the `monitor` user password update in the Secrets object (technical secrets used by the Operator, and restarts Percona XtraDB Cluster Pods in cases when PMM is enabled or when the sidecar container references the internal Secrets object `internal-<clustername>` (technical users secrets used by Operator, `internal-cluster1` by default) as follows:

```
pxc:
sidecars:
- name: metrics
  image: my_repo/my_custom_monitoring_solution:1.0
  env:
    - name: MYSQLD_EXPORTER_PASSWORD
      valueFrom:
        secretKeyRef:
          name: internal-cluster1
          key: monitor
  ...
```

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.

Adding a sidecar container

You can add sidecar containers to Percona XtraDB Cluster, HAProxy, and ProxySQL Pods. Just use `sidecars` subsection ing the `pxc`, `haproxy`, or `proxysql` section of the `deploy/cr.yaml` configuration file. In this subsection, you should specify the name and image of your container and possibly a command to run:

```
spec:
  pxc:
    ....
    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
```

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

```
$ kubectl describe pod cluster1-pxc-0
```

Expected output

```
....
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)
....
```

Getting shell access to a sidecar container

You can login to your sidecar container as follows:

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

Mount volumes into sidecar containers

It is possible to mount volumes into sidecar containers.

Following subsections describe different volume types, which were tested with sidecar containers and are known to work.

Persistent Volume

You can use Persistent volumes when you need dynamically provisioned storage which doesn't depend on the Pod lifecycle.

To use such volume, you should *claim* durable storage with persistentVolumeClaim without specifying any non-important details.

> **Important**
>
> You can use PVCs with sidecar containers only when you deploy a new cluster. Updates to running cluster are not supported.

The following example requests 1G storage with `sidecar-volume-claim` PersistentVolumeClaim, and mounts the correspondent Persistent Volume to the `my-sidecar-1` container's filesystem under the `/volume1` directory:

```
...
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
    volumeMounts:
    - mountPath: /volume1
      name: sidecar-volume-claim
  sidecarPVCs:
  - apiVersion: v1
    kind: PersistentVolumeClaim
    metadata:
      name: sidecar-volume-claim
    spec:
      resources:
        requests:
          storage: 1Gi
      volumeMode: Filesystem
      accessModes:
        - ReadWriteOnce
```

Secret

You can use a secret volume to pass the information which needs additional protection (e.g. passwords), to the container. Secrets are stored with the Kubernetes API and mounted to the container as RAM-stored files.

You can mount a secret volume as follows:

```
...
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
    volumeMounts:
    - mountPath: /secret
      name: sidecar-secret
  sidecarVolumes:
  - name: sidecar-secret
    secret:
      secretName: mysecret
```

The above example creates a `sidecar-secret` volume (based on already existing `mysecret` Secret object) and mounts it to the `my-sidecar-1` container's filesystem under the `/secret` directory.

> **Note**
>
> Don't forget you need to create a Secret Object before you can use it.

configMap

You can use a configMap volume to pass some configuration data to the container. Secrets are stored with the Kubernetes API and mounted to the container as RAM-stored files.

You can mount a configMap volume as follows:

```
...
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
    volumeMounts:
    - mountPath: /config
      name: sidecar-config
  sidecarVolumes:
  - name: sidecar-config
    configMap:
      name: myconfigmap
```

The above example creates a `sidecar-config` volume (based on already existing `myconfigmap` configMap object) and mounts it to the `my-sidecar-1` container's filesystem under the `/config` directory.

Note

Don't forget you need to create a configMap Object before you can use it.

Add external PersistentVolumeClaims to the Operator

One of the first things to think about when you run a database cluster on Kubernetes is data persistence. You want to make sure that if a Pod restarts, your data doesn't vanish with it. That's where Persistent Volumes (PVs) and Persistent Volume Claims (PVCs) come in.

A PV is a chunk of storage in your Kubernetes cluster, provisioned either by a cluster admin or dynamically via a storage class. A PVC is how your Pod asks for that storage. When you deploy a database cluster using an Operator, each pod automatically gets its own PVC. That PVC binds to a Persistent Volume that matches the pod's needs such as size, access mode, and storage class. This ensures that your database has durable, pod-specific storage.

Sometimes your database needs access to data it doesn't own. An example of such data is externally generated lookup tables, reference files, or large binary objects. These aren't part of the database's internal storage, but they may be essential to its operation.

To ensure the smooth operation of your client applications and the database itself, you can add external PVCs that store such data to the Operator. This gives you a clean separation between the cluster's internal storage and shared or external data. You can update or replace the shared data independently of the cluster lifecycle, while still staying fully compatible with the Operator's management flow.

You can add external PVCs either when creating the cluster or at runtime. You can mount them into PXC pods, ProxySQL, or HAProxy, depending on where your database needs them. Since the Operator doesn't create nor manage these PVCs, they must already exist in the same namespace as your cluster before you deploy or update it.

To add an external PVC, edit your `deploy/cr.yaml` file and include the following under the `pxc.extraPVCs`, `proxysql.extraPVCs`, or `haproxy.extraPVCs` sections:

```
pxc:
  extraPVCs:
    - name: shared-data
      claimName: my-existing-pvc
      mountPath: /mnt/shared-data
      readOnly: false
```

After you apply the changes, the Operator will mount the existing PVC `my-existing-pvc` into your PXC pod at `/mnt/shared-data`. You're in control of the data, and the database gets exactly what it needs.

Pause/resume Percona XtraDB Cluster

There may be external situations when it is needed to shutdown the Percona XtraDB 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
```

Pausing the cluster may take some time, and when the process is over, you will see only the Operator Pod running:

```
$ kubectl get pods
NAME                                               READY   STATUS    RESTARTS   AGE
percona-xtradb-cluster-operator-79966668bd-rswbk   1/1     Running   0          12m
```

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

Starting the cluster will take time. The process is over when all Pods have reached their Running status:

```
NAME                                               READY   STATUS    RESTARTS   AGE
cluster1-haproxy-0                                 2/2     Running   0          6m17s
cluster1-haproxy-1                                 2/2     Running   0          4m59s
cluster1-haproxy-2                                 2/2     Running   0          4m36s
cluster1-pxc-0                                     3/3     Running   0          6m17s
cluster1-pxc-1                                     3/3     Running   0          5m3s
cluster1-pxc-2                                     3/3     Running   0          3m56s
percona-xtradb-cluster-operator-79966668bd-rswbk   1/1     Running   0          9m54s
```

Crash Recovery

What does the full cluster crash mean?

A full cluster crash is a situation when all database instances where shut down in random order. Being rebooted after such situation, Pod is continuously restarting, and generates the following errors in the log:

```
It may not be safe to bootstrap the cluster from this node. It was not the last one to leave the cluster and may not contain all
the updates.
To force cluster bootstrap with this node, edit the grastate.dat file manually and set safe_to_bootstrap to 1
```

Note

To avoid this, shutdown your cluster correctly as it is written in Pause/resume Percona XtraDB Cluster.

The Percona Operator for MySQL based on Percona XtraDB Cluster provides two ways of recovery after a full cluster crash.

The Operator is providing automatic crash recovery (by default) and semi-automatic recovery starting from the version `1.7`. For the previous Operator versions, crash recovery can be done manually.

Automatic Crash Recovery

Crash recovery can be done automatically. This behavior is controlled by the `pxc.autoRecovery` option in the `deploy/cr.yaml` configuration file.

The default value for this option is `true`, which means that automatic recovery is turned on.

If this option is set to `false`, automatic crash recovery is not done, but semi-automatic recovery is still possible.

In this case you need to get the log from pxc container from all Pods using the following command:

```
$ for i in $(seq 0 $(($(kubectl get pxc cluster1 -o jsonpath='{.spec.pxc.size}')-1))); do echo "###############cluster1-pxc-
$i###############"; kubectl logs cluster1-pxc-$i -c pxc | grep '(seqno):' ; done
```

The output of this command should be similar to the following one:

```
###############cluster1-pxc-0##############
It is cluster1-pxc-0.cluster1-pxc.default.svc.cluster.local node with sequence number (seqno): 18
###############cluster1-pxc-1##############
It is cluster1-pxc-1.cluster1-pxc.default.svc.cluster.local node with sequence number (seqno): 18
###############cluster1-pxc-2##############
It is cluster1-pxc-2.cluster1-pxc.default.svc.cluster.local node with sequence number (seqno): 19
```

Now find the Pod with the largest `seqno` (it is `cluster1-pxc-2` in the above example).

Now execute the following commands to start this instance:

```
$ kubectl exec cluster1-pxc-2 -c pxc -- sh -c 'kill -s USR1 1'
```

Manual Crash Recovery

Warning

This method includes a lot of operations, and therefore, it is intended for advanced users only!

This method involves the following steps:

- swap the original Percona XtraDB Cluster image with the debug image, which does not reboot after the crash, and force all Pods to run it,
- find the Pod with the most recent Percona XtraDB Cluster data, run recovery on it, start `mysqld`, and allow the cluster to be restarted,
- revert all temporary substitutions.

Let's assume that a full crash did occur for the cluster named `cluster1`, which is based on three Percona XtraDB Cluster Pods.

Note

The following commands are written for Percona XtraDB Cluster 8.0. The same steps are also for Percona XtraDB Cluster 5.7 unless specifically indicated otherwise.

1. Check the current Update Strategy with the following command to make sure Smart Updates are turned off during the recovery:

```
$ kubectl get pxc cluster1 -o jsonpath='{.spec.updateStrategy}'
```

If the returned value is `SmartUpdate`, please change it to `onDelete` with the following command:

```
$ kubectl patch pxc cluster1 --type=merge --patch '{"spec": {"updateStrategy": "OnDelete" }}'
```

2. Change the normal PXC image inside the cluster object to the debug image:

Note

Please make sure the Percona XtraDB Cluster version for the debug image matches the version currently in use in the cluster. You can run the following command to find out which Percona XtraDB Cluster image is in use:

```
$ kubectl get pxc cluster1 -o jsonpath='{.spec.pxc.image}'
```

```
$ kubectl patch pxc cluster1 --type="merge" -p '{"spec":{"pxc":{"image":"percona/percona-xtradb-cluster:8.0.44-35.1-debug"}}}'
```

Note

For Percona XtraDB Cluster 5.7 this command should be as follows:

```
$ kubectl patch pxc cluster1 --type="merge" -p '{"spec":{"pxc":{"image":"percona/percona-xtradb-cluster:5.7.44-31.65-debug"}}}'
```

1. Restart all Pods:

```
$ for i in $(seq 0 $(($(kubectl get pxc cluster1 -o jsonpath='{.spec.pxc.size}')-1))); do kubectl delete pod cluster1-pxc-$i --force --grace-period=0; done
```

2. Wait until the Pod `0` is ready, and execute the following code (it is required for the Pod liveness check):

```
$ for i in $(seq 0 $(($(kubectl get pxc cluster1 -o jsonpath='{.spec.pxc.size}')-1))); do until [[ $(kubectl get pod cluster1-pxc-$i -o jsonpath='{.status.phase}') == 'Running' ]]; do sleep 10; done; kubectl exec cluster1-pxc-$i -- touch /var/lib/mysql/sst_in_progress; done
```

3. Wait for all Percona XtraDB Cluster Pods to start, and execute the following code to make sure no mysqld processes are running:

```
$ for i in $(seq $(($(kubectl get pxc cluster1 -o jsonpath='{.spec.pxc.size}')-1))); do pid=$(kubectl exec cluster1-pxc-$i -- ps -C mysqld-ps -o pid=); if [[ -n "$pid" ]]; then kubectl exec cluster1-pxc-$i -- kill -9 $pid; fi;  done
```

4. Wait for all Percona XtraDB Cluster Pods to start, then find the Percona XtraDB Cluster instance with the most recent data - i.e. the one with the highest sequence number (seqno):

```
$ for i in $(seq 0 $(($(kubectl get pxc cluster1 -o jsonpath='{.spec.pxc.size}')-1))); do echo "###############cluster1-pxc-$i###############"; kubectl exec cluster1-pxc-$i -- cat /var/lib/mysql/grastate.dat; done
```

The output of this command should be similar to the following one:

```
###############cluster1-pxc-0##############
# GALERA saved state
version: 2.1
uuid:    7e037079-6517-11ea-a558-8e77af893c93
seqno:   18
safe_to_bootstrap: 0
###############cluster1-pxc-1##############
# GALERA saved state
version: 2.1
uuid:    7e037079-6517-11ea-a558-8e77af893c93
seqno:   18
safe_to_bootstrap: 0
###############cluster1-pxc-2##############
# GALERA saved state
version: 2.1
uuid:    7e037079-6517-11ea-a558-8e77af893c93
seqno:   19
safe_to_bootstrap: 0
```

Now find the Pod with the largest `seqno` (it is `cluster1-pxc-2` in the above example).

5. Now execute the following commands *in a separate shell* to start this instance:

```
$ kubectl exec cluster1-pxc-2 -- mysqld --wsrep_recover
$ kubectl exec cluster1-pxc-2 -- sed -i 's/safe_to_bootstrap: 0/safe_to_bootstrap: 1/g' /var/lib/mysql/grastate.dat
$ kubectl exec cluster1-pxc-2 -- sed -i 's/wsrep_cluster_address=.*/wsrep_cluster_address=gcomm:\/\//g' /etc/mysql/node.cnf
$ kubectl exec cluster1-pxc-2 -- mysqld
```

The `mysqld` process will initialize the database once again, and it will be available for the incoming connections.

6. Go back *to the previous shell* and return the original Percona XtraDB Cluster image because the debug image is no longer needed:

Note

Please make sure the Percona XtraDB Cluster version for the debug image matches the version currently in use in the cluster.

```
$ kubectl patch pxc cluster1 --type="merge" -p '{"spec":{"pxc":{"image":"percona/percona-xtradb-cluster:8.0.44-35.1"}}}'
```

Note

For Percona XtraDB Cluster 5.7 this command should be as follows:

```
$ kubectl patch pxc cluster1 --type="merge" -p '{"spec":{"pxc":{"image":"percona/percona-xtradb-cluster:5.7.44-31.65"}}}'
```

1. Restart all Pods besides the `cluster1-pxc-2` Pod (the recovery donor).

```
$ for i in $(seq 0 $(($(kubectl get pxc cluster1 -o jsonpath='{.spec.pxc.size}')-1))); do until [[ $(kubectl get pod
cluster1-pxc-$i -o jsonpath='{.status.phase}') == 'Running' ]]; do sleep 10; done; kubectl exec cluster1-pxc-$i -- rm
/var/lib/mysql/sst_in_progress; done
$ kubectl delete pods --force --grace-period=0 cluster1-pxc-0 cluster1-pxc-1
```

2. Wait for the successful startup of the Pods which were deleted during the previous step, and finally remove the `cluster1-pxc-2` Pod:

```
$ kubectl delete pods --force --grace-period=0 cluster1-pxc-2
```

3. After the Pod startup, the cluster is fully recovered.

Note

If you have changed the update strategy on the 1st step, don't forget to revert it back to `SmartUpdate` with the following command:

```
$ kubectl patch pxc cluster1 --type=merge --patch '{"spec": {"updateStrategy": "SmartUpdate" }}'
```

Clone a cluster with the same data set

A good practice is to test a new functionality or an upgraded version of the database in a testing / staging environment. As a developer, you would want the data in the staging database cluster, so that your applications can start immediately.

The `dataSource` functionality allows doing just that. Instead of creating a new PVC for a new cluster, you can clone the existing one. This enables you to spin up a new cluster with the data in it almost in no time which is especially beneficial if you use CI/CD for that.

For example, you have the production Percona XtraDB Cluster `cluster1`. To test a new feature in your app, you need a staging cluster `cluster2` with the data set from `cluster1`.

To create it, create the `cluster2-cr.yaml` Custom Resource manifest. You can use the existing `deploy/cr.yaml` for convenience. Specify the PVC from `cluster1` as the `dataSource` for it:

```
pxc:
  volumeSpec:
    persistentVolumeClaim:
      dataSource:
        name: cluster1-pvc
        kind: PersistentVolumeClaim
```

This configuration instructs the Operator to create a direct clone of the PVC from `cluster1`. If you have a snapshot of the PVC, you can use that as a data source for your staging cluster. Here's how you define it:

```
persistentVolumeClaim:
  dataSource:
    name: cluster1-pvc-snapshot1
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

To create a database cluster, apply the `cluster2-cr.yaml`.

Troubleshooting

Initial troubleshooting

Percona Operator for MySQL uses Custom Resources to manage options for the various components of the cluster.

- `PerconaXtraDBCluster` Custom Resource with Percona XtraDB Cluster options (it has handy `pxc` shortname also),
- `PerconaXtraDBClusterBackup` and `PerconaXtraDBClusterRestore` Custom Resources contain options for Percona XtraBackup used to backup Percona XtraDB Cluster and to restore it from backups (`pxc-backup` and `pxc-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 pxc
```

Expected output

```
NAME       ENDPOINT                   STATUS   PXC   PROXYSQL   HAPROXY   AGE
cluster1   cluster1-haproxy.default   ready    3                3         33d
```

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

```
perconaxtradbclusterbackups      pxc-backup,pxc-backups      pxc.percona.com/v1      true      PerconaXtraDBClusterBackup
perconaxtradbclusterrestores     pxc-restore,pxc-restores    pxc.percona.com/v1      true      PerconaXtraDBClusterRestore
perconaxtradbclusters            pxc,pxcs                    pxc.percona.com/v1      true      PerconaXtraDBCluster
```

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
```

Expected output

```
NAME                                               READY   STATUS    RESTARTS   AGE
cluster1-haproxy-0                                 2/2     Running   0          6m17s
cluster1-haproxy-1                                 2/2     Running   0          4m59s
cluster1-haproxy-2                                 2/2     Running   0          4m36s
cluster1-pxc-0                                     3/3     Running   0          6m17s
cluster1-pxc-1                                     3/3     Running   0          5m3s
cluster1-pxc-2                                     3/3     Running   0          3m56s
percona-xtradb-cluster-operator-79966668bd-rswbk   1/1     Running   0          9m54s
```

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-haproxy-0` Pod 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-pxc-0
```

Expected output

```
...
Name:         cluster1-pxc-0
Namespace:    default
...
Controlled By:  StatefulSet/cluster1-pxc
Init Containers:
 pxc-init:
...
Containers:
 pmm-client:
...
 pxc:
...
   Restart Count:  0
   Limits:
     cpu:     1
     memory:  2G
   Requests:
     cpu:      1
     memory:   2G
   Liveness:   exec [/var/lib/mysql/liveness-check.sh] delay=300s timeout=5s period=10s #success=1 #failure=3
   Readiness:  exec [/var/lib/mysql/readiness-check.sh] delay=15s timeout=15s period=30s #success=1 #failure=5
   Environment Variables from:
     pxc-env-vars-pxc  Secret  Optional: true
   Environment:
...
   Mounts:
...
Volumes:
...
Events:                      <none>
```

This gives a lot of information about containers, resources, container status and also events. So, describe output should be checked to see any abnormalities.

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 `pxc` of the `cluster1-pxc-0` Pod.

- Run `date` command:

```
$ kubectl exec -ti cluster1-pxc-0 -c pxc -- date
```

Expected output

```
Thu Nov 24 10:01:17 UTC 2022
```

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-pxc-0 -c pxc -- time` would show the following result:

```
error: Internal error occurred: error executing command in container: failed to exec in container: failed to start exec
"71bdb96a65af89d3672cd0d69a8f2c1068542a97b1938e7f6f17d29a87d76453": OCI runtime exec failed: exec failed: unable to start
container process: exec: "time": executable file not found in $PATH: unknown
```

- Print `/var/log/mysqld.log` file to a terminal:

```
$ kubectl exec -ti cluster1-pxc-0 -c pxc -- cat /var/log/mysqld.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-pxc-0 -c pxc -- bash
bash-4.4$ hostname
cluster1-pxc-0
bash-4.4$ ls /var/log/mysqld.log
/var/log/mysqld.log
bash-4.4$ exit
exit
$
```

Avoid the restart-on-fail loop for Percona XtraDB Cluster containers

The restart-on-fail loop takes place when the container entry point fails (e.g. `mysqld` crashes). In such a situation, Pod is continuously restarting. Continuous restarts prevent to get console access to the container, and so a special approach is needed to make fixes.

You can prevent such infinite boot loop by putting the Percona XtraDB Cluster containers into the infinity loop *without* starting mysqld. This behavior of the container entry point is triggered by the presence of the `/var/lib/mysql/sleep-forever` file.

For example, you can do it for the `pxc` container of an appropriate Percona XtraDB Cluster instance as follows:

```
$ kubectl exec -it cluster1-pxc-0 -c pxc -- sh -c 'touch /var/lib/mysql/sleep-forever'
```

If `pxc` container can’t start, you can use `logs` container instead:

```
$ kubectl exec -it cluster1-pxc-0 -c logs -- sh -c 'touch /var/lib/mysql/sleep-forever'
```

The instance will restart automatically and run in its usual way as soon as you remove this file (you can do it with a command similar to the one you have used to create the file, just substitute `touch` to `rm` in it).

Check the Events

Kubernetes Events always provide a wealth of information and should always be checked while troubleshooting issues.

Events can be checked by the following command

```
$ kubectl get events
```

Expected output

```
LAST SEEN   TYPE      REASON                  OBJECT                                                                  MESSAGE
38m         Normal    Provisioning            persistentvolumeclaim/xb-cron-pxc-pxc-backup-stora-202211231300-3qf7g    External provisioner is
provisioning volume for claim "default/xb-cron-pxc-pxc-backup-stora-202211231300-3qf7g"
...
```

Events capture many information happening at Kubernetes level and provide valuable information. By default, the ordering of events cannot be guaranteed. Use the following command to sort the output in a reverse chronological fashion.

```
$ kubectl get events --sort-by=".lastTimestamp"
```

Expected output

```
LAST SEEN   TYPE      REASON                  OBJECT                                                  MESSAGE
13m         Normal    Created                 pod/xb-cron-pxc-pxc-backup-stora-2022112313300-3qf7g-brxmv              Created container
xtrabackup
13m         Normal    Started                 pod/xb-cron-pxc-pxc-backup-stora-2022112313300-3qf7g-brxmv              Started container
xtrabackup
12m         Normal    Completed               job/xb-cron-pxc-pxc-backup-stora-2022112313300-3qf7g                    Job completed
...
```

When there are too many events and there is a need of filtering output, tools like yq, jq can be used to filter specific items or know the structure of the events.

Example:

```
$ kubectl get events -oyaml | yq .items[0]
```

Expected output

```
apiVersion: v1
count: 2
eventTime: null
firstTimestamp: "2022-11-24T05:30:00Z"
involvedObject:
 apiVersion: v1
 kind: Pod
 name: xb-cron-pxc-pxc-backup-stora-202211245300-3qf7g-bpm5s
 namespace: default
 resourceVersion: "41813970"
 uid: c2463e2a-65a0-4fc2-b5c3-86d88bba6b5b
kind: Event
lastTimestamp: "2022-11-24T05:30:03Z"
message: '0/6 nodes are available: 6 pod has unbound immediate PersistentVolumeClaims.'
metadata:
 creationTimestamp: "2022-11-24T05:30:00Z"
 name: xb-cron-pxc-pxc-backup-stora-202211245300-3qf7g-bpm5s.172a6e3851f6710c
 namespace: default
 resourceVersion: "94245"
 uid: d56ea5b8-3b15-4a22-a6ea-a4f641fcc54e
reason: FailedScheduling
reportingComponent: ""
reportingInstance: ""
source:
 component: default-scheduler
type: Warning
```

Flag `--field-selector` can be used to filter out the output as well. For example, the following command provides events of Pod only:

```
$ kubectl get events --field-selector involvedObject.kind=Pod
```

More fields can be added to the field-selector flag for filtering events further. Example: the following command provides events of Pod by name `xb-cron-pxc-pxc-backup-stora-202211245300-3qf7g-bpm5s`.

```
$ kubectl get events --field-selector involvedObject.kind=Pod,involvedObject.name=xb-cron-pxc-pxc-backup-stora-202211245300-3qf7g-bpm5s
```

Expected output

```
LAST SEEN   TYPE      REASON                  OBJECT                                                  MESSAGE
53m         Warning   FailedScheduling        pod/xb-cron-pxc-pxc-backup-stora-202211245300-3qf7g-bpm5s   0/6 nodes are available: 6 pod has
unbound immediate PersistentVolumeClaims.
53m         Normal    NotTriggerScaleUp       pod/xb-cron-pxc-pxc-backup-stora-202211245300-3qf7g-bpm5s   pod didn't trigger scale-up: 3 pod has
unbound immediate PersistentVolumeClaims
```

Same way you can query events for other Kubernetes object (StatefulSet, Custom Resource, etc.) to investigate any problems to them:

```
$ kubectl get events --field-selector involvedObject.kind=PerconaXtraDBCluster,involvedObject.name=cluster1
```

Expected output

```
LAST SEEN   TYPE      REASON                    OBJECT                        MESSAGE
10m         Warning   AsyncReplicationNotReady  perconaservermysql/cluster1   cluster1-mysql-1: [not_replicating]
...
```

Alternatively, you can see events for a specific object in the output of `kubectl describe` command:

```
$ kubectl describe ps cluster1
```

Expected output

```
Name:         cluster1
...
Events:
  Type     Reason                   Age                From           Message
  ----     ------                   ----               ----           -------
  Warning  AsyncReplicationNotReady  10m (x23 over 13m)    ps-controller  cluster1-mysql-1: [not_replicating]
...
```

Check `kubectl get events --help` to know about more options.

Note

It is important to note that events are stored in the etcd for only 60 minutes. Ensure that events are checked within 60 minutes of the issue. Kubernetes cluster administrators might also use event exporters for storing the events.

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
`--container=<container-name>`	Print log of a specific container in case of multiple containers in a Pod
`--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)
`--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-pxc-0` Pod.

- Check logs of the `pxc` container:

```
$ kubectl logs cluster1-pxc-0 -c pxc
```

- Check logs of the `pmm-client` container:

```
$ kubectl logs cluster1-pxc-0 -c pmm-client
```

- Filter logs of the `pxc` container which are not older than 600 seconds:

```
$ kubectl logs cluster1-pxc-0 -c pxc --since=600s
```

- Check logs of a previous instantiation of the `pxc` container, if any:

```
$ kubectl logs cluster1-pxc-0 -c pxc --previous
```

- Check logs of the `pxc` container, parsing the output with jq JSON processor:

```
$ kubectl logs cluster1-pxc-0 -c pxc -f | jq -R 'fromjson?'
```

Cluster-level logging

Cluster-level logging involves collecting logs from all Percona XtraDB Cluster Pods in the cluster to some persistent storage. This feature gives the logs a lifecycle independent of nodes, Pods and containers in which they were collected. Particularly, it ensures that Pod logs from previous failures are available for later review.

Log collector is turned on by the `logcollector.enabled` key in the `deploy/cr.yaml` configuration file (`true` by default).

The Operator collects logs using Fluent Bit Log Processor, which supports many output plugins and has broad forwarding capabilities. If necessary, Fluent Bit filtering and advanced features can be configured via the `logcollector.configuration` key in the `deploy/cr.yaml` configuration file.

Logs are stored for 7 days and then rotated.

Collected logs can be examined using the following command:

```
$ kubectl logs cluster1-pxc-0 -c logs
```

Note

Technically, logs are stored on the same Persistent Volume, which is used with the corresponding Percona XtraDB Cluster Pod. Therefore collected logs can be found in `DATADIR` (`var/lib/mysql/`). Also, there is an additional Secrets object for Fluent Bit passwords and other similar data, e.g. for output plugins. The name of this Secrets object can be found in the `logCollectorSecretName` option of the Custom Resource (it is set to `my-log-collector-secrets` in the `deploy/cr.yaml` configuration file by default).

Check Storage-related objects

Storage-related objects worth to check in case of problems are the following ones:

- Persistent Volume Claims (PVC) and Persistent Volumes (PV), which are playing a key role in stateful applications.
- Storage Class, which automates the creation of Persistent Volumes and the underlying storage.

It is important to remember that PVC is namespace-scoped, but PV and Storage Class are cluster-scoped.

Check the PVC

You can check all the PVC with the following command (use your namespace name instead of the `<namespace>` placeholder):

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

Expected output

```
NAME                STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
datadir-pxc-pxc-0   Bound    pvc-f3e7097f-accd-4f5d-9c9d-6f29b54a368b   24Gi       RWO            standard       42d
datadir-pxc-pxc-1   Bound    pvc-3ec336a8-69de-4cbc-aff8-700d41696447   24Gi       RWO            standard       42d
datadir-pxc-pxc-2   Bound    pvc-207e8a3e-1c83-4eae-b3f2-cf126f89ba9e   24Gi       RWO            standard       42d
```

The fields in the output of this command provide the following insights:

- **STATUS**: shows the state of the PVC:
  - For normal working of an application, the status should be `Bound`.
  - If the status is not `Bound`, further investigation is required.
- **VOLUME**: is the name of the Persistent Volume with which PVC is Bound to. Obviously, this field will be occupied only when a PVC is Bound.
- **CAPACITY**: it is the size of the volume claimed.
- **STORAGECLASS**: it indicates the Kubernetes storage class used for dynamic provisioning of Volume.
- **ACCESS MODES**: Access mode indicates how Volume is used with the Pods. Access modes should have write permission if the application needs to write data, which is obviously true in case of databases.

Now you can check a specific PVC for more details using its name as follows:

```
$ kubectl get pvc datadir-pxc-pxc-0 -n <namespace> -oyaml # output stripped for brevity, name of PVC may vary
```

Expected output

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  ...
  name: datadir-pxc-pxc-0
  namespace: default
  ...
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 25G
  storageClassName: standard
  volumeMode: Filesystem
  volumeName: pvc-f3e7097f-accd-4f5d-9c9d-6f29b54a368b
status:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 24Gi
  phase: Bound
```

Check the PV

It is important to remember that PV is a cluster-scoped Object. If you see any issues with attaching a Volume to a Pod, PV and PVC might be looked upon.

Check all the PV present in the Kubernetes cluster as follows:

```
$ kubectl get pv
```

Expected output

```
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
STORAGECLASS   REASON   AGE
pvc-207e8a3e-1c83-4eae-b3f2-cf126f89ba9e   24Gi       RWO            Delete           Bound       default/datadir-pxc-pxc-2
standard                43d
pvc-3ec336a8-69de-4cbc-aff8-700d41696447   24Gi       RWO            Delete           Bound       default/datadir-pxc-pxc-1
standard                43d
pvc-f3e7097f-accd-4f5d-9c9d-6f29b54a368b   24Gi       RWO            Delete           Bound       default/datadir-pxc-pxc-0
standard                43d
```

Now you can check a specific PV for more details using its name as follows:

```
$ kubectl get pv pvc-f3e7097f-accd-4f5d-9c9d-6f29b54a368b -oyaml
```

Expected output

```
apiVersion: v1
kind: PersistentVolume
metadata:
  ...
  name: pvc-f3e7097f-accd-4f5d-9c9d-6f29b54a368b
 ...
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 24Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: datadir-pxc-pxc-0
    namespace: default
    resourceVersion: "912868"
    uid: f3e7097f-accd-4f5d-9c9d-6f29b54a368b
  gcePersistentDisk:
    fsType: ext4
    pdName: pvc-f3e7097f-accd-4f5d-9c9d-6f29b54a368b
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: topology.kubernetes.io/zone
          operator: In
          values:
          - us-central1-a
        - key: topology.kubernetes.io/region
          operator: In
          values:
          - us-central1
  persistentVolumeReclaimPolicy: Delete
  storageClassName: standard
  volumeMode: Filesystem
status:
  phase: Bound
```

Fields to check if there are any issues in binding with PVC, are the `claimRef` and `nodeAffinity`.

The `claimRef` one indicates to which PVC this volume is bound to. This means that if by any chance PVC is deleted (e.g. by the appropriate finalizer), this section needs to be modified so that it can bind to a new PVC.

The `spec.nodeAffinity` field may influence the PV availability as well: for example, it can make Volume accessed in one availability zone only.

Check the StorageClass

StorageClass is also a cluster-scoped object, and it indicates what type of underlying storage is used for the Volumes.

You can set StorageClass in `pxc.volumeSpec.persistentVolumeClaim.storageClassName`, `proxysql.volumeSpec.persistentVolumeClaim.storageClassName`, and `backup.storages.STORAGE-NAME.persistentVolumeClaim.storageClassName` Custom Resource options.

The following command checks all the storage class present in the Kubernetes cluster, and allows to see which storage class is the default one:

```
$ kubectl get sc
```

Expected output

```
NAME                PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
premium-rwo         pd.csi.storage.gke.io   Delete          WaitForFirstConsumer   true                   44d
standard (default)  kubernetes.io/gce-pd    Delete          Immediate              true                   44d
standard-rwo        pd.csi.storage.gke.io   Delete          WaitForFirstConsumer   true                   44d
```

If some PVC does not refer any storage class explicitly, it means that the default storage class is used. Ensure there is only one default Storage class.

You can check a specific storage class as follows:

```
$ kubectl get sc standard -oyaml
```

Expected output

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 annotations:
   storageclass.kubernetes.io/is-default-class: "true"
 creationTimestamp: "2022-10-09T06:28:03Z"
 labels:
   addonmanager.kubernetes.io/mode: EnsureExists
 name: standard
 resourceVersion: "906"
 uid: 933d37db-990b-4b2d-bf3a-dd091d0b00ae
parameters:
 type: pd-standard
provisioner: kubernetes.io/gce-pd
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

Important things to observe here are the following ones:

- Check if the provisioner and parameters are indicating the type of storage you intend to provision.
- Check the volumeBindingMode especially if the storage cannot be accessed across availability zones. “WaitForFirstConsumer” volumeBindingMode ensures volume is provisioned only after a Pod requesting the Volume is created.
- If you are going to rely on the Operator storage scaling functionality, ensure the storage class supports PVC expansion (it should have `allowVolumeExpansion: true` in the output of the above command).

Special debug images

For the cases when Pods are failing for some reason or just show abnormal behavior, the Operator can be used with a special *debug images*. Percona XtraDB Cluster debug image has the following specifics:

- it avoids restarting on fail,
- it contains additional tools useful for debugging (sudo, telnet, gdb, etc.),
- it has debug mode enabled for the logs.

There are debug versions for all Percona XtraDB Cluster images: they have same names as normal images with a special `-debug` suffix in their version tag: for example, `percona-xtradb-cluster:8.4.7-7.1-debug`.

To use the debug image instead of the normal one, find the needed image name in the list of certified images and set it for the proper key in the `deploy/cr.yaml` configuration file. For example, set the following value of the `pxc.image` key to use the Percona XtraDB Cluster debug image:

- `percona/percona-xtradb-cluster:8.4.7-7.1-debug` for Percona XtraDB Cluster 8.4,
- `percona/percona-xtradb-cluster:8.0.44-35.1-debug` for Percona XtraDB Cluster 8.0,
- `percona/percona-xtradb-cluster:5.7.44-31.65-debug` for Percona XtraDB Cluster 5.7.

The Pod should be restarted to get the new image.

Note

When the Pod is continuously restarting, you may have to delete it to apply image changes.

HOWTOs

Install Percona XtraDB Cluster with customized parameters

You can customize the configuration of Percona XtraDB Cluster and install it with customized parameters.

To check available configuration options, see `deploy/cr.yaml` and Custom Resource Options.

`kubectl`

To customize the configuration, do the following:

1. Clone the repository with all manifests and source code by executing the following command:

   ```
   $ git clone -b v1.19.0 https://github.com/percona/percona-xtradb-cluster-operator
   ```

2. Edit the required options and apply the modified `deploy/cr.yaml` file as follows:

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

Helm

To install Percona XtraDB Cluster with custom parameters, 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 Percona XtraDB Cluster in the `pxc` namespace, with disabled backups and 20 Gi storage:

Command line

```
$ helm install my-db percona/pxc-db --version 1.19.0 --namespace pxc \
  --set pxc.volumeSpec.resources.requests.storage=20Gi \
  --set backup.enabled=false
```

YAML file

You can specify customized options in a YAML file instead of using separate command line parameters. The resulting file similar to the following example looks as follows:

values.yaml

```
allowUnsafeConfigurations: true
sharding:
  enabled: false
pxc:
  size: 3
  volumeSpec:
    pvc:
      resources:
        requests:
          storage: 2Gi
backup:
  enabled: false
```

Apply the resulting YAML file as follows:

```
$ helm install my-db percona/pxc-db --namespace pxc -f values.yaml
```

Percona Operator for MySQL based on Percona XtraDB Cluster single-namespace and multi-namespace deployment

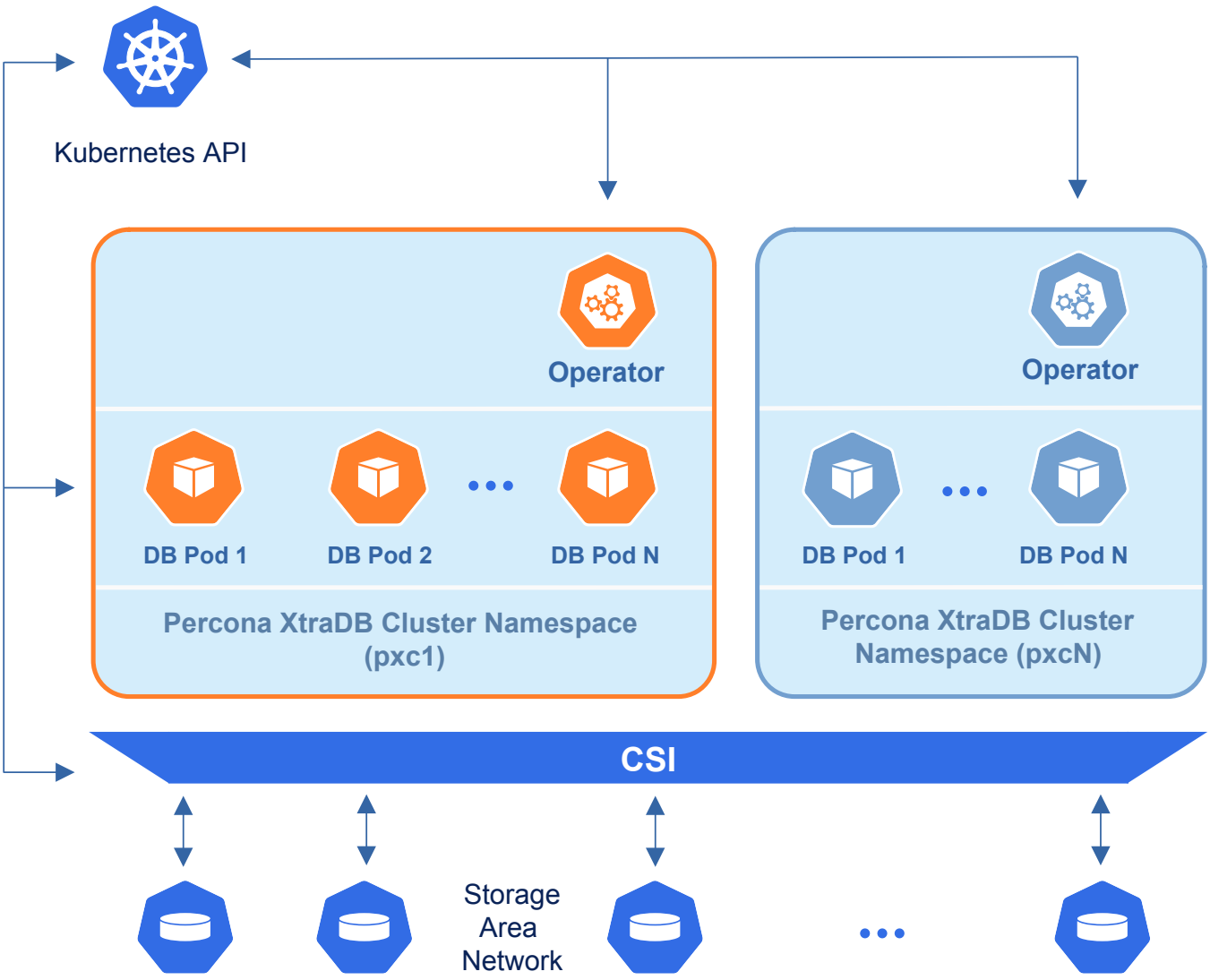
There are two design patterns that you can choose from when deploying Percona Operator for MySQL based on Percona XtraDB Cluster and database clusters in Kubernetes:

- 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 MySQL based on Percona XtraDB Cluster for each scenario.

Namespace-scope

By default, Percona Operator for MySQL functions in a specific Kubernetes namespace. You can create one during the 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.

Let's say you will use a Kubernetes Namespace called `percona-db-1`.

1. Clone `percona-xtradb-cluster-operator` repository:

```
$ git clone -b v1.19.0 git@github.com:percona/percona-xtradb-cluster-operator.git
$ cd percona-xtradb-cluster-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:

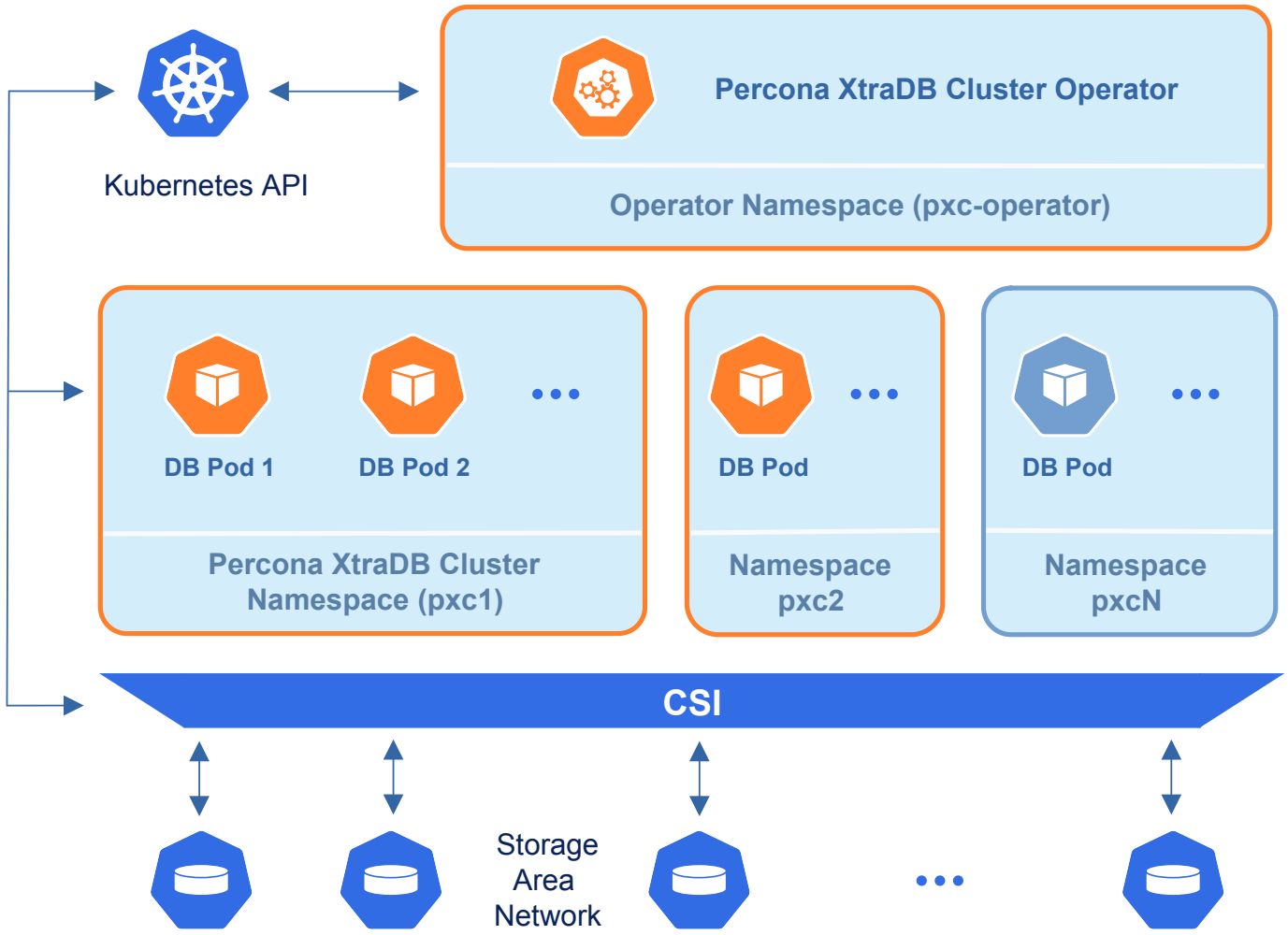
```
$ kubectl apply -f deploy/cr.yaml -n percona-db-2
```

> **Note**
>
> Cluster names may be the same in different namespaces.

Installing the Operator in cluster-wide mode

Sometimes it is more convenient to have one Operator watching for Percona XtraDB Cluster custom resources in several namespaces.

We recommend running Percona Operator for MySQL 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:



Note

Please take into account that if several Operators are configured to watch the same namespace, it is entirely unpredictable which one will get ownership of the Custom Resource in it, so this situation should be avoided.

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` environment variable should be set to a comma-separated list of the namespaces to be watched by the Operator (or an empty string to watch all namespaces). See Operator environment variables for more details. Prior to the Operator version 1.12.0 it was necessary to mention the Operator's own namespace in the list of watched namespaces, but now this limitation has gone.

Note

Installing the Operator cluster-wide on OpenShift via the the Operator Lifecycle Manager (OLM) requires making different selections in the OLM web-based UI instead of patching YAML files.

The following simple example shows how to install Operator cluster-wide on Kubernetes.

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

```
$ git clone -b v1.19.0 https://github.com/percona/percona-xtradb-cluster-operator
$ cd percona-xtradb-cluster-operator
```

2. Let's suppose that Operator's namespace should be the `pxc-operator` one. Create it as follows:

   ```
   $ kubectl create namespace pxc-operator
   ```

   Namespaces to be watched by the Operator should be created in the same way if not exist. Let's say the Operator should watch the `pxc` namespace:

   ```
   $ kubectl create namespace pxc
   ```

3. Edit the `deploy/cw-bundle.yaml` configuration file to set proper namespaces:

   ```
   ...
   subjects:
   - kind: ServiceAccount
     name: percona-xtradb-cluster-operator
     namespace: "pxc"
   ...
   env:
            - name: WATCH_NAMESPACE
              value: "pxc"
   ...
   ```

4. Apply the `deploy/cw-bundle.yaml` file with the following command:

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

5. After the Operator is started, Percona XtraDB Cluster can be created at any time by applying the `deploy/cr.yaml` configuration file, like in the case of normal installation:

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

   The creation process will 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 pxc
   ```

   Expected output

   ```
   NAME       ENDPOINT                  STATUS   PXC   PROXYSQL   HAPROXY   AGE
   cluster1   cluster1-haproxy.default   ready    3               3         5m51s
   ```

Verifying the cluster operation

It may take ten minutes to get the cluster started. When `kubectl get pxc` command finally shows you the cluster status as `ready`, you can try to connect to the cluster.

1. You will need the login and password for the admin user to access the cluster. Use `kubectl get secrets` command to see the list of Secrets objects (by default the Secrets object you are interested in has `cluster1-secrets` name). You can use the following command to get the password of the `root` user:

   ```
   $ kubectl get secrets --namespace=pxc cluster1-secrets --template='{{.data.root | base64decode}}{{"\n"}}'
   ```

2. Run a container with `mysql` client and connect its console output to your terminal. The following command will do this, naming the new Pod `percona-client`:

   ```
   $ kubectl run -i --rm --tty percona-client --image=percona:5.7 --restart=Never --env="POD_NAMESPACE=pxc" -- bash -il
   ```

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

   Now run `mysql` tool in the percona-client command shell using the password obtained from the secret instead of the `<root_password>` placeholder. The command will look different depending on whether your cluster provides load balancing with HAProxy (the default choice) or ProxySQL:

with HAProxy (default)

```
$ mysql -h cluster1-haproxy -uroot -p'<root_password>'
```

with ProxySQL

```
$ mysql -h cluster1-proxysql -uroot -p'<root_password>'
```

Note

Some Kubernetes-based environments are specifically configured to have communication across Namespaces is not allowed by default network policies. In this case, you should specifically allow the Operator communication across the needed Namespaces. Following the above example, you would need to allow ingress traffic for the `pxc-operator` Namespace from the `pxc` Namespace, and also from the `default` Namespace. You can do it with the NetworkPolicy resource, specified in the YAML file as follows:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: percona
  namespace: pxc-operator
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: pxc
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: default
  podSelector: {}
  policyTypes:
  - Ingress
```

Don't forget to apply the resulting file with the usual `kubectl apply` command.

You can find more details about Network Policies in the official Kubernetes documentation.

Upgrading the Operator in cluster-wide mode

Cluster-wide Operator is upgraded similarly to a single-namespace one. Both deployment variants provide you with the same three upgradable components:

- the Operator;
- Custom Resource Definition (CRD),
- Database Management System (Percona XtraDB Cluster).

To upgrade the cluster-wide Operator you follow the standard upgrade scenario concerning the Operator's namespace and a different YAML configuration file: the one with a special `cw-` prefix, `deploy/cw-rbac.yaml`. The resulting steps will look as follows.

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-xtradb-cluster-operator/v1.19.0/deploy/crd.yaml
   $ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-xtradb-cluster-operator/v1.19.0/deploy/cw-rbac.yaml
   ```

2. Now you should apply a patch to your deployment, supplying the 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 (for older releases, please refer to the old releases documentation archive). For example, updating to the `1.19.0` version in the `pxc-operator` namespace should look as follows.

   ```
   $ kubectl patch deployment percona-xtradb-cluster-operator \
     -p'{"spec":{"template":{"spec":{"containers":[{"name":"percona-xtradb-cluster-operator","image":"percona/percona-xtradb-cluster-operator:1.19.0"}]}}}}' -n pxc-operator
   ```

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-xtradb-cluster-operator -n pxc-operator
   ```

Use docker images from a custom registry

Using images from a private Docker registry may be useful in different situations: it may be related to storing images inside of a company, for privacy and security reasons, etc. In such cases, Percona Distribution for MySQL Operator based on Percona XtraDB Cluster allows to use a custom registry, and the following instruction illustrates how this can be done by the example of the Operator deployed in the OpenShift environment.

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 pxc
Now using project "pxc" 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 find out 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. Now you can use the obtained token and address to login to the registry:

```
$ docker login -u admin -p ADO8CqCDappWR4hxjfDqwijEHei31yXAvWg61Jg210s 172.30.162.173:5000
Login Succeeded
```

4. Pull the needed image by its SHA digest:

```
$ docker pull docker.io/perconalab/percona-xtradb-cluster-
operator@sha256:841c07eef30605080bfe80e549f9332ab6b9755fcbc42aacbf86e4ac9ef0e444
Trying to pull repository docker.io/perconalab/percona-xtradb-cluster-operator ...
sha256:841c07eef30605080bfe80e549f9332ab6b9755fcbc42aacbf86e4ac9ef0e444: Pulling from docker.io/perconalab/percona-xtradb-
cluster-operator
Digest: sha256:841c07eef30605080bfe80e549f9332ab6b9755fcbc42aacbf86e4ac9ef0e444
Status: Image is up to date for docker.io/perconalab/percona-xtradb-cluster-
operator@sha256:841c07eef30605080bfe80e549f9332ab6b9755fcbc42aacbf86e4ac9ef0e444
```

   You can find correct names and SHA digests in the current list of the Operator-related images officially certified by Percona.

5. The following way is used to push an image to the custom registry (into the OpenShift pxc project):

```
$ docker tag \
    docker.io/perconalab/percona-xtradb-cluster-
operator@sha256:841c07eef30605080bfe80e549f9332ab6b9755fcbc42aacbf86e4ac9ef0e444 \
    172.30.162.173:5000/pxc/percona-xtradb-cluster-operator:1.19.0
$ docker push 172.30.162.173:5000/pxc/percona-xtradb-cluster-operator:1.19.0
```

6. Check the image in the OpenShift registry with the following command:

```
$ oc get is
NAME                            DOCKER REPO                                                    TAGS      UPDATED
percona-xtradb-cluster-operator   docker-registry.default.svc:5000/pxc/percona-xtradb-cluster-operator   1.19.0     2 hours
ago
```

7. When the custom registry image is Ok, put a Docker Repo + Tag string (it should look like `docker-registry.default.svc:5000/pxc/percona-xtradb-cluster-operator:1.19.0`) into the `initImage` option in `deploy/operator.yaml` configuration file.

8. Repeat steps 3-5 for other images, updating the `image\` `options in the corresponding sections of the the ``deploy/cr.yaml` file.

   > **Note**
   >
   > Don't forget to set upgradeoptions.apply option to `Disabled`. Otherwise Smart Upgrade functionality will try using the image recommended by the Version Service instead of the custom one.

   Please note it is possible to specify `imagePullSecrets` option for the images, if the registry requires authentication.

9. Now follow the standard Percona Operator for MySQL installation instruction.

How to use backups and asynchronous replication to move an external database to Kubernetes

The Operator enables you to restore a database from a backup made outside of Kubernetes environment to the target Kubernetes cluster using Percona XtraBackup. In such a way you can migrate your external database to Kubernetes. Using asynchronous replication between source and target environments enables you to reduce downtime and prevent data loss for your application.

This document provides the steps how to migrate Percona Server for MySQL 8.0 deployed on premises to the Kubernetes cluster managed by the Operator using asynchronous replication. We recommend testing this migration in a non-production environment first, before applying it in production.

Requirements

1. The MySQL version for source and target environments must be 8.0.22 and higher since asynchronous replication is available starting with MySQL version 8.0.22.
2. You must be running Percona XtraBackup as the backup tool on source environment. For how to install Percona XtraBackup, see the installation instructions
3. The storage used to save the backup should be one of the supported cloud storages: AWS S3 or compatible storage, or Azure Blob Storage.

Configure target environment

1. Deploy Percona Operator for MySQL and use it to create Percona XtraDB Cluster following any of the official installation guides.
2. Create the YAML file with the credentials for accessing the storage, needed to create the Kubernetes Secrets object. As and example here, we will use Amazon S3 storage. You will need to create a Secret with the following data to store backups on the Amazon S3:
   - the `metadata.name` key is the name which you will 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 following the deploy/backup-s3.yaml example:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: my-cluster-name-backup-s3
   type: Opaque
   data:
     AWS_ACCESS_KEY_ID: UkVQTEFDRS1XSVRILUFXUy1BQ0NFU1MtS0VZ
     AWS_SECRET_ACCESS_KEY: UkVQTEFDRS1XSVRILUFXUy1TRUNSRVQtS0VZ
   ```

   Note

   You can use the following command to get a base64-encoded string from a plain text one:

   in Linux

   ```
   $ echo -n 'plain-text-string' | base64 --wrap=0
   ```

   in macOS

   ```
   $ echo -n 'plain-text-string' | base64
   ```

3. Once the editing is over, create the Kubernetes Secret object as follows:

   ```
   $ kubectl apply -f deploy/backup-s3.yaml
   ```

4. You will need passwords for the `monitor`, `operator`, `xtrabackup` and `replication` system users created by the Operator. Use `kubectl get secrets` command to see the list of Secrets objects (by default the Secrets object you are interested in has `cluster1-secrets` name). When you know the Secrets name, you can get password for a specific user as follows:

   ```
   $ kubectl get secrets cluster1-secrets --template='{{.data.<user_name> | base64decode}}{{"\n"}}'
   ```

Repeat this command 4 times, substituting with `monitor`, `operator`, `xtrabackup` and `replication`. You will further use these passwords when preparing the source environment.

Prepare the source environment

1. Use official installation instructions for either Percona Server for MySQL or Percona XtraDB Cluster to have the database up and running in your source environment (skip this step if one of them is already installed).
2. Use official installation instructions for Percona XtraBackup to have it up and running in your source environment (skip this step if it is already installed).
3. Configure the replication with Global Transaction Identifiers (GTID). This step is required if you are running Percona Sever for MySQL. If you run Percona XtraDB cluster, replication is already configured.

   Edit the `my.cnf` configuration file as follows:

   ```
   [mysqld]
   enforce_gtid_consistency=ON
   gtid_mode=ON
   ```

4. Create the `monitor`, `operator`, `xtrabackup`, and `replication` system users which will be needed by the Operator to restore a backup. User passwords must match the ones you have found out in your target environment.

   Use the following commands to create users with the actual passwords instead of the `monitor_password`, `operator_password`, `xtrabackup_password`, and `replication_password` placeholders:

   ```
   CREATE USER 'monitor'@'%' IDENTIFIED BY 'monitor_password' WITH MAX_USER_CONNECTIONS 100;
   GRANT SELECT, PROCESS, SUPER, REPLICATION CLIENT, RELOAD ON *.* TO 'monitor'@'%';
   GRANT SERVICE_CONNECTION_ADMIN ON *.* TO 'monitor'@'%';

   CREATE USER 'operator'@'%' IDENTIFIED BY 'operator_password';
   GRANT ALL ON *.* TO 'operator'@'%' WITH GRANT OPTION;

   CREATE USER 'xtrabackup'@'%' IDENTIFIED BY 'xtrabackup_password';
   GRANT ALL ON *.* TO 'xtrabackup'@'%';

   CREATE USER 'replication'@'%' IDENTIFIED BY 'replication_password';
   GRANT REPLICATION SLAVE ON *.* to 'replication'@'%';
   FLUSH PRIVILEGES;
   ```

Make a backup in the source environment

1. Export the storage credentials as environment variables. Following the above example, here is a command which shows how to export the AWS S3 credentials:

   ```
   $ export AWS_ACCESS_KEY_ID=XXXXXX
   $ export AWS_SECRET_ACCESS_KEY=XXXXXX
   ```

   Don't forget to replace the `XXXX` placeholders with your actual Amazon access key ID and secret access key values.
2. Make the backup of your database and upload it to the storage using xbcloud. Replace the values for the `--target-dir`, `--password`, `--s3-bucket` with your values in the following command:

   ```
   $ xtrabackup --backup --stream=xbstream --target-dir=/tmp/backups/ --extra-lsndir=/tmp/backups/  --password=root_password |
   xbcloud put --storage=s3 --parallel=10 --md5 --s3-bucket="mysql-testing-bucket" "db-test-1"
   ```

Restore from a backup in the target environment

If your source database didn't have any data, skip this step and proceed with the asynchronous replication configuration. Otherwise, restore the database in the target environment.

1. To restore a backup, you will use the special restore configuration file. The example of such file is deploy/backup/restore.yaml. For example. your `restore.yaml` file may have the following contents:

   restore.yaml

```
apiVersion: pxc.percona.com/v1
kind: PerconaXtraDBClusterRestore
metadata:
  name: restore1
spec:
  pxcCluster: cluster1
  backupSource:
    destination: s3://mysql-testing-bucket/db-test-1
    s3:
      credentialsSecret: my-cluster-name-backup-s3
      region: us-west-2
```

Don't forget to replace the path to the backup and the credentials with your values.

2. Restore from the backup:

```
$ kubectl apply -f restore.yml
```

You can find more information on restoring backup to a new Kubernetes-based environment and see more examples in a dedicated HowTo.

Configure asynchronous replication in the Kubernetes cluster

This step will allow you to avoid data loss for your application, configuring the asynchronous replication between the source database and the target cluster.

1. Edit the Custom Resource manifest `deploy/cr.yaml` in your target environment to configure the `spec.pxc.replicationChannels` section.

cr.yaml

```
spec:
  ...
  pxc:
    ...
    replicationChannels:
    - name: ps_to_pxc1
      isSource: false
      sourcesList:
        - host: <source_ip>
          port: 3306
          weight: 100
```

Apply the changes for your Custom Resource as usual:

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

2. Verify the replication by connecting to a Percona XtraDB Cluster node. You can do it with `mysql` tool, and you will need the `root` system user password created by the Operator for this. The password can be obtained in a same way we used for other system users:

```
$ kubectl get secrets cluster1-secrets -o yaml -o jsonpath='{.data.root}' | base64 --decode | tr '\n' ' ' && echo " "
```

When you know the `root` password, you can use `kubectl` command as follows (substitute `root_password` with the actual password you have just obtained):

```
$ kubectl exec -it cluster1-pxc-0 -c pxc -- mysql -uroot -proot_password -e "show replica status \G"
```

Expected output

```
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: <ip-of-source-db>
                  Master_User: replication
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: binlog.000004
          Read_Master_Log_Pos: 529
               Relay_Log_File: cluster1-pxc-0-relay-bin-ps_to_pxc1.000002
                Relay_Log_Pos: 738
        Relay_Master_Log_File: binlog.000004
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 529
              Relay_Log_Space: 969
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
                  Master_UUID: 9741945e-148d-11ec-89e9-5ee1a3cf433f
             Master_Info_File: mysql.slave_master_info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay log; waiting for more updates
           Master_Retry_Count: 3
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set: 9741945e-148d-11ec-89e9-5ee1a3cf433f:1-2
            Executed_Gtid_Set: 93f1e7bf-1495-11ec-80b2-06e6016a7c3d:1,
9647dc03-1495-11ec-a385-7e3b2511dacb:1-7,
9741945e-148d-11ec-89e9-5ee1a3cf433f:1-2
                Auto_Position: 1
         Replicate_Rewrite_DB:
                 Channel_Name: ps_to_pxc1
           Master_TLS_Version:
       Master_public_key_path:
        Get_master_public_key: 0
            Network_Namespace:
```

Promote the Kubernetes cluster as primary

After you reconfigured your application to work with the new Percona XtraDB Cluster in Kubernetes, you can promote this cluster as primary.

1. Stop the replication. Edit the `deploy/cr.yaml` manifest and comment the `replicationChannels` subsection:

cr.yaml

```
...
spec:
  ...
  pxc:
    ...
    #replicationChannels:
    #- name: ps_to_pxc1
    #  isSource: false
    #  sourcesList:
    #    - host: <source_ip>
    #      port: 3306
    #      weight: 100
```

2. Stop the `mysqld` service in your source environment to make sure no new data is written:

```
$ sudo systemctl stop mysqld
```

3. Apply the changes to the Kubernetes cluster in your target environment:

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

As a result, replication is stopped and the read-only mode is disabled for the Percona XtraDB Cluster.

This document is based on the blog post Migration of a MySQL Database to a Kubernetes Cluster Using Asynchronous Replication by *Slava Sarzhan*.

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.

The Operator is compatible with both PMM versions 2 and 3. We recommend using the latest PMM version 3 for optimal monitoring capabilities.

The steps in this tutorial are for PMM 3.

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 of PMM 3.

   See PMM upgrade documentation for how you can upgrade to PMM3. Also check the Automatic migration of API keys page.
2. Helm v3.
3. kubectl.
4. PMM 3 Service account token or PMM2 API key.

Configure authentication

PMM3 (recommended)

PMM3 uses Grafana service accounts to control access to PMM server components and resources. To authenticate in PMM server, you need a service account token. Generate a service account and token. Specify the Admin role for the service account.

The token must have the format `glsa_**********_9e35351b`.

Warning

When you create a service account token, you can select its lifetime: it can be either a permanent token that never expires or the one with the expiration date. PMM server cannot rotate service account tokens after they expire. So you must take care of reconfiguring PMM Client in this case.

PMM2

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.

From PMM UI

Generate the PMM API key

From command line

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)
```

Warning

The API key is not rotated.

Install the Victoria Metrics Kubernetes monitoring stack

Quick install

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:
   - `PMM-SERVER-TOKEN` - The PMM Server service account token
   - `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 identifier 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/refs/tags/v0.1.1/vm-operator-k8s-stack/quick-
   install.sh | bash -s -- --api-key <PMM-SERVER-TOKEN> --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/refs/tags/v0.1.1/vm-operator-k8s-stack/quick-install.sh | bash -s --
--api-key <PMM-SERVER-TOKEN> --pmm-server-url <PMM-SERVER-URL> --k8s-cluster-id <UNIQUE-K8s-CLUSTER-IDENTIFIER> --namespace <NAMESPACE> --node-
exporter-enabled
```

Install manually

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 token key with base64.

   Linux

   ```
   $ echo -n <API-key> | base64 --wrap=0
   ```

   macOS

   ```
   $ 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

   ```
   apiVersion: v1
   data:
     api_key: <base-64-encoded-pmm-server-token>
   kind: Secret
   metadata:
     name: pmm-token-vmoperator
     #namespace: default
   type: Opaque
   ```

4. Create the Secrets object using the YAML file you created previously. Replace the `<filename>` placeholder with your value.

   ```
   $ kubectl apply -f pmm-api-vmoperator.yaml -n <namespace>
   ```

5. Check that the secret is created. The following command checks the secret for the resource named `pmm-token-vmoperator` (as defined in the `metadata.name` option in the secrets file). If you defined another resource name, specify your value.

```
$ kubectl get secret pmm-token-vmoperator -n <namespace>
```

Create a ConfigMap to mount for `kube-state-metrics`

The `kube-state-metrics` (KSM) is a simple service that listens to the Kubernetes API server and generates metrics about the state of various objects - Pods, Deployments, Services and Custom Resources.

To define what metrics the `kube-state-metrics` should capture, create the ConfigMap and mount it to a container.

Use the example `configmap.yaml` configuration file to create the ConfigMap.

```
$ kubectl apply -f https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/refs/tags/v0.1.1/vm-operator-k8s-stack/ksm-configmap.yaml -n <namespace>
```

As a result, you have the `customresource-config-ksm` ConfigMap created.

Install the Victoria Metrics Kubernetes monitoring stack

1. Add the dependency repositories of victoria-metrics-k8s-stack chart.

   ```
   $ helm repo add grafana https://grafana.github.io/helm-charts
   $ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
   ```

2. Add the Victoria Metrics Kubernetes monitoring stack repository.

   ```
   $ helm repo add vm https://victoriametrics.github.io/helm-charts/
   ```

3. Update the repositories.

   ```
   $ helm repo update
   ```

4. Install the Victoria Metrics Kubernetes monitoring stack Helm chart. You need to specify the following configuration:

   - the URL to access the PMM server in the `externalVM.write.url` option in the format `<PMM-SERVER-URL>/victoriametrics/api/v1/write`. The URL can contain either the IP address or the hostname of the PMM server.
   - the unique name or an ID of the Kubernetes cluster in the `vmagent.spec.externalLabels.k8s_cluster_id` option. Ensure to set different values if you are sending metrics from multiple Kubernetes clusters to the same PMM Server.
   - the `<namespace>` placeholder with your value. The Namespace must be the same as the Namespace for the Secret and ConfigMap

   ```
   $ helm install vm-k8s vm/victoria-metrics-k8s-stack \
   -f https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/refs/tags/v0.1.1/vm-operator-k8s-stack/values.yaml \
   --set externalVM.write.url=<PMM-SERVER-URL>/victoriametrics/api/v1/write \
   --set vmagent.spec.externalLabels.k8s_cluster_id=<UNIQUE-CLUSTER-IDENTIFIER/NAME> \
   -n <namespace>
   ```

   To illustrate, say your PMM Server URL is `https://pmm-example.com`, the cluster ID is `test-cluster` and the Namespace is `monitoring-system`. Then the command would look like this:

   ```
   $ helm install vm-k8s vm/victoria-metrics-k8s-stack \
   -f https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/refs/tags/v0.1.1/vm-operator-k8s-stack/values.yaml \
   --set externalVM.write.url=https://pmm-example.com/victoriametrics/api/v1/write \
   --set vmagent.spec.externalLabels.k8s_cluster_id=test-cluster \
   -n monitoring-system
   ```

Validate the successful installation

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

Sample output

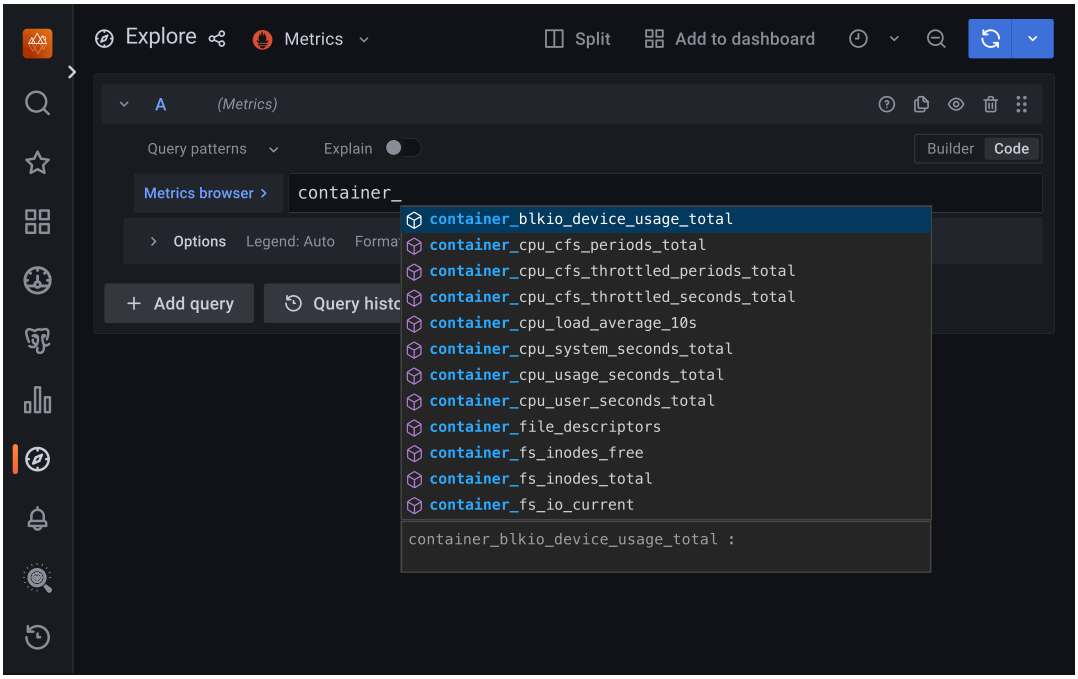
```
vm-k8s-stack-kube-state-metrics-d9d85978d-9pzbs                  1/1     Running   0          28m
vm-k8s-stack-victoria-metrics-operator-844d558455-gvg4n          1/1     Running   0          28m
vmagent-vm-k8s-stack-victoria-metrics-k8s-stack-55fd8fc4fbcxwhx  2/2     Running   0          28m
```

What Pods are running depends on the configuration chosen in values used while installing `victoria-metrics-k8s-stack` chart.

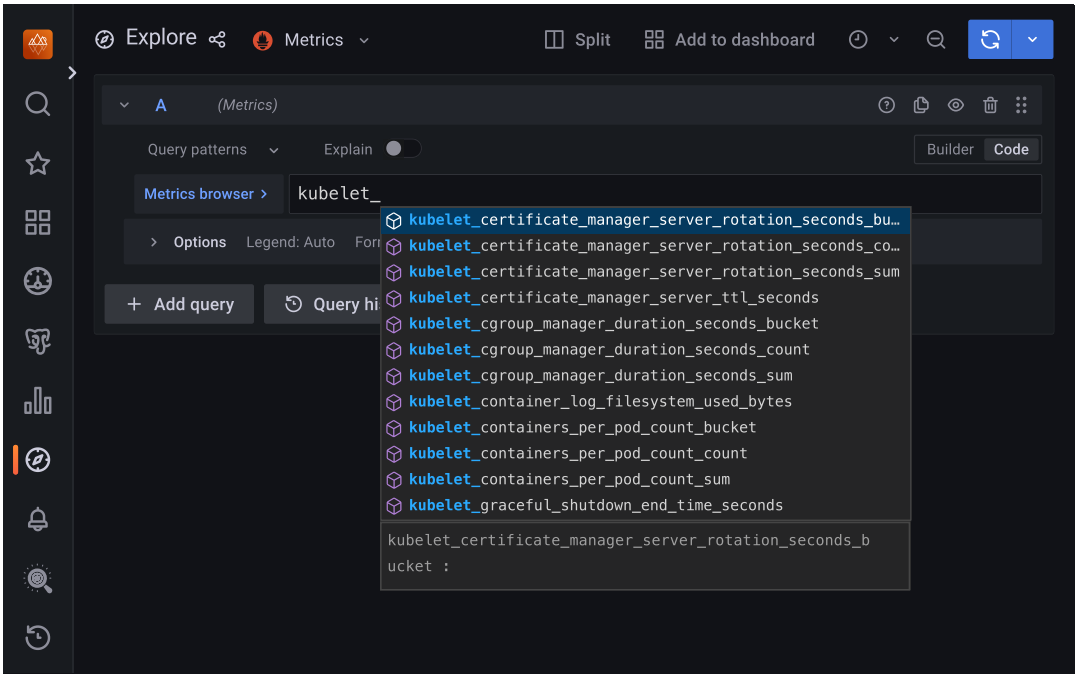
Verify metrics capture

1. Connect to the PMM server.

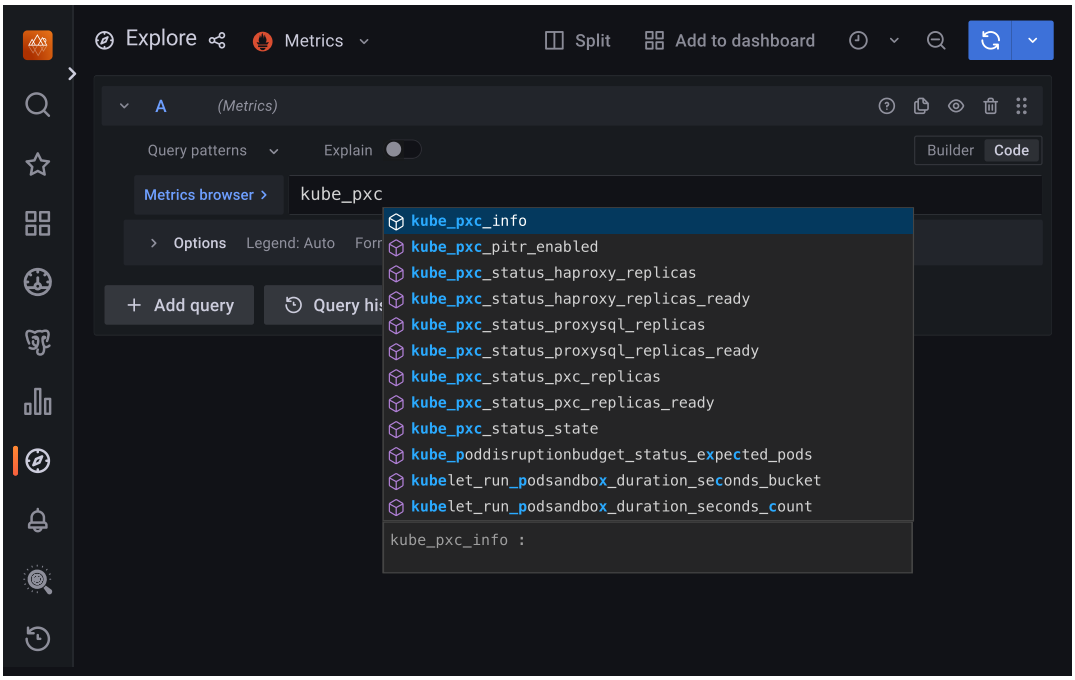
2. Click **Explore** and switch to the **Code** mode.
3. Check that the required metrics are captured, type the following in the Metrics browser dropdown:
   - cadvisor:



   - kubelet:



   - kube-state-metrics metrics that also include Custom resource metrics for the Operator and database deployed in your Kubernetes cluster:



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

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/refs/tags/v0.1.1/vm-operator-k8s-stack/cleanup.sh) --namespace <NAMESPACE>
```

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/refs/tags/v0.1.1/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>
```

Delete Percona Operator for MySQL based on Percona XtraDB Cluster

You may have different reasons to clean up your Kubernetes environment: moving from trial deployment to a production one, testing experimental configurations and the like. In either case, you need to remove some (or all) of these objects:

- Percona XtraDB Cluster managed by the Operator
- Percona Operator for MySQL itself
- Custom Resource Definition deployed with the Operator
- Resources like PVCs and Secrets

Delete the database cluster

To delete the database cluster means to delete the Custom Resource associated with it.

Note

There are 4 finalizers defined in the Custom Resource, which define whether to delete or preserve TLS-related objects and data volumes when the cluster is deleted.

- `finalizers.percona.com/delete-pxc-pods-in-order`: if present, PXC pods are deleted in order on cluster deletion. PVC is not deleted.
- `finalizers.percona.com/delete-ssl`: if present, objects, created for SSL (Secret, certificate, and issuer) are deleted along with the cluster deletion (will not delete PVC).
- `finalizers.percona.com/delete-pxc-pvc`: if present, Persistent Volume Claims for the database cluster Pods are deleted along with the cluster deletion.
- `finalizers.percona.com/delete-proxysql-pvc`: if present, Persistent Volume Claims for ProxySQL Pods are deleted along with the cluster deletion.

The default finalizer in `deploy/cr.yaml` configuration file is `finalizers.percona.com/delete-pxc-pods-in-order`, this allows you to recreate the cluster without losing data, credentials for the system users, etc. You can always delete TLS-related objects and PVCs manually, if needed.

The steps are the following:

1. List the Custom Resources. Replace the `<namespace>` placeholder with your value

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

2. Delete the Custom Resource with the name of your cluster

```
$ kubectl delete pxc <cluster_name> -n <namespace>
```

It may take a while to stop and delete the cluster.

Sample output

```
perconaxtradbcluster.pxc.percona.com "cluster1" deleted
```

3. Check that the cluster is deleted by listing the Custom Resources again:

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

Sample output

```
No resources found in <namespace> namespace.
```

Delete the Operator

Choose the instructions relevant to the way you installed the Operator.

kubectl

To uninstall the Operator, delete the Deployments related to it.

1. List the deployments. Replace the `<namespace>` placeholder with your namespace.

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

2. Delete the `percona-*` deployment

```
$ kubectl delete deploy percona-xtradb-cluster-operator -n <namespace>
```

Sample output

```
deployment.apps "percona-xtradb-cluster-operator" deleted
```

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.
```

4. If you are not just deleting the Operator and XtraDB Cluster from a specific namespace, but want to clean up your entire Kubernetes environment, you can also delete the CustomResourceDefinitions (CRDs).

   Warning: CRDs in Kubernetes are non-namespaced but are available to the whole environment. This means that you shouldn't delete CRDs if you still have the Operator and database cluster in some namespace.

   Get the list of CRDs.

```
$ kubectl get crd
```

5. Delete the `percona*.pxc.percona.com` CRDs

```
$ kubectl delete crd perconaxtradbclusterbackups.pxc.percona.com perconaxtradbclusterrestores.pxc.percona.com
perconaxtradbclusters.pxc.percona.com
```

Sample output

```
customresourcedefinition.apiextensions.k8s.io "perconaxtradbclusterbackups.pxc.percona.com" deleted
customresourcedefinition.apiextensions.k8s.io "perconaxtradbclusterrestores.pxc.percona.com" deleted
customresourcedefinition.apiextensions.k8s.io "perconaxtradbclusters.pxc.percona.com" deleted
```

Helm

To delete the Operator, do the following:

1. List the Helm charts:

```
$ helm list -n <namespace>
```

Sample output

```
cluster1    <namespace>         1           2023-10-31 10:18:10.763049 +0100 CET    deployed    pxc-db-1.19.0         1.19.0
my-op       <namespace>         1           2023-10-31 10:15:18.41444 +0100 CET     deployed    pxc-operator-1.19.0   1.19.0
```

2. Delete the release object for Percona XtraDB Cluster

```
$ helm uninstall cluster1 --namespace <namespace>
```

3. Delete the release object for the Operator

```
$ helm uninstall my-op --namespace <namespace>
```

Clean up resources

By default, TLS-related objects and data volumes remain in Kubernetes environment after you delete the cluster to allow you to recreate it without losing the data.

You can automate resource cleanup by turning on `percona.com/delete-pxc-pvc` and/or `percona.com/delete-ssl` finalizers). You can also delete TLS-related objects and PVCs manually.

To manually clean up resources, do the following:

1. Delete Persistent Volume Claims.

    1. List PVCs. Replace the `<namespace>` placeholder with your namespace:

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

    Sample output

    ```
    NAME                     STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
    datadir-cluster1-pxc-0   Bound    pvc-be4e2398-6fc9-456a-836b-9f0bc36d2a16   6Gi        RWO            standard-rwo   3m57s
    datadir-cluster1-pxc-1   Bound    pvc-8a9ed524-2f79-4ed1-9265-a09947084e08   6Gi        RWO            standard-rwo   2m41s
    datadir-cluster1-pxc-2   Bound    pvc-830fccfb-ced6-4fab-b85a-866aa435a2c7   6Gi        RWO            standard-rwo   91s
    ```

    2. Delete PVCs related to your cluster. The following command deletes PVCs for the `cluster1` cluster:

    ```
    $ kubectl delete pvc datadir-cluster1-pxc-0 datadir-cluster1-pxc-1 datadir-cluster1-pxc-2 -n <namespace>
    ```

    Sample output

    ```
    persistentvolumeclaim "datadir-cluster1-pxc-0" deleted
    persistentvolumeclaim "datadir-cluster1-pxc-1" deleted
    persistentvolumeclaim "datadir-cluster1-pxc-2" deleted
    ```

    Note that it also deletes user secrets if you have enabled the `percona.com/delete-pxc-pvc` finalizer. To prevent it from happening, disable the finalizer.

2. Delete the Secrets

    1. List Secrets:

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

    2. Delete the Secret:

    ```
    $ kubectl delete secret <secret_name> -n <namespace>
    ```

Reference

Custom Resource options reference

Percona Operator for MySQL uses Custom Resources to manage options for the various components of the cluster.

- `PerconaXtraDBCluster` Custom Resource with Percona XtraDB Cluster options,
- `PerconaXtraDBClusterBackup` and `PerconaXtraDBClusterRestore` Custom Resources contain options for Percona XtraBackup used to backup Percona XtraDB Cluster and to restore it from backups.

PerconaXtraDBCluster Custom Resource options

`PerconaXtraDBCluster` Custom Resource contains options for Percona XtraDB Cluster and can be configured via the deploy/cr.yaml configuration file.

The metadata part contains the following keys:

- `name` (`cluster1` by default) sets the name of your Percona XtraDB Cluster; it should include only URL-compatible characters, not exceed 22 characters, start with an alphabetic character, and end with an alphanumeric character;
- `finalizers` subsection:
  - `percona.com/delete-pods-in-order` if present, activates the Finalizer which controls the proper Pods deletion order in case of the cluster deletion event (on by default).
  - `percona.com/delete-pxc-pvc` if present, activates the Finalizer which deletes Persistent Volume Claims for Percona XtraDB Cluster Pods after the cluster deletion event (off by default). It also deletes user Secrets.
  - `percona.com/delete-proxysql-pvc` if present, activates the Finalizer which deletes Persistent Volume Claim for ProxySQL Pod after the cluster deletion event (off by default).
  - `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).

The toplevel spec elements of the deploy/cr.yaml are the following ones:

`allowUnsafeConfigurations`

Prevents users from configuring a cluster with unsafe parameters such as:

- using an invalid number of Percona XtraDB Cluster nodes, which is:
- less than 3
- more than 5
- an even number (for production use, we recommend a minimum of 3 nodes and a maximum of 5 nodes),
- deploying less than 2 ProxySQL or HAProxy Pods,
- starting the cluster without TLS/SSL certificates.

This option is deprecated and will be removed in future releases. Use `unsafeFlags` subsection instead.

Value type	Example
boolean	`false`

`enableCRValidationWebhook`

Enables or disables schema validation before applying `cr.yaml` file (works only in cluster-wide mode due to access restrictions).

Value type	Example
boolean	`true`

`enableVolumeExpansion`

Enables or disables automatic storage scaling / volume expansion.

Value type	Example
boolean	`false`

`pause`

Pause/resume: setting it to `true` gracefully stops the cluster, and setting it to `false` after shut down starts the cluster back.

Value type	Example
boolean	`false`

`secretsName`

A name for users secrets.

Value type	Example
string	`cluster1-secrets`

`crVersion`

Version of the Operator the Custom Resource belongs to.

Value type	Example
string	`1.19.0`

`ignoreAnnotations`

The list of annotations to be ignored by the Operator.

Value type	Example
subdoc	`iam.amazonaws.com/role`

`ignoreLabels`

The list of labels to be ignored by the Operator.

Value type	Example
subdoc	`rack`

`vaultSecretName`

A secret for the HashiCorp Vault to carry on Data at Rest Encryption.

Value type	Example
string	`keyring-secret-vault`

`sslSecretName`

A secret with TLS certificate generated for *external* communications, see Transport Layer Security (TLS) for details.

Value type	Example
string	`cluster1-ssl`

`sslInternalSecretName`

A secret with TLS certificate generated for *internal* communications, see Transport Layer Security (TLS) for details.

Value type	Example
string	`cluster1-ssl-internal`

`logCollectorSecretName`

A secret for the Fluent Bit Log Collector.

Value type	Example
string	`my-log-collector-secrets`

`initImage`

An alternative image for the initial Operator installation. **This option is deprecated and will be removed in future releases**. Use `initContainer.image` instead.

Value type	Example
string	`percona/percona-xtradb-cluster-operator:1.19.0`

`updateStrategy`

A strategy the Operator uses for upgrades.

Value type	Example
string	`SmartUpdate`

Unsafe flags section

The `unsafeFlags` section in the deploy/cr.yaml file contains various configuration options to prevent users from configuring a cluster with unsafe parameters.

`unsafeFlags.tls`

Allows users to configure a cluster without TLS/SSL certificates (if `false`, the Operator will detect unsafe parameters, set cluster status to `error`, and print error message in logs).

Value type	Example
boolean	`false`

`unsafeFlags.pxcSize`

Allows users to configure a cluster with unsafe node counts. For production use, we recommend a minimum of 3 nodes and a maximum of 5 nodes. Even numbers of nodes are not recommended due to quorum voting issues in Galera replication. Higher numbers of nodes (for example, 7) are not recommended due to increased latency from synchronous replication overhead.

Setting this to `true` allows configurations such as:

- Less than 3 nodes
- More than 5 nodes
- Even numbers of nodes

If the option is set to `false`, the Operator detects unsafe parameters, sets clusters status to `error`, and prints error message in logs.

Note: Using unsafe configurations may result in reduced availability, split-brain scenarios, or performance degradation. We cannot guarantee proper operation of the cluster with unsafe node counts.

Value type	Example
boolean	`false`

`unsafeFlags.proxySize`

Allows users to configure a cluster with less than 2 ProxySQL or HAProxy Pods (if `false`, the Operator will detect unsafe parameters, set cluster status to `error`, and print error message in logs).

Value type	Example
boolean	`false`

`unsafeFlags.backupIfUnhealthy`

Allows running a backup even if the cluster status is not `ready`.

Value type	Example
boolean	`false`

initContainer configuration section

The `initContainer` section in the deploy/cr.yaml file allows providing an alternative image with various options for the initial Operator installation.

`initContainer.image`

An alternative image for the initial Operator installation.

Value type	Example
string	`percona/percona-xtradb-cluster-operator:1.19.0`

`initContainer.containerSecurityContext`

A custom Kubernetes Security Context for a Container for the image used for the initial Operator installation.

Value type	Example

subdoc	`privileged: false`
	`runAsUser: 1001`
	`runAsGroup: 1001`

`initContainer.resources.requests.memory`

The Kubernetes memory requests for an image used while the initial Operator installation.

Value type	Example

string	`1G`

`initContainer.resources.requests.cpu`

Kubernetes CPU requests for an image used while the initial Operator installation.

Value type	Example
string	`600m`

`initContainer.resources.limits.memory`

Kubernetes memory limits for an image used while the initial Operator installation.

Value type	Example
string	`1G`

`initContainer.resources.limits.cpu`

Kubernetes CPU limits for an image used while the initial Operator installation.

Value type	Example
string	`1`

TLS (extended cert-manager configuration section)

The `tls` section in the deploy/cr.yaml file contains various configuration options for additional customization of the TLS cert-manager.

`tls.enabled`

Enables or disables the TLS encryption. If set to `false`, it also requires setting `unsafeFlags.tls option to` true`.

Value type	Example
boolean	`true`

`tls.certValidityDuration`

Validity period for TLS certificates. Minimum required validity is 1 hour. Durations lower than 1 hour are rejected. Setting the duration to exactly 1 hour prevents the Operator from generating the correct certificate object.

Value type	Example
string	`2160h`

`tls.caValidityDuration`

Validity period for CA certificate. Minimum accepted duration is 730 hours (approximately 30 days). Setting this value to exactly 730 hours prevents the Operator from generating the correct certificate object. You must set this value greater than 730 hours.

Value type	Example
string	`26280h`

`tls.SANs`

Additional domains (SAN) to be added to the TLS certificate within the extended cert-manager configuration.

Value type	Example
subdoc	

`tls.issuerConf.name`

A cert-manager issuer name.

Value type	Example
string	`special-selfsigned-issuer`

`tls.issuerConf.kind`

A cert-manager issuer type.

Value type	Example
string	`ClusterIssuer`

`tls.issuerConf.group`

A cert-manager issuer group. Should be `cert-manager.io` for built-in cert-manager certificate issuers.

Value type	Example
string	`cert-manager.io`

Upgrade options section

The `upgradeOptions` section in the deploy/cr.yaml file contains various configuration options to control Percona XtraDB Cluster upgrades.

`upgradeOptions.versionServiceEndpoint`

The Version Service URL used to check versions compatibility for upgrade.

Value type	Example
string	`https://check.percona.com`

`upgradeOptions.apply`

Specifies how updates are processed by the Operator. `Never` or `Disabled` will completely disable automatic upgrades, otherwise it can be set to `Latest` or `Recommended` or to a specific version string of Percona XtraDB Cluster (e.g. `8.0.19-10.1`) that is wished to be version-locked (so that the user can control the version running, but use automatic upgrades to move between them).

Value type	Example
string	`Disabled`

`upgradeOptions.schedule`

Scheduled time to check for updates, specified in the crontab format.

Value type	Example
string	`0 2 \* \* \*`

PXC section

The `pxc` section in the deploy/cr.yaml file contains general configuration options for the Percona XtraDB Cluster.

`pxc.size`

The size of the Percona XtraDB cluster must be 3 or 5 for High Availability. Other values are allowed if the `spec.unsafeFlags.pxcSize` key is set to true.

Value type	Example
int	`3`

`pxc.image`

The Docker image of the Percona cluster used (actual image names for Percona XtraDB Cluster 8.0 and Percona XtraDB Cluster 5.7 can be found in the list of certified images).

Value type	Example
string	`percona/percona-xtradb-cluster:8.0.44-35.1`

`pxc.autoRecovery`

Turns Automatic Crash Recovery on or off.

Value type	Example
boolean	`true`

`pxc.expose.enabled`

Enable or disable exposing Percona XtraDB Cluster instances with dedicated IP addresses.

Value type	Example
boolean	`true`

`pxc.expose.type`

The Kubernetes Service Type used for exposure.

Value type	Example
string	`LoadBalancer`

`pxc.expose.loadBalancerClass`

Define the implementation of the load balancer you want to use. This setting enables you to select a custom or specific load balancer class instead of the default one provided by the cloud provider.

Value type	Example

string	`"eks.amazonaws.com/nlb"`

`pxc.expose.trafficPolicy`

Specifies whether Service should route external traffic to cluster-wide or node-local endpoints (it can influence the load balancing effectiveness) **This option is deprecated and will be removed in future releases**. Use `pxc.expose.externalTrafficPolicy` instead.

Value type	Example
string	`Local`

`pxc.expose.externalTrafficPolicy`

Specifies whether Service for Percona XtraDB Cluster should route external traffic to cluster-wide or to node-local endpoints (it can influence the load balancing effectiveness).

Value type	Example
string	`Local`

`pxc.expose.internalTrafficPolicy`

Specifies whether Service for Percona XtraDB Cluster should route internal traffic to cluster-wide or to node-local endpoints (it can influence the load balancing effectiveness).

Value type	Example
string	`Local`

`pxc.expose.loadBalancerSourceRanges`

The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations).

Value type	Example
string	`10.0.0.0/8`

`pxc.expose.loadBalancerIP`

The static IP-address for the load balancer. **This field is deprecated and scheduled for removal in version 1.21.0.**.

`loadBalancerIP` has been officially deprecated upstream in Kubernetes due to its inconsistent behavior across cloud providers and lack of dual-stack support. As a result, its usage is strongly discouraged.

We recommend using cloud provider-specific annotations instead, as they offer more predictable and portable behavior for managing load balancer IP assignments.

Value type	Example
string	`127.0.0.1`

`pxc.expose.annotations`

The Kubernetes annotations.

Value type	Example
string	`networking.gke.io/load-balancer-type: "Internal"`

`pxc.replicationChannels.name`

Name of the replication channel for cross-site replication.

Value type	Example
S string	`pxc1_to_pxc2`

`pxc.replicationChannels.isSource`

Should the cluster act as Source (`true`) or Replica (`false`) in cross-site replication.

Value type	Example
boolean	`false`

`pxc.replicationChannels.configuration.sourceRetryCount`

Number of retries Replica should do when the existing connection source fails.

Value type	Example
1 int	`3`

`pxc.replicationChannels.configuration.sourceConnectRetry`

The interval between reconnection attempts in seconds to be used by Replica when the the existing connection source fails.

Value type	Example
1 int	`60`

`pxc.replicationChannels.configuration.ssl`

Turns SSL for replication channels on or off.

Value type	Example
boolean	`false`

`pxc.replicationChannels.configuration.sslSkipVerify`

Turns the host name identity verification for SSL-based replication on or off.

Value type	Example
boolean	`true`

`pxc.replicationChannels.configuration.ca`

The path name of the Certificate Authority (CA) certificate file to be used if the SSL for replication channels is turned on.

Value type	Example
S string	`/etc/mysql/ssl/ca.crt`

`pxc.replicationChannels.sourcesList.host`

For the cross-site replication Replica cluster, this key should contain the hostname or IP address of the Source cluster.

Value type	Example
S string	`10.95.251.101`

`pxc.replicationChannels.sourcesList.port`

For the cross-site replication Replica cluster, this key should contain the Source port number.

Value type	Example
1 int	`3306`

`pxc.replicationChannels.sourcesList.weight`

For the cross-site replication Replica cluster, this key should contain the Source cluster weight (varies from `1` to `100`, the cluster with the higher number will be selected as the replication source first).

Value type	Example
1 int	`100`

`pxc.readinessDelaySec`

Adds a delay before a run check to verify the application is ready to process traffic.

This field is deprecated and will be removed in version 1.22.0. Use the `pxc.readinessProbes.initialDelaySeconds` option instead.

Value type	Example
1 int	`15`

`pxc.livenessDelaySec`

Adds a delay before the run check ensures the application is healthy and capable of processing requests.

This field is deprecated and will be removed in version 1.22.0. Use the `pxc.livenessProbes.initialDelaySeconds` option instead.

Value type	Example
1 int	`300`

`pxc.configuration`

The `my.cnf` file options to be passed to Percona XtraDB cluster nodes.

Value type	Example

S string	`|`
	`[mysqld]`
	`wsrep_debug=ON`
	`wsrep-provider_options=gcache.size=1G;gcache.recover=yes`

`pxc.imagePullSecrets.name`

The Kubernetes ImagePullSecret.

Value type	Example
S string	`private-registry-credentials`

`pxc.priorityClassName`

The Kubernetes Pod priority class.

Value type	Example
S string	`high-priority`

`pxc.schedulerName`

The Kubernetes Scheduler.

Value type	Example
S string	`mycustom-scheduler`

`pxc.annotations`

The Kubernetes annotations.

Value type	Example
label	`iam.amazonaws.com/role: role-arn`

`pxc.labels`

Labels are key-value pairs attached to objects.

Value type	Example
label	`rack: rack-22`

`pxc.readinessProbes.initialDelaySeconds`

Number of seconds to wait before performing the first readiness probe.

Value type	Example
1 int	`15`

`pxc.readinessProbes.timeoutSeconds`

Number of seconds after which the readiness probe times out.

Value type	Example
1 int	`15`

`pxc.readinessProbes.periodSeconds`

How often (in seconds) to perform the readiness probe.

Value type	Example
1 int	`30`

`pxc.readinessProbes.successThreshold`

Minimum consecutive successes for the readiness probe to be considered successful after having failed.

Value type	Example
1 int	`1`

`pxc.readinessProbes.failureThreshold`

When the readiness probe fails, Kubernetes will try this number of times before marking the Pod Unready.

Value type	Example
1 int	`5`

`pxc.livenessProbes.initialDelaySeconds`

Number of seconds to wait before performing the first liveness probe.

Value type	Example
1 int	`300`

`pxc.livenessProbes.timeoutSeconds`

Number of seconds after which the liveness probe times out.

Value type	Example
1 int	`5`

`pxc.livenessProbes.periodSeconds`

How often (in seconds) to perform the liveness probe.

Value type	Example
1 int	`10`

`pxc.livenessProbes.successThreshold`

Minimum consecutive successes for the liveness probe to be considered successful after having failed.

Value type	Example
1 int	`1`

`pxc.livenessProbes.failureThreshold`

When the liveness probe fails, Kubernetes will try this number of times before restarting the container.

Value type	Example
1 int	`3`

`pxc.envVarsSecret`

A secret with environment variables, see Define environment variables for details.

Value type	Example
S string	`my-env-var-secrets`

`pxc.mysqlAllocator`

Specifies which memory allocator to use for the MySQL process. Available since Operator version 1.19.0 for Percona XtraDB Cluster 8.0 and later.

Supported values: `jemalloc`, `tcmalloc`. When left empty or omitted, the default `libc` allocator is used.

Value type	Example
S string	`jemalloc`

Warning

If you have `LD_PRELOAD` set in a Secret referenced by `pxc.envVarsSecret`, that value takes precedence over the `mysqlAllocator` option. The Operator checks for `LD_PRELOAD` in the Secret first, and if found, uses that value regardless of the `mysqlAllocator` setting.

`pxc.resources.requests.memory`

The Kubernetes memory requests for a Percona XtraDB Cluster container.

Value type	Example
S string	`1G`

`pxc.resources.requests.cpu`

Kubernetes CPU requests for a Percona XtraDB Cluster container.

Value type	Example
S string	`600m`

`pxc.resources.requests.ephemeral-storage`

Kubernetes Ephemeral Storage requests for a Percona XtraDB Cluster container.

Value type	Example
S string	`1G`

`pxc.resources.limits.memory`

Kubernetes memory limits for a Percona XtraDB Cluster container.

Value type	Example

string	`1G`

`pxc.resources.limits.cpu`

Kubernetes CPU limits for a Percona XtraDB Cluster container.

Value type	Example
string	`1`

`pxc.resources.limits.ephemeral-storage`

Kubernetes Ephemeral Storage limits for a Percona XtraDB Cluster container.

Value type	Example
string	`1G`

`pxc.nodeSelector`

Kubernetes nodeSelector.

Value type	Example
label	`disktype: ssd`

`pxc.topologySpreadConstraints.labelSelector.matchLabels`

The Label selector for the Kubernetes Pod Topology Spread Constraints.

Value type	Example
label	`app.kubernetes.io/name: percona-xtradb-cluster-operator`

`pxc.topologySpreadConstraints.maxSkew`

The degree to which Pods may be unevenly distributed under the Kubernetes Pod Topology Spread Constraints.

Value type	Example
int	1

`pxc.topologySpreadConstraints.topologyKey`

The key of node labels for the Kubernetes Pod Topology Spread Constraints.

Value type	Example
string	`kubernetes.io/hostname`

`pxc.topologySpreadConstraints.whenUnsatisfiable`

What to do with a Pod if it doesn't satisfy the Kubernetes Pod Topology Spread Constraints.

Value type	Example

string	`DoNotSchedule`

`pxc.affinity.topologyKey`

The Operator topology key node anti-affinity constraint.

Value type	Example
string	`kubernetes.io/hostname`

`pxc.affinity.advanced`

In cases where the Pods require complex tuning the advanced option turns off the `topologyKey` effect. This setting allows the standard Kubernetes affinity constraints of any complexity to be used.

Value type	Example
subdoc	

`pxc.tolerations`

Kubernetes Pod tolerations.

Value type	Example
subdoc	`node.alpha.kubernetes.io/unreachable`

`pxc.podDisruptionBudget.maxUnavailable`

The Kubernetes podDisruptionBudget specifies the number of Pods from the set unavailable after the eviction.

Value type	Example
int	`1`

`pxc.podDisruptionBudget.minAvailable`

The Kubernetes podDisruptionBudget Pods that must be available after an eviction.

Value type	Example
int	`0`

`pxc.volumeSpec.emptyDir`

The Kubernetes emptyDir volume The directory created on a node and accessible to the Percona XtraDB Cluster Pod containers.

Value type	Example
string	`{}`

`pxc.volumeSpec.hostPath.path`

Kubernetes hostPath The volume that mounts a directory from the host node's filesystem into your Pod. The path property is required.

Value type	Example

string	`/data`

`pxc.volumeSpec.hostPath.type`

The Kubernetes hostPath. An optional property for the hostPath.

Value type	Example
string	`Directory`

`pxc.volumeSpec.persistentVolumeClaim.storageClassName`

Set the Kubernetes storage class to use with the Percona XtraDB Cluster PersistentVolumeClaim.

Value type	Example
string	`standard`

`pxc.volumeSpec.persistentVolumeClaim.accessModes`

The Kubernetes PersistentVolumeClaim access modes for the Percona XtraDB cluster.

Value type	Example
array	`[ReadWriteOnce]`

`pxc.volumeSpec.persistentVolumeClaim.dataSource.name`

The name of PVC used as a data source to create the Percona XtraDB Cluster Volumes by cloning.

Value type	Example
string	`new-snapshot-test`

`pxc.volumeSpec.persistentVolumeClaim.dataSource.kind`

The Kubernetes DataSource type.

Value type	Example
string	`VolumeSnapshot`

`pxc.volumeSpec.persistentVolumeClaim.dataSource.apiGroup`

The Kubernetes API group to use for PVC Data Source.

Value type	Example
string	`snapshot.storage.k8s.io`

`pxc.extraPVCs.name`

The name of the volume provisioned for the external PersistentVolumeClaim that you mount to Percona XtraDB Cluster pods. In such a way you can attach pre-existing storage volumes to your database instances for use cases such as importing data, sharing configuration files, or accessing external datasets. For more information, see Add external PersistentVolumeClaims to the Operator. You can configure external PVCs for both new and running clusters.

Value type	Example
string	`shared-data`

`pxc.extraPVCs.claimName`

The name of the existing PersistentVolumeClaim to mount. This PVC must exist in the same namespace as your cluster before you apply the configuration. The Operator will mount the existing PVC to all PXC pods.

Value type	Example
string	`my-existing-pvc`

`pxc.extraPVCs.mountPath`

The path inside the container where the volume will be mounted.

Value type	Example
string	`/mnt/shared-data`

`pxc.extraPVCs.subPath`

An optional folder within the volume to mount. If not specified, the volume's root is mounted.

Value type	Example
string	`config`

`pxc.extraPVCs.readOnly`

Optional read-only flag. If set to `true`, the volume will be mounted as read-only. Defaults to `false`.

Value type	Example
boolean	`false`

`pxc.gracePeriod`

The Kubernetes grace period when terminating a Pod.

Value type	Example
int	`600`

`pxc.containerSecurityContext`

A custom Kubernetes Security Context for a Container to be used instead of the default one.

Value type	Example
subdoc	`privileged: true`

`pxc.podSecurityContext`

A custom Kubernetes Security Context for a Pod to be used instead of the default one.

Value type	Example
subdoc	`fsGroup: 1001` `supplementalGroups: [1001, 1002, 1003]`

`pxc.serviceAccountName`

The Kubernetes Service Account for Percona XtraDB Cluster Pods.

Value type	Example
string	`percona-xtradb-cluster-operator-workload`

`pxc.imagePullPolicy`

The policy used to update images.

Value type	Example
string	`Always`

`pxc.runtimeClassName`

Name of the Kubernetes Runtime Class for Percona XtraDB Cluster Pods.

Value type	Example
string	`image-rc`

`pxc.sidecars.image`

Image for the custom sidecar container for Percona XtraDB Cluster Pods.

Value type	Example
string	`busybox`

`pxc.sidecars.command`

Command for the custom sidecar container for Percona XtraDB Cluster Pods.

Value type	Example
array	`["/bin/sh"]`

`pxc.sidecars.args`

Command arguments for the custom sidecar container for Percona XtraDB Cluster Pods.

Value type	Example
array	`["-c", "while true; do trap 'exit 0' SIGINT SIGTERM SIGQUIT SIGKILL; done;"]`

`pxc.sidecars.name`

Name of the custom sidecar container for Percona XtraDB Cluster Pods.

Value type	Example
string	`my-sidecar-1`

`pxc.sidecars.resources.requests.memory`

The Kubernetes memory requests for a Percona XtraDB Cluster sidecar container.

Value type	Example
string	`1G`

`pxc.sidecars.resources.requests.cpu`

Kubernetes CPU requests for a Percona XtraDB Cluster sidecar container.

Value type	Example
string	`500m`

`pxc.sidecars.resources.limits.memory`

Kubernetes memory limits for a Percona XtraDB Cluster sidecar container.

Value type	Example
string	`2G`

`pxc.sidecars.resources.limits.cpu`

Kubernetes CPU limits for a Percona XtraDB Cluster sidecar container.

Value type	Example
string	`600m`

`pxc.lifecycle.preStop.exec.command`

Command for the preStop lifecycle hook for Percona XtraDB Cluster Pods.

Value type	Example
array	`["/bin/true"]`

`pxc.lifecycle.postStart.exec.command`

Command for the postStart lifecycle hook for Percona XtraDB Cluster Pods.

Value type	Example
array	`["/bin/true"]`

HAProxy section

The `haproxy` section in the deploy/cr.yaml file contains configuration options for the HAProxy service.

`haproxy.enabled`

Enables or disables load balancing with HAProxy Services.

Value type	Example
boolean	`true`

`haproxy.size`

The number of the HAProxy Pods to provide load balancing. It should be 2 or more unless the `spec.unsafeFlags.proxySize` key is set to true.

Value type	Example
int	`2`

`haproxy.image`

HAProxy Docker image to use.

Value type	Example
string	`percona/percona-xtradb-cluster-operator:1.19.0-haproxy`

`haproxy.imagePullPolicy`

The policy used to update images.

Value type	Example
string	`Always`

`haproxy.imagePullSecrets.name`

The Kubernetes imagePullSecrets for the HAProxy image.

Value type	Example
string	`private-registry-credentials`

`haproxy.readinessDelaySec`

Adds a delay before a run check to verify the application is ready to process traffic.

This field is deprecated and will be removed in version 1.22.0. Use the `haproxy.readinessProbes.initialDelaySeconds` option instead.

Value type	Example
int	`15`

`haproxy.livenessDelaySec`

Adds a delay before the run check ensures the application is healthy and capable of processing requests.

This field is deprecated and will be removed in version 1.22.0. Use the `haproxy.livenessProbes.initialDelaySeconds` option instead.

Value type	Example

int	`300`

`haproxy.configuration`

The custom HAProxy configuration file contents.

Value type	Example
string	

`haproxy.annotations`

The Kubernetes annotations metadata.

Value type	Example
label	`iam.amazonaws.com/role: role-arn`

`haproxy.labels`

Labels are key-value pairs attached to objects.

Value type	Example
label	`rack: rack-22`

`haproxy.readinessProbes.initialDelaySeconds`

Number of seconds to wait before performing the first readiness probe.

Value type	Example
int	`15`

`haproxy.readinessProbes.timeoutSeconds`

Number of seconds after which the readiness probe times out.

Value type	Example
int	`1`

`haproxy.readinessProbes.periodSeconds`

How often (in seconds) to perform the readiness probe.

Value type	Example
int	`5`

`haproxy.readinessProbes.successThreshold`

Minimum consecutive successes for the readiness probe to be considered successful after having failed.

Value type	Example

1 int	`1`

`haproxy.readinessProbes.failureThreshold`

When the readiness probe fails, Kubernetes will try this number of times before marking the Pod Unready.

Value type	Example
1 int	`3`

`haproxy.serviceType`

Specifies the type of Kubernetes Service to be used for HAProxy. **This option is deprecated and will be removed in future releases**. Use `haproxy.exposePrimary.type` instead.

Value type	Example
S string	`ClusterIP`

`haproxy.externalTrafficPolicy`

Specifies whether Service for HAProxy should route external traffic to cluster-wide or to node-local endpoints (it can influence the load balancing effectiveness). **This option is deprecated and will be removed in future releases**. Use `haproxy.exposePrimary.externalTrafficPolicy` instead.

Value type	Example
S string	`Cluster`

`haproxy.livenessProbes.initialDelaySeconds`

Number of seconds to wait before performing the first liveness probe.

Value type	Example
1 int	`60`

`haproxy.livenessProbes.timeoutSeconds`

Number of seconds after which the liveness probe times out.

Value type	Example
1 int	`5`

`haproxy.livenessProbes.periodSeconds`

How often (in seconds) to perform the liveness probe.

Value type	Example
1 int	`30`

`haproxy.livenessProbes.successThreshold`

Minimum consecutive successes for the liveness probe to be considered successful after having failed.

Value type	Example
1 int	`1`

`haproxy.resources.requests.memory`

The Kubernetes memory requests for the main HAProxy container.

Value type	Example
S string	`1G`

`haproxy.resources.requests.cpu`

Kubernetes CPU requests for the main HAProxy container.

Value type	Example
S string	`600m`

`haproxy.resources.limits.memory`

Kubernetes memory limits for the main HAProxy container.

Value type	Example
S string	`1G`

`haproxy.resources.limits.cpu`

Kubernetes CPU limits for the main HAProxy container.

Value type	Example
S string	`700m`

`haproxy.envVarsSecret`

A secret with environment variables, see Define environment variables for details.

Value type	Example
S string	`my-env-var-secrets`

`haproxy.priorityClassName`

The Kubernetes Pod Priority class for HAProxy.

Value type	Example
S string	`high-priority`

`haproxy.schedulerName`

The Kubernetes Scheduler.

Value type	Example
S string	`mycustom-scheduler`

`haproxy.nodeSelector`

Kubernetes nodeSelector.

Value type	Example
label	`disktype: ssd`

`haproxy.sidecarResources.requests.memory`

The Kubernetes memory requests for default sidecar containers such as `haproxy-monit` running inside HAProxy Pods.

Value type	Example
S string	`1G`

`haproxy.sidecarResources.requests.cpu`

Kubernetes CPU requests for default sidecar containers such as `haproxy-monit` running inside HAProxy Pods.

Value type	Example
S string	`500m`

`haproxy.sidecarResources.limits.memory`

Kubernetes memory limits for default sidecar containers such as `haproxy-monit` running inside HAProxy Pods.

Value type	Example
S string	`2G`

`haproxy.sidecarResources.limits.cpu`

Kubernetes CPU limits for default sidecar containers such as `haproxy-monit` running inside HAProxy Pods.

Value type	Example
S string	`600m`

`haproxy.topologySpreadConstraints.labelSelector.matchLabels`

The Label selector for the Kubernetes Pod Topology Spread Constraints.

Value type	Example
label	`app.kubernetes.io/name: percona-xtradb-cluster-operator`

`haproxy.topologySpreadConstraints.maxSkew`

The degree to which Pods may be unevenly distributed under the Kubernetes Pod Topology Spread Constraints.

Value type	Example
1 int	1

`haproxy.topologySpreadConstraints.topologyKey`

The key of node labels for the Kubernetes Pod Topology Spread Constraints.

Value type	Example
S string	`kubernetes.io/hostname`

`haproxy.topologySpreadConstraints.whenUnsatisfiable`

What to do with a Pod if it doesn't satisfy the Kubernetes Pod Topology Spread Constraints.

Value type	Example
S string	`DoNotSchedule`

`haproxy.affinity.topologyKey`

The Operator topology key node anti-affinity constraint.

Value type	Example
S string	`kubernetes.io/hostname`

`haproxy.affinity.advanced`

If available it makes a topologyKey node affinity constraint to be ignored.

Value type	Example
≡ subdoc	

`haproxy.tolerations`

Kubernetes Pod tolerations.

Value type	Example
≡ subdoc	`node.alpha.kubernetes.io/unreachable`

`haproxy.extraPVCs.name`

The name of the volume provisioned for the external PersistentVolumeClaim that you mount to HAProxy pods. In such a way you can attach pre-existing storage volumes to your database instances for use cases such as importing data, sharing configuration files, or accessing external datasets. For more information, see Add external PersistentVolumeClaims to the Operator. You can configure external PVCs for both new and running clusters.

Value type	Example
S string	`shared-data`

`haproxy.extraPVCs.claimName`

The name of the existing PersistentVolumeClaim to mount. This PVC must exist in the same namespace as your cluster before you apply the configuration. The Operator will mount the existing PVC to all HAProxy pods.

Value type	Example
S string	`my-existing-pvc`

`haproxy.extraPVCs.mountPath`

The path inside the container where the volume will be mounted.

Value type	Example
S string	`/mnt/shared-data`

`haproxy.extraPVCs.subPath`

An optional folder within the volume to mount. If not specified, the volume's root is mounted.

Value type	Example
S string	`config`

`haproxy.extraPVCs.readOnly`

Optional read-only flag. If set to `true`, the volume will be mounted as read-only. Defaults to `false`.

Value type	Example
boolean	`false`

`haproxy.podDisruptionBudget.maxUnavailable`

The Kubernetes podDisruptionBudget specifies the number of Pods from the set unavailable after the eviction.

Value type	Example
1 int	`1`

`haproxy.podDisruptionBudget.minAvailable`

The Kubernetes podDisruptionBudget Pods that must be available after an eviction.

Value type	Example
1 int	`0`

`haproxy.gracePeriod`

The Kubernetes grace period when terminating a Pod.

Value type	Example
1 int	`30`

`haproxy.exposePrimary.enabled`

Enables or disables the HAProxy primary instance Service. This field is deprecated starting with the Operator version 1.17.0.

Value type	Example
boolean	`false`

`haproxy.exposePrimary.type`

Specifies the type of Kubernetes Service to be used for HAProxy primary instance Service.

Value type	Example
string	`ClusterIP`

`haproxy.exposePrimary.loadBalancerClass`

Define the implementation of the load balancer you want to use. This setting enables you to select a custom or specific load balancer class instead of the default one provided by the cloud provider.

Value type	Example
string	`"eks.amazonaws.com/nlb"`

`haproxy.exposePrimary.externalTrafficPolicy`

Specifies whether Service for HAProxy should route external traffic to cluster-wide or to node-local endpoints (it can influence the load balancing effectiveness).

Value type	Example
string	`Cluster`

`haproxy.exposePrimary.internalTrafficPolicy`

Specifies whether Service for HAProxy primary instance should route internal traffic to cluster-wide or to node-local endpoints (it can influence the load balancing effectiveness).

Value type	Example
string	`Cluster`

`haproxy.exposePrimary.loadBalancerSourceRanges`

The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations).

Value type	Example
string	`10.0.0.0/8`

`haproxy.exposePrimary.loadBalancerIP`

The static IP-address for the load balancer. **This field is deprecated and scheduled for removal in version 1.21.0.**.

`loadBalancerIP` has been officially deprecated upstream in Kubernetes due to its inconsistent behavior across cloud providers and lack of dual-stack support. As a result, its usage is strongly discouraged.

We recommend using cloud provider-specific annotations instead, as they offer more predictable and portable behavior for managing load balancer IP assignments.

Value type	Example
string	`127.0.0.1`

`haproxy.serviceLabels`

The Kubernetes labels for the load balancer Service. **This option is deprecated and will be removed in future releases**. Use `haproxy.exposePrimary.labels` instead.

Value type	Example
label	`rack: rack-22`

`haproxy.exposePrimary.labels`

The Kubernetes labels for the load balancer Service.

Value type	Example
label	`rack: rack-22`

`haproxy.serviceAnnotations`

The Kubernetes annotations metadata for the load balancer Service. **This option is deprecated and will be removed in future releases**. Use `haproxy.exposePrimary.annotations` instead.

Value type	Example
string	`service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp`

`haproxy.exposePrimary.annotations`

The Kubernetes annotations metadata for the load balancer Service.

Value type	Example
string	`service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp`

`haproxy.replicasServiceEnabled`

Enables or disables `haproxy-replicas` Service. This Service (on by default) forwards requests to all Percona XtraDB Cluster instances, and it *should not be used for write requests*! **This option is deprecated and will be removed in future releases**. Use `haproxy.exposeReplicas.enabled` instead.

Value type	Example
boolean	`false`

`haproxy.exposeReplicas.enabled`

Enables or disables `haproxy-replicas` Service. This Service default forwards requests to all Percona XtraDB Cluster instances, and it **should not be used for write requests**!

Value type	Example
boolean	`true`

`haproxy.exposeReplicas.onlyReaders`

Setting it to `true` excludes current MySQL primary instance (writer) from the list of Pods, to which `haproxy-replicas` Service directs connections, leaving only the reader instances.

Value type	Example
boolean	`false`

`haproxy.exposeReplicas.loadBalancerSourceRanges`

The range of client IP addresses from which the load balancer should be reachable (if not set, no limitations).

Value type	Example
string	`10.0.0.0/8`

`haproxy.exposeReplicas.loadBalancerIP`

The static IP-address for the replicas load balancer. **This field is deprecated and scheduled for removal in version 1.21.0.**.

`loadBalancerIP` has been officially deprecated upstream in Kubernetes due to its inconsistent behavior across cloud providers and lack of dual-stack support. As a result, its usage is strongly discouraged.

We recommend using cloud provider-specific annotations instead, as they offer more predictable and portable behavior for managing load balancer IP assignments.

Value type	Example
string	`127.0.0.1`

`haproxy.exposeReplicas.type`

Specifies the type of Kubernetes Service to be used for HAProxy replicas.

Value type	Example
string	`ClusterIP`

`haproxy.exposeReplicas.loadBalancerClass`

Define the implementation of the load balancer you want to use. This setting enables you to select a custom or specific load balancer class instead of the default one provided by the cloud provider.

Value type	Example
string	`"eks.amazonaws.com/nlb"`

`haproxy.replicasExternalTrafficPolicy`

Specifies whether Service for HAProxy replicas should route external traffic to cluster-wide or to node-local endpoints (it can influence the load balancing effectiveness). **This option is deprecated and will be removed in future releases**. Use `haproxy.exposeReplicas.externalTrafficPolicy` instead.

Value type	Example
string	`Cluster`

`haproxy.exposeReplicas.externalTrafficPolicy`

Specifies whether Service for HAProxy replicas should route external traffic to cluster-wide or to node-local endpoints (it can influence the load balancing effectiveness).

Value type	Example
S string	`Cluster`

`haproxy.exposeReplicas.internalTrafficPolicy`

Specifies whether Service for HAProxy replicas should route internal traffic to cluster-wide or to node-local endpoints (it can influence the load balancing effectiveness).

Value type	Example
S string	`Cluster`

`haproxy.exposeReplicas.labels`

The Kubernetes labels for the `haproxy-replicas` Service.

Value type	Example
label	`rack: rack-22`

`haproxy.exposeReplicas.annotations`

The Kubernetes annotations metadata for the `haproxy-replicas` Service.

Value type	Example
S string	`service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp`

`haproxy.healthCheck.interval`

Interval in milliseconds between HAProxy health checks. The minimum value is 1000 milliseconds.

Value type	Example
1 int	`10000`

`haproxy.healthCheck.fall`

Number of consecutive failed health checks before HAProxy marks a server as down. The minimum value is 1.

Value type	Example
1 int	`2`

`haproxy.healthCheck.rise`

Number of consecutive successful health checks before HAProxy marks a server as up and running. The minimum value is 1.

Value type	Example
1 int	`1`

`haproxy.containerSecurityContext`

A custom Kubernetes Security Context for a Container to be used instead of the default one.

Value type	Example
subdoc	`privileged: true`

`haproxy.podSecurityContext`

A custom Kubernetes Security Context for a Pod to be used instead of the default one.

Value type	Example
subdoc	`fsGroup: 1001` `supplementalGroups: [1001, 1002, 1003]`

`haproxy.serviceAccountName`

The Kubernetes Service Account for the HAProxy Pod.

Value type	Example
string	`percona-xtradb-cluster-operator-workload`

`haproxy.runtimeClassName`

Name of the Kubernetes Runtime Class for the HAProxy Pod.

Value type	Example
string	`image-rc`

`haproxy.sidecars.image`

Image for the custom sidecar container for the HAProxy Pod.

Value type	Example
string	`busybox`

`haproxy.sidecars.command`

Command for the custom sidecar container for the HAProxy Pod.

Value type	Example
array	`["/bin/sh"]`

`haproxy.sidecars.args`

Command arguments for the custom sidecar container for the HAProxy Pod.

Value type	Example
array	`["-c", "while true; do trap 'exit 0' SIGINT SIGTERM SIGQUIT SIGKILL; done;"]`

`haproxy.sidecars.name`

Name of the custom sidecar container for the HAProxy Pod.

Value type	Example
S string	`my-sidecar-1`

`haproxy.sidecars.resources.requests.memory`

The Kubernetes memory requests for custom sidecar containers running inside the HAProxy Pod.

Value type	Example
S string	`1G`

`haproxy.sidecars.resources.requests.cpu`

Kubernetes CPU requests for custom sidecar containers running inside the HAProxy Pod.

Value type	Example
S string	`500m`

`haproxy.sidecars.resources.limits.memory`

Kubernetes memory limits for custom sidecar containers running inside the HAProxy Pod.

Value type	Example
S string	`2G`

`haproxy.sidecars.resources.limits.cpu`

Kubernetes CPU limits for custom sidecar containers running inside the HAProxy Pod.

Value type	Example
S string	`600m`

`haproxy.lifecycle.preStop.exec.command`

Command for the preStop lifecycle hook for HAProxy Pods.

Value type	Example
[] array	`["/bin/true"]`

`haproxy.lifecycle.postStart.exec.command`

Command for the postStart lifecycle hook for HAProxy Pods.

Value type	Example
[] array	`["/bin/true"]`

ProxySQL section

The `proxysql` section in the deploy/cr.yaml file contains configuration options for the ProxySQL daemon.

`proxysql.enabled`

Enables or disables load balancing with ProxySQL Services **ProxySQL can be enabled only at cluster creation time**; otherwise you will be limited to HAProxy load balancing.

Value type	Example
boolean	`false`

`proxysql.size`

The number of the ProxySQL daemons to provide load balancing. It should be 2 or more unless the `spec.unsafeFlags.proxySize` key is set to true.

Value type	Example
int	`2`

`proxysql.image`

ProxySQL Docker image to use.

Value type	Example
string	`percona/percona-xtradb-cluster-operator:1.19.0-proxysql`

`proxysql.imagePullPolicy`

The policy used to update images.

Value type	Example
string	`Always`

`proxysql.imagePullSecrets.name`

The Kubernetes imagePullSecrets for the ProxySQL image.

Value type	Example
string	`private-registry-credentials`

`proxysql.configuration`

The custom ProxySQL configuration file contents.

Value type	Example
string	

`proxysql.annotations`

The Kubernetes annotations metadata.

Value type	Example
label	`iam.amazonaws.com/role: role-arn`

`proxysql.labels`

Labels are key-value pairs attached to objects.

Value type	Example
label	`rack: rack-22`

`proxysql.expose.enabled`

Enable or disable exposing ProxySQL nodes with dedicated IP addresses.

Value type	Example
boolean	`false`

`proxysql.expose.type`

Specifies the type of Kubernetes Service to be used.

Value type	Example
string	`ClusterIP`

`proxysql.expose.loadBalancerClass`

Define the implementation of the load balancer you want to use. This setting enables you to select a custom or specific load balancer class instead of the default one provided by the cloud provider.

Value type	Example
string	`"eks.amazonaws.com/nlb"`

`proxysql.externalTrafficPolicy`

Specifies whether Service for ProxySQL should route external traffic to cluster-wide or to node-local endpoints (it can influence the load balancing effectiveness). **This option is deprecated and will be removed in future releases**. Use `proxysql.expose.externalTrafficPolicy` instead.

Value type	Example
string	`Local`

`proxysql.expose.externalTrafficPolicy`

Specifies whether Service for ProxySQL should route external traffic to cluster-wide or to node-local endpoints (it can influence the load balancing effectiveness).

Value type	Example
string	`Local`

`proxysql.expose.internalTrafficPolicy`

Specifies whether Service for ProxySQL should route internal traffic to cluster-wide or to node-local endpoints (it can influence the load balancing effectiveness).

Value type	Example
S string	`Local`

`proxysql.serviceAnnotations`

The Kubernetes annotations metadata for the load balancer Service. **This option is deprecated and will be removed in future releases**. Use `proxysql.expose.annotations` instead.

Value type	Example
label	`service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp`

`proxysql.expose.annotations`

The Kubernetes annotations metadata for the load balancer Service.

Value type	Example
label	`service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp`

`proxysql.serviceLabels`

The Kubernetes labels for the load balancer Service. **This option is deprecated and will be removed in future releases**. Use `proxysql.expose.labels` instead.

Value type	Example
label	`rack: rack-22`

`proxysql.expose.labels`

The Kubernetes labels for the load balancer Service.

Value type	Example
label	`rack: rack-22`

`proxysql.loadBalancerSourceRanges`

The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations). **This option is deprecated and will be removed in future releases**. Use `proxysql.expose.loadBalancerSourceRanges` instead.

Value type	Example
S string	`10.0.0.0/8`

`proxysql.expose.loadBalancerSourceRanges`

The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations).

Value type	Example
S string	`10.0.0.0/8`

`proxysql.expose.loadBalancerIP`

The static IP-address for the load balancer. **This field is deprecated and scheduled for removal in version 1.21.0.**.

`loadBalancerIP` has been officially deprecated upstream in Kubernetes due to its inconsistent behavior across cloud providers and lack of dual-stack support. As a result, its usage is strongly discouraged.

We recommend using cloud provider-specific annotations instead, as they offer more predictable and portable behavior for managing load balancer IP assignments.

Value type	Example
S string	`127.0.0.1`

`proxysql.resources.requests.memory`

The Kubernetes memory requests for the main ProxySQL container.

Value type	Example
S string	`1G`

`proxysql.resources.requests.cpu`

Kubernetes CPU requests for the main ProxySQL container.

Value type	Example
S string	`600m`

`proxysql.resources.limits.memory`

Kubernetes memory limits for the main ProxySQL container.

Value type	Example
S string	`1G`

`proxysql.resources.limits.cpu`

Kubernetes CPU limits for the main ProxySQL container.

Value type	Example
S string	`700m`

`proxysql.envVarsSecret`

A secret with environment variables, see Define environment variables for details.

Value type	Example
S string	`my-env-var-secrets`

`proxysql.scheduler.enabled`

Enables the external ProxySQL scheduler for even distribution of read/write traffic across Percona XtraDB Cluster nodes. Available since Operator version 1.19.0

See ProxySQL scheduler for more information.

Value type	Example
boolean	`true`

`proxysql.scheduler.writerIsAlsoReader`

Controls whether the writer node is included in the read pool. When set to `false`, the writer node is excluded from receiving read queries. If the cluster loses its last reader, the writer is automatically elected as a reader regardless of this setting. Available since Operator version 1.19.0

Value type	Example
boolean	`true`

`proxysql.scheduler.checkTimeoutMilliseconds`

The maximum time (in milliseconds) allowed for checking a backend PXC node. If checking a node exceeds this timeout, it is not processed. Available since Operator version 1.19.0

Value type	Example
int	`2000`

`proxysql.scheduler.successThreshold`

The number of successful checks required before a failed node is restored to the pool. Available since Operator version 1.19.0

Value type	Example
int	`1`

`proxysql.scheduler.failureThreshold`

The number of failed checks required before a node is marked as DOWN and removed from the pool. Available since Operator version 1.19.0

Value type	Example
int	`3`

`proxysql.scheduler.pingTimeoutMilliseconds`

The connection timeout (in milliseconds) used to test the connection to a PXC server. Available since Operator version 1.19.0

Value type	Example
int	`1000`

`proxysql.scheduler.nodeCheckIntervalMilliseconds`

How frequently (in milliseconds) the scheduler runs to check node health and update the server configuration. Available since Operator version 1.19.0

Value type	Example
int	`2000`

`proxysql.scheduler.maxConnections`

The maximum number of connections from ProxySQL to each backend PXC server. Available since Operator version 1.19.0

Value type	Example
1 int	`1000`

`proxysql.priorityClassName`

The Kubernetes Pod Priority class for ProxySQL.

Value type	Example
S string	`high-priority`

`proxysql.schedulerName`

The Kubernetes Scheduler.

Value type	Example
S string	`mycustom-scheduler`

`proxysql.nodeSelector`

Kubernetes nodeSelector.

Value type	Example
label	`disktype: ssd`

`proxysql.sidecarResources.requests.memory`

The Kubernetes memory requests for default sidecar containers such as `proxysql-monit` running inside ProxySQL Pods.

Value type	Example
S string	`1G`

`proxysql.sidecarResources.requests.cpu`

Kubernetes CPU requests for default sidecar containers such as `proxysql-monit` running inside ProxySQL Pods.

Value type	Example
S string	`500m`

`proxysql.sidecarResources.limits.memory`

Kubernetes memory limits for default sidecar containers such as `proxysql-monit` running inside ProxySQL Pods.

Value type	Example
S string	`2G`

`proxysql.sidecarResources.limits.cpu`

Kubernetes CPU limits for default sidecar containers such as `proxysql-monit` running inside ProxySQL Pods.

Value type	Example
string	`600m`

`proxysql.topologySpreadConstraints.labelSelector.matchLabels`

The Label selector for the Kubernetes Pod Topology Spread Constraints.

Value type	Example
label	`app.kubernetes.io/name: percona-xtradb-cluster-operator`

`proxysql.topologySpreadConstraints.maxSkew`

The degree to which Pods may be unevenly distributed under the Kubernetes Pod Topology Spread Constraints.

Value type	Example
int	1

`proxysql.topologySpreadConstraints.topologyKey`

The key of node labels for the Kubernetes Pod Topology Spread Constraints.

Value type	Example
string	`kubernetes.io/hostname`

`proxysql.topologySpreadConstraints.whenUnsatisfiable`

What to do with a Pod if it doesn't satisfy the Kubernetes Pod Topology Spread Constraints.

Value type	Example
string	`DoNotSchedule`

`proxysql.affinity.topologyKey`

The Operator topology key node anti-affinity constraint.

Value type	Example
string	`kubernetes.io/hostname`

`proxysql.affinity.advanced`

If available it makes a topologyKey node affinity constraint to be ignored.

Value type	Example
subdoc	

`proxysql.tolerations`

Kubernetes Pod tolerations.

Value type	Example
subdoc	`node.alpha.kubernetes.io/unreachable`

`proxysql.volumeSpec.emptyDir`

The Kubernetes emptyDir volume The directory created on a node and accessible to the Percona XtraDB Cluster Pod containers.

Value type	Example
string	`{}`

`proxysql.volumeSpec.hostPath.path`

Kubernetes hostPath The volume that mounts a directory from the host node's filesystem into your Pod. The path property is required.

Value type	Example
string	`/data`

`proxysql.volumeSpec.hostPath.type`

The Kubernetes hostPath. An optional property for the hostPath.

Value type	Example
string	`Directory`

`proxysql.volumeSpec.persistentVolumeClaim.storageClassName`

Set the Kubernetes storage class to use with the Percona XtraDB Cluster PersistentVolumeClaim.

Value type	Example
string	`standard`

`proxysql.volumeSpec.persistentVolumeClaim.accessModes`

The Kubernetes PersistentVolumeClaim access modes for the Percona XtraDB cluster.

Value type	Example
array	`[ReadWriteOnce]`

`proxysql.volumeSpec.resources.requests.storage`

The Kubernetes PersistentVolumeClaim size for the Percona XtraDB cluster.

Value type	Example
string	`6Gi`

`proxysql.extraPVCs.name`

The name of the volume provisioned for the external PersistentVolumeClaim that you mount to ProxySQL pods. In such a way you can attach pre-existing storage volumes to your database instances for use cases such as importing data, sharing configuration files, or accessing external datasets. For more information, see Add external PersistentVolumeClaims to the Operator. You can configure external PVCs for both new and running clusters.

Value type	Example
S string	`shared-data`

`proxysql.extraPVCs.claimName`

The name of the existing PersistentVolumeClaim to mount. This PVC must exist in the same namespace as your cluster before you apply the configuration. The Operator will mount the existing PVC to all ProxySQL pods.

Value type	Example
S string	`my-existing-pvc`

`proxysql.extraPVCs.mountPath`

The path inside the container where the volume will be mounted.

Value type	Example
S string	`/mnt/shared-data`

`proxysql.extraPVCs.subPath`

An optional folder within the volume to mount. If not specified, the volume's root is mounted.

Value type	Example
S string	`config`

`proxysql.extraPVCs.readOnly`

Optional read-only flag. If set to `true`, the volume will be mounted as read-only. Defaults to `false`.

Value type	Example
boolean	`false`

`proxysql.podDisruptionBudget.maxUnavailable`

The Kubernetes podDisruptionBudget specifies the number of Pods from the set unavailable after the eviction.

Value type	Example
1 int	`1`

`proxysql.podDisruptionBudget.minAvailable`

The Kubernetes podDisruptionBudget Pods that must be available after an eviction.

Value type	Example

int	`0`

`proxysql.gracePeriod`

The Kubernetes grace period when terminating a Pod.

Value type	Example
int	`30`

`proxysql.containerSecurityContext`

A custom Kubernetes Security Context for a Container to be used instead of the default one.

Value type	Example
subdoc	`privileged: true`

`proxysql.podSecurityContext`

A custom Kubernetes Security Context for a Pod to be used instead of the default one.

Value type	Example
subdoc	`fsGroup: 1001` `supplementalGroups: [1001, 1002, 1003]`

`proxysql.serviceAccountName`

The Kubernetes Service Account for the ProxySQL Pod.

Value type	Example
string	`percona-xtradb-cluster-operator-workload`

`proxysql.runtimeClassName`

Name of the Kubernetes Runtime Class for the ProxySQL Pod.

Value type	Example
string	`image-rc`

`proxysql.sidecars.image`

Image for the custom sidecar container for the ProxySQL Pod.

Value type	Example
string	`busybox`

`proxysql.sidecars.command`

Command for the custom sidecar container for the ProxySQL Pod.

Value type	Example

array	`["/bin/sh"]`

`proxysql.sidecars.args`

Command arguments for the custom sidecar container for the ProxySQL Pod.

Value type	Example
array	`["-c", "while true; do trap 'exit 0' SIGINT SIGTERM SIGQUIT SIGKILL; done;"]`

`proxysql.sidecars.name`

Name of the custom sidecar container for the ProxySQL Pod.

Value type	Example
string	`my-sidecar-1`

`proxysql.sidecars.resources.requests.memory`

The Kubernetes memory requests for the custom sidecar containers deployed in ProxySQL Pods.

Value type	Example
string	`1G`

`proxysql.sidecars.resources.requests.cpu`

Kubernetes CPU requests for the custom sidecar containers deployed in ProxySQL Pods.

Value type	Example
string	`500m`

`proxysql.sidecars.resources.limits.memory`

Kubernetes memory limits for the custom sidecar containers deployed in ProxySQL Pods.

Value type	Example
string	`2G`

`proxysql.sidecars.resources.limits.cpu`

Kubernetes CPU limits for the custom sidecar containers deployed in ProxySQL Pods.

Value type	Example
string	`600m`

`proxysql.sidecars.securityContext`

A custom Kubernetes Security Context for a sidecar container to be used instead of the default one.

Value type	Example

subdoc	privileged: false

`proxysql.lifecycle.preStop.exec.command`

Command for the preStop lifecycle hook for ProxySQL Pods.

Value type	Example
array	`["/bin/true"]`

`proxysql.lifecycle.postStart.exec.command`

Command for the postStart lifecycle hook for ProxySQL Pods.

Value type	Example
array	`["/bin/true"]`

Log Collector section

The `logcollector` section in the deploy/cr.yaml file contains configuration options for Fluent Bit Log Collector.

`logcollector.enabled`

Enables or disables cluster-level logging with Fluent Bit.

Value type	Example
boolean	`true`

`logcollector.image`

Log Collector Docker image to use.

Value type	Example
string	`percona/percona-xtradb-cluster-operator:1.6.0-logcollector`

`logcollector.configuration`

Additional configuration options (see Fluent Bit official documentation for details).

Value type	Example
subdoc	

`logcollector.resources.requests.memory`

The Kubernetes memory requests for a Log Collector sidecar container in a Percona XtraDB Cluster Pod.

Value type	Example
string	`100M`

`logcollector.resources.requests.cpu`

Kubernetes CPU requests for a Log collector sidecar container in a Percona XtraDB Cluster Pod.

Value type	Example
string	`200m`

Users section

The `users` section in the deploy/cr.yaml file contains various configuration options to configure custom MySQL users via the Custom Resource.

`users.name`

The username of the MySQL user.

Value type	Example
string	`my-user`

`users.dbs`

Databases that the user authenticates against. If the specified database is not present, the Operator will create it. When no databases specified, it defaults to all databases (*). If the user sets administrative grants like SHUTDOWN, this field has to be omitted because administrative privileges are set on a global level.

Value type	Example
array	`- db1` `-db2`

`users.hosts`

Hosts that the users are supposed to connect from (if not specified, defaults to '%' - similar to what is happening in MySQL).

Value type	Example
array	`- localhost`

`users.passwordSecretRef.name`

Name of the secret that contains the user's password. If not provided, the Operator will create the `<cluster-name>-<custom-user-name>-secret` secret and generate password automatically.

Value type	Example
string	`my-user-password`

`users.passwordSecretRef.key`

Key in the secret that corresponds to the value of the user's password (`password` by default).

Value type	Example
string	`password`

`spec.users.withGrantOption`

Defines if the user has grant options.

Value type	Example
boolean	`false`

`users.grants`

Privileges granted to the user.

Value type	Example

array	`- SELECT`
	`- DELETE`
	`- INSERT`

PMM section

The `pmm` section in the deploy/cr.yaml file contains configuration options for Percona Monitoring and Management.

`pmm.enabled`

Enables or disables monitoring Percona XtraDB cluster with PMM.

Value type	Example
boolean	`false`

`pmm.image`

PMM client Docker image to use.

Value type	Example
string	`percona/pmm-client:2.44.1-1`

`pmm.serverHost`

Address of the PMM Server to collect data from the cluster.

Value type	Example
string	`monitoring-service`

pmm.customClusterName

A custom name to define for a cluster. PMM Server uses this name to properly parse the metrics and display them on dashboards. Using a custom name is useful for clusters deployed in different data centers - PMM Server connects them and monitors them as one deployment. Another use case is for clusters deployed with the same name in different namespaces - PMM treats each cluster separately.

Value type	Example
string	`testClusterName`

`pmm.serverUser`

The PMM Server User. The PMM Server password should be configured using Secrets.

Value type	Example

string	`admin`

`pmm.resources.requests.memory`

The Kubernetes memory requests for a PMM container.

Value type	Example
string	`150M`

`pmm.resources.requests.cpu`

Kubernetes CPU requests for a PMM container.

Value type	Example
string	`300m`

`pmm.pxcParams`

Additional parameters which will be passed to the pmm-admin add mysql command for `pxc` Pods.

Value type	Example
string	`--disable-tablestats-limit=2000`

`pmm.proxysqlParams`

Additional parameters which will be passed to the pmm-admin add proxysql command for `proxysql` Pods.

Value type	Example
string	`--custom-labels=CUSTOM-LABELS`

`pmm.containerSecurityContext`

A custom Kubernetes Security Context for a Container to be used instead of the default one.

Value type	Example
subdoc	`privileged: false`

`pmm.readinessProbes.initialDelaySeconds`

The number of seconds to wait before performing the first readiness probe.

Value type	Example
int	`15`

`pmm.readinessProbes.timeoutSeconds`

The number of seconds after which the readiness probe times out.

Value type	Example

int	`15`

`pmm.readinessProbes.periodSeconds`

How often to perform the readiness probe. Measured in seconds.

Value type	Example
int	`30`

`pmm.readinessProbes.successThreshold`

The number of successful probes required to mark the container successful.

Value type	Example
int	`1`

`pmm.readinessProbes.failureThreshold`

The number of failed probes required to mark the container unready.

Value type	Example
int	`5`

`pmm.livenessProbes.initialDelaySeconds`

The number of seconds to wait before performing the first liveness probe.

Value type	Example
int	`300`

`pmm.livenessProbes.timeoutSeconds`

The number of seconds after which the liveness probe times out.

Value type	Example
int	`5`

`pmm.livenessProbes.periodSeconds`

How often to perform the liveness probe. Measured in seconds.

Value type	Example
int	`10`

`pmm.livenessProbes.successThreshold`

The number of successful probes required to mark the container successful.

Value type	Example

1 int	`1`

`pmm.livenessProbes.failureThreshold`

The number of failed probes required to mark the container unhealthy.

Value type	Example
1 int	`3`

Backup section

The `backup` section in the deploy/cr.yaml file contains the following configuration options for the regular Percona XtraDB Cluster backups.

`backup.allowParallel`

Enables or disables running backup jobs in parallel. By default, parallel backup jobs are enabled. A user can disable them to prevent the cluster overload.

Value type	Example
S string	`true`

`backup.image`

The Percona XtraDB cluster Docker image to use for the backup.

Value type	Example
S string	`percona/percona-xtradb-cluster-operator:1.19.0-backup`

`backup.ttlSecondsAfterFinished`

The time for a backup or restore job to live after it finishes. After this time expires, the Operator removes the backup/restore Job and associated Pod.

Applies to both on-demand/scheduled backups and restores.

When the value is too low, the Operator applies the internal.percona.com/keep-job finalizer to allow the operation to finish. After the operation completes with the Succeeded or Failed status, the finalizer is removed and the Job is cleaned up.

Value type	Example
1 int	`3600`

`backup.backoffLimit`

The number of retries to make a backup (by default, 10 retries are made).

Value type	Example
1 int	`6`

`backup.activeDeadlineSeconds`

The timeout value in seconds, after which backup job will automatically fail.

Value type	Example

int	`3600`

`backup.startingDeadlineSeconds`

The maximum time in seconds for a backup to start. The Operator compares the timestamp of the backup object against the current time. If the backup is not started within the set time, the Operator automatically marks it as “failed”.

You can override this setting for a specific backup in the `deploy/backup/backup.yaml` configuration file.

Value type	Example
int	`300`

backup.suspendedDeadlineSeconds

The maximum time in seconds for a backup to remain in a suspended state. The Operator compares the timestamp when the backup job was suspended against the current time. After the defined suspension time expires, the backup is automatically marked as “failed”.

You can override this setting for a specific backup in the `deploy/backup/backup.yaml` configuration file.

Value type	Example
int	`1200`

`backup.imagePullSecrets.name`

The Kubernetes imagePullSecrets for the specified image.

Value type	Example
string	`private-registry-credentials`

`backup.storages.STORAGE-NAME.type`

The cloud storage type used for backups. Only `s3`, `azure`, and `filesystem` types are supported.

Value type	Example
string	`s3`

`backup.storages.STORAGE-NAME.verifyTLS`

Enable or disable verification of the storage server TLS certificate. Disabling it may be useful e.g. to skip TLS verification for private S3-compatible storage with a self-issued certificate.

Value type	Example
boolean	`true`

`backup.storages.STORAGE-NAME.s3.credentialsSecret`

The Kubernetes secret for backups. It should contain `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` keys.

Value type	Example
string	`my-cluster-name-backup-s3`

`backup.storages.STORAGE-NAME.s3.bucket`

The Amazon S3 bucket name for backups.

Value type	Example
S string	

`backup.storages.STORAGE-NAME.s3.region`

The AWS region to use. Please note **this option is mandatory** for Amazon and all S3-compatible storages.

Value type	Example
S string	`us-east-1`

`backup.storages.STORAGE-NAME.s3.endpointUrl`

The endpoint URL of the S3-compatible storage to be used (not needed for the original Amazon S3 cloud).

Value type	Example
S string	

`backup.storages.STORAGE-NAME.s3.caBundle.name`

The name of the Secret that stores custom TLS certificates for TLS communication with S3 storage. See Configure TLS verification with custom certificates for S3 storage for more information.

Value type	Example
S string	`s3-ca-bundle-secret`

`backup.storages.STORAGE-NAME.s3.caBundle.key`

The custom CA certificate for TLS communication with S3 storage. See Configure TLS verification with custom certificates for S3 storage for more information.

Value type	Example
S string	`ca.crt`

`backup.storages.STORAGE-NAME.volume.persistentVolumeClaim.type`

The persistent volume claim storage type.

Value type	Example
S string	`filesystem`

`backup.storages.STORAGE-NAME.volume.persistentVolumeClaim.storageClassName`

Set the Kubernetes Storage Class to use with the Percona XtraDB Cluster backups PersistentVolumeClaims for the `filesystem` storage type.

Value type	Example
S string	`standard`

`backup.storages.STORAGE-NAME.volume.persistentVolumeClaim.accessModes`

The Kubernetes PersistentVolume access modes.

Value type	Example
array	`[ReadWriteOne]`

`backup.storages.STORAGE-NAME.volume.persistentVolumeClaim.resources.requests.storage`

Storage size for the PersistentVolume.

Value type	Example
string	`6Gi`

`backup.storages.STORAGE-NAME.annotations`

The Kubernetes annotations.

Value type	Example
label	`iam.amazonaws.com/role: role-arn`

`backup.storages.STORAGE-NAME.labels`

Labels are key-value pairs attached to objects.

Value type	Example
label	`rack: rack-22`

`backup.storages.STORAGE-NAME.resources.requests.memory`

The Kubernetes memory requests for a Percona XtraBackup container.

Value type	Example
string	`1G`

`backup.storages.STORAGE-NAME.resources.requests.cpu`

Kubernetes CPU requests for a Percona XtraBackup container.

Value type	Example
string	`600m`

`backup.storages.STORAGE-NAME.resources.limits.memory`

Kubernetes memory limits for a Percona XtraBackup container.

Value type	Example
string	`1.5G`

`backup.storages.STORAGE-NAME.resources.limits.cpu`

Kubernetes CPU limits for a Percona XtraBackup container.

Value type	Example
S string	`700m`

`backup.storages.STORAGE-NAME.nodeSelector`

Kubernetes nodeSelector.

Value type	Example
label	`disktype: ssd`

`backup.storages.STORAGE-NAME.topologySpreadConstraints.labelSelector.matchLabels`

The Label selector for the Kubernetes Pod Topology Spread Constraints.

Value type	Example
label	`app.kubernetes.io/name: percona-xtradb-cluster-operator`

`backup.storages.STORAGE-NAME.topologySpreadConstraints.maxSkew`

The degree to which Pods may be unevenly distributed under the Kubernetes Pod Topology Spread Constraints.

Value type	Example
1 int	`1`

`backup.storages.STORAGE-NAME.topologySpreadConstraints.topologyKey`

The key of node labels for the Kubernetes Pod Topology Spread Constraints.

Value type	Example
S string	`kubernetes.io/hostname`

`backup.storages.STORAGE-NAME.topologySpreadConstraints.whenUnsatisfiable`

What to do with a Pod if it doesn't satisfy the Kubernetes Pod Topology Spread Constraints.

Value type	Example
S string	`DoNotSchedule`

`backup.storages.STORAGE-NAME.affinity.nodeAffinity`

The Operator node affinity constraint.

Value type	Example
subdoc	

`backup.storages.STORAGE-NAME.tolerations`

Kubernetes Pod tolerations.

Value type	Example
subdoc	`backupWorker`

`backup.storages.STORAGE-NAME.priorityClassName`

The Kubernetes Pod priority class.

Value type	Example
string	`high-priority`

`backup.storages.STORAGE-NAME.schedulerName`

The Kubernetes Scheduler.

Value type	Example
string	`mycustom-scheduler`

`backup.storages.STORAGE-NAME.containerSecurityContext`

A custom Kubernetes Security Context for a Container to be used instead of the default one.

Value type	Example
subdoc	`privileged: true`

`backup.storages.STORAGE-NAME.podSecurityContext`

A custom Kubernetes Security Context for a Pod to be used instead of the default one.

Value type	Example
subdoc	`fsGroup: 1001` `supplementalGroups: [1001, 1002, 1003]`

`backup.storages.STORAGE-NAME.containerOptions.env`

The environment variables set as key-value pairs for the backup container.

Value type	Example
subdoc	`- name: VERIFY_TLS` `value: “false”`

`backup.storages.STORAGE-NAME.containerOptions.args.xtrabackup`

Custom command line options for the `xtrabackup` Percona XtraBackup tool.

Value type	Example
subdoc	`- “–someflag=abc”`

`backup.storages.STORAGE-NAME.containerOptions.args.xbcloud`

Custom command line options for the `xbcloud` Percona XtraBackup tool.

Value type	Example
subdoc	`- "--someflag=abc"`

`backup.storages.STORAGE-NAME.containerOptions.args.xbstream`

Custom command line options for the `xbstream` Percona XtraBackup tool.

Value type	Example
subdoc	`- "--someflag=abc"`

`backup.schedule.name`

The backup name.

Value type	Example
string	`sat-night-backup`

`backup.schedule.schedule`

Scheduled time to make a backup specified in the crontab format.

Value type	Example
string	`0 0 \* \* 6`

`backup.schedule.keep`

The amount of most recent backups to store. Older backups are automatically deleted. Set `keep` to zero or completely remove it to disable automatic deletion of backups. **This option is deprecated and will be removed in version 1.21.0**.

Value type	Example
int	`3`

`backup.schedule.retention.type`

Defines how to retain backups. The type of retention defaults to `count`.

Value type	Example
string	`count`

`backup.schedule.retention.count`

Defines the number of backups to store. Older backups are automatically deleted from the cluster.

Value type	Example
string	`count`

`backup.schedule.retention.deleteFromStorage`

Defines if the backups are deleted from the cloud storage too. Supported only for AWS and Azure storage. Does not apply to backups made to Persistent Volume.

Value type	Example
boolean	`true`

`backup.schedule.storageName`

The name of the storage for the backups configured in the `storages` or `fs-pvc` subsection.

Value type	Example
string	`s3-us-west`

`backup.pitr.enabled`

Enables or disables point-in-time-recovery functionality.

Value type	Example
boolean	`false`

`backup.pitr.storageName`

The name of the storage for the backups configured in the `storages` subsection, which will be reused to store binlog for point-in-time-recovery.

Value type	Example
string	`s3-us-west`

`backup.pitr.timeBetweenUploads`

Seconds between running the binlog uploader.

Value type	Example
int	`60`

`backup.pitr.timeoutSeconds`

Timeout in seconds for the binlog to be uploaded; the binlog uploader container will be restarted after exceeding this timeout |

Value type	Example
int	`60`

`backup.pitr.resources.requests.memory`

The Kubernetes memory requests for a binlog collector Pod.

Value type	Example
string	`0.1G`

`backup.pitr.resources.requests.cpu`

Kubernetes CPU requests for a binlog collector Pod.

Value type	Example
S string	`100m`

`backup.pitr.resources.limits.memory`

Kubernetes memory limits for a binlog collector Pod. | Value type | Example | | ---------- | --------- | | S string | `1G` |

`backup.pitr.resources.limits.cpu`

Kubernetes CPU limits for a binlog collector Pod.

Value type	Example
S string	`700m`

Password generation section

This section contains the options to customize password generation for user Secrets

`passwordGenerationOptions.symbols`

Specify what special symbols to use when generating user passwords. The passwords must contain only ASCII characters to ensure authentication to MySQL.

Value type	Example
S string	`"!#$%&()*+,-.<=>?@[]^_{}~"`

`passwordGenerationOptions.maxLength`

Specify the max password length for user passwords

Value type	Example
1 int	`20`

`passwordGenerationOptions.minLength`

Specify the min password length for user passwords

Value type	Example
1 int	`16`

PerconaXtraDBClusterBackup Custom Resource options

A Backup resource is a Kubernetes object that tells the Operator how to backup your database. The deploy/backup/backup.yaml file is a template for creating backup resources.

It defines the `PerconaXtraDBClusterBackup` resource.

This document describes all available options that you can use to customize a backup.

`apiVersion`

Specifies the API version of the Custom Resource. `pxc.percona.com` indicates the group, and `v1` is the version of the API.

`kind`

Defines the type of resource being created: `PerconaXtraDBClusterBackup`.

`metadata`

The metadata part contains the following keys:

- `name` sets the name of your backup resource;
- `finalizers` subsection:
  - `percona.com/delete-backup` if present, enables deletion of backup files from a backup storage when the backup object is removed (manually or by schedule). When used with the Persistent Volume as the backup storage, the finalizer deletes the PVC.
  - `internal.percona.com/keep-job` ensures the backup Job is not cleaned up due to expired TTL until the backup operation finishes. After the operation completes with the Succeeded or Failed status, the finalizer is removed and the Job is cleaned up.

`spec` section

The toplevel spec elements of the deploy/backup/backup.yaml are the following ones:

`pxcCluster`

The name of the Percona XtraDB Cluster to back up.

Value type	Example
S string	`cluster1`

`storageName`

The name of the storage configuration defined in your `deploy/cr.yaml` file in the `spec.backup.storages` subsection.

Value type	Example
S string	`fs-pvc`

`activeDeadlineSeconds`

The timeout value in seconds, after which backup job will automatically fail.

Value type	Example
1 int	`3600`

`startingDeadlineSeconds`

The maximum time in seconds for a backup to reach the Starting state. The Operator compares the timestamp of the backup object against the current time. If the backup is not started within the set time, the Operator automatically marks it as “failed”.

Value type	Example
1 int	`300`

`suspendedDeadlineSeconds`

The maximum time in seconds for a backup to remain in a suspended state. The Operator compares the timestamp when the backup job was suspended against the current time. After the defined suspension time expires, the backup is automatically marked as “failed”.

Value type	Example
1 int	`1200`

`runningDeadlineSeconds`

The maximum time in seconds for a backup job to reach the Running state. The Operator compares the timestamp when the backup job started running against the current time. After the defined running time expires, the backup is automatically marked as “failed”.

Value type	Example
1 int	`300`

`containerOptions.env`

The environment variables set as key-value pairs for the backup container.

Value type	Example
subdoc	`- name: VERIFY_TLS` `value: “false”`

`containerOptions.args.xbcloud`

Custom command line options for the `xbcloud` Percona XtraBackup tool.

Value type	Example
subdoc	`- “–someflag=abc”`

`containerOptions.args.xbstream`

Custom command line options for the `xbstream` Percona XtraBackup tool.

Value type	Example
subdoc	`- “–someflag=abc”`

PerconaXtraDBClusterRestore Custom Resource options

A Restore resource is a Kubernetes object that tells the Operator how to restore your database from a specific backup. The deploy/backup/restore.yaml file is a template for creating restore resources.

It defines the `PerconaXtraDBClusterRestore` resource.

The metadata part contains the following keys:

- `name` sets the name of your restore resource;
- `annotations` subsection:
  - `percona.com/headless-service` if present, activates the headless service for the restore.

`spec` section

The toplevel spec elements of the deploy/backup/restore.yaml are the following ones:

`pxcCluster`

The name of the Percona XtraDB Cluster to restore the backup to.

Value type	Example
S string	`cluster1`

`backupName`

The name of the backup which should be restored.

Value type	Example
S string	`backup1`

`containerOptions.env`

The environment variables set as key-value pairs for the restore container.

Value type	Example
subdoc	`- name: VERIFY_TLS` `value: "false"`

`containerOptions.args.xtrabackup`

Custom command line options for the `xtrabackup` Percona XtraBackup tool.

Value type	Example
subdoc	`- "–someflag=abc"`

`containerOptions.args.xbcloud`

Custom command line options for the `xbcloud` Percona XtraBackup tool.

Value type	Example
subdoc	`- "–someflag=abc"`

`containerOptions.args.xbstream`

Custom command line options for the `xbstream` Percona XtraBackup tool.

Value type	Example
subdoc	`- "--someflag=abc"`

`resources.requests.memory`

The Kubernetes memory requests for the restore job.

Value type	Example
string	`100M`

`resources.requests.cpu`

Kubernetes CPU requests for the restore job.

Value type	Example
string	`100m`

`resources.limits.memory`

Kubernetes memory limits for the restore job.

Value type	Example
string	`200M`

`resources.limits.cpu`

Kubernetes CPU limits for the restore job.

Value type	Example
string	`200m`

backupSource section

The `backupSource` section in the deploy/backup/restore.yaml file contains configuration options for restoring from external backup sources.

`backupSource.verifyTLS`

Enable or disable verification of the storage server TLS certificate. Disabling it may be useful e.g. to skip TLS verification for private S3-compatible storage with a self-issued certificate.

Value type	Example
boolean	`true`

`backupSource.destination`

Path to the backup in the storage. The format depends on the storage type: `s3://S3-BUCKET-NAME/BACKUP-NAME` for S3-compatible storage or `azure://CONTAINER-NAME/BACKUP-NAME` for Azure Blob storage.

Value type	Example
S string	`s3://my-bucket/my-backup`

`backupSource.s3.bucket`

The Amazon S3 bucket name for backups.

Value type	Example
S string	`S3-BINLOG-BACKUP-BUCKET-NAME-HERE`

`backupSource.s3.credentialsSecret`

The Kubernetes secret for backups. It should contain `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` keys.

Value type	Example
S string	`my-cluster-name-backup-s3`

`backupSource.s3.endpointUrl`

The endpoint URL of the S3-compatible storage to be used (not needed for the original Amazon S3 cloud).

Value type	Example
S string	`https://s3.us-west-2.amazonaws.com/`

`backupSource.s3.region`

The AWS region to use. Please note **this option is mandatory** for Amazon and all S3-compatible storages.

Value type	Example
S string	`us-west-2`

`backupSource.s3.caBundle.name`

The name of the Secret that stores custom TLS certificates for TLS communication with S3 storage.

Value type	Example
S string	`minio-ca-bundle`

`backupSource.s3.caBundle.key`

The key in the Secret that corresponds to the custom CA certificate file used to sign TLS certificates.

Value type	Example
S string	`tls.crt`

`backupSource.azure.container`

The container name of the Azure Blob storage.

Value type	Example
string	`<your-container-name>`

`backupSource.azure.credentialsSecret`

The Kubernetes secret for Azure Blob storage backups.

Value type	Example
string	`my-cluster-name-backup-azure`

pitr section

The `pitr` section in the deploy/backup/restore.yaml file contains configuration options for point-in-time-recovery.

`pitr.type`

The type of point-in-time recovery. Supported values are `latest` to restore to the latest available point in time, `date` to restore to a specific date, or `gtid` to restore to a specific GTID.

Value type	Example
string	`latest`

`pitr.date`

The exact date and time for point-in-time recovery, specified in the format `"yyyy-mm-dd hh:mm:ss"`.

Value type	Example
string	`"2024-01-15 14:30:00"`

`pitr.gtid`

The exact GTID for point-in-time recovery, specified in the format `"aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee:nnn"`.

Value type	Example
string	`"aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee:123"`

`pitr.backupSource.verifyTLS`

Enable or disable verification of the storage server TLS certificate for binlog backups. Disabling it may be useful e.g. to skip TLS verification for private S3-compatible storage with a self-issued certificate.

Value type	Example
boolean	`true`

`pitr.backupSource.storageName`

The name of the storage for binlog backups configured in the `spec.backup.storages` subsection.

Value type	Example
string	`STORAGE-NAME-HERE`

`pitr.backupSource.s3.bucket`

The Amazon S3 bucket name for binlog backups.

Value type	Example
S string	`S3-BINLOG-BACKUP-BUCKET-NAME-HERE`

`pitr.backupSource.s3.credentialsSecret`

The Kubernetes secret for binlog backups. It should contain `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` keys.

Value type	Example
S string	`my-cluster-name-backup-s3`

`pitr.backupSource.s3.endpointUrl`

The endpoint URL of the S3-compatible storage to be used for binlog backups (not needed for the original Amazon S3 cloud).

Value type	Example
S string	`https://s3.us-west-2.amazonaws.com/`

`pitr.backupSource.s3.region`

The AWS region to use for binlog backups. Please note **this option is mandatory** for Amazon and all S3-compatible storages.

Value type	Example
S string	`us-west-2`

`pitr.backupSource.s3.caBundle.name`

The name of the Secret that stores custom TLS certificates for TLS communication with S3 storage for binlog backups.

Value type	Example
S string	`minio-ca-bundle`

`pitr.backupSource.s3.caBundle.key`

The key in the Secret that corresponds to the custom CA certificate file used to sign TLS certificates for binlog backups.

Value type	Example
S string	`tls.crt`

Percona certified images

Find Percona's certified Docker images that you can use with the Percona Operator for MySQL based on Percona XtraDB Cluster in the following table.

Images released with the Operator version 1.19.0:

Image	Digest
percona/percona-xtradb-cluster-operator:1.19.0	6ccbac5e74f5b5309fd4788c5b8d91d5abd01850a4a356ad9eff9f82d20afb51
percona/percona-xtradb-cluster-operator:1.19.0 (ARM64)	1ed2a5ab22ee7588aa17ec2339876dc72c9724dc9a81506ff449a2b1aa085024
percona/percona-xtradb-cluster:8.4.7-7.1	5b18775ad62a1c5f8d8bffc63a1518360d2e7a82c1bed7cbd8a15011f6cdff9f
percona/percona-xtradb-cluster:8.4.7-7.1 (ARM64)	4c3785f5befd001ca3ae035f42c9b586447b874158b0d9b26afb8ff87658829f
percona/percona-xtradb-cluster:8.0.44-35.1	f91026ec8427ace53dc31f3b00ec14cebdc0868bda921ae0713e8ad3af71ba1f
percona/percona-xtradb-cluster:8.0.44-35.1 (ARM64)	33a0f32c1d42cf6e74f45aeebd6422cfdea6c8c8bc3cce600e46c4661b0183be
percona/percona-xtradb-cluster:5.7.44-31.65	36fafdef46485839d4ff7c6dc73b4542b07031644c0152e911acb9734ff2be85
percona/percona-xtrabackup:8.4.0-5.1	1b81d06b1beb6a126b493d11532a5c71d1b1c2a1d13cb655e3cc5760c0896035
percona/percona-xtrabackup:8.4.0-5.1 (ARM64)	ca40d7975ae39bd5dd652487a1389b823cbf788e9948db6cf53ebb0d3f57c51b
percona/percona-xtrabackup:8.0.35-34.1	967bafa0823c90aa8fa9c25a9012be36b0deef64e255294a09148d77ce6aea68
percona/percona-xtrabackup:8.0.35-34.1 (ARM64)	83f814dca9ed398b585938baa86508bda796ba301e34c948a5106095d27bf86e
percona/percona-xtrabackup:2.4.29	11b92a7f7362379fc6b0de92382706153f2ac007ebf0d7ca25bac2c7303fdf10
percona/fluentbit:4.0.1-1	65bdf7d38cbceed6b6aa6412aea3fb4a196000ac6c66185f114a0a62c4a442ad
percona/fluentbit:4.0.1-1 (ARM64)	dabda77b298b67d30d7f53b5cdb7215ad19dabb22b9543e3fd8aedb74ab24733
percona/pmm-client:3.5.0	352aee74f25b3c1c4cd9dff1f378a0c3940b315e551d170c09953bf168531e4a
percona/pmm-client:3.5.0 (ARM64)	cbbb074d51d90a5f2d6f1d98a05024f6de2ffdcb5acab632324cea4349a820bd
percona/pmm-client:2.44.1-1	52a8fb5e8f912eef1ff8a117ea323c401e278908ce29928dafc23fac1db4f1e3
percona/pmm-client:2.44.1-1 (ARM64)	390bfd12f981e8b3890550c4927a3ece071377065e001894458047602c744e3b
percona/haproxy:2.8.17	ef8486b39a1e8dca97b5cdf1135e6282be1757ad188517b889d12c5a3470eeda
percona/haproxy:2.8.17 (ARM64)	bbc5b3b66ac985d1a4500195539e7dff5196245a5a842a6858ea0848ec089967
percona/proxysql2:2.7.3-1.2	719d0ab363c65c7f75431bbed7ec0d9f2af7e691765c489da954813c552359a2
percona/proxysql2:2.7.3-1.2 (ARM64)	4c4d094652c9f2eb097be5d92dcc05da61c9e8699ac7321def959d5a205a89f7
percona/proxysql3:3.0.1-1.2	f3fb43d4ef2467f207ecd66c51414520a100a0474807f307775a985303c56ec5
percona/proxysql3:3.0.1-1.2 (ARM64)	d21ba769b9e364a1a0c1d5e9d3b6287e8051efcf79cd6ec3df5756278961bbec

Find images for previous versions

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	MySQL	Percona XtraBackup	HA Proxy	ProxySQL
1.19.0	8.4, 8.0, 5.7	8.4.0-5.1 for MySQL 8.4, 8.0.35-34.1 for MySQL 8.0, 2.4.29 for MySQL 5.7	2.8.17	2.7.3-1.2
1.18.0	8.4 (Tech preview), 8.0, 5.7	8.4.0-3 for MySQL 8.4, 8.0.35-34.1 for MySQL 8.0, 2.4.29 for MySQL 5.7	2.8.15	2.7.3
1.17.0	8.4 (Tech preview), 8.0, 5.7	8.4.0-1 for MySQL 8.4, 8.0.35-32 for MySQL 8.0, 2.4.29 for MySQL 5.7	2.8.14	2.7.1-1
1.16.1	8.4 (Tech preview), 8.0, 5.7	8.4.0-1 for MySQL 8.4, 8.0.35-30.1 for MySQL 8.0, 2.4.29 for MySQL 5.7	2.8.11	2.7.1
1.16.0	8.4 (Tech preview), 8.0, 5.7	8.4.0-1 for MySQL 8.4, 8.0.35-30.1 for MySQL 8.0, 2.4.29 for MySQL 5.7	2.8.11	2.7.1
1.15.1	8.0, 5.7	8.0.35-30.1 for MySQL 8.0, 2.4.29-1 for MySQL 5.7	2.8.5	2.5.5
1.14.1	8.0, 5.7	8.0.35-30.1 for MySQL 8.0, 2.4.29-1 for MySQL 5.7	2.8.5-1	2.5.5-1.1
1.15.0	8.0, 5.7	8.0.35-30.1 for MySQL 8.0, 2.4.29-1 for MySQL 5.7	2.8.5	2.5.5
1.14.0	8.0, 5.7	8.0.35-30.1 for MySQL 8.0, 2.4.29-1 for MySQL 5.7	2.8.5-1	2.5.5-1.1
1.13.0	8.0, 5.7	8.0.32-26 for MySQL 8.0, 2.4.28 for MySQL 5.7	2.6.12	2.5.1-1.1
1.12.0	8.0, 5.7	8.0.30-23 for MySQL 8.0, 2.4.26 for MySQL 5.7	2.5.6	2.4.4
1.11.0	8.0, 5.7	8.0.27-19 for MySQL 8.0, 2.4.26 for MySQL 5.7	2.4.15	2.3.2
1.10.0	8.0, 5.7	8.0.23-16 for MySQL 8.0, 2.4.24 for MySQL 5.7	2.3.14	2.0.18
1.9.0	8.0, 5.7	8.0.23-16 for MySQL 8.0, 2.4.23 for MySQL 5.7	2.3.10	2.0.18
1.8.0	8.0, 5.7	8.0.23-16 for MySQL 8.0, 2.4.22 for MySQL 5.7	2.3.2	2.0.17
1.7.0	8.0, 5.7	8.0.22-15 for MySQL 8.0, 2.4.21 for MySQL 5.7	2.1.7	2.0.15
1.6.0	8.0, 5.7	8.0.14 for MySQL 8.0, 2.4.20 for MySQL 5.7	2.1.7	2.0.14
1.5.0	8.0, 5.7	8.0.13 for MySQL 8.0, 2.4.20 for MySQL 5.7	2.1.7	2.0.12
1.4.0	8.0, 5.7	8.0.11 for MySQL 8.0, 2.4.20 for MySQL 5.7	-	2.0.10
1.3.0	5.7	2.4.18	-	2.0.6
1.2.0	5.7	2.4.14	-	2.0.6
1.1.0	5.7	2.4.14	-	2.0.4

Platforms:

Operator	GKE	EKS	Openshift	AKS	Minikube
1.19.0	1.31 - 1.33	1.32 - 1.34	4.17 - 4.20	1.32 - 1.34	1.37.0
1.18.0	1.30 - 1.33	1.30 - 1.33	4.15 - 4.19	1.30 - 1.33	1.36.0

1.17.0	1.29 - 1.32	1.30 - 1.32	4.14 - 4.18	1.30 - 1.32	1.35.0
1.16.1	1.28 - 1.30	1.28 - 1.31	4.15.42 - 4.17.8	1.28 - 1.31	1.34.0
1.16.0	1.28 - 1.30	1.28 - 1.31	4.15.42 - 4.17.8	1.28 - 1.31	1.34.0
1.15.1	1.27 - 1.30	1.28 - 1.30	4.13.46 - 4.16.7	1.28 - 1.30	1.33.1
1.14.1	1.25 - 1.29	1.24 - 1.29	4.12.50 - 4.14.13	1.26 - 1.28	1.32.0
1.15.0	1.27 - 1.30	1.28 - 1.30	4.13.46 - 4.16.7	1.28 - 1.30	1.33.1
1.14.0	1.25 - 1.29	1.24 - 1.29	4.12.50 - 4.14.13	1.26 - 1.28	1.32.0
1.13.0	1.24 - 1.27	1.23 - 1.27	4.10 - 4.13	1.24 - 1.26	1.30
1.12.0	1.21 - 1.24	1.21 - 1.24	4.10 - 4.11	1.22 - 1.24	1.28
1.11.0	1.20 - 1.23	1.20 - 1.22	4.7 - 4.10	-	1.23
1.10.0	1.19 - 1.22	1.17 - 1.21	4.7 - 4.9	-	1.22
1.9.0	1.16, 1.20	1.19	3.11, 4.7	-	1.19
1.8.0	1.16, 1.20	1.19	3.11, 4.7	-	1.19
1.7.0	1.15, 1.17	1.15	3.11, 4.6	-	1.16
1.6.0	1.15, 1.17	1.15	3.11, 4.5	-	1.10
1.5.0	1.13, 1.15	1.15	3.11, 4.2	-	1.16
1.4.0	1.13, 1.15	1.15	3.11, 4.2	-	1.16
1.3.0	1.11, 1.14	-	3.11, 4.1	-	1.12
1.2.0	+	-	3.11	-	+
1.1.0	+	-	3.11	-	+

More detailed information about the cluster components for the current version of the Operator can be found in the system requirements and in the list of certified images. For previous releases of the Operator, you can check the same pages in the documentation archive.

Percona Operator for MySQL API Documentation

Percona Operator for MySQL based on Percona XtraDB Cluster provides an aggregation-layer extension for the Kubernetes API. Please refer to the official Kubernetes API documentation on the API access and usage details. The following subsections describe the Percona XtraDB Cluster API provided by the Operator.

Prerequisites

1. Create the namespace name you will use, if not exist:

   ```
   $ kubectl create namespace my-namespace-name
   ```

   Trying to create an already-existing namespace will show you a self-explanatory error message. Also, you can use the `default` namespace.

   Note

   In this document `default` namespace is used in all examples. Substitute `default` with your namespace name if you use a different one.

2. Prepare

   ```
   # set correct API address
   KUBE_CLUSTER=$(kubectl config view --minify -o jsonpath='{.clusters[0].name}')
   API_SERVER=$(kubectl config view -o jsonpath="{.clusters[?(@.name==\"$KUBE_CLUSTER\")].cluster.server}" | sed -e
   's#https://##')

   # create service account and get token
   kubectl apply --server-side -f deploy/crd.yaml -f deploy/rbac.yaml -n default
   KUBE_TOKEN=$(kubectl get secret $(kubectl get serviceaccount percona-xtradb-cluster-operator -o jsonpath='{.secrets[0].name}'
   -n default) -o jsonpath='{.data.token}' -n default | base64 --decode )
   ```

Create new Percona XtraDB Cluster

Description:

```
The command to create a new Percona XtraDB Cluster with all its resources
```

Kubectl Command:

```
$ kubectl apply -f percona-xtradb-cluster-operator/deploy/cr.yaml
```

URL:

```
https://$API_SERVER/apis/pxc.percona.com/v{{ apiversion }}/namespaces/default/perconaxtradbclusters
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XPOST "https://$API_SERVER/apis/pxc.percona.com/v{{ apiversion }}/namespaces/default/perconaxtradbclusters" \
            -H "Content-Type: application/json" \
            -H "Accept: application/json" \
            -H "Authorization: Bearer $KUBE_TOKEN" \
            -d "@cluster.json"
```

Request Body (cluster.json):

Example

```
{
   "apiVersion":"pxc.percona.com/v1-5-0",
```

```
"kind":"PerconaXtraDBCluster",
"metadata":{
   "name":"cluster1",
   "finalizers":[
      "delete-pxc-pods-in-order"
   ]
},
"spec":{
   "secretsName":"my-cluster-secrets",
   "vaultSecretName":"keyring-secret-vault",
   "sslSecretName":"my-cluster-ssl",
   "sslInternalSecretName":"my-cluster-ssl-internal",
   "allowUnsafeConfigurations":true,
   "pxc":{
      "size":3,
      "image":"percona/percona-xtradb-cluster:8.0.19-10.1",
      "resources":{
         "requests":null
      },
      "affinity":{
         "antiAffinityTopologyKey":"none"
      },
      "podDisruptionBudget":{
         "maxUnavailable":1
      },
      "volumeSpec":{
         "persistentVolumeClaim":{
            "resources":{
               "requests":{
                  "storage":"6Gi"
               }
            }
         }
      },
      "gracePeriod":600
   },
   "proxysql":{
      "enabled":true,
      "size":3,
      "image":"percona/percona-xtradb-cluster-operator:1.5.0-proxysql",
      "resources":{
         "requests":null
      },
      "affinity":{
         "antiAffinityTopologyKey":"none"
      },
      "volumeSpec":{
         "persistentVolumeClaim":{
            "resources":{
               "requests":{
                  "storage":"2Gi"
               }
            }
         }
      },
      "podDisruptionBudget":{
         "maxUnavailable":1
      },
      "gracePeriod":30
   },
   "pmm":{
      "enabled":false,
      "image":"percona/percona-xtradb-cluster-operator:1.5.0-pmm",
      "serverHost":"monitoring-service",
      "serverUser":"pmm"
   },
   "backup":{
      "image":"percona/percona-xtradb-cluster-operator:1.5.0-pxc8.0-backup",
      "serviceAccountName":"percona-xtradb-cluster-operator",
      "storages":{
         "s3-us-west":{
            "type":"s3",
            "s3":{
               "bucket":"S3-BACKUP-BUCKET-NAME-HERE",
               "credentialsSecret":"my-cluster-name-backup-s3",
               "region":"us-west-2"
            }
         },
         "fs-pvc":{
            "type":"filesystem",
            "volume":{
               "persistentVolumeClaim":{
                  "accessModes":[
                     "ReadWriteOnce"
                  ],
                  "resources":{
                     "requests":{
                        "storage":"6Gi"
                     }
                  }
               }
            }
         }
```

```
         },
         "schedule":[
            {
               "name":"sat-night-backup",
               "schedule":"0 0 * * 6",
               "keep":3,
               "storageName":"s3-us-west"
            },
            {
               "name":"daily-backup",
               "schedule":"0 0 * * *",
               "keep":5,
               "storageName":"fs-pvc"
            }
         ]
      }
   }
}
```

Inputs:

> **Metadata**:
>
> 1. Name (String, min-length: 1) : `contains name of cluster`
> 2. Finalizers (list of string, Default: [“delete-pxc-pods-in-order”]) `contains steps to do when deleting the cluster`
>
> **Spec**:
>
> 1. secretsName (String, min-length: 1) : `contains name of secret to create for the cluster`
> 2. vaultSecretName (String, min-length: 1) : `contains name of vault secret to create for the cluster`
> 3. sslInternalSecretName (String, min-length: 1) : `contains name of ssl secret to create for the cluster`
> 4. allowUnsafeConfigurations (Boolean, Default: false) : `allow unsafe configurations to run`
>
> pxc:
>
> 1. Size (Int , min-value: 1, default, 3) : `number of Percona XtraDB Cluster nodes to create`
> 2. Image (String, min-length: 1) : `contains image name to use for Percona XtraDB Cluster nodes`
> 3. volumeSpec : storage (SizeString, default: “6Gi”) : `contains the size for the storage volume of Percona XtraDB Cluster nodes`
> 4. gracePeriod (Int, default: 600, min-value: 0) : `contains the time to wait for Percona XtraDB Cluster node to shutdown in milliseconds`
>
> proxysql:
>
> 1. Enabled (Boolean, default: true) : `enabled or disables proxysql`
>
> pmm:
>
> 1. serverHost (String, min-length: 1) : `service name for monitoring`
> 2. serverUser (String, min-length: 1) : `name of pmm user`
> 3. image (String, min-length: 1) : `name of pmm image`
>
> backup:
>
> 1. Storages (Object) : `contains the storage destinations to save the backups in`
> 2. schedule:
>   a. name (String, min-length: 1) : `name of backup job`
>   b. schedule (String, Cron format: `"\* \* \* \* \*"`) : `contains cron schedule format for when to run cron jobs`
>   c. keep (Int, min-value = 1) : `number of backups to keep`
>   d. storageName (String, min-length: 1) : `name of storage object to use`

Response:

Example

```
{
   "apiVersion":"pxc.percona.com/v1-5-0",
   "kind":"PerconaXtraDBCluster",
   "metadata":{
      "creationTimestamp":"2020-05-27T22:23:58Z",
      "finalizers":[
         "delete-pxc-pods-in-order"
      ],
      "generation":1,
      "managedFields":[
         {
            "apiVersion":"pxc.percona.com/v1-5-0",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:metadata":{
                  "f:finalizers":{

                  }
               },
               "f:spec":{
                  ".":{

                  },
                  "f:allowUnsafeConfigurations":{

                  },
                  "f:backup":{
                     ".":{

                     },
                     "f:image":{

                     },
                     "f:schedule":{

                     },
                     "f:serviceAccountName":{

                     },
                     "f:storages":{
                        ".":{

                        },
                        "f:fs-pvc":{
                           ".":{

                           },
                           "f:type":{

                           },
                           "f:volume":{
                              ".":{

                              },
                              "f:persistentVolumeClaim":{
                                 ".":{

                                 },
                                 "f:accessModes":{

                                 },
                                 "f:resources":{
                                    ".":{

                                    },
                                    "f:requests":{
                                       ".":{

                                       },
                                       "f:storage":{

                                       }
                                    }
                                 }
                              }
                           }
                        },
                        "f:s3-us-west":{
                           ".":{

                           },
                           "f:s3":{
                              ".":{

                              },
                              "f:bucket":{

                              },
                              "f:credentialsSecret":{

                              },
                              "f:region":{
```

```
                }
            },
            "f:type":{

            }
         }
      }
   },
   "f:pmm":{
      ".":{

      },
      "f:enabled":{

      },
      "f:image":{

      },
      "f:serverHost":{

      },
      "f:serverUser":{

      }
   },
   "f:proxysql":{
      ".":{

      },
      "f:affinity":{
         ".":{

         },
         "f:antiAffinityTopologyKey":{

         }
      },
      "f:enabled":{

      },
      "f:gracePeriod":{

      },
      "f:image":{

      },
      "f:podDisruptionBudget":{
         ".":{

         },
         "f:maxUnavailable":{

         }
      },
      "f:resources":{
         ".":{

         },
         "f:requests":{

         }
      },
      "f:size":{

      },
      "f:volumeSpec":{
         ".":{

         },
         "f:persistentVolumeClaim":{
            ".":{

            },
            "f:resources":{
               ".":{

               },
               "f:requests":{
                  ".":{

                  },
                  "f:storage":{

                  }
               }
            }
         }
      }
   },
   "f:pxc":{
      ".":{
```

```
                    },
                    "f:affinity":{
                       ".":{

                       },
                       "f:antiAffinityTopologyKey":{

                       }
                    },
                    "f:gracePeriod":{

                    },
                    "f:image":{

                    },
                    "f:podDisruptionBudget":{
                       ".":{

                       },
                       "f:maxUnavailable":{

                       }
                    },
                    "f:resources":{
                       ".":{

                       },
                       "f:requests":{

                       }
                    },
                    "f:size":{

                    },
                    "f:volumeSpec":{
                       ".":{

                       },
                       "f:persistentVolumeClaim":{
                          ".":{

                          },
                          "f:resources":{
                             ".":{

                             },
                             "f:requests":{
                                ".":{

                                },
                                "f:storage":{

                                }
                             }
                          }
                       }
                    }
                 },
                 "f:secretsName":{

                 },
                 "f:sslInternalSecretName":{

                 },
                 "f:sslSecretName":{

                 },
                 "f:vaultSecretName":{

                 }
              }
           },
           "manager":"kubectl",
           "operation":"Update",
           "time":"2020-05-27T22:23:58Z"
        }
     ],
     "name":"cluster1",
     "namespace":"default",
     "resourceVersion":"8694",
     "selfLink":"/apis/pxc.percona.com/v1-5-0/namespaces/default/perconaxtradbclusters/cluster1",
     "uid":"e9115e2a-49df-4ebf-9dab-fa5a550208d3"
  },
  "spec":{
     "allowUnsafeConfigurations":false,
     "backup":{
        "image":"percona/percona-xtradb-cluster-operator:1.5.0-pxc8.0-backup",
        "schedule":[
           {
              "keep":3,
              "name":"sat-night-backup",
              "schedule":"0 0 * * 6",
              "storageName":"s3-us-west"
```

```
            },
            {
               "keep":5,
               "name":"daily-backup",
               "schedule":"0 0 * * *",
               "storageName":"fs-pvc"
            }
         ],
         "serviceAccountName":"percona-xtradb-cluster-operator",
         "storages":{
            "fs-pvc":{
               "type":"filesystem",
               "volume":{
                  "persistentVolumeClaim":{
                     "accessModes":[
                        "ReadWriteOnce"
                     ],
                     "resources":{
                        "requests":{
                           "storage":"6Gi"
                        }
                     }
                  }
               }
            },
            "s3-us-west":{
               "s3":{
                  "bucket":"S3-BACKUP-BUCKET-NAME-HERE",
                  "credentialsSecret":"my-cluster-name-backup-s3",
                  "region":"us-west-2"
               },
               "type":"s3"
            }
         }
      },
      "pmm":{
         "enabled":false,
         "image":"percona/percona-xtradb-cluster-operator:1.5.0-pmm",
         "serverHost":"monitoring-service",
         "serverUser":"pmm"
      },
      "proxysql":{
         "affinity":{
            "antiAffinityTopologyKey":"none"
         },
         "enabled":true,
         "gracePeriod":30,
         "image":"percona/percona-xtradb-cluster-operator:1.5.0-proxysql",
         "podDisruptionBudget":{
            "maxUnavailable":1
         },
         "resources":{
            "requests":null
         },
         "size":3,
         "volumeSpec":{
            "persistentVolumeClaim":{
               "resources":{
                  "requests":{
                     "storage":"2Gi"
                  }
               }
            }
         }
      },
      "pxc":{
         "affinity":{
            "antiAffinityTopologyKey":"none"
         },
         "gracePeriod":600,
         "image":"percona/percona-xtradb-cluster:8.0.19-10.1",
         "podDisruptionBudget":{
            "maxUnavailable":1
         },
         "resources":{
            "requests":null
         },
         "size":3,
         "volumeSpec":{
            "persistentVolumeClaim":{
               "resources":{
                  "requests":{
                     "storage":"6Gi"
                  }
               }
            }
         }
      },
      "secretsName":"my-cluster-secrets",
      "sslInternalSecretName":"my-cluster-ssl-internal",
      "sslSecretName":"my-cluster-ssl",
      "vaultSecretName":"keyring-secret-vault"
   }
```

```
}
```

List Percona XtraDB Clusters

Description:

```
Lists all Percona XtraDB Clusters that exist in your kubernetes cluster
```

Kubectl Command:

```
$ kubectl get pxc
```

URL:

```
https://$API_SERVER/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters?limit=500
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XGET "https://$API_SERVER/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters?limit=500" \
            -H "Accept:
application/json;as=Table;v=v1;g=meta.k8s.io,application/json;as=Table;v=v1beta1;g=meta.k8s.io,application/json" \
            -H "Authorization: Bearer $KUBE_TOKEN"
```

Request Body:

```
None
```

Response:

Example


```
{
   "kind":"Table",
   "apiVersion":"meta.k8s.io/v1",
   "metadata":{
      "selfLink":"/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters",
      "resourceVersion":"10528"
   },
   "columnDefinitions":[
      {
         "name":"Name",
         "type":"string",
         "format":"name",
         "description":"Name must be unique within a namespace. Is required when creating resources, although some resources may allow a client to
request the generation of an appropriate name automatically. Name is primarily intended for creation idempotence and configuration definition.
Cannot be updated. More info: http://kubernetes.io/docs/user-guide/identifiers#names",
         "priority":0
      },
      {
         "name":"Endpoint",
         "type":"string",
         "format":"",
         "description":"Custom resource definition column (in JSONPath format): .status.host",
         "priority":0
      },
      {
         "name":"Status",
         "type":"string",
         "format":"",
         "description":"Custom resource definition column (in JSONPath format): .status.state",
         "priority":0
      },
      {
         "name":"PXC",
         "type":"string",
         "format":"",
         "description":"Ready pxc nodes",
         "priority":0
      },
      {
```

```
        "name":"proxysql",
        "type":"string",
        "format":"",
        "description":"Ready pxc nodes",
        "priority":0
      },
      {
        "name":"Age",
        "type":"date",
        "format":"",
        "description":"Custom resource definition column (in JSONPath format): .metadata.creationTimestamp",
        "priority":0
      }
   ],
   "rows":[
      {
         "cells":[
            "cluster1",
            "cluster1-proxysql.default",
            "ready",
            "3",
            "3",
            "8m37s"
         ],
         "object":{
            "kind":"PartialObjectMetadata",
            "apiVersion":"meta.k8s.io/v1",
            "metadata":{
               "name":"cluster1",
               "namespace":"default",
               "selfLink":"/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters/cluster1",
               "uid":"e9115e2a-49df-4ebf-9dab-fa5a550208d3",
               "resourceVersion":"10517",
               "generation":1,
               "creationTimestamp":"2020-05-27T22:23:58Z",
               "finalizers":[
                  "delete-pxc-pods-in-order"
               ],
               "managedFields":[
                  {
                     "manager":"kubectl",
                     "operation":"Update",
                     "apiVersion":"pxc.percona.com/v1-5-0",
                     "time":"2020-05-27T22:23:58Z",
                     "fieldsType":"FieldsV1",
                     "fieldsV1":{
                        "f:metadata":{
                           "f:finalizers":{

                           }
                        },
                        "f:spec":{
                           ".":{

                           },
                           "f:allowUnsafeConfigurations":{

                           },
                           "f:backup":{
                              ".":{

                              },
                              "f:image":{

                              },
                              "f:schedule":{

                              },
                              "f:serviceAccountName":{

                              },
                              "f:storages":{
                                 ".":{

                                 },
                                 "f:fs-pvc":{
                                    ".":{

                                    },
                                    "f:type":{

                                    },
                                    "f:volume":{
                                       ".":{

                                       },
                                       "f:persistentVolumeClaim":{
                                          ".":{

                                          },
                                          "f:accessModes":{

                                          },
```

```
                        "f:resources":{
                           ".":{

                           },
                           "f:requests":{
                              ".":{

                              },
                              "f:storage":{

                              }
                           }
                        }
                     }
                  }
               },
               "f:s3-us-west":{
                  ".":{

                  },
                  "f:s3":{
                     ".":{

                     },
                     "f:bucket":{

                     },
                     "f:credentialsSecret":{

                     },
                     "f:region":{

                     }
                  },
                  "f:type":{

                  }
               }
            }
         },
         "f:pmm":{
            ".":{

            },
            "f:image":{

            },
            "f:serverHost":{

            },
            "f:serverUser":{

            }
         },
         "f:proxysql":{
            ".":{

            },
            "f:affinity":{
               ".":{

               },
               "f:antiAffinityTopologyKey":{

               }
            },
            "f:enabled":{

            },
            "f:gracePeriod":{

            },
            "f:image":{

            },
            "f:podDisruptionBudget":{
               ".":{

               },
               "f:maxUnavailable":{

               }
            },
            "f:resources":{

            },
            "f:size":{

            },
            "f:volumeSpec":{
               ".":{

               },
```

```
                        "f:persistentVolumeClaim":{
                           ".":{

                           },
                           "f:resources":{
                              ".":{

                              },
                              "f:requests":{
                                 ".":{

                                 },
                                 "f:storage":{

                                 }
                              }
                           }
                        }
                     }
                  },
                  "f:pxc":{
                     ".":{

                     },
                     "f:affinity":{
                        ".":{

                        },
                        "f:antiAffinityTopologyKey":{

                        }
                     },
                     "f:gracePeriod":{

                     },
                     "f:image":{

                     },
                     "f:podDisruptionBudget":{
                        ".":{

                        },
                        "f:maxUnavailable":{

                        }
                     },
                     "f:resources":{

                     },
                     "f:size":{

                     },
                     "f:volumeSpec":{
                        ".":{

                        },
                        "f:persistentVolumeClaim":{
                           ".":{

                           },
                           "f:resources":{
                              ".":{

                              },
                              "f:requests":{
                                 ".":{

                                 },
                                 "f:storage":{

                                 }
                              }
                           }
                        }
                     }
                  },
                  "f:secretsName":{

                  },
                  "f:sslInternalSecretName":{

                  },
                  "f:sslSecretName":{

                  },
                  "f:vaultSecretName":{

                  }
               }
            }
         }
      },
      {
         "manager":"percona-xtradb-cluster-operator",
```

```
                  "operation":"Update",
                  "apiVersion":"pxc.percona.com/v1",
                  "time":"2020-05-27T22:32:31Z",
                  "fieldsType":"FieldsV1",
                  "fieldsV1":{
                     "f:spec":{
                        "f:backup":{
                           "f:storages":{
                              "f:fs-pvc":{
                                 "f:podSecurityContext":{
                                    ".":{

                                    },
                                    "f:fsGroup":{

                                    },
                                    "f:supplementalGroups":{

                                    }
                                 },
                                 "f:s3":{
                                    ".":{

                                    },
                                    "f:bucket":{

                                    },
                                    "f:credentialsSecret":{

                                    }
                                 }
                              },
                              "f:s3-us-west":{
                                 "f:podSecurityContext":{
                                    ".":{

                                    },
                                    "f:fsGroup":{

                                    },
                                    "f:supplementalGroups":{

                                    }
                                 }
                              }
                           }
                        },
                        "f:pmm":{
                           "f:resources":{

                           }
                        },
                        "f:proxysql":{
                           "f:podSecurityContext":{
                              ".":{

                              },
                              "f:fsGroup":{

                              },
                              "f:supplementalGroups":{

                              }
                           },
                           "f:sslInternalSecretName":{

                           },
                           "f:sslSecretName":{

                           },
                           "f:volumeSpec":{
                              "f:persistentVolumeClaim":{
                                 "f:accessModes":{

                                 }
                              }
                           }
                        },
                        "f:pxc":{
                           "f:podSecurityContext":{
                              ".":{

                              },
                              "f:fsGroup":{

                              },
                              "f:supplementalGroups":{

                              }
                           },
                           "f:sslInternalSecretName":{

                           },
```

```
                            "f:sslSecretName":{

                            },
                            "f:vaultSecretName":{

                            },
                            "f:volumeSpec":{
                               "f:persistentVolumeClaim":{
                                  "f:accessModes":{

                                  }
                               }
                            }
                         }
                      },
                      "f:status":{
                         ".":{

                         },
                         "f:conditions":{

                         },
                         "f:host":{

                         },
                         "f:observedGeneration":{

                         },
                         "f:proxysql":{
                            ".":{

                            },
                            "f:ready":{

                            },
                            "f:size":{

                            },
                            "f:status":{

                            }
                         },
                         "f:pxc":{
                            ".":{

                            },
                            "f:ready":{

                            },
                            "f:size":{

                            },
                            "f:status":{

                            }
                         },
                         "f:state":{

                         }
                      }
                   }
                }
             ]
          }
       }
    ]
}
```


Get status of Percona XtraDB Cluster

Description:

```
Gets all information about the specified Percona XtraDB Cluster
```

Kubectl Command:

```
$ kubectl get pxc/cluster1 -o json
```

URL:

```
https://$API_SERVER/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters/cluster1
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XGET "https://$API_SERVER/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters/cluster1" \
            -H "Accept: application/json" \
            -H "Authorization: Bearer $KUBE_TOKEN"
```

Request Body:

```
None
```

Response:

Example


```
{
   "apiVersion":"pxc.percona.com/v1",
   "kind":"PerconaXtraDBCluster",
   "metadata":{
      "annotations":{
         "kubectl.kubernetes.io/last-applied-configuration":"{\"apiVersion\":\"pxc.percona.com/v1\",\"kind\":\"PerconaXtraDBCluster\",\"metadata\":
{\"annotations\":{},\"creationTimestamp\":\"2020-05-27T22:23:58Z\",\"finalizers\":[\"delete-pxc-pods-in-
order\"],\"generation\":1,\"managedFields\":[{\"apiVersion\":\"pxc.percona.com/v1-5-0\",\"fieldsType\":\"FieldsV1\",\"fieldsV1\":{\"f:metadata\":
{\"f:finalizers\":{}},\"f:spec\":{\".\":{},\"f:allowUnsafeConfigurations\":{},\"f:backup\":{\".\":{},\"f:image\":{},\"f:schedule\":
{},\"f:serviceAccountName\":{},\"f:storages\":{\".\":{},\"f:fs-pvc\":{\".\":{},\"f:type\":{},\"f:volume\":{\".\":{},\"f:persistentVolumeClaim\":
{\".\":{},\"f:accessModes\":{},\"f:resources\":{\".\":{},\"f:requests\":{\".\":{},\"f:storage\":{}}}}}},\"f:s3-us-west\":{\".\":{},\"f:s3\":{\".\":
{},\"f:bucket\":{},\"f:credentialsSecret\":{},\"f:region\":{}},\"f:type\":{}}}},\"f:pmm\":{\".\":{},\"f:image\":{},\"f:serverHost\":
{},\"f:serverUser\":{}},\"f:proxysql\":{\".\":{},\"f:affinity\":{\".\":{},\"f:antiAffinityTopologyKey\":{}},\"f:enabled\":{},\"f:gracePeriod\":
{},\"f:image\":{},\"f:podDisruptionBudget\":{\".\":{},\"f:maxUnavailable\":{}},\"f:resources\":{},\"f:size\":{},\"f:volumeSpec\":{\".\":
{},\"f:persistentVolumeClaim\":{\".\":{},\"f:resources\":{\".\":{},\"f:requests\":{\".\":{},\"f:storage\":{}}}}}},\"f:pxc\":{\".\":
{},\"f:affinity\":{\".\":{},\"f:antiAffinityTopologyKey\":{}},\"f:gracePeriod\":{},\"f:image\":{},\"f:podDisruptionBudget\":{\".\":
{},\"f:maxUnavailable\":{}},\"f:resources\":{},\"f:size\":{},\"f:volumeSpec\":{\".\":{},\"f:persistentVolumeClaim\":{\".\":{},\"f:resources\":
{\".\":{},\"f:requests\":{\".\":{},\"f:storage\":{}}}}},\"f:secretsName\":{},\"f:sslInternalSecretName\":{},\"f:sslSecretName\":
{},\"f:vaultSecretName\":{}}},\"manager\":\"kubectl\",\"operation\":\"Update\",\"time\":\"2020-05-27T22:23:58Z\"},
{\"apiVersion\":\"pxc.percona.com/v1\",\"fieldsType\":\"FieldsV1\",\"fieldsV1\":{\"f:spec\":{\"f:backup\":{\"f:storages\":{\"f:fs-pvc\":
{\"f:podSecurityContext\":{\".\":{},\"f:fsGroup\":{},\"f:supplementalGroups\":{}},\"f:s3\":{\".\":{},\"f:bucket\":{},\"f:credentialsSecret\":
{}}},\"f:s3-us-west\":{\"f:podSecurityContext\":{\".\":{},\"f:fsGroup\":{},\"f:supplementalGroups\":{}}}}},\"f:pmm\":{\"f:resources\":
{}},\"f:proxysql\":{\"f:podSecurityContext\":{\".\":{},\"f:fsGroup\":{},\"f:supplementalGroups\":{}},\"f:sslInternalSecretName\":
{},\"f:sslSecretName\":{},\"f:volumeSpec\":{\"f:persistentVolumeClaim\":{\"f:accessModes\":{}}}},\"f:pxc\":{\"f:podSecurityContext\":{\".\":
{},\"f:fsGroup\":{},\"f:supplementalGroups\":{}},\"f:sslInternalSecretName\":{},\"f:sslSecretName\":{},\"f:vaultSecretName\":{},\"f:volumeSpec\":
{\"f:persistentVolumeClaim\":{\"f:accessModes\":{}}}},\"f:status\":{\".\":{},\"f:conditions\":{},\"f:host\":{},\"f:observedGeneration\":
{},\"f:proxysql\":{\".\":{},\"f:ready\":{},\"f:size\":{},\"f:status\":{}},\"f:pxc\":{\".\":{},\"f:ready\":{},\"f:size\":{},\"f:status\":
{}},\"f:state\":{}}},\"manager\":\"percona-xtradb-cluster-operator\",\"operation\":\"Update\",\"time\":\"2020-05-
27T23:06:47Z\"}],\"name\":\"cluster1\",\"namespace\":\"default\",\"resourceVersion\":\"15878\",\"selfLink\":\"/apis/pxc.percona.com/v1/namespaces/d
efault/perconaxtradbclusters/cluster1\",\"uid\":\"e9115e2a-49df-4ebf-9dab-fa5a550208d3\"},\"spec\":{\"allowUnsafeConfigurations\":true,\"backup\":
{\"image\":\"percona/percona-xtradb-cluster-operator:1.5.0-pxc8.0-debug-backup\",\"schedule\":[{\"keep\":3,\"name\":\"sat-night-
backup\",\"schedule\":\"0 0 * * 6\",\"storageName\":\"s3-us-west\"},{\"keep\":5,\"name\":\"daily-backup\",\"schedule\":\"0 0 * *
*\",\"storageName\":\"fs-pvc\"}],\"serviceAccountName\":\"percona-xtradb-cluster-operator\",\"storages\":{\"fs-pvc\":
{\"type\":\"filesystem\",\"volume\":{\"persistentVolumeClaim\":{\"accessModes\":[\"ReadWriteOnce\"],\"resources\":{\"requests\":
{\"storage\":\"6Gi\"}}}}},\"s3-us-west\":{\"s3\":{\"bucket\":\"S3-BACKUP-BUCKET-NAME-HERE\",\"credentialsSecret\":\"my-cluster-name-backup-
s3\",\"region\":\"us-west-2\"},\"type\":\"s3\"}}},\"pmm\":{\"enabled\":false,\"image\":\"percona/percona-xtradb-cluster-operator:1.5.0-
pmm\",\"serverHost\":\"monitoring-service\",\"serverUser\":\"pmm\"},\"proxysql\":{\"affinity\":
{\"antiAffinityTopologyKey\":\"none\"},\"enabled\":true,\"gracePeriod\":30,\"image\":\"percona/percona-xtradb-cluster-operator:1.5.0-
proxysql\",\"podDisruptionBudget\":{\"maxUnavailable\":1},\"resources\":{\"requests\":null},\"size\":5,\"volumeSpec\":{\"persistentVolumeClaim\":
{\"resources\":{\"requests\":{\"storage\":\"2Gi\"}}}}},\"pxc\":{\"affinity\":
{\"antiAffinityTopologyKey\":\"none\"},\"gracePeriod\":600,\"image\":\"percona/percona-xtradb-cluster:8.0.19-10.1\",\"podDisruptionBudget\":
{\"maxUnavailable\":1},\"resources\":{\"requests\":null},\"size\":5,\"volumeSpec\":{\"persistentVolumeClaim\":{\"resources\":{\"requests\":
{\"storage\":\"6Gi\"}}}},\"secretsName\":\"my-cluster-secrets\",\"sslInternalSecretName\":\"my-cluster-ssl-internal\",\"sslSecretName\":\"my-
cluster-ssl\",\"vaultSecretName\":\"keyring-secret-vault\"},\"status\":{\"conditions\":[{\"lastTransitionTime\":\"2020-05-
27T22:23:58Z\",\"status\":\"True\",\"type\":\"Initializing\"},{\"lastTransitionTime\":\"2020-05-
27T22:25:43Z\",\"status\":\"True\",\"type\":\"Ready\"}],\"host\":\"cluster1-proxysql.default\",\"observedGeneration\":1,\"proxysql\":
{\"ready\":3,\"size\":5,\"status\":\"ready\"},\"pxc\":{\"ready\":3,\"size\":5,\"status\":\"ready\"},\"state\":\"ready\"}}\n"
      },
      "creationTimestamp":"2020-05-27T22:23:58Z",
      "finalizers":[
         "delete-pxc-pods-in-order"
      ],
      "generation":6,
      "managedFields":[
         {
            "apiVersion":"pxc.percona.com/v1-5-0",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:metadata":{
                  "f:finalizers":{

                  }
               },
               "f:spec":{
                  ".":{
```

```
                },
                "f:allowUnsafeConfigurations":{

                },
                "f:backup":{
                   ".":{

                   },
                   "f:schedule":{

                   },
                   "f:serviceAccountName":{

                   },
                   "f:storages":{
                      ".":{

                      },
                      "f:fs-pvc":{
                         ".":{

                         },
                         "f:type":{

                         },
                         "f:volume":{
                            ".":{

                            },
                            "f:persistentVolumeClaim":{
                               ".":{

                               },
                               "f:accessModes":{

                               },
                               "f:resources":{
                                  ".":{

                                  },
                                  "f:requests":{
                                     ".":{

                                     },
                                     "f:storage":{

                                     }
                                  }
                               }
                            }
                         }
                      },
                      "f:s3-us-west":{
                         ".":{

                         },
                         "f:s3":{
                            ".":{

                            },
                            "f:bucket":{

                            },
                            "f:credentialsSecret":{

                            },
                            "f:region":{

                            }
                         },
                         "f:type":{

                         }
                      }
                   }
                },
                "f:pmm":{
                   ".":{

                   },
                   "f:image":{

                   },
                   "f:serverHost":{

                   },
                   "f:serverUser":{

                   }
                },
                "f:proxysql":{
                   ".":{
```

```
      },
      "f:affinity":{
         ".":{

         },
         "f:antiAffinityTopologyKey":{

         }
      },
      "f:enabled":{

      },
      "f:gracePeriod":{

      },
      "f:image":{

      },
      "f:podDisruptionBudget":{
         ".":{

         },
         "f:maxUnavailable":{

         }
      },
      "f:resources":{

      },
      "f:volumeSpec":{
         ".":{

         },
         "f:persistentVolumeClaim":{
            ".":{

            },
            "f:resources":{
               ".":{

               },
               "f:requests":{
                  ".":{

                  },
                  "f:storage":{

                  }
               }
            }
         }
      }
   },
   "f:pxc":{
      ".":{

      },
      "f:affinity":{
         ".":{

         },
         "f:antiAffinityTopologyKey":{

         }
      },
      "f:gracePeriod":{

      },
      "f:podDisruptionBudget":{
         ".":{

         },
         "f:maxUnavailable":{

         }
      },
      "f:resources":{

      },
      "f:volumeSpec":{
         ".":{

         },
         "f:persistentVolumeClaim":{
            ".":{

            },
            "f:resources":{
               ".":{

               },
               "f:requests":{
                  ".":{
```

```
                        },
                        "f:storage":{

                        }
                     }
                  }
               }
            }
         },
         "f:secretsName":{

         },
         "f:sslInternalSecretName":{

         },
         "f:sslSecretName":{

         },
         "f:vaultSecretName":{

         }
      }
   },
   "manager":"kubectl",
   "operation":"Update",
   "time":"2020-05-27T22:23:58Z"
},
{
   "apiVersion":"pxc.percona.com/v1",
   "fieldsType":"FieldsV1",
   "fieldsV1":{
      "f:metadata":{
         "f:annotations":{
            ".":{

            },
            "f:kubectl.kubernetes.io/last-applied-configuration":{

            }
         }
      },
      "f:spec":{
         "f:backup":{
            "f:image":{

            }
         },
         "f:proxysql":{
            "f:size":{

            }
         },
         "f:pxc":{
            "f:image":{

            },
            "f:size":{

            }
         }
      }
   },
   "manager":"kubectl",
   "operation":"Update",
   "time":"2020-05-27T23:38:49Z"
},
{
   "apiVersion":"pxc.percona.com/v1",
   "fieldsType":"FieldsV1",
   "fieldsV1":{
      "f:spec":{
         "f:backup":{
            "f:storages":{
               "f:fs-pvc":{
                  "f:podSecurityContext":{
                     ".":{

                     },
                     "f:fsGroup":{

                     },
                     "f:supplementalGroups":{

                     }
                  },
                  "f:s3":{
                     ".":{

                     },
                     "f:bucket":{

                     },
```

```
                        "f:credentialsSecret":{

                        }
                     }
                  },
                  "f:s3-us-west":{
                     "f:podSecurityContext":{
                        ".":{

                        },
                        "f:fsGroup":{

                        },
                        "f:supplementalGroups":{

                        }
                     }
                  }
               }
            },
            "f:pmm":{
               "f:resources":{

               }
            },
            "f:proxysql":{
               "f:podSecurityContext":{
                  ".":{

                  },
                  "f:fsGroup":{

                  },
                  "f:supplementalGroups":{

                  }
               },
               "f:sslInternalSecretName":{

               },
               "f:sslSecretName":{

               },
               "f:volumeSpec":{
                  "f:persistentVolumeClaim":{
                     "f:accessModes":{

                     }
                  }
               }
            },
            "f:pxc":{
               "f:podSecurityContext":{
                  ".":{

                  },
                  "f:fsGroup":{

                  },
                  "f:supplementalGroups":{

                  }
               },
               "f:sslInternalSecretName":{

               },
               "f:sslSecretName":{

               },
               "f:vaultSecretName":{

               },
               "f:volumeSpec":{
                  "f:persistentVolumeClaim":{
                     "f:accessModes":{

                     }
                  }
               }
            }
         },
         "f:status":{
            ".":{

            },
            "f:conditions":{

            },
            "f:host":{

            },
            "f:message":{
```

```
                },
                "f:observedGeneration":{

                },
                "f:proxysql":{
                   ".":{

                   },
                   "f:ready":{

                   },
                   "f:size":{

                   },
                   "f:status":{

                   }
                },
                "f:pxc":{
                   ".":{

                   },
                   "f:message":{

                   },
                   "f:ready":{

                   },
                   "f:size":{

                   },
                   "f:status":{

                   }
                },
                "f:state":{

                }
             }
          },
          "manager":"percona-xtradb-cluster-operator",
          "operation":"Update",
          "time":"2020-05-28T10:42:00Z"
       }
    ],
    "name":"cluster1",
    "namespace":"default",
    "resourceVersion":"35660",
    "selfLink":"/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters/cluster1",
    "uid":"e9115e2a-49df-4ebf-9dab-fa5a550208d3"
 },
 "spec":{
    "allowUnsafeConfigurations":true,
    "backup":{
       "image":"percona/percona-xtradb-cluster-operator:1.5.0-pxc8.0-debug-backup",
       "schedule":[
          {
             "keep":3,
             "name":"sat-night-backup",
             "schedule":"0 0 * * 6",
             "storageName":"s3-us-west"
          },
          {
             "keep":5,
             "name":"daily-backup",
             "schedule":"0 0 * * *",
             "storageName":"fs-pvc"
          }
       ],
       "serviceAccountName":"percona-xtradb-cluster-operator",
       "storages":{
          "fs-pvc":{
             "type":"filesystem",
             "volume":{
                "persistentVolumeClaim":{
                   "accessModes":[
                      "ReadWriteOnce"
                   ],
                   "resources":{
                      "requests":{
                         "storage":"6Gi"
                      }
                   }
                }
             }
          },
          "s3-us-west":{
             "s3":{
                "bucket":"S3-BACKUP-BUCKET-NAME-HERE",
                "credentialsSecret":"my-cluster-name-backup-s3",
                "region":"us-west-2"
             },
             "type":"s3"
```

```
            }
         }
      },
      "pmm":{
         "enabled":false,
         "image":"percona/percona-xtradb-cluster-operator:1.5.0-pmm",
         "serverHost":"monitoring-service",
         "serverUser":"pmm"
      },
      "proxysql":{
         "affinity":{
            "antiAffinityTopologyKey":"none"
         },
         "enabled":true,
         "gracePeriod":30,
         "image":"percona/percona-xtradb-cluster-operator:1.5.0-proxysql",
         "podDisruptionBudget":{
            "maxUnavailable":1
         },
         "resources":{

         },
         "size":3,
         "volumeSpec":{
            "persistentVolumeClaim":{
               "resources":{
                  "requests":{
                     "storage":"2Gi"
                  }
               }
            }
         }
      },
      "pxc":{
         "affinity":{
            "antiAffinityTopologyKey":"none"
         },
         "gracePeriod":600,
         "image":"percona/percona-xtradb-cluster-operator:1.5.0-pxc8.0-debug",
         "podDisruptionBudget":{
            "maxUnavailable":1
         },
         "resources":{

         },
         "size":3,
         "volumeSpec":{
            "persistentVolumeClaim":{
               "resources":{
                  "requests":{
                     "storage":"6Gi"
                  }
               }
            }
         }
      },
      "secretsName":"my-cluster-secrets",
      "sslInternalSecretName":"my-cluster-ssl-internal",
      "sslSecretName":"my-cluster-ssl",
      "vaultSecretName":"keyring-secret-vault"
   },
   "status":{
      "conditions":[
         {
            "lastTransitionTime":"2020-05-27T22:25:43Z",
            "status":"True",
            "type":"Ready"
         },
         {
            "lastTransitionTime":"2020-05-27T23:06:48Z",
            "status":"True",
            "type":"Initializing"
         },
         {
            "lastTransitionTime":"2020-05-27T23:08:58Z",
            "message":"ProxySQL upgrade error: context deadline exceeded",
            "reason":"ErrorReconcile",
            "status":"True",
            "type":"Error"
         },
         {
            "lastTransitionTime":"2020-05-27T23:08:59Z",
            "status":"True",
            "type":"Initializing"
         },
         {
            "lastTransitionTime":"2020-05-27T23:29:59Z",
            "status":"True",
            "type":"Ready"
         },
         {
            "lastTransitionTime":"2020-05-27T23:30:04Z",
            "status":"True",
```

```
            "type":"Initializing"
         },
         {
            "lastTransitionTime":"2020-05-27T23:35:27Z",
            "status":"True",
            "type":"Ready"
         },
         {
            "lastTransitionTime":"2020-05-27T23:35:42Z",
            "status":"True",
            "type":"Initializing"
         },
         {
            "lastTransitionTime":"2020-05-27T23:47:00Z",
            "status":"True",
            "type":"Ready"
         },
         {
            "lastTransitionTime":"2020-05-27T23:47:05Z",
            "status":"True",
            "type":"Initializing"
         },
         {
            "lastTransitionTime":"2020-05-28T09:58:25Z",
            "status":"True",
            "type":"Ready"
         },
         {
            "lastTransitionTime":"2020-05-28T09:58:31Z",
            "status":"True",
            "type":"Initializing"
         },
         {
            "lastTransitionTime":"2020-05-28T10:03:54Z",
            "status":"True",
            "type":"Ready"
         },
         {
            "lastTransitionTime":"2020-05-28T10:04:14Z",
            "status":"True",
            "type":"Initializing"
         },
         {
            "lastTransitionTime":"2020-05-28T10:15:28Z",
            "status":"True",
            "type":"Ready"
         },
         {
            "lastTransitionTime":"2020-05-28T10:15:38Z",
            "status":"True",
            "type":"Initializing"
         },
         {
            "lastTransitionTime":"2020-05-28T10:26:56Z",
            "status":"True",
            "type":"Ready"
         },
         {
            "lastTransitionTime":"2020-05-28T10:27:01Z",
            "status":"True",
            "type":"Initializing"
         },
         {
            "lastTransitionTime":"2020-05-28T10:38:28Z",
            "status":"True",
            "type":"Ready"
         },
         {
            "lastTransitionTime":"2020-05-28T10:38:33Z",
            "status":"True",
            "type":"Initializing"
         }
      ],
      "host":"cluster1-proxysql.default",
      "message":[
         "PXC: pxc: back-off 5m0s restarting failed container=pxc pod=cluster1-pxc-1_default(5b9b16e6-d0f8-4c97-a2d0-294feb9d014b); pxc: back-off
5m0s restarting failed container=pxc pod=cluster1-pxc-2_default(b8ebedd7-42f0-440b-aa5e-509d28926a5e); pxc: back-off 5m0s restarting failed
container=pxc pod=cluster1-pxc-4_default(2dce12f2-9ebc-419c-a92a-9cec68912004); "
      ],
      "observedGeneration":6,
      "proxysql":{
         "ready":3,
         "size":3,
         "status":"ready"
      },
      "pxc":{
         "message":"pxc: back-off 5m0s restarting failed container=pxc pod=cluster1-pxc-1_default(5b9b16e6-d0f8-4c97-a2d0-294feb9d014b); pxc: back-
off 5m0s restarting failed container=pxc pod=cluster1-pxc-2_default(b8ebedd7-42f0-440b-aa5e-509d28926a5e); pxc: back-off 5m0s restarting failed
container=pxc pod=cluster1-pxc-4_default(2dce12f2-9ebc-419c-a92a-9cec68912004); ",
         "ready":2,
         "size":3,
         "status":"initializing"
      },
```

```
      "state":"initializing"
   }
}
```

Scale up/down Percona XtraDB Cluster

Description:

```
Increase or decrease the size of the Percona XtraDB Cluster nodes to fit the
current high availability needs
```

Kubectl Command:

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
"spec": {"pxc":{ "size": "5" }
}}'
```

URL:

```
https://$API_SERVER/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters/cluster1
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XPATCH "https://$API_SERVER/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters/cluster1" \
            -H "Authorization: Bearer $KUBE_TOKEN" \
            -H "Content-Type: application/merge-patch+json"
            -H "Accept: application/json" \
            -d '{
                  "spec": {"pxc":{ "size": "5" }
                  }}'
```

Request Body:

Example

```
{
"spec": {"pxc":{ "size": "5" }
}}
```

Input:

> **spec**:
>
> pxc
>
> 1. size (Int or String, Defaults: 3): `Specify the size of the Percona XtraDB Cluster to scale up or down to`

Response:

Example


```
{
   "apiVersion":"pxc.percona.com/v1",
   "kind":"PerconaXtraDBCluster",
   "metadata":{
      "annotations":{
         "kubectl.kubernetes.io/last-applied-configuration":"{\"apiVersion\":\"pxc.percona.com/v1-5-
0\",\"kind\":\"PerconaXtraDBCluster\",\"metadata\":{\"annotations\":{},\"finalizers\":[\"delete-pxc-pods-in-
order\"],\"name\":\"cluster1\",\"namespace\":\"default\"},\"spec\":{\"allowUnsafeConfigurations\":true,\"backup\":{\"image\":\"percona/percona-
xtradb-cluster-operator:1.5.0-pxc8.0-backup\",\"schedule\":[{\"keep\":3,\"name\":\"sat-night-backup\",\"schedule\":\"0 0 * *
6\",\"storageName\":\"s3-us-west\"},{\"keep\":5,\"name\":\"daily-backup\",\"schedule\":\"0 0 * * *\",\"storageName\":\"fs-
pvc\"}],\"serviceAccountName\":\"percona-xtradb-cluster-operator\",\"storages\":{\"fs-pvc\":{\"type\":\"filesystem\",\"volume\":
{\"persistentVolumeClaim\":{\"accessModes\":[\"ReadWriteOnce\"],\"resources\":{\"requests\":{\"storage\":\"6Gi\"}}}}},\"s3-us-west\":{\"s3\":
{\"bucket\":\"S3-BACKUP-BUCKET-NAME-HERE\",\"credentialsSecret\":\"my-cluster-name-backup-s3\",\"region\":\"us-west-
2\"},\"type\":\"s3\"}}},\"pmm\":{\"enabled\":false,\"image\":\"percona/percona-xtradb-cluster-operator:1.5.0-pmm\",\"serverHost\":\"monitoring-
```

```
service\",\"serverUser\":\"pmm\"},\"proxysql\":{\"affinity\":
{\"antiAffinityTopologyKey\":\"none\"},\"enabled\":true,\"gracePeriod\":30,\"image\":\"percona/percona-xtradb-cluster-operator:1.5.0-
proxysql\",\"podDisruptionBudget\":{\"maxUnavailable\":1},\"resources\":{\"requests\":null},\"size\":3,\"volumeSpec\":{\"persistentVolumeClaim\":
{\"resources\":{\"requests\":{\"storage\":\"2Gi\"}}}}},\"pxc\":{\"affinity\":
{\"antiAffinityTopologyKey\":\"none\"},\"gracePeriod\":600,\"image\":\"percona/percona-xtradb-cluster:8.0.19-10.1\",\"podDisruptionBudget\":
{\"maxUnavailable\":1},\"resources\":{\"requests\":null},\"size\":3,\"volumeSpec\":{\"persistentVolumeClaim\":{\"resources\":{\"requests\":
{\"storage\":\"6Gi\"}}}}},\"secretsName\":\"my-cluster-secrets\",\"sslInternalSecretName\":\"my-cluster-ssl-internal\",\"sslSecretName\":\"my-
cluster-ssl\",\"updateStrategy\":\"RollingUpdate\",\"vaultSecretName\":\"keyring-secret-vault\"}}\n"
      },
      "creationTimestamp":"2020-06-01T16:50:05Z",
      "finalizers":[
         "delete-pxc-pods-in-order"
      ],
      "generation":4,
      "managedFields":[
         {
            "apiVersion":"pxc.percona.com/v1-5-0",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:metadata":{
                  "f:annotations":{
                     ".":{

                     },
                     "f:kubectl.kubernetes.io/last-applied-configuration":{

                     }
                  },
                  "f:finalizers":{

                  }
               },
               "f:spec":{
                  ".":{

                  },
                  "f:allowUnsafeConfigurations":{

                  },
                  "f:backup":{
                     ".":{

                     },
                     "f:image":{

                     },
                     "f:schedule":{

                     },
                     "f:serviceAccountName":{

                     },
                     "f:storages":{
                        ".":{

                        },
                        "f:fs-pvc":{
                           ".":{

                           },
                           "f:type":{

                           },
                           "f:volume":{
                              ".":{

                              },
                              "f:persistentVolumeClaim":{
                                 ".":{

                                 },
                                 "f:accessModes":{

                                 },
                                 "f:resources":{
                                    ".":{

                                    },
                                    "f:requests":{
                                       ".":{

                                       },
                                       "f:storage":{

                                       }
                                    }
                                 }
                              }
                           }
                        },
                        "f:s3-us-west":{
                           ".":{
```

```
            },
            "f:s3":{
               ".":{

               },
               "f:bucket":{

               },
               "f:credentialsSecret":{

               },
               "f:region":{

               }
            },
            "f:type":{

            }
         }
      }
   },
   "f:pmm":{
      ".":{

      },
      "f:image":{

      },
      "f:serverHost":{

      },
      "f:serverUser":{

      }
   },
   "f:proxysql":{
      ".":{

      },
      "f:affinity":{
         ".":{

         },
         "f:antiAffinityTopologyKey":{

         }
      },
      "f:enabled":{

      },
      "f:gracePeriod":{

      },
      "f:image":{

      },
      "f:podDisruptionBudget":{
         ".":{

         },
         "f:maxUnavailable":{

         }
      },
      "f:resources":{

      },
      "f:size":{

      },
      "f:volumeSpec":{
         ".":{

         },
         "f:persistentVolumeClaim":{
            ".":{

            },
            "f:resources":{
               ".":{

               },
               "f:requests":{
                  ".":{

                  },
                  "f:storage":{

                  }
               }
            }
         }
      }
```

```
                },
                "f:pxc":{
                   ".":{

                   },
                   "f:affinity":{
                      ".":{

                      },
                      "f:antiAffinityTopologyKey":{

                      }
                   },
                   "f:gracePeriod":{

                   },
                   "f:podDisruptionBudget":{
                      ".":{

                      },
                      "f:maxUnavailable":{

                      }
                   },
                   "f:resources":{

                   },
                   "f:volumeSpec":{
                      ".":{

                      },
                      "f:persistentVolumeClaim":{
                         ".":{

                         },
                         "f:resources":{
                            ".":{

                            },
                            "f:requests":{
                               ".":{

                               },
                               "f:storage":{

                               }
                            }
                         }
                      }
                   }
                },
                "f:secretsName":{

                },
                "f:sslInternalSecretName":{

                },
                "f:sslSecretName":{

                },
                "f:updateStrategy":{

                },
                "f:vaultSecretName":{

                }
             }
          },
          "manager":"kubectl",
          "operation":"Update",
          "time":"2020-06-01T16:52:30Z"
       },
       {
          "apiVersion":"pxc.percona.com/v1",
          "fieldsType":"FieldsV1",
          "fieldsV1":{
             "f:spec":{
                "f:backup":{
                   "f:storages":{
                      "f:fs-pvc":{
                         "f:podSecurityContext":{
                            ".":{

                            },
                            "f:fsGroup":{

                            },
                            "f:supplementalGroups":{

                            }
                         },
                         "f:s3":{
                            ".":{
```

```
                  },
                  "f:bucket":{

                  },
                  "f:credentialsSecret":{

                  }
               }
            },
            "f:s3-us-west":{
               "f:podSecurityContext":{
                  ".":{

                  },
                  "f:fsGroup":{

                  },
                  "f:supplementalGroups":{

                  }
               }
            }
         }
      },
      "f:pmm":{
         "f:resources":{

         }
      },
      "f:proxysql":{
         "f:podSecurityContext":{
            ".":{

            },
            "f:fsGroup":{

            },
            "f:supplementalGroups":{

            }
         },
         "f:sslInternalSecretName":{

         },
         "f:sslSecretName":{

         },
         "f:volumeSpec":{
            "f:persistentVolumeClaim":{
               "f:accessModes":{

               }
            }
         }
      },
      "f:pxc":{
         "f:podSecurityContext":{
            ".":{

            },
            "f:fsGroup":{

            },
            "f:supplementalGroups":{

            }
         },
         "f:sslInternalSecretName":{

         },
         "f:sslSecretName":{

         },
         "f:vaultSecretName":{

         },
         "f:volumeSpec":{
            "f:persistentVolumeClaim":{
               "f:accessModes":{

               }
            }
         }
      }
   },
   "f:status":{
      ".":{

      },
      "f:conditions":{

      },
```

```
                "f:host":{

                },
                "f:observedGeneration":{

                },
                "f:proxysql":{
                   ".":{

                   },
                   "f:ready":{

                   },
                   "f:size":{

                   },
                   "f:status":{

                   }
                },
                "f:pxc":{
                   ".":{

                   },
                   "f:ready":{

                   },
                   "f:size":{

                   },
                   "f:status":{

                   }
                },
                "f:state":{

                }
             }
          },
          "manager":"percona-xtradb-cluster-operator",
          "operation":"Update",
          "time":"2020-06-03T15:32:11Z"
       },
       {
          "apiVersion":"pxc.percona.com/v1",
          "fieldsType":"FieldsV1",
          "fieldsV1":{
             "f:spec":{
                "f:pxc":{
                   "f:image":{

                   },
                   "f:size":{

                   }
                }
             }
          },
          "manager":"kubectl",
          "operation":"Update",
          "time":"2020-06-03T15:32:14Z"
       }
    ],
    "name":"cluster1",
    "namespace":"default",
    "resourceVersion":"129605",
    "selfLink":"/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters/cluster1",
    "uid":"15e5e7d6-10b2-46cf-85d0-d3fdea3412ca"
 },
 "spec":{
    "allowUnsafeConfigurations":true,
    "backup":{
       "image":"percona/percona-xtradb-cluster-operator:1.5.0-pxc8.0-backup",
       "schedule":[
          {
             "keep":3,
             "name":"sat-night-backup",
             "schedule":"0 0 * * 6",
             "storageName":"s3-us-west"
          },
          {
             "keep":5,
             "name":"daily-backup",
             "schedule":"0 0 * * *",
             "storageName":"fs-pvc"
          }
       ],
       "serviceAccountName":"percona-xtradb-cluster-operator",
       "storages":{
          "fs-pvc":{
             "type":"filesystem",
             "volume":{
                "persistentVolumeClaim":{
```

```
                     "accessModes":[
                        "ReadWriteOnce"
                     ],
                     "resources":{
                        "requests":{
                           "storage":"6Gi"
                        }
                     }
                  }
               }
            },
            "s3-us-west":{
               "s3":{
                  "bucket":"S3-BACKUP-BUCKET-NAME-HERE",
                  "credentialsSecret":"my-cluster-name-backup-s3",
                  "region":"us-west-2"
               },
               "type":"s3"
            }
         }
      },
      "pmm":{
         "enabled":false,
         "image":"percona/percona-xtradb-cluster-operator:1.5.0-pmm",
         "serverHost":"monitoring-service",
         "serverUser":"pmm"
      },
      "proxysql":{
         "affinity":{
            "antiAffinityTopologyKey":"none"
         },
         "enabled":true,
         "gracePeriod":30,
         "image":"percona/percona-xtradb-cluster-operator:1.5.0-proxysql",
         "podDisruptionBudget":{
            "maxUnavailable":1
         },
         "resources":{
            "requests":null
         },
         "size":3,
         "volumeSpec":{
            "persistentVolumeClaim":{
               "resources":{
                  "requests":{
                     "storage":"2Gi"
                  }
               }
            }
         }
      },
      "pxc":{
         "affinity":{
            "antiAffinityTopologyKey":"none"
         },
         "gracePeriod":600,
         "image":"percona/percona-xtradb-cluster:5.7.30-31.43",
         "podDisruptionBudget":{
            "maxUnavailable":1
         },
         "resources":{
            "requests":null
         },
         "size":"5",
         "volumeSpec":{
            "persistentVolumeClaim":{
               "resources":{
                  "requests":{
                     "storage":"6Gi"
                  }
               }
            }
         }
      },
      "secretsName":"my-cluster-secrets",
      "sslInternalSecretName":"my-cluster-ssl-internal",
      "sslSecretName":"my-cluster-ssl",
      "updateStrategy":"RollingUpdate",
      "vaultSecretName":"keyring-secret-vault"
   },
   "status":{
      "conditions":[
         {
            "lastTransitionTime":"2020-06-01T16:50:37Z",
            "message":"create newStatefulSetNode: StatefulSet.apps \"cluster1-pxc\" is invalid: spec.updateStrategy: Invalid value:
apps.StatefulSetUpdateStrategy{Type:\"SmartUpdate\", RollingUpdate:(*apps.RollingUpdateStatefulSetStrategy)(nil)}: must be 'RollingUpdate' or
'OnDelete'",
            "reason":"ErrorReconcile",
            "status":"True",
            "type":"Error"
         },
         {
            "lastTransitionTime":"2020-06-01T16:52:31Z",
```

```
            "status":"True",
            "type":"Initializing"
         },
         {
            "lastTransitionTime":"2020-06-01T16:55:59Z",
            "status":"True",
            "type":"Ready"
         },
         {
            "lastTransitionTime":"2020-06-01T17:19:15Z",
            "status":"True",
            "type":"Initializing"
         }
      ],
      "host":"cluster1-proxysql.default",
      "observedGeneration":3,
      "proxysql":{
         "ready":3,
         "size":3,
         "status":"ready"
      },
      "pxc":{
         "ready":1,
         "size":3,
         "status":"initializing"
      },
      "state":"initializing"
   }
}
```


Update Percona XtraDB Cluster image

Description:

```
Change the image of Percona XtraDB Cluster containers inside the cluster
```

Kubectl Command:

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
"spec": {"pxc":{ "image": "percona/percona-xtradb-cluster:5.7.30-31.43" }
}}'
```

URL:

```
https://$API_SERVER/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters/cluster1
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XPATCH "https://$API_SERVER/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters/cluster1" \
            -H "Authorization: Bearer $KUBE_TOKEN" \
            -H "Accept: application/json" \
            -H "Content-Type: application/merge-patch+json"
            -d '{
              "spec": {"pxc":{ "image": "percona/percona-xtradb-cluster:5.7.30-31.43" }
              }}'
```

Request Body:

Example


```
{
"spec": {"pxc":{ "image": "percona/percona-xtradb-cluster:5.7.30-31.43" }
}}
```


Input:

> **spec**:
>
> pxc:
>
> 1. image (String, min-length: 1) : `name of the image to update for Percona XtraDB Cluster`

Response:

Example


```
{
   "apiVersion":"pxc.percona.com/v1",
   "kind":"PerconaXtraDBCluster",
   "metadata":{
      "annotations":{
         "kubectl.kubernetes.io/last-applied-configuration":"{\"apiVersion\":\"pxc.percona.com/v1-5-
0\",\"kind\":\"PerconaXtraDBCluster\",\"metadata\":{\"annotations\":{},\"finalizers\":[\"delete-pxc-pods-in-
order\"],\"name\":\"cluster1\",\"namespace\":\"default\"},\"spec\":{\"allowUnsafeConfigurations\":true,\"backup\":{\"image\":\"percona/percona-
xtradb-cluster-operator:1.5.0-pxc8.0-backup\",\"schedule\":[{\"keep\":3,\"name\":\"sat-night-backup\",\"schedule\":\"0 0 * *
6\",\"storageName\":\"s3-us-west\"},{\"keep\":5,\"name\":\"daily-backup\",\"schedule\":\"0 0 * * *\",\"storageName\":\"fs-
pvc\"}],\"serviceAccountName\":\"percona-xtradb-cluster-operator\",\"storages\":{\"fs-pvc\":{\"type\":\"filesystem\",\"volume\":
{\"persistentVolumeClaim\":{\"accessModes\":[\"ReadWriteOnce\"],\"resources\":{\"requests\":{\"storage\":\"6Gi\"}}}}},\"s3-us-west\":{\"s3\":
{\"bucket\":\"S3-BACKUP-BUCKET-NAME-HERE\",\"credentialsSecret\":\"my-cluster-name-backup-s3\",\"region\":\"us-west-
2\"},\"type\":\"s3\"}},\"pmm\":{\"enabled\":false,\"image\":\"percona/percona-xtradb-cluster-operator:1.5.0-pmm\",\"serverHost\":\"monitoring-
service\",\"serverUser\":\"pmm\"},\"proxysql\":{\"affinity\":
{\"antiAffinityTopologyKey\":\"none\"},\"enabled\":true,\"gracePeriod\":30,\"image\":\"percona/percona-xtradb-cluster-operator:1.5.0-
proxysql\",\"podDisruptionBudget\":{\"maxUnavailable\":1},\"resources\":{\"requests\":null},\"size\":3,\"volumeSpec\":{\"persistentVolumeClaim\":
{\"resources\":{\"requests\":{\"storage\":\"2Gi\"}}}}},\"pxc\":{\"affinity\":
{\"antiAffinityTopologyKey\":\"none\"},\"gracePeriod\":600,\"image\":\"percona/percona-xtradb-cluster:8.0.19-10.1\",\"podDisruptionBudget\":
{\"maxUnavailable\":1},\"resources\":{\"requests\":null},\"size\":3,\"volumeSpec\":{\"persistentVolumeClaim\":{\"resources\":{\"requests\":
{\"storage\":\"6Gi\"}}}}},\"secretsName\":\"my-cluster-secrets\",\"sslInternalSecretName\":\"my-cluster-ssl-internal\",\"sslSecretName\":\"my-
cluster-ssl\",\"updateStrategy\":\"RollingUpdate\",\"vaultSecretName\":\"keyring-secret-vault\"}}\n"
      },
      "creationTimestamp":"2020-06-01T16:50:05Z",
      "finalizers":[
         "delete-pxc-pods-in-order"
      ],
      "generation":3,
      "managedFields":[
         {
            "apiVersion":"pxc.percona.com/v1-5-0",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:metadata":{
                  "f:annotations":{
                     ".":{

                     },
                     "f:kubectl.kubernetes.io/last-applied-configuration":{

                     }
                  },
                  "f:finalizers":{

                  }
               },
               "f:spec":{
                  ".":{

                  },
                  "f:allowUnsafeConfigurations":{

                  },
                  "f:backup":{
                     ".":{

                     },
                     "f:image":{

                     },
                     "f:schedule":{

                     },
                     "f:serviceAccountName":{

                     },
                     "f:storages":{
                        ".":{

                        },
                        "f:fs-pvc":{
                           ".":{

                           },
                           "f:type":{

                           },
                           "f:volume":{
                              ".":{
```

```
                    },
                    "f:persistentVolumeClaim":{
                       ".":{

                       },
                       "f:accessModes":{

                       },
                       "f:resources":{
                          ".":{

                          },
                          "f:requests":{
                             ".":{

                             },
                             "f:storage":{

                             }
                          }
                       }
                    }
                 }
              },
              "f:s3-us-west":{
                 ".":{

                 },
                 "f:s3":{
                    ".":{

                    },
                    "f:bucket":{

                    },
                    "f:credentialsSecret":{

                    },
                    "f:region":{

                    }
                 },
                 "f:type":{

                 }
              }
           }
        },
        "f:pmm":{
           ".":{

           },
           "f:image":{

           },
           "f:serverHost":{

           },
           "f:serverUser":{

           }
        },
        "f:proxysql":{
           ".":{

           },
           "f:affinity":{
              ".":{

              },
              "f:antiAffinityTopologyKey":{

              }
           },
           "f:enabled":{

           },
           "f:gracePeriod":{

           },
           "f:image":{

           },
           "f:podDisruptionBudget":{
              ".":{

              },
              "f:maxUnavailable":{

              }
           },
           "f:resources":{
```

```
      },
      "f:size":{

      },
      "f:volumeSpec":{
         ".":{

         },
         "f:persistentVolumeClaim":{
            ".":{

            },
            "f:resources":{
               ".":{

               },
               "f:requests":{
                  ".":{

                  },
                  "f:storage":{

                  }
               }
            }
         }
      }
   },
   "f:pxc":{
      ".":{

      },
      "f:affinity":{
         ".":{

         },
         "f:antiAffinityTopologyKey":{

         }
      },
      "f:gracePeriod":{

      },
      "f:podDisruptionBudget":{
         ".":{

         },
         "f:maxUnavailable":{

         }
      },
      "f:resources":{

      },
      "f:size":{

      },
      "f:volumeSpec":{
         ".":{

         },
         "f:persistentVolumeClaim":{
            ".":{

            },
            "f:resources":{
               ".":{

               },
               "f:requests":{
                  ".":{

                  },
                  "f:storage":{

                  }
               }
            }
         }
      }
   },
   "f:secretsName":{

   },
   "f:sslInternalSecretName":{

   },
   "f:sslSecretName":{

   },
   "f:updateStrategy":{
```

```
                },
                "f:vaultSecretName":{

                }
             }
          },
          "manager":"kubectl",
          "operation":"Update",
          "time":"2020-06-01T16:52:30Z"
       },
       {
          "apiVersion":"pxc.percona.com/v1",
          "fieldsType":"FieldsV1",
          "fieldsV1":{
             "f:spec":{
                "f:pxc":{
                   "f:image":{

                   }
                }
             }
          },
          "manager":"kubectl",
          "operation":"Update",
          "time":"2020-06-01T17:18:58Z"
       },
       {
          "apiVersion":"pxc.percona.com/v1",
          "fieldsType":"FieldsV1",
          "fieldsV1":{
             "f:spec":{
                "f:backup":{
                   "f:storages":{
                      "f:fs-pvc":{
                         "f:podSecurityContext":{
                            ".":{

                            },
                            "f:fsGroup":{

                            },
                            "f:supplementalGroups":{

                            }
                         },
                         "f:s3":{
                            ".":{

                            },
                            "f:bucket":{

                            },
                            "f:credentialsSecret":{

                            }
                         }
                      },
                      "f:s3-us-west":{
                         "f:podSecurityContext":{
                            ".":{

                            },
                            "f:fsGroup":{

                            },
                            "f:supplementalGroups":{

                            }
                         }
                      }
                   }
                },
                "f:pmm":{
                   "f:resources":{

                   }
                },
                "f:proxysql":{
                   "f:podSecurityContext":{
                      ".":{

                      },
                      "f:fsGroup":{

                      },
                      "f:supplementalGroups":{

                      }
                   },
                   "f:sslInternalSecretName":{

                   },
                   "f:sslSecretName":{
```

```
                },
                "f:volumeSpec":{
                   "f:persistentVolumeClaim":{
                      "f:accessModes":{

                      }
                   }
                }
             },
             "f:pxc":{
                "f:podSecurityContext":{
                   ".":{

                   },
                   "f:fsGroup":{

                   },
                   "f:supplementalGroups":{

                   }
                },
                "f:sslInternalSecretName":{

                },
                "f:sslSecretName":{

                },
                "f:vaultSecretName":{

                },
                "f:volumeSpec":{
                   "f:persistentVolumeClaim":{
                      "f:accessModes":{

                      }
                   }
                }
             }
          },
          "f:status":{
             ".":{

             },
             "f:conditions":{

             },
             "f:host":{

             },
             "f:message":{

             },
             "f:observedGeneration":{

             },
             "f:proxysql":{
                ".":{

                },
                "f:ready":{

                },
                "f:size":{

                },
                "f:status":{

                }
             },
             "f:pxc":{
                ".":{

                },
                "f:message":{

                },
                "f:ready":{

                },
                "f:size":{

                },
                "f:status":{

                }
             },
             "f:state":{

             }
          }
       },
       "manager":"percona-xtradb-cluster-operator",
```

```
            "operation":"Update",
            "time":"2020-06-01T17:21:36Z"
         }
      ],
      "name":"cluster1",
      "namespace":"default",
      "resourceVersion":"41149",
      "selfLink":"/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusters/cluster1",
      "uid":"15e5e7d6-10b2-46cf-85d0-d3fdea3412ca"
   },
   "spec":{
      "allowUnsafeConfigurations":true,
      "backup":{
         "image":"percona/percona-xtradb-cluster-operator:1.5.0-pxc8.0-backup",
         "schedule":[
            {
               "keep":3,
               "name":"sat-night-backup",
               "schedule":"0 0 * * 6",
               "storageName":"s3-us-west"
            },
            {
               "keep":5,
               "name":"daily-backup",
               "schedule":"0 0 * * *",
               "storageName":"fs-pvc"
            }
         ],
         "serviceAccountName":"percona-xtradb-cluster-operator",
         "storages":{
            "fs-pvc":{
               "type":"filesystem",
               "volume":{
                  "persistentVolumeClaim":{
                     "accessModes":[
                        "ReadWriteOnce"
                     ],
                     "resources":{
                        "requests":{
                           "storage":"6Gi"
                        }
                     }
                  }
               }
            },
            "s3-us-west":{
               "s3":{
                  "bucket":"S3-BACKUP-BUCKET-NAME-HERE",
                  "credentialsSecret":"my-cluster-name-backup-s3",
                  "region":"us-west-2"
               },
               "type":"s3"
            }
         }
      },
      "pmm":{
         "enabled":false,
         "image":"percona/percona-xtradb-cluster-operator:1.5.0-pmm",
         "serverHost":"monitoring-service",
         "serverUser":"pmm"
      },
      "proxysql":{
         "affinity":{
            "antiAffinityTopologyKey":"none"
         },
         "enabled":true,
         "gracePeriod":30,
         "image":"percona/percona-xtradb-cluster-operator:1.5.0-proxysql",
         "podDisruptionBudget":{
            "maxUnavailable":1
         },
         "resources":{
            "requests":null
         },
         "size":3,
         "volumeSpec":{
            "persistentVolumeClaim":{
               "resources":{
                  "requests":{
                     "storage":"2Gi"
                  }
               }
            }
         }
      },
      "pxc":{
         "affinity":{
            "antiAffinityTopologyKey":"none"
         },
         "gracePeriod":600,
         "image":"percona/percona-xtradb-cluster:5.7.30-31.43",
         "podDisruptionBudget":{
            "maxUnavailable":1
```

```
         },
         "resources":{
            "requests":null
         },
         "size":3,
         "volumeSpec":{
            "persistentVolumeClaim":{
               "resources":{
                  "requests":{
                     "storage":"6Gi"
                  }
               }
            }
         }
      },
      "secretsName":"my-cluster-secrets",
      "sslInternalSecretName":"my-cluster-ssl-internal",
      "sslSecretName":"my-cluster-ssl",
      "updateStrategy":"RollingUpdate",
      "vaultSecretName":"keyring-secret-vault"
   },
   "status":{
      "conditions":[
         {
            "lastTransitionTime":"2020-06-01T16:50:37Z",
            "message":"create newStatefulSetNode: StatefulSet.apps \"cluster1-pxc\" is invalid: spec.updateStrategy: Invalid value:
apps.StatefulSetUpdateStrategy{Type:\"SmartUpdate\", RollingUpdate:(*apps.RollingUpdateStatefulSetStrategy)(nil)}: must be 'RollingUpdate' or
'OnDelete'",
            "reason":"ErrorReconcile",
            "status":"True",
            "type":"Error"
         },
         {
            "lastTransitionTime":"2020-06-01T16:52:31Z",
            "status":"True",
            "type":"Initializing"
         },
         {
            "lastTransitionTime":"2020-06-01T16:55:59Z",
            "status":"True",
            "type":"Ready"
         },
         {
            "lastTransitionTime":"2020-06-01T17:19:15Z",
            "status":"True",
            "type":"Initializing"
         }
      ],
      "host":"cluster1-proxysql.default",
      "message":[
         "PXC: pxc: back-off 40s restarting failed container=pxc pod=cluster1-pxc-2_default(87cdf1a8-0fb3-4bc0-b50d-f66a0a73c087); "
      ],
      "observedGeneration":3,
      "proxysql":{
         "ready":3,
         "size":3,
         "status":"ready"
      },
      "pxc":{
         "message":"pxc: back-off 40s restarting failed container=pxc pod=cluster1-pxc-2_default(87cdf1a8-0fb3-4bc0-b50d-f66a0a73c087); ",
         "ready":2,
         "size":3,
         "status":"initializing"
      },
      "state":"initializing"
   }
}
```


Pass custom my.cnf during the creation of Percona XtraDB Cluster

Description:

```
Create a custom config map containing the contents of the file my.cnf to be
passed on to the Percona XtraDB Cluster containers when they are created
```

Kubectl Command:

```
$ kubectl create configmap cluster1-pxc3 --from-file=my.cnf
```

my.cnf (Contains mysql configuration):

```
[mysqld]
max_connections=250
```

URL:

```
https://$API_SERVER/api/v1/namespaces/default/configmaps
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XPOST "https://$API_SERVER/api/v1/namespaces/default/configmaps" \
            -H "Accept: application/json" \
            -H "Authorization: Bearer $KUBE_TOKEN" \
            -d '{"apiVersion":"v1","data":{"my.cnf":"[mysqld]\nmax_connections=250\n"},"kind":"ConfigMap","metadata":
{"creationTimestamp":null,"name":"cluster1-pxc3"}}' \
            -H "Content-Type: application/json"
```

Request Body:

Example

```
{
   "apiVersion":"v1",
   "data":{
      "my.cnf":"[mysqld]\nmax_connections=250\n"
   },
   "kind":"ConfigMap",
   "metadata":{
      "creationTimestamp":null,
      "name":"cluster1-pxc3"
   }
}
```

Input:

1. data (Object {filename : contents(String, min-length:0)}): `contains filenames to create in config map and its contents`
2. metadata: name(String, min-length: 1) : `contains name of the configmap`
3. kind (String): `type of object to create`

Response:

Example


```
{
   "kind":"ConfigMap",
   "apiVersion":"v1",
   "metadata":{
      "name":"cluster1-pxc3",
      "namespace":"default",
      "selfLink":"/api/v1/namespaces/default/configmaps/cluster1-pxc3",
      "uid":"d92c7196-f399-4e20-abc7-b5de62c0691b",
      "resourceVersion":"85258",
      "creationTimestamp":"2020-05-28T14:19:41Z",
      "managedFields":[
         {
            "manager":"kubectl",
            "operation":"Update",
            "apiVersion":"v1",
            "time":"2020-05-28T14:19:41Z",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:data":{
                  ".":{

                  },
                  "f:my.cnf":{

                  }
               }
            }
         }
      ]
   },
   "data":{
      "my.cnf":""
   }
}
```


Backup Percona XtraDB Cluster

Description:

```
Takes a backup of the Percona XtraDB Cluster containers data to be able to
recover from disasters or make a roll-back later
```

Kubectl Command:

```
$ kubectl apply -f percona-xtradb-cluster-operator/deploy/backup/backup.yaml
```

URL:

```
https://$API_SERVER/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusterbackups
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XPOST "https://$API_SERVER/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusterbackups" \
            -H "Accept: application/json" \
            -H "Content-Type: application/json" \
            -d "@backup.json" -H "Authorization: Bearer $KUBE_TOKEN"
```

Request Body (backup.json):

Example


```
{
   "apiVersion":"pxc.percona.com/v1",
   "kind":"PerconaXtraDBClusterBackup",
   "metadata":{
      "name":"backup1"
   },
   "spec":{
      "pxcCluster":"cluster1",
      "storageName":"fs-pvc"
   }
}
```


Input:

1. **metadata**:

> name(String, min-length:1) : `name of backup to create`

1. **spec**:

> 1. pxcCluster(String, min-length:1) : `name of Percona XtraDB Cluster`
>
> 2. storageName(String, min-length:1) : `name of storage claim to use`

Response:

Example


```
{
   "apiVersion":"pxc.percona.com/v1",
   "kind":"PerconaXtraDBClusterBackup",
   "metadata":{
      "creationTimestamp":"2020-05-27T23:56:33Z",
      "generation":1,
      "managedFields":[
         {
            "apiVersion":"pxc.percona.com/v1",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:spec":{
                  ".":{

                  },
                  "f:pxcCluster":{

                  },
                  "f:storageName":{

                  }
               }
            },
            "manager":"kubectl",
            "operation":"Update",
            "time":"2020-05-27T23:56:33Z"
         }
      ],
      "name":"backup1",
      "namespace":"default",
      "resourceVersion":"26024",
      "selfLink":"/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusterbackups/backup1",
      "uid":"95a354b1-e25b-40c3-8be4-388acca055fe"
   },
   "spec":{
      "pxcCluster":"cluster1",
      "storageName":"fs-pvc"
   }
}
```


Restore Percona XtraDB Cluster

Description:

```
Restores Percona XtraDB Cluster data to an earlier version to recover from a
problem or to make a roll-back
```

Kubectl Command:

```
$ kubectl apply -f percona-xtradb-cluster-operator/deploy/backup/restore.yaml
```

URL:

```
https://$API_SERVER/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusterrestores
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XPOST "https://$API_SERVER/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusterrestores" \
            -H "Accept: application/json" \
            -H "Content-Type: application/json" \
            -d "@restore.json" \
            -H "Authorization: Bearer $KUBE_TOKEN"
```

Request Body (restore.json):

Example

```
{
   "apiVersion":"pxc.percona.com/v1",
   "kind":"PerconaXtraDBClusterRestore",
   "metadata":{
      "name":"restore1"
   },
   "spec":{
      "pxcCluster":"cluster1",
      "backupName":"backup1"
   }
}
```

Input:

1. **metadata**:

> name(String, min-length:1): `name of restore to create`

1. **spec**:

> 1. pxcCluster(String, min-length:1) : `name of Percona XtraDB Cluster`
>
> 2. backupName(String, min-length:1) : `name of backup to restore from`

Response:

Example


```
{
   "apiVersion":"pxc.percona.com/v1",
   "kind":"PerconaXtraDBClusterRestore",
   "metadata":{
      "creationTimestamp":"2020-05-27T23:59:41Z",
      "generation":1,
      "managedFields":[
         {
            "apiVersion":"pxc.percona.com/v1",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:spec":{
                  ".":{

                  },
                  "f:backupName":{

                  },
                  "f:pxcCluster":{

                  }
               }
            },
            "manager":"kubectl",
            "operation":"Update",
            "time":"2020-05-27T23:59:41Z"
         }
      ],
      "name":"restore1",
      "namespace":"default",
      "resourceVersion":"26682",
      "selfLink":"/apis/pxc.percona.com/v1/namespaces/default/perconaxtradbclusterrestores/restore1",
      "uid":"770c3471-be17-46fb-b0a6-e706685ab2fc"
   },
   "spec":{
      "backupName":"backup1",
      "pxcCluster":"cluster1"
   }
}
```

Frequently Asked Questions

Why do we need to follow “the Kubernetes way” when Kubernetes was never intended to run databases?

As it is well known, the Kubernetes approach is targeted at stateless applications but provides ways to store state (in Persistent Volumes, etc.) if the application needs it. Generally, a stateless mode of operation is supposed to provide better safety, sustainability, and scalability, it makes the already-deployed components interchangeable. You can find more about substantial benefits brought by Kubernetes to databases in this blog post.

The architecture of state-centric applications (like databases) should be composed in a right way to avoid crashes, data loss, or data inconsistencies during hardware failure. Percona Operator for MySQL provides out-of-the-box functionality to automate provisioning and management of highly available MySQL database clusters on Kubernetes.

How can I contact the developers?

The best place to discuss Percona Operator for MySQL based on Percona XtraDB Cluster with developers and other community members is the community forum.

If you would like to report a bug, use the Percona Operator for MySQL project in JIRA.

What is the difference between the Operator quickstart and advanced installation ways?

As you have noticed, the installation section of docs contains both quickstart and advanced installation guides.

The quickstart guide is simpler. It has fewer installation steps in favor of predefined default choices. Particularly, in advanced installation guides, you separately apply the Custom Resource Definition and Role-based Access Control configuration files with possible edits in them. At the same time, quickstart guides rely on the all-inclusive bundle configuration.

At another point, quickstart guides are related to specific platforms you are going to use (Minikube, Google Kubernetes Engine, etc.) and therefore include some additional steps needed for these platforms.

Generally, rely on the quickstart guide if you are a beginner user of the specific platform and/or you are new to the Percona Distribution for MySQL Operator as a whole.

Which versions of MySQL does the Percona Operator for MySQL support?

Percona Operator for MySQL based on Percona XtraDB Cluster provides a ready-to-use installation of the MySQL-based Percona XtraDB Cluster inside your Kubernetes installation. It works with both MySQL 8.0 and 5.7 branches, and the exact version is determined by the Docker image in use.

Percona-certified Docker images used by the Operator are listed here. As you can see, both Percona XtraDB Cluster 8.0 and 5.7 are supported with the following recommended versions: 8.0.44-35.1 and 5.7.44-31.65. Three major numbers in the XtraDB Cluster version refer to the version of Percona Server in use. More details on the exact Percona Server version can be found in the release notes (8.0, 5.7).

How is HAProxy better than ProxySQL?

Percona Operator for MySQL based on Percona XtraDB Cluster supports both HAProxy and ProxySQL as a load balancer. HAProxy is turned on by default, but both solutions are similar in terms of their configuration and operation under the control of the Operator.

Still, they have technical differences. HAProxy is a general and widely used high availability, load balancing, and proxying solution for TCP and HTTP-based applications. ProxySQL provides similar functionality but is specific to MySQL clusters. As an SQL-aware solution, it is able to provide more tight internal integration with MySQL instances.

Both projects do a really good job with the Operator. The proxy choice should depend mostly on application-specific workload (including object-relational mapping), performance requirements, advanced routing and caching needs with one or another project, components already in use in the current infrastructure, and any other specific needs of the application.

How can I create a directory on the node to use it as a local storage

You can configure hostPath volume to mount some existing file or directory from the node's filesystem into the Pod and use it as a local storage. The directory used for local storage should already exist in the node's filesystem. You can create it through the shell access to the node, with `mkdir` command, as all other directories. Alternatively you can create a Pod which will do this job. Let's suppose you are going to use `/var/run/data-dir` directory as your local storage, describing it in the `deploy/cr.yaml` configuration file as follows:

```
...
pxc:
  ...
  volumeSpec:
     hostPath:
       path: /var/run/data-dir
       type: Directory
  containerSecurityContext:
    privileged: false
  podSecurityContext:
    runAsUser: 1001
    runAsGroup: 1001
    supplementalGroups: [1001]
  nodeSelector:
    kubernetes.io/hostname: a.b.c
```

Create the yaml file (e.g. `mypod.yaml`), with the following contents:

```
apiVersion: v1
kind: Pod
metadata:
  name: hostpath-helper
spec:
  containers:
  - name: init
    image: busybox
    command: ["install", "-o", "1001", "-g", "1001", "-m", "755", "-d", "/mnt/data-dir"]
    volumeMounts:
    - name: root
      mountPath: /mnt
    securityContext:
      runAsUser: 0
  volumes:
  - name: root
    hostPath:
      path: /var/run
  restartPolicy: Never
  nodeSelector:
    kubernetes.io/hostname: a.b.c
```

Don't forget to apply it as usual:

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

How can I add custom sidecar containers to my cluster?

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. Add such sidecar container to the `deploy/cr.yaml` configuration file, specifying its name and image, and possibly a command to run:

```
spec:
  pxc:
    ....
    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
    ....
```

You can add `sidecars` subsection to `pxc`, `haproxy`, and `proxysql` sections.

Note

Custom sidecar containers can easily access other components of your cluster. Therefore they should be used carefully and by experienced users only.

Find more information on sidecar containers in the appropriate documentation page.

How to get core dumps in case of the Percona XtraDB Cluster crash

In the Percona XtraDB Cluster crash case, gathering all possible information for enhanced diagnostics to be shared with Percona Support helps to solve an issue faster. One of such helpful artifacts is core dump.

Percona XtraDB Cluster can create core dumps on crush using libcoredumper. The Operator has this feature turned on by default. Core dumps are saved to `DATADIR` (`var/lib/mysql/`). You can find appropriate core files in the following way (substitute `some-name-pxc-1` with the name of your Pod):

```
$ kubectl exec some-name-pxc-1 -c pxc -it -- sh -c 'ls -alh /var/lib/mysql/ | grep core'
-rw------- 1 mysql mysql 1.3G Jan 15 09:30 core.20210015093005
```

When identified, the appropriate core dump can be downloaded as follows:

```
$ kubectl cp some-name-pxc-1:/var/lib/mysql/core.20210015093005  /tmp/core.20210015093005
```

Note

It is useful to provide Build ID and Server Version in addition to core dump when Creating a support ticket. Both can be found from logs:

```
$ kubectl logs some-name-pxc-1 -c logs

[1] init-deploy-949.some-name-pxc-1.mysqld-error.log: [1610702394.259356066, {"log"=>"09:19:54 UTC - mysqld got signal 11 ;"}]
[2] init-deploy-949.some-name-pxc-1.mysqld-error.log: [1610702394.259356829, {"log"=>"Most likely, you have hit a bug, but this error can also be
caused by malfunctioning hardware."}]
[3] init-deploy-949.some-name-pxc-1.mysqld-error.log: [1610702394.259457282, {"log"=>"Build ID: 5a2199b1784b967a713a3bde8d996dc517c41adb"}]
[4] init-deploy-949.some-name-pxc-1.mysqld-error.log: [1610702394.259465692, {"log"=>"Server Version: 8.0.21-12.1 Percona XtraDB Cluster (GPL),
Release rel12, Revision 4d973e2, WSREP version 26.4.3, wsrep_26.4.3"}]
.....
```

How to choose between HAProxy and ProxySQL when configuring the cluster?

You can configure the Operator to use one of two different proxies, HAProxy (the default choice) and ProxySQL. Both solutions are fully supported by the Operator, but they have some differences in the architecture, which can make one of them more suitable then the other one in some use cases.

The main difference is that HAProxy operates in TCP mode as an OSI level 4 proxy, while ProxySQL implements OSI level 7 proxy, and thus can provide some additional functionality like read/write split, firewalling and caching.

From the other side, utilizing HAProxy for the service is the easier way to go, and getting use of the ProxySQL level 7 specifics requires good understanding of Kubernetes and ProxySQL.

You can enable ProxySQL only at cluster creation time. Otherwise you will be able to use HAProxy only. The switch from HAProxy to ProxySQL is not possible, because ProxySQL does not yet support caching_sha2_password MySQL authentication plugin used by the Operator by default instead of the older mysql_native_password one.

See more detailed functionality and performance comparison of using the Operator with both solutions in this blog post.

Which additional access permissions are used by the Custom Resource validation webhook?

The `spec.enableCRValidationWebhook` key in the deploy/cr.yaml file enables or disables schema validation done by the Operator before applying `cr.yaml` file. This feature works only in cluster-wide mode due to access restrictions. It uses the following additional RBAC permissions:

```
- apiGroups:
  - admissionregistration.k8s.io
  resources:
  - validatingwebhookconfigurations
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
```

Development documentation

How we use artificial intelligence

The technical writer oversees the integration of AI-driven tools and platforms into the documentation workflow, ensuring that AI-generated text meets the standards for clarity, coherence, and accuracy. While AI assists in tasks such as content generation, language enhancement, and formatting optimization, the technical writer is responsible for validating and refining the output to ensure its suitability for the intended audience.

Throughout the documentation process, the technical writer reviews the quality and relevance of AI-generated content in detail and with critical judgment. By leveraging their expertise in language, communication, and subject matter knowledge, the technical writer collaborates with AI systems to refine and tailor the documentation to meet the specific needs and preferences of the audience.

While AI accelerates the documentation process and enhances productivity, the technical writer verifies the information's accuracy and maintains consistency in terminology, style, and tone. The technical writer ensures that the final document reflects the company's commitment to excellence.

Copyright and licensing information

Documentation licensing

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.

Release Notes

Percona Operator for MySQL based on Percona XtraDB Cluster Release Notes

Percona Operator for MySQL based on Percona XtraDB Cluster 1.19.0 (2026-01-19)

Installation

What's new at a glance

- Percona XtraDB Cluster 8.4 is the default version for cluster deployments

Security & Compliance

- Data-at-rest encryption for Percona XtraDB Cluster 8.4
- Custom CA certificates for S3 backups
- Configurable certificate lifetimes
- Security context for ProxySQL sidecars

Performance & reliability

- ProxySQL scheduler (tech preview) - improved query routing
- Faster HAProxy failover (seconds instead of 20s)
- Direct-access backups via sidecar
- Memory allocator configuration

Operational excellence

- Switch between HAProxy and ProxySQL at runtime
- Automatic cleanup of backup/restore jobs
- External PVC mounting for shared data
- Custom password generation rules

Plus: 15+ bug fixes and stability improvements

Release Highlights

This release of Percona Operator for MySQL based on Percona XtraDB Cluster includes the following new features and improvements:

Ensure data security for Percona XtraDB Cluster 8.4 with data-at-rest encryption

Data at rest encryption ensures that sensitive information stored on disk remains protected from unauthorized access, even if the physical media is compromised. It is a foundational safeguard for compliance, trust and security in modern database environments.

The Operator supports data at rest encryption for MySQL 8.0 with HashiCorp Vault using the `keyring_vault` *plugin*. Now, it also supports data at rest encryption for MySQL 8.4, leveraging the `keyring_vault` *component*.

This enhancement enables you to benefit from the rich feature set of the latest major version of Percona XtraDB Cluster 8.4 while ensuring your sensitive data is secured. In doing so, you can meet compliance requirements and protect critical information without added operational complexity. Learn how to configure data at rest encryption for Percona XtraDB Cluster 8.4.

Percona XtraDB Cluster 8.4 is now fully supported

Percona XtraDB Cluster 8.4 is now fully supported and recommended for production deployments. Starting with this release, it becomes the default version for all new database cluster deployments using the Operator. This support enables you to benefit from its latest features, performance improvements, and enhanced security.

Use your own CA certificates for TLS verification

You can now use your organization's custom Certificate Authority (CA) to securely verify TLS communication with S3 storage during backups and restores.

The configuration is straightforward: create the Secret that stores your custom CA and certificates to authorize in the S3 storage. Then reference this Secret and specify the CA certificate in the `caBundle` option in the Operator Custom Resource. The Operator will verify TLS communication against it.

Here's the example configuration:

```
storages:
  minio-s3:
    type: s3
    verifyTLS: true
    s3:
      caBundle:
         name: minio-ca-bundle
         key: tls.crt
```

With this improvement you ensure the following:

- Security without compromise – no more bypassing identity checks.
- Alignment with your internal standards – use the CA your company already trusts.
- Confidence in backup and restore flows – every S3 interaction is properly verified.

Read more about the use of own CA certificates in our documentation

Configure duration for certificates issued by cert-manager

By default, the `cert-manager` generates certificates valid for 90 days. You have now more control over certificate lifetimes and can configure their custom duration when you create a new cluster. This way you can align with your organization's security and compliance policies.

Use the following Custom Resource options to configure certificate duration:

- `.spec.tls.certValidityDuration` – validity period for generated certificates
- `.spec.tls.caValidityDuration` – validity period for the Certificate Authority (CA)

```
  tls:
    enabled: true
    certValidityDuration: 2160h
    caValidityDuration: 26280h
```

Note that the Operator enforces minimum durations to certificates:

- For TLS certificates – 1 hour
- For CA certificate – 730 hours.

Also, we don't recommend setting duration to exactly 1 hour to prevent certificate generation issues. Read more about rules and limitations about certificate duration configuration in our documentation.

This improvement empowers you to fine-tune certificate lifetimes while keeping your cluster secure and stable.

Security context for ProxySQL sidecar containers

You can now define the security context for ProxySQL sidecar containers in the Operator, reducing the risk of unsecured sidecars bypassing Pod restrictions. This improvement lets you set user IDs, privileges, and filesystem access directly, ensuring compliance and strengthening Pod security.

Configure the security context in your custom resource. For example:

```
spec:
  proxysql:
       sidecars:
          securityContext:
              privileged: false
```

With this change, you enforce safer defaults across your deployments and close security gaps at the Pod level.

Improved load balancing with ProxySQL scheduler (tech preview)

The Operator now integrates with the external `pxc_scheduler_handler` tool to improve query routing. This feature is currently in tech preview, so we recommend experimenting with it in test or staging environments before using it in production.

With this scheduler you get finer control over how your SQL queries are routed within your PXC cluster:

- SELECT queries (that don't use FOR UPDATE) are intelligently distributed across all PXC nodes—or all nodes except the writer, depending on your settings.
- Non-SELECT queries and SELECT FOR UPDATE statements are routed to the writer node.
- You don't have to micromanage the writer role: the scheduler automatically ensures only one writer is active at any time.

This means you could see:

- better performance and higher throughput from distributing query loads
- greater reliability with no single point of failure
- Improved cluster health through early detection of replication lag and node issues
- more efficient use of your resources and hardware
- a smoother, more predictable experience for everyone using the database

See our documentation for full information about the scheduler behavior and setup.

The previous internal scheduler remains enabled by default to maintain backward compatibility. You can switch to the new one when you're ready to benefit from smarter query handling.

Customize HAProxy backend health check intervals and failover behavior

You can now control how quickly HAProxy detects and reacts to node failures. Instead of waiting the default 20 seconds while HAProxy performs the failover, you can tune health checks to cut that down to just a few seconds. That means your applications recover faster, and users don't get stuck with hanging sessions when a node goes down.

You no longer need to override the entire HAProxy configuration to achieve this — the operator now gives you simple, direct options in the Custom Resource specification:

```
haproxy:
   healthCheck:
       interval: 10000
       rise: 1
       fall: 2
```

By adjusting how often checks run and how many failures or successes mark a node as "down" or "up," you get faster failover, cleaner client handling, and easier configuration that is safe to upgrade and tailored to your environment.

Switch from HAProxy to ProxySQL at runtime

You can now switch from HAProxy to ProxySQL without redeploying your Percona XtraDB Cluster. Previously, you had to choose ProxySQL only at startup. Now ProxySQL has the `caching_sha2_password` as the default authentication plugin, which gives you the flexibility to start with HAProxy and migrate to ProxySQL later as your needs evolve.

With this release, ProxySQL also includes a new scheduler that enhances SQL awareness, automates read/write splitting, and handles failovers more intelligently. This leads to faster queries, increased reliability, and more efficient cluster resource usage.

Which proxy should you choose?

- **HAProxy:** Choose HAProxy if you need a lightweight, TCP-level load balancer with minimal configuration. Note that for read/write splitting, your clients must connect to different HAProxy ports based on the query type.
- **ProxySQL:** Opt for ProxySQL if you want built-in read/write splitting, advanced query-level control, and automated failover logic right out of the box.

Each proxy brings its own resource requirements and advantages. We offer additional guidance on selecting the right proxy for your environment, plus detailed recommendations on resource planning and best practices. Review these carefully to ensure your choice fits your operational and performance needs.

To switch between proxies, update your Custom Resource to set `haproxy.enabled` to `false` and `proxysql.enabled` to `true`. Apply the changes, and the Operator will handle the transition for you by restarting the relevant proxy Pods.

With this improvement you now control your proxy choice at runtime, and ProxySQL brings smarter routing and resilience right into the Operator.

ProxySQL 3 support

You can now deploy ProxySQL 3 with Percona Operator for MySQL. This gives you more flexibility and control over how your applications connect to MySQL inside Kubernetes. Here's what you get:

- Dual-password support enables you to introduce a new password while the old one is still valid. As a result, you can rotate passwords without downtime
- Enhanced event and query logging gives you real-time visibility into query behavior and application traffic.
- ProxySQL now logs the actual values bound to prepared statements. This helps you debug queries more effectively, since you see what data was passed instead of just placeholders
- Event logs now include metadata about ProxySQL version and format. This makes it easier to track and audit logs across upgrades in your Operator-managed deployments.

Direct-access backups: Improved performance and reliability with sidecars (tech preview)

By default, the Operator makes backups using the SST method. This creates a separate backup Pod with Percona XtraBackup, while the database node enters Donor state and stops serving client requests. SST backups can also fail with cryptic network errors, making root cause analysis and recovery difficult.

Starting with version 1.19.0, you can make backups via the XtraBackup sidecar container. The Operator deploys a sidecar with XtraBackup inside each Percona XtraDB Cluster Pod. This sidecar makes a backup and uploads it to the remote backup storage. The database Pod doesn't change its state to Donor and keeps accepting client requests.

Using the sidecar method provides a direct access to data thus boosting backup performance. The sidecar container constantly runs in the database Pod, so you have constant access to logs and status, which simplifies troubleshooting.

To enable the XtraBackup sidecar container backup method, set `PXCO_FEATURE_GATES=XtrabackupSidecar=true` environment variable in the Operator Deployment. This functionality is in the tech preview stage and currently supports only cloud storages. We encourage you to try it out in your testing or staging environments and leave your feedback.

Future enhancements such as support of PVC volumes, backup encryption and incremental backups are planned for future releases.

To learn more about XtraBackup sidecar container backup method, see our documentation.

Ensure only up to date data is served during backups

When a node donates data during backups or SST, it enters into the DONOR state. At this point, the node should no longer handle client connections. HAProxy's external check correctly blocked new connections to the Donor node but allowed existing sessions to remain active. Those lingering sessions could return slow or outdated results.

The HAProxy default configuration now includes the `on-marked-down shutdown-sessions` directive. As soon as HAProxy marks a node as down, all active connections are immediately closed and clients reconnect to remaining active nodes. This ensures that only fresh, up to date data is served during backups.

Automatic cleanup of backup and restore Jobs and associated Pods

The Operator creates a dedicated Job and Pod for every backup and restore operation. Previously, these Jobs and Pods remained in the cluster even after the operation was finished, and you had to manually delete them to free up resources.

Now, you can offload this task to the Operator. Specify a time-to-live (TTL) for backup and restore Jobs once the operation is finished. When the TTL expires, the Operator automatically deletes the Job and its associated Pod.

Modify your Custom Resource as follows:

```
backup:
  image: perconalab/percona-xtradb-cluster-operator:main-pxc8.0-backup
  ttlSecondsAfterFinished: 3600
```

This setting is global. It applies to all on-demand and scheduled backups, and all restores.

The Operator also ensures reliability: if a backup or restore takes longer than the configured TTL, it applies the `internal.percona.com/keep-job` finalizer to allow the operation to finish. After the operation completes either with the `Succeeded` or the `Failed` status, the finalizer is removed and the Job is cleaned up.

This improvement reduces manual maintenance overhead, gives you control over the processes lifetime for debugging or auditing purposes and helps keep your cluster healthy and efficient. By reducing unnecessary resource buildup, you gain smoother operations, lower maintenance overhead and improved reliability in production environments.

Improved backup identification for point-in-time recovery readiness

When a backup contains binlog gaps, the Operator now creates a `<backup-name>.pitr-not-ready` file in the backup storage. This file makes it easy to identify which backups are appropriate for point in time recovery both in the storage and when listing backup objects.

Before starting a restore, the Operator checks for this marker file and blocks unsafe restores, protecting you from incomplete recovery attempts. If needed, you can override this safeguard by adding the `percona.com/unsafe-pitr` annotation to the Restore object. Use this override with caution, as this is an unsafe configuration.

Attach external PVCs for shared data access across applications and the database cluster

Sometimes your database needs more than its own internal storage. For example, it needs access to reference files, shared configuration files or lookup tables generated outside the database but still essential for queries and procedures.

You can now attach auxiliary pre-existing PVCs and mount that external data directly into your database, ProxySQL or HAProxy pods in a clean, declarative way using the Custom Resource.

This example configuration shows how to attach external PVC to the XtraDB Cluster Pods:

```
pxc:
  extraPVCs:
    - name: extra-data-volume
      claimName: my-extra-storage
      mountPath: /var/lib/mysql-extra
      readOnly: false
```

This improvement gives you a reliable way to separate internal database storage from external domain data, update shared datasets independently, and still benefit from the Operator's automation and resilience.

Customize password generation by the Operator

By default, the Operator generates user passwords using alphanumeric characters plus a set of special symbols. Some tools such as MySQL Prometheus Exporter or mysqlsh don't support certain symbols, which can make those passwords invalid.

To improve compatibility and user experience, you can now customize password generation parameters in the Custom Resource:

```
spec:
 passwordGenerationOptions:
   symbols: "!#$%&()*+,-.<=>?@[]^_{}~"
   maxLength: 20
   minLength: 16
```

This enhancement lets you keep the convenience of automated password generation and at the same time ensure compliance with the tools and environments you integrate with the Operator.

Configure memory allocator in the Operator

By default, Percona Operator for XtraDB Cluster uses the system allocator (`libc`) to manage memory. While this works for most cases, alternative allocators such as `jemalloc` and `tcmalloc` can improve performance and reduce fragmentation in high-traffic workloads.

To have more flexibility, you can now configure the memory allocator directly in the Custom Resource:

```
spec:
   pxc:
     mysqlAllocator: jemalloc
```

Supported values are:

- `jemalloc`
- `tcmalloc`
- `libc` (default, used when no value is set)

If you have already configured the memory allocator via the environment variable, the Operator will respect that setting and use it instead of the Custom Resource value.

This enhancement lets you fine-tune memory management for your cluster while keeping compatibility with existing configurations.

Deprecation, rename, removal

Removed in 1.19.0:

- `proxysql.readinessDelaySec` and `proxysql.livenessDelaySec` fields are removed as redundant

Deprecated (will be removed in 1.22.0):

- `pxc.livenessDelaySec`. Use `pxc.livenessProbes.initialDelaySeconds`
- `pxc.readinessDelaySec`. Use `pxc.readinessProbes.initialDelaySeconds`
- `haproxy.livenessDelaySec`. Use `haproxy.livenessProbes.initialDelaySeconds`
- `haproxy.readinessDelaySec`. Use `haproxy.readinessProbes.initialDelaySeconds`

Changelog

New Features

- K8SPXC-1332 - Added the ability to load custom SSL certificates for backup operations to S3 storage. This enables secure communication with S3-compatible storage using the certificates approved and trusted by your company (Thank you Azam Abdoelbasier for submitting this request).
- K8SPXC-1494 - Added the ability to configure the duration of TLS certificates created by `cert-manager`. This allows users to customize certificate lifecycles to meet their specific security requirements.
- K8SPXC-1576 - Added the ability to use the XtraBackup sidecar container instead of SST for creating backups. This provides an alternative, potentially more efficient backup method.
- K8SPXC-1688 - Added ability to mount pre-existing auxiliary PVCs as volumes to database and proxy pods. This facilitates easier integration with external data and storage resources (Thank you Emin AKTAS for submitting the request and contributing to it).
- K8SPXC-1733 - Added the ability to customize password generation parameters, such as length and character sets via the Custom Resource. This ensures smooth operation of the tools with specific requirements towards password and leverages the automatic password generation of the Operator (Thank you user fydrah for submitting the request and contributing to it).
- K8SPXC-1734 - Introduced configurable HAProxy backend health check parameters that can be tuned without overriding the entire configuration. This simplifies performance tuning for high-availability setups (Thank you Tim Stoop for submitting the request and contributing to it).

Improvements

- K8SPXC-735 - Added the support of the `pxc_scheduler_handler` ProxySQL scheduler for SQL-aware routing and effective read/write splitting. This allows for better utilization of resources by distributing read-only traffic across the entire cluster while routing write requests only to the writer node.
- K8SPXC-992 - Improved binlog naming to prevent potential collisions between different collectors. The updated naming convention ensures unique and consistent identification of binary log files in storage.
- K8SPXC-1144 - Introduced a mechanism to mark S3 backups as PITR-unready if binlog gaps are detected. This prevents users from attempting invalid point-in-time recoveries using inconsistent backups.
- K8SPXC-1214 - Added an option to automatically clean up completed backup and restore jobs and their associated pods. This improvement helps reduce pressure on the Kubernetes API by removing stale resources (Thank you Alexandre Barth for reporting this issue).
- K8SPXC-1319 - Enhanced the operator to support running multiple backup restores for different clusters in parallel. This removes a previous limitation that blocked concurrent restore operations across the environment.
- K8SPXC-1327 - Added the ability to change the memory allocator for MySQL to improve memory management efficiency. This change helps optimize the overall memory footprint of PXC pods.
- K8SPXC-1373 - Improved core dump handling to ensure that crashed instances can recover more reliably after an SST. This enhancement aids in diagnosing and recovering from unexpected server failures.
- K8SPXC-1431 - Improved the delete-backup finalizer logic to correctly handle the deletion of on-demand backup PVCs.
- K8SPXC-1470 - Added the ability to switch between HAProxy and ProxySQL within an existing cluster. This provides users with more flexibility to change their load-balancing solution as their needs evolve.
- K8SPXC-1511 - Added the support of data at rest encryption for Percona XtraDB Cluster 8.4 via the `keyring vault` component. This change ensures compatibility with the latest security architecture of MySQL 8.4.
- K8SPXC-1525 - Deprecated `pxc.livenessDelaySec` option in favor of more consistent liveness probe parameters. The Operator now prioritizes standard probe configurations to manage Pod lifecycle.
- K8SPXC-1568 - Updated the Operator's password generation logic to prevent the '*' character from being used as the first character. This avoids potential issues with certain authentication plugins and command-line tools.
- K8SPXC-1594 - Improved the database upgrade logic to prevent the controller from being blocked during the operation. This ensures that the Operator remains responsive to other cluster changes while an upgrade is in progress.
- K8SPXC-1628 - Added support for `tcmalloc` as an alternative memory allocator in PXC images. This gives users additional options to tune and reduce the

memory footprint of their database workloads.

- K8SPXC-1647 - Implemented extended exit codes for garbd to provide better diagnostic information for different failure scenarios. This helps users identify the root cause of SST and joining issues faster.
- K8SPXC-1668 - Added support for ProxySQL 3, providing users with access to the latest features and performance improvements of the load balancer.
- K8SPXC-1683 - Expanded smart update tests to include PXC 8.4 and PMM 3, ensuring stable upgrade paths for the latest versions.
- K8SPXC-1703 - Added the support of the `generateEmbeddedObjectMeta` option, improving the template handling for sidecars and extra PVCs (Thank you Emin AKTAS for reporting and contributing to this issue).
- K8SPXC-1748 - Eliminated runtime CREATE FUNCTION statements in the PITR collector to avoid unnecessary Galera TOI (Total Order Isolation) events. This reduces the performance impact on the cluster when the PITR sidecar starts or restarts.

Bugs Fixed

- K8SPXC-926 - Fixed an issue where a failed smart update on one cluster could block the Operator from managing other clusters in multi-cluster environments. The controller now handles update failures more gracefully without impacting independent resources.
- K8SPXC-1379 - Fixed an issue where monit container resource values in ProxySQL pods did not correctly reflect the values specified in the PXC cluster definition. The operator now ensures that ProxySQL resource attributes are properly applied to the monitoring sidecar containers.
- K8SPXC-1424 - Resolved a certificate renewal issue where CA certificates in TLS secrets could expire before server certificates were renewed. The Operator logic was updated to align renewal intervals for CA and server certificates when using cert-manager.
- K8SPXC-1581 - Corrected the order of options in the restore prepare job to ensure that `--defaults-file` is passed as the first option to xtrabackup. This fix ensures that custom configuration files are correctly prioritized during the restore process.
- K8SPXC-1617 - Fixed a binlog gap issue in Point-in-Time Recovery (PITR) that caused repeated test failures. Users are now advised to perform a full backup after each PITR restore to ensure data consistency and prevent gaps.
- K8SPXC-1632 - Added missing SecurityContext configurations for ProxySQL sidecar containers to enhance pod security. This change ensures that all sidecars follow the defined security standards of the cluster.
- K8SPXC-1655 - Fixed a failure in xtrabackup when using LZ4 compression on RHEL9-based Percona XtraDB Cluster images. The fix addresses compatibility issues with the latest compression libraries in the operating system environment.
- K8SPXC-1686 - Improved backup error handling to ensure that providing an invalid Percona XtraBackup image results in a failed status. A new timeout field was introduced to prevent backup objects from hanging indefinitely in a starting state.
- K8SPXC-1687 - Fixed the `copy-backup.sh` script to correctly handle and copy cloud-based backups stored in S3 or Azure. This ensures that the utility script works consistently across all supported storage types.
- K8SPXC-1701 - Ensured that MySQL configurations are correctly mounted to the restore prepare job. This fix allows the restore process to use custom MySQL settings defined in the cluster (Thank you Emin AKTAS for reporting and contributing to this issue).
- K8SPXC-1702 - Added the ability to override time zones for backup jobs. This resolves issues where time-dependent operations could fail due to missing timezone definitions (Thank you Emin AKTAS for reporting and contributing to this issue).
- K8SPXC-1721 - Added a sleep interval to the recovery loop to prevent high CPU usage spikes when containers restart.
- K8SPXC-1725 - Fixed a condition where a cluster could return stalled data during a backup or SST operation by adding the on-marked-down shutdown-sessions directive to HAProxy default configuration. This ensures that only fresh, up to date data is served during backups.
- K8SPXC-1726 - Resolved a deployment breakage in the Operator version 1.18.0 via Helm caused by PMM 3 client incompatibilities. The fix ensures a smoother upgrade path for users migrating to newer versions of PMM (Thank you Antonio Falzarano for reporting and contributing to this issue).
- K8SPXC-1760 - Fixed the handling of the crVersion field in the Helm chart templates. The Operator now correctly considers the version defined in `values.yaml` when generating the cluster configuration.
- K8SPXC-1771 - Fixed the issue with excessive logging by the backup controller when backups are suspended or resumed due to an unready cluster. This improves log readability and reduces unnecessary diagnostic noise.
- K8SPXC-1772 - Fixed a state transition bug where unsuspended backups could move directly from 'Starting' to 'Succeeded' without entering the 'Running' state. This ensures accurate tracking and visibility of the backup process status.

Documentation Improvements

- K8SPXC-1747 - Improved the documentation with the information about available environment variables for cluster components and the Operator. Documented the `S3_WORKERS_LIMIT` environment variable to allow throttling of backup deletions.
- K8SPXC-1661 - Updated the operator documentation to reflect that PMM 3 uses service accounts instead of API keys. This ensures that users can correctly configure monitoring integration with the latest versions of PMM.
- K8SPXC-1663 - Improved documentation for Point-in-Time Recovery steps. The updated documentation properly separates and sequences the recovery instructions for improved readability.

Supported Software

The Operator was developed and tested with the following software:

- Percona XtraDB Cluster versions 8.4.7-7.1, 8.0.44-35.1, and 5.7.44-31.65
- Percona XtraBackup versions 8.4.0-5.1, 8.0.35-34.1, and 2.4.29
- HAProxy 2.8.17
- ProxySQL 2.7.3-1.2, 3.0.1-1.2
- LogCollector based on fluent-bit 4.0.1-1
- PMM Client 2.44.1-1 and 3.5.0

Other options may also work but have not been tested.

Supported Platforms

Percona Operators are designed for compatibility with all CNCF-certified Kubernetes distributions. Our release process includes targeted testing and validation on major cloud provider platforms and OpenShift, as detailed below:

- Google Kubernetes Engine (GKE) 1.31 - 1.33
- Amazon Elastic Container Service for Kubernetes (EKS) 1.32 - 1.34
- Azure Kubernetes Service (AKS) 1.32 - 1.34
- OpenShift 4.17 - 4.20
- Minikube 1.37.0 based on Kubernetes 1.34.0

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.

Percona certified images

Find Percona's certified Docker images that you can use with the Percona Operator for MySQL based on Percona XtraDB Cluster in the following table.

Image	Digest
percona/percona-xtradb-cluster-operator:1.19.0	6ccbac5e74f5b5309fd4788c5b8d91d5abd01850a4a356ad9eff9f82d20afb51
percona/percona-xtradb-cluster-operator:1.19.0 (ARM64)	1ed2a5ab22ee7588aa17ec2339876dc72c9724dc9a81506ff449a2b1aa085024
percona/percona-xtradb-cluster:8.4.7-7.1	5b18775ad62a1c5f8d8bffc63a1518360d2e7a82c1bed7cbd8a15011f6cdff9f
percona/percona-xtradb-cluster:8.4.7-7.1 (ARM64)	4c3785f5befd001ca3ae035f42c9b586447b874158b0d9b26afb8ff87658829f
percona/percona-xtradb-cluster:8.0.44-35.1	f91026ec8427ace53dc31f3b00ec14cebdc0868bda921ae0713e8ad3af71ba1f
percona/percona-xtradb-cluster:8.0.44-35.1 (ARM64)	33a0f32c1d42cf6e74f45aeebd6422cfdea6c8c8bc3cce600e46c4661b0183be
percona/percona-xtradb-cluster:5.7.44-31.65	36fafdef46485839d4ff7c6dc73b4542b07031644c0152e911acb9734ff2be85
percona/percona-xtrabackup:8.4.0-5.1	1b81d06b1beb6a126b493d11532a5c71d1b1c2a1d13cb655e3cc5760c0896035
percona/percona-xtrabackup:8.4.0-5.1 (ARM64)	ca40d7975ae39bd5dd652487a1389b823cbf788e9948db6cf53ebb0d3f57c51b
percona/percona-xtrabackup:8.0.35-34.1	967bafa0823c90aa8fa9c25a9012be36b0deef64e255294a09148d77ce6aea68
percona/percona-xtrabackup:8.0.35-34.1 (ARM64)	83f814dca9ed398b585938baa86508bda796ba301e34c948a5106095d27bf86e
percona/percona-xtrabackup:2.4.29	11b92a7f7362379fc6b0de92382706153f2ac007ebf0d7ca25bac2c7303fdf10
percona/fluentbit:4.0.1-1	65bdf7d38cbceed6b6aa6412aea3fb4a196000ac6c66185f114a0a62c4a442ad
percona/fluentbit:4.0.1-1 (ARM64)	dabda77b298b67d30d7f53b5cdb7215ad19dabb22b9543e3fd8aedb74ab24733

percona/pmm-client:3.5.0	352aee74f25b3c1c4cd9dff1f378a0c3940b315e551d170c09953bf168531e4a
percona/pmm-client:3.5.0 (ARM64)	cbbb074d51d90a5f2d6f1d98a05024f6de2ffdcb5acab632324cea4349a820bd
percona/pmm-client:2.44.1-1	52a8fb5e8f912eef1ff8a117ea323c401e278908ce29928dafc23fac1db4f1e3
percona/pmm-client:2.44.1-1 (ARM64)	390bfd12f981e8b3890550c4927a3ece071377065e001894458047602c744e3b
percona/haproxy:2.8.17	ef8486b39a1e8dca97b5cdf1135e6282be1757ad188517b889d12c5a3470eeda
percona/haproxy:2.8.17 (ARM64)	bbc5b3b66ac985d1a4500195539e7dff5196245a5a842a6858ea0848ec089967
percona/proxysql2:2.7.3-1.2	719d0ab363c65c7f75431bbed7ec0d9f2af7e691765c489da954813c552359a2
percona/proxysql2:2.7.3-1.2 (ARM64)	4c4d094652c9f2eb097be5d92dcc05da61c9e8699ac7321def959d5a205a89f7
percona/proxysql3:3.0.1-1.2	f3fb43d4ef2467f207ecd66c51414520a100a0474807f307775a985303c56ec5
percona/proxysql3:3.0.1-1.2 (ARM64)	d21ba769b9e364a1a0c1d5e9d3b6287e8051efcf79cd6ec3df5756278961bbec

Percona Operator for MySQL based on Percona XtraDB Cluster 1.18.0 (2025-08-14)

Installation

Release Highlights

This release of Percona Operator for MySQL based on Percona XtraDB Cluster includes the following new features and improvements:

PMM3 support

The Operator is natively integrated with PMM 3, enabling you to monitor the health and performance of your Percona Distribution for MySQL deployment and at the same time enjoy enhanced performance, new features, and improved security that PMM 3 provides.

Note that the Operator supports both PMM2 and PMM3. The decision on what PMM version is used depends on the authentication method you provide in the Operator configuration: PMM2 uses API keys while PMM3 uses service account token. If the Operator configuration contains both authentication methods with non-empty values, PMM3 takes the priority.

To use PMM, ensure that the PMM client image is compatible with the PMM Server version. Check Percona certified images for the correct client image.

For how to configure monitoring with PMM, see the documentation.

Improved monitoring for clusters in multi-region or multi-namespace deployments in PMM

Now you can define a custom name for your clusters deployed in different data centers. This name helps Percona Management and Monitoring (PMM) Server to correctly recognize clusters as connected and monitor them as one deployment. Similarly, PMM Server identifies clusters deployed with the same names in different namespaces as separate ones and correctly displays performance metrics for you on dashboards.

To assign a custom name, define this configuration in the Custom Resource manifest for your cluster:

```
spec:
  pmm:
    customClusterName: testClusterName
```

More resilient database restores without matching user Secrets

You no longer need matching user Secrets between your backup and your target cluster to perform a restore. The Operator now has a post-restore step that changes user passwords in the restored database to the ones from the local Secret. Also, it creates missing system users and adds missing grants.

This flow is the same regardless of whether you restore to the same cluster or to a completely new one.

The removal of this major roadblock to have a Secret for restores makes your disaster recovery process smoother and more reliable. This enhancement makes managing databases on Kubernetes more robust and operator-friendly.

Improved backup retention for streamlined management of scheduled backups in cloud storage

A new backup retention configuration gives you more control over how backups are managed in storage and retained in Kubernetes.

With the `deleteFromStorage` flag , you can disable automatic deletion from AWS S3 or Azure Blob storage and instead rely on native cloud lifecycle policies. This makes backup cleanup more efficient and better aligned with flexible storage strategies.

The legacy `keep` option is now deprecated and mapped to the new `retention` block for compatibility. We encourage you to start using the `backup.schedule.retention` configuration:

```
schedule:
  - name: "sat-night-backup"
    schedule: "0 0 * * 6"
    retention:
      count: 3
      type: count
      deleteFromStorage: true
    storageName: s3-us-west
```

Note that if you have both `backup.schedule.keep` and `backup.schedule.retention` defined, the `backup.schedule.retention` takes precedence.

Added labels to identify the version of the Operator

Custom Resource Definition (CRD) is compatible with the last three Operator versions. To know which Operator version is attached to it, we've added labels to all Custom Resource Definitions. The labels help you identify the current Operator version and decide if you need to update the CRD. To view the labels, run: `kubectl get crd perconaxtradbclusters.pxc.percona.com --show-labels`.

Cross-site replication is now supported for Percona XtraDB Cluster 8.4

Cross-site replication is now available with Percona XtraDB Cluster 8.4.x, lifting one of the limitations in the Operator for this database version. This enhancement marks a significant step toward general availability of Percona XtraDB Cluster 8.4 in the Operator by enabling multi-site deployments and improving resilience across distributed environments.

Deprecation, Rename and Removal

- The `pxc.expose.loadBalancerIP`, `haproxy.exposePrimary.loadBalancerIP`, `haproxy.exposeReplicas.loadBalancerIP` and `proxysql.expose.loadBalancerIP` keys are deprecated. The `loadBalancerIP` field is also deprecated upstream in Kubernetes due to its inconsistent behavior across cloud providers and lack of dual-stack support. As a result, its usage is strongly discouraged.

We recommend using cloud provider-specific annotations instead, as they offer more predictable and portable behavior for managing load balancer IP assignments.

The `pxc.expose.loadBalancerIP`, `haproxy.exposePrimary.loadBalancerIP`, `haproxy.exposeReplicas.loadBalancerIP` and `proxysql.expose.loadBalancerIP` keys are scheduled for removal in future releases.

- The `backup.schedule.keep` field is deprecated and will be removed after release 1.21.0. We recommend using the `backup.schedule.retention` instead as follows:

```
schedule:
  - name: "sat-night-backup"
    schedule: "0 0 ** 6"
    retention:
      count: 3
      type: count
      deleteFromStorage: true
    storageName: s3-us-west
```

- New repositories for Percona XtraBackup and Logcollector

Now the Operator uses the official Percona Docker images for the `percona-xtrabackup` and `logcollector` components. Pay attention to the new image repositories when you upgrade the Operator and the database. Check the Percona certified images for exact image names.

- Changes for Helm charts:
- PMM3 is now the default. To keep using PMM2, set the `pmm.tag: 2.44.1`
- If you install or upgrade the Operator with default manifests using Helm charts on Openshift 4.19, you must use the `docker.io` registry prefix to guarantee successful download from the DockerHub `percona-xtradb-cluster` repository. Read the Considerations for using OpenShift 4.19 section for more information.

Known limitations

Considerations for using OpenShift 4.19

Starting with OpenShift 4.19, the way images with not fully qualified names are pulled has changed for repositories that share the same repository name on DockerHub and Red Hat Marketplace. By default the tags are pulled from Red Hat Marketplace. Specifying not fully qualified image names may result in the `ImagePullBackOff` error.

- **OLM installation:** Images are provided with the fully qualified names and are pulled from the Red Hat Marketplace/DockerHub registry.
- **Manual install/update with default manifests:** Images must use the `docker.io` registry prefix to guarantee successful download from the Dockerhub `percona-xtradb-cluster` repository.

For manual installation or update, follow the instructions below:

Install on OpenShift 4.19

1. Clone the Operator repository:

```
$ git clone -b v1.18.0 https://github.com/percona/percona-xtradb-cluster-operator
$ cd percona-xtradb-cluster-operator
```

1. Edit the `deploy/bundle.yaml` file.
   - Locate the Deployment custom resource for the Operator.
   - Update the `spec.image` field to

```
docker.io/percona/percona-xtradb-cluster-operator:1.19.0
```

2. Apply the updated `deploy/bundle.yaml` file

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

3. Install Percona XtraDB Cluster:

```
$ oc create -f deploy/secrets.yaml
```

Update the Operator to 1.19.0

1. Check all clusters managed by the Operator to see if `initContainer.image` is set.
   - If defined: skip the next step.
   - If undefined: proceed to step 2.
2. Apply a patch to the clusters with undefined `initContainer.image` to define this image with the `docker.io` registry in the image path:

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
  "spec": {
    "initcontainer": {
      "image": "docker.io/percona/percona-xtradb-cluster-operator:1.17.0"
    }
  }
}'
```

Important! This command triggers the restart of your clusters. Wait till they restart and report the Ready status

1. Update the Operator deployment and specify the `docker.io` registry name in the image path:

```
$ kubectl patch deployment percona-xtradb-cluster-operator \
-p'{"spec":{"template":{"spec":{"containers":[{"name":"percona-xtradb-cluster-operator","image":"docker.io/percona/percona-
xtradb-cluster-operator:1.19.0"}]}}}}'
```

2. Update the Custom Resource version and the database cluster. Specify the `initContainer` image with the `docker.io` registry name in the path. Pay attention to the changed repositories for PXB and logcollector:

```
$ kubectl patch pxc cluster1 --type=merge --patch '{
  "spec": {
    "crVersion": "1.19.0",
    "initContainer": "docker.io/percona/percona-xtradb-cluster-operator:1.19.0",
    "pxc":{ "image": "docker.io/percona/percona-xtradb-cluster:8.0.44-35.1" },
    "proxysql": { "image": "docker.io/percona/proxysql2:2.7.3-1.2" },
    "haproxy":  { "image": "docker.io/percona/haproxy:2.8.17" },
    "backup":   { "image": "docker.io/percona/percona-xtrabackup:8.0.35-34.1" },
    "logcollector": { "image": "docker.io/percona/fluentbit:4.0.1-1" },
    "pmm":      { "image": "docker.io/percona/pmm-client:2.44.1-1" }
       }}'
  }
}'
```

Changleog

New Features

- K8SPXC-1284 - Add the ability to configure protocol for peer-list DNS SRV lookups
- K8SPXC-1599 - Allowed setting `loadBalancerClass` service type and using a custom implementation of a load balancer rather than the cloud provider default one

Improvements

- K8SPXC-1375 - Added a new retention configuration to allow users to delegate backup cleanup to cloud lifecycle policies (Thank you user Tristan for reporting this issue)
- K8SPXC-1376 - Added the ability to restore from backup without a matching Secret resource
- K8SPXC-1399 - Added a documentation how to set up a disaster recovery system and transfer workloads between sites
- K8SPXC-1415 - Updated the `percona-xtrabackup` image to use the official `percona-xtrabackup` Docker image
- K8SPXC-1430 - Improved handling of autogenerated certificates depending on the `delete-ssl` finalizer configuration
- K8SPXC-1448, K8SPXC-1449 - Improved the `pvc-resize` test by using a custom storage class for EKS, reducing errors and improving the quota handling during resize
- K8SPXC-1450 - Improved PVC resizing behavior when reducing the storage size by reverting the values when the quota is reached
- K8SPXC-1472 - Deprecated the `loadBalancerIP` field due to its deprecation upstream
- K8SPXC-1513 - Added PXC 8.4 support for version service
- K8SPXC-1529 - Added support for cross-site replication with MySQL 8.4.0 by adding the use of `authentication_policy` instead of `default_authentication_plugin`
- K8SPXC-1553 - Added support for PMM v3
- K8SPXC-1560 - Added the warning about CRDs not being upgraded automatically after helm upgrade to the output
- K8SPXC-1566 - Improved reconciliation of replicationChannels without proxy Pods by starting the database Pod bypassing the proxy (Thank you Justin Reasoner for contributing to this issue)
- K8SPXC-1569 - Added Labels for Custom Resource Definitions (CRD) to identify the Operator version attached to them
- K8SPXC-1597 - Improve the scheduled backups behavior for a cluster in an unhealthy state by postponing the job until the cluster reports the healthy status
- K8SPXC-1605 - Introduced Azure CLI for checking if backup objects/folders exist in Azure storage
- K8SPXC-1612 - Added the `imagePullSecrets` for PMM image
- K8SPXC-1615 - Added the ability to define a custom cluster name for `pmm-admin` component
- K8SPXC-1624 - Deleted deprecated finalizers code
- K8SPXC-1669 - Improve the backup flow by generating a default endpoint URL for a storage from a region if it is not provided (Thank you Bernard Grymonpon for reporting this issue)
- K8SPXC-1677 - Document the changed behavior with pulling images for default manifests on OpenShift 4.19 and update install and update instructions

Bugs Fixed

- K8SPXC-1312 - Fixed the issue with labels not being updated automatically for point-in-time recovery deployment upon Custom Resource changes
- K8SPXC-1347 - Fixed the issue with point-in-time recovery failing due to TLS configuration mismatch between the server and the point-in-time recovery job by configuring it to use TLS if is required by the server.
- K8SPXC-1382 - Fixed the issue with backup failing on AWS if using IAM profile without credentialsSecret by using credentialsSecret only when explicitly specified and relying on IAM roles instead (Thank you Itiel Olenick for reporting this issue)
- K8SPXC-1541 - Fixed Telemetry module to to consider both empty string "" and comma separated namespaces in cluster-wide mode
- K8SPXC-1548 Fixed the issue with deleting old backups on Google Cloud Storage by url-decoding the object path before deleting it (Thank you Mateusz Gruszkiewicz for reporting this issue)
- K8SPXC-1631 - Fixed the issue with the Operator restarting pod-0 after the cluster is ready. The issue is caused by ConfigMap and StatefulSet being created too close to each other and Kubernetes API can't return the newly created ConfigMap before creating the StatefulSet. The issue is fixed by reconciling the StatefulSet after the reconciliation of ConfigMap is completed.
- K8SPXC-1664 - Fixed the use of the proper script to check PXC nodes when adding them by HAProxy

Supported Software

The Operator was developed and tested with the following software:

- Percona XtraDB Cluster versions 8.4.5-5.1 (Tech preview), 8.0.42-33.1, and 5.7.44-31.65
- Percona XtraBackup versions 8.4.0-3, 8.0.35-34.1, and 2.4.29
- HAProxy 2.8.15-1
- ProxySQL 2.7.3
- LogCollector based on fluent-bit 4.0.1
- PMM Client 2.44.1 and 3.3.1

Other options may also work but have not been tested.

Supported Platforms

Percona Operators are designed for compatibility with all CNCF-certified Kubernetes distributions. Our release process includes targeted testing and validation on major cloud provider platforms and OpenShift, as detailed below for Operator version 1.16.0:

- Google Kubernetes Engine (GKE) 1.30 - 1.33
- Amazon Elastic Container Service for Kubernetes (EKS) 1.30 - 1.33
- Azure Kubernetes Service (AKS) 1.30 - 1.33
- OpenShift 4.15 - 4.19
- Minikube 1.36.0 based on Kubernetes 1.33.1

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.

Percona certified images

Find Percona's certified Docker images that you can use with the Percona Operator for MySQL based on Percona XtraDB Cluster in the following table.

Images released with the Operator version 1.18.0:

Image	Digest
percona/percona-xtradb-cluster-operator:1.18.0 (x86_64)	0eca0b096482c7d09792c15fee00dbdcd0fbf3cd487dab60eb2774b025681e85
percona/percona-xtradb-cluster-operator:1.18.0 (ARM64)	bdb7a0ff6b78e98b16f8b521e91682202b6d404202283b34b8168013d5c06356
percona/haproxy:2.8.15	49e6987a1c8b27e9111ae1f1168dd51f2840eb6d939ffc157358f0f259819006
percona/proxysql2:2.7.3	51fedf9de05e4f130d5b08388511536fb1e1050a24ffc21bedb0f0b61a236567
percona/percona-xtrabackup:8.4.0-3.1	01071522753ad94e11a897859bba4713316d08e493e23555c0094d68da223730
percona/percona-xtrabackup:8.0.35-34.1	2dc127b08971051296d421b22aa861bb0330cf702b4b0246ae31053b0f01911e
percona/percona-xtrabackup:2.4.29	11b92a7f7362379fc6b0de92382706153f2ac007ebf0d7ca25bac2c7303fdf10
percona/fluentbit:4.0.1	a4ab7dd10379ccf74607f6b05225c4996eeff53b628bda94e615781a1f58b779
percona/pmm-client:3.3.1	29a9bb1c69fef8bedc4d4a9ed0ae8224a8623fd3eb8676ef40b13fd044188cb4
percona/pmm-client:2.44.1-1	52a8fb5e8f912eef1ff8a117ea323c401e278908ce29928dafc23fac1db4f1e3
percona/percona-xtradb-cluster:8.4.5-5.1	918c54c11c96bf61bb3f32315ef6b344b7b1d68a0457a47a3804eca3932b2b17
percona/percona-xtradb-cluster:8.0.42-33.1	476851339090e44bb72760ae718fc36beb73a6028a29459e849271649018d546
percona/percona-xtradb-cluster:8.0.41-32.1	d9c84884a12631306d5a33a079e30bf7b65d3d380b07b397d7b1b6a642cc6bff

percona/percona-xtradb-cluster:8.0.39-30.1	6a53a6ad4e7d2c2fb404d274d993414a22cb67beecf7228df9d5d994e7a09966
percona/percona-xtradb-cluster:8.0.36-28.1	b5cc4034ccfb0186d6a734cb749ae17f013b027e9e64746b2c876e8beef379b3
percona/percona-xtradb-cluster:8.0.35-27.1	1ef24953591ef1c1ce39576843d5615d4060fd09458c7a39ebc3e2eda7ef486b
percona/percona-xtradb-cluster:5.7.44-31.65	36fafdef46485839d4ff7c6dc73b4542b07031644c0152e911acb9734ff2be85
percona/percona-xtradb-cluster:5.7.42-31.65	9dab86780f86ec9caf8e1032a563c131904b75a37edeaec159a93f7d0c16c603
percona/percona-xtradb-cluster:5.7.39-31.61	9013170a71559bbac92ba9c2e986db9bda3a8a9e39ee1ee350e0ee94488bb6d7
percona/percona-xtradb-cluster:5.7.36-31.55	c7bad990fc7ca0fde89240e921052f49da08b67c7c6dc54239593d61710be504
percona/percona-xtradb-cluster:5.7.34-31.51	f8d51d7932b9bb1a5a896c7ae440256230eb69b55798ff37397aabfd58b80ccb

Percona Operator for MySQL based on Percona XtraDB Cluster 1.17.0 (2025-04-14)

Installation

Release Highlights

This release of Percona Operator for MySQL based on Percona XtraDB Cluster includes the following new features and improvements:

Improved observability for HAProxy and ProxySQL

Get insights into the HAProxy and ProxySQL performance by connecting to their statistics pages. Use the `cluster-name-haproxy:8084` and `cluster-name-proxysql:6070` endpoints to do so. Learn about other available ports in the documentation.

Improved cluster load management during backups

If parallel backups overload your cluster, you can turn off parallel execution to prevent this. Previously, this meant that you could only run one backup at a time - no new backups could start until the current one was finished. Now, the Operator queues backups and runs them one after another automatically. You can fine-tune the backup sequence by setting the start time for all backups or for a specific on-demand one using the `spec.backup.startingDeadlineSeconds` Custom Resource option. This provides greater control over backup operations.

Another improvement is for the case when your database cluster becomes unhealthy, for example, when a Pod crashes or restarts. The Operator suspends running backups to reduce the cluster's load. Once the cluster recovers and reports a Ready status, the Operator resumes the suspended backup. To further offload the cluster during an unhealthy state, you can configure how long a backup remains suspended by using the `spec.backup.suspendedDeadlineSeconds` Custom Resource option. If this time expires before the cluster recovers, the backup is marked as "failed."

Monitor PMM Client health and status

Percona Monitoring and Management (PMM) is a great tool to monitor the health of your database cluster. Now you can also learn if PMM itself is healthy using probes - a Kubernetes diagnostics mechanism to check the health and status of containers. Use the `spec.pmm.readinessProbes.*` and `spec.pmm.livenessProbes.*` Custom Resource options to fine-tune Readiness and Liveness probes for PMM Client.

Improved observability of binary log backups

Get insights into the success and failure rates of binlog operations, timeliness of processing and uploads and potential gaps or inconsistencies in binlog data with the Prometheus metrics added for the Operator. Gather this data by connecting to the `<pitr-pod-service>:8080/metrics` endpoint. Learn more about the available metrics in the documentation.

Deprecation, Rename and Removal

The `spec.haproxy.exposePrimary.enabled` field is deprecated. If enabled via the `spec.haproxy.enabled`, the HAProxy primary service is already exposed.

New Features

- K8SPXC-747, K8SPXC-1473 - Add the ability to access the statistics pages for HAProxy and ProxySQL
- K8SPXC-1366 - Add the ability to queue backups and run them sequentially, and to optimize the cluster load with the ability to suspend backups for an unhealthy cluster. A user can assign the start time and suspension time to backups to manage them better.
- K8SPXC-1432 - Enable users to configure cluster-wide Operator deployments in OpenShift certified catalog using OLM.

Improvements

- K8SPXC-1367 - Now a user can configure Readiness and Liveness probes for PMM Client container to check its health and status
- K8SPXC-1461 - Improve logging for resizing PVC with the information about successful and failed PVC resize. Log errors on resize attempts if the Storage Class doesn't support resizing.

- K8SPXC-1466 - Mark the containers that provide the service as default ones with the annotation. This enables a user to connect to a Pod without explicitly specifying a container.
- K8SPXC-1473 - Add the ability to connect to the built-in statistics pages for HAProxy and ProxySQL by exposing the ports for those pages
- K8SPXC-1475 - Update the backup image to use AWS CLI instead of MinIO CLI due to the license change
- K8SPXC-1510 - Add the ability to suppress messages about the use of deprecated features in MySQL Error Log by adding the `log_error_suppression_list` key from the `my.cnf` configuration file and defining the message number in the `spec.pxc.configuration` subsection of the Custom Resource manifest. See how to change MySQL options for steps. This improves readability for MySQL error log.
- K8SPXC-1512 - For Percona XtraDB Cluster version 8.4 and above, binary log user defined functions for point-in-time recovery (`binlog_utils_udf`) are now installed as a component instead of a plugin. This improves their compatibility across platforms and provides automatic dependency handling.
- K8SPXC-1542 - Improve binlog upload for large files to Azure blob storage with the ability to define the block size and the number of concurrent writers for the upload (Thanks to user dcaputo-harmoni for contribution)
- K8SPXC-1543 - Set PITR controller reference for binlog-collector deployment the same way as it's set for PXC and proxy StatefulSets. This creates a connection between PITR deployment and cluster resource (Thank you Vlad Gusev for the contribution)
- K8SPXC-1544 - Improve observability of binlog collector by adding the support of basic Prometheus metrics (Thank you Vlad Gusev for the contribution)
- K8SPXC-1567 - Normalize duplicate slashes if the bucket path for binlog collector ends with a slash (`/`) (Thank you Vlad Gusev for the contribution)
- K8SPXC-1596 - Assign a correct status to a backup if data upload fails due to incomplete backup
- K8SPXC-1620 - Fixed the issue with a failing backup by adding a retry logic to the cloud storage cleanup task to check for uploaded files and clean them up before uploading new files

Bugs Fixed

- K8SPXC-1152 Fixed the issue with the restore process being stuck when the Operator is restarted by setting annotations on the `perconaxtradbclusterrestores` object
- K8SPXC-1482 Fixed the issue with the excessive connection resets on every pod recreation because the cluster's peer-list is not aware of Time To Live (TTL) defined for Kubernetes DNS records. Now there's a 30 second waiting period after a peer update (Thank you Vlad Gusev for reporting this issue and contributing to it)
- K8SPXC-1483 - Fixed the bug where the point-in-time recovery collector process hangs if `mysqlbinlog` cannot connect to the database and start. Now the named pipeline is created with the `O_RDONLY` (Open for Read Only) and `O_NONBLOCK` (Non-Blocking Mode) to unlock the point-in-time recovery collector process. (Thank you Vlad Gusev for reporting this issue and contributing to it)
- K8SPXC-1509 - Fixed the bug where the cluster enters the error state temporarily if point-in-time is enabled for it.
- K8SPXC-1534 - Fixed the issue with the inconsistent secret reconciliation by improving the controller's behavior to timely sync the secret cache and create an internal Secret immediately after its reconciliation.
- K8SPXC-1538 - Fixed the issue with the Operator failing when it tries to reconcile the Custom Resource for the `haproxy-replica` service if the `haproxy-primary` service has the type `LoadBalancer` and the `LoadBalancerSourceRanges` value defined. Now the `haproxy-replica` service inherits this configuration.
- K8SPXC-1546, K8SPXC-1549 - Fixed the issue with the PITR pod crashing on attempt to assign a GTID set to each binlog if the database cluster has a large number of binlogs by caching the `binlog->gtid` set pairs
- K8SPXC-1547 - Removed the outdated example from the `backup.yaml` manifest and update the documentation how to track backup progress
- K8SPXC-1616 - Fixed a bug where the ProxySQL fails to be configured if the password for a `proxysqladmin` user starts with a star (`*`) character by reporting an error and making the Operator regenerate a new password that doesn't start with a star (Thank you Chris Fidao for reporting this issue and contribution)

Supported Software

The Operator was developed and tested with the following software:

- Percona XtraDB Cluster versions 8.4.3-3.1 (Tech preview), 8.0.41-32.1, and 5.7.44-31.65
- Percona XtraBackup versions 8.4.0-1, 8.0.35-32, and 2.4.29
- HAProxy 2.8.14
- ProxySQL 2.7.1-1
- LogCollector based on fluent-bit 4.0.0
- PMM Client 2.44.0

Other options may also work but have not been tested.

Supported Platforms

Percona Operators are designed for compatibility with all CNCF-certified Kubernetes distributions. Our release process includes targeted testing and validation on major cloud provider platforms and OpenShift, as detailed below for Operator version 1.16.0:

- Google Kubernetes Engine (GKE) 1.29 - 1.32
- Amazon Elastic Container Service for Kubernetes (EKS) 1.30 - 1.32
- Azure Kubernetes Service (AKS) 1.30 - 1.32
- OpenShift 4.14 - 4.18
- Minikube 1.35.0 based on Kubernetes 1.32.0

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.

Percona certified images

Find Percona's certified Docker images that you can use with the Percona Operator for MySQL based on Percona XtraDB Cluster in the following table.

Images released with the Operator version 1.17.0:

Image	Digest
percona/percona-xtradb-cluster-operator:1.17.0 (x86_64)	da9aa5c7cb546c60624b927bdd273fc3646bc5a027bcc6f138291bad4da9b7b8
percona/percona-xtradb-cluster-operator:1.17.0 (ARM64)	2b61ed62848521071bea18988461e99123ea5d5a92465ab046d0f179b5c0b8ac
percona/haproxy:2.8.14	6de8c402d83b88dae7403c05183fd75100774defa887c05a57ec04bc25be2305
percona/proxysql2:2.7.1	975d5c8cc7b5714a0df4dfd2111391a7a79cfa3a217f1dd6de77a83550812fc4
percona/percona-xtradb-cluster-operator:1.17.0-pxc8.4-backup-pxb8.4.0	3a7a8a47ad12ce783feb089e7035d50f6d5b803cec97a16067f476a426f6fda8
percona/percona-xtradb-cluster-operator:1.17.0-pxc8.0-backup-pxb8.0.35	2f28c09027a249426b2f4393aa8b76971583d80e0c56be37f77dad49cb5cd5c4
percona/percona-xtradb-cluster-operator:1.17.0-pxc5.7-backup-pxb2.4.29	bf494243d9784a016bb4c98bd2690b0fc5fbba1aa7d45d98502dff353fb68bee
percona/percona-xtradb-cluster-operator:1.17.0-logcollector-fluentbit4.0.0	9fc0b4097c93f6dba8441d9bcb2803dc62dd8328b84288294444fbadb347f6d7
percona/pmm-client:2.44.0	19a07dfa8c12a0554308cd11d7d38494ea02a14cfac6c051ce8ff254b7d0a4a7
percona/percona-xtradb-cluster:8.4.3-3.1	b7b198133e70cb1bd9d5cd1730373a62e976fd2b9bb9ca5a696fd970c1ac09bf
percona/percona-xtradb-cluster:8.0.41-32.1	8a6799cbded5524c6979442f8d7097831c8c6481f5106a856b44b2791ccaf0fb
percona/percona-xtradb-cluster:8.0.39-30.1	6a53a6ad4e7d2c2fb404d274d993414a22cb67beecf7228df9d5d994e7a09966
percona/percona-xtradb-cluster:8.0.36-28.1	b5cc4034ccfb0186d6a734cb749ae17f013b027e9e64746b2c876e8beef379b3
percona/percona-xtradb-cluster:8.0.35-27.1	1ef24953591ef1c1ce39576843d5615d4060fd09458c7a39ebc3e2eda7ef486b
percona/percona-xtradb-cluster:8.0.32-24.2	1f978ab8912e1b5fc66570529cb7e7a4ec6a38adbfce1ece78159b0fcfa7d47a
percona/percona-xtradb-cluster:5.7.44-31.65	36fafdef46485839d4ff7c6dc73b4542b07031644c0152e911acb9734ff2be85
percona/percona-xtradb-cluster:5.7.42-31.65	9dab86780f86ec9caf8e1032a563c131904b75a37edeaec159a93f7d0c16c603
percona/percona-xtradb-cluster:5.7.39-31.61	9013170a71559bbac92ba9c2e986db9bda3a8a9e39ee1ee350e0ee94488bb6d7
percona/percona-xtradb-cluster:5.7.36-31.55	c7bad990fc7ca0fde89240e921052f49da08b67c7c6dc54239593d61710be504
percona/percona-xtradb-cluster:5.7.34-31.51	f8d51d7932b9bb1a5a896c7ae440256230eb69b55798ff37397aabfd58b80ccb

Percona Operator for MySQL based on Percona XtraDB Cluster 1.16.1

- **Date**

December 26, 2024

- **Installation**

Installing Percona Operator for MySQL based on Percona XtraDB Cluster

Bugs Fixed

- K8SPXC-1536: Fix a bug where scheduled backups were not working due to a bug in the Operator that was creating Kubernetes resources with the names exceeding the allowed length (Thanks to Vlad Gusev for contribution)

Supported Platforms

The Operator was developed and tested with Percona XtraDB Cluster versions 8.4.2-2.1 (Tech preview), 8.0.39-30.1, and 5.7.44-31.65. Other options may also work but have not been tested. Other software components include:

- Percona XtraBackup versions 8.4.0-1, 8.0.35-30.1 and 2.4.29
- HAProxy 2.8.11
- ProxySQL 2.7.1
- LogCollector based on fluent-bit 3.2.2
- PMM Client 2.44.0

Percona Operators are designed for compatibility with all CNCF-certified Kubernetes distributions. Our release process includes targeted testing and validation on major cloud provider platforms and OpenShift, as detailed below for Operator version 1.16.1:

- Google Kubernetes Engine (GKE) 1.28 - 1.30
- Amazon Elastic Container Service for Kubernetes (EKS) 1.28 - 1.31
- Azure Kubernetes Service (AKS) 1.28 - 1.31
- OpenShift 4.14.42 - 4.17.8
- Minikube 1.34.0 based on Kubernetes 1.31.0

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.

Percona Operator for MySQL based on Percona XtraDB Cluster 1.16.0

- **Date**

December 19, 2024

- **Installation**

Installing Percona Operator for MySQL based on Percona XtraDB Cluster

Release Highlights

Declarative user management (technical preview)

Before the Operator version 1.16.0 custom MySQL users had to be created manually. Now the declarative creation of custom MySQL users is supported via the `users` subsection in the Custom Resource. You can specify a new user in `deploy/cr.yaml` manifest, setting the user's login name and hosts this user is allowed to connect from, PasswordSecretRef (a reference to a key in a Secret resource containing user's password) and as well as databases the user is going to have access to and the appropriate permissions:

```
...
users:
- name: my-user
  dbs:
  - db1
  - db2
  hosts:
  - localhost
  grants:
  - SELECT
  - DELETE
  - INSERT
  withGrantOption: true
  passwordSecretRef:
    name: my-user-pwd
    key: my-user-pwd-key
...
```

See documentation to find more details about this feature with additional explanations and the list of current limitations.

Percona XtraDB Cluster 8.4 support (technical preview)

Percona XtraDB Cluster based on Percona Server for MySQL 8.4 versions is now supported by the Operator in addition to 8.0 and 5.7 versions. The appropriate images for Percona XtraDB Cluster and Percona XtraBackup are included into the list of Percona-certified images. Being a technical preview, Percona XtraDB Cluster 8.4 is not yet recommended for production environments.

New Features

- K8SPXC-377: It is now possible to create and manage users via the Custom Resource
- K8SPXC-1456: Now the user can run Percona XtraDB Cluster Pods initContainers with a security context different from the Pods security context, useful to customize deployment on tuned Kubernetes environments (Thanks to Vlad Gusev for contribution)

Improvements

- K8SPXC-1230 and K8SPXC-1378: Now the Operator assigns labels to all Kubernetes objects it creates (backups/restores, Secrets, Volumes, etc.) to make them clearly distinguishable
- K8SPXC-1411: Enabling/disabling TLS on a running cluster is now possible simply by toggling the appropriate Custom Resource option
- K8SPXC-1451: The automated storage scaling is now disabled by default and needs to be explicitly enabled with the `enableVolumeExpansion` Custom Resource option
- K8SPXC-1462: A restart of Percona XtraDB Cluster Pods is now triggered by the monitor user's password change if the user secret is used within a sidecar container, which can be useful for custom monitoring solutions (Thanks to Vlad Gusev for contribution)
- K8SPXC-1503: Improved logic saves logs from the appearance of a number of temporary non-critical errors related to ProxySQL user sync and non-presence of point-in-time recovery files (Thanks to dcaputo-harmoni for contribution)

- K8SPXC-1500: A new `backup.activeDeadlineSeconds` Custom Resource option was added to fail the backup job automatically after the specified timeout (Thanks to Vlad Gusev for contribution)
- K8SPXC-1532: The peer-list tool used by the Operator was removed from standard HAProxy, ProxySQL and PXC Docker images because recent Operator versions are adding it with the initContainer approach

Bugs Fixed

- K8SPXC-1222: Fix a bug where upgrading a cluster with hundreds of thousands of tables would fail due to a timeout
- K8SPXC-1398: Fix a bug which sporadically prevented the scheduled backup job Pod from successfully completing the process
- K8SPXC-1413 and K8SPXC-1458: Fix the Operator Pod segfault which was occurring when restoring a backup without backupSource Custom Resource subsection or without storage specified in the backupSource
- K8SPXC-1416: Fix a bug where disabling parallel backups in Custom Resource caused all backups to get stuck in presence of any failed backup
- K8SPXC-1420: Fix a bug where HAProxy exposed at the time of point-in-time restore could make conflicting transactions, causing the PITR Pod stuck on the duplicate key error
- K8SPXC-1422: Fix the cluster endpoint change from the external IP to the service name when upgrading the Operator
- K8SPXC-1444: Fix a bug where Percona XtraDB Cluster initial creation state was changing to “error” if the backup restore was taking too long
- K8SPXC-1454: Fix a bug where the Operator erroneously generated SSL secrets when upgrading from 1.14.0 to 1.15.0 with `allowUnsafeConfigurations: true` Custom Resource option

Deprecation, Rename and Removal

Operator versions older than 1.14.1 become incompatible with new HAProxy, ProxySQL and PXC Docker images due to the absence of the peer-list tool in them. If you are still using the older Operator version, make sure to update the Operator before switching to the latest database and proxy images. You can see the list of Percona certified images for the current release, and check image versions certified for previous releases in the documentation archive.

Known limitations

Being a technical preview, Percona XtraDB Cluster 8.4 doesn’t support the full set of features available within 8.0. Percona XtraDB Cluster 8.4 support has following limitations in this Operator release:

- K8SPXC-1529: Cross-site replication is not yet supported,
- K8SPXC-1512: Point-in-time recovery doesn’t work yet,
- K8SPXC-1511: Encryption is not yet supported,
- K8SPXC-1513: Version service does not support XtraDB Cluster 8.4 yet as well.

Supported Platforms

The Operator was developed and tested with Percona XtraDB Cluster versions 8.4.2-2.1 (Tech preview), 8.0.39-30.1, and 5.7.44-31.65. Other options may also work but have not been tested. Other software components include:

- Percona XtraBackup versions 8.4.0-1, 8.0.35-30.1 and 2.4.29
- HAProxy 2.8.11
- ProxySQL 2.7.1
- LogCollector based on fluent-bit 3.2.2
- PMM Client 2.44.0

Percona Operators are designed for compatibility with all CNCF-certified Kubernetes distributions. Our release process includes targeted testing and validation on major cloud provider platforms and OpenShift, as detailed below for Operator version 1.16.0:

- Google Kubernetes Engine (GKE) 1.28 - 1.30
- Amazon Elastic Container Service for Kubernetes (EKS) 1.28 - 1.31
- Azure Kubernetes Service (AKS) 1.28 - 1.31
- OpenShift 4.14.42 - 4.17.8
- Minikube 1.34.0 based on Kubernetes 1.31.0

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.

Percona Operator for MySQL based on Percona XtraDB Cluster 1.15.1

- **Date**

October 16, 2024

- **Installation**

Installing Percona Operator for MySQL based on Percona XtraDB Cluster

Bugs Fixed

- K8SPXC-1476: Fix a bug where upgrade could put the cluster into a non-operational state if using Storage Classes without the Volume expansion capabilities, by introducing a new `enableVolumeExpansion` Custom Resource option toggling this functionality

Deprecation, Change, Rename and Removal

- The new `enableVolumeExpansion` Custom Resource option allows to disable the automated storage scaling with Volume Expansion capability. The default value of this option is `false`, which means that the automated scaling is turned off by default.

Supported Platforms

The Operator was developed and tested with Percona XtraDB Cluster versions 8.0.36-28.1 and 5.7.44-31.65. Other options may also work but have not been tested. Other software components include:

- Percona XtraBackup versions 8.0.35-30.1 and 2.4.29-1
- HAProxy 2.8.5
- ProxySQL 2.5.5
- LogCollector based on fluent-bit 3.1.4
- PMM Client 2.42.0

The following platforms were tested and are officially supported by the Operator 1.15.1:

- Google Kubernetes Engine (GKE) 1.27 - 1.30
- Amazon Elastic Container Service for Kubernetes (EKS) 1.28 - 1.30
- Azure Kubernetes Service (AKS) 1.28 - 1.30
- OpenShift 4.13.46 - 4.16.7
- Minikube 1.33.1 based on Kubernetes 1.30.0

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.

Percona Operator for MySQL based on Percona XtraDB Cluster 1.14.1

- **Date**

October 16, 2024

- **Installation**

Installing Percona Operator for MySQL based on Percona XtraDB Cluster

Bugs Fixed

- K8SPXC-1476: Fix a bug where upgrade could put the cluster into a non-operational state if using Storage Classes without the Volume expansion capabilities, by introducing a new `enableVolumeExpansion` Custom Resource option toggling this functionality

Deprecation, Change, Rename and Removal

- The new `enableVolumeExpansion` Custom Resource option allows to disable the automated storage scaling with Volume Expansion capability. The default value of this option is `false`, which means that the automated scaling is turned off by default.

Supported Platforms

The Operator was developed and tested with Percona XtraDB Cluster versions 8.0.35-27.1 and 5.7.44-31.65. Other options may also work but have not been tested. Other software components include:

- Percona XtraBackup versions 2.4.29-1 and 8.0.35-30.1
- HAProxy 2.8.5-1
- ProxySQL 2.5.5-1.1
- LogCollector based on fluent-bit 2.1.10-1
- PMM Client 2.41.1

The following platforms were tested and are officially supported by the Operator 1.14.1:

- Google Kubernetes Engine (GKE) 1.25 - 1.29
- Amazon Elastic Container Service for Kubernetes (EKS) 1.24 - 1.29
- Azure Kubernetes Service (AKS) 1.26 - 1.28
- OpenShift 4.12.50 - 4.14.13
- Minikube 1.32.0

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.

Percona Operator for MySQL based on Percona XtraDB Cluster 1.15.0

- **Date**

August 20, 2024

- **Installation**

Installing Percona Operator for MySQL based on Percona XtraDB Cluster

Release Highlights

General availability of the automated volume resizing

The possibility to resize Persistent Volumes by just changing the value of the `resources.requests.storage` option in the PerconaXtraDBCluster custom resource, introduced in the previous release as a technical preview, graduates to general availability.

Allowing `haproxy-replica` Service to cycle through the reader instances only

By default haproxy-replica Service directs connections to all Pods of the database cluster in a round-robin manner. The new `haproxy.exposeReplicas.onlyReaders` Custom Resource option allows to modify this behavior: setting it to `true` excludes current MySQL primary instance (writer) from the list, leaving only the reader instances. By default the option is set to false, which means that `haproxy-replicas` sends traffic to all Pods, including the active writer. The feature can be useful to simplify the application logic by splitting read and write MySQL traffic on the Kubernetes level.

Also, it should be noted that changing `haproxy.exposeReplicas.onlyReaders` value will cause HAProxy Pods to restart.

Fixing the overloaded `allowUnsafeConfigurations` flag

In the previous Operator versions `allowUnsafeConfigurations` Custom Resource option was used to allow configuring a cluster with unsafe parameters, such as starting it with less than 3 Percona XtraDB Cluster instances. In fact, setting this option to `true` resulted in a wide range of reduced safety features without the user's explicit intent: disabling TLS, allowing backups in unhealthy clusters, etc.

With this release, a separate `unsafeFlags` Custom Resource section is introduced for the fine-grained control of the safety loosening features:

```
unsafeFlags:
  tls: false
  pxcSize: false
  proxySize: false
  backupIfUnhealthy: false
```

If the appropriate option is set to `false` and the Operator detects unsafe parameters, it sets cluster status to `error`, and prints an error message in the log.

Also, TLS configuration is now enabled or disabled by setting `unsafeFlags.tls` and `tls.enabled` Custom Resource options to `true` or `false`.

New Features

- K8SPXC-1330: A new `haproxy.exposeReplicas.onlyReaders` Custom Resource option makes `haproxy-replicas` Service to forward requests to reader instances of the MySQL cluster, avoiding the primary (writer) instance.
- K8SPXC-1355: Finalizers were renamed to contain fully qualified domain names (FQDNs), avoiding potential conflicts with other finalizer names in the same Kubernetes environment

Improvements

- K8SPXC-1357: HAProxy Pod no longer restarts when the `operator` user's password changes, which is useful or the applications with persistent connection to MySQL
- K8SPXC-1358: Removing `allowUnsafeConfigurations` Custom Resource option in favor of fine-grained safety control in the `unsafeFlags` subsection
- K8SPXC-1368: Kubernetes PVC DataSources for Percona XtraDB Cluster Volumes are now officially supported via the pxc.volumeSpec.persistentVolumeClaim.dataSource subsection in the Custom Resource
- K8SPXC-1385: Dynamic Volume resize now checks resource quotas and the PVC storage limits
- K8SPXC-1423: The `percona.com/delete-pxc-pvc` finalizer is now able to delete also temporary secrets created by the Operator

Bugs Fixed

- K8SPXC-1067: Fix a bug where changing `gracePeriod`, `nodeSelector`, `priorityClassName`, `runtimeClassName`, and `schedulerName` fields in the `haproxy` Custom Resource subsection didn't propagate changes to the haproxy StatefulSet
- K8SPXC-1338: Fix a bug where binary log collector Pod had unnecessary restart during the Percona XtraDB Cluster rolling restart
- K8SPXC-1364: Fix a bug where log rotation functionality didn't work when the `proxy_protocol_networks` option was enabled in the Percona XtraDB Cluster custom configuration
- K8SPXC-1365: Fix `pxc-operator` Helm chart bug where it wasn't able to create namespaces if multiple namespaces were specified in the watchNamespace option
- K8SPXC-1371: Fix a bug in `pxc-db` Helm chart which had wrong Percona XtraDB Cluster version for the 1.14.0 release and tried to downgrade the database in case of the helm chart upgrade
- K8SPXC-1380: Fix a bug due to which values in the resources requests for the restore job Pod were overwritten by the resources limits ones
- K8SPXC-1381: Fix a bug where HAProxy check script was not correctly identifying all the possible "offline" state of a PXC instance, causing applications connects to an instance NOT able to serve the query
- K8SPXC-1382: Fix a bug where creating a backup on S3 storage failed automatically if `s3.credentialsSecret` Custom Resource option was not present
- K8SPXC-1396: The `xtrabackup` user didn't have rights to grant privileges available in its own privilege level to other users, which caused the point-in-time recovery fail due to access denied
- K8SPXC-1408: Fix a bug where the Operator blocked all restores (including ones without PiTR) in case of a binlog gap detected, instead of only blocking PiTR restores
- K8SPXC-1418: Fix a bug where CA Certificate generated by cert-manager had expiration period of 1 year instead of the 3 years period used by the Operator for other generated certificates, including ones used for internal and external communications

Deprecation, Rename and Removal

- Starting from now, `allowUnsafeConfigurations` Custom Resource option is deprecated in favor of a number of options under the `unsafeFlags` subsection. Also, starting from now the Operator will not set safe defaults automatically. Upgrading existing clusters with `allowUnsafeConfiguration=false` and a configuration considered unsafe (i.e. `pxc.size<3` or `tls.enabled=false`) will print errors in the log and the cluster will have `error` status until the values are fixed.
- Finalizers were renamed to contain fully qualified domain names:
  - `delete-pxc-pods-in-order` renamed to `percona.com/delete-pxc-pods-in-order`
  - `delete-ssl` renamed to `percona.com/delete-ssl`
  - `delete-proxysql-pvc` renamed to `percona.com/delete-proxysql-pvc`
  - `delete-pxc-pvc` renamed to `percona.com/delete-pxc-pvc`
- The `pxc-operator` Helm chart now has `createNamespace` option now is set to `false` by default, resulting in not creating any namespaces unless explicitly allowed to do so by the user

Supported Platforms

The Operator was developed and tested with Percona XtraDB Cluster versions 8.0.36-28.1 and 5.7.44-31.65. Other options may also work but have not been tested. Other software components include:

- Percona XtraBackup versions 8.0.35-30.1 and 2.4.29-1
- HAProxy 2.8.5
- ProxySQL 2.5.5
- LogCollector based on fluent-bit 3.1.4
- PMM Client 2.42.0

The following platforms were tested and are officially supported by the Operator 1.15.0:

- Google Kubernetes Engine (GKE) 1.27 - 1.30
- Amazon Elastic Container Service for Kubernetes (EKS) 1.28 - 1.30
- Azure Kubernetes Service (AKS) 1.28 - 1.30
- OpenShift 4.13.46 - 4.16.7
- Minikube 1.33.1 based on Kubernetes 1.30.0

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.

Percona Operator for MySQL based on Percona XtraDB Cluster 1.14.0

- **Date**

March 4, 2024

- **Installation**

Installing Percona Operator for MySQL based on Percona XtraDB Cluster

Release Highlights

Quickstart guide

Within this release, a Quickstart guide was added to the Operator docs, that'll set you up and running in no time! Taking a look at this guide you'll be guided step by step through quick installing (multiple options), connecting to the database, inserting data, making a backup, and even integrating with Percona Monitoring and Management (PMM) to monitor your cluster.

Automated volume resizing

Kubernetes supports the Persistent Volume expansion as a stable feature since v1.24. Using it with the Operator previously involved manual operations. Now this is automated, and users can resize their PVCs by just changing the value of the `resources.requests.storage` option in the PerconaXtraDBCluster custom resource. This feature is in a technical preview stage and is not recommended for production environments.

Update from November 24, 2025 Though the Operator automates the storage resizing, the users must still trigger the process by modifying the Custom Resource and applying the new configuration.

New Features

- K8SPXC-1298: Custom prefix for Percona Monitoring and Management (PMM) allows using one PMM Server to monitor multiple databases even if they have identical cluster names
- K8SPXC-1334: The new `lifecycle.postStart` and `lifecycle.preStop` Custom Resource options allow configuring postStart and preStop hooks for ProxySQL and HAProxy Pods
- K8SPXC-1341: It is now possible to resize Persistent Volume Claims by patching the PerconaXtraDBCluster custom resource. Change `persistentVolumeClaim.resources.requests.storage` and let the Operator do the scaling

Improvements

- K8SPXC-1313: The `kubectl get pxc-backup` command now shows Latest restorable time to make it easier to pick a point-in-time recovery target
- K8SPXC-1237: The Operator now checks if the needed Secrets exist and connects to the storage to check the existence of a backup before starting the restore process
- K8SPXC-1079: Standardize cluster and components service exposure to have unification of the expose configuration across all Percona Operators
- K8SPXC-1147: Improve log messages by printing the `Last_IO_Error` for a replication channel if it's not empty
- K8SPXC-1151: The `kubectl get pxc-restore` command now shows the "Starting cluster" status to indicate that the point-in-time recovery process is finished
- K8SPXC-1230: Add Labels for all Kubernetes objects created by Operator (backups/restores, Secrets, Volumes, etc.) to make them clearly distinguishable
- K8SPXC-1271: Use timeout to avoid backup stalls in case of the S3 upload network issues
- K8SPXC-1293 and K8SPXC-1294: The new `backup.pitr.timeoutSeconds` Custom Resource option allows setting a timeout for the point-in-time recovery process
- K8SPXC-1301: The Operator can now be run locally against a remote Kubernetes cluster, which simplifies the development process, substantially shortening the way to make and try minor code improvements
- K8SPXC-200 and K8SPXC-1128: The new `containerOptions` subsections were added to `pxc-backup`, `pxc-restore`, and `pxc` Custom Resources to allow setting custom options for xtrabackup, xbstream, and xbcloud tools used by the Operator
- K8SPXC-345: The new `topologySpreadConstraints` Custom Resource option allows to use Pod Topology Spread Constraints to achieve even distribution of Pods across the Kubernetes cluster
- K8SPXC-927: The new `serviceLabel` and `serviceAnnotation` Custom Resource options allow setting Service Labels and Annotations for XtraDB Cluster Pods
- K8SPXC-1340: The new Custom Resource option allows setting custom containerSecurityContext for PMM containers (thanks Marko Weiß for report)

- K8SPXC-1254: Upgrade instructions for Percona XtraDB Cluster in multi-namespace (cluster-wide) mode were added to documentation
- K8SPXC-1276 and K8SPXC-1277: HAProxy log format was changed to JSON with additional information such as timestamps to simplify troubleshooting

Bugs Fixed

- K8SPXC-1264: Liveness probe didn't work if `sql_mode` ANSI_QUOTES enabled
- K8SPXC-1067: Fix a bug that caused the Operator not tracking changes in a number of Custom Resource options in the `haproxy` subsection
- K8SPXC-1105: Fix a bug that didn't allow point-in-time recovery backups on S3-compatible storage with using self-signed certificates
- K8SPXC-1106: Fix a bug which caused point-in-time recovery silently not uploading files if a corrupted binlog file existed in /var/lib/mysql
- K8SPXC-1159: Cluster status was repeatedly switching between "ready" and "error" if the password change did not satisfy the complexity and was rejected by MySQL
- K8SPXC-1256: Fix a bug where the Operator was unable to perform a cleanup by deleting a replication channel if the replication was already stopped
- K8SPXC-1263: Fix a bug where point-in-time recovery was failing if the xtrabackup user password was changed in the binary log files
- K8SPXC-1269: Fix a bug due to which switching from HAProxy to ProxySQL was broken for Percona XtraDB Cluster 5.7
- K8SPXC-1274: PXC init container used by XtraDB Cluster and HAProxy instances inherited XtraDB Cluster resource requirements which was too much for HAProxy (Thanks Tristan for reporting)
- K8SPXC-1275: Fix a bug which caused replication error after switching system accounts to caching_sha2_password authentication plugin which became available in the previous release
- K8SPXC-1288: The Operator didn't treat the name for scheduled backup as a mandatory field
- K8SPXC-1302: Fix a bug where the Operator was continuously trying to delete a backup from an S3 bucket that has a retention policy configured and `delete-s3-backup` finalizer present, which could cause out-of-memory issue in case of tight Pod's memory limits
- K8SPXC-1333: Scheduled backup was failing if Percona XtraDB Cluster name was not unique across namespaces
- K8SPXC-1335: Fix a bug where HAProxy was not stopping existing connections to primary in case of Percona XtraDB Cluster instance failover but only routed new ones to another instance
- K8SPXC-1339: Fix a bug where HAProxy was not aware of the IP address change in case of the restarted Percona XtraDB Cluster Pod and couldn't reach it until the DNS cache update
- K8SPXC-1345: Fix a regression where the Operator was unable to customize readinessProbe of the pxc container
- K8SPXC-1350: Fix a bug due to which log rotate could cause locking TOI (Total Order Isolation) DDL operation on the cluster with flush error logs, resulting in unnecessary synchronization on the whole cluster and possible warnings in logs

Deprecation, Rename and Removal

- K8SPXC-1079: Custom Resource options for service exposure of Percona XtraDB Cluster HAProxy Primary, HAProxy Replicas, and ProxySQL were moved to dedicated `pxc.expose`, `haproxy.exposePrimary`, `haproxy.exposeReplicas`, and `proxysql.expose` subsections. This brings more structure to the Custom Resource and implements the same approach across all Percona Operators. Old variants of service exposure options **are now deprecated** and will be removed in next releases
- K8SPXC-1274: The `initImage` Custom Resource option which allows providing an alternative image with various options for the initial Operator installation, was moved to a dedicated subsection and is now available as `initContainer.image`
- K8SPXC-878: The `clustercheck` system user deprecated in v1.12.0 was completely removed in this release

Supported Platforms

The Operator was developed and tested with Percona XtraDB Cluster versions 8.0.35-27.1 and 5.7.44-31.65. Other options may also work but have not been tested. Other software components include:

- Percona XtraBackup versions 2.4.29-1 and 8.0.35-30.1
- HAProxy 2.8.5-1
- ProxySQL 2.5.5-1.1
- LogCollector based on fluent-bit 2.1.10-1
- PMM Client 2.41.1

The following platforms were tested and are officially supported by the Operator 1.14.0:

- Google Kubernetes Engine (GKE) 1.25 - 1.29

- Amazon Elastic Container Service for Kubernetes (EKS) 1.24 - 1.29
- Azure Kubernetes Service (AKS) 1.26 - 1.28
- OpenShift 4.12.50 - 4.14.13
- Minikube 1.32.0

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.

Percona Operator for MySQL based on Percona XtraDB Cluster 1.13.0

- **Date**

July 11, 2023

- **Installation**

Installing Percona Operator for MySQL based on Percona XtraDB Cluster

Release Highlights

- It is now possible to control whether backup jobs are executed in parallel or sequentially, which can be useful to avoid the cluster overload; also, CPU and memory resource limits can now be configured for the backup restore job
- A substantial improvement of the backup documentation was done in this release, making it much easier to read, and the backup restore options have been added to the Custom Resource reference
- We are deeply committed to delivering software that truly sets the bar for quality and stability. With our latest release, we put an all-hands-on-deck approach towards fine-tuning the Operator with minor improvements, along with addressing key bugs reported by our vibrant community. We are extremely grateful to each and every person who submitted feedback and collaborated to help us get to the bottom of these pesky issues.

New Features and improvements

- K8SPXC-1088: It is now possible to configure CPU and memory resources for the backup restore job in the PerconaXtraDBClusterRestore Custom Resource options
- K8SPXC-1166: Starting from now, Docker image tags for Percona XtraBackup include full XtraBackup version instead of the major number used before
- K8SPXC-1189: Improve security and meet compliance requirements by building the Operator based on Red Hat Universal Base Image (UBI) 9 instead of UBI 8
- K8SPXC-1192: Backup and restore documentation was substantially improved to make it easier to work with, and backup restore options have been added to the Custom Resource reference
- K8SPXC-1210: A headless service can now be configured for ProxySQL and HAProxy to make them usable on a tenant network (thanks to Vishal Anarase for contribution)
- K8SPXC-1225: The Operator (system) users are now created with the `PASSWORD EXPIRE NEVER` policy to avoid breaking the cluster due to the password expiration set by the `default_password_lifetime` system variable
- K8SPXC-362: Code clean-up and refactoring for checking if ProxySQL and HAProxy enabled in the Custom Resource (thanks to Vladislav Safronov for contributing)
- K8SPXC-1224: New `backup.allowParallel` Custom Resource option allows to disable running backup jobs in parallel, which can be useful to avoid connection issues caused by the cluster overload
- K8SPXC-1183: The Operator now uses the caching_sha2_password authentication plugin for MySQL 8.0 instead of the older mysql_native_password one

Bugs Fixed

- K8SPXC-1179 and K8SPXC-1183: Fix a bug due to which the Operator didn't use TLS encryption for system users
- K8SPXC-1188: The database Helm chart has improved defaults, including the use of random passwords generated by the Operator, and disabling `delete-pxc-pvc` and `delete-proxysql-pvc` finalizers to avoid possible data loss during migration
- K8SPXC-1220: Fix a bug due to which DNS resolution problem could force HAProxy to remove all Percona XtraDB Cluster instances, including healthy ones
- K8SPXC-1164: Fix a bug which caused the Operator to recreate Secrets in case of the ProxySQL to HAProxy switch with active `delete-proxysql-pvc` finalizer
- K8SPXC-1255: The log rotation was broken for the audit log, causing it to be written to the old file after the rotation
- K8SPXC-687: Fix a bug which caused the backup restoration not starting in the environment which previously had a cluster with a failed restore
- K8SPXC-835 and K8SPXC-1029: Fix a bug which prevented using ProxySQL on the replica cluster in cross-site replication
- K8SPXC-989: Fix a bug which caused on-demand (manual) backup to fail in IPv6-enabled (dual-stack) environments because of the backup script unable to figure out the proper Pod IPv4 address (thanks to Song Yang for contribution)
- K8SPXC-1106: Fix a bug which caused point-in-time recovery failure in case of a corrupted binlog file in `/var/lib/mysql`
- K8SPXC-1122: Fix a bug which made disabling verification of the storage server TLS certificate with `verifyTLS` PerconaXtraDBClusterRestore Custom Resource option not working
- K8SPXC-1135: Fix a bug where a cluster could incorrectly get a READY status while it had a service with an external IP still in pending state

- K8SPXC-1149: Fix `delete-pxc-pvc` finalizer unable to delete TLS Secret used for external communications in case if this Secret had non-customized default name
- K8SPXC-1161: Fix a bug due to which PMM couldn't continue monitoring HAProxy Pods after the PMM Server API key change
- K8SPXC-1163: Fix a bug that made it impossible to delete the cluster in init state in case of enabled finalizers
- K8SPXC-1199: Fix a bug due to which the Operator couldn't restore backups from Azure blob storage if `spec.backupSource.azure.container` was not specified
- K8SPXC-1205: Fix a bug which made the Operator to ignore the `verifyTLS` option for backups deletion caused by the `delete-s3-backup` finalizer (thanks to Christ-Jan Prinse for reporting)
- K8SPXC-1229 and K8SPXC-1197: Fix a bug due to which the Operator was unable to delete backups from Azure blob storage
- K8SPXC-1236: Fix the pxc container entrypoint script printing passwords into the standard output
- K8SPXC-1242: Fix a bug due to which the unquoted password value was passed to the pmm-admin commands, making PMM Client unable to add MySQL service
- K8SPXC-1243: Fix a bug which prevented deleting PMM agent from the PMM Server inventory on Pod termination
- K8SPXC-1126: Fix a bug that `pxc-db` Helm chart had PVC-based backup storage enabled by default, which could be inconvenient for the users storing backups in cloud
- K8SPXC-1265: Fix a bug due to which get pxc-backup command could show backup as failed after the first unsuccessful attempt while backup job was continuing attempts

Known issues and limitations

- K8SPXC-1183: Switching between HAProxy and ProxySQL load balancer can't be done on existing clusters because ProxySQL does not yet support caching_sha2_password authentication plugin; this makes it necessary to choose load balancer at the cluster creation time

Supported Platforms

The Operator was developed and tested with Percona XtraDB Cluster versions 8.0.32-24.2 and 5.7.42-31.65. Other options may also work but have not been tested. Other software components include:

- Percona XtraBackup versions 2.4.28 and 8.0.32-26
- HAProxy 2.6.12
- ProxySQL 2.5.1-1.1
- LogCollector based on fluent-bit 2.1.5
- PMM Client 2.38

The following platforms were tested and are officially supported by the Operator 1.13.0:

- Google Kubernetes Engine (GKE) 1.24 - 1.27
- Amazon Elastic Container Service for Kubernetes (EKS) 1.23 - 1.27
- Azure Kubernetes Service (AKS) 1.24 - 1.26
- OpenShift 4.10 - 4.13
- Minikube 1.30 (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.

Percona Operator for MySQL based on Percona XtraDB Cluster 1.12.0

- **Date**

December 7, 2022

- **Installation**

Installing Percona Operator for MySQL based on Percona XtraDB Cluster

Release Highlights

- Azure Kubernetes Service (AKS) is now officially supported platform, so developers and vendors of the solutions based on the Azure platform can take advantage of the official support from Percona or just use officially certified Percona Operator for MysQL images; also, Azure Blob Storage can now be used for backups
- This release also includes fixes to the following CVEs (Common Vulnerabilities and Exposures): CVE-2021-20329 (potential injections in MongoDB Go Driver used HAProxy, which had no effect on Percona Operator for MySQL), and CVE-2022-42898 (images used by the Operator suffering from the unauthenticated denial of service vulnerability). Users of previous Operator versions are advised to upgrade to version 1.12.0 which resolves this issue

New Features

- K8SPXC-1043 and K8SPXC-1005: Add support for the Azure Kubernetes Service (AKS) platform and allow using Azure Blob Storage for backups
- K8SPXC-1010: Allow using templates to define `innodb_buffer_pool_size` auto-tuning based on container memory limits
- K8SPXC-1082: New `ignoreAnnotations` and `ignoreLabels` Custom Resource options allow to list specific annotations and labels for Kubernetes Service objects, which the Operator should ignore (useful with various Kubernetes flavors which add annotations to the objects managed by the Operator)
- K8SPXC-1120: Add headless service support for the restore Pod to make it possible restoring backups from a Persistent Volume on a tenant network (thanks to Zulh for contribution)
- K8SPXC-1140: The Operator now allows using SSL channel for cross-site replication (thanks to Alvaro Aguilar-Tablada Espinosa for contribution)

Improvements

- K8SPXC-1104: Starting from now, the Operator changed its API version to v1 instead of having a separate API version for each release. Three last API version are supported in addition to `v1`, which substantially reduces the size of Custom Resource Definition to prevent reaching the etcd limit
- K8SPXC-955: Add Custom Resource options to set static IP-address for the HAProxy and ProxySQL LoadBalancers
- K8SPXC-1032: Disable automated upgrade by default to prevent an unplanned downtime for user applications and to provide defaults more focused on strict user's control over the cluster
- K8SPXC-1095: Process the SIGTERM signal to avoid unneeded lags in case of Percona XtraDB Cluster recovery or using the debug image to start up
- K8SPXC-1113: Utilize dual password feature of MySQL 8 to avoid cluster restart when changing password of the `monitor` user
- K8SPXC-1125: The Operator now does not attempt to start Percona Monitoring and Management (PMM) client sidecar if the corresponding secret does not contain the `pmmserver` or `pmmserverkey` key
- K8SPXC-1153: Configuring the log structuring and leveling is now supported using the `LOG_STRUCTURED` and `LOG_LEVEL` environment variables. This reduces the information overload in logs, still leaving the possibility of getting more details when needed, for example, for debugging
- K8SPXC-1123: Starting from now, installing the Operator for cluster-wide (multi-namespace) doesn't require to add Operator's own namespace to the list of watched namespaces (thanks to Bart Vercoulen for reporting this issue)
- K8SPXC-1030: 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

Bugs Fixed

- K8SPXC-1158: Fix CVE-2022-42898 vulnerability found in MIT krb5, which made images used by the Operator vulnerable to DoS attacks
- K8SPXC-1028: Fix a bug that prevented the Operator to automatically tune `innodb_buffer_pool_size` and `innodb_buffer_pool_chunk_size` variables
- K8SPXC-1036: Fix the bug that caused Liveness Probe failure when XtraBackup was running and the `wsrep_sync_wait` option was set, making the instance to be rejected from the cluster
- K8SPXC-1065: Fix a bug due to which, in a pair of scheduled backups close in time, the next backup could overwrite the previous one: bucket destination was made more unique by including seconds

- K8SPXC-1059: Fix a bug due to which `pxc-monit` and `proxysql-monit` containers were printing passwords in their logs (thanks to zlcnju for contribution)
- K8SPXC-1099: Fix CrashLoopBackOff error caused by incorrect (non-atomic) multi-user password change
- K8SPXC-1100: Fix a bug that made it impossible to use slash characters in the monitor user's password
- K8SPXC-1118: Fix a bug due to which the point-in-time recovery collector only reported warnings in logs when the gaps in binlogs were found. Starting from now, such backups are marked as not suitable for consistent PITR, and restoring them with point-in-time recovery fails without manual user's intervention
- K8SPXC-1137: Fix a bug that prevented adding, deleting or updating ProxySQL Service labels/annotations except at the Service creation time
- K8SPXC-1138: Fix a bug due to which not enough responsive scripts for readiness and liveness Probes could be the reason of killing the overloaded database Pods

Supported Platforms

The following platforms were tested and are officially supported by the Operator 1.12.0:

- Google Kubernetes Engine (GKE) 1.21 - 1.24
- Amazon Elastic Container Service for Kubernetes (EKS) 1.21 - 1.24
- Azure Kubernetes Service (AKS) 1.22 - 1.24
- OpenShift 4.10 - 4.11
- Minikube 1.28

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.

Percona Operator for MySQL based on Percona XtraDB Cluster 1.11.0

- **Date**

  June 3, 2022

- **Installation**

  Installing Percona Operator for MySQL based on Percona XtraDB Cluster

Release Highlights

- With this release, the Operator turns to a simplified naming convention and changes its official name to **Percona Operator for MySQL based on Percona XtraDB Cluster**
- The new backup.backoffLimit Custom Resource option allows customizing the number of attempts the Operator should make for backup
- The OpenAPI schema is now generated for the Operator, which allows Kubernetes to validate Custom Resource and protects users from occasionally applying `deploy/cr.yaml` with syntax errors

New Features

- K8SPXC-936: Allow modifying the init script via Custom Resource, which is useful for troubleshooting the Operator's issues
- K8SPXC-758: Allow to skip TLS verification for backup storage, useful for self-hosted S3-compatible storage with a self-signed certificate

Improvements

- K8SPXC-947: Parametrize the number of attempt the Operator should make for backup backup through a Custom Resource option
- K8SPXC-738: Allow to set service labels for HAProxy and ProxySQL in Custom Resource to enable various integrations with cloud providers or service meshes
- K8SPXC-848: PMM container does not cause the crash of the whole database Pod if pmm-agent is not working properly
- K8SPXC-625: Print the total number of binlogs and the number of remaining binlogs in the restore log while point-in-time recovery in progress
- K8SPXC-920: Using the new Percona XtraBackup Exponential Backoff feature decreases the number of occasional unsuccessful backups due to more effective retries timing (Thanks to Dustin Falgout for reporting this issue)
- K8SPXC-823: Make it possible to use API Key to authorize within Percona Monitoring and Management Server

Bugs Fixed

- K8SPXC-985: Fix a bug that caused point-in-time recovery to fail due to incorrect binlog filtering logic
- K8SPXC-899: Fix a bug due to which issued certificates didn't cover all hostnames, making `VERIFY_IDENTITY` client mode not working with HAProxy
- K8SPXC-750: Fix a bug that prevented ProxySQL from connecting to Percona XtraDB Cluster after turning TLS off
- K8SPXC-896: Fix a bug due to which the Operator was unable to create ssl-internal Secret if crash happened in the middle of a reconcile and restart (Thanks to srteam2020 for contribution)
- K8SPXC-725 and K8SPXC-763: Fix a bug due to which ProxySQL StatefulSet, and Services where mistakenly deleted by the Operator when reading stale ProxySQL or HAProxy information (Thanks to srteam2020 for contribution)
- K8SPXC-957: Fix a bug due to which `pxc-db` Helm chart didn't support setting the `replicasServiceType` Custom Resource option (Thanks to Carlos Martell for reporting this issue)
- K8SPXC-534: Fix a bug that caused some SQL queries to fail during the pxc StatefulSet update (Thanks to Sergiy Prykhodko for reporting this issue)
- K8SPXC-1016: Fix a bug due to which an empty SSL secret name in Custom Resource caused the Operator to throw a misleading error message in the log
- K8SPXC-994: Don't use root user in MySQL Pods to perform checks during cluster restoration, which may be helpful when restoring from non-Kubernetes environments
- K8SPXC-961: Fix a bug due to which a user-defined sidecar container image in the Operator Pod could be treated as the initImage (Thanks to Carlos Martell for reporting this issue)
- K8SPXC-934: Fix a bug due to which the the cluster was not starting as Operator didn't create the users Secret if the `secretsName` option was absent in `cr.yaml`

- K8SPXC-926: Fix a bug due to which failed Smart Update for one cluster in cluster-wide made the Operator unusable for other clusters
- K8SPXC-900: Fix a bug where ProxySQL could not apply new configuration settings
- K8SPXC-862: Fix a bug due to which changing resources as integer values without quotes in Custom Resource could lead to cluster getting stuck
- K8SPXC-858: Fix a bug which could cause a single-node cluster to jump temporarily into the Error status during the upgrade
- K8SPXC-814: Fix a bug when Custom Resource status was missing due to invalid variable setting in the manifest

Deprecation, Rename and Removal

- K8SPXC-823: Password-based authorization to Percona Monitoring and Management Server is now deprecated and will be removed in future releases in favor of a token-based one. Password-based authorization was used by the Operator before this release to provide MySQL monitoring, but now using the API Key is the recommended authorization method

Supported Platforms

The following platforms were tested and are officially supported by the Operator 1.11.0:

- OpenShift 4.7 - 4.10
- Google Kubernetes Engine (GKE) 1.20 - 1.23
- Amazon Elastic Container Service for Kubernetes (EKS) 1.20 - 1.22
- Minikube 1.23

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.

Percona Distribution for MySQL Operator 1.10.0

- **Date**

  November 24, 2021

- **Installation**

  For installation please refer to the documentation page

Release Highlights

- Custom sidecar containers allow users to customize Percona XtraDB Cluster and other Operator components without changing the container images. In this release, we enable even more customization, by allowing users to mount volumes into the sidecar containers.
- In this release, we put a lot of effort into fixing bugs that were reported by the community. We appreciate everyone who helped us with discovering these issues and contributed to the fixes.

New Features

- K8SPXC-856: Mount volumes into sidecar containers to enable customization (Thanks to Sridhar L for contributing)

Improvements

- K8SPXC-771: `spec.Backup.serviceAccount` and `spec.automountServiceAccountToken` Custom Resource options can now be used in the Helm chart (Thanks to Gerwin van de Steeg for reporting this issue)
- K8SPXC-794: The `logrotate` command now doesn't use verbose mode to avoid flooding the log with rotate information
- K8SPXC-793: Logs are now strictly following JSON specification to simplify parsing
- K8SPXC-789: New source_retry_count and source_connect_retry options were added to tune source retries for replication between two clusters
- K8SPXC-588: New replicasServiceEnabled option was added to allow disabling the Kubernetes Service for `haproxy-replicas`, which may be useful to avoid the unwanted forwarding of the application write requests to all Percona XtraDB Cluster instances
- K8SPXC-822: Logrotate now doesn't rotate GRA logs (binlog events in ROW format representing the failed transaction) as ordinary log files, storing them for 7 days instead which gives additional time to debug the problem

Bugs Fixed

- K8SPXC-761: Fixed a bug where HAProxy container was not setting explicit USER id, being incompatible with the runAsNonRoot security policy (Thanks to Henno Schooljan for reporting this issue)
- K8SPXC-894: Fixed a bug where trailing white spaces in the `pmm-admin add` command caused reconcile loop on OpenShift
- K8SPXC-831: Fixed a bug that made it possible to have a split-brain situation, when two nodes were starting their own cluster in case of a DNS failure
- K8SPXC-796: Fixed a bug due to which S3 backup deletion didn't delete Pods attached to the backup job if the S3 finalizer was set (Thanks to Ben Langfeld for reporting this issue)
- K8SPXC-876: Stopped using the `service.alpha.kubernetes.io/tolerate-unready-endpoints` deprecated Kubernetes option in the `${clustername}-pxc-unready` service annotation (Thanks to Antoine Habran for reporting this issue)
- K8SPXC-842: Fixed a bug where backup finalizer didn't delete data from S3 if the backup path contained a folder inside of the S3 bucket (Thanks to 申祥瑞 for reporting this issue)
- K8SPXC-812: Fix a bug due to which the Operator didn't support cert-manager versions since v0.14.0 (Thanks to Ben Langfeld for reporting this issue)
- K8SPXC-762: Fix a bug due to which the validating webhook was not accepting scale operation in the Operator cluster-wide mode (Thanks to Henno Schooljan for reporting this issue)
- K8SPXC-893: Fix a bug where HAProxy service failed during the config validation check if there was a resolution fail with one of the PXC addresses
- K8SPXC-871: Fix a bug that prevented removing a Percona XtraDB Cluster manual backup for PVC storage
- K8SPXC-851: Fixed a bug where changing replication user password didn't work
- K8SPXC-850: Fixed a bug where the default weight value wasn't set for a host in a replication channel
- K8SPXC-845: Fixed a bug where using malformed cr.yaml caused stuck cases in cluster deletion

- K8SPXC-838: Fixed a bug due to which the Log Collector and PMM containers with unspecified memory and CPU requests were inheriting them from the PXC container
- K8SPXC-824: Cluster may get into an unrecoverable state with incomplete full crash
- K8SPXC-818: Fixed a bug which made Pods with a custom config inside a Secret or a ConfigMap not restarting at config update
- K8SPXC-783: Fixed a bug where the root user was able to modify the monitor and clustercheck system users, making the possibility of cluster failure or misbehavior

Supported Platforms

The following platforms were tested and are officially supported by the Operator 1.10.0:

- OpenShift 4.7 - 4.9
- Google Kubernetes Engine (GKE) 1.19 - 1.22
- Amazon Elastic Kubernetes Service (EKS) 1.17 - 1.21
- Minikube 1.22

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.

Percona Distribution for MySQL Operator 1.9.0

- **Date**

  August 9, 2021

- **Installation**

  For installation please refer to the documentation page

Release Highlights

- Starting from this release, the Operator changes its official name to **Percona Distribution for MySQL Operator**. This new name emphasizes gradual changes which incorporated a collection of Percona's solutions to run and operate Percona Server for MySQL and Percona XtraDB Cluster, available separately as Percona Distribution for MySQL.
- Now you can see HAProxy metrics in your favorite Percona Monitoring and Management (PMM) dashboards automatically.
- The cross-site replication feature allows an asynchronous replication between two Percona XtraDB Clusters, including scenarios when one of the clusters is outside of the Kubernetes environment. The feature is intended for the following use cases:
  - provide migrations of your Percona XtraDB Cluster to Kubernetes or vice versa,
  - migrate regular MySQL database to Percona XtraDB Cluster under the Operator control, or carry on backward migration,
  - enable disaster recovery capability for your cluster deployment.

New Features

- K8SPXC-657: Use Secrets to store custom configuration with sensitive data for Percona XtraDB Cluster, HAProxy, and ProxySQL Pods
- K8SPXC-308: Implement Percona XtraDB Cluster asynchronous replication within the Operator
- K8SPXC-688: Define environment variables in the Custom Resource to provide containers with additional customizations

Improvements

- K8SPXC-673: HAProxy Pods now come with Percona Monitoring and Management integration and support
- K8SPXC-791: Allow stopping the restart-on-fail loop for Percona XtraDB Cluster and Log Collector Pods without special debug images
- K8SPXC-764: Unblock backups even if just a single instance is available by setting the `allowUnsafeConfigurations` flag to true
- K8SPXC-765: Automatically delete custom configuration ConfigMaps if the variable in Custom Resource was unset (Thanks to Oleksandr Levchenkov for contributing)
- K8SPXC-734: Simplify manual recovery by automatically getting Percona XtraDB Cluster namespace in the pxc container entrypoint script (Thanks to Michael Lin for contributing)
- K8SPXC-656: imagePullPolicy is now set for init container as well to avoid pulling and simplifying deployments in air-gapped environments (Thanks to Herberto Graça for contributing)
- K8SPXC-511: Secret object containing system users passwords is now deleted along with the Cluster if `delete-pxc-pvc` finalizer is enabled (Thanks to Matthias Baur for contributing)
- K8SPXC-772: All Service objects now have Percona XtraDB Cluster labels attached to them to enable label selector usage
- K8SPXC-731: It is now possible to see the overall progress of the provisioning of Percona XtraDB Cluster resources and dependent components in Custom Resource status
- K8SPXC-730: Percona XtraDB Cluster resource statuses in Custom Resource output (e.g. returned by `kubectl get pxc` command) have been improved and now provide more precise reporting
- K8SPXC-697: Add namespace support in the `copy-backup` script
- K8SPXC-321, K8SPXC-556, K8SPXC-568: Restrict the minimal number of ProxySQL and HAProxy Pods and the maximal number of Percona XtraDB Cluster Pods if the unsafe flag is not set
- K8SPXC-554: Reduced the number of various etcd and k8s object updates from the Operator to minimize the pressure on the Kubernetes cluster
- K8SPXC-421: It is now possible to use X Plugin with Percona XtraDB Cluster Pods

Known Issues and Limitations

- K8SPXC-835: ProxySQL will fail to start on a Replica Percona XtraDB Cluster for cross-site replication in this release

Bugs Fixed

- K8SPXC-757: Fixed a bug where manual crash recovery interfered with auto recovery functionality even with the `auto_recovery` flag set to false
- K8SPXC-706: TLS certificates renewal by a cert-manager was failing (Thanks to Jeff Andrews for reporting this issue)
- K8SPXC-785: Fixed a bug where backup to S3 was producing false-positive error messages even if backup was successful
- K8SPXC-642: Fixed a bug where PodDisruptionBudget was blocking the upgrade of HAProxy (Thanks to Davi S Evangelista for reporting this issue)
- K8SPXC-585: Fixed a bug where the Operator got stuck if the wrong user credentials were set in the Secret object (Thanks to Sergiy Prykhodko for reporting this issue)
- K8SPXC-756: Fixed a bug where the Operator was scheduling backups even when the cluster was paused (Thanks to Dmytro for reporting this issue)
- K8SPXC-813: Fixed a bug where backup restore didn't return error on incorrect AWS credentials
- K8SPXC-805: Fixed a bug that made pxc-backups object deletion hang if the Operator couldn't list objects from the S3 bucket (e.g. due to wrong S3 credentials)
- K8SPXC-787: Fixed the "initializing" status of ready clusters caused by the xtrabackup user password change
- K8SPXC-775: Fixed a bug where errors in custom mysqld config settings were not detected by the Operator if the config was modified after the initial cluster was created
- K8SPXC-767: Fixed a bug where on-demand backup hung up if created while the cluster was in the "initializing" state
- K8SPXC-726: Fixed a bug where the `delete-s3-backup` finalizer prevented deleting a backup stored on Persistent Volume
- K8SPXC-682: Fixed auto-tuning feature setting wrong `innodb_buffer_pool_size` value in some cases

Percona Kubernetes Operator for Percona XtraDB Cluster 1.8.0

- **Date**

  April 26, 2021

- **Installation**

  Installing Percona Kubernetes Operator for Percona XtraDB Cluster

Release Highlights

- It is now possible to use `kubectl scale` command to scale Percona XtraDB Cluster horizontally (add or remove Replica Set instances). You can also use Horizontal Pod Autoscaler which will scale your database cluster based on various metrics, such as CPU utilization.
- Support for custom sidecar containers. The Operator makes it possible now to deploy additional (sidecar) containers to the Pod. This feature can be useful to run debugging tools or some specific monitoring solutions, etc. Sidecar containers can be added to pxc, haproxy, and proxysql sections of the `deploy/cr.yaml` configuration file.

New Features

- K8SPXC-528: Support for custom sidecar containers to extend the Operator capabilities
- K8SPXC-647: Allow the cluster scale in and scale out with the `kubectl scale` command or Horizontal Pod Autoscaler
- K8SPXC-643: Operator can now automatically recover Percona XtraDB Cluster after the network partitioning

Improvements

- K8SPXC-442: The Operator can now automatically remove old backups from S3 storage if the retention period is set (thanks to Davi S Evangelista for reporting this issue)
- K8SPXC-697: Add namespace support in the script used to copy backups from remote storage to a local machine
- K8SPXC-627: Point-in-time recovery uploader now chooses the Pod with the oldest binary log in the cluster to ensure log consistency
- K8SPXC-618: Add debug symbols from the percona-xtradb-cluster-server-debuginfo package to the Percona XtraDB Cluster debug docker image to simplify troubleshooting
- K8SPXC-599: It is now possible to recover databases up to a specific transaction with the Point-in-time Recovery feature. Previously the user could only recover to specific date and time
- K8SPXC-598: Point-in-time recovery feature now works with compressed backups
- K8SPXC-536: It is now possible to explicitly set the version of Percona XtraDB Cluster for newly provisioned clusters. Before that, all new clusters were started with the latest PXC version if Version Service was enabled
- K8SPXC-522: Add support for the `runtimeClassName` Kubernetes feature for selecting the container runtime
- K8SPXC-519, K8SPXC-558, and K8SPXC-637: Various improvements of Operator log messages

Known Issues and Limitations

- K8SPXC-701: Scheduled backups are not compatible with Kubernetes 1.20 in cluster-wide mode.

Bugs Fixed

- K8SPXC-654: Use MySQL administrative port for Kubernetes liveness/readiness probes to avoid false positive failures
- K8SPXC-614, K8SPXC-619, K8SPXC-545, K8SPXC-641, K8SPXC-576: Fix multiple bugs due to which changes of various objects in `deploy/cr.yaml` were not applied to the running cluster (thanks to Sergiy Prykhodko for reporting some of these issues)
- K8SPXC-596: Fix a bug due to which liveness probe for `pxc` container could cause zombie processes
- K8SPXC-632: Fix a bug preventing point-in-time recovery when multiple clusters were uploading binary logs to a single S3 bucket
- K8SPXC-573: Fix a bug that prevented using special characters in XtraBackup password (thanks to Gertjan Bijl for reporting this issue)
- K8SPXC-571: Fix a bug where Percona XtraDB Cluster went into a desynced state at backup job crash (Thanks to Dimitrij Hilt for reporting this issue)

- K8SPXC-430: Galera Arbitrator used for backups does not break the cluster anymore in various cases
- K8SPXC-684: Fix a bug due to which point-in-time recovery backup didn't allow specifying the `endpointUrl` for Amazon S3 storage
- K8SPXC-681: Fix operator crash which occurred when non-existing storage name was specified for point-in-time recovery
- K8SPXC-638: Fix unneeded delay in showing logs with the `kubectl logs` command for the logs container
- K8SPXC-609: Fix frequent HAProxy service NodePort updates which were causing issues with load balancers
- K8SPXC-542: Fix a bug due to which backups were taken only for one cluster out of many controlled by one Operator
- CLOUD-611: Stop using the already deprecated runtime/scheme package (Thanks to Jerome Küttner for reporting this issue)

Percona Kubernetes Operator for Percona XtraDB Cluster 1.7.0

- **Date**

  February 2, 2021

- **Installation**

  Installing Percona Kubernetes Operator for Percona XtraDB Cluster

New Features

- K8SPXC-530: Add support for point-in-time recovery
- K8SPXC-564: PXC cluster will now recover automatically from a full crash when Pods are stuck in CrashLoopBackOff status
- K8SPXC-497: Official support for Percona Monitoring and Management (PMM) v.2

NOTE: Monitoring with PMM v.1 configured according to the unofficial instruction will not work after the upgrade. Please switch to PMM v.2.

Improvements

- K8SPXC-485: Percona XtraDB Cluster Pod logs are now stored on Persistent Volumes. Users can debug the issues even after the Pod restart
- K8SPXC-389: User can now change ServiceType for HAProxy replicas Kubernetes service
- K8SPXC-546: Reduce the number of ConfigMap object updates from the Operator to improve performance of the Kubernetes cluster
- K8SPXC-553: Change default configuration of ProxySQL to WRITERS_ARE_READERS=yes so Percona XtraDB Cluster continues operating with a single node left
- K8SPXC-512: User can now limit cluster-wide Operator access to specific namespaces (Thanks to user mgar for contribution)
- K8SPXC-490: Improve error message when not enough memory is set for auto-tuning
- K8SPXC-312: Add schema validation for Custom Resource. Now `cr.yaml` is validated by a WebHook for syntax typos before being applied. It works only in cluster-wide mode due to access restrictions
- K8SPXC-510: Percona XtraDB Cluster operator can now be deployed through RedHat Marketplace
- K8SPXC-543: Check HAProxy custom configuration for syntax errors before applying it to avoid Pod getting stuck in CrashLoopBackOff status (Thanks to user pservit for reporting this issue)

Bugs Fixed

- K8SPXC-544: Add a liveness probe for HAProxy so it is not stuck and automatically restarted when crashed (Thanks to user pservit for reporting this issue)
- K8SPXC-500: Fix a bug that prevented creating a backup in cluster-wide mode if default cr.yaml is used (Thanks to user michael.lin1 for reporting this issue)
- K8SPXC-491: Fix a bug due to which compressed backups didn't work with the Operator (Thanks to user dejw for reporting this issue)
- K8SPXC-570: Fix a bug causing backups to fail with some S3-compatible storages (Thanks to user dimitrij for reporting this issue)
- K8SPXC-517: Fix a bug causing Operator crash if Custom Resource backup section is missing (Thanks to user deamonmv for reporting this issue)
- K8SPXC-253: Fix a bug preventing rolling out Custom Resource changes (Thanks to user bitsbeats for reporting this issue)
- K8SPXC-552: Fix a bug when HAProxy secrets cannot be updated by the user
- K8SPXC-551: Fix a bug due to which cluster was not initialized when the password had an end of line symbol in `secret.yaml`
- K8SPXC-526: Fix a bug due to which not all clusters managed by the Operator were upgraded by the automatic update
- K8SPXC-523: Fix a bug putting cluster into unhealthy status after the clustercheck secret changed
- K8SPXC-521: Fix automatic upgrade job repeatedly looking for an already removed cluster
- K8SPXC-520: Fix Smart update in cluster-wide mode adding version service check job repeatedly instead of doing it only once
- K8SPXC-463: Fix a bug due to which wsrep_recovery log was unavailable after the Pod restart
- K8SPXC-424: Fix a bug due to which HAProxy health-check spammed in logs, making them hardly unreadable
- K8SPXC-379: Fix a bug due to which the Operator user credentials were not added into internal secrets when upgrading from 1.4.0 (Thanks to user pservit for reporting this issue)

Percona Kubernetes Operator for Percona XtraDB Cluster 1.6.0

- **Date**

  October 9, 2020

- **Installation**

  Installing Percona Kubernetes Operator for Percona XtraDB Cluster

New Features

- K8SPXC-394: Support of "cluster-wide" mode for Percona XtraDB Cluster Operator
- K8SPXC-416: Support of the proxy-protocol in HAProxy (to use this feature, you should have a Percona XtraDB Cluster image version `8.0.21` or newer)
- K8SPXC-429: A possibility to restore backups to a new Kubernetes-based environment with no existing Percona XtraDB Cluster Custom Resource
- K8SPXC-343: Helm chart officially provided with the Operator

Improvements

- K8SPXC-144: Allow adding ProxySQL configuration options
- K8SPXC-398: New `crVersion key` in `deploy/cr.yaml` to indicate the API version that the Custom Resource corresponds to (thanks to user mike.saah for contribution)
- K8SPXC-474: The init container now has the same resource requests as the main container of a correspondent Pod (thanks to user yann.leenhardt for contribution)
- K8SPXC-372: Support new versions of cert-manager by the Operator (thanks to user rf_enigm for contribution)
- K8SPXC-317: Possibility to configure the `imagePullPolicy` Operator option (thanks to user imranrazakhan for contribution)
- K8SPXC-462: Add readiness probe for HAProxy
- K8SPXC-411: Extend cert-manager configuration to add additional domains (multiple SAN) to a certificate
- K8SPXC-375: Improve HAProxy behavior in case of switching writer node to a new one and back
- K8SPXC-368: Autoupdate system users by changing the appropriate Secret name

Known Issues and Limitations

- OpenShift 3.11 requires additional configuration for the correct HAProxy operation: the feature gate `PodShareProcessNamespace` should be set to `true`. If getting it enabled is not possible, we recommend using ProxySQL instead of HAProxy with OpenShift 3.11. Other OpenShift and Kubernetes versions are not affected.
- K8SPXC-491: Compressed backups are not compatible with the Operator 1.6.0 (`percona/percona-xtradb-cluster-operator:1.5.0-pxc8.0-backup` or `percona/percona-xtradb-cluster-operator:1.5.0-pxc5.7-backup` image can be used as a workaround if needed).

Bugs Fixed

- K8SPXC-431: HAProxy unable to start on OpenShift with the default `cr.yaml` file
- K8SPXC-408: Insufficient MAX_USER_CONNECTIONS=10 for ProxySQL monitor user (increased to 100)
- K8SPXC-391: HAProxy and PMM cannot be enabled at the same time (thanks to user rf_enigm for reporting this issue)
- K8SPXC-406: Second node (XXX-pxc-1) always selected as a donor (thanks to user pservit for reporting this issue)
- K8SPXC-390: Crash on missing HAProxy PodDisruptionBudget
- K8SPXC-355: Counterintuitive YYYY-DD-MM dates in the S3 backup folder names (thanks to user graham-web for contribution)
- K8SPXC-305: ProxySQL not working in case of passwords with a `%` symbol in the Secrets object (thanks to user ben.wilson for reporting this issue)
- K8SPXC-278: ProxySQL never getting ready status in some environments after the cluster launch due to the `proxysql-monit` Pod crash (thanks to user lots0logs for contribution)
- K8SPXC-274: The 1.2.0 -> 1.3.0 -> 1.4.0 upgrade path not working (thanks to user martin.atroo for reporting this issue)

- K8SPXC-476: SmartUpdate failing to fetch version from Version Service in case of incorrectly formatted Percona XtraDB Cluster patch version higher than the last known one
- K8SPXC-454: After the cluster creation, pxc-0 Pod restarting due to Operator not waiting for cert-manager to issue requested certificates (thanks to user mike.saah for reporting this issue)
- K8SPXC-450: TLS annotations causing unnecessary HAProxy Pod restarts
- K8SPXC-443 and K8SPXC-456: The outdated version service endpoint URL (fix with preserving backward compatibility)
- K8SPXC-435: MySQL root password visible through `kubectl logs`
- K8SPXC-426: mysqld recovery logs not logged to file and not available through `kubectl logs`
- K8SPXC-423: HAProxy not refreshing IP addresses even when the node gets a different address
- K8SPXC-419: Percona XtraDB Cluster incremental state transfers not taken into account by readiness/liveness checks
- K8SPXC-418: HAProxy not routing traffic for 1 donor, 2 joiners
- K8SPXC-417: Cert-manager not compatible with Kubernetes versions below v1.15 due to unnecessarily high API version demand
- K8SPXC-384: Debug images were not fully functional for the latest version of the Operator because of having no infinity loop
- K8SPXC-383: DNS warnings in PXC Pods when using HAProxy
- K8SPXC-364: Smart Updates showing empty “from” versions for non-PXC objects in logs
- K8SPXC-379: The Operator user credentials not added into internal secrets when upgrading from 1.4.0 (thanks to user pservit for reporting this issue)

Percona Kubernetes Operator for Percona XtraDB Cluster 1.5.0

- **Date**

  July 21, 2020

- **Installation**

  Installing Percona Kubernetes Operator for Percona XtraDB Cluster

New Features

- K8SPXC-298: Automatic synchronization of MySQL users with ProxySQL
- K8SPXC-294: HAProxy Support
- K8SPXC-284: Fully automated minor version updates (Smart Update)
- K8SPXC-257: Update Reader members before Writer member at cluster upgrades
- K8SPXC-256: Support multiple PXC minor versions by the Operator

Improvements

- K8SPXC-290: Extend usable backup schedule syntax to include lists of values
- K8SPXC-309: Quickstart Guide on Google Kubernetes Engine (GKE) - link
- K8SPXC-288: Quickstart Guide on Amazon Elastic Kubernetes Service (EKS) - link
- K8SPXC-280: Support XtraBackup compression
- K8SPXC-279: Use SYSTEM_USER privilege for system users on PXC 8.0
- K8SPXC-277: Install GDB in PXC images
- K8SPXC-276: Pod-0 should be selected as Writer if possible
- K8SPXC-252: Automatically manage system users for MySQL and ProxySQL on password rotation via Secret
- K8SPXC-242: Improve internal backup implementation for better stability with PXC 8.0
- CLOUD-404: Support of loadBalancerSourceRanges for LoadBalancer Services
- CLOUD-556: Kubernetes 1.17 added to the list of supported platforms

Bugs Fixed

- K8SPXC-327: CrashloopBackOff if PXC 8.0 Pod restarts in the middle of SST
- K8SPXC-291: PXC Restore failure with “The node was low on resource: ephemeral-storage” error (Thanks to user rjeka for reporting this issue)
- K8SPXC-270: Restore job wiping data from the original backup’s cluster when restoring to another cluster in the same namespace
- K8SPXC-352: Backup cronjob not scheduled in some Kubernetes environments (Thanks to user msavchenko for reporting this issue)
- K8SPXC-275: Outdated documentation on the Operator updates (Thanks to user martin.atroo for reporting this issue)
- K8SPXC-347: XtraBackup failure after uploading a backup, causing the backup process restart in some cases (Thanks to user connde for reporting this issue)
- K8SPXC-373: Pod not cleaning up the SST tmp dir on start
- K8SPXC-326: Changes in TLS Secrets not triggering PXC restart if AllowUnsafeConfig enabled
- K8SPXC-323: Missing `tar` utility in the PXC node docker image
- CLOUD-531: Wrong usage of `strings.TrimLeft` when processing apiVersion
- CLOUD-474: Cluster creation not failing if wrong resources are set

Percona Kubernetes Operator for Percona XtraDB Cluster 1.4.0

- **Date**

  April 29, 2020

- **Installation**

  Installing Percona Kubernetes Operator for Percona XtraDB Cluster

New Features

- K8SPXC-172: Full data-at-rest encryption available in PXC 8.0 is now supported by the Operator. This feature is implemented with the help of the `keyring_vault` plugin which ships with PXC 8.0. By utilizing Vault we enable our customers to follow best practices with encryption in their environment.
- K8SPXC-125: Percona XtraDB Cluster 8.0 is now supported
- K8SPXC-95: Amazon Elastic Container Service for Kubernetes (EKS) was added to the list of the officially supported platforms
- The OpenShift Container Platform 4.3 is now supported

Improvements

- K8SPXC-262: The Operator allows setting ephemeral-storage requests and limits on all Pods
- K8SPXC-221: The Operator now updates observedGeneration status message to allow better monitoring of the cluster rollout or backup/restore process
- K8SPXC-213: A special PXC debug image is now available. It avoids restarting on fail and contains additional tools useful for debugging
- K8SPXC-100: The Operator now implements the crash tolerance on the one member crash. The implementation is based on starting Pods with `mysqld --wsrep_recover` command if there was no graceful shutdown

Bugs Fixed

- K8SPXC-153: S3 protocol credentials were not masked in logs during the PXC backup & restore process
- K8SPXC-222: The Operator got caught in reconciliation error in case of the erroneous/absent API version in the `deploy/cr.yaml` file
- K8SPXC-261: ProxySQL logs were showing the root password
- K8SPXC-220: The inability to update or delete existing CRD was possible because of too large records in etcd, resulting in “request is too large” errors. Only 20 last status changes are now stored in etcd to avoid this problem.
- K8SPXC-52: The Operator produced an unclear error message in case of fail caused by the absent or malformed pxc section in the `deploy/cr.yaml` file
- K8SPXC-269: The `copy-backup.sh` script didn’s work correctly in case of an existing secret with the AWS_ACCESS_KEY_ID/AWS_SECRET_ACCESS_KEY credentials and prevented users from copying backups (e.g. to a local machine)
- K8SPXC-263: The `kubectl get pxc` command was unable to show the correct ProxySQL external endpoint
- K8SPXC-219: PXC Helm charts were incompatible with the version 3 of the Helm package manager
- K8SPXC-40: The cluster was unable to reach “ready” status in case if `ProxySQL.Enabled` field was set to `false`
- K8SPXC-34: Change of the `proxysql.servicetype` filed was not detected by the Operator and thus had no effect

Percona Kubernetes Operator for Percona XtraDB Cluster 1.3.0

Percona announces the *Percona Kubernetes Operator for Percona XtraDB Cluster* 1.3.0 release on January 6, 2020. This release is now the current GA release in the 1.3 series. Install the Kubernetes Operator for Percona XtraDB Cluster by following the instructions.

The Percona Kubernetes Operator for Percona XtraDB Cluster automates the lifecycle and provides a consistent Percona XtraDB Cluster instance. The Operator can be used to create a Percona XtraDB Cluster, or scale an existing Cluster and contains the necessary Kubernetes settings.

The Operator simplifies the deployment and management of the Percona XtraDB Cluster in Kubernetes-based environments. It extends the Kubernetes API with a new custom resource for deploying, configuring and managing the application through the whole life cycle.

The Operator source code is available in our Github repository. All of Percona's software is open-source and free.

New features and improvements:

- CLOUD-412: Auto-Tuning of the MySQL Parameters based on Pod memory resources was implemented in the case of Percona XtraDB Cluster Pod limits (or at least Pod requests) specified in the cr.yaml file.
- CLOUD-411: Now the user can adjust securityContext, replacing the automatically generated securityContext with the customized one.
- CLOUD-394: The Percona XtraDB Cluster, ProxySQL, and backup images size decrease by 40-60% was achieved by removing unnecessary dependencies and modules to reduce the cluster deployment time.
- CLOUD-390: Helm chart for Percona Monitoring and Management (PMM) 2.0 has been provided.
- CLOUD-383: Affinity constraints and tolerations were added to the backup Pod
- CLOUD-430: Image URL in the CronJob Pod template is automatically updated when the Operator detects changed backup image URL

Fixed bugs:

- CLOUD-462: Resource requests/limits were set not for all containers in a ProxySQL Pod
- CLOUD-437: Percona Monitoring and Management Client was taking resources definition from the Percona XtraDB Cluster despite having much lower need in resources, particularly lower memory footprint.
- CLOUD-434: Restoring Percona XtraDB Cluster was failing on the OpenShift platform with customized security settings
- CLOUD-399: The iputils package was added to the backup docker image to provide backup jobs with the ping command for a better network connection handling
- CLOUD-393: The Operator generated various StatefulSets in the first reconciliation cycle and in all subsequent reconciliation cycles, causing Kubernetes to trigger an unnecessary ProxySQL restart once during the cluster creation.
- CLOUD-376: A long-running SST caused the liveness probe check to fail it's grace period timeout, resulting in an unrecoverable failure
- CLOUD-243: Using MYSQL_ROOT_PASSWORD with special characters in a ProxySQL docker image was breaking the entrypoint initialization process

Percona XtraDB Cluster is an open source, cost-effective and robust clustering solution for businesses. It integrates Percona Server for MySQL with the Galera replication library to produce a highly-available and scalable MySQL® cluster complete with synchronous multi-primary replication, zero data loss and automatic node provisioning using Percona XtraBackup.

Help us improve our software quality by reporting any bugs you encounter using our bug tracking system.

Percona Kubernetes Operator for Percona XtraDB Cluster 1.2.0

Percona announces the *Percona Kubernetes Operator for Percona XtraDB Cluster* 1.2.0 release on September 20, 2019. This release is now the current GA release in the 1.2 series. Install the Kubernetes Operator for Percona XtraDB Cluster by following the instructions.

The Percona Kubernetes Operator for Percona XtraDB Cluster automates the lifecycle and provides a consistent Percona XtraDB Cluster instance. The Operator can be used to create a Percona XtraDB Cluster, or scale an existing Cluster and contains the necessary Kubernetes settings.

The Operator simplifies the deployment and management of the Percona XtraDB Cluster in Kubernetes-based environments. It extends the Kubernetes API with a new custom resource for deploying, configuring and managing the application through the whole life cycle.

The Operator source code is available in our Github repository. All of Percona's software is open-source and free.

New features and improvements:

- A Service Broker was implemented for the Operator, allowing a user to deploy Percona XtraDB Cluster on the OpenShift Platform, configuring it with a standard GUI, following the Open Service Broker API.
- Now the Operator supports Percona Monitoring and Management 2, which means being able to detect and register to PMM Server of both 1.x and 2.0 versions.
- A `NodeSelector` constraint is now supported for the backups, which allows using backup storage accessible to a limited set of nodes only (contributed by Chen Min).
- The resource constraint values were refined for all containers to eliminate the possibility of an out of memory error.
- Now it is possible to set the `schedulerName` option in the operator parameters. This allows using storage which depends on a custom scheduler, or a cloud provider which optimizes scheduling to run workloads in a cost-effective way (contributed by Smaine Kahlouch).
- A bug was fixed, which made cluster status oscillate between "initializing" and "ready" after an update.
- A 90 second startup delay which took place on freshly deployed Percona XtraDB Cluster was eliminated.

Percona XtraDB Cluster is an open source, cost-effective and robust clustering solution for businesses. It integrates Percona Server for MySQL with the Galera replication library to produce a highly-available and scalable MySQL® cluster complete with synchronous multi-primary replication, zero data loss and automatic node provisioning using Percona XtraBackup.

Help us improve our software quality by reporting any bugs you encounter using our bug tracking system.

Percona Kubernetes Operator for Percona XtraDB Cluster 1.1.0

Percona announces the general availability of *Percona Kubernetes Operator for Percona XtraDB Cluster* 1.1.0 on July 15, 2019. This release is now the current GA release in the 1.1 series. Install the Kubernetes Operator for Percona XtraDB Cluster by following the instructions.

The Percona Kubernetes Operator for Percona XtraDB Cluster automates the lifecycle and provides a consistent Percona XtraDB Cluster instance. The Operator can be used to create a Percona XtraDB Cluster, or scale an existing Cluster and contains the necessary Kubernetes settings.

The Operator simplifies the deployment and management of the Percona XtraDB Cluster in Kubernetes-based environments. It extends the Kubernetes API with a new custom resource for deploying, configuring and managing the application through the whole life cycle.

The Operator source code is available in our Github repository. All of Percona's software is open-source and free.

New features and improvements:

- Now the Percona Kubernetes Operator allows upgrading Percona XtraDB Cluster to newer versions, either in semi-automatic or in manual mode.
- Also, two modes are implemented for updating the Percona XtraDB Cluster `my.cnf` configuration file: in *automatic configuration update* mode Percona XtraDB Cluster Pods are immediately re-created to populate changed options from the Operator YAML file, while in *manual mode* changes are held until Percona XtraDB Cluster Pods are re-created manually.
- A separate service account is now used by the Operator's containers which need special privileges, and all other Pods run on default service account with limited permissions.
- User secrets are now generated automatically if don't exist: this feature especially helps reduce work in repeated development environment testing and reduces the chance of accidentally pushing predefined development passwords to production environments.
- The Operator is now able to generate TLS certificates itself which removes the need in manual certificate generation.
- The list of officially supported platforms now includes Minikube, which provides an easy way to test the Operator locally on your own machine before deploying it on a cloud.
- Also, Google Kubernetes Engine 1.14 and OpenShift Platform 4.1 are now supported.

Percona XtraDB Cluster is an open source, cost-effective and robust clustering solution for businesses. It integrates Percona Server for MySQL with the Galera replication library to produce a highly-available and scalable MySQL® cluster complete with synchronous multi-primary replication, zero data loss and automatic node provisioning using Percona XtraBackup.

Help us improve our software quality by reporting any bugs you encounter using our bug tracking system.

Percona Kubernetes Operator for Percona XtraDB Cluster 1.0.0

Percona announces the general availability of *Percona Kubernetes Operator for Percona XtraDB Cluster* 1.0.0 on May 29, 2019. This release is now the current GA release in the 1.0 series. Install the Kubernetes Operator for Percona XtraDB Cluster by following the instructions. Please see the GA release announcement. All of Percona's software is open-source and free.

The Percona Kubernetes Operator for Percona XtraDB Cluster automates the lifecycle and provides a consistent Percona XtraDB Cluster instance. The Operator can be used to create a Percona XtraDB Cluster, or scale an existing Cluster and contains the necessary Kubernetes settings.

The Percona Kubernetes Operators are based on best practices for configuration and setup of the Percona XtraDB Cluster. The Operator provides a consistent way to package, deploy, manage, and perform a backup and a restore for a Kubernetes application. Operators deliver automation advantages in cloud-native applications.

The advantages are the following:

- Deploy a Percona XtraDB Cluster environment with no single point of failure and environment can span multiple availability zones (AZs).
- Deployment takes about six minutes with the default configuration.
- Modify the Percona XtraDB Cluster size parameter to add or remove Percona XtraDB Cluster members
- Integrate with Percona Monitoring and Management (PMM) to seamlessly monitor your Percona XtraDB Cluster
- Automate backups or perform on-demand backups as needed with support for performing an automatic restore
- Supports using Cloud storage with S3-compatible APIs for backups
- Automate the recovery from failure of a single Percona XtraDB Cluster node
- TLS is enabled by default for replication and client traffic using Cert-Manager
- Access private registries to enhance security
- Supports advanced Kubernetes features such as pod disruption budgets, node selector, constraints, tolerations, priority classes, and affinity/anti-affinity
- You can use either PersistentVolumeClaims or local storage with hostPath to store your database
- Customize your MySQL configuration using ConfigMap.

Installation

Installation is performed by following the documentation installation instructions for Kubernetes and OpenShift.