# Percona Kubernetes Operator for Percona XtraDB Cluster

***Release 1.4.0***



**Apr 29, 2020**

# CONTENTS

Kubernetes and the OpenShift platform, based on Kubernetes, have added a way to manage containerized systems, including database clusters. This management is achieved by controllers, declared in configuration files. These controllers provide automation with the ability to create objects, such as a container or a group of containers called pods, to listen for an specific event and then perform a task.

This automation adds a level of complexity to the container-based architecture and stateful applications, such as a database. A Kubernetes Operator is a special type of controller introduced to simplify complex deployments. The Operator extends the Kubernetes API with custom resources.

# Part I

# Requirements

CHAPTER

ONE

# SYSTEM REQUIREMENTS

The Operator supports Percona XtraDB Cluster (PXC) 5.7 and 8.0.

The new `caching_sha2_password` authentication plugin which is default in 8.0 is not supported for the ProxySQL compatibility reasons. Therefore both PXC 5.7 and 8.0 are configured with `default_authentication_plugin = mysql_native_password`.

## Officially supported platforms

The following platforms are supported:

- OpenShift 3.11
- OpenShift 4.2
- Google Kubernetes Engine (GKE) 1.13
- GKE 1.15
- Amazon Elastic Kubernetes Service (EKS) 1.15
- Minikube 1.16

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 Private Volumes provisioning.

## Platform-specific limitations

The Operator is subsequent to specific platform limitations.

- Minikube doesn't support multi-node cluster configurations because of its local nature, which is in collision with the default affinity requirements of the Operator. To arrange this, the *Install Percona XtraDB Cluster on Minikube* instruction includes an additional step which turns off the requirement of having not less than three Nodes.

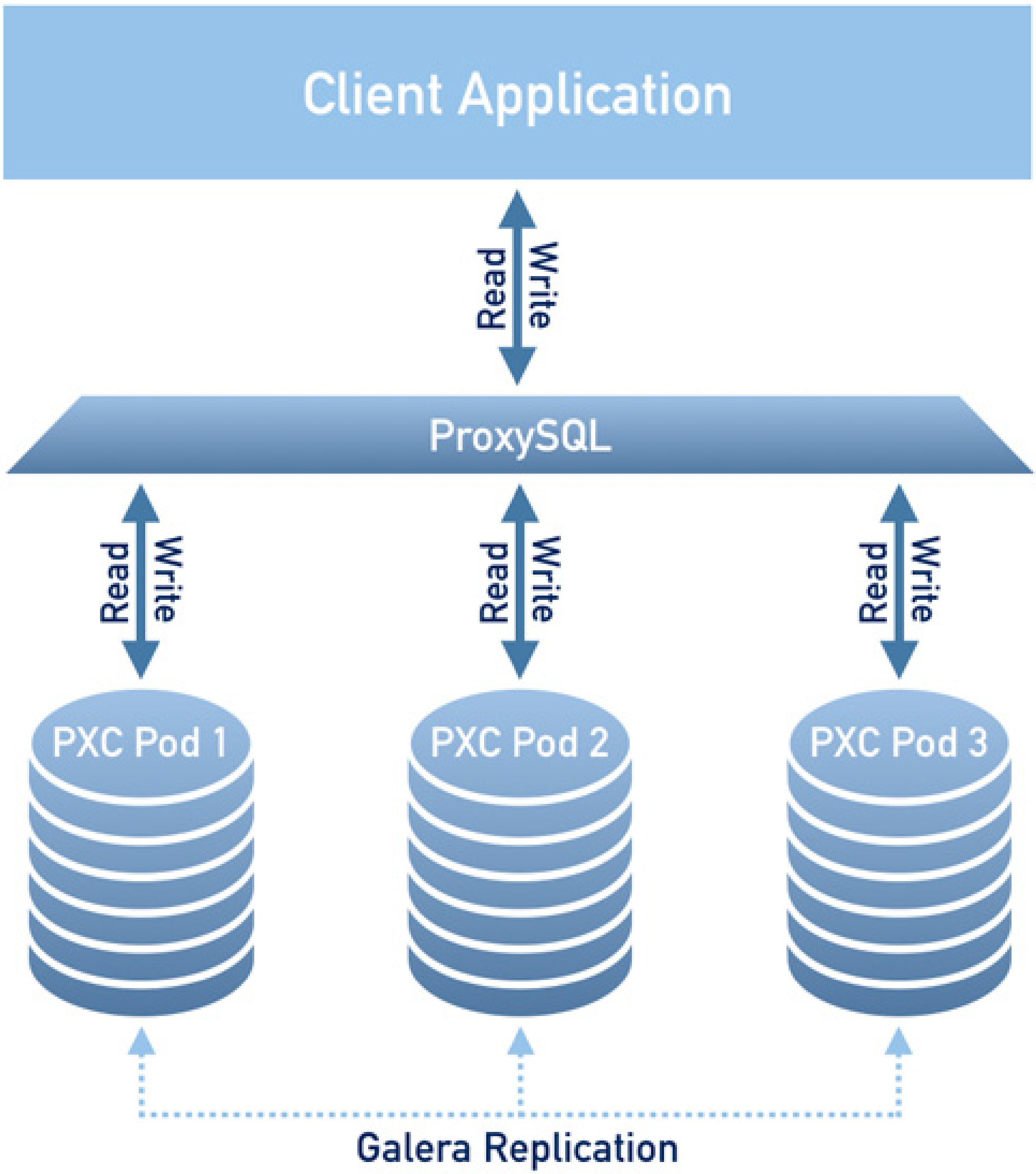
CHAPTER

TWO

# 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-master 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.



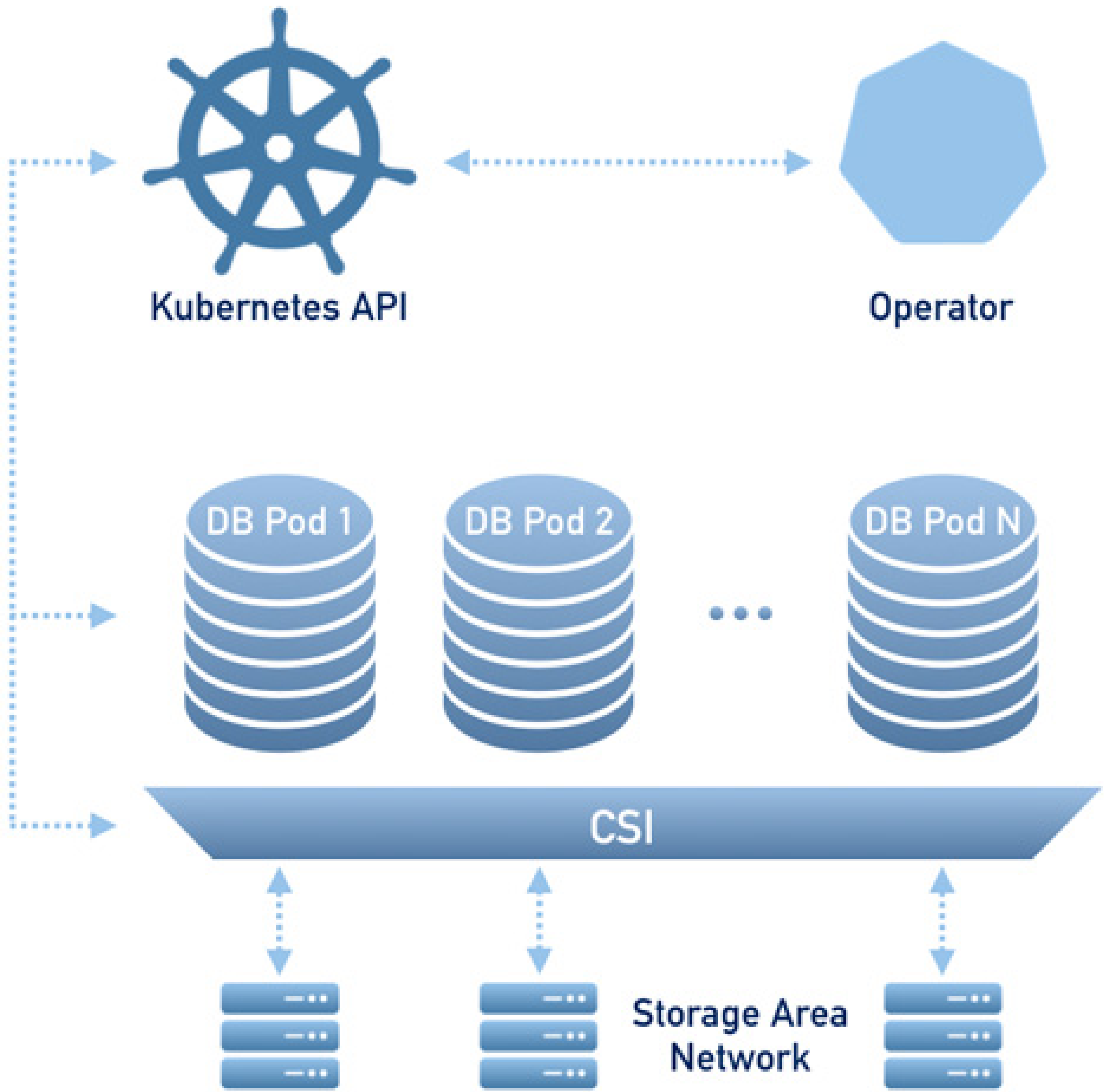

Being a regular MySQL Server instance, each node contains the same set of data synchronized accross nodes. The recommended configuration is to have at least 3 nodes. In a basic setup with this amount of nodes, Percona XtraDB Cluster provides high availability, continuing to function if you take any of the nodes down. Additionally load balancing can be achieved with the ProxySQL daemon, which accepts incoming traffic from MySQL clients and forwards it to backend MySQL servers.

**Note:** Using ProxySQL results in more efficient database workload management in comparison with other load balancers which are not SQL-aware, including built-in ones of the cloud providers, or the Kubernetes NGINX Ingress

Controller.

---

To provide high availability operator uses node affinity to run PXC 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 PXC 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 properly PXC operating.

# Part II

# Installation

CHAPTER

THREE

# INSTALL PERCONA XTRADB CLUSTER ON KUBERNETES

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

```
git clone -b release-1.4.0 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 PXC 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 -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 PXC 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
```

4. Now that's time to add the PXC Users secrets to Kubernetes. They should be placed in the data section of the `deploy/secrets.yaml` file as logins and base64-encoded passwords for the user accounts (see Kubernetes documentation for details).

   **Note:** the following command can be used to get base64-encoded password from a plain text string: `$ echo -n 'plain-text-password' | base64`

   After editing is finished, users secrets should be created (or updated with the new passwords) using the following command:

   ```
   $ kubectl apply -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. The process is over when both operator and replica set pod have reached their Running status:

   ```
   $ kubectl get pods
   NAME                                               READY   STATUS    RESTARTS   AGE
   cluster1-pxc-0                                     1/1     Running   0          5m
   cluster1-pxc-1                                     1/1     Running   0          4m
   cluster1-pxc-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
   ```

7. Check connectivity to newly created cluster

   ```
   $ kubectl run -i --rm --tty percona-client --image=percona:5.7 --restart=Never --␣
   ↪bash -il
   percona-client:/$ mysql -h cluster1-proxysql -uroot -proot_password
   ```

CHAPTER

FOUR

# INSTALL PERCONA XTRADB CLUSTER ON OPENSHIFT

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

```
git clone -b release-1.4.0 https://github.com/percona/percona-xtradb-cluster-
↪operator
cd percona-xtradb-cluster-operator
```

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

1. Now Custom Resource Definition for PXC 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.

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

**Note:** Setting Custom Resource Definition requires your user to have cluster-admin role privileges.

If you want to manage your PXC cluster with a non-privileged user, necessary permissions can be granted by applying the next clusterrole:

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

If you have a cert-manager installed, then you have to execute two more commands to be able 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>
```

2. The next thing to do is to create a new `pxc` project:

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

3. Now RBAC (role-based access control) for PXC 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 OpenShift documentation).

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

   Finally, it's time to start the operator within OpenShift:

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

4. Now that's time to add the PXC Users secrets to OpenShift. They should be placed in the data section of the `deploy/secrets.yaml` file as logins and base64-encoded passwords for the user accounts (see Kubernetes documentation for details).

   **Note:** The following command can be used to get base64-encoded password from a plain text string: `$ echo -n 'plain-text-password' | base64`

   After editing is finished, users secrets should be created (or updated with the new passwords) using the following command:

```
$ oc apply -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:

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

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

```
$ oc get pods
NAME                                              READY   STATUS    RESTARTS   AGE
cluster1-pxc-0                                    1/1     Running   0          5m
cluster1-pxc-1                                    1/1     Running   0          4m
cluster1-pxc-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
```

7. Check connectivity to newly created cluster

```
$ oc run -i --rm --tty percona-client --image=percona:5.7 --restart=Never -- bash␣
↪-il
percona-client:/$ mysql -h cluster1-proxysql -uroot -proot_password
```

CHAPTER

FIVE

# INSTALL PERCONA XTRADB CLUSTER ON MINIKUBE

Installing the PXC Operator 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 PXC Operator on Minikube:

0. Install Minikube, using a way recommended for your system. This includes the installation of the following three components: #. kubectl tool, #. a hypervisor, if it is not already installed, #. 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. After Minikube is successfully started, you can optionally run the Kubernetes dashboard, which visually represents the state of your cluster. Executing `minikube dashboard` will start the dashboard and open it in your default web browser.

1. Clone the percona-xtradb-cluster-operator repository:

```
git clone -b release-1.4.0 https://github.com/percona/percona-xtradb-cluster-
↪operator
cd percona-xtradb-cluster-operator
```

2. Deploy the operator with the following command:

```
kubectl apply -f deploy/bundle.yaml
```

3. Because minikube runs locally, the default `deploy/cr.yaml` file should be edited to adapt the Operator for the the local installation with limited resources. Change the following keys in `pxc` and `proxysql` sections:

   (a) comment `resources.requests.memory` and `resources.requests.cpu` keys (this will fit the Operator in minikube default limitations)

   (b) set `affinity.antiAffinityTopologyKey` key to `"none"` (the Operator will be unable to spread the cluster on several nodes)

   Also, switch `allowUnsafeConfigurations` key to `true` (this option turns off the Operator's control over the cluster configuration, making it possible to deploy Percona XtraDB Cluster as a one-node cluster).

4. Now apply the `deploy/cr.yaml` file with the following command:

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

5. During previous steps, the Operator has generated several secrets, including the password for the `root` user, which you will definitely need to access the cluster. Use `kubectl get secrets` to see the list of Secrets objects (by default Secrets object you are interested in has `my-cluster-secrets` name). Then `kubectl`

`get secret my-cluster-secrets -o yaml` will return the YAML file with generated secrets, including the root password which should look as follows:

```
...
data:
  ...
  root: cm9vdF9wYXNzd29yZA==
```

Here the actual password is base64-encoded, and `echo 'cm9vdF9wYXNzd29yZA==' | base64 --decode` will bring it back to a human-readable form.

6. Check connectivity to a newly created cluster.

   First of all, run percona-client and connect its console output to your terminal (running it may require some time to deploy the correspondent Pod):

```
kubectl run -i --rm --tty percona-client --image=percona:5.7 --restart=Never --␣
↪bash -il
```

   Now run `mysql` tool in the percona-client command shell using the password obtained from the secret:

```
mysql -h cluster1-proxysql -uroot -proot_password
```

CHAPTER

SIX

# SCALE PERCONA XTRADB CLUSTER ON KUBERNETES AND OPENSHIFT

One of the great advantages brought by Kubernetes and the OpenShift platform is the ease of an application scaling. Scaling a Deployment up or down ensures new Pods are created and set to available Kubernetes nodes.

Size of the cluster is controlled by a `size` key in the Custom Resource options configuration, as specified in the Operator Options section. 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, or on the fly, using the following command, which saves the current configuration, updates it and applies the changed version:

```
$ kubectl get pxc/my-cluster -o yaml | sed -e 's/size: 3/size: 5/' | kubectl apply -f␣
↪-
```

In this example we have changed the size of the Percona XtraDB Cluster from `3`, which is a minimum recommended value, to `5` nodes.

**Note:** *Using ``kubectl scale StatefulSet_name`` command to rescale Percona XtraDB Cluster is not recommended, as it makes ``size`` configuration option out of sync, and the next config change may result in reverting the previous number of nodes.*

## Increase the Persistent Volume Claim size

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. In Kubernetes v1.11 the feature was added to allow a user to increase the size of an existing PVC object. The user cannot shrink the size of an existing PVC object. Certain volume types support, be default, expanding PVCs (details about PVCs and the supported volume types can be found in Kubernetes documentation)

The following are the steps to increase the size:

0. Extract and backup the yaml file for the cluster

```
kubectl get pxc cluster1 -o yaml --export > CR_backup.yaml
```

1. Delete the cluster

```
kubectl delete -f CR_backup.yaml
```

2. For each node, edit the yaml to resize the PVC object.

```
kubectl edit pvc datadir-cluster1-pxc-0
```

In the yaml, edit the spec.resources.requests.storage value.

```
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 6Gi
```

Perform the same operation on the other nodes.

```
kubectl edit pvc datadir-cluster1-pxc-1
kubectl edit pvc datadir-cluster1-pxc-2
```

3. In the CR configuration file, use vim or another text editor to edit the PVC size.

```
vim CR_backup.yaml
```

4. Apply the updated configuration to the cluster.

```
kubectl apply -f CR_backup.yaml
```

CHAPTER

SEVEN

# UPDATE PERCONA XTRADB CLUSTER OPERATOR

Starting from the version 1.1.0 the Percona Kubernetes Operator for Percona XtraDB Cluster allows upgrades to newer versions. This upgrade can be done either in semi-automatic or in manual mode.

**Note:** The manual update mode is the recomended way for a production cluster.

**Note:** Only the incremental update to a nearest minor version is supported (for example, update from 1.2.0 to 1.3.0). To update to a newer version, which differs from the current version by more than one, make several incremental updates sequentially.

## Semi-automatic update

1. Edit the `deploy/cr.yaml` file, setting `updateStrategy` key to `RollingUpdate`.
2. Now you should apply a patch to your deployment, supplying necessary image names with a newer version tag. This is done with the `kubectl patch deployment` command. For example, updating to the `1.4.0` version 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:1.4.0"}]}}}}'

kubectl patch pxc cluster1 --type=merge --patch '{
   "metadata": {"annotations":{ "kubectl.kubernetes.io/last-applied-configuration
↪": "{\"apiVersion\":\"pxc.percona.com/v1-3-0\"}" }},
   "spec": {"pxc":{ "image": "percona/percona-xtradb-cluster-operator:1.4.0-pxc" }
↪,
       "proxysql": { "image": "percona/percona-xtradb-cluster-operator:1.4.0-
↪proxysql" },
       "backup":   { "image": "percona/percona-xtradb-cluster-operator:1.4.0-
↪backup" },
       "pmm":      { "image": "percona/percona-xtradb-cluster-operator:1.4.0-pmm"␣
↪}
   }}'
```

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 sts cluster1-pxc
```

# Manual update

1. Edit the `deploy/cr.yaml` file, setting `updateStrategy` key to `OnDelete`.
2. Now you should apply a patch to your deployment, supplying necessary image names with a newer version tag. This is done with the `kubectl patch deployment` command. For example, updating to the `1.4.0` version 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:1.4.0"}]}}}}'

kubectl patch pxc cluster1 --type=merge --patch '{
   "metadata": {"annotations":{ "kubectl.kubernetes.io/last-applied-configuration
↪": "{\"apiVersion\":\"pxc.percona.com/v1-3-0\"}" }},
   "spec": {"pxc":{ "image": "percona/percona-xtradb-cluster-operator:1.4.0-pxc" }
↪,
       "proxysql": { "image": "percona/percona-xtradb-cluster-operator:1.4.0-
↪proxysql" },
       "backup":   { "image": "percona/percona-xtradb-cluster-operator:1.4.0-
↪backup" },
       "pmm":      { "image": "percona/percona-xtradb-cluster-operator:1.4.0-pmm"␣
↪}
   }}'
```

3. 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:

   (a) Delete the Pod using its name with the command like the following one:

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

   (b) 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
   ```

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

CHAPTER

EIGHT

# MONITORING

The Percona Monitoring and Management (PMM) provides an excellent solution to monitor Percona XtraDB Cluster.

## Installing the PMM Server

This first section installs the PMM Server to monitor Percona XtraDB Cluster on Kubernetes or OpenShift. The following steps are optional if you already have installed the PMM Server. The PMM Server available on your network does not require another installation in Kubernetes.

1. The recommended installation approach is based on using helm - the package manager for Kubernetes, which will substantially simplify further steps. So first thing to do is to install helm following its official installation instructions.
2. When the helm is installed, add Percona chart repository and update information of available charts as follows:

```
$ helm repo add percona https://percona-charts.storage.googleapis.com
$ helm repo update
```

3. Now helm can be used to install PMM Server:

```
$ helm install percona/pmm-server --name monitoring --set platform=openshift --
↪set credentials.username=pmm --set "credentials.password=supa|^|pazz"
```

   It is important to specify correct options in the installation command:

   - `platform` should be either `kubernetes` or `openshift` depending on which platform are you using.
   - `name` should correspond to the `serverHost` key in the `pmm` section of the deploy/cr.yaml file with a "-service" suffix, so default `--name monitoring` part of the shown above command corresponds to a `monitoring-service` value of the `serverHost` key.
   - `credentials.username` should correspond to the `serverUser` key in the `pmm` section of the deploy/cr.yaml file.
   - `credentials.password` should correspond to a value of the `pmmserver` secret key specified in `deploy/secrets.yaml` secrets file. Note that password specified in this example is the default development mode password not intended to be used on production systems.

## Installing the PMM Client

The following steps are needed for the PMM client installation:

1. The PMM client installation is initiated by updating the `pmm` section in the deploy/cr.yaml file.

- set `pmm.enabled=true`
- make sure that `serverUser` (the PMM Server user name, `pmm` by default) is the same as one specified for the `credentials.username` parameter on the previous step.
- make sure that `serverHost` (the PMM service name, `monitoring-service` by default) is the same as one specified for the `name` parameter on the previous step, but with additional `-service` suffix.
- make sure that `pmmserver` secret key in the `deploy/secrets.yaml` secrets file is the same as one specified for the `credentials.password` parameter on the previous step (if not, fix it and apply with the `kubectl apply -f deploy/secrets.yaml` command).

When done, apply the edited `deploy/cr.yaml` file:

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

2. To make sure everything gone right, check that correspondent Pods are not continuously restarting (which would occur in case of any errors on the previous two steps):

```
$ kubectl get pods
$ kubectl logs cluster1-pxc-node-0 -c pmm-client
```

3. Find the external IP address (`EXTERNAL-IP` field in the output of `kubectl get service/monitoring-service -o wide`). This IP address can be used to access PMM via *https* in a web browser, with the login/password authentication, already configured and able to show Percona XtraDB Cluster metrics.

CHAPTER

NINE

# 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 XtraDB Cluster Operator 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
```

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:1.2.0
$ docker push 172.30.162.173:5000/pxc/percona-xtradb-cluster-operator:1.2.0
```

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   1.4.0     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:1.4.0`) into the `image:` option in `deploy/operator.yaml` configuration file.

   Please note it is possible to specify `imagePullSecrets` option for all images, if the registry requires authentication.

8. Repeat steps 3-5 for other images, and update corresponding options in the `deploy/cr.yaml` file.

9. Now follow the standard Percona XtraDB Cluster Operator installation instruction.

# Percona certified images

Following table presents Percona's certified images to be used with the Percona XtraDB Cluster Operator:

| Image | Digest |
|---|---|
| percona/percona-xtradb-cluster-operator:1.4.0 | 277d62967e94dc4e7d0569656413967e6a8597842753da05f083543e68c9b719 |
| percona/percona-xtradb-cluster-operator:1.4.0-proxysql | 1ee8b9c291dac955dd98441187476fe8c3b5a4930e9e4dc39b9534376d0cc4f2 |
| percona/percona-xtradb-cluster-operator:1.4.0-pxc8.0 | 58296417cc97378b906e12855cb1f4f2420f06168d2096acc08a93c8afa793f6 |
| percona/percona-xtradb-cluster-operator:1.4.0-pxc8.0-backup | 566ea1f6cf9387a06898d5f7af15189ed577d3af771d5954b2e869593b80cb6b |
| percona/percona-xtradb-cluster-operator:1.4.0-pxc5.7 | 4ff39dab7872a4b45250ca170604f6bce1fcc52510407f6cbd93cd81f5a32d8f |
| percona/percona-xtradb-cluster-operator:1.4.0-pxc5.7-backup | ca8e3fd49d3a2ac15c0b9c44f8ea4e0f8240789de274859a91ec9cd8d8e80763 |
| percona/percona-xtradb-cluster-operator:1.4.0-pmm | 28bbb6693689a15c407c85053755334cd25d864e632ef7fed890bc85726cfb68 |
| percona/percona-xtradb-cluster-operator:1.3.0 | 85cfaf78394e21b722be92015912c39e483f7ae5de1aab114293520a3825eb99 |
| percona/percona-xtradb-cluster-operator:1.3.0-proxysql | 8e40dec83008894aaa438f31233acb90f29969ad660cce26b700075eeaf9d34b |
| percona/percona-xtradb-cluster-operator:1.3.0-pxc | a7d04c0a343fd0b7f08a306bb9f00b6df2f398bb7163990ba787f037c294853e |
| percona/percona-xtradb-cluster-operator:1.3.0-backup | f786d92d96c5036df1785647d323081235c06fad56653ca93ae44af85c2d19e8 |
| percona/percona-xtradb-cluster-operator:1.3.0-pmm | 28bbb6693689a15c407c85053755334cd25d864e632ef7fed890bc85726cfb68 |
| percona/percona-xtradb-cluster-operator:1.2.0 | 841c07eef30605080bfe80e549f9332ab6b9755fcbc42aacbf86e4ac9ef0e444 |
| percona/percona-xtradb-cluster-operator:1.2.0-pxc | d38482fcbe0d0f169e41eefd889404e967e8abc65a6890cbab4dd1f3ea2229df |
| percona/percona-xtradb-cluster-operator:1.2.0-proxysql | 1385b77d3498cebc201426821fda620e0884e8fdaba6756240c9821948864af3 |
| percona/percona-xtradb-cluster-operator:1.2.0-backup | bd45486507321de67ff8ad2fa40c4f55fc20bd15db6369b61c73a5db11bb57cd |
| percona/percona-xtradb-cluster-operator:1.2.0-broker | c0903f41539767fcfe49da815e1c3bfefe4e48a36912b64fb5350b09b51cab32 |
| percona/percona-xtradb-cluster-operator:1.2.0-pmm | 28bbb6693689a15c407c85053755334cd25d864e632ef7fed890bc85726cfb68 |
| percona/percona-xtradb-cluster-operator:1.1.0 | fbfc2fc5c3afc80f18dddc5a1c3439fab89950081cf86c3439a226d4c97198eb |
| percona/percona-xtradb-cluster-operator:1.1.0-pxc | a66a9212760e823af3c666a78e4b480cc7cc0d8be5cfa29c8141319c0036706e |
| percona/percona-xtradb-cluster-operator:1.1.0-proxysql | ac952afb3721eafe86431155da7c3f7f90c4e800491c400a4222b650fd393357 |
| percona/percona-xtradb-cluster-operator:1.1.0-backup | 4852da039dd2a1d3ae9243ec634c14fd9f9e5af18a1fc6c7c9d25d4287dd6941 |
| percona/percona-xtradb-cluster-operator:1.0.0 | b9e97c66a69f448898f8d43b92dd0314aaf53997b70824056dd3d0aec62488eb |
| percona/percona-xtradb-cluster-operator:1.0.0-pxc | 6797c8492cff8092b39cdce75d7d85b9c2d4d08c4f6e0ba7b05c21562a54f168 |
| percona/percona-xtradb-cluster-operator:1.0.0-proxysql | b9360f1a8dc1e57e5ae7442373df02869ddc4da69ef9190190edde70b465235e |
| percona/percona-xtradb-cluster-operator:1.0.0-backup | 652be455c8faf2d610de15e3568ff57fe8630fa353b6d97ff1c6b91d44741f8b |

CHAPTER

TEN

# DEPLOY PERCONA XTRADB CLUSTER WITH SERVICE BROKER

Percona Service Broker provides the Open Service Broker object to facilitate the operator deployment within high-level visual tools. Following steps are needed to use it while installing the Percona XtraDB Cluster on the OpenShift platform:

1. The Percona Service Broker is to be deployed based on the `percona-broker.yaml` file. To use it you should first enable the Service Catalog, which can be done with the following command:

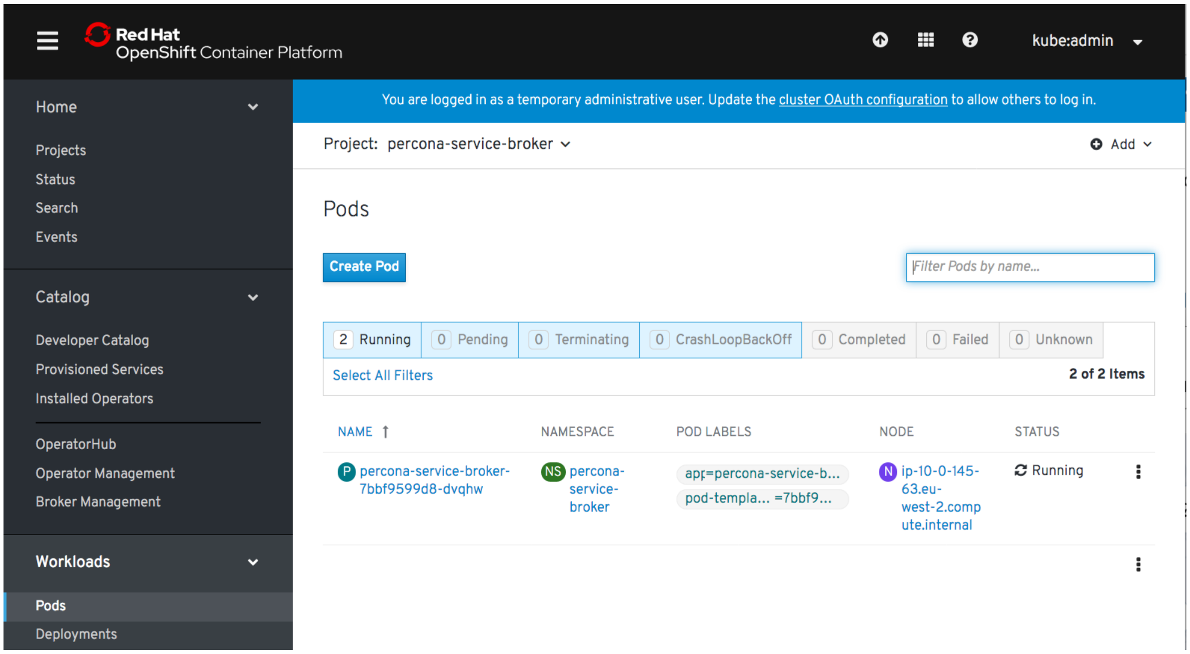
   ```
   $ oc patch servicecatalogapiservers cluster --patch '{"spec":{"managementState":
   ↪"Managed"}}' --type=merge
   $ oc patch servicecatalogcontrollermanagers cluster --patch '{"spec":{
   ↪"managementState":"Managed"}}' --type=merge
   ```

   When Service Catalog is enabled, download and install the Percona Service Broker in a typical OpenShift way:

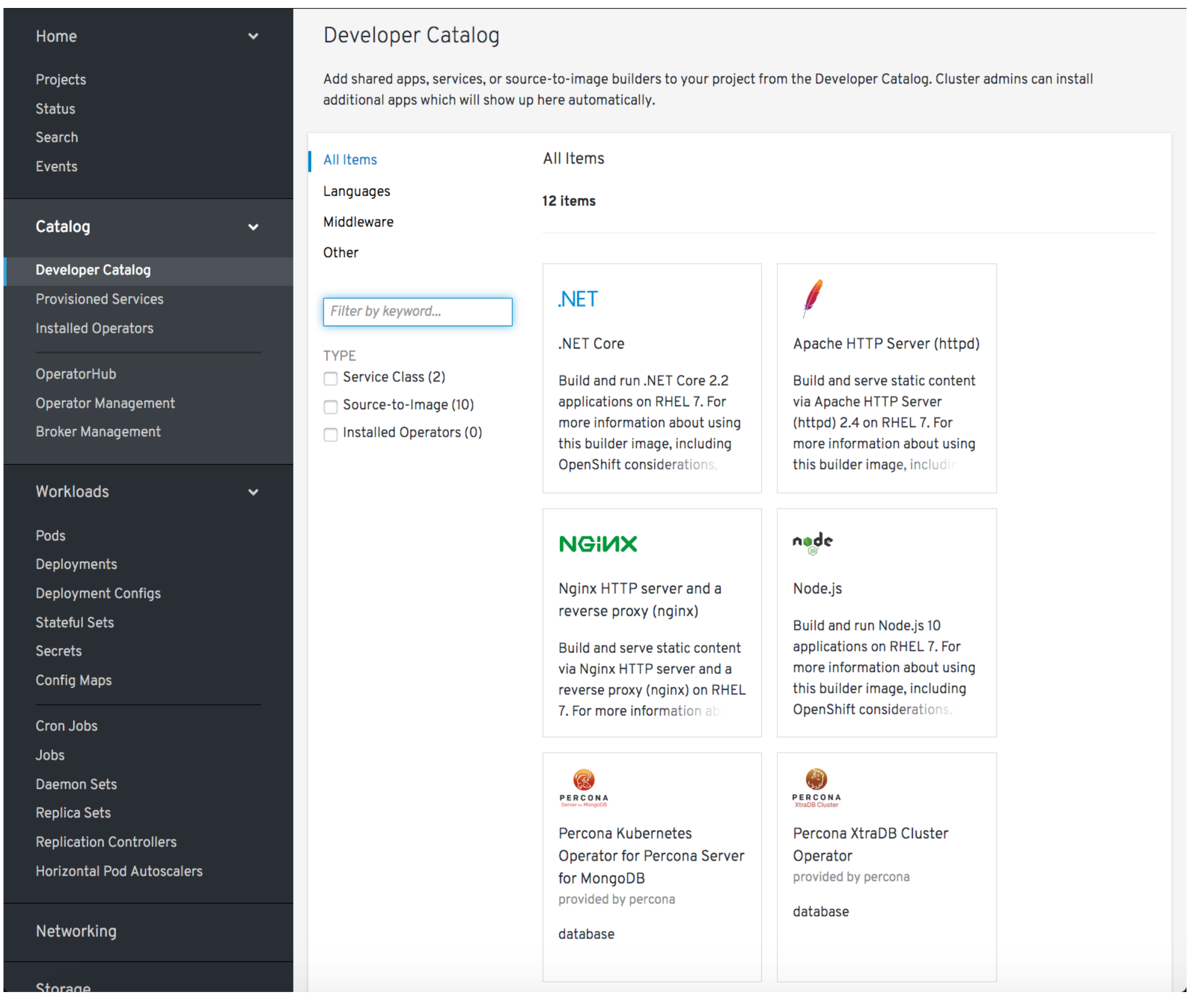
   ```
   $ oc apply -f https://raw.githubusercontent.com/Percona-Lab/percona-dbaas-cli/
   ↪broker/deploy/percona-broker.yaml
   ```

   **Note:** This step should be done only once; the step does not need to be repeated with any other Operator deployments. It will automatically create and setup the needed service and projects catalog with all necessary objects.

2. Now login to your OpenShift Console Web UI and switch to the percona-service-broker project. You can check its Pod running on a correspondent page:

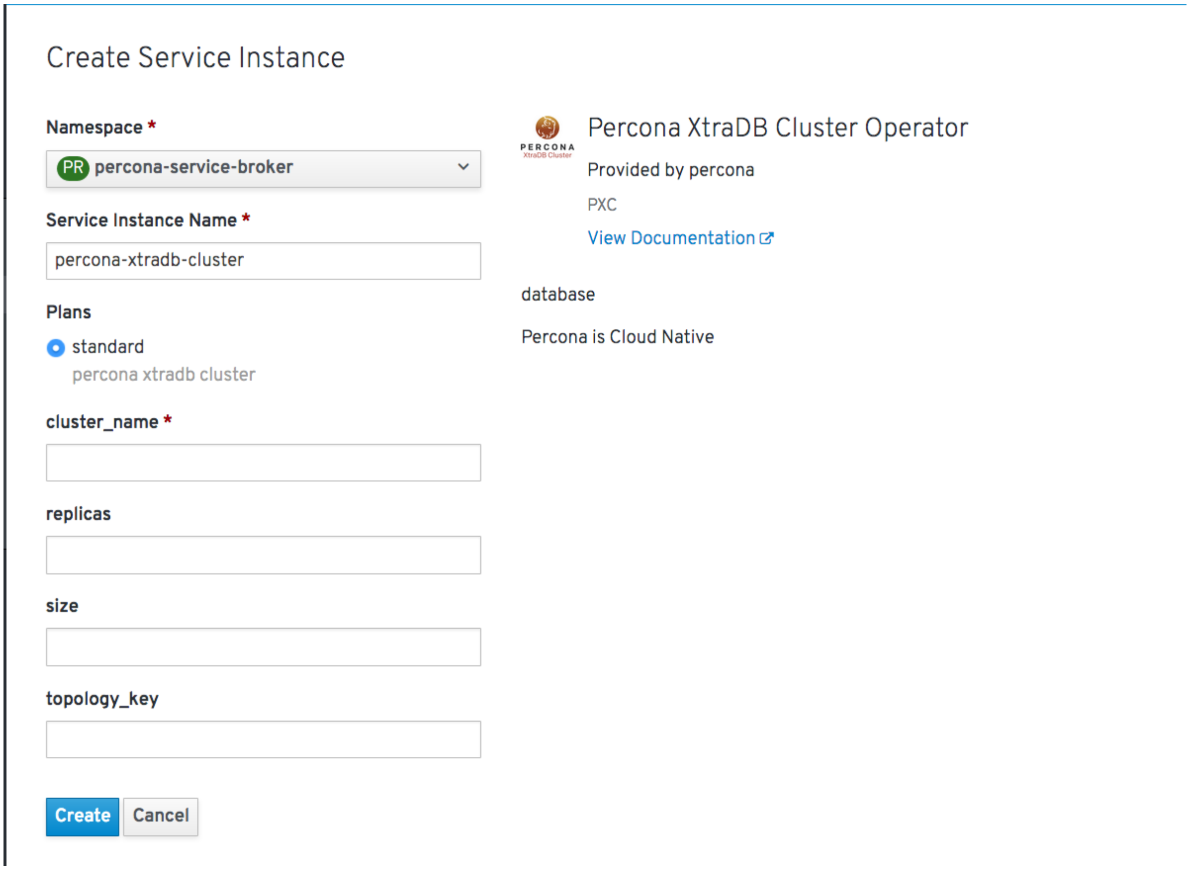



Now switch to the Developer Catalog and select Percona XtraDB Cluster Operator:



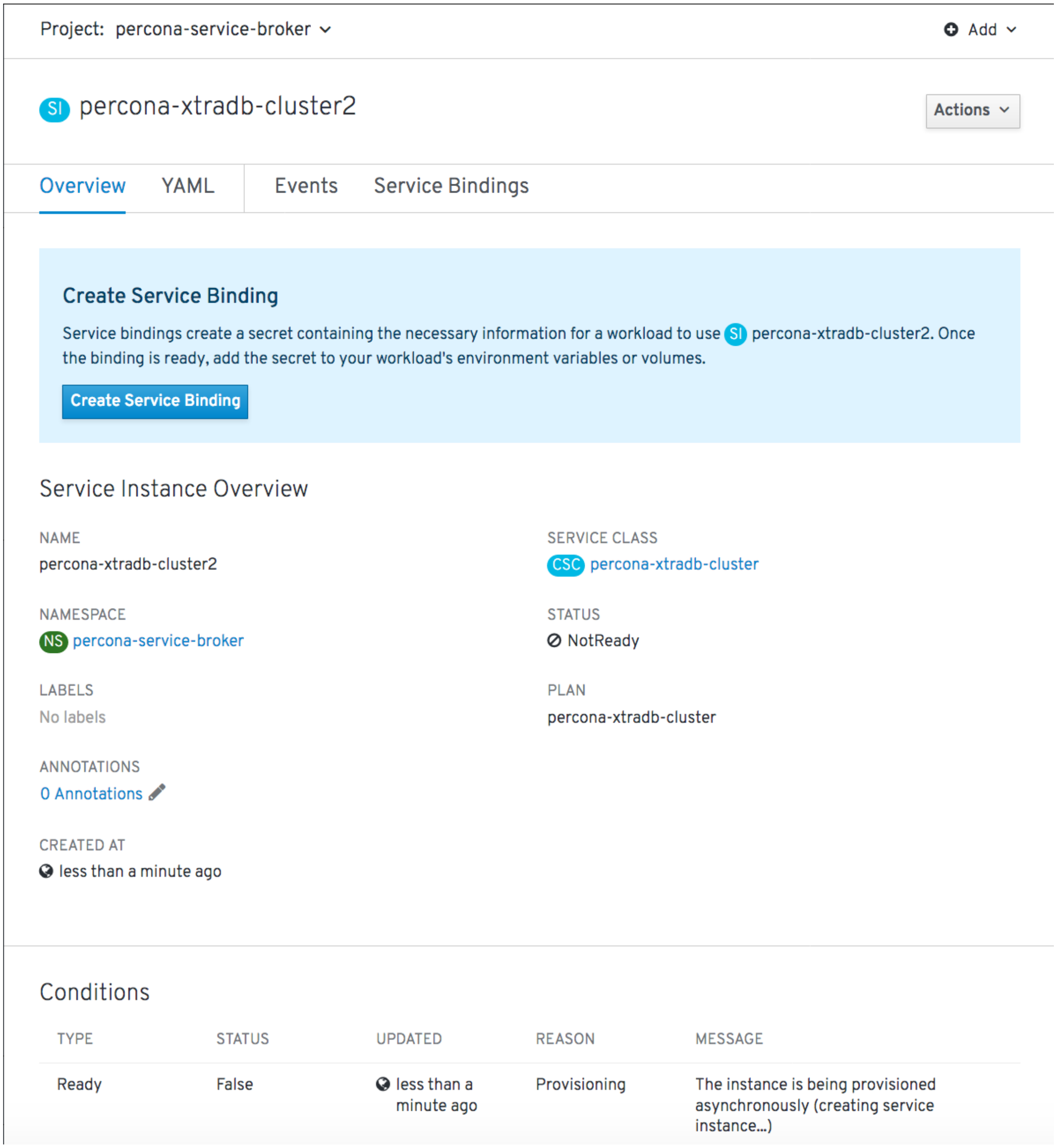
Choose `Percona XtraDB Cluster Operator` item. This will lead you to the Operator page with the *Create Service Instance* button.

3. Clicking the *Create Service Instance* button guides you to the next page:




The two necessary fields are *Service Instance Name* and *Cluster Name*, which should be unique for your project.

4. Clicking the *Create* button gets you to the *Overview* page, which reflects the process of the cluster creation process:




You can also track Pods to see when they are deployed and track any errors.

# Part III

# Configuration

CHAPTER

ELEVEN

# USERS

The Operator requires Kubernetes Secrets to be deployed before the PXC Cluster is started. The name of the required secrets can be set in `deploy/cr.yaml` under the `spec.secretsName` section.

## Unprivileged users

There are no unprivileged (general purpose) user accounts created by default. If you need general purpose users, please run commands below:

```
$ kubectl run -it --rm percona-client --image=percona:5.7 --restart=Never -- mysql -
↪hcluster1-pxc -uroot -proot_password
mysql> GRANT ALL PRIVILEGES ON database1.* TO 'user1'@'%' IDENTIFIED BY 'password1';
```

Sync users on the ProxySQL node:

```
$ kubectl exec -it cluster1-proxysql-0 -- proxysql-admin --config-file=/etc/proxysql-
↪admin.cnf --syncusers
```

Verify that the user was created successfully. If successful, the following command will let you successfully login to MySQL shell via ProxySQL:

```
$ kubectl run -it --rm percona-client --image=percona:5.7 --restart=Never -- bash -il
percona-client:/$ mysql -h cluster1-proxysql -uuser1 -ppassword1
mysql> SELECT * FROM database1.table1 LIMIT 1;
```

You may also try executing any simple SQL statement to ensure the permissions have been successfully granted.

## System Users

*Default Secret name:* `my-cluster-secrets`

*Secret name field:* `spec.secretsName`

The Operator requires system-level PXC users to automate the PXC deployment.

**Warning:** *These users should not be used to run an application.*

| User Purpose | Username | Password Secret Key | Description |
|---|---|---|---|
| Admin | root | root | Database administrative user, should only be used for maintenance tasks |
| ProxySQLAdmin | proxyadmin | proxyadmin | ProxySQL administrative user, can be used to add general-purpose ProxySQL users |
| Backup | xtrabackup | xtrabackup | User to run backups |
| Cluster Check | clustercheck | clustercheck | User for liveness checks and readiness checks |
| PMM Client User | monitor | monitor | User for PMM agent |
| PMM Server Password | should be set through the operator options | pmmserver | Password used to access PMM Server |

## Development Mode

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

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

| Secret Key | Secret Value |
|---|---|
| root | `root_password` |
| xtrabackup | `backup_password` |
| monitor | `monitor` |
| clustercheck | `clustercheckpassword` |
| proxyuser | `s3cret` |
| proxyadmin | `admin_password` |
| pmmserver | `supa|^|pazz` |

**Warning:** *Do not use the default PXC user passwords in production!*

CHAPTER

TWELVE

# CUSTOM RESOURCE OPTIONS

The operator is configured via the spec section of the deploy/cr.yaml file. This file contains the following spec sections to configure three main subsystems of the cluster:

| Key | Value type | Default | Description |
|---|---|---|---|
| pxc | subdoc | | Percona XtraDB Cluster general section |
| proxysql | subdoc | | ProxySQL section |
| pmm | subdoc | | Percona Monitoring and Management section |
| backup | subdoc | | Percona XtraDB Cluster backups section |
| allowUnsafeConfigurations | boolean | `false` | Prevents users from configuring a cluster with unsafe parameters such as starting the cluster with less than 3 nodes or starting the cluster without TLS/SSL certificates |
| secretsName | string | `my-cluster-secrets` | A name for *users secrets* |
| vaultSecretName | string | `keyring-secret-vault` | A secret for the HashiCorp Vault to carry on *Data-at-Rest Encryption* |
| sslSecretName | string | `my-cluster-ssl` | A secret with TLS certificate generated for *external* communications, see *Transport Layer Security (TLS)* for details |
| sslInternalSecretName | string | `my-cluster-ssl-internal` | A secret with TLS certificate generated for *internal* communications, see *Transport Layer Security (TLS)* for details |

## PXC Section

The `pxc` section in the deploy/cr.yaml file contains general configuration options for the Percona XtraDB Cluster.

| | |
|---|---|
| **Key** | pxc.size |
| **Value** | int |
| **Example** | `3` |

Continued on next page

Table 12.1 – continued from previous page

| | |
|---|---|
| **Description** | The size of the Percona XtraDB cluster must be >= 3 for High Availability |
| | |
| **Key** | pxc.image |
| **Value** | string |
| **Example** | `percona/percona-xtradb-cluster-operator:1.4.0-pxc` |
| **Description** | The Docker image of the Percona cluster used |
| | |
| **Key** | pxc.readinessDelaySec |
| **Value** | int |
| **Example** | `15` |
| **Description** | Adds a delay before a run check to verify the application is ready to process traffic |
| | |
| **Key** | pxc.livenessDelaySec |
| **Value** | int |
| **Example** | `300` |
| **Description** | Adds a delay before the run check ensures the application is healthy and capable of processing requests |
| | |
| **Key** | pxc.forceUnsafeBootstrap |
| **Value** | boolean |
| **Example** | `false` |
| **Description** | The setting can be reset in case of a sudden crash when all nodes may be considered unsafe to bootstrap from. The setting lets a node be selected and set to `safe_to_bootstrap` and provides data recovery |
| | |
| **Key** | pxc.configuration |
| **Value** | string |
| **Example** | `\|`<br>`[mysqld]`<br>`wsrep_debug=ON`<br>`wsrep-provider_options=gcache.size=1G;gcache.recover=yes` |
| **Description** | The `my.cnf` file options to be passed to Percona XtraDB cluster nodes |
| | |
| **Key** | pxc.imagePullSecrets.name |
| **Value** | string |
| **Example** | `private-registry-credentials` |
| **Description** | The Kubernetes ImagePullSecret |
| | |
| **Key** | pxc.priorityClassName |
| **Value** | string |
| **Example** | `high-priority` |
| **Description** | The Kubernetes Pod priority class |
| | |
| **Key** | pxc.schedulerName |
| **Value** | string |
| **Example** | `default-scheduler` |
| **Description** | The Kubernetes Scheduler |
| | |
| **Key** | pxc.annotations |
| **Value** | label |

Continued on next page

Table 12.1 – continued from previous page

| | |
|---|---|
| **Example** | `iam.amazonaws.com/role: role-arn` |
| **Description** | The Kubernetes annotations |
| | |
| **Key** | pxc.labels |
| **Value** | label |
| **Example** | `rack: rack-22` |
| **Description** | Labels are key-value pairs attached to objects |
| | |
| **Key** | pxc.resources.requests.memory |
| **Value** | string |
| **Example** | `1G` |
| **Description** | The Kubernetes memory requests for a PXC container |
| | |
| **Key** | pxc.resources.requests.cpu |
| **Value** | string |
| **Example** | `600m` |
| **Description** | Kubernetes CPU requests for a PXC container |
| | |
| **Key** | pxc.resources.limits.memory |
| **Value** | string |
| **Example** | `1G` |
| **Description** | Kubernetes memory limits for a PXC container |
| | |
| **Key** | pxc.nodeSelector |
| **Value** | label |
| **Example** | `disktype: ssd` |
| **Description** | Kubernetes nodeSelector |
| | |
| **Key** | pxc.affinity.topologyKey |
| **Value** | string |
| **Example** | `kubernetes.io/hostname` |
| **Description** | The Operator topology key node anti-affinity constraint |
| | |
| **Key** | pxc.affinity.advanced |
| **Value** | subdoc |
| **Example** | |
| **Description** | 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 |
| | |
| **Key** | pxc.tolerations |
| **Value** | subdoc |
| **Example** | `node.alpha.kubernetes.io/unreachable` |
| **Description** | Kubernetes Pod tolerations |
| | |
| **Key** | pxc.podDisruptionBudget.maxUnavailable |
| **Value** | int |
| **Example** | `1` |
| **Description** | The Kubernetes podDisruptionBudget specifies the number of Pods from the set unavailable after the eviction |

Continued on next page

Table 12.1 – continued from previous page

| | |
|---|---|
| **Key** | pxc.podDisruptionBudget.minAvailable |
| **Value** | int |
| **Example** | `0` |
| **Description** | The Kubernetes podDisruptionBudget Pods that must be available after an eviction |
| | |
| **Key** | pxc.volumeSpec.emptyDir |
| **Value** | string |
| **Example** | `{}` |
| **Description** | The Kubernetes emptyDir volume The directory created on a node and accessible to the PXC Pod containers |
| | |
| **Key** | pxc.volumeSpec.hostPath.path |
| **Value** | string |
| **Example** | `/data` |
| **Description** | Kubernetes hostPath The volume that mounts a directory from the host node's filesystem into your Pod. The path property is required |
| | |
| **Key** | pxc.volumeSpec.hostPath.type |
| **Value** | string |
| **Example** | `Directory` |
| **Description** | The Kubernetes hostPath. An optional property for the hostPath |
| | |
| **Key** | pxc.volumeSpec.persistentVolumeClaim.storageClassName |
| **Value** | string |
| **Example** | `standard` |
| **Description** | Set the Kubernetes storage class to use with the PXC PersistentVolumeClaim |
| | |
| **Key** | pxc.volumeSpec.persistentVolumeClaim.accessModes |
| **Value** | array |
| **Example** | `[ReadWriteOnce]` |
| **Description** | The Kubernetes PersistentVolumeClaim access modes for the Percona XtraDB cluster |
| | |
| **Key** | pxc.volumeSpec.resources.requests.storage |
| **Value** | string |
| **Example** | `6Gi` |
| **Description** | The Kubernetes PersistentVolumeClaim size for the Percona XtraDB cluster |
| | |
| **Key** | pxc.gracePeriod |
| **Value** | int |
| **Example** | `600` |
| **Description** | The Kubernetes grace period when terminating a Pod |
| | |
| **Key** | pxc.containerSecurityContext |
| **Value** | subdoc |
| **Example** | `privileged: true` |
| **Description** | A custom Kubernetes Security Context for a Container to be used instead of the default one |
| | |
| **Key** | pxc.podSecurityContext |
| **Value** | subdoc |

Continued on next page

Table 12.1 – continued from previous page

| | |
|---|---|
| **Example** | `fsGroup: 1001`<br>`supplementalGroups: [1001, 1002, 1003]` |
| **Description** | A custom Kubernetes Security Context for a Pod to be used instead of the default one |

## ProxySQL Section

The `proxysql` section in the deploy/cr.yaml file contains configuration options for the ProxySQL daemon.

| | |
|---|---|
| **Key** | proxysql.enabled |
| **Value** | boolean |
| **Example** | `true` |
| **Description** | Enables or disables load balancing with ProxySQL Services |
| | |
| **Key** | proxysql.size |
| **Value** | int |
| **Example** | `1` |
| **Description** | The number of the ProxySQL daemons to provide load balancing must be = 1 in current release |
| | |
| **Key** | proxysql.image |
| **Value** | string |
| **Example** | `percona/percona-xtradb-cluster-operator:1.4.0-proxysql` |
| **Description** | ProxySQL Docker image to use |
| | |
| **Key** | proxysql.imagePullSecrets.name |
| **Value** | string |
| **Example** | `private-registry-credentials` |
| **Description** | The Kubernetes imagePullSecrets for the ProxySQL image |
| | |
| **Key** | proxysql.annotations |
| **Value** | label |
| **Example** | `iam.amazonaws.com/role: role-arn` |
| **Description** | The Kubernetes annotations metadata |
| | |
| **Key** | proxysql.labels |
| **Value** | label |
| **Example** | `rack: rack-22` |
| **Description** | Labels are key-value pairs attached to objects |
| | |
| **Key** | proxysql.servicetype |
| **Value** | string |
| **Example** | `ClusterIP` |
| **Description** | Specifies the type of Kubernetes Service to be used |
| | |
| **Key** | proxysql.resources.requests.memory |
| **Value** | string |
| **Example** | `1G` |
| **Description** | The Kubernetes memory requests for the main ProxySQL container |
| | |
| **Key** | proxysql.resources.requests.cpu |

Continued on next page

Table 12.2 – continued from previous page

| | |
|---|---|
| **Value** | string |
| **Example** | `600m` |
| **Description** | Kubernetes CPU requests for the main ProxySQL container |
| | |
| **Key** | proxysql.resources.limits.memory |
| **Value** | string |
| **Example** | `1G` |
| **Description** | Kubernetes memory limits for the main ProxySQL container |
| | |
| **Key** | proxysql.resources.limits.cpu |
| **Value** | string |
| **Example** | `700m` |
| **Description** | Kubernetes CPU limits for the main ProxySQL container |
| | |
| **Key** | proxysql.sidecarResources.requests.memory |
| **Value** | string |
| **Example** | `1G` |
| **Description** | The Kubernetes memory requests for the sidecar ProxySQL containers |
| | |
| **Key** | proxysql.sidecarResources.requests.cpu |
| **Value** | string |
| **Example** | `500m` |
| **Description** | Kubernetes CPU requests for the sidecar ProxySQL containers |
| | |
| **Key** | proxysql.sidecarResources.limits.memory |
| **Value** | string |
| **Example** | `2G` |
| **Description** | Kubernetes memory limits for the sidecar ProxySQL containers |
| | |
| **Key** | proxysql.sidecarResources.limits.cpu |
| **Value** | string |
| **Example** | `600m` |
| **Description** | Kubernetes CPU limits for the sidecar ProxySQL containers |
| | |
| **Key** | proxysql.priorityClassName |
| **Value** | string |
| **Example** | `high-priority` |
| **Description** | The Kubernetes Pod Priority class for ProxySQL |
| | |
| **Key** | proxysql.schedulerName |
| **Value** | string |
| **Example** | `default-scheduler` |
| **Description** | The Kubernetes Scheduler |
| | |
| **Key** | proxysql.nodeSelector |
| **Value** | label |
| **Example** | `disktype: ssd` |
| **Description** | Kubernetes nodeSelector |
| | |
| **Key** | proxysql.affinity.topologyKey |

Continued on next page

Table 12.2 – continued from previous page

| | |
|---|---|
| **Value** | string |
| **Example** | `kubernetes.io/hostname` |
| **Description** | The Operator topology key node anti-affinity constraint |
| | |
| **Key** | proxysql.affinity.advanced |
| **Value** | subdoc |
| **Example** | |
| **Description** | If available it makes a topologyKey node affinity constraint to be ignored |
| | |
| **Key** | proxysql.tolerations |
| **Value** | subdoc |
| **Example** | `node.alpha.kubernetes.io/unreachable` |
| **Description** | Kubernetes Pod tolerations |
| | |
| **Key** | proxysql.volumeSpec.emptyDir |
| **Value** | string |
| **Example** | `{}` |
| **Description** | The Kubernetes emptyDir volume The directory created on a node and accessible to the PXC Pod containers |
| | |
| **Key** | proxysql.volumeSpec.hostPath.path |
| **Value** | string |
| **Example** | `/data` |
| **Description** | Kubernetes hostPath The volume that mounts a directory from the host node's filesystem into your Pod. The path property is required |
| | |
| **Key** | proxysql.volumeSpec.hostPath.type |
| **Value** | string |
| **Example** | `Directory` |
| **Description** | The Kubernetes hostPath. An optional property for the hostPath |
| | |
| **Key** | proxysql.volumeSpec.persistentVolumeClaim.storageClassName |
| **Value** | string |
| **Example** | `standard` |
| **Description** | Set the Kubernetes storage class to use with the PXC PersistentVolumeClaim |
| | |
| **Key** | proxysql.volumeSpec.persistentVolumeClaim.accessModes |
| **Value** | array |
| **Example** | `[ReadWriteOnce]` |
| **Description** | The Kubernetes PersistentVolumeClaim access modes for the Percona XtraDB cluster |
| | |
| **Key** | proxysql.volumeSpec.resources.requests.storage |
| **Value** | string |
| **Example** | `6Gi` |
| **Description** | The Kubernetes PersistentVolumeClaim size for the Percona XtraDB cluster |
| | |
| **Key** | proxysql.podDisruptionBudget.maxUnavailable |
| **Value** | int |
| **Example** | `1` |

Continued on next page

Table 12.2 – continued from previous page

| | |
|---|---|
| **Description** | The Kubernetes podDisruptionBudget specifies the number of Pods from the set unavailable after the eviction |
| | |
| **Key** | proxysql.podDisruptionBudget.minAvailable |
| **Value** | int |
| **Example** | `0` |
| **Description** | The Kubernetes podDisruptionBudget Pods that must be available after an eviction |
| | |
| **Key** | proxysql.gracePeriod |
| **Value** | int |
| **Example** | `30` |
| **Description** | The Kubernetes grace period when terminating a Pod |

## PMM Section

The `pmm` section in the deploy/cr.yaml file contains configuration options for Percona Monitoring and Management.

| | |
|---|---|
| **Key** | pmm.enabled |
| **Value** | boolean |
| **Example** | `false` |
| **Description** | Enables or disables monitoring Percona XtraDB cluster with PMM |
| | |
| **Key** | pmm.image |
| **Value** | string |
| **Example** | `perconalab/pmm-client:1.17.1` |
| **Description** | PMM client Docker image to use |
| | |
| **Key** | pmm.serverHost |
| **Value** | string |
| **Example** | `monitoring-service` |
| **Description** | Address of the PMM Server to collect data from the cluster |
| | |
| **Key** | pmm.serverUser |
| **Value** | string |
| **Example** | `pmm` |
| **Description** | The PMM Serve_User. The PMM Server password should be configured using Secrets |
| | |
| **Key** | pmm.resources.requests.memory |
| **Value** | string |
| **Example** | `200M` |
| **Description** | The Kubernetes memory requests for a PMM container |
| | |
| **Key** | pmm.resources.requests.cpu |
| **Value** | string |
| **Example** | `500m` |
| **Description** | Kubernetes CPU requests for a PMM container |

# Backup Section

The `backup` section in the deploy/cr.yaml file contains the following configuration options for the regular Percona XtraDB Cluster backups.

| | |
|---|---|
| **Key** | backup.image |
| **Value** | string |
| **Example** | `percona/percona-xtradb-cluster-operator:1.4.0-backup` |
| **Description** | The Percona XtraDB cluster Docker image to use for the backup |
| | |
| **Key** | backup.imagePullSecrets.name |
| **Value** | string |
| **Example** | `private-registry-credentials` |
| **Description** | The Kubernetes imagePullSecrets for the specified image |
| | |
| **Key** | backup.storages.<storage-name>.type |
| **Value** | string |
| **Example** | `s3` |
| **Description** | The cloud storage type used for backups. Only `s3` and `filesystem` types are supported |
| | |
| **Key** | backup.storages.<storage-name>.s3.credentialsSecret |
| **Value** | string |
| **Example** | `my-cluster-name-backup-s3` |
| **Description** | The Kubernetes secret for backups. It should contain `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` keys. |
| | |
| **Key** | backup.storages.<storage-name>.s3.bucket |
| **Value** | string |
| **Example** | |
| **Description** | The Amazon S3 bucket name for backups |
| | |
| **Key** | backup.storages.s3.<storage-name>.region |
| **Value** | string |
| **Example** | `us-east-1` |
| **Description** | The AWS region to use. Please note **this option is mandatory** for Amazon and all S3-compatible storages |
| | |
| **Key** | backup.storages.s3.<storage-name>.endpointUrl |
| **Value** | string |
| **Example** | |
| **Description** | The endpoint URL of the S3-compatible storage to be used (not needed for the original Amazon S3 cloud) |
| | |
| **Key** | backup.storages.<storage-name>.persistentVolumeClaim.type |
| **Value** | string |
| **Example** | `filesystem` |
| **Description** | The persistent volume claim storage type |
| | |
| **Key** | backup.storages.<storage-name>.persistentVolumeClaim.storageClassName |
| **Value** | string |
| **Example** | `standard` |

Continued on next page

Table 12.3 – continued from previous page

| | |
|---|---|
| **Description** | Set the Kubernetes Storage Class to use with the PXC backups PersistentVolumeClaims for the `filesystem` storage type |
| | |
| **Key** | backup.storages.<storage-name>.persistentVolumeClaim.accessModes |
| **Value** | array |
| **Example** | `[ReadWriteOne]` |
| **Description** | The Kubernetes PersistentVolume access modes |
| | |
| **Key** | backup.storages.<storage-name>.persistentVolumeClaim.storage |
| **Value** | string |
| **Example** | `6Gi` |
| **Description** | Storage size for the PersistentVolume |
| | |
| **Key** | backup.storages.<storage-name>.annotations |
| **Value** | label |
| **Example** | `iam.amazonaws.com/role: role-arn` |
| **Description** | The Kubernetes annotations |
| | |
| **Key** | backup.storages.<storage-name>.labels |
| **Value** | label |
| **Example** | `rack: rack-22` |
| **Description** | Labels are key-value pairs attached to objects |
| | |
| **Key** | backup.storages.<storage-name>.resources.requests.memory |
| **Value** | string |
| **Example** | `1G` |
| **Description** | The Kubernetes memory requests for a PXC container |
| | |
| **Key** | backup.storages.<storage-name>.resources.requests.cpu |
| **Value** | string |
| **Example** | `600m` |
| **Description** | Kubernetes CPU requests for a PXC container |
| | |
| **Key** | backup.storages.<storage-name>.resources.limits.memory |
| **Value** | string |
| **Example** | `1G` |
| **Description** | Kubernetes memory limits for a PXC container |
| | |
| **Key** | backup.storages.<storage-name>.nodeSelector |
| **Value** | label |
| **Example** | `disktype: ssd` |
| **Description** | Kubernetes nodeSelector |
| | |
| **Key** | backup.storages.<storage-name>.affinity.nodeAffinity |
| **Value** | subdoc |
| **Example** | |
| **Description** | The Operator node affinity constraint |
| | |
| **Key** | backup.storages.<storage-name>.tolerations |
| **Value** | subdoc |

Continued on next page

Table 12.3 – continued from previous page

| | |
|---|---|
| **Example** | `backupWorker` |
| **Description** | Kubernetes Pod tolerations |
| | |
| **Key** | backup.storages.<storage-name>.priorityClassName |
| **Value** | string |
| **Example** | `high-priority` |
| **Description** | The Kubernetes Pod priority class |
| | |
| **Key** | backup.storages.<storage-name>.schedulerName |
| **Value** | string |
| **Example** | `default-scheduler` |
| **Description** | The Kubernetes Scheduler |
| | |
| **Key** | backup.storages.<storage-name>.containerSecurityContext |
| **Value** | subdoc |
| **Example** | `privileged: true` |
| **Description** | A custom Kubernetes Security Context for a Container to be used instead of the default one |
| | |
| **Key** | backup.storages.<storage-name>.podSecurityContext |
| **Value** | subdoc |
| **Example** | `fsGroup: 1001`<br>`supplementalGroups: [1001, 1002, 1003]` |
| **Description** | A custom Kubernetes Security Context for a Pod to be used instead of the default one |
| | |
| **Key** | backup.schedule.name |
| **Value** | string |
| **Example** | `sat-night-backup` |
| **Description** | The backup name |
| | |
| **Key** | backup.schedule.schedule |
| **Value** | string |
| **Example** | `0 0 * * 6` |
| **Description** | Scheduled time to make a backup specified in the crontab format |
| | |
| **Key** | backup.schedule.keep |
| **Value** | int |
| **Example** | `3` |
| **Description** | Number of stored backups |
| | |
| **Key** | backup.schedule.storageName |
| **Value** | string |
| **Example** | `s3-us-west` |
| **Description** | The name of the storage for the backups configured in the `storages` or `fs-pvc` subsection |

# CHAPTER THIRTEEN

# PROVIDING BACKUPS

Percona XtraDB Cluster Operator allows doing cluster backup in two ways. *Scheduled backups* are configured in the deploy/cr.yaml file to be executed automatically in proper time. *On-demand backups* can be done manually at any moment.

Backup images are usually stored on Amazon S3 or S3-compatible storage (storing backups on private storage is also possible, but they are described separately).

## Making scheduled backups

Since backups are stored separately on the Amazon S3, a secret with `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` should be present on the Kubernetes cluster. The secrets file with these keys should be created: for example `deploy/backup-s3.yaml` file with the following contents:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-cluster-name-backup-s3
type: Opaque
data:
  AWS_ACCESS_KEY_ID: UkVQTEFDRS1XSVRILUFXUy1BQ0NFU1MtS0VZ
  AWS_SECRET_ACCESS_KEY: UkVQTEFDRS1XSVRILUFXUy1TRUNSRVQtS0VZ
```

The `name` value is the Kubernetes secret name which will be used further, and `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` are the keys to access S3 storage (and obviously they should contain proper values to make this access possible). To have effect secrets file should be applied with the appropriate command to create the secret object, e.g. `kubectl apply -f deploy/backup-s3.yaml` (for Kubernetes).

Backups schedule is defined in the `backup` section of the deploy/cr.yaml file. This section contains following subsections: * `storages` subsection contains data needed to access the S3-compatible cloud to store backups. * `schedule` subsection allows to actually schedule backups (the schedule is specified in crontab format).

Here is an example which uses Amazon S3 storage for backups:

```
...
backup:
  enabled: true
  version: 0.3.0
  ...
  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"
     keep: 3
     storageName: s3-us-west
  ...
```

if you use some S3-compatible storage instead of the original Amazon S3, the endpointURL is needed in the *s3* subsection which points to the actual cloud used for backups and is specific to the cloud provider. For example, using Google Cloud involves the following endpointUrl.

The options within these three subsections are further explained in the Operator Options.

The only option which should be mentioned separately is `credentialsSecret` which is a Kubernetes secret for backups. Value of this key should be the same as the name used to create the secret object (`my-cluster-name-backup-s3` in the last example).

The schedule is specified in crontab format as explained in the Operator Options.

## Making on-demand backup

To make on-demand backup, user should use YAML file with correct names for the backup and the PXC Cluster, and correct PVC settings. The example of such file is deploy/backup/backup.yaml.

When the backup config file is ready, actual backup command is executed:

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

**Note:** *Storing backup settings in a separate file can be replaced by passing its content to the ‘‘kubectl apply‘‘ command as follows:*

```
cat <<EOF | kubectl apply -f-
apiVersion: pxc.percona.com/v1
kind: PerconaXtraDBClusterBackup
metadata:
  name: backup1
spec:
  pxcCluster: cluster1
  storageName: fs-pvc
EOF
```

## Restore the cluster from a previously saved backup

Following steps are needed to restore a previously saved backup:

1. First of all make sure that the cluster is running.
2. Now find out correct names for the backup and the cluster. Available backups can be listed with the following command:

```
kubectl get pxc-backup
```

And the following command will list available clusters:

```
kubectl get pxc
```

3. When both correct names are known, the actual restoration process can be started as follows:

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

**Note:** *Storing backup settings in a separate file can be replaced by passing its content to the ‘‘kubectl apply‘‘ command as follows:*

```
cat <<EOF | kubectl apply -f-
apiVersion: "pxc.percona.com/v1"
kind: "PerconaXtraDBClusterRestore"
metadata:
  name: "restore1"
spec:
  pxcCluster: "cluster1"
  backupName: "backup1"
EOF
```

## Delete the unneeded backup

Deleting 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>
```

## 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
```

# CHAPTER FOURTEEN

# LOCAL STORAGE SUPPORT FOR THE PERCONA XTRADB CLUSTER OPERATOR

Among the wide rage of volume types, supported by Kubernetes, there are two which allow Pod containers to access part of the local filesystem on the node. Two such options are *emptyDir* and *hostPath* volumes.

## 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! Is should be created manually and should have following correct attributives: 1. access permissions 2. ownership 3. SELinux security context

`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.

CHAPTER

FIFTEEN

# BINDING PERCONA XTRADB CLUSTER COMPONENTS TO SPECIFIC KUBERNETES/OPENSHIFT NODES

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

Both `pxc` and `proxysql` sections of the deploy/cr.yaml file contain keys which can be used to do this, depending on what is the best for a particular situation.

## Node selector

`nodeSelector` contains one or more key-value pairs. If the node is not labeled with each key-value pair from the Pod's `nodeSelector`, the Pod will not be able to land on it.

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

```
nodeSelector:
  disktype: ssd
```

## Affinity and anti-affinity

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

Percona XtraDB Cluster Operator provides two approaches for doing this:

- simple way to set anti-affinity for Pods, built-in into the Operator,
- more advanced approach based on using standard Kubernetes constraints.

### Simple approach - use topologyKey of the Percona XtraDB Cluster Operator

Percona XtraDB Cluster Operator provides a `topologyKey` option, which may have one of the following values:

- `kubernetes.io/hostname` - Pods will avoid residing within the same host,
- `failure-domain.beta.kubernetes.io/zone` - Pods will avoid residing within the same zone,
- `failure-domain.beta.kubernetes.io/region` - Pods will avoid residing within the same region,

- `none` - no constraints are applied.

The following example forces Percona XtraDB Cluster Pods to avoid occupying the same node:

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

## Advanced approach - use standard Kubernetes constraints

Previous way can be used with no special knowledge of the Kubernetes way of assigning Pods to specific nodes. Still in some cases more complex tuning may be needed. In this case `advanced` option placed in the deploy/cr.yaml file turns off the effect of the `topologyKey` and allows to use standard Kubernetes affinity constraints of any complexity:

```
affinity:
   advanced:
     podAffinity:
       requiredDuringSchedulingIgnoredDuringExecution:
       - labelSelector:
           matchExpressions:
           - key: security
             operator: In
             values:
             - S1
         topologyKey: failure-domain.beta.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 explanation of the advanced affinity options in Kubernetes documentation.

## Tolerations

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

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

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

## Priority Classes

Pods may belong to some *priority classes*. This allows scheduler to distinguish more and less important Pods to resolve the situation when some higher priority Pod cannot be scheduled without evicting a lower priority one. This can be done adding one or more PriorityClasses in your Kubernetes cluster, and specifying the `PriorityClassName` in the deploy/cr.yaml file:

```
priorityClassName: high-priority
```

See the Kubernetes Pods Priority and Preemption documentation to find out how to define and use priority classes in your cluster.

## Pod Disruption Budgets

Creating the *Pod Disruption Budget* is the Kubernetes style to limits the number of Pods of an application that can go down simultaneously due to such *voluntary disruptions* as cluster administrator's actions during the update of deployments or nodes, etc. By such a way Distribution Budgets allow large applications to retain their high availability while maintenance and other administrative activities.

We recommend to apply Pod Disruption Budgets manually to avoid situation when Kubernetes stopped all your database Pods. See the official Kubernetes documentation for details.

CHAPTER

SIXTEEN

# CHANGING MYSQL OPTIONS

You may require a configuration change for your application. MySQL allows the option to configure the database with a configuration file. You can pass the MySQL options from the my.cnf configuration file to the cluster in one of the following ways:

- Edit the CR.yaml file
- Use a ConfigMap

## Edit the CR.yaml

You can add options from the my.cnf by editing the configuration section of the deploy/cr.yaml.

```
spec:
  secretsName: my-cluster-secrets
  pxc:
    ...
      configuration: |
        [mysqld]
        wsrep_debug=ON
        [sst]
        wsrep_debug=ON
```

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
```

## Make changed options visible to the 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 resources (memory and CPU) **if 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 PXC Pod limits are defined, then limits values are used to calculate these options. If PXC Pod limits are not defined, Operator looks for PXC Pod requests as the basis for calculations. if neither PXC Pod limits nor PXC Pod requests are defined, auto-tuning is not done.

CHAPTER

SEVENTEEN

# CONFIGURING PROXYSQL

You can use ProxySQL admin interface to configure its settings.

Configuring ProxySQL in this way means connecting to it using the MySQL protocol, and two things are needed to do it:

- the ProxySQL Pod name
- the ProxySQL admin password

You can find out ProxySQL Pod name with the `kubectl get pods` command, which will have the following output:

```
$ kubectl get pods
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-pxc-proxysql-0                           1/1     Running   0          5m
percona-xtradb-cluster-operator-dc67778fd-qtspz   1/1     Running   0          6m
```

The next command will print you the needed admin password:

```
kubectl get secrets $(kubectl get pxc -o jsonpath='{.items[].spec.secretsName}') -o␣
↪template='{{ .data.proxyadmin | base64decode }}'
```

When both Pod name and admin password are known, connect to the ProxySQL as follows, substituting `cluster1-pxc-proxysql-0` with the actual Pod name and `admin_password` with the actual password:

```
kubectl exec -it cluster1-pxc-proxysql-0 -- mysql -h127.0.0.1 -P6032 -uproxyadmin -
↪padmin_password
```

.

CHAPTER

# EIGHTEEN

# TRANSPORT LAYER SECURITY (TLS)

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

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

The internal certificate is also used as an authorization method.

TLS security can be configured in several ways. By default, the Operator generates certificates automatically if there are no certificate secrets available. Other options are the following ones:

- The Operator can use a specifically installed *cert-manager* for the automatic certificates generation,
- Certificates can be generated manually.

You can also use pre-generated certificates available in the `deploy/ssl-secrets.yaml` file for test purposes, but we strongly recommend

avoiding their usage on any production system!

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



## Install and use the *cert-manager*

### About the *cert-manager*

A *cert-manager* is a Kubernetes certificate management controller which 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* and deploy the operator, the operator requests a certificate from the *cert-manager*. The *cert-manager* acts as a self-signed issuer and generates certificates. The Percona Operator self-signed issuer is local to the operator namespace. This self-signed issuer is created because PXC requires all certificates are issued by the same CA.

The creation of the self-signed issuer allows you to deploy and use the Percona Operator without creating a clusterissuer separately.

### Installation of the *cert-manager*

The steps to install the *cert-manager* are the following:

- Create a namespace
- Disable resource validations on the cert-manager namespace
- Install the cert-manager.

The following commands perform all the needed actions:

```
kubectl create namespace cert-manager
kubectl label namespace cert-manager certmanager.k8s.io/disable-validation=true
kubectl apply -f https://raw.githubusercontent.com/jetstack/cert-manager/release-0.7/
↪deploy/manifests/cert-manager.yaml
```

After the installation, you can 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.

## Generate certificates manually

To generate certificates manually, follow these steps:

1. Provision a Certificate Authority (CA) to generate TLS certificates
2. Generate a CA key and certificate file with the server details
3. Create the server TLS certificates using the CA keys, certs, and server details

The set of commands generate certificates with the following attributes:

- `Server-pem` - Certificate
- `Server-key.pem` - the private key
- `ca.pem` - Certificate Authority

You should generate certificates twice: one set is for external communications, and another set is for internal ones. A secret created for the external use must be added to `cr.yaml/spec/secretsName`. A certificate generated for internal communications must be added to the `cr.yaml/spec/sslInternalSecretName`.

```
cat <<EOF | cfssl gencert -initca - | cfssljson -bare ca
{
  "CN": "Root CA",
  "key": {
    "algo": "rsa",
    "size": 2048
  }
}
```

```
EOF

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

kubectl create secret generic my-cluster-ssl --from-file=tls.crt=server.pem --
from-file=tls.key=server-key.pem --from-file=ca.crt=ca.pem --
type=kubernetes.io/tls
```

## Run PXC without TLS

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

To disable TLS protocol (e.g. for demonstration purposes) edit the `cr.yaml/spec/allowUnstafeConfigurations` setting to `true` and make sure that there are no certificate secrets available.

CHAPTER

NINETEEN

# DATA-AT-REST ENCRYPTION

Full data-at-rest encryption in Percona XtraDB Cluster is supported by the Operator since version 1.4.0.

**Note:** Data at rest means inactive data stored as files, database records, etc.

To implement these features, the Operator uses `keyring_vault` plugin, which ships with Percona XtraDB Cluster, and utilizes HashiCorp Vault storage for encryption keys.



## Installing Vault

The following steps will deploy Vault on Kubernetes with the Helm 3 package manager. Other Vault installation methods should also work, so the instruction placed here is not obligatory and is for illustration purposes.

1. Clone the official HashiCorp Vault Helm chart from GitHub:

```
$ git clone -b v0.4.0 https://github.com/hashicorp/vault-helm.git
$ cd vault-helm
```

2. Now use Helm to do the installation:

```
$ helm install vault-service ./
```

3. After the installation, Vauld should be first initialized and then unsealed. Initializing Vault is done with the following commands:

```
$ kubectl exec -it pod/vault-service-0 -- vault operator init -key-shares=1 -key-
↪threshold=1 -format=json > /tmp/vault-init
$ unsealKey=$(jq -r ".unseal_keys_b64[]" < /tmp/vault-init)
```

To unseal Vault, execute the following command **for each Pod** of Vault running:

```
$ kubectl exec -it pod/vault-service-0 -- vault operator unseal "$unsealKey"
```

## Configuring Vault

1. First, you should enable secrets within Vault. Get the Vault root token:

```
$ cat /tmp/vault-init | jq -r ".root_token"
```

The output will be like follows:

```
s.VgQvaXl8xGFO1RUxAPbPbsfN
```

Now login to Vault with this token and enable the "pxc-secret" secrets path:

```
$ kubectl exec -it vault-service-0 -- /bin/sh
$ vault login s.VgQvaXl8xGFO1RUxAPbPbsfN
$ vault secrets enable --version=1 -path=pxc-secret kv
```

**Note:** You can also enable audit, which is not mandatory, but useful:

```
$ vault audit enable file file_path=/vault/vault-audit.log
```

2. To enable Vault secret within Kubernetes, create and apply the YAML file, as described further.

2.1. To access the Vault server via HTTP, follow the next YAML file example:

```
apiVersion: v1
kind: Secret
metadata:
  name: some-name-vault
type: Opaque
stringData:
  keyring_vault.conf: |-
     token = s.VgQvaXl8xGFO1RUxAPbPbsfN
     vault_url = vault-service.vault-service.svc.cluster.local
     secret_mount_point = pxc-secret
```

**Note:** the `name` key in the above file should be equal to the `spec.vaultSecretName` key from the `deploy/cr.yaml` configuration file.

2.2. To turn on TLS and access the Vault server via HTTPS, you should do two more things:

- add one more item to the secret: the contents of the `ca.cert` file with your certificate,
- store the path to this file in the `vault_ca` key.

```
apiVersion: v1
kind: Secret
metadata:
  name: some-name-vault
type: Opaque
stringData:
  keyring_vault.conf: |-
    token = = s.VgQvaXl8xGFO1RUxAPbPbsfN
    vault_url = https://vault-service.vault-service.svc.cluster.local
    secret_mount_point = pxc-secret
```

```
    vault_ca = /etc/mysql/vault-keyring-secret/ca.cert
  ca.cert: |-
    -----BEGIN CERTIFICATE-----
    MIIEczCCA1ugAwIBAgIBADANBgkqhkiG9w0BAQQFAD..AkGA1UEBhMCR0Ix
    EzARBgNVBAgTClNvbWUtU3RhdGUxFDASBgNVBAoTC0..0EgTHRkMTcwNQYD
    7vQMfXdGsRrXNGRGnX+vWDZ3/zWI0joDtCkNnqEpVn..HoX
    -----END CERTIFICATE-----
```

**Note:** the `name` key in the above file should be equal to the `spec.vaultSecretName` key from the `deploy/cr.yaml` configuration file.

**Note:** For techincal reasons the `vault_ca` key should either exist or not exist in the YAML file; commented option like `#vault_ca = ...` is not acceptable.

More details on how to install and configure Vault can be found in the official documentation.

## Using the encryption

If using *Percona XtraDB Cluster* 5.7, you should turn encryption on explicitly when you create a table or a tablespace. This can be done by adding the `ENCRYPTION='Y'` part to your SQL statement, like in the following example:

```
CREATE TABLE t1 (c1 INT, PRIMARY KEY pk(c1)) ENCRYPTION='Y';
CREATE TABLESPACE foo ADD DATAFILE 'foo.ibd' ENCRYPTION='Y';
```

**Note:** See more details on encryption in Percona XtraDB Cluster 5.7 here.

If using *Percona XtraDB Cluster* 8.0, the encryption is turned on by default. The following table presents the default values of the correspondent my.cnf configuration options:

| Option | Default value |
|---|---|
| early-plugin-load | keyring_vault.so |
| keyring_vault_config | /etc/mysql/vault-keyring-secret/keyring_vault.conf |
| default_table_encryption | ON |
| table_encryption_privilege_check | ON |
| innodb_undo_log_encrypt | ON |
| innodb_redo_log_encrypt | ON |
| binlog_encryption | ON |
| binlog_rotate_encryption_master_key_at_startup | ON |
| innodb_temp_tablespace_encrypt | ON |
| innodb_parallel_dblwr_encrypt | ON |
| innodb_encrypt_online_alter_logs | ON |
| encrypt_tmp_files | ON |

CHAPTER

# TWENTY

# PAUSE/RESUME PERCONA XTRADB CLUSTER

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

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

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

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

CHAPTER

TWENTYONE

# 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 Percona XtraDB Cluster provides two ways of recovery after a full cluster crash.

- The automated *Bootstrap Crash Recovery method* is the simplest one, but it may cause loss of several recent transactions.
- The manual *Object Surgery Crash Recovery method* includes a lot of operations, but it allows to restore all the data.

## Bootstrap Crash Recovery method

In this case recovery is done automatically. The recovery is triggered by the `pxc.forceUnsafeBootstrap` option set to `true` in the `deploy/cr.yaml` file:

```
pxc:
  ...
  forceUnsafeBootstrap: true
```

Applying this option forces the cluster to start. However, there may exist data inconsistency in the cluster, and several last transactions may be lost. If such data loss is undesirable, experienced users may choose the more advanced manual method described in the next chapter.

# Object Surgery Crash Recovery method

**Warning:** This method is intended for advanced users only!

This method involves the following steps: * swap the original PXC 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 PXC 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 PXC Pods.

**Note:** The following commands are written for PXC 8.0. The same steps are also for PXC 5.7 unless specifically indicated otherwise.

1. Change the normal PXC image inside the cluster object to the debug image:

```
$ kubectl patch pxc cluster1 --type="merge" -p '{"spec":{"pxc":{"image":"percona/
↪percona-xtradb-cluster-operator:1.4.0-pxc8.0-debug"}}}'
```

**Note:** For PXC 5.7 this command should be as follows:

```
$ kubectl patch pxc cluster1 --type="merge" -p '{"spec":{"pxc":{"image":"percona/
↪percona-xtradb-cluster-operator:1.4.0-pxc5.7-debug"}}}'
```

2. 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
```

3. 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
```

4. Wait for all PXC Pods to start, then find the PXC 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 PXC image because the debug image is no longer needed:

```
$ kubectl patch pxc cluster1 --type="merge" -p '{"spec":{"pxc":{"image":"percona/
↪percona-xtradb-cluster-operator:1.4.0-pxc8.0"}}}'
```

**Note:** For PXC 5.7 this command should be as follows:

```
$ kubectl patch pxc cluster1 --type="merge" -p '{"spec":{"pxc":{"image":"percona/
↪percona-xtradb-cluster-operator:1.4.0-pxc5.7"}}}'
```

7. 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
```

8. 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
```

9. After the Pod startup, the cluster is fully recovered.

CHAPTER

# TWENTYTWO

# DEBUG

For the cases when Pods are failing for some reason or just show abnormal behavior, the Operator can be used with a special *debug image* of the Percona XtraDB Cluster, which 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.

Particularly, using this image is useful if the container entry point fails (`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.

To use the debug image instead of the normal one, set the following image name for the `pxc.image` key in the `deploy/cr.yaml` configuration file:

- `percona/percona-xtradb-cluster-operator:1.4.0-pxc8.0-debug` for PXC 8.0,
- `percona/percona-xtradb-cluster-operator:1.4.0-pxc5.7-debug` for PXC 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.

# Part IV

# Reference

CHAPTER

TWENTYTHREE

# PERCONA KUBERNETES OPERATOR FOR PERCONA XTRADB CLUSTER 1.4.0 RELEASE NOTES

## *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't 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-master 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-master 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-master 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.

# INDEX

## Symbols