<table>
<thead>
<tr>
<th>CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I  Requirements</td>
<td>3</td>
</tr>
<tr>
<td>II  Installation</td>
<td>11</td>
</tr>
<tr>
<td>III  Configuration</td>
<td>33</td>
</tr>
<tr>
<td>IV  Reference</td>
<td>59</td>
</tr>
</tbody>
</table>
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 a 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
The following platforms are supported:

- OpenShift >=3.11
- Google Kubernetes Engine (GKE)
- Minikube
**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 across 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
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-1.2.0 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 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. Install cert-manager if it is not up and running yet then generate and apply certificates as secrets according to TLS document `<TLS.html>`:

Pre-generated certificates are available in the `deploy/ssl-secrets.yaml` secrets file for test purposes, but we strongly recommend avoiding their usage on any production system.

```
$ kubectl apply -f <secrets file>
```

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
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster1-pxc-node-0</td>
<td>1/1</td>
<td>Running 0</td>
<td>0</td>
<td>5m</td>
</tr>
<tr>
<td>cluster1-pxc-node-1</td>
<td>1/1</td>
<td>Running 0</td>
<td>0</td>
<td>4m</td>
</tr>
<tr>
<td>cluster1-pxc-node-2</td>
<td>1/1</td>
<td>Running 0</td>
<td>0</td>
<td>2m</td>
</tr>
<tr>
<td>cluster1-pxc-proxysql-0</td>
<td>1/1</td>
<td>Running 0</td>
<td>0</td>
<td>5m</td>
</tr>
<tr>
<td>percona-xtradb-cluster-operator-dc67778fd-qtspz</td>
<td>1/1</td>
<td>Running 0</td>
<td>0</td>
<td>6m</td>
</tr>
</tbody>
</table>

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

```
$ git clone -b release-1.2.0 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.

An extra action is needed if you want to manage PXC cluster from 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,perconaxtradbclusterbackups.pxc.percona.com,perconaxtradbclusterclusterrestores.pxc.percona.com,perconaxtradbclusterclusterreleases.pxc.percona.com/status,issuers.certmanager.k8s.io,certificates.certmanager.k8s.io
$ oc adm policy add-cluster-role-to-user pxc-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. Install cert-manager if it is not up and running yet then generate and apply certificates as secrets according to TLS document <TLS.html>:

Pre-generated certificates are available in the deploy/ssl-secrets.yaml secrets file for test purposes, but we strongly recommend avoiding their usage on any production system. .. code:: bash

$ oc apply -f <secrets file>

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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster1-pxc-node-0</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>5m</td>
</tr>
<tr>
<td>cluster1-pxc-node-1</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>4m</td>
</tr>
<tr>
<td>cluster1-pxc-node-2</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>2m</td>
</tr>
<tr>
<td>cluster1-pxc-proxysql-0</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>5m</td>
</tr>
<tr>
<td>percona-xtradb-cluster-operator-dc67778fd-qtspz</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>6m</td>
</tr>
</tbody>
</table>

7. Check connectivity to newly created cluster

$ oc run -i --rm --tty percona-client --image=percona:5.7 --restart=Never -- bash

percona-client:/$ mysql -h cluster1-proxysql -uroot -proot_password
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: 
   1. `kubectl` tool, 
   2. a hypervisor, if it is not already installed, 
   3. actual Minikube package

After the installation running `minikube start` should 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-1.2.0 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. Edit the `deploy/cr.yaml` file to change the following keys in `pxc` and `proxysql` sections, which would otherwise prevent running Percona XtraDB Cluster on your local Kubernetes installation:

   (a) comment `resources.requests.memory` and `resources.requests.cpu` keys

   (b) set `affinity.antiAffinityTopologyKey` key to "none"

Also, switch `allowUnsafeConfigurations` key to true.

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:
```
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):

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

1. Extract and backup the yaml file for the cluster

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

2. Delete the cluster

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

3. 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
```
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 recommended way for a production cluster.

**Note:** Only the incremental update to a nearest minor version is supported (for example, update from 1.1.0 to 1.2.0). 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 1.2.0 version should look as follows:

   ```bash
   kubectl patch deployment percona-xtradb-cluster-operator \
   -p'{"spec":{"template":{"spec":{"containers":[{"name":"percona-xtradb-cluster-operator","image":"percona/percona-xtradb-cluster-operator:1.2.0"}]}}}}'
   kubectl patch pxc cluster1 --type=merge --patch '{
   "metadata": {"annotations": {"kubectl.kubernetes.io/last-applied-configuration": "/apiVersion": "pxc.percona.com/v1-2-0"}},
   "spec": {"pxc": {"image": "percona/percona-xtradb-cluster-operator:1.2.0-pxc"}}
   }'
   kubectl patch proxy sql cluster1 --type=merge --patch '{
   "proxysql": { "image": "percona/percona-xtradb-cluster-operator:1.2.0-proxy sql" },
   "backup": { "image": "percona/percona-xtradb-cluster-operator:1.2.0-backup" },
   "pmm": { "image": "percona/percona-xtradb-cluster-operator:1.2.0-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:

   ```bash
   kubectl rollout status sts cluster1-pxc
   ```
Manuol 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 1.2.0 version should look as follows:

   ```bash
   kubectl patch deployment percona-xtradb-cluster-operator \\
   -p'"spec":{"template":{"spec":{"containers":[{"name":"percona-xtradb-cluster-operator","image":"percona/percona-xtradb-cluster-operator:1.2.0"}]}}}}'

   kubectl patch pxc cluster1 --type=merge --patch '\
   "metadata": {
   "annotations": {  "kubectl.kubernetes.io/last-applied-configuration": "{"apiVersion": "pxc.percona.com/v1-2-0"}" },
   "spec": { "pxc": { "image": "percona/percona-xtradb-cluster-operator:1.2.0-pxc" } },
   "proxysql": { "image": "percona/percona-xtradb-cluster-operator:1.2.0-proxysql" },
   "backup": { "image": "percona/percona-xtradb-cluster-operator:1.2.0-backup" },
   "pmm": { "image": "percona/percona-xtradb-cluster-operator:1.2.0-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:

   ```bash
   kubectl delete pod cluster1-pxc-2
   
   (b) Wait until Pod becomes ready:

   ```bash
   kubectl get pod cluster1-pxc-2
   
   The output should be like this:

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster1-pxc-2</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>3m33s</td>
</tr>
</tbody>
</table>

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

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

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

   ```bash
   $ helm install percona/pmm-server --name monitoring --set platform=openshift --
   →set credentials.username=pmm --set "credentials.password=supaj^|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:

```sh
$ 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):

```sh
$ 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.
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
   ADO8CqCDappWR4hxjfDqwijEHei31yXAqVq6lgJg210S
   ```

   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 ADO8CqCDappWR4hxjfDqwijEHei31yXAqVq6lgJg210S 172.30.162.173:5000
   Login Succeeded
   ```

4. Pull the needed image by its SHA digest:

   ```
   $ docker pull docker.io/percona/percona-xtradb-cluster-operator@sha256:8895ff4647602dbbabf6e5d1be1611e9d7a9769c3bb3415c3a73aba2adda0
   Trying to pull repository docker.io/percona/percona-xtradb-cluster-operator ...
   sha256:8895ff4647602dbbabf6e5d1be1611e9d7a9769c3bb3415c3a73aba2adda0: Pulling from docker.io/percona/percona-xtradb-cluster-operator
   ```
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@sha256:8895ff4647602dcbcaabf6ea5d1be1611e9d7a9769c3bb3415c3a73aba2adda0 \
  172.30.162.173:5000/pxc/percona-xtradb-cluster-operator:0.3.0
$ docker push 172.30.162.173:5000/pxc/percona-xtradb-cluster-operator:0.3.0
```

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                      0.3.0       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:0.3.0`) 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.


**Percona certified images**

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

### 0.3.0

<table>
<thead>
<tr>
<th>Image</th>
<th>Digest</th>
</tr>
</thead>
<tbody>
<tr>
<td>percona/percona-xtradb-cluster-operator:0.3.0</td>
<td>f4a0d604bb13678cbed72fd26d1b2a287a</td>
</tr>
<tr>
<td>percona/percona-xtradb-cluster-operator:0.3.0-pxc</td>
<td>51a478ff24e6e1631f0e00c7c8b372ad589</td>
</tr>
<tr>
<td>percona/percona-xtradb-cluster-operator:0.3.0-proxysql</td>
<td>673b954ecc795ca457102a6e2f8a9ab389</td>
</tr>
<tr>
<td>percona/percona-xtradb-cluster-operator:0.3.0-backup</td>
<td>a205e8f8699337e3e95d9b9cfc306b8b7183</td>
</tr>
<tr>
<td>perconalab/pmm-client:1.17.1</td>
<td>f762cda2eda9f17bf1242ede70ee72595</td>
</tr>
</tbody>
</table>

26 Chapter 9. Use docker images from a custom registry
## 0.2.0

<table>
<thead>
<tr>
<th>Image</th>
<th>Digest</th>
</tr>
</thead>
<tbody>
<tr>
<td>perconalab/percona-xtradb-cluster-operator:0.2.0</td>
<td>889f4f464762dcbbabf6a5d1be1611e9d7a9769c3bb3415c3a73aba2adda0</td>
</tr>
<tr>
<td>perconalab/pxc-openshift:0.2.0</td>
<td>a9f6568cc71e7eb5bbfe69b3ca561e23bae92a75caba7f0fa88bd3e730be9</td>
</tr>
<tr>
<td>perconalab/proxysql-openshift:0.2.0</td>
<td>c0dd114b82f34312e73419282a695063387c715d3e80677902938f991ef94f13</td>
</tr>
<tr>
<td>perconalab/backupjob-openshift:0.2.0</td>
<td>1ded5511a59fc2cc5a6b23234495e6d243d5f8b55e1b6061781779e19887c9c9</td>
</tr>
<tr>
<td>perconalab/pmm-client:1.17.0</td>
<td>efdce369d5fb29b0a1b03a7026dfbc2efe07b618471aba5db308d0c21b8e118d</td>
</tr>
</tbody>
</table>

## 0.1.0

<table>
<thead>
<tr>
<th>Image</th>
<th>Digest</th>
</tr>
</thead>
<tbody>
<tr>
<td>perconalab/percona-xtradb-cluster-operator:0.1.0</td>
<td>9e4b44ef6859e995d70c0ef7db9be9b9c2875d1116a2b6ff7e5a7f5e5feb39b7</td>
</tr>
<tr>
<td>perconalab/pxc-openshift:0.1.0</td>
<td>c72e845c3f103f105f86f405668a26b22990d6a3ba9c8a1d0467040c6c83f3e53</td>
</tr>
<tr>
<td>perconalab/proxysql-openshift:0.1.0</td>
<td>4826f4161aafe78585b3e377a4aef9a83f4e4860e0bd8576f0e39ee52909d</td>
</tr>
<tr>
<td>perconalab/pmm-client:1.17.0</td>
<td>efdce369d5fb29b0a1b03a7026dfbc2efe07b618471aba5db308d0c21b8e118d</td>
</tr>
</tbody>
</table>
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:

   ```bash
   $ 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:

   ```bash
   $ oc apply -f https://raw.githubusercontent.com/Percona-Lab/percona-dbaas-cli/master/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
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.secrets` section.

### Unprivileged users

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

```shell
$ kubectl run -it --rm percona-client --image=percona:5.7 --restart=Never -- mysql -h cluster1-pxc -uroot -proot_password
```

```
mysql> GRANT ALL PRIVILEGES ON database1.* TO 'user1'@'%' IDENTIFIED BY 'password1';
```

Sync users on the ProxySQL node:

```shell
$ kubectl exec -it cluster1-pxc-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:

```shell
$ kubectl run -it --rm percona-client --image=percona:5.7 --restart=Never -- bash -il
```

```
percona-client:/$ mysql -h cluster1-pxc -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

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

### 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:

<table>
<thead>
<tr>
<th>Secret Key</th>
<th>Secret Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>root_password</td>
</tr>
<tr>
<td>xtrabackup</td>
<td>backup_password</td>
</tr>
<tr>
<td>monitor</td>
<td>monitor</td>
</tr>
<tr>
<td>clustercheck</td>
<td>clustercheckpassword</td>
</tr>
<tr>
<td>proxyuser</td>
<td>s3cret</td>
</tr>
<tr>
<td>proxyadmin</td>
<td>admin_password</td>
</tr>
<tr>
<td>pmmserver</td>
<td>supa</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pxc</td>
<td>subdoc</td>
<td>Percona XtraDB Cluster general section</td>
</tr>
<tr>
<td>proxysql</td>
<td>subdoc</td>
<td>ProxySQL section</td>
</tr>
<tr>
<td>pmm</td>
<td>subdoc</td>
<td>Percona Monitoring and Management section</td>
</tr>
<tr>
<td>backup</td>
<td>subdoc</td>
<td>Percona XtraDB Cluster backups section</td>
</tr>
</tbody>
</table>

**PXC Section**

The `pxc` section in the `deploy/cr.yaml` file contains general configuration options for the Percona XtraDB Cluster.
Table 12.2: PXC Section

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>int</td>
<td>3</td>
<td>The size of the Percona XtraDB cluster must be &gt;= 3 for High Availability</td>
</tr>
<tr>
<td>allowUnsafeConfigurations</td>
<td>string</td>
<td>false</td>
<td>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”</td>
</tr>
<tr>
<td>image</td>
<td>string</td>
<td>percona/percona-xtradb-cluster-operator:1.0.0-pxc</td>
<td>The Docker image of the Percona cluster used.</td>
</tr>
<tr>
<td>readinessDelay-Sec</td>
<td>int</td>
<td>15</td>
<td>Adds a delay before a run check to verify the application is ready to process traffic</td>
</tr>
<tr>
<td>livenessDelay-Sec</td>
<td>int</td>
<td>300</td>
<td>Adds a delay before the run check ensures the application is healthy and capable of processing requests</td>
</tr>
<tr>
<td>forceUnsafeBootstrap</td>
<td>string</td>
<td>false</td>
<td>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.</td>
</tr>
<tr>
<td>configuration</td>
<td>string</td>
<td></td>
<td>mysql [mysqld] wsrep_debug=ON wsrep_provider_options=gcache.size=1G;gcache.recover=yes</td>
</tr>
<tr>
<td>imagePullSecrets.name</td>
<td>string</td>
<td>private-registry-credentials</td>
<td>The Kubernetes ImagePullSecret</td>
</tr>
<tr>
<td>priorityClassName</td>
<td>string</td>
<td>high-priority</td>
<td>The Kubernetes Pod priority class</td>
</tr>
<tr>
<td>annotations</td>
<td>label</td>
<td>iam.amazonaws.com/role:role-arn</td>
<td>The Kubernetes annotations</td>
</tr>
<tr>
<td>labels</td>
<td>label</td>
<td>rack: rack-22</td>
<td>Labels are key-value pairs attached to objects.</td>
</tr>
<tr>
<td>resources.requests.memory</td>
<td>string</td>
<td>1G</td>
<td>The Kubernetes memory requests for a PXC container.</td>
</tr>
<tr>
<td>resources.requests.cpu</td>
<td>string</td>
<td>600m</td>
<td>Kubernetes CPU requests for a PXC container.</td>
</tr>
<tr>
<td>resources.limits.memory</td>
<td>string</td>
<td>1G</td>
<td>Kubernetes memory limits for a PXC container.</td>
</tr>
<tr>
<td>nodeSelector</td>
<td>label</td>
<td>disktype: ssd</td>
<td>Kubernetes nodeSelector</td>
</tr>
<tr>
<td>affinity.topologyKey</td>
<td>string</td>
<td>kubernetes.io/hostname</td>
<td>The Operator topology key <a href="https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity">https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity</a> node anti-affinity constraint</td>
</tr>
<tr>
<td>affinity.advanced</td>
<td>subdoc</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>affinity.tolerations</td>
<td>subdoc</td>
<td>node.alpha.kubernetes.io/unreachable</td>
<td>Kubernetes pod tolerations</td>
</tr>
<tr>
<td>podDisruptionBudget.maxUnavailable</td>
<td>int</td>
<td>1</td>
<td>The Kubernetes podDisruptionBudget specifies the number of pods from the set unavailable after the eviction.</td>
</tr>
<tr>
<td>podDisruptionBudget.minAvailable</td>
<td>int</td>
<td>0</td>
<td>The Kubernetes podDisruptionBudget defines the number of pods that must be available after an eviction.</td>
</tr>
</tbody>
</table>
ProxySQL Section

The `proxysql` section in the `deploy/cr.yaml` file contains configuration options for the ProxySQL daemon.
Table 12.3: proxysql Section

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>true</td>
<td>Enables or disables load balancing with ProxySQL Services</td>
</tr>
<tr>
<td>size</td>
<td>int</td>
<td>1</td>
<td>The number of the ProxySQL daemons to provide load balancing must be 1 in current release.</td>
</tr>
<tr>
<td>image</td>
<td>string</td>
<td><code>percona/percona-xtradb-cluster-operator:1.0-proxysql</code></td>
<td>ProxySQL Docker image to use.</td>
</tr>
<tr>
<td>imagePullSecrets.name</td>
<td>string</td>
<td><code>private-registry-credentials</code></td>
<td>The Kubernetes imagePullSecrets for the ProxySQL image.</td>
</tr>
<tr>
<td>annotations</td>
<td>label</td>
<td><code>iam.amazonaws.com/role:</code> <code>role-arn</code></td>
<td>Kubernetes annotations metadata.</td>
</tr>
<tr>
<td>labels</td>
<td>label</td>
<td><code>rack: rack-22</code></td>
<td>Labels are key-value pairs attached to objects.</td>
</tr>
<tr>
<td>servicetype</td>
<td>string</td>
<td><code>ClusterIP</code></td>
<td>Specifies the type of Kubernetes Service to be used.</td>
</tr>
<tr>
<td>resources.requests.memory</td>
<td>string</td>
<td><code>1G</code></td>
<td>Kubernetes memory requests for a ProxySQL container.</td>
</tr>
<tr>
<td>resources.requests.cpu</td>
<td>string</td>
<td><code>600m</code></td>
<td>Kubernetes CPU requests for a ProxySQL container.</td>
</tr>
<tr>
<td>resources.limits.memory</td>
<td>string</td>
<td><code>1G</code></td>
<td>Kubernetes memory limits for a ProxySQL container.</td>
</tr>
<tr>
<td>resources.limits.cpu</td>
<td>string</td>
<td><code>700m</code></td>
<td>Kubernetes CPU limits for a ProxySQL container.</td>
</tr>
<tr>
<td>priorityClassName</td>
<td>string</td>
<td><code>high-priority</code></td>
<td>The Kubernetes Pod Priority class for ProxySQL.</td>
</tr>
<tr>
<td>nodeSelector</td>
<td>label</td>
<td><code>disktype: ssd</code></td>
<td>Kubernetes nodeSelector</td>
</tr>
<tr>
<td>affinity.topologyKey</td>
<td>string</td>
<td><code>kubernetes.io/hostname</code></td>
<td>The Operator topology key <a href="https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity">https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity</a> node anti-affinity constraint</td>
</tr>
<tr>
<td>affinity.advanced</td>
<td>subdoc</td>
<td><code>node.alpha.kubernetes.io/unreachable</code></td>
<td>If available it makes a topologyKey node affinity constraint to be ignored.</td>
</tr>
<tr>
<td>affinity.tolerations</td>
<td>subdoc</td>
<td></td>
<td>Kubernetes pod tolerations</td>
</tr>
<tr>
<td>volume-Spec.emptyDir</td>
<td>string</td>
<td><code>{}</code></td>
<td>Kubernetes emptyDir volume The directory created on a node and accessible to the PXC pod containers.</td>
</tr>
<tr>
<td>volume-Spec.hostPath.path</td>
<td>string</td>
<td><code>/data</code></td>
<td>Kubernetes hostPath The volume that mounts a directory from the host node’s filesystem into your pod.</td>
</tr>
<tr>
<td>volume-Spec.hostPath.type</td>
<td>string</td>
<td><code>Directory</code></td>
<td>Kubernetes hostPath An optional property for the hostPath.</td>
</tr>
<tr>
<td>volume-Spec.persistentVolumeClaim.storageClassName</td>
<td>string</td>
<td><code>standard</code></td>
<td>Set the Kubernetes storage class to use with the PXC PersistentVolumeClaim.</td>
</tr>
<tr>
<td>volume-Spec.PersistentVolumeClaim.accessModes</td>
<td>array</td>
<td>[ReadWriteOnce]</td>
<td>The Kubernetes PersistentVolumeClaim access modes for the Percona XtraDB cluster.</td>
</tr>
<tr>
<td>volume-Spec.resources.requests.storage</td>
<td>string</td>
<td><code>6Gi</code></td>
<td>The Kubernetes PersistentVolumeClaim size for the Percona XtraDB cluster.</td>
</tr>
<tr>
<td>podDisruption-Budget.maxUnavailable</td>
<td>int</td>
<td>1</td>
<td>Kubernetes podDisruptionBudget specifies the number of pods from the set unavailable after the eviction.</td>
</tr>
<tr>
<td>podDisruption-Budget.minAvailable</td>
<td>int</td>
<td>0</td>
<td>Kubernetes podDisruptionBudget the number of pods that must be available after an eviction.</td>
</tr>
<tr>
<td>gracePeriod</td>
<td>int</td>
<td>30</td>
<td>The Kubernetes grace period when terminating a pod.</td>
</tr>
</tbody>
</table>
PMM Section

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>false</td>
<td>Enables or disables monitoring Percona XtraDB cluster with PMM</td>
</tr>
<tr>
<td>image</td>
<td>string</td>
<td>percona/lab/pmm-client:1.17.1</td>
<td>PMM client Docker image to use.</td>
</tr>
<tr>
<td>serverHost</td>
<td>string</td>
<td>monitoring-service</td>
<td>Address of the PMM Server to collect data from the cluster.</td>
</tr>
<tr>
<td>serverUser</td>
<td>string</td>
<td>pmm</td>
<td>The PMM Serve_User. The PMM Server password should be configured using Secrets.</td>
</tr>
</tbody>
</table>

Backup Section

The `backup` section in the `deploy/cr.yaml` file contains the following configuration options for the regular Percona XtraDB Cluster backups.
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>string</td>
<td>percona-xtradb-cluster:1.0.0-backup</td>
<td>The Percona XtraDB cluster Docker image to use for the backup.</td>
</tr>
<tr>
<td>imagePullSecrets.name</td>
<td>string</td>
<td>private-registry-credentials</td>
<td>The Kubernetes imagePullSecrets for the specified image.</td>
</tr>
<tr>
<td>storages.type</td>
<td>string</td>
<td>s3</td>
<td>The cloud storage type used for backups. Only s3 and filesystem types are supported.</td>
</tr>
<tr>
<td>storages.s3.credentialsSecret</td>
<td>string</td>
<td>my-cluster-name-backup</td>
<td>The Kubernetes secret for backups. It should contain AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY keys.</td>
</tr>
<tr>
<td>storages.s3.bucket</td>
<td>string</td>
<td></td>
<td>The Amazon S3 bucket name for backups.</td>
</tr>
<tr>
<td>storages.s3.region</td>
<td>string</td>
<td>us-east-1</td>
<td>The AWS region to use. Please note <strong>this option is mandatory</strong> for Amazon and all S3-compatible storages.</td>
</tr>
<tr>
<td>storages.s3.endpointUrl</td>
<td>string</td>
<td></td>
<td>The endpoint URL of the S3-compatible storage to be used (not needed for the original Amazon S3 cloud).</td>
</tr>
<tr>
<td>storages.persistentVolumeClaim.type</td>
<td>string</td>
<td>filesystem</td>
<td>The persistent volume claim storage type</td>
</tr>
<tr>
<td>storages.persistentVolumeClaim.storageClassName</td>
<td>string</td>
<td>standard</td>
<td>Set the Kubernetes Storage Class to use with the PXC backups PersistentVolumeClaims for the filesystem storage type.</td>
</tr>
<tr>
<td>storages.persistentVolumeClaim.accessModes</td>
<td>array</td>
<td>[ReadWriteOne]</td>
<td>The Kubernetes PersistentVolume access modes</td>
</tr>
<tr>
<td>storages.persistentVolume</td>
<td>string</td>
<td>6Gi</td>
<td>Storage size for the PersistentVolume.</td>
</tr>
<tr>
<td>schedule.name</td>
<td>string</td>
<td>sat-night-backup</td>
<td>The backup name</td>
</tr>
<tr>
<td>schedule.schedule</td>
<td>string</td>
<td>0 0 * * 6</td>
<td>Scheduled time to make a backup specified in the crontab format</td>
</tr>
<tr>
<td>schedule.keep</td>
<td>int</td>
<td>3</td>
<td>Number of stored backups</td>
</tr>
<tr>
<td>schedule.storageName</td>
<td>string</td>
<td>s3-us-west</td>
<td>The name of the storage for the backups configured in the storages or fs-pvc subsection.</td>
</tr>
</tbody>
</table>
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:

```yaml
apiVersion: v1
cr:
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:

```yaml
... 
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:
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
```
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):

```yaml
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.
BINDING PERCONA XTRADB CLUSTER COMPONENTS TO SPECIFIC KUBERNETES/OPENShift NODES

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

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

```yaml
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:

```yaml
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:

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

The Kubernetes Taints and Tolerations 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:

```yaml
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.
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: * CR.yaml * ConfigMap

**Edit the CR.yaml**

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

```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:

```ini
[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:
Percona Kubernetes Operator for Percona XtraDB Cluster, Release 1.2.0

```
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).
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 two ways: Percona XtraDB Cluster Operator can use a **cert-manager** for automatic certificates generation, but also supports manual certificates generation. The following subsections cover these two ways to configure TLS security with the Operator, as well as explains how to temporarily disable it if needed.

- Install and use the **cert-manager**
  - **About the cert-manager**
  - **Installation of the cert-manager**
- Generate certificates manually
- Run PXC without TLS

**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": {
```

Chapter 17. Transport Layer Security (TLS)
Run PXC without TLS

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

TLS protocol can be disabled (e.g. for demonstration purposes) by editing the cr.yaml/spec/pxc/allowUnstableConfigurations setting to true.
Part IV

Reference
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.
Symbols
1.0.0 (release notes), 62
1.1.0 (release notes), 61
1.2.0 (release notes), 61