Percona Operator for MongoDB documentation

1.15.0 (October 09, 2023)

Percona Technical Documentation Team

Table of contents

1. Percona Operator for MongoDB

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

The Percona Kubernetes Operators are based on best practices for the configuration of a Percona Server for MongoDB replica set. The Operator provides many benefits but saving time, a consistent environment are the most important.

2. Features

- Design and architecture
- Comparison with other solutions

3. Quickstart

- Install with Helm
- Install with kubectl

4. Installation

- System Requirements
- Install on Minikube
- Install on Google Kubernetes Engine (GKE)
- Install on Amazon Elastic Kubernetes Service (AWS EKS)
- Install on Microsoft Azure Kubernetes Service (AKS)
- Generic Kubernetes installation
- Install on OpenShift

5. Configuration

- Application and system users
- Changing MongoDB options
- Anti-affinity and tolerations
- Labels and annotations
- Exposing the cluster
- Local storage support
- Arbiter and non-voting nodes
- MongoDB sharding
- Transport encryption (TLS/SSL)
- Data at rest encryption
- Telemetry

6. Management

7. HOWTOs

8. Reference

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

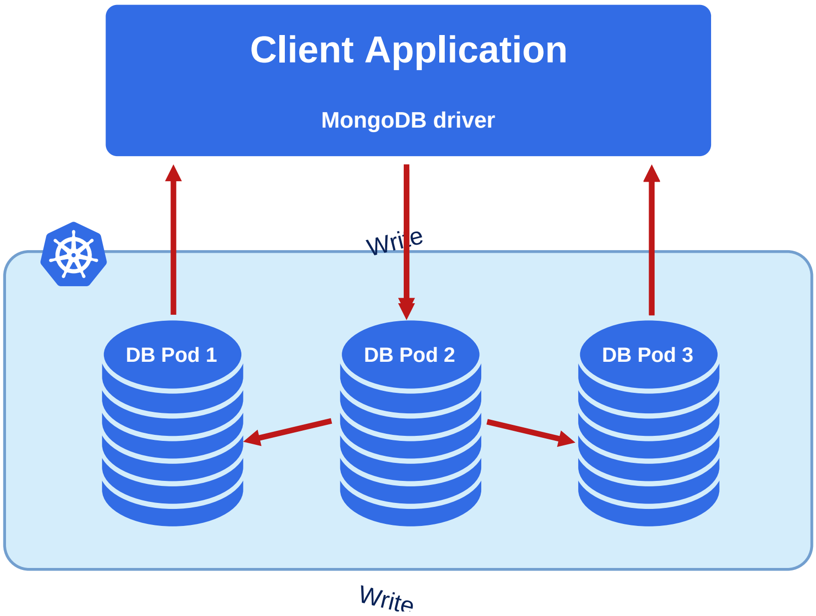
For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-31

9. Features

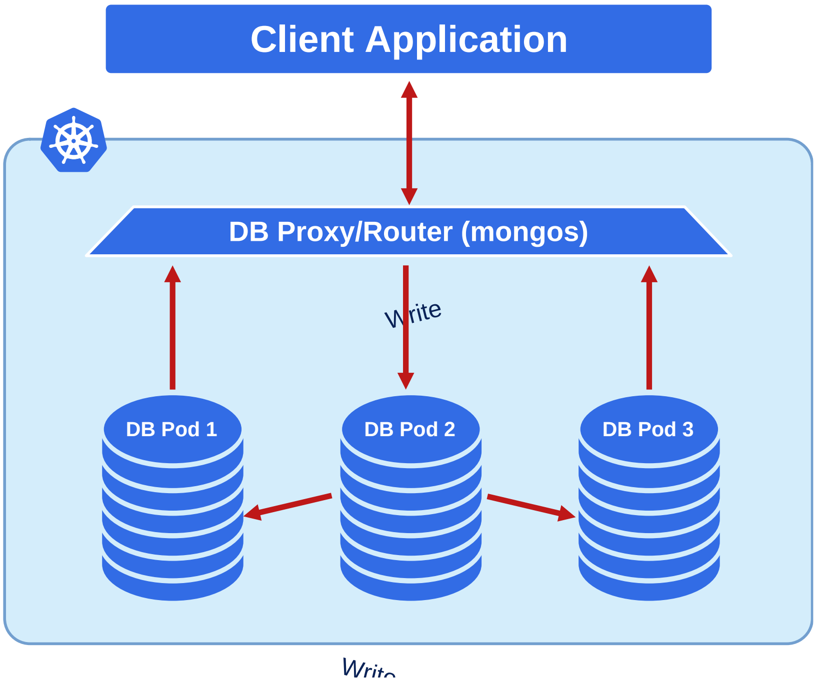
9.1 Design overview

The design of the Operator is tighly bound to the Percona Server for MongoDB replica set or sharded cluster. Replica set cluster is briefly described in the following diagram.



A replica set consists of one primary server and several secondary ones (two in the picture), and the client application accesses the servers via a driver.

In the case of a sharded cluster, each shard is a replica set which contains a subset of data stored in the database, and the `mongos` query router acts as an entry point for client applications. You can find out more details about sharding on a dedicated documentation page, and a simplified diagram is as follows:

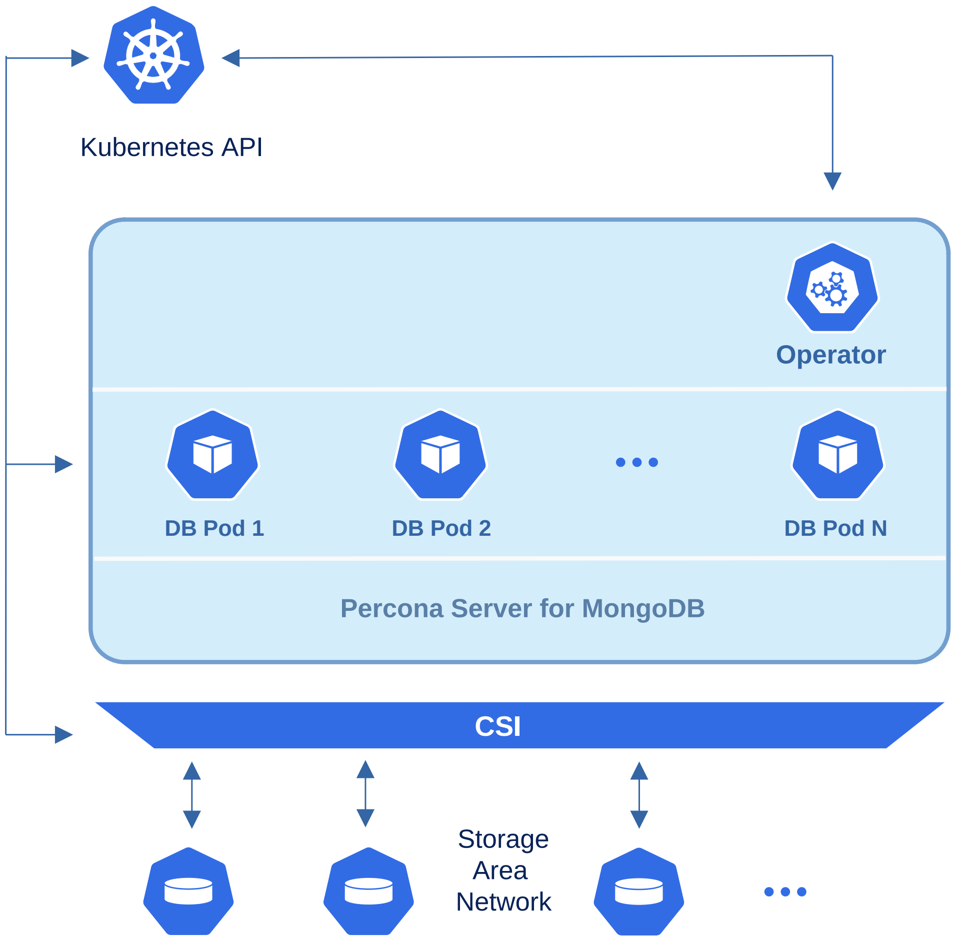


To provide high availability the Operator uses node affinity to run MongoDB instances on separate worker nodes if possible, and the database cluster is deployed as a single Replica Set with at least three nodes. If a node fails, the pod with the mongod process is automatically re-created on another node. If the failed node was hosting the primary server, the replica set initiates elections to select a new primary. If the failed node was running the Operator, Kubernetes will restart the Operator on another node, so normal operation will not be interrupted.

Client applications should use a mongo+srv URI for the connection. This allows the drivers (4.2 and up) to retrieve the list of replica set members from DNS SRV entries without having to list hostnames for the dynamically assigned nodes.

Note

The Operator uses security settings which are more secure than the default Percona Server for MongoDB setup. The initial configuration contains default passwords for all needed user accounts, which should be changed in the production environment, as stated in the installation instructions.



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 *PerconaServerMongoDB* object, and it is implemented as a golang application. Each *PerconaServerMongoDB* object maps to one separate Percona Server for MongoDB setup. The Operator listens to all events on the created objects. When a new PerconaServerMongoDB 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 operating replica set.

Last update: 2023-02-16

9.2 Compare various solutions to deploy MongoDB in Kubernetes

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

- Bitnami Helm chart
- KubeDB
- MongoDB Community Operator
- Percona Operator for MongoDB

9.2.1 Generic

Here is the review of generic features, such as supported MongoDB versions, open source models and more.

Feature/ Product	Percona Operator for MongoDB	Bitnami Helm Chart	KubeDB for MongoDB	MongoDB Community Edtion Operator
Open source model	Apache 2.0	Apache 2.0	Open core	Open core
MongoDB versions	PSMDB 4.4, 5.0, 6.0	MongoDB 5.0	MongoDB 3.4, 3.6. 4.0, 4.1, 4.2	MongoDB 4.2, 4.4, 5.0
Kubernetes conformance	Various versions are tested	No guarantee	No guarantee	No guarantee
Cluster-wide mode	Yes	Not an operator	Enterprise only	Yes
Network exposure	Yes	Yes	No, only through manual config	Enterprise only

9.2.2 Maintenance

Upgrade and scaling are the two most common maintenance tasks that are executed by database administrators and developers.

Feature/ Product	Percona Operator for MongoDB	Bitnami Helm Chart	KubeDB for MongoDB	MongoDB Community Operator
Operator upgrade	Yes	Helm upgrade	Image change	Yes
Database upgrade	Automated minor, manual major	No	Manual minor	Manual mintor and major
Compute scaling	Horizontal and vertical	Horizontal and vertical	Enterprise only	Horizontal only
Storage scaling	Manual	Manual	Enterprise only	Enterprise only

9.2.3 MongoDB topologies

The next comparison is focused on replica sets, arbiters, sharding and other node types.

Feature/ Product	Percona Operator for MongoDB	Bitnami Helm Chart	KubeDB for MongoDB	MongoDB Community Operator
Multi-cluster deployment	Yes	No	No	No
Sharding	Yes	Yes, another chart	Yes	No
Arbiter	Yes	Yes	Yes	Yes
Non-voting nodes	Yes	No	No	No
Hidden nodes	No	Yes	Yes	Yes
Network exposure	Yes	Yes	Manual	Enterprise only

9.2.4 Backups

Here are the backup and restore capabilities of each solution.

Feature/ Product	Percona Operator for MongoDB	Bitnami Helm Chart	KubeDB for MongoDB	MongoDB Community Operator
Scheduled backups	Yes	No	Enterprise only	Enterprise only
Incremental backups	No	No	Enterprise only	No
Point-in-time recovery	Yes	No	No	Enterprise only
Logical backups	Yes	No	No	Enterprise only
Physical backups	Yes	No	No	Enterprise only

9.2.5 Monitoring

Monitoring is crucial for any operations team.

Feature/Product	**Percona Operator for MongoDB**	**Bitnami Helm Chart**	**KubeDB for MongoDB**	**MongoDB Community Operator**
Custom exporters	Yes, through sidecars	mongodb-exporter as a sidecar	mongodb-exporter as a sidecar	Integrate with prometheus operator
Percona Monitoring and Management (PMM)	Yes	No	No	No

9.2.6 Miscellaneous

Finally, let's compare various features that are not a good fit for other categories.

Feature/Product	**Percona Operator for MongoDB**	**Bitnami Helm Chart**	**KubeDB for MongoDB**	**MongoDB Community Operator**
Customize MongoDB configuration	Yes	Yes	Yes	No, only some params
Helm	Yes	Yes	Yes, for operator only	Yes, for operator only
SSL/TLS	Yes	Yes	Enterprise only	Yes
Create users/roles	No, only some params	Yes	No	Yes

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-05-23

10. Quickstart

10.1 Install Percona Server for MongoDB using Helm

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

10.1.1 Pre-requisites

Install Helm following its official installation instructions.

Note

Helm v3 is needed to run the following steps.

10.1.2 Installation

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

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

2. Install Percona Operator for MongoDB:

```
$ helm install my-op percona/psmdb-operator
```

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

Note

If nothing explicitly specified, `helm install` command will work with the `default` namespace and the latest version of the Helm chart.

- To use different namespace, provide its name with the following additional parameter: `--namespace my-namespace`.
- To use different Helm chart version, provide it as follows: `--version 1.15.0`

3. Install Percona Server for MongoDB:

```
$ helm install my-db percona/psmdb-db --namespace my-namespace
```

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

10.1.3 Installing Percona Server for MongoDB with customized parameters

The command above installs Percona Server for MongoDB with default parameters. Custom options can be passed to a `helm install` command as a `--set key=value[,key=value]` argument. The options passed with a chart can be any of the Operator's Custom Resource options.

Note

Parameters from the Replica Set section are treated differently: if you specify *any* parameter from replsets, the Operator *will not* use default values for this Replica Set. So do not specify Replica Set options at all or specify all needed options for the Replica Set.

The following example will deploy a Percona Server for MongoDB Cluster in the `psmdb` namespace, with disabled backups and 20 Gi storage:

```
$ helm install my-db percona/psmdb-db --version 1.15.0 --namespace psmdb \
  --set "replsets[0].name=rs0" --set "replsets[0].size=3" \
  --set "replsets[0].volumeSpec.pvc.resources.requests.storage=20Gi" \
  --set backup.enabled=false --set sharding.enabled=false
```

Also it can be more convenient in some cases to specify customized options in a YAML file instead of using separate command line parameters. The resulting file similar to the above example looks as follows:

```yaml
allowUnsafeConfigurations: true
sharding:
  enabled: false
replsets:
- name: rs0
  size: 3
  volumeSpec:
    pvc:
      resources:
        requests:
          storage: 2Gi
backup:
  enabled: false
```

Apply the resulting YAML file as follows:

```
$ helm install my-db percona/psmdb-db --namespace psmdb -f values.yaml
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-09

10.2 Install Percona Server for MongoDB using kubectl

The kubectl command line utility is a tool used before anything else to interact with Kubernetes and containerized applications running on it. Users can run kubectl to deploy applications, manage cluster resources, check logs, etc.

10.2.1 Pre-requisites

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

1. The **Git** distributed version control system. You can install it following the official installation instructions.
2. The **kubectl** tool to manage and deploy applications on Kubernetes, included in most Kubernetes distributions. Install it, if not present, following the official installation instructions.

10.2.2 Install the Operator and Percona Server for MongoDB

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

1. Deploy the Operator using the following command:

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.15.0/deploy/bundle.yaml
```

Expected output

```
customresourcedefinition.apiextensions.k8s.io/perconaservermongodbs.psmdb.percona.com
serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconaservermongodbbackups.psmdb.percona.com
serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconaservermongodbrestores.psmdb.percona.com
serverside-applied
role.rbac.authorization.k8s.io/percona-server-mongodb-operator serverside-applied
serviceaccount/percona-server-mongodb-operator serverside-applied
rolebinding.rbac.authorization.k8s.io/service-account-percona-server-mongodb-operator
serverside-applied
deployment.apps/percona-server-mongodb-operator serverside-applied
```

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

2. Deploy Percona Server for MongoDB:

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.15.0/deploy/cr.yaml
```

Expected output

```
perconaservermongodb.psmdb.percona.com/my-cluster-name created
```

Note

This deploys default MongoDB cluster configuration, three mongod, three mongos, and three config server instances. Please see deploy/cr.yaml and Custom Resource Options for the configuration options. You can clone the repository with all manifests and source code by executing the following command:

```
$ git clone -b v1.15.0 https://github.com/percona/percona-server-mongodb-operator
```

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

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

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

```
$ kubectl get psmdb
```

Expected output

```
NAME              ENDPOINT                                         STATUS    AGE
my-cluster-name   my-cluster-name-mongos.default.svc.cluster.local ready     5m26s
```

10.2.3 Verifying the cluster operation

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

1. You will need the login and password for the admin user to access the cluster. Use `kubectl get secrets` command to see the list of Secrets objects (by default the Secrets object you are interested in has `my-cluster-name-secrets` name). Then `kubectl get secret my-cluster-name-secrets -o yaml` command will return the YAML file with generated Secrets, including the `MONGODB_DATABASE_ADMIN_USER` and `MONGODB_DATABASE_ADMIN_PASSWORD` strings, which should look as follows:

```
...
data:
  ...
  MONGODB_DATABASE_ADMIN_PASSWORD: aDAzQ0pCY3NSWEZ2ZUIzS1I=
  MONGODB_DATABASE_ADMIN_USER: ZGF0YWJhc2VBZG1pbg==
```

   Here the actual login name and password are base64-encoded. Use `echo 'aDAzQ0pCY3NSWEZ2ZUIzS1I=' | base64 --decode` command to bring it back to a human-readable form.

2. Run a container with a MongoDB client and connect its console output to your terminal. The following command will do this, naming the new Pod `percona-client`:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:
4.4.24-23 --restart=Never -- bash -il
```

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

3. Now run `mongo` tool in the percona-client command shell using the login (which is normally `databaseAdmin`), a proper password obtained from the Secret, and a proper namespace name instead of the `<namespace name>` placeholder. The command will look different depending on whether sharding is on (the default behavior) or off:

if sharding is on

```
$ mongosh "mongodb://databaseAdmin:databaseAdminPassword@my-cluster-name-
mongos.<namespace name>.svc.cluster.local/admin?ssl=false"
```

if sharding is off

```
$ mongosh "mongodb+srv://databaseAdmin:databaseAdminPassword@my-cluster-name-
rs0.<namespace name>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
```

Note

If using MongoDB versions earler than 6.x (such as 4.4.24-23 or 5.0.20-17 instead of the default 6.0.9-7 variant), substitute `mongosh` command with `mongo` in the above example.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-13

11. Installation

11.1 System Requirements

The Operator was developed and tested with Percona Server for MongoDB 4.4.24-23, 5.0.20-17, and 6.0.9-7. Other options may also work but have not been tested. The Operator 1.15.0 also uses Percona Backup for MongoDB 2.3.0.

11.1.1 Officially supported platforms

The following platforms were tested and are officially supported by the Operator 1.15.0:

- Google Kubernetes Engine (GKE) 1.24-1.28
- Amazon Elastic Container Service for Kubernetes (EKS) 1.24-1.28
- OpenShift Container Platform 4.11 - 4.13
- Azure Kubernetes Service (AKS) 1.25-1.28
- Minikube 1.31.2 (based on Kubernetes 1.28)

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

11.1.2 Resource Limits

A cluster running an officially supported platform contains at least 3 Nodes and the following resources (if sharding is turned off):

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

Consider using 4 CPU and 6 GB of RAM if sharding is turned on (the default behavior).

Also, the number of Replica Set Nodes should not be odd if Arbiter is not enabled.

Note

Use Storage Class with XFS as the default filesystem if possible to achieve better MongoDB performance.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

11.2 Install Percona Server for MongoDB on Minikube

Installing the Percona Operator for MongoDB on Minikube is the easiest way to try it locally without a cloud provider. Minikube runs Kubernetes on GNU/Linux, Windows, or macOS system using a system-wide hypervisor, such as VirtualBox, KVM/QEMU, VMware Fusion or Hyper-V. Using it is a popular way to test Kubernetes application locally prior to deploying it on a cloud.

The following steps are needed to run Percona Operator for MongoDB on minikube:

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

   a. kubectl tool,

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

   c. actual minikube package

   After the installation, run `minikube start --memory=5120 --cpus=4 --disk-size=30g` (parameters increase the virtual machine limits for the CPU cores, memory, and disk, 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.

2. Deploy the operator using the following command:

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-
mongodb-operator/v1.15.0/deploy/bundle.yaml
```

3. Deploy MongoDB cluster with:

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-
operator/v1.15.0/deploy/cr-minimal.yaml
```

Note

This deploys a one-shard MongoDB cluster with one replica set with one node, one mongos node and one config server node. The deploy/cr-minimal.yaml is for minimal non-production deployment. For more configuration options please see deploy/cr.yaml and Custom Resource Options. You can clone the repository with all manifests and source code by executing the following command:

```
$ git clone -b v1.15.0 https://github.com/percona/percona-server-mongodb-operator
```

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

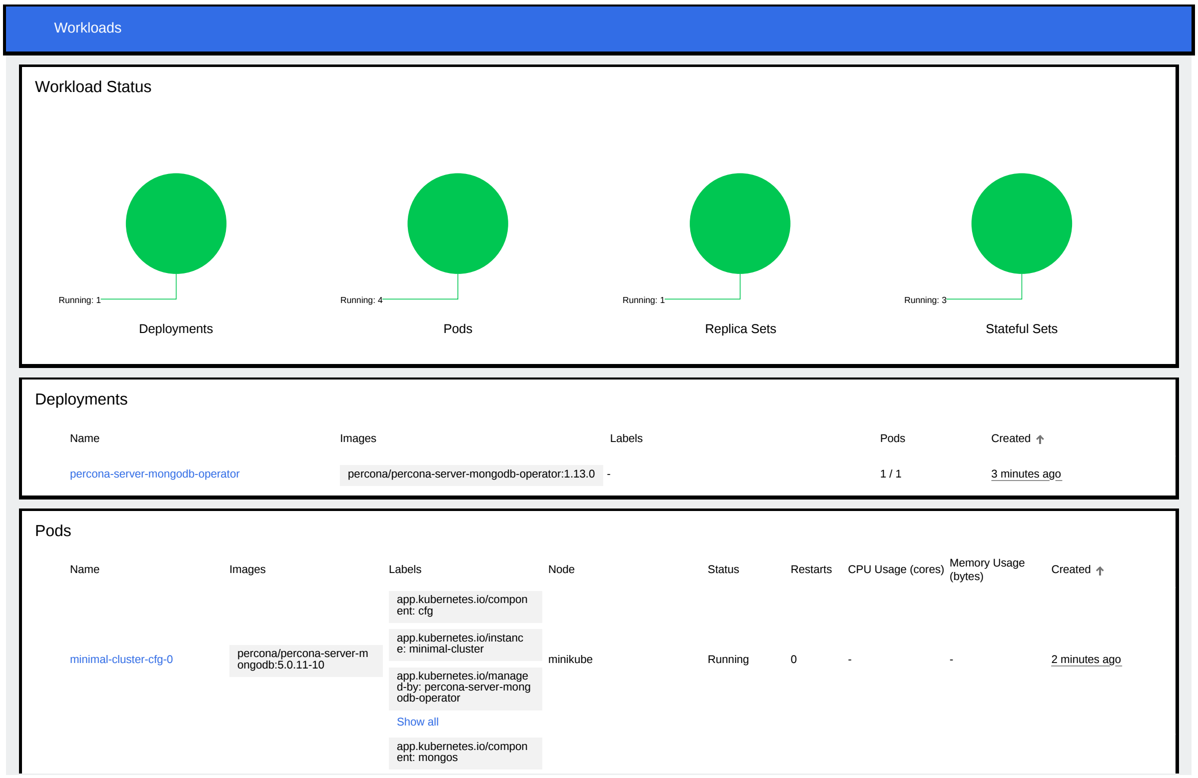
```
$ kubectl apply -f deploy/cr.yaml
```

The creation process may take some time.

The process is over when both operator and replica set pod have reached their Running status. `kubectl get pods` output should look like this:

```
NAME                                              READY   STATUS    RESTARTS   AGE
percona-server-mongodb-operator-d859b69b6-t44vk   1/1     Running   0          50s
minimal-cluster-cfg-0                             1/1     Running   0          41s
minimal-cluster-mongos-0                          1/1     Running   0          36s
minimal-cluster-rs0-0                             1/1     Running   0          39s
```

You can also track the progress via the Kubernetes dashboard:



4. During previous steps, the Operator has generated several secrets, including the password for the admin user, which you will 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 `minimal-cluster-name-secrets` name). Then `kubectl get secret minimal-cluster-name-secrets -o yaml` will return the YAML file with generated secrets, including the `MONGODB_USER_ADMIN` and `MONGODB_USER_ADMIN_PASSWORD` strings, which should look as follows:

```
...
data:
  ...
  MONGODB_USER_ADMIN_PASSWORD: aDAzQ0pCY3NSWEZ2ZUIzS1I=
  MONGODB_USER_ADMIN_USER: dXNlckFkbWlu
```

   Here the actual login name and password are base64-encoded, and `echo 'aDAzQ0pCY3NSWEZ2ZUIzS1I=' | base64 --decode` will bring it back to a human-readable form.

5. Check connectivity to a newly created cluster.

   First of all, run a container with a MongoDB client and connect its console output to your terminal. The following command will do this, naming the new Pod `percona-client`:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:
4.4.24-23 --restart=Never -- bash -il
```

   Executing it may require some time to deploy the correspondent Pod. Now run `mongo` tool in the percona-client command shell using the login (which is `userAdmin`) and password obtained from the secret:

```
$ mongo "mongodb://userAdmin:userAdmin123456@minimal-cluster-name-
mongos.default.svc.cluster.local/admin?ssl=false"
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-30

11.3 Install Percona Server for MongoDB on Google Kubernetes Engine (GKE)

This guide shows you how to deploy Percona Operator for MongoDB on Google Kubernetes Engine (GKE). The document assumes some experience with the platform. For more information on the GKE, see the Kubernetes Engine Quickstart.

11.3.1 Prerequisites

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

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

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

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

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

11.3.2 Create and configure the GKE cluster

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

```
$ gcloud container clusters create my-cluster-name --project <project name> --zone us-
central1-a --cluster-version 1.25 --machine-type n1-standard-4 --num-nodes=3
```

Note

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

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

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

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

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

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

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

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

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

Expected output

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

11.3.3 Install the Operator and deploy your MongoDB cluster

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

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

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

   Deploy the Operator using the following command:

   ```
   $ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-
   mongodb-operator/v1.15.0/deploy/bundle.yaml
   ```

   Expected output

   ```
   customresourcedefinition.apiextensions.k8s.io/perconaservermongodbs.psmdb.percona.com
   serverside-applied
   customresourcedefinition.apiextensions.k8s.io/perconaservermongodbbackups.psmdb.percona.com
   serverside-applied
   customresourcedefinition.apiextensions.k8s.io/perconaservermongodbrestores.psmdb.percona.com
   serverside-applied
   role.rbac.authorization.k8s.io/percona-server-mongodb-operator serverside-applied
   serviceaccount/percona-server-mongodb-operator serverside-applied
   rolebinding.rbac.authorization.k8s.io/service-account-percona-server-mongodb-operator
   serverside-applied
   deployment.apps/percona-server-mongodb-operator serverside-applied
   ```

2. The operator has been started, and you can deploy your MongoDB cluster:

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-
operator/v1.15.0/deploy/cr.yaml
```

Expected output

```
perconaservermongodb.psmdb.percona.com/my-cluster-name created
```

Note

This deploys default MongoDB cluster configuration, three mongod, three mongos, and three config server instances. Please see deploy/cr.yaml and Custom Resource Options for the configuration options. You can clone the repository with all manifests and source code by executing the following command:

```
$ git clone -b v1.15.0 https://github.com/percona/percona-server-mongodb-operator
```

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

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

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

```
$ kubectl get psmdb
```

Expected output

```
NAME              ENDPOINT                                           STATUS    AGE
my-cluster-name   my-cluster-name-mongos.default.svc.cluster.local   ready     5m26s
```

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

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

Name	Status	Type	Namespace	Cluster
core		API Group		
Pod		Kind		
my-cluster-name-cfg-0	Running	Pod	default	my-cluster-name
my-cluster-name-cfg-1	Running	Pod	default	my-cluster-name
my-cluster-name-cfg-2	Running	Pod	default	my-cluster-name
my-cluster-name-mongos-0	Running	Pod	default	my-cluster-name
my-cluster-name-mongos-1	Running	Pod	default	my-cluster-name
my-cluster-name-mongos-2	Running	Pod	default	my-cluster-name
my-cluster-name-rs0-0	Running	Pod	default	my-cluster-name
my-cluster-name-rs0-1	Running	Pod	default	my-cluster-name
my-cluster-name-rs0-2	Running	Pod	default	my-cluster-name
percona-server-mongodb-operator-665cd69f9b-xg5dl	Running	Pod	default	my-cluster-name

11.3.4 Verifying the cluster operation

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

1. You will need the login and password for the admin user to access the cluster. Use `kubectl get secrets` command to see the list of Secrets objects (by default the Secrets object you are interested in has `my-cluster-name-secrets` name). Then `kubectl get secret my-cluster-name-secrets -o yaml` command will return the YAML file with generated Secrets, including the `MONGODB_DATABASE_ADMIN_USER` and `MONGODB_DATABASE_ADMIN_PASSWORD` strings, which should look as follows:

```
...
data:
  ...
  MONGODB_DATABASE_ADMIN_PASSWORD: aDAzQ0pCY3NSWEZ2ZUIzS1I=
  MONGODB_DATABASE_ADMIN_USER: ZGF0YWJhc2VBZG1pbg==
```

   Here the actual login name and password are base64-encoded. Use `echo 'aDAzQ0pCY3NSWEZ2ZUIzS1I=' | base64 --decode` command to bring it back to a human-readable form.

2. Run a container with a MongoDB client and connect its console output to your terminal. The following command will do this, naming the new Pod `percona-client`:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:
4.4.24-23 --restart=Never -- bash -il
```

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

3. Now run `mongo` tool in the percona-client command shell using the login (which is normally `databaseAdmin`), a proper password obtained from the Secret, and a proper namespace name instead of the `<namespace name>` placeholder. The command will look different depending on whether sharding is on (the default behavior) or off:

if sharding is on

```
$ mongosh "mongodb://databaseAdmin:databaseAdminPassword@my-cluster-name-
mongos.<namespace name>.svc.cluster.local/admin?ssl=false"
```

if sharding is off

```
$ mongosh "mongodb+srv://databaseAdmin:databaseAdminPassword@my-cluster-name-
rs0.<namespace name>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
```

> **Note**
>
> If using MongoDB versions earler than 6.x (such as 4.4.24-23 or 5.0.20-17 instead of the default 6.0.9-7 variant), substitute `mongosh` command with `mongo` in the above example.

11.3.5 Troubleshooting

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

```
$ kubectl get pods
```

Expected output

```
NAME                                              READY   STATUS    RESTARTS   AGE
my-cluster-name-cfg-0                             2/2     Running   0          11m
my-cluster-name-cfg-1                             2/2     Running   1          10m
my-cluster-name-cfg-2                             2/2     Running   1          9m
my-cluster-name-mongos-0                          1/1     Running   0          11m
my-cluster-name-mongos-1                          1/1     Running   0          11m
my-cluster-name-mongos-2                          1/1     Running   0          11m
my-cluster-name-rs0-0                             2/2     Running   0          11m
my-cluster-name-rs0-1                             2/2     Running   0          10m
my-cluster-name-rs0-2                             2/2     Running   0          9m
percona-server-mongodb-operator-665cd69f9b-xg5dl  1/1     Running   0          37m
```

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

```
$ kubectl describe pod my-cluster-name-rs0-2
```

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

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

Alternatively, you can examine your Pods via the object browser

The errors will look as follows:

Name	Status	Type	Namespace	Cluster
core		API Group		
Pod		Kind		
my-cluster-name-cfg-0	Running	Pod	default	my-cluster-name
my-cluster-name-cfg-1	Running	Pod	default	my-cluster-name
my-cluster-name-cfg-2	Running	Pod	default	my-cluster-name
my-cluster-name-mongos-0	Running	Pod	default	my-cluster-name
my-cluster-name-mongos-1	Running	Pod	default	my-cluster-name
my-cluster-name-mongos-2	Running	Pod	default	my-cluster-name
my-cluster-name-rs0-0	Running	Pod	default	my-cluster-name
my-cluster-name-rs0-1	Running	Pod	default	my-cluster-name
my-cluster-name-rs0-2	Unschedulable	Pod	default	my-cluster-name
percona-server-mongodb-operator-665cd69f9b-xg5dl	Running	Pod	default	my-cluster-name

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

0/3 nodes are available: 3 node(s) didn't match Pod's node affinity/selector. SHOW DETAILS

11.3.6 Removing the GKE cluster

There are several ways that you can delete the cluster.

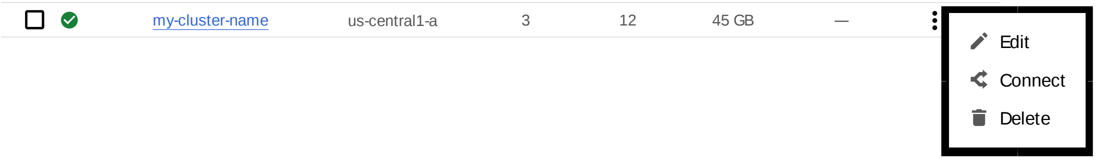
You can clean up the cluster with the `gcloud` command as follows:

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

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

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

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



The cluster deletion may take time.

Warning

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

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-13

11.4 Install Percona Server for MongoDB on Amazon Elastic Kubernetes Service (EKS)

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

11.4.1 Prerequisites

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

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

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

11.4.2 Create the EKS cluster

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

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

    After you have settled all the needed details, create your EKS cluster following the official cluster creation instructions.
2. After you have created the EKS cluster, you also need to install the Amazon EBS CSI driver on your cluster. See the official documentation on adding it as an Amazon EKS add-on.

11.4.3 Install the Operator and deploy your MongoDB cluster

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

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

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

    Deploy the Operator using the following command:

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-
mongodb-operator/v1.15.0/deploy/bundle.yaml
```

Expected output

```
customresourcedefinition.apiextensions.k8s.io/perconaservermongodbs.psmdb.percona.com
serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconaservermongodbbackups.psmdb.percona.com
serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconaservermongodbrestores.psmdb.percona.com
serverside-applied
role.rbac.authorization.k8s.io/percona-server-mongodb-operator serverside-applied
serviceaccount/percona-server-mongodb-operator serverside-applied
rolebinding.rbac.authorization.k8s.io/service-account-percona-server-mongodb-operator
serverside-applied
deployment.apps/percona-server-mongodb-operator serverside-applied
```

2. The operator has been started, and you can deploy your MongoDB cluster:

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-
operator/v1.15.0/deploy/cr.yaml
```

Expected output

```
perconaservermongodb.psmdb.percona.com/my-cluster-name created
```

Note

This deploys default MongoDB cluster configuration, three mongod, three mongos, and three config server instances. Please see deploy/cr.yaml and Custom Resource Options for the configuration options. You can clone the repository with all manifests and source code by executing the following command:

```
$ git clone -b v1.15.0 https://github.com/percona/percona-server-mongodb-operator
```

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

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

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

```
$ kubectl get psmdb
```

Expected output

```
NAME              ENDPOINT                                           STATUS   AGE
my-cluster-name   my-cluster-name-mongos.default.svc.cluster.local   ready    5m26s
```

11.4.4 Verifying the cluster operation

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

1. You will need the login and password for the admin user to access the cluster. Use `kubectl get secrets` command to see the list of Secrets objects (by default the Secrets object you are interested in has `my-cluster-name-secrets` name). Then `kubectl get secret my-cluster-name-secrets -o yaml` command will return the YAML file with generated Secrets, including the `MONGODB_DATABASE_ADMIN_USER` and `MONGODB_DATABASE_ADMIN_PASSWORD` strings, which should look as follows:

```
...
data:
  ...
  MONGODB_DATABASE_ADMIN_PASSWORD: aDAzQ0pCY3NSWEZ2ZUIzS1I=
  MONGODB_DATABASE_ADMIN_USER: ZGF0YWJhc2VBZG1pbg==
```

   Here the actual login name and password are base64-encoded. Use `echo 'aDAzQ0pCY3NSWEZ2ZUIzS1I=' | base64 --decode` command to bring it back to a human-readable form.

2. Run a container with a MongoDB client and connect its console output to your terminal. The following command will do this, naming the new Pod `percona-client`:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:
4.4.24-23 --restart=Never -- bash -il
```

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

3. Now run `mongo` tool in the percona-client command shell using the login (which is normally `databaseAdmin`), a proper password obtained from the Secret, and a proper namespace name instead of the `<namespace name>` placeholder. The command will look different depending on whether sharding is on (the default behavior) or off:

if sharding is on

```
$ mongosh "mongodb://databaseAdmin:databaseAdminPassword@my-cluster-name-
mongos.<namespace name>.svc.cluster.local/admin?ssl=false"
```

if sharding is off

```
$ mongosh "mongodb+srv://databaseAdmin:databaseAdminPassword@my-cluster-name-
rs0.<namespace name>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
```

Note

If using MongoDB versions earler than 6.x (such as 4.4.24-23 or 5.0.20-17 instead of the default 6.0.9-7 variant), substitute `mongosh` command with `mongo` in the above example.

11.4.5 Troubleshooting

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

```
$ kubectl get pods
```

Expected output

```
NAME                                               READY   STATUS    RESTARTS   AGE
my-cluster-name-cfg-0                              2/2     Running   0          11m
my-cluster-name-cfg-1                              2/2     Running   1          10m
my-cluster-name-cfg-2                              2/2     Running   1          9m
my-cluster-name-mongos-0                           1/1     Running   0          11m
my-cluster-name-mongos-1                           1/1     Running   0          11m
my-cluster-name-mongos-2                           1/1     Running   0          11m
my-cluster-name-rs0-0                              2/2     Running   0          11m
my-cluster-name-rs0-1                              2/2     Running   0          10m
my-cluster-name-rs0-2                              2/2     Running   0          9m
percona-server-mongodb-operator-665cd69f9b-xg5dl   1/1     Running   0          37m
```

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

```
$ kubectl describe pod my-cluster-name-rs0-2
```

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

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

11.4.6 Removing the EKS cluster

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

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

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

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

The cluster deletion may take time.

Warning

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

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-07-06

11.5 Install Percona Server for MongoDB on Azure Kubernetes Service (AKS)

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

11.5.1 Prerequisites

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

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

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

11.5.2 Create and configure the AKS cluster

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

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

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

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

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

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

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

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

11.6 Install the Operator and deploy your MongoDB cluster

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

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

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

Deploy the Operator using the following command:

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.15.0/deploy/bundle.yaml
```

Expected output

```
customresourcedefinition.apiextensions.k8s.io/perconaservermongodbs.psmdb.percona.com
serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconaservermongodbbackups.psmdb.percona.com
serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconaservermongodbrestores.psmdb.percona.com
serverside-applied
role.rbac.authorization.k8s.io/percona-server-mongodb-operator serverside-applied
serviceaccount/percona-server-mongodb-operator serverside-applied
rolebinding.rbac.authorization.k8s.io/service-account-percona-server-mongodb-operator
serverside-applied
deployment.apps/percona-server-mongodb-operator serverside-applied
```

2. The operator has been started, and you can deploy your MongoDB cluster:

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.15.0/deploy/cr.yaml
```

Expected output

```
perconaservermongodb.psmdb.percona.com/my-cluster-name created
```

Note

This deploys default MongoDB cluster configuration, three mongod, three mongos, and three config server instances. Please see deploy/cr.yaml and Custom Resource Options for the configuration options. You can clone the repository with all manifests and source code by executing the following command:

```
$ git clone -b v1.15.0 https://github.com/percona/percona-server-mongodb-operator
```

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

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

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

```
$ kubectl get psmdb
```

Expected output

```
NAME             ENDPOINT                                        STATUS   AGE
my-cluster-name  my-cluster-name-mongos.default.svc.cluster.local  ready    5m26s
```

11.6.1 Verifying the cluster operation

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

1. You will need the login and password for the admin user to access the cluster. Use `kubectl get secrets` command to see the list of Secrets objects (by default the Secrets object you are interested in has `my-cluster-name-secrets` name). Then `kubectl get secret my-cluster-name-secrets -o yaml` command will return the YAML file with generated Secrets, including the `MONGODB_DATABASE_ADMIN_USER` and `MONGODB_DATABASE_ADMIN_PASSWORD` strings, which should look as follows:

```
...
data:
  ...
  MONGODB_DATABASE_ADMIN_PASSWORD: aDAzQ0pCY3NSWEZ2ZUIzS1I=
  MONGODB_DATABASE_ADMIN_USER: ZGF0YWJhc2VBZG1pbg==
```

   Here the actual login name and password are base64-encoded. Use `echo 'aDAzQ0pCY3NSWEZ2ZUIzS1I=' | base64 --decode` command to bring it back to a human-readable form.

2. Run a container with a MongoDB client and connect its console output to your terminal. The following command will do this, naming the new Pod `percona-client`:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:
4.4.24-23 --restart=Never -- bash -il
```

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

3. Now run `mongo` tool in the percona-client command shell using the login (which is normally `databaseAdmin`), a proper password obtained from the Secret, and a proper namespace name instead of the `<namespace name>` placeholder. The command will look different depending on whether sharding is on (the default behavior) or off:

if sharding is on

```
$ mongosh "mongodb://databaseAdmin:databaseAdminPassword@my-cluster-name-
mongos.<namespace name>.svc.cluster.local/admin?ssl=false"
```

if sharding is off

```
$ mongosh "mongodb+srv://databaseAdmin:databaseAdminPassword@my-cluster-name-
rs0.<namespace name>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
```

Note

If using MongoDB versions earler than 6.x (such as 4.4.24-23 or 5.0.20-17 instead of the default 6.0.9-7 variant), substitute `mongosh` command with `mongo` in the above example.

11.6.2 Troubleshooting

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

```
$ kubectl get pods
```

Expected output

```
NAME                                              READY   STATUS    RESTARTS   AGE
my-cluster-name-cfg-0                             2/2     Running   0          11m
my-cluster-name-cfg-1                             2/2     Running   1          10m
my-cluster-name-cfg-2                             2/2     Running   1          9m
my-cluster-name-mongos-0                          1/1     Running   0          11m
my-cluster-name-mongos-1                          1/1     Running   0          11m
my-cluster-name-mongos-2                          1/1     Running   0          11m
my-cluster-name-rs0-0                             2/2     Running   0          11m
my-cluster-name-rs0-1                             2/2     Running   0          10m
my-cluster-name-rs0-2                             2/2     Running   0          9m
percona-server-mongodb-operator-665cd69f9b-xg5dl  1/1     Running   0          37m
```

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

```
$ kubectl describe pod my-cluster-name-rs0-2
```

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

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

11.6.3 Removing the AKS cluster

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

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

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

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

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

Warning

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

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-13

11.7 Install Percona server for MongoDB on Kubernetes

1. Clone the percona-server-mongodb-operator repository:

```
$ git clone -b v1.15.0 https://github.com/percona/percona-server-mongodb-operator
$ cd percona-server-mongodb-operator
```

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

2. The Custom Resource Definition for Percona Server for MongoDB should be created from the `deploy/crd.yaml` file. The Custom Resource Definition extends the standard set of resources which Kubernetes "knows" about with the new items, in our case these items are the core of the operator. Apply it as follows:

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

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

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

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

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

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

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

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

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

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

5. Start the operator within Kubernetes:

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

6. Add the MongoDB Users secrets to Kubernetes. These secrets should be placed as plain text in the stringData section of the `deploy/secrets.yaml` file as login name and passwords for the user accounts (see Kubernetes documentation for details).

   After editing the yaml file, MongoDB Users secrets should be created using the following command:

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

   More details about secrets can be found in Users.

7. 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.

8. After the operator is started, Percona Server for MongoDB cluster can be created with the following command:

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

   The creation process may take some time. The process is over when all Pods have reached their Running status. You can check it with the following command:

   ```
   $ kubectl get pods
   ```

   The result should look as follows:

   ```
   NAME                                              READY   STATUS    RESTARTS   AGE
   my-cluster-name-cfg-0                             2/2     Running   0          11m
   my-cluster-name-cfg-1                             2/2     Running   1          10m
   my-cluster-name-cfg-2                             2/2     Running   1          9m
   my-cluster-name-mongos-0                          1/1     Running   0          11m
   my-cluster-name-mongos-1                          1/1     Running   0          11m
   my-cluster-name-mongos-2                          1/1     Running   0          11m
   my-cluster-name-rs0-0                             2/2     Running   0          11m
   my-cluster-name-rs0-1                             2/2     Running   0          10m
   my-cluster-name-rs0-2                             2/2     Running   0          9m
   percona-server-mongodb-operator-665cd69f9b-xg5dl  1/1     Running   0          37m
   ```

9. Check connectivity to a newly created cluster.

   First of all, run a container with a MongoDB client and connect its console output to your terminal. The following command will do this, naming the new Pod `percona-client`:

   ```
   $ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:
   4.4.24-23 --restart=Never -- bash -il
   ```

   Executing it may require some time to deploy the correspondent Pod. Now run `mongo` tool in the percona-client command shell using the login (which is `userAdmin`) with a proper password obtained from the Secret, and a proper namespace name instead of the `<namespace name>` placeholder:

   ```
   percona-client:/$ mongo "mongodb://userAdmin:userAdmin123456@my-cluster-name-
   mongos.<namespace name>.svc.cluster.local/admin?ssl=false"
   ```

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-14

11.8 Install Percona Server for MongoDB on OpenShift

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

Installing Percona Server for MongoDB on OpenShift includes two steps:

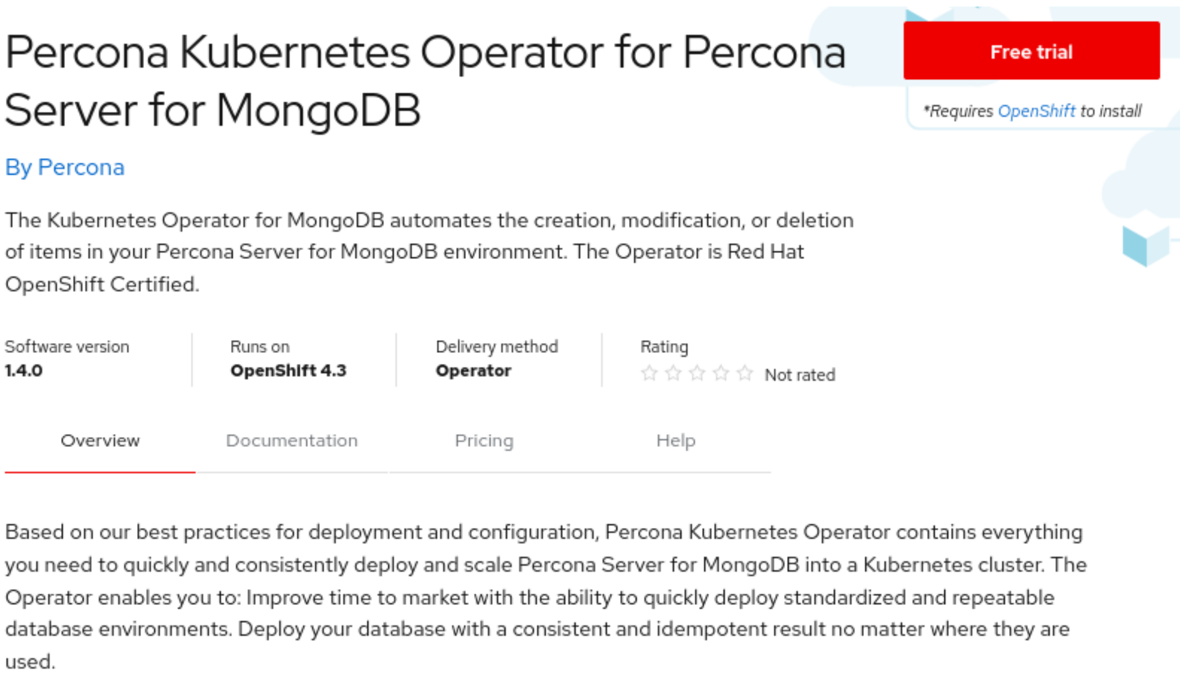
- Installing the Percona Operator for MongoDB,
- Install Percona Server for MongoDB using the Operator.

11.8.1 Install the Operator

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

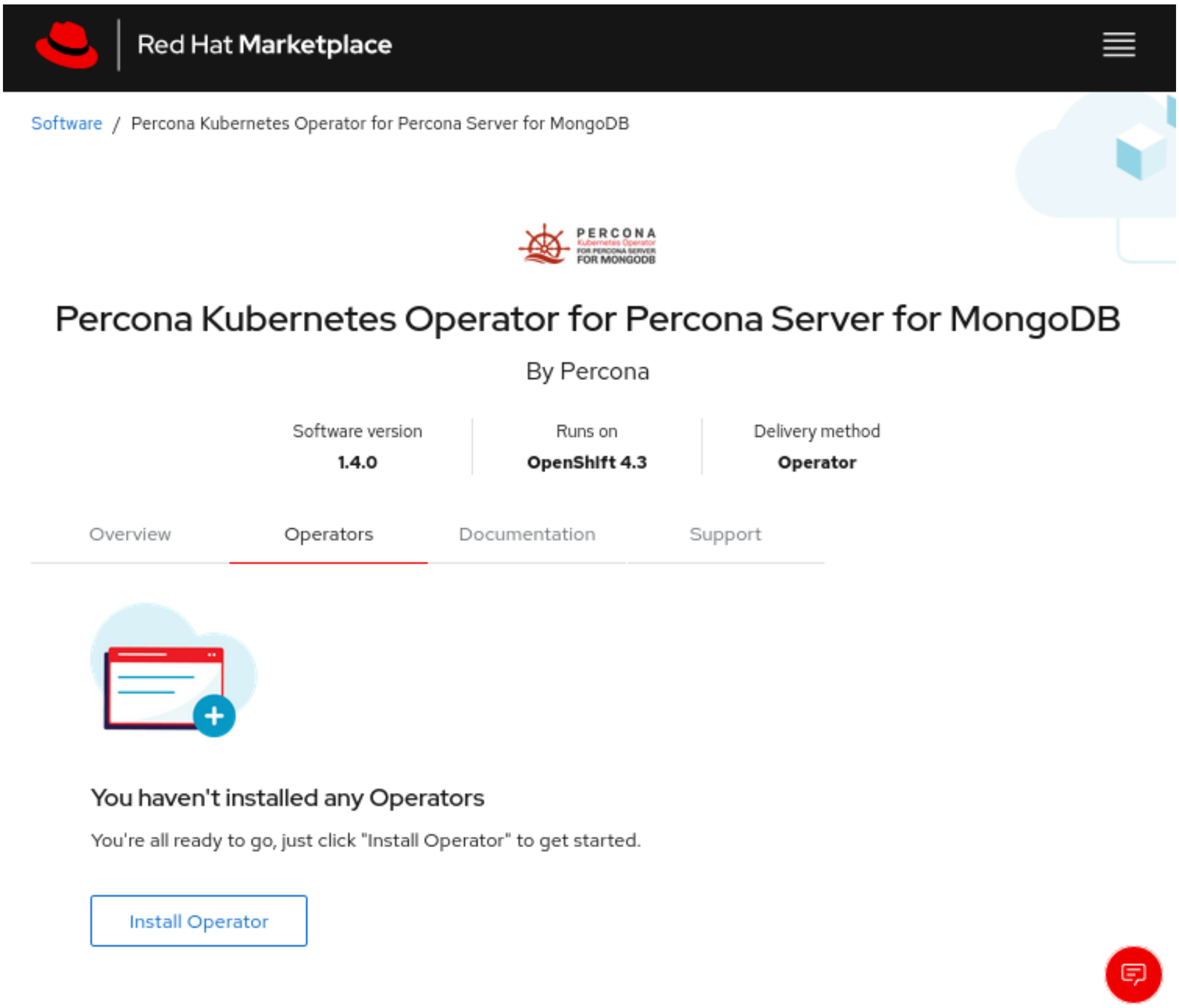
Install the Operator via the Red Hat Marketplace

1. login to the Red Hat Marketplace and register your cluster following the official instructions.
2. Go to the Percona Operator for MongoDB page and click the Free trial button:



   Here you can “purchase” the Operator for 0.0 USD.

3. When finished, chose `Workspace->Software` in the system menu on the top and choose the Operator:



Click the `Install Operator` button.

Install the Operator via the command-line interface

1. Clone the percona-server-mongodb-operator repository:

```
$ git clone -b v1.15.0 https://github.com/percona/percona-server-mongodb-operator
$ cd percona-server-mongodb-operator
```

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

2. The Custom Resource Definition for Percona Server for MongoDB should be created from the `deploy/crd.yaml` file. The Custom Resource Definition extends the standard set of resources which Kubernetes "knows" about with the new items, in our case these items are the core of the operator.

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

   Apply it as follows:

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

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

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

```
$ oc create clusterrole psmdb-admin --verb="*" --
resource=perconaservermongodbs.psmdb.percona.com,perconaservermongodbs.psmdb.percona.com/
status,perconaservermongodbbackups.psmdb.percona.com,perconaservermongodbbackups.psmdb.percon
status,perconaservermongodbrestores.psmdb.percona.com,perconaservermongodbrestores.psmdb.perc
status
$ oc adm policy add-cluster-role-to-user psmdb-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=iissuers.certmanager.k8s.io,certificates.certmanager.k8s.io
$ oc adm policy add-cluster-role-to-user cert-admin <some-user>
```

3. Create a new `psmdb` project:

   ```
   $ oc new-project psmdb
   ```

4. Add role-based access control (RBAC) for Percona Server for MongoDB is configured with the `deploy/rbac.yaml` file. RBAC is based on clearly defined roles and corresponding allowed actions. These actions are allowed on specific Kubernetes resources. The details about users and roles can be found in OpenShift documentation.

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

5. Start the Operator within OpenShift:

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

11.8.2 Install Percona Server for MongoDB

1. Add the MongoDB Users secrets to OpenShift. These secrets should be placed as plain text in the stringData section of the `deploy/secrets.yaml` file as login name and passwords for the user accounts (see Kubernetes documentation for details).

   After editing the yaml file, the secrets should be created with the following command:

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

   More details about secrets can be found in Users.

2. Now certificates should be generated. By default, the Operator generates certificates automatically, and no actions are required at this step. Still, you can generate and apply your own certificates as secrets according to the TLS instructions.

3. Percona Server for MongoDB cluster can be created at any time with the following steps:

   a. Uncomment the `deploy/cr.yaml` field `#platform:` and edit the field to `platform: openshift`. The result should be like this:

```
apiVersion: psmdb.percona.com/v1alpha1
kind: PerconaServerMongoDB
metadata:
  name: my-cluster-name
spec:
  platform: openshift
...
```

   b. (optional) In you're using minishift, please adjust antiaffinity policy to `none`

```
    affinity:
      antiAffinityTopologyKey: "none"
...
```

   c. Create/apply the Custom Resource file:

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

   The creation process will take time. The process is complete when all Pods have reached their Running status. You can check it with the following command:

```
$ oc get pods
```

   The result should look as follows:

```
NAME                                              READY   STATUS    RESTARTS   AGE
my-cluster-name-cfg-0                             2/2     Running   0          11m
my-cluster-name-cfg-1                             2/2     Running   1          10m
my-cluster-name-cfg-2                             2/2     Running   1          9m
my-cluster-name-mongos-0                          1/1     Running   0          11m
my-cluster-name-mongos-1                          1/1     Running   0          11m
my-cluster-name-mongos-2                          1/1     Running   0          11m
my-cluster-name-rs0-0                             2/2     Running   0          11m
my-cluster-name-rs0-1                             2/2     Running   0          10m
my-cluster-name-rs0-2                             2/2     Running   0          9m
percona-server-mongodb-operator-665cd69f9b-xg5dl  1/1     Running   0          37m
```

4. Check connectivity to newly created cluster.

   First of all, run a container with a MongoDB client and connect its console output to your terminal. The following command will do this, naming the new Pod `percona-client`:

```
$ oc run -i --rm --tty percona-client --image=percona/percona-server-mongodb:4.4.24-23 --restart=Never -- bash -il
```

   Executing it may require some time to deploy the correspondent Pod. Now run `mongo` tool in the percona-client command shell using the login (which is `userAdmin`) with a proper password obtained from the Secret:

```
percona-client:/$ mongo "mongodb://userAdmin:userAdmin123456@my-cluster-name-mongos.psmdb.svc.cluster.local/admin?ssl=false"
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-14

12. Configuration

12.1 Users

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

- *application-level users*: the unprivileged user accounts,
- *system-level users*: the accounts needed to automate the cluster deployment and management tasks, such as MongoDB Health checks.

As these two groups of user accounts serve different purposes, they are considered separately in the following sections.

12.1.1 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 -i --rm --tty percona-client --image=percona/percona-server-mongodb:4.4.24-23
--restart=Never -- bash -il
mongodb@percona-client:/$ mongo "mongodb+srv://userAdmin:userAdmin123456@my-cluster-name-
rs0.psmdb.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
rs0:PRIMARY> db.createUser({
    user: "myApp",
    pwd: "myAppPassword",
    roles: [
      { db: "myApp", role: "readWrite" }
    ],
    mechanisms: [
       "SCRAM-SHA-1"
    ]

})
```

Now check the newly created user:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:4.4.24-23
--restart=Never -- bash -il
mongodb@percona-client:/$ mongo "mongodb+srv://myApp:myAppPassword@my-cluster-name-
rs0.psmdb.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
rs0:PRIMARY> use myApp
rs0:PRIMARY> db.test.insert({ x: 1 })
rs0:PRIMARY> db.test.findOne()
```

12.1.2 System Users

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

Credentials for these users are stored as a Kubernetes Secrets object. The Operator requires Kubernetes Secret before the database cluster is started. It will either use existing Secret or create a new Secret with randomly generated passwords if it didn't exist. The name of the required Secret should be set in the `spec.secrets.users` option of the `deploy/cr.yaml` configuration file.

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

Secret name field: `spec.secrets.users`

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

User Purpose	Username Secret Key	Password Secret Key
Backup/ Restore	MONGODB_BACKUP_USER	MONGODB_BACKUP_PASSWORD
Cluster Admin	MONGODB_CLUSTER_ADMIN_USER	MONGODB_CLUSTER_ADMIN_PASSWORD
Cluster Monitor	MONGODB_CLUSTER_MONITOR_USER	MONGODB_CLUSTER_MONITOR_PASSWORD
Database Admin	MONGODB_DATABASE_ADMIN_USER	MONGODB_DATABASE_ADMIN_PASSWORD
User Admin	MONGODB_USER_ADMIN_USER	MONGODB_USER_ADMIN_PASSWORD
PMM Server	PMM_SERVER_USER	PMM_SERVER_PASSWORD

Password-based authorization method for PMM is deprecated since the Operator 1.13.0. Use token-based authorization instead.

- Backup/Restore - MongoDB Role: backup, restore, clusterMonitor
- Cluster Admin - MongoDB Roles: clusterAdmin
- Cluster Monitor - MongoDB Role: clusterMonitor
- Database Admin - MongoDB Roles: readWriteAnyDatabase, readAnyDatabase, dbAdminAnyDatabase, backup, restore, clusterMonitor
- User Admin - MongoDB Role: userAdmin

If you change credentials for the `MONGODB_CLUSTER_MONITOR` user, the cluster Pods will go into restart cycle, and the cluster can be not accessible through the `mongos` service until this cycle finishes.

Note

In some situations it can be needed to reproduce system users in a bare-bone MongoDB. For example, that's a required step in the migration scenarios to move existing on-prem MongoDB database to Kubernetes-based MongoDB cluster managed by the Operator. You can use the following example script which produces a text file with mongo shell commands to create needed system users with appropriate roles:

gen_users.sh

```
clusterAdminPass="clusterAdmin"
userAdminPass="userAdmin"
clusterMonitorPass="clusterMonitor"
backupPass="backup"

# mongo shell
cat <<EOF > user-mongo-shell.txt
use admin
db.createRole(
{
"roles": [],
role: "pbmAnyAction",
"privileges" : [
                {
                        "resource" : {
                                "anyResource" : true
                        },
                        "actions" : [
                                "anyAction"
                        ]
                }
        ],

})

db.createUser( { user: "clusterMonitor", pwd: "$clusterMonitorPass", roles:
[ "clusterMonitor" ] } )
db.createUser( { user: "userAdmin", pwd: "$userAdminPass", roles:
[ "userAdminAnyDatabase" ] } )
db.createUser( { user: "clusterAdmin", pwd: "$clusterAdminPass", roles:
[ "clusterAdmin" ] } )
db.createUser( { user: "backup", pwd: "$backupPass", roles: [ "readWrite", "backup",
"clusterMonitor", "restore", "pbmAnyAction" ] } )
EOF
```

YAML Object Format

The default name of the Secrets object for these users is `my-cluster-name-secrets` and can be set in the CR for your cluster in `spec.secrets.users` to something different. When you create the object yourself, the corresponding YAML file should match the following simple format:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-cluster-name-secrets
type: Opaque
stringData:
  MONGODB_BACKUP_USER: backup
  MONGODB_BACKUP_PASSWORD: backup123456
  MONGODB_DATABASE_ADMIN_USER: databaseAdmin
  MONGODB_DATABASE_ADMIN_PASSWORD: databaseAdmin123456
  MONGODB_CLUSTER_ADMIN_USER: clusterAdmin
```

```
  MONGODB_CLUSTER_ADMIN_PASSWORD: clusterAdmin123456
  MONGODB_CLUSTER_MONITOR_USER: clusterMonitor
  MONGODB_CLUSTER_MONITOR_PASSWORD: clusterMonitor123456
  MONGODB_USER_ADMIN_USER: userAdmin
  MONGODB_USER_ADMIN_PASSWORD: userAdmin123456
  PMM_SERVER_USER: admin
  PMM_SERVER_PASSWORD: admin
  PMM_SERVER_API_KEY: apikey
```

The example above matches what is shipped in `deploy/secrets.yaml` which contains default passwords and default API key. You should NOT use these in production, but they are present to assist in automated testing or simple use in a development environment.

As you can see, because we use the `stringData` type when creating the Secrets object, all values for each key/value pair are stated in plain text format convenient from the user's point of view. But the resulting Secrets object contains passwords stored as `data` - i.e., base64-encoded strings. If you want to update any field, you'll need to encode the value into base64 format. To do this, you can run `echo -n "password" | base64 --wrap=0` (or just `echo -n "password" | base64` in case of Apple macOS) in your local shell to get valid values. For example, setting the Database Admin user's password to `new_password` in the `my-cluster-name-secrets` object can be done with the following command:

in Linux

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

in macOS

```
$ kubectl patch secret/my-cluster-name-secrets -p '{"data":
{"MONGODB_DATABASE_ADMIN_PASSWORD": "'$(echo -n new_password | base64)'"}}'
```

Note

The operator creates and updates an additional Secrets object named based on the cluster name, like `internal-my-cluster-name-users`. It is used only by the Operator and should undergo no manual changes by the user. This object contains secrets with the same passwords as the one specified in `spec.secrets.users` (e.g. `my-cluster-name-secrets`). When the user updates `my-cluster-name-secrets`, the Operator propagates these changes to the internal `internal-my-cluster-name-users` Secrets object.

Password Rotation Policies and Timing

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

Note

Please don't change `secrets.users` option in CR, make changes inside the secrets object itself.

12.1.3 Development Mode

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

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

Secret Key	Secret Value
MONGODB_BACKUP_USER	backup
MONGODB_BACKUP_PASSWORD	backup123456
MONGODB_DATABASE_ADMIN_USER	databaseAdmin
MONGODB_DATABASE_ADMIN_PASSWORD	databaseAdmin123456
MONGODB_CLUSTER_ADMIN_USER	clusterAdmin
MONGODB_CLUSTER_ADMIN_PASSWORD	clusterAdmin123456
MONGODB_CLUSTER_MONITOR_USER	clusterMonitor
MONGODB_CLUSTER_MONITOR_PASSWORD	clusterMonitor123456
MONGODB_USER_ADMIN_USER	userAdmin
MONGODB_USER_ADMIN_PASSWORD	userAdmin123456
PMM_SERVER_USER	admin
PMM_SERVER_PASSWORD	admin
PMM_SERVER_API_KEY	apikey

Warning

Do not use the default MongoDB Users and/or default PMM API key in production!

12.1.4 MongoDB Internal Authentication Key (optional)

Default Secret name: `my-cluster-name-mongodb-key`

Secret name field: `spec.secrets.key`

By default, the operator will create a random, 1024-byte key for MongoDB Internal Authentication if it does not already exist. If you would like to deploy a different key, create the secret manually before starting the operator.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

12.2 Changing MongoDB Options

You may require a configuration change for your application. MongoDB allows configuring the database with a configuration file, as many other database management systems do. You can pass options to MongoDB instances in the cluster in one of the following ways:

- edit the `deploy/cr.yaml` file,
- use a ConfigMap,
- use a Secret object.

You can pass configuration settings separately for **mongod** Pods, **mongos** Pods, and **Config Server** Pods.

Often there's no need to add custom options, as the Operator takes care of providing MongoDB with reasonable defaults. Also, attempt to change some MongoDB options will be ignored: you can't change TLS/SSL options, as it would break the behavior of the Operator.

12.2.1 Edit the `deploy/cr.yaml` file

You can add MongoDB configuration options to the replsets.configuration, sharding.mongos.configuration, and sharding-configsvrreplset-configuration keys of the `deploy/cr.yaml`. Here is an example:

```
spec:
  ...
  replsets:
    - name: rs0
      size: 3
      configuration: |
        operationProfiling:
          mode: slowOp
        systemLog:
          verbosity: 1
      ...
```

See the official manual for the complete list of options, as well as specific Percona Server for MongoDB documentation pages.

12.2.2 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.

You should give the ConfigMap a specific name, which is composed of your cluster name and a specific suffix:

- `my-cluster-name-rs0-mongod` for the Replica Set (mongod) Pods,
- `my-cluster-name-cfg-mongod` for the Config Server Pods,
- `my-cluster-name-mongos` for the mongos Pods,

Note

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

```
$ kubectl get psmdb
```

For example, let's define a `mongod.conf` configuration file and put there several MongoDB options we used in the previous example:

```
operationProfiling:
  mode: slowOp
systemLog:
  verbosity: 1
```

You can create a ConfigMap from the `mongod.conf` file with the `kubectl create configmap` command. It has the following syntax:

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

The following example defines `my-cluster-name-rs0-mongod` as the ConfigMap name and the `mongod.conf` file as the data source:

```
$ kubectl create configmap my-cluster-name-rs0-mongod --from-file=mongod.conf=mongod.conf
```

To view the created ConfigMap, use the following command:

```
$ kubectl describe configmaps my-cluster-name-rs0-mongod
```

Note

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

12.2.3 Use a Secret Object

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

You should create a Secret object with a specific name, composed of your cluster name and a specific suffix:

- `my-cluster-name-rs0-mongod` for the Replica Set Pods,
- `my-cluster-name-cfg-mongod` for the Config Server Pods,
- `my-cluster-name-mongos` for the mongos Pods,

Note

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

```
$ kubectl get psmdb
```

Configuration options should be put inside a specific key:

- `data.mongod` key for Replica Set (mongod) and Config Server Pods,
- `data.mongos` key for mongos Pods.

Actual options should be encoded with Base64.

For example, let's define a `mongod.conf` configuration file and put there several MongoDB options we used in the previous example:

```
operationProfiling:
  mode: slowOp
systemLog:
  verbosity: 1
```

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

in Linux

```
$ cat mongod.conf | base64 --wrap=0
```

in macOS

```
$ cat mongod.conf | base64
```

Note

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

```
$ echo "ICAgICAgb3BlcmF0aW9uUHJvZmlsaW5nOgogICAgICAgIG1vZGU6IHNsb3dPc\
AogICAgICBzeXN0ZW1Mb2c6CiAgICAgICAgdmVyYm9zaXR5OiAxCg==" | base64 --decode
```

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

```
apiVersion: v1
kind: Secret
metadata:
  name: my-cluster-name-rs0-mongod
data:
  mongod.conf: "ICAgICAgb3BlcmF0aW9uUHJvZmlsaW5nOgogICAgICAgIG1vZGU6IHNsb3dPc\
   AogICAgICBzeXN0ZW1Mb2c6CiAgICAgICAgdmVyYm9zaXR5OiAxCg=="
```

When ready, apply it with the following command:

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

Note

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

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-03

12.3 Binding Percona Server for MongoDB components to Specific Kubernetes/OpenShift Nodes

The operator does a good job of automatically assigning new pods to nodes to achieve balanced distribution across the cluster. There are situations when you must ensure that pods land on specific nodes: for example, for the advantage of speed on an SSD-equipped machine, or reduce costs by choosing nodes in the same availability zone.

The appropriate (sub)sections (`replsets`, `replsets.arbiter`, `backup`, etc.) of the deploy/cr.yaml file contain the keys which can be used to do assign pods to nodes.

12.3.1 Node selector

The `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
```

12.3.2 Affinity and anti-affinity

Affinity defines eligible pods that can be scheduled on the node which already has pods with specific labels. Anti-affinity defines pods that are not eligible. This approach is reduces costs by ensuring several pods with intensive data exchange occupy the same availability zone or even the same node or, on the contrary, to spread the pods on different nodes or even different availability zones for high availability and balancing purposes.

Percona Operator for MongoDB 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 antiAffinityTopologyKey of the Percona Operator for MongoDB

Percona Operator for MongoDB provides an `antiAffinityTopologyKey` 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 Server for MongoDB Pods to avoid occupying the same node:

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

Advanced approach - use standard Kubernetes constraints

The previous method can be used without 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, the `advanced` option placed in the deploy/cr.yaml file turns off the effect of the `antiAffinityTopologyKey` and allows the use of the 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.

12.3.3 Topology Spread Constraints

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

Pod topology spread constraints are controlled by the `topologySpreadConstraints` subsection, which can be put into `replsets`, `sharding.configsvrReplSet`, and `sharding.mongos` sections of the `deploy/cr.yaml` configuration file as follows:

```
topologySpreadConstraints:
  - labelSelector:
      matchLabels:
        app.kubernetes.io/name: percona-server-mongodb
    maxSkew: 1
```

```
topologyKey: kubernetes.io/hostname
whenUnsatisfiable: DoNotSchedule
```

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

12.3.4 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 equal variant requires a corresponding `value` for comparison).

Toleration should have a specified `effect`, such as the following:

- `NoSchedule` - less strict
- `PreferNoSchedule`
- `NoExecute`

When a *taint* with the `NoExecute` effect is assigned to a Node, any Pod configured to not tolerating this *taint* is removed from the node. This removal can be immediate or after the `tolerationSeconds` interval. The following example defines this effect and the removal interval:

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

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

12.3.5 Priority Classes

Pods may belong to some *priority classes*. This flexibility allows the scheduler to distinguish more and less important Pods when needed, such as the situation when a higher priority Pod cannot be scheduled without evicting a lower priority one. This ability can be accomplished by 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.

12.3.6 Pod Disruption Budgets

Creating the Pod Disruption Budget is the Kubernetes method to limit the number of Pods of an application that can go down simultaneously due to *voluntary disruptions* such as the cluster administrator's actions during a deployment update. Distribution Budgets allow large applications to retain their high availability during maintenance and other administrative activities. The `maxUnavailable` and `minAvailable` options in the deploy/cr.yaml file can be used to set these limits. The recommended variant is the following:

```
podDisruptionBudget:
   maxUnavailable: 1
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-05

12.4 Labels and annotations

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

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

12.4.1 Setting labels and annotations in the Custom Resource

You can set labels and/or annotations as key/value string pairs in the Custom Resource metadata section of the `deploy/cr.yaml` as follows:

```yaml
apiVersion: psmdb.percona.com/v1
kind: PerconaServerMongoDB
metadata:
  name: my-cluster-name
  annotations:
    percona.com/issue-vault-token: "true"
  labels:
    ...
```

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

```
$ kubectl get pod my-cluster-name-rs0-0 -o jsonpath='{.metadata.annotations}'
```

12.4.2 Using labels and annotations with objects created by the Operator

You can assign labels and annotations to various objects created by the Operator (e.g. Services used to expose components of the cluster, Persistent Volume Claims, etc.) with labels and annotations options in the appropriate subsections of the Custom Resource, as seen in the Custom Resource options reference and the deploy/cr.yaml configuration file.

Sometimes various Kubernetes flavors can add their own annotations to the objects managed by the Operator.

The Operator keeps track of all changes to its objects and can remove annotations that appeared without its participation.

If there are no annotations or labels in the Custom Resource expose subsections, the Operator does nothing if a new label or annotation is added to the object.

If the Service per Pod mode is not used, the Operator **won't remove any annotations and labels** from any Services related to *this expose subsection*. Though, it is still possible to add annotations and labels via the Custom Resource in this case. Use the appropriate `expose.serviceAnnotations` and `expose.serviceLabels` fields.

Else, if the Service per Pod mode is active, the Operator removes unknown annotations and labels from Services *created by the Operator for Pods*. Yet it is still possible to specify which annotations and labels should be preserved (not wiped out) by the Operator. List them in the `spec.ignoreAnnotations` or `spec.ignoreLabels` fields of the `deploy/cr.yaml`, as follows:

```yaml
spec:
  ignoreAnnotations:
    - some.custom.cloud.annotation/smth
  ignoreLabels:
    - some.custom.cloud.label/smth
...
```

The Operator will keep any Service annotation or label, key of which **starts** with the specified string. For example, the following annotations and labels will be **not removed** after applying the above `cr.yaml` fragment:

```yaml
kind: Service
apiVersion: v1
metadata:
  name: my-cluster-name-cfg
  ...
  labels:
    some.custom.cloud.label/smth: somethinghere
    ...
  annotations:
    some.custom.cloud.annotation/smth: somethinghere
    ...
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-08

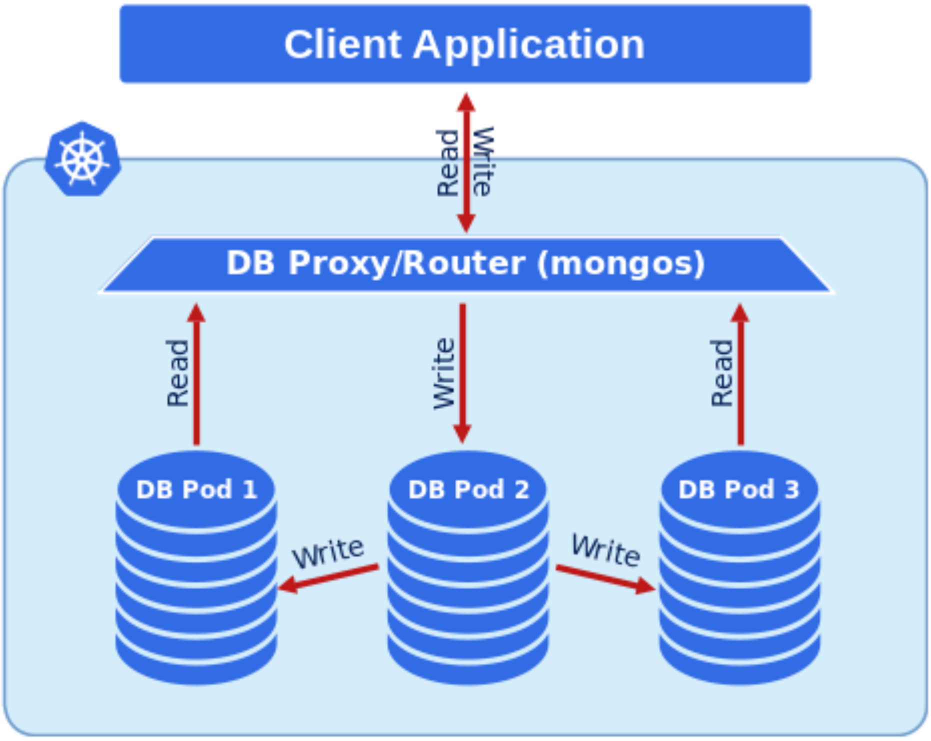
12.5 Exposing cluster

The Operator provides entry points for accessing the database by client applications in several scenarios. In either way the cluster is exposed with regular Kubernetes Service objects, configured by the Operator.

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

12.5.1 Using single entry point in a sharded cluster

If Percona Server for MongoDB Sharding mode is turned **on** (default behavior), then database cluster runs special `mongos` Pods - query routers, which acts as an entry point for client applications,



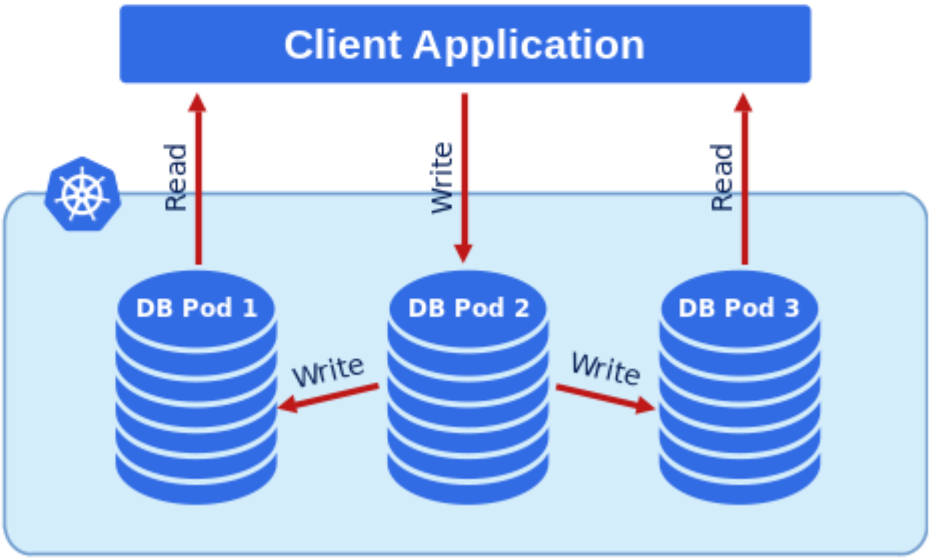
If this feature is enabled, the URI looks like follows (taking into account the need in a proper password obtained from the Secret, and a proper namespace name instead of the `<namespace name>` placeholder):

```
$ mongo "mongodb://userAdmin:userAdminPassword@my-cluster-name-mongos.<namespace
name>.svc.cluster.local/admin?ssl=false"
```

You can find more on sharding in the official MongoDB documentation.

12.5.2 Accessing replica set Pods

If Percona Server for MongoDB Sharding mode is turned **off**, the application needs access to all MongoDB Pods of the replica set:



When Kubernetes creates Pods, each Pod has an IP address in the internal virtual network of the cluster. Creating and destroying Pods is a dynamic process, therefore binding communication between Pods to specific IP addresses would cause problems as things change over time as a result of the cluster scaling, maintenance, etc. Due to this changing environment, you should connect to Percona Server for MongoDB via Kubernetes internal DNS names in URI (e.g. using `mongodb+srv://userAdmin:userAdmin123456@<cluster-name>-rs0.<namespace>.svc.cluster.local/admin?replicaSet=rs0&ssl=false` to access one of the Replica Set Pods).

In this case, the URI looks like follows (taking into account the need in a proper password obtained from the Secret, and a proper namespace name instead of the `<namespace name>` placeholder):

```
$ mongodb://databaseAdmin:databaseAdminPassword@my-cluster-name-rs0.<namespace
name>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
```

12.5.3 Service per Pod

URI-based access is strictly recommended.

Still sometimes you cannot communicate with the Pods using the Kubernetes internal DNS names. To make Pods of the Replica Set accessible, Percona Operator for MongoDB can assign a Kubernetes Service to each Pod.

This feature can be configured in the `replsets` (for MondgoDB instances Pod) and `sharding` (for mongos Pod) sections of the deploy/cr.yaml file:

- set `expose.enabled` option to `true` to allow exposing Pods via services,
- set `expose.exposeType` option specifying the IP address type to be used:
  - `ClusterIP` - expose the Pod's service with an internal static IP address. This variant makes MongoDB Pod only reachable from within the Kubernetes cluster.
  - `NodePort` - expose the Pod's service on each Kubernetes node's IP address at a static port. ClusterIP service, to which the node port will be routed, is automatically created in this variant. As an advantage, the service will be reachable from outside the cluster by node address and port number, but the address will be bound to a specific Kubernetes node.
  - `LoadBalancer` - expose the Pod's service externally using a cloud provider's load balancer. Both ClusterIP and NodePort services are automatically created in this variant.

If this feature is enabled, URI looks like `mongodb://databaseAdmin:databaseAdminPassword@<ip1>:<port1>,<ip2>:<port2>,<ip3>:<port3>/admin?replicaSet=rs0&ssl=false` All IP adresses should be *directly* reachable by application.

12.5.4 Controlling hostnames in replset configuration

Starting from v1.14, the Operator configures replica set members using local fully-qualified domain names (FQDN), which are resolvable and available only from inside the Kubernetes cluster. Exposing the replica set using the options described above will not affect hostname usage in the replica set configuration.

Note

Before v1.14, the Operator used the exposed IP addresses in the replica set configuration in the case of the exposed replica set.

It is still possible to restore the old behavior. For example, it may be useful to have the replica set configured with external IP addresses for multi-cluster deployments. The `clusterServiceDNSMode` field in the Custom Resource controls this Operator behavior. You can set `clusterServiceDNSMode` to one of the following values:

1. **`Internal`**: Use local FQDNs (i.e., `cluster1-rs0-0.cluster1-rs0.psmdb.svc.cluster.local`) in replica set configuration even if the replica set is exposed. **This is the default value.**
2. **`ServiceMesh`**: Use a special FQDN using the Pod name (i.e., `cluster1-rs0-0.psmdb.svc.cluster.local`), assuming it's resolvable and available in all clusters.
3. **`External`**: Use exposed IP in replica set configuration if replica set is exposed; else, use local FQDN. **This copies the behavior of the Operator v1.13.**

If backups are enabled in your cluster, you need to restart replset and config servers after changing `clusterServiceDNSMode`. This option changes the hostnames inside the replset configuration and running pbm-agents don't discover the change until they're restarted. You may have errors in `backup-agent` container logs and your backups may not work until you restarted the agents.

Restart can be done manually with the `kubectl rollout restart sts <clusterName>-<replsetName>` command executed for each replica set in the `spec.replsets`; also, if sharding enabled, do the same for config servers with `kubectl rollout restart sts <clusterName>-cfg`. Alternatively, you can simply restart your cluster.

Warning

You should be careful with the `clusterServiceDNSMode=External` variant. Using IP addresses instead of DNS hostnames is discouraged in MongoDB. IP addresses make configuration changes and recovery more complicated. Also, they are particularly problematic in scenarios where IP addresses change (i.e., deleting and recreating the cluster).

12.5.5 Exposing replica set with split-horizon DNS

Split-horizon DNS provides each replica set Pod with a set of DNS URIs for external usage. This allows to communicate with replica set Pods both from inside the Kubernetes cluster and from outside of Kubernetes.

Split-horizon can be configured via the `replset.horizons` subsection in the Custom Resource options. Set it in the `deploy/cr.yaml` configuration file as follows:

```
...
replsets:
  - name: rs0
```

```
      expose:
        enabled: true
        exposeType: LoadBalancer
      horizons:
        cluster1-rs0-0:
          external: rs0-0.mycluster.xyz
          external-2: rs0-0.mycluster2.xyz
        cluster1-rs0-1:
          external: rs0-1.mycluster.xyz
          external-2: rs0-1.mycluster2.xyz
        cluster1-rs0-2:
          external: rs0-2.mycluster.xyz
          external-2: rs0-2.mycluster2.xyz
```

URIs for external usage are specified as key-value pairs, where the key is an arbitrary name and the value is the actual URI.

Split horizon has following limitations:

- connecting with horizon domains is only supported if client connects using TLS certificates
- duplicating domain names in horizons is not allowed by MongoDB
- using IP addresses in horizons is not allowed by MongoDB
- horizons should be set for *all Pods of a replica set* or not set at all
- horizons should be configured on an existing cluster (creating a new cluster with pre-configured horizons is currently not supported)

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

12.6 Local Storage support for the Percona Operator for MongoDB

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

12.6.1 emptyDir

A Pod emptyDir volume is created when the Pod is assigned to a Node. The volume is initially empty and is erased when the Pod is removed from the Node. The containers in the Pod can read and write the files in the emptyDir volume.

The `emptyDir` options in the deploy/cr.yaml file can be used to turn the emptyDir volume on by setting the directory name.

The `emptyDir` is useful when you use Percona Memory Engine.

12.6.2 hostPath

A hostPath volume mounts an existing file or directory from the host node's filesystem into the Pod. If the pod is removed, the data persists in the host node's filesystem.

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, you must created the hostPath manually and should have following attributes:

- access permissions,
- ownership,
- SELinux security context.

The `hostPath` volume is useful when you perform manual actions during the first run and require improved disk performance. Consider using the tolerations settings to avoid a 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.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-18

12.7 Using Replica Set Arbiter nodes and non-voting nodes

Percona Server for MongoDB replication model is based on elections, when nodes of the Replica Set choose which node becomes the primary node.

The need for elections influences the choice of the number of nodes in the cluster. Elections are the reason to avoid even number of nodes, and to have at least three and not more than seven participating nodes.

Still, sometimes there is a contradiction between the number of nodes suitable for elections and the number of nodes needed to store data. You can solve this contradiction in two ways:

- Add *Arbiter* nodes, which participate in elections, but do not store data,
- Add *non-voting* nodes, which store data but do not participate in elections.

12.7.1 Adding Arbiter nodes

Normally, each node stores a complete copy of the data, but there is also a possibility, to reduce disk IO and space used by the database, to add an arbiter node. An arbiter cannot become a primary and does not have a complete copy of the data. The arbiter does have one election vote and can be the odd number for elections. The arbiter does not demand a persistent volume.

Percona Operator for MongoDB has the ability to create Replica Set Arbiter nodes if needed. This feature can be configured in the Replica Set section of the deploy/cr.yaml file:

- set `arbiter.enabled` option to `true` to allow Arbiter instances,
- use `arbiter.size` option to set the desired amount of Arbiter instances.

For example, the following keys in `deploy/cr.yaml` will create a cluster with 4 data instances and 1 Arbiter:

```
....
replsets:
  ....
  size: 4
  ....
  arbiter:
    enabled: true
    size: 1
    ....
```

Note

You can find description of other possible options in the replsets.arbiter section of the Custom Resource options reference.

12.7.2 Adding non-voting nodes

Non-voting member is a Replica Set node which does not participate in the primary election process. This feature is required to have more than 7 nodes, or if there is a node in the edge location, which obviously should not participate in the voting process.

Note

Non-voting nodes support has technical preview status and is not recommended for production environments.

Note

It is possible to add a non-voting node in the edge location through the `externalNodes` option. Please see cross-site replication documentation for details.

Percona Operator for MongoDB has the ability to configure non-voting nodes in the Replica Set section of the deploy/cr.yaml file:

- set `nonvoting.enabled` option to `true` to allow non-voting instances,
- use `nonvoting.size` option to set the desired amount of non-voting instances.

For example, the following keys in `deploy/cr.yaml` will create a cluster with 3 data instances and 1 non-voting instance:

```
....
replsets:
  ....
  size: 3
  ....
  nonvoting:
    enabled: true
    size: 1
    ....
```

Note

You can find description of other possible options in the replsets.nonvoting section of the Custom Resource options reference.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-18

12.8 Percona Server for MongoDB Sharding

12.8.1 About sharding

Sharding provides horizontal database scaling, distributing data across multiple MongoDB Pods. It is useful for large data sets when a single machine's overall processing speed or storage capacity turns out to be not enough. Sharding allows splitting data across several machines with a special routing of each request to the necessary subset of data (so-called *shard*).

A MongoDB Sharding involves the following components:

- `shard` - a replica set which contains a subset of data stored in the database (similar to a traditional MongoDB replica set),
- `mongos` - a query router, which acts as an entry point for client applications,
- `config servers` - a replica set to store metadata and configuration settings for the sharded database cluster.

Note

Percona Operator for MongoDB 1.6.0 supported only one shard of a MongoDB cluster; still, this limited sharding support allowed using `mongos` as an entry point instead of provisioning a load-balancer per replica set node. Multiple shards are supported starting from the Operator 1.7.0. Also, before the Operator 1.12.0 mongos were deployed by the Deployment object, and starting from 1.12.0 they are deployed by the StatefulSet one.

12.8.2 Turning sharding on and off

Sharding is controlled by the `sharding` section of the `deploy/cr.yaml` configuration file and is turned on by default.

To enable sharding, set the `sharding.enabled` key to `true` (this will turn existing MongoDB replica set nodes into sharded ones). To disable sharding, set the `sharding.enabled` key to `false`.

When sharding is turned on, the Operator runs replica sets with config servers and mongos instances. Their number is controlled by `configsvrReplSet.size` and `mongos.size` keys, respectively.

Config servers have `cfg` replica set name by default, which is used by the Operator in StatefulSet and Service names. If this name needs to be customized (for example when migrating MongoDB cluster from barebone installation to Kubernetes), you can override the default `cfg` variant using `replsets.configuration` Custom Resource option in `deploy/cr.yaml` as follows:

```
...
configuration: |
  replication:
    replSetName: customCfgRS
    ...
```

Note

Config servers for now can properly work only with WiredTiger engine, and sharded MongoDB nodes can use either WiredTiger or InMemory one.

By default replsets section of the `deploy/cr.yaml` configuration file contains only one replica set, `rs0`. You can add more replica sets with different names to the `replsets` section in a similar way. Please take into account that having more than one replica set is possible only with the sharding turned on.

> **Note**
>
> The Operator will be able to remove a shard only when it contains no application (non-system) collections.

12.8.3 Checking connectivity to sharded and non-sharded cluster

With sharding turned on, you have `mongos` service as an entry point to access your database. If you do not use sharding, you have to access `mongod` processes of your replica set.

1. You will need the login and password for the admin user to access the cluster. Use `kubectl get secrets` command to see the list of Secrets objects (by default the Secrets object you are interested in has `my-cluster-name-secrets` name). Then `kubectl get secret my-cluster-name-secrets -o yaml` command will return the YAML file with generated Secrets, including the `MONGODB_DATABASE_ADMIN_USER` and `MONGODB_DATABASE_ADMIN_PASSWORD` strings, which should look as follows:

```
...
data:
  ...
  MONGODB_DATABASE_ADMIN_PASSWORD: aDAzQ0pCY3NSWEZ2ZUIzS1I=
  MONGODB_DATABASE_ADMIN_USER: ZGF0YWJhc2VBZG1pbg==
```

   Here the actual login name and password are base64-encoded. Use `echo 'aDAzQ0pCY3NSWEZ2ZUIzS1I=' | base64 --decode` command to bring it back to a human-readable form.

2. Run a container with a MongoDB client and connect its console output to your terminal. The following command will do this, naming the new Pod `percona-client`:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:
4.4.24-23 --restart=Never -- bash -il
```

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

3. Now run `mongo` tool in the percona-client command shell using the login (which is normally `databaseAdmin`), a proper password obtained from the Secret, and a proper namespace name instead of the `<namespace name>` placeholder. The command will look different depending on whether sharding is on (the default behavior) or off:

if sharding is on

```
$ mongosh "mongodb://databaseAdmin:databaseAdminPassword@my-cluster-name-
mongos.<namespace name>.svc.cluster.local/admin?ssl=false"
```

if sharding is off

```
$ mongosh "mongodb+srv://databaseAdmin:databaseAdminPassword@my-cluster-name-
rs0.<namespace name>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
```

> **Note**
>
> If using MongoDB versions earler than 6.x (such as 4.4.24-23 or 5.0.20-17 instead of the default 6.0.9-7 variant), substitute `mongosh` command with `mongo` in the above example.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

12.9 Transport Layer Security (TLS)

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

- Internal - communication between Percona Server for MongoDB instances in the cluster
- External - communication between the client application and the cluster

The internal certificate is also used as an authorization method.

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

- the Operator can use a specifically installed *cert-manager*, which will automatically generate and renew short-term TLS certificates,
- 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.

12.9.1 Install and use the *cert-manager*

About the *cert-manager*

The 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 Percona Server for MongoDB requires all certificates issued by the same CA (Certificate authority).

Self-signed issuer allows you to deploy and use the Percona Operator without creating a cluster issuer separately.

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 apply -f https://github.com/jetstack/cert-manager/releases/download/v1.12.4/cert-
manager.yaml --validate=false
```

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:

```
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-7d59dd4888-tmjqq              1/1     Running   0          3m8s
cert-manager-cainjector-85899d45d9-8ncw9   1/1     Running   0          3m8s
cert-manager-webhook-84fcdcd5d-697k4       1/1     Running   0          3m8s
```

Once you create the database with the Operator, it will automatically trigger cert-manager to create certificates. Whenever you check certificates for expiration, you will find that they are valid and short-term.

12.9.2 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 the `spec.secrets.ssl` key of the `deploy/cr.yaml` file. A certificate generated for internal communications must be added to the `spec.secrets.sslInternal` key of the `deploy/cr.yaml` file.

Note

If you only create the external certificate, then the Operator will not generate the internal one, but instead use certificate you have provided for both external and internal communications.

Supposing that your cluster name is `my-cluster-name`, the instructions to generate certificates manually are as follows:

```
$ CLUSTER_NAME=my-cluster-name
$ NAMESPACE=default
$ cat <<EOF | cfssl gencert -initca - | cfssljson -bare ca
  {
    "CN": "Root CA",
    "names": [
      {
        "O": "PSMDB"
      }
    ],
    "key": {
      "algo": "rsa",
      "size": 2048
    }
  }
EOF

$ cat <<EOF > ca-config.json
  {
```

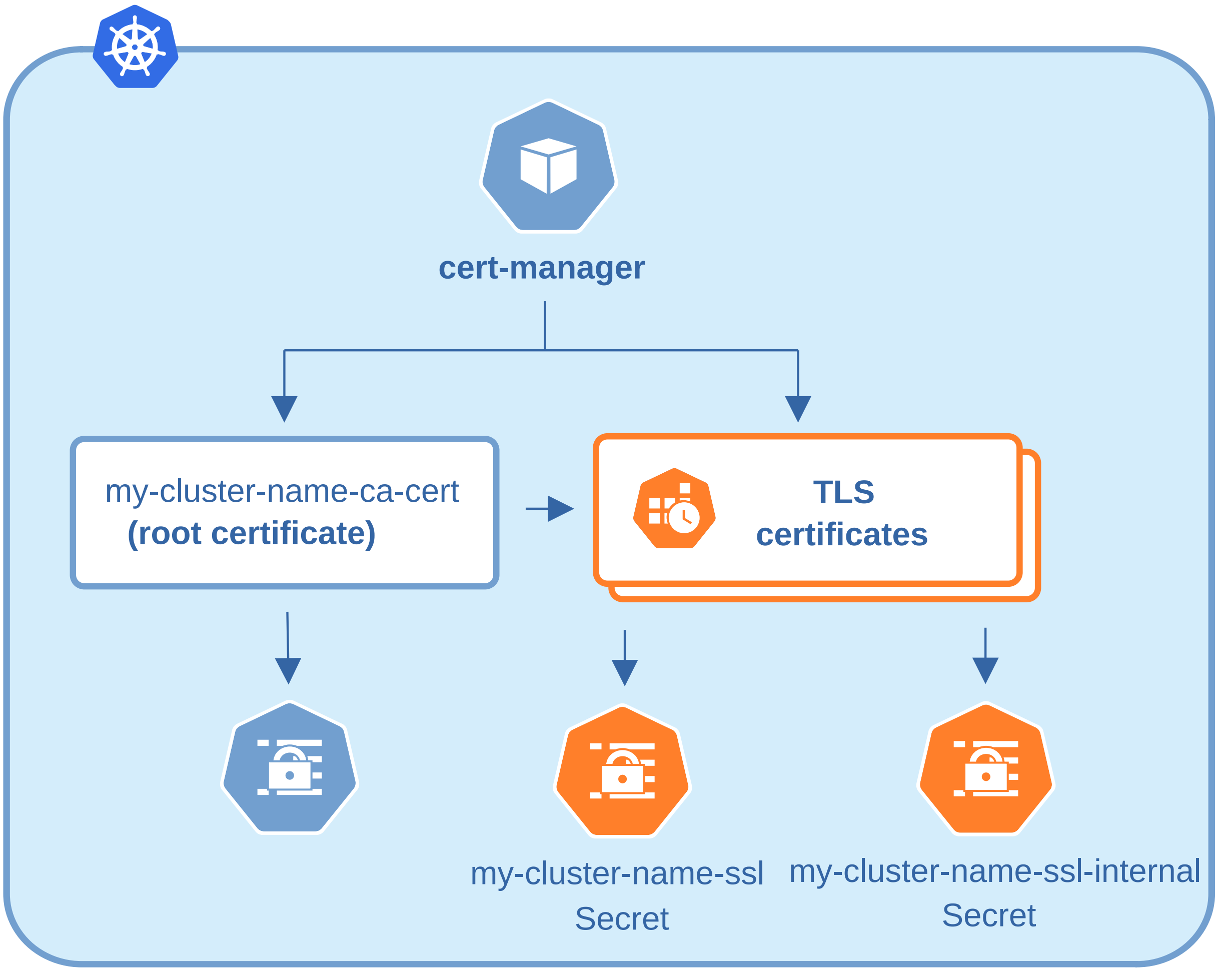
```
    "signing": {
      "default": {
        "expiry": "87600h",
        "usages": ["signing", "key encipherment", "server auth", "client auth"]
      }
    }
  }
EOF

$ cat <<EOF | cfssl gencert -ca=ca.pem  -ca-key=ca-key.pem -config=./ca-config.json - |
cfssljson -bare server
  {
    "hosts": [
      "localhost",
      "${CLUSTER_NAME}-rs0",
      "${CLUSTER_NAME}-rs0.${NAMESPACE}",
      "${CLUSTER_NAME}-rs0.${NAMESPACE}.svc.cluster.local",
      "*.${CLUSTER_NAME}-rs0",
      "*.${CLUSTER_NAME}-rs0.${NAMESPACE}",
      "*.${CLUSTER_NAME}-rs0.${NAMESPACE}.svc.cluster.local"
    ],
    "names": [
      {
        "O": "PSMDB"
      }
    ],
    "CN": "${CLUSTER_NAME/-rs0}",
    "key": {
      "algo": "rsa",
      "size": 2048
    }
  }
EOF
$ cfssl bundle -ca-bundle=ca.pem -cert=server.pem | cfssljson -bare server

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

$ cat <<EOF | cfssl gencert -ca=ca.pem  -ca-key=ca-key.pem -config=./ca-config.json - |
cfssljson -bare client
  {
    "hosts": [
      "${CLUSTER_NAME}-rs0",
      "${CLUSTER_NAME}-rs0.${NAMESPACE}",
      "${CLUSTER_NAME}-rs0.${NAMESPACE}.svc.cluster.local",
      "*.${CLUSTER_NAME}-rs0",
      "*.${CLUSTER_NAME}-rs0.${NAMESPACE}",
      "*.${CLUSTER_NAME}-rs0.${NAMESPACE}.svc.cluster.local"
    ],
    "names": [
      {
        "O": "PSMDB"
      }
    ],
    "CN": "${CLUSTER_NAME/-rs0}",
    "key": {
      "algo": "rsa",
      "size": 2048
    }
  }
EOF
```

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

12.9.3 Update certificates

If a cert-manager is used, it should take care of updating the certificates. If you generate certificates manually, you should take care of updating them in proper time.

TLS certificates issued by cert-manager are short-term ones, valid for 3 months. They are reissued automatically on schedule and without downtime.



Versions of the Operator prior 1.9.0 have used 3 month root certificate, which caused issues with the automatic TLS certificates update. If that's your case, you can make the Operator update along with the official instruction.

Note

If you use the cert-manager version earlier than 1.9.0, and you would like to avoid downtime while updating the certificates after the Operator update to 1.9.0 or newer version, force the certificates regeneration by a cert-manager.

Check your certificates for expiration

1. First, check the necessary secrets names (`my-cluster-name-ssl` and `my-cluster-name-ssl-internal` by default):

    ```
    $ kubectl get certificate
    ```

    You will have the following response:

    ```
    NAME                          READY   SECRET                        AGE
    my-cluster-name-ssl           True    my-cluster-name-ssl           49m
    my-cluster-name-ssl-internal  True    my-cluster-name-ssl-internal  49m
    ```

2. Optionally you can also check that the certificates issuer is up and running:

    ```
    $ kubectl get issuer
    ```

    The response should be as follows:

    ```
    NAME                             READY   AGE
    my-cluster-name-psmdb-issuer     True    61m
    my-cluster-name-psmdb-ca-issuer  True    61m
    ```

    Note

    The presence of two issuers has the following meaning. The `my-cluster-name-psmdb-ca-issuer` issuer is used to create a self signed CA certificate (`my-cluster-name-ca-cert`), and then the `my-cluster-name-psmdb-issuer` issuer is used to create SSL certificates (`my-cluster-name-ssl` and `my-cluster-name-ssl-internal`) signed by the `my-cluster-name-ca-cert` CA certificate.

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

    ```
    $ {
      kubectl get secret/my-cluster-name-ssl-internal -o jsonpath='{.data.tls\.crt}' | base64 --decode | openssl x509 -noout -dates
      kubectl get secret/my-cluster-name-ssl -o jsonpath='{.data.ca\.crt}' | base64 --decode | openssl x509 -noout -dates
      }
    ```

    The resulting output will be self-explanatory:

    ```
    notBefore=Apr 25 12:09:38 2022 GMT notAfter=Jul 24 12:09:38 2022 GMT
    notBefore=Apr 25 12:09:38 2022 GMT notAfter=Jul 24 12:09:38 2022 GMT
    ```

Update certificates without downtime

If you don’t use cert-manager and have *created certificates manually*, you can follow the next steps to perform a no-downtime update of these certificates *if they are still valid*.

Note

For already expired certificates, follow the alternative way.

Having non-expired certificates, you can roll out new certificates (both CA and TLS) with the Operator as follows.

1. Generate a new CA certificate (`ca.pem`). Optionally you can also generate a new TLS certificate and a key for it, but those can be generated later on step 6.
2. Get the current CA (`ca.pem.old`) and TLS (`tls.pem.old`) certificates and the TLS certificate key (`tls.key.old`):

```
$ kubectl get secret/my-cluster-name-ssl-internal -o jsonpath='{.data.ca\.crt}' | base64
--decode > ca.pem.old
$ kubectl get secret/my-cluster-name-ssl-internal -o jsonpath='{.data.tls\.crt}' | base64
--decode > tls.pem.old
$ kubectl get secret/my-cluster-name-ssl-internal -o jsonpath='{.data.tls\.key}' | base64
--decode > tls.key.old
```

3. Combine new and current `ca.pem` into a `ca.pem.combined` file:

```
$ cat ca.pem ca.pem.old >> ca.pem.combined
```

4. Create a new Secrets object with *old* TLS certificate (`tls.pem.old`) and key (`tls.key.old`), but a *new combined* `ca.pem` (`ca.pem.combined`):

```
$ kubectl delete secret/my-cluster-name-ssl-internal
$ kubectl create secret generic my-cluster-name-ssl-internal --from-
file=tls.crt=tls.pem.old --from-file=tls.key=tls.key.old --from-
file=ca.crt=ca.pem.combined --type=kubernetes.io/tls
```

5. The cluster will go through a rolling reconciliation, but it will do it without problems, as every node has old TLS certificate/key, and both new and old CA certificates.
6. If new TLS certificate and key weren't generated on step 1, do that now.
7. Create a new Secrets object for the second time: use new TLS certificate (`server.pem` in the example) and its key (`server-key.pem`), and again the combined CA certificate (`ca.pem.combined`):

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

8. The cluster will go through a rolling reconciliation, but it will do it without problems, as every node already has a new CA certificate (as a part of the combined CA certificate), and can successfully allow joiners with new TLS certificate to join. Joiner node also has a combined CA certificate, so it can authenticate against older TLS certificate.
9. Create a final Secrets object: use new TLS certificate (`server.pmm`) and its key (`server-key.pem`), and just the new CA certificate (`ca.pem`):

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

10. The cluster will go through a rolling reconciliation, but it will do it without problems: the old CA certificate is removed, and every node is already using new TLS certificate and no nodes rely on the old CA certificate any more.

Update certificates with downtime

If your certificates have been already expired (or if you continue to use the Operator version prior to 1.9.0), you should move through the *pause* - *update Secrets* - *unpause* route as follows.

1. Pause the cluster in a standard way, and make sure it has reached its paused state.
2. If cert-manager is used, delete issuer and TLS certificates:

```
$ {
  kubectl delete issuer/my-cluster-name-psmdb-ca-issuer issuer/my-cluster-name-psmdb-
issuer
  kubectl delete certificate/my-cluster-name-ssl certificate/my-cluster-name-ssl-internal
  }
```

3. Delete Secrets to force the SSL reconciliation:

```
$ kubectl delete secret/my-cluster-name-ssl secret/my-cluster-name-ssl-internal
```

4. Check certificates to make sure reconciliation have succeeded.
5. Unpause the cluster in a standard way, and make sure it has reached its running state.

12.9.4 Run Percona Server for MongoDB 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) set the `spec.allowUnsafeConfigurations` key to `true` in the `deploy/cr.yaml` file and and make sure that there are no certificate secrets available. This is the only condition under which the cluster will work without TLS.

> **Warning**
>
> Normally, the Operator prevents users from configuring a cluster with unsafe parameters (starting it with less than 3 replica set instances or without TLS, etc.), automatically changing such unsafe parameters to safe defaults. If you switch the cluster to the *unsafe configurations permissive mode*, you will not be able to switch it back by setting `spec.allowUnsafeConfigurations` key to `false`, the flag will be ignored.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

12.10 Data at rest encryption

Data at rest encryption in Percona Server for MongoDB is supported by the Operator since version 1.1.0.

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

Data at rest encryption is turned on by default. The Operator implements it by either using encryption key stored in a Secret, or obtaining encryption key from the HashiCorp Vault key storage.

12.10.1 Using encryption key Secret

1. The `secrets.encryptionKey` key in the `deploy/cr.yaml` file should specify the name of the encryption key Secret:

   ```
   secrets:
     ...
     encryptionKey: my-cluster-name-mongodb-encryption-key
   ```

   Encryption key Secret will be created automatically by the Operator if it doesn't exist. If you would like to create it yourself, take into account that the key must be a 32 character string encoded in base64.

2. The `replsets.configuration`, `replsets.nonvoting.configuration`, and `sharding.configsvrReplSet.configuration` keys should include the following two MongoDB encryption-specific options:

   ```
   ...
   configuration: |
     ...
     security:
       enableEncryption: true
       encryptionCipherMode: "AES256-CBC"
       ...
   ```

   The `enableEncryption` option should be set to `true` (the default value). The `security.encryptionCipherMode` option should specify a proper cipher mode for decryption: either `AES256-CBC` (the default value) or `AES256-GCM`.

Don't forget to apply the modified `cr.yaml` configuration file as usual:

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

12.10.2 Using HashiCorp Vault storage for encryption keys

Starting from the version 1.13, the Operator supports using HashiCorp Vault storage for encryption keys - a universal, secure and reliable way to store and distribute secrets without depending on the operating system, platform or cloud provider.

> **Warning**
>
> Vault integration has technical preview status and is not yet recommended for production environments.

The Operator will use Vault if the `deploy/cr.yaml` configuration file contains the following items:

- a `secrets.vault` key equal to the name of a specially created Secret,
- `configuration` keys for mongod and config servers with a number of Vault-specific options.

The Operator itself neither installs Vault, nor configures it; both operations should be done manually, as described in the following parts.

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. Read more about installation in Vault's documentation.

1. Add helm repo and install:

```
$ helm repo add hashicorp https://helm.releases.hashicorp.com
"hashicorp" has been added to your repositories

$ helm install vault hashicorp/vault
```

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

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

Configuring Vault

1. First, you should enable secrets within Vault. For this you will need a Vault token. Percona Server for MongoDB can use any regular token which allows all operations inside the secrets mount point. In the following example we are using the *root token* to be sure the permissions requirement is met, but actually there is no need in root permissions. We don't recommend using the root token on the production system.

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

The output will show you the token:

```
s.VgQvaXl8xGFO1RUxAPbPbsfN
```

Now login to Vault with this token to enable the key-value secret engine:

```
$ kubectl exec -it vault-0 -- /bin/sh
$ vault login s.VgQvaXl8xGFO1RUxAPbPbsfN
```

Expected output

```
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key                  Value
---                  -----
token                s.VgQvaXl8xGFO1RUxAPbPbsfN
token_accessor       iMGp477aReYkPBWrR42Z3L6R
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]`
```

Now enable the key-value secret engine with the following command:

```
$ vault secrets enable -path secret kv-v2
```

Expected output

```
Success! Enabled the kv-v2 secrets engine at: secret/
```

Note

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

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

Expected output

```
Success! Enabled the file audit device at: file/
```

2. Now generate Secret with the Vault root token using `kubectl command` (don't forget to substitute the token from the example with your real root token) and add necessary options to `configuration` keys in your `deploy/cr.yaml`:

without TLS, to access the Vault server via HTTP

Generate Secret:

```
$ kubectl create secret generic vault-secret --from-
literal=token="s.VgQvaXl8xGFO1RUxAPbPbsfN"
```

Now modify your `deploy/cr.yaml`:

First set the `secrets.encryptionKey` key to the name of your Secret created on the previous step. Then Add Vault-specific options to the `replsets.configuration`, `replsets.nonvoting.configuration`, and `sharding.configsvrReplSet.configuration` keys, using the following template:

```
...
configuration: |
  ...
  security:
    enableEncryption: true
    vault:
      serverName: vault
      port: 8200
      tokenFile: /etc/mongodb-vault/token
      secret: secret/data/dc/<cluster name>/<path>
      disableTLSForTesting: true
    ...
```

with TLS, to access the Vault server via HTTPS

Generate Secret, using the path to your `ca.crt` certificate instead of the `<path to CA>` placeholder (see the Operator TLS guide, if needed):

```
kubectl create secret generic vault-secret --from-
literal=token="s.VgQvaXl8xGFO1RUxAPbPbsfN" --from-file=ca.crt=<path to CA>/ca.crt
```

Now modify your `deploy/cr.yaml`:

First set the `secrets.encryptionKey` key to the name of your Secret created on the previous step. Then Add Vault-specific options to the `replsets.configuration`, `replsets.nonvoting.configuration`, and `sharding.configsvrReplSet.configuration` keys, using the following template:

```
...
configuration: |
  ...
  security:
    enableEncryption: true
    vault:
      serverName: vault
      port: 8200
      tokenFile: /etc/mongodb-vault/token
      secret: secret/data/dc/<cluster name>/<path>
      serverCAFile: /etc/mongodb-vault/ca.crt
    ...
```

While adding options, modify this template as follows: * substitute the `<cluster name>` placeholder with your real cluster name, * substitute the placeholder with `rs0` when adding options to `replsets.configuration` and `replsets.nonvoting.configuration`, * substitute the placeholder with `cfg` when adding options to `sharding.configsvrReplSet.configuration`.

Finally, apply your modified `cr.yaml` as usual:

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

3. To verify that everything was configured properly, use the following log filtering command (substitute the `<cluster name>` and `<namespace>` placeholders with your real cluster name and namespace):

```
$ kubectl logs <cluster name>-rs0-0 -c mongod -n <namespace> | grep -i "Encryption keys
DB is initialized successfully"
```

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

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-12-20

12.11 Telemetry

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

The following information is gathered:

- ID of the Custom Resource (the `metadata.uid` field)
- Kubernetes version
- Platform (is it Kubernetes or Openshift)
- Is PMM enabled, and the PMM Version
- Operator version
- Mongo version
- Percona Backup for MongoDB (PBM) version
- Is sharding enabled (starting from the Operator version 1.13)
- Is Hashicorp Vault enabled (starting from the Operator version 1.13)
- Is the Operator deployed in a cluster-wide mode (starting from the Operator version 1.13)
- Is the Operator deployed with Helm
- Are sidecar containers used
- Are backups used, are point-in-time recovery and/or scheduled physical backup features used, if so
- How large is the cluster

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

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

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

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

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

```
$ helm install my-db percona/psmdb-db --version 1.15.0 --namespace my-namespace --set
disable_telemetry="true"
```

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

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

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

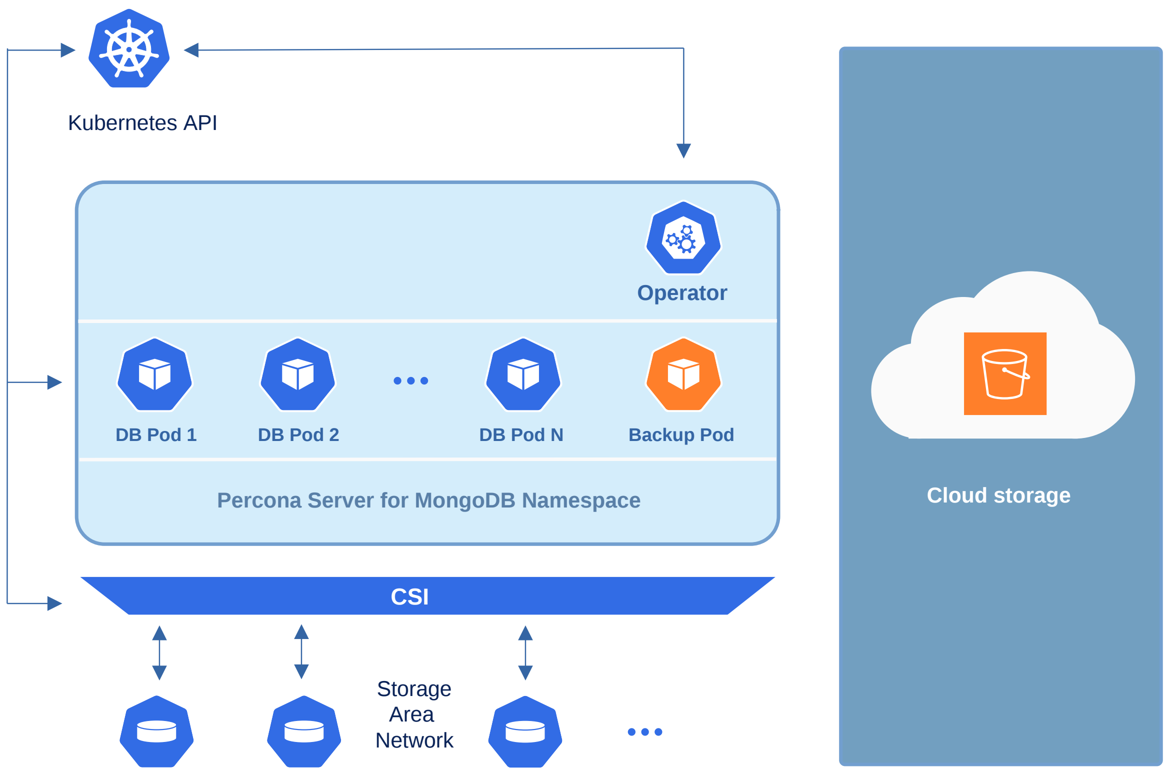
Last update: 2023-03-13

13. Management

13.1 Backup and restore

13.1.1 About backups

The Operator usually stores Server for MongoDB backups outside the Kubernetes cluster: on Amazon S3 or S3-compatible storage, or on Azure Blob Storage.



Backups are done by the Operator using the Percona Backup for MongoDB tool.

The 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.

The Operator can do either *logical* or *physical* backups.

- *Logical backup* means querying the Percona Server for MongoDB for the database data and writing the retrieved data to the remote backup storage.
- *Physical backup* means copying physical files from the Percona Server for MongoDB `dbPath` data directory to the remote backup storage.

Logical backups use less storage, but are much slower than physical backup/restore.

Also, logical backups are stable, while physical backups are available since the Operator version 1.14.0 and still have the **technical preview stauts**.

> **Warning**
>
> Logical backups made with the Operator versions before 1.9.0 are incompatible for restore with the Operator 1.9.0 and later. That is because Percona Backup for MongoDB 1.5.0 used by the newer Operator versions processes system collections Users and Roles differently. The recommended approach is to **make a fresh backup after upgrading the Operator to version 1.9.0**.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-03

13.1.2 Configure storage for backups

You can configure storage for backups in the `backup.storages` subsection of the Custom Resource, using the deploy/cr.yaml configuration file.

You should also create the Kubernetes Secret object with credentials needed to access the storage.

Amazon S3 or S3-compatible storage

1. To store backups on the Amazon S3, you need to create a Secret with the following values:
   - the `metadata.name` key is the name which you wll further use to refer your Kubernetes Secret,
   - the `data.AWS_ACCESS_KEY_ID` and `data.AWS_SECRET_ACCESS_KEY` keys are base64-encoded credentials used to access the storage (obviously these keys should contain proper values to make the access possible).

   Create the Secrets file with these base64-encoded keys following the deploy/backup-s3.yaml example:

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

   Note

   You can use the following command to get a base64-encoded string from a plain text one:

   in Linux

```
$ echo -n 'plain-text-string' | base64 --wrap=0
```

   in macOS

```
$ echo -n 'plain-text-string' | base64
```

   Once the editing is over, create the Kubernetes Secret object as follows:

```
$ kubectl apply -f deploy/backup-s3.yaml
```

2. Put the data needed to access the S3-compatible cloud into the `backup.storages` subsection of the Custom Resource.
   - `storages.<NAME>.type` should be set to `s3` (substitute the part with some arbitrary name you will later use to refer this storage when making backups and restores).
   - `storages.<NAME>.s3.credentialsSecret` key should be set to the name used to refer your Kubernetes Secret (`my-cluster-name-backup-s3` in the last example).
   - `storages.<NAME>.s3.bucket` and `storages.<NAME>.s3.region` should contain the S3 bucket and region. Also you can use `storages.<NAME>.s3.prefix` option to specify the path (sub-folder) to the backups inside the S3 bucket. If prefix is not set, backups are stored in the root directory.
   - if you use some S3-compatible storage instead of the original Amazon S3, add the endpointURL key in the `s3` subsection, which should point to the actual cloud used for backups. This value and is specific to the cloud provider. For example, using Google Cloud involves the following endpointUrl:

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

   The options within the `storages.<NAME>.s3` subsection are further explained in the Operator Custom Resource options.

   Here is an example of the deploy/cr.yaml configuration file which configures Amazon S3 storage for backups:

```
...
backup:
  ...
  storages:
    s3-us-west:
      type: s3
```

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-13

13.1.3 Making scheduled backups

Backups schedule is defined in the `backup` section of the Custom Resource and can be configured via the deploy/cr.yaml file.

1. The `backup.enabled` key should be set to `true`,
2. The `backup.storages` subsection should contain at least one configured storage.
3. The `backup.tasks` subsection allows to actually schedule backups:
    - set the `name` key to some arbitray backup name (this name will be needed later to restore the bakup).
    - specify the `schedule` option with the desired backup schedule in crontab format).
    - set the `enabled` key to `true` (this enables making the `<backup name>` backup along with the specified schedule.
    - set the `storageName` key to the name of your already configured storage.
    - you can optionally set the `keep` key to the number of backups which should be kept in the storage.
    - you can optionally set the `type` key to `physical` if you would like to make physical backups instead of logical ones (please see the physical backups limitations). Otherwise set this key to `logical`, or just omit it.

Here is an example of the `deploy/cr.yaml` with a scheduled Saturday night backup kept on the Amazon S3 storage:

```
...
backup:
  enabled: true
  storages:
    s3-us-west:
      type: s3
      s3:
        bucket: S3-BACKUP-BUCKET-NAME-HERE
        region: us-west-2
        credentialsSecret: my-cluster-name-backup-s3
  tasks:
   - name: "sat-night-backup"
     enabled: true
     schedule: "0 0 * * 6"
     keep: 3
     type: logical
     storageName: s3-us-west
  ...
```

Note

If you plan to restore backup to a new Kubernetes-based environment, make sure you will be able to create there a Secrets object with the same user passwords as in the original cluster. More details about secrets can be found in System Users. The name of the current Secrets object you will need to recreate can be found out from the `spec.secrets` key in the `deploy/cr.yaml` (`my-cluster-name-secrets` by default).

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

13.1.4 Making on-demand backup

1. To make an on-demand backup, you should first check your Custom Resource for the necessary options and make changes, if needed, using the `deploy/cr.yaml` configuration file:
   - the `backup.enabled` key should be set to `true`,
   - `backup.storages` subsection should contain at least one configured storage.

   You can apply changes in the `deploy/cr.yaml` file with the usual `kubectl apply -f deploy/cr.yaml` command.
2. Now use *a special backup configuration YAML file* with the following keys:
   - `metadata.name` key should be set to the **backup name** (this name will be needed later to restore the bakup),
   - `spec.clusterName` key should be set to the name of your cluster (prior to the Operator version 1.12.0 this key was named `spec.psmdbCluster`),
   - `spec.storageName` key should be set to the name of your already configured storage.
   - optionally you can set the `spec.type` key to `physical` if you would like to make physical backups instead of logical ones (please see the physical backups limitations). Otherwise set this key to `logical`, or just omit it.

   You can find the example of such file in deploy/backup/backup.yaml:

   ```
   apiVersion: psmdb.percona.com/v1
   kind: PerconaServerMongoDBBackup
   metadata:
     finalizers:
     - delete-backup
     name: backup1
   spec:
     clusterName: my-cluster-name
     storageName: s3-us-west
     type: logical
   ```
3. Run the actual backup command using this file:

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

Note

If you plan to restore backup to a new Kubernetes-based environment, make sure you will be able to create there a Secrets object with the same user passwords as in the original cluster. More details about secrets can be found in System Users. The name of the current Secrets object you will need to recreate can be found out from the `spec.secrets` key in the `deploy/cr.yaml` (`my-cluster-name-secrets` by default).

1. You can track the backup process with the `PerconaServerMongoDBBackup` Custom Resource as follows:

```
$ kubectl get psmdb-backup
```

Expected output

```
NAME      CLUSTER           STORAGE      DESTINATION            STATUS     COMPLETED   AGE
backup1   my-cluster-name   s3-us-west   2022-09-08T03:22:19Z   running                49s
```

It should show the status as `READY` when the backup process is over.

If you have any issues with the backup, you can view logs from the backup-agent container of the appropriate Pod as follows:

```
$ kubectl logs pod/my-cluster-name-rs0 -c backup-agent
```

Alternatively, getting ssh access to the same container will allow you to carry on Percona Backup for MongoDB diagnostics.

Note

In both cases you will need the name of the Pod that made the backup. You can find the `pbmPodName` field in the output of the `kubectl get psmdb-backup <backup_name> -o yaml` command.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

13.1.5 Storing operations logs for point-in-time recovery

Point-in-time recovery functionality allows users to roll back the cluster to a specific date and time. Technically, this feature involves saving operations log updates to the cloud storage.

Currently, point-in-time recovery functionality can be used with logical backups only.

To be used, it requires setting the backup.pitr.enabled key in the `deploy/cr.yaml` configuration file:

```yaml
backup:
  ...
  pitr:
    enabled: true
```

Note

It is necessary to have at least one full backup to use point-in-time recovery. Percona Backup for MongoDB will not upload operations logs if there is no full backup. This is true for new clusters and also true for clusters which have been just recovered from backup.

Percona Backup for MongoDB uploads operations logs to the same bucket/container, where full backup is stored. This makes point-in-time recovery functionality available only if there is a single bucket/container in spec.backup.storages. Otherwise point-in-time recovery will not be enabled and there will be an error message in the operator logs.

Note

Adding a new bucket or container when point-in-time recovery is enabled will not break it, but put error message about the additional bucket in the Operator logs as well.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-13

13.1.6 Enable server-side encryption for backups

Encrypting database backups is done separately for physical and logical backups. Physical backups are encrypted if data-at-rest encryption is turned on. Logical backups need to be encrypted on the cloud.

There is a possibility to enable server-side encryption for backups stored on S3. Starting from the version 1.15.0, the Operator supports Server Side Encryption either with AWS Key Management Service (KMS), or just encrypt/decrypt backups with AES-256 encryption algorithm with any S3-compatible storage.

To enable server-side encryption for backups, use backup.storages.backup.storages.<storage-name>.s3.serverSideEncryption section in the `deploy/cr.yaml` configuration file.

Encryption with keys stored in AWS KMS

To use the server-side AWS KMS encryption, specify the following Custom Resource options in the `deploy/cr.yaml` configuration file:

```
backup:
  ...
  storages:
    my-s3:
      type: s3
      s3:
        bucket: my-backup-bucket
        serverSideEncryption:
          kmsKeyID: <kms_key_ID>
          sseAlgorithm: aws:kms
```

Here `<kms_key_ID>` should be substituted with the ID of your customer-managed key stored in the AWS KMS. It should look similar to the following example value: `128887dd-d583-43f2-b3f9-d12036d32b12`.

Encryption with localy-stored keys on any S3-compatible storage

The Operator also supports server-side encryption with customer-provided keys that are stored on the client side. During the backup/restore process, encryption key will be provided by the Operator as part of the requests to the S3 storage, and the S3 storage will them to encrypt/decrypt the data using the AES-256 encryption algorithm. This allows to use server-side encryption on S3-compatible storages different from AWS KMS (the feature was tested with the AWS and MinIO storages).

To use the server-side encryption wit locally-stored keys, specify the following Custom Resource options in the `deploy/cr.yaml` configuration file:

```
backup:
  ...
  storages:
    my-s3:
      type: s3
      s3:
        bucket: my-backup-bucket
        serverSideEncryption:
          sseCustomerAlgorithm: AES256
          sseCustomerKey: <your_encryption_key_in_base64>
    ...
```

Here `<your_encryption_key_in_base64>` should be substituted with the actual encryption key encoded in base64.

Note

You can use the following command to get a base64-encoded string from a plain text one:

in Linux

```
$ echo -n 'plain-text-string' | base64 --wrap=0
```

in macOS

```
$ echo -n 'plain-text-string' | base64
```

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

13.1.7 Restore the cluster from a previously saved backup

The backup is normally restored on the Kubernetes cluster where it was made, but restoring it on a different Kubernetes-based environment with the installed Operator is also possible.

Following things are needed to restore a previously saved backup:

- Make sure that the cluster is running.
- Find out correct names for the **backup** and the **cluster**. Available backups can be listed with the following command:

  ```
  $ kubectl get psmdb-backup
  ```

  And the following command will list available clusters:

  ```
  $ kubectl get psmdb
  ```

Note

If you have configured storing operations logs for point-in-time recovery, you will have possibility to roll back the cluster to a specific date and time. Otherwise, restoring backups without point-in-time recovery is the only option.

When the correct names for the backup and the cluster are known, backup restoration can be done in the following way.

Without point-in-time recovery

1. Set appropriate keys in the deploy/backup/restore.yaml file.
   - set `spec.clusterName` key to the name of the target cluster to restore the backup on,
   - set `spec.backupName` key to the name of your backup,

   ```
   apiVersion: psmdb.percona.com/v1
   kind: PerconaServerMongoDBRestore
   metadata:
     name: restore1
   spec:
     clusterName: my-cluster-name
     backupName: backup1
   ```

2. After that, 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: psmdb.percona.com/v1
   kind: PerconaServerMongoDBRestore
   metadata:
     name: restore1
   spec:
     clusterName: my-cluster-name
     backupName: backup1
   EOF
   ```

With point-in-time recovery

1. Set appropriate keys in the deploy/backup/restore.yaml file.
   - set `spec.clusterName` key to the name of the target cluster to restore the backup on
   - set `spec.backupName` key to the name of your backup
   - put additional restoration parameters to the `pitr` section:
     - `type` key can be equal to one of the following options
       - `date` - roll back to specific date
       - `latest` - recover to the latest possible transaction
     - `date` key is used with `type=date` option and contains value in datetime format

   The resulting `restore.yaml` file may look as follows:

   ```
   apiVersion: psmdb.percona.com/v1
   kind: PerconaServerMongoDBRestore
   metadata:
     name: restore1
   spec:
     clusterName: my-cluster-name
     backupName: backup1
     pitr:
       type: date
       date: YYYY-MM-DD hh:mm:ss
   ```

2. Run the actual restoration process:


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

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

13.1.8 Delete the unneeded backup

The maximum amount of stored backups is controlled by the backup.tasks.keep option (only successful backups are counted). Older backups are automatically deleted, so that amount of stored backups do not exceed this number. Setting `keep=0` or removing this option from `deploy/cr.yaml` disables automatic deletion of backups.

Manual deleting of a previously saved backup requires not more than the backup name. This name can be taken from the list of available backups returned by the following command:

```
$ kubectl get psmdb-backup
```

When the name is known, backup can be deleted as follows:

```
$ kubectl delete psmdb-backup/<backup-name>
```

Note

Deleting a backup used as a base for point-in-time recovery (PITR) is possible only starting from the Operator version 1.15.0. Also, deleting such a backup will delete the stored operations log updates based on this backup.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-05

13.2 Update Database and Operator

Starting from the version 1.1.0 the Percona Operator for MongoDB allows upgrades to newer versions. The upgradable components of the cluster are the following ones:

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

The list of recommended upgrade scenarios includes two variants:

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

13.2.1 Upgrading the Operator and CRD

Note

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

Only the incremental update to a nearest version of the Operator is supported (for example, update from 1.5.0 to 1.6.0). To update to a newer version, which differs from the current version by more than one, make several incremental updates sequentially.

Note

Starting from version 1.14.0, the Operator configures replica set members using local fully-qualified domain names (FQDN). Before this version, it used exposed IP addresses in the replica set configuration in case of the exposed replica set. If you have your replica set exposed and upgrade to 1.14.0, the replica set configuration will change to use FQDN. If you don't want such reconfiguration to happen, set `clusterServiceDNSMode` Custom Resource option to `External` before the upgrade.

Warning

Starting from the Operator version 1.15.0 the `spec.mongod` section (deprecated since 1.12.0) is finally removed from the Custom Resource configuration. If you have encryption disabled using the deprecated `mongod.security.enableEncryption` option, you need to set encryption disabled via the custom configuration before upgrade:

```
spec:
  ...
  replsets:
    - name: rs0
      ...
      configuration: |
        security:
          enableEncryption: false
        ...
```

Manual upgrade

The upgrade includes the following steps.

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

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.15.0/deploy/crd.yaml
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.15.0/deploy/rbac.yaml
```

2. Now you should apply a patch to your deployment, supplying necessary image name with a newer version tag. You can find the proper image name for the current Operator release in the list of certified images. updating to the `1.15.0` version should look as follows:

```
$ kubectl patch deployment percona-server-mongodb-operator \
   -p'{"spec":{"template":{"spec":{"containers":[{"name":"percona-server-mongodb-operator","image":"percona/percona-server-mongodb-operator:1.15.0"}]}}}}'
```

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

```
$ kubectl rollout status deployments percona-server-mongodb-operator
```

Note

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

Upgrade via helm

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

1. In case if you installed the Operator with no customized parameters, the upgrade can be done as follows:

```
$ helm upgrade my-op percona/psmdb-operator --version 1.15.0
```

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

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

   Note

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

```
$ helm upgrade my-op percona/psmdb-operator --version 1.15.0 -f my-values.yaml
```

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

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.15.0/deploy/crd.yaml
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.15.0/deploy/rbac.yaml
```

Note

You can use `helm upgrade` to upgrade the Operator only. The Database (Percona Server for MongoDB) should be upgraded in the same way whether you used helm to install it or not.

13.2.2 Upgrading Percona Server for MongoDB

The following section presumes that you are upgrading your cluster within the *Smart Update strategy*, when the Operator controls how the objects are updated. Smart Update strategy is on when the `updateStrategy` key in the Custom Resource configuration file is set to `SmartUpdate` (this is the default value and the recommended way for upgrades).

Note

As an alternative, the `updateStrategy` key can be used to turn off *Smart Update strategy*. You can find out more on this in the appropriate section.

Manual upgrade

Manual update of Percona Server for MongoDB can be done as follows:

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

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

2. Now apply a patch to your Custom Resource, setting necessary Custom Resource version and image names with a newer version tag.

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

Patching Custom Resource is done with the `kubectl patch psmdb` command. Actual image names can be found in the list of certified images. For example, updating `my-cluster-name` cluster to the `1.15.0` version should look as follows:

```
$ kubectl patch psmdb my-cluster-name --type=merge --patch '{
   "spec": {
      "crVersion":"1.15.0",
      "image": "percona/percona-server-mongodb:4.4.24-23",
      "backup": { "image": "percona/percona-backup-mongodb:2.3.0" },
      "pmm": { "image": "percona/pmm-client:2.39.0" }
   }}'
```

> **Warning**
>
> The above command upgrades various components of the cluster including PMM Client. It is highly recommended to upgrade PMM Server **before** upgrading PMM Client. If it wasn't done and you would like to avoid PMM Client upgrade, remove it from the list of images, reducing the last of two patch commands as follows:
>
> ```
> $ kubectl patch psmdb my-cluster-name --type=merge --patch '{
>    "spec": {
>       "crVersion":"1.15.0",
>       "image": "percona/percona-server-mongodb:4.4.24-23",
>       "backup": { "image": "percona/percona-backup-mongodb:2.3.0" }
>    }}'
> ```

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

```
$ kubectl rollout status sts my-cluster-name-rs0
```

The update process is successfully finished when all Pods have been restarted (including the mongos and Config Server nodes, if Percona Server for MongoDB Sharding is on).

Automated upgrade

Smart Update strategy allows you to automate upgrades even more. In this case the Operator can either detect the availability of the new Percona Server for MongoDB version, or rely on the user's choice of the version. To check the availability of the new version, the Operator will query a special *Version Service* server at scheduled times to obtain fresh information about version numbers and valid image paths.

If the current version should be upgraded, the Operator updates the Custom Resource to reflect the new image paths and carries on sequential Pods deletion, allowing StatefulSet to redeploy the cluster Pods with the new image. You can configure Percona Server for MongoDB upgrade via the `deploy/cr.yaml` configuration file as follows:

1. Make sure that `spec.updateStrategy` option is set to `SmartUpdate`.
2. Change `spec.crVersion` option to match the version of the Custom Resource Definition upgrade you have done while upgrading the Operator:

```
...
spec:
  crVersion: 1.15.0
  ...
```

> **Note**
>
> If you don't update crVersion, minor version upgrade is the only one to occur. For example, the image `percona-server-mongodb:5.0.7-6` can be upgraded to `percona-server-mongodb:5.0.11-10`.

3. Set the `upgradeOptions.apply` option from `Disabled` to one of the following values:
   - `Recommended` - automatic upgrade will choose the most recent version of software flagged as Recommended (for clusters created from scratch, the Percona Server for MongoDB 6.0 version will be selected instead of the Percona Server for MongoDB 5.0 or 4.4 version regardless of the image path; for already existing clusters, the 6.0 vs. 5.0 or 4.4 branch choice will be preserved),
   - `6.0-recommended`, `5.0-recommended`, `4.4-recommended` - same as above, but preserves specific major MongoDB version for newly provisioned clusters (ex. 6.0 will not be automatically used instead of 5.0),
   - `Latest` - automatic upgrade will choose the most recent version of the software available (for clusters created from scratch, the Percona Server for MongoDB 6.0 version will be selected instead of the Percona Server for MongoDB 5.0 or 4.4 version regardless of the image path; for already existing clusters, the 6.0 vs. 5.0 or 4.4 branch choice will be preserved),
   - `6.0-latest`, `5.0-latest`, `4.4-latest` - same as above, but preserves specific major MongoDB version for newly provisioned clusters (ex. 6.0 will not be automatically used instead of 5.0),
   - *version number* - specify the desired version explicitly (version numbers are specified as 6.0.9-7, 5.0.20-17, etc.). Actual versions can be found in the list of certified images.
4. Make sure the `versionServiceEndpoint` key is set to a valid Version Server URL (otherwise Smart Updates will not occur).

   Percona's Version Service (default)

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

   Version Service inside your cluster

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

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

> **Note**
>
> Version Service is never checked if automatic updates are disabled in the `upgradeOptions.apply` option. If automatic updates are enabled, but the Version Service URL can not be reached, no updgrades will be performed.

5. Use the `upgradeOptions.schedule` option to specify the update check time in CRON format.

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

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

Note

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

6. Don't forget to apply your changes to the Custom Resource in the usual way:

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

Note

When automatic upgrades are disabled by the `apply` option, Smart Update functionality will continue working for changes triggered by other events, such as rotating a password, or changing resource values.

Major version automated upgrades

Normally automatic upgrade takes place within minor versions (for example, from `4.4.16-16` to `4.4.18-18`) of MongoDB. Major versions upgrade (for example moving from `5.0-recommended` to `6.0-recommended`) is more complicated task which might potentially affect how data is stored and how applications interacts with the database (in case of some API changes).

Such upgrade is supported by the Operator within one major version at a time: for example, to change Percona Server for MongoDB major version from 4.4 to 6.0, you should first upgrade it to 5.0, and later make a separate upgrade from 5.0 to 6.0. The same is true for major version downgrades.

Note

It is recommended to take a backup before upgrade, as well as to perform upgrade on staging environment.

Major version upgrade can be initiated using the upgradeOptions.apply key in the `deploy/cr.yaml` configuration file:

```
spec:
  upgradeOptions:
    apply: 5.0-recommended
```

Note

When making downgrades (e.g. changing version from 5.0 to 4.4), make sure to remove incompatible features that are persisted and/or update incompatible configuration settings. Compatibility issues between major MongoDB versions can be found in upstream documentation.

By default the Operator doesn't set FeatureCompatibilityVersion (FCV) to match the new version, thus making sure that backwards-incompatible features are not automatically enabled with the major version upgrade (which is recommended and safe behavior). You can turn this backward compatibility off at any moment (after the upgrade or even before it) by setting the upgradeOptions.setFCV flag in the `deploy/cr.yaml` configuration file to `true`.

Note

With setFeatureCompatibilityVersion set major version rollback is not currently supported by the Operator. Therefore it is recommended to stay without enabling this flag for some time after the major upgrade to ensure the likelihood of downgrade is minimal. Setting `setFCV` flag to `true` simultaneously with the `apply` flag should be done only if the whole procedure is tested on staging and you are 100% sure about it.

13.2.3 More on upgrade strategies

The recommended way to upgrade your cluster is to use the *Smart Update strategy*, when the Operator controls how the objects are updated. Smart Update strategy is on when the `updateStrategy` key in the Custom Resource configuration file is set to `SmartUpdate` (this is the default value and the recommended way for upgrades).

Alternatively, you can set this key to `RollingUpdate` or `OnDelete`, which means that you will have to follow the low-level Kubernetes way of database upgrades. But take into account, that `SmartUpdate` strategy is not just for simplifying upgrades. Being turned on, it allows to disable automatic upgrades, and still controls restarting Pods in a proper order for changes triggered by other events, such as updating a ConfigMap, rotating a password, or changing resource values. That's why `SmartUpdate` strategy is useful even when you have no plans to automate upgrades at all.

Last update: 2023-03-22

13.3 Scale Percona Server for MongoDB 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.

The size of the cluster is controlled by the `size` key in the Custom Resource options configuration.

Note

The Operator will not allow to scale Percona Server for MongoDB with the `kubectl scale statefulset <StatefulSet name>` command as it puts `size` configuration options out of sync.

You can change size separately for different components of your cluster by setting this option in the appropriate subsections:

- replsets.size allows to set the size of the MongoDB Replica Set,
- replsets.arbiter.size allows to set the number of Replica Set Arbiter instances,
- sharding.configsvrReplSet.size allows to set the number of Config Server instances,
- sharding.mongos.size allows to set the number of mongos instances.

For example, the following update in `deploy/cr.yaml` will set the size of the MongoDB Replica Set to `5` nodes:

```
....
replsets:
  ....
  size: 5
  ....
```

Don't forget to apply changes as usual, running the `kubectl apply -f deploy/cr.yaml` command.

CONTACT US

For free technical help, visit the Percona Community Forum.

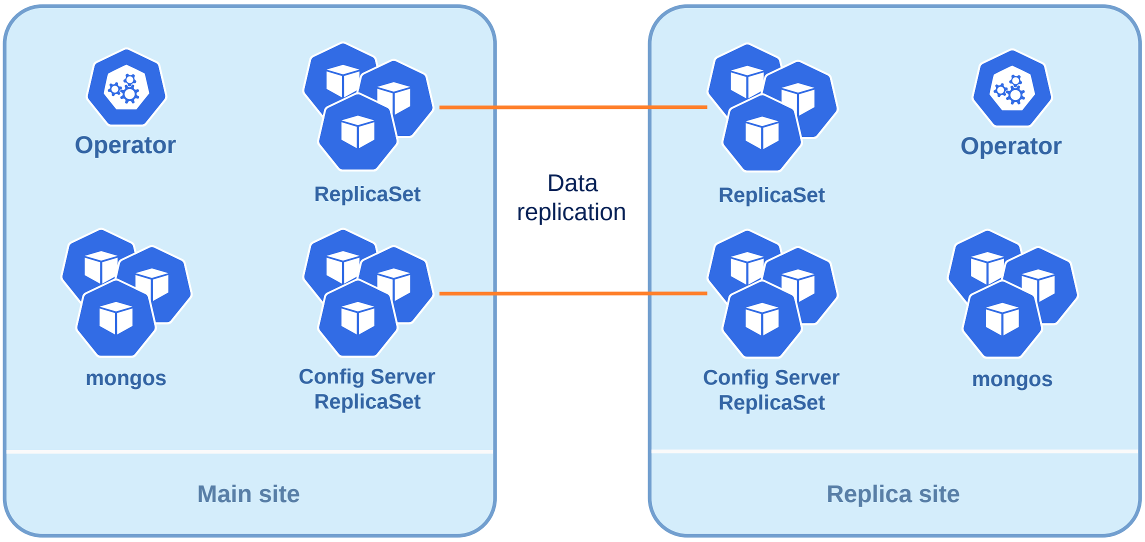
To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-18

13.4 Set up Percona Server for MongoDB cross-site replication

The cross-site replication involves configuring one MongoDB site as *Main*, and another MongoDB site as *Replica* to allow replication between them:



The Operator automates configuration of *Main* and *Replica* MongoDB sites, but the feature itself is not bound to Kubernetes. Either *Main* or *Replica* can run outside of Kubernetes, be regular MongoDB and be out of the Operators' control.

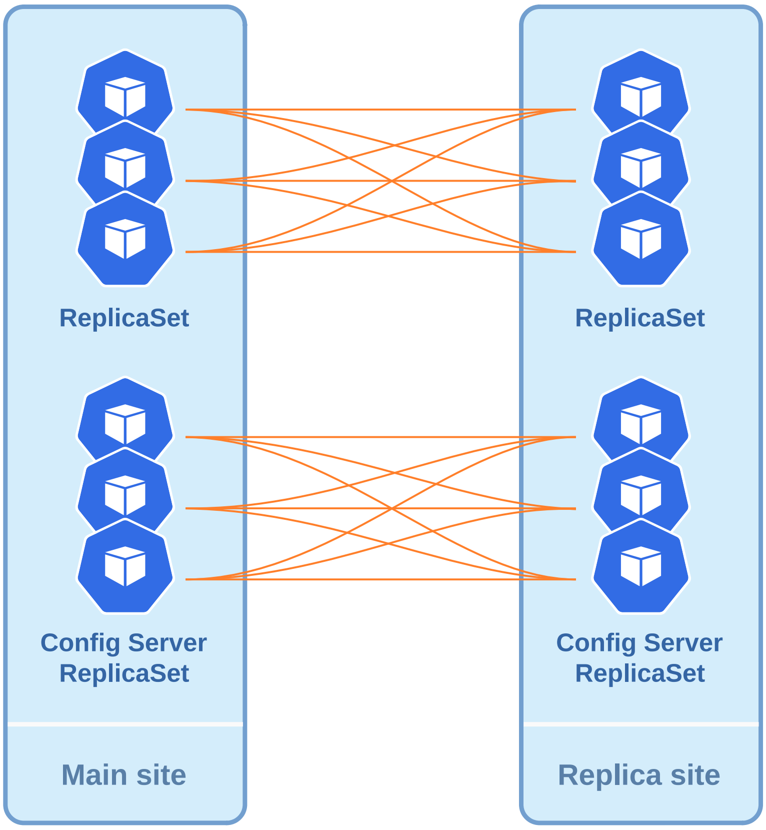
This feature can be useful in several cases:

- simplify the migration of the MongoDB cluster to and from Kubernetes
- add remote nodes to the replica set for disaster recovery

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

13.4.1 Exposing instances of the MongoDB cluster

You need to expose all Replica Set nodes (including Config Servers) through a dedicated service to ensure that *Main* and *Replica* can reach each other, like in a full mesh:



Note

Starting from v1.14, the operator configures the replset using local DNS hostnames even if the replset is exposed. If you want to have IP addresses in the replset configuration to achieve a multi-cluster deployment, you need to set `clusterServiceDNSMode` to `External`.

This is done through the `replsets.expose`, `sharding.configsvrReplSet.expose`, and `sharding.mongos.expose` sections in the `deploy/cr.yaml` configuration file as follows.

```
spec:
  replsets:
  - rs0:
    expose:
      enabled: true
      exposeType: LoadBalancer
    ...
  sharding:
    configsvrReplSet:
      expose:
        enabled: true
        exposeType: LoadBalancer
      ...
```

The above example is using the LoadBalancer Kubernetes Service object, but there are other options (ClusterIP, NodePort, etc.).

Note

The above example will create a LoadBalancer per each Replica Set Pod. In most cases, this Load Balancer should be internet-facing for cross-region replication to work.

To list the endpoints assigned to Pods, list the Kubernetes Service objects by executing `kubectl get services -l "app.kubernetes.io/instance=CLUSTER_NAME"` command.

13.4.2 Configuring cross-site replication on Main site

The cluster managed by the Operator should be able to reach external nodes of the Replica Sets. You can provide needed information in the `replsets.externalNodes` and `sharding.configsvrReplset.externalNodes` subsections of the `deploy/cr.yaml` configuration file. Following keys can be set to specify each external *Replica*, both for its Replica Set and Config Server instances:

- set `host` to URL or IP address of the external replset instance,
- set `port` to the port number of the external node (or rely on the `27017` default value),
- set `priority` to define the priority of the external node (`2` is default for all local members of the cluster; external nodes should have lower priority to avoid unmanaged node being elected as a primary; `0` adds the node as a non-voting member),
- set `votes` to the number of votes an external node can cast in a replica set election (`0` is default and should be used for non-voting members of the cluster).

Here is an example:

```yaml
spec:
  unmanaged: false
  replsets:
  - name: rs0
    externalNodes:
    - host: rs0-1.percona.com
      port: 27017
      priority: 0
      votes: 0
    - host: rs0-2.percona.com
    ...
  sharding:
    configsvrReplSet:
      size: 3
      externalNodes:
        - host: cfg-1.percona.com
          port: 27017
          priority: 0
          votes: 0
        - host: cfg-2.percona.com
        ...
```

The *Main* site will be ready for replication when you apply changes as usual:

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

Getting the cluster secrets and certificates to be copied from Main to Replica

Main and *Replica* should have same Secrets objects (to have same users credentials) and certificates. So you may need to copy them from *Main*. Names of the corresponding objects are set in the `users`, `ssl`, and `sslInternal` keys of the Custom Resource `secrets` subsection (`my-cluster-name-secrets`, `my-cluster-name-ssl`, and `my-cluster-name-ssl-internal` by default).

If you can get Secrets from an existing cluster by executing the `kubectl get secret` command for *each* Secrets object you want to acquire:

```
$ kubectl get secret my-cluster-name-secrets -o yaml > my-cluster-secrets.yaml
```

Next remove the `annotations`, `creationTimestamp`, `resourceVersion`, `selfLink`, and `uid` metadata fields from the resulting file to make it ready for the *Replica*.

You will need to further apply these secrets on Replica.

13.4.3 Configuring cross-site replication on Replica instances

When the Operator creates a new cluster, a lot of things are happening, such as electing the Primary, generating certificates, and picking specific names. This should not happen if we want the Operator to run the *Replica* site, so first of all the cluster should be put into unmanaged state by setting the `unmanaged` key in the `deploy/cr.yaml` configuration file to true. Also you should set `updateStrategy` key to `OnDelete` and `backup.enabled` to `false`, because Smart Updates and backups are not allowed on unmanaged clusters.

Note

Setting `unmanaged` to true will not only prevent the Operator from controlling the Replica Set configuration, but it will also result in not generating certificates and users credentials for new clusters.

Here is an example:

```yaml
spec:
  unmanaged: true
  updateStrategy: OnDelete
  replsets:
  - name: rs0
    size: 3
    ...
  backup:
    enabled: false
  ...
```

Main and *Replica* sites should have same Secrets objects, so don't forget to apply Secrets from your *Main* site. Names of the corresponding objects are set in the `users`, `ssl`, and `sslInternal` keys of the Custom Resource `secrets` subsection (`my-cluster-name-secrets`, `my-cluster-name-ssl`, and `my-cluster-name-ssl-internal` by default).

Copy your secrets from an existing cluster and apply each of them on your *Replica* site as follows:

```
$  kubectl apply -f my-cluster-secrets.yaml
```

The *Replica* site will be ready for replication when you apply changes as usual:

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

13.4.4 Enabling multi-cluster Services

Kubernetes multi-cluster Services (MCS) is a cross-cluster discovery and invocation of Services. MCS-enabled Services become discoverable and accessible across clusters with a virtual IP address.

This feature allows splitting applications into multiple clusters combined in one *fleet*, which can be useful to separate logically standalone parts (i.e. stateful and stateless ones), or to address privacy and scalability requirements, etc.

Multi-cluster Services should be supported by the cloud provider. It is supported by Google Kubernetes Engine (GKE), and by Amazon Elastic Kubernetes Service (EKS).

Configuring your cluster for multi-cluster Services includes two parts:

- configure MCS with your cloud provider,
- make needed preparations with the Operator.

To set up MCS for a specific cloud provider you should follow official guides, for example ones from Google Kubernetes Engine (GKE), or from Amazon Elastic Kubernetes Service (EKS).

> **Warning**
>
> For EKS, you also need to create ClusterProperty objects prior to enabling multi-cluster services.
>
> ```
> apiVersion: about.k8s.io/v1alpha1
>   kind: ClusterProperty
> metadata:
>   name: cluster.clusterset.k8s.io
> spec:
>   value: [Your Cluster identifier]
> ---
> apiVersion: about.k8s.io/v1alpha1
> kind: ClusterProperty
> metadata:
>   name: clusterset.k8s.io
> spec:
>   value: [Your ClusterSet identifier]
> ```
>
> Check AWS MCS controller repository for more information.

Setting up the Operator for MCS results in registering Services for export to other clusters using the ServiceExport object, and using ServiceImport one to import external services. Set the following options in the `multiCluster` subsection of the `deploy/cr.yaml` configuration file to make it happen:

- `multiCluster.enabled` should be set to `true`,
- `multiCluster.DNSSuffix` string should be equal to the cluster domain suffix for multi-cluster Services used by Kubernetes (`svc.clusterset.local` by default).

The following example in the `deploy/cr.yaml` configuration file is rather straightforward:

```
...
multiCluster:
  enabled: true
  DNSSuffix: svc.clusterset.local
...
```

Apply changes as usual with the `kubectl apply -f deploy/cr.yaml` command.

Note

If you want to enable multi-cluster services in a new cluster, we recommended deploying the cluster first with `multiCluster.enabled` set to `false` and enable it after replset is initialized. Having MCS enabled from the start is prone to errors on replset initialization.

The initial ServiceExport creation and sync with the clusters of the fleet takes approximately five minutes. You can check the list of services for export and import with the following commands:

```
$ kubectl get serviceexport
```

Expected output

```
NAME                        AGE
my-cluster-name-cfg         22m
my-cluster-name-cfg-0       22m
my-cluster-name-cfg-1       22m
my-cluster-name-cfg-2       22m
my-cluster-name-mongos      22m
my-cluster-name-rs0         22m
my-cluster-name-rs0-0       22m
my-cluster-name-rs0-1       22m
my-cluster-name-rs0-2       22m
```

```
$ kubectl get serviceimport
```

Expected output

```
NAME                        TYPE           IP                   AGE
my-cluster-name-cfg         Headless                            22m
my-cluster-name-cfg-0       ClusterSetIP   ["10.73.200.89"]     22m
my-cluster-name-cfg-1       ClusterSetIP   ["10.73.192.104"]    22m
my-cluster-name-cfg-2       ClusterSetIP   ["10.73.207.254"]    22m
my-cluster-name-mongos      ClusterSetIP   ["10.73.196.213"]    22m
my-cluster-name-rs0         Headless                            22m
my-cluster-name-rs0-0       ClusterSetIP   ["10.73.206.24"]     22m
my-cluster-name-rs0-1       ClusterSetIP   ["10.73.207.20"]     22m
my-cluster-name-rs0-2       ClusterSetIP   ["10.73.193.92"]     22m
```

Note

ServiceExport objects are created automatically by the Percona Server for MongoDB Operator. ServiceImport objects, on the other hand, are not controlled by the operator. If you need to troubleshoot ServiceImport objects you must check the MCS controller installed by your cloud provider.

After ServiceExport object is created, exported Services can be resolved from any Pod in any fleet cluster as `SERVICE_EXPORT_NAME.NAMESPACE.svc.clusterset.local`.

Note

This means that ServiceExports with the same name and namespace will be recognized as a single combined Service.

MCS can charge cross-site replication with additional limitations specific to the cloud provider. For example, GKE demands all participating Pods to be in the same project. Also, `default` Namespace should be used with caution: your cloud provider may not allow exporting Services from it to other clusters.

Applying MCS to an existing cluster

Additional actions are needed to turn on MCS for the **already-existing non-MCS cluster**.

- You need to restart the Operator after editing the `multiCluster` subsection keys and applying `deploy/cr.yaml`. Find the Operator's Pod name in the output of the `kubectl get pods` command (it will be something like `percona-server-mongodb-operator-d859b69b6-t44vk`) and delete it as follows:

```
$ kubectl delete percona-server-mongodb-operator-d859b69b6-t44vk
```

- If you are enabling MCS for a running cluster after upgrading from the Operator version `1.11.0` or below, you need rotating multi-domain (SAN) certificates. Do this by pausing the cluster and deleting TLS Secrets.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-03

13.5 Monitoring

Percona Monitoring and Management (PMM) provides an excellent solution of monitoring Percona Server for MongoDB.

> **Note**
>
> Only PMM 2.x versions are supported by the Operator.

PMM is a client/server application. *PMM Client* runs on each node with the database you wish to monitor: it collects needed metrics and sends gathered data to *PMM Server*. As a user, you connect to PMM Server to see database metrics on a number of dashboards.

That's why PMM Server and PMM Client need to be installed separately.

13.5.1 Installing PMM Server

PMM Server runs as a *Docker image*, a *virtual appliance*, or on an *AWS instance*. Please refer to the official PMM documentation for the installation instructions.

13.5.2 Installing PMM Client

The following steps are needed for the PMM client installation in your Kubernetes-based environment:

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

- set `pmm.enabled=true`
- set the `pmm.serverHost` key to your PMM Server hostname or IP address (it should be resolvable and reachable from within your cluster)
- authorize PMM Client within PMM Server in one of two ways:

with token-based authorization (recommended)

Acquire the API Key from your PMM Server and set `PMM_SERVER_API_KEY` in the deploy/secrets.yaml secrets file to this obtained API Key value. Keep in mind that you need an API Key with the "Admin" role. The API Key won't be rotated automatically.

with password-based authorization

check that the `PMM_SERVER_USER` key in the deploy/secrets.yaml secrets file contains your PMM Server user name (`admin` by default), and make sure the `PMM_SERVER_PASSWORD` key in the deploy/secrets.yaml secrets file contains the password specified for the PMM Server during its installation.

Password-based authorization method is deprecated since the Operator 1.13.0.

> **Note**
>
> You use `deploy/secrets.yaml` file to *create* Secrets Object. The file contains all values for each key/value pair in a convenient plain text format. But the resulting Secrets contain passwords stored as base64-encoded strings. If you want to *update* password field, you'll need to encode the value into base64 format. To do this, you can run `echo -n "password" | base64 --wrap=0` (or just `echo -n "password" | base64` in case of Apple macOS) in your local shell to get valid values. For example, setting the PMM Server API Key to `new_key` in the `my-cluster-name-secrets` object can be done with the following command:
>
> **in Linux**
>
> ```
> $ kubectl patch secret/my-cluster-name-secrets -p '{"data":{"PMM_SERVER_API_KEY": '$(echo -n new_key | base64 --wrap=0)'}}'
> ```
>
> **in macOS**
>
> ```
> $ kubectl patch secret/my-cluster-name-secrets -p '{"data":{"PMM_SERVER_API_KEY": '$(echo -n new_key | base64)'}}'
> ```

Apply changes with the `kubectl apply -f deploy/secrets.yaml` command.

- Starting from the Operator version 1.12.0, MongoDB operation profiling is disabled by default, and you should enable it to make PMM Query Analytics work. You can pass options to MongoDB in several ways, for example in the `configuration` subsection of the `deploy/cr.yaml`:

```yaml
spec:
  ...
  replsets:
    - name: rs0
      size: 3
      configuration: |
        operationProfiling:
          slowOpThresholdMs: 200
          mode: slowOp
          rateLimit: 100
```

- you can also use `pmm.mongodParams` and `pmm.mongosParams` keys to specify additional parameters for the pmm-admin add mongodb command for `mongod` and `mongos` Pods respectively, if needed.

> **Note**
>
> Please take into account that Operator automatically manages common MongoDB Service Monitoring parameters mentioned in the officiall `pmm-admin add mongodb` documentation, such like username, password, service-name, host, etc. Assigning values to these parameters is not recommended and can negatively affect the functionality of the PMM setup carried out by the Operator.

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

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

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

```
$ kubectl get pods
$ kubectl logs my-cluster-name-rs0-0 -c pmm-client
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-05-19

13.6 Using sidecar containers

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

Note

Custom sidecar containers can easily access other components of your cluster. Therefore they should be used carefully and by experienced users only.

13.6.1 Adding a sidecar container

You can add sidecar containers to Percona Distribution for MongoDB Replica Set, Config Servers, and mongos Pods. Just use `sidecars` subsection in the `replsets`, `sharding.configsvrReplSet`, and `sharding.mongos` of the `deploy/cr.yaml` configuration file. In this subsection, you should specify the name and image of your container and possibly a command to run:

```yaml
spec:
  replsets:
    ....
    sidecars:
    - image: busybox
      command: ["/bin/sh"]
      args: ["-c", "while true; do echo echo $(date -u) 'test' >> /dev/null; sleep 5; done"]
      name: rs-sidecar-0
    ....
```

Apply your modifications as usual:

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

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

```
$ kubectl describe pod my-cluster-name-rs0-0
```

Expected output

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

13.6.2 Getting shell access to a sidecar container

You can login to your sidecar container as follows:

```
$ kubectl exec -it my-cluster-name-rs0-0 -c rs-sidecar-0 -- sh
/ #
```

13.6.3 Mount volumes into sidecar containers

It is possible to mount volumes into sidecar containers.

Following subsections describe different volume types, which were tested with sidecar containers and are known to work.

Persistent Volume

You can use Persistent volumes when you need dynamically provisioned storage which doesn't depend on the Pod lifecycle. To use such volume, you should *claim* durable storage with persistentVolumeClaim without specifying any non-important details.

The following example requests 1G storage with `sidecar-volume-claim` PersistentVolumeClaim, and mounts the correspondent Persistent Volume to the `rs-sidecar-0` container's filesystem under the `/volume0` directory:

```
...
sidecars:
- image: busybox
  command: ["/bin/sh"]
  args: ["-c", "while true; do echo echo $(date -u) 'test' >> /dev/null; sleep 5; done"]
  name: rs-sidecar-0
  volumeMounts:
  - mountPath: /volume0
```

```yaml
    name: sidecar-volume-claim
sidecarPVCs:
- apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    name: sidecar-volume-claim
  spec:
    resources:
      requests:
        storage: 1Gi
    volumeMode: Filesystem
    accessModes:
      - ReadWriteOnce
```

Note

Sidecar containers for *mongos* Pods have limited Persistent volumes support: `sharding.mongos.sidecarPVCs` option can be used if there is a single mongos in deployment or when `ReadWriteMany/ReadOnlyMany` access modes are used (but these modes are available not in every storage).

Secret

You can use a secret volume to pass the information which needs additional protection (e.g. passwords), to the container. Secrets are stored with the Kubernetes API and mounted to the container as RAM-stored files.

You can mount a secret volume as follows:

```yaml
...
sidecars:
- image: busybox
  command: ["/bin/sh"]
  args: ["-c", "while true; do echo echo $(date -u) 'test' >> /dev/null; sleep 5; done"]
  name: rs-sidecar-0
  volumeMounts:
  - mountPath: /secret
    name: sidecar-secret
sidecarVolumes:
- name: sidecar-secret
  secret:
    secretName: mysecret
```

The above example creates a `sidecar-secret` volume (based on already existing `mysecret` Secret object) and mounts it to the `rs-sidecar-0` container's filesystem under the `/secret` directory.

Note

Don't forget you need to create a Secret Object before you can use it.

configMap

You can use a configMap volume to pass some configuration data to the container. Secrets are stored with the Kubernetes API and mounted to the container as RAM-stored files.

You can mount a configMap volume as follows:

```yaml
...
sidecars:
- image: busybox
  command: ["/bin/sh"]
  args: ["-c", "while true; do echo echo $(date -u) 'test' >> /dev/null; sleep 5; done"]
  name: rs-sidecar-0
  volumeMounts:
  - mountPath: /config
    name: sidecar-config
sidecarVolumes:
- name: sidecar-config
  configMap:
    name: myconfigmap
```

The above example creates a `sidecar-config` volume (based on already existing `myconfigmap` configMap object) and mounts it to the `rs-sidecar-0` container's filesystem under the `/config` directory.

Note

Don't forget you need to create a configMap Object before you can use it.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-12-20

13.7 Pause/resume Percona Server for MongoDB

There may be external situations when it is needed to shutdown the 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`.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-13

14. Troubleshooting

14.1 Initial troubleshooting

Percona Operator for MongoDB uses Custom Resources to manage options for the various components of the cluster.

- `PerconaServerMongoDB` Custom Resource with Percona Server for MongoDB options (it has handy `psmdb` shortname also),
- `PerconaServerMongoDBBackup` and `PerconaServerMongoDBRestore` Custom Resources contain options for Percona Backup for MongoDB used to backup Percona Server for MongoDB and to restore it from backups (`psmdb-backup` and `psmdb-restore` shortnames are available for them).

The first thing you can check for the Custom Resource is to query it with `kubectl get` command:

```
$ kubectl get psmdb
```

Expected output

```
NAME              ENDPOINT                                        STATUS   AGE
my-cluster-name   my-cluster-name-mongos.default.svc.cluster.local   ready    5m26s
```

The Custom Resource should have `Ready` status.

Note

You can check which Percona's Custom Resources are present and get some information about them as follows:

```
$ kubectl api-resources | grep -i percona
```

Expected output

```
perconaservermongodbbackups       psmdb-backup    psmdb.percona.com/v1
true          PerconaServerMongoDBBackup
perconaservermongodbrestores      psmdb-restore   psmdb.percona.com/v1
true          PerconaServerMongoDBRestore
perconaservermongodbs             psmdb           psmdb.percona.com/v1
true          PerconaServerMongoDB
```

14.1.1 Check the Pods

If Custom Resource is not getting `Ready` status, it makes sense to check individual Pods. You can do it as follows:

```
$ kubectl get pods
```

Expected output

```
NAME                                                READY   STATUS    RESTARTS   AGE
my-cluster-name-cfg-0                               2/2     Running   0          11m
my-cluster-name-cfg-1                               2/2     Running   1          10m
my-cluster-name-cfg-2                               2/2     Running   1          9m
my-cluster-name-mongos-0                            1/1     Running   0          11m
my-cluster-name-mongos-1                            1/1     Running   0          11m
my-cluster-name-mongos-2                            1/1     Running   0          11m
my-cluster-name-rs0-0                               2/2     Running   0          11m
my-cluster-name-rs0-1                               2/2     Running   0          10m
my-cluster-name-rs0-2                               2/2     Running   0          9m
percona-server-mongodb-operator-665cd69f9b-xg5dl    1/1     Running   0          37m
```

The above command provides the following insights:

- `READY` indicates how many containers in the Pod are ready to serve the traffic. In the above example, `my-cluster-name-rs0-0` Pod has all two containers ready (2/2). For an application to work properly, all containers of the Pod should be ready.
- `STATUS` indicates the current status of the Pod. The Pod should be in a `Running` state to confirm that the application is working as expected. You can find out other possible states in the official Kubernetes documentation.
- `RESTARTS` indicates how many times containers of Pod were restarted. This is impacted by the Container Restart Policy. In an ideal world, the restart count would be zero, meaning no issues from the beginning. If the restart count exceeds zero, it may be reasonable to check why it happens.
- `AGE`: Indicates how long the Pod is running. Any abnormality in this value needs to be checked.

You can find more details about a specific Pod using the `kubectl describe pods <pod-name>` command.

```
$ kubectl describe pods my-cluster-name-rs0-0
```

Expected output

```
...
Name:          my-cluster-name-rs0-0
Namespace:     default
...
Controlled By:  StatefulSet/my-cluster-name-rs0
Init Containers:
 mongo-init:
...
Containers:
 mongod:
...
   Restart Count:  0
   Limits:
     cpu:      300m
     memory:  500M
   Requests:
     cpu:       300m
     memory:   500M
   Liveness:   exec [/opt/percona/mongodb-healthcheck k8s liveness --ssl --sslInsecure --
sslCAFile /etc/mongodb-ssl/ca.crt --sslPEMKeyFile /tmp/tls.pem --startupDelaySeconds 7200]
delay=60s timeout=10s period=30s #success=1 #failure=4
   Readiness:  tcp-socket :27017 delay=10s timeout=2s period=3s #success=1 #failure=8
   Environment Variables from:
     internal-my-cluster-name-users  Secret  Optional: false
   Environment:
...
   Mounts:
...
Volumes:
...
Events:                      <none>
```

This gives a lot of information about containers, resources, container status and also events. So, describe output should be checked to see any abnormalities.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-09-25

14.2 Exec into the containers

If you want to examine the contents of a container "in place" using remote access to it, you can use the `kubectl exec` command. It allows you to run any command or just open an interactive shell session in the container. Of course, you can have shell access to the container only if container supports it and has a "Running" state.

In the following examples we will access the container `mongod` of the `my-cluster-name-rs0-0` Pod.

- Run `date` command:

```
$ kubectl exec -ti my-cluster-name-rs0-0 -c mongod -- date
```

Expected output

```
Thu Nov 24 10:01:17 UTC 2022
```

You will see an error if the command is not present in a container. For example, trying to run the `time` command, which is not present in the container, by executing `kubectl exec -ti my-cluster-name-rs0-0 -c mongod -- time` would show the following result:

```
OCI runtime exec failed: exec failed: unable to start container process: exec: "time":
executable file not found in $PATH: unknown command terminated with exit code 126
```

- Print `/var/log/mongo/mongod.log` file to a terminal:

```
$ kubectl exec -ti my-cluster-name-rs0-0 -c mongod -- cat /var/log/mongo/mongod.log
```

- Similarly, opening an Interactive terminal, executing a pair of commands in the container, and exiting it may look as follows:

```
$ kubectl exec -ti my-cluster-name-rs0-0 -c mongod -- bash
[mongodb@my-cluster-name-rs0-0 db]$ cat /etc/hostname
my-cluster-name-rs0-0
[mongodb@my-cluster-name-rs0-0 db]$ ls /var/log/mongo/mongod.log
/var/log/mongo/mongod.log
[mongodb@my-cluster-name-rs0-0 db]$ exit
exit
$
```

14.2.1 Avoid the restart-on-fail loop for Percona Server for MongoDB containers

The restart-on-fail loop takes place when the container entry point fails (e.g. `mongod` crashes). In such a situation, Pod is continuously restarting. Continuous restarts prevent to get console access to the container, and so a special approach is needed to make fixes.

You can prevent such infinite boot loop by putting the Percona Server for MongoDB containers into the "infinite sleep" *without* starting mongod. This behavior of the container entry point is triggered by the presence of the `/data/db/sleep-forever` file. The feature is available for both replica set and confg server Pods.

For example, you can do it for the `mongod` container of an appropriate Percona Server for MongoDB Pod as follows:

```
$ kubectl exec -it my-cluster-name-cfg-0 -c mongod -- sh -c 'touch /data/db/sleep-forever'
```

If `mongod` container can't start, you can use `backup-agent` container instead:

```
$ kubectl exec -it my-cluster-name-cfg-0 -c backup-agent -- sh -c 'touch /data/db/sleep-forever'
```

The instance will restart automatically and run in its usual way as soon as you remove this file (you can do it with a command similar to the one you have used to create the file, just substitute `touch` to `rm` in it).

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

14.3 Check the Logs

Logs provide valuable information. It makes sense to check the logs of the database Pods and the Operator Pod. Following flags are helpful for checking the logs with the `kubectl logs` command:

Flag	Description
`--container=<container-name>`	Print log of a specific container in case of multiple containers in a Pod
`--follow`	Follows the logs for a live output
`--since=<time>`	Print logs newer than the specified time, for example: `--since="10s"`
`--timestamps`	Print timestamp in the logs (timezone is taken from the container)
`--previous`	Print previous instantiation of a container. This is extremely useful in case of container restart, where there is a need to check the logs on why the container restarted. Logs of previous instantiation might not be available in all the cases.

In the following examples we will access containers of the `my-cluster-name-rs0-0` Pod.

- Check logs of the `mongod` container:

```
$ kubectl logs my-cluster-name-rs0-0 -c mongod
```

- Check logs of the `pmm-client` container:

```
$ kubectl logs my-cluster-name-rs0-0 -c pmm-client
```

- Filter logs of the `mongod` container which are not older than 600 seconds:

```
$ kubectl logs my-cluster-name-rs0-0 -c mongod --since=600s
```

- Check logs of a previous instantiation of the `mongod` container, if any:

```
$ kubectl logs my-cluster-name-rs0-0 -c mongod --previous
```

- Check logs of the `mongod` container, parsing the output with jq JSON processor:

```
$ kubectl logs my-cluster-name-rs0-0 -c mongod -f | jq -R 'fromjson?'
```

14.3.1 Changing logs representation

You can also change the representation of logs: either use structured representation, which produces a parcing-friendly JSON, or use traditional console-frienldy logging with specific level. Changing representation of logs is possible by editing the `deploy/operator.yml` file, which sets the following environment variables with self-speaking names and values:

```
env:
    ...
    name: LOG_STRUCTURED
    value: 'false'
    name: LOG_LEVEL
```

```
    value: INFO
    ...
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-09-25

14.4 Special debug images

For the cases when Pods are failing for some reason or just show abnormal behavior, the Operator can be used with a special *debug image* of the Percona Server for MongoDB, which has the following specifics:

- it avoids restarting on fail,
- it contains additional tools useful for debugging (sudo, telnet, gdb, mongodb-debuginfo package, etc.),
- extra verbosity is added to the mongodb daemon.

Particularly, using this image is useful if the container entry point fails (`mongod` 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 `image` key in the `deploy/cr.yaml` configuration file:

`percona/percona-server-mongodb:6.0.9-7-debug`

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.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-09-25

15. HOWTOs

15.1 How to integrate Percona Operator for MongoDB with OpenLDAP

LDAP services provided by software like OpenLDAP, Microsoft Active Directory, etc. are widely used by enterprises to control information about users, systems, networks, services and applications and the corresponding access rights for the authentication/authorization process in a centralized way.

The following guide covers a simple integration of the already-installed OpenLDAP server with Percona Distribution for MongoDB and the Operator. You can know more about LDAP concepts and LDIF files used to configure it, and find how to install and configure OpenLDAP in the official OpenLDAP and Percona Server for MongoDB documentation.

15.1.1 The OpenLDAP side

You can add needed OpenLDAP settings will the following LDIF portions:

```
0-percona-ous.ldif: |-
  dn: ou=perconadba,dc=ldap,dc=local
  objectClass: organizationalUnit
  ou: perconadba
1-percona-users.ldif: |-
  dn: uid=percona,ou=perconadba,dc=ldap,dc=local
  objectClass: top
  objectClass: account
  objectClass: posixAccount
  objectClass: shadowAccount
  cn: percona
  uid: percona
  uidNumber: 1100
  gidNumber: 100
  homeDirectory: /home/percona
  loginShell: /bin/bash
  gecos: percona
  userPassword: {crypt}x
  shadowLastChange: -1
  shadowMax: -1
  shadowWarning: -1
2-group-cn.ldif: |-
  dn: cn=admin,ou=perconadba,dc=ldap,dc=local
  cn: admin
  objectClass: groupOfUniqueNames
  objectClass: top
  ou: perconadba
  uniqueMember: uid=percona,ou=perconadba,dc=ldap,dc=local
```

Also a read-only user should be created for the database-issued user lookups. If everything is done correctly, the following command should work, resetting he percona user password:

```
$ ldappasswd -s percona -D "cn=admin,dc=ldap,dc=local" -w password -x
"uid=percona,ou=perconadba,dc=ldap,dc=local"
```

Note

If you are not sure about the approach to make references between user and group objects, OpenDAP overlays provide one of the possible ways to go.

15.1.2 The MongoDB and Operator side

The following steps will look different depending on whether sharding is on (the default behavior) or off.

if sharding is off

In order to get MongoDB connected with OpenLDAP in case of a a non-sharded (ReplicaSet) MongoDB cluster we need to configure two things:

- Mongod
- Internal mongodb role

Create configuration Secrets for mongod (in `my_mongod.conf` file):

```
security:
  authorization: "enabled"
  ldap:
    authz:
      queryTemplate: '{USER}?memberOf?base'
    servers: "openldap"
    transportSecurity: none
    bind:
      queryUser: "cn=readonly,dc=ldap,dc=local"
      queryPassword: "password"
    userToDNMapping:
      '[
          {
            match : "(.+)",
            ldapQuery: "OU=perconadba,DC=ldap,DC=local??sub?(uid={0})"
          }
   ]'
setParameter:
  authenticationMechanisms: 'PLAIN,SCRAM-SHA-1'
```

Note

This fragment provides mongod with LDAP-specific parameters, such as FQDN of the LDAP server (`server`), explicit lookup user, domain rules, etc.

Put the snippet on you local machine and create a Kubernetes Secret object named based on your MongoDB cluster name:

```
$ kubectl create secret generic <your_cluster_name>-rs0-mongod --from-
file=mongod.conf=my_mongod.conf
```

Next step is to start the MongoDB cluster up as it's described in Install Percona server for MongoDB on Kubernetes. On successful completion of the steps from this doc, we are to proceed with setting the roles for the 'external' (managed by LDAP) user inside the MongoDB. For this, log into MongoDB as administrator:

```
$ mongo "mongodb+srv://userAdmin:<userAdmin_password>@<your_cluster_name>-
rs0.<your_namespace>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
```

Note

LDAP over TLS is not yet supproted by the Operator.

When logged in, execute the following:

```
mongos> db.getSiblingDB("admin").createRole(
{
 role: "cn=admin,ou=perconadba,dc=ldap,dc=local",
 privileges: [],
 roles : [
   {
     "role" : "readAnyDatabase",
     "db" : "admin"
```

When logged in, execute the following:

```
mongos> db.runCommand({connectionStatus:1})
```

The output should be like follows:

```
{
 "authInfo" : {
   "authenticatedUsers" : [
     {
       "user" : "percona",
       "db" : "$external"
     }
   ],
   "authenticatedUserRoles" : [
     {
       "role" : "restore",
       "db" : "admin"
     },
     {
       "role" : "readAnyDatabase",
       "db" : "admin"
     },
     {
       "role" : "clusterMonitor",
       "db" : "admin"
     },
     {
       "role" : "dbAdminAnyDatabase",
       "db" : "admin"
     },
     {
       "role" : "backup",
       "db" : "admin"
     },
     {
       "role" : "cn=admin,ou=perconadba,dc=ldap,dc=local",
       "db" : "admin"
     },
     {
       "role" : "readWriteAnyDatabase",
       "db" : "admin"
     }
   ]
 },
 "ok" : 1,
 "$clusterTime" : {
   "clusterTime" : Timestamp(1663067287, 4),
   "signature" : {
     "hash" : BinData(0,"ZaLGSVj4ZwZrngXZSOqXB5rx+oo="),
     "keyId" : NumberLong("7142816031004688408")
   }
 },
 "operationTime" : Timestamp(1663067287, 4)
}
mongos>
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-13

15.2 Use Docker images from a custom registry

Using images from a private Docker registry may required for privacy, security or other reasons. In these cases, Percona Operator for MongoDB allows the use of a custom registry This following example of the Operator deployed in the OpenShift environment demonstrates the process:

1. Log into the OpenShift and create a project.

```
$ oc login
```

Expected output

```
Authentication required for https://192.168.1.100:8443 (openshift)
Username: admin
Password:
Login successful.
```

```
$ oc new-project psmdb
```

Expected output

```
Now using project "psmdb" on server "https://192.168.1.100:8443".
```

2. You need obtain the following objects to configure your custom registry access:
   - A user token
   - the registry IP address

   You can view the token with the following command:

```
$ oc whoami -t
```

Expected output

```
ADO8CqCDappWR4hxjfDqwijEHei31yXAvWg61Jg210s
```

The following command returns the registry IP address:

```
$ kubectl get services/docker-registry -n default
```

Expected output

```
NAME              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
docker-registry   ClusterIP   172.30.162.173   <none>        5000/TCP   1d
```

3. Use the user token and the registry IP address to login to the registry:

```
$ docker login -u admin -p ADO8CqCDappWR4hxjfDqwijEHei31yXAvWg61Jg210s 172.30.162.173:5000
```

Expected output

```
Login Succeeded
```

4. Use the Docker commands to pull the needed image by its SHA digest:

```
$ docker pull docker.io/perconalab/percona-server-
mongodb@sha256:991d6049059e5eb1a74981290d829a5fb4ab0554993748fde1e67b2f46f26bf0
```

Expected output

```
Trying to pull repository docker.io/perconalab/percona-server-mongodb ...
sha256:991d6049059e5eb1a74981290d829a5fb4ab0554993748fde1e67b2f46f26bf0: Pulling from
docker.io/perconalab/percona-server-mongodb
Digest: sha256:991d6049059e5eb1a74981290d829a5fb4ab0554993748fde1e67b2f46f26bf0
Status: Image is up to date for docker.io/perconalab/percona-server-
mongodb@sha256:991d6049059e5eb1a74981290d829a5fb4ab0554993748fde1e67b2f46f26bf0
```

You can find correct names and SHA digests in the current list of the Operator-related images officially certified by Percona.

5. The following method can push an image to the custom registry for the example OpenShift `psmdb` project:

```
$ docker tag \
    docker.io/perconalab/percona-server-
mongodb@sha256:991d6049059e5eb1a74981290d829a5fb4ab0554993748fde1e67b2f46f26bf0 \
    172.30.162.173:5000/psmdb/percona-server-mongodb:4.4.24-23
$ docker push 172.30.162.173:5000/psmdb/percona-server-mongodb:4.4.24-23
```

6. Verify the image is available in the OpenShift registry with the following command:

```
$ oc get is
```

Expected output

```
NAME                              DOCKER
REPO                                                             TAGS              UPDATED
percona-server-mongodb            docker-registry.default.svc:5000/psmdb/percona-server-
mongodb   4.4.24-23   2 hours ago
```

7. When the custom registry image is available, edit the the `image:` option in `deploy/operator.yaml` configuration file with a Docker Repo + Tag string (it should look like `docker-registry.default.svc:5000/psmdb/percona-server-mongodb:4.4.24-23`)

Note

If the registry requires authentication, you can specify the `imagePullSecrets` option for all images.

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

Note

Don't forget to set upgradeoptions.apply option to `Disabled`. Otherwise Smart Upgrade functionality will try using the image recommended by the Version Service instead of the custom one.

9. Now follow the standard Percona Operator for MongoDB installation instruction.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-12-20

15.3 Creating a private S3-compatible cloud for backups

As it is mentioned in backups, any cloud storage which implements the S3 API can be used for backups. The one way to setup and implement the S3 API storage on Kubernetes or OpenShift is Minio - the S3-compatible object storage server deployed via Docker on your own infrastructure.

Setting up Minio to be used with Percona Operator for MongoDB backups involves the following steps:

1. Install Minio in your Kubernetes or OpenShift environment and create the correspondent Kubernetes Service as follows:

```
$ helm install \
  --name minio-service \
  --version 8.0.5 \
  --set accessKey=some-access-key \
  --set secretKey=some-secret-key \
  --set service.type=ClusterIP \
  --set configPath=/tmp/.minio/ \
  --set persistence.size=2G \
  --set environment.MINIO_REGION=us-east-1 \
  stable/minio
```

   Don't forget to substitute default `some-access-key` and `some-secret-key` strings in this command with actual unique key values. The values can be used later for access control. The `storageClass` option is needed if you are using the special Kubernetes Storage Class for backups. Otherwise, this setting may be omitted. You may also notice the `MINIO_REGION` value which is may not be used within a private cloud. Use the same region value here and on later steps (`us-east-1` is a good default choice).

2. Create an S3 bucket for backups:

```
$ kubectl run -i --rm aws-cli --image=perconalab/awscli --restart=Never -- \
  bash -c 'AWS_ACCESS_KEY_ID=some-access-key \
  AWS_SECRET_ACCESS_KEY=some-secret-key \
  AWS_DEFAULT_REGION=us-east-1 \
  /usr/bin/aws \
  --endpoint-url http://minio-service:9000 \
  s3 mb s3://operator-testing'
```

   This command creates the bucket named `operator-testing` with the selected access and secret keys (substitute `some-access-key` and `some-secret-key` with the values used on the previous step).

3. Now edit the backup section of the deploy/cr.yaml file to set proper values for the `bucket` (the S3 bucket for backups created on the previous step), `region`, `credentialsSecret` and the `endpointUrl` (which should point to the previously created Minio Service).

```
...
backup:
  enabled: true
  version: 0.3.0
  ...
  storages:
    minio:
      type: s3
      s3:
        bucket: operator-testing
        region: us-east-1
        credentialsSecret: my-cluster-name-backup-minio
        endpointUrl: http://minio-service:9000
  ...
```

   The option which should be specially mentioned is `credentialsSecret` which is a Kubernetes secret for backups. Sample backup-s3.yaml can be used to create this secret object. Check that the object contains the proper `name` value and is equal to the one specified for `credentialsSecret`, i.e. `my-cluster-name-backup-minio` in the backup to Minio example, and also contains the proper `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` keys. After you have finished editing the file, the secrets object are created or updated when you run the following command:

```
$ kubectl apply -f deploy/backup-s3.yaml
```

4. When the setup process is completed, making the backup is based on a script. Following example illustrates how to make an on-demand backup:

```
$ kubectl run -it --rm pbmctl --image=percona/percona-server-mongodb-operator:0.3.0-
backup-pbmctl --restart=Never -- \
   run backup \
   --server-address=<cluster-name>-backup-coordinator:10001 \
   --storage <storage> \
   --compression-algorithm=gzip \
   --description=my-backup
```

Don't forget to specify the name of your cluster instead of the `<cluster-name>` part of the Backup Coordinator URL (the cluster name is specified in the deploy/cr.yaml file). Also substitute `<storage>` with the actual storage name located in a subsection inside of the `backups` in the deploy/cr.yaml file. In the earlier example this value is `minio`.

5. To restore a previously saved backup you must specify the backup name. With the proper Backup Coordinator URL and storage name, you can obtain a list of the available backups:

```
$ kubectl run -it --rm pbmctl --image=percona/percona-server-mongodb-operator:0.3.0-
backup-pbmctl --restart=Never -- list backups --server-address=<cluster-name>-backup-
coordinator:10001
```

Now, restore the backup, using backup name instead of the `backup-name` parameter:

```
$ kubectl run -it --rm pbmctl --image=percona/percona-server-mongodb-operator:0.3.0-
backup-pbmctl --restart=Never -- \
   run restore \
   --server-address=<cluster-name>-backup-coordinator:10001 \
   --storage <storage> \
   backup-name
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-31

15.4 How to restore backup to a new Kubernetes-based environment

The Operator allows restoring a backup not only on the Kubernetes cluster where it was made, but also on any Kubernetes-based environment with the installed Operator.

When restoring to a new Kubernetes-based environment, make sure it has a Secrets object with the same user passwords as in the original cluster. More details about secrets can be found in System Users. The name of the required Secrets object can be found out from the `spec.secrets` key in the `deploy/cr.yaml` (`my-cluster-name-secrets` by default).

You will need correct names for the **backup** and the **cluster**. If you have access to the original cluster, available backups can be listed with the following command:

```
$ kubectl get psmdb-backup
```

And the following command will list available clusters:

```
$ kubectl get psmdb
```

Note

If you have configured storing operations logs for point-in-time recovery, you will have possibility to roll back the cluster to a specific date and time. Otherwise, restoring backups without point-in-time recovery is the only option.

When the correct names for the backup and the cluster are known, backup restoration can be done in the following way.

Without point-in-time recovery

1. Set appropriate keys in the deploy/backup/restore.yaml file.
    - set `spec.clusterName` key to the name of the target cluster to restore the backup on,
    - set `spec.backupSource` subsection to point on the appropriate cloud storage. This `backupSource` subsection should contain the backup type (either `logical` or `physical`), and a `destination` key, followed by necessary storage configuration keys, same as in the `deploy/cr.yaml` file:

    ```
    ...
    backupSource:
      type: logical
      destination: s3://S3-BUCKET-NAME/BACKUP-NAME
      s3:
        credentialsSecret: my-cluster-name-backup-s3
        region: us-west-2
        endpointUrl: https://URL-OF-THE-S3-COMPATIBLE-STORAGE
    ```

    As you have noticed, `destination` value is composed of three parts in case of S3-compatible storage: the `s3://` prefix, the s3 bucket name, and the actual backup name, which you have already found out using the `kubectl get psmdb-backup` command). For Azure Blob storage, you don't put the prefix, and use your container name as an equivalent of a bucket.

    - you can also use a `storageName` key to specify the exact name of the storage (the actual storage should be already defined in the `backup.storages` subsection of the `deploy/cr.yaml` file):

    ```
    ...
    storageName: s3-us-west
    backupSource:
      destination: s3://S3-BUCKET-NAME/BACKUP-NAME
    ```

2. After that, the actual restoration process can be started as follows:

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

With point-in-time recovery

1. Set appropriate keys in the deploy/backup/restore.yaml file.
    - set `spec.clusterName` key to the name of the target cluster to restore the backup on
    - put additional restoration parameters to the `pitr` section:
        - `type` key can be equal to one of the following options
            - `date` - roll back to specific date
            - `latest` - recover to the latest possible transaction
        - `date` key is used with `type=date` option and contains value in datetime format
    - set `spec.backupSource` subsection to point on the appropriate cloud storage. For S3-compatible storage this `backupSource` subsection should contain a `destination` key equal to the s3 bucket with a special `s3://` prefix, followed by necessary S3 configuration keys, same as in `deploy/cr.yaml` file:

    ```
    apiVersion: psmdb.percona.com/v1
    kind: PerconaServerMongoDBRestore
    metadata:
      name: restore1
    spec:
      clusterName: my-cluster-name
      pitr:
        type: date
        date: YYYY-MM-DD hh:mm:ss
    backupSource:
      destination: s3://S3-BUCKET-NAME/BACKUP-NAME
      s3:
        credentialsSecret: my-cluster-name-backup-s3
        region: us-west-2
        endpointUrl: https://URL-OF-THE-S3-COMPATIBLE-STORAGE
    ```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

15.5 How to use backups to move the external database to Kubernetes

The Operator allows restoring a backup not only on the Kubernetes cluster where it was made, but also on any Kubernetes-based environment with the installed Operator, and the backup/restore tool actually used by the Operator is the Percona Backup for MongoDB. That makes it possible to *move* external MongoDB Cluster to Kubernetes with Percona Backup for MongoDB.

Note

There are other scenarios for migrating MongoDB database to Kubernetes as well. For example, this blogpost covers migration based on the regular MongoDB replication capabilities.

Backups can be stored either locally, or remotely (on Amazon S3 or S3-compatible storage, or on Azure Blob Storage). S3-compatible storage to be used for backups.

1. Make sure the following prerequisite requirements are satisfied within your setup:
   - Percona Backup for MongoDB packages are installed on the replica set nodes of the source cluster following the official installation instructions, and the authentication of the pbm-agent is configured to allow it accessing your database.
   - The Operator and the *destination* cluster should be installed in the Kuberentes-based environment. For simplicity, it's reasonable to have the same topology of the *source* and *destination* clusters, although Percona Backup for MongoDB allows replset-remapping as well.
2. Configure the cloud storage for backups on your *source* cluster following the official guide. For example, using the Amazon S3 storage can be configured with the following YAML file:

   ``` yaml title="pbm_config.yaml" type: s3 s3: region: us-west-2 bucket: pbm-test-bucket credentials: access-key-id: secret-access-key:

   ````
   After putting all needed details into the file (`AWS_ACCESS_KEY_ID`,
   `AWS_SECRET_ACCESS_KEY`, the S3 bucket and region in the above example),
   provide the config file to the pbm-agent on all nodes as follows:

   ``` {.bash data-prompt="$" }
   $ pbm config --file pbm_config.yaml
   ````
3. Start the pbm-agent:

   ```
   $ sudo systemctl start pbm-agent
   ```
4. Now you can make backup as follows:

   ```
   $ pbm backup --wait
   ```

   The command output will contain the *backup name*, which you will further use to restore the backup:

   ```
   Starting backup '2022-06-15T08:18:44Z'....
   Waiting for '2022-06-15T08:18:44Z' backup.......... done

   pbm-conf> pbm status -s backups

   Backups:
   ========
   FS  /data/pbm
     Snapshots:
       2022-06-15T08:18:44Z 28.23KB <logical> [complete: 2022-06-15T08:18:49Z]
   ```
5. The rest of operations will be carried out on your *destination* cluster in a Kubernetes-based environment of your choice. These actions are described in the How to restore backup to a new Kubernetes-based environment guide. Just use the proper name of the backup (`2022-06-15T08:18:44Z`) in the above example, and proper parameters specific to your cloud storage (e.g. the `pbm-test-bucket` bucket name we used above).

CONTACT US

For free technical help, visit the Percona Community Forum.

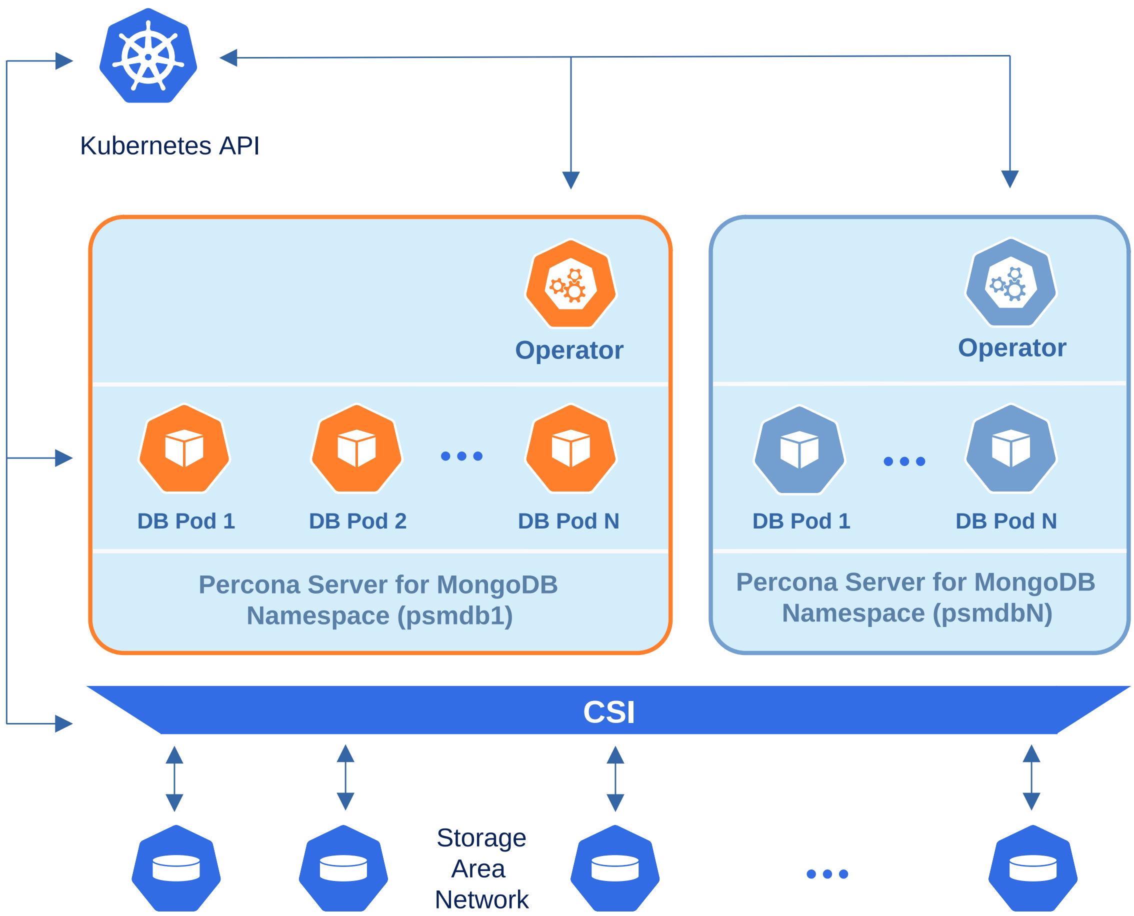
To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-13

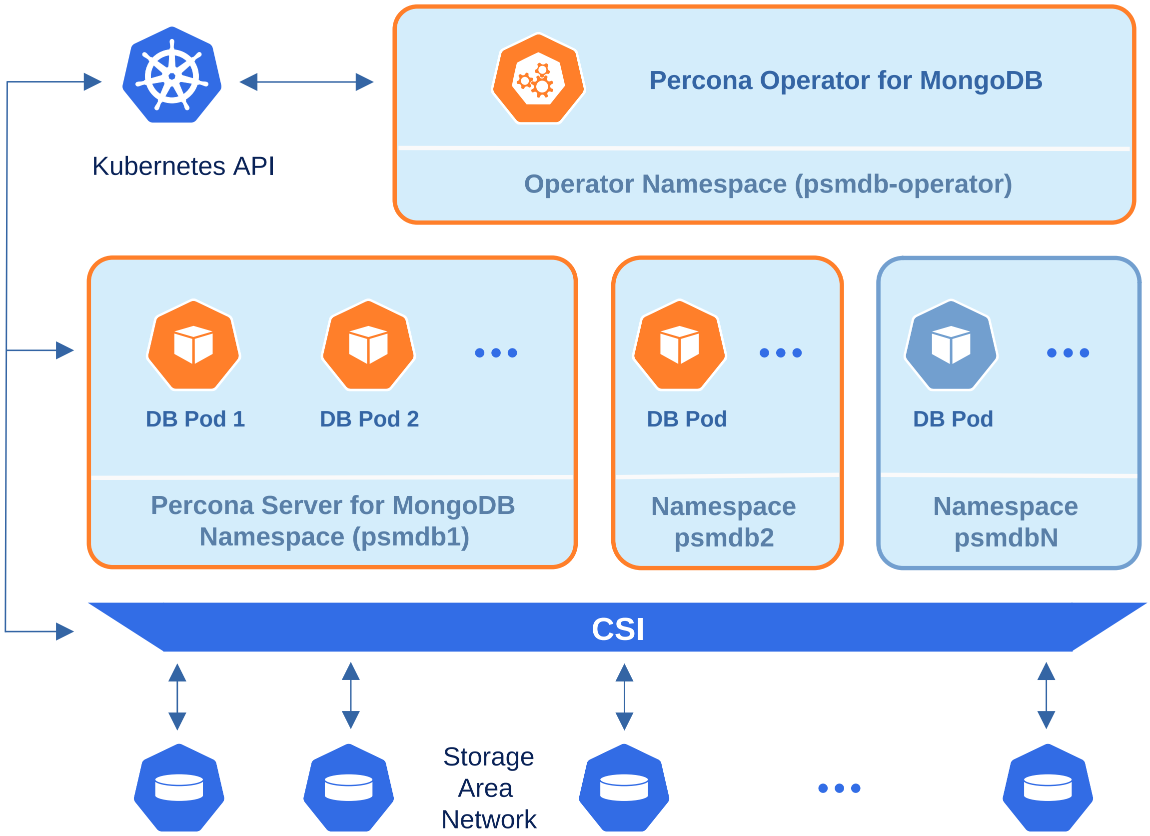
15.6 Install Percona Operator for MongoDB in multi-namespace (cluster-wide) mode

By default, Percona Operator for MongoDB functions in a specific Kubernetes namespace. You can create one during installation (like it is shown in the installation instructions) or just use the `default` namespace. This approach allows several Operators to co-exist in one Kubernetes-based environment, being separated in different namespaces:



Still, sometimes it is more convenient to have one Operator watching for Percona Server for MongoDB Custom Resources in several namespaces.

We recommend running Percona Operator for MongoDB in a traditional way, limited to a specific namespace. But it is possible to run it in so-called *cluster-wide* mode, one Operator watching several namespaces, if needed:



Note

Please take into account that if several Operators are configured to watch the same namespace, it is entirely unpredictable which one will get ownership of the Custom Resource in it, so this situation should be avoided.

To use the Operator in such *cluster-wide* mode, you should install it with a different set of configuration YAML files, which are available in the `deploy` folder and have filenames with a special `cw-` prefix: e.g. `deploy/cw-bundle.yaml`.

While using this cluster-wide versions of configuration files, you should set the following information there:

- `subjects.namespace` option should contain the namespace which will host the Operator,
- `WATCH_NAMESPACE` key-value pair in the `env` section should have `value` equal to a comma-separated list of the namespaces to be watched by the Operator, *and* the namespace in which the Operator resides (or just a blank string to make the Operator deal with *all namespaces* in a Kubernetes cluster).

The following simple example shows how to install Operator cluster-wide on Kubernetes.

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

```
$ git clone -b v1.15.0 https://github.com/percona/percona-server-mongodb-operator
$ cd percona-server-mongodb-operator
```

2. Let's suppose that Operator's namespace should be the `psmdb-operator` one. Create it as follows:

```
$ kubectl create namespace psmdb-operator
```

Namespaces to be watched by the Operator should be created in the same way if not exist. Let's say the Operator should watch the `psmdb` namespace:

```
$ kubectl create namespace psmdb
```

3. Edit the `deploy/cw-bundle.yaml` configuration file to set proper namespaces:

```
...
subjects:
- kind: ServiceAccount
  name: percona-server-mongodb-operator
  namespace: "psmdb-operator"
...
env:
         - name: WATCH_NAMESPACE
           value: "psmdb"
...
```

4. Apply the `deploy/cw-bundle.yaml` file with the following command:

```
$ kubectl apply -f deploy/cw-bundle.yaml -n psmdb-operator
```

5. After the Operator is started, Percona Server for MongoDB can be created at any time by applying the `deploy/cr.yaml` configuration file, like in the case of normal installation:

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

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

```
$ kubectl get psmdb
```

Expected output

```
NAME              ENDPOINT                                      STATUS   AGE
my-cluster-name   my-cluster-name-mongos.psmdb.svc.cluster.local   ready    5m26s
```

15.6.1 Verifying the cluster operation

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

1. You will need the login and password for the admin user to access the cluster. Use `kubectl get secrets` command to see the list of Secrets objects (by default the Secrets object you are interested in has `my-cluster-name-secrets` name). Then `kubectl get secret my-cluster-name-secrets -o yaml` command will return the YAML file with generated Secrets, including the `MONGODB_DATABASE_ADMIN` and `MONGODB_DATABASE_ADMIN_PASSWORD` strings, which should look as follows:

```
...
data:
  ...
  MONGODB_DATABASE_ADMIN_PASSWORD: aDAzQ0pCY3NSWEZ2ZUIzS1I=
  MONGODB_DATABASE_ADMIN_USER: ZGF0YWJhc2VBZG1pbg==
```

   Here the actual login name and password are base64-encoded. Use `echo 'aDAzQ0pCY3NSWEZ2ZUIzS1I=' | base64 --decode` command to bring it back to a human-readable form.

2. Run a container with a MongoDB client and connect its console output to your terminal. The following command will do this, naming the new Pod `percona-client`:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:6.0.9-7
--restart=Never --env="POD_NAMESPACE=psmdb" -- bash -il
```

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

3. Now run `mongo` tool in the percona-client command shell using the login (which is normally `databaseAdmin`) and a proper password obtained from the Secret. The command will look different depending on whether sharding is on (the default behavior) or off:

if sharding is on

```
$ mongo "mongodb://databaseAdmin:databaseAdminPassword@my-cluster-name-
mongos.psmdb.svc.cluster.local/admin?ssl=false"
```

if sharding is off

```
$ mongo "mongodb+srv://databaseAdmin:databaseAdminPassword@my-cluster-name-
rs0.psmdb.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
```


Last update: 2023-05-23

15.7 How to carry on low-level manual upgrades of Percona Server for MongoDB

Percona Operator for MongoDB supports upgrades of the database management system (Percona Server for MongoDB) starting from the Operator version 1.1.0. The Operator 1.5.0 had automated such upgrades with a new upgrade strategy called Smart Update. Smart Update automates the upgrade process while giving the user full control over updates, so it is the most convenient upgrade strategy.

Still there may be use cases when automatic upgrade of Percona Server for MongoDB is not an option (for example, you may be using Percona Server for MongoDB with the Operator version 1.5.0 or earlier), and you have to carry on upgrades manually.

Percona Server for MongoDB can be upgraded manually using one of the following *upgrade strategies*:

- *Rolling Update*, initiated manually and controlled by Kubernetes,
- *On Delete*, done by Kubernetes on per-Pod basis when Pods are manually deleted.

Warning

In case of Smart Updates, the Operator can either detect the availability of the Percona Server for MongoDB version or rely on the user's choice of the version. In both cases Pods are restarted by the Operator automatically in the order, which assures the primary instance to be updated last, preventing possible connection issues until the whole cluster is updated to the new settings. Kubernetes-controlled Rolling Update can't guarantee that Pods update order is optimal from the Percona Server for MongoDB point of view.

15.7.1 Rolling Update strategy and semi-automatic updates

Semi-automatic update of Percona Server for MongoDB can be done as follows:

1. Edit the `deploy/cr.yaml` file, setting `updateStrategy` key to `RollingUpdate`.
2. Now you should apply a patch to your Custom Resource, setting necessary image names with a newer version tag.

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

Patching Custom Resource is done with the `kubectl patch psmdb` command. Actual image names can be found in the list of certified images. For example, updating to the `1.15.0` version should look as follows:

```
$ kubectl patch psmdb my-cluster-name --type=merge --patch '{
   "spec": {
      "crVersion":"1.15.0",
      "image": "percona/percona-server-mongodb:4.4.24-23",
      "backup": { "image": "percona/percona-backup-mongodb:2.3.0" },
      "pmm": { "image": "percona/pmm-client:2.39.0" }
   }}'
```

> **Warning**
>
> The above command upgrades various components of the cluster including PMM Client. It is highly recommended to upgrade PMM Server **before** upgrading PMM Client. If it wasn't done and you would like to avoid PMM Client upgrade, remove it from the list of images, reducing the last of two patch commands as follows:
>
> ```
> $ kubectl patch psmdb my-cluster-name --type=merge --patch '{
>    "spec": {
>       "crVersion":"1.15.0",
>       "image": "percona/percona-server-mongodb:4.4.24-23",
>       "backup": { "image": "percona/percona-backup-mongodb:2.3.0" }
>    }}'
> ```

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 my-cluster-name-rs0
```

15.7.2 Manual upgrade (the On Delete strategy)

Manual update of Percona Server for MongoDB can be done as follows:

1. Edit the `deploy/cr.yaml` file, setting `updateStrategy` key to `OnDelete`.
2. Now you should apply a patch to your Custom Resource, setting necessary image names with a newer version tag.

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

   Patching Custom Resource is done with the `kubectl patch psmdb` command. Actual image names can be found in the list of certified images. For example, updating to the `1.15.0` version should look as follows.

   ```
   $ kubectl patch psmdb my-cluster-name --type=merge --patch '{
      "spec": {
         "crVersion":"1.15.0",
         "image": "percona/percona-server-mongodb:4.4.24-23",
         "backup": { "image": "percona/percona-backup-mongodb:2.3.0" },
         "pmm": { "image": "percona/pmm-client:2.39.0" }
      }}'
   ```

   > **Warning**
   >
   > The above command upgrades various components of the cluster including PMM Client. It is highly recommended to upgrade PMM Server **before** upgrading PMM Client. If it wasn't done and you would like to avoid PMM Client upgrade, remove it from the list of images, reducing the last of two patch commands as follows:
   >
   > ```
   > $ kubectl patch psmdb my-cluster-name --type=merge --patch '{
   >    "spec": {
   >       "crVersion":"1.15.0",
   >       "image": "percona/percona-server-mongodb:4.4.24-23",
   >       "backup": { "image": "percona/percona-backup-mongodb:2.3.0" }
   >    }}'
   > ```

3. The Pod with the newer Percona Server for MongoDB image will start after you delete it. Delete targeted Pods manually one by one to make them restart in the desired order:
   a. Delete the Pod using its name with the command like the following one:

   ```
   $ kubectl delete pod my-cluster-name-rs0-2
   ```

   b. Wait until Pod becomes ready:

   ```
   $ kubectl get pod my-cluster-name-rs0-2
   ```

   The output should be like this:

   ```
   NAME                    READY   STATUS    RESTARTS   AGE
   my-cluster-name-rs0-2   1/1     Running   0          3m33s
   ```

4. The update process is successfully finished when all Pods have been restarted (including the mongos and Config Server nodes, if Percona Server for MongoDB Sharding is on).

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-22

15.8 Monitor Kubernetes

Monitoring the state of the database is crucial to timely identify and react to performance issues. Percona Monitoring and Management (PMM) solution enables you to do just that.

However, the database state also depends on the state of the Kubernetes cluster itself. Hence it's important to have metrics that can depict the state of the Kubernetes cluster.

This document describes how to set up monitoring of the Kubernetes cluster health. This setup has been tested with the PMM server as the centralized data storage and the Victoria Metrics Kubernetes monitoring stack as the metrics collector. These steps may also apply if you use another Prometheus-compatible storage.

15.8.1 Considerations

In this setup, we use Victoria Metrics Kubernetes monitoring stack Helm chart. When customizing the chart's values, consider the following:

- Since we use the PMM Server for monitoring, there is no need to store the data in Victoria Metrics Operator. Therefore, the Victoria Metrics Helm chart is installed with the `vmsingle.enabled` and `vmcluster.enabled` parameters set to `false` in this setup.
- The Prometheus node exporter is not installed by default since it requires privileged containers with the access to the host file system. If you need the metrics for Nodes, enable the Prometheus node exporter by setting the `prometheus-node-exporter.enabled` flag in the Victoria Metrics Helm chart to `true`.
- Check all the role-based access control (RBAC) rules of the `victoria-metrics-k8s-stack` chart and the dependencies chart, and modify them based on your requirements.

15.8.2 Pre-requisites

To set up monitoring of Kubernetes, you need the following:

1. PMM Server up and running. You can run PMM Server as a Docker image, a virtual appliance, or on an AWS instance. Please refer to the official PMM documentation for the installation instructions.
2. Helm v3.
3. kubectl.

15.8.3 Procedure

Set up authentication in PMM Server

To access the PMM Server resources and perform actions on the server, configure authentication.

1. Get the PMM API key. The key must have the role "Admin".

   From PMM UI

   Generate the PMM API key

   From command line

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

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

   Note

   The API key is not rotated.

2. Encode the API key with base64.

   in Linux

```
$ echo -n <API-key> | base64 --wrap=0
```

   in macOS

```
$ echo -n <API-key> | base64
```

3. Create the Namespace where you want to set up monitoring. The following command creates the Namespace `monitoring-system`. You can specify a different name. In the latter steps, specify your namespace instead of the `<namespace>` placeholder.

```
$ kubectl create namespace monitoring-system
```

4. Create the YAML file for the Kubernetes Secrets and specify the base64-encoded API key value within. Let's name this file `pmm-api-vmoperator.yaml`.

```
yaml title="pmm-api-vmoperator.yaml"
apiVersion: v1
data:
  api_key: <base-64-encoded-API-key>
kind: Secret
metadata:
  name: pmm-token-vmoperator
  #namespace: default
type: Opaque
```

5. Create the Secrets object using the YAML file you created previously. Replace the `<filename>` placeholder with your value.

```
$ kubectl apply -f pmm-api-vmoperator.yaml -n <namespace>
```

6. Check that the secret is created. The following command checks the secret for the resource named `pmm-token-vmoperator` (as defined in the `metadata.name` option in the secrets file). If you defined another resource name, specify your value.

```
$ kubectl get secret pmm-token-vmoperator -n <namespace>
```

Create a ConfigMap to mount for `kube-state-metrics`

The `kube-state-metrics` (KSM) is a simple service that listens to the Kubernetes API server and generates metrics about the state of various objects - Pods, Deployments, Services and Custom Resources.

To define what metrics the `kube-state-metrics` should capture, create the ConfigMap and mount it to a container.

Use the example `configmap.yaml` configuration file to create the ConfigMap.

```
$ kubectl apply -f https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/main/vm-
operator-k8s-stack/ksm-configmap.yaml -n <namespace>
```

As a result, you have the `customresource-config-ksm` ConfigMap created.

Install the Victoria Metrics Kubernetes monitoring stack

1. Add the dependency repositories of victoria-metrics-k8s-stack chart.

```
$ helm repo add grafana https://grafana.github.io/helm-charts
$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

2. Add the Victoria Metrics Kubernetes monitoring stack repository.

```
$ helm repo add vm https://victoriametrics.github.io/helm-charts/
```

3. Update the repositories.

```
$ helm repo update
```

4. Install the Victoria Metrics Kubernetes monitoring stack Helm chart. You need to specify the following configuration:
    - the URL to access the PMM server in the `externalVM.write.url` option in the format `<PMM-SERVER-URL>/victoriametrics/api/v1/write`. The URL can contain either the IP address or the hostname of the PMM server.
    - the unique name or an ID of the Kubernetes cluster in the `vmagent.spec.externalLabels.k8s_cluster_id` option. Ensure to set different values if you are sending metrics from multiple Kubernetes clusters to the same PMM Server.

Command line

Use the following command to install the Victoria Metrics Operator and pass the required configuration. The `vm-k8s` value command is the Release name. You can use a different name. Replace the `<namespace>` placeholder with your value. The Namespace must be the same as the Namespace for the Secret and ConfigMap:

```
$ helm install vm-k8s vm/victoria-metrics-k8s-stack \
 -f https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/main/vm-operator-k8s-stack/values.yaml \
 --set externalVM.write.url=<PMM-SERVER-URL>/victoriametrics/api/v1/write \
 --set vmagent.spec.externalLabels.k8s_cluster_id=<UNIQUE-CLUSTER-IDENTIFER/NAME> \
 -n <namespace>
```

To illustrate, say your PMM Server URL is `https://pmm-example.com`, the cluster ID is `test-cluster` and the Namespace is `monitoring-system`. Then the command would look like this:

```
$ helm install vm-k8s vm/victoria-metrics-k8s-stack \
 -f https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/main/vm-operator-k8s-stack/values.yaml \
 --set externalVM.write.url=https://pmm-example.com/victoriametrics/api/v1/write \
 --set vmagent.spec.externalLabels.k8s_cluster_id=test-cluster> \
 -n monitoring-system
```

Configuration file

a. Edit the `values.yaml`

```
externalVM:
  write:
    # Replace PMM-SERVER-URL with valid URL of PMM Server
    url: "https://<PMM-SERVER-URL>//victoriametrics/api/v1/write"

....

vmagent:
  # spec for VMAgent crd
  # https://docs.victoriametrics.com/operator/api.html#vmagentspec
  spec:
    selectAllByDefault: true
    image:
      tag: v1.91.3
    scrapeInterval: 25s
    externalLabels:
      k8s_cluster_id: <cluster-name>
```

Optionally, check the rest of the file and make changes. For example, if you plan to gather metrics for Nodes with the Prometheus node exporter, set the `prometheus-node-exporter.enabled` option to `true`.

b. Run the following command to install the Victoria Metrics kubernetes monitoring stack. The `vm-k8s` value is the Release name. You can use a different name. Replace the `<namespace>` placeholder with your value. The Namespace must be the same as the Namespace for the Secret and ConfigMap.

```
$ kubectl apply -f values.yaml -n <namespace>
```

Note

The example `values.yaml` file is taken from the `victoria-metrics-k8s-stack` version 0.17.5. The fields and default values may differ in newer releases of the `victoria-metrics-k8s-stack` Helm chart. Please check them if you are using a different version of the `victoria-metrics-k8s-stack` Helm chart.

5. Validate the successful installation by checking the Pods.

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

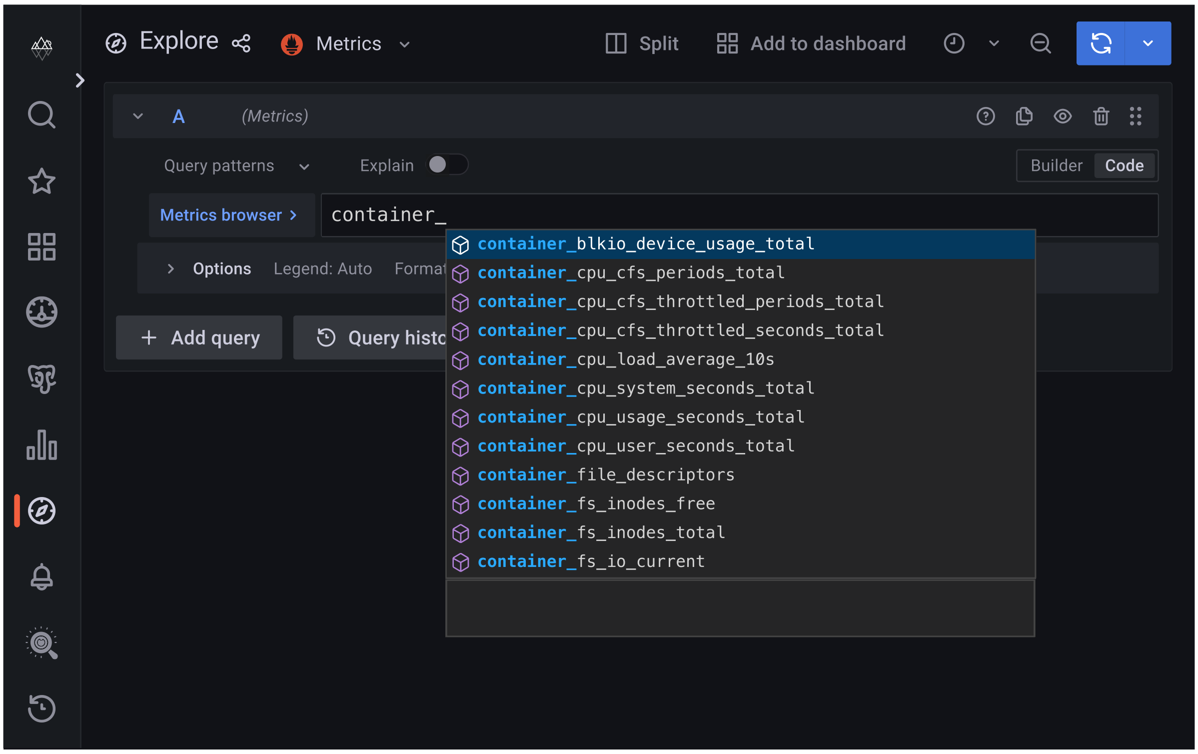
Sample output

```
NAME                                                  READY   STATUS    RESTARTS   AGE
vm-k8s-grafana-5f6bdb8c7c-d5bw5                       3/3     Running   0          90m
vm-k8s-kube-state-metrics-57c5977d4f-6jtbj            1/1     Running   0          81m
vm-k8s-prometheus-node-exporter-kntfk                 1/1     Running   0          90m
vm-k8s-prometheus-node-exporter-mjrvj                 1/1     Running   0          90m
vm-k8s-prometheus-node-exporter-v98c8                 1/1     Running   0          90m
vm-k8s-victoria-metrics-operator-6b7f4f786d-sctp8     1/1     Running   0          90m
vmagent-vm-k8s-victoria-metrics-k8s-stack-fbc86c9db-rz8wk   2/2     Running   0
90m
```

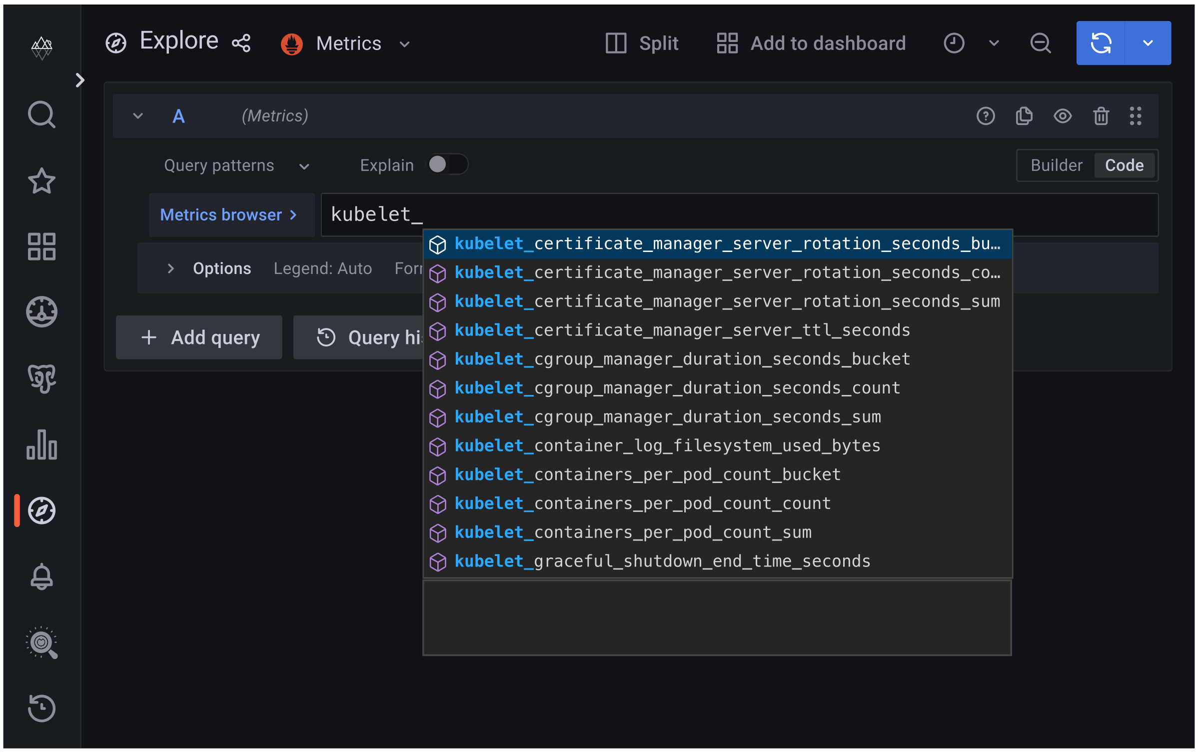
What Pods are running depends on the configuration chosen in values used while installing `victoria-metrics-k8s-stack` chart.

15.8.4 Verify metrics capture

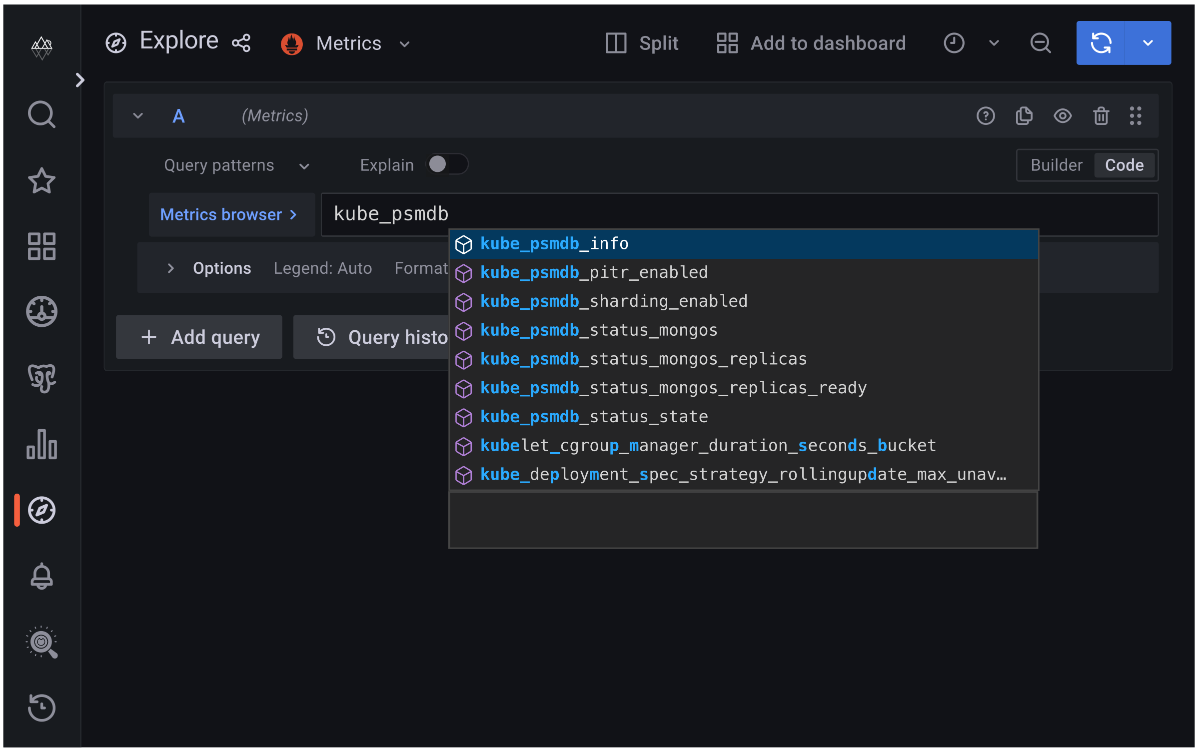
1. Connect to the PMM server.
2. Click **Explore** and switch to the **Code** mode.
3. Check that the required metrics are captured, type the following in the Metrics browser dropdown:
    - cadvisor:



    - kubelet:



- kube-state-metrics metrics that also include Custom resource metrics for the Operator and database deployed in your Kubernetes cluster:



CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-09-11

16. Reference

16.1 Custom Resource options

The operator is configured via the spec section of the deploy/cr.yaml file.

The metadata part of this file contains the following keys:

- `name` (`my-cluster-name` by default) sets the name of your Percona Server for MongoDB Cluster; it should include only URL-compatible characters, not exceed 22 characters, start with an alphabetic character, and end with an alphanumeric character
- `finalizers.delete-psmdb-pods-in-order` if present, activates the Finalizer which controls the proper Pods deletion order in case of the cluster deletion event (on by default)
- `finalizers.delete-psmdb-pvc` if present, activates the Finalizer which deletes appropriate Persistent Volume Claims after the cluster deletion event (off by default)

The spec part of the deploy/cr.yaml file contains the following sections:

Key	Value type	Default	Description
platform	string	`kubernetes`	Override/set the Kubernetes platform: *kubernetes* or *openshift*
pause	boolean	`false`	Pause/resume: setting it to `true` gracefully stops the cluster, and setting it to `false` after shut down starts the cluster back.
unmanaged	boolean	`false`	Unmanaged site in cross-site replication: setting it to `true` forces the Operator to run the cluster in unmanaged state - nodes do not form replica sets, operator does not control TLS certificates
crVersion	string	`1.15.0`	Version of the Operator the Custom Resource belongs to
image	string	`percona/percona-server-mongodb:6.0.9-7`	The Docker image of Percona Server for MongoDB to deploy (actual image names can be found in the list of certified images)
imagePullPolicy	string	`Always`	The policy used to update images
tls.certValidityDuration	string	`2160h`	The validity duration of the external certificate for cert manager (90 days by default). This value is used only at cluster creation time and can't be changed for existing clusters
imagePullSecrets.name	string	`private-registry-credentials`	The Kubernetes ImagePullSecret to access the custom registry
initImage	string	`percona/percona-server-mongodb-operator:1.15.0`	An alternative image for the initial Operator installation
initContainerSecurityContext	subdoc	`{}`	A custom Kubernetes Security Context for a Container for the initImage (image, which can be used instead of the default one while the initial Operator installation)
ClusterServiceDNSSuffix	string	`svc.cluster.local`	The (non-standard) cluster domain to be used as a suffix of the Service name
clusterServiceDNSMode	string	`Internal`	Can be `internal` (local fully-qualified domain names will be used in replset configuration even if the replset is exposed - the default value), `external` (exposed MongoDB instances will use ClusterIP addresses), or `ServiceMesh` (turned on for the exposed Services). Being set, `ServiceMesh` value suprecedes multiCluster settings, and therefore these two modes cannot be combined together.

Key	Value type	Default	Description
allowUnsafeConfigurations	boolean	`false`	Prevents users from configuring a cluster with unsafe parameters: starting it with less than 3 replica set instances, with an even number of replica set instances without additional arbiter, or without TLS/SSL certificates, or running a sharded cluster with less than 3 config server Pods or less than 2 mongos Pods (if `false`, the Operator will automatically change unsafe parameters to safe defaults). **After switching to unsafe configurations permissive mode you will not be able to switch the cluster back by setting `spec.allowUnsafeConfigurations` key to `false`, the flag will be ignored**
updateStrategy	string	`SmartUpdate`	A strategy the Operator uses for upgrades. Possible values are SmartUpdate, RollingUpdate and OnDelete
ignoreAnnotations	subdoc	`service.beta.kubernetes.io/aws-load-balancer-backend-protocol`	The list of annotations to be ignored by the Operator
ignoreLabels	subdoc	`rack`	The list of labels to be ignored by the Operator
multiCluster.enabled	boolean	`false`	Multi-cluster Services (MCS): setting it to `true` enables MCS cluster mode
multiCluster.DNSSuffix	string	`svc.clusterset.local`	The cluster domain to be used as a suffix for multi-cluster Services used by Kubernetes (`svc.clusterset.local` by default)
upgradeOptions	subdoc		Upgrade configuration section
secrets	subdoc		Operator secrets section
replsets	subdoc		Operator MongoDB Replica Set section
pmm	subdoc		Percona Monitoring and Management section
sharding	subdoc		MongoDB sharding configuration section
backup	subdoc		Percona Server for MongoDB backups section

16.1.1 Upgrade Options Section

The `upgradeOptions` section in the deploy/cr.yaml file contains various configuration options to control Percona Server for MongoDB upgrades.

Key	upgradeOptions.versionServiceEndpoint
Value	string
Example	`https://check.percona.com`
Description	The Version Service URL used to check versions compatibility for upgrade

Key	upgradeOptions.apply
Value	string
Example	`disabled`
Description	Specifies how updates are processed by the Operator. `Never` or `Disabled` will completely disable automatic upgrades, otherwise it can be set to `Latest` or `Recommended` or to a specific version string of Percona Server for MongoDB (e.g. `6.0.9-7`) that is wished to be version-locked (so that the user can control the version running, but use automatic upgrades to move between them)

Key	upgradeOptions.schedule
Value	string
Example	`0 2 \* \* \*`
Description	Scheduled time to check for updates, specified in the crontab format

Key	upgradeOptions.setFCV
Value	boolean
Example	`false`
Description	If enabled, FeatureCompatibilityVersion (FCV) will be set to match the version during major version upgrade

16.1.2 Secrets section

Each spec in its turn may contain some key-value pairs. The secrets one has only two of them:

Key	secrets.key
Value	string
Example	`my-cluster-name-mongodb-key`
Description	The secret name for the MongoDB Internal Auth Key. This secret is auto-created by the operator if it doesn't exist.

Key	secrets.users
Value	string
Example	`my-cluster-name-mongodb-users`
Description	The name of the Secrets object for the MongoDB users **required to run the operator.**

Key	secrets.ssl
Value	string
Example	`my-custom-ssl`
Description	A secret with TLS certificate generated for *external* communications, see Transport Layer Security (TLS) for details

Key	secrets.sslInternal
Value	string
Example	`my-custom-ssl-internal`
Description	A secret with TLS certificate generated for *internal* communications, see Transport Layer Security (TLS) for details

Key	secrets.encryptionKey
Value	string
Example	`my-cluster-name-mongodb-encryption-key`
Description	Specifies a secret object with the encryption key

Key	secrets.vault
Value	string
Example	`my-cluster-name-vault`
Description	Specifies a secret object to provide integration with HashiCorp Vault

16.1.3 Replsets Section

The replsets section controls the MongoDB Replica Set.

Key	replsets.name
Value	string
Example	`rs 0`
Description	The name of the MongoDB Replica Set

Key	replsets.size
Value	int
Example	3
Description	The size of the MongoDB Replica Set, must be >= 3 for High-Availability

Key	replsets.terminationGracePeriodSeconds
Value	int
Example	300
Description	The amount of seconds Kubernetes will wait for a clean replica set Pods termination

Key	replsets.topologySpreadConstraints.labelSelector.matchLabels
Value	label
Example	`app.kubernetes.io/name: percona-server-mongodb`
Description	The Label selector for the Kubernetes Pod Topology Spread Constraints

Key	replsets.topologySpreadConstraints.maxSkew
Value	int
Example	1
Description	The degree to which Pods may be unevenly distributed under the Kubernetes Pod Topology Spread Constraints

Key	replsets.topologySpreadConstraints.topologyKey
Value	string
Example	`kubernetes.io/hostname`
Description	The key of node labels for the Kubernetes Pod Topology Spread Constraints

Key	replsets.topologySpreadConstraints.whenUnsatisfiable
Value	string
Example	`DoNotSchedule`
Description	What to do with a Pod if it doesn’t satisfy the Kubernetes Pod Topology Spread Constraints

Key	replsets.configuration
Value	string

Example	<pre>|
	net:
	  tls:
	    mode: preferTLS
	operationProfiling:
	  mode: slowOp
	systemLog:
	  verbosity: 1
	storage:
	  engine: wiredTiger
	  wiredTiger:
	    engineConfig:
	      directoryForIndexes: false
	      journalCompressor: snappy
	    collectionConfig:
	      blockCompressor: snappy
	    indexConfig:
	      prefixCompression: true</pre>

Description	Custom configuration options for mongod. Please refer to the official manual for the full list of options, and specific Percona Server for MongoDB docs.

Key	replsets.affinity.antiAffinityTopologyKey
Value	string
Example	`kubernetes.io/hostname`
Description	The Kubernetes topologyKey node affinity constraint for the Replica Set nodes

Key	replsets.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	replsets.tolerations.key
Value	string
Example	`node.alpha.kubernetes.io/unreachable`
Description	The Kubernetes Pod tolerations key for the Replica Set nodes

Key	replsets.tolerations.operator
Value	string
Example	`Exists`
Description	The Kubernetes Pod tolerations operator for the Replica Set nodes

Key	replsets.tolerations.effect
Value	string
Example	`NoExecute`

Description	The Kubernetes Pod tolerations effect for the Replica Set nodes

Key	replsets.tolerations.tolerationSeconds
Value	int
Example	`6000`
Description	The Kubernetes Pod tolerations time limit for the Replica Set nodes

Key	replsets.priorityClassName
Value	string
Example	`high priority`
Description	The Kuberentes Pod priority class for the Replica Set nodes

Key	replsets.annotations
Value	string
Example	`iam.amazonaws.com/role: role-arn`
Description	The Kubernetes annotations metadata for the Replica Set nodes

Key	replsets.labels
Value	label
Example	`rack: rack-22`
Description	The Kubernetes affinity labels for the Replica Set nodes

Key	replsets.nodeSelector
Value	label
Example	`disktype: ssd`
Description	The Kubernetes nodeSelector affinity constraint for the Replica Set nodes

Key	replsets.storage.engine
Value	string
Example	`wiredTiger`
Description	Sets the storage.engine option https://docs.mongodb.com/manual/reference/configuration-options/#storage.engine`_ for the Replica Set nodes

Key	replsets.storage.wiredTiger.engineConfig.cacheSizeRatio
Value	float
Example	`0.5`
Description	The ratio used to compute the storage.wiredTiger.engineConfig.cacheSizeGB option for the Replica Set nodes

Key	replsets.storage.wiredTiger.engineConfig.directoryForIndexes
Value	bool

Example	`false`
Description	Sets the storage.wiredTiger.engineConfig.directoryForIndexes option for the Replica Set nodes

Key	replsets.storage.wiredTiger.engineConfig.journalCompressor
Value	string
Example	`snappy`
Description	Sets the storage.wiredTiger.engineConfig.journalCompressor option for the Replica Set nodes

Key	replsets.storage.wiredTiger.collectionConfig.blockCompressor
Value	string
Example	`snappy`
Description	Sets the storage.wiredTiger.collectionConfig.blockCompressor option for the Replica Set nodes

Key	replsets.storage.wiredTiger.indexConfig.prefixCompression
Value	bool
Example	`true`
Description	Sets the storage.wiredTiger.indexConfig.prefixCompression option for the Replica Set nodes

Key	replsets.storage.inMemory.engineConfig.inMemorySizeRatio
Value	float
Example	`0.9`
Description	The ratio used to compute the storage.engine.inMemory.inMemorySizeGb option for the Replica Set nodes

Key	replsets.livenessProbe.failureThreshold
Value	int
Example	`4`
Description	Number of consecutive unsuccessful tries of the liveness probe to be undertaken before giving up

Key	replsets.livenessProbe.initialDelaySeconds
Value	int
Example	`60`
Description	Number of seconds to wait after the container start before initiating the liveness probe.

Key	replsets.livenessProbe.periodSeconds
Value	int
Example	`30`

Description	How often to perform a liveness probe (in seconds)

Key	replsets.livenessProbe.timeoutSeconds
Value	int
Example	`10`
Description	Number of seconds after which the liveness probe times out

Key	replsets.livenessProbe.startupDelaySeconds
Value	int
Example	`7200`
Description	Time after which the liveness probe is failed if the MongoDB instance didn't finish its full startup yet

Key	replsets.readinessProbe.failureThreshold
Value	int
Example	`8`
Description	Number of consecutive unsuccessful tries of the readiness probe to be undertaken before giving up

Key	replsets.readinessProbe.initialDelaySeconds
Value	int
Example	`10`
Description	Number of seconds to wait after the container start before initiating the readiness probe

Key	replsets.readinessProbe.periodSeconds
Value	int
Example	`3`
Description	How often to perform a readiness probe (in seconds)

Key	replsets.readinessProbe.successThreshold
Value	int
Example	`1`
Description	Minimum consecutive successes for the readiness probe to be considered successful after having failed

Key	replsets.readinessProbe.timeoutSeconds
Value	int
Example	`2`
Description	Number of seconds after which the readiness probe times out

Key	replsets.runtimeClassName
Value	string

Example	`image-rc`
Description	Name of the Kubernetes Runtime Class for Replica Set Pods

Key	replsets.sidecars.image
Value	string
Example	`busybox`
Description	Image for the custom sidecar container for Replica Set Pods

Key	replsets.sidecars.command
Value	array
Example	`["/bin/sh"]`
Description	Command for the custom sidecar container for Replica Set Pods

Key	replsets.sidecars.args
Value	array
Example	`["-c", "while true; do echo echo $(date -u) 'test' >> /dev/null; sleep 5;done"]`
Description	Command arguments for the custom sidecar container for Replica Set Pods

Key	replsets.sidecars.name
Value	string
Example	`rs-sidecar-1`
Description	Name of the custom sidecar container for Replica Set Pods

Key	replsets.sidecars.volumeMounts.mountPath
Value	string
Example	`/volume1`
Description	Mount path of the custom sidecar container volume for Replica Set Pods

Key	replsets.sidecars.volumeMounts.name
Value	string
Example	`sidecar-volume-claim`
Description	Name of the custom sidecar container volume for Replica Set Pods

Key	replsets.sidecarVolumes.name
Value	string
Example	`sidecar-config`
Description	Name of the custom sidecar container volume for Replica Set Pods

Key	replsets.sidecarVolumes.configMap.name
Value	string

Example	`myconfigmap`
Description	Name of the ConfigMap for a custom sidecar container volume for Replica Set Pods

Key	replsets.sidecarVolumes.secret.secretName
Value	string
Example	`sidecar-secret`
Description	Name of the Secret for a custom sidecar container volume for Replica Set Pods

Key	replsets.sidecarPVCs
Value	subdoc
Example	
Description	Persistent Volume Claim for the custom sidecar container volume for Replica Set Pods

Key	replsets.podDisruptionBudget.maxUnavailable
Value	int
Example	`1`
Description	The Kubernetes Pod distribution budget limit specifying the maximum value for unavailable Pods

Key	replsets.podDisruptionBudget.minAvailable
Value	int
Example	`1`
Description	The Kubernetes Pod distribution budget limit specifying the minimum value for available Pods

Key	replsets.splitHorizons.<replicaset-pod-name>.external
Value	string
Example	`rs0-0.mycluster.xyz`
Description	External URI for Split-horizon for replica set Pods of the exposed cluster

Key	replsets.splitHorizons.<replicaset-pod-name>.external-2
Value	string
Example	`rs0-0.mycluster2.xyz`
Description	External URI for Split-horizon for replica set Pods of the exposed cluster

Key	replsets.expose.enabled
Value	boolean
Example	`false`
Description	Enable or disable exposing MongoDB Replica Set nodes with dedicated IP addresses

Key	replsets.expose.exposeType

Value	string
Example	`ClusterIP`
Description	The IP address type to be exposed

Key	replsets.expose.loadBalancerSourceRanges
Value	string
Example	`10.0.0.0/8`
Description	The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations)

Key	replsets.expose.serviceAnnotations
Value	string
Example	`service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http`
Description	The Kubernetes annotations metadata for the MongoDB mongod daemon

Key	replsets.expose.serviceLabels
Value	string
Example	`rack: rack-22`
Description	The Kubernetes labels for the MongoDB Replica Set Service

Key	replsets.nonvoting.enabled
Value	boolean
Example	`false`
Description	Enable or disable creation of Replica Set non-voting instances within the cluster

Key	replsets.nonvoting.size
Value	int
Example	`1`
Description	The number of Replica Set non-voting instances within the cluster

Key	replsets.nonvoting.afinity.antiAffinityTopologyKey
Value	string
Example	`kubernetes.io/hostname`
Description	The Kubernetes topologyKey node affinity constraint for the non-voting nodes

Key	replsets.nonvoting.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	replsets.nonvoting.tolerations.key
Value	string
Example	`node.alpha.kubernetes.io/unreachable`
Description	The Kubernetes Pod tolerations key for the non-voting nodes

Key	replsets.nonvoting.tolerations.operator
Value	string
Example	`Exists`
Description	The Kubernetes Pod tolerations operator for the non-voting nodes

Key	replsets.nonvoting.tolerations.effect
Value	string
Example	`NoExecute`
Description	The Kubernetes Pod tolerations effect for the non-voting nodes

Key	replsets.nonvoting.tolerations.tolerationSeconds
Value	int
Example	`6000`
Description	The Kubernetes Pod tolerations time limit for the non-voting nodes

Key	replsets.nonvoting.priorityClassName
Value	string
Example	`high priority`
Description	The Kuberentes Pod priority class for the non-voting nodes

Key	replsets.nonvoting.annotations
Value	string
Example	`iam.amazonaws.com/role: role-arn`
Description	The Kubernetes annotations metadata for the non-voting nodes

Key	replsets.nonvoting.labels
Value	label
Example	`rack: rack-22`
Description	The Kubernetes affinity labels for the non-voting nodes

Key	replsets.nonvoting.nodeSelector
Value	label
Example	`disktype: ssd`
Description	The Kubernetes nodeSelector affinity constraint for the non-voting nodes

Key	replsets.nonvoting.podDisruptionBudget.maxUnavailable
Value	int
Example	`1`
Description	The Kubernetes Pod distribution budget limit specifying the maximum value for unavailable Pods among non-voting nodes

Key	replsets.nonvoting.podDisruptionBudget.minAvailable
Value	int
Example	`1`
Description	The Kubernetes Pod distribution budget limit specifying the minimum value for available Pods among non-voting nodes

Key	replsets.nonvoting.resources.limits.cpu
Value	string
Example	`300m`
Description	Kubernetes CPU limit for MongoDB container

Key	replsets.nonvoting.resources.limits.memory
Value	string
Example	`0.5G`
Description	Kubernetes Memory limit for MongoDB container

Key	replsets.nonvoting.resources.requests.cpu
Value	string
Example	`300m`
Description	The Kubernetes CPU requests for MongoDB container

Key	replsets.nonvoting.resources.requests.memory
Value	string
Example	`0.5G`
Description	The Kubernetes Memory requests for MongoDB container

Key	replsets.nonvoting.volumeSpec.emptyDir
Value	string
Example	`{}`
Description	The Kubernetes emptyDir volume, i.e. the directory which will be created on a node, and will be accessible to the MongoDB Pod containers

Key	replsets.nonvoting.volumeSpec.hostPath.path
Value	string
Example	`/data`

Description	Kubernetes hostPath volume, i.e. the file or directory of a node that will be accessible to the MongoDB Pod containers

Key	replsets.nonvoting.volumeSpec.hostPath.type
Value	string
Example	`Directory`
Description	The Kubernetes hostPath volume type

Key	replsets.nonvoting.volumeSpec.persistentVolumeClaim.annotations
Value	string
Example	`service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http`
Description	The Kubernetes annotations metadata for Persistent Volume Claim

Key	replsets.nonvoting.volumeSpec.persistentVolumeClaim.labels
Value	string
Example	`rack: rack-22`
Description	The Kubernetes labels metadata for Persistent Volume Claim

Key	replsets.nonvoting.volumeSpec.persistentVolumeClaim.storageClassName
Value	string
Example	`standard`
Description	The Kubernetes Storage Class to use with the MongoDB container Persistent Volume Claim for the non-voting nodes. Use Storage Class with XFS as the default filesystem if possible, [for better MongoDB performance](https://dba.stackexchange.com/questions/190578/is-xfs-still-the-best-choice-for-mongodb

Key	replsets.nonvoting.volumeSpec.persistentVolumeClaim.accessModes
Value	array
Example	`[ "ReadWriteOnce" ]`
Description	The Kubernetes Persistent Volume access modes for the MongoDB container for the non-voting nodes

Key	replsets.nonvoting.volumeSpec.persistentVolumeClaim.resources.requests.storage
Value	string
Example	`3Gi`
Description	The Kubernetes Persistent Volume size for the MongoDB container for the non-voting nodes

Key	replsets.arbiter.enabled
Value	boolean
Example	`false`
Description	Enable or disable creation of Replica Set Arbiter nodes within the cluster

Key	replsets.arbiter.size
Value	int
Example	`1`
Description	The number of Replica Set Arbiter instances within the cluster

Key	replsets.arbiter.afinity.antiAffinityTopologyKey
Value	string
Example	`kubernetes.io/hostname`
Description	The Kubernetes topologyKey node affinity constraint for the Arbiter

Key	replsets.arbiter.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	replsets.arbiter.tolerations.key
Value	string
Example	`node.alpha.kubernetes.io/unreachable`
Description	The Kubernetes Pod tolerations key for the Arbiter nodes

Key	replsets.arbiter.tolerations.operator
Value	string
Example	`Exists`
Description	The Kubernetes Pod tolerations operator for the Arbiter nodes

Key	replsets.arbiter.tolerations.effect
Value	string
Example	`NoExecute`
Description	The Kubernetes Pod tolerations effect for the Arbiter nodes

Key	replsets.arbiter.tolerations.tolerationSeconds
Value	int
Example	`6000`
Description	The Kubernetes Pod tolerations time limit for the Arbiter nodes

Key	replsets.arbiter.priorityClassName
Value	string
Example	`high priority`
Description	The Kuberentes Pod priority class for the Arbiter nodes

Key	replsets.arbiter.annotations
Value	string
Example	`iam.amazonaws.com/role: role-arn`
Description	The Kubernetes annotations metadata for the Arbiter nodes

Key	replsets.arbiter.labels
Value	label
Example	`rack: rack-22`
Description	The Kubernetes affinity labels for the Arbiter nodes

Key	replsets.arbiter.nodeSelector
Value	label
Example	`disktype: ssd`
Description	The Kubernetes nodeSelector affinity constraint for the Arbiter nodes

Key	replsets.resources.limits.cpu
Value	string
Example	`300m`
Description	Kubernetes CPU limit for MongoDB container

Key	replsets.resources.limits.memory
Value	string
Example	`0.5G`
Description	Kubernetes Memory limit for MongoDB container

Key	replsets.resources.requests.cpu
Value	string
Example	`300m`
Description	The Kubernetes CPU requests for MongoDB container

Key	replsets.resources.requests.memory
Value	string
Example	`0.5G`
Description	The Kubernetes Memory requests for MongoDB container

Key	replsets.volumeSpec.emptyDir
Value	string
Example	`{}`
Description	The Kubernetes emptyDir volume, i.e. the directory which will be created on a node, and will be accessible to the MongoDB Pod containers

Key	replsets.volumeSpec.hostPath.path
Value	string
Example	`/data`
Description	Kubernetes hostPath volume, i.e. the file or directory of a node that will be accessible to the MongoDB Pod containers

Key	replsets.volumeSpec.hostPath.type
Value	string
Example	`Directory`
Description	The Kubernetes hostPath volume type

Key	replsets.volumeSpec.persistentVolumeClaim.annotations
Value	string
Example	`service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http`
Description	The Kubernetes annotations metadata for Persistent Volume Claim

Key	replsets.volumeSpec.persistentVolumeClaim.labels
Value	string
Example	`rack: rack-22`
Description	The Kubernetes labels metadata for Persistent Volume Claim

Key	replsets.volumeSpec.persistentVolumeClaim.storageClassName
Value	string
Example	`standard`
Description	The Kubernetes Storage Class to use with the MongoDB container Persistent Volume Claim. Use Storage Class with XFS as the default filesystem if possible, for better MongoDB performance

Key	replsets.volumeSpec.persistentVolumeClaim.accessModes
Value	array
Example	`[ "ReadWriteOnce" ]`
Description	The Kubernetes Persistent Volume access modes for the MongoDB container

Key	replsets.volumeSpec.persistentVolumeClaim.resources.requests.storage
Value	string
Example	`3Gi`
Description	The Kubernetes Persistent Volume size for the MongoDB container

Key	replsets.hostAliases.ip
Value	string
Example	`"10.10.0.2"`

Description	The IP address for Kubernetes host aliases for replica set Pods

Key	replsets.hostAliases.hostnames
Value	subdoc
Example	
Description	Hostnames for Kubernetes host aliases for replica set Pods

16.1.4 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 Server for MongoDB with PMM

Key	pmm.image
Value	string
Example	`percona/pmm-client:2.39.0`
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.mongodParams
Value	string
Example	`--environment=DEV-ENV --custom-labels=DEV-ENV`
Description	Additional parameters which will be passed to the pmm-admin add mongodb command for `mongod` Pods

Key	pmm.mongosParams
Value	string
Example	`--environment=DEV-ENV --custom-labels=DEV-ENV`
Description	Additional parameters which will be passed to the pmm-admin add mongodb command for `mongos` Pods

16.1.5 Sharding Section

The `sharding` section in the deploy/cr.yaml file contains configuration options for Percona Server for MondoDB sharding.

Key	sharding.enabled
Value	boolean
Example	`true`
Description	Enables or disables Percona Server for MondoDB sharding

Key	sharding.configsvrReplSet.size
Value	int
Example	`3`
Description	The number of Config Server instances within the cluster

Key	sharding.configsvrReplSet.terminationGracePeriodSeconds
Value	int
Example	300
Description	The amount of seconds Kubernetes will wait for a clean config server Pods termination

Key	sharding.configsvrReplSet.topologySpreadConstraints.labelSelector.matchLabels
Value	label
Example	`app.kubernetes.io/name: percona-server-mongodb`
Description	The Label selector for the Kubernetes Pod Topology Spread Constraints

Key	sharding.configsvrReplSet.topologySpreadConstraints.maxSkew
Value	int
Example	1
Description	The degree to which Pods may be unevenly distributed under the Kubernetes Pod Topology Spread Constraints

Key	sharding.configsvrReplSet.topologySpreadConstraints.topologyKey
Value	string
Example	`kubernetes.io/hostname`
Description	The key of node labels for the Kubernetes Pod Topology Spread Constraints

Key	sharding.configsvrReplSet.topologySpreadConstraints.whenUnsatisfiable
Value	string
Example	`DoNotSchedule`
Description	What to do with a Pod if it doesn't satisfy the Kubernetes Pod Topology Spread Constraints

Key	sharding.configsvrReplSet.configuration
Value	string

Example	`|`
	`operationProfiling:`
	`  mode: slowOp`
	`systemLog:`
	`  verbosity: 1`

Description	Custom configuration options for Config Servers. Please refer to the official manual for the full list of options

Key	sharding.configsvrReplSet.livenessProbe.failureThreshold
Value	int
Example	`4`
Description	Number of consecutive unsuccessful tries of the liveness probe to be undertaken before giving up

Key	sharding.configsvrReplSet.livenessProbe.initialDelaySeconds
Value	int
Example	`60`
Description	Number of seconds to wait after the container start before initiating the liveness probe

Key	sharding.configsvrReplSet.livenessProbe.periodSeconds
Value	int
Example	`30`
Description	How often to perform a liveness probe (in seconds)

Key	sharding.configsvrReplSet.livenessProbe.timeoutSeconds
Value	int
Example	`10`
Description	Number of seconds after which the liveness probe times out

Key	sharding.configsvrReplSet.livenessProbe.startupDelaySeconds
Value	int
Example	`7200`
Description	Time after which the liveness probe is failed if the MongoDB instance didn’t finish its full startup yet

Key	sharding.configsvrReplSet.readinessProbe.failureThreshold
Value	int
Example	`3`
Description	Number of consecutive unsuccessful tries of the readiness probe to be undertaken before giving up

Key	sharding.configsvrReplSet.readinessProbe.initialDelaySeconds
Value	int

Example	`10`
Description	Number of seconds to wait after the container start before initiating the readiness probe

Key	sharding.configsvrReplSet.readinessProbe.periodSeconds
Value	int
Example	`3`
Description	How often to perform a readiness probe (in seconds)

Key	sharding.configsvrReplSet.readinessProbe.successThreshold
Value	int
Example	`1`
Description	Minimum consecutive successes for the readiness probe to be considered successful after having failed

Key	sharding.configsvrReplSet.readinessProbe.timeoutSeconds
Value	int
Example	`2`
Description	Number of seconds after which the readiness probe times out

Key	sharding.configsvrReplSet.runtimeClassName
Value	string
Example	`image-rc`
Description	Name of the Kubernetes Runtime Class for Config Server Pods

Key	sharding.configsvrReplSet.sidecars.image
Value	string
Example	`busybox`
Description	Image for the custom sidecar container for Config Server Pods

Key	sharding.configsvrReplSet.sidecars.command
Value	array
Example	`["/bin/sh"]`
Description	Command for the custom sidecar container for Config Server Pods

Key	sharding.configsvrReplSet.sidecars.args
Value	array
Example	`["-c", "while true; do echo echo $(date -u) 'test' >> /dev/null; sleep 5;done"]`
Description	Command arguments for the custom sidecar container for Config Server Pods

Key	sharding.configsvrReplSet.sidecars.name
Value	string

Example	`rs-sidecar-1`
Description	Name of the custom sidecar container for Config Server Pods

Key	sharding.configsvrReplSet.limits.cpu
Value	string
Example	`300m`
Description	Kubernetes CPU limit for Config Server container

Key	sharding.configsvrReplSet.limits.memory
Value	string
Example	`0.5G`
Description	Kubernetes Memory limit for Config Server container

Key	sharding.configsvrReplSet.resources.requests.cpu
Value	string
Example	`300m`
Description	The Kubernetes CPU requests for Config Server container

Key	sharding.configsvrReplSet.requests.memory
Value	string
Example	`0.5G`
Description	The Kubernetes Memory requests for Config Server container

Key	sharding.configsvrReplSet.expose.enabled
Value	boolean
Example	`false`
Description	Enable or disable exposing Config Server nodes with dedicated IP addresses

Key	sharding.configsvrReplSet.expose.exposeType
Value	string
Example	`ClusterIP`
Description	The IP address type to be exposed

Key	sharding.configsvrReplSet.expose.loadBalancerSourceRanges
Value	string
Example	`10.0.0.0/8`
Description	The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations)

Key	sharding.configsvrReplSet.expose.serviceAnnotations
Value	string

Example	`service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http`
Description	The Kubernetes annotations metadata for the Config Server daemon

Key	sharding.configsvrReplSet.expose.serviceLabels
Value	string
Example	`rack: rack-22`
Description	The Kubernetes labels for the Config Server Service

Key	sharding.configsvrReplSet.volumeSpec.emptyDir
Value	string
Example	`{}`
Description	The Kubernetes emptyDir volume, i.e. the directory which will be created on a node, and will be accessible to the Config Server Pod containers

Key	sharding.configsvrReplSet.volumeSpec.hostPath.path
Value	string
Example	`/data`
Description	Kubernetes hostPath volume, i.e. the file or directory of a node that will be accessible to the Config Server Pod containers

Key	sharding.configsvrReplSet.volumeSpec.hostPath.type
Value	string
Example	`Directory`
Description	The Kubernetes hostPath volume type

Key	sharding.configsvrReplSet.volumeSpec.persistentVolumeClaim.annotations
Value	string
Example	`service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http`
Description	The Kubernetes annotations metadata for Persistent Volume Claim

Key	sharding.configsvrReplSet.volumeSpec.persistentVolumeClaim.labels
Value	string
Example	`rack: rack-22`
Description	The Kubernetes labels metadata for Persistent Volume Claim

Key	sharding.configsvrReplSet.volumeSpec.persistentVolumeClaim.storageClassName
Value	string
Example	`standard`
Description	The Kubernetes Storage Class to use with the Config Server container Persistent Volume Claim. Use Storage Class with XFS as the default filesystem if possible, for better MongoDB performance

Key	sharding.configsvrReplSet.volumeSpec.persistentVolumeClaim.accessModes
Value	array
Example	`[ "ReadWriteOnce" ]`
Description	The Kubernetes Persistent Volume access modes for the Config Server container

Key	sharding.configsvrReplSet.volumeSpec.persistentVolumeClaim.resources.requests.storage
Value	string
Example	`3Gi`
Description	The Kubernetes Persistent Volume size for the Config Server container

Key	sharding.configsvrReplSet.hostAliases.ip
Value	string
Example	`"10.10.0.2"`
Description	The IP address for Kubernetes host aliases for replica set Pods

Key	sharding.configsvrReplSet.hostAliases.hostnames
Value	subdoc
Example	
Description	Hostnames for Kubernetes host aliases for config server Pods

Key	sharding.mongos.size
Value	int
Example	`3`
Description	The number of mongos instances within the cluster

Key	sharding.mongos.terminationGracePeriodSeconds
Value	int
Example	300
Description	The amount of seconds Kubernetes will wait for a clean mongos Pods termination

Key	sharding.mongos.topologySpreadConstraints.labelSelector.matchLabels
Value	label
Example	`app.kubernetes.io/name: percona-server-mongodb`
Description	The Label selector for the Kubernetes Pod Topology Spread Constraints

Key	sharding.mongos.topologySpreadConstraints.maxSkew
Value	int
Example	1
Description	The degree to which Pods may be unevenly distributed under the Kubernetes Pod Topology Spread Constraints

Key	sharding.mongos.topologySpreadConstraints.topologyKey
Value	string
Example	`kubernetes.io/hostname`
Description	The key of node labels for the Kubernetes Pod Topology Spread Constraints

Key	sharding.mongos.topologySpreadConstraints.whenUnsatisfiable
Value	string
Example	`DoNotSchedule`
Description	What to do with a Pod if it doesn't satisfy the Kubernetes Pod Topology Spread Constraints

Key	sharding.mongos.configuration
Value	string

Example	`|`
	`systemLog:`
	`  verbosity: 1`

Description	Custom configuration options for mongos. Please refer to the official manual for the full list of options

Key	sharding.mongos.afinity.antiAffinityTopologyKey
Value	string
Example	`kubernetes.io/hostname`
Description	The Kubernetes topologyKey node affinity constraint for mongos

Key	sharding.mongos.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	sharding.mongos.tolerations.key
Value	string
Example	`node.alpha.kubernetes.io/unreachable`
Description	The Kubernetes Pod tolerations key for mongos instances

Key	sharding.mongos.tolerations.operator
Value	string
Example	`Exists`
Description	The Kubernetes Pod tolerations operator for mongos instances

Key	sharding.mongos.tolerations.effect
Value	string

Example	`NoExecute`
Description	The Kubernetes Pod tolerations effect for mongos instances

Key	sharding.mongos.tolerations.tolerationSeconds
Value	int
Example	`6000`
Description	The Kubernetes Pod tolerations time limit for mongos instances

Key	sharding.mongos.priorityClassName
Value	string
Example	`high priority`
Description	The Kuberentes Pod priority class for mongos instances

Key	sharding.mongos.annotations
Value	string
Example	`iam.amazonaws.com/role: role-arn`
Description	The Kubernetes annotations metadata for the mongos instances

Key	sharding.mongos.labels
Value	label
Example	`rack: rack-22`
Description	The Kubernetes affinity labels for mongos instances

Key	sharding.mongos.nodeSelector
Value	label
Example	`disktype: ssd`
Description	The Kubernetes nodeSelector affinity constraint for mongos instances

Key	sharding.mongos.livenessProbe.failureThreshold
Value	int
Example	`4`
Description	Number of consecutive unsuccessful tries of the liveness probe to be undertaken before giving up

Key	sharding.mongos.livenessProbe.initialDelaySeconds
Value	int
Example	`60`
Description	Number of seconds to wait after the container start before initiating the liveness probe

Key	sharding.mongos.livenessProbe.periodSeconds
Value	int

Example	`30`
Description	How often to perform a liveness probe (in seconds)

Key	sharding.mongos.livenessProbe.timeoutSeconds
Value	int
Example	`10`
Description	Number of seconds after which the liveness probe times out

Key	sharding.mongos.livenessProbe.startupDelaySeconds
Value	int
Example	`7200`
Description	Time after which the liveness probe is failed if the MongoDB instance didn't finish its full startup yet

Key	sharding.mongos.readinessProbe.failureThreshold
Value	int
Example	`3`
Description	Number of consecutive unsuccessful tries of the readiness probe to be undertaken before giving up

Key	sharding.mongos.readinessProbe.initialDelaySeconds
Value	int
Example	`10`
Description	Number of seconds to wait after the container start before initiating the readiness probe

Key	sharding.mongos.readinessProbe.periodSeconds
Value	int
Example	`3`
Description	How often to perform a readiness probe (in seconds)

Key	sharding.mongos.readinessProbe.successThreshold
Value	int
Example	`1`
Description	Minimum consecutive successes for the readiness probe to be considered successful after having failed

Key	sharding.mongos.readinessProbe.timeoutSeconds
Value	int
Example	`2`
Description	Number of seconds after which the readiness probe times out

Key	sharding.mongos.runtimeClassName

Value	string
Example	`image-rc`
Description	Name of the Kubernetes Runtime Class for mongos Pods

Key	sharding.mongos.sidecars.image
Value	string
Example	`busybox`
Description	Image for the custom sidecar container for mongos Pods

Key	sharding.mongos.sidecars.command
Value	array
Example	`["/bin/sh"]`
Description	Command for the custom sidecar container for mongos Pods

Key	sharding.mongos.sidecars.args
Value	array
Example	`["-c", "while true; do echo echo $(date -u) 'test' >> /dev/null; sleep 5;done"]`
Description	Command arguments for the custom sidecar container for mongos Pods

Key	sharding.mongos.sidecars.name
Value	string
Example	`rs-sidecar-1`
Description	Name of the custom sidecar container for mongos Pods

Key	sharding.mongos.limits.cpu
Value	string
Example	`300m`
Description	Kubernetes CPU limit for mongos container

Key	sharding.mongos.limits.memory
Value	string
Example	`0.5G`
Description	Kubernetes Memory limit for mongos container

Key	sharding.mongos.resources.requests.cpu
Value	string
Example	`300m`
Description	The Kubernetes CPU requests for mongos container

Key	sharding.mongos.requests.memory

Value	string
Example	`0.5G`
Description	The Kubernetes Memory requests for mongos container

Key	sharding.mongos.expose.exposeType
Value	string
Example	`ClusterIP`
Description	The IP address type to be exposed

Key	sharding.mongos.expose.servicePerPod
Value	boolean
Example	`true`
Description	If set to `true`, a separate ClusterIP Service is created for each mongos instance

Key	sharding.mongos.expose.loadBalancerSourceRanges
Value	string
Example	`10.0.0.0/8`
Description	The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations)

Key	sharding.mongos.expose.serviceAnnotations
Value	string
Example	`service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http`
Description	The Kubernetes annotations metadata for the MongoDB mongos daemon

Key	sharding.mongos.expose.serviceLabels
Value	string
Example	`rack: rack-22`
Description	The Kubernetes labels for the MongoDB mongos Service

Key	sharding.mongos.hostAliases.ip
Value	string
Example	`"10.10.0.2"`
Description	The IP address for Kubernetes host aliases for mongos Pods

Key	sharding.mongos.hostAliases.hostnames
Value	subdoc
Example	
Description	Hostnames for Kubernetes host aliases for mongos Pods

16.1.6 Backup Section

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

Key	backup.enabled
Value	boolean
Example	`true`
Description	Enables or disables making backups

Key	backup.image
Value	string
Example	`percona/percona-server-mongodb-operator:1.15.0-backup`
Description	The Percona Server for MongoDB Docker image to use for the backup

Key	backup.serviceAccountName
Value	string
Example	`percona-server-mongodb-operator`
Description	Name of the separate privileged service account used by the Operator

Key	backup.annotations
Value	string
Example	`sidecar.istio.io/inject: "false"`
Description	The Kubernetes annotations metadata for the backup job

Key	backup.resources.limits.cpu
Value	string
Example	`100m`
Description	Kubernetes CPU limit for backups

Key	backup.resources.limits.memory
Value	string
Example	`0.2G`
Description	Kubernetes Memory limit for backups

Key	backup.resources.requests.cpu
Value	string
Example	`100m`
Description	The Kubernetes CPU requests for backups

Key	backup.resources.requests.memory
Value	string
Example	`0.1G`
Description	The Kubernetes Memory requests for backups

Key	backup.storages.<storage-name>.type
Value	string
Example	`s3`
Description	The cloud storage type used for backups. Only `s3` type is currently supported

Key	backup.storages.<storage-name>.s3.insecureSkipTLSVerify
Value	boolean
Example	`true`
Description	Enable or disable verification of the storage server TLS certificate. Disabling it may be useful e.g. to skip TLS verification for private S3-compatible storage with a self-issued certificate

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.<storage-name>.s3.prefix
Value	string
Example	`""`
Description	The path (sub-folder) to the backups inside the bucket

Key	backup.storages.<storage-name>.s3.uploadPartSize
Value	int
Example	`10485760`
Description	The size of data chunks in bytes to be uploaded to the storage bucket (10 MiB by default)

Key	backup.storages.<storage-name>.s3.maxUploadParts
Value	int
Example	`10000`
Description	The maximum number of data chunks to be uploaded to the storage bucket (10000 by default)

Key	backup.storages.<storage-name>.s3.storageClass
Value	string

Example	`STANDARD`
Description	The storage class name of the S3 storage

Key	backup.storages.<storage-name>.s3.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.<storage-name>.s3.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>.s3.serverSideEncryption.kmsKeyID
Value	string
Example	`""`
Description	The ID of the key stored in the AWS KMS used by the Operator for backups server-side encryption

Key	backup.storages.<storage-name>.s3.serverSideEncryption.sseAlgorithm
Value	string
Example	`aws:kms`
Description	The key management mode used for backups server-side encryption with the encryption keys stored in AWS KMS - `aws:kms` is the only supported value for now

Key	backup.storages.<storage-name>.s3.serverSideEncryption.sseCustomerAlgorithm
Value	string
Example	`AES256`
Description	The key management mode for backups server-side encryption with customer-provided keys - `AES256` is the only supported value for now

Key	backup.storages.<storage-name>.s3.serverSideEncryption.sseCustomerKey
Value	string
Example	`""`
Description	The locally-stored base64-encoded custom encryption key used by the Operator for backups server-side encryption on S3-compatible storages

Key	backup.storages.<storage-name>.azure.credentialsSecret
Value	string
Example	`my-cluster-azure-secret`

Description	The Kubernetes secret for backups. It should contain `AZURE_STORAGE_ACCOUNT_NAME` and `AZURE_STORAGE_ACCOUNT_KEY`

Key	backup.storages.<storage-name>.azure.container
Value	string
Example	`my-container`
Description	Name of the container for backups

Key	backup.storages.<storage-name>.azure.prefix
Value	string
Example	`""`
Description	The path (sub-folder) to the backups inside the container

Key	backup.pitr.enabled
Value	boolean
Example	`false`
Description	Enables or disables point-in-time-recovery functionality

Key	backup.pitr.oplogOnly
Value	boolean
Example	false
Description	If true, Percona Backup for MongoDB saves oplog chunks even without the base backup snapshot (oplog chunks without a base backup can't be used to restore a backup by the Operator, but can still be useful for manual restore operations)

Key	backup.pitr.oplogSpanMin
Value	int
Example	`10`
Description	Number of minutes between the uploads of oplogs

Key	backup.pitr.compressionType
Value	string
Example	`gzip`
Description	The point-in-time-recovery chunks compression format, can be gzip, snappy, lz4, pgzip, zstd, s2, or none

Key	backup.pitr.compressionLevel
Value	int
Example	`6`
Description	The point-in-time-recovery chunks compression level (higher values result in better but slower compression)

Key	backup.tasks.name
Value	string
Example	
Description	The name of the backup

Key	backup.tasks.enabled
Value	boolean
Example	`true`
Description	Enables or disables this exact backup

Key	backup.tasks.schedule
Value	string
Example	`0 0 \* \* 6`
Description	The scheduled time to make a backup, specified in the crontab format

Key	backup.tasks.keep
Value	int
Example	`3`
Description	The amount of most recent backups to store. Older backups are automatically deleted. Set `keep` to zero or completely remove it to disable automatic deletion of backups

Key	backup.tasks.storageName
Value	string
Example	`st-us-west`
Description	The name of the S3-compatible storage for backups, configured in the storages subsection

Key	backup.tasks.compressionType
Value	string
Example	`gzip`
Description	The backup compression format, can be gzip, snappy, lz4, pgzip, zstd, s2, or none

Key	backup.tasks.compressionLevel
Value	int
Example	`6`
Description	The backup compression level (higher values result in better but slower compression)

Key	backup.tasks.type
Value	string
Example	`physical`

Description	The backup type: (can be either `logical` (default) or `physical`; see the Operator backups official documentation for details

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

16.2 Percona certified images

Following table presents Percona's certified docker images to be used with the Percona Operator for Percona Server for MongoDB:

Image	Digest
percona/percona-server-mongodb-operator:1.15.0	d8a5b33db1938d42769cb5a87d34a128332a2d0302eaa6d7c860e7c4667ea3b6
percona/pmm-client:2.39.0	4bae6c2fd94e0108a4204d6665ac72100a9cf5f786174ba040395759aebe47de
percona/percona-backup-mongodb:2.3.0	4b0a3a8dcd12619417aea16c7349cacacb41303b7131191ef65f4adff7ca1926
percona/percona-server-mongodb:6.0.9-7	ef1caec49cc5ea6652731d65ebef468b11da607f2aab11360a47d90aeee5dde5
percona/percona-server-mongodb:6.0.5-4	b6f875974c59d8ea0174675c85f41668460233784cbf2cbe7ce5eca212ac5f6a
percona/percona-server-mongodb:6.0.4-3	df46c596e6f7339badec3b36f7f209689c3f31e5391ef714be0701deef555570
percona/percona-server-mongodb:5.0.20-17	9d1a440f602318551b9e1003a7f01c6af3e6b3f6a633eed37ed2ad6e7d176b9d
percona/percona-server-mongodb:5.0.15-13	f0b5a8291d778d7419c20dcf0d1985a0f33770d05e94dba41db8f071957e9929
percona/percona-server-mongodb:5.0.14-12	38ced404ec798d78943c234ab4f39ee237f3013095a5cd581978cfdf0fbce2f9
percona/percona-server-mongodb:5.0.11-10	da3713525d76a354435e1ab8fda12a06407e7eca8b8e72b9ac0163a34c8eb735
percona/percona-server-mongodb:5.0.7-6	3f4849a17236c3849a513f46caa39fbc6da0414f98d27e080fbe0496fa9e86a2
percona/percona-server-mongodb:4.4.24-23	87fa059b4b4460cbea9f79687e5cb158c9fc8fe75094fd6816657cef35701813
percona/percona-server-mongodb:4.4.19-19	daeed5cc326cb6393d0304c247e96eb74f7844e30e2e38df84aa4368371bff81
percona/percona-server-mongodb:4.4.18-18	fe34309f0491f13b4b6b35503f57391e57b1a87d88f6f842b1ca3606ddaf1e14
percona/percona-server-mongodb:4.4.16-16	402b5e5b08ac73c74a47c72d002251a086f9ad28b0594fbae5c34757b294ce13
percona/percona-server-mongodb:4.4.13-13	059c3c9a0360d6823905e39b52bdcaf76c3929c93408c537f139cee835c2bc0f

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

16.3 Versions compatibility

Versions of the cluster components and platforms tested with different Operator releases are shown below. Other version combinations may also work but have not been tested.

Cluster components:

Operator	MongoDB	Percona Backup for MongoDB
1.15.0	4.4 - 6.0	2.3.0
1.14.0	4.4 - 6.0	2.0.4, 2.0.5
1.13.0	4.2 - 5.0	1.8.1
1.12.0	4.2 - 5.0	1.7.0
1.11.0	4.0, 4.2, 4.4, 5.0	1.6.1
1.10.0	4.0, 4.2, 4.4, 5.0	1.6.0
1.9.0	4.0, 4.2, 4.4	1.5.0
1.8.0	3.6, 4.0, 4.2, 4.4	1.4.1
1.7.0	3.6, 4.0, 4.2, 4.4	1.4.1
1.6.0	3.6, 4.0, 4.2	1.3.4
1.5.0	3.6, 4.0, 4.2	1.3.1
1.4.0	3.6, 4.0, 4.2	1.1.0
1.3.0	3.6, 4.0	0.4.0
1.2.0	3.6, 4.0	0.4.0
1.1.0	3.6, 4.0	0.4.0

Platforms:

Operator	GKE	EKS	Openshift	AKS	Minikube
1.15.0	1.24 - 1.28	1.24 - 1.28	4.11 - 4.13	1.25 - 1.28	1.31.2
1.14.0	1.22 - 1.25	1.22 - 1.24	4.10 - 4.12	1.23 - 1.25	1.29
1.13.0	1.21 - 1.23	1.21 - 1.23	4.10 - 4.11	1.22 - 1.24	1.26
1.12.0	1.19 - 1.22	1.19 - 1.22	4.7 - 4.10	-	1.23
1.11.0	1.19 - 1.22	1.18 - 1.22	4.7 - 4.9	-	1.22
1.10.0	1.17 - 1.21	1.16 - 1.21	4.6 - 4.8	-	1.22
1.9.0	1.17 - 1.21	1.16-1.20	4.7	-	1.20
1.8.0	1.16 - 1.20	1.19	3.11, 4.7	-	1.19
1.7.0	1.15 - 1.17	1.15	3.11, 4.5	-	1.10
1.6.0	1.15 - 1.17	1.15	3.11, 4.5	-	1.10
1.5.0	1.15 - 1.17	1.15	3.11, 4.5	-	1.18
1.4.0	1.13, 1.15	1.15	3.11, 4.2	-	1.16
1.3.0	1.11, 1.14	-	3.11, 4.1	-	1.12
1.2.0	-	-	3.11, 4.0	-	-
1.1.0	-	-	3.11, 4.0	-	-

More detailed information about the cluster components for the current version of the Operator can be found in the system requirements and in the list of certified images. For previous releases of the Operator, you can check the same pages in the documentation archive.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

16.4 Percona Operator for MongoDB API Documentation

Percona Operator for MongoDB provides an aggregation-layer extension for the Kubernetes API. Please refer to the official Kubernetes API documentation on the API access and usage details. The following subsections describe the Percona XtraDB Cluster API provided by the Operator.

16.4.1 Prerequisites

1. Create the namespace name you will use, if not exist:

```
$ kubectl create namespace my-namespace-name
```

Trying to create an already-existing namespace will show you a self-explanatory error message. Also, you can use the `defalut` namespace.

Note

In this document `default` namespace is used in all examples. Substitute `default` with your namespace name if you use a different one.

2. Prepare:

```
# set correct API address
KUBE_CLUSTER=$(kubectl config view --minify -o jsonpath='{.clusters[0].name}')
API_SERVER=$(kubectl config view -o jsonpath="{.clusters[?
(@.name==\"$KUBE_CLUSTER\")].cluster.server}" | sed -e 's#https://##')

# create service account and get token
kubectl apply --server-side -f deploy/crd.yaml -f deploy/rbac.yaml -n default
KUBE_TOKEN=$(kubectl get secret $(kubectl get serviceaccount percona-server-mongodb-
operator -o jsonpath='{.secrets[0].name}' -n default) -o jsonpath='{.data.token}' -n
default | base64 --decode )
```

16.4.2 Create new Percona Server for MongoDB cluster

Description:

```
The command to create a new Percona Server for MongoDB cluster
```

Kubectl Command:

```
$ kubectl apply -f percona-server-mongodb-operator/deploy/cr.yaml
```

URL:

```
https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XPOST "https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/
perconaservermongodbs" \
            -H "Content-Type: application/json" \
            -H "Accept: application/json" \
            -H "Authorization: Bearer $KUBE_TOKEN" \
            -d "@cluster.json"
```

Request Body (cluster.json):

Example

```
{
   "apiVersion": "psmdb.percona.com/v1-5-0",
   "kind": "PerconaServerMongoDB",
   "metadata": {
      "name": "my-cluster-name"
   },
   "spec": {
      "image": "percona/percona-server-mongodb:4.2.8-8",
      "imagePullPolicy": "Always",
      "allowUnsafeConfigurations": false,
      "updateStrategy": "SmartUpdate",
      "secrets": {
         "users": "my-cluster-name-secrets"
      },
      "pmm": {
         "enabled": false,
         "image": "percona/percona-server-mongodb-operator:1.5.0-pmm",
         "serverHost": "monitoring-service"
      },
      "replsets": [
         {
            "name": "rs0",
            "size": 3,
            "affinity": {
               "antiAffinityTopologyKey": "none"
            },
            "podDisruptionBudget": {
               "maxUnavailable": 1
            },
            "expose": {
               "enabled": false,
               "exposeType": "LoadBalancer"
            },
            "arbiter": {
               "enabled": false,
               "size": 1,
               "affinity": {
                  "antiAffinityTopologyKey": "none"
               }
            },
            "resources": {
               "limits": null
            },
            "volumeSpec": {
               "persistentVolumeClaim": {
                  "storageClassName": "standard",
                  "accessModes": [
                     "ReadWriteOnce"
                  ],
                  "resources": {
                     "requests": {
                        "storage": "3Gi"
                     }
                  }
               }
            }
         }
      ],
      "mongod": {
         "net": {
            "port": 27017,
            "hostPort": 0
         },
         "security": {
            "redactClientLogData": false,
```

```
              "enableEncryption": true,
              "encryptionKeySecret": "my-cluster-name-mongodb-encryption-key",
              "encryptionCipherMode": "AES256-CBC"
           },
           "setParameter": {
              "ttlMonitorSleepSecs": 60,
              "wiredTigerConcurrentReadTransactions": 128,
              "wiredTigerConcurrentWriteTransactions": 128
           },
           "storage": {
              "engine": "wiredTiger",
              "inMemory": {
                 "engineConfig": {
                    "inMemorySizeRatio": 0.9
                 }
              },
              "mmapv1": {
                 "nsSize": 16,
                 "smallfiles": false
              },
              "wiredTiger": {
                 "engineConfig": {
                    "cacheSizeRatio": 0.5,
                    "directoryForIndexes": false,
                    "journalCompressor": "snappy"
                 },
                 "collectionConfig": {
                    "blockCompressor": "snappy"
                 },
                 "indexConfig": {
                    "prefixCompression": true
                 }
              }
           },
           "operationProfiling": {
              "mode": "slowOp",
              "slowOpThresholdMs": 100,
              "rateLimit": 100
           }
        },
        "backup": {
           "enabled": true,
           "restartOnFailure": true,
           "image": "percona/percona-server-mongodb-operator:1.5.0-backup",
           "serviceAccountName": "percona-server-mongodb-operator",
           "storages": null,
           "tasks": null
        }
     }
  }
```

Inputs:

Metadata:

1. Name (String, min-length: 1) : `contains name of cluster`

Spec:

1. secrets[users] (String, min-length: 1) : `contains name of secret for the users`
2. allowUnsafeConfigurations (Boolean, Default: false) : `allow unsafe configurations to run`
3. image (String, min-length: 1) : `name of the Percona Server for MongoDB cluster image`

replsets:

1. name (String, min-length: 1) : `name of monogo replicaset`
2. size (Integer, min-value: 1) : `contains size of MongoDB replicaset`
3. expose[exposeType] (Integer, min-value: 1) : `type of service to expose replicaset`
4. arbiter (Object) : `configuration for mongo arbiter`

mongod:

1. net:
    a. port (Integer, min-value: 0) : `contains mongod container port`
    b. hostPort (Integer, min-value: 0) : `host port to expose mongod on`
2. security:
    a. enableEncryption (Boolean, Default: true) : `enable encrypting mongod storage`
    b. encryptionKeySecret (String, min-length: 1) : `name of encryption key secret`
    c. encryptionCipherMode (String, min-length: 1) : `type of encryption cipher to use`
3. setParameter (Object): `configure mongod enginer paramters`
4. storage:
    a. engine (String, min-length: 1, default “wiredTiger”): `name of mongod storage engine`
    b. inMemory (Object) : `wiredTiger engine configuration`
    c. wiredTiger (Object) : `wiredTiger engine configuration`

pmm:

1. serverHost (String, min-length: 1) : `serivce name for monitoring`
2. image (String, min-length: 1) : `name of pmm image`

backup:

1. image (String, min-length: 1) : `name of MngoDB backup docker image`
2. serviceAccountName (String, min-length: 1) `name of service account to use for backup`
3. storages (Object) : `storage configuration object for backup`

Response:

Example


```
{
   "apiVersion":"psmdb.percona.com/v1-5-0",
   "kind":"PerconaServerMongoDB",
   "metadata":{
      "annotations":{
         "kubectl.kubernetes.io/last-applied-configuration":"{\"apiVersion\":\"psmdb.percona.com/
v1-5-0\",\"kind\":\"PerconaServerMongoDB\",\"metadata\":{\"annotations\":{},\"name\":\"my-
cluster-name\",\"namespace\":\"default\"},\"spec\":{\"allowUnsafeConfigurations\":false,
\"backup\":{\"enabled\":true,\"image\":\"percona/percona-server-mongodb-operator:1.5.0-backup\",
\"restartOnFailure\":true,\"serviceAccountName\":\"percona-server-mongodb-operator\",
\"storages\":null,\"tasks\":null},\"image\":\"percona/percona-server-mongodb:4.2.8-8\",
\"imagePullPolicy\":\"Always\",\"mongod\":{\"net\":{\"hostPort\":0,\"port\":27017},
\"operationProfiling\":{\"mode\":\"slowOp\",\"rateLimit\":100,\"slowOpThresholdMs\":100},
\"security\":{\"enableEncryption\":true,\"encryptionCipherMode\":\"AES256-CBC\",
\"encryptionKeySecret\":\"my-cluster-name-mongodb-encryption-key\",
\"redactClientLogData\":false},\"setParameter\":{\"ttlMonitorSleepSecs\":
60,\"wiredTigerConcurrentReadTransactions\":128,\"wiredTigerConcurrentWriteTransactions\":128},
\"storage\":{\"engine\":\"wiredTiger\",\"inMemory\":{\"engineConfig\":{\"inMemorySizeRatio\":
0.9}},\"mmapv1\":{\"nsSize\":16,\"smallfiles\":false},\"wiredTiger\":{\"collectionConfig\":
{\"blockCompressor\":\"snappy\"},\"engineConfig\":{\"cacheSizeRatio\":
0.5,\"directoryForIndexes\":false,\"journalCompressor\":\"snappy\"},\"indexConfig\":
{\"prefixCompression\":true}}}},\"pmm\":{\"enabled\":false,\"image\":\"percona/percona-server-
mongodb-operator:1.5.0-pmm\",\"serverHost\":\"monitoring-service\"},\"replsets\":[{\"affinity\":
{\"antiAffinityTopologyKey\":\"none\"},\"arbiter\":{\"affinity\":{\"antiAffinityTopologyKey\":
\"none\"},\"enabled\":false,\"size\":1},\"expose\":{\"enabled\":false,\"exposeType\":
\"LoadBalancer\"},\"name\":\"rs0\",\"podDisruptionBudget\":{\"maxUnavailable\":1},\"resources\":
{\"limits\":null},\"size\":3,\"volumeSpec\":{\"persistentVolumeClaim\":{\"accessModes\":
[\"ReadWriteOnce\"],\"resources\":{\"requests\":{\"storage\":\"3Gi\"}},\"storageClassName\":
\"standard\"}}}],\"secrets\":{\"users\":\"my-cluster-name-secrets\"},\"updateStrategy\":
\"SmartUpdate\"}}\n"
      },
      "creationTimestamp":"2020-07-24T14:27:58Z",
      "generation":1,
      "managedFields":[
         {
            "apiVersion":"psmdb.percona.com/v1-5-0",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:metadata":{
                  "f:annotations":{
                     ".":{

                     },
                     "f:kubectl.kubernetes.io/last-applied-configuration":{

                     }
                  }
               },
               "f:spec":{
                  ".":{

                  },
                  "f:allowUnsafeConfigurations":{

                  },
                  "f:backup":{
                     ".":{

                     },
                     "f:enabled":{

                     },
                     "f:image":{

                     },
```

```
            "f:restartOnFailure":{

            },
            "f:serviceAccountName":{

            },
            "f:storages":{

            },
            "f:tasks":{

            }
         },
         "f:image":{

         },
         "f:imagePullPolicy":{

         },
         "f:mongod":{
            ".":{

            },
            "f:net":{
               ".":{

               },
               "f:hostPort":{

               },
               "f:port":{

               }
            },
            "f:operationProfiling":{
               ".":{

               },
               "f:mode":{

               },
               "f:rateLimit":{

               },
               "f:slowOpThresholdMs":{

               }
            },
            "f:security":{
               ".":{

               },
               "f:enableEncryption":{

               },
               "f:encryptionCipherMode":{

               },
               "f:encryptionKeySecret":{

               },
               "f:redactClientLogData":{

               }
            },
            "f:setParameter":{
               ".":{

               },
```

```
      "f:ttlMonitorSleepSecs":{

      },
      "f:wiredTigerConcurrentReadTransactions":{

      },
      "f:wiredTigerConcurrentWriteTransactions":{

      }
   },
   "f:storage":{
      ".":{

      },
      "f:engine":{

      },
      "f:inMemory":{
         ".":{

         },
         "f:engineConfig":{
            ".":{

            },
            "f:inMemorySizeRatio":{

            }
         }
      },
      "f:mmapv1":{
         ".":{

         },
         "f:nsSize":{

         },
         "f:smallfiles":{

         }
      },
      "f:wiredTiger":{
         ".":{

         },
         "f:collectionConfig":{
            ".":{

            },
            "f:blockCompressor":{

            }
         },
         "f:engineConfig":{
            ".":{

            },
            "f:cacheSizeRatio":{

            },
            "f:directoryForIndexes":{

            },
            "f:journalCompressor":{

            }
         },
         "f:indexConfig":{
            ".":{
```

```
                                },
                                "f:prefixCompression":{

                                }
                             }
                          }
                       }
                    },
                    "f:pmm":{
                       ".":{

                       },
                       "f:enabled":{

                       },
                       "f:image":{

                       },
                       "f:serverHost":{

                       }
                    },
                    "f:replsets":{

                    },
                    "f:secrets":{
                       ".":{

                       },
                       "f:users":{

                       }
                    },
                    "f:updateStrategy":{

                    }
                 }
              },
              "manager":"kubectl",
              "operation":"Update",
              "time":"2020-07-24T14:27:58Z"
           }
        ],
        "name":"my-cluster-name",
        "namespace":"default",
        "resourceVersion":"1268922",
        "selfLink":"/apis/psmdb.percona.com/v1-5-0/namespaces/default/perconaservermongodbs/my-
cluster-name",
        "uid":"5207e71a-c83f-4707-b892-63aa93fb615c"
     },
     "spec":{
        "allowUnsafeConfigurations":false,
        "backup":{
           "enabled":true,
           "image":"percona/percona-server-mongodb-operator:1.5.0-backup",
           "restartOnFailure":true,
           "serviceAccountName":"percona-server-mongodb-operator",
           "storages":null,
           "tasks":null
        },
        "image":"percona/percona-server-mongodb:4.2.8-8",
        "imagePullPolicy":"Always",
        "mongod":{
           "net":{
              "hostPort":0,
              "port":27017
           },
           "operationProfiling":{
```

```
            "mode":"slowOp",
            "rateLimit":100,
            "slowOpThresholdMs":100
         },
         "security":{
            "enableEncryption":true,
            "encryptionCipherMode":"AES256-CBC",
            "encryptionKeySecret":"my-cluster-name-mongodb-encryption-key",
            "redactClientLogData":false
         },
         "setParameter":{
            "ttlMonitorSleepSecs":60,
            "wiredTigerConcurrentReadTransactions":128,
            "wiredTigerConcurrentWriteTransactions":128
         },
         "storage":{
            "engine":"wiredTiger",
            "inMemory":{
               "engineConfig":{
                  "inMemorySizeRatio":0.9
               }
            },
            "mmapv1":{
               "nsSize":16,
               "smallfiles":false
            },
            "wiredTiger":{
               "collectionConfig":{
                  "blockCompressor":"snappy"
               },
               "engineConfig":{
                  "cacheSizeRatio":0.5,
                  "directoryForIndexes":false,
                  "journalCompressor":"snappy"
               },
               "indexConfig":{
                  "prefixCompression":true
               }
            }
         }
      },
      "pmm":{
         "enabled":false,
         "image":"percona/percona-server-mongodb-operator:1.5.0-pmm",
         "serverHost":"monitoring-service"
      },
      "replsets":[
         {
            "affinity":{
               "antiAffinityTopologyKey":"none"
            },
            "arbiter":{
               "affinity":{
                  "antiAffinityTopologyKey":"none"
               },
               "enabled":false,
               "size":1
            },
            "expose":{
               "enabled":false,
               "exposeType":"LoadBalancer"
            },
            "name":"rs0",
            "podDisruptionBudget":{
               "maxUnavailable":1
            },
            "resources":{
               "limits":null
            },
```

```
            "size":3,
            "volumeSpec":{
               "persistentVolumeClaim":{
                  "accessModes":[
                     "ReadWriteOnce"
                  ],
                  "resources":{
                     "requests":{
                        "storage":"3Gi"
                     }
                  },
                  "storageClassName":"standard"
               }
            }
         }
      ],
      "secrets":{
         "users":"my-cluster-name-secrets"
      },
      "updateStrategy":"SmartUpdate"
   }
}
```


16.4.3 List Percona Server for MongoDB clusters

Description:

```
Lists all Percona Server for MongoDB clusters that exist in your kubernetes cluster
```

Kubectl Command:

```
$ kubectl get psmdb
```

URL:

```
https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs?
limit=500
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XGET "https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/
perconaservermongodbs?limit=500" \
            -H "Accept: application/json;as=Table;v=v1;g=meta.k8s.io,application/
json;as=Table;v=v1beta1;g=meta.k8s.io,application/json" \
            -H "Authorization: Bearer $KUBE_TOKEN"
```

Request Body:

```
None
```

Response:

Example


```
{
   "kind":"Table",
   "apiVersion":"meta.k8s.io/v1",
   "metadata":{
      "selfLink":"/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs",
      "resourceVersion":"1273793"
   },
   "columnDefinitions":[
      {
         "name":"Name",
         "type":"string",
         "format":"name",
         "description":"Name must be unique within a namespace. Is required when creating
resources, although some resources may allow a client to request the generation of an
appropriate name automatically. Name is primarily intended for creation idempotence and
configuration definition. Cannot be updated. More info: http://kubernetes.io/docs/user-guide/
identifiers#names",
         "priority":0
      },
      {
         "name":"Status",
         "type":"string",
         "format":"",
         "description":"Custom resource definition column (in JSONPath format): .status.state",
         "priority":0
      },
      {
         "name":"Age",
         "type":"date",
         "format":"",
         "description":"Custom resource definition column (in JSONPath
format): .metadata.creationTimestamp",
         "priority":0
      }
   ],
   "rows":[
      {
         "cells":[
            "my-cluster-name",
            "ready",
            "37m"
         ],
         "object":{
            "kind":"PartialObjectMetadata",
            "apiVersion":"meta.k8s.io/v1",
            "metadata":{
               "name":"my-cluster-name",
               "namespace":"default",
               "selfLink":"/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs/
my-cluster-name",
               "uid":"5207e71a-c83f-4707-b892-63aa93fb615c",
               "resourceVersion":"1273788",
               "generation":1,
               "creationTimestamp":"2020-07-24T14:27:58Z",
               "annotations":{
                  "kubectl.kubernetes.io/last-applied-configuration":"{\"apiVersion\":
\"psmdb.percona.com/v1-5-0\",\"kind\":\"PerconaServerMongoDB\",\"metadata\":{\"annotations\":{},
\"name\":\"my-cluster-name\",\"namespace\":\"default\"},\"spec\":
{\"allowUnsafeConfigurations\":false,\"backup\":{\"enabled\":true,\"image\":\"percona/percona-
server-mongodb-operator:1.5.0-backup\",\"restartOnFailure\":true,\"serviceAccountName\":
\"percona-server-mongodb-operator\",\"storages\":null,\"tasks\":null},\"image\":\"percona/
percona-server-mongodb:4.2.8-8\",\"imagePullPolicy\":\"Always\",\"mongod\":{\"net\":
{\"hostPort\":0,\"port\":27017},\"operationProfiling\":{\"mode\":\"slowOp\",\"rateLimit\":
100,\"slowOpThresholdMs\":100},\"security\":{\"enableEncryption\":true,\"encryptionCipherMode\":
\"AES256-CBC\",\"encryptionKeySecret\":\"my-cluster-name-mongodb-encryption-key\",
```

```
\"redactClientLogData\":false},\"setParameter\":{\"ttlMonitorSleepSecs\":
60,\"wiredTigerConcurrentReadTransactions\":128,\"wiredTigerConcurrentWriteTransactions\":128},
\"storage\":{\"engine\":\"wiredTiger\",\"inMemory\":{\"engineConfig\":{\"inMemorySizeRatio\":
0.9}},\"mmapv1\":{\"nsSize\":16,\"smallfiles\":false},\"wiredTiger\":{\"collectionConfig\":
{\"blockCompressor\":\"snappy\"},\"engineConfig\":{\"cacheSizeRatio\":
0.5,\"directoryForIndexes\":false,\"journalCompressor\":\"snappy\"},\"indexConfig\":
{\"prefixCompression\":true}}}},\"pmm\":{\"enabled\":false,\"image\":\"percona/percona-server-
mongodb-operator:1.5.0-pmm\",\"serverHost\":\"monitoring-service\"},\"replsets\":[{\"affinity\":
{\"antiAffinityTopologyKey\":\"none\"},\"arbiter\":{\"affinity\":{\"antiAffinityTopologyKey\":
\"none\"},\"enabled\":false,\"size\":1},\"expose\":{\"enabled\":false,\"exposeType\":
\"LoadBalancer\"},\"name\":\"rs0\",\"podDisruptionBudget\":{\"maxUnavailable\":1},\"resources\":
{\"limits\":null},\"size\":3,\"volumeSpec\":{\"persistentVolumeClaim\":{\"accessModes\":
[\"ReadWriteOnce\"],\"resources\":{\"requests\":{\"storage\":\"3Gi\"}},\"storageClassName\":
\"standard\"}}}],\"secrets\":{\"users\":\"my-cluster-name-secrets\"},\"updateStrategy\":
\"SmartUpdate\"}}\n"
               },
               "managedFields":[
                  {
                     "manager":"kubectl",
                     "operation":"Update",
                     "apiVersion":"psmdb.percona.com/v1-5-0",
                     "time":"2020-07-24T14:27:58Z",
                     "fieldsType":"FieldsV1",
                     "fieldsV1":{
                        "f:metadata":{
                           "f:annotations":{
                              ".":{

                              },
                              "f:kubectl.kubernetes.io/last-applied-configuration":{

                              }
                           }
                        },
                        "f:spec":{
                           ".":{

                           },
                           "f:allowUnsafeConfigurations":{

                           },
                           "f:backup":{
                              ".":{

                              },
                              "f:enabled":{

                              },
                              "f:image":{

                              },
                              "f:serviceAccountName":{

                              }
                           },
                           "f:image":{

                           },
                           "f:imagePullPolicy":{

                           },
                           "f:mongod":{
                              ".":{

                              },
                              "f:net":{
                                 ".":{

                                 },
```

```
                                    "f:port":{

                                    }
                                 },
                                 "f:operationProfiling":{
                                    ".":{

                                    },
                                    "f:mode":{

                                    },
                                    "f:rateLimit":{

                                    },
                                    "f:slowOpThresholdMs":{

                                    }
                                 },
                                 "f:security":{
                                    ".":{

                                    },
                                    "f:enableEncryption":{

                                    },
                                    "f:encryptionCipherMode":{

                                    },
                                    "f:encryptionKeySecret":{

                                    }
                                 },
                                 "f:setParameter":{
                                    ".":{

                                    },
                                    "f:ttlMonitorSleepSecs":{

                                    },
                                    "f:wiredTigerConcurrentReadTransactions":{

                                    },
                                    "f:wiredTigerConcurrentWriteTransactions":{

                                    }
                                 },
                                 "f:storage":{
                                    ".":{

                                    },
                                    "f:engine":{

                                    },
                                    "f:inMemory":{
                                       ".":{

                                       },
                                       "f:engineConfig":{
                                          ".":{

                                          },
                                          "f:inMemorySizeRatio":{

                                          }
                                       }
                                    },
                                    "f:mmapv1":{
                                       ".":{
```

```
                },
                "f:nsSize":{

                }
             },
             "f:wiredTiger":{
                ".":{

                },
                "f:collectionConfig":{
                   ".":{

                   },
                   "f:blockCompressor":{

                   }
                },
                "f:engineConfig":{
                   ".":{

                   },
                   "f:cacheSizeRatio":{

                   },
                   "f:journalCompressor":{

                   }
                },
                "f:indexConfig":{
                   ".":{

                   },
                   "f:prefixCompression":{

                   }
                }
             }
          }
       },
       "f:pmm":{
          ".":{

          },
          "f:image":{

          },
          "f:serverHost":{

          }
       },
       "f:secrets":{
          ".":{

          },
          "f:users":{

          }
       },
       "f:updateStrategy":{

       }
    }
 }
},
{
   "manager":"percona-server-mongodb-operator",
   "operation":"Update",
   "apiVersion":"psmdb.percona.com/v1",
   "time":"2020-07-24T15:04:55Z",
```



```
                  "fieldsType":"FieldsV1",
                  "fieldsV1":{
                     "f:spec":{
                        "f:backup":{
                           "f:containerSecurityContext":{
                              ".":{

                              },
                              "f:runAsNonRoot":{

                              },
                              "f:runAsUser":{

                              }
                           },
                           "f:podSecurityContext":{
                              ".":{

                              },
                              "f:fsGroup":{

                              }
                           }
                        },
                        "f:clusterServiceDNSSuffix":{

                        },
                        "f:replsets":{

                        },
                        "f:runUid":{

                        },
                        "f:secrets":{
                           "f:ssl":{

                           },
                           "f:sslInternal":{

                           }
                        }
                     },
                     "f:status":{
                        ".":{

                        },
                        "f:conditions":{

                        },
                        "f:observedGeneration":{

                        },
                        "f:replsets":{
                           ".":{

                           },
                           "f:rs0":{
                              ".":{

                              },
                              "f:ready":{

                              },
                              "f:size":{

                              },
                              "f:status":{

                              }
```

```
                              }
                           },
                           "f:state":{

                           }
                        }
                     }
                  }
               ]
            }
         }
      }
   ]
}
```


16.4.4 Get status of Percona Server for MongoDB cluster

Description:

```
Gets all information about specified Percona Server for MongoDB cluster
```

Kubectl Command:

```
$ kubectl get psmdb/my-cluster-name -o json
```

URL:

```
https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs/my-
cluster-name
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XGET "https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/
perconaservermongodbs/my-cluster-name" \
            -H "Accept: application/json" \
            -H "Authorization: Bearer $KUBE_TOKEN"
```

Request Body:

```
None
```

Response:

Example


```
{
   "apiVersion":"psmdb.percona.com/v1",
   "kind":"PerconaServerMongoDB",
   "metadata":{
      "annotations":{
         "kubectl.kubernetes.io/last-applied-configuration":"{\"apiVersion\":\"psmdb.percona.com/
v1-5-0\",\"kind\":\"PerconaServerMongoDB\",\"metadata\":{\"annotations\":{},\"name\":\"my-
cluster-name\",\"namespace\":\"default\"},\"spec\":{\"allowUnsafeConfigurations\":false,
\"backup\":{\"enabled\":true,\"image\":\"percona/percona-server-mongodb-operator:1.5.0-backup\",
\"restartOnFailure\":true,\"serviceAccountName\":\"percona-server-mongodb-operator\",
\"storages\":null,\"tasks\":null},\"image\":\"percona/percona-server-mongodb:4.2.8-8\",
\"imagePullPolicy\":\"Always\",\"mongod\":{\"net\":{\"hostPort\":0,\"port\":27017},
\"operationProfiling\":{\"mode\":\"slowOp\",\"rateLimit\":100,\"slowOpThresholdMs\":100},
\"security\":{\"enableEncryption\":true,\"encryptionCipherMode\":\"AES256-CBC\",
\"encryptionKeySecret\":\"my-cluster-name-mongodb-encryption-key\",
\"redactClientLogData\":false},\"setParameter\":{\"ttlMonitorSleepSecs\":
60,\"wiredTigerConcurrentReadTransactions\":128,\"wiredTigerConcurrentWriteTransactions\":128},
\"storage\":{\"engine\":\"wiredTiger\",\"inMemory\":{\"engineConfig\":{\"inMemorySizeRatio\":
0.9}},\"mmapv1\":{\"nsSize\":16,\"smallfiles\":false},\"wiredTiger\":{\"collectionConfig\":
{\"blockCompressor\":\"snappy\"},\"engineConfig\":{\"cacheSizeRatio\":
0.5,\"directoryForIndexes\":false,\"journalCompressor\":\"snappy\"},\"indexConfig\":
{\"prefixCompression\":true}}}},\"pmm\":{\"enabled\":false,\"image\":\"percona/percona-server-
mongodb-operator:1.5.0-pmm\",\"serverHost\":\"monitoring-service\"},\"replsets\":[{\"affinity\":
{\"antiAffinityTopologyKey\":\"none\"},\"arbiter\":{\"affinity\":{\"antiAffinityTopologyKey\":
\"none\"},\"enabled\":false,\"size\":1},\"expose\":{\"enabled\":false,\"exposeType\":
\"LoadBalancer\"},\"name\":\"rs0\",\"podDisruptionBudget\":{\"maxUnavailable\":1},\"resources\":
{\"limits\":null},\"size\":3,\"volumeSpec\":{\"persistentVolumeClaim\":{\"accessModes\":
[\"ReadWriteOnce\"],\"resources\":{\"requests\":{\"storage\":\"3Gi\"}},\"storageClassName\":
\"standard\"}}}],\"secrets\":{\"users\":\"my-cluster-name-secrets\"},\"updateStrategy\":
\"SmartUpdate\"}}\n"
      },
      "creationTimestamp":"2020-07-24T14:27:58Z",
      "generation":1,
      "managedFields":[
         {
            "apiVersion":"psmdb.percona.com/v1-5-0",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:metadata":{
                  "f:annotations":{
                     ".":{

                     },
                     "f:kubectl.kubernetes.io/last-applied-configuration":{

                     }
                  }
               },
               "f:spec":{
                  ".":{

                  },
                  "f:allowUnsafeConfigurations":{

                  },
                  "f:backup":{
                     ".":{

                     },
                     "f:enabled":{

                     },
                     "f:image":{

                     },
```

```
            "f:serviceAccountName":{

            }
         },
         "f:image":{

         },
         "f:imagePullPolicy":{

         },
         "f:mongod":{
            ".":{

            },
            "f:net":{
               ".":{

               },
               "f:port":{

               }
            },
            "f:operationProfiling":{
               ".":{

               },
               "f:mode":{

               },
               "f:rateLimit":{

               },
               "f:slowOpThresholdMs":{

               }
            },
            "f:security":{
               ".":{

               },
               "f:enableEncryption":{

               },
               "f:encryptionCipherMode":{

               },
               "f:encryptionKeySecret":{

               }
            },
            "f:setParameter":{
               ".":{

               },
               "f:ttlMonitorSleepSecs":{

               },
               "f:wiredTigerConcurrentReadTransactions":{

               },
               "f:wiredTigerConcurrentWriteTransactions":{

               }
            },
            "f:storage":{
               ".":{

               },
               "f:engine":{
```

```
                        },
                        "f:inMemory":{
                           ".":{

                           },
                           "f:engineConfig":{
                              ".":{

                              },
                              "f:inMemorySizeRatio":{

                              }
                           }
                        },
                        "f:mmapv1":{
                           ".":{

                           },
                           "f:nsSize":{

                           }
                        },
                        "f:wiredTiger":{
                           ".":{

                           },
                           "f:collectionConfig":{
                              ".":{

                              },
                              "f:blockCompressor":{

                              }
                           },
                           "f:engineConfig":{
                              ".":{

                              },
                              "f:cacheSizeRatio":{

                              },
                              "f:journalCompressor":{

                              }
                           },
                           "f:indexConfig":{
                              ".":{

                              },
                              "f:prefixCompression":{

                              }
                           }
                        }
                     }
                  },
                  "f:pmm":{
                     ".":{

                     },
                     "f:image":{

                     },
                     "f:serverHost":{

                     }
                  },
                  "f:secrets":{
```

```
                  ".":{

                  },
                  "f:users":{

                  }
               },
               "f:updateStrategy":{

               }
            }
         },
         "manager":"kubectl",
         "operation":"Update",
         "time":"2020-07-24T14:27:58Z"
      },
      {
         "apiVersion":"psmdb.percona.com/v1",
         "fieldsType":"FieldsV1",
         "fieldsV1":{
            "f:spec":{
               "f:backup":{
                  "f:containerSecurityContext":{
                     ".":{

                     },
                     "f:runAsNonRoot":{

                     },
                     "f:runAsUser":{

                     }
                  },
                  "f:podSecurityContext":{
                     ".":{

                     },
                     "f:fsGroup":{

                     }
                  }
               },
               "f:clusterServiceDNSSuffix":{

               },
               "f:replsets":{

               },
               "f:runUid":{

               },
               "f:secrets":{
                  "f:ssl":{

                  },
                  "f:sslInternal":{

                  }
               }
            },
            "f:status":{
               ".":{

               },
               "f:conditions":{

               },
               "f:observedGeneration":{
```

```
                  },
                  "f:replsets":{
                     ".":{

                     },
                     "f:rs0":{
                        ".":{

                        },
                        "f:ready":{

                        },
                        "f:size":{

                        },
                        "f:status":{

                        }
                     }
                  },
                  "f:state":{

                  }
               }
            },
            "manager":"percona-server-mongodb-operator",
            "operation":"Update",
            "time":"2020-07-24T15:09:40Z"
         }
      ],
      "name":"my-cluster-name",
      "namespace":"default",
      "resourceVersion":"1274523",
      "selfLink":"/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs/my-cluster-
name",
      "uid":"5207e71a-c83f-4707-b892-63aa93fb615c"
   },
   "spec":{
      "allowUnsafeConfigurations":false,
      "backup":{
         "enabled":true,
         "image":"percona/percona-server-mongodb-operator:1.5.0-backup",
         "restartOnFailure":true,
         "serviceAccountName":"percona-server-mongodb-operator",
         "storages":null,
         "tasks":null
      },
      "image":"percona/percona-server-mongodb:4.2.8-8",
      "imagePullPolicy":"Always",
      "mongod":{
         "net":{
            "hostPort":0,
            "port":27017
         },
         "operationProfiling":{
            "mode":"slowOp",
            "rateLimit":100,
            "slowOpThresholdMs":100
         },
         "security":{
            "enableEncryption":true,
            "encryptionCipherMode":"AES256-CBC",
            "encryptionKeySecret":"my-cluster-name-mongodb-encryption-key",
            "redactClientLogData":false
         },
         "setParameter":{
            "ttlMonitorSleepSecs":60,
            "wiredTigerConcurrentReadTransactions":128,
            "wiredTigerConcurrentWriteTransactions":128
```

```
         },
         "storage":{
            "engine":"wiredTiger",
            "inMemory":{
               "engineConfig":{
                  "inMemorySizeRatio":0.9
               }
            },
            "mmapv1":{
               "nsSize":16,
               "smallfiles":false
            },
            "wiredTiger":{
               "collectionConfig":{
                  "blockCompressor":"snappy"
               },
               "engineConfig":{
                  "cacheSizeRatio":0.5,
                  "directoryForIndexes":false,
                  "journalCompressor":"snappy"
               },
               "indexConfig":{
                  "prefixCompression":true
               }
            }
         }
      },
      "pmm":{
         "enabled":false,
         "image":"percona/percona-server-mongodb-operator:1.5.0-pmm",
         "serverHost":"monitoring-service"
      },
      "replsets":[
         {
            "affinity":{
               "antiAffinityTopologyKey":"none"
            },
            "arbiter":{
               "affinity":{
                  "antiAffinityTopologyKey":"none"
               },
               "enabled":false,
               "size":1
            },
            "expose":{
               "enabled":false,
               "exposeType":"LoadBalancer"
            },
            "name":"rs0",
            "podDisruptionBudget":{
               "maxUnavailable":1
            },
            "resources":{
               "limits":null
            },
            "size":3,
            "volumeSpec":{
               "persistentVolumeClaim":{
                  "accessModes":[
                     "ReadWriteOnce"
                  ],
                  "resources":{
                     "requests":{
                        "storage":"3Gi"
                     }
                  },
                  "storageClassName":"standard"
               }
            }
```

```
         }
      ],
      "secrets":{
         "users":"my-cluster-name-secrets"
      },
      "updateStrategy":"SmartUpdate"
   },
   "status":{
      "conditions":[
         {
            "lastTransitionTime":"2020-07-24T14:28:03Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:28:39Z",
            "status":"True",
            "type":"Error"
         },
         {
            "lastTransitionTime":"2020-07-24T14:28:41Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:28:41Z",
            "status":"True",
            "type":"Error"
         },
         {
            "lastTransitionTime":"2020-07-24T14:29:10Z",
            "status":"True",
            "type":"ClusterReady"
         },
         {
            "lastTransitionTime":"2020-07-24T14:49:46Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:50:00Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:52:31Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:52:43Z",
            "status":"True",
            "type":"Error"
         },
         {
            "lastTransitionTime":"2020-07-24T14:53:01Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:53:05Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:53:05Z",
            "status":"True",
            "type":"ClusterReady"
         }
```

```
        ],
        "observedGeneration":1,
        "replsets":{
           "rs0":{
              "ready":3,
              "size":3,
              "status":"ready"
           }
        },
        "state":"ready"
     }
  }
```


16.4.5 Scale up/down Percona Server for MongoDB cluster

Description:

```
Increase or decrease the size of the Percona Server for MongoDB cluster nodes to fit the
current high availability needs
```

Kubectl Command:

```
$ kubectl patch psmdb my-cluster-name --type=merge --patch '{
"spec": {"replsets":{ "size": "5" }
}}'
```

URL:

```
https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs/my-
cluster-name
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XPATCH "https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/
perconaservermongodbs/my-cluster-name" \
            -H "Authorization: Bearer $KUBE_TOKEN" \
            -H "Content-Type: application/merge-patch+json"
            -H "Accept: application/json" \
            -d '{
                  "spec": {"replsets":{ "size": "5" }
                  }}'
```

Request Body:

Example

```
{
"spec": {"replsets":{ "size": "5" }
}}
```

Input:

spec:

replsets

1. size (Int or String, Defaults: 3): `Specifiy the sie of the replsets cluster to scale up or down to`

Response:

Example


```
{
   "apiVersion":"psmdb.percona.com/v1",
   "kind":"PerconaServerMongoDB",
   "metadata":{
      "annotations":{
         "kubectl.kubernetes.io/last-applied-configuration":"{\"apiVersion\":\"psmdb.percona.com/
v1-5-0\",\"kind\":\"PerconaServerMongoDB\",\"metadata\":{\"annotations\":{},\"name\":\"my-
cluster-name\",\"namespace\":\"default\"},\"spec\":{\"allowUnsafeConfigurations\":false,
\"backup\":{\"enabled\":true,\"image\":\"percona/percona-server-mongodb-operator:1.5.0-backup\",
\"restartOnFailure\":true,\"serviceAccountName\":\"percona-server-mongodb-operator\",
\"storages\":null,\"tasks\":null},\"image\":\"percona/percona-server-mongodb:4.2.8-8\",
\"imagePullPolicy\":\"Always\",\"mongod\":{\"net\":{\"hostPort\":0,\"port\":27017},
\"operationProfiling\":{\"mode\":\"slowOp\",\"rateLimit\":100,\"slowOpThresholdMs\":100},
\"security\":{\"enableEncryption\":true,\"encryptionCipherMode\":\"AES256-CBC\",
\"encryptionKeySecret\":\"my-cluster-name-mongodb-encryption-key\",
\"redactClientLogData\":false},\"setParameter\":{\"ttlMonitorSleepSecs\":
60,\"wiredTigerConcurrentReadTransactions\":128,\"wiredTigerConcurrentWriteTransactions\":128},
\"storage\":{\"engine\":\"wiredTiger\",\"inMemory\":{\"engineConfig\":{\"inMemorySizeRatio\":
0.9}},\"mmapv1\":{\"nsSize\":16,\"smallfiles\":false},\"wiredTiger\":{\"collectionConfig\":
{\"blockCompressor\":\"snappy\"},\"engineConfig\":{\"cacheSizeRatio\":
0.5,\"directoryForIndexes\":false,\"journalCompressor\":\"snappy\"},\"indexConfig\":
{\"prefixCompression\":true}}}},\"pmm\":{\"enabled\":false,\"image\":\"percona/percona-server-
mongodb-operator:1.5.0-pmm\",\"serverHost\":\"monitoring-service\"},\"replsets\":[{\"affinity\":
{\"antiAffinityTopologyKey\":\"none\"},\"arbiter\":{\"affinity\":{\"antiAffinityTopologyKey\":
\"none\"},\"enabled\":false,\"size\":1},\"expose\":{\"enabled\":false,\"exposeType\":
\"LoadBalancer\"},\"name\":\"rs0\",\"podDisruptionBudget\":{\"maxUnavailable\":1},\"resources\":
{\"limits\":null},\"size\":3,\"volumeSpec\":{\"persistentVolumeClaim\":{\"accessModes\":
[\"ReadWriteOnce\"],\"resources\":{\"requests\":{\"storage\":\"3Gi\"}},\"storageClassName\":
\"standard\"}}}],\"secrets\":{\"users\":\"my-cluster-name-secrets\"},\"updateStrategy\":
\"SmartUpdate\"}}\n"
      },
      "creationTimestamp":"2020-07-24T14:27:58Z",
      "generation":4,
      "managedFields":[
         {
            "apiVersion":"psmdb.percona.com/v1-5-0",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:metadata":{
                  "f:annotations":{
                     ".":{

                     },
                     "f:kubectl.kubernetes.io/last-applied-configuration":{

                     }
                  }
               },
               "f:spec":{
                  ".":{

                  },
                  "f:allowUnsafeConfigurations":{

                  },
                  "f:backup":{
                     ".":{

                     },
                     "f:enabled":{

                     },
                     "f:image":{

                     },
```

```
                     "f:serviceAccountName":{

                     }
                  },
                  "f:image":{

                  },
                  "f:imagePullPolicy":{

                  },
                  "f:mongod":{
                     ".":{

                     },
                     "f:net":{
                        ".":{

                        },
                        "f:port":{

                        }
                     },
                     "f:operationProfiling":{
                        ".":{

                        },
                        "f:mode":{

                        },
                        "f:rateLimit":{

                        },
                        "f:slowOpThresholdMs":{

                        }
                     },
                     "f:security":{
                        ".":{

                        },
                        "f:enableEncryption":{

                        },
                        "f:encryptionCipherMode":{

                        },
                        "f:encryptionKeySecret":{

                        }
                     },
                     "f:setParameter":{
                        ".":{

                        },
                        "f:ttlMonitorSleepSecs":{

                        },
                        "f:wiredTigerConcurrentReadTransactions":{

                        },
                        "f:wiredTigerConcurrentWriteTransactions":{

                        }
                     },
                     "f:storage":{
                        ".":{

                        },
                        "f:engine":{
```

```
                                    },
                                    "f:inMemory":{
                                       ".":{

                                       },
                                       "f:engineConfig":{
                                          ".":{

                                          },
                                          "f:inMemorySizeRatio":{

                                          }
                                       }
                                    },
                                    "f:mmapv1":{
                                       ".":{

                                       },
                                       "f:nsSize":{

                                       }
                                    },
                                    "f:wiredTiger":{
                                       ".":{

                                       },
                                       "f:collectionConfig":{
                                          ".":{

                                          },
                                          "f:blockCompressor":{

                                          }
                                       },
                                       "f:engineConfig":{
                                          ".":{

                                          },
                                          "f:cacheSizeRatio":{

                                          },
                                          "f:journalCompressor":{

                                          }
                                       },
                                       "f:indexConfig":{
                                          ".":{

                                          },
                                          "f:prefixCompression":{

                                          }
                                       }
                                    }
                                 }
                              },
                              "f:pmm":{
                                 ".":{

                                 },
                                 "f:image":{

                                 },
                                 "f:serverHost":{

                                 }
                              },
                              "f:secrets":{
```

```
            ".":{

            },
            "f:users":{

            }
         },
         "f:updateStrategy":{

         }
      }
   },
   "manager":"kubectl",
   "operation":"Update",
   "time":"2020-07-24T14:27:58Z"
},
{
   "apiVersion":"psmdb.percona.com/v1",
   "fieldsType":"FieldsV1",
   "fieldsV1":{
      "f:spec":{
         "f:backup":{
            "f:containerSecurityContext":{
               ".":{

               },
               "f:runAsNonRoot":{

               },
               "f:runAsUser":{

               }
            },
            "f:podSecurityContext":{
               ".":{

               },
               "f:fsGroup":{

               }
            }
         },
         "f:clusterServiceDNSSuffix":{

         },
         "f:runUid":{

         },
         "f:secrets":{
            "f:ssl":{

            },
            "f:sslInternal":{

            }
         }
      },
      "f:status":{
         ".":{

         },
         "f:conditions":{

         },
         "f:observedGeneration":{

         },
         "f:replsets":{
            ".":{
```

```
                     },
                     "f:rs0":{
                        ".":{

                        },
                        "f:ready":{

                        },
                        "f:size":{

                        },
                        "f:status":{

                        }
                     }
                  },
                  "f:state":{

                  }
               }
            },
            "manager":"percona-server-mongodb-operator",
            "operation":"Update",
            "time":"2020-07-24T15:35:14Z"
         },
         {
            "apiVersion":"psmdb.percona.com/v1",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:spec":{
                  "f:replsets":{
                     ".":{

                     },
                     "f:size":{

                     }
                  }
               }
            },
            "manager":"kubectl",
            "operation":"Update",
            "time":"2020-07-24T15:43:19Z"
         }
      ],
      "name":"my-cluster-name",
      "namespace":"default",
      "resourceVersion":"1279009",
      "selfLink":"/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs/my-cluster-
name",
      "uid":"5207e71a-c83f-4707-b892-63aa93fb615c"
   },
   "spec":{
      "allowUnsafeConfigurations":false,
      "backup":{
         "enabled":true,
         "image":"percona/percona-server-mongodb-operator:1.5.0-backup",
         "restartOnFailure":true,
         "serviceAccountName":"percona-server-mongodb-operator",
         "storages":null,
         "tasks":null
      },
      "image":"percona/percona-server-mongodb:4.2.8-8",
      "imagePullPolicy":"Always",
      "mongod":{
         "net":{
            "hostPort":0,
            "port":27017
```

```
         },
         "operationProfiling":{
            "mode":"slowOp",
            "rateLimit":100,
            "slowOpThresholdMs":100
         },
         "security":{
            "enableEncryption":true,
            "encryptionCipherMode":"AES256-CBC",
            "encryptionKeySecret":"my-cluster-name-mongodb-encryption-key",
            "redactClientLogData":false
         },
         "setParameter":{
            "ttlMonitorSleepSecs":60,
            "wiredTigerConcurrentReadTransactions":128,
            "wiredTigerConcurrentWriteTransactions":128
         },
         "storage":{
            "engine":"wiredTiger",
            "inMemory":{
               "engineConfig":{
                  "inMemorySizeRatio":0.9
               }
            },
            "mmapv1":{
               "nsSize":16,
               "smallfiles":false
            },
            "wiredTiger":{
               "collectionConfig":{
                  "blockCompressor":"snappy"
               },
               "engineConfig":{
                  "cacheSizeRatio":0.5,
                  "directoryForIndexes":false,
                  "journalCompressor":"snappy"
               },
               "indexConfig":{
                  "prefixCompression":true
               }
            }
         }
      },
      "pmm":{
         "enabled":false,
         "image":"percona/percona-server-mongodb-operator:1.5.0-pmm",
         "serverHost":"monitoring-service"
      },
      "replsets":{
         "size":"5"
      },
      "secrets":{
         "users":"my-cluster-name-secrets"
      },
      "updateStrategy":"SmartUpdate"
   },
   "status":{
      "conditions":[
         {
            "lastTransitionTime":"2020-07-24T14:28:03Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:28:39Z",
            "status":"True",
            "type":"Error"
         },
         {
```



```
            "lastTransitionTime":"2020-07-24T14:28:41Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:28:41Z",
            "status":"True",
            "type":"Error"
         },
         {
            "lastTransitionTime":"2020-07-24T14:29:10Z",
            "status":"True",
            "type":"ClusterReady"
         },
         {
            "lastTransitionTime":"2020-07-24T14:49:46Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:50:00Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:52:31Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:52:43Z",
            "status":"True",
            "type":"Error"
         },
         {
            "lastTransitionTime":"2020-07-24T14:53:01Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:53:05Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:53:05Z",
            "status":"True",
            "type":"ClusterReady"
         }
      ],
      "observedGeneration":1,
      "replsets":{
         "rs0":{
            "ready":3,
            "size":3,
            "status":"ready"
         }
      },
      "state":"ready"
   }
}
```


16.4.6 Update Percona Server for MongoDB cluster image

Description:

```
Change the image of Percona Server for MongoDB containers inside the cluster
```

Kubectl Command:

```
$ kubectl patch psmdb my-cluster-name --type=merge --patch '{
"spec": {"psmdb":{ "image": "percona/percona-server-mongodb-operator:1.4.0-mongod4.2" }
}}'
```

URL:

```
https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs/my-
cluster-name
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XPATCH "https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/
perconaservermongodbs/my-cluster-name" \
            -H "Authorization: Bearer $KUBE_TOKEN" \
            -H "Accept: application/json" \
            -H "Content-Type: application/merge-patch+json"
            -d '{
              "spec": {"psmdb":{ "image": "percona/percona-server-mongodb-operator:1.4.0-
mongod4.2" }
              }}'
```

Request Body:

> **Example**
>
> ```
> {
> "spec": { "image ": "percona/percona-server-mongodb:4.2.8-8" }
> }
> ```

Input:

spec:

psmdb:

1. image (String, min-length: 1) : `name of the image to update for Percona Server for MongoDB`

Response:

Example

```
{
   "apiVersion":"psmdb.percona.com/v1",
   "kind":"PerconaServerMongoDB",
   "metadata":{
      "annotations":{
         "kubectl.kubernetes.io/last-applied-configuration":"{\"apiVersion\":\"psmdb.percona.com/
v1-5-0\",\"kind\":\"PerconaServerMongoDB\",\"metadata\":{\"annotations\":{},\"name\":\"my-
cluster-name\",\"namespace\":\"default\"},\"spec\":{\"allowUnsafeConfigurations\":false,
\"backup\":{\"enabled\":true,\"image\":\"percona/percona-server-mongodb-operator:1.5.0-backup\",
\"restartOnFailure\":true,\"serviceAccountName\":\"percona-server-mongodb-operator\",
\"storages\":null,\"tasks\":null},\"image\":\"percona/percona-server-mongodb:4.2.8-8\",
\"imagePullPolicy\":\"Always\",\"mongod\":{\"net\":{\"hostPort\":0,\"port\":27017},
\"operationProfiling\":{\"mode\":\"slowOp\",\"rateLimit\":100,\"slowOpThresholdMs\":100},
\"security\":{\"enableEncryption\":true,\"encryptionCipherMode\":\"AES256-CBC\",
\"encryptionKeySecret\":\"my-cluster-name-mongodb-encryption-key\",
\"redactClientLogData\":false},\"setParameter\":{\"ttlMonitorSleepSecs\":
60,\"wiredTigerConcurrentReadTransactions\":128,\"wiredTigerConcurrentWriteTransactions\":128},
\"storage\":{\"engine\":\"wiredTiger\",\"inMemory\":{\"engineConfig\":{\"inMemorySizeRatio\":
0.9}},\"mmapv1\":{\"nsSize\":16,\"smallfiles\":false},\"wiredTiger\":{\"collectionConfig\":
{\"blockCompressor\":\"snappy\"},\"engineConfig\":{\"cacheSizeRatio\":
0.5,\"directoryForIndexes\":false,\"journalCompressor\":\"snappy\"},\"indexConfig\":
{\"prefixCompression\":true}}}},\"pmm\":{\"enabled\":false,\"image\":\"percona/percona-server-
mongodb-operator:1.5.0-pmm\",\"serverHost\":\"monitoring-service\"},\"replsets\":[{\"affinity\":
{\"antiAffinityTopologyKey\":\"none\"},\"arbiter\":{\"affinity\":{\"antiAffinityTopologyKey\":
\"none\"},\"enabled\":false,\"size\":1},\"expose\":{\"enabled\":false,\"exposeType\":
\"LoadBalancer\"},\"name\":\"rs0\",\"podDisruptionBudget\":{\"maxUnavailable\":1},\"resources\":
{\"limits\":null},\"size\":3,\"volumeSpec\":{\"persistentVolumeClaim\":{\"accessModes\":
[\"ReadWriteOnce\"],\"resources\":{\"requests\":{\"storage\":\"3Gi\"}},\"storageClassName\":
\"standard\"}}}],\"secrets\":{\"users\":\"my-cluster-name-secrets\"},\"updateStrategy\":
\"SmartUpdate\"}}\n"
      },
      "creationTimestamp":"2020-07-24T14:27:58Z",
      "generation":5,
      "managedFields":[
         {
            "apiVersion":"psmdb.percona.com/v1-5-0",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:metadata":{
                  "f:annotations":{
                     ".":{

                     },
                     "f:kubectl.kubernetes.io/last-applied-configuration":{

                     }
                  }
               },
               "f:spec":{
                  ".":{

                  },
                  "f:allowUnsafeConfigurations":{

                  },
                  "f:backup":{
                     ".":{

                     },
                     "f:enabled":{

                     },
                     "f:image":{

                     },
```

```
            "f:serviceAccountName":{

            }
         },
         "f:image":{

         },
         "f:imagePullPolicy":{

         },
         "f:mongod":{
            ".":{

            },
            "f:net":{
               ".":{

               },
               "f:port":{

               }
            },
            "f:operationProfiling":{
               ".":{

               },
               "f:mode":{

               },
               "f:rateLimit":{

               },
               "f:slowOpThresholdMs":{

               }
            },
            "f:security":{
               ".":{

               },
               "f:enableEncryption":{

               },
               "f:encryptionCipherMode":{

               },
               "f:encryptionKeySecret":{

               }
            },
            "f:setParameter":{
               ".":{

               },
               "f:ttlMonitorSleepSecs":{

               },
               "f:wiredTigerConcurrentReadTransactions":{

               },
               "f:wiredTigerConcurrentWriteTransactions":{

               }
            },
            "f:storage":{
               ".":{

               },
               "f:engine":{
```

```
      },
      "f:inMemory":{
         ".":{

         },
         "f:engineConfig":{
            ".":{

            },
            "f:inMemorySizeRatio":{

            }
         }
      },
      "f:mmapv1":{
         ".":{

         },
         "f:nsSize":{

         }
      },
      "f:wiredTiger":{
         ".":{

         },
         "f:collectionConfig":{
            ".":{

            },
            "f:blockCompressor":{

            }
         },
         "f:engineConfig":{
            ".":{

            },
            "f:cacheSizeRatio":{

            },
            "f:journalCompressor":{

            }
         },
         "f:indexConfig":{
            ".":{

            },
            "f:prefixCompression":{

            }
         }
      }
   }
},
"f:pmm":{
   ".":{

   },
   "f:image":{

   },
   "f:serverHost":{

   }
},
"f:secrets":{
```

```
                    ".":{

                    },
                    "f:users":{

                    }
                },
                "f:updateStrategy":{

                }
            }
        },
        "manager":"kubectl",
        "operation":"Update",
        "time":"2020-07-24T14:27:58Z"
    },
    {
        "apiVersion":"psmdb.percona.com/v1",
        "fieldsType":"FieldsV1",
        "fieldsV1":{
            "f:spec":{
                "f:backup":{
                    "f:containerSecurityContext":{
                        ".":{

                        },
                        "f:runAsNonRoot":{

                        },
                        "f:runAsUser":{

                        }
                    },
                    "f:podSecurityContext":{
                        ".":{

                        },
                        "f:fsGroup":{

                        }
                    }
                },
                "f:clusterServiceDNSSuffix":{

                },
                "f:runUid":{

                },
                "f:secrets":{
                    "f:ssl":{

                    },
                    "f:sslInternal":{

                    }
                }
            },
            "f:status":{
                ".":{

                },
                "f:conditions":{

                },
                "f:observedGeneration":{

                },
                "f:replsets":{
                    ".":{
```

```
                     },
                     "f:rs0":{
                        ".":{

                        },
                        "f:ready":{

                        },
                        "f:size":{

                        },
                        "f:status":{

                        }
                     }
                  },
                  "f:state":{

                  }
               }
            },
            "manager":"percona-server-mongodb-operator",
            "operation":"Update",
            "time":"2020-07-24T15:35:14Z"
         },
         {
            "apiVersion":"psmdb.percona.com/v1",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:spec":{
                  "f:image ":{

                  },
                  "f:replsets":{
                     ".":{

                     },
                     "f:size":{

                     }
                  }
               }
            },
            "manager":"kubectl",
            "operation":"Update",
            "time":"2020-07-27T12:21:39Z"
         }
      ],
      "name":"my-cluster-name",
      "namespace":"default",
      "resourceVersion":"1279853",
      "selfLink":"/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs/my-cluster-name",
      "uid":"5207e71a-c83f-4707-b892-63aa93fb615c"
   },
   "spec":{
      "allowUnsafeConfigurations":false,
      "backup":{
         "enabled":true,
         "image":"percona/percona-server-mongodb-operator:1.5.0-backup",
         "restartOnFailure":true,
         "serviceAccountName":"percona-server-mongodb-operator",
         "storages":null,
         "tasks":null
      },
      "image ":"percona/percona-server-mongodb:4.2.8-8",
      "imagePullPolicy":"Always",
      "mongod":{
```

```
        "net":{
           "hostPort":0,
           "port":27017
        },
        "operationProfiling":{
           "mode":"slowOp",
           "rateLimit":100,
           "slowOpThresholdMs":100
        },
        "security":{
           "enableEncryption":true,
           "encryptionCipherMode":"AES256-CBC",
           "encryptionKeySecret":"my-cluster-name-mongodb-encryption-key",
           "redactClientLogData":false
        },
        "setParameter":{
           "ttlMonitorSleepSecs":60,
           "wiredTigerConcurrentReadTransactions":128,
           "wiredTigerConcurrentWriteTransactions":128
        },
        "storage":{
           "engine":"wiredTiger",
           "inMemory":{
              "engineConfig":{
                 "inMemorySizeRatio":0.9
              }
           },
           "mmapv1":{
              "nsSize":16,
              "smallfiles":false
           },
           "wiredTiger":{
              "collectionConfig":{
                 "blockCompressor":"snappy"
              },
              "engineConfig":{
                 "cacheSizeRatio":0.5,
                 "directoryForIndexes":false,
                 "journalCompressor":"snappy"
              },
              "indexConfig":{
                 "prefixCompression":true
              }
           }
        }
     },
     "pmm":{
        "enabled":false,
        "image":"percona/percona-server-mongodb-operator:1.5.0-pmm",
        "serverHost":"monitoring-service"
     },
     "replsets":{
        "size":"5"
     },
     "secrets":{
        "users":"my-cluster-name-secrets"
     },
     "updateStrategy":"SmartUpdate"
  },
  "status":{
     "conditions":[
        {
           "lastTransitionTime":"2020-07-24T14:28:03Z",
           "status":"True",
           "type":"ClusterInitializing"
        },
        {
           "lastTransitionTime":"2020-07-24T14:28:39Z",
           "status":"True",
```

```
            "type":"Error"
         },
         {
            "lastTransitionTime":"2020-07-24T14:28:41Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:28:41Z",
            "status":"True",
            "type":"Error"
         },
         {
            "lastTransitionTime":"2020-07-24T14:29:10Z",
            "status":"True",
            "type":"ClusterReady"
         },
         {
            "lastTransitionTime":"2020-07-24T14:49:46Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:50:00Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:52:31Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:52:43Z",
            "status":"True",
            "type":"Error"
         },
         {
            "lastTransitionTime":"2020-07-24T14:53:01Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:53:05Z",
            "status":"True",
            "type":"ClusterInitializing"
         },
         {
            "lastTransitionTime":"2020-07-24T14:53:05Z",
            "status":"True",
            "type":"ClusterReady"
         }
      ],
      "observedGeneration":1,
      "replsets":{
         "rs0":{
            "ready":3,
            "size":3,
            "status":"ready"
         }
      },
      "state":"ready"
   }
}
```

16.4.7 Backup Percona Server for MongoDB cluster

Description:

```
Takes a backup of the Percona Server for MongoDB cluster containers data to be able to
recover from disasters or make a roll-back later
```

Kubectl Command:

```
$ kubectl apply -f percona-server-mongodb-operator/deploy/backup/backup.yaml
```

URL:

```
https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbbackups
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XPOST "https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/
perconaservermongodbbackups" \
            -H "Accept: application/json" \
            -H "Content-Type: application/json" \
            -d "@backup.json" -H "Authorization: Bearer $KUBE_TOKEN"
```

Request Body (backup.json):

Example

```
{
   "apiVersion":"psmdb.percona.com/v1",
   "kind":"PerconaServerMongoDBBackup",
   "metadata":{
      "name":"backup1",
      "namespace":"default"
   },
   "spec":{
      "psmdbCluster":"my-cluster-name",
      "storageName":"s3-us-west"
   }
}
```

Input:

1. **metadata**:

name(String, min-length:1) : `name of backup to create`

1. **spec**:

```
1. psmdbCluster(String, min-length:1) : `name of Percona Server for MongoDB cluster`

2. storageName(String, min-length:1) : `name of storage claim to use`
```

Response:

Example


```
{
   "apiVersion":"psmdb.percona.com/v1",
   "kind":"PerconaServerMongoDBBackup",
   "metadata":{
      "annotations":{
         "kubectl.kubernetes.io/last-applied-configuration":"{\"apiVersion\":\"psmdb.percona.com/
v1\",\"kind\":\"PerconaServerMongoDBBackup\",\"metadata\":{\"annotations\":{},\"name\":
\"backup1\",\"namespace\":\"default\"},\"spec\":{\"psmdbCluster\":\"my-cluster-name\",
\"storageName\":\"s3-us-west\"}}\n"
      },
      "creationTimestamp":"2020-07-27T13:45:43Z",
      "generation":1,
      "managedFields":[
         {
            "apiVersion":"psmdb.percona.com/v1",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:metadata":{
                  "f:annotations":{
                     ".":{

                     },
                     "f:kubectl.kubernetes.io/last-applied-configuration":{

                     }
                  }
               },
               "f:spec":{
                  ".":{

                  },
                  "f:psmdbCluster":{

                  },
                  "f:storageName":{

                  }
               }
            },
            "manager":"kubectl",
            "operation":"Update",
            "time":"2020-07-27T13:45:43Z"
         }
      ],
      "name":"backup1",
      "namespace":"default",
      "resourceVersion":"1290243",
      "selfLink":"/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbbackups/
backup1",
      "uid":"e695d1c7-898e-44b0-b356-537284f6c046"
   },
   "spec":{
      "psmdbCluster":"my-cluster-name",
      "storageName":"s3-us-west"
   }
}
```

16.4.8 Restore Percona Server for MongoDB cluster

Description:

```
Restores Percona Server for MongoDB cluster data to an earlier version to recover from a
problem or to make a roll-back
```

Kubectl Command:

```
$ kubectl apply -f percona-server-mongodb-operator/deploy/backup/restore.yaml
```

URL:

```
https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/
perconaservermongodbrestores
```

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XPOST "https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/
perconaservermongodbrestores" \
            -H "Accept: application/json" \
            -H "Content-Type: application/json" \
            -d "@restore.json" \
            -H "Authorization: Bearer $KUBE_TOKEN"
```

Request Body (restore.json):

Example

```
{
   "apiVersion":"psmdb.percona.com/v1",
   "kind":"PerconaServerMongoDBRestore",
   "metadata":{
      "name":"restore1",
      "namespace":"default"
   },
   "spec":{
      "backupName":"backup1",
      "clusterName":"my-cluster-name"
   }
}
```

Input:

1. **metadata**:

   name(String, min-length:1): `name of restore to create`

1. **spec**:

```
1. clusterName(String, min-length:1) : `name of Percona Server for MongoDB cluster`

2. backupName(String, min-length:1) : `name of backup to restore from`
```

Response:

Example

```
{
   "apiVersion":"psmdb.percona.com/v1",
   "kind":"PerconaServerMongoDBRestore",
   "metadata":{
      "annotations":{
         "kubectl.kubernetes.io/last-applied-configuration":"{\"apiVersion\":\"psmdb.percona.com/
v1\",\"kind\":\"PerconaServerMongoDBRestore\",\"metadata\":{\"annotations\":{},\"name\":
\"restore1\",\"namespace\":\"default\"},\"spec\":{\"backupName\":\"backup1\",\"clusterName\":
\"my-cluster-name\"}}\n"
      },
      "creationTimestamp":"2020-07-27T13:52:56Z",
      "generation":1,
      "managedFields":[
         {
            "apiVersion":"psmdb.percona.com/v1",
            "fieldsType":"FieldsV1",
            "fieldsV1":{
               "f:metadata":{
                  "f:annotations":{
                     ".":{

                     },
                     "f:kubectl.kubernetes.io/last-applied-configuration":{

                     }
                  }
               },
               "f:spec":{
                  ".":{

                  },
                  "f:backupName":{

                  },
                  "f:clusterName":{

                  }
               }
            },
            "manager":"kubectl",
            "operation":"Update",
            "time":"2020-07-27T13:52:56Z"
         }
      ],
      "name":"restore1",
      "namespace":"default",
      "resourceVersion":"1291198",
      "selfLink":"/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbrestores/
restore1",
      "uid":"17e982fe-ac41-47f4-afba-fea380b0c76e"
   },
   "spec":{
      "backupName":"backup1",
      "clusterName":"my-cluster-name"
   }
}
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-12-20

16.5 Frequently Asked Questions

16.5.1 Why do we need to follow “the Kubernetes way” when Kubernetes was never intended to run databases?

As it is well known, the Kubernetes approach is targeted at stateless applications but provides ways to store state (in Persistent Volumes, etc.) if the application needs it. Generally, a stateless mode of operation is supposed to provide better safety, sustainability, and scalability, it makes the already-deployed components interchangeable. You can find more about substantial benefits brought by Kubernetes to databases in this blog post.

The architecture of state-centric applications (like databases) should be composed in a right way to avoid crashes, data loss, or data inconsistencies during hardware failure. Percona Operator for MongoDB provides out-of-the-box functionality to automate provisioning and management of highly available MongoDB database clusters on Kubernetes.

16.5.2 How can I contact the developers?

The best place to discuss Percona Operator for MongoDB with developers and other community members is the community forum.

If you would like to report a bug, use the Percona Operator for MongoDB project in JIRA.

16.5.3 What is the difference between the Operator quickstart and advanced installation ways?

As you have noticed, the installation section of docs contains both quickstart and advanced installation guides.

The quickstart guide is simpler. It has fewer installation steps in favor of predefined default choices. Particularly, in advanced installation guides, you separately apply the Custom Resource Definition and Role-based Access Control configuration files with possible edits in them. At the same time, quickstart guides rely on the all-inclusive bundle configuration.

At another point, quickstart guides are related to specific platforms you are going to use (Minikube, Google Kubernetes Engine, etc.) and therefore include some additional steps needed for these platforms.

Generally, rely on the quickstart guide if you are a beginner user of the specific platform and/or you are new to the Percona Operator for MongoDB as a whole.

16.5.4 Which versions of MongoDB the Operator supports?

Percona Operator for MongoDB provides a ready-to-use installation of the MongoDB-based database cluster inside your Kubernetes installation. It works with Percona Server for MongoDB 4.2, and 4.4, and the exact version is determined by the Docker image in use.

Percona-certified Docker images used by the Operator are listed here. For example, Percona Server for MongoDB 4.4 is supported with the following recommended version: 4.4.24-23. More details on the exact Percona Server for MongoDB version can be found in the release notes (5.0, 4.4, and 4.2).

16.5.5 How can I add custom sidecar containers to my cluster?

The Operator allows you to deploy additional (so-called *sidecar*) containers to the Pod. You can use this feature to run debugging tools, some specific monitoring solutions, etc. Add such sidecar container to the `deploy/cr.yaml` configuration file, specifying its name and image, and possibly a command to run:

```yaml
spec:
  replsets:
  - name: rs0
    ....
    sidecars:
    - image: busybox
      command: ["/bin/sh"]
      args: ["-c", "while true; do echo echo $(date -u) 'test' >> /dev/null; sleep 5; done"]
      name: rs-sidecar-1
    ....
```

You can add `sidecars` subsection to `replsets`, `sharding.configsvrReplSet`, and `sharding.mongos` sections.

Note

Custom sidecar containers can easily access other components of your cluster. Therefore they should be used carefully and by experienced users only.

Find more information on sidecar containers in the appropriate documentation page.

16.5.6 How to provoke the initial sync of a Pod

There are certain situations where it might be necessary to delete all MongoDB instance data to force the resync. For example, there may be the following reasons:

- rebuilding the node to defragment the database,
- recreating the member failing to sync due to some bug.

In the case of a "regular" MongoDB, wiping the dbpath would trigger such resync. In the case of a MongoDB cluster controlled by the Operator, you will need to do the following steps:

1. Find out the names of the Persistent Volume Claim and Pod you are going to delete (use `kubectl get pvc` command for PVC and `kubectl get pod` one for Pods).
2. Delete the appropriate PVC and Pod. For example, wiping out the `my-cluster-name-rs0-2` Pod should look as follows:

```
$ kubectl delete pod/my-cluster-name-rs0-2 pvc/mongod-data-my-cluster-name-rs0-2
```

The Operator will automatically recreate the needed Pod and PVC after deletion.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-12-20

16.6 Copyright and licensing information

16.6.1 Documentation licensing

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-06-27

16.7 Trademark policy

This Trademark Policy is to ensure that users of Percona-branded products or services know that what they receive has really been developed, approved, tested and maintained by Percona. Trademarks help to prevent confusion in the marketplace, by distinguishing one company's or person's products and services from another's.

Percona owns a number of marks, including but not limited to Percona, XtraDB, Percona XtraDB, XtraBackup, Percona XtraBackup, Percona Server, and Percona Live, plus the distinctive visual icons and logos associated with these marks. Both the unregistered and registered marks of Percona are protected.

Use of any Percona trademark in the name, URL, or other identifying characteristic of any product, service, website, or other use is not permitted without Percona's written permission with the following three limited exceptions.

First, you may use the appropriate Percona mark when making a nominative fair use reference to a bona fide Percona product.

Second, when Percona has released a product under a version of the GNU General Public License ("GPL"), you may use the appropriate Percona mark when distributing a verbatim copy of that product in accordance with the terms and conditions of the GPL.

Third, you may use the appropriate Percona mark to refer to a distribution of GPL-released Percona software that has been modified with minor changes for the sole purpose of allowing the software to operate on an operating system or hardware platform for which Percona has not yet released the software, provided that those third party changes do not affect the behavior, functionality, features, design or performance of the software. Users who acquire this Percona-branded software receive substantially exact implementations of the Percona software.

Percona reserves the right to revoke this authorization at any time in its sole discretion. For example, if Percona believes that your modification is beyond the scope of the limited license granted in this Policy or that your use of the Percona mark is detrimental to Percona, Percona will revoke this authorization. Upon revocation, you must immediately cease using the applicable Percona mark. If you do not immediately cease using the Percona mark upon revocation, Percona may take action to protect its rights and interests in the Percona mark. Percona does not grant any license to use any Percona mark for any other modified versions of Percona software; such use will require our prior written permission.

Neither trademark law nor any of the exceptions set forth in this Trademark Policy permit you to truncate, modify or otherwise use any Percona mark as part of your own brand. For example, if XYZ creates a modified version of the Percona Server, XYZ may not brand that modification as "XYZ Percona Server" or "Percona XYZ Server", even if that modification otherwise complies with the third exception noted above.

In all cases, you must comply with applicable law, the underlying license, and this Trademark Policy, as amended from time to time. For instance, any mention of Percona trademarks should include the full trademarked name, with proper spelling and capitalization, along with attribution of ownership to Percona Inc. For example, the full proper name for XtraBackup is Percona XtraBackup. However, it is acceptable to omit the word "Percona" for brevity on the second and subsequent uses, where such omission does not cause confusion.

In the event of doubt as to any of the conditions or exceptions outlined in this Trademark Policy, please contact trademarks@percona.com for assistance and we will do our very best to be helpful.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-06-27

17. Release notes

17.1 Percona Operator for MongoDB Release Notes

- *Percona Operator for MongoDB* 1.15.0 (2023-10-09)
- *Percona Operator for MongoDB* 1.14.0 (2023-03-13)
- *Percona Operator for MongoDB* 1.13.0 (2022-09-08)
- *Percona Operator for MongoDB* 1.12.0 (2022-05-05)
- *Percona Distribution for MongoDB Operator* 1.11.0 (2021-12-21)
- *Percona Distribution for MongoDB Operator* 1.10.0 (2021-09-30)
- *Percona Distribution for MongoDB Operator* 1.9.0 (2021-07-29)
- *Percona Kubernetes Operator for Percona Server for MongoDB* 1.8.0 (2021-05-06)
- *Percona Kubernetes Operator for Percona Server for MongoDB* 1.7.0 (2021-03-08)
- *Percona Kubernetes Operator for Percona Server for MongoDB* 1.6.0 (2020-12-22)
- *Percona Kubernetes Operator for Percona Server for MongoDB* 1.5.0 (2020-09-07)
- *Percona Kubernetes Operator for Percona Server for MongoDB* 1.4.0 (2020-03-31)
- *Percona Kubernetes Operator for Percona Server for MongoDB* 1.3.0 (2019-12-11)
- *Percona Kubernetes Operator for Percona Server for MongoDB* 1.2.0 (2019-09-20)
- *Percona Kubernetes Operator for Percona Server for MongoDB* 1.1.0 (2019-07-15)
- *Percona Kubernetes Operator for Percona Server for MongoDB* 1.0.0 (2019-05-29)

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

17.2 *Percona Operator for MongoDB* 1.15.0

- **Date**

  October 9, 2023

- **Installation**

  Installing Percona Operator for MongoDB

17.2.1 Release Highlights

Physical Backups now support Point-in-time Recovery (in tech preview)

In the previous 1.14.0 release we added support for Physical Backups and Restores to significantly reduce Recovery Time Objective (RTO.)), especially for big data sets. But the problem with losing data between backups - in other words Recovery Point Objective (RPO) - for physical backups was not solved. With this release users can greatly reduce RPO by leveraging the Point-in-time Recovery feature in the Operators. Under the hood we store logical oplogs along with physical backups into the object storage. Read more about this feature in our documentation.

Encrypted backups with Server Side Encryption (SSE)

Backups stored on S3 compatible storage can now be encrypted with Server Side Encryption (SSE) to pass certain compliance or security requirements. Users can leverage integration with AWS KMS or just encrypt/decrypt backups with AES-256 encryption algorithm. It is important to remember that Operator does not store keys and users can choose which key storage to use.

17.2.2 New Features

- K8SPSMDB-227 The new `topologySpreadConstraints` Custom Resource option allows to use Pod Topology Spread Constraints to achieve even distribution of Pods across the Kubernetes cluster
- K8SPSMDB-792 and K8SPSMDB-974 The new "sleep infinity" mode available for replset and config server containers allows running the Pod without starting mongod useful to examine a problematic Pod that is constantly restarting
- K8SPSMDB-801 It is now possible to delete a backup with its PITR data on retention period or with `delete-backup` finalizer (there were no PITR files deletion in previous versions)
- K8SPSMDB-926 Point-in-time recovery is now supported with physical backups to significantly reduce Recovery Point Objective (RPO)
- K8SPSMDB-961 The new `sharding.balancer.enabled` Custom Resource option allows to disable Load Balancer on a cross-site replication managed cluster

17.2.3 Improvements

- K8SPSMDB-662 Restoring a backup with point-in-time recovery can now be easily done to a latest available position by setting `pitr.type` PerconaServerMongoDBRestore Custom Resource option to `latest`
- K8SPSMDB-774 The Transport encryption documentation now includes details on updating TLS certificates
- K8SPSMDB-807 A custom name for a Replica Set config server instead of the default `cfg` one can be set in the custom configuration, which can be useful for migration purposes

- K8SPSMDB-814 and K8SPSMDB-927 The new `terminationGracePeriodSeconds` Custom Resource option allows to set termination period for Replica Set containers, useful to cleanly shutdown clusters with big data sets
- K8SPSMDB-850 Server Side Encryption for backups with for S3 and S3-compatible storage is now supported (thanks to Mert Gönül for contribution)
- K8SPSMDB-903 The backup destination URI now includes bucket/container name, allowing the user to specify the full path to the backup as an easy to read string
- K8SPSMDB-924 The token associated with the operator's ServiceAccount is no longer printed in the log when a scheduled backup is running; this improves security and avoids logging uninformative elements
- K8SPSMDB-938 Configuring Kubernetes host aliases is now possible for replica set, config server, and mongos Pods
- K8SPSMDB-946 The psmdb-backup object now includes the name of the Pod that made the backup, to save users from searching for the correct Pod to examine the Percona Backup for MongoDB logs (previously it was necessary to check replica set Pods one by one until logs were found)
- K8SPSMDB-976 The Operator now does not start backups if storages or credentials are not set, avoiding fruitless attempts to configure Percona Backup for MongoDB and cluster state repeatedly changing between ready and error
- K8SPSMDB-929 Using split-horizon DNS for the external access to MongoDB Replica Set Pods of the exposed cluster is now possible

17.2.4 Bugs Fixed

- K8SPSMDB-913 Fix a bug due to which restoring a backup on a cluster with mongos exposed via LoabBalancer resulted in recreating mongos Service with a new IP address
- K8SPSMDB-956 Fix a bug that certificate rotation was bringing the sharded MongoDB cluster down (thanks to Stiliyan for reporting)
- K8SPSMDB-854 Backup stucks after cluster was exposed
- K8SPSMDB-977 The out of memory problem could cause cluster got stuck in the "initializing" state at reconciliation
- K8SPSMDB-778 Fix a bug due to which the Operator did not delete arbiter instances during replica set deletion
- K8SPSMDB-791 Fix a bug which prevented setting `LoadBalancerSourceRanges` Custom Resource option when `replsets.expose.exposeType` is set to `Loadbalancer`
- K8SPSMDB-813 Fix a bug due to which secure connection was not used for MongoDB Liveness check (thanks to t-yrka for contribution)
- K8SPSMDB-818 Fix a bug where `clusterMonitor` user had not enough permissions for PMM monitoring with `--enable-all-collectors` flag turned on
- K8SPSMDB-872 The Operator didn't prevent attempts to restore a backup with "error" status, which could cause the cluster got stuck in the "initializing" state
- K8SPSMDB-876 Fix a bug due to which `delete-psmdb-pods-in-order` finalizer, intended to shutdown primary Pod last, affected only shards and did not affect config replica set
- K8SPSMDB-911 Fix a bug where connection string with credentials was included in the backup-agent container logs
- K8SPSMDB-958 Fix insufficient permissions issue that didn't allow to monitor mongos instances with Percona Monitoring and Management (PMM)
- K8SPSMDB-962 Fix a memory leak due to which the Operator's Pod continually increased both CPU and memory usage in cluster-wide mode (with an unmanaged cluster)

- K8SPSMDB-968 Fix a bug due to which the endpoints list returned by `kubectl get psmdb` command contained fully qualified domain names (FQDN) instead of IP addresses when the replset was exposed as a LoadBalancer and the clusterServiceDNSMode was set to Internal

17.2.5 Deprecation and removal

- K8SPSMDB-883 The `spec.mongod` section deprecated in the Operator version 1.12.0 is finally removed from the Custom Resource configuration. If you have encryption disabled using the deprecated `mongod.security.enableEncryption` option, you need to set encryption disabled via the custom configuration before removing `mongod` section (and before upgrade):

```
spec:
  ...
  replsets:
    - name: rs0
      ...
      configuration: |
        security:
          enableEncryption: false
        ...
```

17.2.6 Supported Platforms

The Operator was developed and tested with Percona Server for MongoDB 4.4.24, 5.0.20, and 6.0.9. Other options may also work but have not been tested. The Operator also uses Percona Backup for MongoDB 2.3.0.

The following platforms were tested and are officially supported by the Operator 1.15.0:

- Google Kubernetes Engine (GKE) 1.24-1.28
- Amazon Elastic Container Service for Kubernetes (EKS) 1.24-1.28
- OpenShift Container Platform 4.11 - 4.13
- Azure Kubernetes Service (AKS) 1.25-1.28
- Minikube 1.31.2 (based on Kubernetes 1.28)

This list only includes the platforms that the Percona Operators are specifically tested on as part of the release process. Other Kubernetes flavors and versions depend on the backward compatibility offered by Kubernetes itself.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-10-09

17.3 *Percona Operator for MongoDB* 1.14.0

- **Date**

  March 13, 2023

- **Installation**

  Installing Percona Operator for MongoDB

17.3.1 Release Highlights

- Backups and Restores are critical for business continuity. With this release you can significantly reduce your Recovery Time Objective (RTO) with Physical backups support in the Operator. The feature is now in technical preview.
- MongoDB 6.0 comes with a variety of improvements and new features. It is now fully supported by the Operator. See our documentation to learn how to upgrade.

17.3.2 New Features

- K8SPSMDB-713 Physical backups are now supported by the Operator to recover big data sets faster
- K8SPSMDB-737 MongoDB 6.0 is now officially supported in addition to 4.x and 5.x versions. Read more about version 6 in our blog post
- K8SPSMDB-824 New `ignoreAnnotations` and `ignoreLabels` Custom Resource options allow to list specific annotations and labels for Kubernetes Service objects, which the Operator should ignore (useful with various Kubernetes flavors which add annotations to the objects managed by the Operator)

17.3.3 Improvements

- K8SPSMDB-658 The Operator log messages appearing during the pause/unpause of the cluster were improved to more clearly indicate this event
- K8SPSMDB-708 The new `initContainerSecurityContext` option allows to configure securityContext for the container which can be used instead of the official image during the initial Operator installation
- K8SPSMDB-721 The backup subsystem was improved so that database is not crashing in case if the backup agent is not able to connect to MongoDB (e.g. due to misconfigured password)
- K8SPSMDB-758 The ServiceMesh fully qualified domain names (FQDNs) for config servers are now prioritized if DNSMode is set to ServiceMesh (thanks to Jo Lyshoel for contribution)
- K8SPSMDB-793 It is now possible to set annotations and labels for Persistent Volume Claims for better integration with Cloud Native tools
- K8SPSMDB-803 The Operator now does not attempt to start Percona Monitoring and Management (PMM) client sidecar if the corresponding secret does not contain the `pmmserver` or `pmmserverkey` key
- K8SPSMDB-817 Adding external nodes to the cluster is now allowed even when the replica set is not exposed. This unblocks the creation of complex multi-cluster topologies
- K8SPSMDB-844 Update the RuntimeClass API version to `v1` from the `v1beta1`, which was already deprecated since the Kubernetes version 1.22
- K8SPSMDB-848 Remove formatted strings from log messages to avoid confronting with structured logging based on key-value pairs
- K8SPSMDB-882 Percona Server for MongoDB Helm chart now persists data by default instead of deleting Persistent Volumes after the cluster deletion
- CLOUD-768 Helm charts now use random passwords generated by the Operator by default instead of providing pre-configured passwords specified in the values file

- K8SPSMDB-853 To improve the operator we capture anonymous telemetry and usage data. In this release we add more data points to it
- K8SPSMDB-867 The Operator now configures replset members using local fully-qualified domain names (FQDN) resolvable and available only from inside the cluster instead of using IP addresses; the old behavior can be restored by setting the `clusterServiceDNSMode` option to `External`

17.3.4 Bugs Fixed

- K8SPSMDB-784 Fix a bug due to which the `enableEncryption` MongoDB configuration option was always activated when using psmdb-db Helm Chart
- K8SPSMDB-796 Fix a bug due to which backup failed if replica set was exposed
- K8SPSMDB-854 Fix a bug due to which backup got stuck after the cluster was exposed
- K8SPSMDB-471 Fix a bug due to which in case of scheduled backups with error status `delete-backup` finalizer didn't allow to delete the appropriate failed resources and the Kubernetes namespace (thanks to Aliaksandr Karavai for reporting)
- K8SPSMDB-674 Fix a bug that caused the Operator not deleting unneeded Services after the replica set exposing is turned off
- K8SPSMDB-742 Fix a bug that caused the updates of the `sharding.mongos.expose.serviceAnnotations` option to be silently rejected
- K8SPSMDB-766 and K8SPSMDB-767 Fix a bug where the combination of `delete-psmdb-pods-in-order` and `delete-psmdb-pvc` finalizers was not working
- K8SPSMDB-770 We now mention the namespace name in the log message to ease debugging when the cluster-wide mode is used
- K8SPSMDB-797 Fix the backup/restore documentation not clearly mentioning that user should specify the bucket for the S3 storage
- K8SPSMDB-820 Fix a bug which prevented the parallel backup jobs execution for different MongoDB clusters in the cluster-wide mode
- K8SPSMDB-823 Fix a bug where backups were not working in case of ReplicaSet exposed with NodePort
- K8SPSMDB-836 Fix backups being incorrectly marked as error while still being in starting status
- K8SPSMDB-841 Fix a bug which turned the cluster into unready status after switching from the LoadBalancer expose to ClusterIP
- K8SPSMDB-843 Fix a bug which made the cluster unable to start if it was recreated with the same Custom Resource after delete without deleting PVCs and Secrets
- K8SPSMDB-846 Fix a bug due to which scaling the replica set down to 1 instance caused the last Pod to remain Secondary instead of becoming Primary
- K8SPSMDB-866 Fix the bug due to which the Operator was continuously flooding the log with error messages if the PMM server credentials were missing

17.3.5 Known Issues and Limitations

- K8SPSMDB-875 Physical backups cannot be restored on the clusters with arbiter, non-voting, or delayed members due to current Percona Backup for MongoDB limitations
- K8SPSMDB-846 After switching the cluster to unsafe mode by setting allowUnsafeConfig: true, it is not possible to switch back into safe mode. The user can still scale the cluster safely, but the flag is ignored

17.3.6 Supported Platforms

The Operator was developed and tested with Percona Server for MongoDB 4.4.18, 5.0.14, and 6.0.4. Other options may also work but have not been tested.

The following platforms were tested and are officially supported by the Operator 1.14.0:

- Google Kubernetes Engine (GKE) 1.22 - 1.25
- Amazon Elastic Container Service for Kubernetes (EKS) 1.22 - 1.24
- OpenShift Container Platform 4.10 - 4.12
- Azure Kubernetes Service (AKS) 1.23 - 1.25
- Minikube 1.29

This list only includes the platforms that the Percona Operators are specifically tested on as part of the release process. Other Kubernetes flavors and versions depend on the backward compatibility offered by Kubernetes itself.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2023-03-13

17.4 *Percona Operator for MongoDB* 1.13.0

- **Date**

  September 15, 2022

- **Installation**

  Installing Percona Operator for MongoDB

17.4.1 Release Highlights

- Azure Kubernetes Service (AKS) is now officially supported platform, so developers and vendors of the solutions based on the Azure platform can take advantage of the official support from Percona or just use officially certified Percona Operator for MongoDB images
- Starting from now, the Operator can be installed in multi-namespace (so-called "cluster-wide") mode, when a single Operator can be given a list of namespaces in which to manage Percona Server for MongoDB clusters

17.4.2 New Features

- K8SPSMDB-203 Support for the cluster-wide operator mode allowing one Operator to watch for Percona Server for MongoDB Custom Resources in several namespaces
- K8SPSMDB-287 Support for the HashiCorp Vault for encryption keys as a universal, secure and reliable way to store and distribute secrets without depending on the operating system, platform or cloud provider
- K8SPSMDB-704 Support for the Azure Kubernetes Service (AKS)

17.4.3 Improvements

- K8SPSMDB-515 Allow setting requireTLS mode for MongoDB through the Operator to enforce security by restricting each MongoDB server to use TLS/SSL encrypted connections only
- K8SPSMDB-636 An additional `databaseAdmin` user was added to the list of system users which are automatically created by the Operator. This user is intended to provision databases, collections and perform data modifications
- K8SPSMDB-699 Disable automated upgrade by default to prevent an unplanned downtime for user applications and to provide defaults more focused on strict user's control over the cluster
- K8SPSMDB-725 Configuring the log structuring and leveling is now supported using the `LOG_STRUCTURED` and `LOG_LEVEL` environment variables. This reduces the information overload in logs, still leaving the possibility of getting more details when needed, for example, for debugging
- K8SPSMDB-719 Details about using sharding, Hashicorp Vault and cluster-wide mode were added to telemetry
- K8SPSMDB-715 Starting from now, the Opearator changed its API version to v1 instead of having a separate API version for each release. Three last API version are supported in addition to `v1`, which substantially reduces the size of Custom Resource Definition to prevent reaching the etcd limit
- K8SPSMDB-709 Make it possible to use API Key to authorize within Percona Monitoring and Management Server as a more convenient and modern alternative password-based authentication
- K8SPSMDB-707 Allow to set Service labels for replica set, config servers and mongos in Custom Resource to enable various integrations with cloud providers or service meshes

17.4.4 Bugs Fixed

- K8SPSMDB-702 Fix a bug which resulted in always using the `force` option when reconfiguring MongoDB member, which is normally recommended only for special scenarios such as crash recovery
- K8SPSMDB-730 Fix a bug due to which point-in-time recovery was enabled and consequently disabled when setting Percona Backup for MongoDB compression options without checking whether it was enabled in the Custom Resource
- K8SPSMDB-660 Fix a bug due to which a successful backup could be erroneously marked as failed due to exceeding the start deadline in case of big number of nodes, especially on sharded clusters
- K8SPSMDB-686 Fix a bug that prevented downscaling sharded MongoDB cluster to a non-sharded replica set variant
- K8SPSMDB-691 Fix a bug that produced an error in the Operator log in case of the empty SSL Secret name in Custom Resource
- K8SPSMDB-696 Fix a bug that prevented removing additional annotations previously added under the `spec.replsets.annotations` field
- K8SPSMDB-724 Fix a bug which caused the delete-backup finalizer not working causing backups being not deleted from buckets
- K8SPSMDB-746 Fix a bug due to which the Operator was unable to initialize a three-member replica set with a primary-secondary-arbiter (PSA) architecture
- K8SPSMDB-762 Fix a bug due to which the Operator was running the replSetReconfig MongoDB command at every reconciliation if arbiter was enabled

17.4.5 Deprecation, Rename and Removal

- K8SPSMDB-690 CCustom Resource options under the sharding.mongos.auditLog subsection, deprecated since the Operator version 1.9.0 in favor of using replsets.configuration, were finally removed and cannot be used with the Operator
- K8SPSMDB-709 Password-based authorization to Percona Monitoring and Management Server is now deprecated and will be removed in future releases in favor of a token-based one. Password-based authorization was used by the Operator before this release to provide MongoDB monitoring, but now using the API Key is the recommended authorization method

17.4.6 Supported Platforms

The Operator was developed and tested with Percona Server for MongoDB 4.2.22, 4.4.8, 4.4.10, 4.4.13, 4.4.16, 5.0.2, 5.0.4, and 5.0.11. Other options may also work but have not been tested.

The following platforms were tested and are officially supported by the Operator 1.13.0:

- Google Kubernetes Engine (GKE) 1.21 - 1.23
- Amazon Elastic Container Service for Kubernetes (EKS) 1.21 - 1.23
- OpenShift Container Platform 4.10 - 4.11
- Azure Kubernetes Service (AKS) 1.22 - 1.24
- Minikube 1.26

This list only includes the platforms that the Percona Operators are specifically tested on as part of the release process. Other Kubernetes flavors and versions depend on the backward compatibility offered by Kubernetes itself.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-09-15

17.5 *Percona Operator for MongoDB* 1.12.0

- **Date**

  May 5, 2022

- **Installation**

  Installing Percona Operator for MongoDB

17.5.1 Release Highlights

- With this release, the Operator turns to a simplified naming convention and changes its official name to **Percona Operator for MongoDB**
- The Operator is able now to use the Amazon Web Services feature of authenticating applications running on EC2 instances based on Identity and Access Management (IAM) roles assigned to the instance; this makes it possible to configure S3 backup on AWS without using IAM keys saved in Secrets
- This release brings support for the Multi Cluster Services (MCS). This allows users to deploy MongoDB with Percona Operator across multiple Kubernetes clusters using MCS, which extends the reach of the Service object beyond one cluster, so one Service can be used across multiple clusters. It can be used to provide disaster recovery or perform a migration for MongoDB clusters.
- The OpenAPI schema is now generated for the Operator , which allows Kubernetes to perform Custom Resource validation and saves user from occasionally applying `deploy/cr.yaml` with syntax typos

17.5.2 New Features

- K8SPSMDB-185: Allow using AWS EC2 instances for backups with IAM roles assigned to the instance instead of using stored IAM credentials (Thanks to Oleksii for reporting this issue)
- K8SPSMDB-625: Integrate the Operator with Multi Cluster Services (MCS)
- K8SPSMDB-668: Adding support for enabling replication over a service mesh (Thanks to Jo Lyshoel for contribution)

17.5.3 Improvements

- K8SPSMDB-473: Allow to skip TLS verification for backup storage, useful for self-hosted S3-compatible storage with a self-issued certificate
- K8SPSMDB-644: Make `cacheSizeRatio` parameter available as a custom value in psmdb-db-1.11.0 helm chart (Thanks to Richard CARRE for reporting this issue)
- K8SPSMDB-574: Allow user to choose the validity duration of the external certificate for cert manager
- K8SPSMDB-634: Support point-in-time recovery compression levels for backups (Thanks to Damiano Albani for reporting this issue)
- K8SPSMDB-570: The Operator documentation now includes a How-To on using Percona Server for MongoDB with LDAP authentication and authorization
- K8SPSMDB-537: PMM container does not cause the crash of the whole database Pod if pmm-agent is not working properly
- K8SPSMDB-684: Generate OpenAPI schema for and validate Custom Resource

17.5.4 Bugs Fixed

- K8SPSMDB-597: Fix a bug in the Operator helm chart which caused deleting the watched Namespace on uninstall (Thanks to Andrei Nistor for reporting this issue)

- K8SPSMDB-640: Fix a regression which prevented labels from being applied to Pods after the Custom Resource change
- K8SPSMDB-583: Fix a bug which caused backup crashing if `spec.mongod.net.port` not set or set to zero
- K8SPSMDB-540 and K8SPSMDB-563: Fix a bug which could cause a cluster crash when reducing the configured Replicaset size between deletion and re-creation of the cluster
- K8SPSMDB-608: Fix a bug due to which the password of backup user was printed in backup agent logs (Thanks to Antoine Ozenne for reporting this issue)
- K8SPSMDB-599: A new mongos.expose.servicePerPod option allows deploying a separate ClusterIP Service for each mongos instance, which prevents the failure of a multi-threaded transaction executed with the same driver instance and ended up on a different mongos. Starting from this release, mongos is deployed by StatefulSet instead of Deployment object
- K8SPSMDB-656: Fix a bug which caused cluster name being not displayed in the backup Custom Resource output with `psmdbCluster` set in the backup spec
- K8SPSMDB-653: Fix a bug due to which `spec.ImagePullPolicy` options from `deploy/cr.yaml` wasn't applied to backup and pmm-client images
- K8SPSMDB-632: Fix a bug which caused the Operator to perform Smart Update on the initial deployment
- K8SPSMDB-624: Fix a bug due to which the Operator didn't grant enough permissions to the Cluster Monitor user necessary for Percona Monitoring and Management (PMM) (Thanks to Richard CARRE for reporting this issue)
- K8SPSMDB-618: Improve security and meet compliance requirements by building MongoDB Operator based on Red Hat Universal Base Image (UBI) 8 instead of UBI 7
- K8SPSMDB-602: Fix a thread leak in a mongod container of the Replica Set Pods, which occurred when setting `setFCV` flag to `true` in Custom Resource
- K8SPSMDB-560: Fix a bug due to which `serviceName` tag was not set to all members in the Replica Set
- K8SPSMDB-533: Fix a bug due to which setting password with a special character for a system user was breaking the cluster

17.5.5 Known Issues

- K8SPSMDB-686: The Operator versions 1.11.0 and 1.12.0 can not be downscaled from a sharding to non-sharding/Replica Set configuration on Google Kubernetes Engine (GKE) 1.19-1.21 (GKE 1.22 is not affected)

17.5.6 Deprecation, Rename and Removal

- K8SPSMDB-596: The `spec.mongod` section is removed from the Custom Resource configuration. Starting from now, mongod options should be passed to Replica Sets using `spec.replsets.[].configuration` key, except the following 3 options:
    - `mongod.security.encryptionKeySecret` key was left in a deprecated state in favor of the new `spec.secrets.encryptionKey` option
    - `mongod.storage.wiredTiger.engineConfig.cacheSizeRatio` and `mongod.storage.inMemory.engineConfig.inMemorySizeRatio` options are now only available from the `replsets.storage` section

Before the upgrade, please ensure that you have moved all custom MongoDB parameters to proper places!

- K8SPSMDB-228: The `spec.psmdbCluster` option in the example on-demand backup configuration file `backup/backup.yaml` was renamed to `spec.clusterName` (`psmdbCluster` will be valid till 1.15 version)

17.5.7 Supported Platforms

The following platforms were tested and are officially supported by the Operator 1.12.0:

- OpenShift 4.7 - 4.10
- Google Kubernetes Engine (GKE) 1.19 - 1.22
- Amazon Elastic Container Service for Kubernetes (EKS) 1.19 - 1.22
- Minikube 1.23

This list only includes the platforms that the Percona Operators are specifically tested on as part of the release process. Other Kubernetes flavors and versions depend on the backward compatibility offered by Kubernetes itself.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-08

17.6 *Percona Distribution for MongoDB Operator* 1.11.0

- **Date**

  December 21, 2021

- **Installation**

  For installation please refer to the documentation page

17.6.1 Release Highlights

- In addition to S3-compatible storage, you can now configure backups to use Microsoft Azure Blob storage. This feature makes the Operator fully compatible with Azure Cloud.
- Custom sidecar containers allow users to customize Percona Distribution for MongoDB and other Operator components without changing the container images. In this release, we enable even more customization, by allowing users to mount volumes into the sidecar containers.

17.6.2 New Features

- K8SPSMDB-513: Add support of Microsoft Azure Blob storage for backups

17.6.3 Improvements

- K8SPSMDB-422: It is now possible to set annotations to backup cron jobs (Thanks to Aliaksandr Karavai for contribution)
- K8SPSMDB-534: mongos readiness probe now avoids running listDatabases command for all databases in the cluster to avoid unneeded delays on clusters with an extremely large amount of databases
- K8SPSMDB-527: Timeout parameters for liveness and readiness probes can be customized to avoid false-positives for heavy-loaded clusters
- K8SPSMDB-520: Mount volumes into sidecar containers to enable customization
- K8SPSMDB-463: Update backup status as error if it’s not started for a long time
- K8SPSMDB-388: New `backup.pitr.oplogSpanMin` option controls how often oplogs are uploaded to the cloud storage

17.6.4 Bugs Fixed

- K8SPSMDB-603: Fixed a bug where the Operator checked the presence of CPU limit and not memory limit when deciding whether to set the size of cache memory for WiredTiger
- K8SPSMDB-511 and K8SPSMDB-558: Fixed a bug where Operator changed NodePort port every 20 seconds for a Replica Set service (Thanks to Rajshekar Reddy for reporting this issue)
- K8SPSMDB-608: Fix a bug that resulted in printing the password of backup user the in backup agent logs (Thanks to Antoine Ozenne for reporting this issue)
- K8SPSMDB-592: Fixed a bug where helm chart was incorrectly setting the `serviceAnnotations` and `loadBalancerSourceRanges` for mongos exposure
- K8SPSMDB-568: Fixed a bug where upgrading to MongoDB 5.0 failed when using the `upgradeOptions:apply` option

17.6.5 Supported Platforms

The following platforms were tested and are officially supported by the Operator 1.11.0:

- OpenShift 4.7 - 4.9
- Google Kubernetes Engine (GKE) 1.19 - 1.22
- Amazon Elastic Container Service for Kubernetes (EKS) 1.18 - 1.22
- Minikube 1.22

This list only includes the platforms that the Percona Operators are specifically tested on as part of the release process. Other Kubernetes flavors and versions depend on the backward compatibility offered by Kubernetes itself.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-08

17.7 *Percona Distribution for MongoDB Operator* 1.10.0

- **Date**

  September 30, 2021

- **Installation**

  For installation please refer to the documentation page

17.7.1 Release Highlights

- Starting from this release, the Operator implements as a technical preview the possibility to include non-voting replica set members into the cluster, which do not participate in the primary election process. This feature enables users to deploy non-voting members with the Operator through a Custom Resource object without manual configuration.
- The technical preview of the cross-site replication feature allows users to add external replica set nodes into the cluster managed by the Operator, including scenarios when one of the clusters is outside of the Kubernetes environment. External nodes can be run by another Operator or can be regular MongoDB deployment. The feature is intended for the following use cases:
  - provide migrations of your regular MongoDB database to the Percona Server for MongoDB cluster under the Operator control, or carry on backward migration,
  - deploy cross-regional clusters for Disaster Recovery.

17.7.2 New Features

- K8SPSMDB-479: Allow users to add non-voting members to MongoDB replica, needed to have more than 7 nodes or to create a node in the edge location
- K8SPSMDB-265: Cross region replication feature simplifies the migrations and enables Disaster Recovery capabilities for MongoDB on Kubernetes

17.7.3 Improvements

- K8SPSMDB-537: PMM container should not cause the crash of the whole database Pod if pmm-agent is not working properly
- K8SPSMDB-517: Users can now run Percona Server for MongoDB 5 with the Operator. Version 5 support is added as a technical preview and is not recommended for Production.
- K8SPSMDB-490: Add validation for the Custom Resource name so that cluster name and replica set name do not exceed 51 characters in total

17.7.4 Bugs Fixed

- K8SPSMDB-504: Fixed a race condition that could prevent the cluster with LoadBalancer-exposed replica set members from becoming ready
- K8SPSMDB-470: Fix a bug where ServiceAnnotation and LoadBalancerSourceRanges fields didn't propagate to Kubernetes service (Thanks to Aliaksandr Karavai for reporting this issue)
- K8SPSMDB-531: Fix compatibility issues between Percona Kubernetes Operator for MongoDB and Calico (Thanks to Mykola Kruliv for reporting this issue)
- K8SPSMDB-514: Fix a bug where backup cronJob created by the Operator did not include resources limits and requests, which prevented it to run in the namespaces with resource quotas (Thanks to George Asenov for reporting this issue)

- K8SPSMDB-512: Fix a bug where configuring getLastErrorModes in the replica set causes the Operator to fail to reconcile (Thanks to Adam Watson for contribution)
- K8SPSMDB-553: Fix a bug where wrong S3 credentials caused backup to keep running despite the actual failure
- K8SPSMDB-496: Fix a bug where Pods did not restart if custom MongoDB config was updated with a secret or a configmap

17.7.5 Supported Platforms

The following platforms were tested and are officially supported by the Operator 1.10.0:

- OpenShift 4.6 - 4.8
- Google Kubernetes Engine (GKE) 1.17 - 1.21
- Amazon Elastic Container Service for Kubernetes (EKS) 1.16 - 1.21
- Minikube 1.22

This list only includes the platforms that the Percona Operators are specifically tested on as part of the release process. Other Kubernetes flavors and versions depend on the backward compatibility offered by Kubernetes itself.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-08

17.8 *Percona Distribution for MongoDB Operator* 1.9.0

- **Date**

  June 29, 2021

- **Installation**

  For installation please refer to the documentation page

17.8.1 Release Highlights

- Starting from this release, the Operator changes its official name to **Percona Distribution for MongoDB Operator**. This new name emphasizes graduate changes which incorporated a collection of Percona's solutions to run and operate MongoDB Server, available separately as Percona Distribution for MongoDB.
- It is now possible to restore backups from S3-compatible storage to a new Kubernetes-based environment with no existing Backup Custom Resources
- You can now customize Percona Server for MongoDB by storing custom configuration for Replica Set, mongos, and Config Server instances in ConfigMaps or in Secrets

17.8.2 New Features

- K8SPSMDB-276: Restore backups to a new Kubernetes-based environment with no existing Backup Custom Resource
- K8SPSMDB-444, K8SPSMDB-445: Allow storing custom configuration in ConfigMaps and Secrets

17.8.3 Improvements

- K8SPSMDB-365: Unblock backups even if just a single Replica Set node is available by setting `allowUnsafeConfigurations` flag to true
- K8SPSMDB-453: It is now possible to see the overall progress of the provisioning of MongoDB cluster resources and dependent components in Custom Resource status
- K8SPSMDB-451, K8SPSMDB-398: MongoDB cluster resource statuses in Custom Resource output (e.g. returned by `kubectl get psmdb` command) have been improved and now provide more precise reporting
- K8SPSMDB-425: Remove `mongos.expose.enabled` option from Custom Resource and always expose mongos (with the ClusterIP exposeType by default)
- K8SPSMDB-421: Secret object containing system users passwords is now deleted along with the Cluster if delete-psmdb-pvc finalizer is enabled
- K8SPSMDB-411: Added options to specify custom memory and CPU requirements for Arbiter instances
- K8SPSMDB-329: Reduced the number of various etcd and k8s object updates from the operator to minimize the pressure on the Kubernetes cluster

17.8.4 Bugs Fixed

- K8SPSMDB-437: Fixed a bug where Labels were not set on Persistent Volume Claim objects when set on the respective Pods
- K8SPSMDB-435: Fixed a bug that prevented adding custom Labels to mongos Pods

- K8SPSMDB-423: Fixed a bug where unpause of a cluster did not work when `replsets.expose = LoadBalancer` because of provisioning new Load Balancers with different names (Thanks to Aliaksandr Karavai for reporting this issue)
- K8SPSMDB-494: When upgrading MongoDB clusters with Smart Update, the statuses reported in Custom Resource are now reflecting the real state
- K8SPSMDB-489: Fixed a bug where the status of successful backups could be set to error in case of a cluster crash
- K8SPSMDB-462: Fixed a bug where psmdb-backup object could not be deleted if the backup was not successful
- K8SPSMDB-456: Fixed a bug where Smart Update was not upgrading a MongoDB deployment with a replica set consisting of one node
- K8SPSMDB-455: Fixed a bug that prevented major version downgrade to a specific version number when `upgradeOptions.setFCV` Custom Resource option was not updated to the new version
- K8SPSMDB-485: Fixed TLS documentation that referenced incorrect Secrets names from the cr.yaml configuration file

17.8.5 Deprecation and Removal

- We are simplifying the way the user can customize MongoDB components such as mongod and mongos. It is now possible to set custom configuration through ConfigMaps and Secrets Kubernetes resources. The following options will be deprecated in Percona Distribution for MongoDB Operator v1.9.0+, and completely removed in v1.12.0+:
  - `sharding.mongos.auditLog.\*`
  - `mongod.security.redactClientLogData`
  - `mongod.security.\*`
  - `mongod.setParameter.\*`
  - `mongod.storage.\*`
  - `mongod.operationProfiling.mode`
  - `mongod.auditLog.\*`
- The mongos.expose.enabled option has been completely removed from the Custom Resource as it was causing confusion for the users

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-08

17.9 *Percona Kubernetes Operator for Percona Server for MongoDB* 1.8.0

- **Date**

  May 6, 2021

- **Installation**

  Installing Percona Kubernetes Operator for Percona Server for MongoDB

17.9.1 Release Highlights

- The support for Point-in-time recovery added in this release. Users can now recover to a specific date and time from operations logs stored on S3
- It is now possible to perform a major version upgrade for MongoDB (for example, upgrade 4.2 version to 4.4) with no manual steps

17.9.2 New Features

- K8SPSMDB-387: Add support for point-in-time recovery to recover to a specific date and time
- K8SPSMDB-284: Add support for automated major version MongoDB upgrades

17.9.3 Improvements

- K8SPSMDB-436: The imagePullPolicy option in the `deploy/cr.yaml` configuration file now is applied to init container as well
- K8SPSMDB-400: Simplify secret change logic to avoid Pod restarts when user changes the credentials
- K8SPSMDB-381: Get credentials directly from Secrets instead of the environment variables when initializing the Replica Set
- K8SPSMDB-352: Restrict running run less than 5 Pods of Replica Sets with enabled arbiter unless the `allowUnsafeConfigurations` option is set to true
- K8SPSMDB-332: Restrict running less than 3 Pods of Config Servers unless the `allowUnsafeConfigurations` option is set to true
- K8SPSMDB-331: Restrict running less than 3 mongos Pods unless the `allowUnsafeConfigurations` option is set to true

17.9.4 Bugs Fixed

- K8SPSMDB-384: Fix a bug due to which mongos Pods were failing readiness probes for some period of time during the cluster initialization
- K8SPSMDB-434: Fix a bug due to which nil pointer dereference error was occurring when switching the `sharding.enabled` option from false to true (thanks to srteam2020 for contributing)
- K8SPSMDB-430: Fix a bug due to which a stale apiserver could trigger undesired StatefulSet and PVC deletion when recreating the cluster with the same name (thanks to srteam2020 for contributing)
- K8SPSMDB-428: Fix a bug which caused mongos to fail in case of the empty name field in configsvrReplSet section of the Custom Resource
- K8SPSMDB-418: Fix a bug due to which `serviceAnnotations` changes in the `deploy/cr.yaml` file were not applied to the running cluster
- K8SPSMDB-364: Fix a bug where liveness probe of a mongo container was always failing if the userAdmin password contained special characters

- K8SPSMDB-43: Fix a bug due to which renaming Replica Set in the Custom Resource caused creating new Replica Set without deleting the old one

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-08

17.10 *Percona Kubernetes Operator for Percona Server for MongoDB* 1.7.0

- **Date**

  March 8, 2021

- **Installation**

  Installing Percona Kubernetes Operator for Percona Server for MongoDB

17.10.1 Release Highlights

- This release brings full support for the Percona Server for MongoDB Sharding. Sharding allows you to scale databases horizontally, distributing data across multiple MongoDB Pods, and so it is extremely useful for large data sets. By default of the `deploy/cr.yaml` configuration file contains only one replica set, but when you turn sharding on, you can add more replica sets with different names to the `replsets` section.
- It is now possible to clean up Persistent Volume Claims automatically after the cluster deletion event. This feature is off by default. Particularly it is useful to avoid leftovers in testing environments, where the cluster can be re-created and deleted many times. Support for custom sidecar containers. The Operator makes it possible now to deploy additional (*sidecar*) containers to the Pod. This feature can be useful to run debugging tools or some specific monitoring solutions, etc. The sidecar container can be added to replsets, sharding.configsvrReplSet, and sharding.mongos sections of the `deploy/cr.yaml` configuration file.

17.10.2 New Features

- K8SPSMDB-121: Add support for sharding to scale MongoDB cluster horizontally
- K8SPSMDB-294: Support for custom sidecar container to extend the Operator capabilities
- K8SPSMDB-260: Persistent Volume Claims can now be automatically removed after MongoDB cluster deletion

17.10.3 Improvements

- K8SPSMDB-335: Operator can now automatically remove old backups from S3 if retention period is set
- K8SPSMDB-330: Add support for runtimeClassName Kubernetes feature for selecting the container runtime
- K8SPSMDB-306: It is now possible to explicitly set the version of MongoDB for newly provisioned clusters. Before that, all new clusters were started with the latest MongoDB version if Version Service was enabled
- K8SPSMDB-370: Fix confusing log messages about no backup / restore found which were caused by Percona Backup for MongoDB waiting for the backup metadata
- K8SPSMDB-342: MongoDB container liveness probe will now use TLS to follow best practices and remove noisy log messages from mongod log

17.10.4 Bugs Fixed

- K8SPSMDB-346: Fix a bug which prevented adding/removing labels to Pods without downtime
- K8SPSMDB-366: Fix a bug which prevented enabling Percona Monitoring and Management (PMM) due to incorrect request for the recommended PMM Client image version to the Version Service
- K8SPSMDB-402: running multiple replica sets without sharding enabled should be prohibited

- K8SPSMDB-382: Fix a bug which caused mongos process to fail when using `allowUnsafeConfigurations=true`
- K8SPSMDB-362: Fix a bug due to which changing secrets in a single-shard mode caused mongos Pods to fail

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-08

17.11 *Percona Kubernetes Operator for Percona Server for MongoDB* 1.6.0

- **Date**

  December 22, 2020
- **Installation**

  Installing Percona Kubernetes Operator for Percona Server for MongoDB

17.11.1 New Features

- K8SPSMDB-273: Add support for `mongos` service to expose a single shard of a MongoDB cluster through one entry point instead of provisioning a load-balancer per replica set node. In the following release, we will add support for multiple shards.
- K8SPSMDB-282: Official support for Percona Monitoring and Management (PMM) v.2

Note

Monitoring with PMM v.1 configured according to the unofficial instruction will not work after the upgrade. Please switch to PMM v.2.

17.11.2 Improvements

- K8SPSMDB-258: Add support for Percona Server for MongoDB version 4.4
- K8SPSMDB-319: Show Endpoint in the `kubectl get psmdb` command output to connect to a MongoDB cluster easily
- K8SPSMDB-257: Store the Operator version as a `crVersion` field in the `deploy/cr.yaml` configuration file
- K8SPSMDB-266: Use plain-text passwords instead of base64-encoded ones when creating System Users secrets for simplicity

17.11.3 Bugs Fixed

- K8SPSMDB-268: Fix a bug affecting the support of TLS certificates issued by cert-manager, due to which proper rights were not set for the role-based access control, and Kubernetes versions newer than 1.15 required other certificate issuing sources
- K8SPSMDB-261: Fix a bug due to which cluster pause/resume functionality didn't work in previous releases
- K8SPSMDB-292: Fix a bug due to which not all clusters managed by the Operator were upgraded by the automatic update

17.11.4 Removal

- The MMAPv1 storage engine is no longer supported for all MongoDB versions starting from this version of the Operator. MMAPv1 was already deprecated by MongoDB for a long time. WiredTiger is the default storage engine since MongoDB 3.2, and MMAPv1 was completely removed in MongoDB 4.2.

Note

Upgrade of the Operator from 1.5.0 to 1.6.0 will fail if MMAPv1 is used, but MongoDB cluster will continue to run. It is recommended to migrate your clusters to WiredTiger engine before the upgrade.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-12

17.12 *Percona Kubernetes Operator for Percona Server for MongoDB* 1.5.0

- **Date**

  September 7, 2020

- **Installation**

  Installing Percona Kubernetes Operator for Percona Server for MongoDB

17.12.1 New Features

- K8SPSMDB-233: Automatic management of system users for MongoDB on password rotation via Secret
- K8SPSMDB-226: Official Helm chart for the Operator
- K8SPSMDB-199: Support multiple PSMDB minor versions by the Operator
- K8SPSMDB-198: Fully Automate Minor Version Updates (Smart Update)

17.12.2 Improvements

- K8SPSMDB-192: The ability to set the mongod cursorTimeoutMillis parameter in YAML (Thanks to user xprt64 for the contribution)
- K8SPSMDB-234: OpenShift 4.5 support
- K8SPSMDB-197: Additional certificate SANs useful for reverse DNS lookups (Thanks to user phin1x for the contribution)
- K8SPSMDB-190: Direct API quering with “curl” instead of using “kubectl” tool in scheduled backup jobs (Thanks to user phin1x for the contribution)
- K8SPSMDB-133: A special Percona Server for MongoDB debug image which avoids restarting on fail and contains additional tools useful for debugging
- CLOUD-556: Kubernetes 1.17 / Google Kubernetes Engine 1.17 support

17.12.3 Bugs Fixed

- K8SPSMDB-213: Installation instruction not reflecting recent changes in git tags (Thanks to user geraintj for reporting this issue)
- K8SPSMDB-210: Backup documentation not reflecting changes in Percona Backup for MongoDB
- K8SPSMDB-180: Replset and cluster having “ready” status set before mongo initialization and replicasets configuration finished
- K8SPSMDB-179: The “error” cluster status instead of the “initializing” one during the replset initialization
- CLOUD-531: Wrong usage of `strings.TrimLeft` when processing apiVersion

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-08

17.13 *Percona Kubernetes Operator for Percona Server for MongoDB* 1.4.0

- **Date**

  March 31, 2020

- **Installation**

  Installing Percona Kubernetes Operator for PSMDB

17.13.1 New Features

- K8SPSMDB-89: Amazon Elastic Container Service for Kubernetes (EKS) was added to the list of the officially supported platforms
- K8SPSMDB-113: Percona Server for MongoDB 4.2 is now supported
- OpenShift Container Platform 4.3 is now supported

17.13.2 Improvements

- K8SPSMDB-79: The health check algorithm improvements have increased the overall stability of the Operator
- K8SPSMDB-176: The Operator was updated to use Percona Backup for MongoDB version 1.2
- K8SPSMDB-153: Now the user can adjust securityContext, replacing the automatically generated securityContext with the customized one
- K8SPSMDB-175: Operator now updates observedGeneration status message to allow better monitoring of the cluster rollout or backups/restore process

17.13.3 Bugs Fixed

- K8SPSMDB-182: Setting the `updateStrategy: OnDelete` didn't work if was not specified from scratch in CR
- K8SPSMDB-174: 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.

Help us improve our software quality by reporting any bugs you encounter using our bug tracking system.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-08

17.14 *Percona Kubernetes Operator for Percona Server for MongoDB* 1.3.0

Percona announces the *Percona Kubernetes Operator for Percona Server for MongoDB* 1.3.0 release on December 11, 2019. This release is now the current GA release in the 1.3 series. Install the Kubernetes Operator for Percona Server for MongoDB by following the instructions.

The Operator simplifies the deployment and management of the Percona Server for MongoDB 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.

17.14.1 New Features and Improvements

- CLOUD-415: Non-default cluster domain can now be specified with the new `ClusterServiceDNSSuffix` Operator option.
- CLOUD-395: The Percona Server for MongoDB images size decrease by 42% 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 have been provided.

Percona Server for MongoDB is an enhanced, open source and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB Community Edition. It supports MongoDB protocols and drivers. Percona Server for MongoDB extends MongoDB Community Edition functionality by including the Percona Memory Engine, as well as several enterprise-grade features. It requires no changes to MongoDB applications or code.

Help us improve our software quality by reporting any bugs you encounter using our bug tracking system.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-08

17.15 *Percona Kubernetes Operator for Percona Server for MongoDB* 1.2.0

Percona announces the *Percona Kubernetes Operator for Percona Server for MongoDB* 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 Server for MongoDB by following the instructions.

The Operator simplifies the deployment and management of the Percona Server for MongoDB 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.

17.15.1 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.
- Data-at-rest encryption is now enabled by default unless `EnableEncryption=false` is explicitly specified in the `deploy/cr.yaml` configuration file.
- 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.
- The resource constraint values were refined for all containers to eliminate the possibility of an out of memory error.

17.15.2 Fixed Bugs

- Oscillations of the cluster status between “initializing” and “ready” took place after an update.
- The Operator was removing other cron jobs in case of the enabled backups without defined tasks (contributed by Marcel Heers).

Percona Server for MongoDB is an enhanced, open source and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB Community Edition. It supports MongoDB protocols and drivers. Percona Server for MongoDB extends MongoDB Community Edition functionality by including the Percona Memory Engine, as well as several enterprise-grade features. It requires no changes to MongoDB applications or code.

Help us improve our software quality by reporting any bugs you encounter using our bug tracking system.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-08

17.16 *Percona Kubernetes Operator for Percona Server for MongoDB* 1.1.0

Percona announces the general availability of *Percona Kubernetes Operator for Percona Server for MongoDB* 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 Server for MongoDB by following the instructions. Please see the GA release announcement.

The Operator simplifies the deployment and management of the Percona Server for MongoDB 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.

17.16.1 New Features and Improvements

- Now the Percona Kubernetes Operator allows upgrading Percona Server for MongoDB to newer versions, either in semi-automatic or in manual mode.
- Also, two modes are implemented for updating the Percona Server for MongoDB `mongod.conf` configuration file: in *automatic configuration update* mode Percona Server for MongoDB Pods are immediately re-created to populate changed options from the Operator YAML file, while in *manual mode* changes are held until Percona Server for MongoDB Pods are re-created manually.
- Percona Server for MongoDB data-at-rest encryption is now supported by the Operator to ensure that encrypted data files cannot be decrypted by anyone except those with the decryption key.
- 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 the 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 Server for MongoDB is an enhanced, open source and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB Community Edition. It supports MongoDB protocols and drivers. Percona Server for MongoDB extends MongoDB Community Edition functionality by including the Percona Memory Engine, as well as several enterprise-grade features. It requires no changes to MongoDB applications or code.

Help us improve our software quality by reporting any bugs you encounter using our bug tracking system.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-08

17.17 *Percona Kubernetes Operator for Percona Server for MongoDB* 1.0.0

Percona announces the general availability of *Percona Kubernetes Operator for Percona Server for MongoDB* 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 Server for MongoDB 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 Server for MongoDB automates the lifecycle of your Percona Server for MongoDB environment. The Operator can be used to create a Percona Server for MongoDB replica set, or scale an existing replica set.

The Operator creates a Percona Server for MongoDB replica set with the needed settings and provides a consistent Percona Server for MongoDB instance. The Percona Kubernetes Operators are based on best practices for configuration and setup of the Percona Server for MongoDB.

The Kubernetes Operators provide 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 and may save time while providing a consistent environment.

The advantages are the following:

```
* Deploy a Percona Server for MongoDB 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 Server for MongoDB size parameter to add or remove Percona Server for
MongoDB replica set members


* Integrate with Percona Monitoring and Management (PMM) to seamlessly monitor your Percona
Server for MongoDB


* 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 Percona Server for MongoDB replica set member


* 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
```

```
* Supports a replica set Arbiter member


* Supports Percona Server for MongoDB versions 3.6 and 4.0
```

17.17.1 Installation

Installation is performed by following the documentation installation instructions for Kubernetes and OpenShift.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: 2022-08-08