

Documentation

1.13.0 (September 15, 2022)

Percona Technical Documentation Team

Percona LLC and/or its affiliates, © 2009 - 2022

Table of contents

1. Percona Operator for MongoDB	4
2. Requirements	4
3. Quickstart guides	4
4. Advanced installation guides	4
5. Configuration	4
6. Management	5
7. HOWTOs	5
8. Reference	5
9. Requirements	6
9.1 System Requirements	6
9.2 Design overview	7
9.3 Compare various solutions to deploy MongoDB in Kubernetes	9
10. Quickstart guides	12
10.1 Install Percona Server for MongoDB using Helm	12
10.2 Install Percona Server for MongoDB on Minikube	14
11. Advanced installation guide	18
11.1 Install Percona Server for MongoDB on Google Kubernetes Engine (GKE)	18
11.2 Install Percona Server for MongoDB on Amazon Elastic Kubernetes Service (EKS)	24
11.3 Install Percona Server for MongoDB on Azure Kubernetes Service (AKS)	29
11.4 Install the Operator and deploy your MongoDB cluster	29
11.5 Install Percona server for MongoDB on Kubernetes	33
11.6 Install Percona Server for MongoDB on OpenShift	35
11.7 Use Docker images from a custom registry	40
12. Configuration	43
12.1 Users	43
12.2 Changing MongoDB Options	47
12.3 Binding Percona Server for MongoDB components to Specific Kubernetes/OpenShift Nodes	50
12.4 Exposing cluster	53
12.5 Local Storage support for the Percona Operator for MongoDB	56
12.6 Using Replica Set Arbiter nodes and non-voting nodes	57
12.7 Percona Server for MongoDB Sharding	59
12.8 Transport Layer Security (TLS)	61
12.9 Data at rest encryption	65
12.10 Telemetry	70

13. Management	71
13.1 Providing Backups	71
13.2 Update Percona Operator for MongoDB	81
13.3 Scale Percona Server for MongoDB on Kubernetes and OpenShift	88
13.4 Set up Percona Server for MongoDB cross-site replication	89
13.5 Monitoring	95
13.6 Using sidecar containers	100
13.7 Operator pause	104
13.8 Debug	105
14. HOWTOs	106
14.1 How to integrate Percona Operator for MongoDB with OpenLDAP	106
14.2 Creating a private S3-compatible cloud for backups	108
14.3 Install Percona Server for MongoDB in multi-namespace (cluster-wide) mode	112
15. Reference	116
15.1 Custom Resource options	116
15.2 Percona certified images	151
15.3 Percona Operator for MongoDB API Documentation	152
15.4 Frequently Asked Questions	197
16. Release notes	199
16.1 <i>Percona Operator for MongoDB 1.13.0</i>	199
16.2 <i>Percona Operator for MongoDB 1.12.0</i>	201
16.3 <i>Percona Distribution for MongoDB Operator 1.11.0</i>	204
16.4 <i>Percona Distribution for MongoDB Operator 1.10.0</i>	206
16.5 <i>Percona Distribution for MongoDB Operator 1.9.0</i>	208
16.6 <i>Percona Kubernetes Operator for Percona Server for MongoDB 1.8.0</i>	210
16.7 <i>Percona Kubernetes Operator for Percona Server for MongoDB 1.7.0</i>	212
16.8 <i>Percona Kubernetes Operator for Percona Server for MongoDB 1.6.0</i>	214
16.9 <i>Percona Kubernetes Operator for Percona Server for MongoDB 1.5.0</i>	216
16.10 <i>Percona Kubernetes Operator for Percona Server for MongoDB 1.4.0</i>	217
16.11 <i>Percona Kubernetes Operator for Percona Server for MongoDB 1.3.0</i>	218
16.12 <i>Percona Kubernetes Operator for Percona Server for MongoDB 1.2.0</i>	219
16.13 <i>Percona Kubernetes Operator for Percona Server for MongoDB 1.1.0</i>	220
16.14 <i>Percona Kubernetes Operator for Percona Server for MongoDB 1.0.0</i>	221

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. Requirements

- [System Requirements](#)
- [Design and architecture](#)
- [Comparison with other solutions](#)

3. Quickstart guides

- [Install with Helm](#)
- [Install on Minikube](#)

4. Advanced installation guides

- [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](#)
- [Use private registry](#)

5. Configuration

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

6. Management

- [Backup and restore](#)
- [Upgrade MongoDB and the Operator](#)
- [Horizontal and vertical scaling](#)
- [Multi-cluster and multi-region deployment](#)
- [Monitor with Percona Monitoring and Management \(PMM\)](#)
- [Add sidecar containers](#)
- [Restart or pause the cluster](#)
- [Debug and troubleshoot](#)

7. HOWTOs

- [OpenLDAP integration](#)
- [Creating a private S3-compatible cloud for backups](#)
- [Install Percona Server for MongoDB in multi-namespace \(cluster-wide\) mode](#)

8. Reference

- [Custom Resource options](#)
- [Percona certified images](#)
- [Operator API](#)
- [Frequently asked questions](#)
- [Release notes](#)

Last update: 2022-09-15

9. Requirements

9.1 System Requirements

The Operator was developed and tested with Percona Server for MongoDB 4.2, 4.4, and 5.0. Other options may also work but have not been tested.

Note

The [MMAPv1 storage engine](#) is no longer supported for all MongoDB versions starting from the Operator version 1.6. 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.

9.1.1 Officially supported platforms

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
- [Azure Kubernetes Service \(AKS\)](#) 1.22 - 1.24
- [OpenShift Container Platform](#) 4.10 - 4.11
- [Minikube](#) 1.26
- [VMWare Tanzu](#)

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

9.1.2 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.

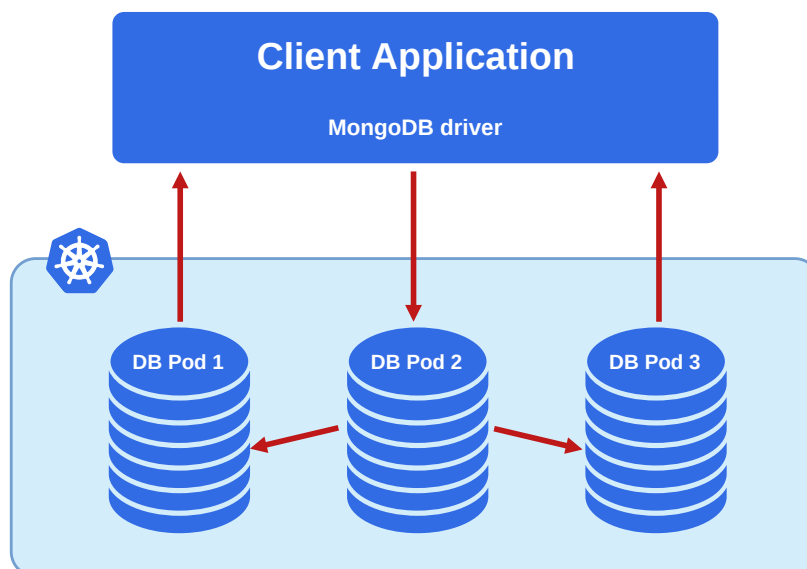
Note

Use Storage Class with XFS as the default filesystem if possible [to achieve better MongoDB performance](#).

Last update: 2022-09-15

9.2 Design overview

The design of the Operator is tightly bound to the Percona Server for MongoDB replica set, which is briefly described in the following diagram.



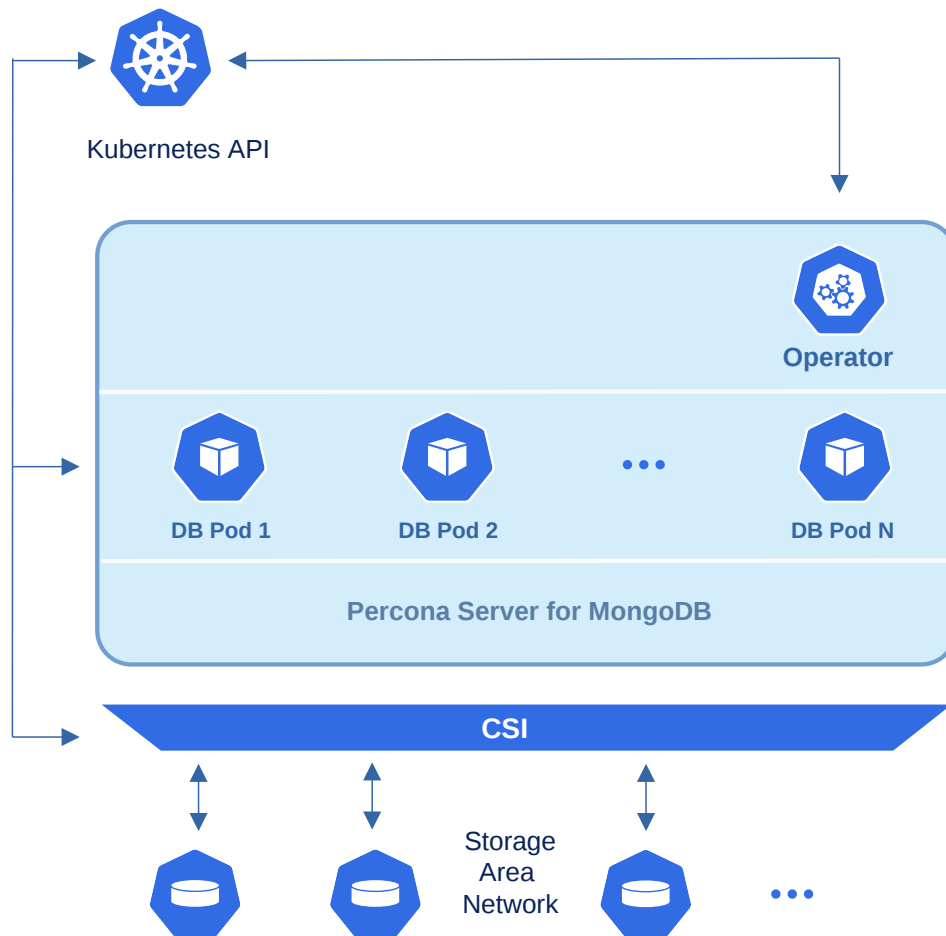
A replica set consists of one primary server and several secondary ones (two in the picture), and the client application accesses the servers via a driver.

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: 2022-09-15

9.3 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](#)

9.3.1 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 Operator
Open source model	Apache 2.0	Apache 2.0	Open core	Open core
MongoDB versions	PSMDB 4.2, 4.4, 5.0	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	No	Not an operator	Enterprise only	Yes
Network exposure	Yes	Yes	No, only through manual config	Enterprise only

9.3.2 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 minor and major
Compute scaling	Horizontal and vertical	Horizontal and vertical	Enterprise only	Horizontal only
Storage scaling	Manual	Manual	Enterprise only	Enterprise only

9.3.3 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

9.3.4 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

9.3.5 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

9.3.6 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

Last update: 2022-08-09

10. Quickstart guides

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](https://github.com/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 `default` namespace. To use different namespace, provide it with the following additional parameter: `--namespace my-namespace`.

3. Install Percona Server for MongoDB:

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

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 --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
```

Last update: 2022-08-18

10.2 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 -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.13.0/deploy/bundle.yaml
```

3. Deploy MongoDB cluster with:

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.13.0/deploy/cr-minimal.yaml
```

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

```
$ git clone -b v1.13.0 https://github.com/percona/percona-server-mongodb-operator
```

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: dXNlcFkKbWlu
```

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:4.4.16-16
--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"
```

Last update: 2022-09-15

11. Advanced installation guide

11.1 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](#).

11.1.1 Prerequisites

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

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

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

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

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

11.1.2 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 1.23 --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:

<input type="checkbox"/>	<input checked="" type="checkbox"/>	my-cluster-name	us-central1-a	3	12	45 GB	—	⋮	<ul style="list-style-type: none"> Edit Connect Delete
--------------------------	-------------------------------------	---------------------------------	---------------	---	----	-------	---	---	---

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
```

11.1.3 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 -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/
v1.13.0/deploy/bundle.yaml
```

Expected output

```
customresourcedefinition.apiextensions.k8s.io/perconaservermongodbs.psmdb.percona.com
serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconaservermongoddbackups.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/v1.13.0/deploy/cr.yaml
```

Expected output

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

Note

This deploys default MongoDB cluster configuration, one mongos node and one config server node. 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 v1.13.0 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

11.1.4 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:4.4.16-16
--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

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

if sharding is off

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

11.1.5 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](#)

11.1.6 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:

<input type="checkbox"/>	<input checked="" type="checkbox"/>	my-cluster-name	us-central1-a	3	12	45 GB	—		 Edit
									 Connect
									 Delete

The cluster deletion may take time.

 **Warning**

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

Last update: 2022-09-15

11.2 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](#).

11.2.1 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](#).

11.2.2 Create the EKS cluster

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 to 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.

The most easy and visually clear way is to describe the desired cluster in YAML and to pass this configuration to the `eksctl` command.

The following example configures a EKS cluster with one **managed node group**:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: test-cluster
  region: eu-west-2

nodeGroups:
  - name: ng-1
    minSize: 3
    maxSize: 5
    instancesDistribution:
      maxPrice: 0.15
      instanceTypes: ["m5.xlarge", "m5.2xlarge"] # At least two instance types should be
specified
    onDemandBaseCapacity: 0
    onDemandPercentageAboveBaseCapacity: 50
```



```
spotInstancePools: 2
tags:
  'iit-billing-tag': 'cloud'
```

When the cluster configuration file is ready, you can actually create your cluster by the following command:

```
$ eksctl create cluster -f ~/cluster.yaml
```

11.2.3 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/v1.13.0/deploy/bundle.yaml
```

Expected output

```
customresourcedefinition.apiextensions.k8s.io/perconaservermongodbs.psmdb.percona.com
serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconaservermongoddbbackups.psmdb.percona.com
serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconaservermongoddbrestores.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/v1.13.0/deploy/cr.yaml
```

Expected output

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

 **Note**

This deploys default MongoDB cluster configuration, one mongos node and one config server node. 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 v1.13.0 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

11.2.4 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:4.4.16-16
--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

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

if sharding is off

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

11.2.5 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.
```

11.2.6 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!

Last update: 2022-09-10

11.3 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](#).

11.3.1 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](#).

11.3.2 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 to 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
```

11.4 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 -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.13.0/deploy/bundle.yaml
```

Expected output

```
customresourcedefinition.apiextensions.k8s.io/perconaservermongodbs.psmdb.percona.com
serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconaservermongoddbbackups.psmdb.percona.com
serverside-applied
customresourcedefinition.apiextensions.k8s.io/perconaservermongoddbrestores.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/v1.13.0/deploy/cr.yaml
```

Expected output

```
perconaservermongodbs.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 v1.13.0 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

11.4.1 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:4.4.16-16
--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

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

if sharding is off

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

11.4.2 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.
```

11.4.3 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!

Last update: 2022-09-15

11.5 Install Percona server for MongoDB on Kubernetes

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

```
$ git clone -b v1.13.0 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 -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:4.4.16-16
--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"
```

Last update: 2022-09-15

11.6 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.

11.6.1 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:

Percona Kubernetes Operator for Percona Server for MongoDB

Free trial

*Requires OpenShift to install

By Percona

The Kubernetes Operator for MongoDB automates the creation, modification, or deletion of items in your Percona Server for MongoDB environment. The Operator is Red Hat OpenShift Certified.

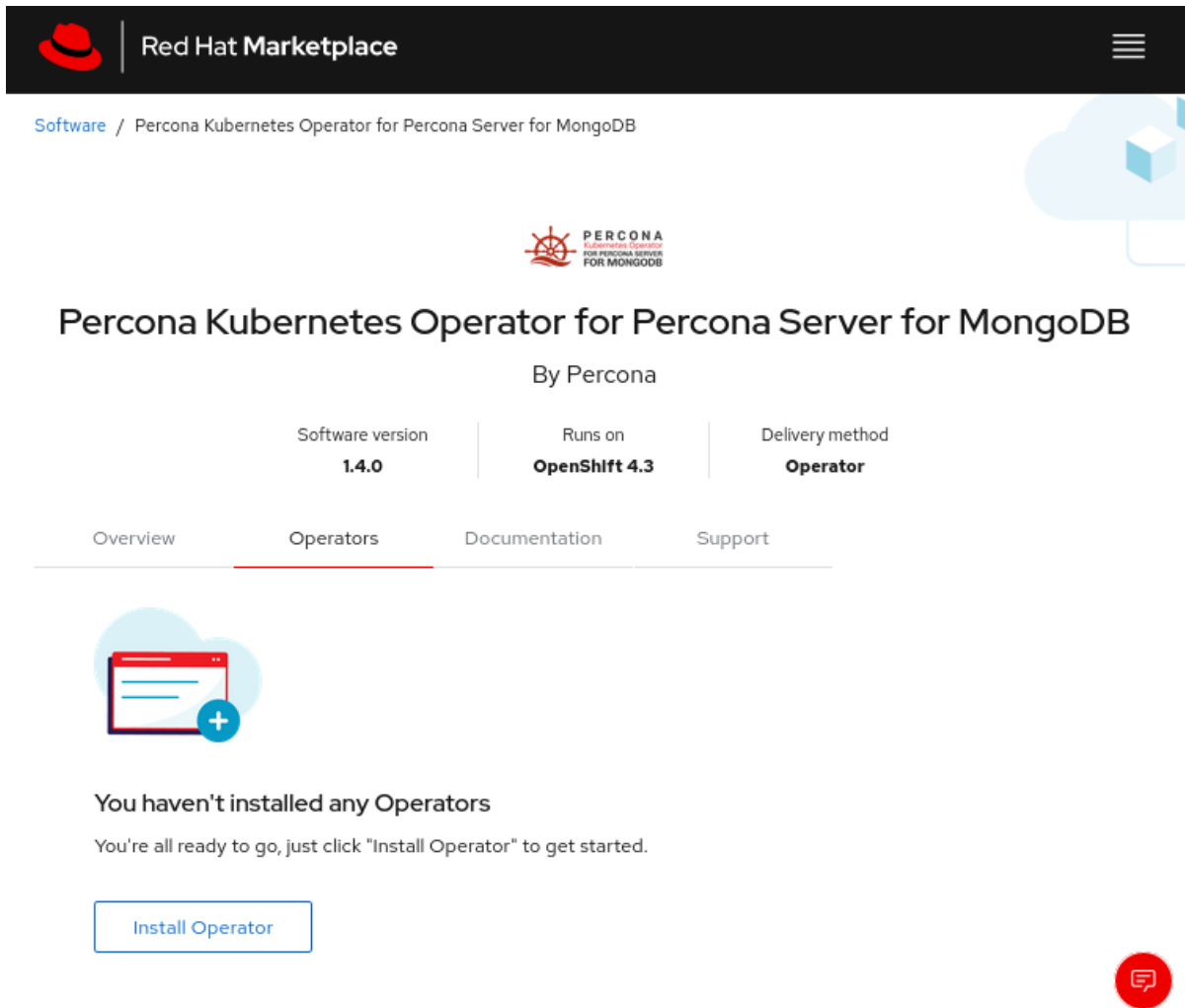
Software version 1.4.0	Runs on OpenShift 4.3	Delivery method Operator	Rating ☆☆☆☆☆ Not rated
----------------------------------	---------------------------------	------------------------------------	---------------------------

- Overview
- Documentation
- Pricing
- Help

Based on our best practices for deployment and configuration, Percona Kubernetes Operator contains everything you need to quickly and consistently deploy and scale Percona Server for MongoDB into a Kubernetes cluster. The Operator enables you to: Improve time to market with the ability to quickly deploy standardized and repeatable database environments. Deploy your database with a consistent and idempotent result no matter where they are used.

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:



Software / Percona Kubernetes Operator for Percona Server for MongoDB

PERCONA
Kubernetes Operator
FOR PERCONA SERVER
FOR MONGODB

Percona Kubernetes Operator for Percona Server for MongoDB

By Percona

Software version	Runs on	Delivery method
1.4.0	OpenShift 4.3	Operator

Overview Operators Documentation Support

You haven't installed any Operators

You're all ready to go, just click "Install Operator" to get started.

[Install 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 v1.13.0 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 -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,perconaservermongoddbackups.psmdb.percona.com,perconaservermongoddbackups.psmdb.percona.c
status,perconaservermongodbrestores.psmdb.percona.com,perconaservermongodbrestores.psmdb.percona
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
```

11.6.2 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:
- 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
...
```

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

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

- 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

- 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:4.4.16-16 --
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"
```

Last update: 2022-09-15

11.7 Use Docker images from a custom registry

Using images from a private Docker registry may be required for privacy, security or other reasons. In these cases, Percona Operator for 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
Authentication required for https://192.168.1.100:8443 (openshift)
Username: admin
Password:
Login successful.
$ oc new-project psmdb
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
AD08CqCDappWR4hxjFDqwijEHei31yXAvWg61Jg210s
```

The following command returns the registry IP address:

```
$ kubectl get services/docker-registry -n default
NAME                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
docker-registry    ClusterIP    172.30.162.173 <none>         5000/TCP    1d
```

3. Use the user token and the registry IP address to login to the registry:

```
$ docker login -u admin -p AD08CqCDappWR4hxjFDqwijEHei31yXAvWg61Jg210s 172.30.162.173:5000
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
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:4.4.16-16
$ docker push 172.30.162.173:5000/psmdb/percona-server-mongodb:4.4.16-16
```

6. Verify the image is available in the OpenShift registry with the following command:

```
$ oc get is
NAME                DOCKER                                TAGS                                UPDATED
REPO
```

```
percona-server-mongodb          docker-registry.default.svc:5000/psmdb/percona-server-  
mongodb 4.4.16-16 2 hours ago
```

- When the custom registry image is available, edit 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:4.4.16-16`)

 **Note**

If the registry requires authentication, you can specify the `imagePullSecrets` option for all images.

- 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.

- Now follow the standard Percona Operator for MongoDB [installation instruction](#).

Last update: 2022-08-18

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:4.4.16-16 --
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:4.4.16-16 --
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.

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

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`

 **Note**

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.

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:

```
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` 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:

```
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.

Last update: 2022-09-15

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.

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:

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

Note

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

```
$ echo "ICAgICAgb3BlcmF0aW9uUHJvZmZsaW5nOgogICAgICAgIG1vZGU6IHNSb3dPc\
AogICAgICBzeXN0ZW1Mb2c6CiAgICAgICAgdmVyYm9zaXR5OXAxCg==" | 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: "ICAgICAgb3BlcmF0aW9uUHJvZmZsaW5nOgogICAgICAgIG1vZGU6IHNSb3dPc\
AogICAgICBzeXN0ZW1Mb2c6CiAgICAgICAgdmVyYm9zaXR5OXAxCg=="
```

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).

Last update: 2022-08-18

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 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 Tolerations

Tolerations allow Pods having them to be able to land onto nodes with matching *taints*. Tolerations are expressed as a `key` with an `operator`, which is either `exists` or `equal` (the `equal` variant requires a corresponding `value` for comparison).

Tolerations 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 Tolerations](#) contains more examples on this topic.

12.3.4 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.

12.3.5 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
```

Last update: 2022-08-18

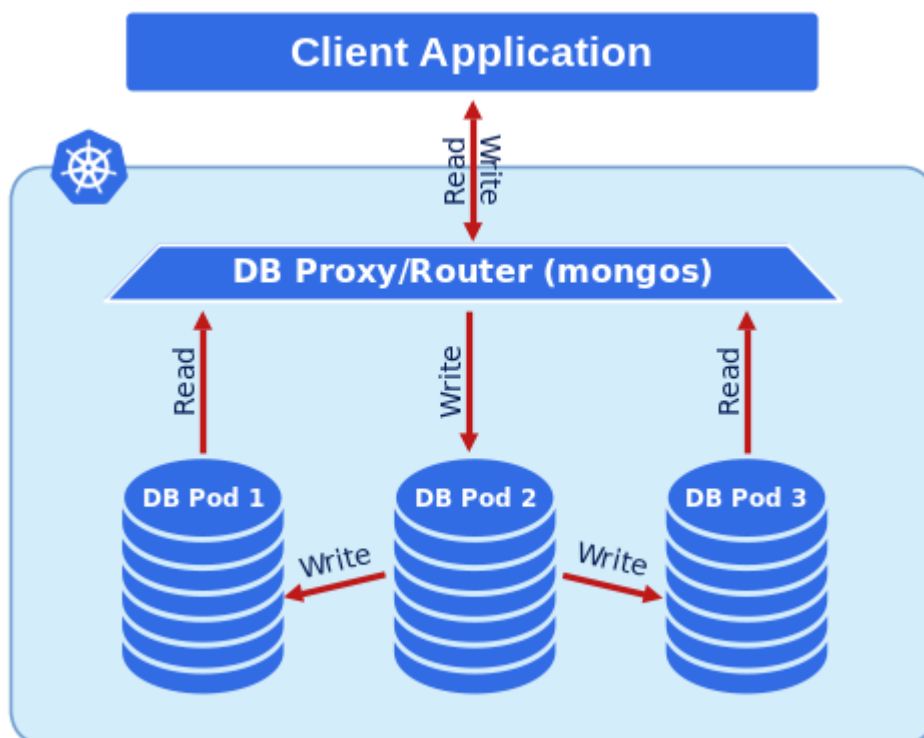
12.4 Exposing cluster

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.

12.4.1 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,



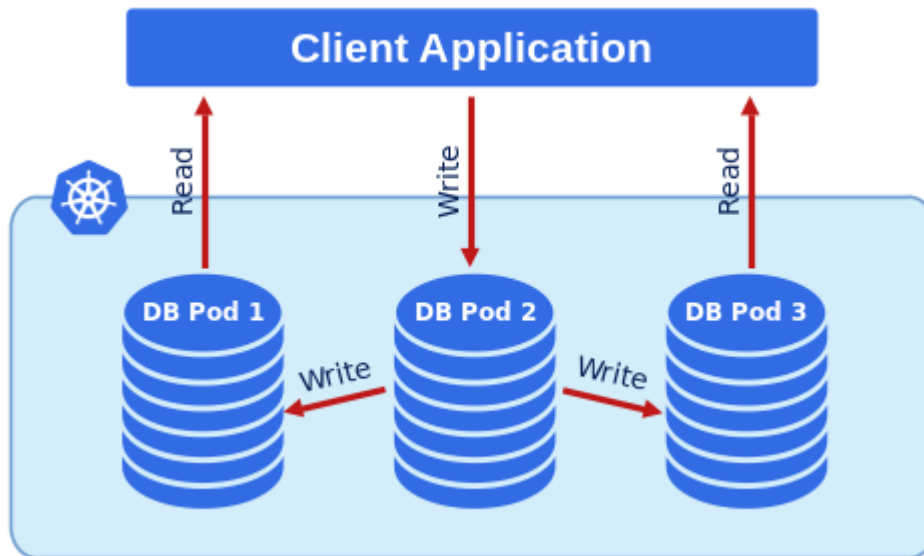
If this feature is enabled, the URI looks like follows (taking into account the need in a proper password obtained from the Secret, and a proper namespace name instead of the `<namespace name>` placeholder):

```
$ 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](#).

12.4.2 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):

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

12.4.3 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 MongoDB 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 addresses should be *directly* reachable by application.

Last update: 2022-09-15

12.5 Local Storage support for the Percona Operator for MongoDB

Among the wide range 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*.

12.5.1 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](#).

12.5.2 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 create 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](#).

Last update: 2022-08-18

12.6 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.

12.6.1 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](#).

12.6.2 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](#).

Last update: 2022-08-18

12.7 Percona Server for MongoDB Sharding

12.7.1 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.

12.7.2 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.

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.

12.7.3 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:4.4.16-16
--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

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

if sharding is off

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

Last update: 2022-09-01

12.8 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.

12.8.1 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/v1.6.1/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-tmjqg	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.

12.8.2 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.

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": [
    {
      "0": "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": [
    {
      "0": "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": [
    {
      "0": "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

```

12.8.3 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-ca	True	61s

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
```

12.8.4 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 make sure that there are no certificate secrets available.

Last update: 2022-08-18

12.9 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.

12.9.1 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:

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

12.9.2 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.VgQvaXl8xGF01RUxAPbPbsfN
```

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

```
$ kubectl exec -it vault-0 -- /bin/sh
$ vault login s.VgQvaXl8xGF01RUxAPbPbsfN
```

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.VgQvaXl8xGF01RUxAPbPbsfN |
| 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

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

Note

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

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

Expected output

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

- 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.VgQvaXl8xGF01RUxAPbPbsfN"
```

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.VgQvaXl8xGF01RUxAPbPbsfN" --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](#).

Last update: 2022-09-15

12.10 Telemetry

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

The following information is gathered:

- ID of the Custom Resource (the `metadata.uid` field)
- Kubernetes version
- Platform (is it Kubernetes or Openshift)
- PMM Version
- Operator version
- 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 Operator deployed in a cluster-wide mode (starting from the Operator version 1.13)

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 1.13.0 --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`:

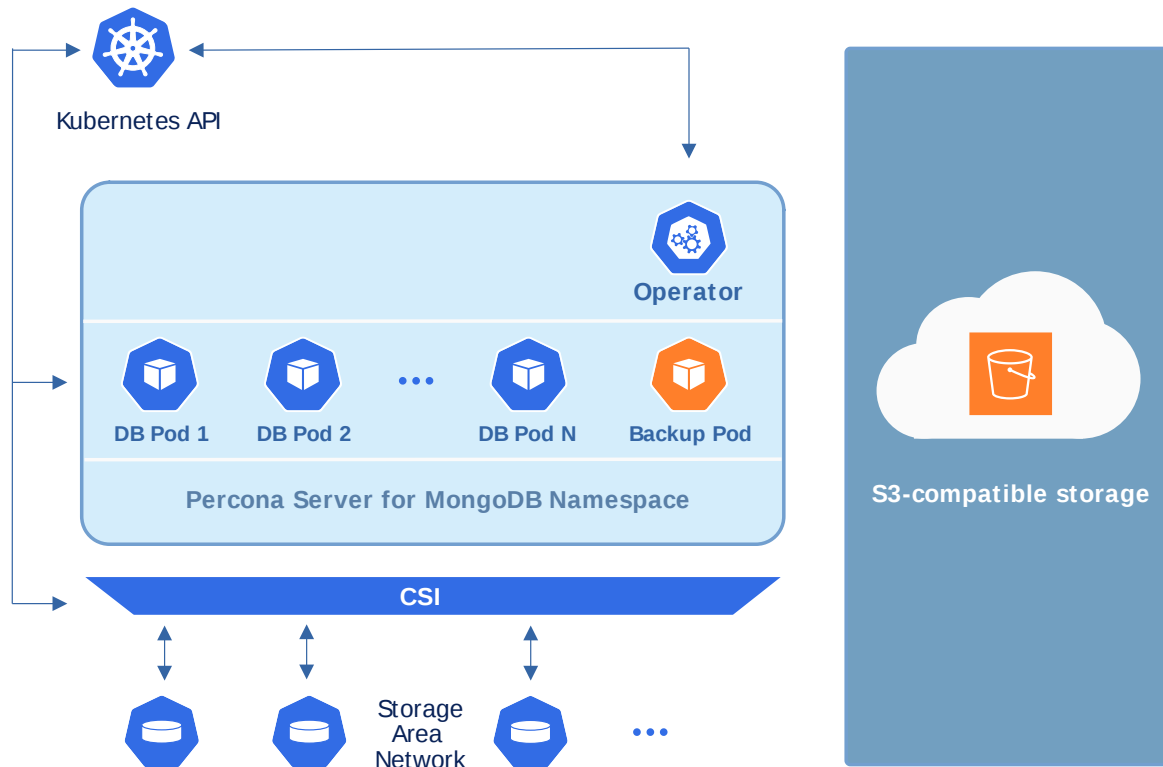
```
env:
  ...
  - name: DISABLE_TELEMETRY
    value: "true"
  ...
```

Last update: 2022-09-01

13. Management

13.1 Providing 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.



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. Both ways use the Percona Backup for MongoDB tool.

Warning

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**.

13.1.1 Making scheduled backups

Backups schedule is defined in the `backup` section of the `deploy/cr.yaml` file. This section contains `backup.enabled` key (it should be set to `true` to enable backups), and the following subsections:

- `storages` subsection contains data needed to access the S3-compatible cloud to store backups,
- `tasks` subsection allows to actually schedule backups (the schedule is specified in `crontab` format).

Backups on Amazon S3 or S3-compatible storage

Since backups are stored separately on the Amazon S3, a secret with `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` should be present on the Kubernetes cluster. The secrets file with these base64-encoded keys should be created: for example `deploy/backup-s3.yaml` file with the following contents.

```
apiVersion: v1
kind: Secret
metadata:
  name: my-cluster-name-backup-s3
type: Opaque
data:
  AWS_ACCESS_KEY_ID: UkVQTEFDRS1XSVRILUFxUy1BQ0NFU1MtS0VZ
  AWS_SECRET_ACCESS_KEY: UKVQTEFDRS1XSVRILUFxUy1TRUNSRVQtS0VZ
```

Note

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

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

All the data needed to access the S3-compatible cloud to store backups should be put into the `backup.storages` subsection, and `backup.tasks` subsection should actually schedule backups in `crontab`-compatible way. Here is an example of `deploy/cr.yaml` which uses Amazon S3 storage for backups:

```
...
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"
    schedule: "0 0 * * 6"
    keep: 3
    storageName: s3-us-west
  ...
```




Using AWS EC2 instances for backups makes it possible to automate access to AWS S3 buckets based on IAM roles for Service Accounts with no need to specify the S3 credentials explicitly.

Following steps are needed to turn this feature on:

- Create the IAM instance profile and the permission policy within where you specify the access level that grants the access to S3 buckets.
- Attach the IAM profile to an EC2 instance.
- Configure an S3 storage bucket and verify the connection from the EC2 instance to it.
- Do not provide `s3.credentialsSecret` for the storage in `deploy/cr.yaml`.

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

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

Also you can use `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.

The options within these three subsections are further explained in the [Operator Custom Resource options](#).

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

Backups on Microsoft Azure Blob storage

Since backups are stored separately on [Azure Blob Storage](#), a secret with `AZURE_STORAGE_ACCOUNT_NAME` and `AZURE_STORAGE_ACCOUNT_KEY` should be present on the Kubernetes cluster. The secrets file with these base64-encoded keys should be created: for example `deploy/backup-azure.yaml` file with the following contents.

```
apiVersion: v1
kind: Secret
metadata:
  name: my-cluster-azure-secret
type: Opaque
data:
  AZURE_STORAGE_ACCOUNT_NAME: UkVQTEFDRS1XSVRILUFXUy1BQ0NFU1MtS0VZ
  AZURE_STORAGE_ACCOUNT_KEY: UkVQTEFDRS1XSVRILUFXUy1TRUNSRVQtS0VZ
```



Note

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

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

All the data needed to access the Azure Blob storage to store backups should be put into the `backup.storages` subsection, and `backup.tasks` subsection should actually schedule backups in crontab-compatible way. Here is an example of `deploy/cr.yaml` which uses Azure Blob storage for backups:

```
...
backup:
  enabled: true
  ...
  storages:
    azure-blob:
      type: azure
      azure:
        container: <your-container-name>
        prefix: psmdb
        credentialsSecret: my-cluster-azure-secret

...
tasks:
- name: "sat-night-backup"
  schedule: "0 0 * * 6"
  keep: 3
  storageName: azure-blob
...
```

The options within these three subsections are further explained in the [Operator Custom Resource options](#).

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

You can use `prefix` option to specify the path (sub-folder) to the backups inside the container. If prefix is not set, backups will be stored in the root directory of the container.

13.1.2 Making on-demand backup

To make an on-demand backup, the user should first make changes in the `deploy/cr.yaml` configuration file: set the `backup.enabled` key to `true` and configure backup storage in the `backup.storages` subsection in a same way it was done for scheduled backups. When the `deploy/cr.yaml` file contains correctly configured keys and is applied with `kubectl` command, use *a special backup configuration YAML file* with the following contents:

- **backup name** in the `metadata.name` key,
- **Percona Server for MongoDB Cluster name** in the `clusterName` key (prior to the Operator version 1.12.0 this key was named `spec.psmdbCluster`),
- **storage name** from `deploy/cr.yaml` in the `spec.storageName` key.

The example of such file is `deploy/backup/backup.yaml`.

When the backup destination is configured and applied with `kubectl apply -f deploy/cr.yaml` command, the actual backup command is executed:

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

 Note

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

```
$ cat <<EOF | kubectl apply -f-
apiVersion: psmdb.percona.com/v1
kind: PerconaServerMongoDBBackup
metadata:
  name: backup1
spec:
  clusterName: my-cluster-name
  storageName: s3-us-west
EOF
```

13.1.3 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 S3-compatible backup storage.

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

```
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 where full backup is stored. This makes point-in-time recovery functionality available only if there is a single bucket 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 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.

13.1.4 Restore the cluster from a previously saved backup

Backup can be restored not only on the Kubernetes cluster where it was made, but also on any Kubernetes-based environment with the installed Operator.

 **Note**

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).

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
```

 **Note**

Obviously, you can make this check only on the same cluster on which you have previously made the backup.

And the following command will list available clusters:

```
$ kubectl get psmdb
```

Restoring without point-in-time recovery

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

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,
- if you are restoring backup on the *same* Kubernetes-based cluster you have used to save this backup, set `spec.backupName` key to the name of your backup,
- if you are restoring backup on the Kubernetes-based cluster *different* from one you have used to save this backup, set `spec.backupSource` subsection instead of `spec.backupName` field to point on the appropriate S3-compatible storage. This `backupSource` subsection should contain a `destination` key, followed by necessary storage configuration keys, same as in `deploy/cr.yaml` file:

```
...
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
```

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
```

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
```

Restoring backup with point-in-time recovery

Following steps are needed to roll back the cluster to a specific date and time:

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:

```
...
spec:
  clusterName: my-cluster-name
  pitr:
    type: date
    date: YYYY-MM-DD hh:mm:ss
```

- if you are restoring backup on the *same* Kubernetes-based cluster you have used to save this backup, set `spec.backupName` key to the name of your backup,
- if you are restoring backup on the Kubernetes-based cluster *different* from one you have used to save this backup, set `spec.backupSource` subsection instead of `spec.backupName` field to point on the appropriate 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:

```
...
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
```

- you can also use a `storageName` key to specify the exact name of the storage (the actual storage should be already defined in the `backup.storages` subsection of the `deploy/cr.yaml` file):

```
...
storageName: s3-us-west
backupSource:
  destination: s3://S3-BUCKET-NAME/BACKUP-NAME
```

2. Run the actual restoration process:

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

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
  pitr:
    type: date
    date: YYYY-MM-DD hh:mm:ss
EOF
```

13.1.5 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>
```

Last update: 2022-08-18

13.2 Update Percona Operator for MongoDB

Starting from the version 1.1.0 the Percona Operator for MongoDB allows upgrades to newer versions. This includes upgrades of the Operator itself, and upgrades of the Percona Server for MongoDB.

13.2.1 Upgrading the Operator

This upgrade can be done either in semi-automatic or in manual mode. **Manual update mode is the recommended way for a production cluster.**

 **Note**

Operational support is provided for the last 3 minor versions of the Operator. Customers will get complete support for the latest minor version. Bug fixes and improvements are not backported to older minor versions.

Semi-automatic upgrade

 Note

Only the incremental update to a nearest minor version 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.


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

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.13.0/deploy/crd.yaml
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.13.0/deploy/rbac.yaml
```

2. Edit the `deploy/cr.yaml` file, setting `updateStrategy` key to `RollingUpdate`, and apply changes with the `kubectl apply -f deploy/cr.yaml` command.
3. Now you should [apply a patch](#) to your deployment, supplying necessary image names with a newer version tag. This is done with the `kubectl patch deployment` command. For example, updating to the `1.13.0` 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:1.13.0"}]}}}}'

$ kubectl patch psmdb my-cluster-name --type=merge --patch '{
  "spec": {
    "crVersion":"1.13.0",
    "image": "percona/percona-server-mongodb:4.4.16-16",
    "backup": { "image": "percona/percona-backup-mongodb:1.8.1" },
    "pmm": { "image": "percona/pmm-client:2.30.0" }
  }}'
```

 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":"1.13.0",
    "image": "percona/percona-server-mongodb:4.4.16-16",
    "backup": { "image": "percona/percona-backup-mongodb:1.8.1" }
  }}'
```

4. 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
```

Manual upgrade

 **Note**

Only the incremental update to a nearest minor 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.

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

```
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.13.0/deploy/crd.yaml
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.13.0/deploy/rbac.yaml
```

2. Edit the `deploy/cr.yaml` file, setting `updateStrategy` key to `OnDelete`, and apply changes with the `kubectl apply -f deploy/cr.yaml` command.
3. Now you should [apply a patch](#) to your deployment, supplying necessary image names with a newer version tag. This is done with the `kubectl patch deployment` command. For example, updating to the `1.13.0` 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:1.13.0"}]}}}}'

$ kubectl patch psmdb my-cluster-name --type=merge --patch '{
  "spec": {
    "crVersion": "1.13.0",
    "image": "percona/percona-server-mongodb:4.4.16-16",
    "backup": { "image": "percona/percona-backup-mongodb:1.8.1" },
    "pmm": { "image": "percona/pmm-client:2.30.0" }
  }}'
```

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:

```
```bash $ kubectl patch psmdb my-cluster-name --type=merge --patch '{
```

```
"spec": { "crVersion": "1.13.0", "image": "percona/percona-server-mongodb:4.4.16-16", "backup": { "image": "percona/percona-backup-mongodb:1.8.1" } } }```
```

4. 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

5. 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).

## 13.2.2 Upgrading Percona Server for MongoDB

Starting from version 1.5.0, the Operator can do fully automatic upgrades to the newer versions of Percona Server for MongoDB within the method named *Smart Updates*.

To have this upgrade method enabled, make sure that the `updateStrategy` key in the `deploy/cr.yaml` configuration file is set to `SmartUpdate`, and apply changes with the `kubectl apply -f deploy/cr.yaml` command.

When automatic updates are enabled, the Operator will carry on upgrades according to the following algorithm. It will query a special *Version Service* server at scheduled times to obtain fresh information about version numbers and valid image paths needed for the upgrade. If the current version should be upgraded, the Operator updates the CR to reflect the new image paths and carries on sequential Pods deletion in a safe order, allowing StatefulSet to redeploy the cluster Pods with the new image.

### Note

Being enabled, Smart Update will force the Operator to take MongoDB version from Version Service and not from the `mongod.image` option during the very first start of the cluster.

The upgrade details are set in the `upgradeOptions` section of the `deploy/cr.yaml` configuration file. Make the following edits to configure updates:

1. Set the `apply` option 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 5.0 version will be selected instead of the Percona Server for MongoDB 4.4 or 4.2 version regardless of the image path; for already existing clusters, the 5.0 vs. 4.4 or 4.2 branch choice will be preserved),
- `5.0-recommended`, `4.4-recommended`, `4.2-recommended` - same as above, but preserves specific major MongoDB version for newly provisioned clusters (ex. 5.0 will not be automatically used instead of 4.4),
- `Latest` - automatic upgrade will choose the most recent version of the software available (for clusters created from scratch, the Percona Server for MongoDB 5.0 version will be selected instead of the Percona Server for MongoDB 4.4 or 4.2 version regardless of the image path; for already existing clusters, the 5.0 vs. 4.4 or 4.2 branch choice will be preserved),
- `5.0-latest`, `4.4-latest`, `4.2-latest` - same as above, but preserves specific major MongoDB version for newly provisioned clusters (ex. 5.0 will not be automatically used instead of 4.4),
- `version number` - specify the desired version explicitly (version numbers are specified as 4.4.16-16, 4.2.22-22, etc.),
- `Never` or `Disabled` - disable automatic upgrades.

#### 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.

2. Make sure the `versionServiceEndpoint` key is set to a valid Version Service URL (otherwise Smart Updates will not occur).

- a. You can use the URL of the official Percona's Version Service (default). Set `versionServiceEndpoint` to `https://check.percona.com`.
- b. 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. If automatic updates are enabled, but Version Service URL can not be reached, upgrades will not occur.

3. Use the `schedule` option to specify the update checks time in CRON format.

4. Don't forget to apply changes with the `kubectl apply -f deploy/cr.yaml` command.

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 * * *"
 ...
```

## Percona Server for MongoDB major version upgrades

Normally automatic upgrade takes place within minor versions (for example, from `4.2.11-12` to `4.2.12-13`) of MongoDB. Major versions upgrade (for example moving from `4.2-recommended` to `4.4-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 4.2 to 5.0, you should first upgrade it to 4.4, and later make a separate upgrade from 4.4 to 5.0. 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 4.4 to 4.2), 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.

---

Last update: 2022-09-15

## 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.

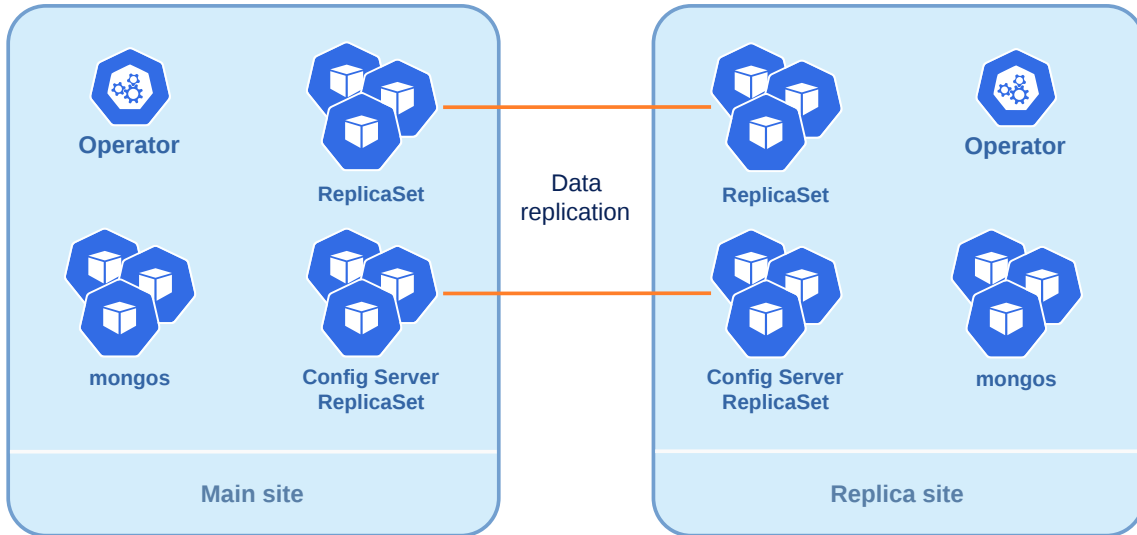
---

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.

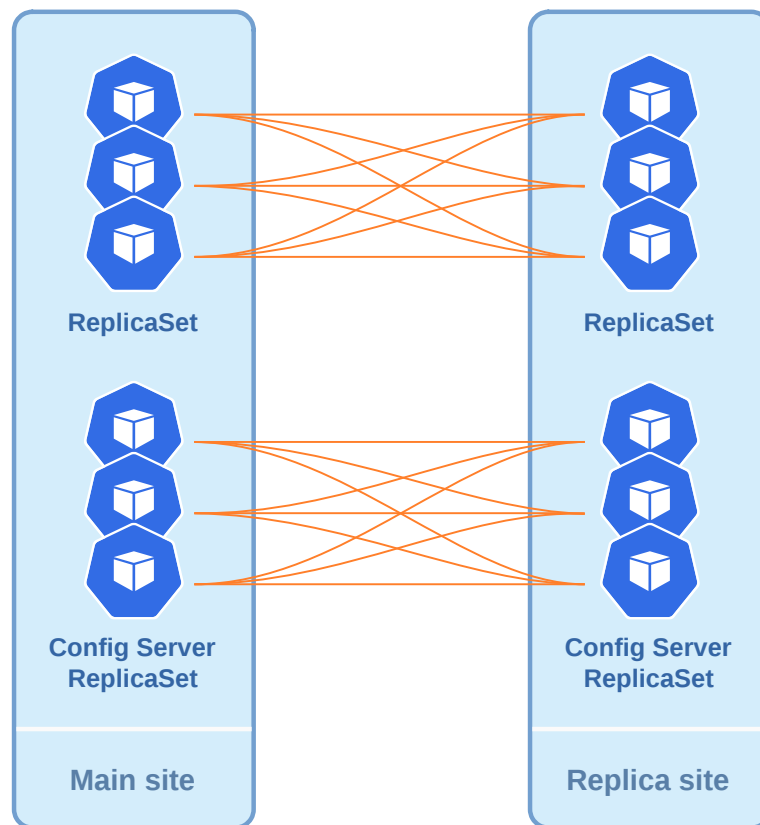
This feature can be useful in several cases:

- 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:



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),

Optionally you can set the following additional keys:

- `priority` key sets the `priority` of the external node ( `2` by 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`),
- `votes` key sets the number of `votes` an external node can cast in a replica set election ( `0` by default, and `0` for non-voting members of the cluster).

Here is an example:

```
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:

```
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 \(EKS\)](#).

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 happened:

- `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.

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 serviceimport
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

$ kubectl get serviceexport
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
```

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](#).

---

Last update: 2022-09-15

## 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.
- 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.

#### 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` in your local shell to get valid values. For example, setting the PMM Server user's password to `new_password` in the `my-cluster-name-secrets` object can be done with the following command:

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

```
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 mongod` 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 official `pmm-admin add mongod` [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
```

---

Last update: 2022-09-15

## 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:

```
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
....
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
 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:

```
...
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:

```
...
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.

Last update: 2022-08-18

## 13.7 Operator pause

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`.

---

Last update: 2022-09-01



## 13.8 Debug

### 13.8.1 Using debug image

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:5.0.11-10-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.

### 13.8.2 Changing logs representation

You can also change the representation of logs: either use structured representation, which produces a parsing-friendly JSON, or use traditional console-friendly logging with specific level. Changing representation of logs is possible by editing the `deploy/operator.yaml` file, which sets the following environment variables with self-speaking names and values:

```
env:
 ...
 name: LOG_STRUCTURED
 value: 'false'
 name: LOG_LEVEL
 value: INFO
 ...
```

---

Last update: 2022-09-15

## 14. HOWTOs

### 14.1 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.

#### 14.1.1 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
```

Also a read-only user should be created for database-issued user lookups. If everything is done correctly, the following command should work

```
$ ldappasswd -s percona -D "cn=admin,dc=ldap,dc=local" -w password -x
"uid=percona,ou=perconadba,dc=ldap,dc=local"
```

#### 14.1.2 The MongoDB and Operator side

In order to get MongoDB connected with OpenLDAP we need to configure both:

- Mongod
- Internal mongod role

As for mongod you may use the following code snippet:

```
security:
 authorization: "enabled"
```

```

ldap:
 authz:
 queryTemplate: 'ou=perconadba,dc=ldap,dc=local??sub?(&(objectClass=group)(uid={USER}))'
 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'

```

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 my-cluster-name-rs0-mongod --from-file=mongod.conf=<path-to-mongod-ldap-configuration>
```

#### Note

LDAP over TLS is not yet supported by the Operator.

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 LDAP user roles inside the MongoDB. For this, log into MongoDB as administrator and execute the following:

```

var admin = db.getSiblingDB("admin")
admin.createRole(
 {
 role: "ou=perconadba,dc=ldap,dc=local",
 privileges: [],
 roles: ["userAdminAnyDatabase"]
 }
)

```

Now the new `percona` user created inside OpenLDAP is able to login to MongoDB as administrator. You can check this with the following command:

```
$ mongo --username percona --password 'percona' --authenticationMechanism 'PLAIN' --
authenticationDatabase '$external' --host <mongodb-rs-endpoint> --port 27017
```

---

Last update: 2022-08-18

## 14.2 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 \
 --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:

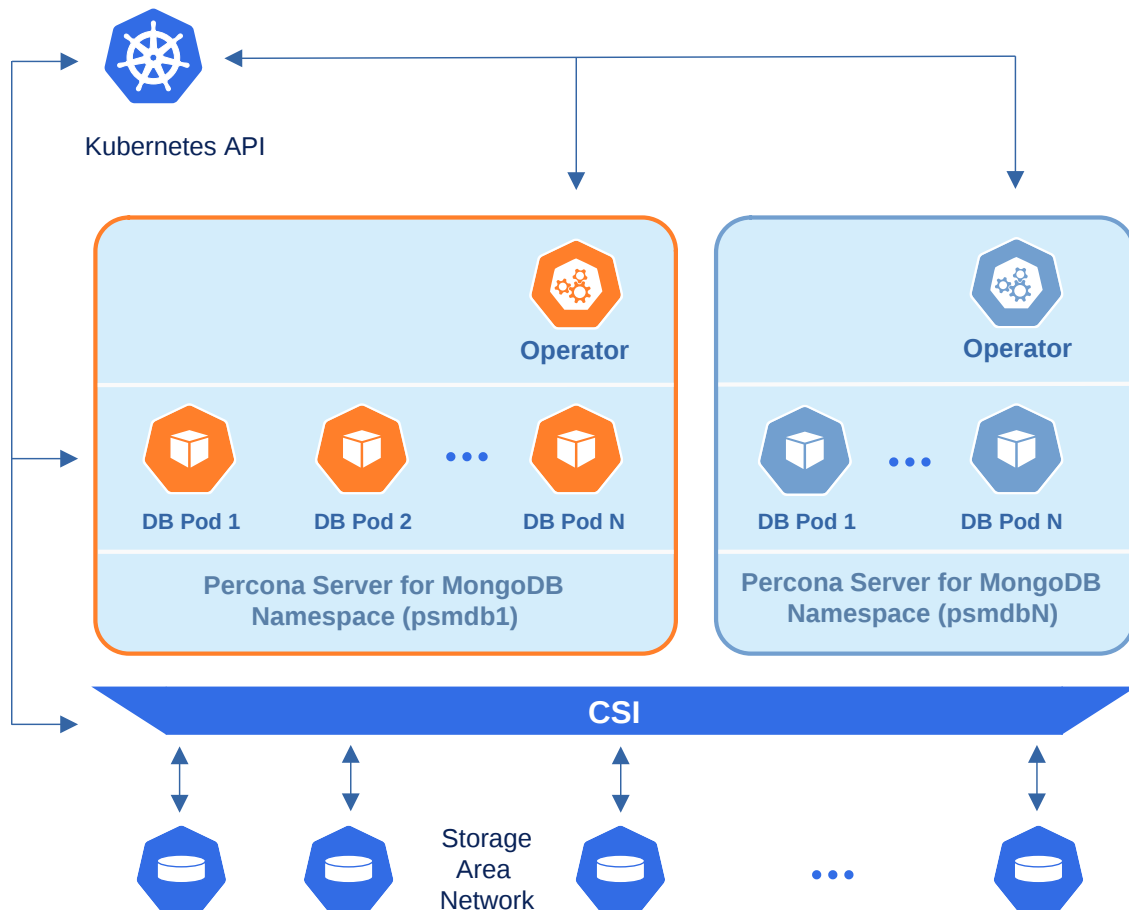
```
$ kubectl run -it --rm pbmctl --image=percona/percona-server-mongodb-operator:0.3.0-backup-
pbmctl --restart=Never -- \
 run restore \
 --server-address=<cluster-name>-backup-coordinator:10001 \
 --storage <storage> \
 backup-name
```

---

Last update: 2022-09-01

## 14.3 Install Percona Server for MongoDB in multi-namespace (cluster-wide) mode

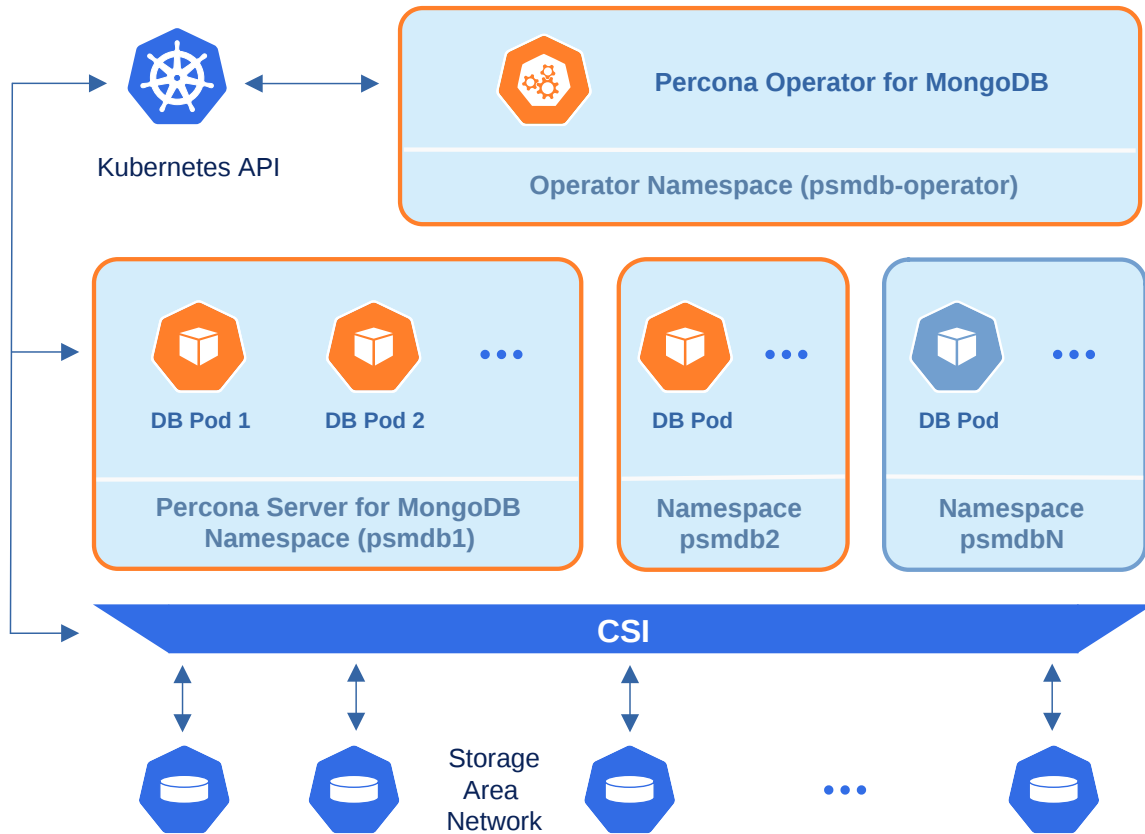
By default, Percona Operator for MongoDB functions in a specific Kubernetes namespace. You can create one during installation (like it is shown in the [installation instructions](#)) or just use the `default` namespace. This approach allows several Operators to co-exist in one Kubernetes-based environment, 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 v1.13.0 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

### 14.3.1 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:5.0.11-10
--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: 2022-09-15

## 15. Reference

### 15.1 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-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: <i>kubernetes</i> or <i>openshift</i>
pause	boolean	false	Pause/resume: setting it to <code>true</code> gracefully stops the cluster, and setting it to <code>false</code> after shut down starts the cluster back.
unmanaged	boolean	false	Unmanaged site in <a href="#">cross-site replication</a> : setting it to <code>true</code> forces the Operator to run the cluster in unmanaged state - nodes do not form replica sets, operator does not control TLS certificates
crVersion	string	1.13.0	Version of the Operator the Custom Resource belongs to
image	string	percona/percona-server-mongodb:4.4.16-16	The Docker image of <a href="#">Percona Server for MongoDB</a> to deploy (actual image names can be found in <a href="#">the list of certified images</a> )
imagePullPolicy	string	Always	The <a href="#">policy used to update images</a>
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 <a href="#">Kubernetes ImagePullSecret</a> to access the <a href="#">custom registry</a>
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 either <code>internal</code> (exposed MongoDB instances will use ClusterIP addresses) or <code>ServiceMesh</code> (turns on for the exposed Services). Being set, <code>ServiceMesh</code> value supersedes <code>multiCluster</code> settings, and therefore these two modes cannot be combined together.

Key	Value type	Default	Description
allowUnsafeConfigurations	boolean	<code>false</code>	Prevents users from configuring a cluster with unsafe parameters: starting it with less than 3 replica set instances, with an <a href="#">even number of replica set instances without additional arbiter</a> , or without TLS/SSL certificates, or running a sharded cluster with less than 3 config server Pods or less than 2 mongos Pods (if <code>false</code> , the Operator will automatically change unsafe parameters to safe defaults)
updateStrategy	string	<code>SmartUpdate</code>	A strategy the Operator uses for <a href="#">upgrades</a> . Possible values are <code>SmartUpdate</code> , <code>RollingUpdate</code> and <code>OnDelete</code>
multiCluster.enabled	boolean	<code>false</code>	<a href="#">Multi-cluster Services (MCS)</a> : setting it to <code>true</code> enables <a href="#">MCS cluster mode</a>
multiCluster.DNSSuffix	string	<code>svc.cluster.set.local</code>	The cluster domain to be used as a suffix for <a href="#">multi-cluster Services</a> used by Kubernetes ( <code>svc.cluster.set.local</code> by default)
upgradeOptions	<a href="#">subdoc</a>		Upgrade configuration section
secrets	<a href="#">subdoc</a>		Operator secrets section
replsets	<a href="#">subdoc</a>		Operator MongoDB Replica Set section
pmm	<a href="#">subdoc</a>		Percona Monitoring and Management section
sharding	<a href="#">subdoc</a>		MongoDB sharding configuration section
mongod	<a href="#">subdoc</a>		Operator MongoDB Mongod configuration section
backup	<a href="#">subdoc</a>		Percona Server for MongoDB backups section

### 15.1.1 Upgrade Options Section

The `upgradeOptions` section in the `deploy/cr.yaml` file contains various configuration options to control Percona Server for MongoDB upgrades.

<b>Key</b>	<code>upgradeOptions.versionServiceEndpoint</code>
<b>Value</b>	string
<b>Example</b>	<code>https://check.percona.com</code>
<b>Description</b>	The Version Service URL used to check versions compatibility for upgrade
<b>Key</b>	<code>upgradeOptions.apply</code>
<b>Value</b>	string
<b>Example</b>	<code>disabled</code>
<b>Description</b>	Specifies how updates are processed by the Operator. <code>Never</code> or <code>Disabled</code> will completely disable automatic upgrades, otherwise it can be set to <code>Latest</code> or <code>Recommended</code> or to a specific version string of Percona Server for MongoDB (e.g. <code>5.0.11-10</code> ) that is wished to be version-locked (so that the user can control the version running, but use automatic upgrades to move between them)
<b>Key</b>	<code>upgradeOptions.schedule</code>
<b>Value</b>	string
<b>Example</b>	<code>0 2 \* \* \*</code>
<b>Description</b>	Scheduled time to check for updates, specified in the <code>crontab</code> format
<b>Key</b>	<code>upgradeOptions.setFCV</code>
<b>Value</b>	boolean
<b>Example</b>	<code>false</code>
<b>Description</b>	If enabled, <code>FeatureCompatibilityVersion</code> (FCV) will be set to match the version during major version upgrade



### 15.1.2 Secrets section

Each spec in its turn may contain some key-value pairs. The secrets one has only two of them:

<b>Key</b>	<code>secrets.key</code>
<b>Value</b>	string
<b>Example</b>	<code>my-cluster-name-mongodb-key</code>
<b>Description</b>	The secret name for the <a href="#">MongoDB Internal Auth Key</a> . This secret is auto-created by the operator if it doesn't exist.
<b>Key</b>	<code>secrets.users</code>
<b>Value</b>	string
<b>Example</b>	<code>my-cluster-name-mongodb-users</code>
<b>Description</b>	The name of the Secrets object for the MongoDB users <b>required to run the operator</b> .
<b>Key</b>	<code>secrets.ssl</code>
<b>Value</b>	string
<b>Example</b>	<code>my-custom-ssl</code>
<b>Description</b>	A secret with TLS certificate generated for <i>external</i> communications, see <a href="#">Transport Layer Security (TLS)</a> for details
<b>Key</b>	<code>secrets.sslInternal</code>
<b>Value</b>	string
<b>Example</b>	<code>my-custom-ssl-internal</code>
<b>Description</b>	A secret with TLS certificate generated for <i>internal</i> communications, see <a href="#">Transport Layer Security (TLS)</a> for details
<b>Key</b>	<code>secrets.encryptionKey</code>
<b>Value</b>	string
<b>Example</b>	<code>my-cluster-name-mongodb-encryption-key</code>
<b>Description</b>	Specifies a secret object with the <a href="#">encryption key</a>
<b>Key</b>	<code>secrets.vault</code>
<b>Value</b>	string
<b>Example</b>	<code>my-cluster-name-vault</code>
<b>Description</b>	Specifies a secret object to <a href="#">provide integration with HashiCorp Vault</a>

### 15.1.3 Replsets Section

The replsets section controls the MongoDB Replica Set.

<b>Key</b>	<code>replsets.name</code>
<b>Value</b>	string
<b>Example</b>	<code>rs 0</code>
<b>Description</b>	The name of the <a href="#">MongoDB Replica Set</a>
<b>Key</b>	<code>replsets.size</code>
<b>Value</b>	int
<b>Example</b>	3
<b>Description</b>	The size of the MongoDB Replica Set, must be $\geq 3$ for <a href="#">High-Availability</a>
<b>Key</b>	<code>replsets.configuration</code>
<b>Value</b>	string
<b>Example</b>	<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>
<b>Description</b>	Custom configuration options for mongod. Please refer to the <a href="#">official manual</a> for the full list of options, and <a href="#">specific Percona Server for MongoDB docs</a> .
<b>Key</b>	<code>replsets.affinity.antiAffinityTopologyKey</code>
<b>Value</b>	string
<b>Example</b>	<code>kubernetes.io/hostname</code>
<b>Description</b>	The <a href="#">Kubernetes topologyKey</a> node affinity constraint for the Replica Set nodes
<b>Key</b>	<code>replsets.affinity.advanced</code>
<b>Value</b>	subdoc
<b>Example</b>	
<b>Description</b>	In cases where the pods require complex tuning the advanced option turns off the <code>topologykey</code> effect. This setting allows the standard Kubernetes affinity constraints of any complexity to be used

<b>Key</b>	<code>replsets.tolerations.key</code>
<b>Value</b>	string
<b>Example</b>	<code>node.alpha.kubernetes.io/unreachable</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> key for the Replica Set nodes
<b>Key</b>	<code>replsets.tolerations.operator</code>
<b>Value</b>	string
<b>Example</b>	<code>Exists</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> operator for the Replica Set nodes
<b>Key</b>	<code>replsets.tolerations.effect</code>
<b>Value</b>	string
<b>Example</b>	<code>NoExecute</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> effect for the Replica Set nodes
<b>Key</b>	<code>replsets.tolerations.tolerationSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>6000</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> time limit for the Replica Set nodes
<b>Key</b>	<code>replsets.priorityClassName</code>
<b>Value</b>	string
<b>Example</b>	<code>high priority</code>
<b>Description</b>	The <a href="#">Kuberentes Pod priority class</a> for the Replica Set nodes
<b>Key</b>	<code>replsets.annotations</code>
<b>Value</b>	string
<b>Example</b>	<code>iam.amazonaws.com/role: role-arn</code>
<b>Description</b>	The <a href="#">Kubernetes annotations</a> metadata for the Replica Set nodes
<b>Key</b>	<code>replsets.labels</code>
<b>Value</b>	label
<b>Example</b>	<code>rack: rack-22</code>
<b>Description</b>	The <a href="#">Kubernetes affinity labels</a> for the Replica Set nodes
<b>Key</b>	<code>replsets.nodeSelector</code>
<b>Value</b>	label
<b>Example</b>	<code>disktype: ssd</code>
<b>Description</b>	The <a href="#">Kubernetes nodeSelector</a> affinity constraint for the Replica Set nodes

<b>Key</b>	<code>replsets.storage.engine</code>
<b>Value</b>	string
<b>Example</b>	<code>wiredTiger</code>
<b>Description</b>	Sets the <code>storage.engine</code> option <a href="https://docs.mongodb.com/manual/reference/configuration-options/#storage.engine`_">https://docs.mongodb.com/manual/reference/configuration-options/#storage.engine`_</a> for the Replica Set nodes
<b>Key</b>	<code>replsets.storage.wiredTiger.engineConfig.cacheSizeRatio</code>
<b>Value</b>	float
<b>Example</b>	<code>0.5</code>
<b>Description</b>	The ratio used to compute the <code>storage.wiredTiger.engineConfig.cacheSizeGB</code> option for the Replica Set nodes
<b>Key</b>	<code>replsets.storage.wiredTiger.engineConfig.directoryForIndexes</code>
<b>Value</b>	bool
<b>Example</b>	<code>false</code>
<b>Description</b>	Sets the <code>storage.wiredTiger.engineConfig.directoryForIndexes</code> option for the Replica Set nodes
<b>Key</b>	<code>replsets.storage.wiredTiger.engineConfig.journalCompressor</code>
<b>Value</b>	string
<b>Example</b>	<code>snappy</code>
<b>Description</b>	Sets the <code>storage.wiredTiger.engineConfig.journalCompressor</code> option for the Replica Set nodes
<b>Key</b>	<code>replsets.storage.wiredTiger.collectionConfig.blockCompressor</code>
<b>Value</b>	string
<b>Example</b>	<code>snappy</code>
<b>Description</b>	Sets the <code>storage.wiredTiger.collectionConfig.blockCompressor</code> option for the Replica Set nodes
<b>Key</b>	<code>replsets.storage.wiredTiger.indexConfig.prefixCompression</code>
<b>Value</b>	bool
<b>Example</b>	<code>true</code>
<b>Description</b>	Sets the <code>storage.wiredTiger.indexConfig.prefixCompression</code> option for the Replica Set nodes
<b>Key</b>	<code>replsets.storage.inMemory.engineConfig.inMemorySizeRatio</code>
<b>Value</b>	float
<b>Example</b>	<code>0.9</code>
<b>Description</b>	The ratio used to compute the <code>storage.engine.inMemory.inMemorySizeGb</code> option for the Replica Set nodes
<b>Key</b>	<code>replsets.livenessProbe.failureThreshold</code>

<b>Value</b>	int
<b>Example</b>	4
<b>Description</b>	Number of consecutive unsuccessful tries of the <a href="#">liveness probe</a> to be undertaken before giving up
<b>Key</b>	<a href="#">replsets.livenessProbe.initialDelaySeconds</a>
<b>Value</b>	int
<b>Example</b>	60
<b>Description</b>	Number of seconds to wait after the container start before initiating the <a href="#">liveness probe</a> .
<b>Key</b>	<a href="#">replsets.livenessProbe.periodSeconds</a>
<b>Value</b>	int
<b>Example</b>	30
<b>Description</b>	How often to perform a <a href="#">liveness probe</a> (in seconds)
<b>Key</b>	<a href="#">replsets.livenessProbe.timeoutSeconds</a>
<b>Value</b>	int
<b>Example</b>	10
<b>Description</b>	Number of seconds after which the <a href="#">liveness probe</a> times out
<b>Key</b>	<a href="#">replsets.livenessProbe.startupDelaySeconds</a>
<b>Value</b>	int
<b>Example</b>	7200
<b>Description</b>	Time after which the <a href="#">liveness probe</a> is failed if the MongoDB instance didn't finish its full startup yet
<b>Key</b>	<a href="#">replsets.readinessProbe.failureThreshold</a>
<b>Value</b>	int
<b>Example</b>	8
<b>Description</b>	Number of consecutive unsuccessful tries of the <a href="#">readiness probe</a> to be undertaken before giving up
<b>Key</b>	<a href="#">replsets.readinessProbe.initialDelaySeconds</a>
<b>Value</b>	int
<b>Example</b>	10
<b>Description</b>	Number of seconds to wait after the container start before initiating the <a href="#">readiness probe</a>
<b>Key</b>	<a href="#">replsets.readinessProbe.periodSeconds</a>
<b>Value</b>	int
<b>Example</b>	3

<b>Description</b>	How often to perform a <a href="#">readiness probe</a> (in seconds)
<b>Key</b>	<code>replsets.readinessProbe.successThreshold</code>
<b>Value</b>	int
<b>Example</b>	<code>1</code>
<b>Description</b>	Minimum consecutive successes for the <a href="#">readiness probe</a> to be considered successful after having failed
<b>Key</b>	<code>replsets.readinessProbe.timeoutSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>2</code>
<b>Description</b>	Number of seconds after which the <a href="#">readiness probe</a> times out
<b>Key</b>	<code>replsets.runtimeClassName</code>
<b>Value</b>	string
<b>Example</b>	<code>image-rc</code>
<b>Description</b>	Name of the Kubernetes Runtime Class for Replica Set Pods
<b>Key</b>	<code>replsets.sidecars.image</code>
<b>Value</b>	string
<b>Example</b>	<code>busybox</code>
<b>Description</b>	Image for the <a href="#">custom sidecar container</a> for Replica Set Pods
<b>Key</b>	<code>replsets.sidecars.command</code>
<b>Value</b>	array
<b>Example</b>	<code>["/bin/sh"]</code>
<b>Description</b>	Command for the <a href="#">custom sidecar container</a> for Replica Set Pods
<b>Key</b>	<code>replsets.sidecars.args</code>
<b>Value</b>	array
<b>Example</b>	<code>["-c", "while true; do echo echo \$(date -u) 'test' &gt;&gt; /dev/null; sleep 5;done"]</code>
<b>Description</b>	Command arguments for the <a href="#">custom sidecar container</a> for Replica Set Pods
<b>Key</b>	<code>replsets.sidecars.name</code>
<b>Value</b>	string
<b>Example</b>	<code>rs-sidecar-1</code>
<b>Description</b>	Name of the <a href="#">custom sidecar container</a> for Replica Set Pods
<b>Key</b>	<code>replsets.sidecars.volumeMounts.mountPath</code>
<b>Value</b>	string

<b>Example</b>	<code>/volume1</code>
<b>Description</b>	Mount path of the <a href="#">custom sidecar container</a> volume for Replica Set Pods
<b>Key</b>	<code>replsets.sidecars.volumeMounts.name</code>
<b>Value</b>	string
<b>Example</b>	<code>sidecar-volume-claim</code>
<b>Description</b>	Name of the <a href="#">custom sidecar container</a> volume for Replica Set Pods
<b>Key</b>	<code>replsets.sidecarVolumes.name</code>
<b>Value</b>	string
<b>Example</b>	<code>sidecar-config</code>
<b>Description</b>	Name of the <a href="#">custom sidecar container</a> volume for Replica Set Pods
<b>Key</b>	<code>replsets.sidecarVolumes.configMap.name</code>
<b>Value</b>	string
<b>Example</b>	<code>myconfigmap</code>
<b>Description</b>	Name of the ConfigMap for a <a href="#">custom sidecar container</a> volume for Replica Set Pods
<b>Key</b>	<code>replsets.sidecarVolumes.secret.secretName</code>
<b>Value</b>	string
<b>Example</b>	<code>sidecar-secret</code>
<b>Description</b>	Name of the Secret for a <a href="#">custom sidecar container</a> volume for Replica Set Pods
<b>Key</b>	<code>replsets.sidecarPVCs</code>
<b>Value</b>	subdoc
<b>Example</b>	
<b>Description</b>	<a href="#">Persistent Volume Claim</a> for the <a href="#">custom sidecar container</a> volume for Replica Set Pods
<b>Key</b>	<code>replsets.podDisruptionBudget.maxUnavailable</code>
<b>Value</b>	int
<b>Example</b>	<code>1</code>
<b>Description</b>	The <a href="#">Kubernetes Pod distribution budget</a> limit specifying the maximum value for unavailable Pods
<b>Key</b>	<code>replsets.podDisruptionBudget.minAvailable</code>
<b>Value</b>	int
<b>Example</b>	<code>1</code>
<b>Description</b>	The <a href="#">Kubernetes Pod distribution budget</a> limit specifying the minimum value for available Pods
<b>Key</b>	<code>replsets.expose.enabled</code>



<b>Value</b>	boolean
<b>Example</b>	<code>false</code>
<b>Description</b>	Enable or disable exposing <a href="#">MongoDB Replica Set</a> nodes with dedicated IP addresses
<b>Key</b>	<code>replsets.expose.exposeType</code>
<b>Value</b>	string
<b>Example</b>	<code>ClusterIP</code>
<b>Description</b>	The <a href="#">IP address type</a> to be exposed
<b>Key</b>	<code>replsets.expose.loadBalancerSourceRanges</code>
<b>Value</b>	string
<b>Example</b>	<code>10.0.0.0/8</code>
<b>Description</b>	The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations)
<b>Key</b>	<code>replsets.expose.serviceAnnotations</code>
<b>Value</b>	string
<b>Example</b>	<code>service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http</code>
<b>Description</b>	The <a href="#">Kubernetes annotations</a> metadata for the MongoDB mongod daemon
<b>Key</b>	<code>replsets.expose.serviceLabels</code>
<b>Value</b>	string
<b>Example</b>	<code>rack: rack-22</code>
<b>Description</b>	The <a href="#">Kubernetes labels</a> for the MongoDB Replica Set Service
<b>Key</b>	<code>replsets.nonvoting.enabled</code>
<b>Value</b>	boolean
<b>Example</b>	<code>false</code>
<b>Description</b>	Enable or disable creation of <a href="#">Replica Set non-voting instances</a> within the cluster
<b>Key</b>	<code>replsets.nonvoting.size</code>
<b>Value</b>	int
<b>Example</b>	<code>1</code>
<b>Description</b>	The number of <a href="#">Replica Set non-voting instances</a> within the cluster
<b>Key</b>	<code>replsets.nonvoting.affinity.antiAffinityTopologyKey</code>
<b>Value</b>	string
<b>Example</b>	<code>kubernetes.io/hostname</code>
<b>Description</b>	The <a href="#">Kubernetes topologyKey</a> node affinity constraint for the non-voting nodes

<b>Key</b>	<code>replsets.nonvoting.affinity.advanced</code>
<b>Value</b>	subdoc
<b>Example</b>	
<b>Description</b>	In cases where the pods require complex tuning the advanced option turns off the <code>topologykey</code> effect. This setting allows the standard Kubernetes affinity constraints of any complexity to be used
<b>Key</b>	<code>replsets.nonvoting.tolerations.key</code>
<b>Value</b>	string
<b>Example</b>	<code>node.alpha.kubernetes.io/unreachable</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> key for the non-voting nodes
<b>Key</b>	<code>replsets.nonvoting.tolerations.operator</code>
<b>Value</b>	string
<b>Example</b>	<code>Exists</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> operator for the non-voting nodes
<b>Key</b>	<code>replsets.nonvoting.tolerations.effect</code>
<b>Value</b>	string
<b>Example</b>	<code>NoExecute</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> effect for the non-voting nodes
<b>Key</b>	<code>replsets.nonvoting.tolerations.tolerationSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>6000</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> time limit for the non-voting nodes
<b>Key</b>	<code>replsets.nonvoting.priorityClassName</code>
<b>Value</b>	string
<b>Example</b>	<code>high priority</code>
<b>Description</b>	The <a href="#">Kuberentes Pod priority class</a> for the non-voting nodes
<b>Key</b>	<code>replsets.nonvoting.annotations</code>
<b>Value</b>	string
<b>Example</b>	<code>iam.amazonaws.com/role: role-arn</code>
<b>Description</b>	The <a href="#">Kubernetes annotations</a> metadata for the non-voting nodes
<b>Key</b>	<code>replsets.nonvoting.labels</code>
<b>Value</b>	label

<b>Example</b>	<code>rack: rack-22</code>
<b>Description</b>	The <a href="#">Kubernetes affinity labels</a> for the non-voting nodes
<b>Key</b>	<code>replsets.nonvoting.nodeSelector</code>
<b>Value</b>	label
<b>Example</b>	<code>disktype: ssd</code>
<b>Description</b>	The <a href="#">Kubernetes nodeSelector</a> affinity constraint for the non-voting nodes
<b>Key</b>	<code>replsets.arbiter.enabled</code>
<b>Value</b>	boolean
<b>Example</b>	<code>false</code>
<b>Description</b>	Enable or disable creation of <a href="#">Replica Set Arbiter</a> nodes within the cluster
<b>Key</b>	<code>replsets.arbiter.size</code>
<b>Value</b>	int
<b>Example</b>	<code>1</code>
<b>Description</b>	The number of <a href="#">Replica Set Arbiter</a> instances within the cluster
<b>Key</b>	<code>replsets.arbiter.affinity.antiAffinityTopologyKey</code>
<b>Value</b>	string
<b>Example</b>	<code>kubernetes.io/hostname</code>
<b>Description</b>	The <a href="#">Kubernetes topologyKey</a> node affinity constraint for the Arbiter
<b>Key</b>	<code>replsets.arbiter.affinity.advanced</code>
<b>Value</b>	subdoc
<b>Example</b>	
<b>Description</b>	In cases where the pods require complex tuning the advanced option turns off the <code>topologykey</code> effect. This setting allows the standard <a href="#">Kubernetes affinity</a> constraints of any complexity to be used
<b>Key</b>	<code>replsets.arbiter.tolerations.key</code>
<b>Value</b>	string
<b>Example</b>	<code>node.alpha.kubernetes.io/unreachable</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> key for the Arbiter nodes
<b>Key</b>	<code>replsets.arbiter.tolerations.operator</code>
<b>Value</b>	string
<b>Example</b>	<code>Exists</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> operator for the Arbiter nodes

<b>Key</b>	<code>replsets.arbiter.tolerations.effect</code>
<b>Value</b>	string
<b>Example</b>	<code>NoExecute</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> effect for the Arbiter nodes
<b>Key</b>	<code>replsets.arbiter.tolerations.tolerationSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>6000</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> time limit for the Arbiter nodes
<b>Key</b>	<code>replsets.arbiter.priorityClassName</code>
<b>Value</b>	string
<b>Example</b>	<code>high priority</code>
<b>Description</b>	The <a href="#">Kuberentes Pod priority class</a> for the Arbiter nodes
<b>Key</b>	<code>replsets.arbiter.annotations</code>
<b>Value</b>	string
<b>Example</b>	<code>iam.amazonaws.com/role: role-arn</code>
<b>Description</b>	The <a href="#">Kubernetes annotations</a> metadata for the Arbiter nodes
<b>Key</b>	<code>replsets.arbiter.labels</code>
<b>Value</b>	label
<b>Example</b>	<code>rack: rack-22</code>
<b>Description</b>	The <a href="#">Kubernetes affinity labels</a> for the Arbiter nodes
<b>Key</b>	<code>replsets.arbiter.nodeSelector</code>
<b>Value</b>	label
<b>Example</b>	<code>disktype: ssd</code>
<b>Description</b>	The <a href="#">Kubernetes nodeSelector</a> affinity constraint for the Arbiter nodes
<b>Key</b>	<code>replsets.resources.limits.cpu</code>
<b>Value</b>	string
<b>Example</b>	<code>300m</code>
<b>Description</b>	<a href="#">Kubernetes CPU limit</a> for MongoDB container
<b>Key</b>	<code>replsets.resources.limits.memory</code>
<b>Value</b>	string
<b>Example</b>	<code>0.5G</code>
<b>Description</b>	<a href="#">Kubernetes Memory limit</a> for MongoDB container

<b>Key</b>	<code>replsets.resources.requests.cpu</code>
<b>Value</b>	string
<b>Example</b>	
<b>Description</b>	The <a href="#">Kubernetes CPU requests</a> for MongoDB container
<b>Key</b>	<code>replsets.resources.requests.memory</code>
<b>Value</b>	string
<b>Example</b>	
<b>Description</b>	The <a href="#">Kubernetes Memory requests</a> for MongoDB container
<b>Key</b>	<code>replsets.volumeSpec.emptyDir</code>
<b>Value</b>	string
<b>Example</b>	<code>{}</code>
<b>Description</b>	The <a href="#">Kubernetes emptyDir volume</a> , i.e. the directory which will be created on a node, and will be accessible to the MongoDB Pod containers
<b>Key</b>	<code>replsets.volumeSpec.hostPath.path</code>
<b>Value</b>	string
<b>Example</b>	<code>/data</code>
<b>Description</b>	<a href="#">Kubernetes hostPath volume</a> , i.e. the file or directory of a node that will be accessible to the MongoDB Pod containers
<b>Key</b>	<code>replsets.volumeSpec.hostPath.type</code>
<b>Value</b>	string
<b>Example</b>	<code>Directory</code>
<b>Description</b>	The <a href="#">Kubernetes hostPath volume type</a>
<b>Key</b>	<code>replsets.volumeSpec.persistentVolumeClaim.storageClassName</code>
<b>Value</b>	string
<b>Example</b>	<code>standard</code>
<b>Description</b>	The <a href="#">Kubernetes Storage Class</a> to use with the MongoDB container <a href="#">Persistent Volume Claim</a> . Use Storage Class with XFS as the default filesystem if possible, [for better MongoDB performance] ( <a href="https://dba.stackexchange.com/questions/190578/is-xfs-still-the-best-choice-for-mongodb">https://dba.stackexchange.com/questions/190578/is-xfs-still-the-best-choice-for-mongodb</a> )
<b>Key</b>	<code>replsets.volumeSpec.persistentVolumeClaim.accessModes</code>
<b>Value</b>	array
<b>Example</b>	<code>[ "ReadWriteOnce" ]</code>
<b>Description</b>	The <a href="#">Kubernetes Persistent Volume</a> access modes for the MongoDB container
<b>Key</b>	<code>replsets.volumeSpec.persistentVolumeClaim.resources.requests.storage</code>

<b>Value</b>	string
<b>Example</b>	<code>3Gi</code>
<b>Description</b>	The <a href="#">Kubernetes Persistent Volume</a> size for the MongoDB container

#### 15.1.4 PMM Section

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

<b>Key</b>	<code>pmm.enabled</code>
<b>Value</b>	boolean
<b>Example</b>	<code>false</code>
<b>Description</b>	Enables or disables monitoring Percona Server for MongoDB with <a href="#">PMM</a>
<b>Key</b>	<code>pmm.image</code>
<b>Value</b>	string
<b>Example</b>	<code>percona/pmm-client:2.30.0</code>
<b>Description</b>	PMM Client docker image to use
<b>Key</b>	<code>pmm.serverHost</code>
<b>Value</b>	string
<b>Example</b>	<code>monitoring-service</code>
<b>Description</b>	Address of the PMM Server to collect data from the Cluster
<b>Key</b>	<code>pmm.mongodParams</code>
<b>Value</b>	string
<b>Example</b>	<code>--environment=DEV-ENV --custom-labels=DEV-ENV</code>
<b>Description</b>	Additional parameters which will be passed to the <a href="#">pmm-admin add mongodb</a> command for <code>mongod</code> Pods
<b>Key</b>	<code>pmm.mongosParams</code>
<b>Value</b>	string
<b>Example</b>	<code>--environment=DEV-ENV --custom-labels=DEV-ENV</code>
<b>Description</b>	Additional parameters which will be passed to the <a href="#">pmm-admin add mongodb</a> command for <code>mongos</code> Pods

### 15.1.5 Sharding Section

The `sharding` section in the `deploy/cr.yaml` file contains configuration options for Percona Server for MongoDB `sharding`.

<b>Key</b>	<code>sharding.enabled</code>
<b>Value</b>	boolean
<b>Example</b>	<code>true</code>
<b>Description</b>	Enables or disables Percona Server for MondoDB sharding
<b>Key</b>	<code>sharding.configsvrReplSet.size</code>
<b>Value</b>	int
<b>Example</b>	<code>3</code>
<b>Description</b>	The number of Config Server instances within the cluster
<b>Key</b>	<code>sharding.configsvrReplSet.configuration</code>
<b>Value</b>	string
<b>Example</b>	<pre>  operationProfiling:   mode: slowOp systemLog:   verbosity: 1</pre>
<b>Description</b>	Custom configuration options for Config Servers. Please refer to the <a href="#">official manual</a> for the full list of options
<b>Key</b>	<code>sharding.configsvrReplSet.livenessProbe.failureThreshold</code>
<b>Value</b>	int
<b>Example</b>	<code>4</code>
<b>Description</b>	Number of consecutive unsuccessful tries of the <a href="#">liveness probe</a> to be undertaken before giving up
<b>Key</b>	<code>sharding.configsvrReplSet.livenessProbe.initialDelaySeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>60</code>
<b>Description</b>	Number of seconds to wait after the container start before initiating the <a href="#">liveness probe</a>
<b>Key</b>	<code>sharding.configsvrReplSet.livenessProbe.periodSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>30</code>
<b>Description</b>	How often to perform a <a href="#">liveness probe</a> (in seconds)
<b>Key</b>	<code>sharding.configsvrReplSet.livenessProbe.timeoutSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>10</code>
<b>Description</b>	Number of seconds after which the <a href="#">liveness probe</a> times out



<b>Key</b>	<code>sharding.configsvrReplSet.livenessProbe.startupDelaySeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>7200</code>
<b>Description</b>	Time after which the liveness probe is failed if the MongoDB instance didn't finish its full startup yet
<b>Key</b>	<code>sharding.configsvrReplSet.readinessProbe.failureThreshold</code>
<b>Value</b>	int
<b>Example</b>	<code>3</code>
<b>Description</b>	Number of consecutive unsuccessful tries of the <code>readiness probe</code> to be undertaken before giving up
<b>Key</b>	<code>sharding.configsvrReplSet.readinessProbe.initialDelaySeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>10</code>
<b>Description</b>	Number of seconds to wait after the container start before initiating the <code>readiness probe</code>
<b>Key</b>	<code>sharding.configsvrReplSet.readinessProbe.periodSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>3</code>
<b>Description</b>	How often to perform a <code>readiness probe</code> (in seconds)
<b>Key</b>	<code>sharding.configsvrReplSet.readinessProbe.successThreshold</code>
<b>Value</b>	int
<b>Example</b>	<code>1</code>
<b>Description</b>	Minimum consecutive successes for the <code>readiness probe</code> to be considered successful after having failed
<b>Key</b>	<code>sharding.configsvrReplSet.readinessProbe.timeoutSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>2</code>
<b>Description</b>	Number of seconds after which the <code>readiness probe</code> times out
<b>Key</b>	<code>sharding.configsvrReplSet.runtimeClassName</code>
<b>Value</b>	string
<b>Example</b>	<code>image-rc</code>
<b>Description</b>	Name of the <code>Kubernetes Runtime Class</code> for Config Server Pods
<b>Key</b>	<code>sharding.configsvrReplSet.sidecars.image</code>
<b>Value</b>	string

<b>Example</b>	<code>busybox</code>
<b>Description</b>	Image for the <a href="#">custom sidecar container</a> for Config Server Pods
<b>Key</b>	<code>sharding.configsvrReplSet.sidecars.command</code>
<b>Value</b>	array
<b>Example</b>	<code>["/bin/sh"]</code>
<b>Description</b>	Command for the <a href="#">custom sidecar container</a> for Config Server Pods
<b>Key</b>	<code>sharding.configsvrReplSet.sidecars.args</code>
<b>Value</b>	array
<b>Example</b>	<code>["-c", "while true; do echo echo \$(date -u) 'test' &gt;&gt; /dev/null; sleep 5;done"]</code>
<b>Description</b>	Command arguments for the <a href="#">custom sidecar container</a> for Config Server Pods
<b>Key</b>	<code>sharding.configsvrReplSet.sidecars.name</code>
<b>Value</b>	string
<b>Example</b>	<code>rs-sidecar-1</code>
<b>Description</b>	Name of the <a href="#">custom sidecar container</a> for Config Server Pods
<b>Key</b>	<code>sharding.configsvrReplSet.limits.cpu</code>
<b>Value</b>	string
<b>Example</b>	<code>300m</code>
<b>Description</b>	Kubernetes CPU limit for Config Server container
<b>Key</b>	<code>sharding.configsvrReplSet.limits.memory</code>
<b>Value</b>	string
<b>Example</b>	<code>0.5G</code>
<b>Description</b>	Kubernetes Memory limit for Config Server container
<b>Key</b>	<code>sharding.configsvrReplSet.resources.requests.cpu</code>
<b>Value</b>	string
<b>Example</b>	<code>300m</code>
<b>Description</b>	The Kubernetes CPU requests for Config Server container
<b>Key</b>	<code>sharding.configsvrReplSet.requests.memory</code>
<b>Value</b>	string
<b>Example</b>	<code>0.5G</code>
<b>Description</b>	The Kubernetes Memory requests for Config Server container
<b>Key</b>	<code>sharding.configsvrReplSet.expose.enabled</code>

<b>Value</b>	boolean
<b>Example</b>	<code>false</code>
<b>Description</b>	Enable or disable exposing <a href="#">Config Server</a> nodes with dedicated IP addresses
<b>Key</b>	<a href="#">sharding.configsvrReplSet.expose.exposeType</a>
<b>Value</b>	string
<b>Example</b>	<code>ClusterIP</code>
<b>Description</b>	The IP address type to be exposed
<b>Key</b>	<a href="#">sharding.configsvrReplSet.expose.loadBalancerSourceRanges</a>
<b>Value</b>	string
<b>Example</b>	<code>10.0.0.0/8</code>
<b>Description</b>	The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations)
<b>Key</b>	<a href="#">sharding.configsvrReplSet.expose.serviceAnnotations</a>
<b>Value</b>	string
<b>Example</b>	<code>service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http</code>
<b>Description</b>	The <a href="#">Kubernetes annotations</a> metadata for the Config Server daemon
<b>Key</b>	<a href="#">sharding.configsvrReplSet.expose.serviceLabels</a>
<b>Value</b>	string
<b>Example</b>	<code>rack: rack-22</code>
<b>Description</b>	The <a href="#">Kubernetes labels</a> for the Config Server Service
<b>Key</b>	<a href="#">sharding.configsvrReplSet.volumeSpec.emptyDir</a>
<b>Value</b>	string
<b>Example</b>	<code>{}</code>
<b>Description</b>	The <a href="#">Kubernetes emptyDir volume</a> , i.e. the directory which will be created on a node, and will be accessible to the Config Server Pod containers
<b>Key</b>	<a href="#">sharding.configsvrReplSet.volumeSpec.hostPath.path</a>
<b>Value</b>	string
<b>Example</b>	<code>/data</code>
<b>Description</b>	<a href="#">Kubernetes hostPath volume</a> , i.e. the file or directory of a node that will be accessible to the Config Server Pod containers
<b>Key</b>	<a href="#">sharding.configsvrReplSet.volumeSpec.hostPath.type</a>
<b>Value</b>	string
<b>Example</b>	<code>Directory</code>

<b>Description</b>	The <a href="#">Kubernetes hostPath</a> volume type
<b>Key</b>	<code>sharding.configsvrReplSet.volumeSpec.persistentVolumeClaim.storageClassName</code>
<b>Value</b>	string
<b>Example</b>	<code>standard</code>
<b>Description</b>	The <a href="#">Kubernetes Storage Class</a> to use with the Config Server container <a href="#">Persistent Volume Claim</a> . Use Storage Class with XFS as the default filesystem if possible, for better MongoDB performance
<b>Key</b>	<code>sharding.configsvrReplSet.volumeSpec.persistentVolumeClaim.accessModes</code>
<b>Value</b>	array
<b>Example</b>	<code>[ "ReadWriteOnce" ]</code>
<b>Description</b>	The <a href="#">Kubernetes Persistent Volume</a> access modes for the Config Server container
<b>Key</b>	<code>sharding.configsvrReplSet.volumeSpec.persistentVolumeClaim.resources.requests.storage</code>
<b>Value</b>	string
<b>Example</b>	<code>3Gi</code>
<b>Description</b>	The <a href="#">Kubernetes Persistent Volume</a> size for the Config Server container
<b>Key</b>	<code>sharding.mongos.size</code>
<b>Value</b>	int
<b>Example</b>	<code>3</code>
<b>Description</b>	The number of <a href="#">mongos</a> instances within the cluster
<b>Key</b>	<code>sharding.mongos.configuration</code>
<b>Value</b>	string
<b>Example</b>	<pre>  systemLog:   verbosity: 1</pre>
<b>Description</b>	Custom configuration options for mongos. Please refer to the <a href="#">official manual</a> for the full list of options
<b>Key</b>	<code>sharding.mongos.affinity.antiAffinityTopologyKey</code>
<b>Value</b>	string
<b>Example</b>	<code>kubernetes.io/hostname</code>
<b>Description</b>	The <a href="#">Kubernetes topologyKey</a> node affinity constraint for mongos
<b>Key</b>	<code>sharding.mongos.affinity.advanced</code>
<b>Value</b>	subdoc
<b>Example</b>	

<b>Description</b>	In cases where the Pods require complex tuning the advanced option turns off the <code>topologykey</code> effect. This setting allows the standard Kubernetes affinity constraints of any complexity to be used
<b>Key</b>	<code>sharding.mongos.tolerations.key</code>
<b>Value</b>	string
<b>Example</b>	<code>node.alpha.kubernetes.io/unreachable</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> key for mongos instances
<b>Key</b>	<code>sharding.mongos.tolerations.operator</code>
<b>Value</b>	string
<b>Example</b>	<code>Exists</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> operator for mongos instances
<b>Key</b>	<code>sharding.mongos.tolerations.effect</code>
<b>Value</b>	string
<b>Example</b>	<code>NoExecute</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> effect for mongos instances
<b>Key</b>	<code>sharding.mongos.tolerations.tolerationSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>6000</code>
<b>Description</b>	The <a href="#">Kubernetes Pod tolerations</a> time limit for mongos instances
<b>Key</b>	<code>sharding.mongos.priorityClassName</code>
<b>Value</b>	string
<b>Example</b>	<code>high priority</code>
<b>Description</b>	The <a href="#">Kuberentes Pod priority class</a> for mongos instances
<b>Key</b>	<code>sharding.mongos.annotations</code>
<b>Value</b>	string
<b>Example</b>	<code>iam.amazonaws.com/role: role-arn</code>
<b>Description</b>	The <a href="#">Kubernetes annotations</a> metadata for the mongos instances
<b>Key</b>	<code>sharding.mongos.labels</code>
<b>Value</b>	label
<b>Example</b>	<code>rack: rack-22</code>
<b>Description</b>	The <a href="#">Kubernetes affinity labels</a> for mongos instances
<b>Key</b>	<code>sharding.mongos.nodeSelector</code>

<b>Value</b>	label
<b>Example</b>	<code>disktype: ssd</code>
<b>Description</b>	The <a href="#">Kubernetes nodeSelector</a> affinity constraint for mongos instances
<b>Key</b>	<code>sharding.mongos.livenessProbe.failureThreshold</code>
<b>Value</b>	int
<b>Example</b>	<code>4</code>
<b>Description</b>	Number of consecutive unsuccessful tries of the <a href="#">liveness probe</a> to be undertaken before giving up
<b>Key</b>	<code>sharding.mongos.livenessProbe.initialDelaySeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>60</code>
<b>Description</b>	Number of seconds to wait after the container start before initiating the <a href="#">liveness probe</a>
<b>Key</b>	<code>sharding.mongos.livenessProbe.periodSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>30</code>
<b>Description</b>	How often to perform a <a href="#">liveness probe</a> (in seconds)
<b>Key</b>	<code>sharding.mongos.livenessProbe.timeoutSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>10</code>
<b>Description</b>	Number of seconds after which the <a href="#">liveness probe</a> times out
<b>Key</b>	<code>sharding.mongos.livenessProbe.startupDelaySeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>7200</code>
<b>Description</b>	Time after which the <a href="#">liveness probe</a> is failed if the MongoDB instance didn't finish its full startup yet
<b>Key</b>	<code>sharding.mongos.readinessProbe.failureThreshold</code>
<b>Value</b>	int
<b>Example</b>	<code>3</code>
<b>Description</b>	Number of consecutive unsuccessful tries of the <a href="#">readiness probe</a> to be undertaken before giving up
<b>Key</b>	<code>sharding.mongos.readinessProbe.initialDelaySeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>10</code>

<b>Description</b>	Number of seconds to wait after the container start before initiating the <a href="#">readiness probe</a>
<b>Key</b>	<code>sharding.mongos.readinessProbe.periodSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>3</code>
<b>Description</b>	How often to perform a <a href="#">readiness probe</a> (in seconds)
<b>Key</b>	<code>sharding.mongos.readinessProbe.successThreshold</code>
<b>Value</b>	int
<b>Example</b>	<code>1</code>
<b>Description</b>	Minimum consecutive successes for the <a href="#">readiness probe</a> to be considered successful after having failed
<b>Key</b>	<code>sharding.mongos.readinessProbe.timeoutSeconds</code>
<b>Value</b>	int
<b>Example</b>	<code>2</code>
<b>Description</b>	Number of seconds after which the <a href="#">readiness probe</a> times out
<b>Key</b>	<code>sharding.mongos.runtimeClassName</code>
<b>Value</b>	string
<b>Example</b>	<code>image-rc</code>
<b>Description</b>	Name of the <a href="#">Kubernetes Runtime Class</a> for mongos Pods
<b>Key</b>	<code>sharding.mongos.sidecars.image</code>
<b>Value</b>	string
<b>Example</b>	<code>busybox</code>
<b>Description</b>	Image for the <a href="#">custom sidecar container</a> for mongos Pods
<b>Key</b>	<code>sharding.mongos.sidecars.command</code>
<b>Value</b>	array
<b>Example</b>	<code>["/bin/sh"]</code>
<b>Description</b>	Command for the <a href="#">custom sidecar container</a> for mongos Pods
<b>Key</b>	<code>sharding.mongos.sidecars.args</code>
<b>Value</b>	array
<b>Example</b>	<code>["-c", "while true; do echo echo \$(date -u) 'test' &gt;&gt; /dev/null; sleep 5;done"]</code>
<b>Description</b>	Command arguments for the <a href="#">custom sidecar container</a> for mongos Pods
<b>Key</b>	<code>sharding.mongos.sidecars.name</code>
<b>Value</b>	string

<b>Example</b>	<code>rs-sidecar-1</code>
<b>Description</b>	Name of the <a href="#">custom sidecar container</a> for mongos Pods
<b>Key</b>	<code>sharding.mongos.limits.cpu</code>
<b>Value</b>	string
<b>Example</b>	<code>300m</code>
<b>Description</b>	Kubernetes CPU limit for mongos container
<b>Key</b>	<code>sharding.mongos.limits.memory</code>
<b>Value</b>	string
<b>Example</b>	<code>0.5G</code>
<b>Description</b>	Kubernetes Memory limit for mongos container
<b>Key</b>	<code>sharding.mongos.resources.requests.cpu</code>
<b>Value</b>	string
<b>Example</b>	<code>300m</code>
<b>Description</b>	The Kubernetes CPU requests for mongos container
<b>Key</b>	<code>sharding.mongos.requests.memory</code>
<b>Value</b>	string
<b>Example</b>	<code>0.5G</code>
<b>Description</b>	The Kubernetes Memory requests for mongos container
<b>Key</b>	<code>sharding.mongos.expose.exposeType</code>
<b>Value</b>	string
<b>Example</b>	<code>ClusterIP</code>
<b>Description</b>	The IP address type to be exposed
<b>Key</b>	<code>sharding.mongos.expose.servicePerPod</code>
<b>Value</b>	boolean
<b>Example</b>	<code>true</code>
<b>Description</b>	If set to <code>true</code> , a separate ClusterIP Service is created for each mongos instance
<b>Key</b>	<code>sharding.mongos.expose.loadBalancerSourceRanges</code>
<b>Value</b>	string
<b>Example</b>	<code>10.0.0.0/8</code>
<b>Description</b>	The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations)
<b>Key</b>	<code>sharding.mongos.expose.serviceAnnotations</code>



<b>Value</b>	string
<b>Example</b>	<code>service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http</code>
<b>Description</b>	The <a href="#">Kubernetes annotations</a> metadata for the MongoDB mongos daemon
<b>Key</b>	<a href="#">sharding.mongos.expose.serviceLabels</a>
<b>Value</b>	string
<b>Example</b>	<code>rack: rack-22</code>
<b>Description</b>	The <a href="#">Kubernetes labels</a> for the MongoDB mongos Service

### 15.1.6 Mongod Section

This section contains the Mongod configuration options. This section is **deprecated** in Percona Operator for MongoDB v1.12.0+, **and will be unavailable** in v1.14.0+. Options were moved to `replsets.configuration`.

<b>Key</b>	<a href="#">mongod.security.encryptionKeySecret</a>
<b>Value</b>	string
<b>Example</b>	<code>my-cluster-name-mongodb-encryption-key</code>
<b>Description</b>	Specifies a secret object with the <a href="#">encryption key</a> <b>Please note that this option is deprecated; use <code>spec.secrets.encryptionKey</code> instead</b>

### 15.1.7 Backup Section

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

<b>Key</b>	<code>backup.enabled</code>
<b>Value</b>	boolean
<b>Example</b>	<code>true</code>
<b>Description</b>	Enables or disables making backups
<b>Key</b>	<code>backup.image</code>
<b>Value</b>	string
<b>Example</b>	<code>percona/percona-server-mongodb-operator:1.13.0-backup</code>
<b>Description</b>	The Percona Server for MongoDB Docker image to use for the backup
<b>Key</b>	<code>backup.serviceAccountName</code>
<b>Value</b>	string
<b>Example</b>	<code>percona-server-mongodb-operator</code>
<b>Description</b>	Name of the separate privileged service account used by the Operator
<b>Key</b>	<code>backup.annotations</code>
<b>Value</b>	string
<b>Example</b>	<code>sidecar.istio.io/inject: "false"</code>
<b>Description</b>	The <a href="#">Kubernetes annotations</a> metadata for the backup job
<b>Key</b>	<code>backup.resources.limits.cpu</code>
<b>Value</b>	string
<b>Example</b>	<code>100m</code>
<b>Description</b>	<a href="#">Kubernetes CPU limit</a> for backups
<b>Key</b>	<code>backup.resources.limits.memory</code>
<b>Value</b>	string
<b>Example</b>	<code>0.2G</code>
<b>Description</b>	<a href="#">Kubernetes Memory limit</a> for backups
<b>Key</b>	<code>backup.resources.requests.cpu</code>
<b>Value</b>	string
<b>Example</b>	<code>100m</code>
<b>Description</b>	The <a href="#">Kubernetes CPU requests</a> for backups
<b>Key</b>	<code>backup.resources.requests.memory</code>
<b>Value</b>	string
<b>Example</b>	<code>0.1G</code>
<b>Description</b>	The <a href="#">Kubernetes Memory requests</a> for backups

<b>Key</b>	<code>backup.storages.&lt;storage-name&gt;.type</code>
<b>Value</b>	string
<b>Example</b>	<code>s3</code>
<b>Description</b>	The cloud storage type used for backups. Only <code>s3</code> type is currently supported
<b>Key</b>	<code>backup.storages.&lt;storage-name&gt;.s3.insecureSkipTLSVerify</code>
<b>Value</b>	boolean
<b>Example</b>	<code>true</code>
<b>Description</b>	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
<b>Key</b>	<code>backup.storages.&lt;storage-name&gt;.s3.credentialsSecret</code>
<b>Value</b>	string
<b>Example</b>	<code>my-cluster-name-backup-s3</code>
<b>Description</b>	The <a href="#">Kubernetes secret</a> for backups. It should contain <code>AWS_ACCESS_KEY_ID</code> and <code>AWS_SECRET_ACCESS_KEY</code> keys.
<b>Key</b>	<code>backup.storages.&lt;storage-name&gt;.s3.bucket</code>
<b>Value</b>	string
<b>Example</b>	
<b>Description</b>	The Amazon S3 bucket name for backups
<b>Key</b>	<code>backup.storages.&lt;storage-name&gt;.s3.prefix</code>
<b>Value</b>	string
<b>Example</b>	<code>""</code>
<b>Description</b>	The path (sub-folder) to the backups inside the <a href="#">bucket</a>
<b>Key</b>	<code>backup.storages.&lt;storage-name&gt;.s3.region</code>
<b>Value</b>	string
<b>Example</b>	<code>us-east-1</code>
<b>Description</b>	The <a href="#">AWS region</a> to use. Please note <b>this option is mandatory</b> for Amazon and all S3-compatible storages
<b>Key</b>	<code>backup.storages.&lt;storage-name&gt;.s3.endpointUrl</code>
<b>Value</b>	string
<b>Example</b>	
<b>Description</b>	The endpoint URL of the S3-compatible storage to be used (not needed for the original Amazon S3 cloud)
<b>Key</b>	<code>backup.storages.&lt;storage-name&gt;.azure.credentialsSecret</code>

<b>Value</b>	string
<b>Example</b>	<code>my-cluster-azure-secret</code>
<b>Description</b>	The <a href="#">Kubernetes secret</a> for backups. It should contain <code>AZURE_STORAGE_ACCOUNT_NAME</code> and <code>AZURE_STORAGE_ACCOUNT_KEY</code>
<b>Key</b>	<code>backup.storages.&lt;storage-name&gt;.azure.container</code>
<b>Value</b>	string
<b>Example</b>	<code>my-container</code>
<b>Description</b>	Name of the <a href="#">container</a> for backups
<b>Key</b>	<code>backup.storages.&lt;storage-name&gt;.azure.prefix</code>
<b>Value</b>	string
<b>Example</b>	<code>""</code>
<b>Description</b>	The path (sub-folder) to the backups inside the <a href="#">container</a>
<b>Key</b>	<code>backup.pitr.enabled</code>
<b>Value</b>	boolean
<b>Example</b>	<code>false</code>
<b>Description</b>	Enables or disables <a href="#">point-in-time-recovery functionality</a>
<b>Key</b>	<code>backup.pitr.oplogSpanMin</code>
<b>Value</b>	int
<b>Example</b>	<code>10</code>
<b>Description</b>	Number of minutes between the uploads of oplogs
<b>Key</b>	<code>backup.pitr.compressionType</code>
<b>Value</b>	string
<b>Example</b>	<code>gzip</code>
<b>Description</b>	The point-in-time-recovery chunks compression format, can be <a href="#">gzip</a> , <a href="#">snappy</a> , <a href="#">lz4</a> , <a href="#">pgzip</a> , <a href="#">zstd</a> , <a href="#">s2</a> , or <a href="#">none</a>
<b>Key</b>	<code>backup.pitr.compressionLevel</code>
<b>Value</b>	int
<b>Example</b>	<code>6</code>
<b>Description</b>	The point-in-time-recovery chunks compression level ( <a href="#">higher values result in better but slower compression</a> )
<b>Key</b>	<code>backup.tasks.name</code>
<b>Value</b>	string
<b>Example</b>	

<b>Description</b>	The name of the backup
<b>Key</b>	<code>backup.tasks.enabled</code>
<b>Value</b>	boolean
<b>Example</b>	<code>true</code>
<b>Description</b>	Enables or disables this exact backup
<b>Key</b>	<code>backup.tasks.schedule</code>
<b>Value</b>	string
<b>Example</b>	<code>0 0 \* \* 6</code>
<b>Description</b>	The scheduled time to make a backup, specified in the <a href="#">crontab format</a>
<b>Key</b>	<code>backup.tasks.keep</code>
<b>Value</b>	int
<b>Example</b>	<code>3</code>
<b>Description</b>	The amount of most recent backups to store. Older backups are automatically deleted. Set <code>keep</code> to zero or completely remove it to disable automatic deletion of backups
<b>Key</b>	<code>backup.tasks.storageName</code>
<b>Value</b>	string
<b>Example</b>	<code>st-us-west</code>
<b>Description</b>	The name of the S3-compatible storage for backups, configured in the <a href="#">storages subsection</a>
<b>Key</b>	<code>backup.tasks.compressionType</code>
<b>Value</b>	string
<b>Example</b>	<code>gzip</code>
<b>Description</b>	The backup compression format, can be <code>gzip</code> , <code>snappy</code> , <code>lz4</code> , <code>pgzip</code> , <code>zstd</code> , <code>s2</code> , or <code>none</code>
<b>Key</b>	<code>backup.tasks.compressionLevel</code>
<b>Value</b>	int
<b>Example</b>	<code>6</code>
<b>Description</b>	The backup compression level (higher values result in better but slower compression)

Last update: 2022-09-15

## 15.2 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:1.13.0	7137ee6ff918bd2366033f198f2919e7aa291ae3b76460e267896bf052251837
percona/pmm-client:2.30.0	de556410de32a49a8a6bc157536881e2baefc8549a1094d6c2c70242a3c792cb
percona/percona-backup-mongodb:1.8.1	80aad4f71ee3ce721f019e0409cc5a21c07376169428bbd04b486da3bf515704
percona/percona-server-mongodb:5.0.11-10	da3713525d76a354435e1ab8fda12a06407e7eca8b8e72b9ac0163a34c8eb735
percona/percona-server-mongodb:5.0.7-6	3f4849a17236c3849a513f46caa39fbc6da0414f98d27e080fbe0496fa9e86a2
percona/percona-server-mongodb:5.0.4-3	4ac4cff1dac52ea109e9a68a61de44c75b62292bb4676cf8efd1e0000d8adf3
percona/percona-server-mongodb:4.4.16-16	402b5e5b08ac73c74a47c72d002251a086f9ad28b0594fbae5c34757b294ce13
percona/percona-server-mongodb:4.4.13-13	059c3c9a0360d6823905e39b52bdcaf76c3929c93408c537f139cee835c2bc0f
percona/percona-server-mongodb:4.4.10-11	ea73a506fa02604660e3ef7d452d142a89587bb5daca15d3cc1b539a9b1000c5
percona/percona-server-mongodb:4.4.8-9	4d29b3557c949f95009eaccf7a8f56215ac609406d230be87b6eaa072e0c1f69
percona/percona-server-mongodb:4.2.22-22	da4634df780563e10a547662c58c8ce28fbe5c98e1ac8b42b4f6be87f292e92b

.....

Last update: 2022-09-15

## 15.3 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.

### 15.3.1 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 `default` namespace.

#### Note

In this document `default` namespace is used in all examples. Substitute `default` with your namespace name if you use a different one.

1. 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)
```

### 15.3.2 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",
 "serviceName": "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 engine 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): service 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. serviceName (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": {
 ".": {
 "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:tllMonitorSleepSecs":{
 },
 "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/perconaservermongodb/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"
 }
 }
}

```

### 15.3.3 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/perconaservermongodbbs",
 "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/perconaservermongodbbs/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, \"serviceName\": \"percona-server-mongodb-operator\", \"storages\": null, \"tasks\": null}, \"image\": \"percona/percona-server-mongodb:4.2.8-8\", \"imagePullPolicy\": \"Always\", \"mongodb\": {\"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: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":{
 }
 }
 }
]
}

```

### 15.3.4 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": {
 ".": {
 "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"
 }
}

```

### 15.3.5 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](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): Specify the size 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": {
 ".": {
 "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"
 }
}

```

### 15.3.6 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": {
 "kubectrl.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": {
 ".": {
 "kubectrl.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,
 "serviceName": "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"
}
}

```



## 15.3.7 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/perconaservermongoddbbackups
```

### Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

### cURL Request:

```
$ curl -k -v -XPOST "https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongoddbbackups" \
 -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": {
 "kubectrl.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:kubectrl.kubernetes.io/last-applied-configuration": {

 }
 },
 "f:spec": {
 ".": {

 },
 "f:psmdbCluster": {

 },
 "f:storageName": {

 }
 }
 },
 "manager": "kubectrl",
 "operation": "Update",
 "time": "2020-07-27T13:45:43Z"
 }
],
 "name": "backup1",
 "namespace": "default",
 "resourceVersion": "1290243",
 "selfLink": "/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongoddbbackups/backup1",
 "uid": "e695d1c7-898e-44b0-b356-537284f6c046"
 },
 "spec": {
 "psmdbCluster": "my-cluster-name",
 "storageName": "s3-us-west"
 }
}

```

### 15.3.8 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/perconaservermongoddbrestores
```

#### Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

#### cURL Request:

```
$ curl -k -v -XPOST "https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongoddbrestores" \
 -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": {
 "kubectrl.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": {
 ".": {
 "kubectrl.kubernetes.io/last-applied-configuration": {
 }
 }
 }
 },
 "f:spec": {
 ".": {
 "f:backupName": {
 },
 "f:clusterName": {
 }
 }
 }
 },
 "manager": "kubectrl",
 "operation": "Update",
 "time": "2020-07-27T13:52:56Z"
 }
],
 "name": "restore1",
 "namespace": "default",
 "resourceVersion": "1291198",
 "selfLink": "/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbretores/restore1",
 "uid": "17e982fe-ac41-47f4-afba-fea380b0c76e"
 },
 "spec": {
 "backupName": "backup1",
 "clusterName": "my-cluster-name"
 }
}
```

Last update: 2022-08-12

## 15.4 Frequently Asked Questions

### 15.4.1 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.

### 15.4.2 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](#).

### 15.4.3 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.

### 15.4.4 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: 4.4.16-16. More details on the exact Percona Server for MongoDB version can be found in the release notes (5.0, 4.4, and 4.2).

### 15.4.5 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:

```
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](#).

## 15.4.6 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.

---

Last update: 2022-08-18

## 16. Release notes

### 16.1 *Percona Operator for MongoDB 1.13.0*

- **Date**

September 15, 2022

- **Installation**

[Installing Percona Operator for MongoDB](#)

#### 16.1.1 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

#### 16.1.2 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 \(AKS\)](#)

#### 16.1.3 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 Operator changed its API version to v1 instead of having a separate API version for each release. Three last API version are supported in addition to `v1`, which substantially reduces the size of Custom Resource Definition to prevent reaching the etcd limit
- [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

### 16.1.4 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

### 16.1.5 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](#)

### 16.1.6 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.

---

Last update: 2022-09-15



## 16.2 Percona Operator for MongoDB 1.12.0

- **Date**

May 5, 2022

- **Installation**

[Installing Percona Operator for MongoDB](#)

### 16.2.1 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

### 16.2.2 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)

### 16.2.3 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

### 16.2.4 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

### 16.2.5 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)

### 16.2.6 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!

- **K8SPSMD-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)

## 16.2.7 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.

---

Last update: 2022-08-08

## 16.3 Percona Distribution for MongoDB Operator 1.11.0

- **Date**

December 21, 2021

- **Installation**

For installation please refer to [the documentation page](#)

### 16.3.1 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.

### 16.3.2 New Features

- [K8SPSMDB-513](#): Add support of Microsoft Azure Blob storage for backups

### 16.3.3 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

### 16.3.4 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

### 16.3.5 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.

---

Last update: 2022-08-08

## 16.4 Percona Distribution for MongoDB Operator 1.10.0

- **Date**

September 30, 2021

- **Installation**

For installation please refer to [the documentation page](#)

### 16.4.1 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.

### 16.4.2 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

### 16.4.3 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

### 16.4.4 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

### 16.4.5 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.

---

Last update: 2022-08-08

## 16.5 Percona Distribution for MongoDB Operator 1.9.0

- **Date**

June 29, 2021

- **Installation**

For installation please refer to [the documentation page](#)

### 16.5.1 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

### 16.5.2 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

### 16.5.3 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

### 16.5.4 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

### 16.5.5 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

---

Last update: 2022-08-08

## 16.6 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

### 16.6.1 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

### 16.6.2 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

### 16.6.3 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

### 16.6.4 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

- [K8SPSMD-43](#): Fix a bug due to which renaming Replica Set in the Custom Resource caused creating new Replica Set without deleting the old one

---

Last update: 2022-08-08

## 16.7 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](#)

### 16.7.1 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.

### 16.7.2 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

### 16.7.3 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

### 16.7.4 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

Last update: 2022-08-08

## 16.8 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

### 16.8.1 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.

### 16.8.2 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

### 16.8.3 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

### 16.8.4 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.

---

Last update: 2022-08-12

## 16.9 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

### 16.9.1 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)

### 16.9.2 Improvements

- [K8SPSMDB-192](#): The ability to set the mongod cursorTimeoutMillis parameter in YAML (Thanks to user xpirt64 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 “kubect!” 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

### 16.9.3 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 replicaset 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

---

Last update: 2022-08-08



## 16.10 Percona Kubernetes Operator for Percona Server for MongoDB 1.4.0

- **Date**

March 31, 2020

- **Installation**

[Installing Percona Kubernetes Operator for PSMDB](#)

### 16.10.1 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

### 16.10.2 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

### 16.10.3 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](#).

---

Last update: 2022-08-08

## 16.11 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.

### 16.11.1 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](#).

---

Last update: 2022-08-08

## 16.12 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.

### 16.12.1 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.

### 16.12.2 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](#).

---

Last update: 2022-08-08

## 16.13 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.

### 16.13.1 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](#).

---

Last update: 2022-08-08

## 16.14 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

### 16.14.1 Installation

Installation is performed by following the documentation installation instructions [for Kubernetes](#) and [OpenShift](#).

---

Last update: 2022-08-08