

Operator for MongoDB

Documentation

1.20.0 (May 19, 2025)

Table of Contents

<u>1 Welcome</u>
2 Get help from Percona
3 Features
3.1 Design and architecture
3.2 Comparison with other solutions
4 Quickstart guides
4.1 Overview
4.2 1. Quick install
4.2.1 With kubectl
4.2.2 With Helm
4.3 2. Connect to Percona Server for MongoDB
4.4 3. Insert data
4.5 4. Make a backup
4.6 5. Monitor the database with PMM
4.7 What's next?
<u>5 Installation</u>
5.1 System requirements
5.2 Install on Minikube
5.3 Install with Everest
5.4 Install on Google Kubernetes Engine (GKE)
5.5 Install on Amazon Elastic Kubernetes Service (AWS EKS)
5.6 Install on Microsoft Azure Kubernetes Service (AKS)
5.7 Generic Kubernetes installation
5.8 Install on OpenShift
<u>6 Upgrade</u>
6.1 About upgrades
6.2 Upgrade CRD and the Operator
6.3 Database upgrade overview
6.4 Minor upgrade
6.4.1 To a specific version
6.4.2 Automatic minor upgrades
6.4.3 Manual upgrade
6.5 Major upgrade
6.6 Upgrade Percona Server for MongoDB on OpenShift
7 Configuration
7.1 Application and system users
7.2 Changing MongoDB options
7.3 Change Percona Backup for MongoDB configuration
7.4 Anti-affinity and tolerations
7.5 Labels and annotations
7.6 Exposing the cluster
7.7 Local storage support
7.8 Arbiter and non-voting nodes
7.9 MongoDB sharding
7.10 Transport encryption (TLS/SSL)

7.10.1 About TLS security

7.10.2 Configure TLS using cert-manager

- 7.10.3 Generate certificates manually
- 7.10.4 Update certificates
- 7.10.5 Disable TLS
- 7.11 Data at rest encryption
- 7.12 Telemetry

8 Management

- 8.1 Backup and restore
 - 8.1.1 About backups
 - 8.1.2 Configure storage for backups
 - 8.1.3 Multiple storages for backups
 - 8.1.4 Making scheduled backups
 - 8.1.5 Making on-demand backup
 - 8.1.6 Storing operations logs for point-in-time recovery
 - 8.1.7 Enable server-side encryption
 - 8.1.8 Restore from a previously saved backup
 - 8.1.9 Delete the unneeded backup
- 8.2 Horizontal and vertical scaling
- 8.3 Multi-cluster and multi-region deployment
 - 8.3.1 About Multi-cluster
 - 8.3.2 Configuring Main site for replication
 - 8.3.3 Configuring Replica site for replication
 - 8.3.4 Backups with cross-site replication
 - 8.3.5 Splitting replica set across multiple data centers
 - 8.3.6 Enabling multi-cluster Services
- 8.4 Monitor with Percona Monitoring and Management (PMM)
- 8.5 Add sidecar containers
- 8.6 Restart or pause the cluster

9 Troubleshooting

- 9.1 Initial troubleshooting
- 9.2 Exec into the container
- 9.3 Check the logs
- 9.4 Special debug images

10 HOWTOs

- 10.1 Install the database with customized parameters
- 10.2 OpenLDAP integration
- 10.3 How to use private registry
- 10.4 Creating a private S3-compatible cloud for backups
- 10.5 Restore backup to a new Kubernetes-based environment
- 10.6 How to use backups to move the external database to Kubernetes
- 10.7 Install Percona Server for MongoDB in multi-namespace (cluster-wide) mode
- 10.8 Monitor Kubernetes
- 10.9 Delete the Operator

11 Reference

- 11.1 Custom Resource options
- 11.2 Backup Resource options
- 11.3 Restore Resource options
- 11.4 Percona certified images
- 11.5 Versions compatibility
- 11.6 Operator API
- 11.7 Frequently asked questions
- 11.8 Copyright and licensing information

11.9 Trademark policy

12 Release notes

12 1	R۵	0200	notes	indev

- 12.2 Percona Operator for MongoDB 1.20.0 (2025-05-19)
- 12.3 Percona Operator for MongoDB 1.19.1 (2025-02-20)
- 12.4 Percona Operator for MongoDB 1.19.0 (2025-01-21)
- 12.5 Percona Operator for MongoDB 1.18.0 (2024-11-14)
- 12.6 Percona Operator for MongoDB 1.17.0 (2024-09-09)
- 12.7 Percona Operator for MongoDB 1.16.2 (2024-07-23)
- 12.8 Percona Operator for MongoDB 1.16.1 (2024-06-24)
- 12.9 Percona Operator for MongoDB 1.16.0 (2024-05-24)
- 12.10 Percona Operator for MongoDB 1.15.0 (2023-10-09)
- 12.11 Percona Operator for MongoDB 1.14.0 (2023-03-13)
- 12.12 Percona Operator for MongoDB 1.13.0 (2022-09-15)
- 12.13 Percona Operator for MongoDB 1.12.0 (2022-05-05)
- 12.14 Percona Distribution for MongoDB Operator 1.11.0 (2021-12-21)
- 12.15 Percona Distribution for MongoDB Operator 1.10.0 (2021-09-30)
- 12.16 Percona Distribution for MongoDB Operator 1.9.0 (2021-07-29)
- 12.17 Percona Kubernetes Operator for Percona Server for MongoDB 1.8.0 (2021-05-06)
- 12.18 Percona Kubernetes Operator for Percona Server for MongoDB 1.7.0 (2021-03-08)
- 12.19 Percona Kubernetes Operator for Percona Server for MongoDB 1.6.0 (2020-12-22)
- 12.20 Percona Kubernetes Operator for Percona Server for MongoDB 1.5.0 (2020-09-07)
- 12.21 Percona Kubernetes Operator for Percona Server for MongoDB 1.4.0 (2020-03-31)
- 12.22 Percona Kubernetes Operator for Percona Server for MongoDB 1.3.0 (2019-12-11)
- 12.23 Percona Kubernetes Operator for Percona Server for MongoDB 1.2.0 (2019-09-20)
- 12.24 Percona Kubernetes Operator for Percona Server for MongoDB 1.1.0 (2019-07-15)
- 12.25 Percona Kubernetes Operator for Percona Server for MongoDB 1.0.0 (2019-05-29)

1 Percona Operator for MongoDB

The <u>Percona Operator for MongoDB</u> [2] is a Kubernetes-native application that uses custom resources to manage the lifecycle of Percona Server for MongoDB clusters. It works as a controller, monitoring the desired database state defined by you (through YAML files) and ensuring your MongoDB deployment matches that state automatically.

The Operator simplifies and automates tasks related to MongoDB cluster management such as:

- · Provisioning and scaling:
 - Automatically creates MongoDB clusters on Kubernetes.
 - Dynamically scales your MongoDB instances up or down based on workload requirements.
- Upgrade: Manages seamless upgrades of MongoDB versions without downtime or data loss.
- Backups and Restores: Simplifies backing up data to external storage (e.g., AWS S3, Azure) and restoring it when needed.
- Self-Healing: Detects and resolves issues such as pod failures, keeping the cluster healthy.
- High Availability: Manages replica sets and failover mechanisms to ensure your database remains available.

What's new in version 1.20.0

2 Get help from Percona

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

2.1 Percona's Community Forum

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

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

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

2.2 Percona experts

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

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

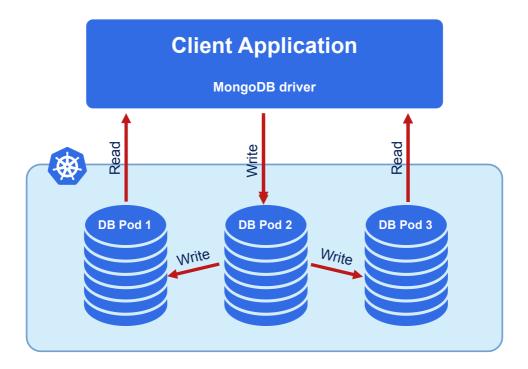
Service	Description
24/7 Expert Support	Our dedicated team of database experts is available 24/7 to assist you with any database issues. We provide flexible support plans tailored to your specific needs.
Hands-On Database Management	Our managed services team can take over the day-to-day management of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

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

3 Features

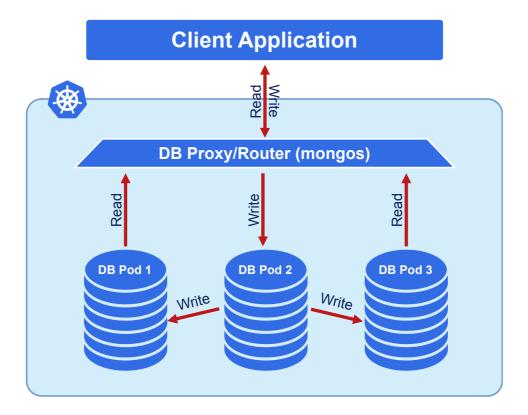
3.1 Design overview

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



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

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

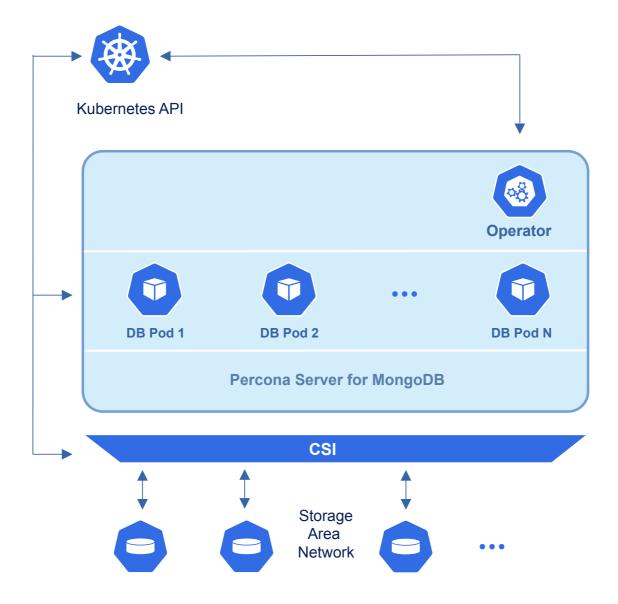


To provide high availability the Operator uses node affinity 🗗 to run MongoDB instances on separate worker nodes if possible, and the database cluster is deployed as a single Replica Set with at least three nodes. If a node fails, the pod with the mongod process is automatically re-created on another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

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



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.

3.2 Compare various solutions to deploy MongoDB in Kubernetes

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

- KubeDB ☐
- MongoDB Community Operator [2]
- Percona Operator for MongoDB [2]

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	MongoDB Enterprise Operator
Open source model	Apache 2.0	Apache 2.0	Open core	Open core	Open core
MongoDB versions	MongoDB 5.0, 6.0, 7.0 *	MongoDB 5.0	MongoDB 3.4, 3.6. 4.0, 4.1, 4.2	MongoDB 4.2, 4.4, 5.0, 6.0, 7.0	MongoDB 4.2, 4.4, 5.0, 6.0, 7.0
Kubernetes conformance	Various versions are tested	No guarantee	No guarantee	No guarantee	No guarantee
Cluster-wide mode	Yes	Not an operator	Enterprise only	Yes	Yes
Network exposure	Yes	Yes	No, only through manual config	No	Yes
Web-based GUI	Percona Everest	0	<u>kubedb-ui</u>	0	<u>Ops Manager</u>

^{*} Percona Operator relies on <u>Percona Server for MongoDB</u> - a free, enhanced, fully compatible MongoDB software alternative for MongoDB Community Server with enterprise-grade features.

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	MongoDB Enterprise Operator
Operator upgrade	Yes	Helm upgrade	Image change	Yes	Yes
Database upgrade	Automated minor, manual major	No	Manual minor	Manual minor and major	Yes
Compute scaling	Horizontal and vertical	Horizontal and vertical	Enterprise only	Horizontal only	Yes
Storage scaling	Yes	Manual	Enterprise only	No	Yes

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	MongoDB Enterprise Operator
Multi-cluster deployment	Yes	No	No	No	Yes
Sharding	Yes	Yes, another chart	Yes	No	Yes
Arbiter	Yes	Yes	Yes	Yes	Yes
Non-voting nodes	Yes	No	No	No	Yes
Hidden nodes	No	Yes	Yes	Yes	Yes
Network exposure	Yes	Yes	Manual	No	Yes
Split Horizon	Yes	No	No	Yes	Yes

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	MongoDB Enterprise Operator
Scheduled backups	Yes	No	Enterprise only	No	Yes
Incremental backups	No	No	Enterprise only	No	No
Point-in-time recovery	Yes	No	No	No	Yes
Logical backups	Yes	No	No	No	Yes
Physical backups	Yes	No	No	No	Yes

Monitoring

Monitoring is crucial for any operations team.

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

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	MongoDB Enterprise Operator
Customize MongoDB configuration	Yes	Yes	Yes	No, only some params	No, only some params
Helm	Yes	Yes	Yes, for operator only	Yes, for operator only	Yes, for operator only
SSL/TLS	Yes	Yes	Enterprise only	Yes	Yes
Create users/roles	Yes	Yes	No	Yes	Yes

4 Quickstart guides

4.1 Overview

Ready to get started with the Percona Operator for MongoDB? These tutorials help you learn some basic operations, such as:

- Install and deploy an Operator
- Connect to Percona Server for MongoDB
- Insert sample data to the database
- Set up and make a logical backup
- Monitor the database health with Percona Monitoring and Management (PMM)

Next steps

Install the Operator \rightarrow

4.2 1. Quick install

4.2.1 Install Percona Server for MongoDB using kubectl

A Kubernetes Operator is a special type of controller introduced to simplify complex deployments. The Operator extends the Kubernetes API with custom resources

The <u>Percona Operator for MongoDB</u> is based on best practices for configuration and setup of a <u>Percona Distribution for MongoDB</u> is based on best practices for configuration and setup of a <u>Percona Distribution for MongoDB</u> is based on best practices for configuration and setup of a <u>Percona Distribution for MongoDB</u> is based on best practices for configuration and setup of a <u>Percona Distribution for MongoDB</u> is based on best practices for configuration and setup of a <u>Percona Distribution for MongoDB</u> is based on best practices for configuration and setup of a <u>Percona Distribution for MongoDB</u> is based on best practices for configuration and setup of a <u>Percona Distribution for MongoDB</u> is based on best practices for configuration and setup of a <u>Percona Distribution for MongoDB</u> is based on best practices for configuration and setup of a <u>Percona Distribution for MongoDB</u> is based on best practices for configuration and setup of a <u>Percona Distribution for MongoDB</u> is based on best practices for configuration and setup of a <u>Percona Distribution for MongoDB</u> is based on best practices.

We recommend installing the Operator with the <u>kubectl</u> C command line utility. It is the universal way to interact with Kubernetes. Alternatively, you can install it using the <u>Helm</u> C package manager.

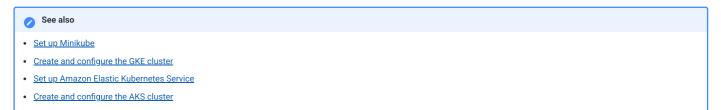


Install with Helm ->

Prerequisites

To install Percona Distribution for MongoDB, you need the following:

- 1. The **kubectl** tool to manage and deploy applications on Kubernetes, included in most Kubernetes distributions. Install not already installed, <u>follow its official</u> installation instructions. [2].
- 2. A Kubernetes environment. You can deploy it on Minikube of for testing purposes or using any cloud provider of your choice. Check the list of our officially supported platforms.

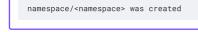


Procedure

Here's a sequence of steps to follow:

1 Create the Kubernetes namespace for your cluster. It is a good practice to isolate workloads in Kubernetes by installing the Operator in a custom namespace. Replace the <namespace> placeholder with your value.

\$ kubectl create namespace <namespace>
Expected output



2 Deploy the Operator <u>using</u> 1 the following command:

 $\$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/bundle.yaml -n <namespace>



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

3 Deploy Percona Server for MongoDB:

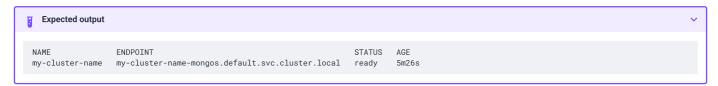
 $\label{lem:compercona} $$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr.yaml -n <namespace>$



4 Check the Operator and the Percona Server for MongoDB Pods status.

\$ kubectl get psmdb -n <namespace>

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



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

The default Percona Server for MongoDB configuration includes three mongod, three mongos, and three config server instances with enabled sharding.

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

Next steps

Connect to Percona Server for MongoDB ightarrow

Useful links

Install Percona Server for MongoDB with customized parameters

4.2.2 Install Percona Server for MongoDB using Helm

Helm 's is the package manager for Kubernetes. A Helm chart is a package that contains all the necessary resources to deploy an application to a Kubernetes cluster.

You can find Percona Helm charts in percona/percona-helm-charts ☐ repository in Github.

Prerequisites

To install and deploy the Operator, you need the following:

- 1. <u>Helm v3</u> ℃.
- 2. kubectl command line utility.
- 3. A Kubernetes environment. You can deploy it locally on Minikube of for testing purposes or using any cloud provider of your choice. Check the list of our officially supported platforms.



- Set up Minikube
- Create and configure the GKE cluster
- Set up Amazon Elastic Kubernetes Service
- Create and configure the AKS cluster

Installation

Here's a sequence of steps to follow:

- 1 Add the Percona's Helm charts repository and make your Helm client up to date with it:
 - \$ helm repo add percona https://percona.github.io/percona-helm-charts/
 - \$ helm repo update
- 2 It is a good practice to isolate workloads in Kubernetes via namespaces. Create a namespace:
 - \$ kubectl create namespace <namespace>
- 3 Install Percona Operator for MongoDB:
 - \$ helm install my-op percona/psmdb-operator --namespace <namespace>

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

Install Percona Server for MongoDB:

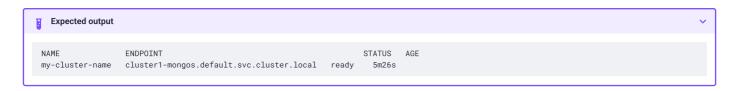
\$ helm install cluster1 percona/psmdb-db --namespace <namespace>

The cluster1 parameter is the name of a new release object \(\tilde{\cluster}\) which is created for the Percona Server for MongoDB when you install its Helm chart (use any name you like).

5 Check the Operator and the Percona Server for MongoDB Pods status.

\$ kubectl get psmdb -n <namespace>

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



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

The default Percona Server for MongoDB configuration includes three mongod, three mongos, and three config server instances with enabled sharding.

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

Next steps

Connect to Percona Server for MongoDB ightarrow

Useful links

Install Percona Server for MongoDB with customized parameters

4.3 2. Connect to Percona Server for MongoDB

In this tutorial, you will connect to the Percona Server for MongoDB cluster you deployed previously.

To connect to Percona Server for MongoDB you need to construct the MongoDB connection URI string. It includes the credentials of the admin user, which are stored in the Secrets object.

Here's how to do it:

1 List the Secrets objects

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

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

via kubectl

my-cluster-name-secrets

via Helm

cluster1-psmdb-db-secrets

- 2 Retrieve the admin user credentials. Replace the secret-name and namespace with your values in the following commands:
 - Retrieve the login

```
\ kubectl get secret <secret-name> -n <namespace> -o yaml -o jsonpath='{.data.MONGODB_DATABASE_ADMIN_USER}' | base64 -- decode | tr '\n' ' && echo " "
```

The default value is databaseAdmin

Retrieve the password

```
$ kubectl get secret <secret-name> -n <namespace> -o yaml -o jsonpath='{.data.MONGODB_DATABASE_ADMIN_PASSWORD}' | base64 -
-decode | tr '\n' ' && echo " "
```

3 Run a container with a MongoDB client and connect its console output to your terminal. The following command does this, naming the new Pod perconaclient:

```
$ kubectl -n <namespace> run -i --rm --tty percona-client --image=percona/percona-server-mongodb:6.0.21-18 --restart=Never -- bash -il
```

Connect to Percona Server for MongoDB. The format of the MongoDB connection URI string is the following:

sharding is on

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

sharding is off

```
\label{lem:mongosh} \verb| "mongodb://databaseAdmin:<databaseAdminPassword>@<cluster-name>-rs0.<namespace>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
```

If you run MongoDB 5.0 and earlier, use the old mongo client instead of mongosh.



Congratulations! You have connected to Percona Server for MongoDB.

Next steps

Insert sample data \rightarrow

4.4 3. Insert sample data

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

MongoDB provides multiple methods for data insert . We will use a For loop to insert some sample documents.

Run the following command:

```
admin> for (var i = 1; i <= 50; i++) {
   db.test.insertOne( { x : i } )
}</pre>
```

If there is no test collection created, MongoDB creates when inserting documents.

```
Cutput

{
   acknowledged: true,
   insertedId: ObjectId("652567e5eedca48f97e1868f")
}
```

2 Query the collection to verify the data insertion

```
admin> db.test.find()
```

```
Output
    { \_id: ObjectId("652567e4eedca48f97e1865e"), x: 1 },
     { _id: ObjectId("652567e4eedca48f97e1865f"), x: 2 },
     { _id: ObjectId("652567e4eedca48f97e18660"), x: 3 },
     { _{id}: ObjectId("652567e4eedca48f97e18661"), x: 4 },
    __id: ObjectId("652567e4eedca48f97e18662"), x: 5 ,
{ _id: ObjectId("652567e4eedca48f97e18663"), x: 6 },
     { _id: ObjectId("652567e4eedca48f97e18664"), x: 7 },
     { _id: ObjectId("652567e4eedca48f97e18665"), x: 8 },
     { _id: ObjectId("652567e4eedca48f97e18666"), x: 9 },
     { _id: ObjectId("652567e4eedca48f97e18667"), x: 10 },
     { _id: ObjectId("652567e4eedca48f97e18668"), x: 11 },
     { _id: ObjectId("652567e4eedca48f97e18669"), x: 12 },
     { _id: ObjectId("652567e4eedca48f97e1866a"), x: 13 },
     { _id: ObjectId("652567e4eedca48f97e1866b"), x: 14 },
     { _id: ObjectId("652567e4eedca48f97e1866c"), x: 15 },
     { _id: ObjectId("652567e4eedca48f97e1866d"), x: 16 },
     { _id: ObjectId("652567e4eedca48f97e1866e"), x: 17 },
    { _id: ObjectId("652567e4eedca48f97e1866f"), x: 18 },
{ _id: ObjectId("652567e4eedca48f97e18670"), x: 19 },
     { _id: ObjectId("652567e4eedca48f97e18671"), x: 20 }
You will have different _id values.
```

Now your cluster has some data in it.

Next steps

Make a backup →

4.5 4. Make a backup

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

Considerations and prerequisites

In this tutorial we use the AWS S3 [2] as the backup storage. You need the following S3-related information:

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

If you don't have access to AWS, you can use any S3-compatible storage like MinIO C3. Also check the list of supported storages.

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

```
$ git clone -b v1.20.0 https://github.com/percona/percona-server-mongodb-operator
$ cd percona-server-mongodb-operator
```



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

Configure backup storage

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

on Linux

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

on MacOS

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

- 2 Edit the deploy/backup-s3.yaml C example Secrets configuration file and specify the following:
 - the metadata.name key is the name which you use to refer your Kubernetes Secret
 - the base64-encoded S3 credentials

deploy/backup-s3.yaml

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

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

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

- 4 Update your deploy/cr.yaml configuration. Specify the following parameters in the backups section:
 - set the storages.<NAME>.type to s3. Substitute the <NAME> part with some arbitrary name that you will later use to refer this storage when making backups and restores.
 - set the storages.<NAME>.s3.credentialsSecret to the name you used to refer your Kubernetes Secret (my-cluster-name-backup-s3 in the previous step).
 - specify the S3 bucket name for the storages.<NAME>.s3.bucket option
 - specify the region in the storages.<NAME>.s3.region option. Also you can use the storages.<NAME>.s3.prefix option to specify the path (a subfolder) to the backups inside the S3 bucket. If prefix is not set, backups are stored in the root directory.

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

If you use a different S3-compatible storage instead of AWS S3, add the endpointURL key in the s3 subsection, which should point to the actual cloud used for backups. This value is specific to the cloud provider. For example, using Google Cloud involves the following endpointUr1:

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

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

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

Make a logical backup

- Before you start, verify the backup configuration in the deploy/cr.yaml file:
 - the backup.enabled key is set to true
 - the backup.storages subsection contains the configured storage.
- 2 To make a backup, you need the configuration file. Edit the sample deploy/backup/backup.yaml 🖒 configuration file and specify the following:
 - metadata.name specify the backup name. You will use this name to restore from this backup
 - spec.clusterName specify the name of your cluster. This is the name you specified when deploying Percona Server for MongoDB.
 - spec.storageName specify the name of your already configured storage.

deploy/backup/backup.yaml

```
apiVersion: psmdb.percona.com/v1
kind: PerconaServerMongoDBBackup
metadata:
    finalizers:
        - percona.com/delete-backup
    name: backup1
spec:
    clusterName: my-cluster-name
    storageName: s3-us-west
    type: logical
```

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

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

4 Track the backup progress.

\$ kubectl get psmdb-backup -n <namespace>



When the status changes to Ready, backup is made.

Troubleshooting

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

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

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

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

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

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

Next steps

Monitor the database \rightarrow

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

In this section you will learn how to monitor Percona Server for MongoDB with Percona Monitoring and Management (PMM) [4].



Only PMM 2.x versions are supported by the Operator.

PMM is a client/server application. It includes the PMM Server and the number of PMM Clients are running on each node with the database you wish to monitor.

A PMM Client collects needed metrics and sends gathered data to the PMM Server. As a user, you connect to the PMM Server to see database metrics on a number of dashboards [3].

PMM Server and PMM Client are installed separately.

Install PMM Server

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

Install PMM Client

To install PMM Client as a side-car container in your Kubernetes-based environment, do the following:

Authorize PMM Client within PMM Server.

Token-based authorization (recommended)

- 1. Generate the PMM Server API Key . Specify the Admin role when getting the API Key.
- ▲ Warning: The API key is not rotated automatically.
- 1 Edit the deploy/secrets.yaml 🚰 secrets file and specify the PMM API key for the PMM_SERVER_API_KEY option.
- 2 Apply the configuration for the changes to take effect.

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

Password-based authorization (deprecated since version 1.13.0)

- Edit the <u>deploy/secrets.yaml</u> secrets file and specify the following:
- 2 The user name of your PMM Server (admin by default) in the PMM_SERVER_USER key
- The password you set for the PMM Server during its installation in the PMM_SERVER_PASSWORD key.
- Apply the configuration for the changes to take effect.

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

- 2 Update the pmm section in the deploy/cr.yaml C file:
 - Set pmm.enabled = true.
 - 🕞 Specify your PMM Server hostname / an IP address for the pmm.serverHost option. The PMM Server IP address should be resolvable and reachable from within your cluster.

pmm:

enabled: true

image: percona/pmm-client:2.44.1
serverHost: monitoring-service

3. Apply the changes:

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

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

```
$ kubectl get pods -n <namespace>
$ kubectl logs <cluster-name>-rs0-0 -c pmm-client -n <namespace>
```

Check the metrics

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

- 1 Log in to PMM server.
- 2 Click MongoDB from the left-hand navigation menu. You land on the Instances Overview page.
- 3 Select your cluster from the Clusters drop-down menu and the desired time range on the top of the page. You should see the metrics.
- 4 Click NongoDB → Other dashboards to see the list of available dashboards that allow you to drill down to the metrics you are interested in.

Next steps

What's next →

4.7 What's next?

Congratulations! You have completed all the steps in the Get started guide.

You have the following options to move forward with the Operator:

- Deepen your monitoring insights by setting up <u>Kubernetes monitoring with PMM</u>
- Control Pods assignment on specific Kubernetes Nodes by setting up <u>affinity / anti-affinity</u>.
- Ready to adopt the Operator for production use and need to delete the testing deployment? Use this guide to do it
- You can also try operating the Operator and database clusters via the web interface with <u>Percona Everest</u> C an open-source web-based database provisioning tool based on Percona Operators. See <u>Get started with Percona Everest</u> C on how to start using it

5 Installation

5.1 System Requirements

The Operator was developed and tested with Percona Server for MongoDB 6.0.21-18, 7.0.18-11, and 8.0.8-3. Other options may also work but have not been tested. The Operator 1.20.0 also uses Percona Backup for MongoDB 2.9.1.

Officially supported platforms

The following platforms were tested and are officially supported by the Operator 1.20.0:

- Google Kubernetes Engine (GKE) 1.30-1.32
- Amazon Elastic Container Service for Kubernetes (EKS) [1.30-1.32
- OpenShift Container Platform 4.14 4.18
- Azure Kubernetes Service (AKS) 1.30-1.32
- Minikube 1.35.0 based on Kubernetes 1.32.0

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

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.



Use Storage Class with XFS as the default filesystem if possible to achieve better MongoDB performance [2].

Installation guidelines

Choose how you wish to install the Operator:

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

5.2 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 [2], 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 <u>using</u> the following command:

 $\$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/bundle.yaml

3. Deploy MongoDB cluster with:

 $\$ \ kubect1 \ apply \ -f \ https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr-minimal.yamlagenergy \ apply \ -f \ https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr-minimal.yamlagenergy \ -f \ https://raw.githubusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr-minimal.yamlagenergy \ -f \ https://raw.githubusercontent.com/percona$

Note

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

\$ git clone -b v1.20.0 https://github.com/percona/percona-server-mongodb-operator

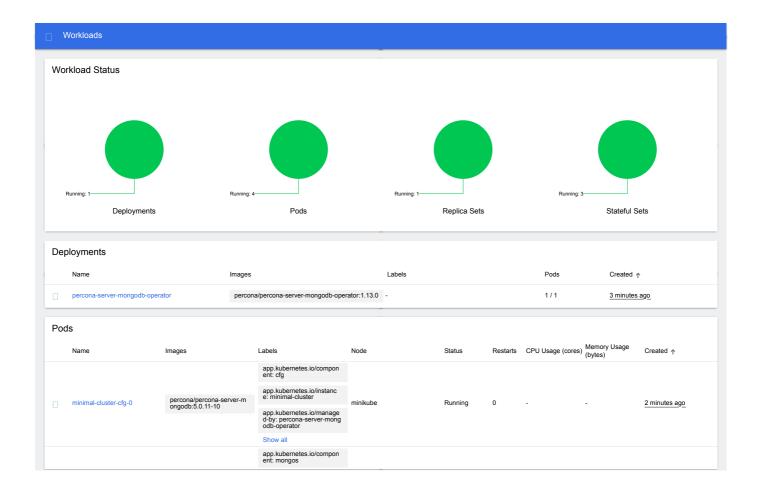
After editing the needed options, apply your modified deploy/cr.yaml file as follows:

\$ kubectl apply -f deploy/cr.yaml

The creation process may take some time.

The process is over when both operator and replica set pod have reached their Running status. kubect1 get pods output should look like this:

You can also track the progress via the Kubernetes dashboard:



Verifying the cluster operation

It may take ten minutes to get the cluster started. When kubect1 get pods command finally shows you the cluster is ready, you can try to connect to the cluster.

To connect to Percona Server for MongoDB you need to construct the MongoDB connection URI string. It includes the credentials of the admin user, which are stored in the Secrets object.

1. List the Secrets objects

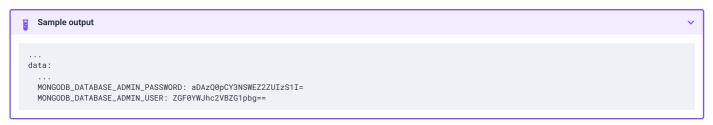
```
$ kubectl get secrets -n <namespace>
```

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

2. View the Secret contents to retrieve the admin user credentials.

```
$ kubectl get secret minimal-cluster-secrets -o yaml
```

The command returns the YAML file with generated Secrets, including the MONGODB_DATABASE_ADMIN_USER and MONGODB_DATABASE_ADMIN_PASSWORD strings, which should look as follows:



The actual login name and password on the output are base64-encoded. To bring it back to a human-readable form, run:

```
$ echo 'MONGODB_DATABASE_ADMIN_USER' | base64 --decode
$ echo 'MONGODB_DATABASE_ADMIN_PASSWORD' | base64 --decode
```

3. Run a container with a MongoDB client and connect its console output to your terminal. The following command does this, naming the new Pod perconaclient:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:7.0.18-11 --restart=Never -- bash -il
```

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

4. Now run mongosh tool inside the percona-client command shell using the admin user credentialds you 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

 $\label{lem:mongodb://databaseAdminPassword@minimal-cluster-mongos.} $$\operatorname{mongodb://databaseAdminPassword@minimal-cluster-mongos.} $$\operatorname{mongodb://databaseAdminPassword@minimal-cluster-mongos.} $$\operatorname{mongodb://databaseAdminPassword@minimal-cluster-mongos.} $$\operatorname{mongodb://databaseAdminPassword@minimal-cluster-mongos.} $$\operatorname{mongodb://databaseAdminPassword@minimal-cluster-mongos.} $$$

if sharding is off

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

5.3 Install Percona Server for MongoDB cluster using Everest

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

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

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

5.4 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 <u>Kubernetes Engine Quickstart</u>.

Prerequisites

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

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

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

- 1. gcloud [7]. This tool is part of the Google Cloud SDK. To install it, select your operating system on the official Google Cloud SDK documentation page [7] and then follow the instructions.
- 2. kubectl. C. It is the Kubernetes command-line tool you will use to manage and deploy applications. To install the tool, run the following command:

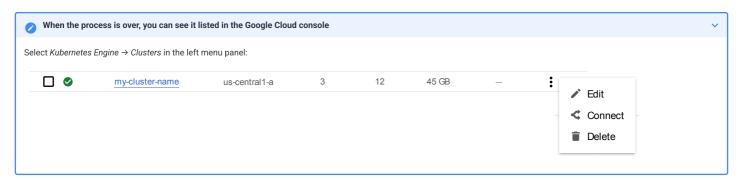
```
$ gcloud auth login
$ gcloud components install kubectl
```

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:



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 kubect1 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 <u>Cloud Identity and Access Management (Cloud IAM)</u> of 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)



 $\verb|cluster-olebinding.rbac.authorization.k8s.io/cluster-admin-binding| | created| \\$

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.

 $\label{thm:continuous} \mbox{Deploy the Operator by applying the $\deg \log \beta$ undle . yaml $\mbox{manifest from the Operator source tree}. $$

For x86_64 architecture

You can apply it without downloading, using the following command:

\$ kubect1 apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/bundle.yaml



For ARM64 architecture

Clone the repository with all manifests and source code by executing the following command:

```
$ git clone -b v1.20.0 https://github.com/percona/percona-server-mongodb-operator
```

Edit the deploy/bundle.yaml file: add the following affinity rules to the spec part of the percona-server-mongodb-operator Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: percona-server-mongodb-operator
spec:
  replicas: 1
  selector
    matchLabels:
      name: percona-server-mongodb-operator
  template:
    metadata:
      labels:
        name: percona-server-mongodb-operator
    spec:
      affinity:
       nodeAffinity:
          required {\tt DuringSchedulingIgnoredDuringExecution:}
            nodeSelectorTerms:
               - matchExpressions:
                  key: kubernetes.io/arch
                  operator: In
                  values:
                     - arm64
```

After editing, apply 🔀 your modified deploy/bundle.yaml file as follows:

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

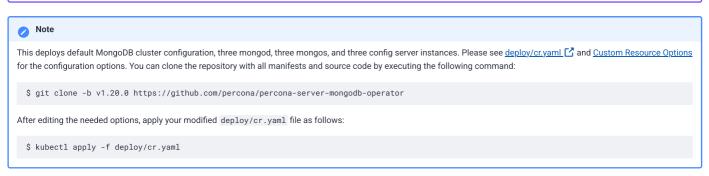


2. The Operator has been started, and you can deploy your MongoDB cluster:

For x86_64 architecture

 $\$ \ kubect1 \ apply \ -f \ https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamluser-mongodb-operator/v1.20.0/deploy/$





For ARM64 architecture

Edit the deploy/cr.yaml file: set the following affinity rules in all affinity subsections:

```
affinity:
advanced:
nodeAffinity:
requiredDuringSchedulingIgnoredDuringExecution:
nodeSelectorTerms:
- matchExpressions:
- key: kubernetes.io/arch
operator: In
values:
- arm64
```

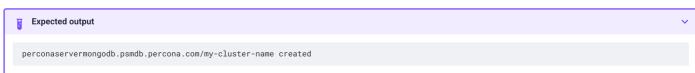
Also, set image and backup.image Custom Resource options to special multi-architecture image versions by adding a -multi suffix to their tags:

```
image: percona/percona-server-mongodb:7.0.18-11-multi
...
backup:
...
image: percona/percona-backup-mongodb:2.9.1-multi
```

Please note, that currently <u>monitoring with PMM</u> is not supported on ARM64 configurations.

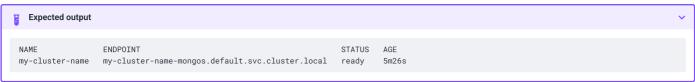
After editing, apply your modified deploy/cr.yaml file as follows:

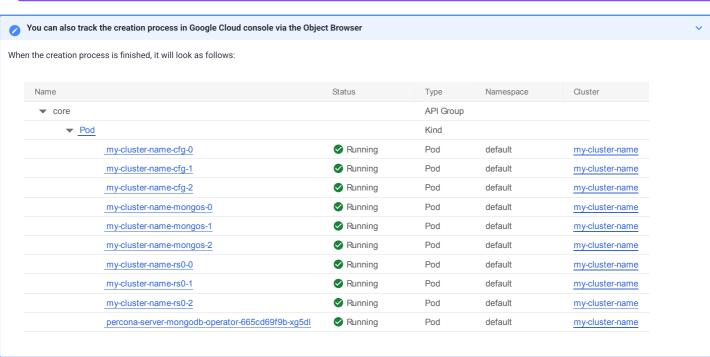
```
$ kubectl apply -f deploy/cr.yaml
```



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
```





Verifying the cluster operation

It may take ten minutes to get the cluster started. When kubect1 get psmdb command finally shows you the cluster status as ready, you can try to connect to the cluster.

To connect to Percona Server for MongoDB you need to construct the MongoDB connection URI string. It includes the credentials of the admin user, which are stored in the Secrets object.

1. List the Secrets objects

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

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

2. View the Secret contents to retrieve the admin user credentials.

```
$ kubectl get secret my-cluster-name-secrets -o yaml
```

The command returns the YAML file with generated Secrets, including the MONGODB_DATABASE_ADMIN_USER and MONGODB_DATABASE_ADMIN_PASSWORD strings, which should look as follows:



The actual login name and password on the output are base64-encoded. To bring it back to a human-readable form, run:

```
$ echo 'MONGODB_DATABASE_ADMIN_USER' | base64 --decode
$ echo 'MONGODB_DATABASE_ADMIN_PASSWORD' | base64 --decode
```

3. Run a container with a MongoDB client and connect its console output to your terminal. The following command does this, naming the new Pod perconaclient:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:7.0.18-11 --restart=Never -- bash -il
```

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

4. Now run mongosh tool inside the percona-client command shell using the admin user credentialds you obtained from the Secret, and a proper namespace name instead of the <namespace name> placeholder. The command will look different depending on whether sharding is on (the default behavior) or off:

if sharding is on

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

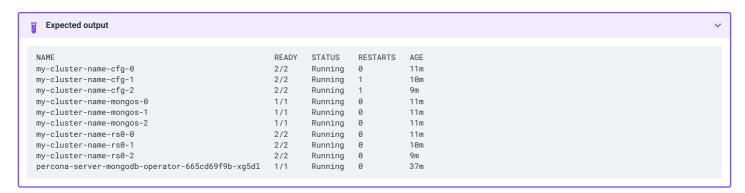
if sharding is off

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

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

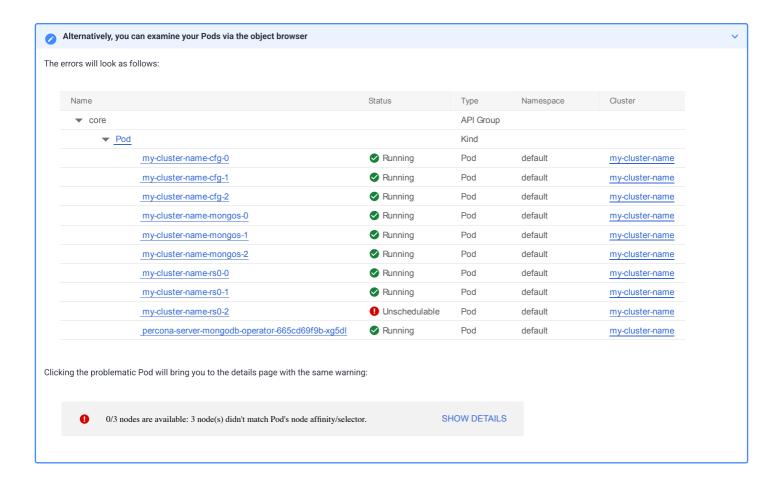


If the command output had shown some errors, you can examine the problematic Pod with the kubectl describe <pod name> command as follows:

```
$ 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.



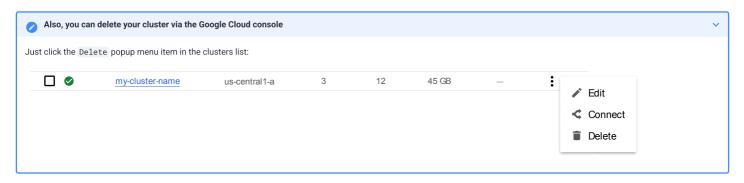
Removing the GKE cluster

There are several ways that you can delete the cluster.

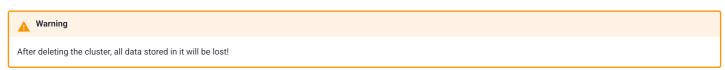
You can clean up the cluster with the gcloud command as follows:

\$ gcloud container clusters delete <cluster name> --zone us-central1-a --project Froject ID>

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



The cluster deletion may take time.



5.5 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 <u>Amazon EKS official documentation</u> C.

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].
- 2. eksctl to simplify cluster creation on EKS. It can be installed along its installation notes on GitHub [2].
- 3. kubectl to manage and deploy applications on Kubernetes. Install it following the official installation instructions [].

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

Create the EKS cluster

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



spot [] instances are not recommended for production environment, but may be useful e.g. for testing purposes.

After you have settled all the needed details, create your EKS cluster following the official cluster creation instructions [].

2. After you have created the EKS cluster, you also need to install the Amazon EBS CSI driver on your cluster. See the official documentation on adding it as an Amazon EKS add-on.

Install the Operator and deploy your MongoDB cluster

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

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

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

Deploy the Operator by applying the deploy/bundle.yaml manifest from the Operator source tree.

For x86_64 architecture

You can apply it without downloading, using the following command:

\$ kubect1 apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-mongodboperator/v1.20.0/deploy/bundle.yaml



For ARM64 architecture

Clone the repository with all manifests and source code by executing the following command:

```
$ git clone -b v1.20.0 https://github.com/percona/percona-server-mongodb-operator
```

Edit the deploy/bundle.yaml file: add the following affinity rules to the spec part of the percona-server-mongodb-operator Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: percona-server-mongodb-operator
spec:
  replicas: 1
  selector
   matchLabels:
      name: percona-server-mongodb-operator
  template:
    metadata:
      labels:
        name: percona-server-mongodb-operator
    spec:
      affinity:
       nodeAffinity:
          required {\tt DuringSchedulingIgnoredDuringExecution:}
            nodeSelectorTerms:
               - matchExpressions:
                  key: kubernetes.io/arch
                  operator: In
                  values:
                     - arm64
```

After editing, apply your modified deploy/bundle.yaml file as follows:

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

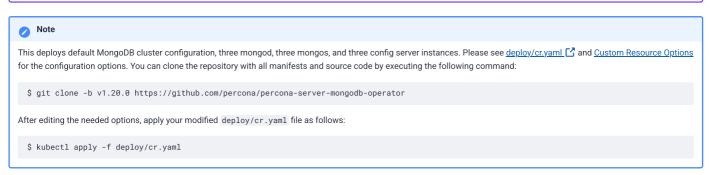


2. The Operator has been started, and you can deploy your MongoDB cluster:

For x86_64 architecture

 $\$ \ kubect1 \ apply \ -f \ https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamlusercontent.com/percona-server-mongodb-operator/v1.20.0/deploy/cr.yamluser-mongodb-operator/v1.20.0/deploy/$





For ARM64 architecture

Edit the deploy/cr.yaml file: set the following affinity rules in all affinity subsections:

```
affinity:
advanced:
nodeAffinity:
requiredDuringSchedulingIgnoredDuringExecution:
nodeSelectorTerms:
- matchExpressions:
- key: kubernetes.io/arch
operator: In
values:
- arm64
```

Also, set image and backup.image Custom Resource options to special multi-architecture image versions by adding a -multi suffix to their tags:

```
image: percona/percona-server-mongodb:7.0.18-11-multi
...
backup:
...
image: percona/percona-backup-mongodb:2.9.1-multi
```

Please note, that currently monitoring with PMM is not supported on ARM64 configurations.

After editing, 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
```



Verifying the cluster operation

It may take ten minutes to get the cluster started. When kubect1 get psmdb command finally shows you the cluster status as ready, you can try to connect to the cluster.

To connect to Percona Server for MongoDB you need to construct the MongoDB connection URI string. It includes the credentials of the admin user, which are stored in the Secrets \(\text{S} \) object.

1. List the Secrets objects

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

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

2. View the Secret contents to retrieve the admin user credentials.

```
$ kubectl get secret my-cluster-name-secrets -o yaml
```

The command returns the YAML file with generated Secrets, including the MONGODB_DATABASE_ADMIN_USER and MONGODB_DATABASE_ADMIN_PASSWORD strings, which should look as follows:



The actual login name and password on the output are base64-encoded. To bring it back to a human-readable form, run:

```
$ echo 'MONGODB_DATABASE_ADMIN_USER' | base64 --decode
$ echo 'MONGODB_DATABASE_ADMIN_PASSWORD' | base64 --decode
```

3. Run a container with a MongoDB client and connect its console output to your terminal. The following command does this, naming the new Pod perconaclient:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:7.0.18-11 --restart=Never -- bash -il
```

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

4. Now run mongosh tool inside the percona-client command shell using the admin user credentialds you 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

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

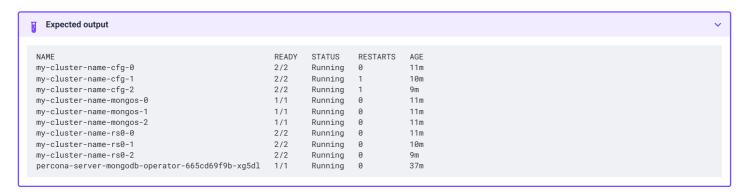
if sharding is off

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

Troubleshooting

If kubect1 get psmdb command doesn't show ready status too long, you can check the creation process with the kubect1 get pods command:

\$ kubectl get pods



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.

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.



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

5.6 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 .

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 <u>official installation instructions for</u> your system [3].
- 2. **kubectl** to manage and deploy applications on Kubernetes. Install it following the official installation instructions .

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

Create and configure the AKS cluster

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

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

You can create your cluster via command line using az aks create command. The following command will create a 3-node cluster named my-cluster-name within some <u>already existing</u> C 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 x86_64 machine type of Standard_B4ms \(\text{S} \) and OS disk size reduced to 30 GiB. If you need ARM64, use different machine type, for example, Standard_D4ps_v5. You can see detailed information about cluster creation options in the AKS official documentation \(\text{S} \).

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 kubect1 be able to use it.

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

Install the Operator and deploy your MongoDB cluster

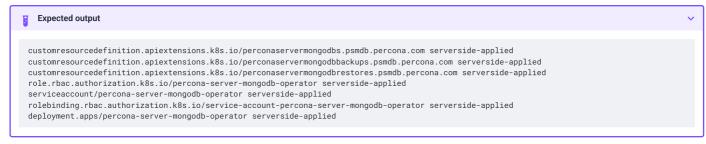
1. Deploy the Operator. By default deployment will be done in the default namespace. If that's not the desired one, you can create a new namespace and/or set the context for the namespace as follows (replace the <namespace name> placeholder with some descriptive name):

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

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

Deploy the Operator, using 'the following command:

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-mongodb-
operator/v1.20.0/deploy/bundle.yaml
```

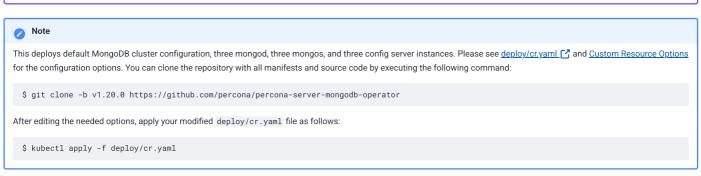


2. The Operator has been started, and you can deploy your MongoDB cluster:

For x86_64 architecture

\$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/cr.yaml

Expected output perconaservermongodb.psmdb.percona.com/my-cluster-name created



For ARM64 architecture

Clone the repository with all manifests and source code by executing the following command:

```
$ git clone -b v1.20.0 https://github.com/percona/percona-server-mongodb-operator
```

Edit the deploy/cr.yaml configuration file: set image and backup.image Custom Resource options to special multi-architecture image versions by adding a -multi suffix to their tags:

```
image: percona/percona-server-mongodb:7.0.18-11-multi
...
backup:
...
image: percona/percona-backup-mongodb:2.9.1-multi
```

Please note, that currently monitoring with PMM is not supported on ARM64 configurations.

After editing, apply your modified deploy/cr.yaml file as follows:

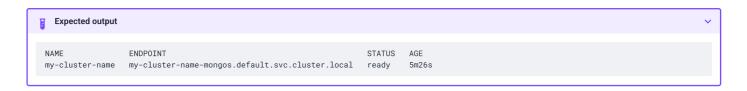
```
$ kubectl apply -f deploy/cr.yaml

Expected output

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

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
```



Verifying the cluster operation

It may take ten minutes to get the cluster started. When kubect1 get psmdb command finally shows you the cluster status as ready, you can try to connect to the cluster.

To connect to Percona Server for MongoDB you need to construct the MongoDB connection URI string. It includes the credentials of the admin user, which are stored in the Secrets \(\text{S} \) object.

1. List the Secrets objects

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

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

2. View the Secret contents to retrieve the admin user credentials.

```
$ kubectl get secret my-cluster-name-secrets -o yaml
```

The command returns the YAML file with generated Secrets, including the MONGODB_DATABASE_ADMIN_USER and MONGODB_DATABASE_ADMIN_PASSWORD strings, which should look as follows:



The actual login name and password on the output are base64-encoded. To bring it back to a human-readable form, run:

```
$ echo 'MONGODB_DATABASE_ADMIN_USER' | base64 --decode
$ echo 'MONGODB_DATABASE_ADMIN_PASSWORD' | base64 --decode
```

3. Run a container with a MongoDB client and connect its console output to your terminal. The following command does this, naming the new Pod perconaclient:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:7.0.18-11 --restart=Never -- bash -il
```

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

4. Now run mongosh tool inside the percona-client command shell using the admin user credentialds you 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

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

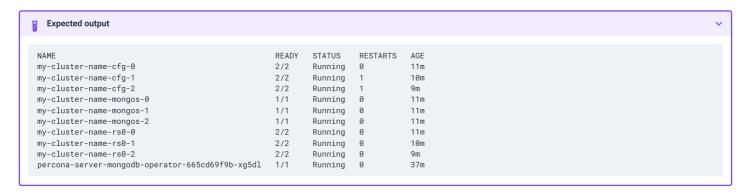
if sharding is off

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

Troubleshooting

If kubect1 get psmdb command doesn't show ready status too long, you can check the creation process with the kubect1 get pods command:

\$ kubectl get pods



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.

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.



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

5.7 Install Percona server for MongoDB on Kubernetes

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

```
$ git clone -b v1.20.0 https://github.com/percona/percona-server-mongodb-operator
$ cd percona-server-mongodb-operator
```



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 sa follows:

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

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

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

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

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

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

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

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



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 C 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 <u>TLS instructions</u>.
- 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. 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

Verifying the cluster operation

It may take ten minutes to get the cluster started. When kubect1 get psmdb command finally shows you the cluster status as ready, you can try to connect to the cluster.

To connect to Percona Server for MongoDB you need to construct the MongoDB connection URI string. It includes the credentials of the admin user, which are stored in the Secrets object.

1. List the Secrets objects

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

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

2. View the Secret contents to retrieve the admin user credentials.

```
$ kubectl get secret my-cluster-name-secrets -o yaml
```

The command returns the YAML file with generated Secrets, including the MONGODB_DATABASE_ADMIN_USER and MONGODB_DATABASE_ADMIN_PASSWORD strings, which should look as follows:



The actual login name and password on the output are base64-encoded. To bring it back to a human-readable form, run:

```
$ echo 'MONGODB_DATABASE_ADMIN_USER' | base64 --decode
$ echo 'MONGODB_DATABASE_ADMIN_PASSWORD' | base64 --decode
```

3. Run a container with a MongoDB client and connect its console output to your terminal. The following command does this, naming the new Pod perconaclient:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:7.0.18-11 --restart=Never -- bash -il
```

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

4. Now run mongosh tool inside the percona-client command shell using the admin user credentialds you obtained from the Secret, and a proper namespace name instead of the <namespace name> placeholder. The command will look different depending on whether sharding is on (the default behavior) or off:

if sharding is on

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

if sharding is off

 $\label{lem:condition} $$ mongodh+srv://databaseAdmin:databaseAdminPassword@my-cluster-name-rs0.<namespace name>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"$

5.8 Install Percona Server for MongoDB on OpenShift

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

Installing Percona Server for MongoDB on OpenShift includes two steps:

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

Install the Operator

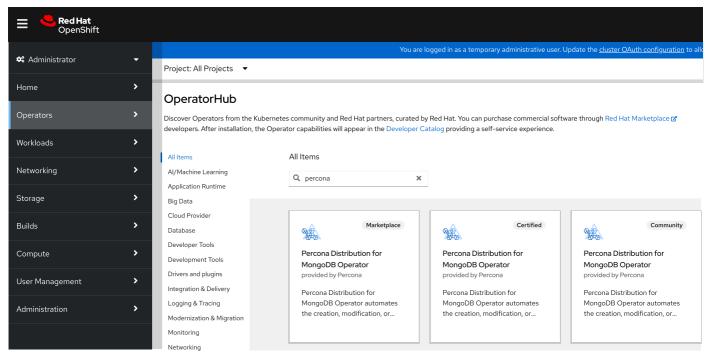
You can install Percona Operator for MongoDB on OpenShift using the web interface (the Operator Lifecycle Manager C), or using the command line interface.

Install the Operator via the Operator Lifecycle Manager (OLM)

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

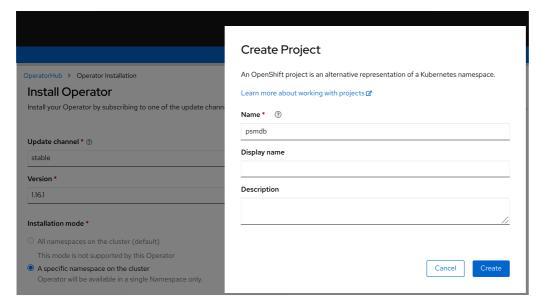
Following steps will allow you to deploy the Operator and Percona Server for MongoDB on your OLM installation:

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



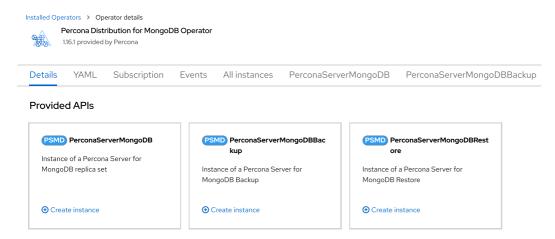
Then click "Contiune", and "Install".

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



Click "Install" button to actually install the Operator.

3. When the installation finishes, you can deploy your MongoDB cluster. In the "Operator Details" you will see Provided APIs (Custom Resources, available for installation). Click "Create instance" for the PerconaServerMongoDB Custom Resource.



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

Install the Operator via the command-line interface

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

\$ git clone -b v1.20.0 https://github.com/percona/percona-server-mongodb-operator
\$ cd percona-server-mongodb-operator

Note

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

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

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

Apply it
☐ as follows:

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

```
Note
```

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

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

```
$ oc create clusterrole psmdb-admin --verb="*" --
resource = perconaser vermongodbs.psmdb.percona.com, perconaser vermongodbs.psmdb.percona.com/status, perconaser vermongodbbackups and the perconaser vermongodbs.psmdb.perconaser vermongodbs.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.psmdb.
 . \verb|psmdb.percona.com|, perconaser vermong odbbackups.psmdb.percona.com|, status, perconaser vermong odbrestores.psmdb.percona.com|, perconaser vermong odbrestores.psmdb.perconaser vermong observation odbrestores
onaservermongodbrestores.psmdb.percona.com/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 <a>C¹.

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

5. Start the Operator within OpenShift:

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

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 [4] for details).

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

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

More details about secrets can be found in Users.

- 2. Now certificates should be generated. By default, the Operator generates certificates automatically, and no actions are required at this step. Still, you can generate and apply your own certificates as secrets according to the TLS instructions.
- 3. Percona Server for MongoDB cluster can be created at any time with the following steps:
 - a. Uncomment the deploy/cr.yaml field #platform: and edit the field to platform: openshift. The result should be like this:

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

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

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

c. Create/apply the Custom Resource file:

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

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

$ oc get psmdb

| Expected output | STATUS | AGE | my-cluster-name | my-cluster-nam
```

Verifying the cluster operation

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

To connect to Percona Server for MongoDB you need to construct the MongoDB connection URI string. It includes the credentials of the admin user, which are stored in the Secrets 2 object.

1. List the Secrets objects

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

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

2. View the Secret contents to retrieve the admin user credentials.

```
$ oc get secret my-cluster-name-secrets -o yaml
```

The command returns the YAML file with generated Secrets, including the MONGODB_DATABASE_ADMIN_USER and MONGODB_DATABASE_ADMIN_PASSWORD strings, which should look as follows:

```
Sample output

...
data:
...
MONGODB_DATABASE_ADMIN_PASSWORD: aDAZQ0pCY3NSWEZ2ZUIZS1I=
MONGODB_DATABASE_ADMIN_USER: ZGF0YWJhc2VBZG1pbg==
```

The actual login name and password on the output are base64-encoded. To bring it back to a human-readable form, run:

```
$ echo 'MONGODB_DATABASE_ADMIN_USER' | base64 --decode
$ echo 'MONGODB_DATABASE_ADMIN_PASSWORD' | base64 --decode
```

3. Run a container with a MongoDB client and connect its console output to your terminal. The following command does this, naming the new Pod perconaclient:

```
$ oc run -i --rm --tty percona-client --image=percona/percona-server-mongodb:7.0.18-11 --restart=Never -- bash -il
```

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

4. Now run mongosh tool inside the percona-client command shell using the admin user credentialds you obtained from the Secret, and a proper namespace name instead of the <namespace name> placeholder. The command will look different depending on whether sharding is on (the default behavior) or off:

if sharding is on

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

if sharding is off

 $\label{lem:condition} $$ mongodh+srv://databaseAdmin:databaseAdminPassword@my-cluster-name-rs0.<namespace name>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"$

6 Upgrade

6.1 Update Percona Operator for MongoDB

You can upgrade Percona Operator for MongoDB to newer versions. The upgrade process consists of these steps:

- · Upgrade the Operator
- Upgrade the database (Percona Server for MongoDB)

Update scenarios

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

Full upgrade (CRD, Operator, and the database).

When to use this scenario:

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

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

Upgrade only the database

When to use this scenario:

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

When choosing this scenario, consider the following:

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

Update strategies

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

- Smart Update is the automated way to update the database cluster. The Operator controls how objects are updated. It restarts Pods in a specific order, with the primary instance updated last to avoid connection issues until the whole cluster is updated to the new settings.
 - This update method applies during database upgrades and when making changes like updating a ConfigMap, rotating passwords, or changing resource values. It is the default and recommended way to update.
- Rolling Update is initiated manually and controlled by Kubernetes. [2]. The StatefulSet controller in Kubernetes deletes a Pod, updates it, waits till it reports the Ready status and proceeds to the next Pod. The order for Pod update is the same as for Pod termination. However, this order may not be optimal from the Percona Server for MongoDB point of view.
- On Delete strategy requires a user to manually delete a Pod to make Kubernetes StatefulSet controller recreate it with the updated configuration [4].

To select an update strategy, set the updateStrategy key in the <u>Custom Resource</u> manifest to one of the following:

- SmartUpdate
- RollingUpdate
- OnDelete

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

Update on OpenShift

If you run the Operator on Red Hat Marketplace or you run Red Hat certified Operators on OpenShift or you need to do additional steps during the upgrade. See Upgrade Percona Server for MongoDB on OpenShift for details.

6.2 Upgrade the Operator and CRD

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

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

Considerations

- 1. The Operator version has three digits separated by a dot (.) in the format major .minor .patch . Here's how you can understand the version 1.18.0:
 - 1 major version
 - 18 minor version
 - 0 patch version

You can only upgrade the Operator to the nearest major.minor version. For example, from 1.18.0 to 1.19.0. To upgrade to a newer version, which differs from the current minor.major version by more than one, you need to make several incremental upgrades sequentially. For example, to upgrade the Operator from 1.17.0 to 1.19.1, you need to first upgrade it to 1.18.0, and then to 1.19.1.

Patch versions don't influence the upgrade, so you can safely move from 1.18.0 to 1.19.1.

Check the Release notes index for the list of the Operator versions.

- 2. CRD supports the last 3 minor versions of the Operator. This means it is compatible with the newest Operator version and the two previous minor versions. If the Operator is older than the CRD by no more than two versions, you should be able to continue using the old Operator version. But updating the CRD and Operator is the recommended path.
- 3. Starting from version 1.14.0, the Operator configures replica set members using local fully-qualified domain names (FQDN). Before this version, if you exposed a replica set, it used the exposed IP addresses in the replica set configuration. Therefore, if you upgrade to version 1.14.0 and your replica set is exposed, the replica set configuration will change to use FQDN. To prevent such reconfiguration, set the clusterServiceDNSMode Custom Resource option to External before the upgrade.
- 4. Starting with version 1.12.0, the Operator no longer has a separate API version for each release in CRD. Instead, the CRD has the API version v1. Therefore, if you installed the CRD when the Operator version was **older than 1.12.0**, you must update the API version in the CRD manually to run the upgrade. To check your CRD version, use the following command:

\$ kubectl get crd perconaservermongodbs.psmdb.percona.com -o yaml | yq .status.storedVersions



If the CRD version is other than v1 or has several entries, run the manual update.

5. Starting from the Operator version 1.15.0, the spec.mongod section (deprecated since 1.12.0) is finally removed from the Custom Resource configuration. If you have encryption disabled using the deprecated mongod.security.enableEncryption option, you need to set encryption as disabled via the <a href="mailto:custom-custo

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

6. Starting from the Operator version 1.16.0, MongoDB 4.4 support in the Operator has reached its end-of-life. Make sure that you have a supported MongoDB version before upgrading the Operator to 1.16.0 (you can use <u>major version upgrade functionality</u> to fix it.

7. The Operator versions 1.19.0 and 1.19.1 have a recommended MongoDB version set to 7.0 because point-in-time recovery may fail on MongoDB 8.0 if sharding is enabled and the Operator version is 1.19.x. Therefore, upgrading to Operator 1.19.0/1.19.1 is not recommended for sharded MongoDB 8.0 clusters.

Upgrade manually

The upgrade includes the following steps.

1. For Operators older than v1.12.0: Update the API version in the Custom Resource Definition [4]:

Manually

Via kubectl patch

\$ kubectl patch customresourcedefinitions perconaservermongodbs.psmdb.percona.com --subresource='status' --type='merge' -p
'{"status":{"storedVersions":["v1"]}}'

```
Expected output

customresourcedefinition.apiextensions.k8s.io/perconaservermongodbs.psmdb.percona.com patched
```

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

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-mongodb-
operator/v1.20.0/deploy/crd.yaml
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/rbac.yaml
```

3. Next, update the Percona Server for MongoDB Operator Deployment in Kubernetes by changing the container image of the Operator Pod to the latest version. Find the image name for the current Operator release in the list of certified images. Then apply a patch of to the Operator Deployment and specify the image name and version. Use the following command to update the Operator Deployment to the 1.20.0 version:

```
$ kubectl patch deployment percona-server-mongodb-operator \
    -p'{"spec":{"template":{"spec":{"containers":[{"name":"percona-server-mongodb-operator","image":"percona/percona-server-mongodb-operator:1.20.0"}]}}}}'
```

4. The deployment rollout will be automatically triggered by the applied patch. You can track the rollout process in real time with the kubect1 rollout status command with the name of your cluster:

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



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

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

Find the image names in the list of certified images.

We recommend to update the PMM Server **before** the upgrade of PMM Client. If you haven't done it yet, exclude PMM Client from the list of images to update.

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

For example, to update the cluster with the name my-cluster-name to the 1.20.0 version, the command is as follows:

With PMM Client

```
$ kubectl patch psmdb my-cluster-name --type=merge --patch '{
   "spec": {
      "crVersion":"1.20.0",
      "image": "percona/percona-server-mongodb:7.0.18-11",
      "backup": { "image": "percona/percona-backup-mongodb:2.9.1" },
      "pmm": { "image": "percona/pmm-client:2.44.1" }
}}'
```

Without PMM Client

```
$ kubectl patch psmdb my-cluster-name --type=merge --patch '{
   "spec": {
      "crVersion":"1.20.0",
      "image": "percona/percona-server-mongodb:7.0.18-11",
      "backup": { "image": "percona/percona-backup-mongodb:2.9.1" }
}}'
```

6.

Upgrade via helm

If you have $\underline{\text{installed the Operator using Helm}}$, you can upgrade the Operator with the $\underline{\text{helm}}$ upgrade command.

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

```
$ kubectl apply --server-side -f https://raw.githubusercontent.com/percona/percona-server-mongodb-
operator/v1.20.0/deploy/crd.yaml
$ kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.20.0/deploy/rbac.yaml
```

2. If you installed the Operator with no <u>customized parameters</u> [], the upgrade can be done as follows:

```
$ helm upgrade my-op percona/psmdb-operator --version 1.20.0
```

The my-op parameter in the above example is the name of a <u>release object</u> which which you have chosen for the Operator when installing its Helm chart. If the Operator was installed with some <u>customized parameters</u> , you should list these options in the upgrade command.

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

```
$ helm upgrade my-op percona/psmdb-operator --version 1.20.0 -f my-values.yaml
```

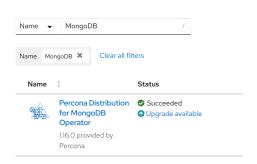
Upgrade via Operator Lifecycle Manager (OLM)

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

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

Installed Operators

Installed Operators are represented by ClusterServiceVersions within this



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

6.3 Upgrade Percona Server for MongoDB

You can decide how to run the database upgrades:

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

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

- · Never the Operator never makes automatic upgrades. You must upgrade the Custom Resource and images manually.
- · Disabled the Operator doesn't not carry on upgrades automatically. You must upgrade the Custom Resource and images manually.
- · Recommended the Operator automatically updates the database and components to the version flagged as Recommended
- · Latest the Operator automatically updates the database and components to the most recent available version
- · version specify the specific database version that you want to update to. The Operator updates the database to it automatically.

6.4 Minor upgrade

6.4.1 Upgrade to a specific version

Assumptions

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

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

Procedure

To upgrade Percona Server for MongoDB to a specific version, do the following:

- 1 Check the version of the Operator you have in your Kubernetes environment. If you need to update it, refer to the Operator upgrade guide.
- 2 Check the <u>Custom Resource</u> manifest configuration to be the following:
 - spec.updateStrategy option is set to SmartUpdate
 - spec.upgradeOptions.apply option is set to Disabled or Never.

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

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

```
$ curl https://check.percona.com/versions/v1/psmdb-operator/1.20.0 |jq -r '.versions[].matrix'
```

You can also find this information in the <u>Versions compatibility</u> matrix.

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

We recommend to update the PMM Server **before** the upgrade of PMM Client. If you haven't done it yet, exclude PMM Client from the list of images to update.

Since this is a working cluster, the way to update the Custom Resource is to apply-a-patch with the kubect1 patch psmdb command.

This example command updates the cluster with the name my-cluster-name to the 1.20.0 version:

With PMM Client

```
$ kubectl patch psmdb my-cluster-name --type=merge --patch '{
   "spec": {
      "crVersion":"1.20.0",
      "image": "percona/percona-server-mongodb:7.0.18-11",
      "backup": { "image": "percona/percona-backup-mongodb:2.9.1" },
      "pmm": { "image": "percona/pmm-client:2.44.1" }
}}'
```

Without PMM Client

```
$ kubectl patch psmdb my-cluster-name --type=merge --patch '{
   "spec": {
      "crVersion":"1.20.0",
      "image": "percona/percona-server-mongodb:7.0.18-11",
      "backup": { "image": "percona/percona-backup-mongodb:2.9.1" }
}}'
```

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

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

The update process is successfully finished when all Pods have been restarted. If you turned on <u>Percona Server for MongoDB Sharding</u>, the mongos and Config Server nodes must be restarted too to complete the upgrade.

6.4.2 Automatic minor upgrade to the latest / recommended version

Assumptions

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

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

Procedure

You can configure the Operator to automatically upgrade Percona Server for MongoDB to the latest available, the recommended or to a specific version of your choice. <u>Learn more about automatic upgrades</u>.

The steps are the following:

- 1 Check the version of the Operator you have in your Kubernetes environment. If you need to update it, refer to the Operator upgrade guide.
- 2 Make sure that spec.updateStrategy option in the Custom Resource manifest is set to SmartUpdate.
- 3 Set the upgradeOptions.apply option in the Custom Resource manifest from Disabled to one of the following values:
 - Recommended automatic upgrade will choose the most recent version of software flagged as Recommended (for clusters created from scratch, the Percona Server for MongoDB 8.0 version will be selected instead of the Percona Server for MongoDB 7.0 or 6.0 version regardless of the image path; for already existing clusters, the 8.0 vs. 7.0 vs. 6.0 branch choice will be preserved),
 - 8.0-recommended, 7.0-recommended, 6.0-recommended same as above, but preserves specific major MongoDB version for newly provisioned clusters (ex. 8.0 will not be automatically used instead of 7.0),
 - Latest automatic upgrade will choose the most recent version of the software available (for clusters created from scratch, the Percona Server for MongoDB 8.0 version will be selected instead of the Percona Server for MongoDB 7.0 or 6.0 version regardless of the image path; for already existing clusters, the 8.0 vs. 7.0 vs. 6.0 branch choice will be preserved),
 - 3.0-latest, 7.0-latest, 6.0-latest same as above, but preserves specific major MongoDB version for newly provisioned clusters (ex. 8.0 will not be automatically used instead of 7.0),
 - version number specify the desired version explicitly (version numbers are specified as 6.0.21-18, 7.0.18-11, etc.). Actual versions can be found in the list of certified images.
 - End of Life versions of MongoDB
 - MongoDB 5.0 support has reached its end-of-life in the Operator version 1.19.0. Therefore, the 5.8-recommended or 5.8-latest values are no longer supported. Users of existing clusters based on Percona Server for MongoDB 5.0 should explicitly switch to a newer major database version before upgrading the Operator to 1.19.0.
 - (a) MongoDB 4.4 support has reached its end-of-life in the Operator version 1.16.0. Therefore, the 4.4-recommended or 4.4-latest values are no longer supported. Users of existing clusters based on Percona Server for MongoDB 4.4 should explicitly switch to a newer major database version before upgrading the Operator to 1.16.0.
- 4 Make sure to set the valid Version Server URL for the versionServiceEndpoint key. The Operator checks the new software versions in the Version Server. If the Operator can't reach the Version Server, the upgrades won't happen.

Percona's Version Service (default)

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

Version Service inside your cluster

Alternatively, you can run Version Service inside your cluster. This can be done with the kubect1 command as follows:

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

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

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

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

Note

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

6 Apply your changes to the Custom Resource:

\$ kubectl apply -f deploy/cr.yaml

6.4.3 Manual upgrades of Percona Server for MongoDB

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

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

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

- Rolling Update, initiated manually and controlled by Kubernetes [2]. Note that the order of Pods restart may not be optimal from the Percona Server for MongoDB point of view.
- On Delete, done by Kubernetes on per-Pod basis ☐ when Pods are manually deleted.

Rolling Update strategy and semi-automatic updates

To run a semi-automatic update of Percona Server for MongoDB, do the following:

- 1 Check the version of the Operator you have in your Kubernetes environment. If you need to update it, refer to the Operator upgrade guide.
- 2 Edit the deploy/cr.yaml file and set the updateStrategy key to RollingUpdate.
- 3 After you applied the patch, the deployment rollout will be triggered automatically. You can track the rollout process in real time using the kubectl rollout status command with the name of your cluster:

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

Manual upgrade (the On Delete strategy)

To upgrade Percona Server for MongoDB manually, do following:

- 1) Check the version of the Operator you have in your Kubernetes environment. If you need to update it, refer to the Operator upgrade guide.
- 2 Edit the deploy/cr.yaml file and set the updateStrategy key to OnDelete.
- 3 The Pod with the newer Percona Server for MongoDB image will start after you delete it. Delete targeted Pods manually one by one to make them restart in the desired order:
 - Delete the Pod using its name with the command like the following one:

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

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
```

The update process is successfully finished when all Pods have been restarted. If you turned on <u>Percona Server for MongoDB Sharding</u>, the mongos and Config Server nodes must be restarted too to complete the upgrade.

6.5 Major version automated upgrades

Major version upgrade is moving from the current major version to the next one. For example, from 6.0.x to 7.0.x.

It is a more complicated task than a minor version upgrade because it might potentially affect how data is stored and how applications interact with the database (in case of some API changes). Therefore, we recommend to test the major version upgrade on a staging environment first.

A major upgrade is supported by the Operator within one major version at a time: for example, from 7.0 to 8.0. To upgrade Percona Server for MongoDB from 6.0 to 8.0, you should first upgrade it to 7.0, and then make a separate upgrade from 7.0 to 8.0. The same is true for major version downgrades.



Important

Before the upgrade, make a backup of your data.

Assumptions

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

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

Procedure

- 1. Check the version of the Operator you have in your Kubernetes environment. If you need to update it, refer to the Operator upgrade guide.
- 2. Set the upgradeOptions.apply key in the deploy/cr.yaml Custom Resource manifest to <version>-recommended:

upgradeOptions: apply: 8.0-recommended

1. Apply the deploy/cr.yaml Custom Resource manifest to start the major version upgrade.

Feature Compatibility Version

By default, the Operator doesn't set FeatureCompatibilityVersion (FCV) of to match the new version. This ensures that backwards-incompatible features are not automatically enabled with the major version upgrade (which the 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 that with this setting, the Operator doesn't yet support major version rollback if the setFeatureCompatibilityVersion is set. Therefore it is recommended to stay without enabling this flag for some time after the major upgrade to ensure that everything works as expected and you won't have to downgrade.

You can set the setFCV flag to true simultaneously with the apply flag but you must be absolutely sure that the whole procedure is tested on staging environment and works as expected.

Downgrades

When making downgrades (e.g. changing version from 7.0 to 6.0), 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 [4].

6.6 Upgrade Database and Operator on OpenShift

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

Upgrading the Operator and CRD

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

```
$ kubectl get deploy percona-server-mongodb-operator -o yaml

Expected output

...

"containerImage": "registry.connect.redhat.com/percona/percona-server-mongodb-operator@sha256:201092cf97c9ceaaaf3b60dd1b24c7c5228d35aab2674345893f4cd4d9bb0e2e",
...
```

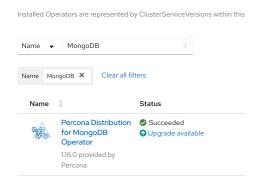
b. Apply a patch C to update the initImage option of your cluster Custom Resource with this value taken from containerImage. Supposing that your cluster name is my-cluster-name, the command should look as follows:

```
$ kubectl patch psmdb my-cluster-name --type=merge --patch '{
    "spec": {
        "initImage":"registry.connect.redhat.com/percona/percona-server-mongodb-
operator@sha256:201092cf97c9ceaaaf3b60dd1b24c7c5228d35aab2674345893f4cd4d9bb0e2e"
    }}'
```

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

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

Installed Operators



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

Upgrading Percona Server for MongoDB

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

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

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

\$ kubectl get deploy percona-server-mongodb-operator -o yaml

```
Expected output

...
    "image": "registry.connect.redhat.com/percona/percona-server-mongodb-operator-
    containers@sha256:5d29132a60b89e660ab738d463bcc9707a17be73dc955aa8da9e50bed4d9ad3e",
    ...
    "initImage": "registry.connect.redhat.com/percona/percona-server-mongodb-
    operator@sha256:8adc57e9445cfcea1ae02798a8f9d6a4958ac89f0620b9c6fa6cf969545dd23f",
    ...
    "pmm": {
        "enabled": true,
        "image": "registry.connect.redhat.com/percona/percona-server-mongodb-operator-
        containers@sha256:165f97cdae2b6def546b0df7f50d88d83c150578bdb9c992953ed866615016f1",
        ...
        "backup": {
        "enabled": true,
        "image": "registry.connect.redhat.com/percona/percona-server-mongodb-operator-
        containers@sha256:a73889d61e996bc4fbc6b256a1284b60232565e128a64e4f94b2c424966772eb",
        ...
        ...
```

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

```
$ kubect1 patch psmdb my-cluster-name --type=merge --patch '{
    "spec": {
        "crVersion":"1.20.0",
        "image": "registry.connect.redhat.com/percona/percona-server-mongodb-operator-
containers@sha256:5d29132a60b89e660ab738d463bcc0707a17be73dc955aa8da9e50bed4d9ad3e",
        "initImage": "registry.connect.redhat.com/percona/percona-server-mongodb-
operator@sha256:8adc57e9445cfcea1ae02798a8f9d6a4958ac89f0620b9c6fa6cf969545dd23f",
        "pmm": {"image": "registry.connect.redhat.com/percona/percona-server-mongodb-operator-
containers@sha256:165f97cdae2b6def546b0df7f50d88d83c150578bdb9c992953ed866615016f1"},
        "backup": {"image": "registry.connect.redhat.com/percona/percona-server-mongodb-operator-
containers@sha256:a73889d61e996bc4fbc6b256a1284b60232565e128a64e4f94b2c424966772eb"}
    }}'
```

```
Marning

The above command upgrades various components of the cluster including PMM Client. If you didn't follow the official recommendation [3] to upgrade PMM Server before upgrading PMM Client, you can avoid PMM Client upgrade by removing it from the list of images as follows:

$ kubectl patch psmdb my-cluster-name --type=merge --patch '{
    "spec": {
        "crVersion":"1.20.0",
        "image": "registry.connect.redhat.com/percona/percona-server-mongodb-operator-containers@sha256:5d29132a60b89e660ab738d463bcc0707a17be73dc955aa8da9e50bed4d9ad3e",
        "initImage": "registry.connect.redhat.com/percona/percona-server-mongodb-operator@sha256:8adc57e9445cfcea1ae02798a8f9d6a4958ac89f96c20b9c6fa6cf969545dd23f",
        "backup": {"image": "registry.connect.redhat.com/percona/percona-server-mongodb-operator-containers@sha256:a73889d61e996bc4fbc6b256a1284b60232565e128a64e4f94b2c424966772eb"}

}}'
```

4. The deployment rollout will be automatically triggered by the applied patch.

7 Configuration

7.1 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.

Unprivileged users

The Operator does not create unprivileged (general purpose) user accounts by default.

Here's how you can create unprivileged users:

- manually in Percona Server for MongoDB
- automatically via Custom Resource (Operator versions 1.17.0 and newer).

Regardless how you create users, their usernames must be unique.

Create users via Custom Resource

Starting from the Operator version 1.17.0, you can create users in Percona Server for MongoDB via the users subsection in the Custom Resource. This is called declarative user management.



Warning

Declarative user management has the technical preview status and is not yet recommended for production environments.

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

For every new user in deploy/cr.yaml configuration file, specify the following:

- a username and the database where a user will be created. The username must be unique for every user.
- · roles on databases in MongoDB which you want to grant to this user.

If you don't want the Operator to generate a user password automatically, you can create a Secret resource that contains the user password. Then specify a reference to this Secret resource in the passwordSecretRef key. You can find a detailed description of the corresponding options in the Custom Resource reference

Here are example configurations.

Secret configuration:

apiVersion: v1 kind: Secret metadata:

name: my-user-password

type: Opaque stringData:

password: mypassword

Custom Resource configuration:

```
users:
  - name: my-user
   db: admin
    passwordSecretRef:
      name: my-user-password
      key: password
    roles:
      - name: clusterAdmin
       db: admin
      - name: userAdminAnyDatabase
        db: admin
```

If you haven't specified the Secret name in the Custom Resource, the Operator creates a Secret named <cluster-name>-custom-user-secret, generates a password for the user and sets it by the key named after the user name.



The Operator doesn't generate passwords for users created in the \$external database. You can't set the passwordSecretRef for these users, too.

Such users are used for authentication via an external authentication source, such as an LDAP server. The user credentials are stored in an external authentication source and their usernames are mapped to those in the \$external database during authentication.

The Operator tracks password changes in the Secret object, and updates the user password in the database. This applies to the manually created users as well: if a user was created manually in the database before creating user via Custom Resource, the existing user is updated. But manual password updates in the database are not tracked: the Operator doesn't overwrite changed passwords with the old ones from the users Secret.

Custom MongoDB roles

Custom MongoDB roles
☐ allow providing fine-grained access control over your MongoDB deployment.

Custom MongoDB roles can be defined in a declarative way via the roles subsection in the Custom Resource.

Warning

Custom roles were introduced in the Operator version 1.18.0. It has technical preview status and is not yet recommended for production environments.

This subsection contains array of roles each with the defined custom name (roles.name), database in which you want to store the user-defined role (roles.db). The roles.privileges.actions allows to set List of custom role actions that users granted this role can perform. For a list of accepted values, see Privilege Actions [in the manual of the corresponding MongoDB version. Actions can be granted for the whole cluster (if roles.privileges.resource.cluster set to true), or be related to a specific database or collection. Adding existing role and database names to the roles .roles subsection allows you to inherit privileges from existing roles. Finally, you can apply authentication restrictions for your custom role based on the IP address ranges for the client and server. The following example shows how roles subsection may look like:

```
roles:
    - role: my-role
      db: admin
      privileges:
         - resource:
            db:
            collection: ''
          actions:
            - find
      authenticationRestrictions:
        - clientSource:
            - 127.0.0.1
          serverAddress:
            - 127.0.0.1
      roles:
         - role: read
          db: admin
        - role: readWrite
          db: admin
```

Find more information about available options and their accepted values in the roles subsection of the Custom Resource reference.

Create users manually

You can create unprivileged users manually. Please run commands below, substituting the <namespace name> placeholder with the real namespace of your database cluster:

if sharding is on

Now check the newly created user:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:7.0.18-11 --restart=Never -- bash -il
mongodb@percona-client:/$ mongosh "mongodb+srv://myApp:myAppPassword@my-cluster-name-rs0.<namespace
name>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
rs0:PRIMARY> use myApp
rs0:PRIMARY> db.test.insert({ x: 1 })
rs0:PRIMARY> db.test.findOne()
```

if sharding is off

Now check the newly created user:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:7.0.18-11 --restart=Never -- bash -il
mongodb@percona-client:/$ mongosh "mongodb+srv://myApp:myAppPassword@my-cluster-name-rs0.<namespace
name>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
rs0:PRIMARY> use myApp
rs0:PRIMARY> db.test.insert({ x: 1 })
rs0:PRIMARY> db.test.findOne()
```

System Users

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

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

Default Secret name: my-cluster-name-secrets

Secret name field: spec.secrets.users



Warning

These users should not be used to run an application.

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

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

- Backup/Restore MongoDB Role: <u>backup</u> [2], <u>restore</u> [2], <u>clusterMonitor</u> [2], <u>readWrite</u> [2], <u>pbmAnyAction</u> [2]
- Cluster Admin MongoDB Roles: clusterAdmin 🖸
- Cluster Monitor MongoDB Role: cluster Monitor MongoDB Role: clustermon MongoDB Role: cl
- Database Admin MongoDB Roles: readWriteAnyDatabase [2], readAnyDatabase [2], dbAdminAnyDatabase [2], backup [2], restore [2], clusterMonitor [2]
- User Admin MongoDB Role: userAdminAnyDatabase 🖸

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

```
Note
In some situations it can be needed to reproduce system users in a bare-bone MongoDB. For example, that's a required step in the migration scenarios [2] to move existing on-prem
MongoDB database to Kubernetes-based MongoDB cluster managed by the Operator. You can use the following example script which produces a text file with mongo shell commands
to create needed system users with appropriate roles:
   gen_users.sh
    clusterAdminPass="clusterAdmin"
    userAdminPass="userAdmin
    clusterMonitorPass="clusterMonitor"
    backupPass="backup"
    # mongo shell
    cat <<EOF > user-mongo-shell.txt
    use admin
    db.createRole(
    "roles": [],
    role: "pbmAnyAction",
    "privileges" : [
                              "resource" : {
                                      "anyResource" : true
                              }.
                              "actions" : [
"anyAction"
                     }
            ],
    })
    db.createUser( { user: "clusterAdmin", pwd: "$clusterAdminPass", roles: [ "clusterAdmin" ] } )
db.createUser( { user: "backup", pwd: "$backupPass", roles: [ "readWrite", "backup", "clusterMonitor", "restore", "pbmAnyAction" ] } )
```

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
  {\tt MONGODB\_DATABASE\_ADMIN\_USER: \ databaseAdmin}
  MONGODB_DATABASE_ADMIN_PASSWORD: databaseAdmin123456
  MONGODB_CLUSTER_ADMIN_USER: clusterAdmin
  MONGODB_CLUSTER_ADMIN_PASSWORD: clusterAdmin123456
  MONGODB CLUSTER MONITOR USER: clusterMonitor
  MONGODB_CLUSTER_MONITOR_PASSWORD: clusterMonitor123456
  MONGODB_USER_ADMIN_USER: userAdmin
  MONGODB_USER_ADMIN_PASSWORD: userAdmin123456
  PMM_SERVER_USER: admin
  PMM_SERVER_PASSWORD: admin
  PMM_SERVER_API_KEY: apikey
```

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

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

in Linux

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

in macOS

 $\label{lem:condition} $$ \text{kubectl patch secret/my-cluster-name-secrets -p '{"data":{"MONGODB_DATABASE_ADMIN_PASSWORD": "'$(echo -n new_password | label{lem:condition}). } \$ base64)'"}}'



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.



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

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



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

MongoDB Internal Authentication Key (optional)

Default Secret name: my-cluster-name-mongodb-keyfile

Secret name field: spec.secrets.key

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

```
apiVersion: v1
kind: Secret
metadata:
   name: my-cluster-name-mongodb-keyfile
type: Opaque
data:
   mongodb-key: <replace-this-value-with-base-64-encoded-text>
```

7.2 Changing MongoDB options

When you deploy a new Percona Server for MongoDB cluster, the Operator spins it up with the set of defaults that ensure its correct operation. However, your application may need additional configuration of MongoDB. You can define this configuration using the mongod.conf configuration file options. You can configure mongod Pods, mongos Pods and config server replica set Pods separately, based on your requirements.

Then you pass these 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

Note that you can't change options that may break the behavior of the Operator. For example, TLS/SSL options. If you try changing such options, your changes will be ignored.

Edit the deploy/cr.yaml file

You can add MongoDB configuration options to the following keys of the deploy/cr.yaml:

- replsets.configuration
- sharding.mongos.configuration
- sharding.configsvrReplSet.configuration

Example

This example shows how to enable rate limit for database profiler and define the default verbosity level for system log:

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

Find the complete list of options in the official manual . Also refer to these pages in Percona Server for MongoDB documentation:

- Profiling rate limit [4]
- Percona memory engine [4]
- <u>Data-at-rest encryption</u>
- Log redaction ☐
- Audit logging

 ✓.

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,



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



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).

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,



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 2.

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

operationProfiling: mode: slowOp systemLog: verbosity: 1

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

in Linux

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

in macOS

\$ cat mongod.conf | base64



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

AogICAgICBzeXN0ZW1Mb2c6CiAgICAgICAgdmVyYm9zaXR50iAxCg==" | 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

AogICAgICBzeXN0ZW1Mb2c6CiAgICAgICAgdmVyYm9zaXR50iAxCg=="

When ready, apply it with the following command:

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



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 <u>Install Percona Server for MongoDB on</u> Kubernetes page).

7.3 Change Percona Backup for MongoDB configuration

The Operator configures Percona Backup for MongoDB (PBM) with the set of default parameters that ensure its correct operation. However, you may want to fine-tune PBM configuration to meet your specific needs. For example, <u>adjust the node priority for backups</u> of to use a specific node. Or configure the <u>parallel</u> download from the storage for a physical restore of the parallel download from the storage for a physical restore.

To change PBM configuration, edit the deploy/cr.yaml Custom Resource manifest and specify the options in the backup.configuration subsection.

Examples

This example shows how to adjust the node priority for backups:

```
spec:
backup:
...
configuration:
backupOptions:
priority:
    "cluster1-rs0-0.cluster1-rs0.psmdb.svc.cluster.local:27017": 2.5
    "cluster1-rs0-1.cluster1-rs0.psmdb.svc.cluster.local:27017": 2.5
```

This example shows how to configure the parallel data download for physical restores:

```
spec:
backup:
...
configuration:
  restoreOptions:
    numDownloadWorkers: 4
    maxDownloadBufferMb: 0
    downloadChunkMb: 32
```

Apply the manifest to pass your configuration to the Operator:

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

Refer to the Custom Resource options for the full list of available options.

7.4 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 of file contain the keys which can be used to do assign pods to nodes.

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

Affinity and anti-affinity

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

Percona Operator for MongoDB provides two approaches for doing this:

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

Simple approach - use antiAffinityTopologyKey of the Percona Operator for MongoDB

Percona Operator for MongoDB provides an antiAffinityTopologyKey option, which may have one of the following values:

- kubernetes.io/hostname Pods will avoid residing within the same host,
- topology.kubernetes.io/zone Pods will avoid residing within the same zone,
- topology.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 <u>deploy/cr.yaml</u> of 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:
       {\tt preferredDuringSchedulingIgnoredDuringExecution:}
       - weight: 100
         podAffinityTerm:
           labelSelector:
             matchExpressions:
             - key: security
               operator: In
               values:
               - S2
           topologyKey: kubernetes.io/hostname
     nodeAffinity:
       required {\tt DuringSchedulingIgnoredDuringExecution:}
         nodeSelectorTerms:
         - matchExpressions:
           - key: kubernetes.io/e2e-az-name
             operator: In
             values:
             - e2e-az1
             - e2e-az2
       {\tt preferredDuringSchedulingIgnoredDuringExecution:}
       - weight: 1
         preference:
           {\tt matchExpressions:}
           - key: another-node-label-key
             operator: In
             values:
             - another-node-label-value
```

See explanation of the advanced affinity options in Kubernetes documentation [4].

Topology Spread Constraints

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

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

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

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

Tolerations

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

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

• NoSchedule - less strict

- PreferNoSchedule
- NoExecute

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

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

The <u>Kubernetes Taints and Toleratins</u> C contains more examples on this topic.

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 [2] to find out how to define and use priority classes in your cluster.

Pod Disruption Budgets

Creating the Pod Disruption Budget C 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 C file can be used to set these limits. The recommended variant is the following:

podDisruptionBudget:
 maxUnavailable: 1

7.5 Labels and annotations

Labels [2] and annotations [2] are used to attach additional metadata information to Kubernetes resources.

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

Setting labels and annotations in the Custom Resource

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

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

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

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

Using labels and annotations with objects created by the Operator

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

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

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

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

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

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

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

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

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

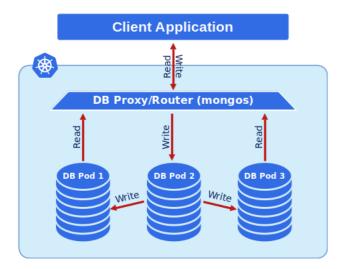
7.6 Exposing the cluster

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 <u>Custom Resource manifest options</u> to expose clusters deployed with the Operator.

Using a single entry point in a sharded cluster

If Percona Server for MongoDB <u>sharding mode</u> is turned **on** (the default behavior), then the database cluster runs special mongos Pods - query routers, which act as entry points for client applications:



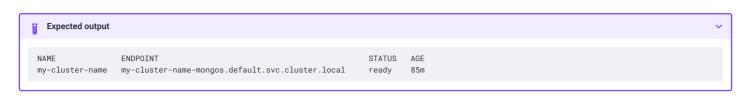
By default, a ClusterIP type Service is created (this is controlled by <u>sharding.mongos.expose.type</u>). The Service works in a round-robin fashion between all the mongos Pods.

The URI looks like this (taking into account the need for a proper password obtained from the Secret, and a proper namespace name instead of the <namespace name> placeholder):

 $\$ \ mongosh \ "mongodb://userAdmin:userAdminPassword@my-cluster-name-mongos.<namespace \ name>.svc.cluster.local/admin:ssl=false". The property of the prope$

You can get the actual Service endpoints by running the following command:

\$ kubectl get psmdb

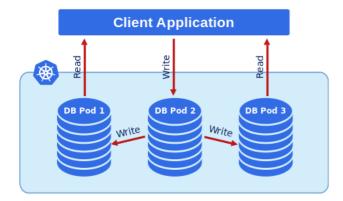




A ClusterIP Service endpoint is only reachable inside Kubernetes. If you need to connect from the outside, you need to expose the mongos Pods by using the NodePort or Load Balancer Service types. See the Connecting from outside Kubernetes section below for details.

Accessing replica set Pods

If Percona Server for MongoDB sharding mode mode is turned off, the application needs to connect to all the 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 by using Kubernetes internal DNS names in the URI.

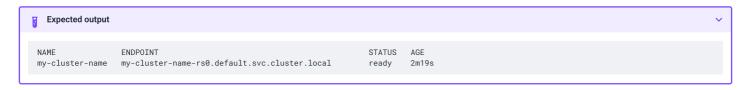
By default, a ClusterIP type Service is created (this is controlled by <u>replsets.expose.type</u>). The Service works in a round-robin fashion between all the mongod Pods of the replica set.

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

 $\label{lem:cluster-name-rs0.} $$ mongooh "mongodb://databaseAdmin:databaseAdminPassword@my-cluster-name-rs0.<namespace name>.svc.cluster.local/admin?replicaSet=rs0&ssl=false"$

You can get the actual Service endpoints by running the following command:

\$ kubectl get psmdb





A ClusterIP Service endpoint is only reachable inside Kubernetes. If you need to connect from the outside, you need to expose the mongod Pods by using the NodePort or Load Balancer Service types. See the Connecting from outside Kubernetes section below for details.

Connecting from outside Kubernetes

If connecting to a cluster from outside Kubernetes, you cannot reach the Pods using the Kubernetes internal DNS names. To make the Pods accessible, Percona Operator for MongoDB can create <u>Kubernetes Services</u>.

- set expose.enabled option to true to allow exposing the Pods via Services,
- set expose.type option specifying the type of Service to be used:
 - ClusterIP expose the Pod with an internal static IP address. This variant makes the Service reachable only from within the Kubernetes cluster.
 - NodePort expose the Pod on each Kubernetes Node's IP address at a static port. A 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, however the address will be bound to a specific Kubernetes Node. The expose externalTrafficPolicy Custom Resource option available in replsets, sharding.configsvrReplSet, and sharding.mongos subsections of the deploy/cr.yaml manifest, controlls if the external traffic will be node-local (Local, external requests will be dropped if there is no available Pod on the Node) or cluster-wide (Cluster, requests can be routed to another Node at the cost of extra latency and not preserving the client IP address).
 - LoadBalancer expose the Pod externally using a cloud provider's load balancer. Both ClusterIP and NodePort Services are automatically created C in this variant.

If the NodePort type is used, the URI looks like this:

 $\verb|mongodb://databaseAdmin:databaseAdminPassword@<node1>:<port1>, <node2>:<port2>, <node3>:<port3>/admin?replicaSet=rs0&ss1=false$

All Node addresses should be directly reachable by the application.

Service per Pod

To make all database Pods accessible, Percona Operator for MongoDB can assign a <u>Kubernetes Service</u> of to each Pod. Particularly, the Service per Pod option allows the application to take care of Cursor tracking instead of relying on a single Service. This solves the problem of CursorNotFound errors when the Service transparently cycles between the mongos instances while client is still iterating the cursor on some large collection.

This feature can be enabled for both sharded and non-sharded clusters by setting the <a href="mailto:sharded-enable-sharded-en

If this feature is enabled with the expose.type: NodePort, the created Services look like this:

```
$ kubectl get svc
                                        CLUSTER-IP
                                                                     PORT(S)
NAME
                          TYPE
                                                       EXTERNAL-IP
                                                                                                   AGE
my-cluster-name-mongos-0
                          NodePort
                                         10.38.158.103
                                                        <none>
                                                                       27017:31689/TCP
                                                                                                   12s
                                                                       27017:31389/TCP
                                         10.38.155.250 <none>
                                                                                                   12s
my-cluster-name-mongos-1
                         NodePort
```

Controlling hostnames in replset configuration

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



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

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

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

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

Restart can be done manually with the kubectl rollout restart sts

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



Warning

You should be careful with the clusterServiceDNSMode=External variant. Using IP addresses instead of DNS hostnames is discouraged in MongoDB. IP addresses make reconfiguration and recovery more complicated, and are **generally problematic in scenarios where IP addresses change**. In particular, if you delete and recreate the cluster with clusterServiceDNSMode=External without deleting its volumes (having percona.com/delete-psmdb-pvc finalizer unset), your cluster will crash and there will be no straightforward way to recover it.

Exposing replica set with split-horizon DNS

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

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

```
replsets:
- name: rs0
expose:
enabled: true
type: LoadBalancer
splitHorizons:
cluster1-rs0-0:
external: rs0-0.mycluster.xyz
external-2: rs0-0.mycluster2.xyz
cluster1-rs0-1:
external: rs0-1.mycluster.xyz
external-2: rs0-1.mycluster2.xyz
cluster1-rs0-2:
external: rs0-2.mycluster2.xyz
external: rs0-2.mycluster2.xyz
```

URIs for external usage are specified as key-value pairs, where the key is an arbitrary name and the value is the actual URI. The URI may include a port number. If nothing is set, the default MongoDB port will be used.

Split horizon has following limitations:

- connecting with horizon domains is only supported if client connects using TLS certificates, and these TLS certificates need to be generated manually.
- duplicating domain names in horizons is not allowed by MongoDB
- using IP addresses in horizons is not allowed by MongoDB
- horizons should be set for all Pods of a replica set or not set at all

7.7 Local Storage support for the Percona Operator for MongoDB

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

emptyDir

A Pod <u>emptyDir volume</u> is 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 [3] file can be used to turn the emptyDir volume on by setting the directory name.

The emptyDir is useful when you use $\underline{\text{Percona Memory Engine}}$ \square .

hostPath

A hostPath volume C 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 <u>deploy/cr.yaml</u> the lie may include path and type keys to set the node's filesystem object path and to specify whether it is a file, a directory, or something else (e.g. a socket):

volumeSpec:
 hostPath:
 path: /data
 type: Directory

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

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

The hostPath volume is useful when you perform manual actions during the first run and require improved disk performance. Consider using the tolerations settings to avoid a cluster migration to different hardware in case of a reboot or a hardware failure.

7.8 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.

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 <u>arbiter node</u>. 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 <a href="https://december.new.org/de

- 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
....
```



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

Preventing Arbiter instances to share Kubernetes Nodes with Replica Set

By default Arbiter instances are allowed to run on the same host as regular Replica Set instances. This may be reasonable in terms of the number of Kubernetes Nodes required for the cluster. But as a result it increases possibility to have 50/50 votes division in case of network partitioning. You can use anti-affinity.constraints to avoid such Pod alocation as follows:

```
arbiter:
  enabled: true
  size: 1
  affinity:
    antiAffinityTopologyKey: "kubernetes.io/hostname"
    advanced:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app.kubernetes.io/component: mongod
              app.kubernetes.io/instance: cluster1
              app.kubernetes.io/managed-by: percona-server-mongodb-operator
              app.kubernetes.io/name: percona-server-mongodb
              app.kubernetes.io/part-of: percona-server-mongodb
              app.kubernetes.io/replset: rs0
          topologyKey: kubernetes.io/hostname
```

Adding non-voting nodes

Non-voting member \square 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 \square , which obviously should not participate in the voting process.



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/ <a hr

- 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 <u>replsets.nonvoting section</u> of the <u>Custom Resource options reference</u>.

7.9 Percona Server for MongoDB Sharding

About sharding

Sharding [1] 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.



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.

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. If backups are disabled (the backup, enabled Custom Resource option set to false), the Operator will turn sharded MongoDB instances into unsharded one by one, so the database cluster will operate without downtime. If backups are enabled (the backup enabled Custom Resource option is true), the Operator will pause the cluster (to avoid Percona Backup for MongoDB misconfiguration), update the instances, and then unpause it back.

Configuring instances of a sharded cluster

When sharding is turned on, the Operator runs replica sets with config servers and mongos instances. Their number is controlled by configsvrRep1Set.size and mongos.size keys, respectively.

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

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



Note

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

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



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

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.

To connect to Percona Server for MongoDB you need to construct the MongoDB connection URI string. It includes the credentials of the admin user, which are stored in the Secrets object.

1. List the Secrets objects

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

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

2. View the Secret contents to retrieve the admin user credentials.

```
$ kubectl get secret my-cluster-name-secrets -o yaml
```

The command returns the YAML file with generated Secrets, including the MONGODB_DATABASE_ADMIN_USER and MONGODB_DATABASE_ADMIN_PASSWORD strings, which should look as follows:

```
Sample output

...

data:
...

MONGODB_DATABASE_ADMIN_PASSWORD: aDAzQ@pCY3NSWEZ2ZUIzS1I=
MONGODB_DATABASE_ADMIN_USER: ZGF@YWJhc2VBZG1pbg==
```

The actual login name and password on the output are base64-encoded. To bring it back to a human-readable form, run:

```
$ echo 'MONGODB_DATABASE_ADMIN_USER' | base64 --decode
$ echo 'MONGODB_DATABASE_ADMIN_PASSWORD' | base64 --decode
```

3. Run a container with a MongoDB client and connect its console output to your terminal. The following command does this, naming the new Pod perconaclient:

```
$ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:7.0.18-11 --restart=Never -- bash -il
```

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

4. Now run mongosh tool inside the percona-client command shell using the admin user credentialds you 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

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

if sharding is off

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

7.10 Transport encryption (TLS/SSL)

7.10.1 Transport Layer Security (TLS)

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. The internal certificate is also used as an authorization method.
- External communication between the client application and the cluster

TLS usage is controlled by the tls.mode Custom Resource option, which can be set to allowTLS, preferTLS (default choice), requireTLS, or disabled:

```
spec:
...
tls:
   mode: preferTLS
```

- · allowTLS means that both TLS and non-TLS incoming connections are accepted, but server doesn't use TLS internally,
- preferTLS turns on TLS for internal communication, and allows both TLS and non-TLS external traffic,
- requireTLS enforces the use of TLS encrypted connections only,
- disabled completely <u>turns TLS off</u>.

TLS Certificates

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.



The tls.allowInvalidCertificates Custom Resource option is set to true by default to allow certificates automatically generated by the Operator. It can be set to false with other variants, such as certificates generated by cert-manager.

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!

TLS configuration

The following sections provide guidelines how to:

- Configure TLS security with the Operator using cert-manager
- Generate certificates manually
- <u>Update certificates</u>
- Disable TLS temporarily

If you are going to use TLS for external traffic, you need to additionally configure your client application. See <u>this blog post</u> of for detailed instruction with examples. Also, you can check the <u>official MongoDB documentation</u>.

For clients outside of your Kubernetes-based environment, you must also expose your cluster.

7.10.2 Install and use the cert-manager

About the cert-manager

The <u>cert-manager</u> of 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.

Install 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/v1.17.2/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:

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

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

7.10.3 Generate certificates manually



Warning

Using manually generated certificates didn't work in the Operator version 1.16.0. The problem is fixed starting from the version 1.16.1.

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.

You can explore pre-generated / development mode sample certificates available as base64-encoded data in the deploy/ssl-secrets.yaml file. Also, check MongoDB certificate requirements in the <u>upstream documentation</u> [2].



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

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

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

if sharding is on

```
$ CLUSTER_NAME=my-cluster-name
$ NAMESPACE=default
$ cat <<EOF | cfssl gencert -initca - | cfssljson -bare ca</pre>
    "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"]
    }
FOF
$ 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",
      "${CLUSTER_NAME}-mongos"
      "${CLUSTER_NAME}-mongos.${NAMESPACE}",
      \verb| "\$\{CLUSTER_NAME\}-mongos.\$\{NAMESPACE\}.svc.cluster.local",\\
      "*.${CLUSTER_NAME}-mongos"
      "*.${CLUSTER_NAME}-mongos.${NAMESPACE}",
      "*.\$\{\texttt{CLUSTER\_NAME}\}-\texttt{mongos.}\$\{\texttt{NAMESPACE}\}.\texttt{svc.cluster.local}",
      "${CLUSTER_NAME}-cfg",
      "${CLUSTER_NAME}-cfg.${NAMESPACE}",
      "${CLUSTER_NAME}-cfg.${NAMESPACE}.svc.cluster.local",
      "*.${CLUSTER_NAME}-cfg"
      "*.${CLUSTER_NAME}-cfg.${NAMESPACE}",
      "*.${CLUSTER_NAME}-cfg.${NAMESPACE}.svc.cluster.local"
    1.
    "names": [
        "0": "PSMDB"
      }
    1.
    "CN": "${CLUSTER_NAME/-rs0}",
    "key": {
      "algo": "rsa",
      "size": 2048
EOF
$ cfssl bundle -ca-bundle=ca.pem -cert=server.pem | cfssljson -bare server
\$ \ \text{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.${NAMESPACE}",
                    "*.\$\{CLUSTER\_NAME\}-rs0.\$\{NAMESPACE\}.svc.cluster.local",
                    "${CLUSTER_NAME}-mongos",
                    "${CLUSTER_NAME}-mongos.${NAMESPACE}",
                    "${CLUSTER_NAME}-mongos.${NAMESPACE}.svc.cluster.local",
                    "*.${CLUSTER_NAME}-mongos"
                    "*.${CLUSTER_NAME}-mongos.${NAMESPACE}",
                    "*.${CLUSTER_NAME}-mongos.${NAMESPACE}.svc.cluster.local",
                    \verb|"${CLUSTER_NAME}-cfg||
                    "${CLUSTER_NAME}-cfg.${NAMESPACE}",
                    "${CLUSTER_NAME}-cfg.${NAMESPACE}.svc.cluster.local",
                    "*.${CLUSTER_NAME}-cfg",
                   "*.${CLUSTER_NAME}-cfg.${NAMESPACE}",
                    "*.\$\{\texttt{CLUSTER\_NAME}\}-\texttt{cfg}.\$\{\texttt{NAMESPACE}\}.\texttt{svc.cluster.local}"
              ],
              "names": [
                         "0": "PSMDB"
              "CN": "${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 --f
file=ca.crt=ca.pem --type=kubernetes.io/tls
```

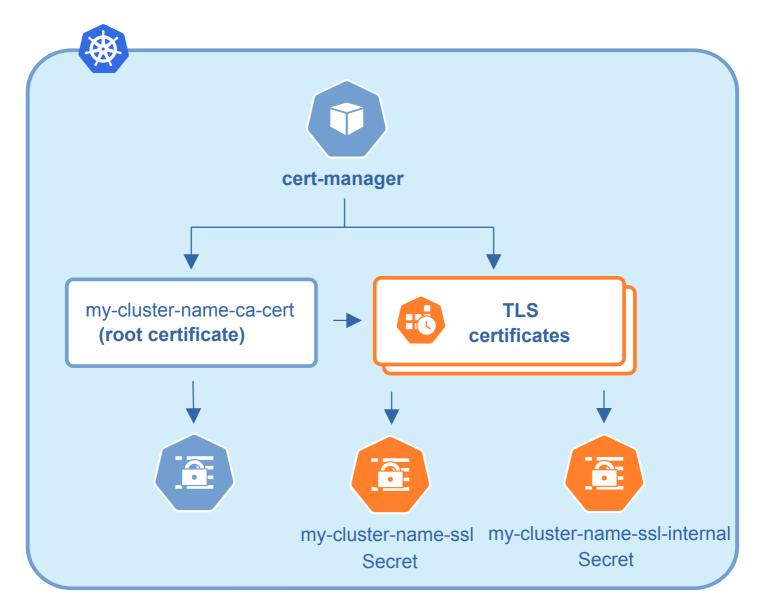
Note

Commands in the above example use rs0 replica set name (the default one). If you set different name in replsets.name Custom Resource option, change these commands

7.10.4 Update certificates

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

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



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

This command is available if you have cert-manager installed; if not, you can still check the necessary secrets names with kubect1 get secrets command.

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

```
$ kubectl get issuer
```

The response should be as follows:

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

Again, this command is provided by cert-manager; if you don't have it installed, you can still use kubectl get secrets.



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

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

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

The resulting output will be self-explanatory:

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

Update certificates without downtime

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



For already expired certificates, follow the alternative way.

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

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

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

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

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

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

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

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

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

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

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

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

Update certificates with downtime

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

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

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

3. Delete Secrets to force the SSL reconciliation:

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

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

Modify certificates generation

There may be reasons to tweak the certificates generation, making it better fit some needs. Of course, maximum flexibility can be obtained with manual certificates generation, but sometimes slight tweaking the already automated job may be enough.

The following example shows how to increase CA duration with cert-manager for a cluster named cluster1:

1. Delete the psmdb Custom Resource in the proper namespace (this will cause deletion of all Pods of the cluster, but later you will recreate the cluster using the same deploy/cr.yaml flie from which it was originally created).



you may need to make sure that finalizers.percona.com/delete-psmdb-pvc is not set if you want to preserver Persistent Volumes with the data.

Deletion command should look as follows:

```
$ kubectl -n <namespace_name> delete psmdb cluster1
```

2. Deletion takes time. Check that all Pods disappear with kubect1 -n <namespace_name> get pods command, and delete certificate related resources:

\$ kubectl -n <namespace_name> delete issuer.cert-manager.io/cluster1-psmdb-ca-issuer issuer.cert-manager.io/cluster1-psmdbissuer certificate.cert-manager.io/cluster1-ssl-internal certificate.cert-manager.io/cluster1-ssl certificate.certmanager.io/cluster1-ca-cert secret/cluster1-ssl secret/cluster1-ssl-internal

3. Create your own custom CA:

```
my_new_ca.yml
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
 name: cluster1-psmdb-ca-issuer
spec:
 selfSigned: {}
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
 name: cluster1-ca-cert
spec:
 commonName: cluster1-ca
 duration: 10000h0m0s
 isCA: true
  issuerRef:
   kind: Issuer
   name: cluster1-psmdb-ca-issuer
  renewBefore: 730h0m0s
  secretName: cluster1-ca-cert
```

Apply it as usual, with the kubectl -n <namespace_name> apply -f my_new_ca.yml command.

4. Recreate the cluster from the original deploy/cr.yaml configuration file:

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

5. Verify certificate duration in usual way.

7.10.5 Run Percona Server for MongoDB without TLS

You can run Percona Server for MongoDB without TLS. For example, for testing or demonstration purposes. However, we recommend that you have the TLS protocol enabled.

You can start a new cluster without TLS or disable the TLS protocol for a running cluster. See the corresponding sections for steps.

Disable TLS for a new cluster

To disable TLS protocol for a new cluster, edit the deploy/cr.yaml Custom Resource manifest as follows:

- set the tls.mode key to disabled
- set the unsafeFlags.tls to true.

```
spec:
...
unsafeFlags
tls: true
...
tls:
mode: disabled
```

Apply the manifest:

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

Disable TLS for a running cluster

To disable TLS protocol for a running cluster, follow these steps:

1. Pause the cluster. Since the cluster is running, run the kubectl patch command to update the cluster configuration. Replace the <namespace> placeholder with your namespace. For example, for the cluster with the name my-cluster-name, the command is:

```
$ kubectl patch psmdb my-cluster-name -n <namespace> --type json -p='[{"op":"add","path":"/spec/pause","value":true}]'
```

2. Wait for the cluster to be paused. Check the status with the kubectl get psmdb command:

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



3. Disable the TLS protocol by setting the following configuration in the deploy/cr.yaml Custom Resource manifest:

```
spec:
...
unsafeFlags
tls: true
...
tls:
mode: disabled
```

4. Apply the changes:

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

5. Now resume the cluster with the kubectl patch command:

```
$ kubectl patch psmdb my-cluster-name -n <namespace> --type json -p='[{"op":"add","path":"/spec/pause","value":false}]'
```

6. Wait for the cluster to be resumed. Check the status with the kubectl get psmdb command.

Re-enable TLS

To re-enable TLS protocol for a running cluster, follow these steps:

1. Pause the cluster. Edit the deploy/cr.yaml Custom Resource manifest and set spec.pause key to true:

```
spec:
pause: true
```

2. Apply the changes:

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

3. Wait for the cluster to be paused. Check the status with the kubectl get psmdb command:

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

4. Enable the TLS protocol by setting the following configuration in the deploy/cr.yaml Custom Resource manifest:

```
spec:
...
unsafeFlags
  tls: false
  ...
tls:
  mode: preferTLS
```

5. Apply the changes:

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

6. Now resume the cluster. Edit the deploy/cr.yaml Custom Resource manifest and set the spec.pause key to false:

```
spec:
pause: false
```

7. Apply the changes:

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

8. Wait for the cluster to be resumed. Check the status with the kubectl get psmdb command.

7.11 Data at rest encryption

<u>Data at rest encryption in Percona Server for MongoDB</u> ☐ is supported by the Operator since version 1.1.0.



Data at rest C 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.

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].

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

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

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

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

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

Using HashiCorp Vault storage for encryption keys

Starting from the version 1.13, the Operator supports using <u>HashiCorp Vault</u> of 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)</pre>
```

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 <u>Vault token</u> . 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.VgQvaX18xGF01RUxAPbPbsfN
```

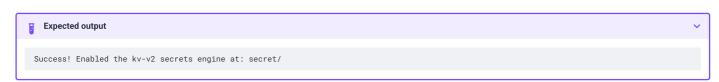
Now login to Vault with this token to enable the key-value secret engine:

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



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

\$ vault secrets enable -path secret kv-v2





2. Now generate Secret with the Vault root token using kubect1 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: |
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 [4].

7.12 Telemetry

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

The following information is gathered:

- ID of the Custom Resource (the metadata.uid field)
- · Kubernetes version
- · Platform (is it Kubernetes or Openshift)
- Is PMM enabled, and the PMM Version
- Operator version
- · Mongo version
- · Percona Backup for MongoDB (PBM) version
- Is sharding enabled (starting from the Operator version 1.13)
- Is <u>Hashicorp Vault</u> enabled (starting from the Operator version 1.13)
- Is the Operator deployed in a <u>cluster-wide mode</u> (starting from the Operator version 1.13)
- Is Volume Expansion enabled (starting from the Operator version 1.19)
- Are <u>multi-cluster Services</u> enabled (starting from the Operator version 1.19)
- Does the Operator manage <u>custom MongoDB users</u> and/or <u>custom MongoDB roles</u> (starting from the Operator version 1.19)
- Is the Operator deployed with Helm
- Are sidecar containers used
- · Are backups used, are point-in-time recovery and/or scheduled physical backup features used, if so
- · How large is the cluster

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

Telemetry is enabled by default and is sent to the <u>Version Service server</u> 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 <a href="mailto:

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 1.20.0 --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"
...
```

8 Management

8.1 Backup and restore

8.1.1 About backups

You can back up your data in two ways:

- On-demand. You can do them manually at any moment.
- Scheduled backups. Configure backups and their schedule in the deploy/cr.yaml. The Operator makes them automatically according to the specified schedule.

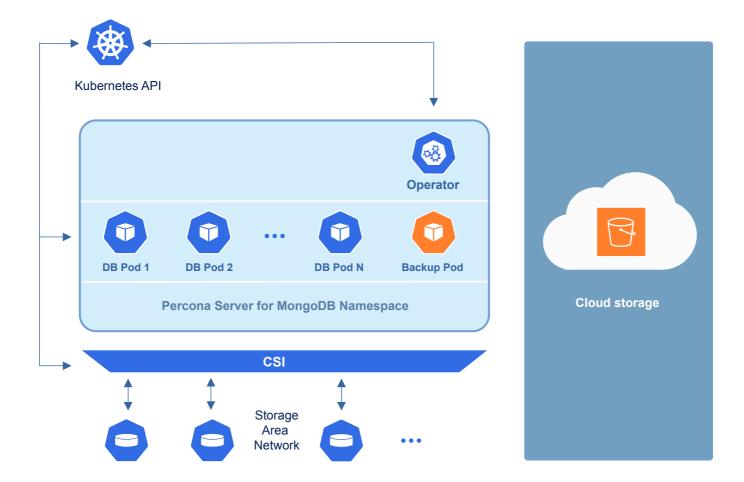
To make backups and restores, the Operator uses the <u>Percona Backup for MongoDB (PBM)</u> of tool. The Operator runs PBM as <u>a sidecar container</u> to the database Pods. It configures PBM in the following cases:

- when it creates a new cluster if you defined the <u>backup storage configuration</u> for it.
- when you configure the backup storage for a backup
- when you start a restore on a new cluster and defined the backup storage configuration within the backupSource subsection of the Restore resource.

Backup storage

You can store Percona Server for MongoDB backups outside the Kubernetes cluster using the following remote backup storages:

- MinIO ☐ S3-compatible storage



Multiple backup storages

Starting with version 1.20.0, the Operator natively supports multiple backup storages [2], inheriting this feature from Percona Backup for MongoDB (PBM). This means you don't have to wait till the Operator reconfigures a cluster after you select a different storage for a backup or a restore. And you can make a point-in-time recovery from any backup stored on any storage - PBM and the Operator maintain the data consistency for you.

Find more information in the <u>Multiple storages for backups</u> chapter.

Backup types

Backup type	Version added	Status	Description	Important considerations
Full logical	Initial	GA	Queries Percona Server for MongoDB for database data and writes this data to the remote storage	 - Uses less storage but is slower than physical backups - Supports selective restore since 1.18.0 - Supports point-in-time recovery - Incompatible for restores with backups made with Operator versions before 1.9.0. Make a new backup after the upgrade to the Operator 1.9.0.
Full physical	1.14.0	GA (1.16.0)	Copies physical files from MongoDB dbPath data directory to remote storage	- Faster backup/restore than logical - Better for large datasets - Supports point-in-time recovery since 1.15.0
Physical incremental	1.20.0	Tech preview	Copies only data changed after the previous backup	- Speeds up backup/restore - Reduces network load and storage consumption - Requires a base incremental backup to start the incremental chain - Base backup and increments must bet taken from the same node - New base backup is needed if a node is down or if the cluster was restored from a backup

8.1.2 Configure storage for backups

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



Warning

Remote storage for backups has the technical preview status

You should also create the Kubernetes Secret [5] object with credentials needed to access the storage.

Amazon S3 or S3-compatible storage

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

Create the Secrets file with these base64-encoded keys following the $\underline{deploy/backup\text{-}s3.yaml}$ \square example:

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

Note

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

in Linux

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

in macOS

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

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

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

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

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

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

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

```
backup:
...
storages:
s3-us-west:
type: s3
s3:
bucket: S3-BACKUP-BUCKET-NAME-HERE
region: us-west-2
credentialsSecret: my-cluster-name-backup-s3
...
```

Finally, make sure that your storage has enough resources to store backups, which is especially important in the case of large databases. It is clear that you need enough free space on the storage. Beside that, S3 storage <u>upload limitats</u> of include the maximum number 10000 parts, and backing up large data will result in larger chunk sizes, which in turn may cause S3 server to run out of RAM, especially within the default memory limits.

Automating access to Amazon s3 based on IAM roles

Using AWS EC2 instances for backups makes it possible to automate access to AWS S3 buckets based on <u>Identity Access Management (IAM) roles</u> of Service Accounts with *no need to specify the S3 credentials explicitly*.

You can use either make and use the *IAM instance profile*, or configure *IAM roles for Service Accounts* (both ways heavily rely on AWS specifics, and need following the official Amazon documentation to be configured).

Using IAM instance profile

Following steps are needed to turn this feature on:

- 1. Create the IAM instance profile [7] and the permission policy within where you specify the access level that grants the access to S3 buckets.
- 2. Attach the IAM profile to an EC2 instance.
- 3. Configure an S3 storage bucket in the Custom Resource and verify the connection from the EC2 instance to it.
- ${\tt 4.}\,\textit{Do}\,\textit{not}\,\textit{provide}\,\, {\tt s3.credentialsSecret}\,\, {\tt for}\,\, {\tt the}\,\, {\tt storage}\,\, {\tt in}\,\, {\tt deploy/cr.yaml.}$

Using IAM role for service account

IRSA [3] is the native way for the cluster running on Amazon Elastic Kubernetes Service (AWS EKS) to access the AWS API using permissions configured in AWS IAM roles.

Assuming that you have deployed the MongoDB Operator and the database cluster on EKS, following our installation steps, and your EKS cluster has OpenID Connect issuer URL (OIDC) OID enabled, the the high-level steps to configure it are the following:

- 1. Create an IAM role for your OIDC, and attach to the created role the policy that defines the access to an S3 bucket. See official Amazon documentation of ordetails.
- 2. Find out service accounts used for the Operator and for the database cluster. Service account for the Operator is percona-server-mongodb-operator (it is set by the serviceAccountName key in the deploy/operator.yaml or deploy/bundle.yaml manifest) The cluster's default account is default (it can be set with serviceAccountName Custom Resource option in the replsets, sharding.configsvrReplSet, and sharding.mongos subsections of the deploy/cr.yaml manifest).
- 3. Annotate both service accounts with the needed IAM roles. The commands should look as follows:

```
$ kubectl -n <cluster namespace> annotate serviceaccount default eks.amazonaws.com/role-arn:
arn:aws:iam::111122223333:role/my-role --overwrite
$ kubectl -n <operator namespace> annotate serviceaccount percona-server-mongodb-operator eks.amazonaws.com/role-arn:
arn:aws:iam::111122223333:role/my-role --overwrite
```

Don't forget to substitute the operator namespace> and <cluster namespace> placeholders with the real namespaces, and use your IAM role instead
 of the eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/my-role example.

4. Configure an S3 storage bucket in the Custom Resource and verify the connection from the EC2 instance to it. *Do not provide* s3.credentialsSecret for the storage in deploy/cr.yaml.



If IRSA-related credentials are defined, they have the priority over any IAM instance profile. S3 credentials in a secret, if present, override any IRSA/IAM instance profile related credentials and are used for authentication instead.

Microsoft Azure Blob storage

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

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

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

Vou can use the following command to get a base64-encoded string from a plain text one: in Linux \$ echo -n 'plain-text-string' | base64 --wrap=0 in macOS \$ echo -n 'plain-text-string' | base64

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

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

- 2. Put the data needed to access the Azure Blob storage into the backup.storages subsection of the Custom Resource.
 - storages.<NAME>.type should be set to azure` (substitute the part with some arbitrary name you will later use to refer this storage when making backups and restores).
 - storages.<NAME>.azure.credentialsSecret key should be set to the name used to refer your Kubernetes Secret (my-cluster-azure-secret in the last example).
 - storages.<NAME>.azure.container option should contain the name of the Azure container. Also you can use storages.<NAME>.azure.prefix option to specify the path (sub-folder) to the backups inside the container. If prefix is not set, backups are stored in the root directory of the container.

These and other options within the storages. < NAME > . azure subsection are further described in the Operator Custom Resource options.

Here is an example of the deploy/cr.yaml deploy/cr.yaml

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

Remote file server

You can use fileystem backup storage type to mount a remote file server to a local directory as a sidecar volume, and make Percona Backup for MongoDB using this directory as a storage for backups.

The approach is based on using common Network File System (NFS) protocol . Particularly, this storage type is useful in network-restricted environments without S3-compatible storage, or in cases with a non-standard storage service that still supports NFS access.

1. Add the remote storage as a <u>sidecar volume</u> in the <u>replset</u> section of the Custom Resource (and also in <u>configsvrReplSet</u> in case of a sharded cluster). You will need to specify the server hostname and some directory on it, as in the following example:

```
replsets:
- name: rs0
...
sidecarVolumes:
- name: backup-nfs-vol
   nfs:
    server: "nfs-service.storage.svc.cluster.local"
    path: "/psmdb-my-cluster-name-rs0"
...
```

The backup-nfs-vol name specified above will be used to refer this sidecar volume in the backup section.

2. Now put the mount point (the local directory path to which the remote storage will be mounted) and the name of your sidecar volume into the backup.volumeMounts subsection of the Custom Resource:

```
backup:
...
volumeMounts:
- mountPath: /mnt/nfs/
  name: backup-nfs-vol
...
```

3. Finally, storage of the filesystem type needs to be configured in the backup.storages subsection. It needs only the mount point:

```
backup:
    enabled: true
    ...
    storages:
    backup-nfs:
    type: filesystem
    filesystem:
       path: /mnt/nfs/
```

8.1.3 Multiple storages for backups

You can define several storage locations for backups in the Operator. However, previously you were limited to only a single storage for point-in-time recovery, because Percona Backup for MongoDB (PBM) couldn't maintain oplog consistency across multiple storages. Also, you had to wait for the Operator to reconfigure the cluster and sync metadata after you make the next backup or a restore to a different storage.

This behavior is improved. The Operator differentiates the storages as the main storage and profiles. The difference between them is that the Operator uses the main storage to save both backups and oplog chunks for point-in-time recovery. Profiles are used only for backups. This is done for data consistency and to enable point-in-time recovery from a backup on any storage.

Define the main storage

When you configure only one storage, the Operator automatically uses it as the main one until you add more. When you add another storage, you must mark which one is the main using the main: true flag in the deploy/cr.yaml Custom Resource manifest.

```
storages:
s3-us-west:
main: true
type: s3
```

Note that you can have only one main storage. All other storages are added as profiles.

To check the list of profiles, connect to the database Pod and run the pbm profile list command. For example, for the cluster cluster1, the command looks as follows:

```
$ kubectl exec cluster1-rs0-0 -c backup-agent -- pbm profile list
```

You can run other profile management commands [] in the same way.

Change the main storage

You can change the main storage by reassigning the main:true flag for another one. The Operator then:

- · Resyncs the metadata for the new main storage
- Deletes the profile for it
- Adds the previous main storage as a profile

Pass storage configuration via restore objects

Usually you define the storage configuration within the deploy/cr.yaml Custom Resource manifest. You can also pass it to the Operator within the backupSource option of a Restore object. For example, when you restore the failed site after a disaster. The Operator then checks the current configuration and:

- If there is no storage configured there, it uses the one from the Restore object as the main storage until you define it in the deploy/cr.yaml file.
- If the deploy/cr.yaml Custom Resource manifest has the storage configured and it differs from the one from the Restore object, the Operator adds this storage as a profile.

Backup metadata resync

The Operator resyncs the metadata in the following cases:

For the main storage:

- When the main storage changes
- When you manually start a resync using the pbm config --force resync

For the profile storage:

- When you start a restore from a backup on a profile
- When you manually resync the metadata on a profile using the pbm profile sync <storage-name> command

For the main storage and profiles:

• When you added an annotation percona.com/resync-pbm=true to the deploy/cr.yaml Custom Resource manifest.

Note that resync is a resource consuming task and we don't recommend to run it manually. Read more when you need to run it in PBM documentation 🖸

The improved support for multiple backup storages brings the following benefits:

- Enables you to make a point-in-time recovery from any storage with guaranteed data consistency
- Reduces the load on the cluster for reconfiguration when the storage for a next backup changes

Upgrade considerations

You must specify the main storage during the upgrade. If you use a single storage, it will automatically be marked as main in the Custom Resource manifest. If you use multiple storages, you must define one of them as main.

The following command shows how to set the s3-us-west storage as the main one:

For more information about the upgrades, see the <u>Update documentation</u>

8.1.4 Making scheduled backups

You can automate the backup process with scheduled backups. You define a schedule and the Operator runs backups automatically according to it. This provides reliability and efficiency to your backups strategy and ensures your data is timely and regularly backed up with no gaps.

To configure scheduled backups, modify the backups section of the deploy/cr.yaml C Custom Resource manifest. Specify the following configuration:

- 1. backup.enabled set to true,
- 2. backup.storages subsection define at least one configured storage.
- 3. Configure the backup.tasks subsection:
 - name specify a backup name. You will need this name when you restore from this backup.
 - schedule specify the desired backup schedule in crontab format (2).
 - · enabled set this key to true. This enables making the <backup name> backup along with the specified schedule.
 - storageName specify the name of your <u>already configured storage</u>.
 - · keep define the number of backups to keep in the storage. This key is optional. It applies to base incremental backups but is ignored for increments.
 - type specify what type of backup to make. If you leave it empty, the Operator makes a logical backup by default.

Note that the percona.com/delete-backup finalizer applies for an incremental base backup but is ignored for increments. This means that when an incremental base backup is deleted, PBM also deletes all increments that derived from it from the backup storage. There is the limitation that the Backup resource for the base incremental backup is deleted but the Backup resources for increments remain in the Operator. This is because the Operator doesn't control their deletion outsourcing this task to PBM. This limitation will be fixed in future releases.

Examples

Logical

This example shows how to set up backups to run every Saturday night and store them in Amazon S3:

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

Physical

This example shows how to set up backups to run every Saturday night and store them in Amazon S3:

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

Incremental

To run incremental backups, consider the following:

- 1. You must use the same storage for the base backup and subsequent incremental ones
- 2. The percona.com/delete-backup finalizer and the .spec.backup.tasks.[].keep option are is considered for incremental base backup but are ignored for increments. This means that when a base backup is deleted, PBM deletes all increments that derive from it.

There is the limitation that the Backup resource for the base incremental backup is deleted but the Backup resources for increments remain in the Operator. This is because the Operator doesn't control their deletion outsourcing this task to PBM. This limitation will be fixed in future releases.

This example shows how to set up incremental base backups to run every Sunday at 5 a.m and subsequent incremental backups every night at 1:00 a.m. and store them in Amazon S3:

```
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
   - name: weekly-s3-us-west-incremental
     enabled: true
    schedule: "0 1 * * *"
    type: incremental
    storageName: s3-us-west
    compressionType: gzip
    compressionLevel: 6
   - name: weekly-s3-us-west-incremental-base
    enabled: true
    schedule: "0 5 * * 0"
    type: incremental-base
     storageName: s3-us-west
    compressionType: gzip
    compressionLevel: 6
```

Restore to a new Kubernetes environment

To <u>restore from a backup to a new Kubernetes-based environment</u>, you must create a Secrets object there with the same user passwords as in the original cluster.

Find the Secrets name object on the source cluster in the spec.secrets key in the 'deploy/cr.yaml. Use this name to recreate the Secrets on the target cluster.

Find more details about secrets in **System Users**.

8.1.5 Making on-demand backups

An on-demand backup is a backup that you start manually at any time. You create a Backup resource and the Operator uses it to make a backup. A backup can be any of the <u>supported backup types</u>.

If you want to run backups automatically, according to the schedule, see Make scheduled backups tutorial.

Here's what you need to do to run on-demand backups:

Modify the Custom Resource manifest

- 1 Edit the deploy/cr.yaml configuration file and specify the following configuration:
 - Set the backup.enabled key to true,
 - Check that you have defined at least one <u>configured storage</u> in the backup.storages subsection.
- 2 Apply the changes. Don't forget to replace the <namespace> placeholder with your namespace:

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

Create a Backup resource

To create a Backup resource, you need a special custom resource manifest. The deploy/backup/backup.yaml [] is the example manifest that you can use.

- 1 Specify the following configuration:
 - et metadata. name is the name of the backup. You will need this name when you restore from this backup. The default name is backup. 1.
 - spec.clusterName is the name of your cluster (prior to the Operator version 1.12.0 this key was named spec.psmdbCluster). Run kubectl get psmdb -n <namespace> to find out the cluster name.
 - spec.storageName is the name of your <u>already configured storage</u>.
 - spec.type is the <u>backup type</u>. If you leave it empty, the Operator makes a logical backup by default.

Examples

Logical

```
apiVersion: psmdb.percona.com/v1
kind: PerconaServerMongoDBBackup
metadata:
    finalizers:
    - percona.com/delete-backup
    name: backup1
spec:
    clusterName: my-cluster-name
    storageName: s3-us-west
    type: logical
```

Physical

```
apiVersion: psmdb.percona.com/v1
kind: PerconaServerMongoDBBackup
metadata:
    finalizers:
        - percona.com/delete-backup
        name: backup1
spec:
    clusterName: my-cluster-name
    storageName: s3-us-west
    type: physical
```

Incremental

To make incremental backups, consider the following:

- 1 Make the incremental base backup first. The Operator needs the base to start the chain of increments and save only changes from previous backup.
- 2 Use the same storage for base backup and increments.
- 3 The percona.com/delete-backup finalizer is considered for incremental base backup but is ignored for increments. This means that when a base backup is deleted, PBM deletes all increments that derive from it.

There is the limitation that the Backup resource for the base incremental backup is deleted but the Backup resources for increments remain in the Operator. This is because the Operator doesn't control their deletion outsourcing this task to PBM. This limitation will be fixed in future releases.

Here's the configuration example for the base incremental backup

```
apiVersion: psmdb.percona.com/v1
kind: PerconaServerMongoDBBackup
metadata:
    finalizers:
        percona.com/delete-backup
        name: backup1
spec:
    clusterName: my-cluster-name
    storageName: s3-us-west
    type: incremental-base
```

This configuration example is for subsequent incremental backups:

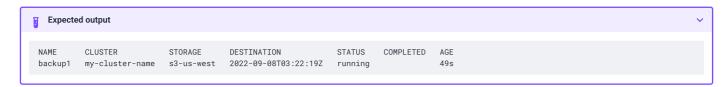
```
apiVersion: psmdb.percona.com/v1
kind: PerconaServerMongoDBBackup
metadata:
   name: backup1
spec:
   clusterName: my-cluster-name
   storageName: s3-us-west
   type: incremental
```

2 Apply the backup.yaml manifest to start a backup:

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

3 You can track the backup process with the PerconaServerMongoDBBackup Custom Resource as follows:

\$ kubectl get psmdb-backup



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

Troubleshooting

If you have any issues with a backup, here's how you can troubleshoot it:

1. View information about a backup:

\$ kubectl describe psmdb-backup backup1

```
Expected output
              backup1
 Namespace: my-namespace
 Labels:
              <none>
 Annotations: <none>
 API Version: psmdb.percona.com/v1
 Kind:
              PerconaServerMongoDBBackup
 Metadata:
  Creation Timestamp: 2025-04-22T11:32:21Z
  Finalizers:
    percona.com/delete-backup
   Generation:
  Resource Version: 136319
                    46670473-4fd0-465d-b944-1be3717485a0
  Cluster Name: my-cluster-name
  Storage Name: gcp-cs
  Type:
               incremental-base
 Status:
  Completed: 2025-04-22T11:32:42Z
   Destination:
                   s3://my-bucket/demand-backup-incremental/2025-04-22T11:32:26Z
  Last Transition: 2025-04-22T11:32:42Z
  Pbm Name:
                   2025-04-22T11:32:26Z
   Pbm Pod:
                   my-cluster-name-rs0.2.my-cluster-name-rs0.demand-backup-incremental-10277.svc.cluster.local:27017
    rs0: my-cluster-name-rs0-2.my-cluster-name-rs0.demand-backup-incremental-10277.svc.cluster.local:27017
   Replset Names:
    rs0
   s3:
    Bucket:
                        my-bucket
    Credentials Secret: gcp-cs-secret
    Endpoint URL:
                       https://storage.googleapis.com
    Prefix:
                       demand-backup-incremental
    Region:
                        us-east-1
    Server Side Encryption:
   Start: 2025-04-22T11:32:26Z
   State:
   Storage Name: gcp-cs
   Type:
                incremental-base
```

2. Check logs from the backup-agent container of the appropriate Pod as follows. Find the Pod name in the pbm Pod field in the output from the previous step.

Or use the following command to get the Pod name:

```
$ kubectl get psmdb-backup -o yaml | grep pbmPod
```

Now connect to the backup-agent of this Pod:

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

3. Access the same container via ssh and carry on Percona Backup for MongoDB diagnostics [4].

Restore to a new Kubernetes environment

To <u>restore from a backup to a new Kubernetes-based environment</u>, you must create a Secrets object there with the same user passwords as in the original cluster.

Find the Secrets name object on the source cluster in the spec.secrets key in the 'deploy/cr.yaml. Use this name to recreate the Secrets on the target cluster.

Find more details about secrets in **System Users**.

8.1.6 Storing operations logs for point-in-time recovery

Point-in-time recovery enables you to roll back your cluster to a specific date and time. The Operator first restores a backup and then applies an operation log (oplog) - the changes that occurred to the operations up to the defined moment. To do so, the Operator saves oplog to the cloud storage.

Starting from the Operator version 1.15.0, you can do a point-in-time recovery from both logical and physical backups.

To start saving oplog, set the backup.pitr.enabled key in the deploy/cr.yaml configuration file:

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

You must have a full backup to use point-in-time recovery. Without a full backup, Percona Backup for MongoDB will not upload operations logs. You must have a full backup for a new cluster and for a cluster that you restored from a backup.

After you enabled point-in-time recovery, it takes 10 minutes for a first oplog chunk to be uploaded. The default time period between uploads is 10 minutes. You can adjust this time by setting the new duration for the 'backup.pitr.oplogSpanMin` option.

Point-in-time recovery with multiple storages

Version 1.20.0 and above

The Operator natively supports <u>multiple storages for backups</u> inheriting this functionality from Percona Backup for MongoDB. This allows you to enable point-in-time recovery and make backups on a storage of your choice. PBM saves oplog only to the main storage to ensure data consistency for all backups on all storages. As a result, you can <u>make a point-in-time restore</u> from any backup on any storage.

Version 1.19.1 and earlier

You must have a single storage defined in the <u>spec.backup.storages</u> option to enable point-in-time recovery. This is because PBM writes oplog to the same bucket where the backup snapshot is saved.

If you defined several storages and try to enable point-in-time recovery, PBM won't know where to save oplog and can't therefore guarantee data consistency for the restore. Therefore, point-in-time recovery is not allowed for multiple storages. You will see the error message in the Operator logs.

8.1.7 Enable server-side encryption for backups

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

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

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

Encryption with keys stored in AWS KMS

To use the server-side AWS KMS encryption, specify the ID of your customer-managed key. C and other needed options as follows:

with kmsKeyID in Custom Resource

Set the following Custom Resource options in the deploy/cr.yaml configuration file:

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

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

with kmsKeyID in Secret object

You can avoid storing your kmsKeyID in Custom Resource, and put it into a dedicated Secrets object. Define your secret in YAML as follows:

```
deploy/sse-secret.yaml

apiVersion: v1
kind: Secret
metadata:
   name: my-cluster-name-sse
type: Opaque
stringData:
   KMS_KEY_ID: <kms_key_ID>
```

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

When the YAML file is ready, apply it to create the Secret:

```
$ kubectl create -f deploy/sse-secret.yaml
```

After creating the Secret, set the following Custom Resource options in the deploy/cr.yaml configuration file:

```
secrets:
...
sse: my-cluster-name-sse
...
backup:
...
storages:
  my-s3:
  type: s3
  s3:
  bucket: my-backup-bucket
  serverSideEncryption:
    sseAlgorithm: aws:kms
```

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

The Operator also supports server-side encryption with customer-provided keys that are stored on the client side. During the backup/restore process, encryption key will be provided by the Operator as part of the requests to the S3 storage, and the S3 storage will use them to encrypt/decrypt the data with the AES-256 encryption algorithm. This allows to use server-side encryption on S3-compatible storages different from AWS KMS (the feature was tested with the AWS C3 and MinIO C3 storages).

To use the server-side encryption with locally-stored keys, specify your encryption key and other needed options:

with encryption key in Custom Resource

Set the following Custom Resource options in the deploy/cr.yaml configuration file:

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

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

with encryption key in Secret object

You can avoid storing your encryption key in Custom Resource, and put it into a dedicated Secrets object. Define your secret in YAML as follows:

```
deploy/sse-secret.yaml

apiVersion: v1
kind: Secret
metadata:
   name: my-cluster-name-sse
type: Opaque
stringData:
   SSE_CUSTOMER_KEY: <your_encryption_key_in_base64>
```

 $Here < your_encryption_key_in_base 64 > should be substituted with the actual encryption key encoded in base 64.$

When the YAML file is ready, apply it to create the Secret:

```
$ kubectl create -f deploy/sse-secret.yaml
```

After creating the Secret, set the following Custom Resource options in the deploy/cr.yaml configuration file:

```
secrets:
...
sse: my-cluster-name-sse
...
backup:
...
storages:
  my-s3:
  type: s3
  s3:
    bucket: my-backup-bucket
    serverSideEncryption:
    sseCustomerAlgorithm: AES256
...
```

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

in Linux

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

in macOS

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

8.1.8 Restore the cluster from a previously saved backup

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

Following things are needed to restore a previously saved backup:

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

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

And the following command will list available clusters:

\$ kubectl get psmdb



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

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

Without point-in-time recovery

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

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

2. After that, the actual restoration process can be started as follows:

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

```
Note
```

Storing backup settings in a separate file can be replaced by passing its content to the kubect1 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</pre>
```

With point-in-time recovery

1. Set appropriate keys in the deploy/backup/restore.yaml deploy

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

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

Note

Full backup objects available with the kubect1 get psmdb-backup command have a "Latest restorable time" information field handy when selecting a backup to restore. You can easily query the backup for this information as follows:

```
$ kubectl get psmdb-backup <backup_name> -o jsonpath='{.status.latestRestorableTime}'
```

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
```

Selective restore

Starting with the version 1.18.0, you can restore a desired subset of data from a full logical backup. Selective logical backups are not yet supported.

Selective restores are controlled by the additional selective section in the PerconaServerMongoDBRestore Custom Resource. There you can specify a specific database or a collection that you wish to restore:

```
spec:
    selective:
    withUsersAndRoles: true
    namespaces:
    - "db1.collection1"
    - "db2.collection2"
```

The selective.namespaces field allows you to specify several "namespaces" (subsets of data) as a list. Each "namespace" is represented as a pair of database and collection names, or just database_name.* to get everything from the specific database. Specifying "*" as an item in the namespaces means restoring all databases and collections.

Also, you can use selective .withUsersAndRoles set to true to restore a custom database with users and roles from a full backup.

Selective restores support only logical backups and have a number of other limitations. See the full list of <u>current selective restore limitations</u> \(\tilde{\su} \) in Percona Backup for MongoDB documentation.

8.1.9 Delete the unneeded backup

The maximum amount of stored backups is controlled by the <u>backup.tasks.keep</u> 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>



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

8.2 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.

Scaling can be vertical and horizontal. Vertical scaling adds more compute or storage resources to MongoDB nodes; horizontal scaling is about adding more nodes to the cluster. High availability looks technically similar, because it also involves additional nodes, but the reason is maintaining liveness of the system in case of server or network failures.

Vertical scaling

Scale compute

To add more resources to your MongoDB replica set instances, edit the following section in the Custom Resource:

```
spec:
    replsets:
    resources:
    requests:
    memory: 4G
    cpu: 2
    limits:
    memory: 4G
    cpu: 2
```

Use our reference documentation for the <u>Custom Resource options</u> for more details about other components.

Scale storage

Kubernetes manages storage with a PersistentVolume (PV), a segment of storage supplied by the administrator, and a PersistentVolumeClaim (PVC), a request for storage from a user. In Kubernetes v1.11 the feature was added to allow a user to increase the size of an existing PVC object (considered stable since Kubernetes v1.24). The user cannot shrink the size of an existing PVC object.

Starting from the version 1.16.0, the Operator allows to scale Percona Server for MongoDB storage automatically by changing the appropriate Custom Resource option, if the volume type supports PVCs expansion.

Automated scaling with Volume Expansion capability



AllowVolumeExpansion: true

Automated storage scaling by the Operator is in a technical preview stage and is not recommended for production environments.

Certain volume types support PVCs expansion (exact details about PVCs and the supported volume types can be found in Kubernetes documentation C).

You can run the following command to check if your storage supports the expansion capability:

\$ kubectl describe sc <storage class name> | grep AllowVolumeExpansion

Expected output

V

You can enable automated scaling with the enableVolumeExpansion Custom Resource option (turned off by default). When enabled, the Operator will automatically expand such storage for you when you change the replace. NAME>.volumeSpec.persistentVolumeClaim.resources.requests.storage and/or configsvrReplSet.volumeSpec.persistentVolumeClaim.resources.requests.storage options in the Custom Resource.

For example, you can do it by editing and applying the deploy/cr.yaml file:

```
spec:
...
enableVolumeExpansion: true
...
replsets:
...
volumeSpec:
   persistentVolumeClaim:
   resources:
    requests:
    storage: <NEW STORAGE SIZE>
```

Apply changes as usual:

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

Manual scaling without Volume Expansion capability

Manual scaling is the way to go if your version of the Operator is older than 1.16.0, your volumes have type which does not support Volume Expansion, or you just do not rely on automated scaling.

You will need to delete Pods one by one and their persistent volumes to resync the data to the new volumes. This can also be used to shrink the storage.

1. Update the Custom Resource with the new storage size by editing and applying the deploy/cr.yaml file:

```
spec:
...
replsets:
...
volumeSpec:
persistentVolumeClaim:
    resources:
    requests:
    storage: <NEW STORAGE SIZE>
```

Apply the Custom Resource update in a usual way:

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

2. Delete the StatefulSet with the orphan option

```
$ kubectl delete sts <statefulset-name> --cascade=orphan
```

The Pods will not go down and the Operator is going to recreate the StatefulSet:

\$ kubectl get sts <statefulset-name>



3. Scale up the cluster (Optional)

Changing the storage size would require us to terminate the Pods, which decreases the computational power of the cluster and might cause performance issues. To improve performance during the operation we are going to change the size of the cluster from 3 to 5 nodes:

```
spec:
...
replsets:
...
size: 5
```

Apply the change:

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

New Pods will already have new storage:

```
$ kubectl get pvc
```



4. Delete PVCs and Pods with old storage size one by one. Wait for data to sync before you proceeding to the next node.

```
$ kubectl delete pvc <PVC NAME>
$ kubectl delete pod <POD NAME>
```

The new PVC is going to be created along with the Pod.

Horizontal scaling

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



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,
- <u>sharding.configsvrReplSet.size</u> allows to set the number of <u>Config Server instances</u> [2],
- <u>sharding.mongos.size</u> allows to set the number of <u>mongos</u> ☐ instances.

For example, the following update in deploy/cr.yaml will set the size of the MongoDB Replica Set to 5 nodes:

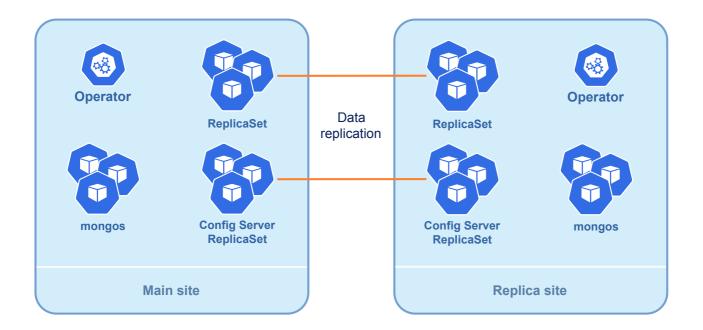
```
spec:
    ...
replsets:
    ...
    size: 5
```

Don't forget to apply changes as usual, running the kubectl apply -f deploy/cr.yaml command.

8.3 Multi-cluster and multi-region deployment

8.3.1 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:



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
- keep the replica set of the database cluster in different data centers to get a fault-tolerant system.

Prerequisites

- Every node in Main and Replica clusters need to be reachable through network.
- · User credentials should be the same in each cluster.
- TLS certificates should be the same in each cluster.

Glossary

- Main cluster: The cluster which the primary node runs and accepts write traffic. It's the managed cluster if it's running on Kubernetes.
- Replica cluster: The cluster which is configured to replicate from main cluster. It's the unmanaged cluster if it's running on Kubernetes.
- Managed cluster: The cluster controlled by operator. The operator controls everything from Replica Set configuration to users credentials. It's the default deployment of the operator.
- Unmanaged cluster: The cluster controlled by operator but the operator isn't responsible for managing Replica Set configuration [].

Topologies

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.

You need to have a single *Main* cluster but you can have multiple *Replica* clusters as long as you don't have more than 50 members in Replica Set. This limitation comes from MongoDB itself, for more information please check <u>MongoDB docs</u> .

Main and Replica clusters on Kubernetes

If you want both Main and Replica clusters to run on Kubernetes, overall steps will look like:

- 1. Deploy the Main cluster on a Kubernetes cluster (or use an existing one)
- 2. Get TLS secrets from the Main cluster and apply them to the namespace in Kubernetes cluster to which you'll deploy the Replica cluster
- 3. Deploy Replica cluster on a Kubernetes cluster
- 4. Add nodes from the Replica cluster to the Main cluster as external nodes

Main cluster on Kubernetes and Replica cluster outside of Kubernetes

If you want Main cluster to run on Kubernetes, but Replica cluster outside of Kubernetes, overall steps will look like:

- 1. Deploy the Main cluster on a Kubernetes cluster (or use an existing one)
- 2. Get TLS secrets from the Main cluster to configure the Replica cluster
- 3. Deploy the Replica cluster on wherever you want
- 4. Add nodes from the Replica cluster to the Main cluster as external nodes

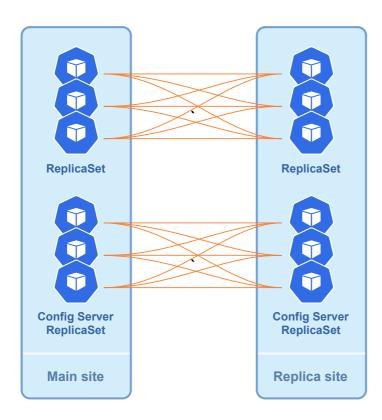
Main cluster outside of Kubernetes and Replica cluster on Kubernetes

If you want Main cluster to run outside of Kubernetes but Replica cluster on Kubernetes, overall steps will look like:

- 1. Deploy the Main cluster on wherever you want (or use an existing one)
- 2. Get TLS certificates and create a Kubernetes Secret with them
- 3. Get user credentials and create a Kubernetes Secret with them
- 4. Deploy the Replica cluster on a Kubernetes cluster
- 5. Add nodes from the Replica cluster to the Main cluster using Mongo client

Exposing instances of the MongoDB cluster

You need to expose all Replica Set nodes (including Config Servers) through a dedicated Service to ensure that both the *Main* and the *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
     type: LoadBalancer
  sharding:
   configsvrReplSet:
      expose:
       enabled: true
       type: LoadBalancer
```

The above example is using the LoadBalancer Kubernetes Service object, and the result will be a LoadBalancer per each Replica Set Pod. In most cases, this Load Balancer should be Internet-facing for cross-region replication to work. Also, there are other options except the LoadBalancer (ClusterIP, NodePort, etc.).



Starting from v1.14, the Operator configures the replset using local DNS hostnames even if the replset is exposed. If you want to have IP addresses in the replset configuration to achieve a multi-cluster deployment, $\underline{you\;need}$ to set clusterServiceDNSMode to External.

To list the endpoints assigned to Pods, list the Kubernetes Service objects by executing kubectl get services -1 "app.kubernetes.io/instance=CLUSTER_NAME" command.

8.3.2 Configuring cross-site replication on the Main site

The cluster managed by the Operator should be able to reach external nodes of the Replica Sets. You can provide needed information in the replsets.externalNodes and sharding.configsvrReplset.externalNodes subsections of the deploy/cr.yaml configuration file. Following keys can be set to specify each external Replica, both for its Replica Set and Config Server instances:

- · set host to URL or IP address of the external replset instance,
- set port to the port number of the external node (or rely on the 27017 default value),
- set priority to define the priority [2] of the external node (2) is default for all local members of the cluster; external nodes should have lower priority to avoid unmanaged node being elected as a primary; 0 adds the node as a non-voting member),
- set votes to the number of votes an external node can cast in a replica set election (0 is default and should be used for non-voting members of the cluster).

Here is an example:

```
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
```



Don't forget to expose instances of the Main cluster!

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 secrets.ssl, secrets.sslInternal, secrets.users, and secrets.keyfile Custom Resource options. The default names are the following ones:

```
* `my-cluster-name-ssl` (SSL certificates for client connections),
* `my-cluster-name-ssl-internal` (SSL certificates for replication),
* `my-cluster-name-secrets` (user credentials),
* `my-cluster-name-mongodb-keyfile` (encryption key file).
```

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.

8.3.3 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, because Smart Updates are not allowed on unmanaged clusters. Also, the Operator versions prior to 1.19.0 did not support backups on unmanaged clusters, so set backup.enabled to false for the Operator 1.18.0 and older.



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

Here is an example:

```
spec:
  unmanaged: true
 updateStrategy: OnDelete
  replsets:
  - name: rs0
   size: 3
```

The Main and Replica sites should have the same Secrets objects, so don't forget to apply Secrets from your Main site. Names of the corresponding objects are set in the secrets.ssl, secrets.sslInternal, secrets.users, and secrets.keyfile Custom Resource options (my-cluster-name-ssl, my-clustername-ssl-internal, my-cluster-name-secrets, and my-cluster-name-mongodb-keyfile by default).

Note

Replica will not start if the TLS secrets and the encryption key are not copied. If users are not copied, the replica will join the replica set, but it will be restarting due to failed liveness

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

Note

Don't forget that you need to expose instances of the Replica cluster!

8.3.4 Backups with cross-site replication

Before the Operator version 1.19.0 Backups were supported for the primary (managed) cluster only. Now backups can be taken on primary and replica clusters.

Still, backups on cross-site configurations have some specifics.

- Even though you can run backups in unmanaged clusters, you can't run restores on them.
- Even if the backup is started in primary (managed) cluster, most likely it will be taken from a secondary instance, even if such instance is on a separate cluster, because Percona Backup for MongoDB (PBM) automatically assigns lower priority to primary member to avoid affecting the write performance. This can be overwritten with <u>custom PBM configuration</u>.
- PBM configuration is shared across all clusters. The Operator will reconfigure PBM every time it runs a backup, and setting PBM configuration in one cluster will affect other clusters too. For example, setting backup.configuration.backupOptions.oplogSpanMin to 2 in a secondary cluster will be applied to primary cluster as well.

8.3.5 Splitting replica set across multiple data centers

Splitting the replica set of the database cluster over multiple Kubernetes clusters can be useful to get a fault-tolerant system in which all replicas are in different data centers.

The Operator cannot deploy MongoDB replicas to other data centers, but this solution can be achieved with a number of Operator deployments, equal to the size of your replica set. So, you will need at least 3 Operator instances: one Operator to control the replica set via cross-site replication, and at least two Operators to bootstrap the unmanaged clusters. Each cluster will contain replica set with only one member, and the *Main* site will manage instances from other sites as external nodes. All configuration of the replica set is done manually.

The solution has the following limitations to consider:

- · setting it up involves a number of manual operations, and the same applies to scaling such a manually configured replica,
- backups are supported on the Main site only, not on the Replica sites.

Configuring the Main site

You will use the externally reachable URI for each of your replica set instances, manually overwiriting its default local fully-qualified domain name (FQDN) in the Custom Resouce manifest. Also you will need including all these host names into TLS certificates. So the first thing needed is the list of these externally reachable names. In the above example we will use the following ones:

- r1.percona.local:443 URI for the cluster-name-rs0-0 (1st replica set instance),
- r2.percona.local:443 for the 2nd replica set instance,
- r3.percona.local:443 for the 3rd replica set instance.

Following steps will allow you to prepare the Main site for cross-site replication, keeping in mind the multiple data centers deployment:

TLS certificates generated by the Operator are not suitable and it's required to generate certificates manually on the Main site before creating a database cluster, with all names from replsetOverrides and externalNodes.

- 1. Use TLS cerifficates manual generation instruction to prepare TLS certificates with the host names from your prepared list.
- 2. Deploy your Main site as usual, with these manually generated certificates. Don't forget to turn on Pods exposure on your Main cluster.
- 3. Now override hostname of the first replica in the replica set configuration by using the replsets.replset0verrides subsection in the Custom Resource options manifest with the externally reachable endpoint from your externally reachable URI list:

```
unsafeFlags:
replsetSize: true
replsets:
- name: rs0
size: 1
replsetOverrides:
cluster-name-rs0-0:
host: r1.percona.local:443
...
```

The unsafeFlags.replsetSize option in the above example is needed to create replica set with less than 3 instances.

The actual approach to make the URI reachable from the outside of your Kubernetes culster depends on the exposure type. It is different in case of the NodePort exposure , Load balancer of the cloud provider , etc. Operator won't perform any validation for hostnames. It's user's responsibility to ensure connectivity.

```
You can also add custom tags to the replset members, just to make their identication easier:

...
replsetOverrides:
cluster-name-rs0-0:
host: r1.percona.local:443
tags:
team: cloud
...
```

Configuring Replica sites

To configure Replica sites, you should <u>deploy your Relica sites</u>, repeating the following steps for each Kubernetes cluster you are adding:

- 1. Copy secrets from the Main site, rename them according to the cluster name you use on the Replica site (if needed), and apply.
 - cluster1-ssl (SSL certificates for client connections),
 - cluster1-ssl-internal (SSL certificates for replication),
 - cluster1-secrets (user credentials),
 - cluster1-mongodb-encryption-key (encryption key).
- 2. Deploy the database cluster on the Replica site. Don't forgetting the following:
 - a. All Replica sites must be deployed with the unmanaged: true Custom Resource option. This will stop the Operator in the Replica cluster from touching the MongoDB replset configuration. Starting from this moment, only the Operator of the Main cluster will be able to modify it.
 - b. Backups must be disabled with the backup.enabaled: false Custom Resource option.
 - c. The updateStrategy Custom Resource option must be set to RollingUpdate or OnDelete.
 - d. In order to create a single-instance replica set, you will need to the unsafeFlags.replsetSize option to true as you did on the Main site.
- 3. Now add the new Replica site's Pod to your Main site's externalNodes subsection of the Custom Resource options manifest:

```
replsets:
- name: rs0
size: 1
replsetOverrides:
cluster1-rs0-0:
host: r1.percona.local:443
externalNodes:
- host: r2.percona.local:443
votes: 1
priority: 1
```

8.3.6 Enabling multi-cluster Services

Kubernetes multi-cluster Services (MCS) C 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) [4], and by Amazon Elastic Kubernetes Service (EKS) [5].

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 <u>from Google Kubernetes Engine (GKE)</u> C, or <u>from Amazon Elastic Kubernetes Service (EKS)</u> C.

```
For EKS, you also need to create ClusterProperty objects prior to enabling multi-cluster services.

apiVersion: about.k8s.io/v1alpha1
   kind: ClusterProperty
metadata:
   name: cluster.clusterset.k8s.io
spec:
   value: [Your Cluster identifier]
---
apiVersion: about.k8s.io/v1alpha1
kind: ClusterProperty
metadata:
   name: clusterset.k8s.io
spec:
   value: [Your ClusterSet identifier]

Check AWS MCS controller repository C for more information.
```

Setting up the Operator for MCS results in registering Services for export to other clusters <u>using the ServiceExport object</u> [2], and using ServiceImport one to import external services. Set the following options in the <u>multiCluster</u> subsection of the <u>deploy/cr.yaml</u> configuration file to make it happen:

- multiCluster.enabled should be set to true,
- multiCluster.DNSSuffix string should be equal to the cluster domain suffix for multi-cluster Services used by Kubernetes (svc.clusterset.local by default C).

The following example in the deploy/cr.yaml configuration file is rather straightforward:

```
...
multiCluster:
  enabled: true
  DNSSuffix: svc.clusterset.local
...
```

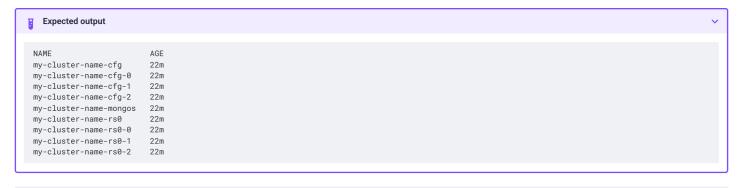
Apply changes as usual with the kubectl apply -f deploy/cr.yaml command.



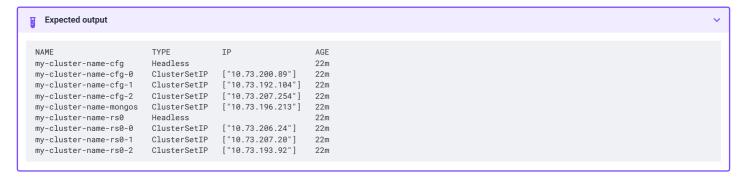
If you want to enable multi-cluster services in a new cluster, we recommended deploying the cluster first with multiCluster.enabled set to false and enable it after replset is initialized. Having MCS enabled from the start is prone to errors on replset initialization.

The initial ServiceExport creation and sync with the clusters of the fleet takes approximately five minutes. You can check the list of services for export and import with the following commands:

```
$ kubectl get serviceexport
```



\$ kubectl get serviceimport





ServiceExport objects are created automatically by the Percona Server for MongoDB Operator. ServiceImport objects, on the other hand, are not controlled by the operator. If you need to troubleshoot ServiceImport objects you must check the MCS controller installed by your cloud provider.

After ServiceExport object is created, exported Services can be resolved from any Pod in any fleet cluster as SERVICE_EXPORT_NAME.NAMESPACE.svc.clusterset.local.

Note

This means that ServiceExports with the same name and namespace will be recognized as a single combined Service.

MCS can charge cross-site replication with additional limitations specific to the cloud provider. For example, GKE demands all participating Pods to be in the same <u>project</u>. Also, default Namespace should be used with caution: your cloud provider <u>may not allow</u> of 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 <u>pausing the cluster</u> and deleting <u>TLS Secrets</u>.

8.4 Monitor database with Percona Monitoring and Management (PMM)

In this section you will learn how to monitor Percona Server for MongoDB with Percona Monitoring and Management (PMM) [4].



Only PMM 2.x versions are supported by the Operator.

PMM is a client/server application. It includes the PMM Server C and the number of PMM Clients C running on each node with the database you wish to monitor.

A PMM Client collects needed metrics and sends gathered data to the PMM Server. As a user, you connect to the PMM Server to see database metrics on a number of dashboards [3].

PMM Server and PMM Client are installed separately.

Install PMM Server

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

Install PMM Client

To install PMM Client as a side-car container in your Kubernetes-based environment, do the following:

Authorize PMM Client within PMM Server.

Token-based authorization (recommended)

- 1. Generate the PMM Server API Key . Specify the Admin role when getting the API Key.
- ▲ Warning: The API key is not rotated automatically.
- 1 Edit the deploy/secrets.yaml 🚰 secrets file and specify the PMM API key for the PMM_SERVER_API_KEY option.
- 2 Apply the configuration for the changes to take effect.

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

Password-based authorization (deprecated since version 1.13.0)

- Edit the <u>deploy/secrets.yaml</u> secrets file and specify the following:
- 2 The user name of your PMM Server (admin by default) in the PMM_SERVER_USER key
- The password you set for the PMM Server during its installation in the PMM_SERVER_PASSWORD key.
- Apply the configuration for the changes to take effect.

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

- 2 Update the pmm section in the deploy/cr.yaml C file:
 - Set pmm.enabled = true.
 - 🕞 Specify your PMM Server hostname / an IP address for the pmm.serverHost option. The PMM Server IP address should be resolvable and reachable from within your cluster.

```
pmm:
   enabled: true
   image: percona/pmm-client:2.44.1
   serverHost: monitoring-service
```

3. Apply the changes:

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

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

```
$ kubectl get pods -n <namespace>
$ kubectl logs <cluster-name>-rs0-0 -c pmm-client -n <namespace>
```

Check the metrics

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

- 1 Log in to PMM server.
- 2 Click MongoDB from the left-hand navigation menu. You land on the Instances Overview page.
- 3 Select your cluster from the Clusters drop-down menu and the desired time range on the top of the page. You should see the metrics.
- 4 Click ♦ MongoDB → Other dashboards to see the list of available dashboards that allow you to drill down to the metrics you are interested in.

Enable profiling

Starting from the Operator version 1.12.0, MongoDB operation profiling is disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analytics Contact disabled by default. To analyze query execution on the PMM Query Analy

For example, update the configuration subsection of the deploy/cr.yaml:

```
spec:
...
replsets:
    - name: rs0
    size: 3
    configuration: |
        operationProfiling:
        slowOpThresholdMs: 200
        mode: slowOp
        rateLimit: 100
```

Optionally, you can specify additional parameters for the pmm-admin add mongodb C command in the pmm.mongodParams and pmm.mongosParams keys for mongod and mongos Pods respectively.

• Info: Please take into account that the Operator automatically manages common MongoDB Service Monitoring parameters C, such as username, password, service-name, host, etc. Assigning values to these parameters is not recommended and can negatively affect the functionality of the PMM setup carried out by the Operator.

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

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

Update the secrets file

The deploy/secrets.yaml file contains all values for each key/value pair in a convenient plain text format. But the resulting Secrets Objects contains passwords stored as base64-encoded strings. If you want to *update* the password field, you need to encode the new password into the base64 format and pass it to the Secrets Object.

To encode a password or any other parameter, run the following command:

on Linux

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

on macOS

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

For example, to set the new PMM API key in the $\mbox{my-cluster-name-secrets}$ object, do the following:

in Linux

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

8.5 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.



Custom sidecar containers can easily access other components of your cluster [2]. Therefore they should be used carefully and by experienced users only.

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

```
Expected output
 Containers:
 rs-sidecar-0:
  Container ID: docker://f0c3437295d0ec819753c581aae174a0b8d062337f80897144eb8148249ba742
   Image ID:
                 docker-pullable://busybox@sha256:139abcf41943b8bcd4bc5c42ee71ddc9402c7ad69ad9e177b0a9bc4541f14924
                 <none>
  Host Port:
                 <none>
  Command:
    /bin/sh
  Args:
    - C
     while true; do echo echo $(date -u) 'test' >> /dev/null; sleep 5; done
               Running
  State:
                  Thu, 11 Nov 2021 10:38:15 +0300
    Started:
   Restart Count: 0
   Environment:
                  <none>
   Mounts:
     /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-fbrbn (ro)
```

Getting shell access to a sidecar container

You can login to your sidecar container as follows:

```
\ kubectl exec -it my-cluster-name-rs0-0 -c rs-sidecar-0 -- sh
/ #
```

Mount volumes into sidecar containers

It is possible to mount volumes into sidecar containers.

Following subsections describe different volume types [4], which were tested with sidecar containers and are known to work. They allow either dynamically provisioning volumes for sidecar containers or mounting existing volumes.

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 <u>secret volume</u> of 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 via the sidecarVolumes subsection 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 1) and mounts it to the rs-sidecar-0 container's filesystem under the /secret directory.



Don't forget you need to <u>create a Secret Object</u> ☐ before you can use it.

configMap

You can use a configMap volume of 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 via the sidecarVolumes subsection 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 [2]) and mounts it to the rs-sidecar-0 container's filesystem under the /config directory.



Don't forget you need to $\underline{\text{create a configMap Object }} \ \underline{\complement}'$ before you can use it.

8.6 Pause/resume Percona Server for MongoDB

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

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

spec:
.....
pause: true

To start the cluster after it was shut down just revert the <code>spec.pause</code> key to false.

9 Troubleshooting

9.1 Initial troubleshooting

Percona Operator for MongoDB uses Custom Resources [2] to manage options for the various components of the cluster.

- PerconaServerMongoDB Custom Resource with Percona Server for MongoDB options (it has handy psmdb shortname also),
- PerconaServerMongoDBBackup and PerconaServerMongoDBRestore Custom Resources contain options for Percona Backup for MongoDB used to backup Percona Server for MongoDB and to restore it from backups (psmdb-backup and psmdb-restore shortnames are available for them).

The first thing you can check for the Custom Resource is to query it with kubectl get command:

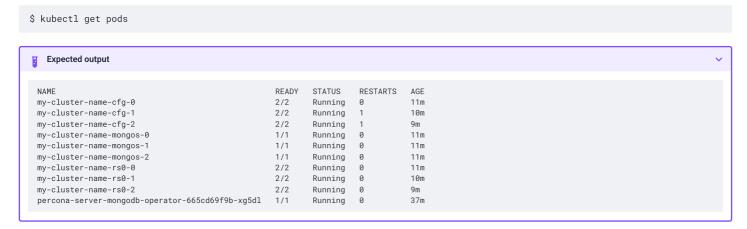


The Custom Resource should have Ready status.



Check the Pods

If Custom Resource is not getting Ready status, it makes sense to check individual Pods. You can do it as follows:



The above command provides the following insights:

- READY indicates how many containers in the Pod are ready to serve the traffic. In the above example, my-cluster-name-rs0-0 Pod has all two containers ready (2/2). For an application to work properly, all containers of the Pod should be ready.
- STATUS indicates the current status of the Pod. The Pod should be in a Running state to confirm that the application is working as expected. You can find out other possible states in the official Kubernetes documentation .
- RESTARTS indicates how many times containers of Pod were restarted. This is impacted by the Container Restart Policy. C. In an ideal world, the restart count would be zero, meaning no issues from the beginning. If the restart count exceeds zero, it may be reasonable to check why it happens.

• AGE: Indicates how long the Pod is running. Any abnormality in this value needs to be checked.

You can find more details about a specific Pod using the kubectl describe pods <pod-name> command.

\$ kubectl describe pods my-cluster-name-rs0-0

```
Expected output
 Name:
                my-cluster-name-rs0-0
 Namespace:
                default
 Controlled By: StatefulSet/my-cluster-name-rs0
 Init Containers:
  mongo-init:
 Containers:
  mongod:
     Restart Count: 0
    Limits:
       cpu:
       memory: 500M
     Requests:
      cpu:
                  300m
       memory:
                 500M
 Liveness: exec [/opt/percona/mongodb-healthcheck k8s liveness --ssl --sslInsecure --sslCAFile /etc/mongodb-ssl/ca.crt --sslPEMKeyFile /tmp/tls.pem --startupDelaySeconds 7200] delay=60s timeout=10s period=30s #success=1 #failure=4
    Readiness: tcp-socket:27017 delay=10s timeout=2s period=3s #success=1 #failure=8 Environment Variables from:
       internal-my-cluster-name-users Secret Optional: false
    Environment:
    Mounts:
 Volumes:
 Events:
                                   <none>
```

This gives a lot of information about containers, resources, container status and also events. So, describe output should be checked to see any abnormalities.

9.2 Exec into the containers

If you want to examine the contents of a container "in place" using remote access to it, you can use the kubectlexec command. It allows you to run any command or just open an interactive shell session in the container. Of course, you can have shell access to the container only if container supports it and has a "Running" state.

In the following examples we will access the container mongod of the my-cluster-name-rs0-0 Pod.

· Run date command:

```
$ kubectl exec -ti my-cluster-name-rs0-0 -c mongod -- date

Expected output

Thu Nov 24 10:01:17 UTC 2022
```

You will see an error if the command is not present in a container. For example, trying to run the time command, which is not present in the container, by executing kubectl exec -ti my-cluster-name-rs0-0 -c mongod -- time would show the following result:

OCI runtime exec failed: exec failed: unable to start container process: exec: "time": executable file not found in \$PATH: unknown command terminated with exit code 126

• Print /var/log/mongo/mongod.log file to a terminal:

```
$ kubectl exec -ti my-cluster-name-rs0-0 -c mongod -- cat /var/log/mongo/mongod.log
```

· Similarly, opening an Interactive terminal, executing a pair of commands in the container, and exiting it may look as follows:

```
$ kubectl exec -ti my-cluster-name-rs0-0 -c mongod -- bash
[mongodb@my-cluster-name-rs0-0 db]$ cat /etc/hostname
my-cluster-name-rs0-0
[mongodb@my-cluster-name-rs0-0 db]$ ls /var/log/mongo/mongod.log
/var/log/mongo/mongod.log
[mongodb@my-cluster-name-rs0-0 db]$ exit
exit
$
```

Avoid the restart-on-fail loop for Percona Server for MongoDB containers

The restart-on-fail loop takes place when the container entry point fails (e.g. mongod crashes). In such a situation, Pod is continuously restarting. Continuous restarts prevent to get console access to the container, and so a special approach is needed to make fixes.

You can prevent such infinite boot loop by putting the Percona Server for MongoDB containers into the "infinite sleep" without starting mongod. This behavior of the container entry point is triggered by the presence of the /data/db/sleep-forever file. The feature is available for both replica set and confg server Pods.

For example, you can do it for the mongod container of an appropriate Percona Server for MongoDB Pod as follows:

```
$ kubectl exec -it my-cluster-name-cfg-0 -c mongod -- sh -c 'touch /data/db/sleep-forever'
```

If mongod container can't start, you can use backup-agent container instead:

```
$ kubectl exec -it my-cluster-name-cfg-0 -c backup-agent -- sh -c 'touch /data/db/sleep-forever'
```

The instance will restart automatically and run in its usual way as soon as you remove this file (you can do it with a command similar to the one you have used to create the file, just substitute touch to rm in it).

9.3 Check the Logs

Logs provide valuable information. It makes sense to check the logs of the database Pods and the Operator Pod. Following flags are helpful for checking the logs with the kubectl logs command:

Flag	Description
container= <container-name></container-name>	Print log of a specific container in case of multiple containers in a Pod
follow	Follows the logs for a live output
since= <time></time>	Print logs newer than the specified time, for example:since="10s"
timestamps	Print timestamp in the logs (timezone is taken from the container)
previous	Print previous instantiation of a container. This is extremely useful in case of container restart, where there is a need to check the logs on why the container restarted. Logs of previous instantiation might not be available in all the cases.

In the following examples we will access containers of the my-cluster-name-rs0-0 Pod.

• Check logs of the mongod container:

```
$ kubectl logs my-cluster-name-rs0-0 -c mongod
```

• Check logs of the pmm-client container:

```
$ kubectl logs my-cluster-name-rs0-0 -c pmm-client
```

• Filter logs of the mongod container which are not older than 600 seconds:

```
$ kubectl logs my-cluster-name-rs0-0 -c mongod --since=600s
```

• Check logs of a previous instantiation of the mongod container, if any:

```
$ kubectl logs my-cluster-name-rs0-0 -c mongod --previous
```

• Check logs of the mongod container, parsing the output with jq JSON processor ☐:

```
$ kubectl logs my-cluster-name-rs0-0 -c mongod -f | jq -R 'fromjson?'
```

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.yml file, which sets the following environment variables with self-speaking names and values:

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

9.4 Special debug images

For the cases when Pods are failing for some reason or just show abnormal behavior, the Operator can be used with a special debug image of the Percona Server for MongoDB, which has the following specifics:

- · it avoids restarting on fail,
- it contains additional tools useful for debugging (sudo, telnet, gdb, mongodb-debuginfo package, etc.),
- extra verbosity is added to the mongodb daemon.

Images are available for Percona server for MongoDB versions 5.0 and 6.0, not for 7.0.

Particularly, using this image is useful if the container entry point fails (mongod crashes). In such a situation, Pod is continuously restarting. Continuous restarts prevent to get console access to the container, and so a special approach is needed to make fixes.

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

percona/percona-server-mongodb:6.0.21-18-debug

The Pod should be restarted to get the new image.



When the Pod is continuously restarting, you may have to delete it to apply image changes.

10 HOWTOs

10.1 Install Percona Server for MongoDB with customized parameters

You can customize the configuration of Percona Server for MongoDB and install it with customized parameters.

To check available configuration options, see deploy/cr.yaml C and Custom Resource Options.

kubect1

To customize the configuration, do the following:

1. Clone the repository with all manifests and source code by executing the following command:

```
$ git clone -b v1.20.0 https://github.com/percona/percona-server-mongodb-operator
```

2. Edit the required options and apply the modified deploy/cr.yaml file as follows:

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

Helm

To install Percona Server for MongoDB with custom parameters, use the following command:

```
$ helm install --set key=value
```

You can pass any of the Operator's Custom Resource options as a --set key=value[,key=value] argument.

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

Command line

```
$ helm install my-db percona/psmdb-db --version 1.20.0 --namespace psmdb \
    --set "replsets.rs0.name=rs0" --set "replsets.rs0.size=3" \
    --set "replsets.rs0.volumeSpec.pvc.resources.requests.storage=20Gi" \
    --set backup.enabled=false --set sharding.enabled=false
```

YAML file

You can specify customized options in a YAML file instead of using separate command line parameters. The resulting file similar to the following example looks as follows:

Apply the resulting YAML file as follows:

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

Configure ports for MongoDB cluster components

By default, the Operator starts Percona Server for MongoDB with the default port 27017 for all cluster components: mongod, mongos and configsvrReplSet Pods. Starting with version 1.20.0, you can start a new cluster with custom ports for all components or for a specific one.

Here's how to do it.

kubectl

1. Edit the deploy/cr.yaml file and specify the following configuration:

```
spec:
...
replsets:
    - name: rs0
    configuration: |
    net:
        port: 27018
sharding:
configsvrReplSet:
    configuration: |
    net:
        port: 27019
mongos:
    configuration: |
    net:
        port: 27017
```

2. Apply the deploy/cr.yaml to deploy Percona Server for MongoDB:

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

Helm

1. Create a yaml file with the desired configuration. For example, values.yaml:

2. Install Percona Server for MongoDB with the specified configuration:

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

10.2 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 The files used to configure it, and find how to install and configure OpenLDAP in the official OpenLDAP The and OpenLDAP The files used to configure it, and find how to install and configure OpenLDAP The official OpenLDAP The open the official OpenLDAP The open the open that the open the open that t

The OpenLDAP side

You can add needed OpenLDAP settings will the following LDIF C 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: {crvpt}x
  shadowLastChange: -1
  shadowMax: -1
  shadowWarning: -1
2-group-cn.ldif: |-
  dn: cn=admin,ou=perconadba,dc=ldap,dc=local
  cn: admin
  objectClass: groupOfUniqueNames
  objectClass: top
  ou: perconadba
  uniqueMember: uid=percona, ou=perconadba, dc=ldap, dc=local
```

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

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



If you are not sure about the approach to make references between user and group objects, OpenDAP overlays C provide one of the possible ways to go.

The MongoDB and Operator side

The following steps will look different depending on whether sharding is on (the default behavior) or off.

If sharding is off

In order to get MongoDB connected with OpenLDAP in case of a a non-sharded (ReplicaSet) MongoDB cluster we need to configure two things:

- Mongod
- · Internal mongodb role

Create configuration Secrets for mongod:

my_mongod.conf security: authorization: "enabled" ldap: queryTemplate: '{USER}?memberOf?base' servers: "openldap" transportSecurity: none bind: queryUser: "cn=readonly,dc=ldap,dc=local" queryPassword: "password" userToDNMapping: '[match : "(.+)", ldapQuery: "OU=perconadba,DC=ldap,DC=local??sub?(uid={0})" setParameter: authenticationMechanisms: 'PLAIN, SCRAM-SHA-1'

Note

This fragment provides mongod with LDAP-specific parameters, such as FQDN of the LDAP server (server), explicit lookup user, domain rules, etc.

Put the snippet on you local machine and create a Kubernetes Secret object named based on your MongoDB cluster name:

```
$ kubectl create secret generic <your_cluster_name>-rs0-mongod --from-file=mongod.conf=my_mongod.conf
```

Next step is to start the MongoDB cluster up as it's described in <u>Install Percona server for MongoDB on Kubernetes</u>. On successful completion of the steps from this doc, we are to proceed with setting the roles for the 'external' (managed by LDAP) user inside the MongoDB. For this, log into MongoDB as administrator:

```
$ mongo "mongodb+srv://userAdmin:<userAdmin_password>@<your_cluster_name>-rs0.<your_namespace>.svc.cluster.local/admin?
replicaSet=rs0&ssl=false"
```

When logged in, execute the following:

```
mongos> db.getSiblingDB("admin").createRole(
{
 role: "cn=admin,ou=perconadba,dc=ldap,dc=local",
 privileges: [],
 roles : [
  {
   "role" : "readAnyDatabase",
     "db" : "admin"
   },
     "role" : "dbAdminAnyDatabase",
     "db" : "admin"
     "role" : "clusterMonitor",
"db" : "admin"
     "role" : "readWriteAnyDatabase",
"db" : "admin"
     "role" : "restore",
     "db" : "admin"
     "role" : "backup",
     "db" : "admin"
```

Note

Extra roles listed in the above example are just to show more than one possible variant.

Now the new percona user created inside OpenLDAP is able to login to MongoDB as administrator. Verify whether the user role has been identified correctly with the following command:

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

When logged in, execute the following:

```
mongos> db.runCommand({connectionStatus:1})
```

The output should be like follows:

```
"authInfo" : {
   "authenticatedUsers" : [
       "user" : "percona",
       "db" : "$external"
   "authenticatedUserRoles" : [
       "role" : "restore",
       "db" : "admin"
       "role" : "readAnyDatabase",
       "db" : "admin"
       "role" : "clusterMonitor",
"db" : "admin"
       "role" : "dbAdminAnyDatabase",
       "db" : "admin"
       "role" : "backup",
       "db" : "admin"
     }.
       "role" : "cn=admin,ou=perconadba,dc=ldap,dc=local",
       "db" : "admin"
       "role" : "readWriteAnyDatabase",
       "db" : "admin"
   1
 },
 "ok" : 1,
 "$clusterTime" : {
   "clusterTime" : Timestamp(1663067287, 4),
   "signature" : {
     "hash" : BinData(0, "ZaLGSVj4ZwZrngXZSOqXB5rx+oo="),
     "keyId" : NumberLong("7142816031004688408")
   }
 "operationTime" : Timestamp(1663067287, 4)
mongos>
```

If sharding is on

In order to get MongoDB connected with OpenLDAP in this case we need to configure three things:

- Mongod
- Internal mongodb role

Both the routing interface (mongos) and the configuraion ReplicaSet (mongod) have to be configured to make the LDAP server a part of the Authentication/Authorization chain.



mongos is just a router between shards and underlying database instances, and configuration ReplicaSet is responsible for keeping information about database users and roles. Thus, the router can perform only authentication, while authorization is the responsibility of the configuration ReplicaSet.

Create configuration Secrets for the router and the configuration ReplicaSet respectively.

Secret for the router should look as follows:

Put the snippet on you local machine and create a Kubernetes Secret object named based on your MongoDB cluster name:

```
$ kubectl create secret generic <your_cluster_name>-mongos --from-file=mongos.conf=my_mongos.conf
```

Secret for the configuration ReplicaSet should look as follows:

```
my_mongod.conf
security:
  authorization: "enabled"
  ldap:
   authz:
     queryTemplate: '{USER}?memberOf?base'
    servers: "openldap"
    transportSecurity: none
    bind:
      queryUser: "cn=readonly,dc=ldap,dc=local"
      queryPassword: "password"
    userToDNMapping:
      '[
            match : "(.+)",
            ldapQuery: "OU=perconadba, DC=ldap, DC=local??sub?(uid={0})"
    ]'
setParameter:
  authenticationMechanisms: 'PLAIN, SCRAM-SHA-1'
```

Put the snippet on you local machine and create a Kubernetes Secret object named based on your MongoDB cluster name:

```
$ kubectl create secret generic <your_cluster_name>-cfg-mongod --from-file=mongod.conf=my_mongod.conf
```

Both files are pretty much the same except the authz subsection, which is only present for the configuration ReplicaSet.

Next step is to start the MongoDB cluster up as it's described in <u>Install Percona server for MongoDB on Kubernetes</u>. On successful completion of the steps from this doc, we are to proceed with setting the roles for the 'external' (managed by LDAP) user inside the MongoDB. For this, log into MongoDB as administrator:

```
$ mongo "mongodb://userAdmin:<userAdmin_password>@<your_cluster_name>-mongos.<your_namespace>.svc.cluster.local/admin?
ssl=false"
```

When logged in, execute the following:

```
mongos> db.getSiblingDB("admin").createRole(
{
role: "cn=admin,ou=perconadba,dc=ldap,dc=local",
privileges: [],
roles : [
  {
  "role" : "readAnyDatabase",
     "db" : "admin"
   },
     "role" : "dbAdminAnyDatabase",
     "db" : "admin"
     "role" : "clusterMonitor",
     "db" : "admin"
     "role" : "readWriteAnyDatabase",
"db" : "admin"
     "role" : "restore",
     "db" : "admin"
     "role" : "backup",
     "db" : "admin"
```

Note

Extra roles listed in the above example are just to show more than one possible variant.

Now the new percona user created inside OpenLDAP is able to login to MongoDB as administrator. Verify whether the user role has been identified correctly with the following command:

```
$ mongo --username percona --password 'percona' --authenticationMechanism 'PLAIN' --authenticationDatabase '$external' --host <your_cluster_name>-mongos --port 27017
```

When logged in, execute the following:

```
mongos> db.runCommand({connectionStatus:1})
```

The output should be like follows:

```
"authInfo" : {
   "authenticatedUsers" : [
       "user" : "percona",
       "db" : "$external"
   "authenticatedUserRoles" : [
       "role" : "restore",
       "db" : "admin"
       "role" : "readAnyDatabase",
       "db" : "admin"
       "role" : "clusterMonitor",
       "db" : "admin"
       "role" : "dbAdminAnyDatabase",
       "db" : "admin"
       "role" : "backup",
       "db" : "admin"
     }.
       "role" : "cn=admin,ou=perconadba,dc=ldap,dc=local",
       "db" : "admin"
       "role" : "readWriteAnyDatabase",
       "db" : "admin"
  1
 },
 "ok" : 1,
 "$clusterTime" : {
   "clusterTime" : Timestamp(1663067287, 4),
   "signature" : {
     "hash" : BinData(0, "ZaLGSVj4ZwZrngXZSOqXB5rx+oo="),
     "keyId" : NumberLong("7142816031004688408")
  }
 "operationTime" : Timestamp(1663067287, 4)
mongos>
```

Using LDAP over TLS connection

LDAP over TLS 🖸 allows you to use Transport Layer Security, encrypting your communication between MongoDB and OpenLDAP server.

Here are the needed modifications to <u>The MongoDB and Operator side</u> subsection which will enable it:

1. First, create a secret that contains the SSL certificate to connect to LDAP. The following example creates it from the file with CA certificate (the one you use in /etc/openldap/ldap.conf), naming the new secret my-ldap-secret:

```
$ kubectl create secret generic my-ldap-secret --from-file=ca.crt=ldap-ca.pem
```

2. Set the secrets.ldapSecret Custom Resource option to the name of your newly created secret. Your modified deploy/cr.yaml may look as follows:

```
...
secrets:
...
ldapSecret: my-ldap-secret
```

3. It is also necessary to change the value of transportSecurity to tls in mongod and mongos configurations. The configuration is similar to one described at the
The MongoDB and Operator side">MongoDB and Operator side subsection:

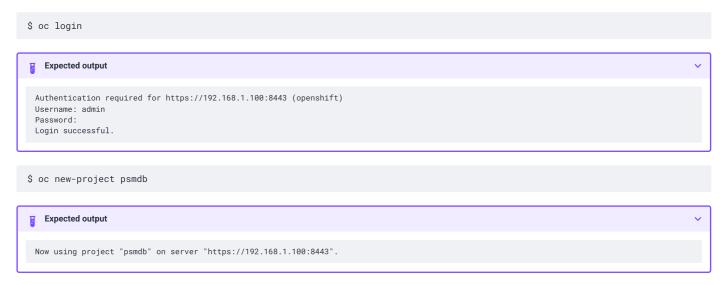
Changed mongod configuration should look as follows:

```
``` yaml title="my_mongod.conf" hl_lines="7"
security:
 authorization: "enabled"
 ldap:
 authz:
 queryTemplate: '{USER}?memberOf?base'
 servers: "openldap"
 transportSecurity: tls
 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'
If **sharding is on**, you will also need to change mongos configuration:
```yaml title="my_mongos.conf" hl_lines="4"
security:
 ldap:
    servers: "openldap"
    transportSecurity: tls
   bind:
      queryUser: "cn=readonly,dc=ldap,dc=local"
      queryPassword: "password"
    userToDNMapping:
      '[
            match : "(.+)",
            ldapQuery: "OU=perconadba, DC=ldap, DC=local??sub?(uid={0})"
    1'
setParameter:
 authenticationMechanisms: 'PLAIN, SCRAM-SHA-1'
```

10.3 Use Docker images from a custom registry

Using images from a private Docker registry may required for privacy, security or other reasons. In these cases, Percona Operator for MongoDB allows the use of a custom registry This following example of the Operator deployed in the OpenShift environment demonstrates the process:

1. Log into the OpenShift and create a project.



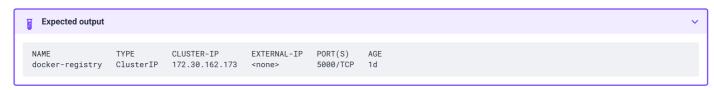
- 2. You need obtain the following objects to configure your custom registry access:
 - · A user token
 - the registry IP address

\$ oc whoami -t

You can view the token with the following command:

Expected output ADO8CqCDappWR4hxjfDqwijEHei31yXAvWg61Jg210s The following command returns the registry IP address:

\$ kubectl get services/docker-registry -n default



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

\$ docker login -u admin -p ADO8CqCDappWR4hxjfDqwijEHei31yXAvWg61Jg210s 172.30.162.173:5000 Expected output Login Succeeded

4. Use the Docker commands to pull the needed image by its SHA digest:

 $\label{locker} $$ docker pull 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:7.0.18-11
$ docker push 172.30.162.173:5000/psmdb/percona-server-mongodb:7.0.18-11
```

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

\$ oc get is



7. When the custom registry image is available, edit the the image: option in deploy/operator.yaml configuration file with a Docker Repo + Tag string (it should look like docker-registry.default.svc:5000/psmdb/percona-server-mongodb:7.0.18-11)



If the registry requires authentication, you can specify the <code>imagePullSecrets</code> option for all images.

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



Don't forget to set <u>upgradeoptions.apply</u> option to <u>Disabled</u>. Otherwise <u>Smart Upgrade functionality</u> will try using the image recommended by the Version Service instead of the custom one.

9. Now follow the standard Percona Operator for MongoDB $\underline{installation\ instruction}.$

10.4 Creating a private \$3-compatible cloud for backups

As it is mentioned in <u>backups</u>, 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 <u>Minio</u> - the S3-compatible object storage server deployed via Docker on your own infrastructure.

Setting up Minio to be used with Percona Operator for MongoDB backups involves the following steps:

1. Install Minio in your Kubernetes or OpenShift environment and create the correspondent Kubernetes Service as follows:

```
$ helm install \
    --name minio-service \
    --version 8.0.5 \
    --set accessKey=some-access-key \
    --set secretKey=some-secret-key \
    --set service.type=ClusterIP \
    --set configPath=/tmp/.minio/ \
    --set persistence.size=2G \
    --set environment.MINIO_REGION=us-east-1 \
    stable/minio
```

Don't forget to substitute default some-access-key and some-secret-key strings in this command with actual unique key values. The values can be used later for access control. The storageClass option is needed if you are using the special <u>Kubernetes Storage Class</u> of for backups. Otherwise, this setting may be omitted. You may also notice the <u>MINIO_REGION</u> 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 <u>deploy/cr.yaml</u> T 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 <u>Kubernetes secret</u> of probackups. Sample <u>backup-s3.yaml</u> 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 If file). Also substitute <storage> with the actual storage name located in a subsection inside of the backups in the deploy/cr.yaml If 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
```

10.5 How to restore backup to a new Kubernetes-based environment

The Operator allows restoring a backup not only on the Kubernetes cluster where it was made, but also on any Kubernetes-based environment with the installed Operator.

When restoring to a new Kubernetes-based environment, make sure it has a Secrets object with the same user passwords as in the original cluster. If restoring a physical backup, the encryption key of the target cluster needs to be set accordingly as well. More details about secrets can be found in <u>System Users</u>. The name of the required Secrets object can be found out from the spec.secrets key in the deploy/cr.yaml (my-cluster-name-secrets by default).

You will need correct names for the backup and the cluster. If you have access to the original cluster, available backups can be listed with the following command:

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

And the following command will list available clusters:

\$ kubectl get psmdb



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

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

Without point-in-time recovery

- 1. Set appropriate keys in the deploy/backup/restore.yaml deploy
 - set spec.clusterName key to the name of the target cluster to restore the backup on,
 - set spec.backupSource subsection to point on the appropriate cloud storage. This backupSource subsection should contain the backup type (either logical or physical), and a destination key, followed by necessary storage configuration keys, same as in the deploy/cr.yaml file:

```
backupSource:
  type: logical
  destination: s3://S3-BUCKET-NAME/BACKUP-NAME
    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 kubect1 get psmdb-backup command). For Azure Blob storage, you don't put the prefix, and use your container name as an equivalent of a bucket.

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

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

2. After that, the actual restoration process can be started as follows:

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

With point-in-time recovery

- 1. Set appropriate keys in the deploy/backup/restore.yaml deploy
 - set spec.clusterName key to the name of the target cluster to restore the backup on
 - put additional restoration parameters to the pitr section:
 - type key can be equal to one of the following options
 - date roll back to specific date
 - latest recover to the latest possible transaction
 - date key is used with type=date option and contains value in datetime format
 - set spec.backupSource subsection to point on the appropriate cloud storage. For S3-compatible storage this backupSource subsection should contain a destination key equal to the s3 bucket with a special s3:// prefix, followed by necessary S3 configuration keys, same as in deploy/cr.yaml file:

```
apiVersion: psmdb.percona.com/v1
kind: PerconaServerMongoDBRestore
metadata:
    name: restore1
spec:
    clusterName: my-cluster-name
pitr:
    type: date
    date: YYYY-MM-DD hh:mm:ss
backupSource:
    destination: s3://S3-BUCKET-NAME/BACKUP-NAME
    s3:
        credentialsSecret: my-cluster-name-backup-s3
        region: us-west-2
        endpointUrl: https://URL-OF-THE-S3-COMPATIBLE-STORAGE
```

• 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
```

10.6 How to use backups to move the external database to Kubernetes

The Operator allows restoring a backup not only on the Kubernetes cluster where it was made, but also on any Kubernetes-based environment with the installed Operator, and the backup/restore tool actually used by the Operator is the Percona Backup for MongoDB. C. That makes it possible to move external MongoDB Cluster to Kubernetes with Percona Backup for MongoDB.



There are other scenarios for migrating MongoDB database to Kubernetes as well. For example, this blogpost C covers migration based on the regular MongoDB replication capabilities.

Backups can be stored either locally, or remotely (on Amazon S3 or S3-compatible storage [2]), or on Azure Blob Storage [2]). S3-compatible storage to be used for backups.

- 1. Make sure the following prerequisite requirements are satisfied within your setup:
 - Percona Backup for MongoDB packages are installed on the replica set nodes of the source cluster following the official installation instructions [4], and the authentication of the pbm-agent is configured of to allow it accessing your database.
 - The Operator and the destination cluster should be installed in the Kuberentes-based environment. For simplicity, it's reasonable to have the same topology of the source and destination clusters, although Percona Backup for MongoDB allows replset-remapping. C as well.
- 2. Configure the cloud storage for backups on your source cluster following the official guide [4]. For example, using the Amazon S3 storage can be configured with the following YAML file:

```
pbm_config.yaml
type: s3
 region: us-west-2
  bucket: pbm-test-bucket
  credentials:
    access-key-id: <your-access-key-id-here>
    secret-access-key: <your-secret-key-here>
```

After putting all needed details into the file (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, the S3 bucket and region in the above example), provide the config file to the pbm-agent on all nodes as follows:

```
$ pbm config --file pbm_config.yaml
```

3. Start the pbm-agent:

```
$ sudo systemctl start pbm-agent
```

4. Now you can make backup as follows:

```
$ pbm backup --wait
```

The command output will contain the backup name, which you will further use to restore the backup:

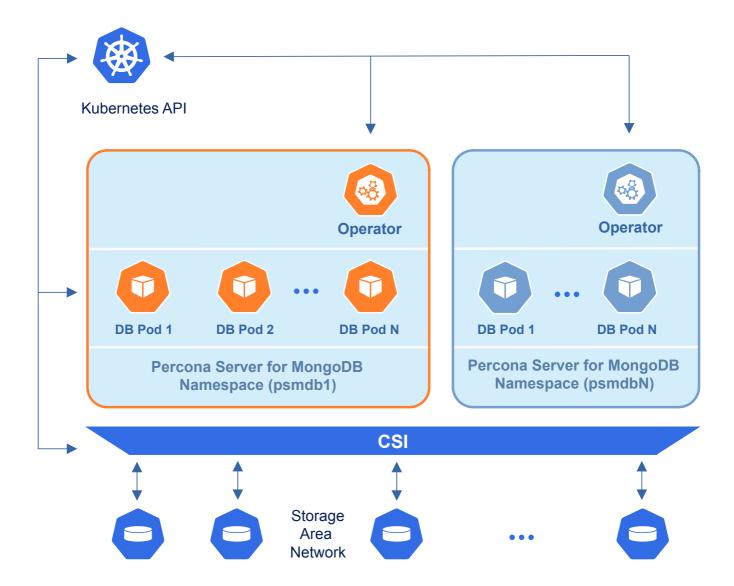
```
Starting backup '2022-06-15T08:18:44Z'....
Waiting for '2022-06-15T08:18:44Z' backup..... done
pbm-conf> pbm status -s backups
Backups:
FS /data/pbm
  Snapshots:
   2022-06-15T08:18:44Z 28.23KB <logical> [complete: 2022-06-15T08:18:49Z]
```

5. The rest of operations will be carried out on your destination cluster in a Kubernetes-based environment of your choice. These actions are described in the How to restore backup to a new Kubernetes-based environment guide. Just use the proper name of the backup (2022-06-15T08:18:44Z) in the above example, and proper parameters specific to your cloud storage (e.g. the pbm-test-bucket bucket name we used above).

10.7 Install Percona Operator for MongoDB in multi-namespace (cluster-wide) mode

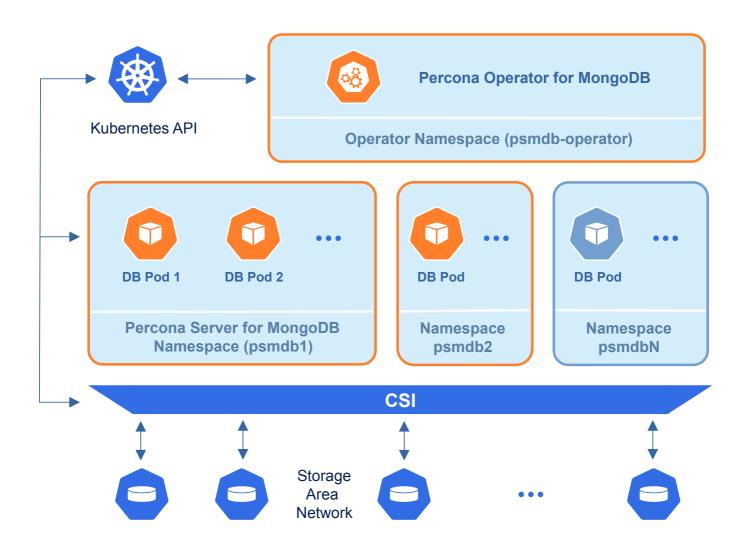
Difference between single-namespace and multi-namespace Operator deployment

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:





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.

Installing the Operator in cluster-wide mode

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 v1.20.0 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:

4. Apply ☐ the deploy/cw-bundle.yaml file with the following command:

```
$ kubectl apply -f deploy/cw-bundle.yaml --server-side -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 by quering the PerconaServerMongoDB Custom Resource (it has handy psmdb shortname also) with the following command:

```
$ kubectl get psmdb -n psmdb
```



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 'aDAzQ@pCY3NSWEZ2ZUIzS1I=' | 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 perconaclient:

```
\ kubectl run -i --rm --tty percona-client --image=percona/percona-server-mongodb:6.0.21-18 --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

```
\$ \ mongosh \ "mongodb://databaseAdmin:databaseAdminPassword@my-cluster-name-mongos.psmdb.svc.cluster.local/admin?ssl=false" \ and the substitution of the substitut
```

if sharding is off

 $\label{lem:password@my-cluster-name-rs0.psmdb.svc.cluster.local/admin? replicaSet=rs0\&ssl=false" \\$

10.8 Monitor Kubernetes

Monitoring the state of the database is crucial to timely identify and react to performance issues. <u>Percona Monitoring and Management (PMM) solution enables</u> you to do just that.

However, the database state also depends on the state of the Kubernetes cluster itself. Hence it's important to have metrics that can depict the state of the Kubernetes cluster

This document describes how to set up monitoring of the Kubernetes cluster health. This setup has been tested with the PMM server C as the centralized data storage and the Victoria Metrics Kubernetes monitoring stack as the metrics collector. These steps may also apply if you use another Prometheus-compatible storage.

Pre-requisites

To set up monitoring of Kubernetes, you need the following:

- 1. PMM Server up and running. You can run PMM Server as a Docker image, a virtual appliance, or on an AWS instance. Please refer to the official PMM documentation. [2] for the installation instructions.
- 2. <u>Helm v3</u> 4.
- 3. kubectl

 ☐.
- 4. The PMM Server API key. The key must have the role "Admin".

Get the PMM API key:

From PMM UI

Generate the PMM API key

∑ From command line

You can query your PMM Server installation for the API Key using curl and jq utilities. Replace <login>:<password>@<server_host> placeholders with your real PMM Server login, password, and hostname in the following command:

```
$ API_KEY=$(curl --insecure -X POST -H "Content-Type: application/json" -d {"name":"operator", "role": "Admin"}'
"https://<login>:<password>@<server_host>/graph/api/auth/keys" | jq .key)
```



The API key is not rotated.

Install the Victoria Metrics Kubernetes monitoring stack

Quick install

- 1. To install the Victoria Metrics Kubernetes monitoring stack with the default parameters, use the quick install command. Replace the following placeholders with your values:
 - API-KEY The API key of your PMM Server
 - PMM-SERVER-URL The URL to access the PMM Server
 - UNIQUE-K8s-CLUSTER-IDENTIFIER Identifier for the Kubernetes cluster. It can be the name you defined during the cluster creation.

You should use a unique identifier for each Kubernetes cluster. The use of the same identifier for more than one Kubernetes cluster will result in the conflicts during the metrics collection.

NAMESPACE - The namespace where the Victoria metrics Kubernetes stack will be installed. If you haven't created the namespace before, it will be created
during the command execution.

We recommend to use a separate namespace like $\,$ monitoring-system .

\$ curl -fsL https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/refs/tags/v0.1.1/vm-operator-k8s-stack/quick-install.sh | bash -s -- --api-key <API-KEY> --pmm-server-url <PMM-SERVER-URL> --k8s-cluster-id <UNIQUE-K8s-CLUSTER-IDENTIFIER> --namespace <NAMESPACE>



The Prometheus node exporter is not installed by default since it requires privileged containers with the access to the host file system. If you need the metrics for Nodes, add the --node-exporter-enabled flag as follows:

\$ curl -fsL https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/refs/tags/v0.1.1/vm-operator-k8s-stack/quick-install.sh | bash -s -- --api-key <API-KEY> --pmm-server-url <PMM-SERVER-URL> --k8s-cluster-id <UNIQUE-K8s-CLUSTER-IDENTIFIER> --namespace <NAMESPACE> --node-exporter-enabled

Install manually

You may need to customize the default parameters of the Victoria metrics Kubernetes stack.

- Since we use the PMM Server for monitoring, there is no need to store the data in Victoria Metrics Operator. Therefore, the Victoria Metrics Helm chart is installed with the vmsingle.enabled and vmcluster.enabled parameters set to false in this setup.
- Check all the role-based access control (RBAC) rules of the victoria-metrics-k8s-stack chart and the dependencies chart, and modify them based on your requirements.

Configure authentication in PMM

To access the PMM Server resources and perform actions on the server, configure authentication.

1. Encode the PMM Server API key with base64.

A Linux

```
$ echo -n <API-key> | base64 --wrap=0
```

macOS

```
$ echo -n <API-key> | base64
```

2. Create the Namespace where you want to set up monitoring. The following command creates the Namespace monitoring-system. You can specify a different name. In the latter steps, specify your namespace instead of the <namespace> placeholder.

```
$ kubectl create namespace monitoring-system
```

3. Create the YAML file for the Kubernetes Secrets 🖸 and specify the base64-encoded API key value within. Let's name this file pmm-api-vmoperator.yaml.

pmm-api-vmoperator.yaml

```
apiVersion: v1
data:
   api_key: <base-64-encoded-API-key>
kind: Secret
metadata:
   name: pmm-token-vmoperator
   #namespace: default
type: Opaque
```

4. Create the Secrets object using the YAML file you created previously. Replace the <filename> placeholder with your value.

```
$ kubectl apply -f pmm-api-vmoperator.yaml -n <namespace>
```

5. Check that the secret is created. The following command checks the secret for the resource named pmm-token-vmoperator (as defined in the metadata. name option in the secrets file). If you defined another resource name, specify your value.

```
$ kubectl get secret pmm-token-vmoperator -n <namespace>
```

Create a ConfigMap to mount for kube-state-metrics

The <u>kube-state-metrics (KSM)</u> L⁷ is a simple service that listens to the Kubernetes API server and generates metrics about the state of various objects - Pods, Deployments, Services and Custom Resources.

To define what metrics the kube-state-metrics should capture, create the ConfigMap of and mount it to a container.

Use the <u>example configmap.yaml configuration file</u> ☐ to create the ConfigMap.

```
$ kubectl apply -f https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/refs/tags/v0.1.1/vm-operator-k8s-stack/ksm-configmap.yaml -n <namespace>
```

As a result, you have the customresource-config-ksm ConfigMap created.

Install the Victoria Metrics Kubernetes monitoring stack

1. Add the dependency repositories of victoria-metrics-k8s-stack ☐ chart.

```
$ helm repo add grafana https://grafana.github.io/helm-charts
$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

2. Add the Victoria Metrics Kubernetes monitoring stack repository.

```
$ helm repo add vm https://victoriametrics.github.io/helm-charts/
```

3. Update the repositories.

```
$ helm repo update
```

- 4. Install the Victoria Metrics Kubernetes monitoring stack Helm chart. You need to specify the following configuration:
 - the URL to access the PMM server in the externalVM.write.url option in the format <PMM-SERVER-URL>/victoriametrics/api/v1/write. The URL can contain either the IP address or the hostname of the PMM server.
 - the unique name or an ID of the Kubernetes cluster in the vmagent.spec.externalLabels.k8s_cluster_id option. Ensure to set different values if you are sending metrics from multiple Kubernetes clusters to the same PMM Server.
 - the <namespace> placeholder with your value. The Namespace must be the same as the Namespace for the Secret and ConfigMap.

```
$ helm install vm-k8s vm/victoria-metrics-k8s-stack \
-f https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/refs/tags/v0.1.1/vm-operator-k8s-stack/values.yaml \
--set externalVM.write.url=<PMM-SERVER-URL>/victoriametrics/api/v1/write \
--set vmagent.spec.externalLabels.k8s_cluster_id=<UNIQUE-CLUSTER-IDENTIFER/NAME> \
-n <namespace>
```

To illustrate, say your PMM Server URL is https://pmm-example.com, the cluster ID is test-cluster and the Namespace is monitoring-system. Then the command would look like this:

```
$ helm install vm-k8s vm/victoria-metrics-k8s-stack \
-f https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/refs/tags/v0.1.1/vm-operator-k8s-stack/values.yaml \
--set externalVM.write.url=https://pmm-example.com/victoriametrics/api/v1/write \
--set vmagent.spec.externalLabels.k8s_cluster_id=test-cluster \
-n monitoring-system
```

Validate the successful installation

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

```
Sample output

vm-k8s-stack-kube-state-metrics-d9d85978d-9pzbs 1/1 Running 0 28m

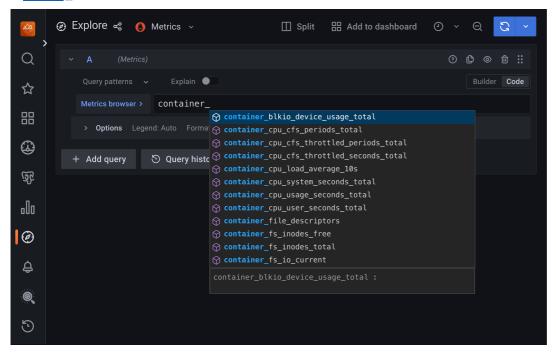
vm-k8s-stack-victoria-metrics-operator-844d558455-gvg4n 1/1 Running 0 28m

vmagent-vm-k8s-stack-victoria-metrics-k8s-stack-55fd8fc4fbcxwhx 2/2 Running 0 28m
```

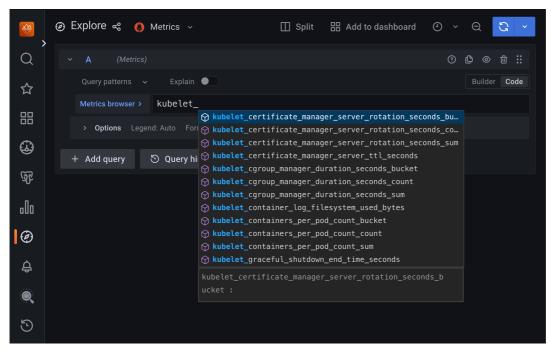
What Pods are running depends on the configuration chosen in values used while installing victoria-metrics-k8s-stack chart.

Verify metrics capture

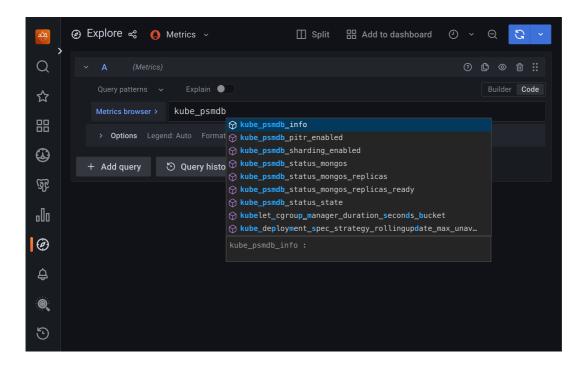
- 1. Connect to the PMM server.
- 2. Click Explore and switch to the Code mode.
- 3. Check that the required metrics are captured, type the following in the Metrics browser dropdown:
 - <u>cadvisor</u> ☐:



kubelet:



• kube-state-metrics 🖸 metrics that also include Custom resource metrics for the Operator and database deployed in your Kubernetes cluster:



Uninstall Victoria metrics Kubernetes stack

To remove Victoria metrics Kubernetes stack used for Kubernetes cluster monitoring, use the cleanup script. By default, the script removes all the <u>Custom</u>
<u>Resource Definitions(CRD)</u> and Secrets associated with the Victoria metrics Kubernetes stack. To keep the CRDs, run the script with the --keep-crd flag.

Remove CRDs

Replace the <NAMESPACE> placeholder with the namespace you specified during the Victoria metrics Kubernetes stack installation:

```
$ bash <(curl -fsL https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/refs/tags/v0.1.1/vm-operator-k8s-
stack/cleanup.sh) --namespace <NAMESPACE>
```

Replace the <NAMESPACE> placeholder with the namespace you specified during the Victoria metrics Kubernetes stack installation:

```
\ bash <(curl -fsL https://raw.githubusercontent.com/Percona-Lab/k8s-monitoring/refs/tags/v0.1.1/vm-operator-k8s-stack/cleanup.sh) --namespace <NAMESPACE> --keep-crd
```

Check that the Victoria metrics Kubernetes stack is deleted:

```
$ helm list -n <namespace>
```

The output should provide the empty list.

If you face any issues with the removal, uninstall the stack manually:

```
$ helm uninstall vm-k8s-stack -n < namespace>
```

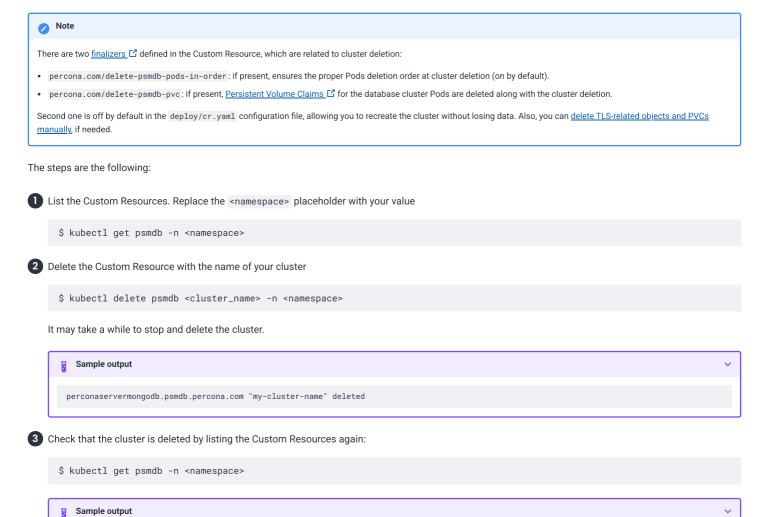
10.9 Delete Percona Operator for MongoDB

You may have different reasons to clean up your Kubernetes environment: moving from trial deployment to a production one, testing experimental configurations and the like. In either case, you need to remove some (or all) of these objects:

- · Percona Distribution for MongoDB cluster managed by the Operator
- · Percona Operator for MongoDB itself
- · Custom Resource Definition deployed with the Operator
- · Resources like PVCs and Secrets

Delete the database cluster

To delete the database cluster means to delete the Custom Resource associated with it.



Delete the Operator

Choose the instructions relevant to the way you installed the Operator. $\label{eq:choose} % \begin{center} \b$

No resources found in <namespace> namespace.

Use kubectl

To uninstall the Operator, delete the $\underline{\text{Deployments}}$ \boxed{C} related to it.

- 1 List the deployments. Replace the <namespace> placeholder with your namespace.
 \$ kubect1 get deploy -n <namespace>
 2 Delete the percona-* deployment
 \$ kubect1 delete deploy percona-server-mongodb-operator -n <namespace>
 3 Check that the Operator is deleted by listing the Pods. As a result you should have no Pods related to it.
 \$ kubect1 get pods -n <namespace>
- 4 If you are not just deleting the Operator and MongoDB cluster from a specific namespace, but want to clean up your entire Kubernetes environment, you can also delete the CustomRecourceDefinitions (CRDs) .

▲ Warning: CRDs in Kubernetes are non-namespaced but are available to the whole environment. This means that you shouldn't delete CRDs if you still have the Operator and database cluster in some namespace.

Get the list of CRDs.

- \$ kubectl get crd
- 5 Delete the percona*.psmdb.percona.com CRDs

No resources found in <namespace> namespace.

- $\$ \ kubectl \ delete \ crd \ perconaservermongodbs.percona.com \ perconaservermongodbrestores.psmdb.percona.com \ perconaservermongodbs.psmdb.percona.com \ perconaservermongodbs.psmdb.psmdb.perconaservermongodbs.psmdb.perconaservermongodbs.psmdb.perconaservermongodb$
- Customresourcedefinition.apiextensions.k8s.io "perconaservermongodbbackups.psmdb.percona.com" deleted customresourcedefinition.apiextensions.k8s.io "perconaservermongodbrestores.psmdb.percona.com" deleted customresourcedefinition.apiextensions.k8s.io "perconaservermongodbs.psmdb.percona.com" deleted

Use Helm

To delete the Operator, do the following:

1 List the Helm charts:

\$ helm list -n <namespace>



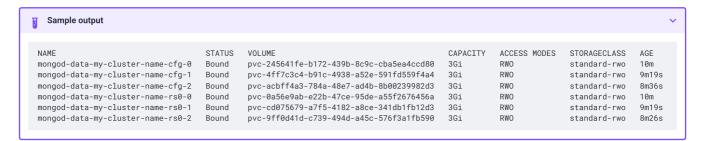
- 2 Delete the <u>release object</u> of for Percona Server for MongoDB
 - \$ helm uninstall cluster1 --namespace <namespace>
- 3 Delete the <u>release object</u> ☐ for the Operator
 - \$ helm uninstall my-op --namespace <namespace>

Clean up resources

By default, TLS-related objects and data volumes remain in Kubernetes environment after you delete the cluster to allow you to recreate it without losing the data. If you wish to delete them, do the following:

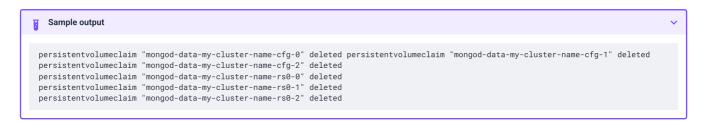
- 1 Delete Persistent Volume Claims.
 - List PVCs. Replace the <namespace> placeholder with your namespace:

\$ kubectl get pvc -n <namespace>



Delete PVCs related to your cluster. The following command deletes PVCs for the my-cluster-name cluster:

\$ kubect1 delete pvc mongod-data-my-cluster-name-cfg-0 mongod-data-my-cluster-name-cfg-1 mongod-data-my-cluster-name-cfg-2 mongod-data-my-cluster-name-rs0-1 mongod-data-my-cluster-name-rs0-2 -n
<namespace>



- 2 Delete the Secrets
 - List Secrets:
 - \$ kubectl get secrets -n <namespace>
 - 2 Delete the Secret:
 - \$ kubectl delete secret <secret_name> -n <namespace>

11 Reference

11.1 Custom Resource options

A Custom Resource (CR) is how you configure the Operator to manage Percona Server for MongoDB. It defines a custom resource of type PerconaServerMongoDB.

This document explains every section of the deploy/cr.yaml Custom Resource manifest and describes available options.

apiVersion

Specifies the API version of the Custom Resource. psmdb.percona.com indicates the group, and v1 is the version of the API.

This tells Kubernetes which version of the custom resource definition (CRD) to use.

kind

Defines the type of resource being created.

metadata

The metadata part of the deploy/cr.yaml contains metadata about the resource, such as its name and other attributes. It includes the following keys:

- name sets the name of your Percona Server for MongoDB Cluster. The name must follow these rules:
 - include only <u>URL-compatible characters</u> [2],
 - not exceed 22 characters,
 - · start and end with an alphanumeric character

The default name is my-cluster-name.

- finalizers ensure safe deletion of resources in Kubernetes under certain conditions. This subsection includes the following finalizers:
 - percona.com/delete-psmdb-pods-in-order if present, activates the <u>Finalizer</u> * which controls the proper Pods deletion order in case of the cluster deletion event (on by default)
 - percona.com/delete-psmdb-pvc if present, activates the Finalizer 🖸 which deletes appropriate Persistent Volume Claims 🖸 after the cluster deletion event (off by default)
 - percona.com/delete-pitr-chunks if present, activates the Finalizer 🖸 which deletes all point-in-time recovery chunks from the cloud storage on cluster deletion (off by default)

Toplevel spec elements

The spec part of the deploy/cr.yaml If file contains the following keys and sections:

platform

Override/set the Kubernetes platform: kubernetes or openshift.

Value type	Example
s string	kubernetes

pause

Pause/resume: setting it to true gracefully stops the cluster, and setting it to false after shut down starts the cluster back.

Value type	Example
→ boolean	false

unmanaged

Setting it to true instructs the Operator to run the cluster in unmanaged state - the Operator does not form replica sets, and does not generate TLS certificates or user credentials. This can be useful for migration scenarios and for cross-site replication.

Value type	Example
ு boolean	false

${\tt enable Volume Expansion}$

Enables or disables <u>automatic storage scaling / volume expansion</u>.

Value type	Example
ு boolean	false

crVersion

Version of the Operator the Custom Resource belongs to.

Value type	Example
S string	1.20.0

image

The Docker image of Percona Server for MongoDB 🖸 to deploy (actual image names can be found in the list of certified images).

Value type	Example
S string	percona/percona-server-mongodb:6.0.21-18

imagePullPolicy

The policy used to update images ☐.

Value type	Example
S string	Always

$\verb|imagePullSecrets.name| \\$

The Kubernetes ImagePullSecret \(\text{ to access the } \(\text{custom registry} \).

Value type	Example
s string	private-registry-credentials

initImage

An alternative image for the initial Operator installation.

Value type	Example
S string	percona/percona-server-mongodb-operator:1.20.0

initContainerSecurityContext

A custom <u>Kubernetes Security Context for a Container</u> of the initImage (image, which can be used instead of the default one while the initial Operator installation).

Value type	Example
≡ subdoc	{}

ClusterServiceDNSSuffix

The (non-standard) cluster domain to be used as a suffix of the Service name.

Value type	Example
s string	svc.cluster.local

clusterServiceDNSMode

Can be internal (local fully-qualified domain names will be used in replset configuration even if the replset is exposed - the default value), external (exposed MongoDB instances will use ClusterIP addresses, should be applied with caution) or ServiceMesh (use a special FQDN based on the Pod name). Being set, ServiceMesh value suprecedes multiCluster settings, and therefore these two modes cannot be combined together.

Va	alue type	Example
S	string	Internal

allowUnsafeConfigurations

Prevents users from configuring a cluster with unsafe parameters: starting it with less than 3 replica set instances, with an even number of replica set instances without additional arbiter, or without TLS/SSL certificates, or running a sharded cluster with less than 3 config server Pods or less than 2 mongos Pods (if false, the Operator will automatically change unsafe parameters to safe defaults). After switching to unsafe configurations permissive mode you will not be able to switch the cluster back by setting spec.allowUnsafeConfigurations key to false, the flag will be ignored. This option is deprecated and will be removed in future releases. Use unsafeFlags subsection instead

Value type	Example
□ boolean	false

updateStrategy

A strategy the Operator uses for <u>upgrades</u>. Possible values are <u>SmartUpdate</u>, <u>RollingUpdate</u> C and <u>OnDelete</u> C.

Value type	Example
s string	SmartUpdate

ignoreAnnotations

The list of annotations to be ignored by the Operator.

Value type	Example
≡ subdoc	service.beta.kubernetes.io/aws-load-balancer-backend-protocol

ignoreLabels

The list of labels to be ignored by the Operator.

Value type	Example
≡ subdoc	rack

multiCluster.enabled

Multi-cluster Services (MCS): setting it to true enables MCS cluster mode [♂.

Value type	Example
□ boolean	false

multiCluster.DNSSuffix

The cluster domain to be used as a suffix for multi-cluster Services used by Kubernetes (svc.clusterset.local by default C).

Value type	Example
S string	svc.clusterset.local

Unsafe flags section

The unsafeFlags section in the deploy/cr.yaml file contains various configuration options to prevent users from configuring a cluster with unsafe parameters.

Once you enable permissive mode with unsafe settings, you cannot disable it by simply turning the same settings back (e.g. by setting a configuration option to false). These settings will be ignored if you try to revert them. Reverting the cluster to a secure state may require additional steps or reinitialization.

unsafeFlags.tls

Prevents users from configuring a cluster without TLS/SSL certificates (if false, the Operator will automatically change unsafe parameters to safe defaults).

Value type	Example
ு boolean	false

unsafeFlags.replsetSize

Prevents users from configuring a cluster with unsafe parameters: starting it with less than 3 replica set instances or with an <u>even number of replica set instances without additional arbiter</u> (if false, the Operator will automatically change unsafe parameters to safe defaults).

Value type	Example
→ boolean	false

unsafeFlags.mongosSize

Prevents users from configuring a sharded cluster with less than 3 config server Pods or less than 2 mongos Pods (if false, the Operator will automatically change unsafe parameters to safe defaults).

Value type	Example	
→ boolean	false	

unsafeFlags.terminationGracePeriod

Prevents users from configuring a sharded cluster without termination grace period for replica set, config servers and mongos Pods.

Value type	Example
◯ boolean	false

unsafeFlags.backupIfUnhealthy

Prevents running backup on a cluster with <u>failed health checks</u> \(\bigsize{\textsf.} \).

Value type	Example
ு boolean	false

TLS (extended cert-manager configuration section)

The tls section in the deploy/cr.yaml 🖸 file contains various configuration options for additional customization of the Transport Layer Security.

tls.mode

Controls if the <u>TLS encryption</u> should be used and/or enforced. Can be disabled, allowTLS, preferTLS, or requireTLS. If set to disabled, it also requires setting unsafeFlags.tls option to true`.

Value type	Example
S string	preferTLS

tls.certValidityDuration

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.

Value type	Example
S string	2160h

tls.allowInvalidCertificates

If true, the mongo shell will not attempt to validate the server certificates. Should be true (default variant) to use self-signed certificates generated by the Operator when there is no cert-manager.

Value type	Example
ு boolean	true

'tls.issuerConf.name'

A cert-manager issuer name 2.

Value type	Example
s string	special-selfsigned-issuer

'tls.issuerConf.kind'

A cert-manager issuer type 2.

'tls.issuerConf.group'

A <u>cert-manager issuer group</u> [2]. Should be cert-manager.io for built-in cert-manager certificate issuers.

Value type	Example
s string	cert-manager.io

Upgrade Options Section

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

upgradeOptions.versionServiceEndpoint

The Version Service URL used to check versions compatibility for upgrade.

Value type	Example
S string	https://check.percona.com

upgradeOptions.apply

Specifies how <u>updates are processed</u> by the Operator. Never or <u>Disabled</u> will completely disable automatic upgrades, otherwise it can be set to <u>Latest</u> or Recommended or to a specific version stringof Percona Server for MongoDB (e.g. 6.0.21-18) that is wished to be version-locked (so that the user can control the version running, but use automatic upgrades to move between them).

Value type	Example
s string	disabled

upgradeOptions.schedule

Scheduled time to check for updates, specified in the <u>crontab format</u> [4].

Value type	Example
S string	0 2 * * *

upgradeOptions.setFCV

If enabled, FeatureCompatibilityVersion (FCV) 🖸 will be set to match the version during major version upgrade.

Value type	Example
□ boolean	false

Secrets section

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

secrets.keyFile

The secret name for the MongoDB Internal Auth Key file []. This secret is auto-created by the operator if it doesn't exist.

Value type	Example
s string	my-cluster-name-mongodb-keyfile

secrets.users

The name of the Secrets object for the MongoDB users required to run the operator.

Value type	Example
s string	my-cluster-name-secrets

secrets.sse

The name of the Secrets object for <u>server side encryption credentials</u>

Value type	Example
s string	my-cluster-name-sse

secrets.ssl

A secret with TLS certificate generated for external communications, see Transport Layer Security (TLS) for details.

Value type	Example
S string	my-custom-ssl

secrets.sslInternal

A secret with TLS certificate generated for internal communications, see <u>Transport Layer Security (TLS)</u> for details.

Value type	Example
S string	my-custom-ssl-internal

secrets.encryptionKey

Value type	Example
S string	my-cluster-name-mongodb-encryption-key

secrets.vault

Specifies a secret object to provide integration with HashiCorp Vault.

Value type	Example
string	my-cluster-name-vault

secrets.ldapSecret

 $Specifies \ a \ secret \ object \ for \ \underline{LDAP \ over \ TLS} \ connection \ between \ MongoDB \ and \ OpenLDAP \ server.$

Value type	Example
s string	my-ldap-secret

Replsets Section

The replsets section controls the MongoDB Replica Set.

replsets.name

The name of the MongoDB Replica Set
☐.

Value type	Example
S string	rs 0

replsets.size

Value type	Example
■ int	3

${\tt replsets.terminationGracePeriodSeconds}$

The amount of seconds Kubernetes will wait for a clean replica set Pods termination.

Value type	Example
int int	300

'replsets.serviceAccountName'

Name of the separate privileged service account for Replica Set Pods.

Value type	Example
S string	default

replsets.topology Spread Constraints.label Selector.match Labels

The label selector for the <u>Kubernetes Pod Topology Spread Constraints</u> [2].

Value type	Example
□ label	app.kubernetes.io/name: percona-server-mongodb

${\tt replsets.topologySpreadConstraints.maxSkew}$

The degree to which Pods may be unevenly distributed under the <u>Kubernetes Pod Topology Spread Constraints</u> ☑.

V	Value type	Example
	int int	1

${\tt replsets.topologySpreadConstraints.topologyKey}$

The key of node labels for the <u>Kubernetes Pod Topology Spread Constraints</u> $\[\]$.

Value type	Example
S string	kubernetes.io/hostname

${\tt replsets.topologySpreadConstraints.whenUnsatisfiable}$

What to do with a Pod if it doesn't satisfy the Kubernetes Pod Topology Spread Constraints .

Value type	Example
S string	DoNotSchedule

replsets.replsetOverrides.MEMBER-NAME.host

Use if you need to <u>override the replica set members FQDNs with custom host names</u>. Each key (MEMBER-NAME) under replset0verrides should be name of a Pod. The Operator won't perform any validation for hostnames, so it's the user's responsibility to ensure connectivity.

Value type	Example
s string	my-cluster-name-rs0-0.example.net:27017

replsets.replsetOverrides.MEMBER-NAME.priority

Use if you need to override the <u>replica set members priorities</u> \square .

Value type	Example
1 int	3

${\tt replsets.replsetOverrides.MEMBER-NAME.tags}$

Optional custom tags which can be added to the replset members to make their identication easier.

Value type	Example
□ label	key: value-0

replsets.externalNodes.host

The URL or IP address of the external replset instance.

Value type	Example
S string	34.124.76.90

replsets.externalNodes.port

The port number of the external replset instance.

Value type	Example
S string	27017

replsets.externalNodes.votes

The number of votes of the external replset instance.

Valu	ne type	Example
S st	tring	0

${\tt replsets.externalNodes.priority}$

The priority of the external replset instance.

Value type	Example
s string	В

replsets.configuration

Custom configuration options for mongod. Please refer to the official manual of for the full list of options, and specific of Server of for MongoDB of docs of the full list of options, and specific of Server of Serve

```
Value type
                                      Example

≡ subdoc

                                      operationProfiling:
                                        mode: slowOp
                                       systemLog:
                                         verbosity: 1
                                       storage:
                                         engine: wiredTiger
                                         wiredTiger:
                                           engineConfig:
                                             directoryForIndexes: false
                                             journalCompressor: snappy
                                           collectionConfig:
                                            blockCompressor: snappy
                                           \verb"indexConfig":
                                             prefixCompression: true
```

${\tt replsets.affinity.antiAffinityTopologyKey}$

The <u>Kubernetes topologyKey</u> ☐ node affinity constraint for the Replica Set nodes.

Value type	Example
S string	kubernetes.io/hostname

replsets.affinity.advanced

In cases where the pods require complex tuning the advanced option turns off the topologykey effect. This setting allows the standard Kubernetes affinity constraints of any complexity to be used.

Value type	Example
≡ subdoc	

replsets.tolerations.key

Value type	Example
S string	node.alpha.kubernetes.io/unreachable

replsets.tolerations.operator

The <u>Kubernetes Pod tolerations</u> ✓ operator for the Replica Set nodes.

Value type	Example
s string	Exists

replsets.tolerations.effect

The <u>Kubernetes Pod tolerations</u> **☐** effect for the Replica Set nodes.

Value type	Example
S string	NoExecute

${\tt replsets.tolerations.tolerationSeconds}$

The Kubernetes Pod tolerations \square time limit for the Replica Set nodes.

Value type	Example
1 int	6000

replsets.primary Prefer Tag Selector.region

Ensures the MongoDB instance is selected as Primary based on specified region

Value type	Example
S string	us-west-2

replsets.primaryPreferTagSelector.zone

Ensures the MongoDB instance is selected as Primary based on specified zone

Va	llue type	Example
S	string	us-west-2c

replsets.priorityClassName

Value type	Example
S string	high priority

replsets.annotations

The <u>Kubernetes annotations</u> <u>Marketing</u> metadata for the Replica Set nodes.

Value type	Example
S string	iam.amazonaws.com/role: role-arn

replsets.labels

The <u>Kubernetes affinity labels</u> of the Replica Set nodes.

Value	type	Example
□ lab	pel	rack: rack-22

${\tt replsets.nodeSelector}$

The <u>Kubernetes nodeSelector</u> \square affinity constraint for the Replica Set nodes.

Value type	Example
□ label	disktype: ssd

replsets.storage.engine

 $Sets the storage.engine\ option\ \underline{https://docs.mongodb.com/manual/reference/configuration-options/\#storage.engine}\ _\ for\ the\ Replica\ Set\ nodes.$

Value type	Example
S string	wiredTiger

replsets.storage.wiredTiger.engineConfig.cacheSizeRatio

The ratio used to compute the storage.wiredTiger.engineConfig.cacheSizeGB option 🖸 for the Replica Set nodes.

Value type	Example
.∞ float	0.5

replsets.storage.wired Tiger.engine Config.directory For Indexes

Sets the storage.wiredTiger.engineConfig.directoryForIndexes option config.directoryForIndexes option config.directoryForIndexes option config.directoryForIndexes option config.directoryForIndexes config.direc

Value type	Example
→ boolean	false

${\tt replsets.storage.wiredTiger.engineConfig.journalCompressor}$

Sets the $\underline{storage.wiredTiger.engineConfig.journalCompressor\ option\ \boxed{``} for the Replica Set\ nodes.$

Value type	Example
S string	snappy

replsets. storage.wired Tiger.collection Config.block Compressor

Sets the $\underline{storage.wiredTiger.collectionConfig.blockCompressor option}$ \square for the Replica Set nodes.

Value type	Example
S string	snappy

replsets.storage.wiredTiger.indexConfig.prefixCompression

Sets the $\underline{storage.wiredTiger.indexConfig.prefixCompression option}$ \square for the Replica Set nodes.

Value type	Example
③ boolean	true

${\tt replsets.storage.in Memory.engine Config.in Memory Size Ratio}$

The ratio used to compute the storage.engine.inMemory.inMemory.inMemorySizeGb option option optional optional</a

Value type	Example
∞ float	0.9

replsets.livenessProbe.failureThreshold

Number of consecutive unsuccessful tries of the <u>liveness probe</u> ☐ to be undertaken before giving up.

Value ty	ype	Example
1 int		4

${\tt replsets.livenessProbe.initialDelaySeconds}$

Number of seconds to wait after the container start before initiating the <u>liveness probe</u> .

Value type	Example
int int	60

${\tt replsets.livenessProbe.periodSeconds}$

How often to perform a <u>liveness probe</u> ☐ (in seconds).

Value type	Example
int int	30

${\tt replsets.livenessProbe.timeoutSeconds}$

Number of seconds after which the <u>liveness probe</u> ☐ times out.

Value type	Example
• int	10

replsets.livenessProbe.startupDelaySeconds

Time after which the liveness probe is failed if the MongoDB instance didn't finish its full startup yet.

Value type	Example
1 int	7200

${\tt replsets.readinessProbe.failureThreshold}$

Number of consecutive unsuccessful tries of the <u>readiness probe</u> ☐ to be undertaken before giving up.

Value type	Example
1 int	8

${\tt replsets.readinessProbe.initialDelaySeconds}$

Number of seconds to wait after the container start before initiating the <u>readiness probe</u> [4].

Value type	Example
1 int	10

replsets.readinessProbe.periodSeconds

How often to perform a <u>readiness probe</u> ☐ (in seconds).

Value type	Example
int int	3

${\tt replsets.readinessProbe.successThreshold}$

 $\begin{tabular}{ll} Minimum consecutive successes for the $\underline{readiness\ probe}$ $\ \square$$ to be considered successful after having failed. \\ \end{tabular}$

Value type	Example
int int	1

${\tt replsets.readinessProbe.timeoutSeconds}$

Number of seconds after which the <u>readiness probe</u> ' times out.

Va	alue type	Example
0	int int	2

'replsets.containerSecurityContext'

A custom <u>Kubernetes Security Context for a Container</u> to be used instead of the default one.

Value type	Example
≡ subdoc	privileged: false

'replsets.podSecurityContext'

Value type	Example
≡ subdoc	runAsUser: 1001 runAsGroup: 1001 supplementalGroups: [1001]

replsets.runtimeClassName

Name of the <u>Kubernetes Runtime Class</u> ☐ for Replica Set Pods.

Value type	Example
S string	image-rc

replsets.sidecars.image

Image for the <u>custom sidecar container</u> for Replica Set Pods.

Value type	Example
S string	busybox

replsets.sidecars.command

Command for the <u>custom sidecar container</u> for Replica Set Pods.

Value type	Example
1 array	["/bin/sh"]

replsets.sidecars.args

Command arguments for the $\underline{\text{custom sidecar container}}$ for Replica Set Pods.

Value type	Example
array	["-c", "while true; do echo echo \$(date -u) 'test' >> /dev/null; sleep 5;done"]

replsets.sidecars.name

Name of the <u>custom sidecar container</u> for Replica Set Pods.

Value type	Example
S string	rs-sidecar-1

replsets.sidecars.volumeMounts.mountPath

Mount path of the <u>custom sidecar container</u> volume for Replica Set Pods.

Value type	Example
S string	/volume1

replsets.sidecars.volumeMounts.name

Name of the $\underline{\text{custom sidecar container}}$ volume for Replica Set Pods.

Value type	Example
s string	sidecar-volume-claim

${\tt replsets.sidecarVolumes.name}$

Name of the <u>custom sidecar container</u> volume for Replica Set Pods.

Value type	Example
S string	sidecar-config

replsets.sidecarVolumes.configMap.name

Name of the ConfigMap ☐ for a custom sidecar container volume for Replica Set Pods.

Value type	Example
s string	myconfigmap

replsets.sidecarVolumes.secret.secretName

Name of the <u>Secret</u> ☐ for a <u>custom sidecar container</u> volume for Replica Set Pods.

Value type	Example
S string	sidecar-secret

${\tt replsets.sidecar Volumes.nfs.server}$

The hostname of the NFS server that will provide remote filesystem to the custom sidecar container volume for Replica Set Pods.

Value type	Example
S string	nfs-service.storage.svc.cluster.local

replsets.sidecarVolumes.nfs.path

The path on the NFS server that will be provided as a remote filesystem to the <u>custom sidecar container</u> volume for Replica Set Pods.

Value type	Example
s string	/psmdb-some-name-rs0

replsets.sidecarPVCs

 $\underline{ \text{Persistent Volume Claim}} \ \underline{ \text{Cl}} \ \text{for the} \ \underline{ \text{custom sidecar container}} \ \text{volume for Replica Set Pods}.$

Value type	Example
≡ subdoc	

${\tt replsets.podDisruptionBudget.maxUnavailable}$

The Kubernetes Pod distribution budget [Ilmit specifying the maximum value for unavailable Pods.

Value type	Example
■ int	1

replsets.podDisruptionBudget.minAvailable

The Kubernetes Pod distribution budget 🖸 limit specifying the minimum value for available Pods.

Value type	Example
1 int	1

replsets.splitHorizons.REPLICASET-POD-NAME.external

External URI for Split-horizon for replica set Pods of the exposed cluster.

Value type	Example
S string	rs0-0.mycluster.xyz

${\tt replsets.splitHorizons.REPLICASET-POD-NAME.external-2}$

External URI for $\underline{\text{Split-horizon}}$ for replica set Pods of the exposed cluster.

Value type	Example
S string	rs0-0.mycluster2.xyz

replsets.expose.enabled

Enable or disable exposing MongoDB Replica Set 🖸 nodes with dedicated IP addresses.

Value	e type	Example
ு bo	oolean	false

replsets.expose.type

The <u>IP address type</u> ✓ to be exposed.

Value type	Example	
s string	ClusterIP	

replsets.expose.loadBalancerClass

Define the implementation of the load balancer you want to use. This setting enables you to select a custom or specific load balancer class instead of the default one provided by the cloud provider.

Value type	Example
S string	eks.amazonaws.com/nlb

${\tt replsets.expose.loadBalancerSourceRanges}$

The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations).

Value type	Example
S string	10.0.0.0/8

replsets.expose.annotations

The Kubernetes annotations $\ \square$ metadata for the MongoDB mongod daemon.

Value type	Example
s string	service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http

replsets.expose.labels

The Kubernetes labels [2] for the MongoDB Replica Set Service.

Value type	Example
S string	rack: rack-22

replsets.expose.internalTrafficPolicy

Specifies whether Service for MongoDB instances should <u>route internal traffic to cluster-wide or to node-local endpoints</u> (it can influence the load balancing effectiveness).

Value type	Example
ு boolean	Local

replsets.expose.externalTrafficPolicy

Specifies whether Service for MongoDB instances should route external traffic [to cluster-wide (Cluster) or to node-local (Local) endpoints. It can influence the load balancing effectiveness.

Value type	Example
string	Local

replsets.nonvoting.enabled

Enable or disable creation of Replica Set non-voting instances within the cluster.

Value type	Example	
ு boolean	false	

replsets.nonvoting.size

The number of Replica Set non-voting instances within the cluster.

Value type	Example
int int	1

replsets.nonvoting.podSecurityContext

A custom <u>Kubernetes Security Context for a Pod</u> C to be used instead of the default one.

Value type	Example
≡ subdoc	{}

replsets.nonvoting.containerSecurityContext

A custom <u>Kubernetes Security Context for a Container</u> of to be used instead of the default one.

Value type	Example
≡ subdoc	()

replsets.nonvoting.afinity.antiAffinityTopologyKey

The Kubernetes topologyKey [] node affinity constraint for the non-voting nodes.

Value type	Example
S string	kubernetes.io/hostname

replsets.nonvoting.affinity.advanced

In cases where the pods require complex tuning the advanced option turns off the topologykey effect. This setting allows the standard Kubernetes affinity constraints of any complexity to be used.

Value type	Example
≡ subdoc	

replsets.nonvoting.tolerations.key

The <u>Kubernetes Pod tolerations</u> [] key for the non-voting nodes.

Value type	Example
S string	node.alpha.kubernetes.io/unreachable

replsets.nonvoting.tolerations.operator

The <u>Kubernetes Pod tolerations</u> ✓ operator for the non-voting nodes.

,	Value type	Example
	S string	Exists

replsets.nonvoting.tolerations.effect

The $\underline{\text{Kubernetes Pod tolerations}} \ \ \square$ effect for the non-voting nodes.

Value type	Example
S string	NoExecute

${\tt replsets.nonvoting.tolerations.tolerationSeconds}$

The Kubernetes Pod tolerations \square time limit for the non-voting nodes.

\	/alue type	Example
	1 int	6000

replsets.nonvoting.priorityClassName

The <u>Kuberentes Pod priority class</u> **☐** for the non-voting nodes.

Value type	Example
S string	high priority

replsets.nonvoting.annotations

The <u>Kubernetes annotations</u> <u>Material of the non-voting nodes.</u>

Value type	Example
S string	iam.amazonaws.com/role: role-arn

replsets.nonvoting.labels

Value type	Example
□ label	rack: rack-22

replsets.nonvoting.nodeSelector

The <u>Kubernetes nodeSelector</u> **☐** affinity constraint for the non-voting nodes.

Value type	Example
□ label	disktype: ssd

replsets.nonvoting.podDisruptionBudget.maxUnavailable

The Kubernetes Pod distribution budget [2] limit specifying the maximum value for unavailable Pods among non-voting nodes.

Value type	Example
■ int	1

${\tt replsets.nonvoting.podDisruptionBudget.minAvailable}$

The Kubernetes Pod distribution budget [2] limit specifying the minimum value for available Pods among non-voting nodes.

Value type	Example
int int	1

replsets.nonvoting.resources.limits.cpu

Kubernetes CPU limit ☐ for MongoDB container.

Va	alue type	Example
S	string	300m

replsets.nonvoting.resources.limits.memory

Kubernetes Memory limit ☐ for MongoDB container.

Value type	Example	
S string	0.5G	

replsets.nonvoting.resources.requests.cpu

The Kubernetes CPU requests CF for MongoDB container.

Value type	Example
s string	300m

replsets.nonvoting.resources.requests.memory

The $\underline{\text{Kubernetes Memory requests}}$ $\boxed{\text{$\square$}}$ for MongoDB container.

Value type	Example
s string	0.5G

replsets.nonvoting.volumeSpec.emptyDir

The Kubernetes emptyDir volume [4], i.e. the directory which will be created on a node, and will be accessible to the MongoDB Pod containers.

Value type	Example
S string	0

${\tt replsets.nonvoting.volumeSpec.hostPath.path}$

Kubernetes hostPath volume [2], i.e. the file or directory of a node that will be accessible to the MongoDB Pod containers.

Value type	Example
s string	/data

${\tt replsets.nonvoting.volumeSpec.hostPath.type}$

The Kubernetes hostPath volume type [2].

Value type	Example
S string	Directory

replsets.nonvoting.volume Spec.persistent Volume Claim.annotations

The <u>Kubernetes annotations</u> \square metadata for <u>Persistent Volume Claim</u> \square .

Value type	Example
S string	service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http

replsets.nonvoting.volume Spec.persistent Volume Claim.labels

The Kubernetes labels 🔀 metadata for Persistent Volume Claim 🖸.

Value type	Example
S string	rack: rack-22

${\tt replsets.nonvoting.volumeSpec.persistentVolumeClaim.storageClassName}$

The <u>Kubernetes Storage Class</u> of to use with the MongoDB container <u>Persistent Volume Claim</u> of for the non-voting nodes. Use Storage Class with XFS as the default filesystem if possible, [for better MongoDB performance [4](https://dba.stackexchange.com/questions/190578/is-xfs-still-the-best-choice-for-mongodb.

Value type	Exam	nple
S string	stan	ndard

replsets.nonvoting.volume Spec.persistent Volume Claim.access Modes

The <u>Kubernetes Persistent Volume</u> access modes for the MongoDB container for the non-voting nodes.

Value type	Example
tij array	["ReadWriteOnce"]

replsets.nonvoting.volume Spec.persistent Volume Claim.resources.requests.storage

The <u>Kubernetes Persistent Volume</u> size for the MongoDB container for the non-voting nodes.

Value type	Example
S string	3Gi

replsets.arbiter.enabled

Enable or disable creation of Replica Set Arbiter ☐ nodes within the cluster.

Value type	Example
ு boolean	false

replsets.arbiter.size

The number of Replica Set Arbiter \square instances within the cluster.

Value type	Example
■ int	1

replsets.arbiter.afinity.antiAffinityTopologyKey

The <u>Kubernetes topologyKey</u> ✓ node affinity constraint for the Arbiter.

Value type	Example
S string	kubernetes.io/hostname

replsets.arbiter.affinity.advanced

In cases where the pods require complex tuning the advanced option turns off the topologykey effect. This setting allows the standard Kubernetes affinity constraints of any complexity to be used.

Value type	Example
≡ subdoc	

replsets.arbiter.tolerations.key

The <u>Kubernetes Pod tolerations</u> **☐** key for the Arbiter nodes.

Value type	Example
S string	node.alpha.kubernetes.io/unreachable

${\tt replsets.arbiter.tolerations.operator}$

The <u>Kubernetes Pod tolerations</u> ☐ operator for the Arbiter nodes.

Value type	Example
S string	Exists

${\tt replsets.arbiter.tolerations.effect}$

The $\underline{\text{Kubernetes Pod tolerations}}$ $\underline{\text{C}}$ effect for the Arbiter nodes.

Value type	Example
S string	NoExecute

${\tt replsets.arbiter.tolerations.tolerationSeconds}$

Value type	Example
1 int	6000

replsets.arbiter.priorityClassName

The <u>Kuberentes Pod priority class</u> ☐ for the Arbiter nodes.

Value type	Example
S string	high priority

replsets.arbiter.annotations

The <u>Kubernetes annotations</u> <u>Marketing</u> metadata for the Arbiter nodes.

Value type	Example
S string	iam.amazonaws.com/role: role-arn

replsets.arbiter.labels

The Kubernetes affinity labels for the Arbiter nodes.

Value type	Example
□ label	rack: rack-22

replsets.arbiter.nodeSelector

The <u>Kubernetes nodeSelector</u> [affinity constraint for the Arbiter nodes.

Value type	Example
□ label	disktype: ssd

replsets.resources.limits.cpu

Kubernetes CPU limit
☐ for MongoDB container.

Value type	Example
S string	300m

replsets.resources.limits.memory

Value type	Example
S string	0.5G

replsets.resources.requests.cpu

The <u>Kubernetes CPU requests</u> ☐ for MongoDB container.

Value type	Example
s string	300m

replsets.resources.requests.memory

The <u>Kubernetes Memory requests</u> ☐ for MongoDB container.

Value type	Example
S string	0.5G

replsets.volumeSpec.emptyDir

The Kubernetes emptyDir volume [4], i.e. the directory which will be created on a node, and will be accessible to the MongoDB Pod containers.

Value type	Example	
s string	0	

replsets.volumeSpec.hostPath.path

<u>Kubernetes hostPath volume</u> **[** d, i.e. the file or directory of a node that will be accessible to the MongoDB Pod containers.

Value type	Example
s string	/data

replsets.volumeSpec.hostPath.type

The <u>Kubernetes hostPath volume type</u>
☐.

Value type	Example
S string	Directory

${\tt replsets.volumeSpec.persistentVolumeClaim.annotations}$

The <u>Kubernetes annotations</u> [] metadata for <u>Persistent Volume Claim</u> [].

Value type	Example
S string	service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http

replsets.volumeSpec.persistentVolumeClaim.labels

The Kubernetes labels 🖸 metadata for Persistent Volume Claim 🖒.

Value type	Example
S string	rack: rack-22

replsets.volumeSpec.persistentVolumeClaim.storageClassName

The <u>Kubernetes Storage Class</u> of to use with the MongoDB container <u>Persistent Volume Claim</u>. Use Storage Class with XFS as the default filesystem if possible, <u>for better MongoDB performance</u>.

Value type	Example
S string	standard

replsets.volume Spec.persistent Volume Claim.access Modes

The Kubernetes Persistent Volume 🖸 access modes for the MongoDB container.

Value type	Example
[] array	["ReadWriteOnce"]

repl sets. volume Spec.persistent Volume Claim.resources.requests. storage

Value ty	ре	Example
S string)	3Gi

replsets.hostAliases.ip

The IP address for <u>Kubernetes host aliases</u> [for replica set Pods.

Value type	Example
s string	"10.10.0.2"

replsets.hostAliases.hostnames

Hostnames for Kubernetes host aliases \square for replica set Pods.

,	Value type	Example
	≡ subdoc	

PMM Section

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

pmm.enabled

Enables or disables monitoring Percona Server for MongoDB with $\underline{\sf PMM}$ $\underline{\sf C}'$.

Value type	Example
ு boolean	false

pmm.image

PMM Client docker image to use.

Value type	Example
s string	percona/pmm-client:2.44.1

pmm.serverHost

Address of the PMM Server to collect data from the Cluster.

Value type	Example
S string	monitoring-service

pmm.containerSecurityContext

A custom <u>Kubernetes Security Context for a Container</u> of to be used instead of the default one.

Value type	Example
≡ subdoc	0

pmm.customClusterName

A custom name to define for a cluster. PMM Server uses this name to properly parse the metrics and display them on dashboards. Using a custom name is useful for clusters deployed in different data centers - PMM Server connects them and monitors them as one deployment. Another use case is for clusters deployed with the same name in different namespaces - PMM treats each cluster separately.

Value type	Example
S string	mongo-cluster

pmm.mongodParams

Additional parameters which will be passed to the pmm-admin add mongodb [command for mongod Pods.

Value type	Example
S string	environment=DEV-ENVcustom-labels=DEV-ENV

pmm.mongosParams

Additional parameters which will be passed to the $\underline{pmm-admin\ add\ mongodb}$ [2] command for mongos Pods.

Value type	Example
S string	environment=DEV-ENVcustom-labels=DEV-ENV

Sharding Section

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

sharding.enabled

Enables or disables Percona Server for MondoDB sharding ...

Value type	Example
→ boolean	true

sharding.configsvrReplSet.size

The number of Config Server instances within the cluster.

Value type	Example
1 int	3

sharding.configsvrReplSet.terminationGracePeriodSeconds

The amount of seconds Kubernetes will wait for a clean config server Pods termination.

Value type	Example
■ int	300

'sharding.configsvrReplSet.serviceAccountName'

Name of the separate privileged service account for Config Server Pods.

Value type	Example	
S string	default	

sharding.configsvrReplSet.topologySpreadConstraints.labelSelector.matchLabels

The label selector for the <u>Kubernetes Pod Topology Spread Constraints</u> [7].

Value type	Example
□ label	app.kubernetes.io/name: percona-server-mongodb

sharding.configsvrReplSet.topologySpreadConstraints.max Skew

The degree to which Pods may be unevenly distributed under the Kubernetes Pod Topology Spread Constraints [4].

Value type	Example
1 int	1

sharding.configsvrReplSet.topologySpreadConstraints.topologyKey

The key of node labels for the Kubernetes Pod Topology Spread Constraints [-].

Value type	Example
S string	kubernetes.io/hostname

sharding.configs vrReplSet.topology Spread Constraints.when Unsatisfiable

What to do with a Pod if it doesn't satisfy the <u>Kubernetes Pod Topology Spread Constraints</u> \square .

Value type	Example
s string	DoNotSchedule

$\verb|sharding.configsvrReplSet.externalNodes.host|\\$

The URL or IP address of the external config server instance.

Value type	Example
S string	34.124.76.90

${\tt sharding.configsvrReplSet.externalNodes.port}$

The port number of the external config server instance.

Valu	ue type	Example
5 5	string	27017

sharding.configsvrReplSet.externalNodes.votes

The number of <u>votes</u> of the <u>external config server instance</u>.

Value type	Example
S string	0

${\tt sharding.configsvrReplSet.externalNodes.priority}$

The <u>priority</u> ☐ of the <u>external config server instance</u>.

Value type	Example
S string	0

sharding.configsvrReplSet.configuration

Custom configuration options for Config Servers. Please refer to the official manual 🔀 for the full list of options.

Value type	Example
S string	 operationProfiling: mode: slowOp systemLog: verbosity: 1

sharding.configsvrReplSet.livenessProbe.failureThreshold

Number of consecutive unsuccessful tries of the $\underline{\text{liveness probe}}$ $\underline{\text{C}}$ to be undertaken before giving up.

Value type	Example
1 int	4

sharding.configsvrReplSet.livenessProbe.initialDelaySeconds

Number of seconds to wait after the container start before initiating the <u>liveness probe</u> ☐.

Value type	Example
int int	60

sharding.configsvrReplSet.livenessProbe.periodSeconds

How often to perform a <u>liveness probe</u> \square (in seconds).

Value type	Example	
1 int	30	

sharding.configs vr Repl Set. 1 iveness Probe. time out Seconds

Number of seconds after which the <u>liveness probe</u> ☐ times out.

,	Value type	Example
	1 int	10

sharding.configsvrReplSet.liveness Probe.startup Delay Seconds

Time after which the liveness probe is failed if the MongoDB instance didn't finish its full startup yet.

Value type	Example
1 int	7200

sharding.configsvrReplSet.readinessProbe.failureThreshold

Number of consecutive unsuccessful tries of the $\underline{readiness\ probe}$ \square to be undertaken before giving up.

Value type	Example
s int	3

sharding.configs vrReplSet.readiness Probe.initial Delay Seconds

Number of seconds to wait after the container start before initiating the <u>readiness probe</u> ☑.

Value type	Example
int int	10

sharding.configsvr Repl Set.readiness Probe.period Seconds

How often to perform a <u>readiness probe</u> \square (in seconds).

Value type	Example
1 int	3

sharding.configsvrReplSet.readinessProbe.successThreshold

Minimum consecutive successes for the <u>readiness probe</u> ☐ to be considered successful after having failed.

Value type	Example
1 int	1

sharding.configsvrReplSet.readiness Probe.time out Seconds

Number of seconds after which the <u>readiness probe</u> 🖸 times out.

Value type	Example
1 int	2

's harding.configs vr Repl Set.container Security Context'

A custom <u>Kubernetes Security Context for a Container</u> 🖸 to be used instead of the default one.

Value type	Example
≡ subdoc	privileged: false

's harding.configs vr Repl Set.pod Security Context'

A custom <u>Kubernetes Security Context for a Pod</u> C to be used instead of the default one.

Value type	Example
≡ subdoc	runAsUser: 1001 runAsGroup: 1001 supplementalGroups: [1001]

${\tt sharding.configsvrReplSet.runtimeClassName}$

Name of the Kubernetes Runtime Class \Box for Config Server Pods.

Value type	Example
S string	image-rc

sharding.configsvrReplSet.sidecars.image

Image for the <u>custom sidecar container</u> for Config Server Pods.

Value type	Example
S string	busybox

sharding.configsvrReplSet.sidecars.command

Command for the <u>custom sidecar container</u> for Config Server Pods.

Value type	Example
[] array	["/bin/sh"]

${\tt sharding.configsvrReplSet.sidecars.args}$

Command arguments for the $\underline{\text{custom sidecar container}}$ for Config Server Pods.

Value type	Example
[] array	["-c", "while true; do echo echo \$(date -u) 'test' >> /dev/null; sleep 5;done"]

$\verb|sharding.configsvrReplSet.sidecars.name| \\$

Name of the $\underline{\text{custom sidecar container}}$ for Config Server Pods.

Value typ	ре	Example
s string		rs-sidecar-1

sharding.configsvrReplSet.sidecarVolumes.name

Name of the <u>custom sidecar container</u> volume for Config Server Pods.

Value type	Example
s string	sidecar-config

sharding.configsvr Repl Set.side car Volumes.nfs.server

The hostname of the NFS server that will provide remote filesystem to the <u>custom sidecar container</u> volume for Config Server Pods.

Value type	Example
S string	nfs-service.storage.svc.cluster.local

sharding.configs vr Repl Set.side car Volumes.nfs.path

The path on the NFS server that will be provided as a remote filesystem to the <u>custom sidecar container</u> volume for Config Server Pods.

Value type	Example
S string	/psmdb-some-name-rs0

sharding.configsvrReplSet.limits.cpu

Kubernetes CPU limit ☐ for Config Server container.

Value type	Example
S string	300m

sharding.configsvrReplSet.limits.memory

Kubernetes Memory limit
☐ for Config Server container.

Value type	Example
S string	0.5G

${\tt sharding.configsvrReplSet.resources.requests.cpu}$

The Kubernetes CPU requests [2] for Config Server container.

Value type	Example
s string	300m

sharding.configsvrReplSet.requests.memory

The <u>Kubernetes Memory requests</u> ☐ for Config Server container.

Value type	Example
S string	0.5G

$\verb|sharding.configsvrReplSet.expose.enabled|\\$

Enable or disable exposing Config Server \(\textstyle \text{ nodes with dedicated IP addresses.} \)

Value typ	e e	Example
⊸ boolea	an	false

sharding.configsvrReplSet.expose.type

The $\underline{\mathsf{IP}}$ address type \square to be exposed.

Value type	Example
S string	ClusterIP

sharding.configs vr Repl Set.expose.load Balancer Class

Define the implementation of the load balancer you want to use. This setting enables you to select a custom or specific load balancer class instead of the default one provided by the cloud provider.

Value type	Example
S string	eks.amazonaws.com/nlb

sharding.configsvrReplSet.expose.loadBalancerSourceRanges

The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations).

Value type	Example
s string	10.0.0.0/8

sharding.configsvrReplSet.expose.annotations

The <u>Kubernetes annotations</u> [2] metadata for the Config Server daemon.

Value type	Example
S string	service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http

sharding.configsvrReplSet.expose.labels

The Kubernetes labels for the Config Server Service.

Value type	Example
S string	rack: rack-22

${\tt sharding.configsvrReplSet.expose.internalTrafficPolicy}$

Specifies whether Service for config servers should <u>route internal traffic to cluster-wide or to node-local endpoints</u> [2] (it can influence the load balancing effectiveness).

Value type	Example
→ boolean	Local

sharding.configsvrReplSet.expose.externalTrafficPolicy

Specifies whether Service for config servers should route external traffic of to cluster-wide (Cluster) or to node-local (Local) endpoints. It can influence the load balancing effectiveness.

Value type	Example
S string	Local

sharding.configsvrReplSet.volumeSpec.emptyDir

The Kubernetes emptyDir volume. [2], i.e. the directory which will be created on a node, and will be accessible to the Config Server Pod containers.

Value type	Example
s string	{}

sharding.configsvrReplSet.volumeSpec.hostPath.path

Kubernetes hostPath volume [1], i.e. the file or directory of a node that will be accessible to the Config Server Pod containers.

Value type	Example
S string	/data

sharding.configsvrReplSet.volumeSpec.hostPath.type

The Kubernetes hostPath volume type [2].

Value type	Example
S string	Directory

sharding.configsvrReplSet.volumeSpec.persistentVolumeClaim.annotations

The <u>Kubernetes annotations</u> \(\textstyle{\textstyle{L}} \) metadata for <u>Persistent Volume Claim</u> \(\textstyle{\textstyle{L}} \).

Value typ	oe .	Example
S string		service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http

sharding.configsvrReplSet.volumeSpec.persistentVolumeClaim.labels

The Kubernetes labels [] metadata for Persistent Volume Claim [].

Value type	Example
s string	rack: rack-22

sharding.configsvrReplSet.volumeSpec.persistentVolumeClaim.storageClassName

The <u>Kubernetes Storage Class</u> of to use with the Config Server container <u>Persistent Volume Claim</u>. Use Storage Class with XFS as the default filesystem if possible, <u>for better MongoDB performance</u>.

Value type	Example
S string	standard

sharding.configsvrReplSet.volumeSpec.persistentVolumeClaim.accessModes

The Kubernetes Persistent Volume $\ \ \Box$ access modes for the Config Server container.

Value type	Example
t) array	["ReadWriteOnce"]

sharding.configsvrReplSet.volumeSpec.persistent VolumeClaim.resources.requests.storage and the property of t

The $\underline{\text{Kubernetes Persistent Volume}}$ \square size for the Config Server container.

Valu	e type	Example
S st	tring	3Gi

sharding.configsvrReplSet.hostAliases.ip

The IP address for <u>Kubernetes host aliases</u>
☐ for replica set Pods.

Value type	Example
s string	"10.10.0.2"

$sharding.configsvr Repl Set.host A \verb|liases.host| names$

Value type	Example
≡ subdoc	

sharding.mongos.size

The number of mongos ☐ instances within the cluster.

Value type	Example
int int	3

$\verb|sharding.mongos.terminationGracePeriodSeconds|\\$

The amount of seconds Kubernetes will wait for a clean mongos Pods termination.

Value type	Example
int int	300

'sharding.mongos.serviceAccountName'

Name of the separate privileged service account for mongos Pods.

Value type	Example
s string	default

sharding.mongos.topology Spread Constraints.label Selector.match Labels

The label selector for the <u>Kubernetes Pod Topology Spread Constraints</u> .

Value type	Example
□ label	app.kubernetes.io/name: percona-server-mongodb

sharding.mongos.topologySpreadConstraints.maxSkew

The degree to which Pods may be unevenly distributed under the Kubernetes Pod Topology Spread Constraints [4].

Value type	Example
1 int	1

sharding.mongos.topology Spread Constraints.topology Key

The key of node labels for the <u>Kubernetes Pod Topology Spread Constraints</u> C.

Value type	Example
S string	kubernetes.io/hostname

sharding.mongos.topology Spread Constraints.when Unsatisfiable

Value type	Example
S string	DoNotSchedule

sharding.mongos.configuration

Custom configuration options for mongos. Please refer to the $\underline{\text{official manual}}$ \underline{C} for the full list of options.

Value type	Example
S string	 systemLog: verbosity: 1

sharding.mongos.a finity.antiAffinity Topology Key

The $\underline{\text{Kubernetes topologyKey}}$ \square node affinity constraint for mongos.

Value type	Example
S string	kubernetes.io/hostname

${\tt sharding.mongos.affinity.advanced}$

In cases where the Pods require complex tuning the advanced option turns off the topologykey effect. This setting allows the standard Kubernetes affinity constraints of any complexity to be used.

Value type	Example
≡ subdoc	

sharding.mongos.tolerations.key

The <u>Kubernetes Pod tolerations</u> \square key for mongos instances.

Value type	Example
S string	node.alpha.kubernetes.io/unreachable

sharding.mongos.tolerations.operator

The <u>Kubernetes Pod tolerations</u> operator for mongos instances.

Value type	Example
S string	Exists

sharding.mongos.tolerations.effect

Value type	Example
s string	NoExecute

$\verb|sharding.mongos.tolerations.tolerationSeconds|\\$

The Kubernetes Pod tolerations \square time limit for mongos instances.

Value type	Example	
■ int	6000	

sharding.mongos.priorityClassName

The <u>Kuberentes Pod priority class</u> **☐** for mongos instances.

Value type	Example
s string	high priority

sharding.mongos.annotations

The Kubernetes annotations \square metadata for the mongos instances.

Value type	Example
s string	iam.amazonaws.com/role: role-arn

sharding.mongos.labels

The Kubernetes affinity labels 🖸 for mongos instances.

Value type	Example
□ label	rack: rack-22

sharding.mongos.nodeSelector

The $\underline{\text{Kubernetes nodeSelector}}$ \square affinity constraint for mongos instances.

Value type	Example
□ label	disktype: ssd

$sharding. mongos. \\ liveness Probe. \\ failure Threshold$

Number of consecutive unsuccessful tries of the <u>liveness probe</u> ☐ to be undertaken before giving up.

Value type	Example
1 int	4

sharding.mongos.liveness Probe.initial Delay Seconds

Number of seconds to wait after the container start before initiating the liveness probe .

Value type	Example
1 int	60

sharding.mongos.liveness Probe.period Seconds

How often to perform a <u>liveness probe</u> \square (in seconds).

Value type	Example
1 int	30

sharding.mongos.livenessProbe.timeoutSeconds

Number of seconds after which the <u>liveness probe</u> ☐ times out.

Value type	Example
1 int	10

sharding.mongos.livenessProbe.startupDelaySeconds

Time after which the liveness probe is failed if the MongoDB instance didn't finish its full startup yet.

Value type	Example	
■ int	7200	

sharding.mongos.readiness Probe.failure Threshold

Number of consecutive unsuccessful tries of the <u>readiness probe</u> ☐ to be undertaken before giving up.

Value type	Example
■ int	3

sharding.mongos.readiness Probe.initial Delay Seconds

Number of seconds to wait after the container start before initiating the <u>readiness probe</u> ☐.

Value	e type	Example
1 in	nt	10

$\verb|sharding.mongos.readinessProbe.periodSeconds|\\$

How often to perform a <u>readiness probe</u> ☐ (in seconds).

Value type	Example
1 int	3

sharding.mongos.readiness Probe.success Threshold

Minimum consecutive successes for the readiness probe C to be considered successful after having failed.

Value type	Example
1 int	1

$\verb|sharding.mongos.readinessProbe.timeoutSeconds|\\$

Number of seconds after which the <u>readiness probe</u> 🔀 times out.

Value type	Example
int int	2

'sharding.mongos.containerSecurityContext'

A custom <u>Kubernetes Security Context for a Container</u> C to be used instead of the default one.

Value type	Example
≡ subdoc	privileged: false

'sharding.mongos.podSecurityContext'

A custom <u>Kubernetes Security Context for a Pod</u> ' to be used instead of the default one.

Value type	Example
≡ subdoc	runAsUser: 1001 runAsGroup: 1001 supplementalGroups: [1001]

$\verb|sharding.mongos.runtimeClassName| \\$

Name of the <u>Kubernetes Runtime Class</u> ☐ for mongos Pods.

Value type	Example
S string	image-rc

sharding.mongos.sidecars.image

Image for the <u>custom sidecar container</u> for mongos Pods.

Value type	Example
S string	busybox

sharding.mongos.sidecars.command

Command for the $\underline{\text{custom sidecar container}}$ for mongos Pods.

Value type	Example
[] array	["/bin/sh"]

sharding.mongos.sidecars.args

Command arguments for the $\underline{\text{custom sidecar container}}$ for mongos Pods.

Value type	Example
☐ array	["-c", "while true; do echo echo \$(date -u) 'test' >> /dev/null; sleep 5;done"]

sharding.mongos.sidecars.name

Name of the <u>custom sidecar container</u> for mongos Pods.

Value type	Example
s string	rs-sidecar-1

sharding.mongos.limits.cpu

Kubernetes CPU limit ☐ for mongos container.

Value type	Example
s string	300m

sharding.mongos.limits.memory

<u>Kubernetes Memory limit</u> ☐ for mongos container.

Value type	Example
S string	0.5G

sharding.mongos.resources.requests.cpu

The <u>Kubernetes CPU requests</u> ☐ for mongos container.

Value type	Example
S string	300m

sharding.mongos.requests.memory

The <u>Kubernetes Memory requests</u> ☐ for mongos container.

Value type	Example
S string	0.5G

sharding.mongos.expose.type

The $\underline{\mathsf{IP}}$ address type \square to be exposed.

Value type	Example
S string	ClusterIP

${\tt sharding.mongos.expose.servicePerPod}$

If set to true, a separate ClusterIP Service is created for each mongos instance.

Value type	Example
→ boolean	true

sharding.mongos.expose.loadBalancerClass

Define the implementation of the load balancer you want to use. This setting enables you to select a custom or specific load balancer class instead of the default one provided by the cloud provider.

Value type	Example
S string	eks.amazonaws.com/nlb

sharding.mongos.expose.loadBalancerSourceRanges

The range of client IP addresses from which the load balancer should be reachable (if not set, there is no limitations).

Value type	Example
S string	10.0.0.0/8

sharding.mongos.expose.annotations

Value type	Example
S string	service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http

sharding.mongos.expose.labels

The Kubernetes labels for the MongoDB mongos Service.

Value type	Example
S string	rack: rack-22

'sharding.mongos.expose.nodePort'

The Node port number To to be allocated for the MongoDB mongos Service when the sharding.mongos.expose.type is set to the NodePort, and sharding.mongos.expose.servicePerPod is not turned on.

Value type	Example
■ int	32017

${\tt sharding.mongos.internalTrafficPolicy}$

Specifies whether Services for the mongos instances should <u>route internal traffic to cluster-wide or to node-local endpoints</u> (it can influence the load balancing effectiveness).

v	alue type	Example
	₃ boolean	Local

${\tt sharding.mongos.externalTrafficPolicy}$

Specifies whether Service for the mongos instances should route external traffic of to cluster-wide (Cluster) or to node-local (Local) endpoints. It can influence the load balancing effectiveness.

Value type	Example
s string	Local

$\verb|sharding.mongos.hostAliases.ip|$

The IP address for <u>Kubernetes host aliases</u> \square for mongos Pods.

Value type	Example
S string	"10.10.0.2"

$\verb|sharding.mongos.hostAliases.hostnames|\\$

Hostnames for <u>Kubernetes host aliases</u> ☐ for mongos Pods.

Value type	Example
≡ subdoc	

Roles section

The roles section in the deploy/cr.yaml C file contains various configuration options to configure custom MongoDB user roles via the Custom Resource.

roles.role

The <u>cusom MongoDB role</u>
☐ name.

Value type	Example
s string	myClusterwideAdmin

roles.db

Database in which you want to store the user-defined role.

Value type	Example
s string	`admin

roles.authenticationRestrictions.clientSource

List of the IP addresses or CIDR blocks from which users assigned this role can connect. MongoDB servers reject connection requests from users with this role if the requests come from a client that is not present in this array.

Value type	Example
≡ subdoc	127.0.0.1

roles.authenticationRestrictions.serverAddress

List of the IP addresses or CIDR blocks to which users assigned this role can connect. MongoDB servers reject connection requests from users with this role if the client requests to connect to a server that is not present in this array.

Value type	Example
≡ subdoc	127.0.0.1

roles.privileges.actions

List of custom role actions that users granted this role can perform: For a list of accepted values, see Privilege Actions. 🗗 in the MongoDB Manual.

Value type	Example
≡ subdoc	addShard

roles.privileges.resource.db

Database for which the custom role actions apply. An empty string ("") indicates that the privilege actions apply to all databases.

Value type	Example
S string	***

roles.privileges.resource.collection

Collection for which the custom role actions apply. An empty string ("") indicates that the privilege actions apply to all of the database's collections.

Value type	Example
S string	**

roles.privileges.resource.cluster

If true, the custom role actions apply to all databases and collections in the MongoDB deployment. False by default. If set to true, values for roles.privileges.resource.db and roles.privileges.resource.collection shouldn't be provided.

Value type	Example
◯ boolean	true

roles.roles

An array of roles (with names of the role and the database) from which this role inherits privileges, if any.

Value type	Example
≡ subdoc	role: read db: admin

Users section

The users section in the deploy/cr.yaml L file contains various configuration options to configure custom MongoDB users via the Custom Resource.

users.name

The username of the MongoDB user.

Value type	Example
s string	my-user

users.db

Database that the user authenticates against.

Value type	Example
S string	admin

users.passwordSecretRef.name

Name of the secret that contains the user's password. If passwordSecretRef is not present, password will be generated automatically.

Value type	Example
S string	my-user-password

users.passwordSecretRef.key

Key in the secret that corresponds to the value of the user's password (password by default).

Value ty	уре	Example
s strin	g	password

users.roles.role.name

Name of the MongoDB role assigned to the user. As built-in roles, so custom roles are supported.

Value type	Example
S string	clusterAdmin

users.roles.role.db

Database that the MongoDB role applies to.

Value type	Example
s string	admin

Backup Section

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

backup.enabled

Enables or disables making backups.

Value type	Example
	true

backup.image

The Percona Server for MongoDB Docker image to use for the backup.

Value type	Example
S string	percona/percona-server-mongodb-operator:1.20.0-backup

backup.serviceAccountName

Name of the separate privileged service account for backups; service account for backups is not used by the Operator any more, and the option is deprecated since the Operator version 1.16.0.

Value type	Example
s string	percona-server-mongodb-operator

backup.annotations

Value type	Example
S string	sidecar.istio.io/inject: "false"

backup.resources.limits.cpu

Kubernetes CPU limit ☐ for backups.

Value type	Example
S string	300m

backup.resources.limits.memory

Kubernetes Memory limit ' for backups.

Value type	Example
string	1.26

backup.resources.requests.cpu

The <u>Kubernetes CPU requests</u> [☐ for backups.

Value type	Example
S string	300m

backup.resources.requests.memory

The Kubernetes Memory requests \square for backups.

Value type	Example
S string	16

'backup.containerSecurityContext'

A custom $\underline{\text{Kubernetes Security Context for a Container}}$ to be used instead of the default one.

Value type	Example
≡ subdoc	privileged: false

${\tt backup.storages.STORAGE-NAME.main}$

Marks the storage as main. All other storages you define are added as profiles. The Operator saves backups to all storages but it saves oplog chunks for point-in-time recovery only to the main storage. You can define only one storage as main. Read more about <u>multiple storages for backups</u>.

Value type	Example
→ boolean	true

backup.storages.STORAGE-NAME.type

The cloud storage type used for backups. Only s3, azure, and filesystem types are supported.

Value type	Example
S string	s3

backup.storages.STORAGE-NAME.s3.insecureSkipTLSVerify

Enable or disable verification of the storage server TLS certificate. Disabling it may be useful e.g. to skip TLS verification for private S3-compatible storage with a self-issued certificate.

Value type	Example
ு boolean	true

backup.storages.STORAGE-NAME.s3.credentialsSecret

The Kubernetes secret C for backups. It should contain AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY keys.

Value type	Example
s string	my-cluster-name-backup-s3

backup.storages.STORAGE-NAME.s3.bucket

The <u>Amazon S3 bucket</u> ☐ name for backups.

Value type	Example
S string	

backup.storages.STORAGE-NAME.s3.prefix

The path (sub-folder) to the backups inside the bucket ...

Value type	Example
s string	**

backup.storages.STORAGE-NAME.s3.uploadPartSize

The size of data chunks in bytes to be uploaded to the storage bucket (10 MiB by default).

Value type	Example
int int	10485760

backup.storages.STORAGE-NAME.s3.maxUploadParts

The maximum number of data chunks to be uploaded to the storage bucket (10000 by default).

Value ty	ре	Example
1 int		10000

backup.storages. STORAGE-NAME.s3.storage Class

The <u>storage class name</u> ☐ of the S3 storage.

Value type	Example
S string	STANDARD

backup.storages.STORAGE-NAME.s3.retryer.numMaxRetries

The maximum number of retries to upload data to S3 storage.

Value type	Example
1 int	3

backup.storages.STORAGE-NAME.s3.retryer.minRetryDelay

The minimum time in milliseconds to wait till the next retry.

Value type	Example
1 int	10

backup.storages.STORAGE-NAME.s3.retryer.maxRetryDelay

The maximum time in minutes to wait till the next retry.

Value type	Example	
1 int	5	

backup.storages.STORAGE-NAME.s3.region

The $\underline{\mathsf{AWS}}$ region $\underline{\mathsf{C}}$ to use. Please note this option is mandatory for Amazon and all S3-compatible storages.

Value type	Example
s string	us-east-1

backup.storages.STORAGE-NAME.s3.endpointUrl

The URL of the S3-compatible storage to be used (not needed for the original Amazon S3 cloud).

Value type	Example
s string	

backup.storages.STORAGE-NAME.s3.serverSideEncryption.kmsKeyID

The ID of the key stored in the AWS KMS [2] used by the Operator for backups server-side encryption

Value type	Example
S string	***

backup.storages.STORAGE-NAME.s3.serverSideEncryption.sseAlgorithm

The key management mode used for <u>backups server-side encryption</u> with the encryption keys stored in <u>AWS KMS</u> C - aws:kms is the only supported value for now.

Value type	Example
S string	aws:kms

backup.storages.STORAGE-NAME.s3.serverSideEncryption.sseCustomerAlgorithm

The key management mode for backups server-side encryption with customer-provided keys - AES256 is the only supported value for now.

Value type	Example	
S string	AES256	

backup.storages.STORAGE-NAME.s3.serverSideEncryption.sseCustomerKey

The locally-stored base64-encoded custom encryption key used by the Operator for <u>backups server-side encryption</u> on S3-compatible storages.

Value type	Example
s string	и и

backup.storages.STORAGE-NAME.azure.credentials Secret

The Kubernetes secret C for backups. It should contain AZURE_STORAGE_ACCOUNT_NAME and AZURE_STORAGE_ACCOUNT_KEY |

Value type	Example
s string	my-cluster-azure-secret

backup.storages.STORAGE-NAME.azure.container

Name of the container of for backups.

Value type	Example
S string	my-container

backup.storages.STORAGE-NAME.azure.prefix

The path (sub-folder) to the backups inside the container ...

Value type	Example
S string	•••

'backup.storages.STORAGE-NAME.azure.endpointUrl'

The private endpoint URL to use instead of the public endpoint.

Value type	Example
S string	https://accountName.blob.core.windows.net

${\tt backup.storages.STORAGE-NAME.filesystem.path}$

The mount point for a remote filesystem configured to store backups.

Value type	Example
s string	/mnt/nfs/

$backup.volume {\tt Mounts.mountPath}$

Mount path for the remote backup storage.

Value type	Example
S string	/mnt/nfs/

backup.volumeMounts.name

Name of the remote backup storage.

\	Value type	Example
1	s string	backup-nfs

backup.pitr.enabled

Enables or disables point-in-time-recovery functionality.

Value type	Example	
□ boolean	false	

backup.pitr.oplogOnly

If true, Percona Backup for MongoDB saves oplog chunks even without the base logical backup snapshot (oplog chunks without a base backup can't be used with logical backups to restore a backup by the Operator, but can still be useful for manual restore operations.

Value type	Example
ு boolean	false

backup.pitr.oplogSpanMin

Number of minutes between the uploads of oplogs.

Value type	Example
1 int	10

backup.pitr.compressionType

The point-in-time-recovery chunks compression format, can be gzip, snappy, Iz4, pgzip, zstd, s2, or none [2].

Value type	Example
S string	gzip

backup.pitr.compressionLevel

The point-in-time-recovery chunks compression level (higher values result in better but slower compression. C).

Value type	Example
■ int	6

backup.configuration.backupOptions.priority

The list of mongod nodes and their priority for making backups.

Value type	Example
≡ subdoc	"localhost:28019": 2.5 "localhost:27018": 2.5

backup.configuration.backup Options.timeouts.starting Status

The wait time in seconds Percona Backup for MongoDB should use to start physical backups on all shards. The 0 (zero) value resets the timeout to the default 33 seconds.

Value type	Example
1 int	33

backup.configuration.backupOptions.oplogSpanMin

The duration (in minutes) of oplog slices saved by Percona Backup for MongoDB with the logical backup snapshot.

Value type	Example
int int	10

${\tt backup.configuration.restoreOptions.batchSize}$

The number of documents Percona Backup for MongoDB should buffer.

Value type	Example
■ int	500

backup.configuration.restoreOptions.numInsertionWorkers

The number of workers that Percona Backup for MongoDB should use to add the documents to buffer.

Value type	Example
1 int	10

backup.configuration.restore Options.num Download Workers

The number of workers that Percona Backup for MongoDB should use to request data chunks from the storage during the restore.

Value type	Example
int int	4

backup.configuration.restore Options.max Download Buffer Mb

The maximum size of the in-memory buffer that Percona Backup for MongoDB should use use when downloading files from the S3 storage.

Value type	Example
int int	0

backup.configuration.restore Options.download Chunk Mb

The size of the data chunk in MB, that Percona Backup for MongoDB should use when downloading from the S3 storage.

Value type	Example
■ int	32

backup.configuration.restore Options.mongod Location

The custom path to mongod binaries which Percona Backup for MongoDB should use during restore.

Value type	Example
S string	/usr/bin/mongo

backup.configuration.restore Options.mongod Location Map

The list of custom paths to mongod binaries on every node, which Percona Backup for MongoDB should use during restore.

Value type	Example
≡ subdoc	"node01:2017": /usr/bin/mongo "node03:27017": /usr/bin/mongo

backup.tasks.name

The name of the backup.

Value type	Example
S string	

backup.tasks.enabled

Enables or disables this exact backup.

Value type	Example
ு boolean	true

backup.tasks.schedule

The scheduled time to make a backup, specified in the $\underline{\text{crontab format}}$ \square .

Value type	Example
SI string	0 0 * * 6

backup.tasks.keep

The amount of most recent backups to store. Older backups are automatically deleted. Set keep to zero or completely remove it to disable automatic deletion of backups.

Value	type	Example
1 int		3

backup.tasks.storageName

The name of the S3-compatible storage for backups, configured in the storages subsection.

Value type	Example
S string	st-us-west

backup.tasks.compression Type

The backup compression format, $\underline{can\ be\ gzip,\ snappy,\ lz4,\ pgzip,\ zstd,\ s2,\ or\ none}$

Value	e type	Example
S st	tring	gzip

${\tt backup.tasks.compressionLevel}$

The backup compression level (<u>higher values result in better but slower compression</u> $\[\[\] \]).$

Value type	Example
½ int	6

backup.tasks.type

 $The \ backup \ type: (can be \ either \ logical \ (default) \ or \ physical; see \ \underline{the \ Operator \ backups \ official \ documentation}} \ for \ details.$

Value type	Example
S string	physical

11.2 Backup Resource options

A Backup resource is a Kubernetes object that tells the Operator how to create and manage your database backups. The deploy/backup/backup.yaml file is a template for creating backup resources when you make an on-demand backup. It defines the PerconaServerMongoDBBackup resource.

This document describes all available options that you can use to customize your backups.

apiVersion

Specifies the API version of the Custom Resource. psmdb.percona.com indicates the group, and v1 is the version of the API.

kind

 $\label{percona} \mbox{Defines the type of resource being created: } \mbox{PerconaServerMongoDBBackup} \ .$

metadata

The metadata part of the deploy/backup/backup.yaml contains metadata about the resource, such as its name and other attributes. It includes the following keys:

- finalizers ensure safe deletion of resources in Kubernetes under certain conditions. This subsection includes the following finalizers:
 - percona.com/delete-backup deletes the backup resource after the backup data is deleted from storage. Note that it is ignored for incremental backups.
- name The name of the backup resource used to identify it in your deployment. You also use the backup name for the restore operation.

spec

This subsection includes the configuration of a backup resource.

clusterName

Specifies the name of the MongoDB cluster to back up.

Value type	Example
s string	my-cluster-name

storageName

Specifies the name of the storage where to save a backup. It must match the name you specified in the spec.backup.storages subsection of the deploy/cr.yaml file.

Value type	Example
s string	s3-us-west

type

Specifies the backup type. Supported types are: logical, physical, incremental-base, incremental.

Value type	Example
s string	physical

${\tt compressionType}$

Specifies the compression algorithm for backups. Supported values are: gzip, pgzip, zstd, snappy. Read more about compression types in the Configure backup compression & section of PBM documentation.

Value type	Example
S string	gzip

${\tt compressionLevel}$

Specifies the compression level. Note that the higher value you specify, the more time and computing resources it will take to compress the data. The default value depends on the compression method used. Read more about compression levels in the Configure backup compression © Section of PBM documentation.

Value type	Example
S string	6

11.3 Restore Resource options

A Restore resource is a Kubernetes object that tells the Operator how to restore your database from a specific backup. The deploy/backup/restore.yaml file is a template for creating restore resources. It defines the PerconaServerMongoDBRestore resource.

This document describes all available options that you can use to customize a restore.

apiVersion

Specifies the API version of the Custom Resource. psmdb.percona.com indicates the group, and v1 is the version of the API.

kind

Defines the type of resource being created: PerconaServerMongoDBRestore.

metadata

The metadata part of the deploy/backup/restore.yaml contains metadata about the resource, such as its name and other attributes. It includes the following keys:

• name - The name of the restore object used to identify it in your deployment. You use this name to track the restore operation status and view information about it.

spec

This section includes the configuration of a restore resource.

clusterName

Specifies the name of the MongoDB cluster to restore.

Value type	Example
S string	my-cluster-name

backupName

Specifies the name of a backup to be used for a restore. This backup should be from the same cluster.

Value type	Example
s string	backup1

The selective subsection

Controls the selective restore, which enables you to restore a specific subset of namespaces - databases and collections.

selective.withUsersAndRoles

Allows restoring specified custom databases with users and roles that were created against them. Read more about <u>Selective restore with users and roles</u> in PBM documentation.

Value type	Example
S boolean	true

selective.namespaces

Specifies the list of namespaces to restore. The namespace has the format <db.collection>

Value type	Example
≡ array	["db1.collection1", "db2.collection2"]

The pitr subsection

Controls how to make a point-in-time restore

pitr.type

Specifies the type of a point-in-time restore. Available options:

- date restore to a specific date.
- latest recover to the latest possible transaction

Value type	Example
S string	date

pitr.date

Specifies the timestamp for the restore in the datetime format YYYY-MM-DD hh:mm:ss.

Use it together with the type=date option.

Value type	Example
S string	YYYY-MM-DD hh:mm:ss

The backupSource subsection

 $Contains \ the \ configuration \ options \ to \ restore \ from \ a \ backup \ made \ in \ a \ different \ cluster, \ name space, \ or \ Kubernetes \ environment.$

backupSource.type

Specifies the backup type. Available options: physical, logical, incremental

Value type	Example
S string	physical

backupSource.destination

Specifies the path to the backup on the storage

Value type	Example
S string	s3://bucket-name/backup-destination/

backupSource.s3.credentialsSecret

Specifies the Secrets object name with the credentials to access the storage with a backup.

Value type	Example
s string	my-cluster-name-backup-s3

backupSource.s3.serverSideEncryption.kmsKeyID

Specifies your customer-managed key stored in the AWS Key Management Service (AWS KMS). This key is used to encrypt backup data uploaded to S3 buckets if you don't wish to use the default server-side encryption with Amazon S3 managed keys (SSE-S3)

Value type	Example
s string	1234abcd-12ab-34cd-56ef-1234567890ab

backup Source.s 3. server Side Encryption.s se Algorithm

The encryption algorithm used to encrypt data

Valu	ue type	Example
S	string	AES256

backup Source.s 3. server Side Encryption.s se Customer Algorithm

The encryption algorithm used for server-side encryption with customer-provided keys (SSE-C).

v	alue type	Example
3	string	AES256

$backup Source.s \\ 3.server Side Encryption.sse Customer Key$

The customer-provided encryption key.

Value type	Example
string	Y3VzdG9tZXIta2V5

backupSource.s3.region

The AWS region of to use. Please note this option is mandatory for Amazon and all S3-compatible storages.

Value type	Example
S string	us-west-2

backupSource.s3.bucket

The Amazon S3 bucket anne for backups.

Value type	Example
S string	

backupSource.s3.endpointUrl

The URL of the S3-compatible storage to be used (not needed for the original Amazon S3 cloud).

Value type	Example
S string	https://s3.us-west-2.amazonaws.com/

backupSource.s3.prefix

The path to the data directory in the bucket. If undefined, backups are stored in the bucket's root directory.

Value type	Example
s string	

backupSource.azure.credentialsSecret

Specifies the Secrets object name with the credentials to access the Azure Blob storage with a backup.

Val	lue type	Example
S	string	

backupSource.azure.prefix

The path to the data directory in the bucket. If undefined, backups are stored in the bucket's root directory.

Value type	Example
s string	

backupSource.s3.container

The name of the storage container. See the <u>naming conventions</u>

Value type	Example
s string	

11.4 Percona certified images

Find Percona's certified Docker images that you can use with the Percona Operator for MongoDB in the following table:

Images released with the Operator version 1.20.0:

Image	Digest			
percona/percona-server-mongodb-operator:1.20.0 (x86_64)	01da3139b0f7f64a27f3642ca06581ea065a02891b13ce2375d61471011d6dd4			
percona/percona-server-mongodb-operator:1.20.0 (ARM64)	26d885398af42d18928f51f070aff770df900eb5ddf46e3e0bc2570720089bb1			
percona/pmm-client:2.44.1	8b2eaddffd626f02a2d5318ffebc0c277fe8457da6083b8cfcada9b6e6168616			
percona/pmm-client:2.44.1 (ARM64)	337fecd4afdb3f6daf2caa2b341b9fe41d0418a0e4ec76980c7f29be9d08b5ea			
percona/percona-backup-mongodb:2.9.1 (x86_64)	976bfbaa548eb70dd90bf0bd2dcfe40b2994d749ef644af3a0590f4856e4d7e2			
percona/percona-backup-mongodb:2.9.1 (ARM64)	ebc6e5c5aa3ed97991d3fd90e9201597b485ddc0eae8d7ee4311ecb785c03bf0			
percona/percona-server-mongodb:8.0.8-3 (x86_64)	e4580ca292f07fd7800e139121aea4b2c1dfa6aa34f3657d25a861883fd3de41			
percona/percona-server-mongodb:8.0.8-3 (ARM64)	96cfee2102499aba05e63ca7862102c2b1da1cf9f4eea0cbea3793a07c183925			
percona/percona-server-mongodb:8.0.4-1-multi (x86_64)	873b201ce3d66d97b1225c26db392c5043a73cc19ee8db6f2dc1b8efd4783bcf			
percona/percona-server-mongodb:8.0.4-1-multi (ARM64)	222ccf746ad4ffdfccf41b41edaa0d318d28f663e13c9629f8dad5a5078434e5			
percona/percona-server-mongodb:7.0.18-11 (x86_64)	0115a72f5e60d86cb4f4b7eae32118c0910e8c96831e013de12798a1771c4c91			
percona/percona-server-mongodb:7.0.18-11 (ARM64)	86c17067f3e233f522612389ed2500231cbb22ce93524c476b9aa8d464d06f0b			
percona/percona-server-mongodb:7.0.15-9-multi (x86_64)	7bffdf2e71c121e2ab37b4fa7e2f513237abdd65266da384bf8197cee1316917			
percona/percona-server-mongodb:7.0.15-9-multi (ARM64)	fdc4875df82572267445811445ebf517f63e509be54d1a2599fe58e1c525e1d8			
percona/percona-server-mongodb:7.0.14-8-multi (x86_64)	ed932d4e7231dcb793bf609f781226a8393aa8958b103339f4a503a8f70ed17e			
percona/percona-server-mongodb:7.0.14-8-multi (ARM64)	052f84ee926ad9b5146f08a7e887820342d65b757a284c2f0ea8e937bb51cd7b			
percona/percona-server-mongodb:7.0.12-7	7f00e19878bd143119772cd5468f1f0f9857dfcd2ae2f814d52ef3fa7cff6899			
percona/percona-server-mongodb:6.0.21-18 (x86_64)	579d2fdc617ea42ab2be8c2682955b489dbf49ab19771b7a5d9c77da4dd323e7			
percona/percona-server-mongodb:6.0.21-18 (ARM64)	b9d2b7e8c4a97b2d20e2aaccfbd183f65f8ccd9f2ea13939515e18e02bc64871			
percona/percona-server-mongodb:6.0.19-16-multi (x86_64)	c8ff08c4b8a96679e2daf4845873fdd4d2c48646b84db19f0c5fe02e8f3808b4			
percona/percona-server-mongodb:6.0.19-16-multi (ARM64)	6908b28ced260b762cd38a642c06dd802cbef0a43ab5f22afe7b583b234ebcec			
percona/percona-server-mongodb:6.0.18-15-multi (x86_64)	d197ce16ab0eed6df25e632b92dea5ce448e549e02028f39b78f5730c2ffef36			
percona/percona-server-mongodb:6.0.18-15-multi (ARM64)	7fd1d8f74f71dea6ad423e8e202a0617bdd1e8783f2b5cb071b5281685ce0adf			
percona/percona-server-mongodb:6.0.16-13	1497e58e39497d8425ccd053898dc323338d6eb3f0e3c4c223f9d5a468da7931			
percona/percona-server-mongodb:6.0.15-12	f12dd271d78cf3e70088fea0c420e8c03703457d8a5959b645053546bff94dea			

Image tag format

Image tags have the format:

[component_name]-[component_version]

where:

- component_name is the name of the component. For example, percona-server-mongodb
- \bullet $\,$ component_version is the version of the component. For example, $\,$ 8 . 0 . 4-1 .

Note, that PMM Client images have their own tags. They contain the version of PMM.

Find images for previous versions 🖸

11.5 Versions compatibility

Versions of the cluster components and platforms tested with different Operator releases are shown below. Other version combinations may also work but have not been tested.

Cluster components

Operator	MongoDB [ご	Percona Backup for MongoDB ☐		
1.20.0	6.0 - 8.0	2.9.1		
1.19.1	6.0 - 8.0	2.8.0		
1.19.0	6.0 - 8.0	2.8.0		
1.18.0	5.0 - 7.0	2.7.0		
1.17.0	5.0 - 7.0 2.5.0			
1.16.2	5.0 - 7.0	2.4.1		
1.16.1	5.0 - 7.0	2.4.1		
1.16.0	5.0 - 7.0	2.4.1		
1.15.0	4.4 - 6.0	2.3.0		
1.14.0	4.4 - 6.0	2.0.4, 2.0.5		
1.13.0	4.2 - 5.0	1.8.1		
<u>1.12.0</u> 4.2 - 5.0		1.7.0		
1.11.0	4.0, 4.2, 4.4, 5.0	1.6.1		
1.10.0	4.0, 4.2, 4.4, 5.0	1.6.0		
1.9.0	4.0, 4.2, 4.4	1.5.0		
1.8.0	3.6, 4.0, 4.2, 4.4	1.4.1		
1.7.0	3.6, 4.0, 4.2, 4.4	1.4.1		
1.6.0	3.6, 4.0, 4.2	1.3.4		
1.5.0	3.6, 4.0, 4.2	1.3.1		
1.4.0	3.6, 4.0, 4.2	1.1.0		
1.3.0	3.6, 4.0	0.4.0		
1.2.0	3.6, 4.0	0.4.0		
<u>1.1.0</u> 3.6, 4.0		0.4.0		

Platforms

Operator	<u>GKE</u> [♂	<u>EKS</u> ௴	Openshift 🖸	AKS [간	Minikube ☑
1.20.0	1.30 - 1.32	1.30 - 1.32	4.14 - 4.18	1.30 - 1.32	1.35.0
1.19.1	1.28 - 1.30	1.29 - 1.31	4.14.44 - 4.17.11	1.28 - 1.31	1.34.0
1.19.0	1.28 - 1.30	1.29 - 1.31	4.14.44 - 4.17.11	1.28 - 1.31	1.34.0
1.18.0	1.28 - 1.30	1.28 - 1.31	4.13.52 - 4.17.3	1.28 - 1.31	1.34.0
1.17.0	1.27 - 1.30	1.28 - 1.30	4.13.48 - 4.16.9	1.28 - 1.30	1.33.1
<u>1.16.2</u>	1.26 - 1.29	1.26 - 1.29	4.12.56 - 4.15.11	1.27 - 1.29	1.33
1.16.1	1.26 - 1.29	1.26 - 1.29	4.12.56 - 4.15.11	1.27 - 1.29	1.33
<u>1.16.0</u>	1.26 - 1.29	1.26 - 1.29	4.12.56 - 4.15.11	1.27 - 1.29	1.33
<u>1.15.0</u>	1.24 - 1.28	1.24 - 1.28	4.11 - 4.13	1.25 - 1.28	1.31.2
1.14.0	1.22 - 1.25	1.22 - 1.24	4.10 - 4.12	1.23 - 1.25	1.29
1.13.0	1.21 - 1.23	1.21 - 1.23	4.10 - 4.11	1.22 - 1.24	1.26
1.12.0	1.19 - 1.22	1.19 - 1.22	4.7 - 4.10	-	1.23
1.11.0	1.19 - 1.22	1.18 - 1.22	4.7 - 4.9	-	1.22
<u>1.10.0</u>	1.17 - 1.21	1.16 - 1.21	4.6 - 4.8	-	1.22
1.9.0	1.17 - 1.21	1.16-1.20	4.7	-	1.20
1.8.0	1.16 - 1.20	1.19	3.11, 4.7	-	1.19
1.7.0	1.15 - 1.17	1.15	3.11, 4.5	-	1.10
1.6.0	1.15 - 1.17	1.15	3.11, 4.5	-	1.10
1.5.0	1.15 - 1.17	1.15	3.11, 4.5	-	1.18
1.4.0	1.13, 1.15	1.15	3.11, 4.2	-	1.16
1.3.0	1.11, 1.14	-	3.11, 4.1	-	1.12
1.2.0	-	-	3.11, 4.0	-	-
1.1.0	-	-	3.11, 4.0	-	-

More detailed information about the cluster components for the current version of the Operator can be found in the system requirements and in the list of certified images. For previous releases of the Operator, you can check the same pages in the documentation archive.

11.6 Percona Operator for MongoDB API Documentation

Percona Operator for MongoDB provides an <u>aggregation-layer extension for the Kubernetes API</u> . Please refer to the <u>official Kubernetes API documentation</u> . To the API access and usage details. The following subsections describe the Percona XtraDB Cluster API provided by the Operator.

Prerequisites

1. Create the namespace name you will use, if not exist:

```
$ kubectl create namespace my-namespace-name
```

Trying to create an already-existing namespace will show you a self-explanatory error message. Also, you can use the defalut namespace.



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

2. Prepare:

```
# set correct API address
KUBE_CLUSTER=$(kubectl config view --minify -o jsonpath='{.clusters[0].name}')
API_SERVER=$(kubectl config view -o jsonpath="{.clusters[?(@.name==\"$KUBE_CLUSTER\")].cluster.server}" | sed -e
's#https://##')

# create service account and get token
kubectl apply --server-side -f deploy/crd.yaml -f deploy/rbac.yaml -n default
KUBE_TOKEN=$(kubectl get secret $(kubectl get serviceaccount percona-server-mongodb-operator -o
jsonpath='{.secrets[0].name}' -n default) -o jsonpath='{.data.token}' -n default | base64 --decode )
```

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

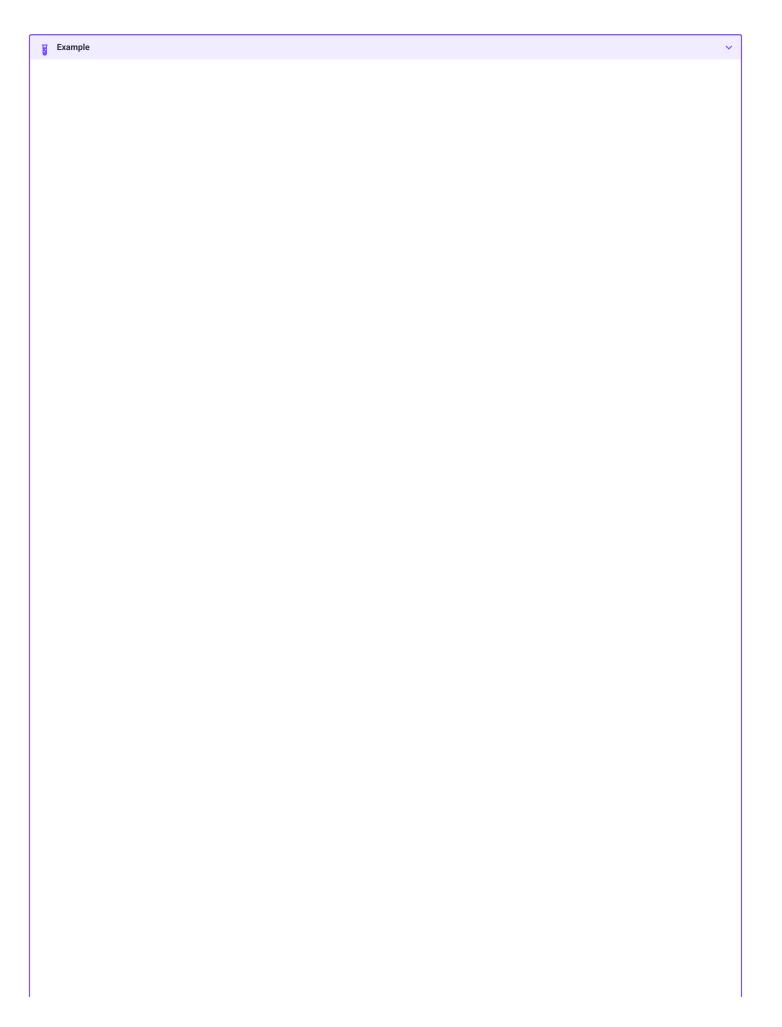
Authentication:

```
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):

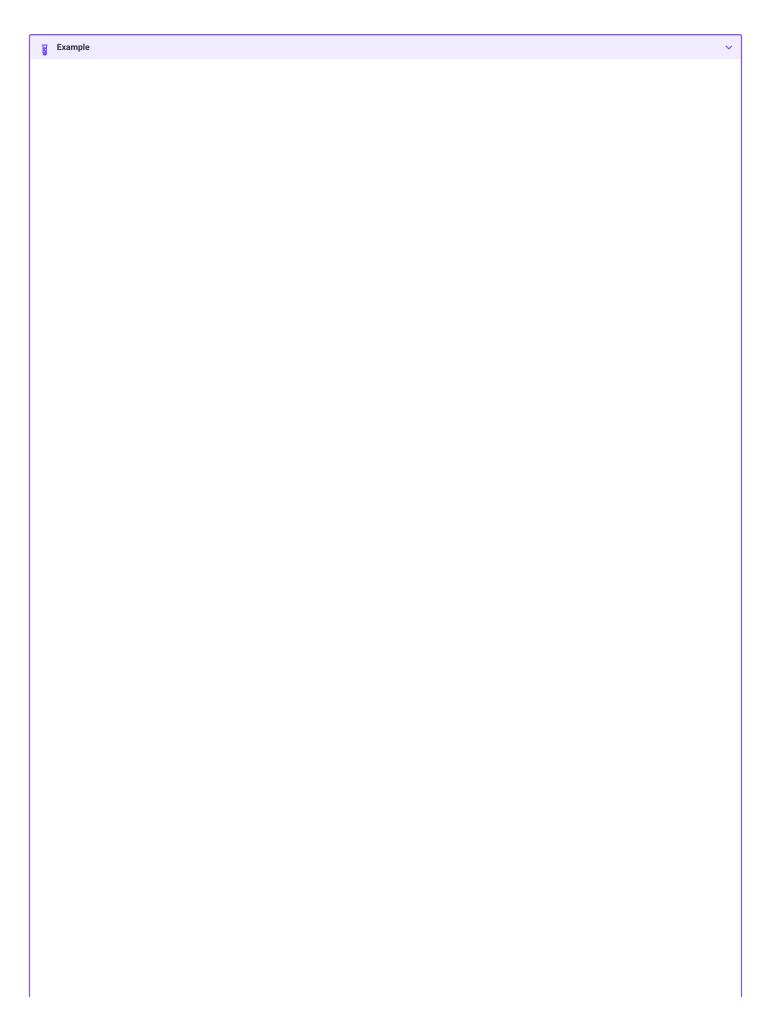


```
"apiVersion": "psmdb.percona.com/v1-5-0",
"kind": "PerconaServerMongoDB",
"metadata": {
   "name": "my-cluster-name"
"spec": {
   "image": "percona/percona-server-mongodb:4.2.8-8",
"imagePullPolicy": "Always",
"allowUnsafeConfigurations": false,
   "updateStrategy": "SmartUpdate",
   "secrets": {
    "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,
           "wired Tiger Concurrent Read Transactions": 128,\\
          "wired Tiger Concurrent Write Transactions": 128\\
       "storage": {
   "engine": "wiredTiger",
   "inMemory": {
              "engineConfig": {
                  "inMemorySizeRatio": 0.9
             }
           "mmapv1": {
              "nsSize": 16,
              "smallfiles": false
           "wiredTiger": {
              "engineConfig": {
```

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.\ enable Encryption\ (Boolean, Default: true): \ \textbf{enable}\ \textbf{encrypting}\ \textbf{mongod}\ \textbf{storage}
     b. encryptionKeySecret (String, min-length: 1): name of encryption key secret
     c. encryptionCipherMode (String, min-length: 1): type of encryption cipher to use
 3. setParameter (Object): configure mongod enginer paramters
 4. storage:
     a. engine (String, min-length: 1, default "wiredTiger"): name of mongod storage engine
     b. inMemory (Object): wiredTiger engine configuration
     c. wiredTiger (Object): wiredTiger engine configuration
pmm:
 1. serverHost (String, min-length: 1): serivce name for monitoring
 2. image (String, min-length: 1): name of pmm image
backup:
 1. image (String, min-length: 1): name of MngoDB backup docker image
 2. serviceAccountName (String, min-length: 1) name of service account to use for backup
 3. storages (Object): storage configuration object for backup
```

Response:



```
"apiVersion":"psmdb.percona.com/v1-5-0",
                   "kind":"PerconaServerMongoDB",
                   "metadata":{
                                "annotations":{
                                              "kubectl.kubernetes.io/last-applied-configuration":"{\normaline} ``" {\normaline} ``" {\n
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 \verb|\|", \verb|\|"restartOnFailure||":true, \verb|\|"serviceAccountName||": \verb|\|"percona-server-mongodb-leaves | leaves |
 operator'",\"storages\":null,\"tasks\":null},\"image\":\"percona/percona-server-mongodb:4.2.8-8\",\"imagePullPolicy\":\"Always\",\"mongod\":
{\"net\":{\"hostPort\":0,\"port\":27017},\"operationProfiling\":\"slowOp\",\"rateLimit\":100,\"slowOpThresholdMs\":100},\"security\":
{\"enableEncryption\":true,\"encryptionCipherMode\":\"AES256-CBC\",\"encryptionKeySecret\":\"my-cluster-name-mongodb-encryption-
  key\",\"redactClientLogData\":false},\"setParameter\":
  {\"tlMonitorSleepSecs\":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\":
    [\ \{\ "affinity\ ": \{\ "antiAffinity\ TopologyKey\ ":\ "none\ "\},\ "arbiter\ ": \{\ "affinity\ ":\ "affinity\ ":\ "affinity\ ":\ "arbiter\ ":\ "affinity\ ":\ "affinity\ ":\ "arbiter\ ":\ "affinity\ ":\ "arbiter\ ":\ "affinity\ ":\ "affinity\ ":\ "arbiter\ ":\ "arbiter\ ":\ "affinity\ ":\ "arbiter\ ":\ "a
  {\"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\":\"36i\"}},\"storageClassName\":\"standard\"}}],\"secrets\":\"imy-cluster-name-
  secrets\"},\"updateStrategy\":\"SmartUpdate\"}}\n"
                                "creationTimestamp":"2020-07-24T14:27:58Z",
                                 "generation":1,
                                 "managedFields":[
                                                           "apiVersion":"psmdb.percona.com/v1-5-0",
"fieldsType":"FieldsV1",
                                                              "fieldsV1":{
                                                                             "f:metadata":{
                                                                                         "f:annotations":{
                                                                                                         ".":{
                                                                                                       },
"f:kubectl.kubernetes.io/last-applied-configuration":{
                                                                                        }
                                                                        },
"f:spec":{
                                                                                          ".":{
                                                                                           "f:allowUnsafeConfigurations":{
                                                                                            f:backup":{
                                                                                                       ".":{
                                                                                                      },
"f:enabled":{
                                                                                                          "f:image":{
                                                                                                       },
"f:restartOnFailure":{
                                                                                                        },
"f:serviceAccountName":{
                                                                                                       },
"f:storages":{
                                                                                                             f:tasks":{
                                                                                       },
"f:image":{
                                                                                        },
"f:imagePullPolicy":{
                                                                                              f:mongod":{
                                                                                                            '.":{
                                                                                                        },
"f:net":{
                                                                                                                        f:hostPort":{
```

```
},
"f:port":{
},
"f:operationProfiling":{
   ".":{
   },
"f:mode":{
   },
"f:rateLimit":{
   },
"f:slowOpThresholdMs":{
},
"f:security":{
   },
"f:enableEncryption":{
   },
"f:encryptionCipherMode":{
   },
"f:encryptionKeySecret":{
   },
"f:redactClientLogData":{
},
"f:setParameter":{
    ".":{
   },
"f:ttlMonitorSleepSecs":{
    },
"f:wiredTigerConcurrentReadTransactions":{
   },
"f:wiredTigerConcurrentWriteTransactions":{
},
"f:storage":{
   ".":{
   },
"f:engine":{
   },
"f:inMemory":{
    ".":{
       },
"f:engineConfig":{
   ".":{
         },
"f:inMemorySizeRatio":{
   },
"f:mmapv1":{
    ".":{
       },
"f:nsSize":{
       },
"f:smallfiles":{
   },
"f:wiredTiger":{
        ".":{
       },
"f:collectionConfig":{
```

```
},
"f:blockCompressor":{
                               },
"f:engineConfig":{
                                     ".":{
                                    },
"f:cacheSizeRatio":{
                                    },
"f:directoryForIndexes":{
                                    },
"f:journalCompressor":{
                                },
"f:indexConfig":{
                                    ".":{
                                    },
"f:prefixCompression":{
                   },
"f:pmm":{
    " ":{
                       },
"f:enabled":{
                       },
"f:image":{
                       },
"f:serverHost":{
                   },
"f:replsets":{
                    },
"f:secrets":{
                        ".":{
                       },
"f:users":{
                    "f:updateStrategy":{
               }
           },
"manager":"kubectl",
"operation":"Update",
           "time":"2020-07-24T14:27:58Z"
   "name":"my-cluster-name",
"namespace":"default",
"resourceVersion":"1268922",
"selfLink":"/apis/psmdb.percona.com/v1-5-0/namespaces/default/perconaservermongodbs/my-cluster-name",
"uid":"5207e71a-c83f-4707-b892-63aa93fb615c"
},
"spec":{
    "allowUnsafeConfigurations":false,
    "backup":{
       "enabled":true,
       "image": "percona/percona-server-mongodb-operator:1.5.0-backup",
       "restartOnFailure":true,
       "service Account Name": "percona-server-mongodb-operator",\\
       "storages":null,
"tasks":null
   },
"image":"percona/percona-server-mongodb:4.2.8-8",
"imagePullPolicy":"Always",
    "mongod":{
    "net":{
           "hostPort":0,
```

```
"operationProfiling":{
        "mode":"slowOp",
"rateLimit":100,
        "slowOpThresholdMs":100
    "security":{
       "enableEncryption":true,
"encryptionCipherMode":"AES256-CBC",
"encryptionKeySecret":"my-cluster-name-mongodb-encryption-key",
        "redact {\tt ClientLogData":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
   }
    "enabled":false,
    "image":"percona/percona-server-mongodb-operator:1.5.0-pmm",
    "serverHost":"monitoring-service"
},
"replsets":[
       "affinity":{
    "antiAffinityTopologyKey":"none"
        "arbiter":{
               "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"
}
}
```

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:

Authentication:

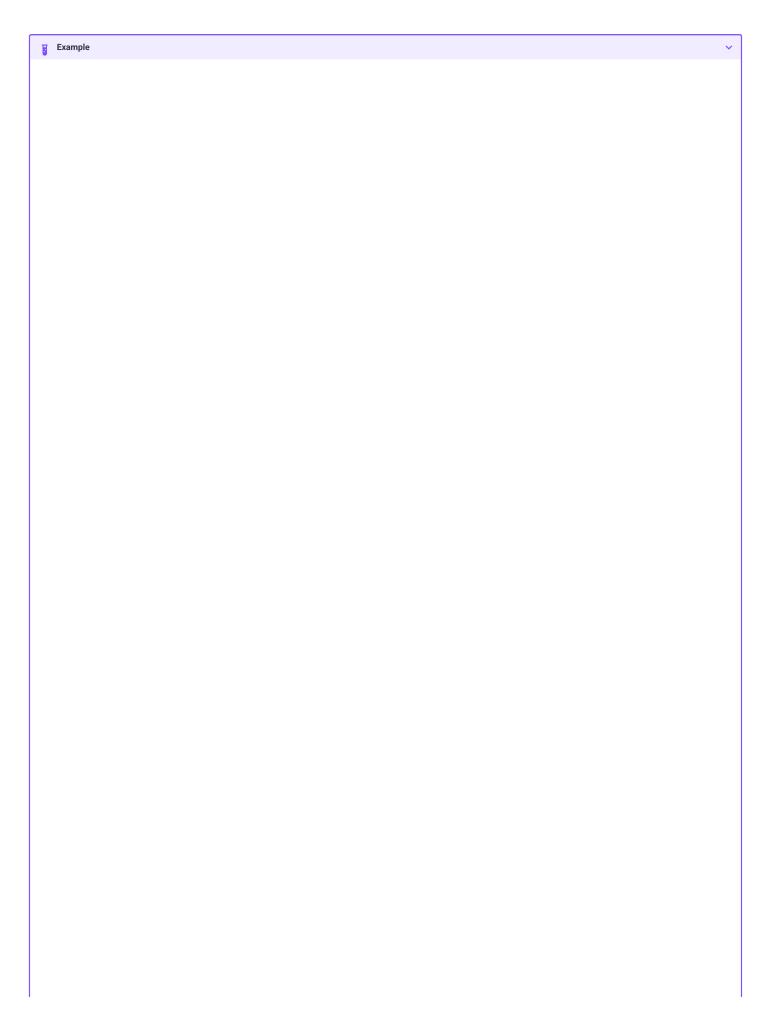
Authorization: Bearer \$KUBE_TOKEN

cURL Request:

Request Body:

None

Response:



```
"kind":"Table",
       apiVersion":"meta.k8s.io/v1".
       "metadata":{
             selfLink":"/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs",
            "resourceVersion":"1273793"
       "columnDefinitions":[
                 "name": "Name"
                  "type":"string"
                 "format":"name"
                 "description":"Name must be unique within a namespace. Is required when creating resources, although some resources may allow a client
to request the generation of an appropriate name automatically. Name is primarily intended for creation idempotence and configuration
definition. Cannot be updated. More info: http://kubernetes.io/docs/user-guide/identifiers#names",
                 "priority":0
                 "name": "Status",
                 "type":"string",
                 "format":
                 "description": "Custom resource definition column (in JSONPath format): .status.state",
                 "priority":0
                 "name":"Age"
                 "type": "date",
                 "description":"Custom resource definition column (in JSONPath format): .metadata.creationTimestamp",
       rows":[
                "cells":[
                        "my-cluster-name",
                      "ready",
                      "37m
                        "kind":"PartialObjectMetadata",
                       "apiVersion":"meta.k8s.io/v1",
                       "metadata":{
                            "name":"my-cluster-name",
                            "namespace":"default",
"selfLink":"/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs/my-cluster-name",
                            "uid":"5207e71a-c83f-4707-b892-63aa93fb615c",
                            "resourceVersion":"1273788",
                            "generation":1.
                            "creationTimestamp":"2020-07-24T14:27:58Z",
                                  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-
key\",\"redactClientLogData\":false},\"setParameter\'
\label{thm:continuous} \begin{tabular}{ll} $$ \ \ '''' ttlMonitorSleepSecs''':60, ''wiredTigerConcurrentReadTransactions''':128, \''wiredTigerConcurrentWriteTransactions''':128, \''wiredTigerConcurrentWriteTransactions'''':128, \''
{\"engine\":\"wiredTiger\",\"inMemory\":{\"engineConfig\":{\"inMemorySizeRatio\":0.9}},\"mmapv1\":
{\"nsSize\":16,\"smallfiles\":false},\"wiredTiger\":{\"collectionConfig\":{\"blockCompressor\":\"snappy\"},\"engineConfig\":
 \\"cacheSizeRatio\":0.5,\"directoryForIndexes\":false,\"journalCompressor\":\"snappy\"},\"indexConfiq\":{\"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\":
{\"anabled\":false,\"exposeType\":\"LoadBalancer\"},\"name\":\"rs0\",\"podDisruptionBudget\":\\"maxUnavailable\":1},\"resources\":
{\"limits\":null},\"size\":3,\"volumeSpec\":\\"persistentVolumeClaim\":\\"accessModes\":\\"ReadWriteOnce\"],\"resources\":\\"requests\":\\"storage\":\"3Gi\"}},\"storageClassName\":\"standard\"}}],\"secrets\":\\"users\":\\"my-cluster-name-
secrets\"},\"updateStrategy\":\"SmartUpdate\"}}\n'
                              'managedFields":[
                                       "manager": "kubectl",
                                       "operation":"Update",
"apiVersion":"psmdb.percona.com/v1-5-0",
                                        "time": "2020-07-24T14:27:58Z",
                                         fieldsType":"FieldsV1".
                                       "fieldsV1":{
                                              f:metadata":{
                                                  "f:annotations":{
                                                          .":{
                                                        "f:kubectl.kubernetes.io/last-applied-configuration":{
```

```
},
"f:spec":{
   ".":{
    },
"f:allowUnsafeConfigurations":{
   },
"f:backup":{
    ".":{
       },
"f:enabled":{
       },
"f:image":{
        },
"f:serviceAccountName":{
    },
"f:image":{
    },
"f:imagePullPolicy":{
   },
"f:mongod":{
    ".":{
       },
"f:net":{
    ".":{
           },
"f:port":{
        },
"f:operationProfiling":{
           },
"f:mode":{
           },
"f:rateLimit":{
           },
"f:slowOpThresholdMs":{
       },
"f:security":{
           ".":{
           },
"f:enableEncryption":{
           },
"f:encryptionCipherMode":{
           },
"f:encryptionKeySecret":{
       },
"f:setParameter":{
   ".":{
           },
"f:ttlMonitorSleepSecs":{
           },
"f:wiredTigerConcurrentReadTransactions":{
           },
"f:wiredTigerConcurrentWriteTransactions":{
       },
"f:storage":{
   ".":{
           },
"f:engine":{
```

```
},
"f:inMemory":{
   ".":{
                       },
"f:engineConfig":{
   ".":{
                             },
"f:inMemorySizeRatio":{
                   },
"f:mmapv1":{
    ".":{
                        },
"f:nsSize":{
                   },
"f:wiredTiger":{
    ".":{
                        },
"f:collectionConfig":{
   ".":{
                             },
"f:blockCompressor":{
                        },
"f:engineConfig":{
   ".":{
                            },
"f:cacheSizeRatio":{
                             },
"f:journalCompressor":{
                       },
"f:indexConfig":{
   ".":{
                             },
"f:prefixCompression":{
              },
"f: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":{
) } } 
1
```

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:

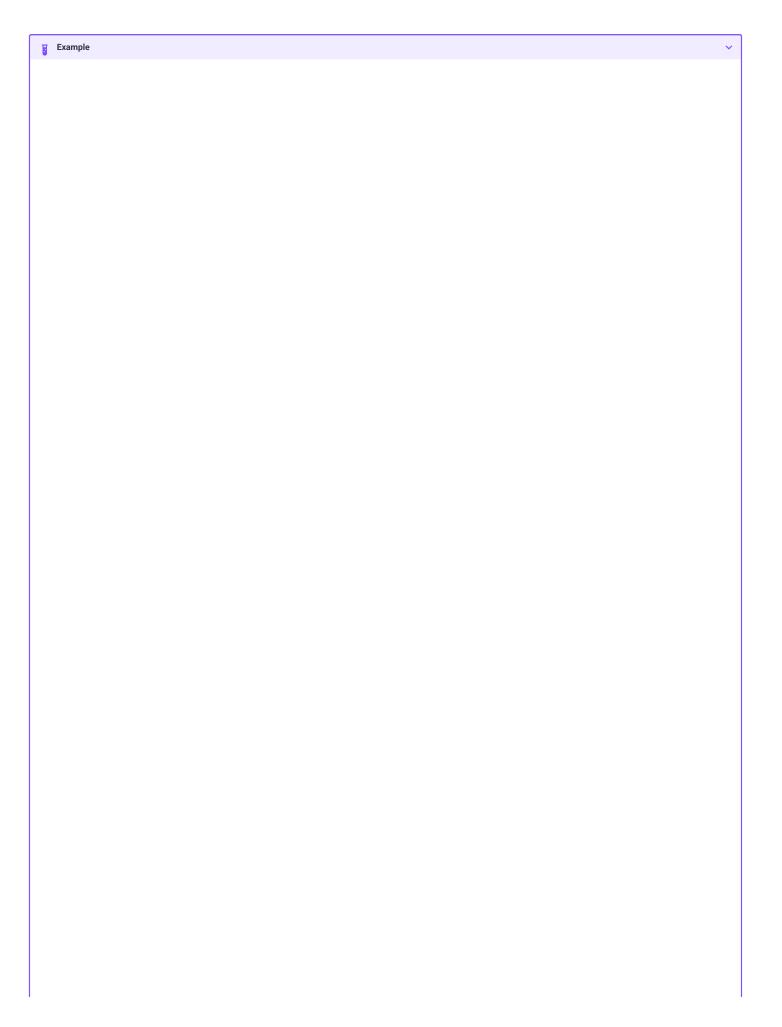
Authentication:

Authorization: Bearer \$KUBE_TOKEN

cURL Request:

Request Body:

None



```
"apiVersion": "psmdb.percona.com/v1",
                   "kind":"PerconaServerMongoDB",
                   "metadata":{
                                "annotations":{
                                               "kubectl.kubernetes.io/last-applied-configuration":"{\normaline} ``" {\normaline} ``" {\n
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 \verb|\|", \verb|\|"restartOnFailure|":true, \verb|\|"serviceAccountName|": \verb|\|"percona-server-mongodb-leading of the control of
 operator'",\"storages\":null,\"tasks\":null},\"image\":\"percona/percona-server-mongodb:4.2.8-8\",\"imagePullPolicy\":\"Always\",\"mongod\":
{\"net\":{\"hostPort\":0,\"port\":27017},\"operationProfiling\":\"slowOp\",\"rateLimit\":100,\"slowOpThresholdMs\":100},\"security\":
{\"enableEncryption\":true,\"encryptionCipherMode\":\"AES256-CBC\",\"encryptionKeySecret\":\"my-cluster-name-mongodb-encryption-
  key\",\"redactClientLogData\":false},\"setParameter\":
  {\"tlMonitorSleepSecs\":60,\"wiredTigerConcurrentReadTransactions\":128,\"wiredTigerConcurrentWriteTransactions\":128},\"storage\":
{\"engine\":\"wiredTiger\",\"inMemory\":{\"inMemorySizeRatio\":0.9}},\"mmapv1\":
{\"nsSize\":16,\"smallfiles\":false},\"wiredTiger\":{\"collectionConfig\":{\"blockCompressor\":\"snappy\"},\"engineConfig\":
    {\"cacheSizeRatio\":0.5,\"directoryForIndexes\":false,\"journalCompressor\":\"snappy\"},\"indexConfig\":{\"prefixCompression\":true}}}},\"pmm\":
    [\ \{\ "affinity\ ": \{\ "antiAffinity\ TopologyKey\ ":\ "none\ "\},\ "arbiter\ ": \{\ "affinity\ ":\ "affinity\ ":\ "affinity\ ":\ "arbiter\ ":\ "affinity\ ":\ "affinity\ ":\ "arbiter\ ":\ "arbiter\ ":\ "affinity\ ":\ "arbiter\ ":\ "
  {\"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\":\"36i\"}},\"storageClassName\":\"standard\"}}],\"secrets\":\"imy-cluster-name-
  \verb|secrets|"}, \verb|"updateStrategy|": \verb|"SmartUpdate|"}| \verb|n"|
                                "creationTimestamp":"2020-07-24T14:27:58Z",
                                 "generation":1,
                                 "managedFields":[
                                                           "apiVersion":"psmdb.percona.com/v1-5-0",
"fieldsType":"FieldsV1",
                                                              "fieldsV1":{
                                                                             "f:metadata":{
                                                                                          "f:annotations":{
                                                                                                          ".":{
                                                                                                       },
"f:kubectl.kubernetes.io/last-applied-configuration":{
                                                                                         }
                                                                         },
"f:spec":{
                                                                                           ".":{
                                                                                            "f:allowUnsafeConfigurations":{
                                                                                            f:backup":{
                                                                                                       ".":{
                                                                                                          "f:enabled":{
                                                                                                           "f:image":{
                                                                                                          "f:serviceAccountName":{
                                                                                                      }
                                                                                           "f:image":{
                                                                                         },
"f:imagePullPolicy":{
                                                                                        },
"f:mongod":{
                                                                                                        ".":{
                                                                                                            "f:net":{
                                                                                                                      },
"f:port":{
                                                                                                              f:operationProfiling":{
                                                                                                                     },
"f:mode":{
```

```
},
"f:rateLimit":{
    },
"f:slowOpThresholdMs":{
},
"f:security":{
    ".":{
    },
"f:enableEncryption":{
    },
"f:encryptionCipherMode":{
    },
"f:encryptionKeySecret":{
},
"f:setParameter":{
    ".":{
    },
"f:ttlMonitorSleepSecs":{
    },
"f:wiredTigerConcurrentReadTransactions":{
    },
"f:wiredTigerConcurrentWriteTransactions":{
},
"f:storage":{
    ".":{
    },
"f:engine":{
    },
"f:inMemory":{
       },
"f:engineConfig":{
   ".":{
          },
"f:inMemorySizeRatio":{
   },
"f:mmapv1":{
    ".":{
       },
"f:nsSize":{
    },
"f:wiredTiger":{
       },
"f:collectionConfig":{
   ".":{
           },
"f:blockCompressor":{
       },
"f:engineConfig":{
          },
"f:cacheSizeRatio":{
           },
"f:journalCompressor":{
       },
"f:indexConfig":{
```

```
},
"f:prefixCompression":{
           },
"f: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,
      \verb|"image":"percona/percona-server-mongodb-operator: 1.5.0-backup",\\
      "restartOnFailure":true,
      "service Account Name": "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,
     \verb|"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",
             "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"
}
```

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:

 $https://\$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs/my-cluster-name. The property of the$

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

Request Body:

```
Example

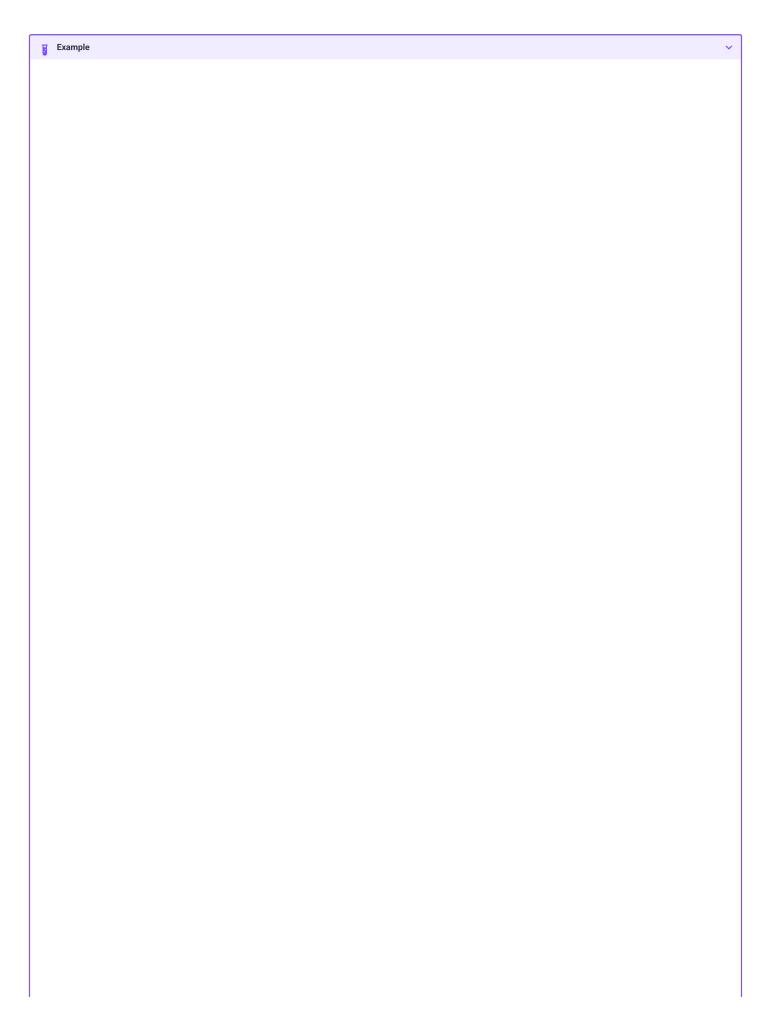
{
    "spec": {"replsets":{ "size": "5" }
}}
```

Input:

spec:

replsets

1. size (Int or String, Defaults: 3): Specifiy the sie of the replsets cluster to scale up or down to



```
"apiVersion": "psmdb.percona.com/v1",
                    "kind":"PerconaServerMongoDB",
                    "metadata":{
                                "annotations":{
                                               "kubectl.kubernetes.io/last-applied-configuration":"{\normaline} ``" {\normaline} ``" {\n
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 \verb|\|", \verb|\|"restartOnFailure|":true, \verb|\|"serviceAccountName|": \verb|\|"percona-server-mongodb-leading of the control of
 operator'",\"storages\":null,\"tasks\":null},\"image\":\"percona/percona-server-mongodb:4.2.8-8\",\"imagePullPolicy\":\"Always\",\"mongod\":
{\"net\":{\"hostPort\":0,\"port\":27017},\"operationProfiling\":\"slowOp\",\"rateLimit\":100,\"slowOpThresholdMs\":100},\"security\":
{\"enableEncryption\":true,\"encryptionCipherMode\":\"AES256-CBC\",\"encryptionKeySecret\":\"my-cluster-name-mongodb-encryption-
  key\",\"redactClientLogData\":false},\"setParameter\":
  {\"tlMonitorSleepSecs\":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\":
    [\ \{\ "affinity\ ": \{\ "antiAffinity\ TopologyKey\ ":\ "none\ "\}, \ "arbiter\ ": \{\ "affinity\ ": \{, \ "af
  {\"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\":\"36i\"}},\"storageClassName\":\"standard\"}}],\"secrets\":\"imy-cluster-name-
  secrets\"},\"updateStrategy\":\"SmartUpdate\"}}\n"
                                "creationTimestamp":"2020-07-24T14:27:58Z",
                                 "generation":4,
                                 "managedFields":[
                                                            "apiVersion":"psmdb.percona.com/v1-5-0",
"fieldsType":"FieldsV1",
                                                               "fieldsV1":{
                                                                              "f:metadata":{
                                                                                           "f:annotations":{
                                                                                                           ".":{
                                                                                                         },
"f:kubectl.kubernetes.io/last-applied-configuration":{
                                                                                          }
                                                                          },
"f:spec":{
                                                                                            ".":{
                                                                                             "f:allowUnsafeConfigurations":{
                                                                                             f:backup":{
                                                                                                         ".":{
                                                                                                        },
"f:enabled":{
                                                                                                            "f:image":{
                                                                                                           "f:serviceAccountName":{
                                                                                                        }
                                                                                            "f:image":{
                                                                                          },
"f:imagePullPolicy":{
                                                                                         },
"f:mongod":{
                                                                                                          ".":{
                                                                                                             "f:net":{
                                                                                                                       },
"f:port":{
                                                                                                                f:operationProfiling":{
                                                                                                                      },
"f:mode":{
```

```
},
"f:rateLimit":{
    },
"f:slowOpThresholdMs":{
},
"f:security":{
    ".":{
    },
"f:enableEncryption":{
    },
"f:encryptionCipherMode":{
    },
"f:encryptionKeySecret":{
},
"f:setParameter":{
    },
"f:ttlMonitorSleepSecs":{
    },
"f:wiredTigerConcurrentReadTransactions":{
    },
"f:wiredTigerConcurrentWriteTransactions":{
},
"f:storage":{
    ".":{
    },
"f:engine":{
    },
"f:inMemory":{
       },
"f:engineConfig":{
   ".":{
          },
"f:inMemorySizeRatio":{
   },
"f:mmapv1":{
    ".":{
       },
"f:nsSize":{
    },
"f:wiredTiger":{
       },
"f:collectionConfig":{
   ".":{
           },
"f:blockCompressor":{
       },
"f:engineConfig":{
          },
"f:cacheSizeRatio":{
           },
"f:journalCompressor":{
       },
"f:indexConfig":{
```

```
},
"f:prefixCompression":{
           },
"f: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"
       1.
       "name":"my-cluster-name",
"namespace":"default",
       "resourceVersion":"1279009",
       "selfLink": "/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbs/my-cluster-name", and the contract of the c
       "uid":"5207e71a-c83f-4707-b892-63aa93fb615c"
"spec":{
       "allow Unsafe Configurations": 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":{
   "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":{
    "redy":3,
    "size":3,
    "size":3,
    "status":"ready"
}
},
"status":"ready"
}
}
```

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:

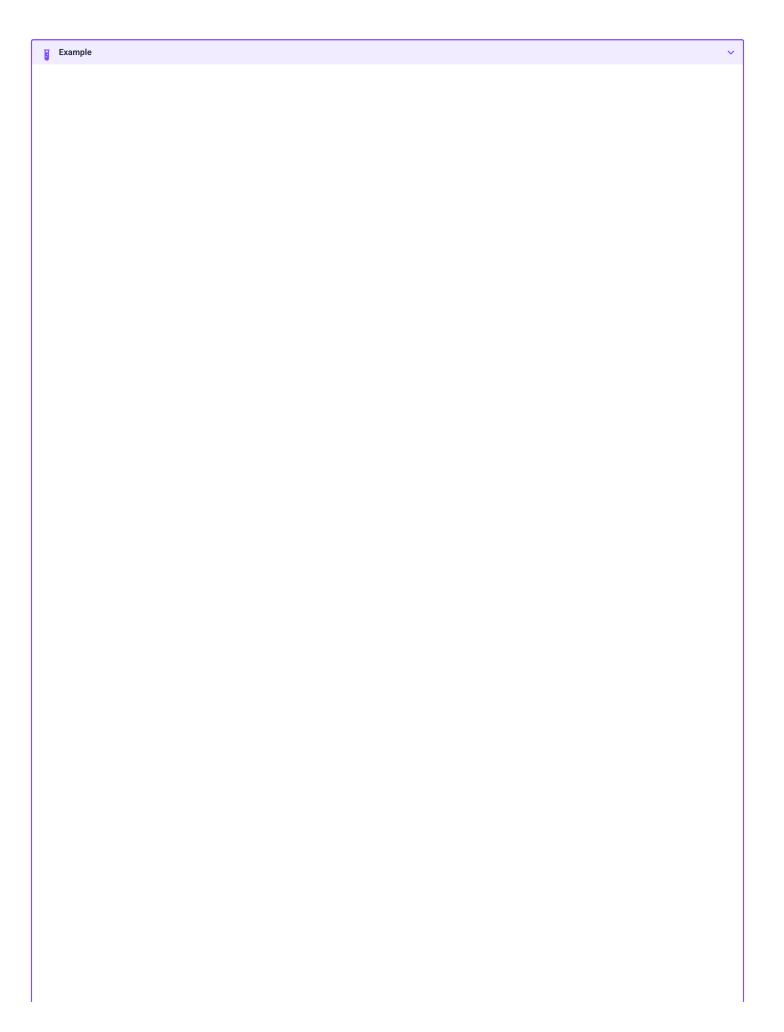
Request Body:

```
{
    "spec": { "image ": "percona/percona-server-mongodb:4.2.8-8" }
}
```

Input:

spec:
psmdb:

 $1.\ image\ (String, min-length: 1): name\ of\ the\ image\ to\ update\ for\ Percona\ Server\ for\ MongoDB$



```
"apiVersion": "psmdb.percona.com/v1",
                   "kind":"PerconaServerMongoDB",
                   "metadata":{
                                "annotations":{
                                               "kubectl.kubernetes.io/last-applied-configuration":"{\normaline} ``" {\normaline} ``" {\n
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 \verb|\|", \verb|\|"restartOnFailure|":true, \verb|\|"serviceAccountName|": \verb|\|"percona-server-mongodb-leading of the control of
 operator'",\"storages\":null,\"tasks\":null},\"image\":\"percona/percona-server-mongodb:4.2.8-8\",\"imagePullPolicy\":\"Always\",\"mongod\":
{\"net\":{\"hostPort\":0,\"port\":27017},\"operationProfiling\":\"slowOp\",\"rateLimit\":100,\"slowOpThresholdMs\":100},\"security\":
{\"enableEncryption\":true,\"encryptionCipherMode\":\"AES256-CBC\",\"encryptionKeySecret\":\"my-cluster-name-mongodb-encryption-
  key\",\"redactClientLogData\":false},\"setParameter\":
  {\"tlMonitorSleepSecs\":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\":
    [\ \{\ "affinity\ ": \{\ "antiAffinity\ TopologyKey\ ":\ "none\ "\},\ "arbiter\ ": \{\ "affinity\ ":\ "affinity\ ":\ "affinity\ ":\ "arbiter\ ":\ "affinity\ ":\ "affinity\ ":\ "arbiter\ ":\ "arbiter\ ":\ "affinity\ ":\ "arbiter\ ":\ "
  {\"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\":\"36i\"}},\"storageClassName\":\"standard\"}}],\"secrets\":\"imy-cluster-name-
  secrets\"},\"updateStrategy\":\"SmartUpdate\"}}\n"
                                "creationTimestamp":"2020-07-24T14:27:58Z",
                                 "generation":5,
                                 "managedFields":[
                                                           "apiVersion":"psmdb.percona.com/v1-5-0",
"fieldsType":"FieldsV1",
                                                              "fieldsV1":{
                                                                             "f:metadata":{
                                                                                          "f:annotations":{
                                                                                                         ".":{
                                                                                                       },
"f:kubectl.kubernetes.io/last-applied-configuration":{
                                                                                         }
                                                                         },
"f:spec":{
                                                                                           ".":{
                                                                                            "f:allowUnsafeConfigurations":{
                                                                                            f:backup":{
                                                                                                       ".":{
                                                                                                      },
"f:enabled":{
                                                                                                          "f:image":{
                                                                                                         "f:serviceAccountName":{
                                                                                                      }
                                                                                           "f:image":{
                                                                                         },
"f:imagePullPolicy":{
                                                                                        },
"f:mongod":{
                                                                                                        ".":{
                                                                                                            "f:net":{
                                                                                                                      },
"f:port":{
                                                                                                               f:operationProfiling":{
                                                                                                                     },
"f:mode":{
```

```
},
"f:rateLimit":{
    },
"f:slowOpThresholdMs":{
},
"f:security":{
    ".":{
    },
"f:enableEncryption":{
    },
"f:encryptionCipherMode":{
    },
"f:encryptionKeySecret":{
},
"f:setParameter":{
    ".":{
    },
"f:ttlMonitorSleepSecs":{
    },
"f:wiredTigerConcurrentReadTransactions":{
    },
"f:wiredTigerConcurrentWriteTransactions":{
},
"f:storage":{
    ".":{
    },
"f:engine":{
    },
"f:inMemory":{
       },
"f:engineConfig":{
   ".":{
          },
"f:inMemorySizeRatio":{
   },
"f:mmapv1":{
    ".":{
       },
"f:nsSize":{
    },
"f:wiredTiger":{
       },
"f:collectionConfig":{
   ".":{
           },
"f:blockCompressor":{
       },
"f:engineConfig":{
          },
"f:cacheSizeRatio":{
           },
"f:journalCompressor":{
       },
"f:indexConfig":{
```

```
},
"f:prefixCompression":{
           },
"f: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", and the control of the co
           "uid":"5207e71a-c83f-4707-b892-63aa93fb615c'
},
"spec":{
    "-110
           "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,
          "wired Tiger Concurrent Write Transactions": 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",
```

Backup Percona Server for MongoDB cluster

Description:

Takes a backup of the Percona Server for MongoDB cluster containers data to be able to recover from disasters or make a roll-back later

Kubectl Command:

```
$ kubectl apply -f percona-server-mongodb-operator/deploy/backup/backup.yaml
```

URL:

 $https://\$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbbackups$

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

```
$ curl -k -v -XPOST "https://$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbbackups" \
    -H "Accept: application/json" \
    -H "Content-Type: application/json" \
    -d "@backup.json" -H "Authorization: Bearer $KUBE_TOKEN"
```

Request Body (backup.json):

```
{
    "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:

```
    psmdbCluster(String, min-length:1): `name of Percona Server for MongoDB cluster`
    storageName(String, min-length:1): `name of storage claim to use`
```

Response:

```
Example
     "apiVersion":"psmdb.percona.com/v1",
     "kind":"PerconaServerMongoDBBackup",
     "metadata":{
        "annotations":{
            "kubectl.kubernetes.io/last-applied-configuration":"
 {\"apiVersion\":\"psmdb.percona.com/v1\",\"kind\":\"PerconaServerMongoDBBackup\",\"metadata\":\"annotations\":
{},\"name\":\"backup1\",\"namespace\":\"default\"},\"spec\":\"yperdonaServerMongoDBBackup\",\"my-cluster-name\",\"storageName\":\"s3-us-west\"}}\n"
         "creationTimestamp":"2020-07-27T13:45:43Z",
        "generation":1,
         "managedFields":[
                "apiVersion":"psmdb.percona.com/v1",
               "fieldsType":"FieldsV1",
                "fieldsV1":{
                   "f:metadata":{
                      "f:annotations":{
                          ".":{
                          "f:kubectl.kubernetes.io/last-applied-configuration":{
                      }
                  },
"f:spec":{
".;
                       ".":{
                      },
"f:psmdbCluster":{
                       "f:storageName":{
                "manager":"kubectl",
                "operation":"Update"
               "time":"2020-07-27T13:45:43Z"
         "name":"backup1",
        "namespace":"default"
        "resourceVersion":"1290243",
        "selfLink": "/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbbackups/backup1", \\
        "uid":"e695d1c7-898e-44b0-b356-537284f6c046"
      "spec":{
        "psmdbCluster": "my-cluster-name",
         "storageName":"s3-us-west"
```

Restore Percona Server for MongoDB cluster

Description:

Restores Percona Server for MongoDB cluster data to an earlier version to recover from a problem or to make a roll-back

Kubectl Command:

```
$ kubectl apply -f percona-server-mongodb-operator/deploy/backup/restore.yaml
```

URL:

 $https://\$API_SERVER/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbrestores$

Authentication:

```
Authorization: Bearer $KUBE_TOKEN
```

cURL Request:

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:

```
    clusterName(String, min-length:1): `name of Percona Server for MongoDB cluster`
    backupName(String, min-length:1): `name of backup to restore from`
```

```
Example
      "apiVersion":"psmdb.percona.com/v1",
"kind":"PerconaServerMongoDBRestore",
      "metadata":{
          "annotations":{
              "kubectl.kubernetes.io/last-applied-configuration":"
 {\"apiVersion\":\"psmdb.percona.com/v1\",\"hind\":\"PerconaServerMongoDBRestore\",\"metadata\":{\"annotations\":
{},\"name\":\"restore1\",\"namespace\":\"default\"},\"spec\":\"backupName\":\"backup1\",\"clusterName\":\"my-cluster-name\"}\n"
          "creationTimestamp":"2020-07-27T13:52:56Z",  
          "generation":1,
          "managedFields":[
                  "apiVersion":"psmdb.percona.com/v1",
"fieldsType":"FieldsV1",
                  "fieldsV1":{
                      "f:metadata":{
                          "f:annotations":{
                              ".":{
                              },
"f:kubectl.kubernetes.io/last-applied-configuration":{
                      },
"f:spec":{
                            ·.":{
                          },
"f:backupName":{
                          },
"f:clusterName":{
                     }
                  "manager":"kubectl",
"operation":"Update"
                  "time":"2020-07-27T13:52:56Z"
             }
          "name":"restore1",
"namespace":"default"
         "resourceVersion":"1291198",
"selfLink":"/apis/psmdb.percona.com/v1/namespaces/default/perconaservermongodbrestores/restore1",
"uid":"17e982fe-ac41-47f4-afba-fea380b0c76e"
      "spec":{
          "backupName":"backup1",
          "clusterName": "my-cluster-name"
```

11.7 Frequently Asked Questions

Why do we need to follow "the Kubernetes way" when Kubernetes was never intended to run databases?

As it is well known, the Kubernetes approach is targeted at stateless applications but provides ways to store state (in Persistent Volumes, etc.) if the application needs it. Generally, a stateless mode of operation is supposed to provide better safety, sustainability, and scalability, it makes the already-deployed components interchangeable. You can find more about substantial benefits brought by Kubernetes to databases in this blog post t.

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.

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 [4].

If you would like to report a bug, use the Percona Operator for MongoDB project in JIRA [].

What is the difference between the Operator quickstart and advanced installation ways?

As you have noticed, the installation section of docs contains both quickstart and advanced installation guides.

The quickstart guide is simpler. It has fewer installation steps in favor of predefined default choices. Particularly, in advanced installation guides, you separately apply the Custom Resource Definition and Role-based Access Control configuration files with possible edits in them. At the same time, quickstart guides rely on the all-inclusive bundle configuration.

At another point, quickstart guides are related to specific platforms you are going to use (Minikube, Google Kubernetes Engine, etc.) and therefore include some additional steps needed for these platforms.

Generally, rely on the quickstart guide if you are a beginner user of the specific platform and/or you are new to the Percona Operator for MongoDB as a whole.

Which versions of MongoDB does the Operator support?

Percona Operator for MongoDB works with Percona Server for MongoDB 5.0, 6.0, and 7.0, and the exact version is determined by the Docker image in use.

Percona-certified Docker images that can be used by the Operator are listed <u>here</u>. For example, Percona Server for MongoDB 6.0 is supported with the following recommended version: 6.0.21-18. More details on the exact Percona Server for MongoDB version can be found in the release notes (5.0 5, 6.0 5, and 7.0 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
   ....
```

 $You \ can \ add \ sidecars \ subsection \ to \ replsets, \ sharding.configsvrReplSet, and \ sharding.mongos \ sections.$



Custom sidecar containers can easily access other components of your cluster [2]. Therefore they should be used carefully and by experienced users only.

Find more information on sidecar containers in the appropriate documentation page.

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.

11.8 Copyright and licensing information

Documentation licensing

Percona Operator for MongoDB documentation is (C)2009-2023 Percona LLC and/or its affiliates and is distributed under the <u>Creative Commons Attribution 4.0</u>
International License [].

11.9 Trademark policy

This <u>Trademark Policy</u> is to ensure that users of Percona-branded products or services know that what they receive has really been developed, approved, tested and maintained by Percona. Trademarks help to prevent confusion in the marketplace, by distinguishing one company's or person's products and services from another's.

Percona owns a number of marks, including but not limited to Percona, XtraDB, Percona XtraDB, XtraBackup, Percona XtraBackup, Percona Server, and Percona Live, plus the distinctive visual icons and logos associated with these marks. Both the unregistered and registered marks of Percona are protected.

Use of any Percona trademark in the name, URL, or other identifying characteristic of any product, service, website, or other use is not permitted without Percona's written permission with the following three limited exceptions.

First, you may use the appropriate Percona mark when making a nominative fair use reference to a bona fide Percona product.

Second, when Percona has released a product under a version of the GNU General Public License ("GPL"), you may use the appropriate Percona mark when distributing a verbatim copy of that product in accordance with the terms and conditions of the GPL.

Third, you may use the appropriate Percona mark to refer to a distribution of GPL-released Percona software that has been modified with minor changes for the sole purpose of allowing the software to operate on an operating system or hardware platform for which Percona has not yet released the software, provided that those third party changes do not affect the behavior, functionality, features, design or performance of the software. Users who acquire this Percona-branded software receive substantially exact implementations of the Percona software.

Percona reserves the right to revoke this authorization at any time in its sole discretion. For example, if Percona believes that your modification is beyond the scope of the limited license granted in this Policy or that your use of the Percona mark is detrimental to Percona, Percona will revoke this authorization. Upon revocation, you must immediately cease using the applicable Percona mark. If you do not immediately cease using the Percona mark upon revocation, Percona may take action to protect its rights and interests in the Percona mark. Percona does not grant any license to use any Percona mark for any other modified versions of Percona software; such use will require our prior written permission.

Neither trademark law nor any of the exceptions set forth in this Trademark Policy permit you to truncate, modify or otherwise use any Percona mark as part of your own brand. For example, if XYZ creates a modified version of the Percona Server, XYZ may not brand that modification as "XYZ Percona Server" or "Percona XYZ Server", even if that modification otherwise complies with the third exception noted above.

In all cases, you must comply with applicable law, the underlying license, and this Trademark Policy, as amended from time to time. For instance, any mention of Percona trademarks should include the full trademarked name, with proper spelling and capitalization, along with attribution of ownership to Percona Inc. For example, the full proper name for XtraBackup is Percona XtraBackup. However, it is acceptable to omit the word "Percona" for brevity on the second and subsequent uses, where such omission does not cause confusion.

In the event of doubt as to any of the conditions or exceptions outlined in this Trademark Policy, please contact <u>trademarks@percona.com</u> for assistance and we will do our very best to be helpful.

12 Release notes

12.1 Percona Operator for MongoDB Release Notes

- Percona Operator for MongoDB 1.20.0 (2025-05-19)
- Percona Operator for MongoDB 1.19.1 (2025-02-20)
- Percona Operator for MongoDB 1.19.0 (2025-01-21)
- Percona Operator for MongoDB 1.18.0 (2024-11-14)
- Percona Operator for MongoDB 1.17.0 (2024-09-09)
- Percona Operator for MongoDB 1.16.2 (2024-07-23)
- Percona Operator for MongoDB 1.16.1 (2024-06-24)
- Percona Operator for MongoDB 1.16.0 (2024-05-24)
- Percona Operator for MongoDB 1.15.0 (2023-10-09)
- Percona Operator for MongoDB 1.14.0 (2023-03-13)
- Percona Operator for MongoDB 1.13.0 (2022-09-08)
- Percona Operator for MongoDB 1.12.0 (2022-05-05)
- Percona Distribution for MongoDB Operator 1.11.0 (2021-12-21)
- Percona Distribution for MongoDB Operator 1.10.0 (2021-09-30)
- Percona Distribution for MongoDB Operator 1.9.0 (2021-07-29)
- Percona Kubernetes Operator for Percona Server for MongoDB 1.8.0 (2021-05-06)
- Percona Kubernetes Operator for Percona Server for MongoDB 1.7.0 (2021-03-08)
- Percona Kubernetes Operator for Percona Server for MongoDB 1.6.0 (2020-12-22)
- Percona Kubernetes Operator for Percona Server for MongoDB 1.5.0 (2020-09-07)
- Percona Kubernetes Operator for Percona Server for MongoDB 1.4.0 (2020-03-31)
- Percona Kubernetes Operator for Percona Server for MongoDB 1.3.0 (2019-12-11)
- Percona Kubernetes Operator for Percona Server for MongoDB 1.2.0 (2019-09-20)
- Percona Kubernetes Operator for Percona Server for MongoDB 1.1.0 (2019-07-15)
- Percona Kubernetes Operator for Percona Server for MongoDB 1.0.0 (2019-05-29)

Percona Operator for MongoDB 1.20.0 (2025-05-19)

Get started with the Operator \rightarrow

Release Highlights

This release of Percona Operator for MongoDB includes the following new features and improvements:

Point-in-time recovery from any backup storage

The Operator now natively supports <u>multiple backup storages</u> inheriting this feature from Percona Backup for MongoDB (PBM). This enables you to make a point-in-time recovery from any backup stored on any storage - PBM and the Operator maintain the data consistency for you. And you no longer have to wait till the Operator reconfigures a cluster after you select a different storage for a backup or a restore. As a result, overall performance of your backup flow improves.

Improve RTO with the added support of incremental physical backups (tech preview)

Using <u>incremental physical backups</u> in the Operator, you can now back up only the changes happened since the previous backup. Since increments are smaller in size than the whole backup, the backup completion is faster and you also save on the storage and data transfer costs. Using incremental backups and point-in-time recovery improves your recovery time objective (RTO).

You do need the base backup to start the incremental backup chain and you must make the whole chain from the same storage. Also note that the percona.com/delete-backup finalizer and the .spec.backup.tasks.[].keep option apply for the incremental base backup but are ignored for subsequent incremental backups.

Improved monitoring for clusters in multi-region or multi-namespace deployments in PMM

Now you can define a custom name for your clusters deployed in different data centers. This name helps Percona Management and Monitoring (PMM) Server to correctly recognize clusters as connected and monitor them as one deployment. Similarly, PMM Server identifies clusters deployed with the same names in different namespaces as separate ones and correctly displays performance metrics for you on dashboards.

To assign a custom name, define this configuration in the Custom Resource manifest for your cluster:

```
spec:
   pmm:
   customClusterName: mongo-cluster
```

Changelog

New Features

- K8SPSMDB-1237 Added support for incremental physical backups
- K8SPSMDB-1329 Allowed setting loadBalancerClass service type and using a custom implementation of a load balancer rather than the cloud provider
 default one.

Improvements

- K8SPSMDB-621 Set PBM_MONGODB_URI env variable in PBM container to avoid defining it for every shell session and improve setup automation (Thank you Damiano Albani for reporting this issue)
- K8SPSMDB-1219 Improved the support of multiple storages for backups by using the Multi Storage support functionality in PBM. This enables users to make point-in-time recovery from any storage
- K8SPSMDB-1223 Improved the MONGODB_PBM_URI connection string construction by enabling every pbm-agent to connect to local mongoDB directly
- K8SPSMDB-1226 Documented how to pass custom configuration for PBM
- K8SPSMDB-1234 Added the ability to use non default ports 27017 for MongoDB cluster components: mongod, mongos and configsvrReplSet Pods
- K8SPSMDB-1236 Added a check for a username to be unique when defining it via the Custom Resource manifest
- K8SPSMDB-1253 Made the SmartUpdate the default update strategy

- K8SPSMDB-1276 Added logic to the getMongoUri function to compare the content of the existing TLS and CA certificate files with the secret data. Files are only overwritten if the data has changed, preventing redundant writes and ensuring smoother operations during backup checks. (Thank you Anton Averianov for reporting and contributing to this issue)
- K8SPSMDB-1316 Added the ability to define a custom cluster name for pmm-admin component
- K8SPSMDB-1325 Added the directShardOperations role for a mongo user used for monitoring MongoDB 8 and above
- K8SPSMDB-1337 Add imagePullSecrets for PMM and backup images

Bugs Fixed

- K8SPSMDB-1197 Fixed the healthcheck log rotation routine to delete log file created 1 day before.
- <u>K8SPSMDB-1231</u> Fixed the issue with a single-node cluster to temporarily report the Error state during initial provisioning by ignoring the No mongod containers in running state error.
- K8SPSMDB-1239 Fixed the issue with cron jobs running simultaneously
- K8SPSMDB-1245 Improved Telemetry for cluster-wide deployments to handle both an empty value and a comma-separated list of namespaces
- K8SPSMDB-1256 Fixed the issue with PBM failing with the length of read message too large error by verifying the existence of TLS files when constructing the PBM_MONGODB_URI connection string URI
- K8SPSMDB-1263 Fixed the issue with the Operator losing connection to mongod pods during backup and throwing an error by retrying to connect and proceed with the backup
- K8SPSMDB-1274 Disable balancer before logical restore to meet the PBM restore requirements
- K8SPSMDB-1275 Fixed the issue with the Operator failing when the getLastErrorModes write concern value is set for a replica set by using the data type for a value that matches MongoDB behavior (Thank you user clrxbl for reporting and contributing to this issue)
- <u>K8SPSMDB-1294</u> Fixed the API mismatch error with the multi-cluster Services (MCS) enabled in the Operator by using the DiscoveryClient.ServerPreferredResources method to align with the kubect1 behavior.
- K8SPSMDB-1302 Fixed the issue with the Operator being stuck during physical restore when the update strategy is set to SmartUpdate
- K8SPSMDB-1306 Fixed the Operator panics if a user configures PBM priorities without timeouts
- <u>K8SPSMDB-1347</u> Fixed the issue with the Operator throwing errors when auto generating password for multiple users by properly updating the secret after a password generation

Upgrade considerations

The <u>added support for multiple backup storages</u> requires specifying the main storage. If you use a single storage, it will automatically be marked as main in the Custom Resource manifest during the upgrade. If you use multiple storages, you must define one of them as the main storage when you upgrade to version 1.20.0. The following command shows how to set the s3-us-west storage as the main one:

Supported software

The Operator was developed and tested with the following software:

- Percona Server for MongoDB 6.0.21-18, 7.0.18-11, and 8.0.8-3.
- Percona Backup for MongoDB 2.9.1.

Other options may also work but have not been tested.

Supported platforms

Percona Operators are designed for compatibility with all <u>CNCF-certified</u> <u>C</u> Kubernetes distributions. Our release process includes targeted testing and validation on major cloud provider platforms and OpenShift, as detailed below for Operator version 1.20.0:

- Google Kubernetes Engine (GKE) [1.30-1.32
- Amazon Elastic Container Service for Kubernetes (EKS) [1.30-1.32
- OpenShift Container Platform 2 4.14 4.18
- Azure Kubernetes Service (AKS) [1.30-1.32
- Minikube
 ☐ 1.35.0 based on Kubernetes 1.32.0

This list only includes the platforms that the Percona Operators are specifically tested on as part of the release process. Other Kubernetes flavors and versions depend on the backward compatibility offered by Kubernetes itself.

Percona certified images

Find Percona's certified Docker images that you can use with the Percona Operator for MongoDB in the following table:

Images released with the Operator version 1.20.0:

Image	Digest
percona/percona-server-mongodb-operator:1.20.0 (x86_64)	01da3139b0f7f64a27f3642ca06581ea065a02891b13ce2375d61471011d6dd4
percona/percona-server-mongodb-operator:1.20.0 (ARM64)	26d885398af42d18928f51f070aff770df900eb5ddf46e3e0bc2570720089bb1
percona/pmm-client:2.44.1	8b2eaddffd626f02a2d5318ffebc0c277fe8457da6083b8cfcada9b6e6168616
percona/pmm-client:2.44.1 (ARM64)	337fecd4afdb3f6daf2caa2b341b9fe41d0418a0e4ec76980c7f29be9d08b5ea
percona/percona-backup-mongodb:2.9.1 (x86_64)	976bfbaa548eb70dd90bf0bd2dcfe40b2994d749ef644af3a0590f4856e4d7e2
percona/percona-backup-mongodb:2.9.1 (ARM64)	ebc6e5c5aa3ed97991d3fd90e9201597b485ddc0eae8d7ee4311ecb785c03bf0
percona/percona-server-mongodb:8.0.8-3 (x86_64)	e4580ca292f07fd7800e139121aea4b2c1dfa6aa34f3657d25a861883fd3de41
percona/percona-server-mongodb:8.0.8-3 (ARM64)	96cfee2102499aba05e63ca7862102c2b1da1cf9f4eea0cbea3793a07c183925
percona/percona-server-mongodb:8.0.4-1-multi (x86_64)	873b201ce3d66d97b1225c26db392c5043a73cc19ee8db6f2dc1b8efd4783bcf
percona/percona-server-mongodb:8.0.4-1-multi (ARM64)	222ccf746ad4ffdfccf41b41edaa0d318d28f663e13c9629f8dad5a5078434e5
percona/percona-server-mongodb:7.0.18-11 (x86_64)	0115a72f5e60d86cb4f4b7eae32118c0910e8c96831e013de12798a1771c4c91
percona/percona-server-mongodb:7.0.18-11 (ARM64)	86c17067f3e233f522612389ed2500231cbb22ce93524c476b9aa8d464d06f0b
percona/percona-server-mongodb:7.0.15-9-multi (x86_64)	7bffdf2e71c121e2ab37b4fa7e2f513237abdd65266da384bf8197cee1316917
percona/percona-server-mongodb:7.0.15-9-multi (ARM64)	fdc4875df82572267445811445ebf517f63e509be54d1a2599fe58e1c525e1d8
percona/percona-server-mongodb:7.0.14-8-multi (x86_64)	ed932d4e7231dcb793bf609f781226a8393aa8958b103339f4a503a8f70ed17e
percona/percona-server-mongodb:7.0.14-8-multi (ARM64)	052f84ee926ad9b5146f08a7e887820342d65b757a284c2f0ea8e937bb51cd7b
percona/percona-server-mongodb:7.0.12-7	7f00e19878bd143119772cd5468f1f0f9857dfcd2ae2f814d52ef3fa7cff6899

Image	Digest
percona/percona-server-mongodb:6.0.21-18 (x86_64)	579d2fdc617ea42ab2be8c2682955b489dbf49ab19771b7a5d9c77da4dd323e7
percona/percona-server-mongodb:6.0.21-18 (ARM64)	b9d2b7e8c4a97b2d20e2aaccfbd183f65f8ccd9f2ea13939515e18e02bc64871
percona/percona-server-mongodb:6.0.19-16-multi (x86_64)	c8ff08c4b8a96679e2daf4845873fdd4d2c48646b84db19f0c5fe02e8f3808b4
percona/percona-server-mongodb:6.0.19-16-multi (ARM64)	6908b28ced260b762cd38a642c06dd802cbef0a43ab5f22afe7b583b234ebcec
percona/percona-server-mongodb:6.0.18-15-multi (x86_64)	d197ce16ab0eed6df25e632b92dea5ce448e549e02028f39b78f5730c2ffef36
percona/percona-server-mongodb:6.0.18-15-multi (ARM64)	7fd1d8f74f71dea6ad423e8e202a0617bdd1e8783f2b5cb071b5281685ce0adf
percona/percona-server-mongodb:6.0.16-13	1497e58e39497d8425ccd053898dc323338d6eb3f0e3c4c223f9d5a468da7931
percona/percona-server-mongodb:6.0.15-12	f12dd271d78cf3e70088fea0c420e8c03703457d8a5959b645053546bff94dea

Find previous version images in the $\underline{\text{documentation archive}}$

Percona Operator for MongoDB 1.19.1

Date

February 20, 2025

Installation

Installing Percona Operator for MongoDB

Bugs Fixed

• K8SPSMDB-1274: Revert to disabling MongoDB balancer during restores to follow requirements of Percona Backup for MongoDB 2.8.0.

Known limitations

• <u>PBM-1493</u>: For sharded MongoDB 8.0 deployments, Percona Operator for MongoDB versions 1.19.0 and 1.19.1 have a known issue causing point-in-time recovery failures. Avoid upgrading to these Operator versions until a fix is released in Percona Backup for MongoDB and added into the newer versions of the Operator

Supported Platforms

The Operator was developed and tested with Percona Server for MongoDB 6.0.19-16, 7.0.15-9, and 8.0.4-1. Other options may also work but have not been tested. The Operator also uses Percona Backup for MongoDB 2.8.0.

Percona Operators are designed for compatibility with all <u>CNCF-certified</u> <u>C</u> Kubernetes distributions. Our release process includes targeted testing and validation on major cloud provider platforms and OpenShift, as detailed below for Operator version 1.19.1:

- Google Kubernetes Engine (GKE) 1.28-1.30
- Amazon Elastic Container Service for Kubernetes (EKS) [1.29-1.31
- OpenShift Container Platform 2 4.14.44 4.17.11
- Azure Kubernetes Service (AKS) [1.28-1.31
- Minikube ☑ 1.34.0 based on Kubernetes 1.31.0

Percona Operator for MongoDB 1.19.0

Date

January 21, 2025

Installation

Installing Percona Operator for MongoDB

Release Highlights

Using remote file server for backups (tech preview)

The new filesystem backup storage type was added in this release in addition to already existing s3 and azure types. It allows users to mount a remote file server to a local directory, and make Percona Backup for MongoDB using this directory as a storage for backups. The approach is based on common Network File System (NFS) protocol, and should be useful in network-restricted environments without S3-compatible storage or in cases with a non-standard storage service supporting NFS access.

To use NFS-capable remote file server as a backup storage, user needs to mount the remote storage as a sidecar volume in the replsets section of the Custom Resource (and also configsyrReplSet in case of a sharded cluster):

```
replsets:
...
sidecarVolumes:
- name: backup-nfs
nfs:
server: "nfs-service.storage.svc.cluster.local"
path: "/psmdb-some-name-rs0"
...
```

Finally, this new storage needs to be configured in the same Custom Resource as a normal storage for backups:

```
backup:
...
storages:
  backup-nfs:
  filesystem:
    path: /mnt/nfs/
    type: filesystem
...
volumeMounts:
- mountPath: /mnt/nfs/
    name: backup-nfs
```

See more in our documentation about this storage type.

Generated passwords for custom MongoDB users

A new improvement for the <u>declarative management of custom MongoDB users</u> brings the possibility to use automatic generation of users passwords. When you specify a new user in <u>deploy/cr.yaml</u> configuration file, you can omit specifying a reference to an already existing Secret with the user's password, and the Operator will generate it automatically:

```
users:
- name: my-user
db: admin
roles:
- name: clusterAdmin
db: admin
- name: userAdminAnyDatabase
db: admin
```

Find more details on this automatically created Secret in our documentation.

Percona Server for MongoDB 8.0 support

Percona Server for MongoDB 8.0 is now supported by the Operator in addition to 6.0 and 7.0 versions. The appropriate images are now included into the <u>list of Percona-certified images</u>. See <u>this blogpost</u> of the details about the latest MongoDB 8.0 features with the added reliability and performance improvements.

New Features

- K8SPSMDB-1109: Backups can now be stored on a remote file server
- K8SPSMDB-921: IAM Roles for Service Accounts (IRSA) allow automating access to AWS S3 buckets based on Identity Access Management with no need to specify the S3 credentials explicitly
- <u>K8SPSMDB-1133</u>: Manual change of Replica Set Member Priority in Percona Server MongoDB Operator <u>is now possible</u> with the new replsetOverrides. MEMBER-NAME.priority Custom Resource option
- K8SPSMDB-1164: Add the possibility to create users in the \$external database for external authentication purposes

Improvements

- K8SPSMDB-1123: Percona Server for MongoDB 8.0 is now supported
- K8SPSMDB-1171: The declarative user management was enchanced with the possibility to automatically generate passwords
- K8SPSMDB-1174: Telemetry was improved to to track whether the custom users and roles management, automatic volume expansion, and multi-cluster services features are enabled
- K8SPSMDB-1179: It is now possible to configure externalTrafficPolicy for mongod, configsvr and mongos instances
- <u>K8SPSMDB-1205</u>: Backups in unmanaged clusters <u>are now supported</u>, removing a long-standing limitation of <u>cross-site replication</u> that didn't allow backups on replica clusters

Bugs Fixed

- K8SPSMDB-1215: Fix a bug where ExternalTrafficPolicy was incorrectly set for LoadBalancer and NodePort services (Thanks to Anton Averianov for contributing)
- K8SPSMDB-675: Fix a bug where disabling sharding failed on a running cluster with enabled backups
- K8SPSMDB-754: Fix a bug where some error messages had "INFO" log level and therefore were not seen in logs with the "ERROR" log level turned on
- K8SPSMDB-1088: Fix a bug which caused the Operator starting two backup operations if the user patches the backup object while its state is empty or Waiting
- K8SPSMDB-1156: Fix a bug that prevented the Operator with enabled backups to recover from invalid TLS configurations (Thanks to KOS for reporting)
- K8SPSMDB-1172: Fix a bug where backup user's password username with special characters caused Percona Backup for MongoDB to fail
- K8SPSMDB-1212: Stop disabling balancer during restores, because it is not required for Percona Backup for MongoDB 2.x

Deprecation, Rename and Removal

- The psmdbCluster option from the deploy/backup/backup.yaml manifest used for <u>on-demand backups</u>, which was deprecated since the Operator version 1.12.0 in favor of the clusterName option, has been removed and is no longer supported.
- Percona Server for MongoDB 5.0 has reached its end of life and in no longer supported by the Operator

Supported Platforms

The Operator was developed and tested with Percona Server for MongoDB 6.0.19-16, 7.0.15-9, and 8.0.4-1. Other options may also work but have not been tested. The Operator also uses Percona Backup for MongoDB 2.8.0.

Percona Operators are designed for compatibility with all <u>CNCF-certified</u> <u>C</u> Kubernetes distributions. Our release process includes targeted testing and validation on major cloud provider platforms and OpenShift, as detailed below for Operator version 1.19.0:

- Google Kubernetes Engine (GKE)
 ☐ 1.28-1.30
- Amazon Elastic Container Service for Kubernetes (EKS) [₹ 1.29-1.31]
- OpenShift Container Platform 2 4.14.44 4.17.11

- Azure Kubernetes Service (AKS) [1.28-1.31
- Minikube 🖸 1.34.0 based on Kubernetes 1.31.0

Percona Operator for MongoDB 1.18.0

Date

November 14, 2024

Installation

Installing Percona Operator for MongoDB

Release Highlights

Enhancements of the declarative user management

The <u>declarative management of custom MongoDB users</u> was improved compared to its initial implementation in the previous release, where the Operator did not track and sync user-related changes in the Custom Resource and the database. Also, starting from now you can create custom MongoDB roles on various databases just like users in the <u>deploy/cr.yaml</u> manifest:

```
roles:
- name: clusterAdmin
db: admin
- name: userAdminAnyDatabase
db: admin
```

See the documentation to find more details about this feature.

Support for selective restores

Percona Backup for MongoDB 2.0.0 has introduced a new functionality that allows partial restores, which means selectively restoring only with the desired subset of data. Now the Operator also supports this feature, allowing you to restore a specific database or a collection from a backup. You can achieve this by using an additional selective section in the PerconaServerMongoDBRestore Custom Resource:

```
spec:
    selective:
    withUsersAndRoles: true
    namespaces:
    - "db.collection"
```

You can find more on selective restores and their limitations in our documentation.

Splitting the replica set of the database cluster over multiple Kubernetes clusters

Recent improvements in cross-site replication made it possible to keep the replica set of the database cluster in different data centers. The Operator itself cannot deploy MongoDB replicas to other data centers, but this still can be achieved with a number of Operator deployments, equal to the size of your replica set: one Operator to control the replica set via cross-site replication, and at least two Operators to bootstrap the unmanaged clusters with other MongoDB replica set instances. Splitting the replica set of the database cluster over multiple Kubernetes clusters can be useful to get a fault-tolerant system in which all replicas are in different data centers. You can find more about configuring such a multi-datacenter MongoDB cluster and the limitations of this solution on the dedicated documentation page.

New Features

- K8SPSMDB-894: It is now possible to restore a subset of data (a specific database or a collection) from a backup which is useful to reduce time on restore operations when fixing corrupted data fragment
- K8SPSMDB-1113: The new percona.com/delete-pitr-chunks finalizer allows the deletion of PITR log files from the backup storage when deleting a cluster so that leftover data does not continue to take up space in the cloud
- K8SPSMDB-1124 and K8SPSMDB-1146: Declarative user management now covers creating and managing user roles, and syncs user-related changes between the Custom Resource and the database
- K8SPSMDB-1140 and K8SPSMDB-1141: Multi-datacenter cluster deployment is now possible

Improvements

- <u>K8SPSMDB-739</u>: A number of Service exposure options in the replsets, sharding.configsvrReplSet, and sharding.mongos were renamed for unification with other Percona Operators
- <u>K8SPSMDB-1002</u>: New Custom Resource options under the <u>replsets.primaryPreferTagSelector</u> subsection allow providing Primary instance selection preferences based on specific zone and region, which may be especially useful within the planned zone switchover process (Thanks to sergelogvinov for contribution)
- K8SPSMDB-1096: Restore logs were improved to contain pbm-agent logs in mongod containers, useful to debug failures in the backup restoration process
- K8SPSMDB-1135: Split-horizon DNS for external (unmanaged) nodes is now configurable via the replsets.externalNodes subsection in Custom Resource
- K8SPSMDB-1152: Starting from now, the Operator uses multi-architecture images of Percona Server for MongoDB and Percona Backup for MongoDB, making it easier to deploy a cluster on ARM
- <u>K8SPSMDB-1160</u>: The <u>PVC resize</u> feature introduced in previous release can now be enabled or disabled via the enableVolumeExpansion Custom Resource option (false by default), which protects the cluster from storage resize triggered by mistake
- K8SPSMDB-1132: A new secrets.keyFile Custom Resource option allows to configure custom name for the Secret with the MongoDB internal auth key file

Bugs Fixed

- K8SPSMDB-912: Fix a bug where the full backup connection string including the password was visible in logs in case of the Percona Backup for MongoDB errors
- <u>K8SPSMDB-1047</u>: Fix a bug where the Operator was changing <u>writeConcernMajorityJournalDefault</u> to "true" during the replica set reconfiguring, ignoring the value set by user
- K8SPSMDB-1168: Fix a bug where successful backups could obtain a failed state in case of the Operator configured with watchAllNamespaces: true and having the same name for MongoDB clusters across multiple namespaces (Thanks to Markus Küffner for contribution)
- K8SPSMDB-1170: Fix a bug that prevented deletion of a cluster with the active percona.com/delete-psmdb-pods-in-order finalizer in case of the cluster error state (e.g. when mongo replset failed to reconcile)
- K8SPSMDB-1184: Fix a bug where the Operator failed to reconcile when using the container security context with readOnlyRootFilesystem set to true (Thanks to applejag for contribution)
- K8SPSMDB-1180: Fix a bug where rotation functionality didn't work for scheduled backups

Deprecation, Rename and Removal

- The new enableVolumeExpansion Custom Resource option allows users to disable the <u>automated storage scaling with Volume Expansion capability</u>. The default value of this option is false, which means that the automated scaling is turned off by default.
- A number of Service exposure Custom Resource options in the replsets, sharding.configsvrReplSet, and sharding.mongos subsections were
 renamed to provide a unified experience with other Percona Operators:
 - expose.serviceAnnotations option renamed to expose.annotations
 - expose.serviceLabels option renamed to expose.labels
 - expose.exposeType option renamed to expose.type

Supported Platforms

The Operator was developed and tested with Percona Server for MongoDB 5.0.29-25, 6.0.18-15, and 7.0.14-8. Other options may also work but have not been tested. The Operator also uses Percona Backup for MongoDB 2.7.0.

The following platforms were tested and are officially supported by the Operator 1.18.0:

- Google Kubernetes Engine (GKE) [1.28-1.30
- Amazon Elastic Container Service for Kubernetes (EKS) [1.28-1.31
- OpenShift Container Platform 2 4.13.52 4.17.3
- Azure Kubernetes Service (AKS) 🖸 1.28-1.31
- Minikube 2 1.34.0 based on Kubernetes 1.31.0

Percona Operator for MongoDB 1.17.0

Date

September 09, 2024

Installation

Installing Percona Operator for MongoDB

Release Highlights

Declarative user management (technical preview)

Before the Operator version 1.17.0 custom MongoDB users had to be created manually. Now the declarative creation of custom MongoDB users is supported via the users subsection in the Custom Resource. You can specify a new user in deploy/cr.yaml manifest, setting the user's login name and database,

PasswordSecretRef (a reference to a key in a Secret resource containing user's password) and as well as MongoDB roles on various databases which should be assigned to this user:

```
users:
- name: my-user
db: admin
passwordSecretRef:
    name: my-user-password
    key: my-user-password-key
roles:
    - name: clusterAdmin
    db: admin
- name: userAdminAnyDatabase
    db: admin
```

See documentation to find more details about this feature with additional explanations and the list of current limitations.

Liveness check improvements

Several improvements in logging were made related to the liveness checks, to allow getting more information for debugging, and to make these logs persist on failures to allow further examination.

Liveness check logs are stored in the /data/db/mongod-data/logs/mongodb-healthcheck.log file, which can be accessed in the corresponding Pod if needed. Starting from now, Liveness check generates more log messages, and the default log level is set to DEBUG.

Each time the health check fails, the current log is saved to a gzip compressed file named mongodb-healthcheck-<timestamp>.log.gz, and the mongodb-healthcheck.log log file is reset. Logs older than 24 hours are automatically deleted.

New Features

• K8SPSMDB-253: It is now possible to create and manage users via the Custom Resource

Improvements

- K8SPSMDB-899: Add Labels for all Kubernetes objects created by Operator (backups/restores, Secrets, Volumes, etc.) to make them clearly distinguishable
- <u>K8SPSMDB-919</u>: The Operator now checks if the needed Secrets exist and connects to the storage to check the validity of credentials and the existence of a backup before starting the restore process
- K8SPSMDB-934: Liveness checks are providing more debug information and keeping separate log archives for each failure with the 24 hours retention
- K8SPSMDB-1057: Finalizers were renamed to contain fully qualified domain names (FQDNs), avoiding potential conflicts with other finalizer names in the same Kubernetes environment
- K8SPSMDB-1108: The new Custom Resource option allows setting custom containerSecurityContext for PMM containers
- K8SPSMDB-994: Remove a limitation where it wasn't possible to create a new cluster with splitHorizon enabled, leaving the only way to enable it later on the running cluster

Bugs Fixed

- K8SPSMDB-925: Fix a bug where the Operator generated "failed to start balancer" and "failed to get mongos connection" log messages when using Mongos with servicePerPod and LoadBalancer services, while the cluster was operating properly
- K8SPSMDB-1105: The memory requests and limits for backups were increased in the deploy/cr.yaml configuration file example to reflect the Percona Backup for MongoDB minimal pbm-agents requirement of 1 Gb RAM needed for stable operation
- K8SPSMDB-1074: Fix a bug where MongoDB Cluster could not failover in case of all Pods downtime and exposeType Custom Resource option set to either NodePort or LoadBalancer
- K8SPSMDB-1089: Fix a bug where it was impossible to delete a cluster in error state with finalizers present
- K8SPSMDB-1092: Fix a bug where Percona Backup for MongoDB log messages during physical restore were not accessible with the kubect1 logs command
- K8SPSMDB-1094: Fix a bug where it wasn't possible to create a new cluster with upgradeOptions.setFCV Custom Resource option set to true
- K8SPSMDB-1110: Fix a bug where nil Custom Resource annotations were causing the Operator panic

Deprecation, Rename and Removal

Finalizers were renamed to contain fully qualified domain names to comply with the Kubernetes standards.

- PerconaServerMongoDB Custom Resource:
 - delete-psmdb-pods-in-order finalizer renamed to percona.com/delete-psmdb-pods-in-order
 - delete-psmdb-pvc finalizer renamed to percona.com/delete-psmdb-pvc
- PerconaServerMongoDBBackup Custom Resource:
 - delete-backup finalizer renamed to percona.com/delete-backup

Key change in <u>psmdb-db</u> <u>Helm chart</u>: the parameter for defining <u>system users</u> is renamed from users to <u>systemUsers</u>. The users parameter now handles the new <u>Declarative user management</u> feature. This change impacts users upgrading to this version via Helm: make sure that values manifests are changed accordingly.

Supported Platforms

The Operator was developed and tested with Percona Server for MongoDB 5.0.28-24, 6.0.16-13, and 7.0.12-7. Other options may also work but have not been tested. The Operator also uses Percona Backup for MongoDB 2.5.0.

The following platforms were tested and are officially supported by the Operator 1.17.0:

- Google Kubernetes Engine (GKE) 1.27-1.30
- <u>Amazon Elastic Container Service for Kubernetes (EKS)</u> ☐ 1.28-1.30
- OpenShift Container Platform 🖸 4.13.48 4.16.9
- Azure Kubernetes Service (AKS) [1.28-1.30
- Minikube ☐ 1.33.1

Percona Operator for MongoDB 1.16.2

Date

July 23, 2024

Installation

Installing Percona Operator for MongoDB

Bugs Fixed

• K8SPSMDB-1117: Fix a bug where the Operator incorrectly compares G with Gi and and tries to downscale PVC size after upgrade

Supported Platforms

The Operator was developed and tested with Percona Server for MongoDB 5.0.26-22, 6.0.15-12, and 7.0.8-5. Other options may also work but have not been tested. The Operator also uses Percona Backup for MongoDB 2.4.1.

The following platforms were tested and are officially supported by the Operator 1.16.2:

- Google Kubernetes Engine (GKE) [1.26-1.29
- Amazon Elastic Container Service for Kubernetes (EKS) [1.26-1.29
- OpenShift Container Platform 2 4.12.56 4.15.11
- Azure Kubernetes Service (AKS) 🖸 1.27-1.29
- <u>Minikube</u> ☐ 1.33.0

Percona Operator for MongoDB 1.16.1

Date

June 24, 2024

Installation

Installing Percona Operator for MongoDB

Bugs Fixed

• K8SPSMDB-1101: Fix a bug where manually generated TLS certificates couldn't be applied because Operator was replacing them with auto-generated ones

Supported Platforms

The Operator was developed and tested with Percona Server for MongoDB 5.0.26-22, 6.0.15-12, and 7.0.8-5. Other options may also work but have not been tested. The Operator also uses Percona Backup for MongoDB 2.4.1.

The following platforms were tested and are officially supported by the Operator 1.16.1:

- Google Kubernetes Engine (GKE) 🖸 1.26-1.29
- Amazon Elastic Container Service for Kubernetes (EKS) [1.26-1.29
- OpenShift Container Platform 2 4.12.56 4.15.11
- Azure Kubernetes Service (AKS) 🖸 1.27-1.29
- <u>Minikube</u> ☐ 1.33.0

Percona Operator for MongoDB 1.16.0

Date

May 24, 2024

Installation

Installing Percona Operator for MongoDB

Release Highlights

General availability of Physical Backups

Two releases ago we added experimental support for Physical Backups and Restores to significantly reduce Recovery Time Objective (RTO [3]), especially for big data sets. With this release Percona announces the general availability of physical backups and restores for Percona Server for MongoDB with the Operator.

Automated volume expansion

Kubernetes supports the Persistent Volume expansion as a stable feature since v1.24. Using it with the Operator previously involved manual operations. Now this is automated, and users can resize their PVCs by just changing the value of the resources.requests.storage option in the PerconaServerMongoDB custom resource. This feature is in a technical preview stage and is not recommended for production environments.

Support for MongoDB 7

Starting from this release, MongoDB 7.0 is now supported. Read our take on top-5 changes in MongoDB version 7 in this blog post. 🗗

Support for ARM architecture (technical preview)

ARM architecture meets the intensive growth of its usage nowadays, both in a segment of highly efficient cloud computing based on systems like AWS Graviton, and the Internet of Things or Edge. Officially certified images for ARM are now available for the Operator, as well as Percona Server for MongoDB and Percona Backup for MongoDB, while database monitoring based on PMM Client is yet to follow.

Fixing the overloaded allowUnsafeConfigurations flag

In the previous Operator versions allowUnsafeConfigurations Custom Resource option was used to allow configuring a cluster with unsafe parameters, such as starting it with less than 3 replica set instances. In fact, setting this option to true resulted in a wide range of reduced safety features without the user's explicit intent: disabling TLS, allowing backups in unhealthy clusters, etc.

With this release, a separate unsafeFlags Custom Resource section is introduced for the fine-grained control of the safety loosening features:

unsafeFlags:
 tls: false
 replsetSize: false
 mongosSize: false
 terminationGracePeriod: false
 backupIfUnhealthy: false

Also, TLS configuration is now enabled or disabled by a special tls.mode Custom Resource option, which can be set to disabled, allowTLS, preferTLS, or requireTLS values.

New Features

- K8SPSMDB-1000: Users who store backups on Azure Blob Storage can now use private endpoints
- K8SPSMDB-1055: The kubect1 get psmdb-backup command now shows latest restorable time to make it easier to pick a point-in-time recovery target
- K8SPSMDB-491: It is now possible to specify the existing cert-manager issuer which should be used by the Operator
- K8SPSMDB-733: It is now possible to <u>resize Persistent Volume Claims</u> by patching the PerconaServerMongoDB custom resource: change persistentVolumeClaim.resources.requests.storage and let the Operator do the scaling

Improvements

- K8SPSMDB-1004: Exposing replica set with split-horizon DNS allows to specify URIs with non-standard port numbers, which are particularly useful with the NodePort service type
- K8SPSMDB-1013: MongoDB 7.0 is now supported.
- K8SPSMDB-1015: Information about backup and restore operations is now included in the Operator's logs
- <u>K8SPSMDB-951</u>, <u>K8SPSMDB-979</u> and <u>K8SPSMDB-1021</u>: The Operator now allows setting custom configuration for Percona Backup for MongoDB through the set of new Custom Resource options under backup.configuration.backupOptions, backup.configuration.restoreOptions, and backup.storages.s3.retryer subsections
- K8SPSMDB-1029: Mongod is now run in quiet mode ☐ by default to reduce the amount of log messages
- K8SPSMDB-1032: It is now possible to define TCP port for mongos Service when it is exposed through a NodePort (thanks to Mike Devresse for contribution)
- K8SPSMDB-1062: The Operator now sets appProtocol C to mongo for Service objects, which is useful for service mesh implementations (thanks to Søren Mathiasen for contribution)
- K8SPSMDB-732: Integration of the Operator with OpenLDAP can now be secured by using TLS connections
- <u>K8SPSMDB-755</u>: New allowInvalidCertificates option allows to <u>enable or disable</u> bypassing MongoDB Shell checks for the certificates presented by the mongod/mongos instance, useful for self-signed certificates
- K8SPSMDB-948: Officially certified images for ARM architecture are now available for the Operator, as well as Percona Server for MongoDB and Percona Backup for MongoDB
- K8SPSMDB-993: To avoid backup fail on clusters where Percona Backup for MongoDB resync process takes too long, the Operator now checks, if there is still a resync operation working, with exponentially increasing interval and total wait time until failure equal to 8715 seconds
- <u>K8SPSMDB-995</u>: The Operator now allows storing key for <u>backups server-side AWS KMS encryption</u> in a Secret configurable with the <u>secrets.sse</u> Custom Resource option
- K8SPSMDB-780: Removing allowUnsafeConfigurations Custom Resource option in favor of fine-grained safety control in the unsafeFlags subsection
- K8SPSMDB-1042: Helm chart for Percona Server for MongoDB now accepts replica set options as the map argument instead of the array one used in previous releases; this simplifies how arguments are specified in the command line and allows to specify only part of the replica set parameters, relying on the default values for the other part. Take this change into account if you are installing database via helm and want to use set of custom options from previous releases

Bugs Fixed

- <u>K8SPSMDB-1011</u>: Fix a bug where custom logins for system users stopped working after deleting and recreating back the users Secret (thanks for Patrick Wolleb for report)
- <u>K8SPSMDB-1014</u>: Fix a bug that certificate rotation was bringing the sharded MongoDB cluster down for clusters originally created with the Operator version prior to 1.15.0 (thanks to Stiliyan Stefanov for reporting)
- K8SPSMDB-1018: Fix a bug where MongoDB container startup would fail if the MongoDB image being used contained the numactl package
- K8SPSMDB-1024: Fix a bug where environment variable wasn't properly updated in the Percona Backup for MongoDB container entry script (thanks to Rockawear for contribution)
- <u>K8SPSMDB-1035</u>: Fixed a bug where the empty secretName field was not allowed for backup jobs that might not need it when accessing AWS S3 buckets based on IAM roles (thanks to Sergey Zelenov for contribution)
- K8SPSMDB-1036: Fix a bug due to which restoring backup to a new cluster was broken by incompatibility with Percona Backup for MongoDB 2.3.0
- K8SPSMDB-1038: Fix a bug where mongos Services were deleted if the cluster was set to paused state
- K8SPSMDB-1039: Fix a bug which prevented deleting PMM agent from the PMM Server inventory on Pod termination
- K8SPSMDB-1058: A minor missing privileges issue caused flooding MongoDB logs with "Checking authorization failed" errors
- K8SPSMDB-1070: Fix a bug where panic was happening in delete-psmdb-pods-in-order finalizer if the cluster was deleted prior to creating Pods
- K8SPSMDB-940: Fix a bug due to which the Operator didn't allow to set serviceAccount for mongos Pods
- K8SPSMDB-985: Fix a bug where pbmPod key in backup object was only showing one replica/pod

Deprecation and removal

- Starting from now, allowUnsafeConfigurations Custom Resource option is deprecated in favor of a number of options under the unsafeFlags subsection. Setting allowUnsafeConfigurations won't have any effect; upgrading existing clusters with allowUnsafeConfigurations=true will cause everything under unsafeFlags set to true and TLS funuctionality disabled
- MongoDB 4.4 support in the Operator has reached its end-of-life. Starting from now Percona will not provide <u>officially certified images</u> for it. Make sure that you have a supported MongoDB version before upgrading the Operator to 1.16.0. You can use <u>major version upgrade functionality</u>.

Supported Platforms

The Operator was developed and tested with Percona Server for MongoDB 5.0.26-22, 6.0.15-12, and 7.0.8-5. Other options may also work but have not been tested. The Operator also uses Percona Backup for MongoDB 2.4.1.

The following platforms were tested and are officially supported by the Operator 1.16.0:

- Google Kubernetes Engine (GKE) 🖸 1.26-1.29
- Amazon Elastic Container Service for Kubernetes (EKS) [1.26-1.29
- OpenShift Container Platform 2 4.12.56 4.15.11
- Azure Kubernetes Service (AKS) [1.27-1.29
- Minikube ☐ 1.33.0

Percona Operator for MongoDB 1.15.0

Date

October 9, 2023

Installation

Installing Percona Operator for MongoDB

Release Highlights

Physical Backups now support Point-in-time Recovery (in tech preview)

In the previous 1.14.0 release we added support for Physical Backups and Restores to significantly reduce Recovery Time Objective (RTO_C.)), especially for big data sets. But the problem with losing data between backups - in other words Recovery Point Objective (RPO) - for physical backups was not solved. With this release users can greatly reduce RPO by leveraging the Point-in-time Recovery feature in the Operators. Under the hood we store logical oplogs along with physical backups into the object storage. Read more about this feature in our documentation.

Encrypted backups with Server Side Encryption (SSE)

Backups stored on S3 compatible storage <u>can now be encrypted</u> with Server Side Encryption (SSE) to pass certain compliance or security requirements. Users can leverage integration with AWS KMS or just encrypt/decrypt backups with AES-256 encryption algorithm. It is important to remember that Operator does not store keys and users can choose which key storage to use.

New Features

- K8SPSMDB-227 The new topologySpreadConstraints Custom Resource option allows to use Pod Topology Spread Constraints C to achieve even distribution of Pods across the Kubernetes cluster
- K8SPSMDB-792 and K8SPSMDB-974 The new "sleep infinity" mode available for replset and config server containers allows <u>running the Pod without starting</u> mongod useful to examine a problematic Pod that is constantly restarting
- <u>K8SPSMDB-801</u> It is now possible to delete a backup with its PITR data on retention period or with delete-backup finalizer (there were no PITR files deletion in previous versions)
- K8SPSMDB-926 Point-in-time recovery is now supported with physical backups to significantly reduce Recovery Point Objective (RPO)
- K8SPSMDB-961 The new sharding.balancer.enabled Custom Resource option allows to disable Load Balancer on a cross-site replication managed cluster

Improvements

- K8SPSMDB-662 Restoring a backup with point-in-time recovery can now be easily done to a latest available position by setting pitr.type
 PerconaServerMongoDBRestore Custom Resource option to latest
- K8SPSMDB-774 The Transport encryption documentation now includes details on <u>updating TLS certificates</u>
- K8SPSMDB-807 A custom name for a Replica Set config server instead of the default cfg one can be set in the custom configuration, which can be useful for migration purposes
- <u>K8SPSMDB-814</u> and <u>K8SPSMDB-927</u> The new terminationGracePeriodSeconds Custom Resource option allows to set termination period for Replica Set containers, useful to cleanly shutdown clusters with big data sets
- K8SPSMDB-850 Server Side Encryption for backups with for S3 and S3-compatible storage is now supported (thanks to Mert Gönül for contribution)
- K8SPSMDB-903 The backup destination URI now includes bucket/container name, allowing the user to specify the full path to the backup as an easy to read string
- K8SPSMDB-924 The token associated with the operator's ServiceAccount is no longer printed in the log when a scheduled backup is running; this improves security and avoids logging uninformative elements
- K8SPSMDB-938 Configuring Kubernetes host aliases is now possible for replica set, config server, and mongos Pods
- K8SPSMDB-946 The psmdb-backup object now includes the name of the Pod that made the backup, to save users from searching for the correct Pod to examine the Percona Backup for MongoDB logs (previously it was necessary to check replica set Pods one by one until logs were found)

- K8SPSMDB-976 The Operator now does not start backups if storages or credentials are not set, avoiding fruitless attempts to configure Percona Backup for MongoDB and cluster state repeatedly changing between ready and error
- K8SPSMDB-929 Using split-horizon DNS for the external access to MongoDB Replica Set Pods of the exposed cluster is now possible

Bugs Fixed

- K8SPSMDB-913 Fix a bug due to which restoring a backup on a cluster with mongos exposed via LoabBalancer resulted in recreating mongos Service with a new IP address
- K8SPSMDB-956 Fix a bug that certificate rotation was bringing the sharded MongoDB cluster down (thanks to Stiliyan for reporting)
- K8SPSMDB-854 Backup stucks after cluster was exposed
- K8SPSMDB-977 The out of memory problem could cause cluster got stuck in the "initializing" state at reconciliation
- K8SPSMDB-778 Fix a bug due to which the Operator did not delete arbiter instances during replica set deletion
- <u>K8SPSMDB-791</u> Fix a bug which prevented setting LoadBalancerSourceRanges Custom Resource option when replsets.expose.exposeType is set to Loadbalancer
- K8SPSMDB-813 Fix a bug due to which secure connection was not used for MongoDB Liveness check (thanks to t-yrka for contribution)
- K8SPSMDB-818 Fix a bug where clusterMonitor user had not enough permissions for PMM monitoring with --enable-all-collectors flag turned on
- K8SPSMDB-872 The Operator didn't prevent attempts to restore a backup with "error" status, which could cause the cluster got stuck in the "initializing" state
- <u>K8SPSMDB-876</u> Fix a bug due to which delete-psmdb-pods-in-order finalizer, intended to shutdown primary Pod last, affected only shards and did not affect config replica set
- K8SPSMDB-911 Fix a bug where connection string with credentials was included in the backup-agent container logs
- K8SPSMDB-958 Fix insufficient permissions issue that didn't allow to monitor mongos instances with Percona Monitoring and Management (PMM)
- <u>K8SPSMDB-962</u> Fix a memory leak due to which the Operator's Pod continually increased both CPU and memory usage in cluster-wide mode (with an unmanaged cluster)
- K8SPSMDB-968 Fix a bug due to which the endpoints list returned by kubect1 get psmdb command contained fully qualified domain names (FQDN) instead of IP addresses when the replset was exposed as a LoadBalancer and the clusterServiceDNSMode was set to Internal

Deprecation and removal

• <u>K8SPSMDB-883</u> The spec.mongod section deprecated in the Operator version 1.12.0 is finally removed from the Custom Resource configuration. If you have encryption disabled using the deprecated mongod.security.enableEncryption option, you need to set encryption disabled with <u>custom configuration</u> before removing mongod section (and before upgrade):

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

Supported Platforms

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

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

- Google Kubernetes Engine (GKE) [1.24-1.28
- Amazon Elastic Container Service for Kubernetes (EKS) 1.24-1.28
- OpenShift Container Platform 2 4.11 4.13

- Azure Kubernetes Service (AKS) [1.25-1.28
- Minikube 🖸 1.31.2 (based on Kubernetes 1.28)

Percona Operator for MongoDB 1.14.0

Date

March 13, 2023

Installation

Installing Percona Operator for MongoDB

Release Highlights

- Backups and Restores are critical for business continuity. With this release you can significantly reduce your Recovery Time Objective (RTO) with Physical backups support in the Operator. The feature is now in technical preview.
- MongoDB 6.0 comes with a variety of improvements and new features. It is now fully supported by the Operator. See our documentation to learn how to upgrade.

New Features

- K8SPSMDB-713 Physical backups are now supported by the Operator to recover big data sets faster
- K8SPSMDB-737 MongoDB 6.0 is now officially supported in addition to 4.x and 5.x versions. Read more about version 6 in our blog.post 🗗
- <u>K8SPSMDB-824</u> New ignoreAnnotations and ignoreLabels Custom Resource options allow to list <u>specific annotations and labels</u> for Kubernetes Service objects, which the Operator should ignore (useful with various Kubernetes flavors which add annotations to the objects managed by the Operator)

Improvements

- K8SPSMDB-658 The Operator log messages appearing during the pause/unpause of the cluster were improved to more clearly indicate this event
- <u>K8SPSMDB-708</u> The new initContainerSecurityContext option allows to configure securityContext for the container which can be used instead of the official image during the initial Operator installation
- K8SPSMDB-721 The backup subsystem was improved so that database is not crashing in case if the backup agent is not able to connect to MongoDB (e.g. due to misconfigured password)
- K8SPSMDB-758 The ServiceMesh fully qualified domain names (FQDNs) for config servers are now prioritized if DNSMode is set to ServiceMesh (thanks to Jo Lyshoel for contribution)
- K8SPSMDB-793 It is now possible to set annotations and labels for Persistent Volume Claims for better integration with Cloud Native tools
- <u>K8SPSMDB-803</u> The Operator now does not attempt to start Percona Monitoring and Management (PMM) client sidecar if the corresponding secret does not contain the pmmserver or pmmserverkey key
- K8SPSMDB-817 Adding external nodes to the cluster is now allowed even when the replica set is not exposed. This unblocks the creation of complex multicluster topologies
- K8SPSMDB-844 Update the RuntimeClass API version to v1 from the v1beta1, which was already deprecated since the Kubernetes version 1.22
- K8SPSMDB-848 Remove formatted strings from log messages to avoid confronting with structured logging based on key-value pairs
- K8SPSMDB-882 Percona Server for MongoDB Helm chart now persists data by default instead of deleting Persistent Volumes after the cluster deletion
- <u>CLOUD-768</u> Helm charts now use random passwords generated by the Operator by default instead of providing pre-configured passwords specified in the values file
- K8SPSMDB-853 To improve the operator we capture anonymous telemetry and usage data. In this release we add more data points to it
- K8SPSMDB-867 The Operator now configures replset members using local fully-qualified domain names (FQDN) resolvable and available only from inside the cluster instead of using IP addresses; the old behavior can be restored by setting the clusterServiceDNSMode option to External

Bugs Fixed

- K8SPSMDB-784 Fix a bug due to which the enableEncryption MongoDB configuration option was always activated when using psmdb-db Helm Chart
- K8SPSMDB-796 Fix a bug due to which backup failed if replica set was exposed

- K8SPSMDB-854 Fix a bug due to which backup got stuck after the cluster was exposed
- K8SPSMDB-471 Fix a bug due to which in case of scheduled backups with error status delete-backup finalizer didn't allow to delete the appropriate failed resources and the Kubernetes namespace (thanks to Aliaksandr Karavai for reporting)
- K8SPSMDB-674 Fix a bug that caused the Operator not deleting unneeded Services after the replica set exposing is turned off
- K8SPSMDB-742 Fix a bug that caused the updates of the sharding.mongos.expose.serviceAnnotations option to be silently rejected
- <u>K8SPSMDB-766</u> and <u>K8SPSMDB-767</u> Fix a bug where the combination of delete-psmdb-pods-in-order and delete-psmdb-pvc finalizers was not working
- K8SPSMDB-770 We now mention the namespace name in the log message to ease debugging when the cluster-wide mode is used
- K8SPSMDB-797 Fix the backup/restore documentation not clearly mentioning that user should specify the bucket for the S3 storage
- K8SPSMDB-820 Fix a bug which prevented the parallel backup jobs execution for different MongoDB clusters in the cluster-wide mode
- K8SPSMDB-823 Fix a bug where backups were not working in case of ReplicaSet exposed with NodePort
- K8SPSMDB-836 Fix backups being incorrectly marked as error while still being in starting status
- K8SPSMDB-841 Fix a bug which turned the cluster into unready status after switching from the LoadBalancer expose to ClusterIP
- K8SPSMDB-843 Fix a bug which made the cluster unable to start if it was recreated with the same Custom Resource after delete without deleting PVCs and Secrets
- K8SPSMDB-846 Fix a bug due to which scaling the replica set down to 1 instance caused the last Pod to remain Secondary instead of becoming Primary
- K8SPSMDB-866 Fix the bug due to which the Operator was continuously flooding the log with error messages if the PMM server credentials were missing

Known Issues and Limitations

- K8SPSMDB-875 Physical backups cannot be restored on the clusters with <u>arbiter, non-voting</u>, or <u>delayed</u> T members due to current Percona Backup for MongoDB limitations
- K8SPSMDB-846 After switching the cluster to unsafe mode by setting allowUnsafeConfig: true, it is not possible to switch back into safe mode. The user can still scale the cluster safely, but the flag is ignored

Supported Platforms

The Operator was developed and tested with Percona Server for MongoDB 4.4.18, 5.0.14, and 6.0.4. Other options may also work but have not been tested.

The following platforms were tested and are officially supported by the Operator 1.14.0:

- Google Kubernetes Engine (GKE)
 ☐ 1.22 1.25
- Amazon Elastic Container Service for Kubernetes (EKS) [1.22 1.24
- OpenShift Container Platform 2 4.10 4.12
- Azure Kubernetes Service (AKS) ☐ 1.23 1.25
- Minikube
 ☐ 1.29

Percona Operator for MongoDB 1.13.0

Date

September 15, 2022

Installation

Installing Percona Operator for MongoDB

Release Highlights

- <u>Azure Kubernetes Service (AKS)</u> 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 <u>can be installed in multi-namespace (so-called "cluster-wide") mode</u>, when a single Operator can be given a list of namespaces in which to manage Percona Server for MongoDB clusters

New Features

- K8SPSMDB-203 Support for the <u>cluster-wide operator mode</u> 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)

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 <u>automated upgrade</u> by default to prevent an unplanned downtime for user applications and to provide defaults more focused on strict user's control over the cluster
- K8SPSMDB-725 Configuring the log structuring and leveling is now supported using the LOG_STRUCTURED and LOG_LEVEL environment variables. This reduces the information overload in logs, still leaving the possibility of getting more details when needed, for example, for debugging
- K8SPSMDB-719 Details about using sharding, Hashicorp Vault and cluster-wide mode were added to telemetry
- K8SPSMDB-715 Starting from now, the Opearator changed its API version to v1 instead of having a separate API version for each release. Three last API version are supported in addition to v1, which substantially reduces the size of Custom Resource Definition to prevent reaching the etcd limit
- <u>K8SPSMDB-709</u> Make it possible <u>to use API Key</u> 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

Bugs Fixed

- <u>K8SPSMDB-702</u> 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

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
- <u>K8SPSMDB-709</u> 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

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
- OpenShift Container Platform 2 4.10 4.11
- Azure Kubernetes Service (AKS) 1.22 1.24
- Minikube [1.26

Percona Operator for MongoDB 1.12.0

Date

May 5, 2022

Installation

Installing Percona Operator for MongoDB

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 <u>Identity and Access</u>

 <u>Management (IAM) roles assigned to the instance</u>; this makes it possible to configure S3 backup on AWS without using IAM keys saved in Secrets
- This release brings <u>support for the Multi Cluster Services (MCS)</u>. 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

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)

Improvements

- K8SPSMDB-473: Allow to skip TLS verification for backup storage, useful for self-hosted S3-compatible storage with a self-issued certificate
- <u>K8SPSMDB-644</u>: 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

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 recreation 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)
- <u>K8SPSMDB-599</u>: A new <u>mongos.expose.servicePerPod</u> 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
- <u>K8SPSMDB-624</u>: 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

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)

Deprecation, Rename and Removal

- <u>K8SPSMDB-596</u>: The spec.mongod section is removed from the Custom Resource configuration. Starting from now, mongod options should be passed to Replica Sets using spec.replsets.[].configuration key, except the following 3 options:
 - mongod.security.encryptionKeySecret key was left in a deprecated state in favor of the new spec.secrets.encryptionKey option
 - mongod.storage.wiredTiger.engineConfig.cacheSizeRatio and mongod.storage.inMemory.engineConfig.inMemorySizeRatio options are now only available from the replsets.storage section

Before the upgrade, please ensure that you have moved all custom MongoDB parameters to proper places!

• K8SPSMDB-228: The spec.psmdbCluster option in the example on-demand backup configuration file backup/backup.yaml was renamed to spec.clusterName (psmdbCluster will be valid till 1.15 version)

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

Percona Distribution for MongoDB Operator 1.11.0

Date

December 21, 2021

Installation

For installation please refer to the documentation page

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.
- <u>Custom sidecar containers</u> 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.

New Features

• K8SPSMDB-513: Add support of Microsoft Azure Blob storage for backups

Improvements

- K8SPSMDB-422: It is now possible to set annotations to backup cron jobs (Thanks to Aliaksandr Karavai for contribution)
- <u>K8SPSMDB-534</u>: 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

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
- <u>K8SPSMDB-511</u> and <u>K8SPSMDB-558</u>: 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

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

Percona Distribution for MongoDB Operator 1.10.0

Date

September 30, 2021

Installation

For installation please refer to the documentation page

Release Highlights

- Starting from this release, the Operator implements as a technical preview the possibility to <u>include non-voting replica set members</u> 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 <u>cross-site replication</u> 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.

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

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

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

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

Percona Distribution for MongoDB Operator 1.9.0

Date

June 29, 2021

Installation

For installation please refer to the documentation page

Release Highlights

- Starting from this release, the Operator changes its official name to **Percona Distribution for 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 <u>Percona Distribution for MongoDB</u>.
- 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 <u>storing custom configuration</u> for Replica Set, mongos, and Config Server instances in ConfigMaps or in Secrets

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

Improvements

- K8SPSMDB-365: Unblock backups even if just a single Replica Set node is available by setting allowUnsafeConfigurations flag to true
- <u>K8SPSMDB-453</u>: 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

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
- <u>K8SPSMDB-423</u>: 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
- <u>K8SPSMDB-455</u>: 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

Deprecation and Removal

- We are simplifying the way the user can customize MongoDB components such as mongod and mongos. <u>It is now possible</u> 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

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

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

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

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

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
- <u>K8SPSMDB-434</u>: Fix a bug due to which nil pointer dereference error was occurring when switching the sharding.enabled option from false to true (thanks to srteam2020 for contributing)
- K8SPSMDB-430: Fix a bug due to which a stale apiserver could trigger undesired StatefulSet and PVC deletion when recreating the cluster with the same name (thanks to srteam2020 for contributing)
- K8SPSMDB-428: Fix a bug which caused mongos to fail in case of the empty name field in configsvrReplSet section of the Custom Resource
- K8SPSMDB-418: Fix a bug due to which serviceAnnotations changes in the deploy/cr.yaml file were not applied to the running cluster
- K8SPSMDB-364: Fix a bug where liveness probe of a mongo container was always failing if the userAdmin password contained special characters
- K8SPSMDB-43: Fix a bug due to which renaming Replica Set in the Custom Resource caused creating new Replica Set without deleting the old one

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

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 <u>custom sidecar containers</u>. 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 <u>replsets</u>, <u>sharding.configsvrReplSet</u>, and <u>sharding.mongos</u> sections of the <u>deploy/cr.yaml</u> configuration file.

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

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

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

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

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 loadbalancer 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



Monitoring with PMM v.1 configured according to the unofficial instruction [3] will not work after the upgrade. Please switch to PMM v.2.

Improvements

- K8SPSMDB-258: Add support for Percona Server for MongoDB version 4.4
- K8SPSMDB-319: Show Endpoint in the kubect1 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

Bugs Fixed

- K8SPSMDB-268: Fix a bug affecting the support of TLS certificates issued by cert-manager [2], 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

Removal

• The MMAPv1 storage engine [2] 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

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

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)

Improvements

- K8SPSMDB-192: The ability to set the mongod cursorTimeoutMillis parameter in YAML (Thanks to user xprt64 for the contribution)
- K8SPSMDB-234: OpenShift 4.5 support
- K8SPSMDB-197: Additional certificate SANs useful for reverse DNS lookups (Thanks to user phin1x for the contribution)
- K8SPSMDB-190: Direct API quering with "curl" instead of using "kubectl" tool in scheduled backup jobs (Thanks to user phin1x for the contribution)
- K8SPSMDB-133: A special Percona Server for MongoDB debug image which avoids restarting on fail and contains additional tools useful for debugging
- <u>CLOUD-556</u>: Kubernetes 1.17 / Google Kubernetes Engine 1.17 support

Bugs Fixed

- K8SPSMDB-213: Installation instruction not reflecting recent changes in git tags (Thanks to user geraintj for reporting this issue)
- K8SPSMDB-210: Backup documentation not reflecting changes in Percona Backup for MongoDB
- K8SPSMDB-180: Replset and cluster having "ready" status set before mongo initialization and replicasets configuration finished
- K8SPSMDB-179: The "error" cluster status instead of the "initializing" one during the replset initialization
- CLOUD-531: Wrong usage of strings. TrimLeft when processing apiVersion

Percona Kubernetes Operator for Percona Server for MongoDB 1.4.0

Date

March 31, 2020

Installation

Installing Percona Kubernetes Operator for PSMDB

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

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

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 $\underline{\text{our bug tracking system}}$ \square .

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. <u>Install the Kubernetes Operator for Percona Server for MongoDB by following the instructions</u>.

The Operator simplifies the deployment and management of the <u>Percona Server for MongoDB</u> of 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. [2]. All of Percona's software is open-source and free.

New Features and Improvements

- CLOUD-415: Non-default cluster domain can now be specified with the new ClusterServiceDNSSuffix Operator option.
- <u>CLOUD-395</u>: 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 C 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 ☑.

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. <u>Install the Kubernetes Operator for Percona Server for MongoDB by following the instructions.</u>

The Operator simplifies the deployment and management of the <u>Percona Server for MongoDB</u> of 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. [2]. All of Percona's software is open-source and free.

New Features and Improvements

- A Service Broker was implemented for the Operator, allowing a user to deploy Percona XtraDB Cluster on the OpenShift Platform, configuring it with a standard GUI, following the Open Service Broker API.
- Now the Operator supports Percona Monitoring and Management 2 [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.

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 [4]).

Percona Server for MongoDB C 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 4.

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. <u>Install the Kubernetes Operator for Percona Server for MongoDB by following the instructions</u>. Please see the <u>GA release announcement</u>.

The Operator simplifies the deployment and management of the <u>Percona Server for MongoDB</u> of 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. [2]. All of Percona's software is open-source and free.

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.
- <u>User secrets</u> 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 [2] 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 <a>I⁻. <a>I⁻.

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. <u>Install the Kubernetes Operator for Percona Server for MongoDB by following the instructions</u>. Please see the <u>GA release announcement</u>. C. 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

Installation

Installation is performed by following the documentation installation instructions for Kubernetes and OpenShift.