MySQL Operator for Kubernetes

Abstract

MySQL Operator for Kubernetes manages MySQL InnoDB Cluster setups inside a Kubernetes Cluster. MySQL Operator for Kubernetes manages the full lifecycle with setup and maintenance including automating upgrades and backups.

This documentation is a work in progress; expect future changes to both content and structure.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2023-03-09 (revision: 75126)

Table of Contents

Preface and Legal Notices

MySQL Operator for Kubernetes manages MySQL InnoDB Cluster setups inside a Kubernetes Cluster. MySQL Operator for Kubernetes manages the full lifecycle with set up and maintenance including automating upgrades and backups. This is the MySQL Operator for Kubernetes manual.

Licensing information. This product may include third-party software, used under license. If you are using a *Commercial* release of MySQL Operator for Kubernetes, see the MySQL Operator for Kubernetes Commercial License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a *Community* release of MySQL Workbench, see the MySQL Operator for Kubernetes Community License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Community release.

Legal Notices

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Chapter 1 Introduction

MySQL and Kubernetes share terminology. For example, a Node might be a Kubernetes Node or a MySQL Node, a Cluster might be a MySQL InnoDB Cluster or Kubernetes Cluster, and a ReplicaSet is a feature in both MySQL and Kubernetes. This documentation prefers the long names but these overloaded terms may still lead to confusion; context is important.

Kubernetes

The Kubernetes system uses Controllers to manage the life-cycle of containerized workloads by running them as Pods in the Kubernetes system. Controllers are general-purpose tools that provide capabilities for a broad range of services, but complex services require additional components and this includes operators. An Operator is software running inside the Kubernetes cluster, and the operator interacts with the Kubernetes API to observe resources and services to assist Kubernetes with the life-cycle management.

MySQL Operator for Kubernetes

The MySQL Operator for Kubernetes is an operator focused on managing one or more MySQL InnoDB Clusters consisting of a group of MySQL Servers and MySQL Routers. The MySQL Operator itself runs in a Kubernetes cluster and is controlled by a Kubernetes Deployment to ensure that the MySQL Operator remains available and running.

The MySQL Operator is deployed in the 'mysql-operator' Kubernetes namespace by default; and watches all InnoDB Clusters and related resources in the Kubernetes cluster. To perform these tasks, the operator subscribes to the Kubernetes API server to update events and connects to the managed MySQL Server instance as needed. On top of the Kubernetes controllers, the operator configures the MySQL servers, replication using MySQL Group Replication, and MySQL Router.

MySQL InnoDB Cluster

Once an InnoDB Cluster (InnoDBCluster) resource is deployed to the Kubernetes API Server, MySQL Operator for Kubernetes creates resources including:

- A Kubernetes StatefulSet for the MySQL Server instances.

  This manages the Pods and assigns the corresponding storage Volume. Each Pod managed by this StatefulSet runs multiple containers. Several provide a sequence of initialisation steps for preparing the MySQL Server configuration and data directory, and then two containers remain active for operational mode. One of those containers (named 'mysql') runs the MySQL Server itself, and the other (named 'sidecar') is a Kubernetes sidecar responsible for local management of the node in coordination with the operator itself.

- A Kubernetes Deployment for the MySQL Routers.

  MySQL Routers are stateless services routing the application to the current Primary or a Replica, depending on the application's choice. The operator can scale the number of routers up or down as required by the Cluster's workload.

A MySQL InnoDB Cluster deployment creates these Kubernetes Services:

- One service is the name of the InnoDB Cluster. It serves as primary entry point for an application and sends incoming connections to the MySQL Router. They provide stable name in the form '{clustername}.svc.cluster.local' and expose specific ports.

  See also Section 3.4, “MySQL InnoDB Cluster Service Explanation” and Chapter 4, *Connecting to MySQL InnoDB Cluster*.

- A second service named '{clustername}-instances' provides stable names to the individual servers. Typically these should not be directly used; instead use the main service to reliably reach the current primary or secondary as needed. However, for maintenance or monitoring purposes, direct access to an instance might be needed. Each pod instance has MySQL Shell installed.

MySQL Operator for Kubernetes creates and manages additional resources that should not be manually modified, including:

- A Kubernetes ConfigMap named '{clustername}-initconf' that contains configuration information for the MySQL Servers.

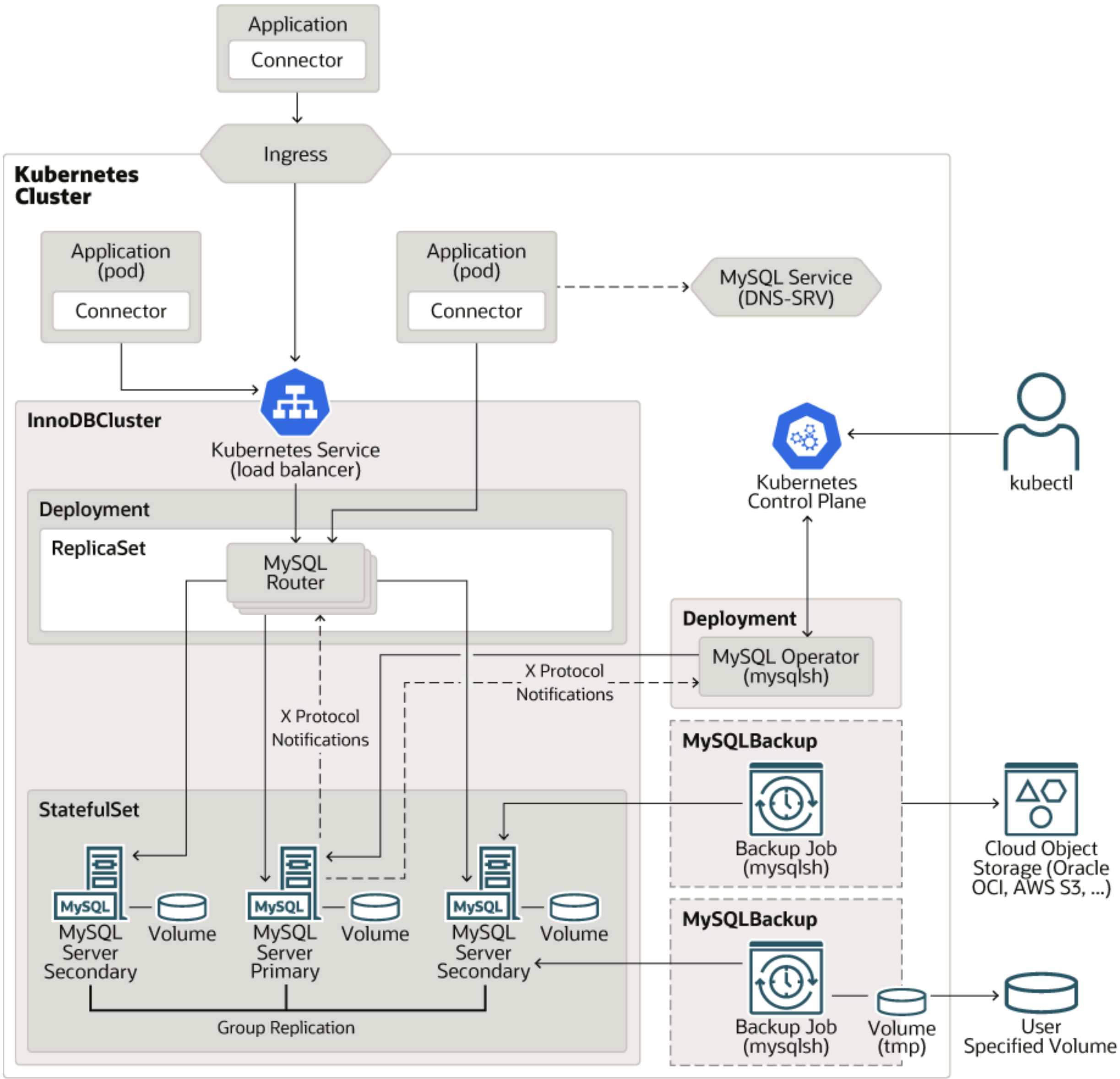
  To modify the generated `my.cnf` configuration file, see Section 3.3, “Manifest Changes for InnoDBCluster”.

- A sequence of Kubernetes Secrets with credentials for different parts of the system; names include '{clustername}.backup', '{clustername.privsecrets}', and '{clustername.router}'.

  For a list of MySQL accounts (and associated Secrets) created by the operator, see Section 3.5, “MySQL Accounts Created by InnoDBCluster Deployment”.

MySQL Operator for Kubernetes Architecture

Figure 1.1 MySQL Operator for Kubernetes Architecture Diagram



Chapter 2 Installing MySQL Operator for Kubernetes

Table of Contents

Two different installation methods are documented here; using either `helm` or manually applying manifests using `kubectl`. This documentation assumes that `kubectl` is available on a system configured with the desired Kubernetes context; and all examples use a Unix-like command line.

> **Note**
>
> MySQL Operator for Kubernetes requires these three container images to function: MySQL Operator for Kubernetes, MySQL Router, and MySQL Server.

2.1 Install using Helm Charts

Helm is an optional package manager for Kubernetes that helps manage Kubernetes applications; Helm uses charts to define, install, and upgrade Kubernetes Operators. For Helm specific usage information, see the Helm Quickstart and Installing Helm guides. Alternatively, see Section 2.2, "Install using Manifest Files".

Add the Helm repository:

```
$> helm repo add mysql-operator https://mysql.github.io/mysql-operator/
$> helm repo update
```

Install MySQL Operator for Kubernetes, this example defines the release name as `my-mysql-operator` using a new namespace named `mysql-operator`:

```
$> helm install my-mysql-operator mysql-operator/mysql-operator \
   --namespace mysql-operator --create-namespace
```

This latest MySQL Operator for Kubernetes can be downloaded from DockerHub and deployed. The operator deployment is customizable through other options that override built-in defaults. For example, useful when using a local (air-gapped) private container registry to use with the operator.

To use MySQL Operator for Kubernetes to create MySQL InnoDB Clusters, see Chapter 3, *MySQL InnoDB Cluster*.

2.2 Install using Manifest Files

This document assumes a familiarity with kubectl, and that you have it installed. Alternatively, see Section 2.1, "Install using Helm Charts".

MySQL Operator for Kubernetes can be installed using raw manifest files with `kubectl`; first install the Custom Resource Definition (CRD) used by MySQL Operator for Kubernetes:

```
$> kubectl apply -f https://raw.githubusercontent.com/mysql/mysql-operator/trunk/deploy/deploy-crds.yam

// Output is similar to:
customresourcedefinition.apiextensions.k8s.io/innodbclusters.mysql.oracle.com created
customresourcedefinition.apiextensions.k8s.io/mysqlbackups.mysql.oracle.com created
customresourcedefinition.apiextensions.k8s.io/clusterkopfpeerings.zalando.org created
customresourcedefinition.apiextensions.k8s.io/kopfpeerings.zalando.org created
```

Next deploy MySQL Operator for Kubernetes, which also includes RBAC definitions as noted in the output:

```
$> kubectl apply -f https://raw.githubusercontent.com/mysql/mysql-operator/trunk/deploy/deploy-operator.yam

// Output is similar to:
clusterrole.rbac.authorization.k8s.io/mysql-operator created
clusterrole.rbac.authorization.k8s.io/mysql-sidecar created
clusterrolebinding.rbac.authorization.k8s.io/mysql-operator-rolebinding created
clusterkopfpeering.zalando.org/mysql-operator created
namespace/mysql-operator created
serviceaccount/mysql-operator-sa created
deployment.apps/mysql-operator created
```

Verify that the operator is running by checking the deployment that's managing the operator inside the mysql-operator namespace, a configurable namespace defined by `deploy-operator.yaml`:

```
$> kubectl get deployment mysql-operator --namespace mysql-operator
```

After MySQL Operator for Kubernetes is ready, the output should look similar to this:

```
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
mysql-operator   1/1     1            1           37s
```

To use MySQL Operator for Kubernetes to create MySQL InnoDB Clusters, see Chapter 3, *MySQL InnoDB Cluster*.

Chapter 3 MySQL InnoDB Cluster

Table of Contents

Examples and documentation assumes the current default namespace is used, which defaults to 'default' although it can be modified, for example:

```
$> kubectl create namespace newdefaultnamespace
$> kubectl config set-context --current --namespace=newdefaultnamespace
```

Examples typically use 'innodbcluster' as the resource name but may use plural and short names as defined in `deploy-crds.yaml`:

```
 names:
    kind: InnoDBCluster
    listKind: InnoDBClusterList
    singular: innodbcluster
    plural: innodbclusters
    shortNames:
      - ic
      - ics
```

3.1 Deploy using Helm

Potential values for creating a MySQL InnoDB Cluster are visible here:

```
$> helm show values mysql-operator/mysql-innodbcluster
```

Public Registry

The most common Helm repository is the public https://artifacthub.io/, which is used by these examples.

This example uses all default values in the default namespace with mycluster as the release name:

```
$> helm install mycluster mysql-operator/mysql-innodbcluster
```

The manifest for this simple installation looks similar to this:

```
$> helm get manifest mycluster

---
# Source: mysql-innodbcluster/templates/service_account_cluster.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mycluster-sa
  namespace: default
---
# Source: mysql-innodbcluster/templates/cluster_secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: mycluster-cluster-secret
```

```
  namespace: default
stringData:
  rootUser: root
  rootHost: "%"
  rootPassword: sakila
---
# Source: mysql-innodbcluster/templates/deployment_cluster.yaml
apiVersion: mysql.oracle.com/v2
kind: InnoDBCluster
metadata:
  name: mycluster
  namespace: default
spec:
  instances: 3
  router:
    instances: 1
  secretName: mycluster-cluster-secret
  imagePullPolicy : IfNotPresent
  baseServerId: 1000
  version: 8.0.31
  serviceAccountName: mycluster-sa
  tls:
    useSelfSigned: false
```

Alternatively set options using command-line parameters:

```
$> helm install mycluster mysql-operator/mysql-innodbcluster \
    --set credentials.root.user='root' \
    --set credentials.root.password='sakila' \
    --set credentials.root.host='%' \
    --set serverInstances=3 \
    --set routerInstances=1 \
    --set tls.useSelfSigned=true
```

To view user-supplied values for an existing cluster:

```
$> helm get values mycluster

USER-SUPPLIED VALUES:
credentials:
  root:
    host: '%'
    password: sakila
    user: root
routerInstances: 1
serverInstances: 3
tls:
  useSelfSigned: true
```

See also Chapter 4, *Connecting to MySQL InnoDB Cluster*.

3.2 Deploy using kubectl

To create an InnoDB Cluster with `kubectl`, first create a secret containing credentials for a new MySQL root user, a secret named 'mypwds' in this example:

```
$> kubectl create secret generic mypwds \
        --from-literal=rootUser=root \
        --from-literal=rootHost=% \
        --from-literal=rootPassword="sakila"
```

Use that newly created user to configure a new MySQL InnoDB Cluster. This example's InnoDBCluster definition creates three MySQL server instances and one MySQL Router instance:

```
apiVersion: mysql.oracle.com/v2
kind: InnoDBCluster
metadata:
  name: mycluster
```

```
spec:
  secretName: mypwds
  tlsUseSelfSigned: true
  instances: 3
  router:
    instances: 1
```

Assuming a file named `mycluster.yaml` contains this definition, install this simple cluster:

```
$> kubectl apply -f mycluster.yaml
```

Optionally observe the process by watching the `innodbcluster` type for the default namespace:

```
$> kubectl get innodbcluster --watch
```

Output looks similar to this:

```
NAME           STATUS     ONLINE    INSTANCES    ROUTERS    AGE
mycluster      PENDING    0         3            1          10s
```

Until reaching ONLINE status:

```
NAME         STATUS    ONLINE    INSTANCES    ROUTERS    AGE
mycluster    ONLINE    3         3            1          2m6s
```

To demonstrate, this example connects with MySQL Shell to show the host name:

```
$> kubectl run --rm -it myshell --image=mysql/mysql-operator -- mysqlsh root@mycluster --sql
If you don't see a command prompt, try pressing enter.
******

MySQL mycluster SQL> SELECT @@hostname

+-------------+
| @@hostname  |
+-------------+
| mycluster-0 |
+-------------+
```

This shows a successful connection that was routed to the mycluster-0 pod in the MySQL InnoDB Cluster. For additional information about connecting, see Chapter 4, *Connecting to MySQL InnoDB Cluster*.

3.3 Manifest Changes for InnoDBCluster

This section covers common options defined while setting up a MySQL InnoDB Cluster. For a full list of options, see Table 7.1, “Spec table for InnoDBCluster”.

Here's a simple example that uses most defaults:

```
apiVersion: mysql.oracle.com/v2
kind: InnoDBCluster
metadata:
  name: mycluster
spec:
  secretName: mypwds
  tlsUseSelfSigned: true
```

Here's an expanded version of that with optional changes:

```
apiVersion: mysql.oracle.com/v2
kind: InnoDBCluster
```

```
metadata:
  name: mycluster
spec:
  secretName: mypwds
  tlsUseSelfSigned: true
  instances: 3
  version: 8.0.31
  router:
    instances: 1
    version: 8.0.31
  datadirVolumeClaimTemplate:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 40Gi
  initDB:
    clone:
      donorUrl: mycluster-0.mycluster-instances.another.svc.cluster.local:3306
      rootUser: root
      secretKeyRef:
        name: mypwds
  mycnf: |
    [mysqld]
    max_connections=162
```

Below are explanations of each change made to initial the *InnoDBCluster* configuration.

Router and Server Versions and Instances

By default, MySQL Operator for Kubernetes installs MySQL Server with the same version as the Operator, and installs Router with the same version as MySQL Server. It also installs 3 MySQL instances and 1 Router instance by default. Optionally configure each:

```
spec:
  instances: 3
  version: 8.0.31
  router:
    instances: 1
    version: 8.0.31
```

Setting PersistentVolumeClaim Size

Set a MySQL instance's storage configuration. For storing the MySQL Server's Data Directory (datadir), a PersistentVolumeClaim (PVC) is used for each MySQL Server pod. Each PVC follows the naming scheme datadir-*`{clustername}-[0-9]`*. A `datadirVolumeClaimTemplate` template allows setting different options, including size and storage class. For example:

```
  datadirVolumeClaimTemplate:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 40Gi
```

For additional configuration information, see the official Storage: Persistent Volumes documentation. The `datadirVolumeClaimTemplate` object is set to `x-kubernetes-preserve-unknown-fields: true`.

> **Note**
>
> MySQL Operator for Kubernetes currently does not support storage resizing.

For a related `MySQLBackup` example that uses a `PersistentVolumeClaim`, see Section 6.1, “Handling MySQL Backups”.

The initDB Object

Optionally initialize an InnoDBCluster with a database using the initDB object; it's only used when the InnoDBCluster is created. It accepts `clone` or `dump` definitions.

This simple initDB *clone* example clones a remote MySQL instance from a cluster. The donor MySQL server's credentials are stored in a Secret on the target server with a 'rootPassword' key for the 'rootUser'.

```
initDB:
  clone:
    donorUrl: mycluster-0.mycluster-instances.another.svc.cluster.local:3306
    rootUser: root
    secretKeyRef:
      name: mypwds
```

MySQL Server restarts after populating with the clone operation, and a "1" is seen in the restart column of the associated pods. Cloning utilizes MySQL Server's The Clone Plugin and behaves accordingly.

For a *dump* example (instead of *clone*), see Section 6.2, “Bootstrap a MySQL InnoDB Cluster from a Dump using Helm”.

Modify my.cnf Settings

Use the mycnf option to add custom configuration additions to the `my.cnf` for each MySQL instance. This example adds a [mysqld] section that sets max_connections to 162:

```
mycnf: |
  [mysqld]
  max_connections=162
```

This is added to the generated `my.cnf`; the default `my.cnf` template is visible in the *initconf* container's ConfigMap. An example to see this template: *kubectl get cm ${CLUSTER_NAME}-initconf -o json* | *jq -r '.data["my.cnf.in"]'*.

3.4 MySQL InnoDB Cluster Service Explanation

For connecting to the InnoDB Cluster, a *Service* is created inside the Kubernetes cluster. The exported ports represent read-write and read-only ports for both the MySQL Protocol and X Protocol.

```
$> kubectl describe service mycluster
```

Output looks similar to this:

```
Name:              mycluster
Namespace:         default
Labels:            mysql.oracle.com/cluster=mycluster
                   tier=mysql
Annotations:       <none>
Selector:          component=mysqlrouter,mysql.oracle.com/cluster=mycluster,tier=mysql
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.106.33.215
IPs:               10.106.33.215
Port:              mysql  3306/TCP
TargetPort:        6446/TCP
Endpoints:         172.17.0.12:6446
Port:              mysqlx  33060/TCP
TargetPort:        6448/TCP
Endpoints:         172.17.0.12:6448
Port:              mysql-alternate  6446/TCP
```

```
TargetPort:         6446/TCP
Endpoints:          172.17.0.12:6446
Port:               mysqlx-alternate  6448/TCP
TargetPort:         6448/TCP
Endpoints:          172.17.0.12:6448
Port:               mysql-ro  6447/TCP
TargetPort:         6447/TCP
Endpoints:          172.17.0.12:6447
Port:               mysqlx-ro  6449/TCP
TargetPort:         6449/TCP
Endpoints:          172.17.0.12:6449
Port:               router-rest  8443/TCP
TargetPort:         8443/TCP
Endpoints:          172.17.0.12:8443
Session Affinity:   None
Events:             <none>
```

An alternative view showing services named `mycluster` and `mycluster-instances`:

```
$> kubectl get service
```

Output looks similar to this:

```
NAMESPACE     NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
default       kubernetes             ClusterIP   10.96.0.1        <none>        443/TCP
default       mycluster              ClusterIP   10.102.198.226   <none>        3306/TCP,33060/TCP,6446/TCP,
default       mycluster-instances    ClusterIP   None             <none>        3306/TCP,33060/TCP,33061/TCP
```

The long host name used to connect to an InnoDB Cluster from within a Kubernetes cluster is `{innodbclustername}.{namespace}.svc.cluster.local`, which routes to the current primary/replica using MySQL Router, depending on the port. Acceptable host name forms:

```
{innodbclustername}.{namespace}.svc.cluster.local
{innodbclustername}.{namespace}.svc
{innodbclustername}.{namespace}
{innodbclustername}
```

Using these names goes to the Kubernetes LoadBalancer (part of Kubernetes Service), which redirects to MySQL Router. MySQL Router then talks to the individual server based on the role, such as PRIMARY or SECONDARY.

For example, assuming 'mycluster' as the InnoDB Cluster name in the 'default' namespace:

```
mycluster.default.svc.cluster.local
```

Using only {innodbclustername} as the host name assumes the session's context is either the default namespace or set accordingly. Alternatively you may use the clusterIP instead of a host name; here's an example that retrieves it:

```
$> kubectl get service/mycluster -o jsonpath='{.spec.clusterIP}'
```

See also Chapter 4, *Connecting to MySQL InnoDB Cluster*.

3.5 MySQL Accounts Created by InnoDBCluster Deployment

MySQL Operator for Kubernetes creates and/or utilizes several MySQL accounts as when creating an InnoDB Cluster. Internal accounts created and only used by MySQL Operator for Kubernetes may be used by users but they must not be changed (dropped, password changes, grant changes, and so on).

Typically the only account a system administrator uses is the 'root' user, whereas other MySQL users are considered internal to the MySQL InnoDB Cluster installation.

Table 3.1 MySQL accounts created and/or used by MySQL Operator.

MySQL User	Purpose	Creator	Description
`root`	General system administration by the user	MySQL Operator for Kubernetes as defined by the user	Defined when InnoDB Cluster is created using a user-supplied Kubernetes secret object as referenced by the `secretsName` configuration option. It's typically `root@'%'` but can be overridden using the `rootUser` and `rootHost` configuration options. You may want to create less-privileged MySQL accounts with this user.
`localroot`	Used by Operator to perform local administration tasks	MySQL Operator for Kubernetes	This local root account specific to MySQL Operator for Kubernetes, and is used by the MySQL sidecar container for local maintenance tasks like creating other accounts, configuring instances, and verifying replication status. It should not be used or edited by users. It's created with auth_socket authentication and PROXY with full privileges and no password.
`mysqladmin`	Administration tasks by the Operator	MySQL Operator for Kubernetes	Used to administer the InnoDB Cluster, credentials managed by the "{clustername}-privsecrets" Kubernetes secret
`mysqlbackup`	Administration tasks by the Operator	MySQL Operator for Kubernetes	Used to create backups and manage backup jobs, credentials managed by the "{clustername}-backup" Kubernetes secret
`mysqlrouter`	Administration tasks by the Operator	MySQL Operator for Kubernetes	Tasks include managing MySQL Router instances to access cluster metadata; credentials managed by the "{clustername}-

MySQL User	Purpose	Creator	Description
			router" Kubernetes secret
`mysqlhealthchecker`	Internal health checks	MySQL Operator for Kubernetes	A local account used for health checks only (liveness and readiness probes); created with auth_socket authentication and no privileges.
`mysql_innodb_cluster_{server_id}`	Internal recovery users that enable connections between the servers in the cluster	MySQL InnoDB Cluster	One per MySQL instance, for additional information see Internal User Accounts Created by InnoDB Cluster.
`mysql.infoschema`	Reserved	MySQL Server	See Reserved Accounts.
`mysql.session`	Reserved	MySQL Server	See Reserved Accounts.
`mysql.sys`	Reserved	MySQL Server	See Reserved Accounts.

Related: Deploying MySQL Operator for Kubernetes creates a Kubernetes service account with a name defaulting to `mysql-operator-sa` in the bundled `deploy-operator.yaml` and Helm deployment template.

For a list of all ports used by MySQL services, see MySQL Port Reference.

3.6 Upgrading a pre-release InnoDB Cluster to 8.0.30-2.0.6 GA

The first MySQL Operator for Kubernetes General Availability (GA) release improved the security configuration and this complicates the upgrade process from pre-GA preview releases. A migration from a preview release (v8.0.28-2.0.3 and earlier) to a GA release without downtime is not possible.

Important

This guide does not apply to future upgrades, say from 8.0.30-2.0.6 to a future release.

Important

If you run multiple clusters then these operations must be done for all clusters in order. In other words, perform step #1 for all clusters, then step #2 for all clusters, and so on.

Prerequisites

A successful migration assumes InnoDB Cluster is managed by MySQL Operator 8.0.28-2.0.3, and that you have credentials to an account (rootUser) with root-style privileges. The InnoDB Cluster must have a minimum of two running MySQL instances. Have a backup before proceeding.

The InnoDBCluster is named `mycluster` in this guide, and assumes it's in the default namespace.

Summary of the Upgrade

The upgrade process described here terminates the old InnoDB Cluster, and removes all data directories except for one. The old MySQL Operator for Kubernetes is then shut down. Then, a temporary MySQL server is configured to use the remaining data directory, which then becomes the donor to initialize a new InnoDB Cluster with a single instance. After the data is cloned, the temporary (donor) server is shutdown, the old data directory is deleted, and the new InnodB Cluster is scaled up to three instances.

Perform the Upgrade

`Step #1`: Terminate the old InnoDB Cluster and remove all data directories except for one. This example uses kubectl and observes the termination process:

```
$> kubectl delete innodbcluster mycluster
$> kubectl get pods -w
```

Deleting an InnoDB Cluster does not remove its associated PersistentVolumeClaims, as seen with:

```
$> kubectl get pvc
```

Delete all but the one with the highest number. For example, with a three-node cluster keep the one with index 2:

```
$> kubectl delete pvc datadir-mycluster-0  datadir-mycluster-1
```

Be careful to keep one as otherwise the data can not be recovered; in this case, do not delete datadir-mycluster-2.

`Step #2`: Terminate the MySQL Operator for Kubernetes; do so by deleting the associated Deployment that controls it:

```
$> kubectl delete deployment -n mysql-operator mysql-operator
```

`Step #3`: Create a temporary MySQL Server using the datadir PersistentVolumeClaim, but first store the credentials in a Secret:

```
$> kubectl create secret generic myfixer \
     --from-literal=rootUser="root" \
     --from-literal=rootPassword="YOUR_PASSWORD"
```

This Secret is used to set up the temporary server, and also used to clone the data into the new InnoDB Cluster.

Save the following manifest to a file, say to `myfixer.yaml`, and then apply it. If datadir-mycluster-2 is not the datadir PVC you kept then modify the name accordingly.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mycnf
data:
  my.cnf: |
    [mysqld]
    plugin_load_add=auth_socket.so
    loose_auth_socket=FORCE_PLUS_PERMANENT
    skip_log_error
    log_error_verbosity=3
    skip_log_bin
    skip_slave_start=1
---
apiVersion: v1
kind: Pod
metadata:
  name: myfixer
spec:
  restartPolicy: Never
  containers:
  - image: mysql/mysql-server:8.0.30
    imagePullPolicy: IfNotPresent
    name: myfixer
```

```
    args: [ 'mysqld', '--defaults-file=/mycnf/my.cnf' ]
    env:
    - name: MYSQL_ROOT_PASSWORD
      valueFrom:
        secretKeyRef:
          key: rootPassword
          name: myfixer

    volumeMounts:
    - name: datadir
      mountPath: /var/lib/mysql
    - name: mycnf
      mountPath: /mycnf

  volumes:
  - name: datadir
    persistentVolumeClaim:
      claimName: datadir-mycluster-2
  - name: mycnf
    configMap:
      name: mycnf
```

Apply it:

```
$> kubectl apply -f myfixer.yaml
```

Retrieve the IP of the Pod when it's ready:

```
$> kubectl get pod -o wide myfixer
```

The sixth column has the IP address, you can verify the server started correctly by using a shell session using the credentials you stored (enter rootUser's password when you see "If you don't see a command prompt, try pressing enter."):

```
$> kubectl run testshell --restart=Never --rm --image=mysql/mysql-operator:8.0.30-2.0.6 -it -- mysqlsh -urc
```

`Step #4`: Deploy a new MySQL Operator for Kubernetes, as describe in the installation documentation. For example:

```
$> kubectl apply -f https://raw.githubusercontent.com/mysql/mysql-operator/trunk/deploy/deploy-crds.yaml
$> kubectl apply -f https://raw.githubusercontent.com/mysql/mysql-operator/trunk/deploy/deploy-operator.yam
```

`Step #5`: Deploy your new InnoDB Cluster following the installation documentation, and by using the temporary server created earlier as the donor; but first create a secret for your administrative user as described in the manual. For example:

```
$> kubectl create secret generic mypwds \
        --from-literal=rootUser="root" \
        --from-literal=rootHost="%" \
        --from-literal=rootPassword="YOUR_PASSWORD"
```

The following manifest shows the initDB definition, and this example assumes to a file named `ic.yaml`.

```
apiVersion: mysql.oracle.com/v2
kind: InnoDBCluster
metadata:
  name: mycluster
spec:
  tlsUseSelfSigned: true
  instances: 1              # Important deploy a single instance only
  secretName: mypwds        # Name of Secret for new InnoDBCluster
  version: 8.0.30           # Important must equal the server used as temporary donor
  router:
    instances: 1
  initDB:
    clone:
      donorUrl: root@{IP_HERE}:3306  # Use your myfixer user/ip here
      secretKeyRef:
```

```
        name: myfixer # Secret on the temporary server
  # Add custom options like configuration settings, storage configuration etc. as needed
```

Apply it:

```
$> kubectl apply -f ic.yaml
```

Now observe the status until the single MySQL instance cluster is ready. The time needed depends on the amount of data to clone.

```
$> kubectl get ic mycluster -w
```

`Step #6`: Remove the temporary server and scale up the InnoDB Cluster.

After confirming your new InnoDB Cluster is online and has your cloned data, remove the temporary server and its associated configuration, and also the old data directory's PersistentVolumeClaim:

```
$> kubectl delete -f  myfixer.yaml
$> kubectl delete secret myfixer

$> kubectl delete pvc datadir-mycluster-2
```

Scale up the InnoDB Cluster MySQL instances as desired, for example:

```
$> kubectl patch ic mycluster --type=merge -p '{"spec": {"instances": 3}}'
```

Chapter 4 Connecting to MySQL InnoDB Cluster

Table of Contents

This section utilizes the *{innodbclustername}.{namespace}.svc.cluster.local* form when connecting; and typically refers to the *{innodbclustername}* shorthand form that assumes the default namespace. See Section 3.4, "MySQL InnoDB Cluster Service Explanation" for additional information.

4.1 Connect with MySQL Shell

Create a new container with MySQL Shell to administer a MySQL InnoDB Cluster. This is the preferred method, although every MySQL Operator for Kubernetes and MySQL InnoDB Cluster container also has MySQL Shell installed if you need to troubleshoot a specific pod.

These examples assume the InnoDB Cluster is named 'mycluster' and using the 'default' namespace.

Create the new container with MySQL Shell; this example uses the MySQL Operator for Kubernetes image but other images work too, such as `mysql/mysql-server:8.0`.

This example creates a new container named "myshell" using the "mysql/mysql-operator" image, and immediately executes MySQL Shell:

```
$> kubectl run --rm -it myshell --image=mysql/mysql-operator -- mysqlsh
If you don't see a command prompt, try pressing enter.

MySQL JS >
```

Now connect to the InnoDB Cluster from within MySQL Shell's interface:

```
MySQL JS>  \connect root@mycluster

Creating a session to 'root@mycluster'
Please provide the password for 'root@mycluster': ******

MySQL mycluster JS>
```

The `root@mycluster` shorthand works as it assumes port 3306 (MySQL Router redirects to 6446) and the `default` namespace.

Optionally pass in additional arguments to `mysqlsh`, for example:

```
$> kubectl run --rm -it myshell --image=mysql/mysql-operator -- mysqlsh root@mycluster --sql
If you don't see a command prompt, try pressing enter.
******

MySQL mycluster SQL>
```

The "******" represents entering the MySQL user's password to MySQL Shell as MySQL Shell prompts for a password by default. The `root@mycluster` represents user root on host *mycluster*, and assumes the *default* namespace. Setting "*-sql* initiates MySQL Shell into SQL mode.

Troubleshooting a Specific Container

Every MySQL Operator for Kubernetes and MySQL InnoDB Cluster container has MySQL Shell installed, so for troubleshooting you may need to connect to a specific pod in the cluster. For example, connecting to a pod named mycluster-0:

```
$> kubectl --namespace default exec -it mycluster-0 -- bash
```

```
Defaulted container "sidecar" out of: sidecar, mysql, initconf (init), initmysql (init)
bash-4.4#

bash-4.4# mysqlsh root@localhost
Please provide the password for 'root@localhost': ******

...
```

4.2 Connect with Port Forwarding

Optionally use port forwarding to create a redirection from your local machine to easily use a MySQL client such as MySQL Workbench. We'll use port 3306 for a read-write connection to the primary on port 6446:

```
$> kubectl port-forward service/mycluster 3306

Forwarding from 127.0.0.1:3306 -> 6446
Forwarding from [::1]:3306 -> 6446
```

To test, open a second terminal using the MySQL command line or MySQL Shell with the InnoDB Cluster user's credentials:

```
$> mysql -h127.0.0.1 -uroot -p
```

To demonstrate the connection to a local MySQL instance:

```
mysql> select @@hostname;
+-------------+
| @@hostname  |
+-------------+
| mycluster-0 |
+-------------+
```

Not seeing a port-forward to 127.0.0.1:3306 in this example means a local MySQL installation is likely installed and active on the system.

Using port names instead of port numbers also works:

```
$> kubectl port-forward service/mycluster mysql
Forwarding from 127.0.0.1:3306 -> 6446
Forwarding from [::1]:3306 -> 6446
^C

$> kubectl port-forward service/mycluster mysql-ro
Forwarding from 127.0.0.1:6447 -> 6447
Forwarding from [::1]:6447 -> 6447
```

A list of port names with their associated ports:

```
mysql:            3306
mysqlx:           33060
mysql-alternate:  6446
mysqlx-alternate: 6448
mysql-ro:         6447
mysqlx-ro:        6449
router-rest:      8443
```

For a list of all ports used by MySQL services, see MySQL Port Reference. The ports used here are from MySQL Router.

Chapter 5 Private Registries

Table of Contents

Tasks related to using private registries. This section is a work-in-progress.

5.1 Install MySQL Operator for Kubernetes from Private Registry using Helm

If the private registry is not authenticated, and after pushing the MySQL Operator for Kubernetes image to your private registry, execute the following on the host where helm is installed; and adjust the variable values as needed:

```
export REGISTRY="..."   # like 192.168.20.199:5000
export REPOSITORY="..." # like "mysql"
export NAMESPACE="mysql-operator"
helm install mysql-operator helm/mysql-operator \
    --namespace $NAMESPACE \
    --create-namespace \
    --set image.registry=$REGISTRY \
    --set image.repository=$REPOSITORY \
    --set envs.imagesDefaultRegistry="$REGISTRY" \
    --set envs.imagesDefaultRepository="$REPOSITORY"
```

Authenticated private registries need to create a namespace for MySQL Operator for Kubernetes, and also add a Kubernetes docker-registry secret in the namespace; then execute `helm install` with arguments that look similar to:

```
export REGISTRY="..."   # like 192.168.20.199:5000
export REPOSITORY="..." # like "mysql"
export NAMESPACE="mysql-operator"
export DOCKER_SECRET_NAME="priv-reg-secret"

kubectl create namespace $NAMESPACE

kubectl -n $NAMESPACE create secret docker-registry $DOCKER_SECRET_NAME \
        --docker-server="https://$REGISTRY/v2/" \
        --docker-username=user --docker-password=pass \
        --docker-email=user@example.com

helm install mysql-operator helm/mysql-operator \
        --namespace $NAMESPACE \
        --set image.registry=$REGISTRY \
        --set image.repository=$REPOSITORY \
        --set image.pullSecrets.enabled=true \
        --set image.pullSecrets.secretName=$DOCKER_SECRET_NAME \
        --set envs.imagesPullPolicy='IfNotPresent' \
        --set envs.imagesDefaultRegistry="$REGISTRY" \
        --set envs.imagesDefaultRepository="$REPOSITORY"
```

To confirm the installation, check the status with commands such as `helm list -n $NAMESPACE` and `kubectl -n $NAMESPACE get pods`.

5.2 Install InnoDB Cluster from Private Registry using Helm

For example:

```
export REGISTRY="..."   # like 192.168.20.199:5000
export REPOSITORY="..." # like "mysql"
export NAMESPACE="mynamespace"
export DOCKER_SECRET_NAME="priv-reg-secret"

$> kubectl create namespace $NAMESPACE

$> kubectl -n $NAMESPACE create secret docker-registry $DOCKER_SECRET_NAME \
           --docker-server="https://$REGISTRY/v2/" \
           --docker-username=user --docker-password=pass \
           --docker-email=user@example.com

$> helm install mycluster mysql-operator/mysql-innodbcluster \
           --namespace $NAMESPACE \
           --set credentials.root.user='root' \
           --set credentials.root.password='sakila' \
           --set credentials.root.host='%' \
           --set serverInstances=3 \
           --set routerInstances=1 \
           --set image.registry=$REGISTRY \
           --set image.repository=$REPOSITORY \
           --set image.pullSecrets.enabled=true \
           --set image.pullSecrets.secretName=$DOCKER_SECRET_NAME \
```

5.3 Copy Image to Private Registry using Docker

Using air-gapped or sanctioned images to avoid pulling images from the internet is another use case and described here.

Note

MySQL Operator for Kubernetes requires these three container images to function: MySQL Operator for Kubernetes, MySQL Router, and MySQL Server.

1. Choose the desired MySQL Operator for Kubernetes version. For example, latest is defined in helm/mysql-operator/Chart.yaml. For example, `8.0.31-2.0.7`.
2. Execute `docker pull mysql/mysql-operator:VERSION` where VERSION is the desired MySQL Operator for Kubernetes version.
3. Execute `docker save mysql/mysql-operator:VERSION -o mysql-operator.tar` to export the container image where VERSION is the desired MySQL Operator for Kubernetes version.
4. Copy `mysql-operator.tar` to a host with access to the private registry.
5. Execute `docker load -i mysql-operator.yaml` to load the image into the local Docker cache on that host.
6. Execute `docker tag mysql/mysql-server:VERSION registry:port/repo/mysql-server:VERSION` to retag the image as preparation for pushing to the private registry; adjust VERSION accordingly.
7. Execute `docker push registry:port/repo/mysql-server:VERSION` to push the newly created tag to the private registry; adjust VERSION accordingly.
8. If you won't need the image from the importing host cache, then you can delete it with `docker rmi mysql/mysql-operator:VERSION registry:port/repo/mysql-server:VERSION`. This removes it from the host but the registry itself won't be affected. Adjust VERSION accordingly.

Alternatively, you can use the following commands to pull and push in one command. Execute it on a host with DockerHub access. If applicable, this host also needs access to the secure (bastion) host that can access the private registry. Modify the variable values to fit your needs. The command does not consume local space for a tarball but will stream the container image over SSH.

```
export BASTION_USER='k8s'
export BASTION_HOST='k8'
export REGISTRY="..." # for example 192.168.20.199:5000
export REPOSITORY="..." # for example mysql
export OPERATOR_VERSION=$(grep appVersion helm/mysql-operator/Chart.yaml | cut -d '"' -f2)
docker pull mysql/mysql-operator:$OPERATOR_VERSION
docker save mysql/mysql-operator:$OPERATOR_VERSION | \
    ssh $BASTION_USER@$BASTION_HOST \
         "docker load && \
          docker tag mysql/mysql-operator:$OPERATOR_VERSION $REGISTRY/$REPOSITORY/mysql-operator:$OPERA
          docker push $REGISTRY/$REPOSITORY/mysql-operator:$OPERATOR_VERSION && \
          docker rmi mysql/mysql-operator:$OPERATOR_VERSION $REGISTRY/$REPOSITORY/mysql-operator:$OPERA
docker rmi mysql/mysql-operator:$OPERATOR_VERSION
```

5.4 Copy Image to Private Registry using Skopeo

Similar to Section 5.3, “Copy Image to Private Registry using Docker”, but you might use Skopeo. Skopeo is a container utility that can also run as a container. The following example copies the operator image from DockerHub to a private registry. It needs to run on a host that has Docker or Podman, and also that has access to both DockerHub and your private registry. Change the variable names to fit your environment, and change docker to podman if using Podman. The `OPERATOR_VERSION` is the MySQL Operator for Kubernetes version, such as `8.0.31-2.0.7`.

```
export REGISTRY="..." # for example 192.168.20.199:5000
export REPOSITORY="..." # for example mysql
export OPERATOR_VERSION=$(grep appVersion helm/mysql-operator/Chart.yaml | cut -d '"' -f2)
docker run --rm quay.io/skopeo/stable copy docker://mysql/mysql-operator:$OPERATOR_VERSION docker://$RE
```

For authenticated private registries, append `--dest-creds user:pass` to the skopeo command. Also append `--dest-tls-verify=false` if it does not use TLS.

Chapter 6 MySQL Operator Cookbook

Table of Contents

Tasks related to using MySQL Operator for Kubernetes.

6.1 Handling MySQL Backups

There are three main topics related to MySQL backups:

- *Backup profile*: describes the general backup structure that includes storage, schedule, and MySQL Shell dump related options. Defining profiles is optional, and profiles are separated by name.
- *Backup request*: requesting a backup initiates a new object that creates a new pod to perform the backup.
- *Backup schedule*: defined as a cron expression for regular backups, or with no schedule when performing one-off backups.

See also Chapter 7, *MySQL Operator Custom Resource Properties* for a list of all `MySQLBackup` resource options.

Backup Profiles with backupProfiles

Backup profiles are defined and reused for regular backups and one-off backups using the backupProfiles specification object. A profile is defined and called from within the InnoDB Cluster specification object, or values can be defined from within the individual backup requests without a profile.

How a Backup is Created

MySQL Operator for Kubernetes supports the `dumpInstance()` command using MySQL Shell by defining the associated dumpInstance specification object that contains the `dumpOptions` and `storage` specification objects:

- The optional `dumpOptions` value is a dictionary of key-value pairs passed directly to MySQL Shell's DumpInstance() function. See Instance Dump Utility, Schema Dump Utility, and Table Dump Utility for a list of relevant options.

  MySQL Operator for Kubernetes adds definitions by default, such as defining `threads` based on what the system claims as its CPU count; but these values can be overridden.

- The `storage` configuration specification offers two options as of MySQL Operator for Kubernetes 8.0.29: `persistentVolumeClaim` or `ociObjectStorage` (OCI refers to Oracle Cloud Infrastructure).

  > **Note**
  >
  > *Limitations*: Restore capability is not available for persistentVolumeClaim as of MySQL Operator for Kubernetes 8.0.29, and ociObjectStorage use is specific to the Oracle Cloud Infrastructure (OCI).

- The `backupSchedules schedule` utilizes the Kubernetes CronJob controller for regular backups.

A PersistentVolumeClaim Scheduled Backup Example

This example uses PersistentVolumeClaim (PVC), sets a daily backup schedule, and defines a backup profile named "myfancyprofile" in the backupProfiles object.

Note

This example defines a single backupProfile and schedule but could define multiple profiles and schedules depending need. For example, a volatile table may have hourly backups in addition to the full nightly backup.

```
apiVersion: mysql.oracle.com/v2
kind: InnoDBCluster
metadata:
  name: mycluster
spec:
  instances: 3
  router:
    instances: 1
  secretName: mypwds
  tlsUseSelfSigned: true
  backupProfiles:
  - name: myfancyprofile  # Embedded backup profile
    dumpInstance:         # MySQL Shell Dump
      dumpOptions:
        excludeTables: "[world.country]" # Example to exclude one table
      storage:
        persistentVolumeClaim:
          claimName: myexample-pvc # store to this pre-existing PVC
      backupSchedules:
        - name: mygreatschedule
          schedule: "0 0 * * *" # Daily, at midnight
          backupProfileName:  myfancyprofile # reference the desired backupProfiles's name
          enabled: true # backup schedules can be temporarily disabled
```

This example requires a `PersistentVolumeClaim` definition named "myexample-pvc"; see the official Kubernetes Persistent Volumes documentation for `PersistentVolumeClaim` specifics. A simple example:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: myexample-pv
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /tmp
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myexample-pvc
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

The example "mycluster" InnoDB Cluster definition uses a secret named "mypwds" for its root user, for example:

```
$> kubectl create secret generic mypwds \
        --from-literal=rootUser=root \
        --from-literal=rootHost=% \
        --from-literal=rootPassword="sakila"
```

After creating the example InnoDB Cluster, you may want to execute a one-off backup using an existing profile, such as:

```
apiVersion: mysql.oracle.com/v2
kind: MySQLBackup
metadata:
    name: a-cool-one-off-backup
spec:
  clusterName: mycluster
  backupProfileName: myfancyprofile
```

Executing this creates a pod with a name similar to *a-cool-one-off-backup-20220330-215635-t6thv* that executes the backup, and it remains in a Completed state after the backup operation.

Using OciObjectStorage

Using the same example but for Oracle Cloud Infrastructure (OCI) instead of a PVC, modify dumpInstance.`storage` from PrivateVolumeClaim to an ociObjectStorage object similar to:

```
    dumpInstance:
      storage:
        ociObjectStorage:
          prefix: someprefix         # a prefix (directory) used for ObjectStorage
          bucketName: bucket         # the ObjectStorage bucket
          credentials: backup-apikey # a secret with credentials ...
```

The backup-apikey secret used in this OCI example looks similar to:

```
    apiVersion: v1
    kind: Secret
    type: Opaque
    metadata:
      name: backup-apikey
    stringData:
      fingerprint: 06:e9:e1:c6:e5:df:81:f3:......
      passphrase: ....
      privatekey: |
        -----BEGIN RSA PRIVATE KEY-----
        MIIEogIBAAKCAQEAwmQ1JGOGUBNwyJuq4msGpBfK24toKrWaqAkbZ1Z/XLOFLvEE
       ....
      region: us-ashburn-1..
      tenancy: ocid1.tenancy...
      user: ocid1.user.....
```

An example method to create the Secret; values are found in the configuration file downloaded from OCI, which is used with the OCI command-line tool.

```
$> kubectl create secret generic <secret_name> \
        --from-literal=user=<userid> \
        --from-literal=fingerprint=<fingerprint> \
        --from-literal=tenancy=<tenancy> \
        --from-literal=region=<region> \
        --from-literal=passphrase=<passphrase> \
        --from-file=privatekey=<path_to_api_key.pem>
```

Using profiles (`backupProfileName`) is optional, so instead it may look like the following with the same settings. This example restores to a new InnoDB Cluster from ociObjectStorage:

```
    apiVersion: mysql.oracle.com/v2
    kind: InnoDBCluster
    metadata:
      name: newcluster
    spec:
```

```
      instances: 3
      router:
        instances: 1
      secretName: newpwds
      tlsUseSelfSigned: true
      baseServerId: 2000
      initDB:
        dump:
          name: some-name
          storage:
            ociObjectStorage:
              prefix: someprefix
              bucketName: bucket
              credentials: restore-apikey
```

The secret (`restore-apikey`) could be the same as the backup example (`backup-apikey`) but may be a different user with different permissions, such as no write permissions to the OS.

Cloning

Data can be initialized using a backup or by cloning an existing and running MySQL instance using `iniDB` and its `donorURL` option:

```
apiVersion: mysql.oracle.com/v2
kind: InnoDBCluster
metadata:
  name: copycluster
spec:
  instances: 1
  secretName: pwds
  tlsUseSelfSigned: true
  initDB:
    clone:
      donorUrl: root@mycluster-0.mycluster-instances.testns.svc.cluster.local:3306
      secretKeyRef:
        name: donorpwds
```

The `donorpwds` secret contains a single field named rootPassword, so for example you could reuse the main secretName used when creating the original cluster (named `mypwds` in the example). This utilizes MySQL's cloning plugin, so standard limitations apply (such as requiring the same MySQL versions). Cloning can theoretically also be used for creating backups.

6.2 Bootstrap a MySQL InnoDB Cluster from a Dump using Helm

A MySQL InnoDB Cluster can be initialized with a database dump created by MySQL Shell or by MySQL Operator for Kubernetes. The backup could reside on a persistent volume accessible from the cluster, but our example uses an OCI Object Storage bucket.

Using an OCI Object Storage bucket

If you are boostrapping from OCI OS, then the following must be known:

- The credentials of the user who has access to OCI OS
- The OCI OS Object Prefix (plays the role of a directory). The following Helm variables must be set:
  - `initDB.dump.name`: a name for the dump that follows the Kubernetes rules for naming an identifier, such as *dump-20210916-140352*.
  - `initDB.dump.ociObjectStorage.prefix`: the prefix from list above
  - `initDB.dump.ociObjectStorage.bucketName`: the bucket name from the list above
  - `initDB.dump.ociObjectStorage.credentials`: name of the Kubernetes secret that holds the credentials for accessing the OCI OS bucket

For the credentials secret, the following information is needed: OCI OS User Name, Fingerprint, Tenancy Name, Region Name, Passphrase, and the Private Key of the user.

- The OCI OS Bucket Name

The OCI command-line tool provides this information in `$HOME/config` under the `[DEFAULT]` section. Once obtained, execute:

```
export NAMESPACE="mynamespace"
export OCI_CREDENTIALS_SECRET_NAME="oci-credentials"
export OCI_USER="..."                # like ocid1.user.oc1....
export OCI_FINGERPRINT="..."         # like 90:01:..:..:....
export OCI_TENANCY="..."             # like ocid1.tenancy.oc1...
export OCI_REGION="..."              # like us-ashburn-1
export OCI_PASSPHRASE="..."          # set to empty string if no passphrase
export OCI_PATH_TO_PRIVATE_KEY="..." # like $HOME/.oci/oci_api_key.pem

kubectl -n $NAMESPACE create secret generic $OCI_CREDENTIALS_SECRET_NAME \
        --from-literal=user="$OCI_USER" \
        --from-literal=fingerprint="$OCI_FINGERPRINT" \
        --from-literal=tenancy="$OCI_TENANCY" \
        --from-literal=region="$OCI_REGION" \
        --from-literal=passphrase="$OCI_PASSPHRASE" \
        --from-file=privatekey="$OCI_PATH_TO_PRIVATE_KEY"
```

With the OCI secret created, now create the cluster that'll be initialized from the dump in OCI OS:

```
export NAMESPACE="mynamespace"
export OCI_DUMP_PREFIX="..."  # like dump-20210916-140352
export OCI_BUCKET_NAME="..."  # like idbcluster_backup
export OCI_CREDENTIALS_SECRET_NAME="oci-credentials"
kubectl create namespace $NAMESPACE
helm install mycluster mysql-operator/mysql-innodbcluster \
        --namespace $NAMESPACE \
        --set credentials.root.user='root' \
        --set credentials.root.password='sakila' \
        --set credentials.root.host='%' \
        --set serverInstances=3 \
        --set routerInstances=1 \
        --set initDB.dump.name="initdb-dump" \
        --set initDB.dump.ociObjectStorage.prefix="$OCI_DUMP_PREFIX" \
        --set initDB.dump.ociObjectStorage.bucketName="$OCI_BUCKET_NAME" \
        --set initDB.dump.ociObjectStorage.credentials="$OCI_CREDENTIALS_SECRET_NAME"
```

6.3 Viewing Logs

Information helpful for debugging and finding relevant log information.

Log locations include each InnoDBCluster Pod, which are divided into a set of containers. There are two operative containers (`mysql`, and `sidecar`) and three initializer containers (`initconf`, `initmysql`, and `fixdatadir`) as described below here:

Table 6.1 Containers associated with an InnoDBCluster Pod

Container Name	Description
`sidecar`	Initialization, including initial setup of data (initDB) and ongoing maintenance tasks for a specific instance, such as TLS certification updates
`mysql`	The MySQL Server itself
`initconf`	It prepares MySQL configuration files for a specific host. For example, to view its ConfigMap: *kubectl get cm {cluster_name}-initconf -o json*
`initmysql`	Initializes the MySQL Server, including its data directory.

Container Name	Description
`fixdatadir`	Sets appropriate permissions and ownership of the MySQL data directory, upon initialization.

There's also the dynamic MySQL Operator for Kubernetes and MySQL Router pods.

Examples that assume a basic setup as per `samples/sample-cluster.yaml` which looks like:

```
$> kubectl get pods

NAME                                READY   STATUS    RESTARTS   AGE
mycluster-0                         2/2     Running   0          99m
mycluster-1                         2/2     Running   0          99m
mycluster-2                         2/2     Running   0          99m
mycluster-router-6d49485474-ftw9r   1/1     Running   0          97m

$> kubectl get pods --namespace mysql-operator

NAME                              READY   STATUS    RESTARTS   AGE
mysql-operator-586f9f5d5b-7wtgl   1/1     Running   0          3h48m
```

Viewing operational Pod logs for debugging active operations:

```
$> kubectl logs mycluster-0 -c sidecar
...
[2022-04-21 19:15:08,571] sidecar              [INFO    ] My pod is mycluster-0 in default
[2022-04-21 19:15:08,571] sidecar              [INFO    ] Bootstrapping
[2022-04-21 19:15:10,600] sidecar              [INFO    ] Configuring mysql pod default/mycluster-0, confi
[2022-04-21 19:15:10,626] sidecar              [INFO    ] Creating root account root@%
[2022-04-21 19:15:10,670] sidecar              [INFO    ] Creating account mysqladmin@%
[2022-04-21 19:15:10,694] sidecar              [INFO    ] Admin account created
...
```

```
$> kubectl logs mycluster-0 -c mysql

[Entrypoint] MySQL Docker Image 8.0.29-1.2.8-server
2022-04-21T19:15:05.969998Z 0 [Note] [MY-010747] [Server] Plugin 'FEDERATED' is disabled.
2022-04-21T19:15:05.971625Z 0 [Note] [MY-010733] [Server] Shutting down plugin 'MyISAM'
2022-04-21T19:15:05.971700Z 0 [Note] [MY-010733] [Server] Shutting down plugin 'CSV'
[Entrypoint] Starting MySQL 8.0.29-1.2.8-server
2022-04-21T19:15:07.085923Z 0 [Note] [MY-010949] [Server] Basedir set to /usr/.
2022-04-21T19:15:07.085959Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.29) starting as p
2022-04-21T19:15:07.094129Z 0 [Note] [MY-012366] [InnoDB] Using Linux native AIO
2022-04-21T19:15:07.094464Z 0 [Note] [MY-010747] [Server] Plugin 'FEDERATED' is disabled.
2022-04-21T19:15:07.155815Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
...
```

```
$> kubectl logs mycluster-router-6d49485474-ftw9r
...
# Bootstrapping MySQL Router instance at '/tmp/mysqlrouter'...
...
## MySQL Classic protocol
- Read/Write Connections: localhost:6446
- Read/Only Connections:  localhost:6447
...
```

```
$> kubectl logs mysql-operator-586f9f5d5b-7wtgl -n mysql-operator
...
2022-04-21 17:06:13: Info: Credential store mechanism is going to be disabled.
2022-04-21 17:06:13: Info: Loading startup files...
2022-04-21 17:06:13: Info: Loading plugins...
[2022-04-21 17:06:14,758] kopf.activities.star [INFO    ] MySQL Operator/operator.py=2.0.4 timestamp=2022-
[2022-04-21 17:06:14,758] kopf.activities.star [INFO    ] OPERATOR_VERSION   =2.0.4
[2022-04-21 17:06:14,758] kopf.activities.star [INFO    ] OPERATOR_EDITION   =community
[2022-04-21 17:06:14,758] kopf.activities.star [INFO    ] OPERATOR_EDITIONS  =['community', 'enterprise']
[2022-04-21 17:06:14,758] kopf.activities.star [INFO    ] SHELL_VERSION      =8.0.29
[2022-04-21 17:06:14,758] kopf.activities.star [INFO    ] DEFAULT_VERSION_TAG=8.0.29
[2022-04-21 17:06:14,758] kopf.activities.star [INFO    ] SIDECAR_VERSION_TAG=8.0.29-2.0.4
...
Incremental state recovery is now in progress.
```

```
* Waiting for distributed recovery to finish...
...
[2022-04-21 19:15:54,926] kopf.objects         [INFO    ] cluster probe: status=ClusterDiagStatus.ONLIN
                                                            online=[<MySQLPod mycluster-0>, <MySQLPod myc
[2022-04-21 19:15:54,930] kopf.objects         [INFO    ] Handler 'on_pod_event' succeeded.
...
```

Viewing Pods specific to the InnoDBCluster's initialization:

```
$> kubectl logs mycluster-0 -c initmysql
...
[Entrypoint] Initializing database
2022-04-21T19:14:40.315937Z 0 [Note] [MY-010949] [Server] Basedir set to /usr/.
2022-04-21T19:14:40.315977Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (mysqld 8.0.29) initializi
2022-04-21T19:14:40.317974Z 0 [Note] [MY-010458] [Server] --initialize specified on an existing data di
2022-04-21T19:14:40.323039Z 0 [Note] [MY-010938] [Server] Generating a new UUID: 4af9d5b7-c1a7-11ec-966
2022-04-21T19:14:40.329396Z 0 [Note] [MY-012366] [InnoDB] Using Linux native AIO
2022-04-21T19:14:40.330470Z 0 [Note] [MY-010747] [Server] Plugin 'FEDERATED' is disabled.
2022-04-21T19:14:40.398051Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
...
2022-04-21T19:14:42.103049Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2022-04-21T19:14:42.201557Z 1 [Note] [MY-011088] [Server] Data dictionary initializing version '80023'.
2022-04-21T19:14:43.367575Z 1 [Note] [MY-010007] [Server] Installed data dictionary with version 80023
2022-04-21T19:14:43.678363Z 2 [Note] [MY-011019] [Server] Created system views with I_S version 80023.
...
[Entrypoint] running /docker-entrypoint-initdb.d/initdb-localroot.sql
...
2022-04-21T19:15:01.834127Z 12 [System] [MY-013172] [Server] Received SHUTDOWN from user root. Shutting
...
2022-04-21T19:15:03.832074Z 0 [System] [MY-010910] [Server] /usr/sbin/mysqld: Shutdown complete (mysqld
[Entrypoint] Server shut down
[Entrypoint] MySQL init process done. Ready for start up.
[Entrypoint] MYSQL_INITIALIZE_ONLY is set, exiting without starting MySQL...
```

```
$> kubectl logs mycluster-0 -c initconf

2022-04-21 19:14:35: Info: Credential store mechanism is going to be disabled.
2022-04-21 19:14:35: Info: Loading startup files...
2022-04-21 19:14:35: Info: Loading plugins...
2022-04-21T19:14:37 - [INFO] [initmysql] MySQL Operator/init_main.py=2.0.4 timestamp=2022-04-21T14:43:1
2022-04-21T19:14:37 - [INFO] [initmysql] Configuring mysql pod default/mycluster-0, datadir=/var/lib/my
total 0
/mnt:
total 8
drwxrwsrwx 3 root mysql 4096 Apr 21 19:14 initconf
drwxrwsrwx 2 root mysql 4096 Apr 21 19:14 mycnfdata

/mnt/initconf:
total 0
lrwxrwxrwx 1 root mysql 19 Apr 21 19:14 00-basic.cnf -> ..data/00-basic.cnf
lrwxrwxrwx 1 root mysql 31 Apr 21 19:14 01-group_replication.cnf -> ..data/01-group_replication.cnf
lrwxrwxrwx 1 root mysql 17 Apr 21 19:14 02-ssl.cnf -> ..data/02-ssl.cnf
lrwxrwxrwx 1 root mysql 19 Apr 21 19:14 99-extra.cnf -> ..data/99-extra.cnf
lrwxrwxrwx 1 root mysql 27 Apr 21 19:14 initdb-localroot.sql -> ..data/initdb-localroot.sql
lrwxrwxrwx 1 root mysql 23 Apr 21 19:14 livenessprobe.sh -> ..data/livenessprobe.sh
lrwxrwxrwx 1 root mysql 16 Apr 21 19:14 my.cnf.in -> ..data/my.cnf.in
lrwxrwxrwx 1 root mysql 24 Apr 21 19:14 readinessprobe.sh -> ..data/readinessprobe.sh

/mnt/mycnfdata:
total 0
2022-04-21T19:14:38 - [INFO] [initmysql] Setting up configurations for mycluster-0 server_id=1000
                                         report_host=mycluster-0.mycluster-instances.default.svc.cluste
2022-04-21T19:14:38 - [INFO] [initmysql] Configuration done
```

For `initconf`, you might view their ConfigMap, for example:

```
$> kubectl get configmap mycluster-initconf -o yaml
```

Copied here is the [data] object:

```
data:
  00-basic.cnf: |
    # Basic configuration.
    # Do not edit.
    [mysqld]
    plugin_load_add=auth_socket.so
    loose_auth_socket=FORCE_PLUS_PERMANENT
    skip_log_error
    log_error_verbosity=3
  01-group_replication.cnf: |
    # GR and replication related options
    # Do not edit.
    [mysqld]
    log_bin=mycluster
    enforce_gtid_consistency=ON
    gtid_mode=ON
    relay_log_info_repository=TABLE
    skip_slave_start=1
  02-ssl.cnf: |
    # SSL configurations
    # Do not edit.
    [mysqld]
    # ssl-ca=/etc/mysql-ssl/ca.pem
    # ssl-crl=/etc/mysql-ssl/crl.pem
    # ssl-cert=/etc/mysql-ssl/tls.crt
    # ssl-key=/etc/mysql-ssl/tls.key

    loose_group_replication_recovery_use_ssl=1
    # loose_group_replication_recovery_ssl_verify_server_cert=1

    # loose_group_replication_recovery_ssl_ca=/etc/mysql-ssl/ca.pem
    ## loose_group_replication_recovery_ssl_crl=/etc/mysql-ssl/crl.pem
    # loose_group_replication_recovery_ssl_cert=/etc/mysql-ssl/tls.crt
    # loose_group_replication_recovery_ssl_key=/etc/mysql-ssl/tls.key
  99-extra.cnf: |
    # Additional user configurations taken from spec.mycnf in InnoDBCluster.
    # Do not edit directly.
    [mysqld]
    innodb_buffer_pool_size=200M
    innodb_log_file_size=2G
  my.cnf.in: |
    # Server identity related options (not shared across instances).
    # Do not edit.
    [mysqld]
    server_id=@@SERVER_ID@@
    report_host=@@HOSTNAME@@
    datadir=/var/lib/mysql
    loose_mysqlx_socket=/var/run/mysqld/mysqlx.sock
    socket=/var/run/mysqld/mysql.sock
    local-infile=1

    [mysql]
    socket=/var/run/mysqld/mysql.sock

    [mysqladmin]
    socket=/var/run/mysqld/mysql.sock

    !includedir /etc/my.cnf.d
```

Chapter 7 MySQL Operator Custom Resource Properties

Resource Types

- InnoDBCluster
- MySQLBackup

InnoDBCluster

Table 7.1 Spec table for InnoDBCluster

Name	Type	Description	Required
apiVersion	string	mysql.oracle.com/v2	true
kind	string	InnoDBCluster	true
metadata	object	Refer to the Kubernetes API documentation	true
`spec`	object		true
`status`	object		false

InnoDBCluster.spec

Parent

Table 7.2 Spec table for InnoDBCluster.spec

Name	Type	Description	Required
`secretName`	string	Name of a generic type Secret containing root/default account password	true
`backupProfiles`	[]object	Backup profile specifications for the cluster, which can be referenced from backup schedules and one-off backup jobs	false
`backupSchedules`	[]object	Schedules for periodically executed backups	false
`baseServerId`	integer	Base value for MySQL server_id for instances in the cluster • `Default`: 1000 • `Minimum`: 0 • `Maximum`: 4294967195	false
`datadirVolumeClaimTemplate`	object	Template for a PersistentVolumeClaim, to be used as datadir	false
`edition`	string	MySQL Server Edition (community or enterprise)	false

Name	Type	Description	Required
imagePullPolicy	string	Defaults to Always, but set to IfNotPresent in deploy-operator.yaml when deploying Operator	false
imagePullSecrets	[]object		false
imageRepository	string	Repository from where images must be pulled from; defaults to mysql for community and container-registry.oracle.com/mysql for enterprise	false
initDB	object		false
instances	integer	Number of MySQL replica instances for the cluster • Default: 1 • Minimum: 1 • Maximum: 9	false
keyring	object	Keyring specification	false
mycnf	string	Custom configuration additions for my.cnf	false
podAnnotations	object		false
podLabels	object		false
podSpec	object		false
router	object	MySQL Router specification	false
serviceAccountName	string		false
tlsCASecretName	string	Name of a generic type Secret containing CA (ca.pem) and optional CRL (crl.pem) for SSL	false
tlsSecretName	string	Name of a TLS type Secret containing Server certificate and private key for SSL	false
tlsUseSelfSigned	boolean	Enables use of self-signed TLS certificates, reducing or disabling TLS based security verifications • Default: false	false
version	string	MySQL Server version	false

InnoDBCluster.spec.backupProfiles[index]

Parent

Table 7.3 Spec table for InnoDBCluster.spec.backupProfiles[index]

Name	Type	Description	Required
`name`	string	Embedded backup profile, referenced as backupProfileName elsewhere	true
`dumpInstance`	object		false
`podAnnotations`	object		false
`podLabels`	object		false
`snapshot`	object		false

InnoDBCluster.spec.backupProfiles[index].dumpInstance

Parent

Table 7.4 Spec table for InnoDBCluster.spec.backupProfiles[index].dumpInstance

Name	Type	Description	Required
`dumpOptions`	object	A dictionary of key-value pairs passed directly to MySQL Shell's DumpInstance()	false
`storage`	object		false

InnoDBCluster.spec.backupProfiles[index].dumpInstance.storage

Parent

Table 7.5 Spec table for InnoDBCluster.spec.backupProfiles[index].dumpInstance.storage

Name	Type	Description	Required
`ociObjectStorage`	object		false
`persistentVolumeClaim`	object	Specification of the PVC to be used. Used 'as is' in pod executing the backup.	false
`s3`	object		false

InnoDBCluster.spec.backupProfiles[index].dumpInstance.storage.ociObjectStora

Parent

Table 7.6 Spec table for InnoDBCluster.spec.backupProfiles[index].dumpInstance.storage.ociObjectStorage

Name	Type	Description	Required
`bucketName`	string	Bucket name where backup is stored	true

Name	Type	Description	Required
credentials	string	Secret name with data for accessing the bucket	true
prefix	string	Path in bucket where backup is stored	false

InnoDBCluster.spec.backupProfiles[index].dumpInstance.storage.s3

Parent

Table 7.7 Spec table for InnoDBCluster.spec.backupProfiles[index].dumpInstance.storage.s3

Name	Type	Description	Required
bucketName	string	Name of the bucket where the dump is stored	true
config	string	Secret name with S3 configuration and credentials	true
endpoint	string	Override endpoint URL	false
prefix	string	Path in the bucket where the dump files are stored	false
profile	string	Profile being used in configuration files • Default:	false

InnoDBCluster.spec.backupProfiles[index].snapshot

Parent

Table 7.8 Spec table for InnoDBCluster.spec.backupProfiles[index].snapshot

Name	Type	Description	Required
storage	object		false

InnoDBCluster.spec.backupProfiles[index].snapshot.storage

Parent

Table 7.9 Spec table for InnoDBCluster.spec.backupProfiles[index].snapshot.storage

Name	Type	Description	Required
ociObjectStorage	object		false
persistentVolumeClaim	object	Specification of the PVC to be used. Used 'as is' in pod executing the backup.	false
s3	object		false

InnoDBCluster.spec.backupProfiles[index].snapshot.storage.ociObjectStorage

Parent

Table 7.10 Spec table for InnoDBCluster.spec.backupProfiles[index].snapshot.storage.ociObjectStorage

Name	Type	Description	Required
bucketName	string	Bucket name where backup is stored	true
credentials	string	Secret name with data for accessing the bucket	true
prefix	string	Path in bucket where backup is stored	false

InnoDBCluster.spec.backupProfiles[index].snapshot.storage.s3

Parent

Table 7.11 Spec table for InnoDBCluster.spec.backupProfiles[index].snapshot.storage.s3

Name	Type	Description	Required
bucketName	string	Name of the bucket where the dump is stored	true
config	string	Secret name with S3 configuration and credentials	true
endpoint	string	Override endpoint URL	false
prefix	string	Path in the bucket where the dump files are stored	false
profile	string	Profile being used in configuration files • Default:	false

InnoDBCluster.spec.backupSchedules[index]

Parent

Table 7.12 Spec table for InnoDBCluster.spec.backupSchedules[index]

Name	Type	Description	Required
name	string	Name of the backup schedule	true
schedule	string	The schedule of the job, syntax as a cron expression	true
backupProfile	object	backupProfile specification if backupProfileName is not specified	false
backupProfileName	string	Name of the backupProfile to be used	false
deleteBackupData	boolean	Whether to delete the backup data in case the	false

Name	Type	Description	Required
		MySQLBackup object created by the job is deleted • Default: false	
enabled	boolean	Whether the schedule is enabled or not • Default: true	false

InnoDBCluster.spec.backupSchedules[index].backupProfile

Parent

Description: backupProfile specification if backupProfileName is not specified

Table 7.13 Spec table for InnoDBCluster.spec.backupSchedules[index].backupProfile

Name	Type	Description	Required
dumpInstance	object		false
podAnnotations	object		false
podLabels	object		false

InnoDBCluster.spec.backupSchedules[index].backupProfile.dumpInstance

Parent

Table 7.14 Spec table for InnoDBCluster.spec.backupSchedules[index].backupProfile.dumpInstance

Name	Type	Description	Required
dumpOptions	object	A dictionary of key-value pairs passed directly to MySQL Shell's DumpInstance()	false
storage	object		false

InnoDBCluster.spec.backupSchedules[index].backupProfile.dumpInstance.storage

Parent

Table 7.15 Spec table for InnoDBCluster.spec.backupSchedules[index].backupProfile.dumpInstance.storage

Name	Type	Description	Required
ociObjectStorage	object		false
persistentVolumeClaim	object	Specification of the PVC to be used. Used 'as is' in pod executing the backup.	false
s3	object		false

InnoDBCluster.spec.backupSchedules[index].backupProfile.dumpInstance.storage.c

Parent

Table 7.16 Spec table for InnoDBCluster.spec.backupSchedules[index].backupProfile.dumpInstance.storage.ociObjectStorage

Name	Type	Description	Required
`bucketName`	string	Bucket name where backup is stored	true
`credentials`	string	Secret name with data for accessing the bucket	true
`prefix`	string	Path in bucket where backup is stored	false

InnoDBCluster.spec.backupSchedules[index].backupProfile.dumpInstance.storag

Parent

Table 7.17 Spec table for InnoDBCluster.spec.backupSchedules[index].backupProfile.dumpInstance.storage.s3

Name	Type	Description	Required
`bucketName`	string	Name of the bucket where the dump is stored	true
`config`	string	Secret name with S3 configuration and credentials	true
`endpoint`	string	Override endpoint URL	false
`prefix`	string	Path in the bucket where the dump files are stored	false
`profile`	string	Profile being used in configuration files • `Default`:	false

InnoDBCluster.spec.imagePullSecrets[index]

Parent

Table 7.18 Spec table for InnoDBCluster.spec.imagePullSecrets[index]

Name	Type	Description	Required
`name`	string		false

InnoDBCluster.spec.initDB

Parent

Table 7.19 Spec table for InnoDBCluster.spec.initDB

Name	Type	Description	Required
`clone`	object		false
`dump`	object		false

InnoDBCluster.spec.initDB.clone

Parent

Table 7.20 Spec table for InnoDBCluster.spec.initDB.clone

Name	Type	Description	Required
donorUrl	string	URL of the cluster to clone from	true
secretKeyRef	object		true
rootUser	string	User name used for cloning • Default: root	false

InnoDBCluster.spec.initDB.clone.secretKeyRef

Parent

Table 7.21 Spec table for InnoDBCluster.spec.initDB.clone.secretKeyRef

Name	Type	Description	Required
name	string	Secret name with key 'rootPassword' storing the password for the user specified in rootUser	true

InnoDBCluster.spec.initDB.dump

Parent

Table 7.22 Spec table for InnoDBCluster.spec.initDB.dump

Name	Type	Description	Required
storage	object		true
name	string	Name of the dump. Not used by the operator, but a descriptive hint for the cluster administrator	false
options	object	A dictionary of key-value pairs passed directly to MySQL Shell's loadDump()	false
path	string	Path to the dump in the PVC. Use when specifying persistentVolumeClaim. Omit for ociObjectStorage or S3.	false

InnoDBCluster.spec.initDB.dump.storage

Parent

Table 7.23 Spec table for InnoDBCluster.spec.initDB.dump.storage

Name	Type	Description	Required
ociObjectStorage	object		false

Name	Type	Description	Required
persistentVolumeClaim	object	Specification of the PVC to be used. Used 'as is' in the cloning pod.	false
s3	object		false

InnoDBCluster.spec.initDB.dump.storage.ociObjectStorage

Parent

Table 7.24 Spec table for InnoDBCluster.spec.initDB.dump.storage.ociObjectStorage

Name	Type	Description	Required
bucketName	string	Name of the bucket where the dump is stored	true
credentials	string	Secret name with data for accessing the bucket	true
prefix	string	Path in the bucket where the dump files are stored	true

InnoDBCluster.spec.initDB.dump.storage.s3

Parent

Table 7.25 Spec table for InnoDBCluster.spec.initDB.dump.storage.s3

Name	Type	Description	Required
bucketName	string	Name of the bucket where the dump is stored	true
config	string	Secret name with S3 configuration and credentials	true
prefix	string	Path in the bucket where the dump files are stored	true
endpoint	string	Override endpoint URL	false
profile	string	Profile being used in configuration files • Default:	false

InnoDBCluster.spec.keyring

Parent

Description: Keyring specification

Table 7.26 Spec table for InnoDBCluster.spec.keyring

Name	Type	Description	Required
encryptedFile	object	Keyring 'Encrypted File' specification	false

Name	Type	Description	Required
file	object	Keyring 'File' specification	false
oci	object	Keyring 'OCI' specification	false

InnoDBCluster.spec.keyring.encryptedFile

Parent

Description: Keyring 'Encrypted File' specification

Table 7.27 Spec table for InnoDBCluster.spec.keyring.encryptedFile

Name	Type	Description	Required
fileName	string	Full path to the keyring file name inside the storage volume	true
password	string	Name of a secret that contains password for the keyring in the key 'keyring_password'	true
storage	object	Specification of the volume to be mounted where the keyring file resides	true
readOnly	boolean	Whether to open the keyring file in read-only mode • Default: false	false

InnoDBCluster.spec.keyring.file

Parent

Description: Keyring 'File' specification

Table 7.28 Spec table for InnoDBCluster.spec.keyring.file

Name	Type	Description	Required
fileName	string	Full path to the keyring file name inside the storage volume	true
storage	object	Specification of the volume to be mounted where the keyring file resides	true
readOnly	boolean	Whether to open the keyring file in read-only mode • Default: false	false

InnoDBCluster.spec.keyring.oci

Parent

Description: Keyring 'OCI' specification

Table 7.29 Spec table for InnoDBCluster.spec.keyring.oci

Name	Type	Description	Required
keyFingerprint	string	Private key fingerprint	true
keySecret	string	A secret that contains the private key under the field 'privatekey'	true
tenancy	string	Tenancy identifier in the form ocid1.tenancy.oc1...	true
user	string	User identifier in the form of ocid1.user.oc1...	true
caCertificate	string	Secret that contains ca.crt field with CA certificate bundle file that the keyring_oci plugin uses for Oracle Cloud Infrastructure certificate verification	false
compartment	string	Compartment identifier in the form ocid1.compartment.oc1...	false
endpoints	object		false
masterKey	string	Master key identified in the form ocid1.key.oc1...	false
virtualVault	string	Vault identifier in the form ocid1.vault.oc1...	false

InnoDBCluster.spec.keyring.oci.endpoints

Parent

Table 7.30 Spec table for InnoDBCluster.spec.keyring.oci.endpoints

Name	Type	Description	Required
encryption	string	Encryption endpoint URI like {identifier}-crypto.kms.{region}.oraclecloud.com	false
management	string	Management endpoint URI like {identifier}-management.kms.{region}.oraclecloud.com	false
secrets	string	Secrets endpoint URI like secrets.vaults.{region}.oci.oraclecloud.com	false
vaults	string	Vaults endpoint URI like vaults.{region}.oci.oraclecloud.com	false

InnoDBCluster.spec.router

Parent

Description: MySQL Router specification

Table 7.31 Spec table for InnoDBCluster.spec.router

Name	Type	Description	Required
instances	integer	Number of MySQL Router instances to deploy • Default: 1 • Minimum: 0	false
podAnnotations	object		false
podLabels	object		false
podSpec	object		false
tlsSecretName	string	Name of a TLS type Secret containing MySQL Router certificate and private key used for SSL	false
version	string	Override MySQL Router version	false

MySQLBackup

Table 7.32 Spec table for MySQLBackup

Name	Type	Description	Required
apiVersion	string	mysql.oracle.com/v2	true
kind	string	MySQLBackup	true
metadata	object	Refer to the Kubernetes API documentation	true
spec	object		false
status	object		false

MySQLBackup.spec

Parent

Table 7.33 Spec table for MySQLBackup.spec

Name	Type	Description	Required
clusterName	string		true
addTimestampToBackupDirectory	boolean	• Default: true	false
backupProfile	object	backupProfile specification if backupProfileName is not specified	false
backupProfileName	string		false
deleteBackupData	boolean	• Default: false	false

MySQLBackup.spec.backupProfile

Parent

Description: backupProfile specification if backupProfileName is not specified

Table 7.34 Spec table for MySQLBackup.spec.backupProfile

Name	Type	Description	Required
`dumpInstance`	object		false
`podAnnotations`	object		false
`podLabels`	object		false

MySQLBackup.spec.backupProfile.dumpInstance

Parent

Table 7.35 Spec table for MySQLBackup.spec.backupProfile.dumpInstance

Name	Type	Description	Required
`dumpOptions`	object	A dictionary of key-value pairs passed directly to MySQL Shell's DumpInstance()	false
`storage`	object		false

MySQLBackup.spec.backupProfile.dumpInstance.storage

Parent

Table 7.36 Spec table for MySQLBackup.spec.backupProfile.dumpInstance.storage

Name	Type	Description	Required
`ociObjectStorage`	object		false
`persistentVolumeClaim`	object	Specification of the PVC to be used. Used 'as is' in pod executing the backup.	false
`s3`	object		false

MySQLBackup.spec.backupProfile.dumpInstance.storage.ociObjectStorage

Parent

Table 7.37 Spec table for MySQLBackup.spec.backupProfile.dumpInstance.storage.ociObjectStorage

Name	Type	Description	Required
`bucketName`	string	Bucket name where backup is stored	true
`credentials`	string	Secret name with data for accessing the bucket	true
`prefix`	string	Path in bucket where backup is stored	false

MySQLBackup.spec.backupProfile.dumpInstance.storage.s3

Parent

Table 7.38 Spec table for MySQLBackup.spec.backupProfile.dumpInstance.storage.s3

Name	Type	Description	Required
`bucketName`	string	Name of the bucket where the dump is stored	true
`config`	string	Secret name with S3 configuration and credentials	true
`endpoint`	string	Override endpoint URL	false
`prefix`	string	Path in the bucket where the dump files are stored	false
`profile`	string	Profile being used in configuration files • `Default`:	false

MySQLBackup.status

Parent

Table 7.39 Spec table for MySQLBackup.status

Name	Type	Description	Required
`bucket`	string		false
`completionTime`	string		false
`elapsedTime`	string		false
`method`	string		false
`ociTenancy`	string		false
`output`	string		false
`size`	string		false
`source`	string		false
`spaceAvailable`	string		false
`startTime`	string		false
`status`	string		false

Resource Types

- ClusterKopfPeering
- KopfPeering

ClusterKopfPeering

Table 7.40 Spec table for ClusterKopfPeering

Name	Type	Description	Required
apiVersion	string	zalando.org/v1	true
kind	string	ClusterKopfPeering	true
metadata	object	Refer to the Kubernetes API documentation	true

Name	Type	Description	Required
status	object		false

KopfPeering

Table 7.41 Spec table for KopfPeering

Name	Type	Description	Required
apiVersion	string	zalando.org/v1	true
kind	string	KopfPeering	true
metadata	object	Refer to the Kubernetes API documentation	true
status	object		false