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 set up and maintenance including automating upgrades and backups.

For working with Kubernetes, a good understanding of Docker and containers is suggested. Containers are isolated processes groups running on a system and are operating system features on different systems. This documentation assumes a familiarity with Kubernetes.

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 typically uses the long names but these overloaded terms may still lead to confusion; context is important.

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

For legal information, see the Legal Notices.

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

Document generated on: 2022-02-15 (revision: 71961)
# Table of Contents

Preface and Legal Notices .......................................................................................................................... v

1 Installing MySQL Operator for Kubernetes .............................................................................................. 1
   1.1 Installation Using Helm Charts ............................................................................................................. 1
   1.2 Installing the MySQL Operator using Manifest Files ............................................................................. 1
   1.3 Setting Up a MySQL InnoDB Cluster: Simple ......................................................................................... 2

2 MySQL Operator Cookbook ......................................................................................................................... 5
   2.1 Copying Image to a Private Registry using Helm ..................................................................................... 5
   2.2 Copying Image to a Private Registry using Docker .................................................................................. 6
   2.3 Copying Image to a Private Registry using Skopeo .................................................................................. 6
   2.4 Using Helm for Bootstrapping a MySQL InnoDB Cluster from a Dump ................................................. 7

3 MySQL Operator Properties ......................................................................................................................... 9
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 setup 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

Copyright © 2009, 2022, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

**U.S. GOVERNMENT END USERS:** Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.
This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

This documentation is in pre-General Availability status and is intended for demonstration and preliminary use only. It may not be specific to the hardware on which you are using the software. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to this documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

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 Installing MySQL Operator for Kubernetes

Table of Contents

1.1 Installation Using Helm Charts ................................................................. 1
1.2 Installing the MySQL Operator using Manifest Files .................................. 1
1.3 Setting Up a MySQL InnoDB Cluster: Simple ........................................... 2

Two different installation methods are documented here; use either Helm or the kubectl command line tool. This documentation assumes that kubectl is available on the system.

1.1 Installation Using Helm Charts

Helm is a package manager for Kubernetes. It helps install and configure Kubernetes Operators and assign their resources. For Helm specific usage information, see the Helm Quickstart and Installing Helm guides. Alternatively, see Section 1.2, “Installing the MySQL Operator using Manifest Files”.

Add the helm repository:

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

Install MySQL Operator for Kubernetes, and in our example we’re defining the release name as `my-mysql-operator` (it can be anything) and optionally installing it under a new namespace named `mysql-operator` (it can also be anything) rather than the default namespace.

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

This latest MySQL Operator for Kubernetes is downloaded from DockerHub and deployed. The operator deployment is customizable through other options that override built-in defaults. For example, if you have an air-gapped Kubernetes installation and use your own private container registry, there is a way to use it with the operator.

Optionally install an example MySQL InnoDB Cluster. As we did with the operator above, our example is setting and creating a new namespace:

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

1.2 Installing the MySQL Operator using Manifest Files

This document assumes a familiarity with kubectl, and that you have it installed. Alternatively, see Section 1.1, “Installation Using Helm Charts”.

MySQL Operator for Kubernetes can be installed using raw manifest files with kubectl:

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

CRD propagation can take several seconds depending on the size of the Kubernetes cluster, so wait five or so seconds between this and the following command.

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

If the command fails due to missing CRDs, wait for a few seconds and try again. Verify that the operator is running by checking the deployment that’s managing the operator inside the mysql-operator namespace:

```bash
$> kubectl get deployment -n mysql-operator mysql-operator --output wide
```

After MySQL Operator for Kubernetes is ready, the output should look similar to:
1.3 Setting Up a MySQL InnoDB Cluster: Simple

Using Helm

Public Registry

This sets up a simple MySQL InnoDB Cluster using command line parameters to set options:

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

Private Registry

This is a similar example but uses a private registry:

```
export REGISTRY="..." # like 192.168.20.199:5000
export REPOSITORY="..." # like "mysql"
export NAMESPACE="mynamespace"
export DOCKER_SECRET_NAME="priv-reg-secret"
export DOCKER_USER="user"
export DOCKER_USER_PASS="pass"
$> 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 \\
   --set image.pullSecrets.username=$DOCKER_USER \\
   --set image.pullSecrets.password=$DOCKER_USER_PASS \\
   --set image.pullSecrets.email='user@example.com'
```

Using kubectl

To create an InnoDB Cluster, first create a secret containing credentials for a new MySQL root user:

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

Use that newly created user to create the InnoDB Cluster, and this example uses a simple cluster definition:
apiVersion: mysql.oracle.com/v2alpha1
text:
kind: InnoDBCluster
metadata:
  name: mycluster
spec:
  secretName: mypwds
  instances: 3
  router:
    instances: 1

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

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

This sample created an InnoDB Cluster named mycluster with three MySQL server instances and one MySQL Router instance. The process can be observed using:

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

Output looks similar to:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>ONLINE</th>
<th>INSTANCES</th>
<th>ROUTERS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>mycluster</td>
<td>PENDING</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>10s</td>
</tr>
</tbody>
</table>

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:

```
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 Families:       <none>
IP:                10.43.203.248
IPs:               <none>
Port:              mysql  6446/TCP
TargetPort:        6446/TCP
Endpoints:         <none>
Port:              mysqlx  6448/TCP
TargetPort:        6448/TCP
Endpoints:         <none>
Port:              mysql-ro  6447/TCP
TargetPort:        6447/TCP
Endpoints:         <none>
Port:              mysqlx-ro  6449/TCP
TargetPort:        6449/TCP
Endpoints:         <none>
Session Affinity:   None
Events:            <none>
```

Using Kubernetes port forwarding you can create a redirection from your local machine, so that you can use any MySQL Client, like MySQL Shell or MySQL Workbench, to inspect or using the server.

For a read-write connection to the primary using the MySQL protocol:

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

And then in a second terminal using MySQL Shell, and when prompted enter the MySQL root password defined earlier:
$> mysqlsh -h127.0.0.1 -P6446 -uroot -p
Chapter 2 MySQL Operator Cookbook

Table of Contents

2.1 Copying Image to a Private Registry using Helm ............................................. 5
2.2 Copying Image to a Private Registry using Docker ........................................ 6
2.3 Copying Image to a Private Registry using Skopeo ........................................ 6
2.4 Using Helm for Bootstrapping a MySQL InnoDB Cluster from a Dump .................. 7

Tasks related to using MySQL Operator for Kubernetes.

2.1 Copying Image to a 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:

```bash
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; and then execute `helm install` with arguments that looks similar to:

```bash
export REGISTRY="..."   # like 192.168.20.199:5000
export REPOSITORY="..." # like "mysql"
export NAMESPACE="mysql-operator"
export DOCKER_SECRET_NAME="priv-reg-secret"
export DOCKER_USER="user"
export DOCKER_USER_PASS="pass"
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 image.pullSecrets.username="$DOCKER_USER"
   --set image.pullSecrets.password="$DOCKER_USER_PASS"
   --set image.pullSecrets.email='user@example.com'
   --set envs.imagesPullPolicy='IfNotPresent'
   --set envs.imagesDefaultRegistry=$REGISTRY
   --set envs.imagesDefaultRepository=$REPOSITORY
```

To confirm, check the status with commands such as `helm list -n $NAMESPACE` and `kubectl -n $NAMESPACE get pods`. 
**2.2 Copying Image to a Private Registry using Docker**

1. Get the desired MySQL Operator for Kubernetes version, for example the latest is defined in `helm/mysql-operator/Chart.yaml`. For example, `8.0.29-2.0.3`.

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. This host also needs access to the 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.

```bash
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:$OPERATOR_VERSION && 
docker push $REGISTRY/$REPOSITORY/mysql-operator:$OPERATOR_VERSION && 
docker rmi mysql/mysql-operator:$OPERATOR_VERSION $REGISTRY/$REPOSITORY/mysql-operator:$OPERATOR_VERSION"

2.3 Copying Image to a Private Registry using 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 `8.0.29-2.0.3`.

```bash
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 rm quay.io/skopeo/stable
```

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.
2.4 Using Helm for Bootstrapping a MySQL InnoDB Cluster from a Dump

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 bootstrapping 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

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, then execute:

```bash
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:

```bash
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 $NAMESPACE 
  --set credentials.root.user='root' 
  --set credentials.root.password='sakila' 
  --set serverInstances=3 
  --set routerInstances=3 
  --set initDB.dump.name="initdb-dump"
```
--set initDB.dump ociObjectStorage.prefix=${OCI_DUMP_PREFIX} \
--set initDB.dump ociObjectStorage.bucketName=${OCI_BUCKET_NAME} \
--set initDB.dump ociObjectStorage.credentials=${OCI_CREDENTIALS_SECRET_NAME}
### Chapter 3 MySQL Operator Properties

**CRD specification properties for MySQL Operator**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>secretName</td>
<td>Name of a secret containing root/default account password; this is a required field.</td>
<td>string</td>
</tr>
<tr>
<td>sslSecretName</td>
<td></td>
<td>string</td>
</tr>
<tr>
<td>version</td>
<td>MySQL Server version</td>
<td>string</td>
</tr>
<tr>
<td>edition</td>
<td>MySQL Server Edition; either commercial or community.</td>
<td>string</td>
</tr>
<tr>
<td>imageRepository</td>
<td>Repository from where images must be pulled from. Default value is 'mysql'.</td>
<td>string</td>
</tr>
<tr>
<td>imagePullPolicy</td>
<td>Available values are <strong>IfNotPresent</strong>, <strong>Always</strong>, or <strong>Never</strong>. For usage details, see the official Image pull policy documentation.</td>
<td>string</td>
</tr>
<tr>
<td>imagePullSecrets</td>
<td>For usage details, see the official Using imagePullSecrets documentation.</td>
<td>array</td>
</tr>
<tr>
<td>serviceAccountName</td>
<td>The ServiceAccount name used to run the pod. For usage details, see the official Configure Service Accounts for Pods documentation.</td>
<td>string</td>
</tr>
<tr>
<td>baseServerId</td>
<td>Base value for MySQL server_id for instances in the cluster.</td>
<td>integer</td>
</tr>
<tr>
<td>datadirVolumeClaimTemplate</td>
<td>Template for a <strong>PersistentVolumeClaim</strong>, to be used as the <strong>datadir</strong>.</td>
<td>object</td>
</tr>
<tr>
<td>mycnf</td>
<td>Custom configuration additions for MySQL server's <strong>my.cnf</strong>.</td>
<td>string</td>
</tr>
<tr>
<td>instances</td>
<td>Number of MySQL replica instances for the cluster.</td>
<td>integer</td>
</tr>
<tr>
<td>podSpec</td>
<td></td>
<td>object</td>
</tr>
</tbody>
</table>
initDB

Type: object

router

Properties for the Router.

Type: object

Properties:

- **instances**: The number of MySQL router instances to deploy; defaults to 0.
- **version**: Optionally override the MySQL Router version.
- **podSpec**: A podSpec object for the Router.

backupProfiles

Backup profile specifications for the cluster, which can be referenced from backup schedules and one-off backup jobs.

Type: array

Properties:

- **name**
- **dumpInstance**
- **snapshot**

backupSchedules

Schedules for periodically executed backups

Type: array

Properties:

- **name**
- **backupProfileName**
- **backupProfile**
- **deleteBackupData**