

VMware Tanzu SQL with MySQL for Kubernetes v1.1 Documentation

VMware Tanzu SQL with MySQL for Kubernetes 1.1

You can find the most up-to-date technical documentation on the VMware website at:

<https://docs.vmware.com/>

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Copyright © 2022 VMware, Inc. All rights reserved. [Copyright and trademark information.](#)

Contents

VMware Tanzu™ SQL with MySQL for Kubernetes	7
About Tanzu MySQL for Kubernetes	7
About MySQL 8.0	7
MySQL 8.0 Features	7
More about MySQL	7
Product Snapshot	7
License	8
General Limitations	8
Release Notes	9
Supported Platforms	9
Version 1.1.0	9
Features	9
Changes	9
Resolved Issues	10
Known Issues	10
Limitations	10
Version 1.0.0	11
Features	11
Known Issues	11
Limitations	12
Architecture	13
VMware Tanzu™ SQL with MySQL for Kubernetes Architecture	13
Architecture of a Single-Node MySQL Instance	13
Architecture of an HA MySQL Instance	13
Controllers and Custom Resource Definitions (CRDs)	14
MySQL	14
MySQLBackupLocation	15
MySQLBackup	15
MySQLBackupSchedule	16
MySQLRestore	16
Installing the Tanzu SQL for Kubernetes Operator	18
Prerequisites	18

Install Tanzu MySQL Operator via the Tanzu Network Registry	19
Access the Resources	19
Create Namespace and Secret	19
Review values.yaml and Create Overrides If Needed	20
Use Helm CLI to Install the Operator	20
Install Tanzu Operator via a Downloaded Archive File	21
Prerequisites	21
Download and unpack the Tanzu MySQL distribution	21
Load the Images to a Local Docker Registry	22
Push the Images to a Private Container Registry	22
Configure a Kubernetes Secret for Accessing the Private Container Registry	22
Deploy the Tanzu MySQL Operator	23
Edit the Operator Configuration	23
Create the MySQL Operator	24
Upgrading the Tanzu SQL for Kubernetes Operator	26
Use the Helm CLI to Upgrade the Operator	26
Creating and Deleting MySQL Instances	28
Prerequisites	28
Download the Deployment Templates	28
Create a MySQL Instance	28
Delete a MySQL Instance	29
About PVCs	29
About Deleting a MySQL Instance	30
Procedure	30
Updating MySQL Instances	31
Prerequisites	31
Scale storageSize	31
Expected Downtime When Scaling storageSize	32
Scale CPU and Memory Resources	33
Expected Downtime When Scaling CPU and Memory Resources	33
Change Other Configurations	33
Upgrading MySQL Instances	35
Prerequisites	35
List Instances By Version	35
Upgrade an Instance	35

Configuring MySQL Instances for High Availability	37
Overview	37
About Planning for Long-Lived HA MySQL Instances	37
About Explicit Node Selection and Anti-Affinity	37
About Scaling Single Node to HA	37
About Scaling HA to Single Node	37
About Backing Up HA Instances	37
Prerequisites	37
Create an HA MySQL Instance	38
Convert a Single-Node MySQL Instance to an HA MySQL Instance	38
Inspect an HA MySQL Instance	38
Move an HA MySQL Instance to a Single-Node MySQL Instance	39
Delete an HA MySQL Instance	40
Monitoring MySQL Instances in Kubernetes	41
Overview	41
Prerequisites	41
Verifying MySQL Metrics	41
Using Prometheus Operator to Scrape the Tanzu MySQL for Kubernetes Metrics	42
Using TLS for the Metrics Endpoint	43
MySQL Server Exporter Collector Flag Reference	43
Rotating MySQL Credentials	44
Overview	44
Prerequisites	44
Option 1: Delete the Kubernetes Secret	44
Option 2: Patch the Kubernetes Secret with a Custom Password	45
Accessing MySQL Instances	47
Prerequisites	47
(Optional) Verify MySQL Instance Settings	47
Get Root Access to the MySQL Server	47
Access the MySQL Server from an External IP Address	48
Turn Off External Access	49
Access MySQL From Cluster-Hosted Applications	49
Configuring TLS for MySQL Instances	50
Overview	50
Prerequisites	50
Create the TLS Secret Manually	50

Create TLS Secret with cert-manager	51
Configure MySQL for TLS	52
Connect to a Load-Balanced MySQL Instance over TLS	52
Connecting Apps to MySQL Instances	54
Prerequisites	54
Create a Database and Privileged MySQL User for the App	54
Configure Your App with MySQL User and Connectivity Information	54
Binding a TAS Application to a MySQL Instance	58
Prerequisites	58
Binding an Application	58
Troubleshooting	58
Backing Up and Restoring MySQL Instances	60
Overview	60
About Synchronization of Backups with the External Blobstore	60
Back Up Tanzu MySQL for Kubernetes Data	60
Create a MySQLBackupLocation Resource	60
Create a MySQLBackupSchedule Resource	61
Name and Location for Backup Artifacts	62
Perform an On-Demand Backup	62
List Existing MySQLBackup Resources	63
Delete Old Backup Artifacts	63
Restore	64
Restore from a Backup	64
Prerequisites	64
Procedure	64
Restoring a Backup to a Different Namespace or Kubernetes Cluster	65
Troubleshoot Backup and Restore	66
Property Reference for the MySQL Resource	67
Property Reference for Backup and Restore	70
Properties for the MySQLBackupLocation Resource	70
Properties for the Secret	70
Properties for the MySQLBackupSchedule Resource	70
Properties for the MySQLBackup Resource	71
Properties for the MySQLRestore Resource	71

VMware Tanzu™ SQL with MySQL for Kubernetes

This topic provides an overview of VMware Tanzu™ SQL with MySQL for Kubernetes (Tanzu MySQL for Kubernetes).

About Tanzu MySQL for Kubernetes

Tanzu MySQL for Kubernetes is a Kubernetes Operator that automates provisioning, management, and operations of dedicated MySQL instances running on Kubernetes. By default, MySQL instances provisioned by Tanzu MySQL for Kubernetes are configured with secure and functional settings to meet app developer expectations for a general-use relational database.

Kubernetes Operators are software extensions to Kubernetes that provide custom resources for the management of apps, services, and their components.

The Operator provides a consistent way to deploy MySQL instances to Kubernetes and to run them. This includes “day two”, continuous operations. Apps running on Kubernetes can use MySQL instances deployed using the Operator.

Tanzu MySQL for Kubernetes packages a collection of open-source software to help you deploy and manage one or more instances of the MySQL database on Kubernetes. It includes the following components:

- [Percona Server 8.0](#)
- [Percona XtraBackup 8.0](#)

About MySQL 8.0

MySQL is a powerful open-source relational database in use since the mid-90s. Developers have relied on MySQL as a first step to storing, processing, and sharing data. As its user community has grown, MySQL has become a robust system capable of handling a wide variety of use cases and very significant workloads. Unlike other traditional databases that centralize and consolidate data, MySQL lends itself to dedicated deployment, which supports building “cloud native” apps.

MySQL 8.0 Features

The major features of MySQL 8.0 include:

- Multi-Version Concurrency Control (MVCC)
- Tablespaces
- Asynchronous replication
- Nested transactions (savepoints)
- Online/hot backups
- Redo logging for fault tolerance
- Crash-safe DDL
- Unicode
- NoSQL Document Store interface
- Native JSON datatype
- Native Geographic Information System (GIS) support

More about MySQL

If you are new to MySQL and want general information about MySQL, see the [MySQL documentation](#).

Product Snapshot

The following table provides version and version-support information about Tanzu MySQL for Kubernetes.

Element	Details
Version	1.1.0
Release date	August 12, 2021
Compatible Kubernetes versions	1.16 or later

License

This product is distributed under the [VMware EULA](#) license.

General Limitations

Tanzu MySQL for Kubernetes has the following limitation:

- This product is intended to be used with any Kubernetes distribution v1.16 or later. However, given the number of Kubernetes vendors, versions, and configurations, not all of them have been tested with Tanzu MySQL for Kubernetes.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Release Notes

These are the release notes for VMware Tanzu™ SQL with MySQL for Kubernetes.

Supported Platforms

The Tanzu MySQL for Kubernetes releases support the components listed below:

Tanzu MySQL for Kubernetes Version	Percona Server	Percona XtraBackup
1.1.0	8.0.25-15	8.0.25-17
1.0.0	8.0.22-13	8.0.23-16

IMPORTANT: VMware does not support deployments that have been modified by adding layers to the packaged Docker images, or deployments that reference images other than the VMware MySQL Operator. VMware does not support changing the contents of the deployed containers and pods in any way.

Version 1.1.0

Release Date: August 17, 2021

Features

- Release 1.1.0 supports MySQL Server 8.0.25. For details on MySQL Server 8.0.25 see [MySQL Release Notes](#) and [Percona Server Release Notes](#).
- Release 1.1.0 supports Percona XtraBackup 8.0.25. For details on Percona XtraBackup 8.0.25 see [Percona XtraBackup Release Notes](#).
- Tanzu MySQL instances are now created with a version number. Users may specify the `spec.version` field in the instance definition to specify which Tanzu MySQL version to deploy. Users can now list instances by version number. For more details, see [List Instances By Version](#).
- When installing VMware Tanzu™ SQL with MySQL for Kubernetes, users can specify a default version for new Tanzu MySQL instances. For more details, see [Installing the Operator](#).
- Tanzu MySQL for Kubernetes now requires installing Cert Manager before creating or updating new instances. For more details, see [Installing the Operator](#).
- Tanzu MySQL for Kubernetes instances now expose Prometheus compatible metrics endpoints. Metrics are secured behind a self-signed TLS certificate. For more details see [Monitoring MySQL Instances in Kubernetes](#).
- This release supports an additional auto-recovery feature: an HA cluster that loses quorum can now recover its state.
- Customers can now upgrade from a previous release to a new release. For more information on the process, see [Upgrading MySQL Instances](#).
- This release prevents the users from making the following changes:
 - Users are prevented from scaling an HA instance to a single-node instance.
 - This release now displays an error when the user attempts to change the field `storageClassName`.
 - This release now displays an error when the user attempts to reduce `storageSize` on a running MySQL instance.
 - Users are prevented from downgrading MySQL instances to a lower version.

Changes

- `instanceImage` has been deprecated and removed from the Tanzu MySQL for Kubernetes Helm chart. This release introduces two new chart values, `registry` and

`defaultInstanceVersion`. Users should specify `registry: <my.private-registry>` in their `values-override.yaml`. For more details, see [Installing the Tanzu SQL for Kubernetes Operator](#).

- `imagePullSecret` has been renamed to `imagePullSecretName` for the Helm chart and the MySQL resource, and it's now optional. Users who are maintaining a deployment file for the object should rename the field or drop the field entirely. If omitted, the `imagePullSecretName` will default to the Helm chart setting.
- Users can now alter `spec.storageSize` in a MySQL object to scale up the PV (Persistent Volume), if the underlying storage class supports `onlineVolumeExpansion`. See [Scale storageSize](#).
- Before installing the Tanzu MySQL operator, this release now requires cert-manager installation. For details, see [Prerequisites for Installing via the Tanzu Network Registry](#).

Resolved Issues

- This release addresses a vulnerability [CVE-2020-14040](#) in the `mysqld_exporter` process.
- [177801601] - Users are now prevented from scaling down an existing HA cluster to a single node instance. If users set the `spec.highAvailability.enabled` to `false` in a running HA cluster, the operation fails and returns an error.
- [177369907] - When a user creates a backup with `spec.storageArtifactName`, when the backup starts, the `timeStarted` is now correctly updated.
- [177648443] - The Tanzu MySQL for Kubernetes Operator now checks that the DNS name resolution is consistent before repairing an offline cluster.
- [177801601] - Scaling down an HA instance to a single node is prevented by the Tanzu MySQL for Kubernetes Operator.
- [177069136] - During an HA instance initialization, the proxy container starts only after all Tanzu MySQL HA instances are fully initialized.

Known Issues

- **Existing or new 1.0.0 Tanzu MySQL instances require an upgrade:** New Tanzu MySQL 1.0.0 instances, or existing 1.0.0 instances that restart, do not reach a Running state. The 1.1.0 Tanzu MySQL Operator tries to grant permissions to the `performance_schema.keyring_component_status` system table that is introduced in Percona 8.0.25, and does not exist in Percona 8.0.22 and the 1.0.0 instances. As a workaround, upgrade existing 1.0.0 Tanzu MySQL instances to 1.1.0, and avoid creating new 1.0.0 instances.
- **Proxy connection limitation:** The proxy in an HA instance supports up to 100 connection errors from the same source IP. When the limit is reached, the proxy does not process any further connection requests. For a workaround, restart the proxy pod in order to reset the connection error counter.
- **`imagePullSecretName` required for restores:** MySQLRestore requires `imagePullSecretName` to be provided in the `instanceTemplate`.
- **S3 backups cannot exceed 50 GB:** AWS limits the multipart upload to 10,000 parts per upload. The Tanzu MySQL default part size is 5MB, which results in an upper limit of 50 GB for the full backup. For more details, see [Amazon S3 multipart upload limits](#).

Limitations

- Tanzu MySQL metrics are exposed to a metrics collector that is currently installed inside the Kubernetes cluster.
- HA instances have no anti-affinity capability, so the pods of an HA instance may be scheduled to the same Kubernetes node.
- Backups are only supported on S3-compatible blobstores that support the S3 ListObjectsV2 API. Notably, Google Cloud Storage (GCS) does not support this API in its S3-compatibility mode.
- To rotate MySQL system account passwords, you must manually restart the pods. For details, see [Rotating MySQL Credentials](#).
- TLS is required for external connections to the database. There is no supported option to disable this requirement.

- Some common operations require an administrator to run `kubectl exec` to access a pod. Some examples are:
 - Checking for an HA instance that is not tolerant to additional members leaving the replication group.
 - Configuring schemas and users for an application.
- Backups are unencrypted. Enable S3 server-side encryption and ensure that the `MySQLBackupLocation` object is configured with a secure endpoint (`spec.endpoint` should begin with `https://`). For more information about server-side encryption, see the [AWS documentation](#).

Version 1.0.0

Release Date: April 15, 2021

This is the first generally available (GA) release of VMware Tanzu™ SQL with MySQL for Kubernetes. For an overview of this product, see [About Tanzu MySQL for Kubernetes](#).

Features

- **MySQL for Kubernetes Operator:** VMware Tanzu™ SQL with MySQL for Kubernetes implements the Kubernetes Operator pattern to provision and manage on-demand Tanzu MySQL instances. Tanzu MySQL for Kubernetes supports single-node MySQL database instances.

For more information about Tanzu MySQL for Kubernetes features and compatibility, see [Overview](#). For information about Tanzu MySQL for Kubernetes architecture, see [Architecture](#). For more information about the Kubernetes Operator pattern, see the [Kubernetes documentation](#).

- **Installation Using Helm:** Kubernetes admins can use Helm to install the Tanzu MySQL for Kubernetes Operator. This simplifies the installation process while maintaining flexibility in configuration. For information about how to install the Tanzu MySQL for Kubernetes Operator, see [Installing the Operator](#).
- **Backup and Restore:** Tanzu MySQL for Kubernetes automates on-demand and scheduled full physical backups using the [Percona XtraBackup](#) tool. It also automates restoring and managing backups. For more information, see [Backing Up and Restoring MySQL Instances](#).
- **Sane and Secure Server Defaults:** Tanzu MySQL for Kubernetes configures MySQL server settings to optimize for security and performance. Certain server settings, like `max-connections`, are auto-tuned based on the compute resources and persistent storage provisioned for the Tanzu MySQL for Kubernetes instance.

To see all MySQL server settings configured, follow the procedure in [\(Optional\) Verify MySQL Server Settings](#).

- **MySQL credential management:** Tanzu MySQL for Kubernetes uses Kubernetes automation to simplify rotating MySQL user credentials. For more information, see [Rotating MySQL Credentials](#).
- **High Availability:** Support for creating a high-availability cluster configuration with three members. For information, see [Architecture](#) and [Configuring MySQL Instances for High Availability](#).
- **Configure TLS:** TanzuMySQL instances only accept encrypted client connections. Users can now configure a TanzuMySQL instance for TLS by creating a TLS Secret. For more information, see [Configuring TLS for MySQL Instances](#).
- **Allow off-platform connections:** Users can connect to a TanzuMySQL instance from outside the Kubernetes cluster using an external load balancer. For more information, see [Connect to the MySQL Server with an External IP Address](#) in [Accessing MySQL Instances](#).

Known Issues

- **Upgrades from beta releases are not supported:** Upgrades from beta releases to the VMware Tanzu™ SQL with MySQL for Kubernetes v1.0.0 release are not supported. Download and install the latest version.
- **Scaling down an HA instance is not supported:** Creating an HA instance and scaling back to a single pod is not supported and may incur data loss. The VMware Tanzu™ SQL with MySQL

for Kubernetes operator does not prevent you from changing the property, and changing the property will result in unknown behavior. If the pods for an HA instance crash while the cluster and its metadata are still being created, the cluster may not recover automatically.

- **Pods may restart when creating a HA instance:** Proxy pods may spontaneously restart while the cluster is being initialized.
- **S3 backups cannot exceed 50 GB:** AWS limits the multipart upload to 10,000 parts per upload. The Tanzu MySQL default part size is 5MB, which results in an upper limit of 50 GB for the full backup. For more details, see [Amazon S3 multipart upload limits](#).

Limitations

- HA instances have no anti-affinity capability, so the pods of an HA instance may be scheduled to the same Kubernetes node.
- Backups are only supported on S3-compatible blobstores that support the S3 ListObjectsV2 API. Notably, Google Cloud Storage (GCS) does not support this API in its S3-compatibility mode.
- Changing `spec.storageSize` in a MySQL object to scale the PersistentVolume is not supported. See [Scale storageSize](#) for a workaround that expands the PersistentVolume.
- To rotate MySQL system account passwords, you must manually restart pods. For details, see [Rotating MySQL Credentials](#).
- TLS is required for external connections to the database. There is no supported option to disable this requirement.
- Some common operations require an administrator to run `kubect1 exec` to access a pod. Some examples are:
 - Checking for an HA instance that is not tolerant to additional members leaving the replication group.
 - Configuring schemas and users for an application.
- Backups are unencrypted. Enable S3 server-side encryption and ensure that the MySQLBackupLocation object is configured with a secure endpoint (`spec.endpoint` should begin with `https://`). For more on server-side encryption, see the [AWS documentation](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Architecture

This topic provides some architectural information about VMware Tanzu™ SQL with MySQL for Kubernetes.

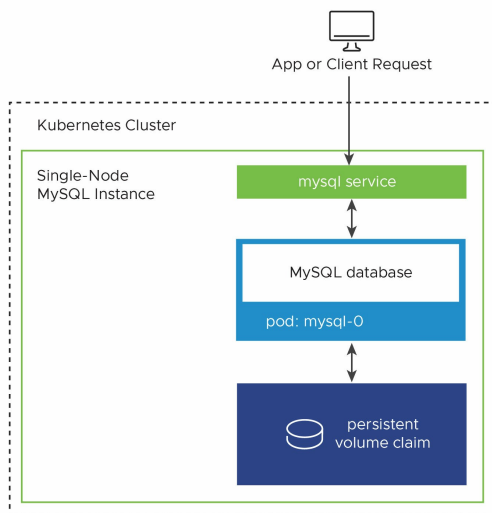
VMware Tanzu™ SQL with MySQL for Kubernetes Architecture

This section illustrates the two topologies of MySQL instances:

- Single-node instances
- High-availability (HA) instances

Architecture of a Single-Node MySQL Instance

The diagram below shows the architecture of a single-node MySQL instance:



[Click here to view a larger version of this diagram](#)

In a single-node MySQL instance, the app or client makes a request to the MySQL service. The service communicates with the single database Pod, named `mysql-0`. The database Pod stores data in a persistent volume claim (PVC).

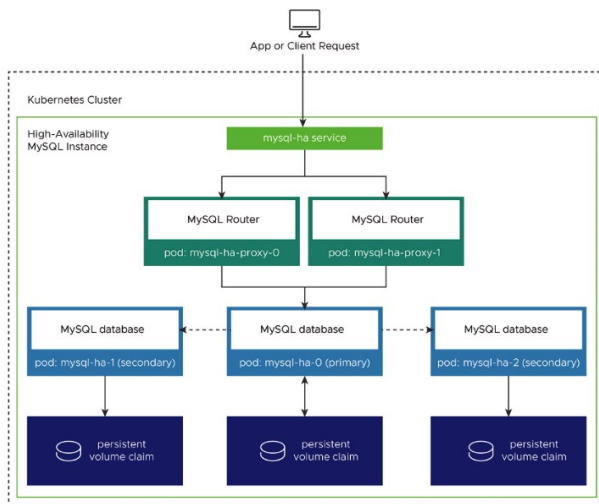
Architecture of an HA MySQL Instance

High availability offers automatic failover, ensuring that app requests operate continuously and without extended downtime.

VMware Tanzu™ SQL with MySQL for Kubernetes uses InnoDB Cluster to provide a highly-available MySQL instance on Kubernetes. For information about InnoDB Cluster features, see the [MySQL documentation](#).

InnoDB Cluster uses Group Replication and other MySQL technologies. For information about Group Replication, see the [MySQL documentation](#).

The diagram below shows the architecture of an HA MySQL instance:



[Click here to view a larger version of this diagram](#)

In a high-availability (HA) MySQL instance, the topology consists of five Pods:

- One primary MySQL database Pod
- Two secondary MySQL database failover Pods
- Two proxy Pods

The primary database is a read-write database. All external connections, such as from apps, are directed by the proxy to the primary read-write database. The two secondary databases are read-only and are updated as replicas of the primary.

Group Replication ensures that data is synchronously replicated from the primary to the secondary database Pods. If the primary Pod becomes unresponsive, app requests are redirected to a secondary Pod, which gets promoted to primary. All app requests continue to that promoted primary, and a new secondary MySQL Pod is created to replace the failed Pod.

If a proxy Pod fails, Kubernetes directs traffic to the remaining proxy Pod while re-creating the failed Pod in the background.

Controllers and Custom Resource Definitions (CRDs)

VMware Tanzu™ SQL with MySQL for Kubernetes provides extensions to the Kubernetes API through custom resources. The product provides five separate custom resources:

- **MySQL**: A managed MySQL instance
- **MySQLBackupLocation**: Defines an S3 storage bucket for backups
- **MySQLBackup**: Requests for a backup
- **MySQLBackupSchedule**: Defines a backup schedule
- **MySQLRestore**: Requests a data restore

MySQL

Applying this resource causes the Kubernetes Operator to create a StatefulSet with a single Pod and three containers. One container runs the MySQL database software, one runs the components to support backups, and the third runs the `mysqld_exporter` (for monitoring). The MySQL Pod mounts a persistent volume claim (PVC) which holds the MySQL data.

When a MySQL instance is created, the MySQL Operator automatically creates a Kubernetes service with a service type of `ClusterIP` which routes connections to the MySQL Pod.

```

---
apiVersion: with.sql.tanzu.vmware.com/v1
kind: MySQL
metadata:
  name: mysql-sample
spec:
  resources:
    mysql:

```

```
requests:
  memory: 1Gi
storageSize: 1Gi
imagePullSecretName: tanzu-mysql-image-registry
```

After applying this resource, the Operator creates the low-level Kubernetes resources to run this MySQL instance.

You can see the status of the instance by running:

```
kubectl get mysqls.with.sql.tanzu.vmware.com
```

Initially, this instance is in a pending state:

```
$ kubectl get mysqls.with.sql.tanzu.vmware.com
NAME          READY   STATUS    VERSION   AGE
mysql-sample   true    Pending   1.1.0     2s
```

Later, the instance transitions to a running state:

```
$ kubectl get mysqls.with.sql.tanzu.vmware.com
NAME          READY   STATUS    AGE
mysql-sample   true    Running   27s
```

MySQLBackupLocation

This resource describes metadata and credentials for storing backups in an S3-compatible object store.

MySQLBackup resources reference this backup location to know where to save backup artifacts. The Operator creates or maintains a MySQLBackup resource for every artifact found in the backup location. Consequently, creating a MySQLBackupLocation that contains existing artifacts generates new, corresponding MySQLBackup resources.

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backuplocation-sample-creds
stringData:
  # S3 Credentials
  accessKeyId: "my-s3-access-key-id"
  secretAccessKey: "my-s3-secret-access-key"
---
apiVersion: with.sql.tanzu.vmware.com/v1
kind: MySQLBackupLocation
metadata:
  name: backuplocation-sample
spec:
  storage:
    # For S3 or Minio:
    s3:
      bucket: "s3bucket"
      bucketPath: "optional/prefix/for/backup/objects/"
      region: "us-east-1"
      endpoint: "" # optional, default to AWS
      forcePathStyle: false
      secret:
        name: backuplocation-sample-creds
```

Credentials for S3 are stored in a Kubernetes secret with the keys `accessKeyId`, `secretAccessKey`. MySQLBackupLocation resources do not currently have any associated status, but you can list them using `kubectl`. For example:

```
$ kubectl get mysqlbackuplocations.with.sql.tanzu.vmware.com
NAME                AGE
backuplocation-sample 2m38s
```

MySQLBackup

This resource triggers the instance to run the Percona XtraBackup utility and upload a backup artifact to the blobstore. Blobstore artifacts are represented as MySQLBackup resources for subsequent restore. The resource requests for a backup to be taken as soon as possible. For a resource that requests a scheduled backup, see [MySQLBackupSchedule](#) below. For example:

```
---
apiVersion: with.sql.tanzu.vmware.com/v1
kind: MySQLBackup
metadata:
  name: backup-sample
spec:
  location:
    name: backuplocation-sample
  instance:
    name: mysql-sample
```

You can see the status of the backup using kubectl. For example:

```
$ kubectl get mysqlbackups.with.sql.tanzu.vmware.com
NAME          STATUS    SOURCE INSTANCE    TIME STARTED          TIME COMPLETED
backup-sample  Succeeded mysql-sample  2020-12-07T15:52:12Z  2020-12-07T15:52:16Z
```

MySQLBackupSchedule

This resource defines a schedule to automatically create MySQLBackup resources. For example:

```
---
apiVersion: with.sql.tanzu.vmware.com/v1
kind: MySQLBackupSchedule
metadata:
  name: mysqlbackupschedule-sample
spec:
  backupTemplate:
    spec:
      location:
        name: backuplocation-sample
      instance:
        name: mysql-sample
      schedule: "@daily"
```

MySQLBackupSchedule resources do not currently have any associated status, but you can list them using kubectl. For example:

```
$ kubectl get mysqlbackupschedules.with.sql.tanzu.vmware.com
NAME          AGE
mysqlbackupschedule-sample  8s
```

MySQLRestore

This resource instructs the Operator to create a new PVC and a new MySQL instance from an existing MySQLBackup resource. The backup artifact for the MySQLBackup resource is loaded into the new PVC before the MySQL database is started. For example:

```
---
apiVersion: with.sql.tanzu.vmware.com/v1
kind: MySQLRestore
metadata:
  name: restore-sample
spec:
  backup:
    name: backup-sample
  instanceTemplate:
    metadata:
      name: mysql-sample
    spec:
      storageSize: 1Gi
      imagePullSecretName: tanzu-mysql-image-registry
```

You can see the status of the restore using kubectl. For example:

```
$ kubectl get mysqlrestores.with.sql.tanzu.vmware.com
NAME          STATUS    SOURCE BACKUP    TARGET INSTANCE    TIME STARTED          TI
ME COMPLETED
restore-sample  Running   backup-sample    mysql-sample        2020-12-07T15:56:26Z
```

The restore retrieves and unpacks the backup artifact and creates the MySQL instance. The restore time is proportional to the size of the backup artifact. For example:

```
$ kubectl get mysqlrestores.with.sql.tanzu.vmware.com
NAME          STATUS    SOURCE BACKUP    TARGET INSTANCE    TIME STARTED
```



```
TIME COMPLETED
restore-sample    Succeeded    backup-sample    mysql-sample     2020-12-07T15:56:26Z
2020-12-07T15:56:41Z
```

At the end of the restore process, you have a MySQL instance with the **READY** state **true** and with **STATUS running**. For example:

```
$ kubectl get mysqls.with.sql.tanzu.vmware.com
NAME          READY  STATUS   AGE
mysql-sample  true   Running  30s
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Installing the Tanzu SQL for Kubernetes Operator

Page last updated:

This topic describes how Kubernetes admins install, configure, and deploy the VMware Tanzu™ SQL with MySQL for Kubernetes using two different methods.

- Use [Installing Tanzu MySQL Operator via Tanzu Network Registry](#) for a faster installation process, and if your server hosts have access to the internet.
- Use [Installing Tanzu MySQL Operator via Downloadable Archive File](#) if your server hosts do not have access to the internet, or if you want to install from a private registry.

Prerequisites

Before you deploy the Tanzu MySQL Operator, you need:

- Access to [Tanzu Network](#) and [Tanzu Network Registry](#). You can use the same credentials for both sites.
- [Docker](#) running and configured on your local computer, to access the Kubernetes cluster and Docker registry.
- A running Kubernetes cluster.
- The `kubectl` command-line tool, configured and authenticated to communicate with your Kubernetes cluster.
- The Helm v3 command-line tool installed. For more information, see [Installing Helm](#) from the Helm documentation.
- `cluster-admin` ClusterRole access to the Kubernetes cluster. For more information, see the [Kubernetes documentation](#).
- [Cert Manager](#) installed on the Kubernetes cluster.

Install cert-manager by running these commands from your local client:

```
$ kubectl create namespace cert-manager
$ helm repo add jetstack https://charts.jetstack.io
$ helm repo update
$ helm install cert-manager jetstack/cert-manager --namespace cert-manager --version <1.latest> --set installCRDs=true
```

where:

- `--namespace cert-manager` is the namespace used for cert manager in the Kubernetes cluster
- `--version <1.latest>` is the latest cert-manager version available (minimum above 1.0.2)
- `--set installCRDs=true` ensures cert manager installs all types necessary to create certificates

To verify the installation run:

```
$ kubectl get all --namespace=cert-manager
```

The output should be similar to:

NAME	READY	STATUS	RESTARTS	AGE
pod/cert-manager-57b65b7fc-x8vjt	1/1	Running	5	4d19h
pod/cert-manager-cainjector-5f988f74c6-tgk25	1/1	Running	15	4d19h
pod/cert-manager-webhook-7cf554f879-b5ss9	1/1	Running	4	4d19h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
service/cert-manager	ClusterIP	10.106.253.7	<none>	9402/T
CP 4d19h				
service/cert-manager-webhook	ClusterIP	10.108.17.113	<none>	443/TC
P 4d19h				
NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/cert-manager	1/1	1	1	4d19
h				
deployment.apps/cert-manager-cainjector	1/1	1	1	4d19
h				
deployment.apps/cert-manager-webhook	1/1	1	1	4d19
h				
NAME	DESIRED	CURRENT	READY	
AGE				
replicaset.apps/cert-manager-57b65b7fc	1	1	1	
4d19h				
replicaset.apps/cert-manager-cainjector-5f988f74c6	1	1	1	
4d19h				
replicaset.apps/cert-manager-webhook-7cf554f879	1	1	1	
4d19h				

For more advanced security scenarios, see [Configuring TLS for MySQL Instances](#).

Install Tanzu MySQL Operator via the Tanzu Network Registry

This section describes how to access the resources, and deploy the Tanzu MySQL Operator using the Tanzu Network Registry. To install the Tanzu MySQL Operator, you must use `kubectl` to create a namespace and secret. Then, use the Helm CLI to install the Operator.

Access the Resources

To install the Tanzu MySQL Operator, you must download the Helm chart and images.

To download the resources:

1. Set the environment variable to enable OCI support in the Helm v3 client by running:

```
export HELM_EXPERIMENTAL_OCI=1
```

If you skip this step, the following error message might appear:

```
Error: this feature has been marked as experimental and is not enabled by default.
```

2. Use Helm to log in to the Tanzu Network Registry by running:

```
helm registry login registry.pivotal.io
```

Follow the prompts to enter the email address and password for your VMware Tanzu Network account.

3. Download the `tanzu-mysql-operator-chart` using Helm:

```
$ helm chart pull registry.pivotal.io/tanzu-mysql-for-kubernetes/tanzu-mysql-operator-chart:1.1.0
1.1.0: Pulling from registry.pivotal.io/tanzu-mysql-for-kubernetes/tanzu-mysql-operator-chart
ref:      <span style="color: #77bf00;">registry.pivotal.io/tanzu-mysql-for-kubernetes/tanzu-mysql-operator-chart:1.1.0</span>
digest:   2b6e1d010ab1737dcc5a426b223a06db1c616107d2ceaf368db6fda48d96a61a
size:     4.2 KiB
name:     tanzu-sql-with-mysql-operator
version:  1.1.0
Status: Downloaded newer chart for registry.pivotal.io/tanzu-mysql-for-kubernetes/tanzu-mysql-operator-chart:1.1.0
```

Create Namespace and Secret

To create the namespace and secret:

1. Create the namespace by running:

```
kubectl create namespace tanzu-mysql-for-kubernetes-system
```

2. Set your kubectl context to the newly-created namespace by running:

```
kubectl config set-context --current --namespace tanzu-mysql-for-kubernetes-system
```

3. Create a Kubernetes secret for accessing registry containing the Tanzu MySQL images by running:

```
kubectl create secret docker-registry tanzu-image-registry --docker-server=https://registry.pivotal.io/ \
--docker-username=DOCKER-USERNAME --docker-password=DOCKER-PASSWORD
```

4. Download the Helm chart to your current working directory on your local machine by running:

```
helm chart export REGISTRY-URL
```

Where `REGISTRY-URL` is the reference to the Tanzu MySQL Helm chart.

The value of `REGISTRY-URL` has the following pattern:

```
registry.pivotal.io/tanzu-mysql-for-kubernetes/tanzu-mysql-operator-chart:VERSION-NUMBER-TAG
```

Where `VERSION-NUMBER-TAG` is the version of the Helm chart.

This downloads a directory named `tanzu-sql-with-mysql-operator/` into your current working directory that contains:

- The Tanzu MySQL Helm chart
- Custom Resource Definitions (CRDs)
- Role-Based Access Control (RBAC) definitions required to install the operator with Helm

For example:

```
$ helm chart export registry.pivotal.io/tanzu-mysql-for-kubernetes/tanzu-mysql-operator-chart:1.1.0

ref:      registry.pivotal.io/tanzu-mysql-for-kubernetes/tanzu-mysql-operator-chart:1.1.0
digest:   2b6e1d010ab1737dcc5a426b223a06db1c616107d2ceaf368db6fda48d96a61a
size:     4.2 KiB
name:     tanzu-sql-with-mysql-operator
version:  1.1.0
Exported chart to tanzu-sql-with-mysql-operator/
```

Review values.yaml and Create Overrides If Needed

In most situations, the default values supplied in the `values.yaml` file do not need to be changed.

However, if any of the following are true, follow the steps described in [Edit the Operator Configuration](#) to override the defaults:

- You deployed the Tanzu SQL for Kubernetes images from a registry other than `registry.pivotal.io`.
- You did not use the default `tanzu-image-registry` for the secret name in Step 4 above.
- You want to allocate different CPU or memory resources for your Operator.
- You want to specify an alternate default version for new MySQL instances.

Use Helm CLI to Install the Operator

To install the Operator using the Helm CLI:

1. Verify that you are in the same working directory as where you downloaded the Helm chart in Step 4 of [Create Namespace and Secret](#) above.
2. Install the Tanzu MySQL Operator.

If you created a custom `values-override.yaml` in [Review values.yaml and Create Overrides](#) [If Needed](#) above, then run the following Helm command, including the `--values` flag with the location of your override file:

```
helm install --values=/your/values-override.yaml tanzu-sql-with-mysql-operator
./tanzu-sql-with-mysql-operator/
```

Otherwise, omit the `--values` flag:

```
helm install tanzu-sql-with-mysql-operator ./tanzu-sql-with-mysql-operator/
```

3. See that the Operator has installed successfully by running:

```
kubectl get all
```

The Tanzu MySQL Operator has finished installing when you see the value of the `STATUS` column for the Tanzu MySQL Operator Pod is `Running`. See example output:

```
$ kubectl get all
NAME                                     READY   STATUS
RESTARTS   AGE
pod/mysql-for-kubernetes-controller-manager-84d76dfb77-lq5mb  1/1     Running
0          21s

NAME                                     READY   UP-TO-DATE
AVAILABLE   AGE
deployment.apps/mysql-for-kubernetes-controller-manager  1/1     1
1          23s

NAME                                     DESIRED
CURRENT   READY   AGE
replicaset.apps/mysql-for-kubernetes-controller-manager-84d76dfb77  1
1          1      23s
```

Note: You can only have one Tanzu MySQL operator installed in a Kubernetes cluster.

Install Tanzu Operator via a Downloaded Archive File

Choose this method if:

- The installation destination (for example an air-gapped network) cannot access the [VMware Tanzu Network](#).
- Or you wish to load the Operator and instance images to a private Docker registry.

This section covers:

- Downloading and unpacking the distribution from [VMware Tanzu Network](#)
- Loading the images to a local Docker registry
- Pushing the images from the local Docker registry to a private container registry
- Configuring a Kubernetes secret for accessing the private container registry
- Deploying the MySQL Operator

Prerequisites

Before you deploy the Tanzu MySQL Operator, review the list of [Prerequisites](#) that apply to both installation methods.

Download and unpack the Tanzu MySQL distribution

1. Download the Tanzu MySQL distribution from [VMware Tanzu Network](#). The download filename has the format: `tanzu-mysql-for-kubernetes-<version>.tgz`
2. Unpack the downloaded software:

```
$ cd ~/Downloads
$ tar xzf tanzu-mysql-for-kubernetes-.tgz
```

This command unpacks the distribution into a new directory named `tanzu-mysql-for-kubernetes-<version>`, for example `tanzu-mysql-for-kubernetes-1.1.0`.

3. Change to the new `tanzu-mysql-for-kubernetes-<version>` directory.

```
$ cd ./tanzu-mysql-for-kubernetes-1.1.0
```

Load the Images to a Local Docker Registry

1. Load the MySQL instance image to the Docker registry.

```
$ docker load -i ./images/tanzu-mysql-instance
```

2. Load the MySQL Operator image to the Docker registry.

```
$ docker load -i ./images/tanzu-mysql-operator
```

3. Verify that the two Docker images are now available.

```
$ docker images
REPOSITORY                                TAG
IMAGE ID      CREATED      SIZE
registry.pivotal.io/tanzu-mysql-for-kubernetes/tanzu-mysql-instance  1.1.0
5f0ad7bad51b   5 weeks ago  792MB
registry.pivotal.io/tanzu-mysql-for-kubernetes/tanzu-mysql-operator  1.1.0
eb1458d2e1bd   5 weeks ago  76MB
```

Push the Images to a Private Container Registry

Push the VMware Tanzu™ SQL with MySQL for Kubernetes Docker images to the container registry of your choice. Set each image's project and image repo name, tag the images, and then push them using the Docker command `docker push`.

This example tags and pushes the images to the Google Cloud Registry, using the default (core) project name for the example Google Cloud account.

```
$ gcloud auth configure-docker

$ PROJECT=$(gcloud config list core/project --format='value(core.project)')
$ REGISTRY="gcr.io/${PROJECT}"

$ INSTANCE_IMAGE_NAME="${REGISTRY}/tanzu-mysql-instance:$(cat ./images/tanzu-mysql-instance-tag)"
$ docker tag $(cat ./images/tanzu-mysql-instance-id) ${INSTANCE_IMAGE_NAME}
$ docker push ${INSTANCE_IMAGE_NAME}

$ OPERATOR_IMAGE_NAME="${REGISTRY}/tanzu-mysql-operator:$(cat ./images/tanzu-mysql-operator-tag)"
$ docker tag $(cat ./images/tanzu-mysql-operator-id) ${OPERATOR_IMAGE_NAME}
$ docker push ${OPERATOR_IMAGE_NAME}
```

Configure a Kubernetes Secret for Accessing the Private Container Registry

Create a `docker-registry` type secret to allow the Kubernetes cluster to authenticate with the private container registry so it can pull images. These examples create a secret named `tanzu-image-registry` using Harbor, Google Cloud Registry (GCR), and Amazon Elastic Container Registry (ECR).

Important: The commands below create the secret in the current configured namespace for the `kubectl` context. Only pods created in the same namespace can reference the secret. To create a secret in a different namespace, use the `--namespace` flag. You should create an identical secret in the Operator namespace and in each namespace where you will create Tanzu MySQL instances.

Harbor

```
$ kubectl create secret docker-registry tanzu-image-registry \
  --docker-server=${HARBOR_URL} \
  --docker-username=${HARBOR_USER} \
  --docker-password="${HARBOR_PASSWORD}"
```

GCR

```
$ kubectl create secret docker-registry tanzu-image-registry \
  --docker-server=https://gcr.io \
  --docker-username=_json_key \
  --docker-password="$(cat ~/.key.json)"
```

For information about how to obtain the `key.json` service account file, see [Creating service account](#)

[credentials](#) in the GKE documentation.

ECR

```
$ TOKEN=$(aws ecr --region=$REGION get-authorization-token --output text --query author
izationData[].authorizationToken | base64 -d | cut -d: -f2)
$ kubectl create secret docker-registry tanzu-image-registry \
  --docker-server=https://${ACCOUNT}.dkr.ecr.${REGION}.amazonaws.com \
  --docker-username=AWS \
  --docker-password="${TOKEN}"
```

Next follow [Deploying a MySQL Operator](#). The MySQL Operator will use this secret to allow the Kubernetes cluster to authenticate with the container registry to pull images.

Deploy the Tanzu MySQL Operator

The MySQL Operator is the controller for MySQL instances resources. You install the MySQL Operator using the Helm package manager.

Edit the Operator Configuration

1. Go to the directory where you unpacked the Tanzu MySQL distribution.

```
$ cd tanzu-mysql-for-kubernetes-*
```

The file `charts/tanzu-sql-with-mysql-operator/values.yaml` in the VMware Tanzu™ SQL with MySQL for Kubernetes directory specifies the location of the MySQL Operator and instance images. By default it contains the following values:

```
---
imagePullSecretName: tanzu-image-registry
operatorImage: registry.pivotal.io/tanzu-mysql-for-kubernetes/tanzu-mysql-opera
tor:1.1.0-rc.8
registry: "registry.pivotal.io/tanzu-mysql-for-kubernetes/"
defaultInstanceVersion: 1.1.0-rc.8
resources: {}
```

2. Create a copy of `values.yaml` and name the new file `operator-values-overrides.yaml`. Save this file in the same directory as the `values.yaml` file. In this file, you can specify the custom container registry and secret. For manual changes, you may also set individual parameters using the `--set` flag on the command line.

See [Helm Values Files](#) in the Helm documentation for more information.

If you are using a single node Minikube environment, it is not necessary to override the `operator/values.yaml` file because Minikube pulls the images from its local Docker registry.

Determine which values in the `values.yaml` file need to be changed for your environment. Use the table below as a guide.

Key	Value Type	Description
<code>imagePullSecret</code>	String	Name of image secret. This value must match the name of the Kubernetes secret you create in Create Namespace and Secret above.
<code>operatorImage</code>	URI	Reference to the Tanzu MySQL Operator image. If you uploaded the Operator image to a private registry, you must change this reference to pull the Operator image from your registry.
<code>registry</code>	URI	The registry from which to download Tanzu MySQL images.
<code>defaultInstanceVersion</code>	String	The default version of Tanzu MySQL to use when creating new instances.
<code>resources</code>	List	Limits and requests for CPU and memory for the Tanzu MySQL Operator. You can change these values to scale your resources.

3. Edit the values that you want to change.
4. Delete the sections of the file that you do not change.
5. Save the `operator-values-overrides.yaml` file in a location of your choice or the same

directory as the `values.yaml` file.

Create the MySQL Operator

1. Use Helm to install the MySQL Operator in your Kubernetes cluster.

```
$ helm install --wait my-mysql-operator Operator/
```

where:

- `--wait` flag waits for the Operator deployment to complete before any image installation starts
- `my-mysql-operator` is the custom name you provide for your MySQL Operator
- `operator/` is the location of the MySQL Operator helm chart

or

```
$ helm install --wait my-mysql-operator -f charts/tanzu-sql-with-mysql-operator/operator-values-overrides.yaml charts/tanzu-sql-with-mysql-operator/
```

Replace `charts/tanzu-sql-with-mysql-operator/operator-values-overrides.yaml` with your custom location.

To create the Operator in a namespace different than the default, use:

```
$ helm install --wait my-mysql-operator charts/tanzu-sql-with-mysql-operator/ \
  --values=operator-values-overrides.yaml \
  --namespace=mysql-for-kubernetes-system \
  --create-namespace
```

The command displays a message similar to:

```
NAME: my-mysql-operator
LAST DEPLOYED: Wed Jun 16 13:28:05 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Note: The secret namespace in step [Create a Kubernetes Access Secret](#) must match the Operator namespace.

2. Use `watch kubectl get all` to monitor the progress of the deployment. The deployment is complete when the MySQL Operator pod status changes to `Running`.

```
$ watch kubectl get all
```

You may also check the logs to confirm the Operator is running properly:

```
$ kubectl logs -l app=tanzu-mysql-operator
```

Use the label `app=tanzu-mysql-operator` to search across resources created by the MySQL Operator Helm chart:

```
$ kubectl get all -l app=tanzu-mysql-operator -n mysql-for-kubernetes-system
```

where `-n mysql-for-kubernetes-system` defines the namespace. The output would be similar to:

NAME	READY	STATUS	RES
TARTS			
AGE			
pod/tanzu-mysql-operator-69765b8b74-rtms7	1/1	Running	
0			
3d19h			
NAME	READY	UP-TO-DATE	AVA
ILABLE			
AGE			
deployment.apps/tanzu-mysql-operator	1/1	1	
1			
3d19h			
NAME	DESIRED	CURRENT	REA
DY			
AGE			
replicaset.apps/tanzu-mysql-operator-69765b8b74	1	1	
1			
3d19h			

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Upgrading the Tanzu SQL for Kubernetes Operator

This topic describes how operators upgrade the VMware Tanzu™ SQL with MySQL for Kubernetes Operator.

Use the Helm CLI to Upgrade the Operator

To upgrade Tanzu MySQL for Kubernetes, you must download resources and upgrade the Tanzu MySQL for Kubernetes Operator using Helm.

To upgrade the Operator using the Helm CLI:

1. Set the environment variable to enable OCI support in the Helm v3 client by running:

```
export HELM_EXPERIMENTAL_OCI=1
```

If you skip this step, the following error message might appear:

```
Error: this feature has been marked as experimental and is not enabled by default.
```

2. Use Helm to log in to the Tanzu Network Registry by running:

```
helm registry login registry.pivotal.io
```

Follow the prompts to enter the email address and password for your VMware Tanzu Network account.

3. Verify that you have a running Tanzu MySQL for Kubernetes Operator by running:

```
helm -n tanzu-mysql-for-kubernetes-system ls
```

For example:

```
$ helm -n tanzu-mysql-for-kubernetes-system ls
NAME                                NAMESPACE                                REVISION
UPDATED                             STATUS                                CHART
tanzu-mysql-operator               tanzu-mysql-for-kubernetes-system        1
2021-04-01 12:15:07.16966 -0500 CDT  deployed                                tanzu-sql-with-mysql-o
perator-1.0.0                      1.0.0
```

4. Download the Helm chart to your current working directory on your local machine by running these commands:

```
helm chart pull REGISTRY-URL
```

```
helm chart export REGISTRY-URL
```

Where **REGISTRY-URL** is the reference to the Tanzu MySQL for Kubernetes Helm chart. The value of **REGISTRY-URL** has the following pattern:

```
registry.pivotal.io/tanzu-mysql-for-kubernetes/tanzu-mysql-operator-chart:VERSION-NUMBER-TAG
```

Where **VERSION-NUMBER-TAG** is the version of the Helm chart.

This downloads a directory named **tanzu-sql-with-mysql-operator/** into your current working directory

5. Go to where the new release has been downloaded and apply the new CRDs by running:

```
kubect1 apply -f ./tanzu-sql-with-mysql-operator/crds/
```

Note: You can ignore the warnings in the output.

6. Upgrade the Operator by running:

```
helm upgrade tanzu-mysql-operator ./tanzu-sql-with-mysql-operator
```

When you see **deployed** in the **STATUS** column, the Tanzu MySQL for Kubernetes Helm chart has upgraded:

```
$ helm -n tanzu-mysql-for-kubernetes-system history tanzu-sql-with-mysql-operator
REVISION      UPDATED              STATUS      CHART
1             Thu Jan 14 17:47:53 2021  superseded  tanzu-sql-with-mysql-operator-1.0.0
               1.0.0              Install complete
2             Thu Jan 14 18:09:05 2021  deployed    tanzu-sql-with-mysql-operator-1.0.1
               1.0.1              Upgrade complete
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Creating and Deleting MySQL Instances

This topic describes how to create and delete MySQL instances.

Prerequisites

Before you can create or delete MySQL instances, you must have:

- The Kubernetes Command Line Interface (kubectl) installed. For more information, see the [Kubernetes documentation](#).
- Full admin access to all Kubernetes resources in your developer namespace.
For information about Roles and RoleBindings that your Kubernetes cluster admin needs to create, see the [Kubernetes documentation](#).
- The URL and credentials to access the registry that stores the Tanzu MySQL for Kubernetes images. This can be the VMware Tanzu Network registry or the private registry configured for your environment. If you do not have access to the registry credentials, contact your Kubernetes admin to have these set up for you in your namespace.
- The VMware Tanzu™ SQL with MySQL for Kubernetes deployment templates. See [Download the Deployment Templates](#).

Download the Deployment Templates

In order to create VMware Tanzu™ SQL with MySQL for Kubernetes instances, you must first download the deployment templates from Tanzu Network. You use these templates to create configuration files that specify the Tanzu MySQL for Kubernetes resources that you wish to create.

Note: You only need to download the templates once (before you first create a MySQL instance).

To access the templates:

1. Log in to [VMware Tanzu Network](#).
2. Visit the **VMware Tanzu™ SQL with MySQL for Kubernetes** product page.
3. Click **VMware Tanzu SQL with MySQL for Kubernetes v1.1.0**. This will download a .tgz archive file to your local machine.
4. Expand the downloaded .tgz file. Open the directory `tanzu-mysql-for-kubernetes-1.1.0`. The templates are located in the `samples` subdirectory.

The templates include the following:

- `backup.yaml` (for the MySQLBackup resource)
- `backuplocation.yaml` (for the MySQLBackupLocation resource)
- `backupschedule.yaml` (for the MySQLBackupSchedule resource)
- `mysql.yaml` (for the MySQL resource; this template is used to create MySQL instances)
- `restore.yaml` (for the MySQLRestore resource)
- `tls-secret.yaml` (for a [TLS Secret](#))

You do not need to download the **Artifact References** from the Tanzu MySQL for Kubernetes product page.

Create a MySQL Instance

To create a MySQL instance:

1. Target the namespace where you want to create the MySQL instance:

```
kubectl config set-context --current --namespace=DEVELOPMENT-NAMESPACE
```

Where `DEVELOPMENT-NAMESPACE` is the namespace in which you want to create the instance.

For example:

```
$ kubectl config set-context --current --namespace=my-namespace
```

- From your namespace, Kubernetes must be able to access the registry that stores the Tanzu MySQL for Kubernetes images. To allow this, create an imagePullSecret by running:

```
kubectl --namespace=DEVELOPMENT-NAMESPACE create secret docker-registry tanzu-mysql-image-registry --docker-server=REGISTRY-SERVER-URL --docker-username=DOCKER-USERNAME --docker-password=DOCKER-PASSWORD
```

Where:

- `DEVELOPMENT-NAMESPACE` is the namespace in which you want to create the instance
- `REGISTRY-SERVER-URL` is the VMware Tanzu Network registry or the private registry configured for your environment
- `DOCKER-USERNAME` and `DOCKER-PASSWORD` are the credentials used to pull images from the registry.

For example:

```
$ kubectl create secret --namespace=my-namespace \
docker-registry tanzu-mysql-image-registry \
--docker-server=https://registry.pivotal.io/ \
--docker-username=sample-username \
--docker-password=sample-password

secret/tanzu-mysql-image-registry created
```

- Find the `mysql.yaml` deployment template that you downloaded in the TGZ file from VMware Tanzu Network. For how to download deployment templates, see [Download the Deployment Templates](#).
- Create a copy of the `mysql.yaml` file and give it a unique name.

For example:

```
$ cp ~/Downloads/tanzu-mysql-deployment-templates-1.0.0/samples/mysql.yaml testdb.yaml
```

- Edit the file. For information about the properties that you can set for the MySQL resource, see [Property Reference for the MySQL Resource](#).
- Deploy a MySQL instance to Kubernetes by running:

```
kubectl -n DEVELOPMENT-NAMESPACE apply -f FILENAME
```

Where `FILENAME` is the name of the configuration file you created in Step 4 above.

For example:

```
$ kubectl -n my-namespace apply -f testdb.yaml
mysql.with.sql.tanzu.vmware.com/mysql-sample created
```

- Verify that the instance was created successfully by running:

```
kubectl -n DEVELOPMENT-NAMESPACE get mysql INSTANCE-NAME
```

Where `INSTANCE-NAME` is the value that you configured for `metadata.name` in your file

For example:

```
$ kubectl -n my-namespace get mysql mysql-sample
NAME          READY   STATUS    AGE
mysql-sample  true    Running   2m17s
```

Delete a MySQL Instance

This section provides some conceptual information about persistent volume claims (PVCs) and deleting an instance. It also provides the procedure for how to delete an instance.

About PVCs

When you create a MySQL instance, the MySQL Operator also creates PVCs. The PVCs are attached to the instance and contain the data for the MySQL database. Single-node instances have one PVC, and high-availability (HA) instances have three.

The PVC name contains the instance name and the MySQL Pod number. The PVC name is of the form `mysql-data-INSTANCE-NAME-N`, for example, `mysql-data-mysql-sample-0`.

About Deleting a MySQL Instance

There are two steps to deleting an instance. The first step is to delete the instance itself, and the second step is to delete the PVCs.

There are situations where you want to delete the instance but not delete the PVC. For example, in a test environment, you might delete the instance to save costs but keep the PVC storing the data. If you later create a new instance with the same name as the deleted instance, the old PVCs automatically reattach to the new instance and you have access to your data again.

Note: If you delete the Pod or the StatefulSet associated with the MySQL resource, the Operator re-creates it for you. To permanently delete the instance, you need to delete the MySQL resource, as described in step 1 below.

Procedure

To delete a MySQL instance:

1. Delete the MySQL instance by running:

```
kubectl -n DEVELOPMENT-NAMESPACE delete mysql INSTANCE-NAME
```

Where:

- `DEVELOPMENT-NAMESPACE` is the namespace where you created the instance.
- `INSTANCE-NAME` is the name of the instance you want to delete.

For example:

```
$ kubectl -n my-namespace delete mysql mysql-sample
my-namespace "mysql-sample" deleted
```

2. (Optional) Delete the PVCs by running:

Warning: This command is destructive. When you destroy your PVCs, you delete all data associated with your Tanzu MySQL for Kubernetes database.

```
kubectl delete pvc -l app.kubernetes.io/instance=INSTANCE-NAME
```

Where `INSTANCE-NAME` is the name of the MySQL instance that you deleted above.

For example:

```
$ kubectl delete pvc -l app.kubernetes.io/instance=mysql-sample
persistentvolumeclaim "mysql-data-mysql-sample-0" deleted
```

Note: For HA MySQL instances, the command deletes all three PVCs associated with the MySQL instance.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Updating MySQL Instances

Page last updated:

This topic describes how to scale or change configurations on a VMware Tanzu™ SQL with MySQL for Kubernetes instance.

Prerequisites

Before you update the MySQL instance, you must have:

- **Access and permissions to the MySQL instance.**
- **The Kubernetes Command Line Interface (kubectl) installed:** For more information, see the [Kubernetes documentation](#).

Scale storageSize

For storage classes that support `storageSize`, you can expand the `storageSize` but not reduce it.

To scale `storageSize`:

1. Target the namespace where you want to scale `storageSize`:

```
kubectl config set-context --current --namespace=DEVELOPMENT-NAMESPACE
```

Where `DEVELOPMENT-NAMESPACE` is the namespace in which you want to scale `storageSize`.

For example:

```
$ kubectl config set-context --current --namespace=my-namespace
```

2. Look up the storage class associated with the MySQL Pod's Persistent Volume Claim (PVC):

```
kubectl get pvc mysql-data-INSTANCE-NAME-N -o jsonpath={.spec.storageClassName}
```

Where:

- `INSTANCE-NAME` is the value that you configured for `metadata.name` for your MySQL resource.
- `N` is the index of the Pod in the MySQL instance.

For example:

```
$ kubectl get pvc mysql-data-mysql-sample-0 -o jsonpath={.spec.storageClassName}
standard
```

3. Check if the storage class allows volume expansion. Look for the `ALLOWVOLUMEEXPANSION` column by running:

```
kubectl get storageclass CLASS-RETURNED
```

Where `CLASS-RETURNED` is the `storageclass` associated with the MySQL Pod's PVC from the previous step.

For example:

```
$ kubectl get storageclass standard
NAME                PROVISIONER          RECLAIMPOLICY  VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION  AGE
standard (default)  kubernetes.io/gce-pd  Delete         Immediate
true                91s
```

4. If the output does not show the `ALLOWVOLUMEEXPANSION` column, run:

```
kubectl get storageclasses.storage.k8s.io standard \
```

```
-o custom-columns='NAME:.metadata.name,ALLOWVOLUMEEXPANSION:.allowVolumeExpansion'
```

- If you are using Kubernetes v1.11 and volume expansion is supported, change the `allowVolumeExpansion` field to `true` in StorageClass objects. Only PVCs created from a StorageClass with `allowVolumeExpansion` set to `true` are allowed to perform volume expansion. For more information, see the [Kubernetes documentation](#).
- Edit the PVC `spec.resources.requests.storage` field to increase the volume size by running:

```
kubectl patch pvc mysql-data-INSTANCE-NAME-N -p \
'{"spec": {"resources": {"requests": {"storage": "NEW-REQUESTED-SIZE"}}}}'
```

Where:

- `INSTANCE-NAME` is the value that you configured for `metadata.name` for your MySQL resource.
- `N` is the index of the Pod in the MySQL instance.
- `NEW-REQUESTED-SIZE` is the increased volume size.

For example:

```
$ kubectl patch pvc mysql-data-mysql-sample-0 -p '{"spec": {"resources": {"requests": {"storage": "50Gi"}}}}'
```

- Do one of the following:
 - If the storageClass supports online expansion:** Wait for the PVC to automatically resize. For more information, see the [Kubernetes documentation](#).
 - Otherwise:** Wait for the PVC to have the `FileSystemResizePending` condition. Delete the MySQL Pod to unmount the PVC and allow it to resize.

```
kubectl wait --for=condition=FileSystemResizePending pvc/mysql-data-INSTANCE-NAME-N
```

```
kubectl rollout restart statefulset INSTANCE-NAME
```

For example:

```
$ kubectl wait --for=condition=FileSystemResizePending pvc/mysql-data-mysql-sample-0
persistentvolumeclaim/mysql-data-mysql-sample-0 condition met

$ kubectl rollout restart statefulset mysql-sample
statefulset.apps/mysql-sample restarted
```

- After the StatefulSet controller has automatically re-created the MySQL Pod, and the MySQL Pod has successfully restarted, verify that the storage has been resized.

```
kubectl exec INSTANCE-NAME-N -c mysql -- df -h /var/lib/mysql
```

Where:

- `INSTANCE-NAME` is the value that you configured for `metadata.name` for your MySQL resource.
- `N` is the index of the Pod in the MySQL instance.

For example:

```
$ kubectl exec mysql-sample-0 -c mysql -- df -h /var/lib/mysql
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb         3.0G  346M   2.6G  12% /var/lib/mysql
```

Expected Downtime When Scaling storageSize

For HA instances, no downtime is expected.

For single-node instances, the expected downtime when expanding storage is as follows:

- With online expansion: There is no downtime expected in most cases.

- With offline expansion: There is downtime while the MySQL Pod is re-created and the PVC is being resized.

Scale CPU and Memory Resources

To scale CPU or memory resources, update the MySQL configuration `spec.resources` field to change the CPU or memory requirements for the `mysql` or `sidecar` containers.

You can use one of these methods to update the configuration:

- Patch the existing API object in place by running:

```
kubectl patch mysql INSTANCE-NAME --type merge -p \
'{"spec": {"resources": {"mysql": MYSQL-RESOURCES, "mysqlSidecar": SIDECAR-RESOURCES}}}'
```

Where:

- `INSTANCE-NAME` is the value that you configured for `metadata.name` for your MySQL resource.
- `MYSQL-RESOURCES` is a collection of `cpu` and `memory` limits and requests for the `mysql` container.
- `SIDECAR-RESOURCES` is a collection of `cpu` and `memory` limits and requests for the `mysql-sidecar` container.

For example:

```
$ kubectl patch mysql mysql-sample --type merge -p '{"spec": {"resources": {"mysql": {"requests": {"cpu": "500m"}}, "mysqlSidecar": {"limits": {"memory": "64Mi"}}}}}'
```

- If you manage MySQL resources with a set of configuration files in source control, you can also change the fields in the file and apply the changes.

For example:

```
$ kubectl apply -f mysql.yaml
```

For more information on managing CPU and Memory resources, see the [Kubernetes documentation](#).

Expected Downtime When Scaling CPU and Memory Resources

For HA instances, no downtime is expected.

For single-node instances, the expected downtime when changing resource reservations is as follows:

- Brief downtime while Kubernetes re-creates the MySQL Pods.
- Risk of longer downtime if the new requested values exceed the available capacity of the Kubernetes cluster.

Change Other Configurations

Changing `storageClassName` or `imagePullSecretName` on a running MySQL instance is not supported and it returns an error. If a MySQL instance is not running due to errors in these fields, you can change them. The changes are propagated into the StatefulSet to correct the error.

Note: You cannot modify `storageClassName` for a successfully created MySQL instance.

To change `storageClassName` or `imagePullSecretName`, update the MySQL configuration by one of the methods below:

- Patch the existing API object in place:

```
kubectl patch mysql INSTANCE-NAME -p '{"spec": {"PROPERTY": "VALUE"}}'
```

Where:

- `INSTANCE-NAME` is the value that you configured for `metadata.name` for your MySQL resource.
- `PROPERTY` is the property name.

- `VALUE` is the new property value.

For example:

```
$ kubectl patch mysql mysql-sample -p '{"spec": {"imagePullSecretName": "new-secret-name"}}'
```

- If you manage MySQL resources with a set of configuration files in source control, you can change the fields in the file and apply the changes.

For example:

```
$ kubectl apply -f mysql.yaml
```

For information about the properties, see [Property Reference for the MySQL Resource](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Upgrading MySQL Instances

Page last updated:

This topic describes how to upgrade a Tanzu MySQL for Kubernetes instance after [upgrading the operator](#).

Prerequisites

Before you upgrade a MySQL instance, you must have:

- **Upgraded the MySQL operator.** See [Upgrading the Tanzu SQL for Kubernetes Operator](#).
- **Access to and permissions for the MySQL instance.**
- **The Kubernetes Command Line Interface (kubectl) installed.** For more information, see the [Kubernetes documentation](#).

List Instances By Version

Each MySQL instance has a `version` label containing the version of the instance. The instance version corresponds to a version of Tanzu MySQL for Kubernetes.

Note: The `version` label was added in Tanzu MySQL for Kubernetes version 1.1.0. Upgrading the MySQL operator to version 1.1.0 or later causes the `version` label to be applied to existing MySQL instances automatically.

To list all MySQL instances at a specific version, run:

```
kubectl get mysql -l app.kubernetes.io/version=VERSION
```

Where `VERSION` is the version of VMware Tanzu™ SQL with MySQL for Kubernetes.

For example:

```
$ kubectl get mysql -l app.kubernetes.io/version=1.0.0
NAME          READY   STATUS    VERSION   AGE
mysql-sample   true    Running   1.0.0     22m
```

Upgrade an Instance

Each VMware Tanzu™ SQL with MySQL for Kubernetes instance is based on a copy of the `mysql.yaml` deployment template file. The file for the instance should have a unique name (for example, `testdb.yaml`). To upgrade an instance, you must edit this configuration file.

To upgrade a MySQL instance:

1. List all the instances across all namespaces in the cluster:

```
kubectl get mysql -A
```

which returns an output similar to:

NAMESPACE	NAME	READY	STATUS	VERSION	AGE
my-namespace	mysql-sample-2	true	Running	1.0.0	2h31m
acceptance	mysql-sample	true	Running	1.1.0	6h42m

2. Target the namespace of the MySQL instance:

```
kubectl config set-context --current --namespace=DEVELOPMENT-NAMESPACE
```

Where `DEVELOPMENT-NAMESPACE` is the namespace in which the instance was created.

For example:

```
$ kubectl config set-context --current --namespace=my-namespace
```

3. Edit the configuration file for the MySQL instance, setting the `spec.version` property to the desired new version. (This property was added in VMware Tanzu™ SQL with MySQL for Kubernetes version 1.1.0.)

For example:

```
---
apiVersion: with.sql.tanzu.vmware.com/v1
kind: MySQL
metadata:
  name: mysql-sample
spec:
  version: 1.1.0
...
```

4. If you are upgrading the instance from VMware Tanzu™ SQL with MySQL for Kubernetes version 1.0 to version 1.1, you may need to update the `spec.imagePullSecret` property. In version 1.1.0, this property was renamed to `imagePullSecretName`, and the default secret name was changed from `tanzu-mysql-image-registry` to `tanzu-image-registry`. Make sure that you use the correct property name and value in the configuration file for the MySQL instance.

For example, if you wish to continue using a secret named `tanzu-mysql-image-registry`:

```
---
apiVersion: with.sql.tanzu.vmware.com/v1
kind: MySQL
metadata:
  name: mysql-sample
spec:
  version: 1.1.0
  imagePullSecretName: tanzu-mysql-image-registry
...
```

Alternatively, you can remove the property from the configuration file. If you remove the property, the MySQL instance will use the `imagePullSecretName` specified in the `values-override.yaml` or `values.yaml` file for the MySQL operator.

5. Apply your changes by running:

```
kubectl apply -f FILENAME
```

Where `FILENAME` is the name of the configuration file.

For example:

```
$ kubectl apply -f testdb.yaml
mysql.with.sql.tanzu.vmware.com/mysql-sample configured
```

6. Verify that the instance was upgraded successfully by running:

```
kubectl get mysql INSTANCE-NAME
```

Where `INSTANCE-NAME` is the value configured for `metadata.name` in the configuration file.

When the instance has been successfully upgraded, the value of the `READY` column for the instance should be `true` and the value of the `VERSION` column for the instance should report the new version. For example:

```
$ kubectl get mysql mysql-sample
NAME          READY   STATUS    VERSION   AGE
mysql-sample  true    Running   1.1.0     24m
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Configuring MySQL Instances for High Availability

This topic describes how to use VMware Tanzu™ SQL with MySQL for Kubernetes to create high-availability (HA) MySQL instances.

Overview

High-availability (HA) MySQL instances offer automatic failover, ensuring that app requests operate continuously and without extended downtime.

For more information about high availability including an architecture diagram, see [Architecture of an HA MySQL Instance](#) in *Architecture*.

About Planning for Long-Lived HA MySQL Instances

HA MySQL instances are often used in production environments. Before creating an HA instance that you intend to rely on for a long time, carefully consider the resource requests and limits.

If you do not designate resource requests and limits, then Kubernetes schedules Pods onto node resources according to best effort policies. If resources become constrained, your Pods running MySQL risk eviction as Kubernetes manages resources allocation among its Pods. For information about the properties used to set limits and requests, see [Property Reference for the MySQL Resource](#).

About Explicit Node Selection and Anti-Affinity

VMware Tanzu™ SQL with MySQL for Kubernetes does not support explicit node selection or anti-affinity for controlling Pod assignment to Kubernetes nodes. Kubernetes distributes Pods according to its regular placement and scheduling policies including any resource requests.

About Scaling Single Node to HA

You can convert a single-node MySQL instance to an HA MySQL instance with a single command. See [Convert a Single-Node MySQL Instance to an HA MySQL Instance](#) below.

About Scaling HA to Single Node

It is not straightforward to convert an HA instance to a single-node instance. To move from an HA instance to a single-node instance, you must take a backup of the HA instance and restore it to a new single-node instance. See [Move an HA MySQL Instance to a Single-Node MySQL Instance](#) below.

About Backing Up HA Instances

HA MySQL instances are backed up and restored the same way as single-node MySQL instances. Backups from HA instances are created from the primary MySQL Pod. You can restore the backups from HA instances to either single-node or HA instances. For information about backup and restore, see [Backing Up and Restoring MySQL Instances](#).

Prerequisites

Before you can configure high availability, you must have:

- The Kubernetes Command Line Interface (kubectl) installed. For more information, see the [Kubernetes documentation](#).
- Full admin access to all Kubernetes resources in your developer namespace.

For information about Roles and RoleBindings that your Kubernetes cluster admin needs to create, see the [Kubernetes documentation](#).

- The URL and credentials to access the registry that stores the Tanzu MySQL for Kubernetes images. This can be the VMware Tanzu Network registry or the private registry configured

for your environment. If you do not have access to the registry credentials, contact your Kubernetes admin to have these set up for you in your namespace.

- The VMware Tanzu™ SQL with MySQL for Kubernetes deployment templates. See [Download the Deployment Templates](#).
- Reviewed how to create and delete single-node MySQL instances. See [Creating and Deleting MySQL Instances](#).

Create an HA MySQL Instance

Creating an HA MySQL instance is very similar to creating a single-node MySQL instance. By default, MySQL instances are single node.

To create an HA instance:

1. Follow the steps 1–4 in [Create a MySQL Instance](#).
2. Edit your uniquely named copy of `mysql.yaml`:
 1. Set the value of `highAvailability.enabled` to `true`:

```
---
apiVersion: with.sql.tanzu.vmware.com/v1
kind: MySQL
metadata:
  name: mysql-ha-sample
spec:
  storageSize: 1Gi
  imagePullSecretName: tanzu-image-registry
  highAvailability:
    enabled: true
```

2. Edit the other properties for the instance as needed. For information about the properties that you can set for the MySQL resource, see [Property Reference for the MySQL Resource](#).
3. Create and verify the instance by following steps 6 and 7 in [Create a MySQL Instance](#).

Convert a Single-Node MySQL Instance to an HA MySQL Instance

If a single-node MySQL instance already exists, you can edit its YAML file to set high-availability and then redeploy the instance.

Note: Ensure that you want an HA instance before following this procedure. You cannot easily change an HA instance back to a single-node one.

To change a single-node instance into an HA instance:

1. Open the YAML file that you created in step 4 of [Create a MySQL Instance](#) in [Creating and Deleting MySQL Instances](#).
2. Change the value of `spec.highAvailability.enabled` to `true` and save the file.
3. Apply the change and deploy the HA instance by running:

```
kubectl -n DEVELOPMENT-NAMESPACE apply -f FILENAME
```

Where:

- `DEVELOPMENT-NAMESPACE` is the namespace for the instance.
- `FILENAME` is the name of the YAML file edited.

For example:

```
$ kubectl -n my-namespace apply -f testdb.yaml
mysql.with.sql.tanzu.vmware.com/mysql-sample created
```

Inspect an HA MySQL Instance

After you have created your HA MySQL instance, you can inspect the instance to confirm that all the Pods are running.

This inspection procedure can also be used for troubleshooting.

1. Connect to one of the instance Pods, by running:

```
kubectl exec -it POD-NAME --container=mysql -- bash
```

Where `POD-NAME` is the name of the primary Pod or one of the secondary Pods. The Pod name is the instance name appended with an index of 0–2. When the instance is first created, the index for the primary Pod is 0.

For example:

```
$ kubectl exec -it mysql-ha-sample-0 --container=mysql -- bash
```

2. On the Pod, log in to the MySQL database by running:

```
mysql --user=USER --password=PASSWORD
```

Where:

- `USER` is a username on the database. When the database is first created, only the root user exists.
- `PASSWORD` is the password for the database user.

For example:

```
mysql@mysql-sample-0:/$ mysql --user=root --password=$(cat $MYSQL_ROOT_PASSWORD
_FILE)
```

3. Query the database by running:

```
mysql>SELECT * FROM performance_schema.replication_group_members\G;
```

The following example is from a healthy cluster with a single member designated `PRIMARY` and two `SECONDARY` members:

```
mysql>SELECT * FROM performance_schema.replication_group_members\G;
***** 1. row *****
CHANNEL_NAME: group_replication_applier
MEMBER_ID: 157baa2a-8c22-11eb-847c-0242ac110009
MEMBER_HOST: mysql-ha-sample-0.mysql-ha-sample-members.default.svc.cluster.local
MEMBER_PORT: 3306
MEMBER_STATE: ONLINE
MEMBER_ROLE: PRIMARY
MEMBER_VERSION: 8.0.22
***** 2. row *****
CHANNEL_NAME: group_replication_applier
MEMBER_ID: 281ad3c9-8c22-11eb-b3aa-0242ac11000a
MEMBER_HOST: mysql-ha-sample-1.mysql-ha-sample-members.default.svc.cluster.local
MEMBER_PORT: 3306
MEMBER_STATE: ONLINE
MEMBER_ROLE: SECONDARY
MEMBER_VERSION: 8.0.22
***** 3. row *****
CHANNEL_NAME: group_replication_applier
MEMBER_ID: 3c52bb9a-8c22-11eb-aade-0242ac11000b
MEMBER_HOST: mysql-ha-sample-2.mysql-ha-sample-members.default.svc.cluster.local
MEMBER_PORT: 3306
MEMBER_STATE: ONLINE
MEMBER_ROLE: SECONDARY
MEMBER_VERSION: 8.0.22
```

4. Review the value of `MEMBER_STATE`. A healthy cluster shows `ONLINE` for all members. For information about the Group Replication server states, see the [MySQL documentation](#).

Move an HA MySQL Instance to a Single-Node MySQL Instance

To move from an HA MySQL instance to a single-node MySQL instance, you need to back up the HA instance and restore the backup to a single-node instance.

1. Back up the HA instance. See [Back Up Tanzu SQL for Kubernetes Data](#) in *Backing Up and Restoring MySQL Instances*.

2. Restore the backup to a single-instance node. See [Restore from a Backup](#) in *Backing Up and Restoring MySQL Instances*. When you edit your copy of the `restore.yaml` file, set `highAvailability.enabled` to `false`.

Note: You cannot scale down an existing HA instance to a single-node instance by applying `highAvailability.enabled: false` to an HA instance. The `apply` command fails.

Delete an HA MySQL Instance

Deleting an HA MySQL instance is the same as deleting a single-node instance.

To delete an HA instance:

1. See [Delete a MySQL Instance](#) in *Creating and Deleting MySQL Instances*.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Monitoring MySQL Instances in Kubernetes

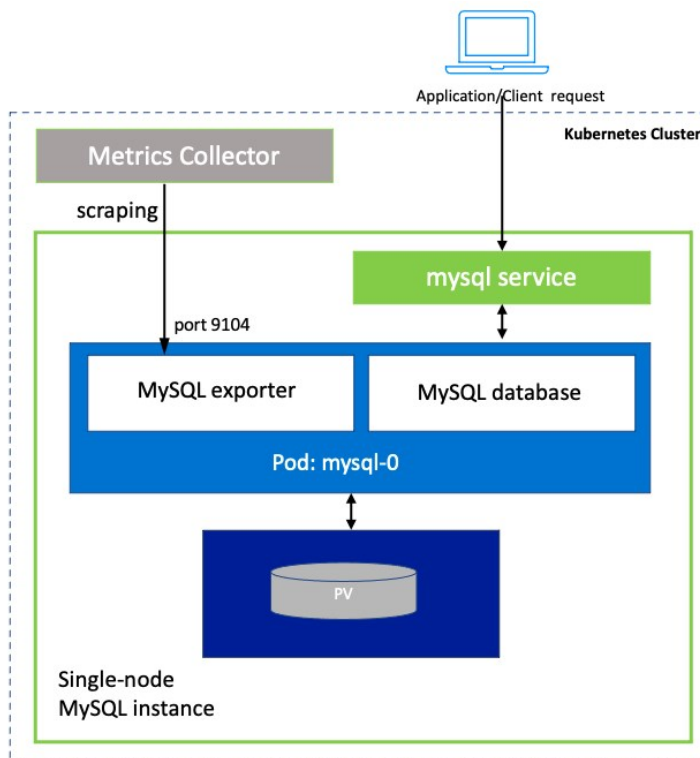
This topic describes how to collect metrics and monitor VMware Tanzu™ SQL with MySQL for Kubernetes instances in a Kubernetes cluster.

Overview

Tanzu MySQL for Kubernetes uses the [MySQL Server Exporter](#), a Prometheus exporter for MySQL server metrics. The Prometheus exporter provides an endpoint for Prometheus to scrape metrics from different application services. The MySQL Server Exporter shares metrics about the MySQL instances.

Upon initialization, each MySQL pod adds a MySQL server exporter container. Prometheus sends HTTPS requests to the exporter. The exporter queries the MySQL database and provides metrics in the Prometheus format on a `/metrics` https endpoint (port 9104) on the pod, conforming to the Prometheus HTTP API.

The diagram below shows the architecture of a single-node MySQL instance with MySQL server exporter, where the metrics are exported on port 9104:



[Click here to view a larger version of this diagram](#)

Prometheus could be your primary consumer of the metrics, but any monitoring tool can take advantage of the `/metrics` endpoint.

Prerequisites

To take advantage of the metrics endpoint, ensure you have a metrics collector in your environment like Prometheus, or Wavefront. For an example installation of Prometheus, see [Using Prometheus Operator to Scrape the Tanzu MySQL for Kubernetes Metrics](#).

Verifying MySQL Metrics

MySQL pods include the exporter that emits metrics by default. To test that the metrics are being

emitted, you may use port forwarding (for more details see [Use Port Forwarding to Access Applications in a Cluster](#) in the Kubernetes documentation):

```
$ kubectl port-forward pod/<mysql-pod-name> 9104:9104
```

And then in another shell, use a tool like `curl` to run:

```
$ curl -k https://localhost:9104/metrics
```

The output shows the metrics collection, for example:

```
mysql_global_status_open_tables 165
```

For a list of the exposed Tanzu MySQL metrics collectors see [MySQL Server Exporter Collector Flag Reference](#).

Using Prometheus Operator to Scrape the Tanzu MySQL for Kubernetes Metrics

[Prometheus Operator](#) allows you to define and manage monitoring instances as Kubernetes resources. This section provides an example of installing the Prometheus Operator using Helm, and an example Prometheus `PodMonitor` CRD that is used to scrape the MySQL metrics.

- Use Helm to install the Prometheus Operator, that defines the custom resource definitions (CRDs) such as `PodMonitor`. The `PodMonitor` CRD defines the configuration details for the MySQL pod monitoring.

```
$ helm install prometheus-community/kube-prometheus-stack \
--create-namespace \
--namespace=prometheus \
--set prometheus.service.port=80 \
--set prometheus.service.type=LoadBalancer \
--set grafana.enabled=false,alertmanager.enabled=false \
--set prometheus-node-exporter.hostRootFsMount=false \
--wait
```

- Confirm the `PodMonitor` CRD exists using:

```
$ kubectl get customresourcedefinitions.apiextensions.k8s.io podmonitors.monitoring.coreos.com
```

- Create a `PodMonitor` that scrapes all MySQL instances every 10 seconds and skips verifying TLS:

```
cat <<EOF | kubectl apply -f -
---
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: tanzu-mysql-instances
  namespace: prometheus
spec:
  namespaceSelector:
    any: true
  selector:
    matchLabels:
      app.kubernetes.io/component: database
      app.kubernetes.io/name: mysql
  podTargetLabels:
    - app.kubernetes.io/instance
  podMetricsEndpoints:
    - port: "mysql-metrics"
      interval: "10s"
      scheme: https
      tlsConfig:
        insecureSkipVerify: true
EOF
```

- To check if Prometheus is successfully monitoring the instances, open the Prometheus UI in the browser and visit the `/targets` URI to check the status of the `PodMonitor` under `podMonitor/prometheus/tanzu-mysql-instances/0 (1/1 up)`.

You should view something similar to:

PodMonitor/prometheus/mysql-instances/0 (1/1 up) [show logs](#)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://100.96.113-9104/metrics	UP	app_kubernetes_io_instance="mysql-sample" containers="mysql-exporter" endpoint="mysql-metrics" instance="100.96.113-9104" job="prometheus/mysql-instances" namespace="acceptance" pod="mysql-sample-0"	7.649s ago	48.260ms	

[Click here to view a larger version of this diagram](#)

For details on the `PodMonitor` API see `PodMonitor` in the Prometheus Operator documentation.

Using TLS for the Metrics Endpoint

During the Tanzu MySQL initialization, the Tanzu MySQL operator creates a metrics related self-signed certificate. The TLS credentials are stored in a Secret named after the MySQL instance name: if the instance name is `mysql-db`, the metrics Secret name is `mysql-db-metrics-tls`.

In order to use TLS for a metrics endpoint, configure Prometheus to use the CA certificate from the Secret. Use the following command to fetch the CA certificate:

```
$ kubectl get secret <MYSQL-INSTANCE-NAME>-metrics-tls -o 'go-template={{index .data "ca.crt" | base64decode}}'
```

where `<MYSQL-INSTANCE-NAME>` is the name of the MySQL instance.

For details on how to configure Prometheus with TLS, see `tls_config` in the [Prometheus Configuration](#) documentation.

MySQL Server Exporter Collector Flag Reference

Collectors are logical groups of metrics. Currently, the following collectors are enabled for Tanzu MySQL for Kubernetes.

- `global_status`
- `global_variables`
- `slave_status`
- `innodb_cmp`
- `innodb_cmpmem`
- `query_response_time`
- `replication_group_members`
- `replication_group_member_stats`
- `replication_applier_status_by_worker`

For more details on the collector list, see [MySQL Server Exporter Collector Flags](#).

[Create a pull request](#) or [raise an issue](#) on the source for this page in GitHub

Rotating MySQL Credentials

Page last updated:

This topic describes how to rotate the MySQL root password and the MySQL backup user password.

Overview

When a user provisions a MySQL instance, the MySQL Operator automatically creates a Kubernetes secret containing the MySQL root password as well as the password for the MySQL backup user.

To adhere to security best practices or to company regulations, VMware recommends rotating a MySQL instance's credentials regularly. In addition, you should rotate a password if it is compromised for any reason.

This topic provides multiple methods for rotating passwords for a MySQL instance.

- **Option 1: Delete the Kubernetes Secret:** Kubernetes automatically re-creates the secret with newly generated passwords.
- **Option 2: Patch the Kubernetes Secret with a Custom Password:** This procedure enables you to configure MySQL with your own custom passwords. It also enables you to rotate the root password and the backup user password individually.

Prerequisites

Before you rotate a MySQL instance's password, you need:

- The Kubernetes Command Line Interface (kubectl) installed. For more information, see the [Kubernetes documentation](#).
- `admin` Role access to the namespace of the MySQL instance for which you want to rotate the root password. For more information about User-facing roles, see the [Kubernetes documentation](#).

Option 1: Delete the Kubernetes Secret

This option deletes the Kubernetes secret containing the MySQL passwords. When the secret is deleted, Kubernetes automatically re-creates the secret with newly generated passwords. This procedure rotates both the MySQL root password and the backup user password.

1. Delete the Kubernetes secret by running:

```
kubectl delete secret INSTANCE-NAME-credentials
```

Where `INSTANCE-NAME` is the name of the MySQL instance.

For example:

```
$ kubectl delete secret mysql-sample-credentials
secret "mysql-sample-credentials" deleted
```

2. Wait until Kubernetes has automatically re-created the secret. You can watch the progress by running:

```
kubectl get secret --watch
```

For example:

```
$ kubectl get secret --watch
```

NAME	TYPE	DATA
A AGE		
default-token-wb7gl	kubernetes.io/service-account-token	3
10d		
mysql-sample-credentials	Opaque	4 4

```

8s
tanzu-mysql-backup-cron-token-c7bnt    kubernetes.io/service-account-token    3
10d
tanzu-mysql-image-registry              kubernetes.io/dockerconfigjson         1
2m3s
tanzu-mysql-token-24cdv                kubernetes.io/service-account-token    3
10d

```

- Update the database with the new passwords by restarting your MySQL instance:

```
kubectl rollout restart statefulset INSTANCE-NAME
```

For example:

```

$ kubectl rollout restart statefulset mysql-sample

statefulset.apps/mysql-sample restarted

```

- Verify that your MySQL instance has finished updating by running:

```
kubectl get mysql INSTANCE-NAME
```

A MySQL instance has finished updating when the value of the `STATUS` column is `Running`.
For example:

```

$ kubectl get mysql mysql-sample

NAME          READY   STATUS    AGE
mysql-sample   true    Running   10d

```

- To verify that the passwords were rotated successfully, try connecting to your MySQL instance. See [Accessing MySQL Instances](#).

Option 2: Patch the Kubernetes Secret with a Custom Password

This option patches the existing Kubernetes secret with a new password. This procedure allows you to configure MySQL with your own custom passwords. You can use this procedure to rotate either the MySQL root password or the backup user password.

- Patch the secret with your custom password by running:

```
kubectl patch secret INSTANCE-NAME-credentials -p='{"stringData":{"PASSWORD-FIELD":"CUSTOM-PASSWORD"}}'
```

Where:

- `INSTANCE-NAME` is the name of the MySQL instance.
- `PASSWORD-FIELD` is either `rootPassword` if you are changing the MySQL root password or `backupPassword` if you are changing the MySQL backup user password.
- `CUSTOM-PASSWORD` is your custom password in plaintext. Kubernetes stores this password as a base64-encoded string in the Kubernetes secret.

For example:

```

$ kubectl patch secret mysql-sample-credentials -p='{"stringData":{"rootPassword":"examplepassword"}}'

secret/mysql-sample-credentials patched

```

- To update the database with the new password, restart your MySQL instance by running:

```
kubectl rollout restart statefulset INSTANCE-NAME
```

For example:

```

$ kubectl rollout restart statefulset mysql-sample

statefulset.apps/mysql-sample restarted

```

- Verify that your MySQL instance has finished updating by running:

```
kubectl get mysql INSTANCE-NAME
```

A MySQL instance has finished updating when the value of the `STATUS` column is `Running`.
For example:

```
$ kubectl get mysql mysql-sample
```

NAME	READY	STATUS	AGE
mysql-sample	true	Running	10d

4. To verify that the password was rotated successfully, try connecting to your MySQL instance.
See [Accessing MySQL Instances](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Accessing MySQL Instances

Page last updated:

This topic describes how to access an instance of VMware Tanzu™ SQL with MySQL for Kubernetes.

Prerequisites

Before accessing a MySQL instance, you must have:

- The Kubernetes Command Line Interface (kubectl) installed. For more information, see the [Kubernetes documentation](#).

(Optional) Verify MySQL Instance Settings

To see all MySQL instance settings configured:

1. Access the MySQL container.

```
kubectl -n DEVELOPMENT-NAMESPACE exec --stdin --tty pod/INSTANCE-NAME-0 -c mysql
-- /bin/bash
```

In this command, replace:

- `DEVELOPMENT-NAMESPACE` with the namespace of the MySQL instance
- `INSTANCE-NAME` with the name of the instance

For example:

```
$ kubectl -n my-namespace exec --stdin --tty pod/mysql-sample-0 -c mysql -- /bin/bash
```

2. Review the configuration files `/etc/mysql/conf.d/base.cnf` and `/etc/mysql/conf.d/autotune.cnf`.

Get Root Access to the MySQL Server

Database administrative operations, such as creating users and databases, require connecting to MySQL as the root database user. Root account connections are only allowed from within the container running the database. Off-container root connections are not permitted.

To connect to the MySQL instance as the MySQL root user:

1. If the MySQL instance is configured for high availability, identify the primary (writable) Pod of the MySQL instance by running:

```
kubectl exec -it INSTANCE-NAME-0 -c mysql -- bash -c \
  "mysql -B -s --user=root --password=$(cat $MYSQL_ROOT_PASSWORD_FILE) \
  --execute 'SELECT MEMBER_HOST FROM performance_schema.replication_group_members \
  WHERE (MEMBER_ROLE=\"PRIMARY\")'"
```

Where `INSTANCE-NAME` is the name of the instance.

For example:

```
$ kubectl exec -it mysql-sample-0 -c mysql -- bash -c \
  "mysql -B -s --user=root --password=$(cat $MYSQL_ROOT_PASSWORD_FILE) \
  --execute 'SELECT MEMBER_HOST FROM performance_schema.replication_group_members \
  WHERE (MEMBER_ROLE=\"PRIMARY\")'"

mysql: [Warning] Using a password on the command line interface can be insecure
mysql-sample-1.mysql-sample-members.default.svc.cluster.local
```

This command returns the Kubernetes-internal domain name of the primary node. The

primary Pod name is the first dot-separated component of the command output. In the example above, the primary Pod name is `mysql-sample-1`.

2. Access the MySQL container by running:

```
kubectl exec --stdin --tty pod/POD-NAME -c mysql -- /bin/bash
```

Where `POD-NAME` is the name of the Pod: the MySQL instance name appended with an index of 0–2. If the MySQL instance is single-node, the index is 0.

For example:

```
$ kubectl exec --stdin --tty pod/mysql-sample-0 -c mysql -- /bin/bash
mysql@mysql-sample-0:/$
```

3. Log in to the MySQL server by running:

```
mysql -uroot -p$(cat $MYSQL_ROOT_PASSWORD_FILE)
```

Access the MySQL Server from an External IP Address

Note: This procedure requires that your cloud provider supports external load balancers. For more information about the `LoadBalancer` service type, see the [Kubernetes documentation](#).

To connect to a MySQL instance from outside of your Kubernetes cluster, you must configure the Kubernetes service for the instance to be of type `LoadBalancer`.

To access the MySQL server from an external IP address:

1. Create a database user to use for the external connection. For more information, see [Create a Database and Privileged MySQL User for the App](#) in *Connecting Apps to MySQL Instances*.
2. Determine if the `ServiceType` is `LoadBalancer` or the default `ClusterIP`:

```
kubectl get service INSTANCE-NAME -o jsonpath={.spec.type}
```

In this command, replace `INSTANCE-NAME` with the value of the `metadata.name` property for the MySQL instance.

For example:

```
$ kubectl get service mysql-sample -o jsonpath={.spec.type}
LoadBalancer
```

3. If the `ServiceType` is not `LoadBalancer`, change the value of the `Spec.ServiceType` property to `LoadBalancer`.
For information about updating an instance, see [Updating MySQL Instances](#). For information about the property, see [Property Reference for the MySQL Resource](#).
4. Find the external IP address allocated for the service.

```
kubectl get service INSTANCE-NAME -o jsonpath={.status.loadBalancer.ingress[0].ip}
```

In this command, replace `INSTANCE-NAME` with the value of the `metadata.name` property for the MySQL instance.

For example:

```
$ kubectl get service mysql-sample -o jsonpath="{.status.loadBalancer.ingress[0]['ip', 'hostname']}"
192.168.64.200
```

This command retrieves either an IP address or a resolvable DNS hostname.

If the command returns a DNS hostname, use the hostname in place of the IP address in the following examples. For example, an AWS load balancer returns a domain name instead of an IP address.

```
$ kubectl get service mysql-sample -o jsonpath="{.status.loadBalancer.ingress[0]['ip', 'hostname']}"
```



```
a4dc8de1b1efe13112-17761231.us-west-2.elb.amazonaws.com
```

5. Log in to the MySQL server.

```
mysql -u USERNAME -pUSER-PASSWORD -P 3306 -h EXTERNAL-IP
```

In this command, replace:

- `USERNAME` with the name of the MySQL user (created in Step 1)
- `USER-PASSWORD` with the password you assigned to the MySQL user (created in Step 1)
- `EXTERNAL-IP` with the external IP address allocated for the MySQL service (retrieved in Step 3)

For example:

```
$ mysql -u report_admin -phunter2 -P 3306 -h a4dc8de1b1efe13112-17761231.us-west-2.elb.amazonaws.com
```

Turn Off External Access

To disable off-platform connections:

1. In the YAML file for the instance, change the value of the `Spec.ServiceType` property to `ClusterIP` and update the MySQL instance. For more information about how to change a property, see [Change Other Configurations](#) in *Updating MySQL Instances*.

Note: When you change the `Spec.ServiceType` from `LoadBalancer` to `ClusterIP`, your cloud provider might automatically delete the associated load balancer for the MySQL instance.

Access MySQL From Cluster-Hosted Applications

Each MySQL instance has a Kubernetes service named after the instance. The service is used to connect an app to its MySQL database.

If an app is deployed on the Kubernetes cluster, the app can access the MySQL instance using the DNS name of the MySQL service. This DNS name is only resolvable within the Kubernetes cluster.

An app should connect to the database using a narrowly-privileged database user created for the specific requirements of the app. For information about creating a user for an app and connecting the app to a database using specific user and configuration information, see [Connecting Apps to MySQL Instances](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Configuring TLS for MySQL Instances

Page last updated:

This topic describes how to configure TLS for a MySQL instance.

Overview

Tanzu MySQL for Kubernetes is configured to require an encrypted connection, and client connections cannot fall back to use an unencrypted connection. By default, the MySQL server uses a self-signed certificate. Clients can connect to the MySQL server, but they cannot verify the connection.

To verify the connection to the MySQL server, the MySQL instance can be configured with a TLS certificate and private key.

To configure a MySQL instance for TLS, you must first create a TLS Secret in the same namespace as the MySQL instance. There are several ways to create the Secret. This topic describes two methods:

- Using the Kubernetes Command Line Interface (kubectl). See [Create the TLS Secret Manually](#) below.
- Using cert-manager. See [Create TLS Secret with cert-manager](#) below.

After creating the secret, you add the name of the secret to your copy of the `mysql.yaml` file, and configure the instance with the updated file, as described in [Configure MySQL for TLS](#) below.

For general information about TLS secrets, see the [Kubernetes documentation](#).

Prerequisites

Before you configure TLS for a MySQL instance, you must have:

- **Access and permissions to the MySQL instance.**
- **The Kubernetes Command Line Interface (kubectl) installed:** For more information, see the [Kubernetes documentation](#).

Create the TLS Secret Manually

This procedure describes how to create the TLS Secret using kubectl. To create the TLS Secret using cert-manager instead, see [Create TLS Secret with cert-manager](#) below.

1. Generate a certificate and private key using a certificate manager, such as OpenSSL, certstrap, or Let's Encrypt.

When creating the certificate, supply the server hostname for the subject alternative names (SANs). The server hostname is the DNS name that you use when connecting your app to the MySQL instance:

- If your apps are deployed in the same Kubernetes cluster as your instance:
The hostname is `INSTANCE-NAME.DEVELOPMENT-NAMESPACE`, for example, `mysql-sample.my-namespace`.
- If your app is deployed outside the Kubernetes cluster and you have configured the MySQL `spec.serviceType` as `LoadBalancer`:
The hostname is the external DNS name of the load balancer. See [Access the MySQL Server from an External IP Address](#) in *Accessing MySQL Instances*.

2. Create the TLS Secret by running:

```
kubectl -n DEVELOPMENT-NAMESPACE create secret generic TLS-SECRET-NAME \
--type kubernetes.io/tls \
--from-file=tls.crt=PATH-TO-CERTIFICATE \
--from-file=tls.key=PATH-TO-PRIVATE-KEY \
--from-file=ca.crt=PATH-TO-CERTIFICATE-AUTHORITY
```

Where:

- `DEVELOPMENT-NAMESPACE` is the namespace for the MySQL instance.
- `TLS-SECRET-NAME` is the name you choose for the TLS Secret.
- `PATH-TO-CERTIFICATE` is the file path to the certificate created in the step above.
- `PATH-TO-PRIVATE-KEY` is the file path to the private key created in the step above.
- `PATH-TO-CERTIFICATE-AUTHORITY` is the file path to the certificate authority that signed the certificate.

For example:

```
$ kubectl -n my-namespace create secret generic mysql-tls-secret \
  --type kubernetes.io/tls \
  --from-file=tls.crt=/path/server.crt \
  --from-file=tls.key=/path/server.key \
  --from-file=ca.crt=/path/server_ca.crt
```

After completing the steps above, you add the secret to the instance by following the instructions in [Configure MySQL for TLS](#) below.

Create TLS Secret with cert-manager

This procedure describes how to create the TLS Secret using cert-manager.

To create the TLS Secret through the kubectl instead, see [Create the TLS Secret Manually](#) above.

1. If you have not yet installed cert-manager on your Kubernetes cluster, install it by running these commands:

```
kubectl create namespace cert-manager
helm repo add jetstack https://charts.jetstack.io
helm repo update
helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager --version v1.0.2 --set installCRDs=true
```

2. Use cert-manager to create either a cluster-wide `ClusterIssuer` resource or a namespace-local `Issuer` resource in the same namespace as your MySQL instance.

For information about the `Issuer` types, see the [cert-manager documentation](#).

Note: Because the MySQL instance requires that the TLS secret include the `ca.crt` key, VMware does not recommend the ACME Issuer.

3. Choose and record a name for the TLS secret name, for example, `mysql-tls-secret`. Use this name as the `spec.secretName` when you create the `Certificate` resource below.
4. Create a certificate request YAML. For instructions, see the [cert-manager documentation](#). The following table suggests values for the YAML parameters.

Enter:	As this parameter in the YAML:
your TLS secret name from step 3 above, for example, <code>mysql-tls-secret</code>	<code>spec.secretName</code>
the name of your <code>ClusterIssuer</code> Or <code>Issuer</code> created in step 2 above	<code>spec.issuerRef.name</code>
the DNS name or the fully qualified DNS name that you use to connect to apps deployed in the same Kubernetes cluster as the MySQL instance: <ul style="list-style-type: none"> • <code>INSTANCE-NAME.DEVELOPMENT-NAMESPACE</code>, for example, <code>mysql-sample.my-namespace</code> • <code>INSTANCE-NAME.DEVELOPMENT-NAMESPACE.svc.CLUSTER-DOMAIN</code>, for example, <code>mysql-sample.my-namespace.svc.cluster.local</code>. The default <code>CLUSTER-DOMAIN</code> is <code>cluster.local</code>. 	<code>spec.dnsNames</code>
the external IP address of the load balancer that you use to connect to apps deployed outside the cluster. To find the address, see step 4 of Access the MySQL Server from an External IP Address .	<code>spec.ipAddresses</code>
any additional DNS names that clients use to access the MySQL instance through a TLS connection	<code>spec.dnsNames</code>
any additional URIs that clients use to access the MySQL instance through a TLS connection you might have configured to access your MySQL	<code>spec.uris</code>
any additional IP addresses that clients use to access the MySQL instance through a TLS connection	<code>spec.ipAddresses</code>

For information about troubleshooting certificate creation, see the [cert-manager documentation](#).

5. Verify that the TLS secret created has the `ca.crt` key by running:

```
kubectl -n DEVELOPMENT-NAMESPACE get secret TLS-SECRET-NAME -o jsonpath="{.data['ca\.crt']}"
```

Where:

- `DEVELOPMENT-NAMESPACE` is the namespace for the MySQL instance.
- `TLS-SECRET-NAME` is the name you chose for the TLS Secret.

For example:

```
$ kubectl -n my-namespace get secret mysql-tls-secret -o jsonpath="{.data['ca\.crt']}"
```

Configure MySQL for TLS

To configure TLS for the MySQL instance:

1. In your copy of the `mysql.yaml` for the MySQL instance, specify `spec.tls.secret.name` as the name of the TLS Secret created in the namespace.
2. Create or update the MySQL instance by running:

```
kubectl apply -f FILENAME -n DEVELOPMENT-NAMESPACE
```

Where:

- `FILENAME` is the name of the configuration file for your MySQL resource.
- `DEVELOPMENT-NAMESPACE` is the namespace for the MySQL instance.

Connect to a Load-Balanced MySQL Instance over TLS

This section shows how to connect to a MySQL instance configured with a load balancing service, `spec.serviceType.LoadBalancer`, from an off-cluster machine running the MySQL command-line client.

In the procedure below, you give the client the certificate of the CA that signed the MySQL TLS certificate. The client uses this CA to authenticate the MySQL server-provided TLS certificate.

To connect the MySQL command-line client to the MySQL instance over TLS:

1. Obtain the certificate of the CA that signed the MySQL server's TLS certificate.

The CA certificate is stored in the TLS secrets `ca.crt` field, base64 encoded. Save that certificate to a local file `ca.crt` by running:

```
kubectl -n DEVELOPMENT-NAMESPACE get secret TLS-SECRET-NAME -o jsonpath="{.data['ca\.crt']}" | base64 -d > ca.crt
```

For example:

```
$ kubectl -n my-namespace get secret mysql-tls-secret -o jsonpath="{.data['ca\.crt']}" | base64 -d > ca.crt
```

2. Create a MySQL username and password to test your connection with. You need to connect as a non-root user because MySQL instances prohibit root connection from remote machines.

For an example of creating a database, user, and password for connection testing, see [Connecting Apps to MySQL Instances](#).

3. Obtain the IP address that your MySQL instance is listening on for connections. Load-balanced instances expose their IP address through their `status.loadBalancer.ingress` property:

```
kubectl get service INSTANCE-NAME -o jsonpath="{.status.loadBalancer.ingress[0].ip}"
```

4. Connect your local MySQL command-line client to your MySQL instance by running:

```
mysql -u USERNAME -pPASSWORD -h IP_ADDRESS --ssl-mode=VERIFY_CA --ssl-ca=CA-CERTPATH
```

Where:

- `USERNAME` is the username you created above, for example, `bn_wordpress`.
- `PASSWORD` is the password for the user you created above, for example, `hunter2`.
- `IP-ADDRESS` is the external IP address from step 3 above.
- `CA-CERTPATH` is the pathname to the certificate file saved in step 1 above, for example, `./ca.crt`.

For example:

```
$ mysql -u bn_wordpress -phunter2 -h 192.168.64.200 --ssl-mode=VERIFY_CA --ssl-ca=./ca.crt

mysql: [Warning] Using a password on the command line interface can be insecure
.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
...
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Connecting Apps to MySQL Instances

Page last updated:

This topic demonstrates how you, as an app developer, connect your app to a VMware Tanzu™ SQL with MySQL for Kubernetes instance. The instructions show how to create a MySQL database and privileged user and then deploy the Bitnami WordPress Stack Helm chart configured to use your database and user.

Prerequisites

Before connecting an app to a MySQL instance, you must have:

- The Kubernetes Command Line Interface (kubectl) installed. For more information, see the [Kubernetes documentation](#).
- A MySQL instance running in the same Kubernetes cluster as the app. The MySQL instance can be in a different namespace from the app. For information about creating an instance, see [Creating and Deleting MySQL Instances](#).

Note: To avoid Kubernetes permissions issues, VMware Tanzu™ SQL with MySQL for Kubernetes recommends that you grant developers admin access to their target namespace.

Create a Database and Privileged MySQL User for the App

1. Log in to the MySQL server as the root user, following the instructions in [Get Root Access to the MySQL Server](#).

For example, to connect to a single-node instance named “mysql-sample”:

```
$ kubectl exec -it mysql-sample-0 -c mysql -- bash
mysql@mysql-sample-0:/$
```

2. Create any MySQL constructs needed by your app, and create a user with privileges to access those constructs as required by your app. **NOTE: Older MySQL client libraries (such as those base on 5.6) need an older password encryption method. We show both type of CREATE USER commands below.**

For example, the following commands create a `bitnami_wordpress` database and `bn_wordpress` user for the Bitnami WordPress Stack deployment:

```
$ mysql -p$(cat $MYSQL_ROOT_PASSWORD_FILE) -u root
mysql> CREATE DATABASE bitnami_wordpress;
Query OK, 1 row affected (0.20 sec)

mysql> CREATE USER 'bn_wordpress'@'%' IDENTIFIED BY 'hunter2';
Query OK, 0 rows affected (0.08 sec)
--- OR, for broader compatibility with older client libraries ---
mysql> CREATE USER 'bn_wordpress'@'%' IDENTIFIED WITH mysql_native_password BY
'hunter2';
Query OK, 0 rows affected (0.08 sec)

mysql> GRANT ALL PRIVILEGES ON bitnami_wordpress.* TO 'bn_wordpress'@'%';
Query OK, 0 rows affected (0.03 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.10 sec)
```

Configure Your App with MySQL User and Connectivity Information

Application configuration will be specific to the app being deployed. This section shows how to deploy the Helm chart for the Bitnami WordPress Stack using TLS certificates generated by cert-manager, the user and database information created in the previous section, and the cluster-internal

domain name for the MySQL service.

This section assumes that the MySQL instance is named `mysql-sample` and is deployed to the `default` Kubernetes namespace.

1. Install cert-manager if it is not already present in the cluster, following the details in [Prerequisites for Installing via the Tanzu Network Registry](#).
2. Generate a certificate for MySQL to present to WordPress. The certificate must meet the WordPress authentication requirements.

WordPress authenticates signed certificates within the cluster. The following command generates two Issuers: the first Issuer, `selfsigned-issuer`, generates a certificate, `ca-certificate`, for the second Issuer, `tls-issuer`. That second Issuer generates the `tls-certificate` that MySQL will present to WordPress.

```
cat <<EOF | kubectl apply -f -
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned-issuer
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ca-certificate
spec:
  isCA: true
  issuerRef:
    name: selfsigned-issuer
  secretName: ca-secret
  commonName: ca-cert
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: tls-issuer
spec:
  ca:
    secretName: ca-secret
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: tls-certificate
spec:
  isCA: false
  dnsNames:
    - mysql-sample.default # See note after the code excerpt
  secretName: mysql-tls-secret
  issuerRef:
    name: tls-issuer
EOF
```

Note: If you are deploying WordPress in the same Kubernetes cluster as the MySQL instance, specify `INSTANCE-NAME.INSTANCE-NAMESPACE` for `spec.dnsNames` above.

3. Edit the YAML file for the MySQL instance so that the instance presents the `tls-certificate` when receiving a client connection request. Then apply the updated YAML file.

In the MySQL instance YAML file, add the spec property `spec.tls.secret.name`. If you are creating a new MySQL instance, you can edit the original deployment YAML file and then apply that file to create the instance. If you are updating an existing MySQL instance, you can use the `kubectl edit` command to edit the YAML configuration in place. For example:

```
$ kubectl edit mysql.with.sql.tanzu.vmware.com/mysql-sample

spec:
  ...
  tls:
    secret:
      name: mysql-tls-secret
  ...
```

For more information about TLS configuration, see [Configuring TLS for MySQL Instances](#).

You can verify that the deployment was successful by inspecting the startup output:

```
$ kubectl logs pod/mysql-sample-0 -c mysql
Initialization complete, now exiting!
2021-04-06T02:07:02.667789Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.22-13) starting as process 1
2021-04-06T02:07:02.675491Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2021-04-06T02:07:02.907693Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2021-04-06T02:07:03.055031Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock
2021-04-06T02:07:03.157475Z 0 [Warning] [MY-010068] [Server] CA certificate /etc/mysql/certs/ca.pem is self signed.
2021-04-06T02:07:03.161660Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported for this channel.
```

4. Deploy the Bitnami WordPress Stack Helm chart, specifying the information for the constructs created in the preceding steps:

- Database username `bn_wordpress`
- Database user password `hunter2`
- Database name `bitnami_wordpress`
- Secret `mysql-tls-secret` (containing the MySQL signed certificate)

The following commands mount a volume, which contains the CA secret, onto the Bitnami container, which reads the mount point into the `WORDPRESS_DATABASE_SSL_CA_FILE` environment variable:

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

$ helm install wp bitnami/wordpress -f - <<EOF
mariadb:
  enabled: false

externalDatabase:
  host: mysql-sample.default.svc.cluster.local
  user: bn_wordpress
  password: hunter2
  database: bitnami_wordpress

extraEnvVars:
- name: "WORDPRESS_DATABASE_SSL_CA_FILE"
  value: "/etc/mysql/tls/ca.crt"

extraVolumes:
- name: mysql-ca
  secret:
    secretName: "mysql-tls-secret"
    items:
    - key: ca.crt
      path: ca.crt

extraVolumeMounts:
- name: mysql-ca
  mountPath: /etc/mysql/tls/
EOF

NAME: wp
LAST DEPLOYED: Tue Dec 8 14:32:08 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
** Please be patient while the chart is being deployed **
```

After successful installation, Helm will display WordPress instructions for connecting as a user or admin to the running WordPress server. You can also retrieve the WordPress user password by running the following command:


```
$ kubectl get secret wp-wordpress -o go-template='{{index .data "wordpress-pass  
word" | base64decode}}'
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Binding a TAS Application to a MySQL Instance

Page last updated:

This topic demonstrates how to connect an application running on VMware Tanzu Application Service for VMs (TAS) to a VMware Tanzu™ SQL with MySQL for Kubernetes instance.

Prerequisites

Before you can connect an application running on TAS to a Tanzu MySQL for Kubernetes instance, you must have:

- The Kubernetes Command Line Interface (kubectl) installed. For more information, see the [Kubernetes documentation](#).
- A Tanzu MySQL for Kubernetes instance installed on a Kubernetes cluster that supports a form of external ingress, for example a [LoadBalancer](#). For more details see [Creating and Deleting MySQL Instances](#).
- An application that is deployed to TAS and can access the Kubernetes cluster ingress points.

Binding an Application

To bind a MySQL instance to an application running on TAS:

1. Deploy an instance of VMware Tanzu™ SQL with MySQL for Kubernetes that uses a public ingress method, such as [LoadBalancer](#).
2. Enable TLS for the instance, following the instructions in [Configuring TLS for MySQL Instances](#).
3. Create a database and a privileged MySQL user for the application, following the procedure [Create a Database and Privileged MySQL User for the App](#).
4. Set the environment variables for your newly created user and for the load balancer hostname:

```
export HOSTNAME=$(kubectl get service mysql-sample -o 'jsonpath={.status.loadBalancer.ingress[0].ip}')
export USER=bn_wordpress
export PASSWORD=hunter2
export DATABASE=for_my_app
```

5. Create a credentials hash for a TAS user-provided service configuration:

```
cf create-user-provided-service k8s-tanzu-sql -p '{"uri":"mysql://$USER:$PASSWORD@$HOSTNAME:3306/$DATABASE?reconnect=true&sslMode=required&ssl_mode=require d\""}'
```

If autoconnect is supported by the application buildpack, as is common with data services, the application will automatically connect to the database instance. If the buildpack does not support autoconnect, you will need to manually use the [uri](#) service credentials to connect the application to the database instance.

Troubleshooting

This procedure, including autowiring, was validated with a Spring Boot application. Autowiring did not work with a Ruby on Rails application, because the version of the MariaDB client library in the Cloud Foundry [cflinuxfs3](#) stack does not support [sslMode](#).

- If your application cannot make SSL connections, start by properly consuming the connection information components and making a manual connection from the application.
- Some techniques for connecting non-Spring applications are provided in the [Activate TLS for Non-Spring Apps](#) section of the MySQL for VMs documentation. The credentials hash

created in the [Binding an Application](#) procedure above does not include individual components of connection information, such as `username` and `hostname`, but you can derive these components by parsing the `uri` in the hash.

- As a last resort, you can disable TLS on the MySQL instance and remove the `ssl_mode` and `sslMode` parameters from the credentials hash.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Backing Up and Restoring MySQL Instances

This topic describes how to back up and restore VMware Tanzu™ SQL with MySQL for Kubernetes.

Overview

Tanzu MySQL for Kubernetes allows you to generate on-demand backups, configure schedules for automated backups, and restore backups to new MySQL instances.

For uploading and retrieving backup artifacts, Tanzu MySQL for Kubernetes currently supports S3, Minio, and other S3-compatible storage.

For backing up and restoring, Tanzu MySQL for Kubernetes uses four of the five Custom Resource Definitions (CRDs):

- **MySQLBackup:** References a MySQL backup artifact that exists in an external blobstore such as S3 or Minio. A new MySQLBackup resource is created every time an on-demand or scheduled backup is generated.
- **MySQLBackupLocation:** References an external blobstore and credentials necessary to access the blobstore.
- **MySQLBackupSchedule:** Represents a CronJob schedule on which to perform backups.
- **MySQLRestore:** References an instance of a restore that was performed. A new MySQLRestore resource is created every time a restore is performed.

For detailed information about the CRDs, see [Controllers and Custom Resource Definitions \(CRDs\)](#) in *Architecture*.

Note: The procedures in this topic require the VMware Tanzu™ SQL with MySQL for Kubernetes deployment templates. If you have not yet downloaded the deployment templates, see [Download the Deployment Templates](#) in *Creating and Deleting MySQL Instances*.

About Synchronization of Backups with the External Blobstore

Tanzu MySQL for Kubernetes syncs MySQLBackup resources in a Kubernetes cluster with the contents of the external blobstore. The external blobstore is treated as the source of truth. This means that, if a [MySQLBackup](#) resource is deleted on the Kubernetes cluster, but the associated backup artifact still exists in the external blobstore, Tanzu MySQL for Kubernetes re-creates the [MySQLBackup](#) resource to match the contents of the external blobstore.

Back Up Tanzu MySQL for Kubernetes Data

Performing backups for Tanzu MySQL for Kubernetes requires creating a MySQLBackupLocation resource that references an external blobstore. Both on-demand backups and scheduled backups use the MySQLBackupLocation to upload backup artifacts to the external blobstore.

Before starting the procedures for backing up a Tanzu MySQL for Kubernetes instance, ensure that you know the configuration details of your external blobstore and how often you want to perform scheduled backups.

Create a MySQLBackupLocation Resource

The purpose of creating this MySQLBackupLocation Resource is to configure the namespace with the location of the blobstore and the credentials to access it.

To create a MySQLBackupLocation resource:

1. Find the [backuplocation.yaml](#) deployment template that you downloaded in the TGZ file from VMware Tanzu Network. For how to download deployment templates, see [Download the Deployment Templates](#) in *Creating and Deleting MySQL Instances*.
2. Create a copy of the [backuplocation.yaml](#) file and give it a unique name.

For example:

```
$ cp ~/Downloads/tanzu-mysql-deployment-templates-1.0.0/samples/backuplocation.yaml testbackuplocation.yaml
```

3. Edit the file with the configuration for your external blobstore. For an explanation of the properties that you can set in this file, see [Properties for the MySQLBackupLocation Resource](#) and [Properties for the Secret](#).
4. Create the MySQLBackupLocation resource in the same namespace as the MySQL instances that you want to back up by running:

```
kubectl apply -f FILENAME -n DEVELOPMENT-NAMESPACE
```

- Where `DEVELOPMENT-NAMESPACE` is the namespace for the MySQL instance.
- Where `FILENAME` is the name of the configuration file you created in Step 2 above.

For example:

```
$ kubectl apply -f testbackuplocation.yaml -n my-namespace
mysqlbackuplocation.with.sql.tanzu.vmware.com/backuplocation-sample created
secret/backuplocation-sample-creds configured
```

5. Verify that the MySQLBackupLocation has been created by running:

```
kubectl get mysqlbackuplocation backuplocation-sample \
-o jsonpath={.spec} -n DEVELOPMENT-NAMESPACE
```

For example:

```
$ kubectl get mysqlbackuplocation backuplocation-sample -o jsonpath={.spec} -n my-namespace
{
  "storage": {
    "s3": {
      "bucket": "bucket-sample",
      "forcePathStyle": false,
      "region": "us-west-1",
      "secret": {
        "name": "backuplocation-sample-creds"
      }
    }
  }
}
```

Create a MySQLBackupSchedule Resource

To set a schedule for automatic backups, create a MySQLBackupSchedule resource:

1. Find the `backupschedule.yaml` deployment template that you downloaded in the TGZ file from VMware Tanzu Network. For how to download deployment templates, see [Download the Deployment Templates](#) in *Creating and Deleting MySQL Instances*.
2. Create a copy of the `backupschedule.yaml` file and give it a unique name.

For example:

```
$ cp ~/Downloads/tanzu-mysql-deployment-templates-1.0.0/samples/backupschedule.yaml testbackupschedule.yaml
```

3. Edit the file with the name of the MySQLBackupLocation resource that you created in [Create a MySQLBackupLocation Resource](#) and the name of the MySQL instance you want scheduled backups of. For an explanation of the properties that you can set in this file, see [Properties for the MySQLBackupSchedule Resource](#).
4. Create the MySQLBackupSchedule resource in the same namespace as the MySQLBackupLocation and MySQL instance that you referenced in the MySQLBackupSchedule YAML file.

```
kubectl apply -f FILENAME -n DEVELOPMENT-NAMESPACE
```

Where `FILENAME` is the name of the configuration file you created in Step 2 above.

For example:

```
$ kubectl apply -f testbackupschedule.yaml -n my-namespace
mysqlbackupschedule.with.sql.tanzu.vmware.com/backupschedule-sample created
```

5. Verify that the MySQLBackupSchedule has been created by running:

```
kubectl get mysqlbackupschedule mysqlbackupschedule-sample -o jsonpath={.spec}
-n DEVELOPMENT-NAMESPACE
```

For example:

```
$ kubectl get mysqlbackupschedule mysqlbackupschedule-sample -o jsonpath={.spec}
-n my-namespace
{
  "backupTemplate": {
    "spec": {
      "instance": {
        "name": "demo-db"
      },
      "location": {
        "name": "demo-backuplocation"
      }
    }
  },
  "schedule": "@daily"
}
```

If you correctly configured both a [MySQLBackupLocation](#) resource and [MySQLBackupSchedule](#) resource for an existing MySQL instance, you see backups being generated and uploaded to the external blobstore.

Name and Location for Backup Artifacts

MySQLBackup resources that are automatically generated as a result of a MySQLBackupSchedule are named `SCHEDULE-NAME-TIMESTAMP`.

By default, Tanzu MySQL for Kubernetes stores backup artifacts under the subfolder structure `yyyy > mm > dd`. You can configure a custom path for backups so that backup artifacts are stored under the subfolder structure `CUSTOM-PATH > yyyy > mm > dd`.

Backup artifacts stored in the external blobstore are named `DATETIME-RANDOM_STRING-backup.xb`.

For example, if a MySQLBackupSchedule name is `mysqlbackupschedule-sample`, the custom backup path is `my-backups/`, and a backup was taken on Thursday, December 10, 2020 at 8:51:03 PM GMT (timestamp `1607633463`), then:

- The MySQLBackup resource on the Kubernetes cluster is named `mysqlbackupschedule-sample-1607633463`
- The backup artifact in the external blobstore is named `20201210T205103-kzw54l-backup.xb`
- The path to the artifact is `my-backups/2020/12/10/`.

Perform an On-Demand Backup

In addition to scheduled backups, you can take individual backups whenever you want.

Prerequisite: A MySQLBackupLocation resource that represents the external blobstore to which you upload the generated backup artifact. To configure the MySQLBackupLocation resource, see [Create a MySQLBackupLocation Resource](#) above.

To take a backup:

1. Find the `backup.yaml` deployment template that you downloaded in the TGZ file from VMware Tanzu Network. For how to download deployment templates, see [Download the Deployment Templates](#) in *Creating and Deleting MySQL Instances*.
2. Create a copy of the `backup.yaml` file and give it a unique name.

For example:

```
$ cp ~/Downloads/tanzu-mysql-deployment-templates-1.0.0/samples/backup.yaml tes
tbackup.yaml
```

3. Edit the file. For an explanation of the properties that you can set for the MySQLBackup resource, see [Properties for the MySQLBackup Resource](#).
4. Trigger the backup by creating the MySQLBackup resource in the same namespace as the

instance by running:

```
kubecttl apply -f FILENAME -n DEVELOPMENT-NAMESPACE
```

Where `FILENAME` is the name of the configuration file you created in Step 2 above.

For example:

```
$ kubecttl apply -f testbackup.yaml -n my-namespace
mysqlbackup.with.sql.tanzu.vmware.com/backup-sample created
```

5. Verify that a backup has been generated and track its progress by running:

```
kubecttl get mysqlbackup backup-sample -n DEVELOPMENT-NAMESPACE
```

For example:

```
$ kubecttl get mysqlbackup backup-sample -n my-namespace
NAME          STATUS      SOURCE INSTANCE    TIME STARTED          TIME COMPLETED
backup-sample Succeeded  mysql-sample      2020-12-01T21:49:26Z  2020-12-01T21:49:30Z
```

For an explanation of what each column means, see [List Existing MySQLBackup Resources](#) below.

List Existing MySQLBackup Resources

You might want to list existing MySQLBackup resources for various reasons, for example:

- To select a backup to restore. For steps to restore a backup, see [Restore](#).
- To see the last successful backup.
- To verify that scheduled backups are running as expected.
- To find old backups that need to be cleaned up. For steps to delete backups, see [Delete Old Backup Artifacts](#).

To see a list of existing MySQLBackup resources:

1. List existing MySQLBackup resources by running:

```
kubecttl get mysqlbackup
```

For example:

```
$ kubecttl get mysqlbackup
NAME          STATUS      SOURCE INSTANCE    TIME STARTED          TIME COMPLETED
backup-sample Failed      mysql-sample
```

2. To understand the output, see the table below:

Column Name	Meaning
<code>STATUS</code>	Represents the current status of the backup. Allowed values are: <ul style="list-style-type: none"> ◦ Pending: The backup has been received but not scheduled on a MySQL Pod. ◦ Running: The backup is being generated and streamed to the external blobstore. ◦ Succeeded: The backup has completed successfully. ◦ Failed: The backup has failed to complete. To troubleshoot a failed backup, see Troubleshoot Backup and Restore below.
<code>SOURCE INSTANCE</code>	The MySQL instance the backup was taken from.
<code>TIME STARTED</code>	The time that the backup process started.
<code>TIME COMPLETED</code>	The time that the backup process finished. If the backup fails, this value is empty.

Delete Old Backup Artifacts

Tanzu SQL for Kubernetes does not natively support retention policies for backup artifacts. You can configure retention policies on your external blobstore. If you do, you must also delete the associated MySQLBackup resources in the Kubernetes cluster, because those are not automatically

deleted by Tanzu SQL for Kubernetes.

To delete a backup:

1. Delete the backup in the external blobstore.
2. On your Kubernetes cluster, delete the MySQLBackup resource by running:

```
kubectl delete mysqlbackup BACKUP-NAME -n DEVELOPMENT-NAMESPACE
```

For example:

```
kubectl delete mysqlbackup backup-sample -n my-namespace
```

Restore

This section discusses two kinds of restore:

- [Restore from a Backup](#)
- [Restore a Backup to a Different Namespace or Kubernetes Cluster](#)

Restore from a Backup

MySQLRestores always restores to a new MySQL instance to avoid overwriting any data on an existing MySQL instance. When the restore is triggered, it automatically creates the new MySQL instance.

Tanzu MySQL for Kubernetes does not allow you to restore a backup to an existing MySQL instance. Although you can perform this manually by copying the MySQL data from the backup artifact onto an existing MySQL instance, VMware strongly discourages you from doing this because you might overwrite existing data on the MySQL instance.

Prerequisites

Before you restore from a backup, you must have:

- An existing MySQLBackup in your current namespace. To select an existing MySQLBackup to restore, see [List Existing MySQLBackup Resources](#).
- A MySQLBackupLocation that represents the bucket where the existing backup artifact is stored. See [Create a MySQLBackupLocation Resource](#) above.

Procedure

To restore from a backup:

1. Find the `restore.yaml` deployment template that you downloaded in the TGZ file from VMware Tanzu Network. For how to download deployment templates, see [Download the Deployment Templates](#) in *Creating and Deleting MySQL Instances*.
2. Create a copy of the `restore.yaml` file and give it a unique name.

For example:

```
$ cp ~/Downloads/tanzu-mysql-deployment-templates-1.0.0/samples/restore.yaml testrestore.yaml
```

3. Edit the file. For information about the properties that you can set for the MySQLRestore resource, see [Property Reference for Backup and Restore](#).

To restore from an HA instance to a non-HA instance, edit the `testrestore.yaml` and amend the `spec.instanceTemplate.spec.highAvailability` field to `false`. For example:

```
---
apiVersion: with.sql.tanzu.vmware.com/v1
kind: MySQLRestore
metadata:
  name: restore-sample
spec:
  backup:
    name: backup-sample
  instanceTemplate:
    metadata:
      name: mysql-sample
    spec:
```



```
storageSize: 1Gi
imagePullSecretName: tanzu-mysql-image-registry
highAvailability:
  enabled: false
```

- Trigger the restore by creating the MySQLRestore resource in the same namespace as the MySQLBackup and MySQLBackupLocation by running:

```
kubectl apply -f FILENAME -n DEVELOPMENT-NAMESPACE
```

Where **FILENAME** is the name of the configuration file you created in Step 2 above.

For example:

```
$ kubectl apply -f testrestore.yaml -n my-namespace
mysqlrestore.with.sql.tanzu.vmware.com/restore-sample created
```

- Verify that a restore has been triggered and track the progress of your restore by running:

```
kubectl get mysqlrestore restore-sample -n DEVELOPMENT-NAMESPACE
```

For example:

```
$ kubectl get mysqlrestore restore-sample -n my-namespace
NAME              STATUS      SOURCE BACKUP  TARGET INSTANCE  TIME STARTED
TIME COMPLETED
restore-sample    Succeeded   backup-sample  mysql-sample     2020-12-01T21:52
:30Z             2020-12-01T21:53:09Z
```

- To understand the output, see the table below:

Column Name	Meaning
STATUS	Represents the current status of the restore process. Allowed values are: <ul style="list-style-type: none"> Pending: The restore has been received but not yet scheduled on a MySQL Pod. Running: The restore is in progress. Succeeded: The restore has completed successfully. Failed: The restore failed. To troubleshoot, see Troubleshoot Backup and Restore below.
SOURCE BACKUP	The name of the backup being restored.
TARGET INSTANCE	The name of the new MySQL instance to be restored with the backup contents.
TIME STARTED	The time that the restore process started.
TIME COMPLETED	The time that the restore process finished. If the restore fails, this value is empty.

Restoring a Backup to a Different Namespace or Kubernetes Cluster

If you want to restore a backup to a different namespace or a different Kubernetes cluster, create a **MySQLBackupLocation** in the target namespace or Kubernetes cluster. Then, Tanzu MySQL for Kubernetes automatically creates **MySQLBackup** resources for the backup artifacts in the external blobstore.

To restore to a different namespace or Kubernetes cluster, you create a BackupLocation in the target namespace:

- Target the destination Kubernetes cluster or namespace.
- Create a MySQLBackupLocation resource that contains the backup artifact to restore. For how to do this, see [Create a MySQLBackupLocation Resource](#).
- Confirm that the MySQLBackup artifact to restore is included in the list by running:

```
kubectl get mysqlbackup
```

For example:

```
$ kubectl get mysqlbackup
NAME              STATUS      SOURCE INSTANCE  TIME STARTED          TIME COMP
LETED
sample-backup    Succeeded   mysql-sample     2020-12-01T21:49:26Z  2020-12-0
1T21:49:30Z
```

4. Trigger a restore by following steps in [Restore from a Backup](#).

Troubleshoot Backup and Restore

Basic troubleshooting begins with reviewing the status for the resource and reading the messages associated with the resource events.

To troubleshoot problems with backup and restore:

1. Detect issues by monitoring the `STATUS` column of any MySQL custom resource. If the status is `Failed` or is stuck in `Pending`, `Scheduled`, or `Running`, then one of the following might be the problem:
 - Misconfiguration
 - Problem with the external blobstore
 - Issues with the MySQL Operator

In this example, the `kubectl get` command outputs a `Failed` status:

```
$ kubectl get mysqlbackup backup-sample
NAME          STATUS    SOURCE INSTANCE    TIME STARTED    TIME COMPLETED
backup-sample  Failed    mysql-sample
```

2. Diagnose the issue by inspecting the Kubernetes events for the resource, for example:

```
$ kubectl get events --field-selector involvedObject.name=backup-sample
LAST SEEN   TYPE      REASON   OBJECT                       MESSAGE
2m43s       Warning   Failed   mysqlbackup/backup-sample    Secret "backuplocation-sample-creds" not found
```

3. Read the message in the `MESSAGE` column to understand why the failure occurred.

In the example above, the `backup-sample` expected a Kubernetes secret called `backuplocation-sample-creds` to exist. Fix this problem by creating the `backuplocation-sample-creds` secret. The template for this secret is located in the `backuplocation.yaml` template.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Property Reference for the MySQL Resource

Page last updated:

This topic contains a property reference table for the MySQL resource. Refer to this table when you are creating the YAML file for a MySQL instance. For more information, see [Create a Mysql Instance](#) in *Creating and Deleting a MySQL Instance*.

The table below explains the properties that you can set in your MySQL resource.

Property	Information	Example
<code>metadata.name</code>	<p>The name of the MySQL instance. Must be unique within a namespace. Cannot be modified after the MySQL instance has been created. For more information, see the Kubernetes documentation.</p> <p>Type: String Required Default: <i>n/a</i></p>	<code>my-instance</code>
<code>spec.storageSize</code>	<p>The size of the persistent volume claims (PVCs) for MySQL instance Pods. After being set, cannot be reduced, but can be increased. For information about allowed size units, see discussion about resource quantities in the kubernetes / community GitHub repository.</p> <p>Type: Resource quantity. Required Default: <i>n/a</i></p>	<code>50Gi</code>
<code>spec.imagePullSecretName</code>	<p>An existing Kubernetes docker-registry secret that can access the registry containing the MySQL image.</p> <p>Type: String Required Default: <i>n/a</i></p>	<code>tanzu-mysql-image-registry</code>
<code>spec.storageClassName</code>	<p>The StorageClass used for PVCs associated with MySQL instance Pods. Cannot be modified after the MySQL instance has been created. For available types, contact your Kubernetes admin. For information about the StorageClass concept, see the Kubernetes documentation.</p> <p>Type: String Optional Default: Standard</p>	<code>standard</code>
<code>spec.serviceType</code>	<p>The type of Service used to provide access to the MySQL database. Only <code>ClusterIP</code> and <code>LoadBalancer</code> are supported. For more information about Kubernetes ServiceTypes, see the Kubernetes documentation.</p> <p>Type: String Optional Default: ClusterIP</p>	<code>LoadBalancer</code>
<code>spec.version</code>	<p>The version of the MySQL instance.</p> <p>Type: String Optional Default: The value of <code>defaultInstanceVersion</code> set by the service administrator in the Tanzu MySQL Helm chart</p>	<code>1.1.0</code>

<code>spec.tls.secret.name</code>	The name of the Kubernetes secret containing the TLS certificate and private key used to support encrypted connections by clients. Type: String Optional Default: <i>n/a</i>	<code>mysql-tls-secret</code>
<code>spec.highAvailability.enabled</code>	Whether the instance is a high-availability cluster (not single-node). After being set to <code>true</code> , cannot be modified. For information about high availability, see Configuring MySQL Instances for High Availability . Type: Boolean Optional Default: False	<code>highAvailability.enabled:</code> <code>false</code>
<code>spec.resources.mysql</code>	The maximum CPU and memory allowed for the MySQL container. If not specified, Kubernetes does its best effort to allocate necessary compute resources for this container. For more information about managing resources for containers, see the Kubernetes documentation . Type: <code>limits.cpu</code> , <code>limits.memory</code> Optional Default: Best effort	<code>limits.cpu: 3 ,</code> <code>limits.memory: 800Mi</code>
<code>spec.resources.mysql</code>	The minimum CPU and memory allowed for the MySQL container. If not specified, defaults to limits, if limits have been explicitly specified. For more information about limits, see the Kubernetes documentation . Type: <code>requests.cpu</code> , <code>requests.memory</code> Optional Default: Best effort	<code>requests.cpu: 2 ,</code> <code>requests.memory: 500Mi</code>
<code>spec.resources.mysqlSidecar</code>	The maximum CPU and memory allowed for the mysql-sidecar container. If not specified, Kubernetes does its best effort to allocate the necessary compute resources for this container. For more information about CPU and memory resources, see the Kubernetes documentation . Type: <code>limits.cpu</code> , <code>limits.memory</code> Optional Default: Best effort	<code>limits.cpu: 2 ,</code> <code>limits.memory: 500Mi</code>
<code>spec.resources.mysqlSidecar</code>	Describes the minimum CPU and memory allowed for the mysql-sidecar container. If left blank, defaults to limits, if limits have been explicitly specified. For more information about limits, see the Kubernetes documentation . Type: <code>requests.cpu</code> , <code>requests.memory</code> Optional Default: Best effort	<code>requests.cpu: 1 ,</code> <code>requests.memory: 200Mi</code>
<code>spec.resources.proxy</code>	Describes the maximum CPU and memory allowed for the proxy container. This container is only created if <code>spec.highAvailability.enabled</code> is <code>true</code> . If not specified, Kubernetes does its best effort to allocate the necessary compute resources for the proxy container. For more information about CPU and memory resources, see the Kubernetes documentation . Type: <code>limits.cpu</code> , <code>limits.memory</code> Optional Default: Best effort	<code>limits.cpu: 1000m,</code> <code>limits.memory: 256Mi</code>

<code>spec.resources.proxy</code>	Describes the minimum CPU and memory allowed for the proxy container. This container is only created if <code>spec.highAvailability.enabled</code> is <code>true</code> . If not specified, defaults to limits, if limits have been explicitly specified. For more information about limits, see the Kubernetes documentation .	<code>requests.cpu: 200m;</code> <code>requests.memory: 48Mi</code>
	Type: requests.cpu, requests.memory Optional Default: Best effort	
<code>spec.resources.metrics</code>	Describes the minimum CPU and memory allowed for the metrics container 'mysql-exporter'. This container is created during MySQL instance initialization. If not specified, defaults to limits, if limits have been explicitly specified.	<code>requests.cpu 100m,</code> <code>requests.memory: 32Mi</code>
	Type: requests.cpu, requests.memory Optional Default: Best effort	
<code>spec.resources.metrics</code>	Describes the limits of CPU and memory for the metrics container 'mysql-exporter'. This container is created during MySQL instance initialization.	<code>limits.cpu: 250m,</code> <code>limits.memory: 75Mi</code>
	Type: limits.cpu, limits.memory Optional Default: Best effort	

Create a pull request or raise an issue on the source for this page in [GitHub](#)

Property Reference for Backup and Restore

In this topic

[Properties for the MySQLBackupLocation Resource](#)

[Properties for the Secret](#)

[Properties for the MySQLBackupSchedule Resource](#)

[Properties for the MySQLBackup Resource](#)

[Properties for the MySQLRestore Resource](#)

Page last updated:

This topic contains property reference tables for the resources that you create for performing backup and restore. For information about backup and restore, see [Backing Up and Restoring MySQL Instances](#).

Properties for the MySQLBackupLocation Resource

The table below explains the properties that can be set for the MySQLBackupLocation resource.

Property	Type	Default	Description	Example	Req?
metadata.name	String	<i>n/a</i>	The name of the MySQLBackupLocation. Must be unique within a namespace.	<code>backuplocation-sample</code>	Yes
spec.storage.s3.bucket	String	<i>n/a</i>	The name of an existing S3-compatible bucket for this backup location.	<code>s3-bucket-sample</code>	Yes
spec.storage.s3.bucketPath	String	<i>n/a</i>	The name of the path where backup artifacts will be uploaded. If a folder in the path does not already exist, it is created automatically. The trailing slash in the path is required.	<code>s3-sample-path/sample-subpath/</code>	No
spec.storage.s3.region	String	us-east-1	The geographic region of the bucket. Some non-AWS S3 implementations do not require this value.	<code>us-west-1</code>	No
spec.storage.s3.endpoint	String	<code>https:// BUCKET-NAME .s3.REGION .amazonaws.com/</code>	The endpoint URL for the configured S3-compatible provider. Leave blank for AWS S3.	<code>http://minio.default:9000</code>	No
spec.storage.s3.forcePathStyle	boolean	false	<code>true</code> forces the use of path-style S3 URLs for compatibility. May be required for some non-AWS S3 providers. <code>false</code> uses virtual hosted-style S3 URLs. For information about AWS S3 Path Deprecation, see the AWS blog site .	<code>false</code>	No
spec.storage.s3.secret.name	String	<i>n/a</i>	The name of the Kubernetes secret that contains the credentials for connecting to S3.	<code>backuplocation-sample-creds</code>	Yes

Properties for the Secret

The table below explains the properties that can be set in the `secret` for the MySQLBackupLocation resource.

Property	Type	Default	Description	Example	Req?
metadata.name	String	<i>n/a</i>	The name of the Secret. Must match <code>spec.storage.s3.secret.name</code> in a BackupLocation. Must be unique within a namespace.	<code>backuplocation-sample-creds</code>	Yes
stringData.accessKeyId	String	<i>n/a</i>	The Access Key ID for an AWS IAM user that has permissions to read/write from the S3 bucket.	<code>AKIAIOSFODNN7EXAMPLE</code>	Yes
stringData.secretAccessKey	String	<i>n/a</i>	The Secret Access Key for an AWS IAM user that has permissions to read/write from the S3 bucket	<code>wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY</code>	Yes

Properties for the MySQLBackupSchedule Resource

The table below explains the properties that can be set for the MySQLBackupSchedule resource.

Property	Type	Default	Description	Example	Req?
metadata.name	String	<i>n/a</i>	The name of the MySQLBackupSchedule. Must be unique within a namespace.	<code>backupschedule-sample</code>	Yes

spec.backupTemplate.spec.location.name	String	<i>n/a</i>	The name of the MySQLBackupLocation that represents the blobstore where backups will be uploaded. Must be in the same namespace as the MySQLBackupSchedule.	<code>backuplocation-sample</code>	Yes
spec.backupTemplate.spec.instance.name	String	<i>n/a</i>	The name of the MySQL instance on which you want scheduled backups for. Must be in the same namespace as the MySQLBackupSchedule.	<code>mysql-sample</code>	Yes
spec.schedule	String (cron schedule)	<i>n/a</i>	The cron schedule for backups. Must be a valid cron schedule.	<code>"0 23 * * 6" (every Saturday at 11PM)</code>	Yes

Properties for the MySQLBackup Resource

The table below explains the properties that can be set for the MySQLBackup resource.

Property	Type	Default	Description	Example	Req?
metadata.name	String	<i>n/a</i>	The name of the MySQLBackup. Must be unique within a namespace.	<code>backup-sample</code>	Yes
spec.location.name	String	<i>n/a</i>	The name of the MySQLBackupLocation that represents the blobstore where the backup will be uploaded. Must be in the same namespace as the MySQL instance.	<code>backuplocation-sample</code>	Yes
spec.instance.name	String	<i>n/a</i>	The name of the MySQL instance on which you want to perform the on-demand backup.	<code>my-instance</code>	Yes

Properties for the MySQLRestore Resource

The table below explains the properties that can be set for the MySQLRestore resource.

Property	Type	Default	Description	Example	Req?
metadata.name	String	<i>n/a</i>	The name of the MySQLRestore. Must be unique within a namespace.	<code>restore-sample</code>	Yes
spec.backup.name	String	<i>n/a</i>	The name of the MySQLBackup that represents the backup artifact to restore. Must be in the same namespace as the MySQLRestore.	<code>backup-sample</code>	Yes
spec.instanceTemplate	MySQL	<i>n/a</i>	The configuration for the MySQL instance that the backup artifact will be restored to. This MySQL instance is created automatically as part of the restore, so the name must not already exist in the namespace. For descriptions of each value in the configuration template, see Create a MySQL Instance .	See example below.*	Yes

* The type MySQL is a nested YAML object. For example:

```
metadata:
  name: mysql-sample
spec:
  storageSize: 1Gi
  imagePullSecret: tanzu-mysql-image-registry
  highAvailability:
    enabled: true
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)