## Overview

The goal of this lab is to deploy the epitome of a stateful service, a database. We will deploy a MySQL cluster with a master node and many read slaves. This will present some unique challenges, the master node must have a consistent hostname (mysql-0 in our case) and it's own persistent data storage. Read slaves cannot start up before the master is ready and must also have their own storage for their own independent copy of the data. Also, our data is super important so we must take frequent backups.

This lab was inspired by the docs. <https://kubernetes.io/docs/tasks/run-application/run-replicated-stateful-application/>

[Overview](#_h81by6i7gprf)

[Instructions](#_cwxa365h7j6n)

[Task 1: Setup our MySQL database](#_z6ht5gjfpopt)

[Step 1: Define a ConfigMap](#_z9ad9pzgs0f1)

[Step 2: Define Services](#_6c1hj16pcmn7)

[Step 3: Define our StatefulSet](#_dxm1e51iah1q)

[Step 4: Create our objects](#_46f75bjcpcsa)

[Step 5: Watch the magic](#_pjjkkyfa2lni)

[Step 6: Create a database and some data](#_kii3q9u3hehg)

[Step 7: Test our reads](#_ifjthmx45xqb)

[Task 2: Scale our mysql cluster](#_mfwirjp9ij8b)

[Step 1: Lets scale up](#_4x6diu8c61)

[Step 2: Scale back down](#_jwt9dbgc76bs)

[Task 3: Schedule Backups](#_66mj27h76r2d)

[Step 1: Create a volume for backups](#_5p7xrdzgtnqx)

[Step 2: Create a CronJob](#_45p61w2cxatq)

[Step 3: Witness a backup](#_z85e6fhg64co)

[Task 4: Cleanup](#_3rra8f1f96jg)

## 

## Instructions

Read this lab like a book, all text is there for a reason!

"→" denotes an action you must take

Use your favorite editor to edit files within the console. I suggest VI, nano, or emacs.

|  |
| --- |
| White boxes with black text denote commands and file contents |

|  |
| --- |
| Black boxes with green text denote example output |

### **Task 1: Setup our MySQL database**

#### **Step 1: Define a ConfigMap**

Our MySQL database requires a configuration file in order to run. In addition, the master instance requires different configuration than the slaves. The following will create a ConfigMap named 'mysql' that we can use to pass the config files down to our pods.

→ Make a new directory for us to work in:

|  |
| --- |
| mkdir mysql  cd mysql |

→ Create a file named ***cm.yaml*** with the following contents:

|  |
| --- |
| apiVersion: v1  kind: ConfigMap  metadata:  name: mysql  labels:  app: mysql  data:  master.cnf: |  # Apply this config only on the master.  [mysqld]  log-bin  slave.cnf: |  # Apply this config only on slaves.  [mysqld]  read-only |

#### **Step 2: Define Services**

Here we will define two services, "mysql" and "mysql-read". The former is a '[headless service](https://kubernetes.io/docs/concepts/services-networking/service/#headless-services)', meaning it doesn't create an endpoint or act as a load balancer, it does however, give all of our pods a consistent DNS hostname we can use (in the format <pod>.<service>). The "mysql-read" service will give us a way to spread the read queries across our slaves.

→ Create a file named **services.yaml** with the following contents, make sure to read through what you copy and paste:

|  |
| --- |
| # Headless service for stable DNS entries of StatefulSet members.  apiVersion: v1  kind: Service  metadata:  name: mysql  labels:  app: mysql  spec:  ports:  - name: mysql  port: 3306  clusterIP: None  selector:  app: mysql  ---  # Client service for connecting to any MySQL instance for reads.  # For writes, you must instead connect to the master: mysql-0.mysql.  apiVersion: v1  kind: Service  metadata:  name: mysql-read  labels:  app: mysql  spec:  ports:  - name: mysql  port: 3306  selector:  app: mysql |

#### **Step 3: Define our StatefulSet**

Our StatefulSet object is a doozy and consists of a few parts and a couple of new concepts.

**Init containers** run first in the pod and must complete before the application containers are started. This can be very useful for any tasks that need to be performed before an application can run.

Our init container simply copies the correct config file depending on if the Pod is for a master or slave. It also uses the mysql image but overrides the command that will run with a bash script, saving us from having to build a custom image just for this init script.

Example Init container section:

|  |
| --- |
| initContainers:  - name: init-mysql  image: mysql:5.7  command:  - bash  - "-c"  - |  set -ex  # This script simply checks if the host name is mysql-0 or not  # if it is, then the mysql master config file is copied,  # otherwise the slave config is copied  # Generate mysql server-id from pod ordinal index.  [[ `hostname` =~ -([0-9]+)$ ]] || exit 1  ordinal=${BASH\_REMATCH[1]}  echo [mysqld] > /mnt/conf.d/server-id.cnf  # Add an offset to avoid reserved server-id=0 value.  echo server-id=$((100 + $ordinal)) >> /mnt/conf.d/server-id.cnf  # Copy appropriate conf.d files from config-map to emptyDir.  if [[ $ordinal -eq 0 ]]; then  cp /mnt/config-map/master.cnf /mnt/conf.d/  else  cp /mnt/config-map/slave.cnf /mnt/conf.d/  fi  volumeMounts:  - name: conf  mountPath: /mnt/conf.d  - name: config-map  mountPath: /mnt/config-map |

**volumeClaimTemplates** allow us to create a brand new PVC for every Pod created in the StatefulSet. In our case, each Pod gets its own 10Gi volume.

Example volumeClaimTemplates section:

|  |
| --- |
| volumeClaimTemplates:  - metadata:  name: data  spec:  accessModes: ["ReadWriteOnce"]  resources:  requests:  storage: 10Gi |

Finally, our Pods will consist of two containers each, *mysql* and *mysql-replicator*, one running the mysql server and another that initializes replication from the master to the slave when the server is ready.

→ Create a file named **statefulset.yaml** with the following contents:

|  |
| --- |
| apiVersion: apps/v1  kind: StatefulSet  metadata:  name: mysql  spec:  selector:  matchLabels:  app: mysql  serviceName: mysql  replicas: 2  template:  metadata:  labels:  app: mysql  spec:  initContainers:  - name: init-mysql  image: mysql:5.7  command:  - bash  - "-c"  - |  set -ex  # Generate mysql server-id from pod ordinal index.  [[ `hostname` =~ -([0-9]+)$ ]] || exit 1  ordinal=${BASH\_REMATCH[1]}  echo [mysqld] > /mnt/conf.d/server-id.cnf  # Add an offset to avoid reserved server-id=0 value.  echo server-id=$((100 + $ordinal)) >> /mnt/conf.d/server-id.cnf  # Copy appropriate conf.d files from config-map to emptyDir.  if [[ $ordinal -eq 0 ]]; then  cp /mnt/config-map/master.cnf /mnt/conf.d/  else  cp /mnt/config-map/slave.cnf /mnt/conf.d/  fi  volumeMounts:  - name: conf  mountPath: /mnt/conf.d  - name: config-map  mountPath: /mnt/config-map  containers:  - name: mysql  image: mysql:5.7  env:  - name: MYSQL\_ALLOW\_EMPTY\_PASSWORD  value: "1"  ports:  - name: mysql  containerPort: 3306  volumeMounts:  - name: data  mountPath: /var/lib/mysql  subPath: mysql  - name: conf  mountPath: /etc/mysql/conf.d  livenessProbe:  exec:  command: ["mysqladmin", "ping"]  initialDelaySeconds: 30  periodSeconds: 10  timeoutSeconds: 5  readinessProbe:  exec:  # Check we can execute queries over TCP.  command: ["mysql", "-h", "127.0.0.1", "-e", "SELECT 1"]  initialDelaySeconds: 5  periodSeconds: 2  timeoutSeconds: 1  - name: mysql-replicator  image: mysql:5.7  command:  - bash  - "-c"  - |  set -ex  [[ `hostname` =~ -([0-9]+)$ ]] || exit 1  ordinal=${BASH\_REMATCH[1]}  if [[ $ordinal -ne 0 ]]; then  echo "Waiting for mysqld to be ready (accepting connections)"  until mysql -h 127.0.0.1 -e "SELECT 1"; do sleep 1; done  echo "  STOP SLAVE;  CHANGE MASTER TO MASTER\_HOST='mysql-0.mysql',  MASTER\_USER='root',  MASTER\_PASSWORD='',  MASTER\_CONNECT\_RETRY=10;  START SLAVE;  GRANT ALL ON \*.\* TO 'root'@'%' WITH GRANT OPTION;  FLUSH PRIVILEGES;" > /tmp/replicate.sql  mysql -h 127.0.0.1 < /tmp/replicate.sql  fi  until false; do sleep 1; done  volumes:  - name: conf  emptyDir: {}  - name: config-map  configMap:  name: mysql  volumeClaimTemplates:  - metadata:  name: data  spec:  accessModes: ["ReadWriteOnce"]  resources:  requests:  storage: 10Gi |

→ Take a minute to look at the file we just created. Notice the "initContainers", "volumes", "volumeMounts", and "volumeClaimTemplates" sections. Also notice that for a couple of our containers, we are overriding the command to execute instead of relying on what is defined in it's image.

#### **Step 4: Create our objects**

→ Run the following to apply all of our object manifests in the current directory.

|  |
| --- |
| kubectl apply -f . |

|  |
| --- |
| configmap/mysql created  service/mysql created  service/mysql-read created  statefulset.apps/mysql created |

#### **Step 5: Watch the magic**

→ List Pods:

|  |
| --- |
| kubectl get pods |

|  |
| --- |
| NAME READY STATUS RESTARTS AGE  mysql-0 2/2 Running 0 38s  mysql-1 0/2 Init:0/1 0 3s |

If you were fast enough, you should see that mysql-0 is starting up first. Once that completes, mysql-1 will start. This is the behavior of a StatefulSet, each replica will be created in order, one-by-one.

Once both of the pods are ready, let's test it.

#### **Step 6: Create a database and some data**

We can easily execute SQL statements from within temporary pods.

→ Lets run some SQL with the following:

|  |
| --- |
| kubectl run mysql-client --image=mysql:5.7 -i --rm --restart=Never --\  mysql -h mysql-0.mysql <<EOF  CREATE DATABASE test;  CREATE TABLE test.messages (message VARCHAR(250));  INSERT INTO test.messages VALUES ('hello');  EOF |

|  |
| --- |
| If you don't see a command prompt, try pressing enter.  pod "mysql-client" deleted |

This command created a Pod that only lived long enough to run our command, then was terminated. The **'--rm'** told kubernetes to clean up all objects created as a result of the command. The **'--restart=Never**' told k8s not to make a deployment object, just a simple Pod. Everything after the '--' will override the containers command.

This can be really convenient, notice that we didn't need to install a mysql client on our local machine or anything, we just ran it in a container that already has it installed.

#### **Step 7: Test our reads**

→ Let's read our data:

|  |
| --- |
| kubectl run mysql-client --image=mysql:5.7 -i -t --rm --restart=Never --\  mysql -h mysql-read -e "SELECT \* FROM test.messages" |

|  |
| --- |
| If you don't see a command prompt, try pressing enter.  Error attaching, falling back to logs: unable to upgrade connection: container mysql-client not found in pod mysql-client\_student2  +---------+  | message |  +---------+  | hello |  +---------+  pod "mysql-client" deleted |

This query may have been served by either of our Pods, as the mysql-read service should be balancing requests between them.

→ Run the following command to query the responding servers ID 5 times:

|  |
| --- |
| kubectl run mysql-client-loop --image=mysql:5.7 -i -t --rm --restart=Never -- bash -ic "for i in {1..5}; do mysql -h mysql-read -e 'SELECT @@server\_id,NOW()'; sleep 1; done" |

|  |
| --- |
| If you don't see a command prompt, try pressing enter.  +-------------+---------------------+  | @@server\_id | NOW() |  +-------------+---------------------+  | 100 | 2019-06-03 00:59:25 |  +-------------+---------------------+  +-------------+---------------------+  | @@server\_id | NOW() |  +-------------+---------------------+  | 101 | 2019-06-03 00:59:26 |  +-------------+---------------------+  +-------------+---------------------+  | @@server\_id | NOW() |  +-------------+---------------------+  | 101 | 2019-06-03 00:59:27 |  +-------------+---------------------+  +-------------+---------------------+  | @@server\_id | NOW() |  +-------------+---------------------+  | 100 | 2019-06-03 00:59:28 |  +-------------+---------------------+  pod "mysql-client-loop" deleted |

You should see output that alternates between our two mysql pods, randomly distributing the load on our database.

### **Task 2: Scale our mysql cluster**

#### **Step 1: Lets scale up**

Update replicas from 2 to 3 in our statefulset.yaml

→ edit statefulset.yaml

|  |
| --- |
| ..  **replicas: 3**  .. |

→ Apply changes:

|  |
| --- |
| kubectl apply -f . |

|  |
| --- |
| configmap/mysql unchanged  service/mysql unchanged  service/mysql-read unchanged  statefulset.apps/mysql configured |

→ List Pods:

|  |
| --- |
| kubectl get pods |

|  |
| --- |
| NAME READY STATUS RESTARTS AGE  mysql-0 2/2 Running 0 8m6s  mysql-1 2/2 Running 0 7m31s  mysql-2 0/2 Init:0/1 0 3s |

You should see a new pod, *mysql-2*, getting created. It's that easy to add read capacity to our mysql cluster.

→ List PVCs:

|  |
| --- |
| kubectl get pvc |

|  |
| --- |
| NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE  data-mysql-0 Bound pvc-45f5d9e2-859a-11e9-ace5-42010a8e0020 10Gi RWO standard 9m14s  data-mysql-1 Bound pvc-5aecfa0b-859a-11e9-ace5-42010a8e0020 10Gi RWO standard 8m39s  data-mysql-2 Bound pvc-0daa2897-859b-11e9-ace5-42010a8e0020 10Gi RWO standard 3m39s |

You should also see a 3rd PVC was created for *mysql-2* to use.

#### **Step 2: Scale back down**

→ edit statefulset.yaml

|  |
| --- |
| ..  replicas: 2  .. |

→ Apply changes:

|  |
| --- |
| kubectl apply -f . |

|  |
| --- |
| configmap/mysql unchanged  service/mysql unchanged  service/mysql-read unchanged  statefulset.apps/mysql configured |

→ List Pods:

|  |
| --- |
| kubectl get pods |

|  |
| --- |
| NAME READY STATUS RESTARTS AGE  mysql-0 2/2 Running 0 10m  mysql-1 2/2 Running 0 10m  mysql-2 2/2 Terminating 0 5m1s |

You should now see *mysql-2* getting deleted. The StatefulSet will remove pods in reverse order when scaling down.

→ List PVCs:

|  |
| --- |
| kubectl get pvc |

|  |
| --- |
| NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE  data-mysql-0 Bound pvc-45f5d9e2-859a-11e9-ace5-42010a8e0020 10Gi RWO standard 10m  data-mysql-1 Bound pvc-5aecfa0b-859a-11e9-ace5-42010a8e0020 10Gi RWO standard 10m  data-mysql-2 Bound pvc-0daa2897-859b-11e9-ace5-42010a8e0020 10Gi RWO standard 5m16s |

You may notice that the new PVC is not deleted. StatefulSets that create PVCs require you to manually delete the PVCs, this is to avoid the chance of data loss or if you know you may need to scale back up, you can keep these around and if *mysql-2* is created again, it will be assigned this exact same PVC (meaning it will have less data it needs replicated to it to sync up with our mysql master).

### **Task 3: Schedule Backups**

It would be a good idea for us to take periodic backups of our database in case we delete or corrupt data somehow so we can restore it.

#### **Step 1: Create a volume for backups**

First thing, we need somewhere to store our backups. Lets create a PV where we can put this data.

→ Create **backup-pvc.yaml** with the following contents:

|  |
| --- |
| apiVersion: v1  kind: PersistentVolumeClaim  metadata:  name: backup-data  spec:  accessModes: ["ReadWriteOnce"]  resources:  requests:  storage: 10Gi |

→ Then apply it:

|  |
| --- |
| kubectl apply -f backup-pvc.yaml |

|  |
| --- |
| persistentvolumeclaim/backup-data created |

#### **Step 2: Create a CronJob**

Now let's test your mettle by creating a CronJob object that will create a Job object every 5 minutes to execute our backup.

Tips:

1. Use the image mysql:5.7

2. Use the following commands to take a backup

|  |
| --- |
| command:  - /bin/sh  - -c  - |  set -ex  export TIMESTAMP=$(date +%Y\_%M\_%d\_%T)  echo "Taking database backup $TIMESTAMP"  mysqldump --all-databases -uroot -h mysql-read > /backups/$TIMESTAMP |

3. Mount your PVC "backup-data" to /backups in the pod, [helpful docs](https://kubernetes.io/docs/concepts/storage/persistent-volumes/#claims-as-volumes)

4. Cron syntax for every 5 minutes is represented as "\*/5 \* \* \* \*"

5. Make use of [kubernetes.io](https://kubernetes.io/) and 'kubectl explain cronjob.spec.jobTemplate.spec'

→ Fill in **backup-cronjob.yaml** with your CronJob definition.

|  |
| --- |
| apiVersion: batch/v1beta1  kind: CronJob  metadata:  name: mysql-backup  spec:  jobTemplate:  spec: # Same as Job.spec  # TODO: Fill in job spec |

→ Then apply it:

|  |
| --- |
| kubectl apply -f backup-cronjob.yaml |

#### **Step 3: Witness a backup**

→ Take a look at our CronJob object. You can see under

|  |
| --- |
| kubectl describe cronjob mysql-backup |

If the job has been scheduled, you will see output at the bottom similar to:

|  |
| --- |
| ...  Last Schedule Time: Fri, 03 May 2019 16:50:00 +0000  Active Jobs: <none>  Events:  Type Reason Age From Message  ---- ------ ---- ---- -------  Normal SuccessfulCreate 18m (x280 over 23h) cronjob-controller (combined from similar events): Created job mysql-backup-1556901300  Normal SawCompletedJob 3m43s (x282 over 23h) cronjob-controller (combined from similar events): Saw completed job: mysql-backup-1556902200 |

Rerun the describe command until you see a job has been run.

→ Once it has run, list the jobs:

|  |
| --- |
| kubectl get jobs |

|  |
| --- |
| NAME COMPLETIONS DURATION AGE  mysql-backup-1559525700 0/1 16s 16s |

→ Pick one and view it's details:

|  |
| --- |
| kubectl describe job/<JOB\_NAME> |

→ Now view it's logs:

|  |
| --- |
| kubectl logs job/<JOB\_NAME> |

Was it successful? Are there any errors in it's logs if it failed?

### **Task 4: Cleanup**

→ Delete all of the objects in our namespace:

|  |
| --- |
| kubectl delete all --all  kubectl delete pvc --all |