Deploying a Stateful Application in the Kubernetes Cluster

Here you can find the instructions for this specific Lab Step.

If you are ready for a real environment experience please start the Lab. Keep in mind that you'll need to start from the first step.

Start Lab

Introduction

Stateful applications are applications that have a memory of what happened in the past. Databases are an example of stateful applications. Kubernetes provides support for stateful applications through StatefulSets and related primitives.

This lab will deploy a replicated MySQL database as a StatefulSet. MySQL is one of the most popular databases in the world. This lab won't focus on the details specific to configuring MySQL. The focus is on the features of Kubernetes that allow a stateful application to be deployed. The Kubernetes community maintains a wide variety of stateful applications that are ready to deploy through Helm. Helm acts as a package manager for Kubernetes and can deploy entire applications to a cluster using templates called charts. A list of charts is available here. In addition to MySQL, there are charts for many other popular databases including MongoDB and PostgreSQL, as well as popular applications like WordPress and Joomla. The concepts illustrated in this lab will help you understand the common elements used to deploy all of these stateful applications in Kubernetes.

Managing your stateless and stateful applications with Kubernetes provides efficiencies and simplifies automation. However, before using StatefulSets for your own stateful applications, you should consider if any of th
apply:

e stateful

using specialized hardware and could effectively run on the same hardware used for stateless applications
- You value flexible reallocation of resources, consolidation, and automation over squeezing the most and having highly predictable performance

If any of the previous bullets apply to your situation, it may make sense to use Kubernetes for your stateful applications.

There are quite a few concepts at work in this lab step. Begin by reviewing the following background section and then jump into the instructions.

Background

ConfigMaps: A type of Kubernetes resource that is used to decouple configuration artifacts from image content to keep containerized applications portable. The configuration data is stored as key-value pairs.

Headless Service: A headless service is a Kubernetes service resource that won't load balance behind a single service IP. Instead, a headless service returns a list of DNS records that point directly to the pods that back the service. A headless service is defined by declaring the clusterIP property in a service spec and setting the value to None. StatefulSets currently require a headless service to identify pods in the cluster network.

Stateful Sets: Similar to Deployments in Kubernetes, StatefulSets manage the deployment and scaling of pods given a container spec.StatefulSets differ from Deployments in that the Pods in a stateful set are not interchangeable. Each pod in a StatefulSet has a persistent identifier that it maintains across any rescheduling. The pods in a StatefulSet are also ordered. This provides a guarantee that one pod can be created before following pods. In this lab, this is useful for ensuring the MySQL primary is provisioned first.

PersistentVolumes (PVs) and PersistentVolumeClaims (PVCs): PVs are Kubernetes resources that represent storage in the cluster. Unlike regular

s do not have a pods over be used by PVs

MySQL replication: This lab uses a single primary, asynchronous replication scheme for MySQL. All database writes are handled by a single primary. The database replicas asynchronously synchronize with the primary. This means the primary will not wait for the data to be copied onto the replicas. This can improve the performance of the primary at the expense of having replicas that are not always exact copies of the primary. Many applications can tolerate slight differences in the data and are able to improve the performance of database read workloads by allowing clients to read from the replicas.

Instructions

1. In your SSH shell, enter the following to declare a ConfigMap to allow primary MySQL pods to be configured differently than replica pods:

Copy code

```
cat <<EOF > mysql-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql
  labels:
    app: mysql
data:
  master.cnf: |
   # Apply this config only on the primary.
   [mysqld]
   log-bin
  slave.cnf: |
    # Apply this config only on replicas.
    [mysqld]
    super-read-only
EOF
```

This ConfigMap will be referenced later in the StatefulSet declaration. The `master.cnf` key maps to a value that declares a MySQL configuration which includes replication logs. The `slave.cnf` key maps to a MySQL configuration that enforces read-only behavior.

2. Create the ConfigMap resource:

Copy code

```
kubectl create -f mysql-configmap.yaml
```

Copy code

```
cat <<EOF > mysql-services.yaml
# Headless service for stable DNS entries of StatefulSet membe
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 read
# For writes, you must instead connect to the primary: mysql-0.
apiVersion: v1
kind: Service
metadata:
  name: mysql-read
  labels:
    app: mysql
spec:
  ports:
  - name: mysql
    port: 3306
  selector:
    app: mysql
EOF
```

Two services are defined:

- A headless service (`clusterIP: None`) for pod DNS resolution. Because the service is named mysql, pods are accessible via pod-name.mysql.
- A service name `mysql-read` to connect to for database reads. This service uses the default ServiceType of ClusterIP which assigns an internal IP address that load balances request to all the pods labeled with `app: mysql`.

Database writes need to be sent to the primary. The primary is the first pod provisioned in the StatefulSet and assigned a name `mysql-0`. The pod is thus accessed by the DNS entry in the headless service for mysql-0.mysql.

4. Create the MySQL services:

Copy code

```
kubectl create -f mysql-services.yaml
```

Copy code

```
cat <<EOF > mysql-storageclass.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: general
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
EOF
```

The built-in `aws-ebs` storage provision is specified along with the type `gp2`.

6. Create the storage class:

Copy code

```
kubectl create -f mysql-storageclass.yaml
```

7. Enter the following command to declare the MySQL StatefulSet:

Copy code

```
cat <<'EOF' > mysql-statefulset.yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  serviceName: mysql
  replicas: 3
  template:
    metadata:
      labels:
        app: mysql
    spec:
      initContainers:
      - name: init-mysql
        image: mysql:5.7.35
        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/se
                                                  onfig-map to e
                                                  onf.d/
```

Skip to content Press option + Q to open this menu

```
        - name: config-map
          mountPath: /mnt/config-map
      - name: clone-mysql
        image: gcr.io/google-samples/xtrabackup:1.0
        command:
        - bash
        - "-c"
        - |
          set -ex
          # Skip the clone if data already exists.
          [[ -d /var/lib/mysql/mysql ]] && exit 0
          # Skip the clone on primary (ordinal index 0).
          [[ `hostname` =~ -([0-9]+)$ ]] || exit 1
          ordinal=${BASH_REMATCH[1]}
          [[ $ordinal -eq 0 ]] && exit 0
          # Clone data from previous peer.
          ncat --recv-only mysql-$(($ordinal-1)).mysql 3307 |
          # Prepare the backup.
          xtrabackup --prepare --target-dir=/var/lib/mysql
        volumeMounts:
        - name: data
          mountPath: /var/lib/mysql
          subPath: mysql
        - name: conf
          mountPath: /etc/mysql/conf.d
      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
        resources:
          requests:
            cpu: 100m
            memory: 200Mi
        livenessProbe:
          exec:
            command: ["mysqladmin", "ping"]
          initialDelaySeconds: 30
          timeoutSeconds: 5
        readinessProbe:
          exec:
            # Check we can execute queries over TCP (skip-netw
            command: ["mysql", "-h", "127.0.0.1", "-e", "SELE
          initialDelaySeconds: 5
          timeoutSeconds: 1
      - name: xtrabackup
        image: gcr.io/google-samples/xtrabackup:1.0
        ports:
        - name: xtrabackup
          containerPort: 3307
        command:
        - bash
        - "-c"
        - |
          set -ex
          cd /var/lib/mysql
                                          data, if any.
                                          n
                                          tial "CHANGE M
                                          ting replica.
```

```
            [[ `cat xtrabackup_binlog_info` =~ ^(.*?)[[:space:
            rm xtrabackup_binlog_info
            echo "CHANGE MASTER TO MASTER_LOG_FILE='${BASH_REM
                  MASTER_LOG_POS=${BASH_REMATCH[2]}" > change_
          fi
          # Check if we need to complete a clone by starting m
          if [[ -f change_master_to.sql.in ]]; then
            echo "Waiting for mysqld to be ready (accepting cc
            until mysql -h 127.0.0.1 -e "SELECT 1"; do sleep 1
            echo "Initializing replication from clone positior
            # In case of container restart, attempt this at-mc
            mv change_master_to.sql.in change_master_to.sql.or
            mysql -h 127.0.0.1 <<EOF
          $(<change_master_to.sql.orig),
            MASTER_HOST='mysql-0.mysql',
            MASTER_USER='root',
            MASTER_PASSWORD='',
            MASTER_CONNECT_RETRY=10;
          START SLAVE;
          EOF
          fi
          # Start a server to send backups when requested by p
          exec ncat --listen --keep-open --send-only --max-cor
            "xtrabackup --backup --slave-info --stream=xbstrea
        volumeMounts:
        - name: data
          mountPath: /var/lib/mysql
          subPath: mysql
        - name: conf
          mountPath: /etc/mysql/conf.d
        resources:
          requests:
            cpu: 100m
            memory: 50Mi
      volumes:
      - name: conf
        emptyDir: {}
      - name: config-map
        configMap:
          name: mysql
  volumeClaimTemplates:
  - metadata:
      name: data
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 2Gi
      storageClassName: general
EOF
```

There is a lot going on in the StatefulSet. Don't focus too much on the bash scripts that are performing MySQL-specific tasks. Some highlights to focus on, following the order they appear in the file are:

- `init-containers`: Run to completion before any containers in the Pod `spec`
  - `init-mysql`: Assigns a unique MySQL server ID starting from 100 for the first pod and incrementing by one, as well as copying the appropriate configuration file from the `config-map`. Note ... `ts` section. The

files from the preceding pod. The `xtrabackup` tool performs the file cloning and persists the data on the `data` volume.

- `spec.containers`: Two containers in the pod
  - `mysql`: Runs the MySQL daemon and mounts the configuration in the `conf` volume and the `data` in the data volume
  - `xtrabackup`: A *sidecar* container that provides additional functionality to the `mysql` container. It starts a server to allow data cloning and begins replication on replicas using the cloned data files.
- `spec.volumes`: `conf` and `config-map` volumes are stored on the node's local disk. They are easily re-generated if a failure occurs and don't require PVs.
- `volumeClaimTemplates`: A template for each pod to create a PVC with. `ReadWriteOnce` `accessMode` allows the PV to be mounted by only one node at a time in read/write mode. The `storageClassName` references the AWS EBS gp2 storage class named `general` that you created earlier.

8. Create the StatefulSet and start watching the associated pods:

Copy code

```
kubectl create -f mysql-statefulset.yaml
kubectl get pods -l app=mysql --watch
```

The `--watch` option causes any updates to the pods to be written to the output. It takes a few minutes to initialize all three replicas.

9. In order to view the logs in S3, open the AWS Management Console by clicking the following button to access the lab's cloud environment:

0%

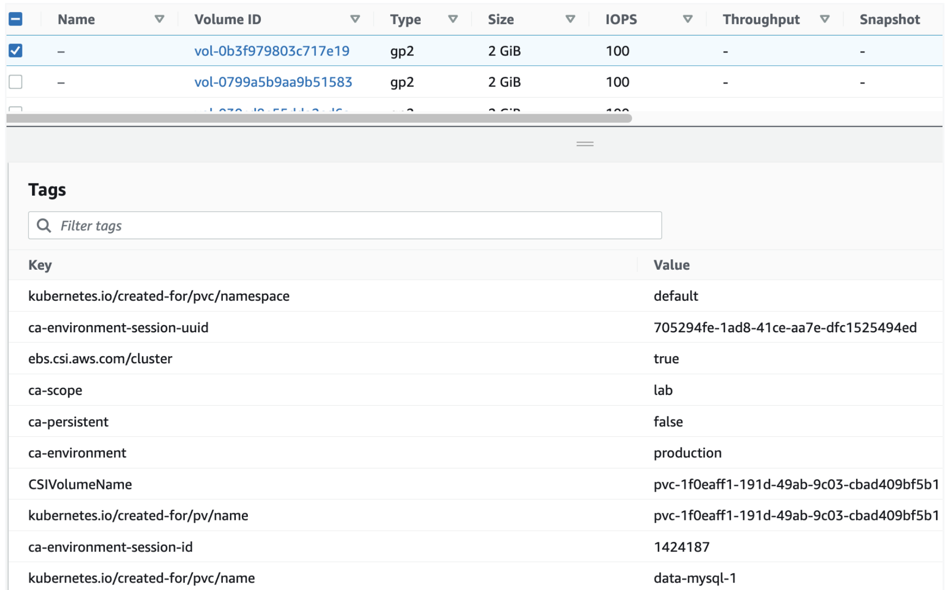
Open Cloud Environment

Loading...

Average setup time: 2m 35s

10. Enter the following credentials created just for your lab session, and click **Sign In**:

The 2GiB PVs are listed here as each pod is created. Notice the **Tags** which relay information about the PV and associated PVC.



Name	Volume ID	Type	Size	IOPS	Throughput	Snapshot
–	vol-0b3f979803c717e19	gp2	2 GiB	100	-	-
–	vol-0799a5b9aa9b51583	gp2	2 GiB	100	-	-

Tags

Filter tags

Key	Value
kubernetes.io/created-for/pvc/namespace	default
ca-environment-session-uuid	705294fe-1ad8-41ce-aa7e-dfc1525494ed
ebs.csi.aws.com/cluster	true
ca-scope	lab
ca-persistent	false
ca-environment	production
CSIVolumeName	pvc-1f0eaff1-191d-49ab-9c03-cbad409bf5b1
kubernetes.io/created-for/pv/name	pvc-1f0eaff1-191d-49ab-9c03-cbad409bf5b1
ca-environment-session-id	1424187
kubernetes.io/created-for/pvc/name	data-mysql-1

12. Return to your SSH shell and press ctrl+C to stop the watch when you see both containers (**2/2**) in the **mysql-2** pod running:

```
NAME      READY     STATUS      RESTARTS   AGE
mysql-0   0/2       Init:0/2    0          3s
mysql-0   0/2       Init:1/2    0          43s
mysql-0   0/2       PodInitializing   0          56s
mysql-0   1/2       Running   0          57s
mysql-0   2/2       Running   0          1m
mysql-1   0/2       Pending   0          0s
mysql-1   0/2       Pending   0          0s
mysql-1   0/2       Pending   0          2s
mysql-1   0/2       Init:0/2    0          2s
mysql-1   0/2       Init:1/2    0          42s
mysql-1   0/2       Init:1/2    0          55s
mysql-1   0/2       PodInitializing   0          1m
mysql-1   1/2       Running   0          1m
mysql-1   2/2       Running   0          1m
mysql-2   0/2       Pending   0          0s
mysql-2   0/2       Pending   0          0s
mysql-2   0/2       Pending   0          1s
mysql-2   0/2       Init:0/2    0          1s
mysql-2   0/2       Init:1/2    0          43s
mysql-2   0/2       Init:1/2    0          54s
mysql-2   0/2       PodInitializing   0          1m
```

Copy code

```
kubectl describe pv
kubectl describe pvc
```

The PV descriptions include the AWS **VolumeIDs**, file system types (**FSType**), and associated PVC (**Claim**). The PVC description includes whether the PVC is currently **Bound** to a pod.

14. Get the StatefulSet to confirm the current number of replicas matches the desired:

Copy code

```
kubectl get statefulset
```

Summary

In this lab step, you created several Kubernetes cluster resources to deploy the MySQL database as an example stateful application:

- A ConfigMap for decoupling primary and replica configuration from the containers
- Two Services: one headless service to manage network identity of pods in the StatefulSet, and one to load balance read access to the MySQL replicas
- A StorageClass to provision EBS PVs dynamically
- A StatefulSet that declared two init-containers, two containers, and one PVC template

You observed the ordered sequence of pods being initialized and the PVs created in AWS to facilitate the StatefulSet.

About Circus Street

COMMUNITY

Join Discord Channel

HELP

Help Center