MySQL on Statefulsets

Author: Vedang Joshi
Published: April 4, 2025 · 3 min read

The following kubernetes documentation and examples are outdated with the recent mysql version
https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

So I took it upon myself to run the latest MySQL version with 2 reader replicas and 1 write replica. In this post, I'll walk you through my approach step-by-step, explaining how I managed to set up a MySQL cluster using Kubernetes StatefulSets. Grab your favorite cup of coffee, and let's dive in!

The Overall Approach

Here's the quick rundown of what I did:

1. **ConfigMap Magic:**
   I started by creating a ConfigMap that bundles two key scripts:
   - `init-master.sh` : This script configures the master node by setting up replication parameters.
   - `init-slave.sh` : This script is designed for the slave nodes, ensuring they connect and replicate from the master.

   These scripts are vital as they initialize the replication setup for MySQL.
2. **StatefulSet Setup:**
   The heart of the deployment is a StatefulSet, which I configured to handle both the master and slave roles based on the pod hostname.
   - **Init Container:**
     Before the main containers kick in, an init container checks the hostname. If it's the first pod (i.e., hostname ends with `-0`), it's designated as the master; otherwise, it's recognized as a slave. It then copies the appropriate script (either `init-master.sh` or `init-slave.sh`) from the ConfigMap.
   - **Main & Sidecar Containers:**
     - **Main Container:** Runs MySQL 8.0 as expected.
     - **Sidecar Container:** Executes the copied script.
       Note: You can't merge the functionality of an init container with a sidecar because the sidecar has to run *after* MySQL is fully up and running. This separation ensures the scripts execute at the appropriate time for proper replication setup.

3. **Service Configuration:**
   Once the StatefulSet is up, I created two kinds of services:
   - **Headless Service:**
     This service lets you connect directly to any container. Specifically, I use it to connect to the master node via `mysql-svc-0` .
   - **Replica Service:**
     With a label selector of `replica` , this service is tailored to connect exclusively to the reader nodes.

4. **Labeling the Readers:**
   The final step was to apply specific labels to the two reader nodes. This labeling makes it easy to differentiate between the master and the replicas when routing read and write traffic.

Putting It All Together

Below is a snippet where you'd copy and paste your YAML files:

```yaml
# - ConfigMap with init-master.sh and init-slave.sh
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-config
data:
  master.cnf: |
    [mysqld]
    log-bin=mysql-bin
    binlog-format=ROW
    server-id=1
  slave.cnf: |
    [mysqld]
    server-id=2
  init-master.sh: |
    #!/bin/bash
    set -ex
    echo "Creating replication user..."
    mysql -h 127.0.0.1 -u root -p${MYSQL_ROOT_PASSWORD} <<EOF
    CREATE USER IF NOT EXISTS 'replication'@'%'
    IDENTIFIED WITH mysql_native_password BY 'repl_password';
    GRANT REPLICATION SLAVE ON *.* TO 'replication'@'%';
    FLUSH PRIVILEGES;
    EOF
  init-slave.sh: |
    #!/bin/bash
    set -ex
    echo "Waiting for master to be ready..."
    until mysql -h 127.0.0.1 -h mysql-0.mysql -u root -p${MYSQL_ROOT_PASSWORD} \
-e "SELECT 1"; do
      echo "Master is not ready yet..."
      sleep 3
    done
    echo "Getting master position..."
    MASTER_STATUS=$(mysql -h mysql-0.mysql -u root -p${MYSQL_ROOT_PASSWORD} \
-e "SHOW MASTER STATUS" --skip-column-names)
    MASTER_LOG_FILE=$(echo $MASTER_STATUS | cut -f 1 -d ' ')
    MASTER_LOG_POS=$(echo $MASTER_STATUS | cut -f 2 -d ' ')
    echo "Stopping replica IO thread if running..."
    mysql -h 127.0.0.1 -u root -p${MYSQL_ROOT_PASSWORD} \
-e "STOP REPLICA IO_THREAD FOR CHANNEL '';"
    echo "Setting up slave with master log file: $MASTER_LOG_FILE \
and position: $MASTER_LOG_POS"
    mysql -h 127.0.0.1 -u root -p${MYSQL_ROOT_PASSWORD} <<EOF
    CHANGE MASTER TO
    MASTER_HOST='mysql-0.mysql',
    MASTER_USER='replication',
    MASTER_PASSWORD='repl_password',
    MASTER_LOG_FILE='$MASTER_LOG_FILE',
    MASTER_LOG_POS=$MASTER_LOG_POS;
    START SLAVE;
    EOF
    mysql -h 127.0.0.1 -u root -p${MYSQL_ROOT_PASSWORD} <<EOF
    SET GLOBAL super_read_only = 1
    EOF
    echo "Slave setup complete!"
---

# - StatefulSet definition for MySQL with init container and sidecar container
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:8.0
        command:
        - bash
        - "-c"
        - |
          set -ex
          # Generate server-id based on ordinal index
          [[ $HOSTNAME =~ -([0-9]+)$ ]] || exit 1
          ordinal=${BASH_REMATCH[1]}
          # Only the first pod (ordinal 0) is master
          if [[ $ordinal -eq 0 ]]; then
            cp /mnt/config-map/master.cnf /etc/mysql/conf.d/
          else
            # Update server-id for slaves
            cp /mnt/config-map/slave.cnf /etc/mysql/conf.d/
            sed -i "s/server-id=2/server-id=$((ordinal + 1))/" \
/etc/mysql/conf.d/slave.cnf
          fi
        volumeMounts:
        - name: conf
          mountPath: /etc/mysql/conf.d
        - name: config-map
          mountPath: /mnt/config-map
      containers:
      - name: mysql
        image: mysql:8.0
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "rootpassword"
        ports:
        - name: mysql
          containerPort: 3306
        volumeMounts:
        - name: data
          mountPath: /var/lib/mysql
        - name: conf
          mountPath: /etc/mysql/conf.d
        - name: config-map
          mountPath: /mnt/config-map
        resources:
          requests:
            cpu: 500m
            memory: 1Gi
        livenessProbe:
          exec:
            command:
            - mysqladmin
            - ping
            - -u
            - root
            - -prootpassword
          initialDelaySeconds: 30
          periodSeconds: 10
          timeoutSeconds: 5
        readinessProbe:
          exec:
            command:
            - mysql
            - -u
            - root
            - -prootpassword
            - -e
            - SELECT 1
          initialDelaySeconds: 5
          periodSeconds: 2
          timeoutSeconds: 1
      - name: replication-init
        image: mysql:8.0
        command:
        - bash
        - "-c"
        - |
          set -ex
          # Wait for MySQL to be ready
          until mysqladmin ping -h 127.0.0.1 -u root -p${MYSQL_ROOT_PASSWORD}; do
            echo "Waiting for MySQL to be ready..."
            sleep 2
          done
          # Determine if master or slave based on hostname
          [[ $HOSTNAME =~ -([0-9]+)$ ]] || exit 1
          ordinal=${BASH_REMATCH[1]}
          if [[ $ordinal -eq 0 ]]; then
            echo "Initializing master..."
            # Execute the script content instead of trying to run the file
            bash -c "$(cat /mnt/config-map/init-master.sh)"
          else
            echo "Initializing slave..."
            # Execute the script content instead of trying to run the file
            bash -c "$(cat /mnt/config-map/init-slave.sh)"
          fi
          # Keep container running
          tail -f /dev/null
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "rootpassword"
        volumeMounts:
        - name: config-map
          mountPath: /mnt/config-map
      volumes:
      - name: conf
        emptyDir: {}
      - name: config-map
        configMap:
          name: mysql-config
  volumeClaimTemplates:
  - metadata:
      name: data
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 10Gi
---

# - Service definitions for headless service and replica service
apiVersion: v1
kind: Service
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  ports:
  - port: 3306
    name: mysql
  clusterIP: None
  selector:
    app: mysql
---

apiVersion: v1
kind: Service
metadata:
  name: mysql-read
  labels:
    app: mysql
    app.kubernetes.io/name: mysql
    readonly: "true"
spec:
  ports:
  - name: mysql
    port: 3306
  selector:
    app: mysql
    replica: "true"
---
```

```
# - Commands for applying labels to reader nodes
kubectl label pods mysql-1 replica=true
kubectl label pods mysql-2 replica=true
```

Final Thoughts

This is a decent approach for using MySQL with StatefulSets. It leverages MySQL's default replication mechanism to create a small MySQL cluster. For a user application, say a Django project, this setup provides database routers that allow you to specify separate configurations for reader and write databases.

Just a quick heads up:
Be mindful that all nodes are in the same Availability Zone (AZ) to avoid incurring extra data transfer charges. While this isn't a production-ready solution, the intention was more to showcase how Kubernetes StatefulSets can be utilized to run MySQL replicas—an educational dive into blending Kubernetes with traditional database replication strategies.

Hope this gives you a clearer picture of how you can run MySQL on Kubernetes with a modern twist. Happy coding and clustering!

Kubernetes Database