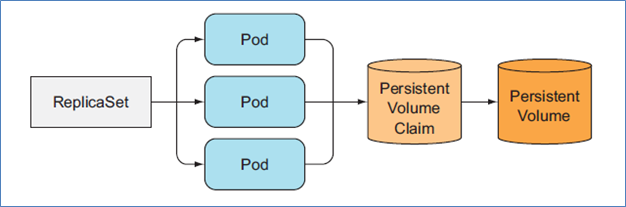
REPLICASET:

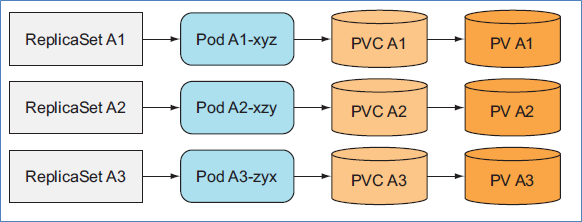


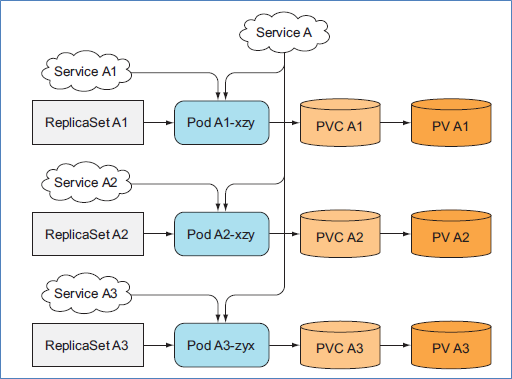
ReplicaSets create multiple pod replicas from a single pod template. These replicas don’t differ from each other, apart from their name and IP address. If the pod template includes a volume, which refers to a specific PersistentVolumeClaim, all replicas of the ReplicaSet will use the exact same PersistentVolumeClaim and therefore the same PersistentVolume bound by the claim.

USING ONE REPLICASET PER POD INSTANCE

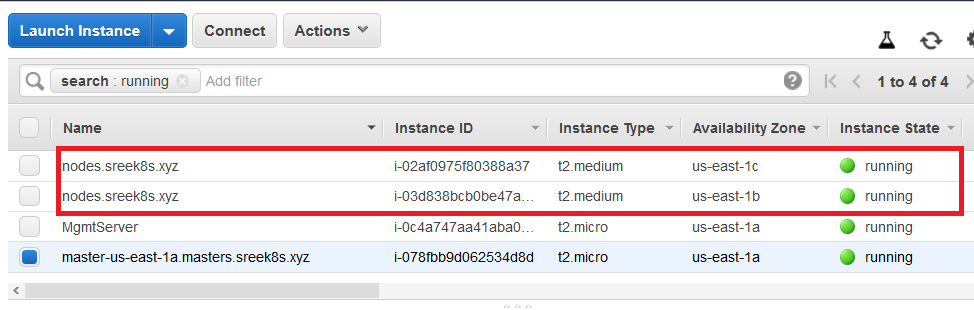
Instead of creating pods directly, you could create multiple ReplicaSets—one for each pod with each ReplicaSet’s desired replica count set to one, and each ReplicaSet’s pod template referencing a dedicated PersistentVolumeClaim. Although this takes care of the automatic rescheduling in case of node failures or accidental pod deletions, it’s much more cumbersome compared to having a single ReplicaSet.

In addition to storage, certain clustered applications also require that each instance has a long-lived stable identity. Pods can be killed from time to time and replaced with new ones. When a ReplicaSet replaces a pod, the new pod is a completely new pod with a new hostname and IP, although the data in its storage volume may be that of the killed pod. For certain apps, starting up with the old instance’s data but with a completely new network identity may cause problems. Why do certain apps mandate a stable network identity? This requirement is fairly common in distributed stateful applications. Certain apps require the administrator to list all the other cluster members and their IP addresses (or hostnames) in each member’s configuration file. But in Kubernetes, every time a pod is rescheduled, the new pod gets both a new hostname and a new IP address, so the whole application cluster would have to be reconfigured every time one of its members is rescheduled.





Statefull set needs following cluster to have a successful mysql cluster. Use t2.medium for nodes.



#https://kubernetes.io/docs/tasks/run-application/run-replicated-stateful-application/

#EFS Need to deploy first before going to deploy the statefull set.

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]

super-read-only

# 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

---

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

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

- 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 master (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 | xbstream -x -C /var/lib/mysql

# 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: 500m

memory: 1Gi

livenessProbe:

exec:

command: ["mysqladmin", "ping"]

initialDelaySeconds: 30

periodSeconds: 10

timeoutSeconds: 5

readinessProbe:

exec:

# Check we can execute queries over TCP (skip-networking is off).

command: ["mysql", "-h", "127.0.0.1", "-e", "SELECT 1"]

initialDelaySeconds: 5

periodSeconds: 2

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

# Determine binlog position of cloned data, if any.

if [[ -f xtrabackup\_slave\_info && "x$(<xtrabackup\_slave\_info)" != "x" ]]; then

# XtraBackup already generated a partial "CHANGE MASTER TO" query

# because we're cloning from an existing slave. (Need to remove the tailing semicolon!)

cat xtrabackup\_slave\_info | sed -E 's/;$//g' > change\_master\_to.sql.in

# Ignore xtrabackup\_binlog\_info in this case (it's useless).

rm -f xtrabackup\_slave\_info xtrabackup\_binlog\_info

elif [[ -f xtrabackup\_binlog\_info ]]; then

# We're cloning directly from master. Parse binlog position.

[[ `cat xtrabackup\_binlog\_info` =~ ^(.\*?)[[:space:]]+(.\*?)$ ]] || exit 1

rm -f xtrabackup\_binlog\_info xtrabackup\_slave\_info

echo "CHANGE MASTER TO MASTER\_LOG\_FILE='${BASH\_REMATCH[1]}',\

MASTER\_LOG\_POS=${BASH\_REMATCH[2]}" > change\_master\_to.sql.in

fi

# Check if we need to complete a clone by starting replication.

if [[ -f change\_master\_to.sql.in ]]; 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 "Initializing replication from clone position"

mysql -h 127.0.0.1 \

-e "$(<change\_master\_to.sql.in), \

MASTER\_HOST='mysql-0.mysql', \

MASTER\_USER='root', \

MASTER\_PASSWORD='', \

MASTER\_CONNECT\_RETRY=10; \

START SLAVE;" || exit 1

# In case of container restart, attempt this at-most-once.

mv change\_master\_to.sql.in change\_master\_to.sql.orig

fi

# Start a server to send backups when requested by peers.

exec ncat --listen --keep-open --send-only --max-conns=1 3307 -c \

"xtrabackup --backup --slave-info --stream=xbstream --host=127.0.0.1 --user=root"

volumeMounts:

- name: data

mountPath: /var/lib/mysql

subPath: mysql

- name: conf

mountPath: /etc/mysql/conf.d

resources:

requests:

cpu: 100m

memory: 100Mi

volumes:

- name: conf

emptyDir: {}

- name: config-map

configMap:

name: mysql

volumeClaimTemplates:

- metadata:

name: data

spec:

accessModes: ["ReadWriteOnce"]

storageClassName: "aws-efs"

resources:

requests:

storage: 10Gi

#Creating DB and Tables

mysql

mysql> CREATE DATABASE books;

USE books;

CREATE TABLE authors (id INT, name VARCHAR(20), email VARCHAR(20));

SHOW TABLES;

INSERT INTO authors (id,name,email) VALUES(1,"Vivek","xuz@abc.com");

INSERT INTO authors (id,name,email) VALUES(2,"Priya","p@gmail.com");

INSERT INTO authors (id,name,email) VALUES(3,"Tom","tom@yahoo.com");

SHOW DATABASES;

SELECT \* FROM authors;

#Log in secondary and check the status

mysql

show databases;

use books;

show tables;

select \* from authors;

mysql> drop database books;

ERROR 1290 (HY000): The MySQL server is running with the --super-read-only option so it cannot execute this statement

#Data mysql-0 and S-Set will create a new pod and data must be there.

kubectl run --generator=run-pod/v1 --rm utils -it --image eddiehale/utils bash

root@utils:/# nslookup mysql

Server: 100.64.0.10

Address: 100.64.0.10#53

Name: mysql.default.svc.cluster.local

Address: 100.96.1.4

Name: mysql.default.svc.cluster.local

Address: 100.96.2.4

Name: mysql.default.svc.cluster.local

Address: 100.96.2.5

apt install python3-pip

pip3 install sqlalchemy

pip3 install pymysql

#For Reading app.py

import sqlalchemy as sal

import pymysql

from sqlalchemy import create\_engine

cnx = create\_engine('mysql+pymysql://root@mysql-read/books')

data = cnx.execute("select \* from authors").fetchall()

#putdata = cnx.execute('INSERT INTO authors (id,name,email) VALUES(4,"Sree","sree@yahoo.com");')

for item in data:

print(item)

#For Writing app-write.py

import sqlalchemy as sal

import pymysql

from sqlalchemy import create\_engine

cnx = create\_engine('mysql+pymysql://root@mysql-0.mysql.default.svc.cluster.local/books')

data = cnx.execute("select \* from authors").fetchall()

putdata = cnx.execute('INSERT INTO authors (id,name,email) VALUES(7,"Munna1","Munna1@yahoo.com");')

for item in data:

print(item)

ku run --generator=run-pod/v1 --image sreeharshav/rollingupdate:v3

ku expose deploy nginx-deployment --target-port=80 --type=ClusterIP --cluster-ip=None

ku expose statefullset nginxstatefulset --target-port=80 --type=ClusterIP --cluster-ip=None --name nginxstatefulset

kubectl run --generator=run-pod/v1 --rm utils -it --image sreeharshav/utils -- bash

SAMPLE STATEFULL-SET:

apiVersion: v1

kind: Service

metadata:

name: nginxstatefulset

labels:

app: nginxstatefulset

spec:

ports:

- name: mynginxstatefulsetsql

port: 80

clusterIP: None

selector:

app: nginxstatefulset

---

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: nginxstatefulset

labels:

app: nginxstatefulset

spec:

replicas: 3

selector:

matchLabels:

app: nginxstatefulset

serviceName: nginxstatefulset

template:

metadata:

labels:

app: nginxstatefulset

spec:

containers:

- name: nginx

image: sreeharshav/testcontainer:v1

ports:

- containerPort: 80