**Deploying Redis Cluster on Kubernetes**

**1. Create a Namespace**

By default, all the [Kubernetes components](https://www.containiq.com/post/kubernetes-components) such as pods, services, [ConfigMaps](https://www.containiq.com/post/kubernetes-configmap), etc. are placed in the default [namespace](https://www.containiq.com/post/kubernetes-namespaces). A namespace is like a folder in the OS. Keep in mind that placing all components in one namespace can create confusion; creating a new namespace will help you manage the pods and services more efficiently.

So create a namespace for a Redis server setup:

kubectl create ns redis

1. **Define a Storage Class**

A [storage class](https://kubernetes.io/docs/concepts/storage/storage-classes/) links to a provisioner, which is a plugin that can reserve disk space or purchase volumes to your cloud provider on your behalf.

The pod in the Kubernetes cluster does not store the data permanently. Rather, the data created inside the pod is deleted whenever the pod is deleted or restarted. To store the data permanently, Kubernetes offers volume and persistent volume.

**Create a storage class, which points to the local storage, using the following manifest code:**

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: local-storage

provisioner: kubernetes.io/no-provisioner

volumeBindingMode: WaitForFirstConsumer

allowVolumeExpansion: true

reclaimPolicy: Delete

**Copy the above code and paste it into a file named sc.yaml.**

**Execute the file using the following command:**

kubectl apply -f sc.yaml

The storage classes are created globally in the Kubernetes cluster. Thus, there is no need to mention the namespace while executing the manifest file for the storage class.

However, for the pods and services, you must mention the namespace while executing the manifest files.

Now that the local storage class is created, you can see the list of available storage classes using the following command:

kubectl get sc

1. **Create a Persistent Volume**

Persistent volumes (PVs) are used to create a storage size. In this demo, you will create a Redis cluster with three pods (one master and two slaves). Therefore, create three PVs.

The following code creates three PVs using the local storage provisioner:

apiVersion: v1

kind: PersistentVolume

metadata:

name: local-pv1spec:

storageClassName: local-storage

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

hostPath:

path: "/storage/data1"

---

apiVersion: v1kind: PersistentVolumemetadata:

name: local-pv2spec:

storageClassName: local-storage

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

hostPath:

path: "/storage/data2"

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: local-pv3spec:

storageClassName: local-storage

capacity:

storage: 2Gi

accessModes:

- ReadWriteOnce

hostPath:

path: "/storage/data3"

**Copy and paste the code into a file named pv.yaml. Execute the file with the following command:**

kubectl apply -f pv.yaml

With three PVs now created, get the PVs list:

kubectl get pv

Because this demo claims the PV using the [volume claim templates](https://www.elastic.co/guide/en/cloud-on-k8s/1.0/k8s-volume-claim-templates.html) of pod definition, you don’t need to create PVCs separately.

**4. Create the ConfigMap**

The ConfigMap in the Kubernetes cluster is a key-value store. You can use the config information of Redis in the Kubernetes cluster as a ConfigMap. There are a couple important things to note here.

First, change the password of master and slave with your desired password, which is needed for authentication.

apiVersion: v1

kind: ConfigMap

metadata:

name: redis-cluster

namespace: redis

data:

update-node.sh: |

#!/bin/sh

REDIS\_NODES="/data/nodes.conf"

sed -i -e "/myself/ s/[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}/${POD\_IP}/" ${REDIS\_NODES}

exec "$@"

redis.conf: |+

cluster-enabled yes

cluster-require-full-coverage no

cluster-node-timeout 15000

cluster-config-file /data/nodes.conf

cluster-migration-barrier 1

appendonly yes

protected-mode no

Keep your master and slave passwords the same; you will need to set up the failover of the master pod. If the master pod dies or restarts, then any slave pod will be made the master using. Using the same password will ensure easy communication between master and slave.

Second, do not hard code the slaveof value. That’s a placeholder for the master instance address. You need to generate this value on the fly due to the master failover. This value is set dynamically on the [StatefulSet](https://www.containiq.com/post/kubernetes-statefulsets) deployment section.

Get the [ConfigMap code](https://gist.github.com/bharathirajatut/dcebde585eba5ac8b1398b8ed653d32d) and save the code in a file named redis-config.yaml. Deploy the ConfigMap in the Redis namespace using the following command:

kubectl apply -n redis -f redis-config.yaml

Get the ConfigMap list in the Redis namespace:

kubectl get configmap -n redis

### **5. Deploy Redis Using StatefulSet**

StatefulSet is a Kubernetes object used to deploy stateful applications such as MySQL, Oracle, MSSQL, and ElasticSearch. You can use the Deployment object if you are planning to deploy stateless applications such as PHP, Jave, or Perl.

The StatefulSet offers ordered pod names starting from zero and recreates the pod with the same name whenever the pod dies or crashes. A pod can fail at any time. The persistent pod identifier uses this feature (recreating the pod with the same name) to match existing persistent volume (storage volume attached to the failed pod) to the newly created pod.

These features are needed when you deploy the stateful application. Therefore, use the StatefulSet controller to deploy the Redis cluster:

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: redisspec:

serviceName: redis

replicas: 3

selector:

matchLabels:

app: redis

template:

metadata:

labels:

app: redis

spec:

initContainers:

- name: config

image: redis:6.2.3-alpine

command: [ "sh", "-c" ]

args:

- |

cp /tmp/redis/redis.conf /etc/redis/redis.conf

echo "finding master..."

MASTER\_FDQN=`hostname -f | sed -e 's/redis-[0-9]\./redis-0./'`

if [ "$(redis-cli -h sentinel -p 5000 ping)" **!=** "PONG" ]; then

echo "master not found, defaulting to redis-0"

if [ "$(hostname)" == "redis-0" ]; then

echo "this is redis-0, not updating config..."

else

echo "updating redis.conf..."

echo "slaveof $MASTER\_FDQN 6379" >> /etc/redis/redis.conf

fi

else

echo "sentinel found, finding master"

MASTER="$(redis-cli -h sentinel -p 5000 sentinel get-master-addr-by-name mymaster | grep -E '(^redis-\d{1,})|([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})')"

echo "master found : $MASTER, updating redis.conf"

echo "slaveof $MASTER 6379" >> /etc/redis/redis.conf

fi

volumeMounts:

- name: redis-config

mountPath: /etc/redis/

- name: config

mountPath: /tmp/redis/

containers:

- name: redis

image: redis:6.2.3-alpine

command: ["redis-server"]

args: ["/etc/redis/redis.conf"]

ports:

- containerPort: 6379

name: redis

volumeMounts:

- name: data

mountPath: /data

- name: redis-config

mountPath: /etc/redis/

volumes:

- name: redis-config

emptyDir: {}

- name: config

configMap:

name: redis-config

volumeClaimTemplates:

- metadata:

name: data

spec:

accessModes: [ "ReadWriteOnce" ]

storageClassName: "local-storage"

resources:

requests:

storage: 500Mi

Save the above code in a file named redis-statefulset.yaml and execute using the following command:

kubectl apply -n redis -f redis-statefulset.yaml

Now, get the list of running pods under the Redis namespace:

kubectl get pods -n redis

Now three pods are up and running: redis-0, redis-1, and redis-2. The redis-0 pod will act as master, and the other pods will act as slaves.

Do note that it is possible that you will receive the following error you when you create the StatefulSet for Redis:

0/2 nodes are available: 2 pod has unbound immediate PersistentVolumeClaims.

The code used for Redis StatefulSet will look for a PV. If you have dynamic provisioning, then the code itself will automatically create the three PVs. If you don’t have dynamic provisioning, then create three PVs with the same access mode in the volume claim template to resolve the error.

### **6. Create Headless Service**

You cannot directly access the application running in the pod. If you want to access the application, you need a Service object in the Kubernetes cluster. Create a headless service for a Redis pod using the following code:

apiVersion: v1kind: Servicemetadata:

name: redisspec:

clusterIP: None

ports:

- port: 6379

targetPort: 6379

name: redis

selector:

app: redis

Headless service means that only internal pods can communicate with each other. They are not exposed to external requests outside of the Kubernetes cluster.

Save the previous code in a file named redis-service.yaml and execute the code with this command:

kubectl apply -n redis -f redis-service.yaml

**Now get the list of running services under the Redis namespace:**

kubectl get service -n redis

**The service endpoints for each Redis pod are given below:**

#syntax

pod\_name.service\_name.namespace.svc.cluster.local

#Example

redis-0.redis.redis.svc.cluster.local

redis-1.redis.redis.svc.cluster.local

redis-2.redis.redis.svc.cluster.local

Now you can access the Redis pods using the above endpoints with Redis client utility tools.