How to Deploy Redis Cluster on Kubernetes

September 30, 2021

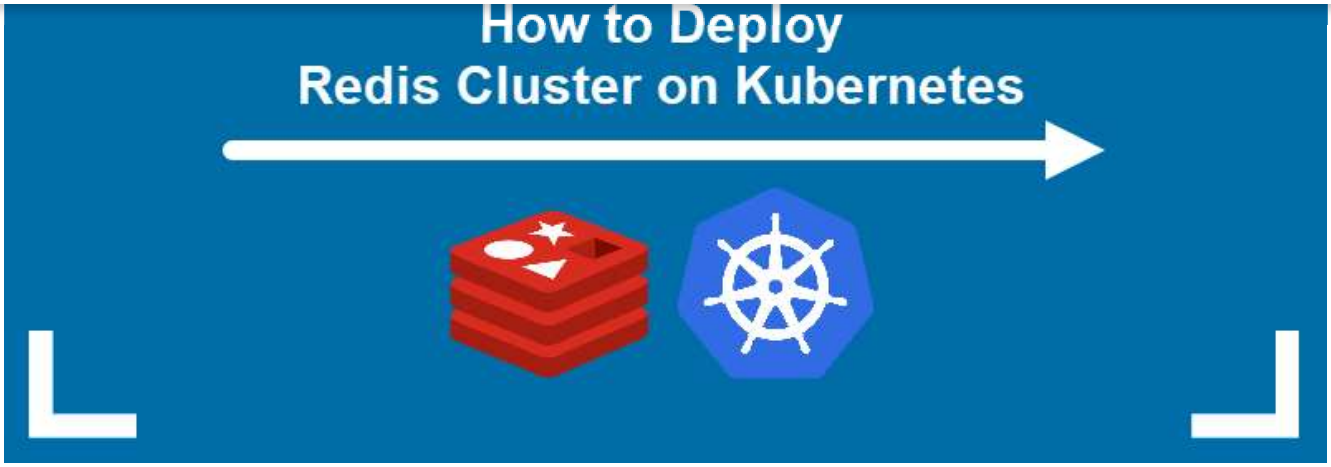
KUBERNETES REDIS

Introduction

Redis is a popular NoSQL database and an in-memory data store supporting multiple abstract data structures. These include strings, lists, hashes, sets, streams, etc. Redis provides syntax for accessing mutable data structures, allowing multiple processes to read and write them in a shared way.

A Redis Cluster is a set of Redis instances that automatically shards data across nodes. Using a cluster gives users the ability to split their datasets among nodes and keep running the database even when some nodes fail.

This tutorial will show you how to deploy a Redis Cluster on Kubernetes using ConfigMap and Helm.



Prerequisites

- A Kubernetes cluster consisting of two or more nodes
- Helm 3 installed
- kubectl 1.14 or above installed

Note: If you are using Minikube, you can simulate a two-node cluster by adding the **`--nodes`** option to the **`start`** command:

```
minikube start --nodes 2
```

Deploying Redis on Kubernetes with ConfigMap

The following steps explain how to configure Redis cache and a pod containing a Redis instance.

1. Using a text editor, create a ConfigMap YAML that will store the Redis configuration.

```
nano redis-conf.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-redis-config
data:
  redis-config: |
    maxmemory 2mb
    maxmemory-policy allkeys-lru
```

The example configuration above sets the **maxmemory** directive and tells Redis to use the maximum of 2 MB of storage for the data set. The **maxmemory-policy** directive defines the procedure to be applied when the memory limit is reached. **allkeys-lru** first removes the less recently used (LRU) keys.

3. Save the file and exit.

4. Create the ConfigMap by applying the YAML file.

```
kubectl apply -f redis-conf.yaml
```

The system confirms that the operation was successful.

```
marko@test-main:~/redis$ kubectl apply -f redis-conf.yaml
configmap/test-redis-config created
marko@test-main:~/redis$
```

5. Create a Redis pod manifest.

```
nano redis-pod.yaml
```

6. Specify your pod configuration.

```
    name: redis
  spec:
    containers:
    - name: redis
      image: redis:5.0.4
      command:
        - redis-server
        - "/redis-master/redis.conf"
      env:
      - name: MASTER
        value: "true"
      ports:
      - containerPort: 6379
      resources:
        limits:
          cpu: "0.1"
      volumeMounts:
      - mountPath: /redis-master-data
        name: data
      - mountPath: /redis-master
        name: config
    volumes:
      - name: data
        emptyDir: {}
      - name: config
        configMap:
          name: test-redis-config
          items:
          - key: redis-config
            path: redis.conf
```

In the example above, the manifest defines **`config`** volume and mounts it at **`/redis-master`** directory on the pod. The **`spec.volumes.items`** section then exposes the **`redis-config`** key

```
kubectl apply -f redis-pod.yaml
```

The system confirms that the pod creation was successful.

```
marko@test-main:~/redis$ kubectl apply -f redis-pod.yaml
pod/redis created
marko@test-main:~/redis$
```

9. Check pod status.

```
kubectl get pod
```

```
marko@test-main:~/redis$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
redis   1/1     Running   0          2m7s
marko@test-main:~/redis$
```

10. Enter the created pod with **`kubectl exec`**.

```
kubectl exec -it redis -- redis-cli
```

The Redis server console appears:

```
marko@test-main:~/redis$ kubectl exec -it redis -- redis-cli
127.0.0.1:6379>
```

11. Use the PING command to check if the server is online.

```
PING
```

```
127.0.0.1:6379> PING
PONG
127.0.0.1:6379>
```

maximum performance of your Redis cluster. Achieve a reliable infrastructure with servers starting at **only $0.10/hour!**

Deploying Redis on Kubernetes with Helm Chart

Helm provides a quick way of setting up a Redis cluster using a pre-made Helm chart.

1. Add the Helm repository containing the Redis chart you wish to install.

```
helm repo add [repo-name] [repo-address]
```

This article uses the Redis chart available in the Bitnami repository.

```
marko@test-main:~/redis$ helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
marko@test-main:~/redis$
```

2. Update local Helm repositories.

```
helm repo update
```

```
marko@test-main:~/redis$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "kubernetes-dashboard" chart repository
...Successfully got an update from the "banzaicloud-stable" chart repository
...Successfully got an update from the "stable" chart repository
...Successfully got an update from the "bitnami" chart repository
Update Complete. ⎈Happy Helming!⎈
marko@test-main:~/redis$
```

3. Use **`helm install`** to install the chart. The basic command is as follows:

```
helm install redis-test bitnami/redis
```

fs-client for both pods.

Since Redis is deployed with non-administrative volume permissions by default, the Redis pod may not be able to communicate with the server. Resolve this problem by setting **volume Permissions** to **true**.

The final **helm install** command should look like this:

```
helm install redis-test --set persistence.storageClass=nfs-client,red
is.replicas.persistence.storageClass=nfs-client bitnami/redis --set v
olumePermissions.enabled=true
```

4. Export the Redis password as an environment variable.

```
export REDIS_PASSWORD=$(kubectl get secret --namespace default redis-
test -o jsonpath="{.data.redis-password}" | base64 --decode)
```

5. Create a Redis client pod that you will use to access the database.

```
kubectl run --namespace default redis-client --restart='Never'  --env
REDIS_PASSWORD=$REDIS_PASSWORD  --image docker.io/bitnami/redis:6.2.5
-debian-10-r63 --command -- sleep infinity
```

The system confirms the creation of the **redis-client** pod.

```
kubectl exec --tty -i redis-client --namespace default -- bash
```

The client console appears.

7. Use the following **redis-cli** command to access the Redis master instance:

```
redis-cli -h redis-test-master -a $REDIS_PASSWORD
```

The console for the master instance displays.

8. To access the replicas, use the same command, but alter the target.

```
redis-cli -h redis-test-replicas -a $REDIS_PASSWORD
```

9. Use the **PING** command to test the connection with the server.

```
PING
```

The **PONG** response confirms that the server is listening.

Conclusion

software.

Was this article helpful? Yes No

Marko Aleksic

Marko Aleksić is a Technical Writer at phoenixNAP. His innate curiosity regarding all things IT, combined with over a decade long background in writing, teaching and working in IT-related fields, led him to technical writing, where he has an opportunity to employ his skills and make technology less daunting to everyone.

Next you should read

Databases, SysAdmin

How to Install Redis on Mac

November 5, 2020

Redis, an in-memory database, stands out with its flexibility and high performance, wide language support, and high availability. In this tutorial, you will learn several ways to install and configure

Databases, DevOps and Development

Redis Data Types with Commands

September 4, 2020

Redis allows you to use various data types such as Lists, Hashes, Sets, and Sorted Sets to store and manage data. Learn how Redis Data Types work and master basic commands..

READ MORE

Databases, SysAdmin, Virtualization

How to Deploy and Run Redis in Docker

July 23, 2020

Using Docker to deploy Redis in a container makes horizontal scaling a straightforward process. Follow this tutorial to deploy Redis in a Docker container in no time.

Databases

How to Install Redis on Ubuntu

July 7, 2020

Apart from its performance and flexibility, Redis stands out with its wide language support, high availability, and automatic partitioning. In this tutorial, learn how to install Redis on Ubuntu.

READ MORE