Redis

I know I'm going to need Redis in the future for some buffering between database and functions. There are lots of uses for Redis, in-memory data structure storage (also known as database in Ram). I had particular use for Redis when I had a lot of `worker scripts` that would normally write to an SQL database (MariaDB), and to the same table and line, which on occasion resulted in lock. So, all the workers started to write to the Redis server, and as Honey badger, Redis don't give a fuck about huge loads and just takes it... I then wrote a simple single `db worker` that on its own time and speed offloaded data to the database.

Redis

You can read much more about Redis here: https://redis.io it's not that difficult.

There are multiple ways to deploy Redis, with multiple master - slave nodes, and so on... but for my purpose, I need only one. If a node fails where it runs, Kubernetes will start it on another, so High Availability is acceptable with small outages, till it starts elsewhere.

Persistence

I usually don't store long term data in Redis and try to consume them as fast as I can, however this whole thing is in Ram; if the pod dies, the data dies with it. Redis supports two types of persistence: regular backups every X seconds or minutes (snapshots), or a kind of log that keeps every transaction as it happens, and this is replayed back when Redis starts up. I'm going for the second one here, since it offers a backup near 1:1 always. Honestly, read about both methods here: https://redis.io/topics/persistence . We are going with `AOF`.

For persistent storage I'm going to use 5GB of Longhorn distributed storage I set up before.

Guide

So, create one directory where we keep configs. This guide assumes you have the same setup as my K3s server, or close enough.

```
mkdir redis-deployment
cd redis-deployment
```

Namespace

I'm going to put everything related to Redis to its own namespace:

```
kubectl create namespace redis-server
```

Persistent storage PVC

Next create a `pvc.yaml` file, and create a claim for 5GB:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: redis-pvc
  namespace: redis-server
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: longhorn
  resources:
    requests:
      storage: 5Gi
```

```
kubectl apply -f pvc.yaml
```

Redis deployment

We have disk space; next, the main deployment. I will use basically the official Redis docker container, since it's produced for arm64, and we do not need to do any special stuff.

Official Docker repository is here: https://hub.docker.com/_/redis/

Create `deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-server
  namespace: redis-server
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis-server
  template:
    metadata:
      labels:
        app: redis-server
        name: redis-server
    spec:
      nodeSelector:
        node-type: worker
      containers:
      - name: redis-server
        image: redis
        args: ["--appendonly", "yes"]
        ports:
          - name: redis-server
            containerPort: 6379
        volumeMounts:
          - name: lv-storage
            mountPath: /data
        env:
            - name: ALLOW_EMPTY_PASSWORD
              value: "yes"
      volumes:
        - name: lv-storage
          persistentVolumeClaim:
            claimName: redis-pvc
```

Nothing special in this deployment; I have explained these values before. The only new argument is:

- ***args*** - which is a way to pass arguments to Docker image. If you look at the official repository, these arguments are the ones that turn on persistence.

```
#Apply
kubectl apply -f deployment.yaml
```

It might take a minute to start, but for me it deployed without issue.

Check:

```
root@control01:/home/ubuntu/redis-kubernetes# kubectl get pods -n redis-server
NAME                            READY   STATUS    RESTARTS   AGE
redis-server-55ddccf85-xfg87   1/1     Running   0          41m
```

Check logs:

```
root@control01:/home/ubuntu/redis-kubernetes# kubectl logs redis-server-55ddccf85-xfg87 -n redis-server
1:C 06 Feb 2021 20:19:53.989 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 06 Feb 2021 20:19:53.989 # Redis version=6.0.10, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 06 Feb 2021 20:19:53.989 # Configuration loaded
1:M 06 Feb 2021 20:19:53.997 * Running mode=standalone, port=6379.
1:M 06 Feb 2021 20:19:53.997 # Server initialized
1:M 06 Feb 2021 20:19:54.004 * Ready to accept connections
```

`Ready to accept connections` is what you want to see 😀.

Service

I like for Redis to have its own IP, so that not only pods in the same namespace can reach it, but everybody around!

You should know by now, that for this I'm using `MetalLB` on my Kubernetes cluster.

Create `service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: redis-server
  namespace: redis-server
spec:
  selector:
    app: redis-server
  type: LoadBalancer
  ports:
    - name: redis-port
      protocol: TCP
      port: 6379
      targetPort: 6379
  loadBalancerIP: 192.168.0.234
```

`192.168.0.234` is the next free IP from my MetalLB IP pool, so this is where my Redis will live.

```
#Apply
kubectl apply -f service.yaml
```

Check if everything is ok:

```
root@control01:/home/ubuntu/redis-kubernetes# kubectl get svc -n redis-server
NAME           TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
redis-server   LoadBalancer   10.43.203.164   192.168.0.234   6379:30771/TCP   41m
```

Looks fine.

/etc/hosts

I'm adding a new entry to every kubernetes node for Redis:

```
echo '192.168.0.234 redis    redis.cube.local' >> /etc/hosts
```

And off you go to every node via Ansible...

```
ansible cube -b -m copy -a "src=/etc/hosts dest=/etc/hosts"
```

Check

```
ubuntu@control01:~$ telnet redis.cube.local 6379
Trying 192.168.0.234...
Connected to redis.
Escape character is '^]'.
```

Done and done. Redis is up and running on Raspberry Pi 4 Kubernetes cluster, Ubuntu arm64. If your mouth is dry like mine now, you will get something to drink and maybe help me to get some liquids too 🙂 (smooth, I know /s).

Last update: October 20, 2021

Comments

Alexander Hagen • a year ago • edited

Thanks for this great article. I managed to get it set-up, except for the final check, where it doesn't get any further than:

```
Microsoft Telnet> open 192.168.12.240 6379
Connecting To 192.168.12.240...
```

Everything looks good to me, so this may have something to do with Traefik.

```
NAME READY STATUS RESTARTS AGE
redis-server-5df7f8964c-4lgh5 1/1 Running 0 108m

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
redis-server LoadBalancer 10.43.27.84 192.168.12.240 6379:32118/TCP 109m

1:C 21 Jun 2021 18:44:28.282 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 21 Jun 2021 18:44:28.282 # Redis version=6.2.4, bits=32, commit=00000000, modified=0, pid=1, just started
1:C 21 Jun 2021 18:44:28.282 # Configuration loaded
1:M 21 Jun 2021 18:44:28.287 * monotonic clock: POSIX clock_gettime
1:M 21 Jun 2021 18:44:28.291 # Warning: 32 bit instance detected but no memory limit set. Setting 3 GB maxmemory limit with 'noeviction' policy now.
1:M 21 Jun 2021 18:44:28.292 * Running mode=standalone, port=6379
```