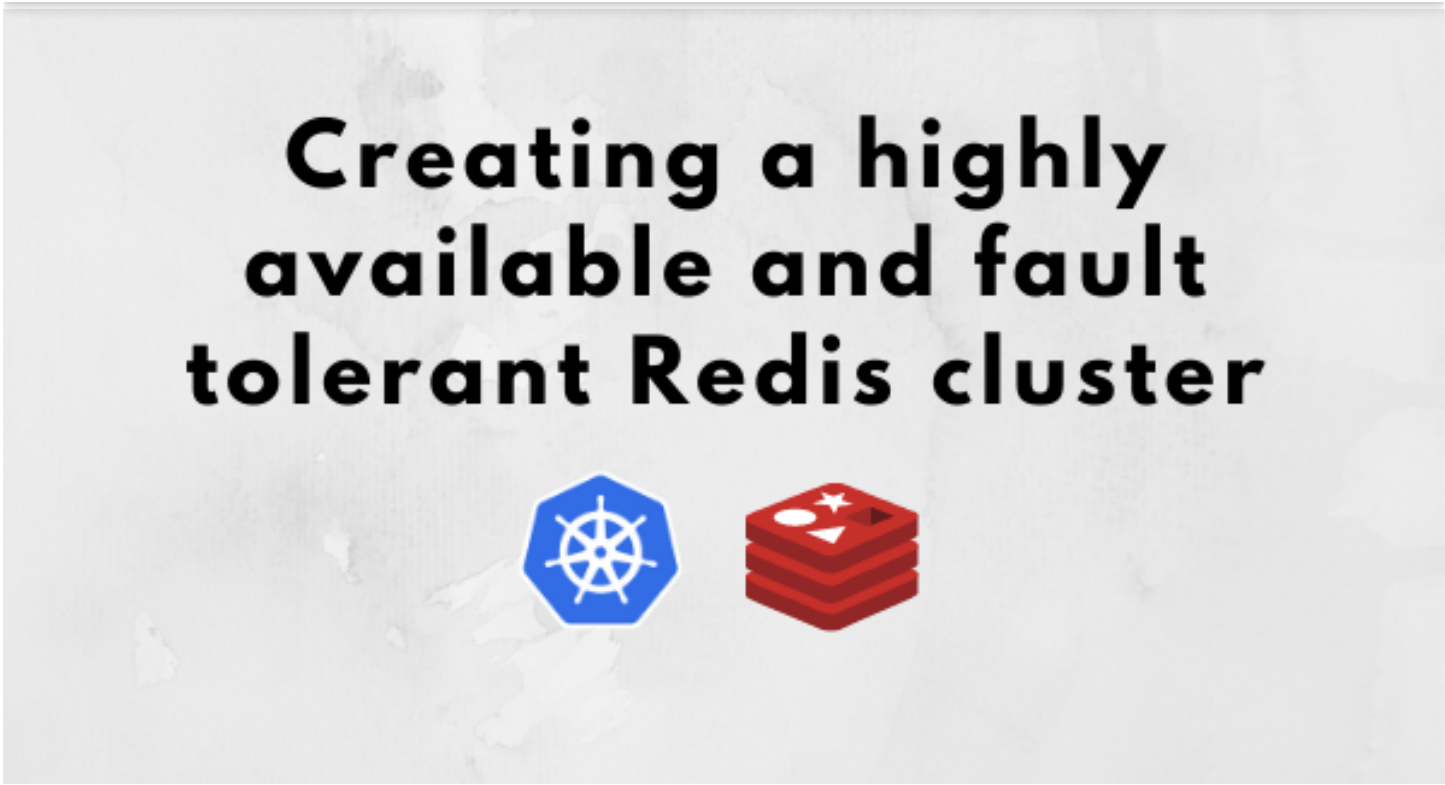
Shishir Khandelwal



Creating a Redis cluster in Kubernetes

Shishir Khandelwal
Engineering @ PayPal | Kubernetes & AWS certified
Published May 19, 2021

+ Follow

Redis is an open source in-memory data structure store. It is commonly used as a database, cache store, and message broker. It supports a variety of data structures such as strings, hashes, lists, sets. In this article, we will explore an approach to create a highly available and fault tolerant Redis cluster in Kubernetes

High availability:- In case one of the pods of redis go down, then other pods should be available to process the commands. The downtime should be minimised. We will use a master-follower setup for this.

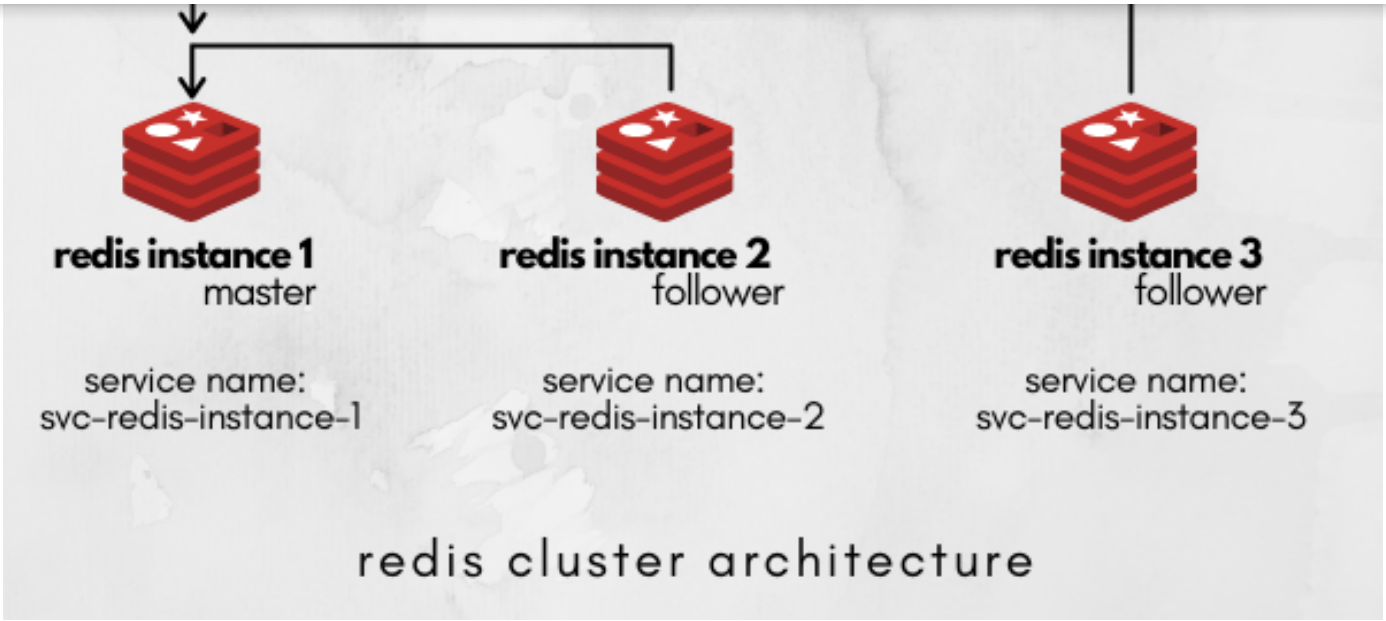
only component which can perform WRITE operations. So in case a master goes down - WRITE operations of the cluster can suffer. To tackle this problem, we will be using Proxies and Sentinels.

Sentinels: Sentinel are components which keep a watch on all redis instances (both master and followers) and keep a record of which redis instance is the master at any given point of time. So in case of any write operation, one must first ask Sentinels about which of the redis instances are master currently. And then send the WRITE operations to that instance. In addition to this, Sentinel also starts failover mechanism for the cluster without any human intervention. (for more info - https://redis.io/topics/sentinel)

Proxies: Application which use redis usually expect one hostname and a port where they can send redis related commands. Since in our present approach, only the sentinel know which of the redis instances is master currently, we use a proxy which will accept commands from your application, ask sentinels about the current master and send the commands to that address. So your application code will require no changes.

Refer to this github repo for the yamls: https://github.com/shishirkh/redis-ha

***Architecture of the redis cluster*:**



- Three redis instances are running as separate deployments. Each of them have their own clusterip service created. All communication between the instances is configured via the clusterip service. This is done so that in case a pod goes down (and it's IP address changes) communication is not affected.

```
--replica-announce-ip <service-name>
```

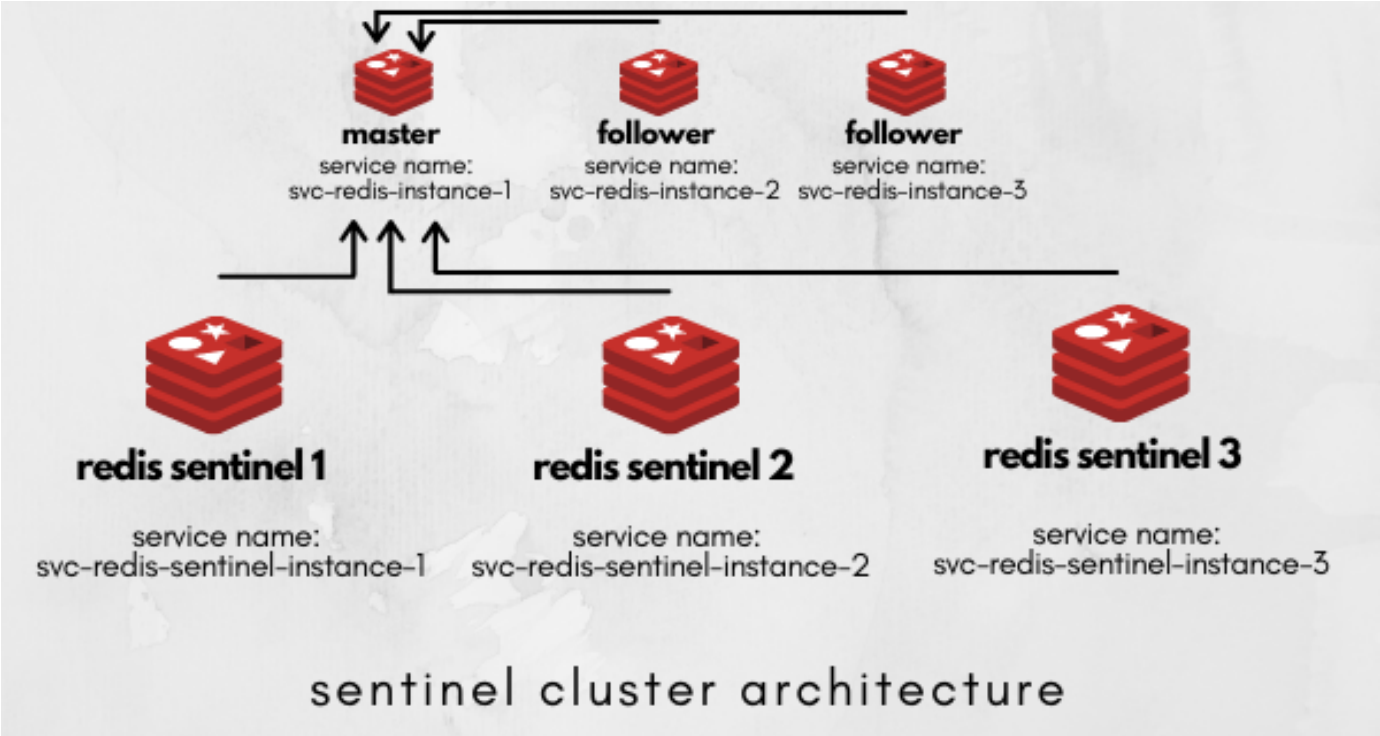
- One of them is assigned as the master initially.

```
--slaveof <master-service-name> 6379
```

- Each of the redis instances need a persistent volume attached at "/dat in order to store the data in case a pod goes down.

```
kubectl apply -f redis/redis-instance-1.yml
kubectl apply -f redis/redis-instance-1.yml
kubectl apply -f redis/redis-instance-3.yml
```

Architecture of the sentinel cluster:



- The architecture of the sentinel cluster is similar to the redis cluster.

them have their own clusterip service created. All communication between the instances is configured via the clusterip service. This is done so that in case a pod goes down (and it's IP address changes) communication is not affected.

About sentinel's configurations:

The way a sentinel works is that to initialize the sentinel, it needs certain things such as the ***redis cluster's master address***. The sentinel's communicate with the master at start to get information about the follower instances of the redis cluster & also about the other sentinels present. The information about the redis cluster master and follower is required in case a failover operation has to be carried out. The information about other sentinel is required so that voting among the sentinels can be carried out.

A special value called the ***Quorum number*** is also required. Quorum number is the minimum no. of votes required before the sentinels can declare any redis master instance as down. Each sentinel has one vote. Ideally, Quorum number should be set as (n+1)/2 where n is the no. of sentinels in the cluster. So, here n is 3. and quorum number should be 2.

There are several other configurations. You can explore all of them here - (http://download.redis.io/redis-stable/sentinel.conf)

Whenever a failover happens, sentinel needs to remember which instance is the new master. The sentinels write this information on their configuration file so that in case their containers are restarted - they still remember who is the current master. To achieve this, several changes need to be done keeping in mind the two possible cases that can occur.

-When sentinel is starting for the first time: *At this time, the configuration file does not exist and has to be created and mounted to a persistent volume. Can we use a configmap to mount the configuration file? No because configmaps attached to a pod cannot be edited from inside the pod. How then can we mount the configuration file on a persistent volume and also keep it editable? We could copy the configuration file to the persistent volume after the persistent volume gets mounted to the pod.*

```
echo "$FILE does not exist."
echo "Copying to destination..."
cp /sentinel-conf-template/sentinel.conf /sentinel-conf-
echo "...file copied. Making file writable..."
chown redis:redis /sentinel-conf-file/sentinel.conf
chmod +x /sentinel-conf-file/sentinel.conf
sed -i "s/\$REDIS_MASTER/$REDIS_MASTER/g" /sentinel-conf
sed -i "s/\$BIND_ADDR/$BIND_ADDR/g" /sentinel-conf-file/
echo "...Done"
```

this time, the configuration file exists already and cannot be overwritten with new content. (because it has information about the current master) For this case, we can simply check. If the file already exists, then we should do nothing.

```
FILE=/sentinel-conf-file/sentinel.conf
if test -f "$FILE"; then
    echo "$FILE exists. Making files writable..."
    chown redis:redis /sentinel-conf-file/sentinel.conf
    chmod +x /sentinel-conf-file/sentinel.conf
    echo "...files are writable now. Done"
```

Combining both the cases to create the sentinel-entrypoint script.

```
#!/bin/sh

echo "Give permissions to the redis-sentinel configurati
chmod -R 0777 $SENTINEL_CONFIG_FILE_DIR
echo "...permissions assigned."

FILE=/sentinel-conf-file/sentinel.conf
if test -f "$FILE"; then
    echo "$FILE exists. Making files writable..."
    chown redis:redis /sentinel-conf-file/sentinel.conf
    chmod +x /sentinel-conf-file/sentinel.conf
    echo "...files are writable now. Done"
else
    echo "$FILE does not exist."
    echo "Copying to destination..."
    cp /sentinel-conf-template/sentinel.conf /sentinel-c
    echo "...file copied. Making file writable..."
    chown redis:redis /sentinel-conf-file/sentinel.conf
    chmod +x /sentinel-conf-file/sentinel.conf
    echo "...files are writable now. Replacing placehold
    sed -i "s/\$SENTINEL_QUORUM/$SENTINEL_QUORUM/g" /sen
```

```
        sed -i "s/\$BIND_ADDR/$BIND_ADDR/g" /sentinel-conf-f
        sed -i "s/\$SENTINEL_SVC_IP/$SENTINEL_SVC_IP/g" /sen
        sed -i "s/\$SENTINEL_SVC_PORT/$SENTINEL_SVC_PORT/g"
        sed -i "s/\$SENTINEL_RESOLVE_HOSTNAMES/$SENTINEL_RES
        sed -i "s/\$SENTINEL_ANNOUNCE_HOSTNAMES/$SENTINEL_AN
        echo "...Done"
    fi


    echo "Starting redis sentinel process now."
    redis-server /sentinel-conf-file/sentinel.conf  --sentin
```

The sentinel.conf looks like this-

```
bind $BIND_ADDR
port 26379
dir /tmp
sentinel monitor mymaster $REDIS_MASTER 6379 $SENTINEL_Q
sentinel down-after-milliseconds mymaster $SENTINEL_DOWN
sentinel parallel-syncs mymaster 1
sentinel failover-timeout mymaster $SENTINEL_FAILOVER
sentinel announce-ip $SENTINEL_SVC_IP
sentinel announce-port $SENTINEL_SVC_PORT
sentinel resolve-hostnames $SENTINEL_RESOLVE_HOSTNAMES
sentinel announce-hostnames $SENTINEL_ANNOUNCE_HOSTNAMES
```

To build the sentinel image:

```
cd /sentinel/docker
docker build -t <image name> .
```

- Each of the sentinel instances need a persistent volume attached at "/sentinel-conf-file/" in order to store the data in case a pod goes dow

```
kubectl apply -f redis/redis-sentinel-1.yml
kubectl apply -f redis/redis-sentinel-2.yml
kubectl apply -f redis/redis-sentinel-3.yml
```

Testing out the sentinel cluster:

- Identify the master in the redis cluster using these commands.

```
kubectl exec -it deployment/deployment-redis-instance-1

kubectl exec -it deployment/deployment-redis-instance-2

kubectl exec -it deployment/deployment-redis-instance-3
```

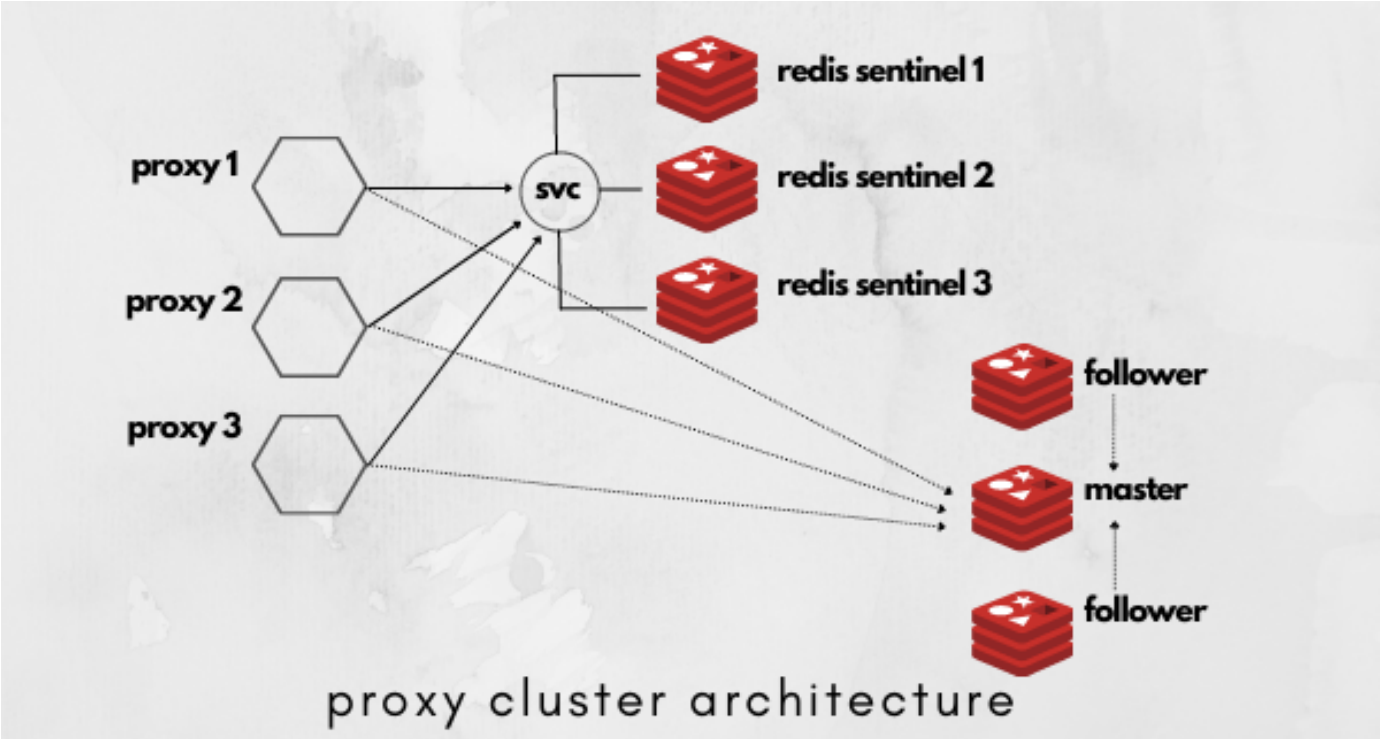
- Verify if each of the sentinels also identify the same master.

```
kubectl exec -it deployment/deployment-redis-sentinel-in

kubectl exec -it deployment/deployment-redis-sentinel-in

kubectl exec -it deployment/deployment-redis-sentinel-in
```

Architecture of the proxy cluster:



- The way the proxy would work is that it will ask any sentinel about which redis instance is the master. It will then redirect all the redis operations to that master.

- As it can communicate with any sentinel, we use a headless service which has all the redis sentinels as endpoints.

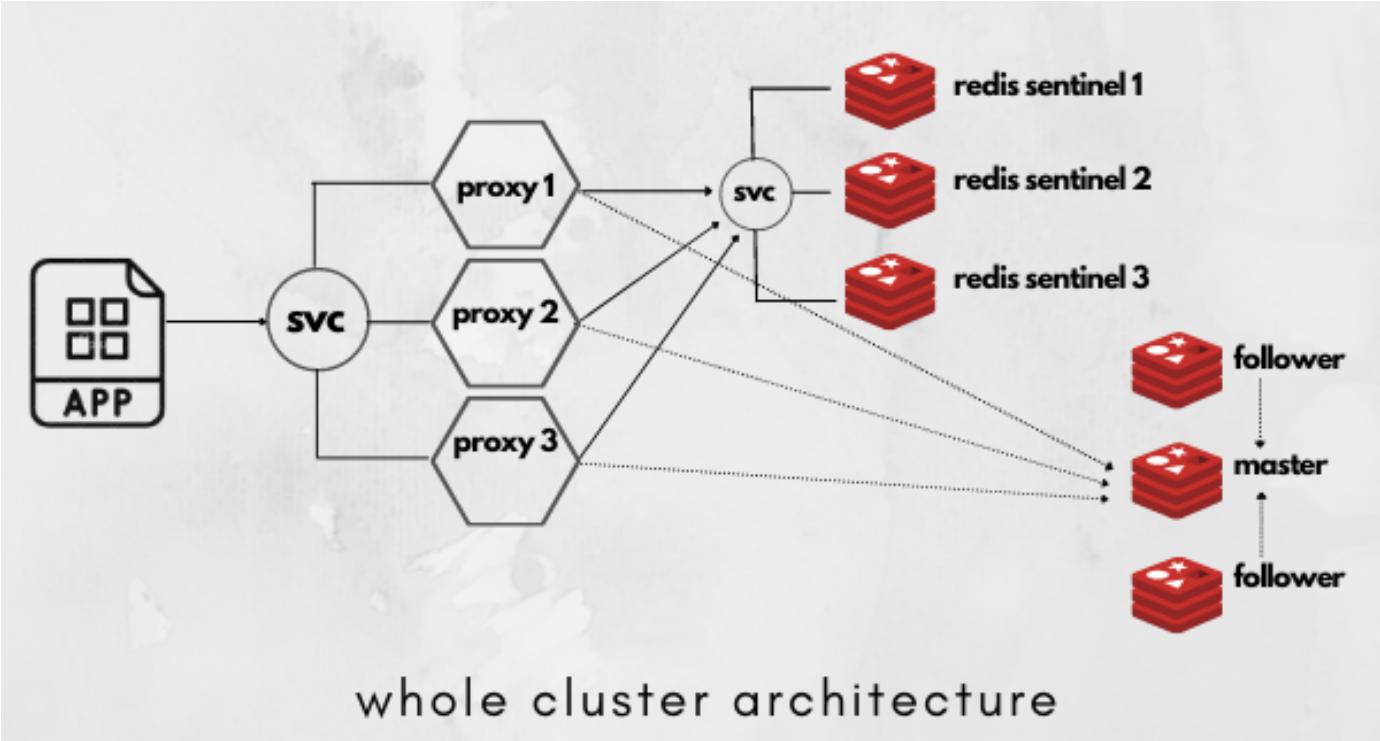
- As each instance of a proxy has the same purpose and to ensure high availability of the proxies - we create multiple instances of the proxy We create a headless service with proxies as the endpoints. This is th service which the application is going to use for redis related operations.

```
kubectl apply -f proxy/proxy.yml
```

Testing out the proxy:

```
-cli -h svc-redis-sentinel-proxy -p 6379 -c info replicatio
```

Finally, the setup should look something like this...Our highly available and fault tolerant redis cluster is ready.



I hope you learnt something. Keep exploring.

Shishir Khandelwal

Richard Cunningham 4w ···

Really nice article, kudos to you and thanks for sharing. A question on the proxy, is this actually needed if our Redis clients have Sentinel support? Or does it add something else?

Like Reply

Michael Falade 4mo ···

Thanks for this helpful article. Pls what URL would a client point to to access the cluster?

Like Reply

John Ruzick 4mo ···

If you're accessing the Redis Cluster from an application running inside Kubernetes, you could reach the cluster via the proxy by just using the proxy service name:port (svc-redis-sentinel-proxy:6379). If you're trying to connect from outside of K8S you would have to setup external access using some type of Ingress setup. Hope this helps!

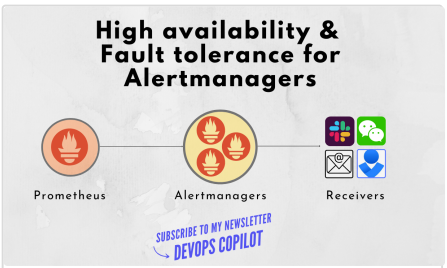
Like Reply | 1 Like

See more comments

To view or add a comment, sign in

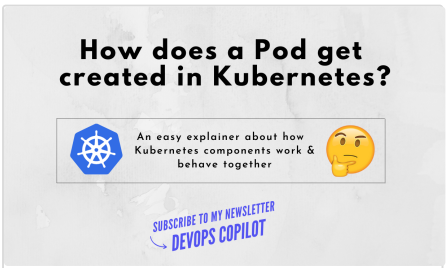
More articles by this author

See all



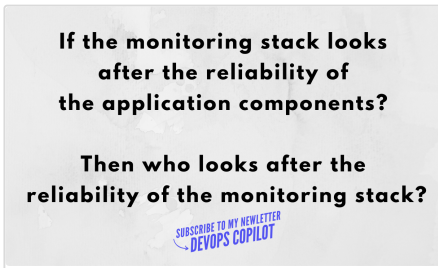
High availability & Fault tolerance fo...

Jan 7, 2022



How does a pod get created in...

Jan 4, 2022



High Availability & Fault Tolerance f...

Dec 27, 2021