Redis Cluster Deployment in GKE

Redis Cluster Deployment

The approach used to deploy the Redis cluster in GKE is done by using the spothome Redis operator.
GitHub - spotahome/redis-operator: Redis Operator creates/configures/manages high availability redis with sentinel automatic failover at op Kubernetes.

Pre-Requisites:

1. Access to GKE namespace
2. Access to GKE Artifact Registry
3. kubectl installed
4. Helm installed
5. docker installed optional
6. Redis Operator Installed by GKE Admin
7. Cilium Policy to allow traffic from Redis-Operator namespace to oc360-loyalty namespace (where redis cluster will be deployed)

Redis Cluster in oc360-loyalty Namespace

a. create a secret for Redis auth with the name redis-auth in the application namespace:
please replace “pass“ with a strong password.

```
echo -n "pass" > password
kubectl create secret generic redis-auth --from-file=password -n oc360-loyalty
```

b. create a cilium policy to allow traffic between redis-operator namespace and the app namespace that will container the redis-cluster

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: allow-redis-loyalty-dev
spec:
  endpointSelector:
    matchLabels: {}
  ingress:
    - fromEndpoints:
        - matchLabels:
            io.kubernetes.pod.namespace: redis-operator-loyalty
  egress:
    - toEndpoints:
        - matchLabels:
            io.kubernetes.pod.namespace: redis-operator-loyalty
```

c. create the RedisFailover resource with the correct secret name in line 27 as created in the previous step.

```
apiVersion: databases.spotahome.com/v1
kind: RedisFailover
metadata:
  name: redisfailover
```

```
spec:
  sentinel:
    replicas: 3
    image: us-east4-docker.pkg.dev/dbce-c360-loyalty-dev-4bcf/oc360-loyalty-dev-artifact-registry/redis:6.
    resources:
      requests:
        cpu: 100m
        memory: 100Mi
      limits:
        cpu: 400m
        memory: 100Mi
  redis:
    replicas: 3
    image: us-east4-docker.pkg.dev/dbce-c360-loyalty-dev-4bcf/oc360-loyalty-dev-artifact-registry/redis:6.
    resources:
      requests:
        cpu: 100m
        memory: 500Mi
      limits:
        cpu: 400m
        memory: 2000Mi
  auth:
    secretPath: redis-auth
```

- Please note that the redis image redis:6.2.6-alpine needs to be in GKE AR as per P&G policies enforced on the GKE Cluster.

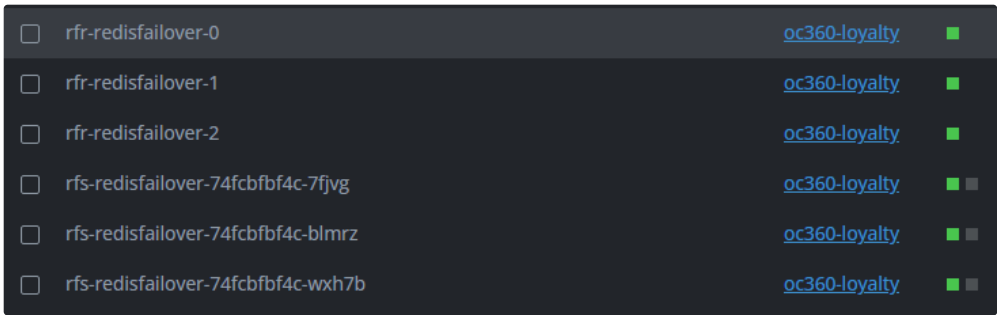
```

docker pull redis:6.2.6-alpine

docker tag redis:6.2.6-alpine us-east4-docker.pkg.dev/dbce-c360-loyalty-dev-4bcf/oc360-loyalty-dev-artifact-re
docker push us-east4-docker.pkg.dev/dbce-c360-loyalty-dev-4bcf/oc360-loyalty-dev-artifact-registry/redis:6.2.6

```

This will create a stateful set with 3 pods with redis-cluster, and a deployment with 3 replicas of sentinel for monitoring and failover, make sure the pods are up and running.



- Once the pods are up and running exec into any sentinel or redis pod (redis-cli is required to test with routes to the redis/sentinel service on ports 6379 & 26379)

```
kubectl exec -i -t -n oc360-loyalty rfs-redisfailover-74fcbfbf4c-7fjvg -c sentinel -- sh -c "clear; (bash || ash
```

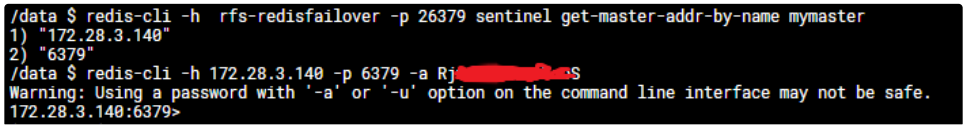
- Test the redis-cluster using the following commands.

```
redis-cli -h  rfs-redisfailover -p 26379 sentinel get-master-addr-by-name mymaster
```

a. -h is the service name of sentinel service and port 26379 where it listens.

b. This command will get the current redis master node IP address so we can connect to it and try to write a key: value pair in cache.

```
redis-cli -h 172.28.3.140 -p 6379 -a “redisauth password“
```



- **Set a key**: This command sets the value of a key

  ```
  SET mykey "Hello Redis"
  ```

- **Get a key**: This command gets the value of a key

  ```
  GET mykey
  ```

  It should return "Hello Redis".

Brief Overview About Sentinel Service and Redis Cluster

Sentinel

The `Sentinel` section configures Redis Sentinel, which is a system designed to help manage Redis instances by providing high availability and failover capabilities. Sentinel performs several critical roles:

- **Monitoring**: Sentinel constantly checks if your Redis instances are functioning correctly and reachable.
- **Notification**: In the event of issues, Sentinel can notify administrators or other systems through various mechanisms, such as APIs or emails, about the problem.
- **Automatic Failover**: Perhaps most importantly, if a Redis master node fails, Sentinel will automatically initiate a failover process by promoting one of the replicas to be the new master. This ensures minimal disruption and downtime.
- **Configuration Provider**: Sentinel also acts as a source of truth for clients about which Redis node is currently the master, helping clients to direct their write operations correctly.
- **Quorum and Consensus**: You configure the number of Sentinels that need to agree on a detected failure before a failover is executed. This prevents false positives and ensures that failover decisions are made reliably.

Redis

The `Redis` section of the `RedisFailover` resource specifies the configuration related to the Redis server instances themselves. Here are the key aspects:

- **Replicas**: This specifies the number of Redis server pods that should be running. These replicas handle the actual data storage and processing tasks of your Redis deployment.
- **Resources**: You can specify the amount of CPU and memory resources. This ensures that each Redis instance has adequate resources to perform efficiently.
- **Persistence**: You can configure persistence settings to ensure that data is not lost when pods are restarted or when failures occur. This often involves setting up persistent volume claims that the Redis pods use to store data. by default we use EmptyDir concept of Kubernetes to store cache data in pods

By defining these components in a `RedisFailover` resource, the Redis Operator can deploy a Redis environment that is resilient to failures and capable of handling production workloads with the expected reliability and performance.