Argo CD - Declarative GitOps CD for Kubernetes

Argo CD is largely stateless. All data is persisted as Kubernetes objects, which in turn is stored in Kubernetes' etcd. Redis is only used as a throw-away cache and can be lost. When lost, it will be rebuilt without loss of service.

NOTE: The HA installation will require at least three different nodes due to pod anti-affinity roles in the specs. Additionally, IPv6 only clusters are not supported.

Implementation:

Below are the components specific to Argo CD HA:

```
shaik [ ~ ]$ kubectl get pods -n argocd -o wide
NAME                                               READY   STATUS    RESTARTS   AGE   IP            NODE
argocd-application-controller-0                    1/1     Running   0          31m   10.244.1.17   aks-agentpool-57942878-vmss000003
argocd-applicationset-controller-54c565cb6b-4pmv7  1/1     Running   0          31m   10.244.1.11   aks-agentpool-57942878-vmss000003
argocd-dex-server-56d48d4bcf-52wtt                 1/1     Running   0          31m   10.244.1.12   aks-agentpool-57942878-vmss000003
argocd-notifications-controller-54d7bdc957-9s2fm   1/1     Running   0          31m   10.244.1.13   aks-agentpool-57942878-vmss000003
argocd-redis-ha-haproxy-d4496b596-69zmp            1/1     Running   0          31m   10.244.2.24   aks-agentpool-57942878-vmss000000
argocd-redis-ha-haproxy-d4496b596-pfdgl            1/1     Running   0          31m   10.244.0.35   aks-agentpool-57942878-vmss000001
argocd-redis-ha-haproxy-d4496b596-q579x            1/1     Running   0          31m   10.244.1.15   aks-agentpool-57942878-vmss000003
argocd-redis-ha-server-0                           3/3     Running   0          31m   10.244.1.18   aks-agentpool-57942878-vmss000003
argocd-redis-ha-server-1                           3/3     Running   0          30m   10.244.2.27   aks-agentpool-57942878-vmss000000
argocd-redis-ha-server-2                           3/3     Running   0          29m   10.244.0.36   aks-agentpool-57942878-vmss000001
argocd-repo-server-54bb99f876-4sxpk                1/1     Running   0          31m   10.244.1.14   aks-agentpool-57942878-vmss000003
argocd-repo-server-54bb99f876-7wvbd                1/1     Running   0          31m   10.244.2.25   aks-agentpool-57942878-vmss000000
argocd-server-c9ddd9797-fjnrz                      1/1     Running   0          31m   10.244.2.26   aks-agentpool-57942878-vmss000000
argocd-server-c9ddd9797-thfxf                      1/1     Running   0          31m   10.244.1.16   aks-agentpool-57942878-vmss000003
```

As highlighted, redis pods are running in different nodes as pod anti-affinity is being used for HA purpose.

Redis Cache:

Redis is used by Argo CD as a throwaway cache, meaning the system still works if redis is not responding/not functioning properly. Redis is highly recommended component.

Because the manifests generated from a git repository will be kept in redis cache, so if redis is missing, they will have to be recreated at every sync request. If the cache is lost, everything needs to be recreated which will results in performance issues.(The application still works)

HA Mode:

The HA setup comes with a StatefulSet - Three replicas for Redis

One Master & Two Slaves. It also comes with HA Proxy deployment that sits infront of Redis.

```
shaik [ ~ ]$ kubectl get pods -n argocd | grep -i redis
argocd-redis-ha-haproxy-d4496b596-69zmp             1/1     Running   0
argocd-redis-ha-haproxy-d4496b596-pfdgl             1/1     Running   0
argocd-redis-ha-haproxy-d4496b596-q579x             1/1     Running   0
argocd-redis-ha-server-0                            3/3     Running   0
argocd-redis-ha-server-1                            3/3     Running   0
argocd-redis-ha-server-2                            3/3     Running   0
```

If the Redis master is down/not functioning properly, then one of the slaves is promoted as the new master and HAProxy will make this transparent to the client applications.

Let's see how master-slave works in redis:

```
shaik [ ~ ]$ kubectl logs argocd-redis-ha-server-0 -c redis  -n argocd
1:C 29 May 2023 07:47:57.184 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 29 May 2023 07:47:57.184 # Redis version=7.0.11, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 29 May 2023 07:47:57.184 # Configuration loaded
1:M 29 May 2023 07:47:57.185 * monotonic clock: POSIX clock_gettime
1:M 29 May 2023 07:47:57.185 * Running mode=standalone, port=6379.
1:M 29 May 2023 07:47:57.185 # Server initialized
1:M 29 May 2023 07:47:57.186 * Ready to accept connections
1:M 29 May 2023 07:48:59.135 * Replica 10.0.25.220:6379 asks for synchronization
1:M 29 May 2023 07:48:59.135 * Full resync requested by replica 10.0.25.220:6379
1:M 29 May 2023 07:48:59.135 * Replication backlog created, my new replication IDs are 'f5936d9c489e2cf341f75b48235c72981f770077' and '0000000000000000000000000000000000000000'
1:M 29 May 2023 07:48:59.135 * Delay next BGSAVE for diskless SYNC
1:M 29 May 2023 07:49:04.507 * Starting BGSAVE for SYNC with target: replicas sockets
1:M 29 May 2023 07:49:04.508 * Background RDB transfer started by pid 42
42:C 29 May 2023 07:49:04.509 * Fork CoW for RDB: current 0 MB, peak 0 MB, average 0 MB
1:M 29 May 2023 07:49:04.509 # Diskless rdb transfer, done reading from pipe, 1 replicas still up.
1:M 29 May 2023 07:49:04.519 * Background RDB transfer terminated with success
1:M 29 May 2023 07:49:04.519 * Streamed RDB transfer with replica 10.0.25.220:6379 succeeded (socket). Waiting for REPLCONF ACK from slave to enable streaming
1:M 29 May 2023 07:49:04.519 * Synchronization with replica 10.0.25.220:6379 succeeded
1:M 29 May 2023 07:49:59.053 * Replica 10.0.115.31:6379 asks for synchronization
1:M 29 May 2023 07:49:59.053 * Full resync requested by replica 10.0.115.31:6379
1:M 29 May 2023 07:49:59.053 * Delay next BGSAVE for diskless SYNC
1:M 29 May 2023 07:50:04.851 * Starting BGSAVE for SYNC with target: replicas sockets
1:M 29 May 2023 07:50:04.852 * Background RDB transfer started by pid 99
99:C 29 May 2023 07:50:04.853 * Fork CoW for RDB: current 0 MB, peak 0 MB, average 0 MB
1:M 29 May 2023 07:50:04.853 # Diskless rdb transfer, done reading from pipe, 1 replicas still up.
1:M 29 May 2023 07:50:04.867 * Background RDB transfer terminated with success
1:M 29 May 2023 07:50:04.867 * Streamed RDB transfer with replica 10.0.115.31:6379 succeeded (socket). Waiting for REPLCONF ACK from slave to enable streaming
1:M 29 May 2023 07:50:04.867 * Synchronization with replica 10.0.115.31:6379 succeeded
```

As highlighted, two replicas are looking for synchronization:

Replica **10.0.25.220**:6379 asks for synchronization
Replica **10.0.115.31**:6379 asks for synchronization

Let's exec into one of the redis cache pod: argocd-redis-ha-server-0
slave-announce-ip 10.0.202.28 - Master Details

```
shaik [ ~ ]$ kubectl exec -it argocd-redis-ha-server-0 -n argocd -- sh
Defaulted container "redis" out of: redis, sentinel, split-brain-fix, config-init (init)
/data $ ls
conf
/data $ cd conf/
/data/conf $ cat redis.conf
dir "/data"
port 6379
rename-command FLUSHDB ""
rename-command FLUSHALL ""
bind 0.0.0.0
maxmemory 0
maxmemory-policy volatile-lru
min-replicas-max-lag 5
min-replicas-to-write 1
rdbchecksum yes
rdbcompression yes
repl-diskless-sync yes
save ""

slave-announce-port 6379
slave-announce-ip 10.0.202.28
```

Check logs of argocd-redis-ha-server-1 pod:
The master details : Connecting to MASTER 10.0.202.28:6379

```
shaik [ ~ ]$ kubectl logs argocd-redis-ha-server-1 -c redis  -n argocd
1:C 29 May 2023 07:48:59.103 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 29 May 2023 07:48:59.103 # Redis version=7.0.11, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 29 May 2023 07:48:59.103 # Configuration loaded
1:S 29 May 2023 07:48:59.103 * monotonic clock: POSIX clock_gettime
1:S 29 May 2023 07:48:59.111 * Running mode=standalone, port=6379.
1:S 29 May 2023 07:48:59.111 # Server initialized
1:S 29 May 2023 07:48:59.118 * Ready to accept connections
1:S 29 May 2023 07:48:59.118 * Connecting to MASTER 10.0.202.28:6379
1:S 29 May 2023 07:48:59.118 * MASTER <-> REPLICA sync started
1:S 29 May 2023 07:48:59.120 * Non blocking connect for SYNC fired the event.
1:S 29 May 2023 07:48:59.121 * Master replied to PING, replication can continue...
1:S 29 May 2023 07:48:59.123 * Partial resynchronization not possible (no cached master)
1:S 29 May 2023 07:49:04.496 * Full resync from master: f5936d9c489e2cf341f75b48235c72981f770077:563
1:S 29 May 2023 07:49:04.497 * MASTER <-> REPLICA sync: receiving streamed RDB from master with EOF to disk
1:S 29 May 2023 07:49:04.497 * MASTER <-> REPLICA sync: Flushing old data
1:S 29 May 2023 07:49:04.497 * MASTER <-> REPLICA sync: Loading DB in memory
1:S 29 May 2023 07:49:04.507 * Loading RDB produced by version 7.0.11
1:S 29 May 2023 07:49:04.507 * RDB age 0 seconds
1:S 29 May 2023 07:49:04.507 * RDB memory usage when created 1.16 Mb
1:S 29 May 2023 07:49:04.507 * Done loading RDB, keys loaded: 0, keys expired: 0.
1:S 29 May 2023 07:49:04.507 * MASTER <-> REPLICA sync: Finished with success
```

Logs of argocd-redis-ha-server-2:

```
shaik [ ~ ]$ kubectl logs argocd-redis-ha-server-2 -c redis  -n argocd
1:C 29 May 2023 07:49:59.029 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 29 May 2023 07:49:59.029 # Redis version=7.0.11, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 29 May 2023 07:49:59.029 # Configuration loaded
1:S 29 May 2023 07:49:59.029 * monotonic clock: POSIX clock_gettime
1:S 29 May 2023 07:49:59.040 * Running mode=standalone, port=6379.
1:S 29 May 2023 07:49:59.040 # Server initialized
1:S 29 May 2023 07:49:59.047 * Ready to accept connections
1:S 29 May 2023 07:49:59.047 * Connecting to MASTER 10.0.202.28:6379
1:S 29 May 2023 07:49:59.047 * MASTER <-> REPLICA sync started
1:S 29 May 2023 07:49:59.049 * Non blocking connect for SYNC fired the event.
1:S 29 May 2023 07:49:59.050 * Master replied to PING, replication can continue...
1:S 29 May 2023 07:49:59.051 * Partial resynchronization not possible (no cached master)
1:S 29 May 2023 07:50:04.850 * Full resync from master: f5936d9c489e2cf341f75b48235c72981f770077:10678
1:S 29 May 2023 07:50:04.852 * MASTER <-> REPLICA sync: receiving streamed RDB from master with EOF to disk
1:S 29 May 2023 07:50:04.852 * MASTER <-> REPLICA sync: Flushing old data
1:S 29 May 2023 07:50:04.852 * MASTER <-> REPLICA sync: Loading DB in memory
1:S 29 May 2023 07:50:04.864 * Loading RDB produced by version 7.0.11
1:S 29 May 2023 07:50:04.864 * RDB age 0 seconds
1:S 29 May 2023 07:50:04.865 * RDB memory usage when created 1.27 Mb
1:S 29 May 2023 07:50:04.865 * Done loading RDB, keys loaded: 1, keys expired: 0.
1:S 29 May 2023 07:50:04.865 * MASTER <-> REPLICA sync: Finished with success
```

From the above screenshots, we can easily conclude:(All three replicas are in sync as shown above)

Master Node:

argocd-redis-ha-server-0

Slave Nodes:

argocd-redis-ha-server-1
argocd-redis-ha-server-2

Redis cache plays an important role as the maintaining manifest/manifest generation is critical operation and Argo CD tries to save the manifest in Redis instance.

As mentioned before, if the cache fails it can be recalculated, but do expect a performance issues.

Kubescape:

“**Kubescape: A CNCF Sandbox Platform for All Kubernetes Security**”

Scanning - Kubescape:

Image: quay.io/argoproj/argocd:latest

STAT.	SCAN TIME	CLUSTER	NAMESPACE	WORKLOAD	CONTAINER NAME	REGISTRY	IMAGE TAG	SEVERITY
	May 29, 2023 13:03:03	zoh-poc-aks	argocd	deployment-argocd-notifications-controller	argocd-notifications-controller	quay.io	quay.io/argoproj/argocd:latest	3 6 15 26 5 0
	May 29, 2023 13:03:05	zoh-poc-aks	argocd	statefulset-argocd-application-controller	argocd-application-controller	quay.io	quay.io/argoproj/argocd:latest	3 6 15 26 5 0
	May 29, 2023 13:03:03	zoh-poc-aks	argocd	deployment-argocd-repo-server	argocd-repo-server	quay.io	quay.io/argoproj/argocd:latest	3 6 15 26 5 0
	May 29, 2023 13:03:07	zoh-poc-aks	argocd	deployment-argocd-applicationset-controller	argocd-applicationset-controller	quay.io	quay.io/argoproj/argocd:latest	3 6 15 26 5 0
	May 29, 2023 13:03:08	zoh-poc-aks	argocd	deployment-argocd-server	argocd-server	quay.io	quay.io/argoproj/argocd:latest	3 6 15 26 5 0
	May 29, 2023 13:03:03	zoh-poc-aks	argocd	deployment-argocd-dex-server	dex	ghcr.io	ghcr.io/dexidp/dex:v2.36.0	3 4 6 0 0 0
	May 29, 2023 13:03:06	zoh-poc-aks	argocd	statefulset-argocd-redis-ha-server	redis		redis:7.0.11-alpine	2 0 0 1 0 0
	May 29, 2023 13:03:06	zoh-poc-aks	argocd	statefulset-argocd-redis-ha-server	sentinel		redis:7.0.11-alpine	2 0 0 1 0 0
	May 29, 2023 13:03:06	zoh-poc-aks	argocd	statefulset-argocd-redis-ha-server	split-brain-fix		redis:7.0.11-alpine	2 0 0 1 0 0
	May 29, 2023 13:03:05	zoh-poc-aks	argocd	deployment-argocd-redis-ha-haproxy	haproxy		haproxy:2.6.12-alpine	0 2 4 0 0 0

Components:

```
shaik [ ~ ]$ kubectl get pods -n kubescape
NAME                              READY   STATUS    RESTARTS   AGE
gateway-67bb6fb7c4-xzlgs          1/1     Running   0          27m
kollector-0                       1/1     Running   0          27m
kubescape-7f998f9b9d-zjhkf        1/1     Running   0          27m
kubevuln-7d6b7c5d4d-ztbnh         1/1     Running   0          27m
operator-56b5d74578-x22nx         1/1     Running   0          27m
otel-collector-54cdb99686-b6mks   1/1     Running   0          27m
shaik [ ~ ]$ kubectl get deploy -n kubescape
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
gateway          1/1     1            1           27m
kubescape        1/1     1            1           27m
kubevuln         1/1     1            1           27m
operator         1/1     1            1           27m
otel-collector   1/1     1            1           27m
```