**Redis Cluster: Architecture, Replication, Sharding and Failover**

*“A query that used to take an hour can run in seconds on cache”.*

Redis Cluster is an active-passive cluster implementation that consists of master and slave nodes. The cluster uses hash partitioning to split the keyspace into 16,384 key slots, with each master responsible for a subset of those slots.

Each slave replicates a specific master and can be reassigned to replicate another master or be elected to a master node as needed.

## Ports Communication

Each node in a cluster requires two TCP ports.

* **One port is used for client connections and communications.** This is the port you would configure into client applications or command-line tools.
* **The second required port is reserved for node-to-node communication(gossip port)** that occurs in a binary protocol and allows the nodes to discuss configuration and node availability.

## Failover

When a master fails or is found to be unreachable by the majority of the cluster as determined by the nodes communication via the gossip port, the remaining masters hold a vote and elect one of the failing masters’ slaves to take its place.

## Rejoining The Cluster

When the failing master eventually rejoins the cluster, it will join as a slave and begin to replicate another master.

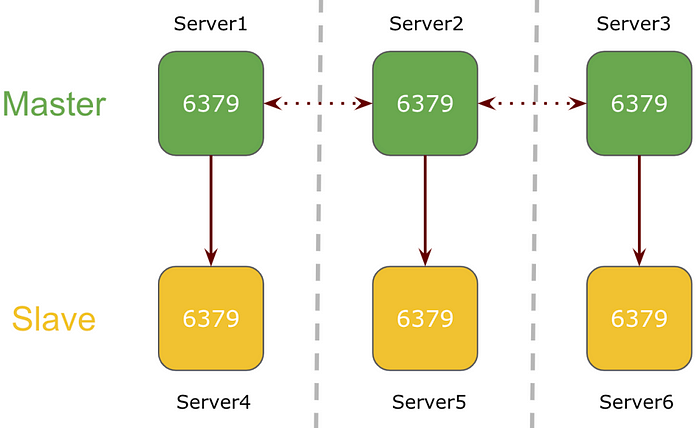
## Sharding

Redis sharded data automatically into the servers.  
Redis has a concept of the **hash slot** in order to split data. All the data are divided into slots.  
There are 16384 slots. These slots are divided by the number of servers.

If there are 3 servers; A, B and C then

* Server 1 contains hash slots from 0 to 5500.
* Server 2 contains hash slots from 5501 to 11000.
* Server 3 contains hash slots from 11001 to 16383.

**6 Node M/S Cluster**

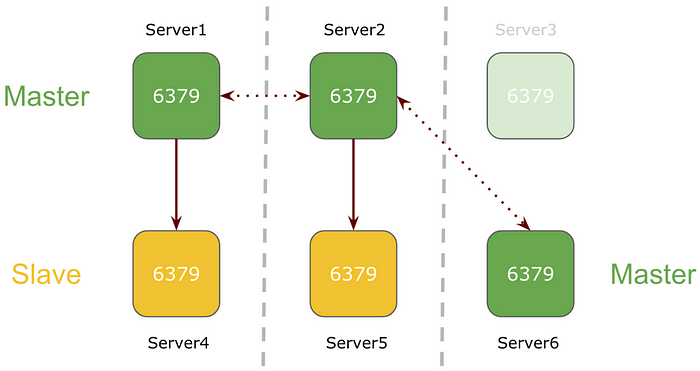


In a 6 node cluster mode, 3 nodes will be serving as a master and the 3 nodes will be their respective slave.

Here, Redis service will be running on port 6379 on all servers in the cluster. Each master server is replicating the keys to its respective Redis slave node assigned during the cluster creation process.

**WHAT IF Redis Goes Down**

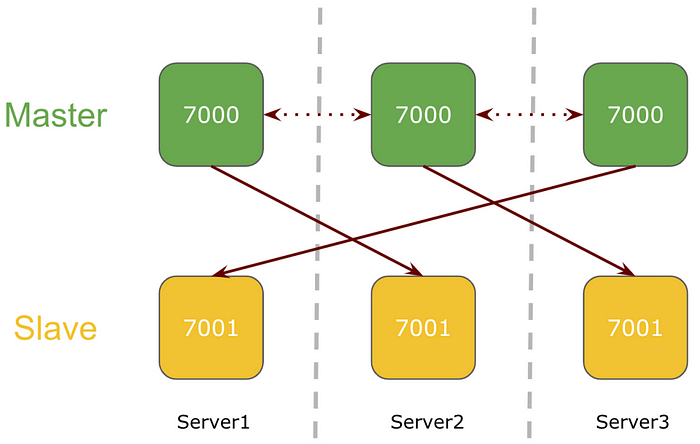
**1 node goes down in a 6 node Redis Cluster**



If one of the nodes goes down in the Redis 6-node cluster setup, its respective slave will be promoted as the master.

In the above example, master Server3 goes down and its slave Server6 is promoted as the master.

**3 Node M/S Cluster**



In a 3 node cluster mode, there will be 2 redis services running on each server on different ports. All 3 nodes will be serving as a master with redis slave on cross nodes.

**1 node goes down in a 3 node Redis Cluster**

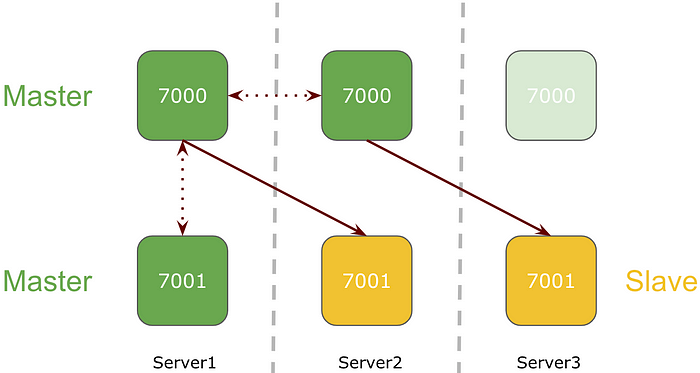
A diagram of a server

Description automatically generated

If one of the nodes goes down in the Redis 3-node cluster setup, its respective slave running on the separate node will be promoted to master.

In the above example, Server 3 goes down and slave running on Server1 is promoted to master.

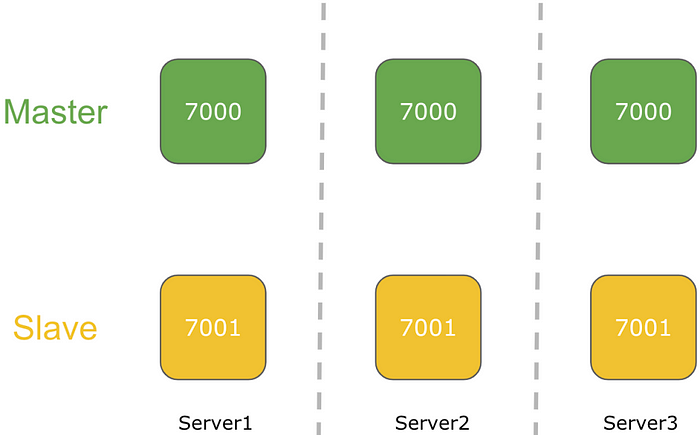
**Redis service goes down on one of the 3 node Redis Cluster**



If Redis service goes down on one of the nodes in Redis 3-node cluster setup, its respective slave will be promoted as the master.

**3-Node Cluster Prerequisites**

While setting up Redis cluster on 3 nodes, I will be following the strategy of having 3 master nodes and 3 slave nodes with one master and one slave running on each node serving redis at different ports. As shown in the diagram Redis service is running on Port 7000 and Port 7001



* 7000 port will serve Redis Master
* 7001 port will serve Redis Slave

**Directory Structure**

We need to design the directory structure to server both Redis configurations.

tree /etc/redis/etc/redis  
`-- cluster  
 |-- 7000  
 | `-- redis\_7000.conf  
 `-- 7001  
 `-- redis\_7001.conf

**Redis Configuration**

Configuration file for Redis service 1

cat /etc/redis/cluster/7000/redis\_7000.confport 7000  
dir /var/lib/redis/7000/  
appendonly yes  
protected-mode no  
cluster-enabled yes  
cluster-node-timeout 5000  
cluster-config-file /etc/redis/cluster/7000/nodes\_7000.conf  
pidfile /var/run/redis\_7000.pid

Configuration file for Redis service 2

cat /etc/redis/cluster/7001/redis\_7001.confport 7001  
dir /var/lib/redis/7001  
appendonly yes  
protected-mode no  
cluster-enabled yes  
cluster-node-timeout 5000  
cluster-config-file /etc/redis/cluster/7001/nodes\_7001.conf  
pidfile /var/run/redis\_7001.pid

**Redis Service File**

As we are managing multiple services on a single instance, we need to update the service file for easier management of Redis services.

Service management file for Redis service 1

cat /etc/systemd/system/redis\_7000.service[Unit]  
Description=Redis persistent key-value database  
After=network.target[Service]  
ExecStart=/usr/bin/redis-server /etc/redis/cluster/7000/redis\_7000.conf --supervised systemd  
ExecStop=/bin/redis-cli -h 127.0.0.1 -p 7000 shutdown  
Type=notify  
User=redis  
Group=redis  
RuntimeDirectory=redis  
RuntimeDirectoryMode=0755  
LimitNOFILE=65535[Install]  
WantedBy=multi-user.target

Service management file for Redis service 2

cat /etc/systemd/system/redis\_7001.service[Unit]  
Description=Redis persistent key-value database  
After=network.target[Service]  
ExecStart=/usr/bin/redis-server /etc/redis/cluster/7001/redis\_7001.conf --supervised systemd  
ExecStop=/bin/redis-cli -h 127.0.0.1 -p 7001 shutdown  
Type=notify  
User=redis  
Group=root  
RuntimeDirectory=/etc/redis/cluster/7001  
RuntimeDirectoryMode=0755  
LimitNOFILE=65535[Install]  
WantedBy=multi-user.target

**Redis Service Status**

**Master Service**

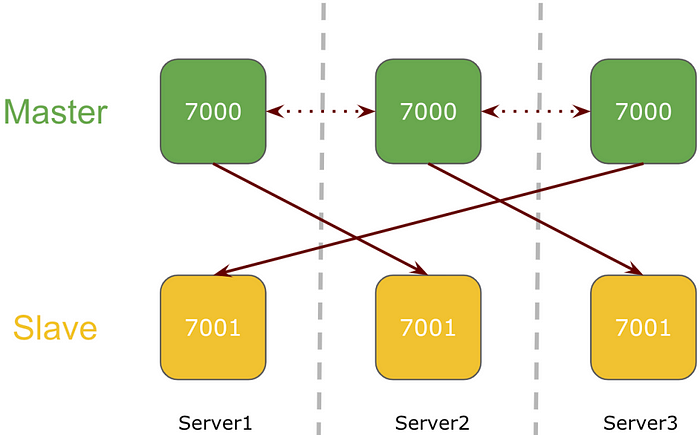
systemctl status redis\_7000.service● redis\_7000.service - Redis persistent key-value database  
 Loaded: loaded (/etc/systemd/system/redis\_7000.service; enabled; vendor preset: disabled)  
 Active: active (running) since Wed 2019-09-25 08:14:15 UTC; 30min ago  
 Process: 2902 ExecStop=/bin/redis-cli -h 127.0.0.1 -p 7000 shutdown (code=exited, status=0/SUCCESS)  
 **Main PID: 2917 (redis-server)**  
 **CGroup: /system.slice/redis\_7000.service**  
 **└─2917 /usr/bin/redis-server \*:7000 [cluster]**  
systemd[1]: Starting Redis persistent key-value database...  
redis-server[2917]: 2917:C 25 Sep 2019 08:14:15.752 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo  
redis-server[2917]: 2917:C 25 Sep 2019 08:14:15.752 # Redis version=5.0.5, bits=64, commit=00000000, modified=0, pid=2917, just started  
redis-server[2917]: 2917:C 25 Sep 2019 08:14:15.752 # Configuration loaded  
redis-server[2917]: 2917:C 25 Sep 2019 08:14:15.752 \* supervised by systemd, will signal readiness  
systemd[1]: Started Redis persistent key-value database.  
redis-server[2917]: 2917:M 25 Sep 2019 08:14:15.754 \* No cluster configuration found, I'm ff3e4300bec02ed4bd1be9af5d83a5b44249c2b2  
redis-server[2917]: 2917:M 25 Sep 2019 08:14:15.756 \* **Running mode=cluster, port=7000**.  
redis-server[2917]: 2917:M 25 Sep 2019 08:14:15.756 # Server initialized  
redis-server[2917]: 2917:M 25 Sep 2019 08:14:15.756 \* **Ready to accept connections**

**Slave Service**

systemctl status redis\_7001.service● redis\_7001.service - Redis persistent key-value database  
 Loaded: loaded (/etc/systemd/system/redis\_7001.service; enabled; vendor preset: disabled)  
 Active: active (running) since Wed 2019-09-25 08:14:15 UTC; 30min ago  
 Process: 2902 ExecStop=/bin/redis-cli -h 127.0.0.1 -p 7001 shutdown (code=exited, status=0/SUCCESS)  
 **Main PID: 2919 (redis-server)**  
 **CGroup: /system.slice/redis\_7001.service**  
 **└─2919 /usr/bin/redis-server \*:7001 [cluster]**  
systemd[1]: Starting Redis persistent key-value database...  
redis-server[2919]: 2917:C 25 Sep 2019 08:14:15.752 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo  
redis-server[2919]: 2917:C 25 Sep 2019 08:14:15.752 # Redis version=5.0.5, bits=64, commit=00000000, modified=0, pid=2917, just started  
redis-server[2919]: 2917:C 25 Sep 2019 08:14:15.752 # Configuration loaded  
redis-server[2919]: 2917:C 25 Sep 2019 08:14:15.752 \* supervised by systemd, will signal readiness  
systemd[1]: Started Redis persistent key-value database.  
redis-server[2919]: 2917:M 25 Sep 2019 08:14:15.754 \* No cluster configuration found, I'm ff3e4300bec02ed4bd1be9af5d83a5b44249c2b2  
redis-server[2919]: 2917:M 25 Sep 2019 08:14:15.756 \* **Running mode=cluster, port=700**1.  
redis-server[2919]: 2917:M 25 Sep 2019 08:14:15.756 # Server initialized  
redis-server[2919]: 2917:M 25 Sep 2019 08:14:15.756 \* **Ready to accept connections**

**Redis Cluster Setup**

Redis itself provides CLI tool to set up cluster.  
In the current 3 node scenario, I opt 7000 port on all nodes to serve Redis master and 7001 port to serve Redis slave.



**redis-cli --cluster create 172.19.33.7:7000 172.19.42.44:7000 172.19.45.201:7000 172.19.33.7:7001 172.19.42.44:7001 172.19.45.201:7001 --cluster-replicas 1**

The first 3 addresses will be the master and the next 3 addresses will be the slaves. It will be a cross node replication, say, Slave of any Mater will reside on a different node and the *cluster-replicas* define the replication factor, i.e each master will have 1 slave.

>>> Performing hash slots allocation on 6 nodes...  
Master[0] -> Slots 0 - 5460  
Master[1] -> Slots 5461 - 10922  
Master[2] -> Slots 10923 - 16383  
Adding replica 172.19.42.44:7001 to 172.19.33.7:7000  
Adding replica 172.19.45.201:7001 to 172.19.42.44:7000  
Adding replica 172.19.33.7:7001 to 172.19.45.201:7000  
**M: ff3e4300bec02ed4bd1be9af5d83a5b44249c2b2 172.19.33.7:7000**  
 **slots:[0-5460] (5461 slots) master**  
**M: 314038a48bda3224bad21c3357dbff8305735d72 172.19.42.44:7000**  
 **slots:[5461-10922] (5462 slots) master**  
**M: 19a2c81b7f489bec35eed474ae8e1ad787327db6 172.19.45.201:7000**  
 **slots:[10923-16383] (5461 slots) master**  
**S: 896b2a7195455787b5d8a50966f1034c269c0259 172.19.33.7:7001**  
 **replicates 19a2c81b7f489bec35eed474ae8e1ad787327db6**  
**S: 89206df4f41465bce81f44e25e5fdfa8566424b8 172.19.42.44:7001**  
 **replicates ff3e4300bec02ed4bd1be9af5d83a5b44249c2b2**  
**S: 20ab4b30f3d6d25045909c6c33ab70feb635061c 172.19.45.201:7001**  
 **replicates 314038a48bda3224bad21c3357dbff8305735d72**  
**Can I set the above configuration? (type 'yes' to accept):**

A dry run will showcase the cluster setup and ask for confirmation.

**Can I set the above configuration? (type 'yes' to accept): yes**>>> Nodes configuration updated  
>>> Assign a different config epoch to each node  
>>> Sending CLUSTER MEET messages to join the cluster  
Waiting for the cluster to join  
..  
>>> Performing Cluster Check (using node 172.19.33.7:7000)  
M: ff3e4300bec02ed4bd1be9af5d83a5b44249c2b2 172.19.33.7:7000  
 slots:[0-5460] (5461 slots) master  
 1 additional replica(s)  
S: 20ab4b30f3d6d25045909c6c33ab70feb635061c 172.19.45.201:7001  
 slots: (0 slots) slave  
 replicates 314038a48bda3224bad21c3357dbff8305735d72  
M: 314038a48bda3224bad21c3357dbff8305735d72 172.19.42.44:7000  
 slots:[5461-10922] (5462 slots) master  
 1 additional replica(s)  
M: 19a2c81b7f489bec35eed474ae8e1ad787327db6 172.19.45.201:7000  
 slots:[10923-16383] (5461 slots) master  
 1 additional replica(s)  
S: 89206df4f41465bce81f44e25e5fdfa8566424b8 172.19.42.44:7001  
 slots: (0 slots) slave  
 replicates ff3e4300bec02ed4bd1be9af5d83a5b44249c2b2  
S: 896b2a7195455787b5d8a50966f1034c269c0259 172.19.33.7:7001  
 slots: (0 slots) slave  
 replicates 19a2c81b7f489bec35eed474ae8e1ad787327db6  
**[OK] All nodes agree about slots configuration.  
>>> Check for open slots...  
>>> Check slots coverage...  
[OK] All 16384 slots covered.**

**Check Cluster Status**

Connect to any of the cluster nodes to check the status of the cluster.

redis-cli -c -h 172.19.33.7 -p 7000**172.19.33.7:7000> cluster nodes  
20ab4b30f3d6d25045909c6c33ab70feb635061c 172.19.45.201:7001@17001 slave 314038a48bda3224bad21c3357dbff8305735d72 0 1569402961000 6 connected  
314038a48bda3224bad21c3357dbff8305735d72 172.19.42.44:7000@17000 master - 0 1569402961543 2 connected 5461-10922**  
19a2c81b7f489bec35eed474ae8e1ad787327db6 **172.19.45.201:7000**@17000 **master** - 0 1569402960538 3 **connected** 10923-16383  
ff3e4300bec02ed4bd1be9af5d83a5b44249c2b2 **172.19.33.7:7000**@17000 **myself,master** - 0 1569402959000 1 **connected** 0-5460  
89206df4f41465bce81f44e25e5fdfa8566424b8 **172.19.42.44:7001**@17001 **slave** ff3e4300bec02ed4bd1be9af5d83a5b44249c2b2 0 1569402960000 5 **connected**  
896b2a7195455787b5d8a50966f1034c269c0259 **172.19.33.7:7001**@17001 **slave** 19a2c81b7f489bec35eed474ae8e1ad787327db6 0 1569402959936 4 **connected**

Redis cluster itself manages the cross node replication, as seen in the above screen, **172.19.42.44:7000** master is associated with **172.19.45.201:7001** slave.

**Data Sharding**

There are 16384 slots. These slots are divided by the number of servers.  
If there are 3 servers; 1, 2, and 3 then

* Server 1 contains hash slots from 0 to 5500.
* Server 2 contains hash slots from 5501 to 11000.
* Server 3 contains hash slots from 11001 to 16383.

redis-cli -c -h 172.19.33.7 -p 7000172.19.33.7:7000> set a 1  
-> Redirected to slot [15495] located at 172.19.45.201:7000  
OK  
172.19.45.201:7000> set b 2  
-> Redirected to slot [3300] located at 172.19.33.7:7000  
OK  
172.19.33.7:7000> set c 3  
-> Redirected to slot [7365] located at 172.19.42.44:7000  
OK  
172.19.42.44:7000> set d 4  
-> Redirected to slot [11298] located at 172.19.45.201:7000  
OK  
172.19.45.201:7000> get b  
-> Redirected to slot [3300] located at 172.19.33.7:7000  
"2"  
172.19.33.7:7000> get a  
-> Redirected to slot [15495] located at 172.19.45.201:7000  
"1"  
172.19.45.201:7000> get c  
-> Redirected to slot [7365] located at 172.19.42.44:7000  
"3"  
172.19.42.44:7000> get d  
-> Redirected to slot [11298] located at 172.19.45.201:7000  
"4"  
172.19.45.201:7000>

**Redis Cluster Failover**

**Stop Master Service**

Let’s stop the Redis master service on Server 3.

systemctl stop redis\_7000.service  
systemctl status redis\_7000.service● redis\_7000.service - Redis persistent key-value database  
 Loaded: loaded (/etc/systemd/system/redis\_7000.service; enabled; vendor preset: disabled)  
 **Active: inactive (dead)** since Wed 2019-09-25 09:32:37 UTC; 23s ago  
 Process: 3232 ExecStop=/bin/redis-cli -h 127.0.0.1 -p 7000 shutdown (code=exited, status=0/SUCCESS)  
 Process: 2892 ExecStart=/usr/bin/redis-server /etc/redis/cluster/7000/redis\_7000.conf --supervised systemd (code=exited, status=0/SUCCESS)  
 Main PID: 2892 (code=exited, status=0/SUCCESS)

**Cluster State (Failover)**

While checking the cluster status, Redis master service running on server 3 at port 7000 is shown fail and disconnected.

A diagram of a computer program

Description automatically generated

At the same moment, its respective slave gets promoted to master which is running on port 7001 on server 1.

A diagram of a computer server

Description automatically generated

redis-cli -c -h 172.19.33.7 -p 7000**172.19.45.201:7000> CLUSTER NODES**  
***314038a48bda3224bad21c3357dbff8305735d72 172.19.42.44:7000@17000 master,fail - 1569403957138 1569403956000 2 disconnected***  
ff3e4300bec02ed4bd1be9af5d83a5b44249c2b2 172.19.33.7:7000@17000 **master** - 0 1569404037252 1 **connected** 0-5460  
896b2a7195455787b5d8a50966f1034c269c0259 172.19.33.7:7001@17001 **slave** 19a2c81b7f489bec35eed474ae8e1ad787327db6 0 1569404036248 4 **connected**  
89206df4f41465bce81f44e25e5fdfa8566424b8 172.19.42.44:7001@17001 **slave** ff3e4300bec02ed4bd1be9af5d83a5b44249c2b2 0 1569404036752 5 **connected**  
***20ab4b30f3d6d25045909c6c33ab70feb635061c 172.19.45.201:7001@17001 master - 0 1569404036000 7 connected 5461-10922***  
19a2c81b7f489bec35eed474ae8e1ad787327db6 172.19.45.201:7000@17000 **myself,master** - 0 1569404035000 3 **connected** 10923-16383

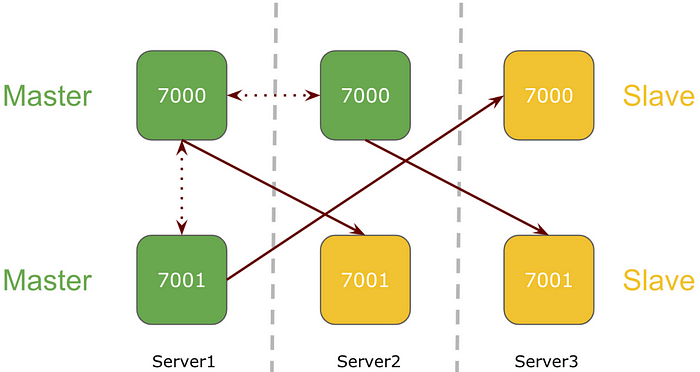
**Restarting Stopped Redis**

Now we will check the behavior of the cluster once we fix or restart the Redis service that we intentionally turned down earlier.

systemctl start redis\_7000.service  
systemctl status redis\_7000.service● redis\_7000.service - Redis persistent key-value database  
 Loaded: loaded (/etc/systemd/system/redis\_7000.service; enabled; vendor preset: disabled)  
 **Active: active (running)** since Wed 2019-09-25 09:35:12 UTC; 8s ago  
 Process: 3232 ExecStop=/bin/redis-cli -h 127.0.0.1 -p 7000 shutdown (code=exited, status=0/SUCCESS)  
 Main PID: 3241 (redis-server)  
 CGroup: /system.slice/redis\_7000.service  
 **└─3241 /usr/bin/redis-server \*:7000 [cluster]**

**Cluster State (Recovery)**

Finally, all redis service is back in running state. The master service that we turned down and restarted has now become a slave to its promoted master.



redis-cli -c -h 172.19.33.7 -p 7000**172.19.45.201:7000> CLUSTER NODES** ***314038a48bda3224bad21c3357dbff8305735d72 172.19.42.44:7000@17000 slave 20ab4b30f3d6d25045909c6c33ab70feb635061c 0 1569404162565 7 connected*** ff3e4300bec02ed4bd1be9af5d83a5b44249c2b2 172.19.33.7:7000@17000 **master** - 0 1569404162000 1 **connected** 0-5460 896b2a7195455787b5d8a50966f1034c269c0259 172.19.33.7:7001@17001 **slave** 19a2c81b7f489bec35eed474ae8e1ad787327db6 0 1569404163567 4 **connected** 89206df4f41465bce81f44e25e5fdfa8566424b8 172.19.42.44:7001@17001 **slave** ff3e4300bec02ed4bd1be9af5d83a5b44249c2b2 0 1569404163000 5 **connected** ***20ab4b30f3d6d25045909c6c33ab70feb635061c 172.19.45.201:7001@17001 master - 0 1569404162000 7 connected 5461-10922*** 19a2c81b7f489bec35eed474ae8e1ad787327db6 172.19.45.201:7000@17000 **myself,master** - 0 1569404161000 3 **connected** 10923-16383

It’s not done yet, further, we can explore having a single endpoint to point from the application. I will am currently working on that and soon will come up with the solution.  
Apart from this monitoring, the Redis Cluster will also be a major aspect to look forward to.  
Till then get your hands dirty playing around the Redis Cluster setup and failover.