REDIS CLUSTER

REDIS + REDUNDANCY + SCALABILITY

Jeff Poole

https://korvus81.github.io/redis-cluster-talk/

https://joind.in/talk/c8087

(press '?' for help, use 'N'/'P' to navigate)

WHO AM I?

vivint.SmartHome™

- Software developer at Vivint on the core platform team
- Previously VoIP developer at Jive Communications
- Before that, a mix of software and hardware (VHDL)
- Coding professionally since 1997 (if you count VB...)

- Redis user (and fan), but no other affiliation

REDIS OVERVIEW

Redis can be used as a Memcached replacement

- In-memory data cache (key-value store)
- Accessible over TCP (so can be shared by multiple processes/servers)
- Very fast

```
>>> import redis

>>> r = redis.StrictRedis() # connects to localhost:6379 by default

>>> r.set("someKey",123)
True

>>> r.get("someKey")
'123'
```

BUT WAIT, THERE'S MORE!

It is also considered a "data structure server" due to it's support for a wide variety of data structures (lists, sets, hashes/maps, etc) and operations on them

Basically a fast, in-memory database for many uses

List example:

```
>>> r.lpush("myList","a","b")   # 2L
>>> r.lpush("myList","c")       # 3L
>>> r.llen("myList")            # 3
>>> r.lindex("myList",2)        # 'a'
>>> r.lrange("myList",0,2)      # ['c', 'b', 'a']
>>> r.lpop("myList")            # 'c'
>>> r.lpop("myList")            # 'b'
>>> r.llen("myList")            # 1
```

Hash(/dictionary/map) example:

```
>>> r.hset("myHash", "name", "Jeff")                  # 1L

>>> r.hset("myHash", "name", "Jeff Poole")            # 0L

>>> r.hset("myHash", "presentation", "Redis Cluster") # 1L

>>> r.hget("myHash", "name")                          # 'Jeff Poole'

>>> r.hgetall("myHash")
{'presentation': 'Redis Cluster', 'name': 'Jeff Poole'}

>>> print repr(r.hvals("myHash")) + " => " + repr(r.hkeys("myHash"))
['Jeff Poole', 'Redis Cluster'] => ['name', 'presentation']
```

Set example:

```
>>> r.sadd("mySet", "a")                              # 1
>>> r.sadd("mySet", "b")                              # 1

>>> r.sadd("myOtherSet", "b")                         # 1
>>> r.sadd("myOtherSet", "c")                         # 1

>>> r.sinter("mySet", "myOtherSet")
set(['b'])

>>> r.sunion("mySet", "myOtherSet")
set(['a', 'c', 'b'])

>>> r.sunionstore("myUnion", "mySet", "myOtherSet")  # 3

>>> r.smembers("myUnion")
set(['a', 'c', 'b'])
```

REDIS CLUSTER FEATURES

Cluster mode adds two major features to Redis.

- Server-side sharding
- Automatic fail-over to slave nodes

SERVER-SIDE SUPPORT FOR SHARDING DATA

- Improves the ability to scale both to more machines and more cores (a single Redis process is effectively single-threaded)
- Can be changed/rebalanced while the environment is running, whereas client-side usually means a code change and separate migration step

SERVER-SIDE SUPPORT FOR SHARDING DATA

- Keys are hashed into 16k slots (0-16383)
- Slots are **manually** mapped to masters
  - Clusters require initial configuration
  - Balancing is a manual process
  - Data for one slot (and therefore any one key) will never be split between hosts
- Can shard on a subset of the key, with "hash tags" used by surrounding the part of the key you want to hash in {}s (so 'abc{slot1}key1' and 'xyz{slot1}key2' will shard to the same slot always)

AUTOMATIC FAIL-OVER TO SLAVE NODES

- Previously, required a Redis Sentinel configuration
- Fail-back is manual, but only requires one command:
  `CLUSTER FAILOVER`, run on the slave you want to become a master

REDIS CLUSTER DOWNSIDES

Cluster mode comes with a few gotchas as well

- Some client libraries are still missing cluster support
- Poor support for remapped IPs/ports (common in Docker setups)
- No cross-slot commands, even if both slots are on the same master
- Cluster sacrifices consistency for availability -- when servers failover, there is a short window for data loss
- No databases (i.e. `SELECT` command does not work)

CLIENTS WITHOUT CLUSTER SUPPORT

- Python - redis-py has no support (use redis-py-cluster instead for now), neither does asyncio_redis (use asyncio-redis-cluster)
- Go - gopkg.in/redis.v3 and github.com/mediocregopher/radix.v2 are OK, github.com/garyburd/redigo does NOT have support (use github.com/chasex/redis-go-cluster)
- Java - Jedis and Lettuce both have cluster support
- Node.js - ioredis has support, node_redis does not (use redis-clustr)
- *Note that the cluster-specific clients often only support Redis servers with cluster mode enabled*

REMAPPED IP/PORTS

Cluster nodes gossip about each other and tell clients about the other nodes. There isn't currently a way to tell nodes what IP/port to advertise, so a cluster will be unusable behind a NAT configuration.

Should have a solution with **https://github.com/antirez/redis/issues/2527** whenever that makes it into a releaase...should be soon.

NO CROSS-SLOT COMMANDS

This means you can't run:

- `MGET a b c`
- `MSET a 1 b 2`
- `EVAL "..." 2 a b`
- `SUNION[STORE] a b c`
- `[B]RPOPLPUSH a b c`
- ...and others...

```
127.0.0.1:6379> cluster keyslot a  # 15495
127.0.0.1:6379> cluster keyslot b  # 3300
127.0.0.1:6379> cluster keyslot c  # 7365
127.0.0.1:6379> mget a b c
# (error) CROSSSLOT Keys in request don't hash to the same slot
```

NO CROSS-SLOT COMMANDS

The best solution is to use the "hash tags" feature to make sure the keys all hash to the same slot:

```
127.0.0.1:6379> cluster keyslot {a}a  # 15495
127.0.0.1:6379> cluster keyslot {a}b  # 15495
127.0.0.1:6379> cluster keyslot {a}c  # 15495
127.0.0.1:6379> mget {a}a {a}b {a}c
# 1) "AAA"
# 2) "BBB"
# 3) "CCC"
```

If performance is not a factor, many drivers will split commands like `MGET a b` into `GET a` and `GET b`, sent to the appropriate masters.

CLUSTER SACRIFICES CONSISTENCY FOR AVAILABILITY

If a master becomes unresponsive from the cluster's perspective, another slave will get promoted to master.

Once the master realizes another master has been promoted, it will step down and sync from the new master.

Any writes to the first master between those two events **will be lost.**

Good news: this requires something like a network partition -- a crashed or rebooted master will take no writes, and should step-down immediately when it returns.

SOME CLUSTER DETAILS

A Redis server in cluster mode has a port for server-to-server communication, which defaults to it's normal port + 4000 (ex: client port 6379 -> server port 10379)

Servers connect to each other in a **full mesh** and communicate information about the cluster (such as slot mappings and unresponsive servers) with a gossip protocol.

Fail-over behavior (when a slave can call an election for a new master) can be controlled with `cluster-node-timeout` and `cluster-slave-validity-factor`

All masters vote in an election

SETTING UP REDIS CLUSTER

CONFIG FILE CHANGES (`REDIS.CONF`):

- `cluster-enabled yes`
- `cluster-config-file node.conf`
  - A file where Redis will save it's view of the cluster.
  - Should not be modified by hand nor shared with any other instances.

CONFIG FILE CHANGES (`REDIS.CONF`):

- `requirepass secret`
  `masterauth secret`
  - If you need a password, set it in **both places**
  - This is important to do even on masters since the roles can switch during failover!

I will use command-line options for demonstration purposes...

START IT UP

```
> redis-server --bind 127.0.0.1 --port 7000 --save '' \
   --cluster-enabled yes --cluster-config-file node-7000.conf
1900:M 26 Jun 19:40:26.747 * No cluster configuration found, I'm a70d66e1b596f44c3b39a7b27b87ad06fa4820dc
                _.-``__ ''-._
           .-`` .-```.  ```\/    _.,_ ''-._
      _.-``    `.  `_.  ''-._           Redis 3.2.0 (00000000/0) 64 bit
  .-`` .-```.  ```\/    _.,_ ''-._
 (    '      ,       .-`  | `,    )     Running in cluster mode
 |`-._`-...-` __...-.``-._|'` _.-'|     Port: 7000
 |    `-._   `._    /     _.-'    |     PID: 1900
  `-._    `-._  `-./  _.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |           http://redis.io
  `-._    `-._`-.__.-'_.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |
  `-._    `-._`-.__.-'_.-'    _.-'
      `-._    `-.__.-'    _.-'
          `-._        _.-'
              `-.__.-'

1900:M 26 Jun 19:40:26.750 # Server started, Redis version 3.2.0
```

START IT UP

```
> redis-server --bind 127.0.0.1 --port 7001 --save '' \
  --cluster-enabled yes --cluster-config-file node-7001.conf
3550:M 26 Jun 19:51:43.754 * No cluster configuration found, I'm b7c8239a7db8b340de07706558b369aa5727ca9f
                _-._
           .-`` __ ``-._
      _.-``    `.  _.  ''-._           Redis 3.2.0 (00000000/0) 64 bit
  .-`` .-```.  ```\/    _.,_ ''-._
 (    '      ,       .-`  | `,    )     Running in cluster mode
 |`-._`-...-` __...-.``-._|'` _.-'|     Port: 7001
 |    `-._   `._    /     _.-'    |     PID: 3550
  `-._    `-._  `-./  _.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |           http://redis.io
  `-._    `-._`-.__.-'_.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |
  `-._    `-._`-.__.-'_.-'    _.-'
      `-._    `-.__.-'    _.-'
          `-._        _.-'
              `-.__.-'

3550:M 26 Jun 19:51:43.755 # Server started, Redis version 3.2.0
```

```
> redis-cli -p 7000 CLUSTER INFO
cluster_state:fail
cluster_slots_assigned:0
cluster_slots_ok:0
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:1
cluster_size:0
cluster_current_epoch:0
cluster_my_epoch:0
cluster_stats_messages_sent:0
cluster_stats_messages_received:0
```

```
> redis-cli -p 7000 CLUSTER NODES
a70d66e1b596f44c3b39a7b27b87ad06fa4820dc :7000 myself,master - 0 0 0 connected
```

```
> redis-cli -p 7000 CLUSTER SLOTS
(empty list or set)
```

The servers don't know about each other, and they have no slots assigned. We need to fix that.

Two ways to set up slot configuration: **manually** or via **redis-trib.rb**

Let's do it manually just to show the steps involved!

First we have the nodes meet:

```
> redis-cli -p 7000 CLUSTER MEET 127.0.0.1 7001
OK
```

```
> redis-cli -p 7000 CLUSTER NODES
a70d66e1b596f44c3b39a7b27b87ad06fa4820dc 127.0.0.1:7000 myself,master ...
b7c8239a7db8b340de07706558b369aa5727ca9f 127.0.0.1:7001 master ...

> redis-cli -p 7001 CLUSTER NODES
b7c8239a7db8b340de07706558b369aa5727ca9f 127.0.0.1:7001 myself,master ...
a70d66e1b596f44c3b39a7b27b87ad06fa4820dc 127.0.0.1:7000 master ...
```

Now let's assign some slots.

```
> redis-cli -p 7000 CLUSTER ADDSLOTS (seq 0 8191)
OK
```

```
> redis-cli -p 7000 CLUSTER SLOTS
1) 1) (integer) 0
   2) (integer) 8191
   3) 1) "127.0.0.1"
      2) (integer) 7000
      3) "a70d66e1b596f44c3b39a7b27b87ad06fa4820dc"
```

Now for the second node...

```
> redis-cli -p 7001 CLUSTER ADDSLOTS (seq 8192 16383)
OK
```

```
> redis-cli -p 7000 CLUSTER SLOTS
1) 1) (integer) 0
   2) (integer) 8191
   3) 1) "127.0.0.1"
      2) (integer) 7000
      3) "a70d66e1b596f44c3b39a7b27b87ad06fa4820dc"
2) 1) (integer) 8192
   2) (integer) 16383
   3) 1) "127.0.0.1"
      2) (integer) 7001
      3) "b7c8239a7db8b340de07706558b369aa5727ca9f"
```

At this point we have assigned all slots (0-16383) and both nodes are aware of the full mapping because they gossip with each other.

Cluster is now up!

```
> redis-cli -p 7000 CLUSTER INFO
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:2
cluster_size:2
cluster_current_epoch:1
cluster_my_epoch:1
cluster_stats_messages_sent:3016
cluster_stats_messages_received:3016
```

How would we set up replication with slave nodes?

```
> redis-server --bind 127.0.0.1 --port 7002 --save ''\
  --cluster-enabled yes --cluster-config-file node-7002.conf
* No cluster configuration found, I'm 3b379c78a4994c757942cff6b6bb01ec6f6ace68
...
```

```
> redis-cli -p 7002 CLUSTER INFO
cluster_state:fail
cluster_slots_assigned:0
cluster_slots_ok:0
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:1
cluster_size:0
cluster_current_epoch:0
cluster_my_epoch:0
cluster_stats_messages_sent:0
cluster_stats_messages_received:0
```

Now we join the cluster...

```
> redis-cli -p 7002 CLUSTER MEET 127.0.0.1 7000
OK
```

(we could have pointed it to ANY node in the cluster)

```
> redis-cli -p 7002 CLUSTER INFO
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:3
cluster_size:2
cluster_current_epoch:2
cluster_my_epoch:2
cluster_stats_messages_sent:68
cluster_stats_messages_received:68
```

And offer to become a replica of the node at :7000

```
> redis-cli -p 7002 CLUSTER REPLICATE a70d66e1b596f44c3b39a7b27b87ad06fa4820dc
OK
```

```
> redis-cli -p 7002 CLUSTER SLOTS
1) 1) (integer) 8192
   2) (integer) 16383
   3) 1) "127.0.0.1"
      2) (integer) 7001
      3) "b7c8239a7db8b340de07706558b369aa5727ca9f"
2) 1) (integer) 0
   2) (integer) 8191
   3) 1) "127.0.0.1"
      2) (integer) 7000
      3) "a70d66e1b596f44c3b39a7b27b87ad06fa4820dc"
   4) 1) "127.0.0.1"
      2) (integer) 7002
      3) "3b379c78a4994c757942cff6b6bb01ec6f6ace68"
```

```
> redis-cli -p 7002 CLUSTER NODES
b7c8239a...5727ca9f 127.0.0.1:7001 master - ... 8192-16383
3b379c78...6f6ace68 127.0.0.1:7002 myself,slave a70d66e1...fa4820dc ...
a70d66e1...fa4820dc 127.0.0.1:7000 master - ... 0-8191
```

THE EASY WAY

If you look in the Redis distribution (or check it out from git), in the `/src` directory you will find a Ruby script called `redis-trib.rb`.

This script can be used for setting up a cluster and for tasks such as re-balancing slots within a cluster.

redis-trib.rb sub-commands are:

- `create`
- `check`
- `info`
- `fix`
- `reshard`
- `rebalance`
- `add-node`
- `del-node`
- `set-timeout`
- `call`
- `import`
- `help`

Starting with the three servers I had running before, I make them forget all cluster state by sending each one `CLUSTER RESET HARD`.

The `HARD` part makes them generate a new ID so they act like brand new instances.

```
> ./redis-trib.rb create \
    --replicas 0 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002
>>> Creating cluster
>>> Performing hash slots allocation on 3 nodes...
Using 3 masters:
127.0.0.1:7000
127.0.0.1:7001
127.0.0.1:7002
M: cd8ca08ac9c1ba83be678da684b7a4416c7d4402 127.0.0.1:7000
   slots:0-5460 (5461 slots) master
M: 845e01037c77f06e327c8c63fbaa97ee101cb92d 127.0.0.1:7001
   slots:5461-10922 (5462 slots) master
M: 00840e9a52dfdfa7299ee14ba5c50ee281184c56 127.0.0.1:7002
   slots:10923-16383 (5461 slots) master
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 127.0.0.1:7000)
M: cd8ca08ac9c1ba83be678da684b7a4416c7d4402 127.0.0.1:7000
   slots:0-5460 (5461 slots) master
M: 845e01037c77f06e327c8c63fbaa97ee101cb92d 127.0.0.1:7001
   slots:5461-10922 (5462 slots) master
M: 00840e9a52dfdfa7299ee14ba5c50ee281184c56 127.0.0.1:7002
   slots:10923-16383 (5461 slots) master
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

PLAYING WITH CLUSTER

Now let's try a few things to demonstrate the features / quirks of Redis Cluster

Let's start with 6 servers

```
> for i in (seq 0 5)
  redis-server --bind 127.0.0.1 --port 700$i --save '' \
    --repl-diskless-sync yes --dbfilename dump$i.rdb \
    --cluster-enabled yes --cluster-config-file node-700$i.conf &
end
```

Now configure them as 3 master nodes each with a slave

```
> set REDIS_NODES (for i in (seq 0 5); echo 127.0.0.1:700$i ; end)
> ./redis-trib.rb create --replicas 1 $REDIS_NODES
```

Verify things are as they should be

```
> redis-cli -p 7000 CLUSTER NODES
6516...965e 127.0.0.1:7000 myself,master - 0 0 1 connected 0-5460
66d2...ee37 127.0.0.1:7001 master - 0 1467586620690 2 connected 5461-10922
23a8...e9b0 127.0.0.1:7002 master - 0 1467586617588 3 connected 10923-16383
3347...ed48 127.0.0.1:7003 slave 6516...965e 0 1467586621723 4 connected
da09...fbed 127.0.0.1:7004 slave 66d2...ee37 0 1467586622762 5 connected
6290...e55a 127.0.0.1:7005 slave 23a8...e9b0 0 1467586619657 6 connected
```

See how slot mechanics work

```
> redis-cli -p 7000 CLUSTER KEYSLOT a
(integer) 15495
> redis-cli -p 7000 CLUSTER KEYSLOT b
(integer) 3300

> redis-cli -p 7000 CLUSTER KEYSLOT 'a{hash_tag}'
(integer) 2515
> redis-cli -p 7000 CLUSTER KEYSLOT 'b{hash_tag}'
(integer) 2515
> redis-cli -p 7000 CLUSTER KEYSLOT 'hash_tag'
(integer) 2515
```

```
> redis-cli -p 7000 SET a 1
(error) MOVED 15495 127.0.0.1:7002
> redis-cli -p 7002 SET a 1
OK
```

```
> redis-cli -p 7000 MSET 'a{hash_tag}' AAA 'b{hash_tag}' BBB
OK
> redis-cli -p 7000 MSET 'a' AAA 'b' BBB
(error) CROSSSLOT Keys in request don't hash to the same slot
> redis-cli -p 7000 MGET 'a{hash_tag}' 'b{hash_tag}'
1) "AAA"
2) "BBB"
```

Let's test fail-over!

```
> redis-cli -p 7000 CLUSTER NODES
6516...965e 127.0.0.1:7000 myself,master - 0 0 1 connected 0-5460
66d2...ee37 127.0.0.1:7001 master - 0 1467587810151 2 connected 5461-10922
23a8...e9b0 127.0.0.1:7002 master - 0 1467587809116 3 connected 10923-16383
3347...ed48 127.0.0.1:7003 slave 6516...965e 0 1467587807048 4 connected
da09...fbed 127.0.0.1:7004 slave 66d2...ee37 0 1467587808078 5 connected
6290...e55a 127.0.0.1:7005 slave 23a8...e9b0 0 1467587806009 6 connected
```

Now block the main thread on the first node for 30 sec...

```
> redis-cli -p 7000 DEBUG SLEEP 30
OK
```

```
> redis-cli -p 7000 CLUSTER NODES
6516...965e 127.0.0.1:7000 myself,slave 3347...ed48 0 0 1 connected
66d2...ee37 127.0.0.1:7001 master - 0 1467588106945 2 connected 5461-10922
23a8...e9b0 127.0.0.1:7002 master - 0 1467588111091 3 connected 10923-16383
3347...ed48 127.0.0.1:7003 master - 0 1467588105911 7 connected 0-5460
da09...fbed 127.0.0.1:7004 slave 66d2...ee37 0 1467588112121 5 connected
6290...e55a 127.0.0.1:7005 slave 23a8...e9b0 0 1467588109022 6 connected
```

Logs (starting with original slave):

```
17:20:53.666 * FAIL message received from 23a8...e9b0 about 6516...965e
17:20:53.666 # Cluster state changed: fail
17:20:53.714 # Start of election delayed for 964 milliseconds (rank #0, offset 31060).
17:20:54.750 # Starting a failover election for epoch 7.
17:20:54.751 # Failover election won: I'm the new master.
17:20:54.751 # configEpoch set to 7 after successful failover
17:20:54.751 # Connection with master lost.
17:20:54.751 * Caching the disconnected master state.
17:20:54.751 * Discarding previously cached master state.
17:20:54.751 # Cluster state changed: ok
```

```
17:21:04.982 # Connection with slave 127.0.0.1:7003 lost.
17:21:04.985 # Failover auth denied to 3347...ed48: its master is up
17:21:04.986 # Configuration change detected.
               Reconfiguring myself as a replica of 3347...ed48
```

```
17:21:04.988 * Clear FAIL state for node 6516...7965e:
               master without slots is reachable again.
```

```
17:21:05.494 * Connecting to MASTER 127.0.0.1:7003
17:21:05.498 * MASTER <-> SLAVE sync started
```

```
17:21:05.499 * Slave 127.0.0.1:7000 asks for synchronization
17:21:05.499 * Full resync requested by slave 127.0.0.1:7000
```

To change back, just tell the old master to initiate a fail-over (this must be run on a slave)

```
> redis-cli -p 7000 CLUSTER FAILOVER
OK
```

```
> redis-cli -p 7000 CLUSTER NODES
6516...965e 127.0.0.1:7000 myself,master - 0 0 8 connected 0-5460
66d2...ee37 127.0.0.1:7001 master - 0 1467589519894 2 connected 5461-10922
23a8...e9b0 127.0.0.1:7002 master - 0 1467589520620 3 connected 10923-16383
3347...ed48 127.0.0.1:7003 slave 6516...965e 0 1467589522687 8 connected
da09...fbed 127.0.0.1:7004 slave 66d2...ee37 0 1467589519584 5 connected
6290...e55a 127.0.0.1:7005 slave 23a8...e9b0 0 1467589521656 6 connected
```

Note that any operation that blocks the main thread can cause a fail-over. For example, `FLUSHALL` on a large dataset may be problematic when the master finishes flushing and finds out it is no longer the master...

REBALANCING / MIGRATING SLOTS

Rebalancing or migrating slots is best done with `redis-trib.rb`, because it is a complex process to do by hand

Say you want to add a new master and slave. Let's make two more servers:

```
> for i in (seq 6 7)
  redis-server --bind 127.0.0.1 --port 700$i --save '' \
    --repl-diskless-sync yes --dbfilename dump$i.rdb \
    --cluster-enabled yes --cluster-config-file node-700$i.conf &
end
```

Add them to the cluster...

```
> ./redis-trib.rb add-node 127.0.0.1:7006 127.0.0.1:7000
...
>>> Send CLUSTER MEET to node 127.0.0.1:7006 to make it join the cluster.
[OK] New node added correctly.

> ./redis-trib.rb add-node 127.0.0.1:7007 127.0.0.1:7000
...
>>> Send CLUSTER MEET to node 127.0.0.1:7007 to make it join the cluster.
[OK] New node added correctly.
```

Make one a slave of the other...

```
> redis-cli -p 7007 CLUSTER NODES |grep :7006
fcd4...e23a 127.0.0.1:7006 myself,master - 0 0 0 connected

> redis-cli -p 7007 CLUSTER REPLICATE fcd4...e23a
OK
```

Then rebalance...

```
> ./redis-trib.rb rebalance --use-empty-masters 127.0.0.1:7000
>>> Performing Cluster Check (using node 127.0.0.1:7000)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
>>> Rebalancing across 4 nodes. Total weight = 4
Moving 1366 slots from 127.0.0.1:7001 to 127.0.0.1:7006
########...########
Moving 1365 slots from 127.0.0.1:7002 to 127.0.0.1:7006
########...########
Moving 1365 slots from 127.0.0.1:7000 to 127.0.0.1:7006
########...########
```

```
> redis-cli -p 7000 CLUSTER NODES
6516...965e 127.0.0.1:7000 myself,master - ... 1365-5460
66d2...ee37 127.0.0.1:7001 master - ... 6827-10922
23a8...e9b0 127.0.0.1:7002 master - ... 12288-16383
3347...ed48 127.0.0.1:7003 slave 6516...965e ...
da09...fbed 127.0.0.1:7004 slave 66d2...ee37 ...
6290...e55a 127.0.0.1:7005 slave 23a8...e9b0 ...
fcd4...e23a 127.0.0.1:7006 master - ... 0-1364 5461-6826 10923-12287
65eb...1e29 127.0.0.1:7007 slave fcd4...e23a ...
```

What if we want to get rid of that part of the cluster? Maybe take the servers down for maintenance?

```
> ./redis-trib.rb reshard 127.0.0.1:7000

>>> Performing Cluster Check (using node 127.0.0.1:7000)
M: 6516050e2962749f3bcd22b8cf2a04e31987965e 127.0.0.1:7000
   slots:1365-5460 (4096 slots) master
   1 additional replica(s)
M: fcd4cb54abcb0db7a02c2dfc04edf83cbdf1e23a 127.0.0.1:7006
   slots:0-1364,5461-6826,10923-12287 (4096 slots) master
   1 additional replica(s)
...(6 more)...
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.

How many slots do you want to move (from 1 to 16384)? 4096
What is the receiving node ID? 6516050e2962749f3bcd22b8cf2a04e31987965e
Please enter all the source node IDs.
  Type 'all' to use all the nodes as source nodes for the hash slots.
  Type 'done' once you entered all the source nodes IDs.
Source node #1:fcd4cb54abcb0db7a02c2dfc04edf83cbdf1e23a
Source node #2:done
```

```
Ready to move 4096 slots.
  Source nodes:
    M: fcd4cb54abcb0db7a02c2dfc04edf83cbdf1e23a 127.0.0.1:7006
   slots:0-1364,5461-6826,10923-12287 (4096 slots) master
   1 additional replica(s)
  Destination node:
    M: 6516050e2962749f3bcd22b8cf2a04e31987965e 127.0.0.1:7000
   slots:1365-5460 (4096 slots) master
   1 additional replica(s)
  Resharding plan:
    Moving slot 0 from fcd4cb54abcb0db7a02c2dfc04edf83cbdf1e23a
    Moving slot 1 from fcd4cb54abcb0db7a02c2dfc04edf83cbdf1e23a
    ...
    Moving slot 12287 from fcd4cb54abcb0db7a02c2dfc04edf83cbdf1e23a
Do you want to proceed with the proposed reshard plan (yes/no)? yes
Moving slot 0 from 127.0.0.1:7006 to 127.0.0.1:7000:
...
Moving slot 12287 from 127.0.0.1:7006 to 127.0.0.1:7000:
```

Notice it has no more slots listed

```
> redis-cli -p 7000 CLUSTER NODES |grep 7006
fcd4...e23a 127.0.0.1:7006 master - 0 1467592749158 10 connected
```

Time to retire those nodes

```
> ./redis-trib.rb del-node 127.0.0.1:7000 fcd4cb54abcb0db7a02c2dfc04edf83cbdf1e23a
>>> Removing node fcd4cb54abcb0db7a02c2dfc04edf83cbdf1e23a from cluster 127.0.0.1:7000
>>> Sending CLUSTER FORGET messages to the cluster...
>>> SHUTDOWN the node.

> ./redis-trib.rb del-node 127.0.0.1:7000 65eb5018b3171e79ec145e37e858489a4d2f1e29
>>> Removing node 65eb5018b3171e79ec145e37e858489a4d2f1e29 from cluster 127.0.0.1:7000
>>> Sending CLUSTER FORGET messages to the cluster...
>>> SHUTDOWN the node.
```

That makes the cluster forget those nodes and shuts them both down. We now are back to our 6-node cluster.

```
> ./redis-trib.rb info 127.0.0.1:7000
127.0.0.1:7000 (6516050e...) -> 2 keys | 8192 slots | 1 slaves.
127.0.0.1:7001 (66d2c906...) -> 0 keys | 4096 slots | 1 slaves.
127.0.0.1:7002 (23a839ff...) -> 1 keys | 4096 slots | 1 slaves.
[OK] 3 keys in 3 masters.
0.00 keys per slot on average.
```

If desired, we can re-balance the keys now (or I could have moved them evenly in the first place)

```
> ./redis-trib.rb rebalance 127.0.0.1:7000
>>> Performing Cluster Check (using node 127.0.0.1:7000)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
>>> Rebalancing across 3 nodes. Total weight = 3
Moving 1366 slots from 127.0.0.1:7000 to 127.0.0.1:7001
########...########
Moving 1365 slots from 127.0.0.1:7000 to 127.0.0.1:7002
########...########
```

QUESTIONS?

Other resources:

- **Redis Cluster Tutorial**
- **Redis Cluster Specification**
- **Cluster-specific Redis Commands**

Contact:

- Email: **jeff@jeffpoole.net**
- Twitter: **@_JeffPoole**
- Talk Feedback: **https://joind.in/talk/c8087**

Slides: **https://korvus81.github.io/redis-cluster-talk/**