Redis Cluster moving slots from one

node to another

Introduction

[CLUSTER SETSLOT](https://redis.io/commands/cluster-setslot) is responsible of changing the state of a hash slot in the receiving node in different ways. It can, depending on the subcommand used:

* MIGRATING subcommand: Set a hash slot in migrating state.
* IMPORTING subcommand: Set a hash slot in importing state.
* STABLE subcommand: Clear any importing / migrating state from hash slot.
* NODE subcommand: Bind the hash slot to a different node.

The command with its set of subcommands is useful in order to start and end cluster live resharding operations, which are accomplished by setting a hash slot in migrating state in the source node, and importing state in the destination node.

Each subcommand is documented below. At the end you'll find a description of how live resharding is performed using this command and other related commands.

Commands

**CLUSTER SETSLOT <slot> MIGRATING <destination-node-id>**

This subcommand sets a slot to *migrating* state. In order to set a slot in this state, the node receiving the command must be the hash slot owner, otherwise an error is returned.

When a slot is set in migrating state, the node changes behavior in the following way:

1. If a command is received about an existing key, the command is processed as usually.
2. If a command is received about a key that does not exists, an ASK redirection is emitted by the node, asking the client to retry only that specific query into destination-node. In this case the client should not update its hash slot to node mapping.
3. If the command contains multiple keys, in case none exist, the behavior is the same as point 2, if all exist, it is the same as point 1, however if only a partial number of keys exist, the command emits a TRYAGAIN error in order for the keys interested to finish being migrated to the target node, so that the multi keys command can be executed.

**CLUSTER SETSLOT <slot> IMPORTING <source-node-id>**

This subcommand is the reverse of MIGRATING, and prepares the destination node to import keys from the specified source node. The command only works if the node is not already owner of the specified hash slot.

When a slot is set in importing state, the node changes behavior in the following way:

* Commands about this hash slot are refused and a MOVED redirection is generated as usually, but in the case the command follows an ASKING command, in this case the command is executed.

keys already migrated to the target node are executed in the target node, so that:

1. New keys are always created in the target node. During a hash slot migration we'll have to move only old keys, not new ones.
2. Commands about keys already migrated are correctly processed in the context of the node which is the target of the migration, the new hash slot owner, in order to guarantee consistency.
3. Without ASKING the behavior is the same as usually. This guarantees that clients with a broken hash slots mapping will not write for error in the target node, creating a new version of a key that has yet to be migrated.

## CLUSTER SETSLOT <slot> STABLE

This subcommand just clears migrating / importing state from the slot. It is mainly used to fix a cluster stuck in a wrong state by redis-trib fix. Normally the two states are cleared automatically at the end of the migration using the SETSLOT ... NODE ... subcommand as explained in the next section.

## CLUSTER SETSLOT <slot> NODE <node-id>

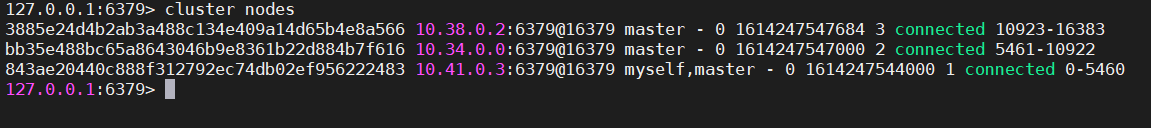
The NODE subcommand is the one with the most complex semantics. It associates the hash slot with the specified node, however the command works only in specific situations and has different side effects depending on the slot state. The following is the set of pre-conditions and side effects of the command:

1. If the current hash slot owner is the node receiving the command, but for effect of the command the slot would be assigned to a different node, the command will return an error if there are still keys for that hash slot in the node receiving the command.
2. If the slot is in migrating state, the state gets cleared when the slot is assigned to another node.
3. If the slot was in importing state in the node receiving the command, and the command assigns the slot to this node (which happens in the target node at the end of the resharding of a hash slot from one node to another), the command has the following side effects: A) the importing state is cleared. B) If the node config epoch is not already the greatest of the cluster, it generates a new one and assigns the new config epoch to itself. This way its new hash slot ownership will win over any past configuration created by previous failovers or slot migrations.
4. It is important to note that step 3 is the only time when a Redis Cluster node will create a new config epoch without agreement from other nodes. This only happens when a manual configuration is operated. However it is impossible that this creates a non-transient setup where two nodes have the same config epoch, since Redis Cluster uses a config epoch collision resolution algorithm.

Practical Example

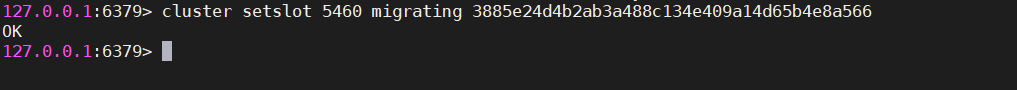
1. Run the cluster node command inside the redis conatiner to get the the node id

**cluster nodes**



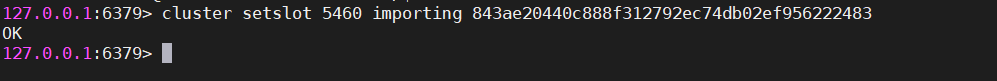
2. Select the slot from source node which need to migrate to the destination node id

**cluster setslot <slot> migrating <dest node id>**



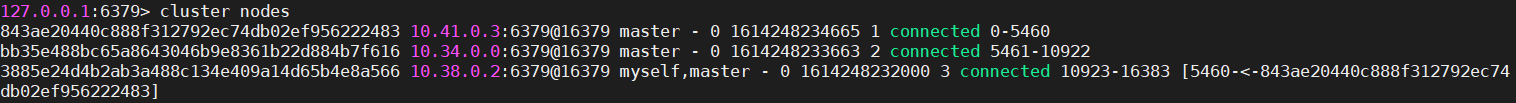
3. On destination node select the same node in importating mode

**cluster setslot <slot> importing <source node id>**



4. After migration go to the destination node and check the cluster node

**5460 slot moved to destination node but not merge**



To merge the slot to the destination run below command

**cluster setslot <slot> node <dest-node-id>**



5. Run the cluster nodes command to see the slot

**In below image you can see the 5460 moved from source to dest node**

**Source node slots become 0-5459 (from 0-5460)**

**Destination node slots become 5460 10923-16383 (from 10923-16383)**

