**Title:** Redis KEYSLOT

**Excerpt:** Redis *KEYSLOT* command is one of the subcommands associated with the container command *CLUSTER* that is used to get the hash slot value of a specified Redis key. It supports hashtags as well. Usually, this command is used to identify the node that a given Redis key is stored in.

**Permalink:** redis-keyslot

**Category:** Redis

# 

# 

# 

# Redis High Availability

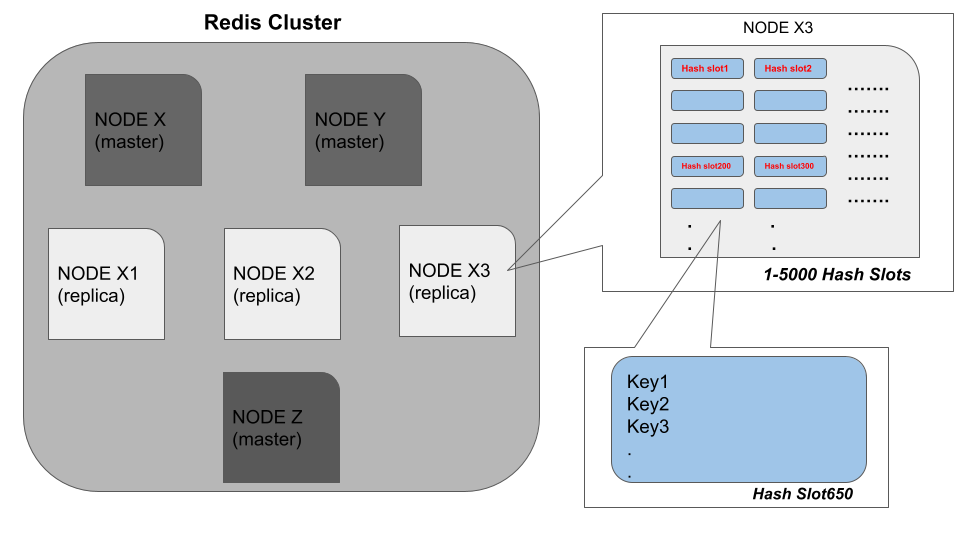
Redis is an in-memory low latency data store that provides the sharding and replication features out of the box. It ensures the high availability of the Redis instance to its stakeholders. The following two mechanisms have been used in the Redis data store to provide high availability.

* Horizontal scaling with clustering
* Redis Sentinel

The horizontal scaling always focuses on increasing the number of instances instead of growing computing power per instance. Redis clustering is based on horizontal scaling where a set of nodes takes care of the instance's continuous functioning.

## Redis Cluster

Redis cluster consists of multiple nodes where each node contains a range of hash slots. Usually, a cluster has 16384 hash slots. Each Redis key is associated with a hash slot and to compute the hash slot per key Redis uses the CRC16 of the Redis key modulo 16384 algorithm. It is based on the master-replica technique where one master node fails, and the respective replica fires in and continues to operate on behalf of the failed node.



Let’s assume that the NODE X fails due to a network failure. Then the Redis cluster will assign the workload of the NODE X to the replica NODE X1 which will ensure smooth functioning. Also, Redis clusters can split the incoming data to different nodes automatically.

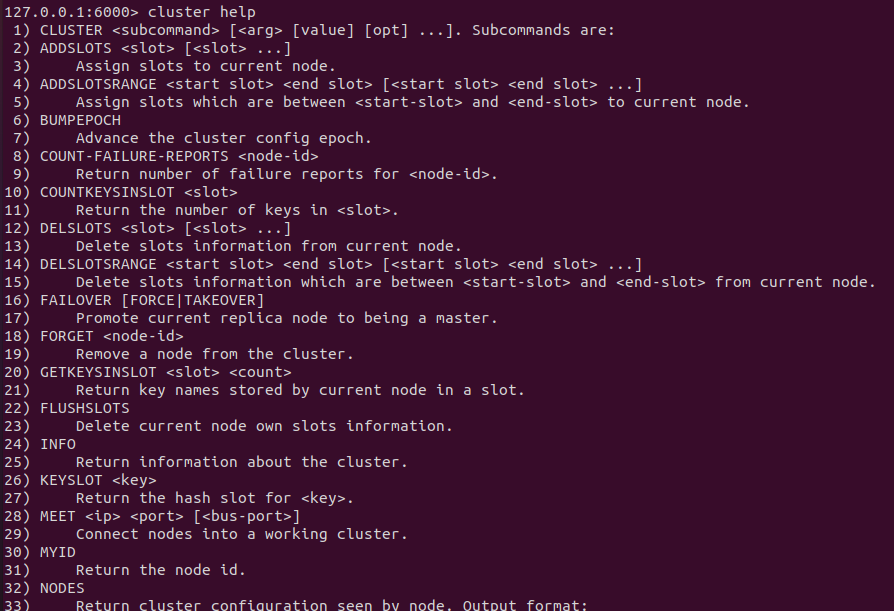
# Redis Cluster Commands

Several commands are available to use against Redis clusters. The CLUSTER command is a container command which has several subcommands. You can use the *cluster* *help* subcommand to check the list of available subcommands with their usage.

| cluster help |
| --- |

As expected it returns an array reply as shown in the following.

Output:



In this guide, we will be focusing on the *KEYSLOT* sub-command that is associated with the container command *CLUSTER*.

# Redis KEYSLOT Command

The *KEYSLOT* command is used to check the hash slot value for a given Redis key which returns an integer value. Whenever you need to know about the associated node for a specific key, this command will be useful. Another use case would be, testing your hashing algorithms by comparing them with the hashes generated by this command and checking for similarity. In addition, the *KEYSLOT* command has linear time complexity.

## Syntax

The following is the syntax for the Redis *KEYSLOT* sub command.

| CLUSTER KEYSLOT redis\_key |
| --- |

***redis\_key***: This is the unique identifier that we use in Redis data stores to identify any value stored.

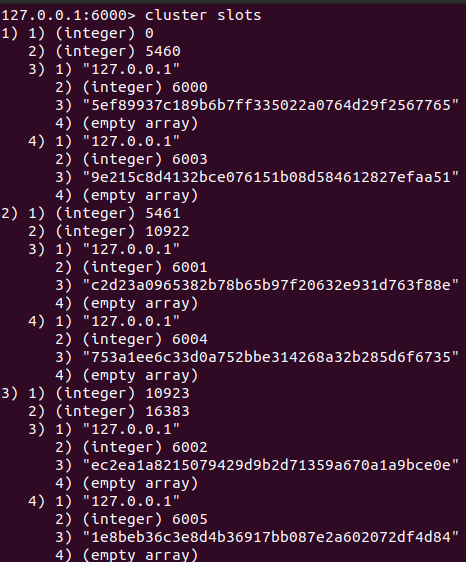
This command returns the hash slot value of the specified Redis key stored in. Usually, this value is an integer ranging from 0 to 16383 for a given cluster.

## Example 01 - Inspect the Hash Slot and Cluster Node for a Given Key

Let’s create a Redis cluster with six nodes where three of them are master nodes and the rest are replicas. We can use the *SLOTS* subcommand to check how the nodes have been assigned with the hash slots.

| cluster slots |
| --- |

As expected, the cluster information has been output as in the following.



Let’s add a couple of keys to our Redis cluster as shown in the following.

| set traveller:1:place france set traveller:2:place india |
| --- |

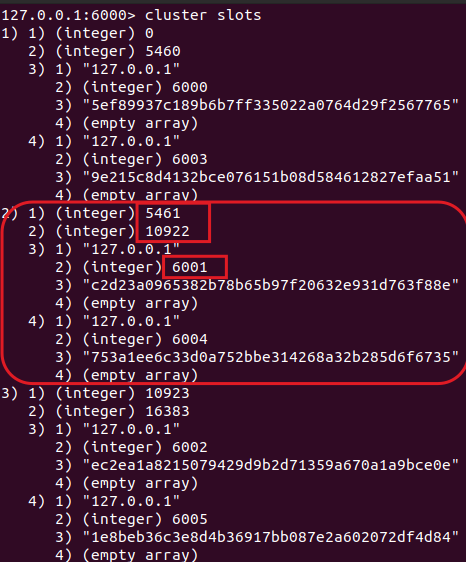
Now we can use the *KEYSLOT* subcommand to check what are the hash slots that each key has been assigned.

| cluster keyslot traveller:1:place |
| --- |

The output looks like the following.



It returned the integer 8963 which means the hash slot is 8963. It should lie in the range 5461 - 10922 shown in the following output.



The master node is 6001 and this key should be stored there. The same procedure can be followed to identify the node for *traveller:2:place* key.

## Example 02 - Handling Multiple Keys by the Same Redis Node

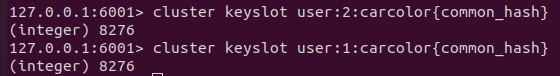
The *KEYSLOT* command supports hashtags where the Redis cluster key hashing algorithm hashes only the characters between the **{** and **}**. Whenever a Redis key follows the mentioned pattern, It will generate the hash slot based on the characters inside the curly braces, and the other characters are not considered. With the hashtags, we can force to store multiple keys in the same node as shown in the following.

Let’s create a couple of key value pairs as shown in the following.

| set user:1:carcolor{common\_hash} red set user:2:carcolor{common\_hash} blue |
| --- |

As shown in the example, the keys contain the common hashtag *{common\_hash}.* Hence, both the hash slot values which are generated by the Redis cluster key hashing algorithm will be the same for both keys. Let’s check the hash slots for both keys using the *KEYSLOT* subcommand as shown in the following.

| cluster keyslot user:2:carcolor{common\_hash} cluster keyslot user:1:carcolor{common\_hash} |
| --- |



As expected, both the commands return the same hash slot value 8276 which means keys are stored in the same node. As per the above examples, you can see how useful the *KEYSLOT* command is when dealing with Redis clusters.

# 

# 

# Conclusion

To summarize, Redis supports sharding, replication, and high availability by horizontal scaling. The horizontal scaling has been achieved using the Redis clusters. As discussed, each cluster is a collection of nodes where a node can hold several hash slots. A cluster can hold 16384 maximum hash slots and around 1000 nodes. As cited, Redis clusters follow the master-replica technique where a master node fails, and the replica will start to function on behalf of the failed node. Redis has a list of commands to operate on clusters. The *KEYSLOT* command is one of those which is used to get the hash slot value for a specific Redis key. This command supports the hash tags as well.