cluster-nodes

- Serialization format
- Notes on published config epochs
- Special slot entries

Each node in a Valkey Cluster has its view of the current cluster configuration, given by the set of known nodes, the state of the connection we have with such nodes, their flags, properties and assigned slots, and so forth.

`CLUSTER NODES` provides all this information, that is, the current cluster configuration of the node we are contacting, in a serialization format which happens to be exactly the same as the one used by the Valkey Cluster itself in order to store on disk the cluster state (however the on disk cluster state has a few additional info appended at the end).

Note that normally clients willing to fetch the map between Cluster hash slots and node addresses should use `CLUSTER SLOTS` instead. `CLUSTER NODES`, that provides more information, should be used for administrative tasks, debugging, and configuration inspections. It is also used by `valkey-cli` in order to manage a cluster.

Serialization format

The output of the command is just a space-separated CSV string, where each line represents a node in the cluster. The following is an example of output on Valkey 7.2.4.

```
07c37dfeb235213a872192d90877d0cd55635b91 127.0.0.1:30004@31004,hostname4 slave
e7d1eecce10fd6bb5eb35b9f99a514335d9ba9ca 0 1426238317239 4 connected
67ed2db8d677e59ec4a4cefb06858cf2a1a89fa1 127.0.0.1:30002@31002,hostname2 master - 0
1426238316232 2 connected 5461-10922
292f8b365bb7edb5e285caf0b7e6ddc7265d2f4f 127.0.0.1:30003@31003,hostname3 master - 0
1426238318243 3 connected 10923-16383
6ec23923021cf3ffec47632106199cb7f496ce01 127.0.0.1:30005@31005,hostname5 slave
67ed2db8d677e59ec4a4cefb06858cf2a1a89fa1 0 1426238316232 5 connected
824fe116063bc5fcf9f4ffd895bc17aee7731ac3 127.0.0.1:30006@31006,hostname6 slave
292f8b365bb7edb5e285caf0b7e6ddc7265d2f4f 0 1426238317741 6 connected
```

```
e7d1eecce10fd6bb5eb35b9f99a514335d9ba9ca 127.0.0.1:30001@31001,hostname1 myself,master - 0 0 1
connected 0-5460
```

Each line is composed of the following fields:

```
<id> <ip:port@cport[,hostname]> <flags> <master> <ping-sent> <pong-recv> <config-epoch> <link-
state> <slot> <slot> ... <slot>
```

The meaning of each field is the following:

1. `id`: The node ID, a 40-character globally unique string generated when a node is created and never changed again (unless `CLUSTER RESET HARD` is used).
2. `ip:port@cport`: The node address that clients should contact to run queries, along with the used cluster bus port. `:0@0` can be expected when the address is no longer known for this node ID, hence flagged with `noaddr`. Note that `ip` can be an IPv6 address, which can contain multiple colons. Here, an IPv6 address is *not* enclosed in square brackets as it would be in other contexts such as URIs. To properly split the `ip:port@cport` into `ip` and `port@cport`, you need to locate the *last* colon.
3. `hostname`: A human readable string that can be configured via the `cluster-annouce-hostname` setting. The max length of the string is 256 characters, excluding the null terminator. The name can contain ASCII alphanumeric characters, '-', and '.' only.
4. `flags`: A list of comma separated flags: `myself`, `master`, `slave`, `fail?`, `fail`, `handshake`, `noaddr`, `nofailover`, `noflags`. Flags are explained below.
5. `master`: If the node is a replica, and the primary is known, the primary node ID, otherwise the "-" character.
6. `ping-sent`: Unix time at which the currently active ping was sent, or zero if there are no pending pings, in milliseconds.
7. `pong-recv`: Unix time the last pong was received, in milliseconds.
8. `config-epoch`: The configuration epoch (or version) of the current node (or of the current primary if the node is a replica). Each time there is a failover, a new, unique, monotonically increasing configuration epoch is created. If multiple nodes claim to serve the same hash slots, the one with the higher configuration epoch wins.
9. `link-state`: The state of the link used for the node-to-node cluster bus. Use this link to communicate with the node. Can be `connected` or `disconnected`.
10. `slot`: A hash slot number or range. Starting from argument number 9, but there may be up to 16384 entries in total (limit never reached). This is the list of hash slots served by this node. If the entry is just a number, it is parsed as such. If it is a range, it is in the form `start-end`, and means that the node is responsible for all the hash slots from `start` to `end` including the start and end values.

Flags are:

- `myself`: The node you are contacting.
- `master`: Node is a primary.
- `slave`: Node is a replica.

- `fail?`: Node is in `PFAIL` state. Not reachable for the node you are contacting, but still logically reachable (not in `FAIL` state).
- `fail`: Node is in `FAIL` state. It was not reachable for multiple nodes that promoted the `PFAIL` state to `FAIL`.
- `handshake`: Untrusted node, we are handshaking.
- `noaddr`: No address known for this node.
- `nofailover`: Replica will not try to failover.
- `noflags`: No flags at all.

Notes on published config epochs

Replicas broadcast their primary's config epochs (in order to get an `UPDATE` message if they are found to be stale), so the real config epoch of the replica (which is meaningless more or less, since they don't serve hash slots) can be only obtained checking the node flagged as `myself`, which is the entry of the node we are asking to generate `CLUSTER NODES` output. The other replicas epochs reflect what they publish in heartbeat packets, which is, the configuration epoch of the primaries they are currently replicating.

Special slot entries

Normally hash slots associated to a given node are in one of the following formats, as already explained above:

1. Single number: 3894
2. Range: 3900-4000

However node hash slots can be in a special state, used in order to communicate errors after a node restart (mismatch between the keys in the AOF/RDB file, and the node hash slots configuration), or when there is a resharding operation in progress. This two states are **importing** and **migrating**.

The meaning of the two states is explained in the Valkey Specification, however the gist of the two states is the following:

- **Importing** slots are yet not part of the nodes hash slot, there is a migration in progress. The node will accept queries about these slots only if the `ASK` command is used.
- **Migrating** slots are assigned to the node, but are being migrated to some other node. The node will accept queries if all the keys in the command exist already, otherwise it will emit what is called an **ASK redirection**, to force new keys creation directly in the importing node.

Importing and migrating slots are emitted in the `CLUSTER NODES` output as follows:

- **Importing slot:** `[slot_number-<-importing_from_node_id]`
- **Migrating slot:** `[slot_number->-migrating_to_node_id]`

The following are a few examples of importing and migrating slots:

- `[93-<-292f8b365bb7edb5e285caf0b7e6ddc7265d2f4f]`
- `[1002-<-67ed2db8d677e59ec4a4cefb06858cf2a1a89fa1]`
- `[77->-e7d1eecce10fd6bb5eb35b9f99a514335d9ba9ca]`
- `[16311->-292f8b365bb7edb5e285caf0b7e6ddc7265d2f4f]`

Note that the format does not have any space, so `CLUSTER NODES` output format is plain CSV with space as separator even when these special slots are emitted. However a complete parser for the format should be able to handle them.

Note that:

1. Migration and importing slots are only added to the node flagged as `myself`. This information is local to a node, for its own slots.
2. Importing and migrating slots are provided as **additional info**. If the node has a given hash slot assigned, it will be also a plain number in the list of hash slots, so clients that don't have a clue about hash slots migrations can just skip these special fields.

> NOTE: If not for backward compatibility, the Valkey project no longer uses the words "master" and "slave". Unfortunately, in the given commands these words are part of the protocol, so we'll be able to remove such occurrences only when this API is naturally deprecated.

□□□ #1
□ □□□ □□□ 2026-10-31 15:12:12 UTC
□ □□□ □□□ 2026-10-31 15:12:12 UTC