Terminology in Redis Enterprise Software

Here are explanations of some of the terms used in Redis Enterprise Software.

Node

A *node* is a physical machine, virtual machine, container or cloud instance on which the RS installation package was installed and the setup process was run in order to make the machine part of the cluster.

Each node is a container for running multiple open source Redis instances, referred to as "shards".

The recommended configuration for a production cluster is an uneven number of nodes, with a minimum of three. Note that in some configurations, certain functionalities might be blocked. For example, if a cluster has only one node you cannot enable database replication, which helps to achieve high availability.

A node is made up of several components, as detailed below, and works together with the other cluster nodes.

Redis instance (shard)

As indicated above, each node serves as a container for hosting multiple database instances, referred to as "shards".

Redis Enterprise Software supports various database configurations:

- **Standard Redis database** - A single Redis shard with no replication or clustering.
- **Highly available Redis database** - Every database master shard has a replica shard, so that if the master shard fails the cluster can automatically fail over to the replica with minimal impact. Master and replica shards are always placed on separate nodes to ensure high availability.
- **Clustered Redis database** - The data stored in the database is split across several shards. The number of shards can be defined by the user. Various performance optimization algorithms define where shards are placed within the cluster. During the lifetime of the cluster, these algorithms might migrate a shard between nodes.
- **Clustered and highly available Redis database** - Each master shard in the clustered database has a replica shard, enabling failover if the master shard fails.

Proxy

Each node includes one zero-latency, multi-threaded proxy (written in low-level C) that masks the underlying system complexity. The proxy oversees forwarding Redis operations to the database shards on behalf of a Redis client.

The proxy simplifies the cluster operation, from the application or Redis client point of view, by enabling the use of a standard Redis client. The zero-latency proxy is built over a cut-through architecture and employs various optimization methods. For example, to help ensure high-throughput and low-latency performance, the proxy might use instruction pipelining even if not instructed to do so by the client.

Database endpoint

Each database is served by a database endpoint that is part of and managed by the proxies. The endpoint oversees forwarding Redis operations to specific database shards.
If the master shard fails and the replica shard is promoted to master, the master endpoint is updated to point to the new master shard.

If the master endpoint fails, the replica endpoint is promoted to be the new master endpoint and is updated to point to the master shard.

Similarly, if both the master shard and the master endpoint fail, then both the replica shard and the replica endpoint are promoted to be the new master shard and master endpoint.

Shards and their endpoints do not have to reside within the same node in the cluster.

In the case of a clustered database with multiple database shards, only one master endpoint acts as the master endpoint for all master shards, forwarding Redis operations to all shards as needed.

### Cluster manager

The cluster manager oversees all node management-related tasks, and the cluster manager in the master node looks after all the cluster related tasks.

The cluster manager is designed in a way that is totally decoupled from the Redis operation. This enables RS to react in a much faster and accurate manner to failure events, so that, for example, a node failure event triggers mass failover operations of all the master endpoints and master shards that are hosted on the failed node.

In addition, this architecture guarantees that each Redis shard is only dealing with processing Redis commands in a shared-nothing architecture, thus maintaining the inherent high-throughput and low-latency of each Redis process. Lastly, this architecture guarantees that any change in the cluster manager itself does not affect the Redis operation.

Some of the primary functionalities of the cluster manager include:

- Deciding where shards are created
- Deciding when shards are migrated and to where
- Monitoring database size
- Monitoring databases and endpoints across all nodes
- Running the database resharding process
- Running the database provisioning and de-provisioning processes
- Gathering operational statistics
- Enforcing license and subscription limitations

### Management UI

Each node runs a web server that is used to provide the user with access to the management user interface (UI). The management UI allows viewing and managing the entire cluster, so it does not matter which node is used for accessing the UI.

### REST API

Each node exposes a REST API to the cluster. Using the REST API, you can automate various tasks that are managed through the management UI. For example, using the REST API you can instruct the cluster to create a new database, update an existing database, or fetch various statistics.

**Updated:** September 23, 2021