Popescu Christian

[Email address]

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.   
When you’re ready to add your content, just click here and start typing.]

[Document title]

[Document subtitle]

**History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Content** |  |
|  |  |  |  |

**Table of Contents**

[1 Introduction 3](#_Toc460496671)

[2 Level 1 3](#_Toc460496672)

[2.1 Level 2 3](#_Toc460496673)

[2.1.1 Level 3 3](#_Toc460496674)

[2.1.1.1 Level 4 3](#_Toc460496675)

[2.1.1.1.1 Level 5 3](#_Toc460496676)

1. Introduction

<https://redis.io/docs/about/>

1. Redis is an open source (BSD licensed), in-memory **data structure store** used as a database, cache, message broker, and streaming engine. Redis provides [data structures](https://redis.io/docs/data-types/) such as [strings](https://redis.io/docs/data-types/strings/), [hashes](https://redis.io/docs/data-types/hashes/), [lists](https://redis.io/docs/data-types/lists/), [sets](https://redis.io/docs/data-types/sets/), [sorted sets](https://redis.io/docs/data-types/sorted-sets/) with range queries, [bitmaps](https://redis.io/docs/data-types/bitmaps/), [hyperloglogs](https://redis.io/docs/data-types/hyperloglogs/), [geospatial indexes](https://redis.io/docs/data-types/geospatial/), and [streams](https://redis.io/docs/data-types/streams/). Redis has built-in [replication](https://redis.io/topics/replication), [Lua scripting](https://redis.io/commands/eval), [LRU eviction](https://redis.io/docs/reference/eviction/), [transactions](https://redis.io/topics/transactions), and different levels of [on-disk persistence](https://redis.io/topics/persistence), and provides high availability via [Redis Sentinel](https://redis.io/topics/sentinel) and automatic partitioning with [Redis Cluster](https://redis.io/topics/cluster-tutorial).
2. You can run **atomic operations** on these types, like [appending to a string](https://redis.io/commands/append); [incrementing the value in a hash](https://redis.io/commands/hincrby); [pushing an element to a list](https://redis.io/commands/lpush); [computing set intersection](https://redis.io/commands/sinter), [union](https://redis.io/commands/sunion) and [difference](https://redis.io/commands/sdiff); or [getting the member with highest ranking in a sorted set](https://redis.io/commands/zrange).
3. To achieve top performance, Redis works with an **in-memory dataset**. Depending on your use case, Redis can persist your data either by periodically [dumping the dataset to disk](https://redis.io/topics/persistence#snapshotting) or by [appending each command to a disk-based log](https://redis.io/topics/persistence#append-only-file). You can also disable persistence if you just need a feature-rich, networked, in-memory cache.
4. Redis supports [asynchronous replication](https://redis.io/topics/replication), with fast non-blocking synchronization and auto-reconnection with partial resynchronization on net split.

Redis also includes:

* Transactions
* Pub/Sub
* Lua scripting
* Keys with a limited time-to-live
* LRU eviction of keys
* Automatic failover

1. Functionalities
   1. Pub/Sub

[SUBSCRIBE](https://redis.io/commands/subscribe), [UNSUBSCRIBE](https://redis.io/commands/unsubscribe) and [PUBLISH](https://redis.io/commands/publish) implement the [Publish/Subscribe messaging paradigm](http://en.wikipedia.org/wiki/Publish/subscribe) where senders (publishers) are not programmed to send their messages to specific receivers (subscribers).

Rather, published messages are characterized into **channels**, without knowledge of what (if any) subscribers there may be.

Subscribers express interest in one or more channels and only receive messages that are of interest, without knowledge of what (if any) publishers there are.

This decoupling of publishers and subscribers allows for greater scalability and a more dynamic network topology.

Redis' Pub/Sub exhibits ***at-most-once*** message delivery semantics. As the name suggests, it means that a message will be delivered once if at all. Once the message is sent by the Redis server, there's no chance of it being sent again. If the subscriber is unable to handle the message (for example, due to an error or a network disconnect) the message is forever lost.

**Note**. If your application requires stronger delivery guarantees, you may want to learn about [Redis Streams](https://redis.io/docs/data-types/streams-tutorial). Messages in streams are persisted, and support both ***at-most-once*** as well as ***at-least-once*** delivery semantics.

* 1. Scale with Redis Cluster

Redis Cluster provides a way to run a Redis installation where data is automatically sharded across multiple Redis nodes. Redis Cluster also provides some degree of availability during partitions—in practical terms, the ability to continue operations when some nodes fail or are unable to communicate. However, the cluster will become unavailable in the event of larger failures (for example, when the majority of masters are unavailable).

**Redis Cluster data sharding**

1. Redis Cluster does not use consistent hashing, but a different form of sharding where every key is conceptually part of what we call a **hash slot**.
2. There are 16384 hash slots in Redis Cluster, and to compute the hash slot for a given key, we simply take the CRC16 of the key modulo 16384.

**Redis Cluster master-replica model**

1. To remain available when a subset of master nodes are failing or are not able to communicate with the majority of nodes, Redis Cluster uses a master-replica model where every hash slot has from 1 (the master itself) to N replicas (N-1 additional replica nodes).
2. In our example cluster with nodes A, B, C, if node B fails the cluster is not able to continue, since we no longer have a way to serve hash slots in the range 5501-11000.
3. However, when the cluster is created (or at a later time), we add a replica node to every master, so that the final cluster is composed of A, B, C that are master nodes, and A1, B1, C1 that are replica nodes. This way, the system can continue if node B fails.
4. Node B1 replicates B, and B fails, the cluster will promote node B1 as the new master and will continue to operate correctly.
5. However, note that if nodes B and B1 fail at the same time, Redis Cluster will not be able to continue to operate.