EInnoVator
Software Engineering School

10

Redis Administration

Jorge Simão, Ph.D.

Content

Redis Overview

Redis is a **NoSql** data-structures store. It resembles a *key–value store* – allowing values to be set and retrieve by key, but in addition to simple strings values can also represent several built-in data-structures, including: *lists*, *hashes* (dictionaries), *sets*, and *sorted sets*.

A distinguish feature of **Redis** is that it loads the full state of the data-store into memory, thus allowing for very fast read&write access. Several techniques can be used to mitigate the memory footprint of **Redis** when the dataset grow beyond the physical memory limits – such as expiring not so much used keys, and partitioning (sharding) data across multiple nodes connected as a cluster.

Redis also supports for replication of data for increased fault-tolerance, using master-slave replication. High-availability and automatic failover is achieved by setting up a small cluster of dedicated monitoring servers known as *sentinels*.

Redis Installation

Redis is very straightforward to install. In a typical Linux installation or MacOS, it builds from the source code out-of-the-box without requiring any configuration. For Windows, there is a distribution managed by Microsoft, that currently has an MSI installer that sets up **Redis** as a Windows service. To start **Redis** manually, both in Linux and Windows, use command **redis-server**. By default, **Redis** listen for client connections on TCP/IP port **6379**. Depending on the OS, kernel version, and current kernel configuration, **Redis** might suggest on start up some OS-level settings for increased performance.

» Example: Installation Redis (Linux)

```
$ wget http://download.redis.io/releases/redis-3.2.0.tar.gz
$ tar -xzf redis-3.2.0.tar.gz
$ cd  redis-3.2.0
$ make
$ make install
```

» Example: Starting Redis

```
$ redis-server
```

```
[12536] 27 May 16:47:10.100 # Warning: no config file specified,
using the default config. In order to specify a config file use
```

```
redis-server /path/to/redis.conf
                _.-``__ ''-._
           _.-``    `.  `_.  ''-._           Redis 3.0.501 (00/0) 64 bit
       .-`` .-```.  ```\/    _.,_ ''-._
  (    '      ,       .-`  | `,    )     Running in standalone mode
  |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
  |    `-._   `._    /     _.-'    |     PID: 12536
   `-._    `-._  `-./  _.-'    _.-'
  |`-._`-._    `-.__.-'    _.-'_.-'|
  |    `-._`-._        _.-'_.-'    |           http://redis.io
   `-._    `-._`-.__.-'_.-'    _.-'
  |`-._`-._    `-.__.-'    _.-'_.-'|
  |    `-._`-._        _.-'_.-'    |
   `-._    `-._`-.__.-'_.-'    _.-'
       `-._    `-.__.-'    _.-'
           `-._        _.-'
               `-.__.-'

[12536] 27 May 16:47:10.124 # Server started, Redis version 3.0.501
[12536] 27 May 16:47:10.383 * DB loaded from disk: 0.258 seconds
[12536] 27 May 16:47:10.383 * The server is now ready to accept
connections on port 6379
```

Configuring Redis

Redis can be configured by starting the **redis-server** with the path to a *configuration file* as argument. The configuration file is a text file with a simple syntax – one setting per line, and each setting has a name followed by one or more arguments. The configuration directive **include <filename>** can be use to split the configuration into multiple files.

» Example: Starting Redis with Configuration File

```
redis-server /path/redis.com
```

» Example: Redis Configuration File

```
port 5000
logfile /var/log/redis.log
```

Table below summarizes some of the **Redis** configuration parameter. Later sections describe additional parameters.

Parameter	Description [default]
port <n>	TCP/IP port for client connections [6379]
bind <ip>	network interface IP to bind socket
unixsocket <filename> unixsocketperm <rwx>	Unix socket path and permission for local connections (e.g. 400)
timeout <sec>	client timeout before disconnect [0]
tcp-keepalive <sec>	Send keepalive msg to client (sec) [5]

pidfile <filename.pid>	Path to created PID file
loglevel <level>	Logging level (debug | verbose |notice | warning)
logfile <filename>	Log file path (e.g. /var/log/redis.log)
syslog-enabled no syslog-ident redis syslog-facility local0	Syslog configuration
databases <n>	Number of Databases (1–N) [16]

Redis CLI

Redis comes with a convenient command-line tool (CLI), that can be used to execute arbitrary **Redis** commands. The CLI support all the commands defined by **Redis** wire-level protocol (text-based). A useful command **CONFIG SET** allows configuration parameters to be change dynamically.

» Example: Starting Redis CLI Interactively

```
$ redis-cli
```

» Example: Execute Commands on Redis CLI

```
> KEYS *
```

» Example: Getting Help from Redis CLI

```
redis-cli --help
```

» Example: Execute Single Command and Return

```
redis-cli keys "*"
```

A single **Redis** server can manage multiple logical databases, up to the value of parameter **databases** (16 by default). Command **SELECT** is used by a client and the **Redis** CLI to select current database.

Table below summarizes some of the **Redis** server administration and configuration commands. (Companion **EInnovator Refcard Redis #9** describes many of the commands used for manipulating keys and data-structure values.)

Command	Description
CONFIG SET	Dynamically set a configuration parameter
CONFIG GET	Get value of configuration parameter
CONFIG REWRITE	Rewrite the configuration file
AUTH	Authenticate with password
SELECT	Switch to database (1-16)
FLUSHDB <db>	Remove all keys from specified database
FLUSH ALL	Remove all keys from all databases
DBSIZE	Get size in byte for current database
INFO	Show details about server
SHUTDOWN	Persistent data and stop server
CLIENT LIST	Get list of connected clients
CLIENT SETNAME CLIENT GETNAME	Set/Get symbolic name for this connection
CLIENT KILL <ip:port>	Kill specified client

Redis Persistence

Redis supports to modes of persistence – based on *snapshot* files (RDB), or *append-only* files (AOF). A snapshot file contains the complete dataset of **Redis** in a compact binary (possibly compressed) format. A snapshot is created by writing the state of **Redis** in-memory to the RDB file. Configuration directive **save** <*sec*> <*nwrites*>, defines a *save-point* (RDB saving rule) specifying that a new file should be created after <*sec*> seconds have pasted since last one, and at least <*nwrites*> write operations (e.g. **SET**, **DEL**, etc.) have been performed. A single save-point **save ""** disable RDB snapshots, and it useful when **Redis** is being used simply as a cache. The commands **SAVE** and **BGSAVE** (background save) can be used to manually request the writing of the RDB file. Consistency during file creation is guaranteed by writing the new RDB to a temporary file first, followed by an atomic **move/mv** operation.

By default, the RDB file is called **dump.rdb** and its stored in same directory as the server was started, but this can be changed with configuration parameters **dbfilename** and **dir**. Two parameters tailor the file fomat: **rdbcompression yes**, makes the file to be compressed in LZF compression, and **rdbchecksum** adds a CRC64 checksum to the end of the file used to check for file corruption at server start-up. Parameter **stop-writes-on-bgsave-error yes** instructs the server to stop accepting write operations if the last RDB snapshot could not be written in success.

» Example: redis.conf w/ Default RDB Settings

```
save 900 1
save 300 10
save 60 10000
stop-writes-on-bgsave-error yes
rdbcompression yes
rdbchecksum yes
dbfilename dump.rdb
dir ./
```

» Example: Manual Saving RDB File

```
> BGSAVE
Background saving started
[5560] 24 Aug 13:53:15.243 * Background saving started by pid 412
```

```
[5560] 24 Aug 13:53:15.696 # fork operation complete
[5560] 24 Aug 13:53:15.746 * Background saving terminated with success
```

In *append-only* (AOF) mode of persistence, the AOF file is appended to contain one entry per write operation that is performed on the server. When the server restarts, the AOF file is *replayed* to recreate the state of the database since last snapshot (RDB file written). The configuration parameter **appendonly yes** is used to enable AOF persistence (by default is off). The AOF file format is text-based (similarly to the wire-level client protocol of **Redis**). This allows for manual inspection and even modification of the file (e.g. remove erroneous operations).

Appending the AOF is not sufficient condition to guarantee the persistence of the data, since the OS file-system manager performs some caching and delays for some time hard-drive writes. The system-call **fsync()** is used to force the OS to *flush* the cache to the physical disk. The configuration parameter **appendfsync** is used to instruct **Redis** when to perform a **fsync()**. Three values are supported: **everysec** – flush every second (recommended), **always** (relatively slower), **no** (for high-performance requirements). With this last setting **appendfsync no** the OS has full control when the flush is performed (usually 30seconds in Linux system). Operations appended to the AOF file but not yet included in last flush, can be lost if the server crashes abruptly.

Since all write operations are recorded in a AOF file, the file can become large. To mitigate this, **Redis** supports AOF file rewrite in which multiple operations are replaced by smaller set that it is functionally equivalent (e.g. multiple consecutive **SET** on the same key can be replaced by only the last one, or multiple **INCR** operation can be replaced by a single **INCRBY**). The command **BGREWRITEAOF** instructs the AOF file to be rewritten on request. Automatic rewriting is also supported and controller by configuration parameter **auto-aof-rewrite-percentage** – specifying that AOF rewrite should be done when it grows by a certain percentage, and **auto-aof-rewrite-min-size** that sets a minimum file size for the rewrite to be done. By default, the AOF file is called **appendonly.aof** and its stored in same folder as the RDB file. This can be changed with configuration parameter **appendfilename**.

» Example: redis.conf w/ Default AOF Settings

```
appendonly no
appendfilename "appendonly.aof"
appendfsync everysec
no-appendfsync-on-rewrite no
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
aof-load-truncated yes
```

» Example: Manual Saving RDB File

```
> BGREWRITEAOF
Background append only file rewriting started
[5560] 24 Aug * Background append only file rewriting started by pid 1144
[5560] 24 Aug * AOF rewrite child asks to stop sending diffs.
[5560] 24 Aug # fork operation complete
[5560] 24 Aug * Background AOF rewrite terminated with success
[5560] 24 Aug * Residual parent diff successfully flushed to the rewritten
AOF (1.00 MB)
[5560] 24 Aug * Background AOF rewrite finished successfully
```

Table below summarizes **Redis** configuration parameters related with persistence (RDB & AOF).

RDB Config Parameter	Description [default]
save <sec> <nwrites>	Save-point rule
dbfilename <filename>	RDB file [*dump.rdb*]
dir <dirname>	Directory for the RDB & AOF files [.]
rdbcompression <yes|no>	Compress the RDB file [yes]
rdbchecksum <yes|no>	Add CRC64 checksum to end of file [yes]
stop-writes-on-bgsave-error <yes|no>	Stop accepting writes on error [yes]
AOF Config Parameter	**Description [default]**
appendonly <no|yes>	Enabled/Disable AOF persistence
appendfilename <filename>	AOF filename [*appendonly.aof]*
appendfsync <everysec| always|no>	When to perform fsync [*everysec*]
auto-aof-rewrite-percentage <%>	Auto-rewrite AOF when grows percentage [100]
auto-aof-rewrite-min-size	Min AOF size for auto-rewrite [*64mb]*
no-appendfsync-on-rewrite no	No fsync on AOF-rewrite | RDB-save
aof-load-truncated yes	Apply AOF even if file is tail truncated

Table below summarizes the **Redis** administration commands related with persistence.

Command	Description
SAVE	Save RDB File
BGSAVE	Background save RDB file
BGREWRITEAOF	Background save AOF file
LASTSAVE	Get *timestamp* of last successful save

Redis Master-Slave Replication

Redis supports *master-slave* replication for increased reliability and scale. A **Redis** server can be made a slave of another server with configuration parameter **slaveof <ip> <port>**.

Slaves contact the master at start-up, and the master send a snapshot with the current state. Any updates done on the server after the snapshot is sent, are propagated to the slaves *semi-synchronously* as separated commands, i.e. without the client of the master waiting. (In **Redis** documentation this is designated as *asynchronous*.) The parameter **repl-disable-tcp-nodelay yes** can also be used to instruct the OS of the master to send the TCP/IP packets immediate without waiting till buffers fill-up to make the propagation of operation faster.

Commonly (and by default) slaves are configured in *read-only* mode, since any updates done in a slave are not propagated to the master. The configuration parameter **slave-read-only** can be used to control this setting. By default slaves are configured to continue operation if the connection to the master is lost. This can be changed with parameter **slave-serve-stale-data**.

By default the master snapshot is written to disk as a regular RDB file before sending it to the slave. Alternatively, an in-memory snapshot can be sent directly to the slave without writing to the disk by setting configuration parameter **repl-diskless-sync yes**. Transfer of the RDB snapshot file to a slave is done by a child (forked) process, and the child needs to know before starting which slaves to make the transfer to. Additional slaves need to wait till the previous transfer(s) complete. The parameter **repl-diskless-sync-delay <sec>** can be set to make the server wait some time before forking the child and start the transfers, in the expectation that others slaves might arrive.

A slave sends a **PING** message to the master periodically to inform the server that its alive and reachable. The parameter **repl-ping-slave-period** defines the frequency of the **PING** message (10 seconds, by default). Parameter **repl-timeout** control the time before the master considers the slave unreachable, or a slave consider the master unreachable.

The master also maintains a buffer with recent operations. When a slave was only temporarily unreachable, the master checks if it is possible to send only the buffered operations rather than a full snapshot. Parameter **repl-backlog-size** and **repl-backlog-ttl** specifies the maximum length of this buffer and the time is kept.

For increased robustness, it also possible to configure the master to have a minimum number of available slaves in order to accept write operations. This is set with parameters **min-slaves-to-write**, and **min-slaves-max-lag** (time before last **PING** for slave to be considered available).

» Example: redis.conf of a Slave

```
slaveof 159.203.129.131 5000
slave-read-only yes
slave-serve-stale-data yes
repl-ping-slave-period 10
repl-timeout 60
```

» Example: redis.conf of a Master

```
repl-diskless-sync yes
repl-diskless-sync-delay 15
repl-timeout 60
repl-disable-tcp-nodelay yes
repl-backlog-size 1mb
repl-backlog-ttl 3600
min-slaves-to-write 2
min-slaves-max-lag 30
```

Table below summarizes **Redis** configuration parameters related with master-slave replication.

Replication Conf Param	Description [default]
slaveof <master-ip> <master-port>	Make server slave of specified master
masterauth <master-password>	Password to authenticated on master
slave-serve-stale-data <yes|no>	Accept commands even if link to master is broken
slave-read-only <yes|no>	Slaves not to accept write commands
repl-diskless-sync <no|yes>	Use in-memory replication snapshoot [no]
repl-diskless-sync-delay <sec>	Wait this time for additional slaves
repl-ping-slave-period	Frequency of PING sent to master [10]
repl-timeout <sec> [60]	Time to consider server unreachable
repl-disable-tcp-nodelay no	Don't wait to send commands to slave
repl-backlog-size <size> (1mb)	Buffer size for commands to send to slaves
repl-backlog-ttl <sec> (3600)	Delete command buffer after ms
slave-priority <priority> [100]	Slave priority in selecting new master
min-slaves-to-write <n>	Min reachable slaves to accept writes
min-slaves-max-lag <sec>	Last slave ping to be reachable
slave-announce-ip 5.5.5.5 slave-announce-port 1234	Announce this slave IP/port (container and NAT support)

Table below summarizes the **Redis** commands for replication management.

Command	Description
SLAVEOF <master-ip> <port>	Become slave of specified master
ROLE	Get current role of node (master | slave | sentinel)

Redis High-Availability with Sentinel

Redis can be setup as a highly-available distributed system by automatically detect that a master is down (or unreachable) and select a slave to become a new master. This is done by running some **Redis** servers in *Sentinel* mode, with command line argument **--sentinel**. A sentinel instance checks if one (or more) configured masters are reachable, and triggers a failover procedure to make a slave a new master when the previous master fails. To ensure true high-availability the sentinel node should also be replicated, and a minimum of 3 sentinels is required.

A configuration file needs to be provided when starting each sentinel. This is a file that describes which masters to monitor. The slaves of each master are automatically discovered when the sentinel connects to the master. Configuration parameter **sentinel monitor <name> <ip> <port> <quorum>** define a named master to be monitored. The quorum setting is the number of sentinels that need to agree (subjectively) that a master is down, to consider it (objectively) down. When master is agreed to be down, a coordination protocol takes places between the sentinels to select which slave to be the new master and initiate the failover. In case a sentinel goes down, is also excluded. Only the majority partition of sentinel nodes is allowed to perform the failover.

» Example: Sentinel Configuration File

```
port 5000
sentinel monitor mymaster 127.0.0.1 6379 2
sentinel down-after-milliseconds mymaster 60000
sentinel failover-timeout mymaster 180000
sentinel parallel-syncs mymaster 1
```

» Example: Starting Sentinels

```
> redis-server sentinel.conf --sentinel
```

Table below summarizes **Redis** configuration parameters related with high-availability using sentinels.

Sentinel Config Param	Description [default]
sentinel monitor <name> <ip> <port> <quorum>	Monitor named master in ip:port. Quorum agreement to consider master down.
sentinel down-after-milliseconds <name> <ms>	Time without PING to consider mast or other sentinel down (dflt:60000)
sentinel failover-timeout <name> <ms>	Maximum time to complete failover. (dflt:180000)
sentinel parallel-syncs <name> <n>	Max number of server informed about new master at same time (dflt:1)

Table below summarizes the **Redis** commands for high-availability management.

Command	Description
SENTINEL SET	Set sentinel configuration property dynamically
SENTINEL MASTER <name>	Show status of named master
SENTINEL GET-MASTER-ADDR-BY-NAME <name>	Get *ip:port* of current master (with given configuration name)
SENTINEL SLAVES <name>	Show slaves of named master
SENTINEL SENTINELS <name>	Show sentinels monitoring named master
SENTINEL RESET	Rest sentinel state & failover process
SENTINEL FAILOVER <name>	Force a failover on named master (select a slave as new master)
SENTINEL CKQUORUM <name>	Check if quorum for failure-detection and failover majority is achievable
SENTINEL FLUSHCONFIG	Force rewrite of sentinel config file

Redis Cluster

Redis servers can be run in a *cluster* mode where data is *sharded* (spread) across the different nodes. This is specially important in **Redis** since this allow to by-pass the limitation of available memory in nodes. Each stored key is assigned to one of 16K *buckets*, trough CRC16%16K hashing, and each node in the cluster is responsible to a configured set of buckets. All members of a cluster maintain a table with the location of each bucket. When a client tries to access a key (in a read or write operation), if the node the client is currently connected to does not hold the key it receive a *redirect* response pointing to the right node holding that particular key. The client is then expected to connect to the node holding the key. Although clients don't control the location the keys, the use of *hash-tags* of the form *prefix{tag}suffix* guarantees co-location for keys with the same *tag* value.

The configuration setting **cluster-enable yes** makes a node to start in cluster mode. Each node maintain a cluster configuration file that hold the current membership and state of the cluster as perceived by the node. The default name for this file is **nodes.conf**, but this can be changed with setting **cluster-config-file**. Each node in the cluster is assigned an unique ID, and this file contains the nodes IDs so the cluster state can be recovered even if the cluster is shutdown for some time.

Data bucket can also be replicated, by using master and slave nodes. Each slave will replicated all buckets managed by the corresponding master node. When a slave considers its master to be down, it will run a failover protocol to try become

the new master.

When a node starts first time it will not be connected to another cluster node. To connect the nodes in a cluster the command **CLUSTER MEET** can be used. More conveniently, the utility **redis-trib** is used with command **create**. When the cluster is created in this way, the assignment of buckets to nodes is done automatically by the tool. It is also possible to later move the location of buckets with command **redis-trib.rb reshard**, specifying the count of buckets to move and the source and destination nodes by identifier.

Client libraries and tools need to have support for clusters and follow the redirects send by nodes. The **Redis** CLI tool can be started with this feature enabled with command-line option **-c**.

» Example: redis.conf Settings for a Cluster Node

```
cluster-enabled yes
cluster-config-file nodes-6379.conf
cluster-node-timeout 15000
cluster-slave-validity-factor 10
cluster-migration-barrier 1
cluster-require-full-coverage yes
```

» Example: Starting a Redis Cluster (3Master + 3Slaves)

```
$ gem install redis

$ ./redis-trib.rb create --replicas 1 127.0.0.1:7000
127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:7003
127.0.0.1:7004 127.0.0.1:7005
```

» Example: Resharding Redis Cluster

```
SET NODE1=`redis-cli -p 7000 cluster nodes|grep myself |awk "{print $1}"`
SET NODE2=`redis-cli -p 7000 cluster nodes|sed -n '2p' |awk "{print $1}"`

$ ./redis-trib.rb reshard --from $NODE1 --to $NODE2
--slots 1000 --yes  127.0.0.1:7000
```

» Example: Starting Redis CLI in Cluster Mode

```
$ redis-cli -c -p 7000

> redis 127.0.0.1:7000> set user:eva:age 33

-> Redirected to slot [12182] located at 127.0.0.1:7002
OK
```

A cluster membership can also be modified dynamically with command **redis-trib add-node**. A node newly added as a master will not be responsible to manage any bucket. So resharding should be done to make the node useful. New slaves nodes can also be added dynamically with option **add-node.--slave**.

» Example: Add New Node To Cluster – As Master

```
./redis-trib.rb add-node 127.0.0.1:7006 127.0.0.1:7000

SET NODE7=`redis-cli -p 7000 cluster nodes|sed -n '7p' |awk "{print $1}"`

$ ./redis-trib.rb reshard --from $NODE0 --to $NODE7
--slots 1000 --yes  127.0.0.1:7000
```

» Example: Add New Node To Cluster – As Slave

```
$ ./redis-trib.rb add-node --slave 127.0.0.1:7006
127.0.0.1:7000
```

» Example: Remove Node From Cluster

```
$ ./redis-trib del-node 127.0.0.1:7000 $NODE3
```

Table below summarizes **Redis** configuration parameters related with cluster management.

Cluster Config Param	Description [default]
cluster-enabled <yes|no>	Enable cluster model [no]
cluster-config-file <file>	Cluster state file [nodes.conf]
cluster-node-timeout <ms>	Timeout to consider node down [15000]
cluster-slave-validity-factor <n>	Timeout factor to promote slave to master [10]
cluster-migration-barrier <n>	Min slaves up for a master before change master[1]
cluster-require-full-coverage <yes|no>	Continue to operate the cluster if some buckets are unavailable [yes]

Table below summarizes some of the **Redis** commands for clustering management.

Command	Description
CLUSTER MEET <ip> <port>	Join cluster through specified node
CLUSTER NODES	Show list of cluster node
CLUSTER SLAVES <id>	Show list of slaves for master with ID
CLUSTER REPLICATE <id>	Make node slave of a master
CLUSTER FORGET <id>	Remove node from cluster
CLUSTER SLOTS	Show details of assigned slot→node
CLUSTER ADDSLOTS <slot>*	Add unassigned slots to connected node

Client Management & Limits

Several configuration parameters define limits and control **Redis** behavior under heave-load. The maximum number of allowed client connection is set with **maxclients**. Parameter **maxmemory** specifies the maximum amount of memory to use. If memory is exhausted, memory commands that would require more memory (e.g. SET on a new key) return an error. Parameter **maxmemory-policy** can also be use to discard keys and release memory when needed. Setting to a value different from the default **noeviction,** will make the server discard keys according to the defined policy (e.g. LRU). When **Redis** is used as a cache the used setting is **allkeys-lru**.

Setting **notify-keyspace-events** is used to specify which server events produce message in **keyspace** and **keyevent** channels (**__keyspace@<db>__:*key cmd***, **__keyevent@<db>__:*cmd key***).

Table below summarizes **Redis** configuration parameters related with clients and connection management and limits.

Clients Config Param	Description [default]
maxclients <n>	[10000]
maxmemory <bytes>	Max memory to use
maxmemory-policy <noeviction | volatile-lru | allkeys-lru | volatile-random | allkeys-random | volatile-ttl>	Key discarding & memory release policy [noeviction]
maxmemory-samples <n>	Key samples for LRU [5]
lua-time-limit <ms>	Max executation time of script before warning [5000]
activerehashing <yes|no>	Active rehash main hash table [yes]
hash-max-ziplist-entries [512] hash-max-ziplist-value [64] list-max-ziplist-size [2] set-max-intset-entries [512] zset-max-ziplist-entries [128] zset-max-ziplist-value [64] hll-sparse-max-bytes [3000]	Limits to us compressed (Zip) representation for data-structures
list-compress-depth <k>	List compression strategy [0]
notify-keyspace-events	Notification policy for server events
<policy>	[“”]

Redis Security

Redis can be configured with a basic level of security, with parameter **requirepass** defining a password that client need to specify with command **AUTH**. When running in untrusty environments the directive **rename-command** in the configuration file can be used to prevent intruder to issue a command. Table below summarizes **Redis** configuration parameters and directives related with security.

Security Conf Param	Description [default]
requirepass <passwd>	Password to authenticate
rename-command <command> <newname>	Rename command

Resources

- Redis Project home page: http://redis.io/
- Redis sample configuration file: https://raw.githubusercontent.com/antirez/redis/3.2/redis.conf
- Redis sample sentinel configuration file: https://raw.githubusercontent.com/antirez/redis/3.2/sentinel.conf

About the Author

Jorge Simão is a software engineer and IT Trainer, with two decades long experience on education delivery both in academia and industry. Expert in a wide range of computing topics, he his an author, trainer, and director (Education & Consulting) at **EInnovator**. He holds a B.Sc., M.Sc., and Ph.D. in Computer Science and Engineering.

Redis Training

Redis Administration course teaches how to install, manage, and configure Redis. Covers topics such as Redis master-slave replication, high-availability, and Redis cluster. Book for a training event in a date&location of your choice: **www.einnovator.org/course/redis-administration**

Redis Development course teaches how build applications that use Redis, with extensive coverage of Spring Data Redis (Java track) & C# drivers (.Net track). Book for a training event in a date&location of your choice: **www.einnovator.org/course/redis-development**

Contacts

Training – Bookings & Inquiries
training@einnovator.org

Consultancy – Partnerships & Inquiries
consulting@einnovator.org

General Info
info@einnovator.org