# How To Install and Secure Redis on Debian 9



Posted September 5, 2018 13.3k REDIS NOSQL DEBIAN DEBIAN 9

By: Justin Ellingwood By: Mark Drake By: Brian Boucheron

Not using **Debian 9**? Choose a different version:

CentOS 7

Ubuntu 18.04

Ubuntu 16.04

## Introduction

Redis is an in-memory key-value store known for its flexibility, performance, and wide language support. This tutorial demonstrates how to install, configure, and secure Redis on a Debian 9 server.

## Prerequisites

To complete this guide, you will need access to a Debian 9 server that has a non-root user with `sudo` privileges and a basic firewall configured. You can set this up by following our Initial Server Setup guide.

When you are ready to begin, log in to your server as your sudo-enabled user and continue below.

## Step 1 — Installing and Configuring Redis

In order to get the latest version of Redis, we will use `apt` to install it from the official Debian repositories.

Update your local `apt` package cache and install Redis by typing:

```
$ sudo apt update
$ sudo apt install redis-server
```

This will download and install Redis and its dependencies. Following this, there is one important configuration change to make in the Redis configuration file, which was generated automatically during the installation.

Open this file with your preferred text editor:

```
$ sudo nano /etc/redis/redis.conf
```

Inside the file, find the `supervised` directive. This directive allows you to declare an init system to manage Redis as a service, providing you with more control over its operation. The `supervised` directive is set to `no` by default. Since you are running Debian, which uses the systemd init system, change this to `systemd`:

/etc/redis/redis.conf

```
. . .

# If you run Redis from upstart or systemd, Redis can interact with your
# supervision tree. Options:
#   supervised no      - no supervision interaction
#   supervised upstart - signal upstart by putting Redis into SIGSTOP mode
#   supervised systemd - signal systemd by writing READY=1 to $NOTIFY_SOCKET
#   supervised auto    - detect upstart or systemd method based on
#                        UPSTART_JOB or NOTIFY_SOCKET environment variables
# Note: these supervision methods only signal "process is ready."
#       They do not enable continuous liveness pings back to your supervisor.
supervised systemd

. . .
```

That's the only change you need to make to the Redis configuration file at this point, so save and close it when you are finished. Then, reload the Redis service file to reflect the changes you made to the configuration file:

```
$ sudo systemctl restart redis
```

With that, you've installed and configured Redis and it's running on your machine. Before you begin using it, though, it's prudent to first check whether Redis is functioning correctly.

## Step 2 — Testing Redis

As with any newly-installed software, it's a good idea to ensure that Redis is functioning as expected before making any further changes to its configuration. We will go over a handful of ways to check that Redis is working correctly in this step.

Start by checking that the Redis service is running:

```
$ sudo systemctl status redis
```

If it is running without any errors, this command will produce output similar to the following:

```
Output
● redis-server.service - Advanced key-value store
   Loaded: loaded (/lib/systemd/system/redis-server.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2018-09-05 20:19:44 UTC; 41s ago
     Docs: http://redis.io/documentation,
           man:redis-server(1)
  Process: 10829 ExecStopPost=/bin/run-parts --verbose /etc/redis/redis-server.post-down.d (code=exit
  Process: 10825 ExecStop=/bin/kill -s TERM $MAINPID (code=exited, status=0/SUCCESS)
  Process: 10823 ExecStop=/bin/run-parts --verbose /etc/redis/redis-server.pre-down.d (code=exited, s
  Process: 10842 ExecStartPost=/bin/run-parts --verbose /etc/redis/redis-server.post-up.d (code=exite
  Process: 10838 ExecStart=/usr/bin/redis-server /etc/redis/redis.conf (code=exited, status=0/SUCCESS
  Process: 10834 ExecStartPre=/bin/run-parts --verbose /etc/redis/redis-server.pre-up.d (code=exited,
 Main PID: 10841 (redis-server)
    Tasks: 3 (limit: 4915)
   CGroup: /system.slice/redis-server.service
           └─10841 /usr/bin/redis-server 127.0.0.1:6379
. . .
```

Here, you can see that Redis is running and is already enabled, meaning that it is set to start up every time the server boots.

**Note:** This setting is desirable for many common use cases of Redis. If, however, you prefer to start up Redis manually every time your server boots, you can configure this with the following command:

```
$ sudo systemctl disable redis
```

To test that Redis is functioning correctly, connect to the server using the command-line client:

```
$ redis-cli
```

In the prompt that follows, test connectivity with the `ping` command:

```
127.0.0.1:6379> ping
```

```
Output
PONG
```

This output confirms that the server connection is still alive. Next, check that you're able to set keys by running:

```
127.0.0.1:6379> set test "It's working!"
```

```
Output
OK
```

Retrieve the value by typing:

```
127.0.0.1:6379> get test
```

Assuming everything is working, you will be able to retrieve the value you stored:

```
Output
"It's working!"
```

After confirming that you can fetch the value, exit the Redis prompt to get back to the shell:

```
127.0.0.1:6379> exit
```

As a final test, we will check whether Redis is able to persist data even after it's been stopped or restarted. To do this, first restart the Redis instance:

```
$ sudo systemctl restart redis
```

Then connect with the command-line client once again and confirm that your test value is still available:

```
$ redis-cli
```

```
127.0.0.1:6379> get test
```

The value of your key should still be accessible:

```
Output
"It's working!"
```

Exit out into the shell again when you are finished:

```
127.0.0.1:6379> exit
```

With that, your Redis installation is fully operational and ready for you to use. However, some of its default configuration settings are insecure and provide malicious actors with opportunities to attack and gain access to your server and its data. The remaining steps in this tutorial cover methods for mitigating these vulnerabilities, as prescribed by the official Redis website. Although these steps are optional and Redis will still function if you choose not to follow them, it is *strongly* recommended that you complete them in order to harden your system's security.

## Step 3 — Binding to localhost

By default, Redis is only accessible from **localhost**. However, if you installed and configured Redis by following a different tutorial than this one, you might have updated the configuration file to allow connections from anywhere. This is not as secure as binding to **localhost**.

To correct this, open the Redis configuration file for editing:

```
$ sudo nano /etc/redis/redis.conf
```

Locate this line and make sure it is uncommented (remove the `#` if it exists):

/etc/redis/redis.conf

```
bind 127.0.0.1
```

Save and close the file when finished (press `CTRL + X`, `Y`, then `ENTER`).

Then, restart the service to ensure that systemd reads your changes:

```
$ sudo systemctl restart redis
```

To check that this change has gone into effect, run the following `netstat` command:

```
$ sudo netstat -lnp | grep redis
```

```
Output
tcp        0      0 127.0.0.1:6379          0.0.0.0:*               LISTEN      10959/redis-server
```

This output shows that the `redis-server` program is bound to **localhost** (`127.0.0.1`), reflecting the change you just made to the configuration file. If you see another IP address in that colu

example), then you should double check that you uncommented the correct line and restart the Redis service again.

Now that your Redis installation is only listening in on **localhost**, it will be more difficult for malicious actors to make requests or gain access to your server. However, Redis isn't currently set to require users to authenticate themselves before making changes to its configuration or the data it holds. To remedy this, Redis allows you to require users to authenticate with a password before making changes via the Redis client ( `redis-cli` ).

## Step 4 — Configuring a Redis Password

Configuring a Redis password enables one of its two built-in security features — the `auth` command, which requires clients to authenticate to access the database. The password is configured directly in Redis's configuration file, `/etc/redis/redis.conf` , so open that file again with your preferred editor:

```
$ sudo nano /etc/redis/redis.conf
```

Scroll to the `SECURITY` section and look for a commented directive that reads:

/etc/redis/redis.conf

```
# requirepass foobared
```

Uncomment it by removing the `#` , and change `foobared` to a secure password.

**Note:** Above the `requirepass` directive in the `redis.conf` file, there is a commented warning:

```
# Warning: since Redis is pretty fast an outside user can try up to
# 150k passwords per second against a good box. This means that you should
# use a very strong password otherwise it will be very easy to break.
#
```

Thus, it's important that you specify a very strong and very long value as your password. Rather than make up a password yourself, you can use the `openssl` command to generate a random one, as in the following example. By piping the output of the first command to the second `openssl` command, as shown here, it will remove any line breaks produced by that the first command:

```
$ openssl rand 60 | openssl base64 -A
```

Your output should look something like:

SCROLL TO TOP

```
Output
```

RBOJ9cCNoGCKhlEBwQLHri1g+atWgn4Xn4HwNUbtzoVxAYxkiYBi7aufl4MILv1nxBqR4L6NNzI0X6cE

After copying and pasting the output of that command as the new value for `requirepass`, it should read:

```
/etc/redis/redis.conf
requirepass RBOJ9cCNoGCKhlEBwQLHri1g+atWgn4Xn4HwNUbtzoVxAYxkiYBi7aufl4MILv1nxBqR4L6NNzI0X6cE
```

After setting the password, save and close the file, then restart Redis:

```
$ sudo systemctl restart redis.service
```

To test that the password works, access the Redis command line:

```
$ redis-cli
```

The following shows a sequence of commands used to test whether the Redis password works. The first command tries to set a key to a value before authentication:

```
127.0.0.1:6379> set key1 10
```

That won't work because you didn't authenticate, so Redis returns an error:

```
Output
(error) NOAUTH Authentication required.
```

The next command authenticates with the password specified in the Redis configuration file:

```
127.0.0.1:6379> auth your_redis_password
```

Redis acknowledges:

```
Output
OK
```

After that, running the previous command again will succeed:

```
127.0.0.1:6379> set key1 10
```

```
Output
OK
```

`get key1` queries Redis for the value of the new key.

```
127.0.0.1:6379> get key1
```

```
Output
"10"
```

After confirming that you're able to run commands in the Redis client after authenticating, you can exit the `redis-cli`:

```
127.0.0.1:6379> quit
```

Next, we'll look at renaming Redis commands which, if entered by mistake or by a malicious actor, could cause serious damage to your machine.

## Step 5 — Renaming Dangerous Commands

The other security feature built into Redis involves renaming or completely disabling certain commands that are considered dangerous.

When run by unauthorized users, such commands can be used to reconfigure, destroy, or otherwise wipe your data. Like the authentication password, renaming or disabling commands is configured in the same `SECURITY` section of the `/etc/redis/redis.conf` file.

Some of the commands that are considered dangerous include: **FLUSHDB**, **FLUSHALL**, **KEYS**, **PEXPIRE**, **DEL**, **CONFIG**, **SHUTDOWN**, **BGREWRITEAOF**, **BGSAVE**, **SAVE**, **SPOP**, **SREM**, **RENAME**, and **DEBUG**. This is not a comprehensive list, but renaming or disabling all of the commands in that list is a good starting point for enhancing your Redis server's security.

Whether you should disable or rename a command depends on your specific needs or those of your site. If you know you will never use a command that could be abused, then you may disable it. Otherwise, it might be in your best interest to rename it.

To enable or disable Redis commands, open the configuration file once more:

```
$ sudo nano  /etc/redis/redis.conf
```

**Warning:** The following steps showing how to disable and rename commands are examples choose to disable or rename the commands that make sense for you. You can review the full list of

commands for yourself and determine how they might be misused at redis.io/commands.

To disable a command, simply rename it to an empty string (signified by a pair of quotation marks with no characters between them), as shown below:

/etc/redis/redis.conf

```
. . .
# It is also possible to completely kill a command by renaming it into
# an empty string:
#
rename-command FLUSHDB ""
rename-command FLUSHALL ""
rename-command DEBUG ""
. . .
```

To rename a command, give it another name as shown in the examples below. Renamed commands should be difficult for others to guess, but easy for you to remember:

/etc/redis/redis.conf

```
. . .
# rename-command CONFIG ""
rename-command SHUTDOWN SHUTDOWN_MENOT
rename-command CONFIG ASC12_CONFIG
. . .
```

Save your changes and close the file.

After renaming a command, apply the change by restarting Redis:

```
$ sudo systemctl restart redis
```

To test the new command, enter the Redis command line:

```
$ redis-cli
```

Then, authenticate:

```
127.0.0.1:6379> auth your_redis_password
```

Output

```
OK
```

Let's assume that you renamed the `CONFIG` command to `ASC12_CONFIG`, as in the preceding example. First, try using the original `CONFIG` command. It should fail, because you've renamed it:

```
127.0.0.1:6379> config get requirepass
```

Output

```
(error) ERR unknown command 'config'
```

Calling the renamed command, however, will be successful. It is not case-sensitive:

```
127.0.0.1:6379> asc12_config get requirepass
```

Output

```
1) "requirepass"
2) "your_redis_password"
```

Finally, you can exit from `redis-cli`:

```
127.0.0.1:6379> exit
```

Note that if you're already using the Redis command line and then restart Redis, you'll need to re-authenticate. Otherwise, you'll get this error if you type a command:

Output

```
NOAUTH Authentication required.
```

Regarding the practice of renaming commands, there's a cautionary statement at the end of the `SECURITY` section in `/etc/redis/redis.conf` which reads:

> `Please note that changing the name of commands that are logged into the AOF file or transmitted to slaves may cause problems.`

***Note:*** *The Redis project chooses to use the terms "master" and "slave" while DigitalOcean generally prefers alternative descriptors. In order to avoid confusion we've chosen to use the terms used in the Redis documentation here.*

That means if the renamed command is not in the AOF file, or if it is but the AOF file has not been transmitted to slaves, then there should be no problem.

So, keep that in mind when you're trying to rename commands. The best time to rename a command is when you're not using AOF persistence, or right after installation, that is, before your Redis-using application has been deployed.

When you're using AOF and dealing with a master-slave installation, consider this answer from the project's GitHub issue page. The following is a reply to the author's question:

> The commands are logged to the AOF and replicated to the slave the same way they are sent, so if you try to replay the AOF on an instance that doesn't have the same renaming, you may face inconsistencies as the command cannot be executed (same for slaves).

Thus, the best way to handle renaming in cases like that is to make sure that renamed commands are applied to all instances in master-slave installations.

## Conclusion

In this tutorial, you installed and configured Redis, validated that your Redis installation is functioning correctly, and used its built-in security features to make it less vulnerable to attacks from malicious actors.

Keep in mind that once someone is logged in to your server, it's very easy to circumvent the Redis-specific security features we've put in place. Therefore, the most important security feature on your Redis server is your firewall (which you configured if you followed the prerequisite Initial Server Setup tutorial), as this makes it extremely difficult for malicious actors to jump that fence.

By: Justin Ellingwood By: Mark Drake By: Brian Boucheron

## Related Tutorials

How To Install Redis from Source on Ubuntu 18.04

How to Secure Your Redis Installation on Ubuntu 18.04

How to Install MongoDB on Ubuntu 18.04

How To Install and Secure Redis on Ubuntu 18.04

How to Install and Secure Redis on Centos7

# 0 Comments

Leave a comment...

Log In to Comment