

How To Use ProxySQL as a Load Balancer for MySQL on Ubuntu 16.04



Posted January 8, 2018 32k

MYSQL

LOAD BALANCING

DATABASES

CLUSTERING

HIGH AVAILABILITY

By: Mateusz Papiernik

Introduction

ProxySQL is an open-source MySQL proxy server, meaning it serves as an intermediary between a MySQL server and the applications that access its databases. ProxySQL can improve performance by distributing traffic among a pool of multiple database servers and also improve availability by automatically failing over to a standby if one or more of the database servers fail.

In this guide, you will set up ProxySQL as a load balancer for multiple MySQL servers with automatic failover. As an example, this tutorial uses a multi-primary replicated cluster of three MySQL servers, but you can use a similar approach with other cluster configurations as well.

Prerequisites

To follow this tutorial, you will need:

- One Ubuntu 16.04 server set up with this initial Ubuntu 16.04 server setup tutorial, including a sudo non-root user and a firewall. This server will become your ProxySQL instance.
- Three MySQL servers configured to form a multi-primary replication group. You can set this up by following the How To Configure MySQL Group Replication on Ubuntu 16.04 tutorial. In the **Choosing Single Primary or Multi-Primary** section, follow the instructions for a **multi-primary** replication group.

Step 1 — Installing ProxySQL

The developers of ProxySQL provide official Ubuntu packages for all ProxySQL releases on their GitHub releases page, so we'll download the latest package version from there and install it.

You can find the latest package on the release list. The naming convention is `proxysql_`**version-distribution**.deb, where `version` is a string like `1.4.4` for version 1.4.4, and `distribution` is a string like `ubuntu16_amd64` for 64-bit Ubuntu 16.04.

Download the latest official package, which is 1.4.4 at the time of writing, into the `/tmp` directory.

```
$ cd /tmp
```

```
$ curl -OL https://github.com/sysown/proxysql/releases/download/v1.4.4/proxysql_1.4.4-ubuntu16_amd64
```

Install the package with `dpkg`, which is used to manage .deb software packages. The `-i` flag indicates that we'd like to install from the specified file.

```
$ sudo dpkg -i proxysql_*
```

At this point, you no longer need the `.deb` file, so you can remove it.

```
$ rm proxysql_*
```

Next, we'll need a MySQL client application to connect to the ProxySQL instance. This is because ProxySQL internally uses a MySQL-compatible interface for administrative tasks. We'll use is the `mysql` command line tool, which is part of the `mysql-client` package available in the Ubuntu repositories.

Update your package repository to make sure you're getting the latest pre-bundled version, then install the `mysql-client` package.

```
$ sudo apt-get update
```

```
$ sudo apt-get install mysql-client
```

You now have all of the requirements to run ProxySQL, but the service doesn't automatically start after installation, so start it manually now.

```
$ sudo systemctl start proxysql
```

ProxySQL should now be running with its default configuration in place. You can check using `systemctl`.

```
$ systemctl status proxysql
```

The output will look similar to this:

Output

- `proxysql.service` - LSB: High Performance Advanced Proxy for MySQL
Loaded: loaded (/etc/init.d/proxysql; bad; vendor preset: enabled)
Active: **active (running)** since Thu 2017-12-21 19:19:20 UTC; 5s ago
Docs: man:systemd-sysv-generator(8)
Process: 12350 ExecStart=/etc/init.d/proxysql start (code=exited, status=0/SUCCESS)

```
Tasks: 23
Memory: 30.9M
CPU: 86ms
CGroup: /system.slice/proxysql.service
├─12355 proxysql -c /etc/proxysql.cnf -D /var/lib/proxysql
└─12356 proxysql -c /etc/proxysql.cnf -D /var/lib/proxysql
```

The **active (running)** line means ProxySQL is installed and running.

Next, we'll increase security by setting the password used to access ProxySQL's administrative interface.

Step 2 — Setting the ProxySQL Administrator Password

The first time you start a new ProxySQL installation, it uses a package-provided configuration file to initialize default values for all of its configuration variables. After this initialization, ProxySQL stores its configuration in a database which you can manage and modify via the command line.

To set the administrator password in ProxySQL, we'll connect to that configuration database and update the appropriate variables.

First, access the administration interface. You'll be prompted for password which, on a default installation, is **admin**.

```
$ mysql -u admin -p -h 127.0.0.1 -P 6032 --prompt='ProxySQLAdmin> '
```

- `-u` specifies the user we want to connect as, which here is **admin**, the default user for administrative tasks like changing configuration settings.
- `-h 127.0.0.1` tells `mysql` to connect to the local ProxySQL instance. We need to define this explicitly because ProxySQL doesn't listen on the socket file that `mysql` assumes by default.
- `-P` specifies the port to connect to. ProxySQL's admin interface listens on `6032`.
- `--prompt` is an optional flag that changes the default prompt, which is normally `mysql>`. Here, we're changing it to `ProxySQLAdmin>` to make it clear that we're connected to the ProxySQL administration interface. This will be helpful to avoid confusion later on when we'll also be connecting to the MySQL interfaces on the replicated database servers..

Once you connect, you will see the `ProxySQLAdmin>` prompt:

```
ProxySQL administration console prompt
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.30 (ProxySQL Admin Module)
```

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

ProxySQLAdmin>

Change the administrative account password by updating (UPDATE) the `admin-admin_credentials` configuration variable in the `global_variables` database. Remember to change `password` in the command below to a strong password of your choice.

```
ProxySQLAdmin> UPDATE global_variables SET variable_value='admin:password' WHERE variable_name='admin_credentials'
```

Output

Query OK, 1 row affected (0.00 sec)

This change won't take immediate effect because of how ProxySQL's configuration system works. It consists of three separate layers:

- **memory**, which is altered when making modifications from the command-line interface.
- **runtime**, which is used by ProxySQL as the effective configuration.
- **disk**, which is used to make a configuration persist across restarts.

Right now, the change you made is in **memory**. To put the change into effect, you have to copy the **memory** settings to the **runtime** realm, then save them to **disk** to make them persist.

```
ProxySQLAdmin> LOAD ADMIN VARIABLES TO RUNTIME;
```

```
ProxySQLAdmin> SAVE ADMIN VARIABLES TO DISK;
```

These `ADMIN` commands handle only variables related to the administrative command-line interface. ProxySQL exposes similar commands, like `MYSQL`, to handle other parts of its configuration. We'll use these later in this tutorial.

Now that ProxySQL is installed and running with a new admin password, let's set up the 3 MySQL nodes so that ProxySQL can monitor them. Keep the ProxySQL interface open, though, because we'll use it later on.

Step 3 — Configuring Monitoring in MySQL

ProxySQL needs to communicate with the MySQL nodes to be able to assess their condition. To do that, it has to be able to connect to each server with a dedicated user.

Here, we will configure the necessary user on the MySQL nodes and install additional SQL functions that allow ProxySQL to query the group replication state.

Because MySQL group replication is already running, the following steps must be performed only on a **single member of the group**.

In a second terminal, log into a server with one of the MySQL nodes.

```
$ ssh sammy@your_mysql_server_ip_1
```

Download the SQL file containing some necessary functions for ProxySQL group replication support to work.

```
$ curl -OL https://gist.github.com/lefred/77ddbde301c72535381ae7af9f968322/raw/5e40b03333a3c148b78aa
```

Note: This file is provided by ProxySQL authors, but in an ad-hoc way: it's a gist in a personal GitHub repository, which means it's possible that it will move or become out of date. In the future, it may be added as a versioned file in the official ProxySQL repository.

You can read more about the context for and contents of this file in the author's blog post about native ProxySQL support for MySQL group replication.

You can view the contents of the file using `less addition_to_sys.sql`.

When you're ready, execute the commands in the file. You will be prompted for the MySQL administrative password.

```
$ mysql -u root -p < addition_to_sys.sql
```

If the command runs successfully, it will produce no output. In that case, all MySQL nodes will now expose the necessary functions for ProxySQL to recognize group replication status.

Next, we have to create a dedicated user that will be used by ProxySQL to monitor health of the instances.

Open the MySQL interactive prompt, which will prompt you for the **root** password again.

```
$ mysql -u root -p
```

Then create the dedicated user, which we called **monitor** here. Make sure to change the password to a strong one.

```
(member1) mysql> CREATE USER 'monitor'@'%' IDENTIFIED BY 'monitorpassword';
```

Grant the user privileges to query the MySQL server's condition to the **monitor** user.

```
(member1) mysql> GRANT SELECT on sys.* to 'monitor'@'%';
```

Finally, apply the changes.

```
(member1) mysql> FLUSH PRIVILEGES;
```

Because of group replication, once you've finished adding the user for health monitoring to one MySQL node, it will be fully configured on all three nodes.

Next, we need to update ProxySQL with the information for that user so it can access the MySQL nodes.

Step 4 — Configuring Monitoring in ProxySQL

To configure ProxySQL to use the new user account when monitoring nodes, we'll **UPDATE** the appropriate configuration variable. This is very similar to the way we set the admin password from Step 2.

Back in the ProxySQL admin interface, update the `mysql-monitor_username` variable to the username of the new account.

```
ProxySQLAdmin> UPDATE global_variables SET variable_value='monitor' WHERE variable_name='mysql-monitor_username';
```

Just like before, the configuration is not automatically applied, so migrate it into **runtime** and save to **disk**. This time, notice that we're using **MYSQL** instead of **ADMIN** to update these variables because we're modifying MySQL configuration variables.

```
ProxySQLAdmin> LOAD MYSQL VARIABLES TO RUNTIME;  
ProxySQLAdmin> SAVE MYSQL VARIABLES TO DISK;
```

The monitoring account is configured on all ends, and the next step is to tell ProxySQL about the nodes themselves.

Step 5 — Adding MySQL Nodes to the ProxySQL Server Pool

To make ProxySQL aware of our three MySQL nodes, we need to tell ProxySQL how to distribute them across its *host groups*, which are designated sets of nodes. Each host group is identified by a positive number, like **1** or **2**. Host groups can route different SQL queries to different sets of hosts when using ProxySQL query routing.

In static replication configurations, host groups can be set arbitrarily. However, ProxySQL's group replication support automatically divides all nodes in a replication group into four logical states:

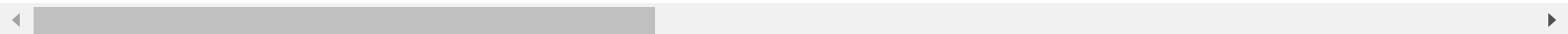
- **writers**, which are MySQL nodes that can accept queries that change data. ProxySQL makes sure to maintain all primary nodes up to the maximum defined amount in this group.
- **backup writers**, which are also MySQL nodes that can accept queries that change data. However, those nodes are not designated as writers; primary nodes exceeding the defined amount of maintained writers are kept in this group, and are promoted to writers if one of the writers fails.
- **readers** are MySQL nodes that cannot accept queries changing data and should be used as read-only nodes. ProxySQL puts only slave nodes here.
- **offline**, which is for nodes that are misbehaving due to issues like lack of connectivity or slow traffic.

Each of these four states have corresponding host groups, but the numerical group identifiers are not assigned automatically.

Putting it all together, we need to tell ProxySQL which identifiers it should use for each state. Here, we use 1 for the **offline** host group, 2 for the **writer** host group, 3 for the **reader** host group, and 4 for the **backup writer** host group.

To set these identifiers, create a new row with those variables and values in the `mysql_group_replication_hostgroups` configuration table.

```
ProxySQLAdmin> INSERT INTO mysql_group_replication_hostgroups (writer_hostgroup, backup_writer_hostg
```



These are the additional variables set in this row and what each one does:

- `active` set to 1 enables ProxySQL's monitoring of these host groups.
- `max_writers` defines how many nodes can act as writers. We used 3 here because in a multi-primary configuration, all nodes can be treated equal, so here we used 3 (the total number of nodes).
- `writer_is_also_reader` set to 1 instructs ProxySQL to treat writers as readers as well.
- `max_transactions_behind` sets the maximum number of delayed transactions before a node is classified as **offline**.

Note: Because our example uses a multi-primary topology in which all nodes can write to the database, we will balance all SQL queries across the **writer** host group. On other topologies, the division between **writer** (primary) nodes and **reader** (secondary) nodes can route read-only queries to different nodes/host groups than write queries. ProxySQL does not automatically do this, but you can set up query routing using rules.

Now that ProxySQL knows how to distribute nodes across host groups, we can add our MySQL servers to the pool. To do so, we need to `INSERT` the IP address and initial host group of each server into the `mysql_servers` table, which contains the list of servers ProxySQL can interact with.

Add each of the three MySQL servers, making sure to replace the example IP addresses in the commands below.

```
ProxySQLAdmin> INSERT INTO mysql_servers(hostgroup_id, hostname, port) VALUES (2, '203.0.113.1', 3306);
ProxySQLAdmin> INSERT INTO mysql_servers(hostgroup_id, hostname, port) VALUES (2, '203.0.113.2', 3306);
ProxySQLAdmin> INSERT INTO mysql_servers(hostgroup_id, hostname, port) VALUES (2, '203.0.113.3', 3306);
```

Here, the `2` value sets all of these nodes to be writers initially, and `3306` sets the default MySQL port.

Just like before, migrate these changes into **runtime** and save them to **disk** to put the changes into effect.

```
ProxySQLAdmin> LOAD MYSQL SERVERS TO RUNTIME;
ProxySQLAdmin> SAVE MYSQL SERVERS TO DISK;
```

ProxySQL should now distribute our nodes across the host groups as specified. Let's check that by executing a `SELECT` query against the `runtime_mysql_servers` table, which exposes the current state of the servers ProxySQL is using.

```
ProxySQLAdmin> SELECT hostgroup_id, hostname, status FROM runtime_mysql_servers;
```

Output

hostgroup_id	hostname	status
2	203.0.113.1	ONLINE
2	203.0.113.2	ONLINE
2	203.0.113.3	ONLINE
3	203.0.113.1	ONLINE
3	203.0.113.2	ONLINE
3	203.0.113.3	ONLINE

6 rows in set (0.01 sec)

In the results table, each server is listed twice: once each for host group IDs `2` and `3`, indicating that all three nodes are both writers and readers. All nodes are marked `ONLINE`, meaning they're ready to be used.

However, before we can use them, we have to configure user credentials to access the MySQL databases on each node.

Step 6 — Creating the MySQL Users

ProxySQL acts as a load balancer; end users connect to ProxySQL, and ProxySQL passes the connection to the chosen MySQL node in turn. To connect to an individual node, ProxySQL reuses the credentials it was accessed with.

To allow access to the databases located on the replication nodes, we need to create a user account with the same credentials as ProxySQL, and grant that user the necessary privileges.

Like in Step 3, the following steps must be performed only on a **single member of the group**. You can choose any one member.

Create a new user called **playgrounduser** identified with the password **playgroundpassword**.

```
(member1) mysql> CREATE USER 'playgrounduser'@'%' IDENTIFIED BY 'playgroundpassword';
```

Give it privileges to fully access the **playground** test database from the original group replication tutorial.

```
(member1) mysql> GRANT ALL PRIVILEGES on playground.* to 'playgrounduser'@'%';
```

Then apply the changes and exit the prompt.

```
(member1) mysql> FLUSH PRIVILEGES;  
(member1) mysql> EXIT;
```

You can verify that the user has been properly created by trying to the database with the newly configured credentials directly on the node.

Re-open the MySQL interface with the new user, which will prompt you for the password.

```
$ mysql -u playgrounduser -p
```

When you're logged in, execute a test query on the **playground** database.

```
(member1) mysql> SHOW TABLES FROM playground;
```

Output

```
+-----+  
| Tables_in_playground |  
+-----+  
| equipment             |
```

```
+-----+
1 row in set (0.00 sec)
```

The visible list of tables in the database showing the `equipment` table created in the original replication tutorial confirms that the user has been created correctly on the nodes.

You can disconnect from the MySQL interface now, but keep the terminal with the connection to the server open. We'll use it to run tests in the final step.

```
(member1) mysql> EXIT;
```

Now we need to create the corresponding user in the ProxySQL server.

Step 7 — Creating the ProxySQL User

The final configuration step is to allow connections to ProxySQL with the **playgrounduser** user, and pass those connections through to the nodes.

To do so, we need to set configuration variables in the `mysql_users` table, which holds user credential information. In the ProxySQL interface, add the username, password, and default host group to the configuration database (which is `2`, for the **writer** host group)

```
ProxySQLAdmin> INSERT INTO mysql_users(username, password, default_hostgroup) VALUES ('playgrounduser', 'playgroundpassword', 2);
```

Migrate the configuration into **runtime** and save to **disk** to put the new configuration into effect.

```
ProxySQLAdmin> LOAD MYSQL USERS TO RUNTIME;
ProxySQLAdmin> SAVE MYSQL USERS TO DISK;
```

To verify that we can connect to the database nodes using these credentials, open another terminal window and SSH to the ProxySQL server. We'll still need the administration prompt later, so don't close it just yet.

```
$ ssh sammy@your_proxysql_server_ip
```

ProxySQL listens on port `6033` for incoming client connections, so try connecting to the real database (not the administration interface) using **playgrounduser** and port `6033`. You'll be prompted for the password, which was `playgroundpassword` in our example.

```
$ mysql -u playgrounduser -p -h 127.0.0.1 -P 6033 --prompt='ProxySQLClient> '
```

Here, we set the prompt to `ProxySQLClient>` so we can distinguish it from the administrative interface prompt. We'll use both in when test the final configuration.

The prompt should open, meaning that the credentials have been accepted by ProxySQL itself.

ProxySQL client prompt

```
$ Welcome to the MySQL monitor.  Commands end with ; or \g.
$ Your MySQL connection id is 31
$ Server version: 5.5.30 (ProxySQL)
$
$ Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.
$
$ Oracle is a registered trademark of Oracle Corporation and/or its
$ affiliates. Other names may be trademarks of their respective
$ owners.
$
$ Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
$
$ ProxySQLClient>
```

Let's execute a simple statement to verify if ProxySQL will connect to one of the nodes. This command queries the database for the hostname of the server it's running on and returns the server hostname as the only output.

```
ProxySQLClient> SELECT @@hostname;
```

According to our configuration, this query should be directed by ProxySQL to one of our three nodes assigned to the **writer** host group. The output should look like the following, where **member1** is the hostname of one of the MySQL nodes.

Output

```
+-----+
| @@hostname |
+-----+
| member1    |
+-----+
1 row in set (0.00 sec)
```

This completes the configuration allowing ProxySQL to load balance connections among the three MySQL nodes.

In the final step, we'll verify that ProxySQL can execute read and write statements on the database and that it handles queries even when some nodes go down.

Step 8 — Verifying the ProxySQL Configuration

We know that connectivity between ProxySQL and the MySQL nodes is working, so the final tests are to ensure that the database permissions allow both read and write statements from ProxySQL, and to make

sure that these statements will still be executed when some of the nodes in the group fail.

Execute a `SELECT` statement in the ProxySQL client prompt to verify that we can read the data from the `playground` database.

```
ProxySQLClient> SELECT * FROM playground.equipment;
```

The output should be similar to the following, containing the three items created in the group replication tutorial. This means that we successfully read data from the MySQL database via ProxySQL.

Output

```
+-----+-----+-----+-----+
| id | type  | quant | color |
+-----+-----+-----+-----+
|  3 | slide |    2  | blue  |
| 10 | swing |   10  | yellow|
| 17 | seesaw|    3  | green |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Next, try writing by inserting some new data into the table representing 5 red drills.

```
ProxySQLClient> INSERT INTO playground.equipment (type, quant, color) VALUES ("drill", 5, "red");
```

Then re-execute the previous `SELECT` command to verify that the data has been inserted.

```
ProxySQLClient> SELECT * FROM playground.equipment;
```

The new drill line in the output means we successfully wrote data to the MySQL database via ProxySQL.

Output

```
+-----+-----+-----+-----+
| id | type  | quant | color |
+-----+-----+-----+-----+
|  3 | slide |    2  | blue  |
| 10 | swing |   10  | yellow|
| 17 | seesaw|    3  | green |
| 24 | drill |    5  | red   |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

We know ProxySQL can fully use the database now, but what happens if a server fails?

From the command line of one of the MySQL servers, stop the MySQL process to simulate a failure.

```
$ systemctl stop mysql
```

After the database stops, try querying data from the `equipment` table again from the ProxySQL client prompt.

```
ProxySQLClient> SELECT * FROM playground.equipment;
```

The output should not change; you should still see the equipment list as before. This means that ProxySQL has noticed that one of the nodes failed and switched to a different one to execute the statement.

We can check that by querying the `runtime_mysql_servers` table from the ProxySQL administration prompt, like in Step 5.

```
ProxySQLAdmin> SELECT hostgroup_id, hostname, status FROM runtime_mysql_servers;
```

The output will look like this:

Output

hostgroup_id	hostname	status
1	203.0.113.1	SHUNNED
2	203.0.113.2	ONLINE
2	203.0.113.3	ONLINE
3	203.0.113.2	ONLINE
3	203.0.113.3	ONLINE

6 rows in set (0.01 sec)

The node we stopped is now **shunned**, which means it's temporarily deemed inaccessible, so all traffic will be distributed across the two remaining online nodes.

ProxySQL will constantly monitor the state of this node, and either bring it back to **online** if it behaves normally, or mark it **offline** if it surpasses the timeout threshold we set in Step 4.

Let's test this monitoring. Switch back to the MySQL server and bring the node back up.

```
$ systemctl start mysql
```

Wait a moment, then query the `runtime_mysql_servers` table from the ProxySQL administration prompt again.

```
ProxySQLAdmin> SELECT hostgroup_id, hostname, status FROM runtime_mysql_servers;
```

ProxySQL will quickly notice the node is available again and mark it as online:

Output

hostgroup_id	hostname	status
2	203.0.113.1	ONLINE
2	203.0.113.2	ONLINE
2	203.0.113.3	ONLINE
3	203.0.113.1	ONLINE
3	203.0.113.2	ONLINE
3	203.0.113.3	ONLINE

6 rows in set (0.01 sec)

You can repeat this test with another node (or two of them) to see that if at least one node will be up, you will be able to freely use your database both for read-only and read-write access.

Conclusion

In this tutorial, you configured ProxySQL to load balance SQL queries across multiple write-enabled MySQL nodes in a multi-primary group replication topology. This kind of configuration can increase performance for heavy database use by distributing the load across multiple servers. It can also provide failover capability in case one of the database servers goes offline.

However, we only covered one node topology as an example here. ProxySQL provides robust query caching, routing, and performance analysis for many other MySQL topologies as well. You can read more about ProxySQL's features and how to solve different database management problems with them on the [official ProxySQL blog](#) and [ProxySQL wiki](#).

By: Mateusz Papiernik

Upvote (13) Subscribe Share

 Editor:
Hazel Virdó



We just made it easier for you to deploy faster.

[TRY FREE](#)

Related Tutorials

[How To Ensure Code Quality with SonarQube on Ubuntu 18.04](#)

[How To Sync and Share Your Files with Seafile on Ubuntu 18.04](#)

[How To Troubleshoot Issues in MySQL](#)



[How To Set Up a Remote Database to Optimize Site Performance with MySQL on Ubuntu 18.04](#)

[How To Set Up Laravel, Nginx, and MySQL with Docker Compose](#)

2 Comments

Leave a comment...

[Log In to Comment](#)

-  [Tthorpe](#) *July 11, 2018*
-  0 After creating a monitor user on the mysql servers with non defaults:

```
CREATE USER 'monitor'@'%' IDENTIFIED BY 'monitorpassword';
```

the guide never points out to change it from default on the proxysql.

I suggest including:

```
UPDATE global_variables SET variable_value="monitorpassword" WHERE variable_name="mysql-moni
```



 [jalpenshah](#) *October 18, 2018*



0

Hi,

I am facing error while executing this step:

```
mysql -u root -p < additiontosys.sql
```

Please note that I am using MariaDB 10.3.10 version.

Error:

No database 'sys' found.

I tried creating 'sys' database and execute the command again. Now I am getting below error:

ERROR 1064 (42000) at line 5: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'RETURNS INT

DETERMINISTIC

RETURN IF(a = 0, b, a)' at line 2

Request your help please

Thank you in advance!



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



Copyright © 2019 DigitalOcean™ Inc.

[Community](#) [Tutorials](#) [Questions](#) [Projects](#) [Tags](#) [Newsletter](#) [RSS](#) 

[Distros & One-Click Apps](#) [Terms, Privacy, & Copyright](#) [Security](#) [Report a Bug](#) [Write for DOnations](#) [Shop](#)