

# How to set up a Docker with Postgres using DigitalOcean

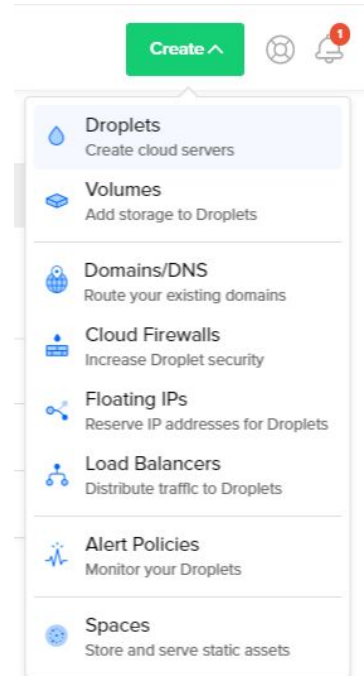
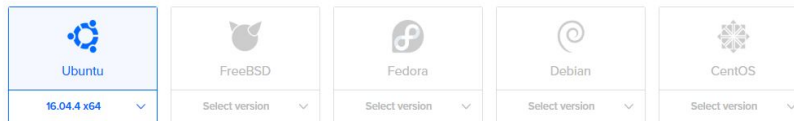
## Step 1: Create your droplet

Create an account on DigitalOcean and once you are able to log in, on the **top-right** side of the screen, hover over the green **"Create"** button and select **"Droplets - Create cloud servers"** (shown on the right). You will come to a new screen asking you what type of Droplet you would like to create. Click on the **'One-click apps' tab** (shown below).

### Create Droplets

Choose an image ?

Distributions Container distributions **One-click apps** Snapshots Backups



Once you're on the 'One-click apps' page, to get started (going from top to bottom):

1. Select on the **'Docker'** for the application.
2. Select your droplet (server) specifications.
  - We used the cheapest specs for this (\$5/Mo - 1GB).
3. Which Datacenter to your droplet hosted on.
  - We chose Toronto (go Canada!).
4. What name to give your server.

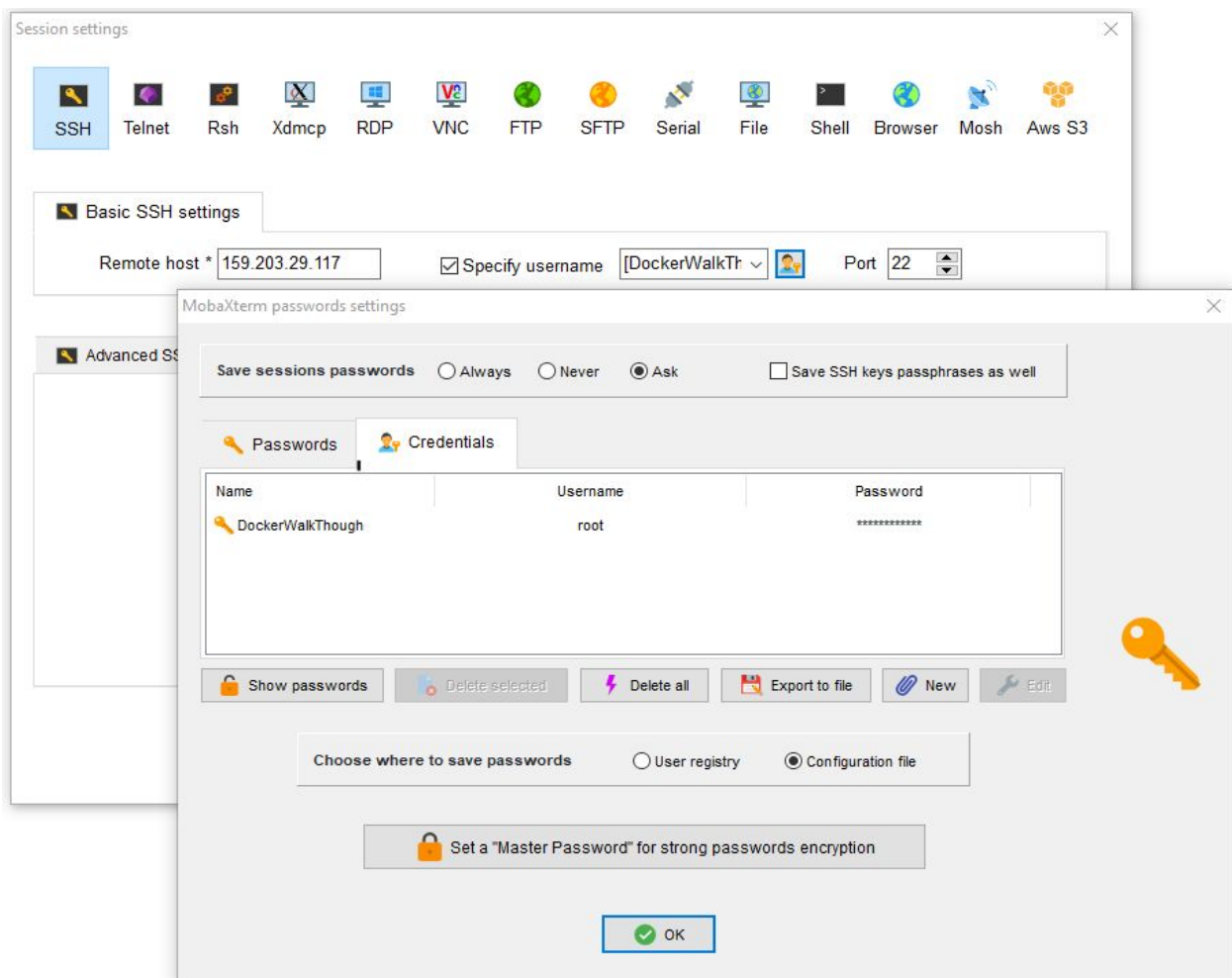
*There are optional other options you can choose from on this page, but we use only what was stated here.* Click 'create' at the bottom once you're ready. This will redirect you to the your docker profile home page and you can see the Droplet being created. **DigitalOcean will email you with details in regards to your newly created Droplet.**

We recommend downloading and installing **MobaXterm** for an alternative method of accessing your server as it offers a visual file structure of your server and allows you to have multiple files open to edit/compare/etc.. things with. Select the **'Installer'** edition on the linked page and install it:

<https://mobaxterm.mobatek.net/download-home-edition.html>.

Once you've installed MobaXterm, click on '**Session**' on the top-left, and make sure you're on the '**SSH**' tab. Fill in the following information from the email from DigitalOcean in regards to your server:

1. Your **Server's IP address** in the '**Remote Host\***' field.
2. Check the '**Specify username**' checkbox. Click on the person icon next to that field.
3. Click on the '**New**' button and fill out the following boxes:
  - A name to identify your server.
  - **The username** to log into your server with.
  - **The password** for the username.
4. Click the '**OK**' button **twice** to go back to the SSH screen.
5. Select your server (with the name you identified it with) from the dropdown menu. Make sure you are on **Port 22**.
6. Click '**OK**' at the bottom and you should be connected to your server.



Congratulations, you are now connected to your Droplet!

## Step 2: Getting Docker set up

Upon logging into your server, you are required to change your password. Change it to something else, or add a character or two to your provided password from DigitalOcean if you are doing this solely for this tutorial. Keep in mind you may want to update your MobaXterm profile with the new password if you used it.

```
Changing password for root.  
(current) UNIX password:  
Enter new UNIX password:  
Retype new UNIX password:  
root@DockerWalkThrough:~#
```

[This section was kindly provided to us from Kaylan and Steven during their presentation.](#)

The goal is to set up a Docker swarm and ensure that each container can communicate with each other.

- 1) Install Docker Machine by running the following commands:

```
wget https://github.com/docker/machine/releases/download/v0.15.0/docker-machine-$(uname -s)-$(uname -m)
```

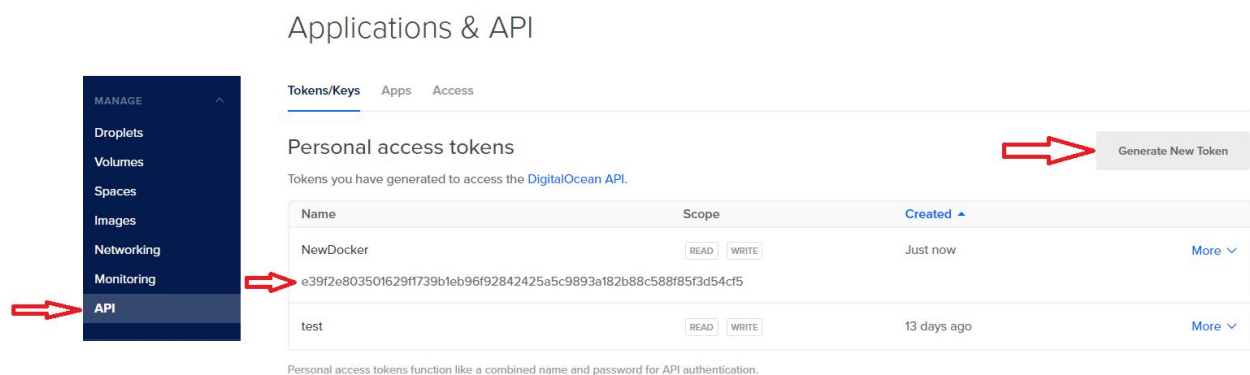
```
mv docker-machine-Linux-x86_64 docker-machine
```

```
chmod +x docker-machine
```

```
sudo mv docker-machine /usr/local/bin
```

- 2) Go back to your DigitalOcean home page and click on '**API**' tab. Click on '**Generate New Token**' on the top right. Give the token a name and it will generate a token that's needed for the next part. Each part is highlighted below with the red arrow.

Applications & API



The screenshot shows the DigitalOcean 'Applications & API' page. On the left sidebar, the 'API' tab is highlighted with a red arrow. In the main content area, the 'Generate New Token' button is highlighted with a red arrow. Below this, a table lists personal access tokens. The first token is named 'NewDocker' and has a scope of 'READ WRITE'. The second token is named 'test' and has a scope of 'READ WRITE'. Both tokens have a 'More' link next to them.

| Name      | Scope      | Created     |
|-----------|------------|-------------|
| NewDocker | READ WRITE | Just now    |
| test      | READ WRITE | 13 days ago |

- 3) Run the following command in and replace **<YOURTOKENAPIKEY>** with the one that was generated in step 2 and **<machinename>** with what you want to call the node. **SAVE THE TOKEN KEY SOMEWHERE AS THE KEY NO LONGER SHOWS IF YOU LEAVE THE PAGE.** You create a new node each time you run this command. This takes a few minutes each time it runs.

```
docker-machine create --driver digitalocean --digitalocean-access-token <YOURTOKENAPIKEY> <machinename>
```

- 4) The first two commands are to help us find our IP addresses. For this instance, we use the Docker IP address. Run the last command to initialize your Docker swarm. Replace **<ip\_address\_of\_host>** with your Docker IP address or the node you want as the swarm leader. Copy the command output that it gives to add a worker to the swarm (*docker swarm join --token ...:2377*).

```
ifconfig
Docker-machine ls
docker swarm init --advertise-addr <ip_address_of_host>
```

```
root@DockerWalkThrough:~# docker swarm init --advertise-addr 159.203.29.117
Swarm initialized: current node (gunnmhyab2x0quwmfqmidjm6v) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-5lzgcrfc6ldjodfhcwmm2kxot9mxrzs99n4ez0gvard2ukci-esgn4icsmuayt9d9g8rj3yqsr 159.203.29.117:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

root@DockerWalkThrough:~#
```

- 5) We can test our connection with our new node by running the following command. To cancel the ping, using CTRL+C to cancel.

- Ping <node\_ip>

Here are some commands that you can use to play around, but will use in a later point:

- Docker-machine ls
- Docker-machine ssh <machine\_name> (from step 3)

## Step 3: Setting up Postgres

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-18-04>

1. With our Docker swarm up and running now, we want to get Postgres going next. Before we can install Postgres, we want to make sure our server is fully up-to-date. Run the following commands and accept any prompts but keep any local version of things install if prompted:

```
Sudo apt update  
Sudo apt upgrade
```

2. After updating the server, we can go ahead and install Postgres. Run the following command:

```
sudo apt install postgresql postgresql-contrib
```

3. Once Postgres is installed, it's important to know that Postgres has its own user list and commands---meaning, if you want to do anything involving the databases, you need to be logged into a Postgres user. The following command logs you in as the 'root' of Postgres.

```
sudo -i -u Postgres
```

```
root@DockerWalkThrough:~# sudo -i -u postgres  
postgres@DockerWalkThrough:~$ exit  
logout  
root@DockerWalkThrough:~#
```

Commands that we can use:

Sudo -i -u postgres - Logs into the postgres user account

Psql - Loads into SQL mode

4. So, we are now logged in as the 'root' for Postgres. While there are a lot of useful commands that can be used, we are simply going to create and use the said database. Type the following commands (filling in <name> to whatever you want):

```
Createdb <name>  
Psql -d <name>
```

- After you type in the last command, you are now in the 'SQL' portion of things. There are similar commands to SQL but a lot of them different, which each requiring a "\ " to start it. Run the following commands:

```
CREATE TABLE cow(name VARCHAR(20), AGE INT);
INSERT INTO cow(name, AGE) VALUES('king', 42);
```

```
postgres@DockerWalkThrough:~$ clear
postgres@DockerWalkThrough:~$ psql -d test
psql (10.5 (Ubuntu 10.5-0ubuntu0.18.04))
Type "help" for help.

test=# \l
               List of databases
  Name      | Owner   | Encoding | Collate | Ctype   | Access privileges
-----+-----+-----+-----+-----+-----
 postgres   | postgres | UTF8      | C.UTF-8 | C.UTF-8 | 
 template0  | postgres | UTF8      | C.UTF-8 | C.UTF-8 | =c/postgres +
            |          |           |         |         | postgres=CTc/postgres
 template1  | postgres | UTF8      | C.UTF-8 | C.UTF-8 | =c/postgres +
            |          |           |         |         | postgres=CTc/postgres
 test       | postgres | UTF8      | C.UTF-8 | C.UTF-8 | 
(4 rows)

test=# \conninfo
You are connected to database "test" as user "postgres" via socket in "/var/run/postgresql" at port "5432".
test=# \dt
Did not find any relations.
test=# CREATE TABLE cow(name VARCHAR(20), AGE INT);
CREATE TABLE
test=# INSERT INTO COW(name, AGE) VALUES('king', 42);
INSERT 0 1
test=# \dt
               List of relations
 Schema | Name | Type | Owner
-----+-----+-----+-----
 public | cow  | table | postgres
(1 row)
```

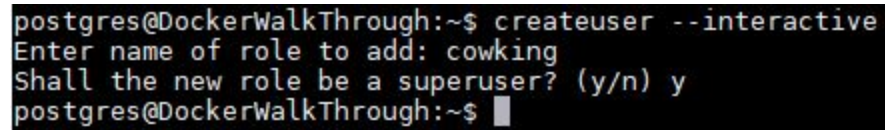
- We now set up a Database and created a table that we can use. There are a lot more commands that we can use for this part, but we can now quit and go back to our server, since we need to set things up for our server and node(s).

```
\q
Exit
```

```
test=# \q
postgres@DockerWalkThrough:~$ exit
logout
root@DockerWalkThrough:~#
```

7. Next, we need to create new 'super user' for later. Run the following commands and create a user and make it a superuser:

```
sudo -i -u postgres  
createuser --interactive
```

A terminal window with a black background and white text. The prompt is 'postgres@DockerWalkThrough:~\$'. The command 'createuser --interactive' has been entered. The output shows 'Enter name of role to add: cowking', followed by 'Shall the new role be a superuser? (y/n) y', and finally the prompt 'postgres@DockerWalkThrough:~\$' with a cursor.

```
postgres@DockerWalkThrough:~$ createuser --interactive  
Enter name of role to add: cowking  
Shall the new role be a superuser? (y/n) y  
postgres@DockerWalkThrough:~$
```

8. Lastly, we need to set up a password for the new user. Run the following commands while modifying the `<user>` you made and the `<password>` you want:

```
psql  
ALTER USER <user> PASSWORD '<password>';
```

## Step 4: Settings

1. Postgres requires port 5432 to be open in order to be connected to remotely. We can check which ports are available on our server by typing the following command:

```
Sudo ufw status
```

2. So we see that port 5432 isn't on our list of accepted ports on our server. Let's add them by typing the following commands:

```
Sudo ufw allow 5432/tcp
```

```
Sudo ufw allow 5432/udp
```

3. Now we need to edit two files that were added when we installed Postgres. If you're using MobaXterm, you can navigate there using file system located on the left side of the application. Edit the files, save them and upload them to the server. For all others, you can run the following commands:

```
Cd /etc/postgresql/10/main/
```

```
Nano postgresql.conf
```

Under the CONNECTIONS AND AUTHENTICATION section, edit the following line:

- `#listen_addresses = 'localhost'`

TO

- `listen_addresses = '*'`
  - **BE SURE TO REMOVE THE '#'**

And do the following commands: CTRL+X, Y, Enter. This saves the changes and exits.

Onto the next file.

```
Nano pg_hba.conf
```

At the bottom of this file, add the following line (lining up the text-to-column):

- Host all all all md5

```
# Allow replication connections from localhost, by a user with the
# replication privilege.
local    replication    all                                     peer
host     replication    all          127.0.0.1/32             md5
host     replication    all          ::1/128                 md5
host     all            all          all                    md5
```

And do the following commands: CTRL+X, Y, Enter. This saves the changes and exits.



4. After editing the two files, we need to simply restart the services. Run the following command:

```
invoke-rc.d postgresql restart
```

5. With the restart of Postgres, everything should be running now on our server. We now need to switch over to our node to set it up as well. Run the following commands found earlier in the tutorial to connect to our node:

```
Docker-machine ls
```

```
Docker-machine ssh <machine_name> (from step 3)
```

```
root@DockerWalkThrough:/# docker-machine ls
NAME      ACTIVE   DRIVER        STATE     URL                         SWARM   DOCKER   ERRORS
DockerTest -        digitalocean   Running   tcp://142.93.190.137:2376   -       v18.06.1-ce
root@DockerWalkThrough:/# docker-machine ssh DockerTest
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-138-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

23 packages can be updated.
8 updates are security updates.

New release '18.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

root@DockerTest:~#
```

6. Once we're connected, we can see that there are updates available. Running the update command is optional, but for this case we did run the update and you have to accept the prompts and it did take about 5-10 minutes to update. **IT IS RECOMMENDED YOU DON'T RUN THE UPDATE FOR THE PURPOSE OF THIS TUTORIAL.**
7. The node will need to have Postgres installed in order to connect to the database on our server. Run the following command:

```
apt install postgresql
```

8. Once it's installed, we now want to connect to our remote database (which is located currently on our server). Run the following command, with replacing `<server_ip>` with your server IP, `<user>` with the user you created and `<database>` with the one you created:

```
psql -h <server_ip> -U <user> -d <database>
```

It will then prompt you for a password. Type in the one you set up for your `<user>`.

9. Congratulations! You should now be connected to your Postgres SQL server from your node. This is a very simple Tutorial using one server and one node. You could in theory do this the opposite way (with the Postgres Database being on the node while you connect to it from the server), or set up another node, host the database on that one and connect to it from the first node.

```
Last login: Tue Nov  6 03:15:26 2018 from 159.203.29.117
root@DockerTest:~# psql -h 159.203.29.117 -U cowking -d test
Password for user cowking:
psql (10.5 (Ubuntu 10.5-0ubuntu0.18.04))
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

test=# SELECT * FROM cow;
 name | age 
-----+-----
 king |  42 
(1 row)

test=# INSERT INTO cow(name, age) VALUES('nodeking', '49');
INSERT 0 1
test=# SELECT * FROM cow;
 name | age 
-----+-----
 king |  42 
nodeking |  49 
(2 rows)

test=# \conninfo
You are connected to database "test" as user "cowking" on host "159.203.29.117" at port "5432".
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
test=# █
```

```
root@DockerWalkThrough:/# sudo -i -u postgres
postgres@DockerWalkThrough:~$ psql -d test
psql (10.5 (Ubuntu 10.5-0ubuntu0.18.04))
Type "help" for help.

test=# SELECT * FROM cow;
 name | age 
-----+-----
 king |  42 
nodeking |  49 
(2 rows)

test=# \conninfo
You are connected to database "test" as user "postgres" via socket in "/var/run/postgresql" at port "5432".
test=# █
```