DigitalOcean Droplet Install Guide

v1.0 — 2017-10-03

Create New (Add) Users

Change root password after login

```
$ ssh root@<ip address droplet>
$ passwd
```

Update system

```
$ apt-get update
$ apt-get upgrade
$ apt-get autoremove && apt-get autoclean
```

To also upgrade the Ubuntu distribution version run:

```
$ apt-get dist-upgrade
```

List all users you can use

```
$ cut -d: -f1 /etc/passwd
```

Remove/delete a user (plus home directory)

```
$ userdel -f <username>
$ rm -r /home/<username>
```

Modify of a user

```
$ usermod -l <new_username> <username>
```

Change password for a user

```
$ passwd <username>
```

Change details for a user (for example real name):

```
$ chfn username
```

Add new user

```
$ adduser <username>
```

[OPTIONAL] Add root privileges

Now, we have a new user account with regular account privileges. However, we may sometimes need to do administrative tasks.

To avoid having to log out of our normal user and log back in as the root account, we can set up what is known as "superuser" or root privileges for our normal account. This will allow our normal user to run commands with administrative privileges by putting the word `sudo` before each command.

To add these privileges to our new user, we need to add the new user to the "sudo" group. By default, on Ubuntu 16.04, users who belong to the "sudo" group are allowed to use the `sudo` command.

As `root`, run this command to add your new user to the sudo group (substitute the highlighted word with your new user):

```
$ usermod -aG sudo sammy
```

Now the (non-root) user can run commands with superuser privileges! For more information about how this works, check out this sudoers tutorial.

Login with new user

```
$ ssh user@hostname/ip-address
```

Note: If the server is set up with public key authentication for users (e.g. this is the case on DigitalOcean if an SSH key was selected during Droplet creation), then follow first **Copy the Public (SSH) Key** section of the **Add Public Key Authentication (SSH)** document to be able to login with newly added (non-root) users.

Add Public Key Authentication (SSH)

The next step in securing the server is to set up public key authentication for user(s). Setting this up will increase the security of the server by requiring a private SSH key to log in.

Generate a Key Pair

If you do not already have an SSH key pair, which consists of a public and private key, you need to generate one. If you already have a key that you want to use, skip to the **Copy the Public (SSH) Key** step.

In Terminal.app:

```
$ ssh-keygen -t rsa
```

Hit return to accept file name and path (or enter a new name).

Next, you will be prompted for a passphrase to secure the key with. You may either enter a passphrase or leave the passphrase blank.

This generates a private key, id_rsa, and a public key, id_rsa.pub, in the .ssh directory of the localuser's home directory. Remember that the private key should not be shared with anyone who should not have access to your servers!

Copy the Public (SSH) Key

After generating an SSH key pair (on your Mac), you will want to copy your public key to your server. There are two easy ways to do this.

Note: The `ssh-copy-id` method will not work on DigitalOcean if an SSH key was selected during Droplet creation. This is because DigitalOcean disables password authentication if an SSH key is present, and the `ssh-copy-id` relies on password authentication to copy the key.

If using DigitalOcean and a SSH key was selected during Droplet creation, use option 2 instead.

Option 1 — Use ssh-copy-id

Run the ssh-copy-id script by specifying the user and IP address of the server that you want to install the key on, like this:

```
$ ssh-copy-id user@hostname/ip-address
```

After providing the password at the prompt, the public key will be added to the remote user's `.ssh/authorized_keys` file. The corresponding private key can now be used to log into the server.

Option 2 — Manually Install the Key

Assuming you generated an SSH key pair using the previous step, use the following command at the terminal of the **local machine** to print your public key (`id_rsa.pub`):

```
$ cat ~/.ssh/id_rsa.pub
```

This should print your public SSH key, select it, and copy it to your clipboard.

To enable the use of SSH key to authenticate as the new remote user, the public key must be added to a special file in the user's home directory.

On the server, as the `root` user (login via ssh), enter the following command to temporarily switch to the new user (substitute `user` name for your own):

```
$ su - user
```

Now you will be in your new user's home directory.

Create a new directory called `.ssh` and restrict its permissions with the following commands:

```
$ mkdir ~/.ssh
$ chmod 700 ~/.ssh
```

Now open a file in `.ssh` called `authorized_keys` with a text editor (nano) to edit the file:

```
$ nano ~/.ssh/authorized_keys
```

Now insert your public key (which copied earlier into the clipboard) by pasting it into the editor.

Hit ctrl-x to exit the file, then y to save the changes that you made, then ENTER to confirm the file name.

Now restrict the permissions of the `authorized_keys` file:

```
$ chmod 600 ~/.ssh/authorized_keys
```

Type this command once to return to the root user:

```
$ exit
```

Now your public key is installed, and you can use SSH keys to log in as your user.

Repeat for other users.

Disable Password Authentication (Recommended)

Note: Only disable password authentication if you installed a public key to your user as recommended in the previous section, step four. Otherwise, you will lock yourself out of your server!

To disable password authentication on your server, follow these steps.

As **root** or your new sudo user, open the SSH daemon configuration:

```
$ sudo nano /etc/ssh/sshd_config
```

Find the line that specifies `PasswordAuthentication`, uncomment it by deleting the preceding `#`, then change its value to “no”. It should look like this after you have made the change:

```
PasswordAuthentication no
```

Here are two other settings that are important for key-only authentication and are set by default. If you haven't modified this file before, you do not need to change these settings:

```
PubkeyAuthentication yes
ChallengeResponseAuthentication no
```

When finished making the changes, save and close the file using the same method as before (e.g. ctrl-x, then Y, then ENTER).

Reload the SSH daemon:

```
$ systemctl reload sshd
```

Install Nginx, PHP-FPM and ImageMagick/GD

Install (or update to) latest nginx ‘mainline’ version

The Ubuntu 16.04 LTS on Digital Ocean droplets comes with nginx x.x.x stable. The latest versions, as of the time of this writing, are 1.12.1 stable and 1.13.3 mainline.

So which one should we upgrade to?

According to nginx:

> We recommend that in general you deploy the NGINX mainline branch at all times.

So let’s install (or upgrade to) 1.13.3 (note that ‘development’ is the legacy name for ‘mainline’):

```
$ add-apt-repository ppa:nginx/development
$ apt-get update
$ apt-get install nginx
```

Note that the installation process may have conflicts with your configuration files. If that’s the case, you will be prompted for an action. I would recommend that you hit `D` to see a diff off the changes and to back that up in a file for later use. Then proceed with `N` (the default) to keep your version. That way, if you find that you do need to change something later, you’ll have a handy diff to refer to.

Finally, run:

```
$ nginx -v
```

And you should see:

```
nginx version: nginx/1.13.3
```

Fix “Failed to read PID from file…” issue

Check nginx's status:

```
$ service nginx status
```

If the output looks like this *and* it contains the line `nginx.service: Failed to read PID from file…`:

```
● nginx.service - A high performance web server and a reverse
proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled;
vendor preset: enab
  Drop-In: /etc/systemd/system/nginx.service.d
           └─override.conf
   Active: active (running) since Thu 2017-09-28 18:34:51 UTC;
17h ago
     Docs: man:nginx(8)
  Process: 5375 ExecStop=/sbin/start-stop-daemon --quiet --stop
--retry QUIT/5 --p
  Process: 12301 ExecReload=/usr/sbin/nginx -g daemon on;
master_process on; -s re
  Process: 5388 ExecStartPost=/bin/sleep 0.1 (code=exited,
status=0/SUCCESS)
  Process: 5383 ExecStart=/usr/sbin/nginx -g daemon on;
master_process on; (code=e
  Process: 5380 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on;
master_process on
 Main PID: 5387 (nginx)
```

```
     Tasks: 2
    Memory: 8.9M
       CPU: 1.287s
    CGroup: /system.slice/nginx.service
           ├─ 5387 nginx: master process /usr/sbin/nginx -g
daemon on; master_proc
           └─12305 nginx: worker process

Sep 29 10:56:47 droplet-name systemd[1]: Starting A high
performance web server…
Sep 29 10:56:47 droplet-name systemd[1]: nginx.service: Failed
to read PID from file /run/nginx.pid: Invalid argument
Sep 29 10:56:47 droplet-name systemd[1]: Started A high
performance web server…
```

… then **fix** this issue with the following workaround:

```
$ mkdir /etc/systemd/system/nginx.service.d
$ printf "[Service]\nExecStartPost=/bin/sleep 0.1\n" >
/etc/systemd/system/nginx.service.d/override.conf
$ systemctl daemon-reload
```

Test and restart nginx:

```
$ nginx -tt && service nginx restart
```

Check status again:

```
$ service nginx status
```

The `nginx.service: Failed to read PID from file…` line should now be gone and you can continue…

Uninstall PHP-FPM and dependencies

Maybe you want to remove older versions of PHP-FMP first...

Check what version is installed, if any:

```
$ dpkg --get-selections | grep php
```

This will output something like this:

```
php5-common                     install
php5-fpm                        install
php5-gd                         install
php5-imagick                    install
php5-json                       install
```

Meaning PHP5-FPM and PHP5-Imagick are installled among some other PHP dependencies.

Now uninstall PHP-FPM and all its dependencies:

```
$ apt-get -y purge php.*
```

Check if PHP-FPM and its dependencies are completely uninstalled:

```
$ dpkg --get-selections | grep php
```

Adding a PPA for PHP 7.1 Packages

A **Personal Package Archive**, or **PPA**, is an Apt repository hosted on Launchpad. PPAs allow third-party developers to build and distribute packages for Ubuntu outside of the official channels. They're often useful sources of beta software, modified builds, and backports to older releases of the operating system.

Ondřej Surý maintains the PHP packages for Debian, and offers a PPA for PHP

7.1 on Ubuntu. Before doing anything else, log in to your system, and add Ondřej's PPA to the system's Apt sources:

```
$ add-apt-repository ppa:ondrej/php
```

Once the PPA is installed, update the local package cache to include its contents:

```
$ apt-get update
```

Install PHP-FPM and dependencies

Install the new PHP-FPM package and its dependencies:

```
$ apt-get install php7.1-fpm
```

Open php.ini:

```
$ nano /etc/php/7.1/fpm/php.ini
```

And change the `;cgi.fix_pathinfo=1` line to (enable by removing the `;` at the beginning of the line):

```
cgi.fix_pathinfo = 0
```

Then (re)start:

```
$ service php7.1-fpm restart
```

and test:

```
$ dpkg --get-selections | grep php
$ php -v
```

Install additional PHP dependencies

At least these additional dependencies are necessary for Kirby CMS to run:

```
$ apt-get install php7.1-mbstring
$ apt-get install php7.1-xml
```

Install Image/Thumb Comversion Extension

Option 1: ImageMagick + PHP Extension (recommended)

Check the version of imagemagick available:

```
$ apt-cache policy imagemagick
```

If you want to uninstall an old(er) version, run (changing the version number to match that installed):

```
dpkg -r imagemagick-6.7.7.10
```

Install ImageMagick:

```
apt-get install imagemagick
```

Sometimes it is necessary to reboot your droplet before to continue:

```
$ reboot
```

Install ImageMagick PHP Extension:

```
$ apt-get install php-imagick
```

(Re)start and check:

```
$ service php7.1-fpm restart
$ nginx -tt && service nginx restart
$ dpkg --get-selections | grep php
```

Options 2: GD PHP Extension

If you don't want to use the better option of ImageMagick, then continue with onstalling the GD dependency...

Check the version of GD available:

```
$ apt-cache policy php-gd
```

Install the GD PHP dependency:

```
$ apt-get install php-gd
```

(Re)start and check:

```
$ service php-fpm7.0 restart
$ nginx -tt && service nginx restart
$ dpkg --get-selections | grep php
```

Configure PHP-FM

PHP-FPM creates and manages a pool of php processes, also called workers that receive and server requests to execute php files from the web directory. Now fpm can run multiple separate pools each with its own uid/gid.

On Ubuntu, the directory that contains the pool configuration files is:

```
/etc/php/7.1/fpm/pool.d/
```

A default file `www.conf` already exists which can be copied to create more pool configuration files. Each file must end with `.conf` to be recognised as a pool configuration file by PHP-FPM.

Copy conf file

Create a copy of the `www.conf` file and rename it:

```
mysite.conf
```

Backup (rename) `www.conf` by adding a tilde: `www.conf~`

Edit conf file

Edit fields in `mysite.conf` file.

First the pool name `[www]`. It is on the top of the file. Rename it to `[mysite]`:

```
; pool name ('www' here)
[mysite]
```

The user and group fields:

```
; will be used.
user = mysite
group = mysite
```

And last edit the socket file path (every pool should have its own separate socket):

```
; Note: This value is mandatory.
listen = /run/php/php7.1-fpm-mysite.sock
```

Now restart PHP-FPM:

```
$ service php7.1-fpm restart
```

source

Create website folder(s)

The (default) location of the web root is `/usr/share/nginx/html`.

Create root folder and domein folder:

```
$ mkdir -p /usr/share/nginx/wwwroot/example.com/www
```

From here on follow the instructions outlined in the **Install Git and Deploy Site Files** document... off course after finishing the final steps below!

Configure nginx

Login (via SFTP) as `root` user with Transmit f.i.

Upload nginx Config Files

Backup the default `nginx.conf` and `mime.types` files by adding a tilde to the file name: `nginx.conf~` and `mime.types~`

Now upload from the `/nginx` folder both the `nginx.conf` and `mime.types` files.

Upload from the `/nginx` folder the config files to: `/etc/nginx`, frist the `hbp5` and (when using Kirby CMS) the `kirby` folders.

Now rename `kirby-example.conf` file to reflect your project: `kirby-mysite.conf`

Update the line `fastcgi_pass unix:/var/run/php7.1-fpm-example.sock;` in `kirby-mysite.conf` to link to the correct mysite socket (to be created later in the 'configure php-fm' see below!):

```
fastcgi_pass unix:/var/run/php/php7.0-fpm-mysite.sock;
```

Sites available

Delete the `default` file in the `sites-available` folder and upload (again from the `/nginx` folder) the following files:

- no-default
- mysite.com
- ssl.mysite.com

(Make sure to delete the `default` symlink in `sites-enabled`!)

Rename and edit the contents of the `mysite.com` and `ssl.mysite.com` files to follow your server/website setup — among other things, include the correct kirby conf file created in the previous step:

```
include kirby/kirby-mysite.conf;
```

Sites enabled

Login with `root` user and active the desired virtual hosts (don’t forget to activate the no-default site too!):

```
$ ln -s /etc/nginx/sites-available/mysite.com /etc/nginx/sites-
enabled/mysite.com
$ ln -s /etc/nginx/sites-available/no-default /etc/nginx/sites-
enabled/no-default
```

And for HTTPS enabled sites:

```
$ ln -s /etc/nginx/sites-available/sll.no-default
/etc/nginx/sites-enabled/ssl.no-default
```

Test and restart nginx:

```
nginx -tt && service nginx restart
```

Add .gitconfig

Login to your droplet

```
$ ssh user@hostname/ip-address
```

Create ‘.gitconfig’ file

Add a `.gitconfig` file to the user's home directory if not already present (in `/root/` or `/home/user`):

```
$ cd ~
$ touch .gitconfig
```

Edit ‘.gitconfig’ file

```
$ nano .gitconfig
```

(or edit via Transmit’s SFTP)

Add the following lines to the `.gitconfig` file:

```
[user]
  name = First Lastname
  email = example@domain.com
[push]
  default = current
[branch]
  autosetuprebase = always
[color]
```

```
  ui = true
[color "diff"]
  meta = yellow bold
  frag = magenta bold
  old = red bold
  new = green bold
  whitespace = red reverse
[color "status"]
  added = yellow
  changed = green
  untracked = cyan
[color "branch"]
  current = yellow reverse
  local = yellow
  remote = green
[alias]
  # Shortcuts
  st = status
  ci = commit
  br = branch
  co = checkout
  df = diff
  dc = diff --cached
  ls = ls-files
  add = "git add ."
  push = "git push origin head"
  pull = "git pull"
  pp = "pull && push"
  # Logs
  # lg = log --color --graph --decorate --pretty=oneline --
abbrev-commit
  lg = log --color --graph --pretty=format:'%Cred%h%Creset -
%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' -
-abbrev-commit
  # Show ignored files:
  ign = ls-files -o -i --exclude-standard
  # Check what has changed
  check = "!f() { git whatchanged $1 -1 | awk '/^:/ {print
$NF}'; }; f"
[core]
  whitespace=fix,-indent-with-non-tab,trailing-space,cr-at-eol
  # A Simple Tweak for Making 'git rebase' Safe on OS X
(http://j.mp/1gtQGuh)
  trustctime = false
  sparsecheckout = true
  # Set to false (or umask) to use permissions reported by umask
  sharedRepository = false
[filter "media"]
  clean = git-media-clean %f
  smudge = git-media-smudge %f
[difftool]
  prompt = false
```

Make sure to edit personal information (user name + email).

Or upload from the `/git` folder the `gitconfig.sample` file to user's home directory (`/root/` or `/home/user`) and rename it to `.gitconfig` .

Install Git + Deploy Site Files

Install Git

To install the latest Git package for Ubuntu, do:

```
$ add-apt-repository ppa:git-core/ppa -y
$ apt-get update
$ apt-get install git
$ apt-get autoremove && apt-get autoclean
$ git --version
```

Or to install the most current *stable* version for Ubuntu, do:

```
$ apt-get install git-core
$ apt-get autoremove && apt-get autoclean
$ git --version
```

Option 1 — Configure Git for ‘bare’ repositories

- This is for ‘bare’ repositories (what is a bare repository?)
- Do you yse git submodule? Please see **option 2** below!

Create ‘public’ folders

Create the following folders with `root` user (`-p` makes sure all the directories that don’t exists already are created, not just the last one):

```
$ mkdir -p /usr/share/nginx/wwwroot/mysite.com/www
$ mkdir -p /usr/share/nginx/wwwroot/mysite.com/stage
$ mkdir -p /usr/share/nginx/repo/mysite.com/www.git
$ mkdir -p /usr/share/nginx/repo/mysite.com/stage.git
```

Update group and user permissions

Now change the group and ownership of the `/public` and `/mysite.git` folders:

```
$ sudo chown -R mysite:mysite
/usr/share/nginx/wwwroot/mysite.com/www
$ sudo chown -R mysite:mysite
/usr/share/nginx/wwwroot/mysite.com/stage
$ sudo chown -R mysite:mysite
/usr/share/nginx/repo/mysite.com/www.git
$ sudo chown -R mysite:mysite
/usr/share/nginx/repo/mysite.com/stage.git
```

Move the `/usr/share/nginx/html/50x.html` file to the newly created `/wwwroot` directory: `/usr/share/nginx/wwwroot`, and then delete the `/html` directory.

Initialize the 'bare' Git repositories

Login (SSH) with `mysite` user and initialize the bare Git repositories:

```
$ cd /usr/share/nginx/repo/mysite.com/www.git
$ git init --bare
```

(Repeat for staging domain!)

Git hooks

Make sure to login (either SSH or SFTP) with the correct `mysite` user when uploading the files, otherwise the file group/owner will be incorrect!

Upload to the `/usr/share/nginx/repo/mysite.com/www.git/hooks` folder of the bare git repository the `post-receive.bare.sample` file, located in the

`/git/hooks` folder (make sure to enter the correct WWW_DIR and GIT_DIR paths) and after uploading rename to `post-receive`.

Set permissions of the `post-receive` file to `775`.

Repeat for staging (and other possible) (sub)domain(s).

Sparse checkout

Upload to the `/usr/share/nginx/repo/mysite.com/www.git/info` folder of the bare git repositories the `sparse-checkout.sample` file (set permissions to `664`), located in the `/git/info` folder (make sure to enter the correct path-to-files) and after uploading rename to `sparse-checkout`.

Add remote stage and production repositories

Now add the remote stage and production repositories to your local repository:

```
$ git remote add stage ssh://mysite@hostname-or-
ip/usr/share/nginx/repo/mysite.com/stage.git
$ git remote add production ssh://mysite@hostname-or-
ip/usr/share/nginx/repo/mysite.com/www.git
```

Option 2 — Configure Git for repositories with submodule(s)

- This is for repositories with submodule(s) (based on this article)
- Don't you use git submodules? Please see **flavour 1** above!

Create ‘public’ folders

Create the following folders with `root` user (`-p` makes sure all the directories that don’t exists already are created, not just the last one):

```
$ mkdir -p /usr/share/nginx/wwwroot/mysite.com/www
$ mkdir -p /usr/share/nginx/wwwroot/mysite.com/stage
```

Update group and user permissions

Change group and ownership of the `/www` and `/stage` folders:

```
$ chown -R mysite:mysite /usr/share/nginx/wwwroot/mysite.com/www
$ chown -R mysite:mysite
/usr/share/nginx/wwwroot/mysite.com/stage
```

Move the `/usr/share/nginx/html/50x.html` file to the newly created `/wwwroot` directory: `/usr/share/nginx/wwwroot`, and then delete the `/html` directory.

Initialize git repositories

Login (SSH) with `mysite` user and initialize the Git repositories:

```
$ cd /usr/share/nginx/wwwroot/mysite.com/www
$ git init
```

(Repeat for staging domain!)

Git config

Make sure to login (either SSH or SFTP) with the correct `mysite` user when uploading the file, otherwise the file group/owner will be incorrect!

Upload to the `/usr/share/nginx/wwwroot/mysite.com/www/.git` folder the `config.submodules.sample` file, located in the `/git` folder, and after uploading rename to `config` (after backing up the original `config` file by adding a tilde: `config~`).

Set permissions of the `config` file to `664`.

Repeat for staging (and other (sub)domains).

Git hooks

Make sure to login (either SSH or SFTP) with the correct `mysite` user when uploading the file, otherwise the file group/owner will be incorrect!

Upload to the `/usr/share/nginx/wwwroot/mysite.com/www/.git/hooks` folder the `post-receive.submodules.sample` file, located in the `/git/hooks` folder and after uploading rename to `post-receive`.

Set permissions of the `post-receive` file to `775`.

Repeat for staging (and other (sub)domains).

Sparse checkout

Now add to the `/usr/share/nginx/wwwroot/mysite.com/www/.git/info` folder the `sparse-checkout.sample` file (set permissions to `664`), located in the `/git/info` folder (make sure to enter the correct path-to-files) and after uploading rename to `sparse-checkout`.

Add remote stage and production repositories

Now add the remote stage and production repositories to your local repository:

```
$ git remote add stage ssh://mysite@hostname-or-
ip/usr/share/nginx/wwwroot/mysite.com/stage
$ git remote add production ssh://mysite@hostname-or-
ip/usr/share/nginx/wwwroot/mysite.com/www
```

Set Up a Basic Firewall

Ubuntu 16.04 servers can use the UFW firewall to make sure only connections to certain services are allowed. We can set up a basic firewall very easily using this application.

Different applications can register their profiles with UFW upon installation. These profiles allow UFW to manage these applications by name. OpenSSH, the service allowing us to connect to our server now, has a profile registered with UFW.

You can see this by typing:

```
$ ufw app list
```

Output:

```
Available applications:
  OpenSSH
```

To make sure that the firewall allows SSH connections to be able to log back in next time, allow these connections by typing:

```
sudo ufw allow OpenSSH
```

Afterwards, the firewall can be enabled by typing:

```
sudo ufw enable
```

Type "Y" and press ENTER to proceed. To see that SSH connections are still allowed, type:

```
sudo ufw status
```

Output:

```
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
```

If you install and configure additional services, you will need to adjust the firewall settings to allow acceptable traffic in. You can learn some common UFW operations in this guide.