DigitalOcean Droplet Deployment Manual

Thomas Castleman

Getting Started

Once you've started a new droplet instance through your DigitalOcean account online, SSH into the root user:

```
ssh root@<DROPLET IP>
```

Since we don't want to use `root` for everything, create a personal user

```
adduser tcastleman
```

Make a `.ssh` directory for storing keys in `/home/tcastleman`

```
mkdir /home/tcastleman/.ssh/
```

From `/home/tcastleman`, copy the root SSH key

```
cp /root/.ssh/authorized_keys /home/tcastleman/.ssh/authorized_keys
```

Make the `authorized_keys` file owned by your personal user

```
chown tcastleman:tcastleman /home/tcastleman/.ssh/authorized_keys
```

Change permissions of that file, so that I can read/write, everyone can read

```
chmod 644 /home/tcastleman/.ssh/authorized_keys
```

Finally, add your personal user to the sudo group

```
usermod -aG sudo tcastleman
```

Now, exit the droplet and attempt to ssh as your personal user. Ensure this works. Check sudoing to ensure it also works.

Hardening SSH

Hardening SSH can make it more difficult for unauthorized individuals to gain access to and control of the droplet.

To do so, we'll edit the `sshd_config` file -- this is the configuration info for other clients using SSH to connect to this server

```
nano /etc/ssh/sshd_config
```

In `sshd_config`, we need to change three important settings:

- Disable root login, by setting `PermitRootLogin` to `no`
- Disable password authentication by setting `PasswordAuthentication` to `no`
- Disable PAM, by setting `UsePAM` to `no`

Installing Node.js

First, su yourself, so you can run the following commands

```
sudo su
```

Update system packages:

```
apt-get update
apt-get upgrade
```

Install some essentials:

```
apt-get install build-essential
apt-get install curl
apt-get install git
apt-get install authbind
```

To install Node.js, we need the Node.js binary distributions, which can be found here.

Find the version you want (for me, 10), and install it:

```
curl -sL https://deb.nodesource.com/setup_10.x | bash -
apt-get install -y nodejs
```

Now, get MySQL

```
apt-get install mysql-server
```

And run the secure installation:

```
mysql_secure_installation
```

Respond to the prompts as follows:

- **No** password for root
- Set a root password (and keep track of this)

- **Yes**, remove anonymous users
- **Yes**, disallow root login remotely
- **Yes**, remove test database
- **Yes**, reload privilege tables

Finally, make a `/var/www` directory and `chmod` it to make it accessible read/write/exec by everyone. This will contain the source code for services running on our server.

```
mkdir /var/www
chmod 777 /var/www
```

Creating a Service

First, create a system user to run the service: (you'll probably want to name it something relevant to the service)

```
sudo adduser service-usr
```

Now, become this user:

```
sudo su service-usr
```

Create an SSH key for this user, so they can interact with the repository (no passphrase)

```
ssh-keygen
```

Now, on the Git repo, add a deploy key (under settings). You can find the public key in the `.ssh` folder of the service user:

```
cat /home/service-usr/.ssh/id_rsa.pub
```

Now, navigate to `/var/www` and clone the desired repo **using SSH**:

```
git clone git@github.com:github-user/repository-name.git
```

Now, ensure the software in the cloned repo has all required components set up as applicable. For example:

- Database built and privileges granted
- Credentials

Finally, we need to add an init script for this service, so first elevate to root (`sudo su`)

In `/etc/init.d`, we'll create a new file with whatever name we'd like the service to have:

```
nano /etc/init.d/service-name
```

And copy and paste the init script found [here](#) into this file. Only a few changes at the very top of the file need to be made before writing (highlighted in blue):

```
#! /bin/bash

### BEGIN INIT INFO
# Provides:          <SERVICE NAME>
# Required-Start:    $remote_fs $named $syslog
# Required-Stop:     $remote_fs $named $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: <SERVICE NAME> frontend/backend
### END INIT INFO

NODE_ENV="production"
PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/local/bin
DAEMON_ARGS="/var/www/<REPOSITORY>/<SERVER FILE>.js"
DESC="<DESCRIPTION OF SERVICE>"
NODEUSER=<SERVICE USER>:<SERVICE USER>
LOCAL_VAR_RUN=/var/run
NAME=node
DAEMON=/usr/bin/$NAME
LOGFILE=/var/log/<SERVICE NAME>.log
```

Once the init script has been configured to provide this service, we need to make it world-executable:

```
chmod a+x /etc/init.d/service-name
```

Then use `update-rc.d` to add the init script to the systemd database, which will make this service available.

```
update-rc.d service-name defaults
```

Now the service can be controlled using

```
sudo service service-name start/stop/restart
```

At this point, the service can be started.

Note that if the service is attempting to listen on a port below 1024, it will not have the necessary privileges to do so.

`authbind` can allow us to circumvent these privilege issues. If you want to set up just *a single* *service* running on port 80, we can use `authbind` to allow the `service-usr` to bind to 80 as follows:

```
cd /etc/authbind/byport
touch 80
chown service-usr:service-usr 80
chmod 755 80
```

If you want to set up multiple services running on different subdomains, however, see the later section regarding Apache.

Using Monit to Keep Services Online

First, install Monit:

```
sudo apt-get install monit
```

To start monitoring a service, create a file under the service name in `/etc/monit/conf.d/` with the following form:

```
check castleman-space with pidfile /var/run/castleman-space.pid
      start program = "/etc/init.d/castleman-space start"
      stop program = "/etc/init.d/castleman-space stop"
      restart program = "/etc/init.d/castleman-space restart"

if failed host castleman.space port 7777 protocol HTTP request / with timeout 10 seconds then restart
```

This example monitors the service castleman-space, which runs on port 7777. You should create a file like this for any and all services you would like to monitor.

To apply the updates, reload Monit:

```
sudo monit reload
```

Monit should now monitor the indicated services and bring them back online if they crash.

DNS Records

Once you have registered a domain name through a domain name registrar like Gandi.net, go to your domain's dashboard.

Under the DNS Records section, set up *only* the following rules:

1. An A record named * which points to your server's IP

2. An A record named @ which points to your server’s IP

This will allow your server full control of handling requests made to subdomains.

Installing & Setting Up Apache

Apache allows us to map specific subdomains on our domain to a given port, which is useful for running multiple services concurrently. For example, I could have a subdomain one.example.com which maps to a service running on port 7979, and two.example.com which maps to another service running on port 8989.

Let’s install Apache2 by running the following commands.

```
sudo apt update
sudo apt -y upgrade
sudo apt-get install apache2
```

Once this is complete, run:

```
sudo a2enmod
```

It should prompt you with something like: *Which module(s) do you want to enable (wildcards ok)?*

Pass in the below arguments:

```
proxy proxy_ajp proxy_http rewrite deflate headers proxy_balancer proxy_connect
proxy_html
```

Pointing Subdomains to Ports

To direct traffic from a subdomain to a service running on a given port, we’ll use a ProxyPass.

For each service you wish to run through a subdomain, add a `.conf` file in `/etc/apache2/sites-available/` that follows this format:

```
<VirtualHost *:80>
    ServerAdmin user@example.com
    ServerName sub.example.com
    ProxyPreserveHost On

    # setup the proxy
    <Proxy *>
        Order allow,deny
        Allow from all
    </Proxy>
    ProxyPass / http://localhost:5000/
    ProxyPassReverse / http://localhost:5000/
</VirtualHost>
```

The above ProxyPass, as an example, passes all requests to the subdomain sub.example.com to a service listening on port 5000. This `.conf` file can be named however you like.

From `/etc/apache2/sites-available/`, enable the site by running `a2ensite` on the configuration file you just created

```
sudo a2ensite example-com.conf
```

To apply these changes, restart apache:

```
sudo systemctl reload apache2
```

Note: To set up the default service Apache should pass to when receiving requests on the root domain, simply edit `/etc/apache2/sites-available/000-default.conf` *using the ProxyPass example above, but make* `ServerName` *the root domain. Then restart Apache.*