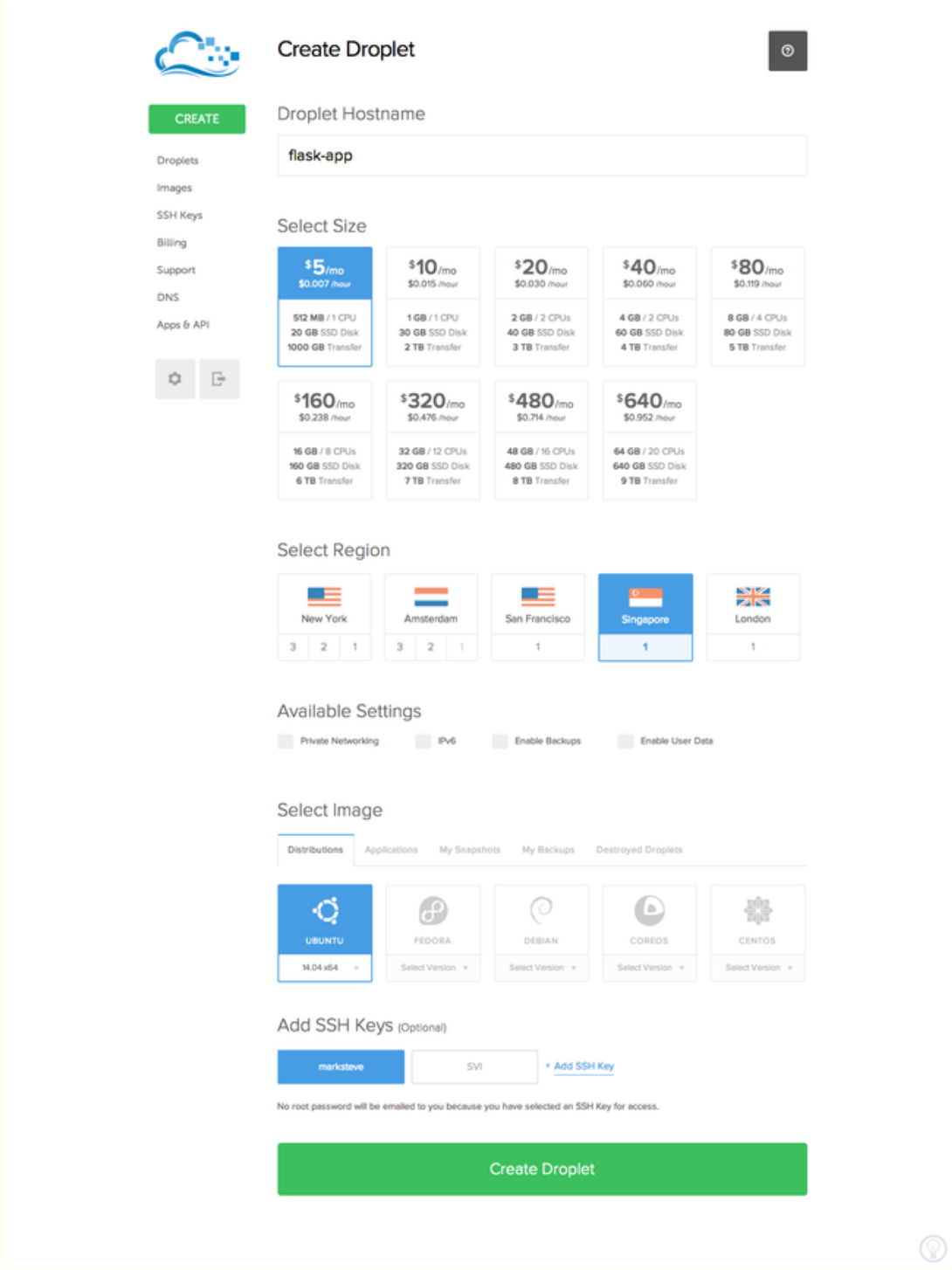
tail /thoughts

Deploying a Flask application inside a DigitalOcean droplet

Published on 2014-12-14

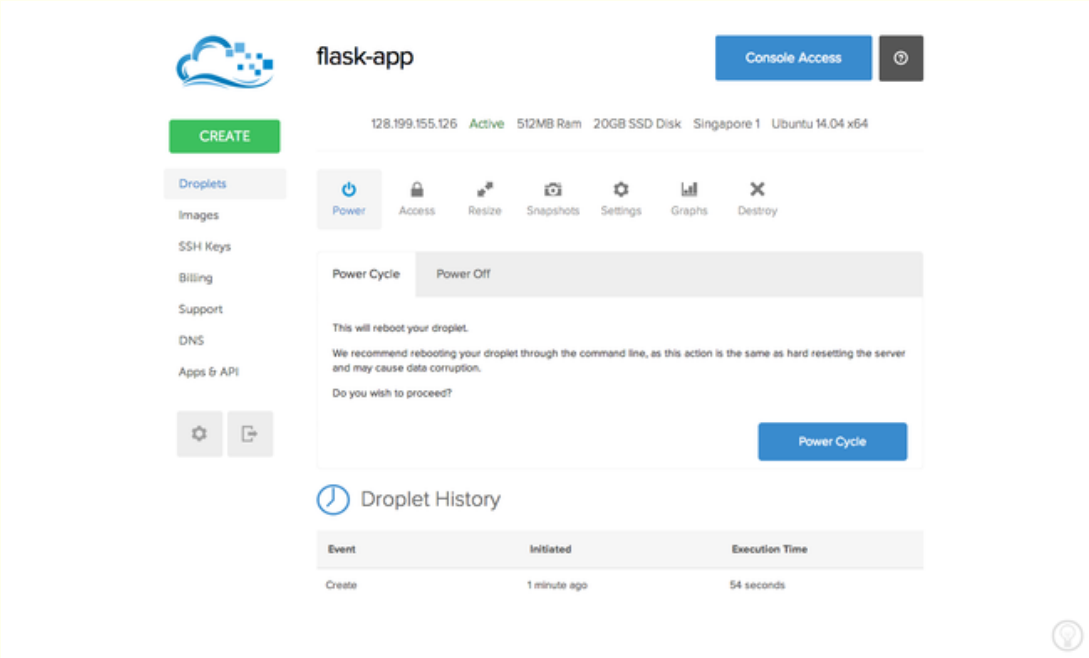
So you've built your first web application in Python using Flask. It works great in your `http://localhost:5000`. Now you want to show it to the world. Or maybe you're shy so just to your friends. You got a $10 coupon for DigitalOcean from some guy in Twitter. Let's use that. Now first things first: Create your droplet.

Droplet like it's hot



Give your droplet a witty name. Select its size. My app just spits out “Hello, world” so I chose the cheapest one. If you have an RDB running, you might want to chose a bigger size. Choose a region near your target audience. I didn’t tick off any of the available settings since I’m deploying a very simple (and useless) app. Select an image for your droplet. I chose the latest Ubuntu stable release (14.04) and I’ll be basing the succeeding instructions from it. Steps for other distros won’t be that different though. Especially for the Debian guys. Finally, add your SSH key. I won’t be discussing how to do this manually in

case you skip it. Click the big green “Create Droplet” button and ready your terminal for some SSH action.



So now you have a shiny new droplet. You’ll notice an IP address under your droplet’s host name. That’s your droplet’s public IP address. (There’s another address that will show up if you ticked “Private Networking” while creating your droplet. Don’t use that.)

Let’s SSH in:

```
ssh root@<DROPLET_PUBLIC_IP_ADDRESS>
# If you're not using default keys (id_rsa)
ssh root@<DROPLET_PUBLIC_IP_ADDRESS> -i <PATH_TO_PRIVATE
```

You’ll probably get something like this:

```
The authenticity of host '128.199.155.126 (128.199.155.1
RSA key fingerprint is 4c:2c:77:83:e7:e6:05:af:17:d9:28:
Are you sure you want to continue connecting (yes/no)?

Type in `yes` if it shows the right fingerprint or if yo
you’re doing. But seriously, you don’t need to worry abo
```

Security?

```
root@flask-app:~#
```

Now we're in. Notice that we're signed in as `root`. We don't want that but I want to keep this guide short and simple so we'll be skipping some security measures. If you really want to know how to secure your server, check this out: My First 5 Minutes On A Server; Or, Essential Security for Linux Servers.

We won't be hardening security but we're not savages. Let's create a user that will run the app for us instead of running it with `root`.

```
useradd -r -m -s /bin/bash deploy
```

This creates a system user named `deploy` with a home directory and uses `bash` as default shell.

Python tools

On to the Python stuff. We need to install `pip` and `virtualenv`.

```
apt-get install python-setuptools
easy_install -U pip
```

I bet you scratched your head after reading the commands above. *"Why is this dude telling me to install `setuptools` to install `pip`?"* The answer is simple: We want the latest and greatest `pip` version. We can get `pip` with `apt-get install python-pip` but it would install an older version. i.e. As of writing, `python-pip` in DO's APT repo is version `1.5.4-1` while latest one from PyPI is `1.5.6`.

```
pip install virtualenv
```

You must have encountered `virtualenvs` while working on your Flask app. If not, then you should start reading about it now. I'll wait… Done? Great!

Your code

I'm assuming you've been pushing your code to a DVCS repo. Most likely with `git`. Maybe hosted in GitHub. DO's Ubuntu image doesn't have `git` installed by default:

```
apt-get install git
```

Before cloning your repo, let's switch to the `deploy` user we created.

```
su - deploy
```

Now let's get your code. I'll be cloning mine as an example:

```
git clone https://github.com/marksteve/flask-hello-world
cd flask-hello-world
```

My app repo looks like this:

```
ls
hello_world.py  requirements.txt
```

I have my Flask app in `hello_world.py`:

```
from flask import Flask

app = Flask(__name__)


@app.route("/")
def index():
```

```
    return "Hello, world"


if __name__ == "__main__":
    app.run(debug=True)
```

And a `requirements.txt` that lists my dependencies:

```
Flask
gunicorn
```

Hold it… What the french is `gunicorn`?

Glad you asked. You see, Flask apps are actually WSGI apps so you can run them using WSGI servers.

Uhm… W-S-G-I?

It's pronounced *wizgy*. WSGI is basically a spec written by Python peeps about how web servers and web applications written in Python should be communicating with each other.

Gunicorn is a WSGI server that's quite easy to use and works out of the box. There are other WSGI servers worth checking out but we'll be sticking with Gunicorn because it's the easiest to work with IMO.

Ok, back to the guide. We're currently inside the code repo's root (in my case it's `/home/deploy/flask-hello-world`). We'll now create a `virtualenv` for our app:

```
virtualenv venv
source venv/bin/activate
```

> **NOTE**: You probably want your DVCS to ignore this `venv` folder. Or you could use `virtualenvwrapper`.

Now that we have our `virtualenv` activated, let's install our Python packages:

```
```

```
pip install -r requirements.txt
```

> **NOTE**: If you have packages that need dependencies installed with sudo access (e.g. Python package `MySQL-python` needs `libmysqlclient-dev` from APT), just press `Ctrl+D` to logout from the `deploy` user session. You'll be dropped back to your `root` session. Run `su - deploy` again to go back.

You can now try running your app with Gunicorn:

```
gunicorn -b 0.0.0.0:8000 hello_world:app
```

The command above runs `gunicorn` listening to any host at port `8000` using the WSGI app named `app` inside the `hello_world` module.

You should be able to see your app working at `http://<DROPLET_PUBLIC_IP_ADDRESS>:8000`.

Gunicorn setup

High five! Your server is up and running! But we have three problems here:

1. We probably want people to access our app using port `80` or `443` (i.e. not needing to type in the port number used).
2. Gunicorn isn't running as a daemon. Close your shell session and your server dies.
3. Gunicorn logs nothing (by default).

Issue #1

There are two ways to address issue #1. We can run Gunicorn with sudo access so it can listen to port `80` or `443`. Or better, we can use Nginx as a reverse proxy to the Gunicorn server. Go back to your `root` session by hitting `Ctrl+D` and install Nginx:

```
apt-add-repository ppa:nginx/stable
```

```
apt-get update
apt-get install nginx
```

Now we need to configure Nginx to point to Gunicorn:

```
vim /etc/nginx/conf.d/flask-app.conf
```

Put this in the configuration file:

```
server {
    listen 80;

    server_name _;

    access_log  /var/log/nginx/access.log;
    error_log  /var/log/nginx/error.log;

    location / {
        proxy_pass         http://127.0.0.1:8000/;
        proxy_redirect     off;

        proxy_set_header   Host             $host;
        proxy_set_header   X-Real-IP        $remote_addr
        proxy_set_header   X-Forwarded-For  $proxy_add_x
    }
}
```

> **NOTE**: If you've pointed a domain to your DO's public IP address, replace `server_name`'s value with that domain name. (e.g. `server_name marksteve.com`)

We also need to disable the default Nginx welcome page.

```
vim /etc/nginx/nginx.conf
```

Comment out the line:

```
include /etc/nginx/sites-enabled/*;
```

Tell Nginx to reload its config to apply our changes.

```
service nginx reload
```

Go back to your app's `virtualenv` and try running `gunicorn` again.

```
su - deploy
cd flask-hello-world
source venv/bin/activate
gunicorn hello_world:app
```

Notice that we didn't set an address for Gunicorn to bind to. By default Gunicorn binds to `127.0.0.1:8000` which is the address we pointed Nginx to.

Try going to `http://<DROPLET_PUBLIC_IP_ADDRESS>`. You should be seeing your app working. Look 'ma! No more port numbers in the address bar!

Issue #2

Easy. Just add `-D/--daemon` to your `gunicorn` arguments.

```
gunicorn -D hello_world:app
```

How do you kill it?

```
killall gunicorn
```

What if you just want to reload your app after pulling some changes?

```
killall -HUP gunicorn
```

Issue #3

Another easy one. Add arguments `--access-logfile FILE` for access logs and `--error-logfile FILE` for error

logs. You can set `FILE` to `-` to write to `stderr`.

`gunicorn` accepts config files so you don't have to write these arguments down everytime.

```
daemon = True
accesslog = "logs/access.log"
errorlog = "logs/error.log"
```

Save this as `gunicorn.conf.py` and tell Gunicorn to use it as its config file.

```
gunicorn -c gunicorn.conf.py hello_world:app
```

Gunicorn has a bunch of other configuration options you can find here.

Recap

By now you should know how to:

1. Create a droplet and access it
2. Create `virtualenv`s and install Python packages within them
3. Run Flask apps with Gunicorn
4. Setup Nginx as a reverse proxy for Gunicorn
5. Configure and run Gunicorn as a daemon process

You do? Then that's it! A simple deployment flow for your Flask apps in DigitalOcean droplets. If you have questions and/or recommendations, feel free to tweet me. I'm @themarksteve.

Say hi: hello@marksteve.com