Setting up an account

Creating an account

It is not safe to use the root account for everything, because of a few reasons. One of which is that it is easy to mess things up. Another of which is that the root account has all the permissions in your system. When you install programs, these programs create their own accounts. These accounts do not have all permissions, so they may be unable to access files and folders you created with the root account.

In short: lets create an account for your everyday usage of this server. I will call it jose.

Whenever there is a section of code, as below, execute it in your server. First, lets add the account:

```
/usr/sbin/adduser jose
```

Then, let’s give the account a password:

```
passwd jose
```

Now, lets allow the account to temporarily gain the permissions only the root account has, so that we can perform tasks otherwise we would not be able to. We will be able to temporarily gain these permissions, but the account will go back to losing these permissions after a short period, for security purposes.

Open the permissions file with the following command. This opens up the file using the default CentOS command-line editor, which is called vi.

```
visudo
```

Under the line giving the root user permissions, lets add a line to allow our new user to gain permissions.

To begin typing in vi press the key i. Then, type the following line below the line for the root user:

```
jose    ALL=(ALL)       ALL
```

To finish editing the file and save, press Esc, then the keys :wq, and press Enter. This will tell vi to write and quit.

Restricting who can log in to the server

We login to the server using the command ssh from our computers. In order to increase security, we are going to disable anybody from logging in directly as the root user, and instead we are only going to allow logging in as the jose user (or your user).

```
vi /etc/ssh/sshd_config
```

Find the following section:

```
Port 25000
Protocol 2
PermitRootLogin yes
UseDNS no
```

And change the line saying PermitRootLogin yes to PermitRootLogin no (you will need to press the key i to type in vi).

Now, press Esc and then the key combination SHIFT+G to go to the end of the file.

Insert a line at the end like the following (replacing the user by your user):

```
AllowUsers jose
```

Then, save and exit by pressing Esc, then :wq and Enter.

Finally, reload the ssh service to apply the changes:

```
service sshd reload
```

In your computer, open a terminal window (if using Mac or Linux), or download PuTTy (if using Windows). You can download PuTTY from here: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html.

Then, type ssh jose@<ip> where ip is the IP address of your server (it's on the e-mail DigitalOcean sent you after creating the droplet).

Type in the password you created for the account.

If all is good, you should log in and see something like this:

```
[jose@pricing-service ~]$
```

Remember to replace the user name by your user, and the droplet name by the one you provided.

If all is good, you can now log out of the root login that you were accessing through the DigitalOcean website, and only use the terminal (or if in Windows, PuTTy) login!

Installing Python

Run the following commands to install some of Python's dependencies. Commands you should type in the terminal are preceded by the symbol $.

```
$ sudo yum -y groupinstall "Development tools"
$ sudo yum -y install zlib-devel bzip2-devel openssl-devel
ncurses-devel sqlite-devel readline-devel tk-devel gdbm-
devel db4-devel libpcap-devel xz-devel
$ sudo yum -y install wget
```

Then, run the following commands to install Python3.5.1:

```
$ wget http://python.org/ftp/python/3.5.1/Python-3.5.1.tar.xz
$ tar xf Python-3.5.1.tar.xz
$ cd Python-3.5.1
$ sudo ./configure --prefix=/usr/local --enable-shared
LDFLAGS="-Wl,-rpath /usr/local/lib"
$ sudo make && sudo make altinstall

$ cd ..

$ export PATH="/usr/local/bin:$PATH"
```

Now lets install virtualenv and change permissions to two folders that we will need afterwards:

```
# We are going to need necessary permissions in the local
account to change stuff around

$ sudo chmod 777 -R /usr/local/bin
$ sudo chmod 777 -R /usr/local/lib

# Then install virtualenv, since we will need them for the
dashboard
$ pip3.5 install virtualenv
```

Installing nginx

Run the following commands to install nginx in your system, and to make sure it re-starts if you restart the server:

```
$ sudo yum -y install epel-release
$ sudo yum -y install nginx
$ sudo chkconfig --levels 235 nginx on
```

Now, modify the nginx config file by executing the following command:

```
$ sudo vi /etc/nginx/conf.d/default.conf
```

Delete everything by pressing d followed by SHIFT+G, and then press i and paste the following content (on the next page):

```
server {
    listen       80;
    real_ip_header X-Forwarded-For;
    set_real_ip_from 127.0.0.1;
    server_name  localhost;

    location / {
        include uwsgi_params;
        uwsgi_pass unix:/var/www/html/pricing-service/
socket.sock;
       uwsgi_modifier1 30;
    }

    error_page  404              /404.html;
    location = /404.html {
        root   /usr/share/nginx/html;
    }

   error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

Next, let’s modify another configuration file by executing the following command:

```
$ sudo vi /etc/nginx/nginx.conf
```

Find the line that says worker_processes 1;, and change it so it says worker_processes 8;.

Then we must add the nginx user to the jose group (the video has it the wrong way round, instead it says we are adding the jose user to the nginx group—small mistake!).

Remember the group will be called the same as the user you created for your server initially.

```
$ sudo usermod -a -G jose nginx
```

Installing the app

We must create the necessary directories for our application to live in first. This is where we'll put our application for nginx and uwsgi to be able to run it.

```
$ sudo mkdir /var/www && sudo mkdir /var/www/html && sudo
mkdir /var/www/html/pricing-service
$ sudo chown jose:jose /var/www/html/pricing-service/
```

Navigate to the /var/www/html/pricing-service folder and clone the git repository. Remember that in order to be able to clone the repository, you will have to create a fork of it, and clone the fork. Do not clone the repository as shown on this document, because it will not work.

```
cd /var/www/html/pricing-service/

git clone git@github.com:schoolofcode-me/price-of-chair.git .
```

Delete everything by pressing d followed by SHIFT+G, and then press i and paste the following content (on the next page):

Generating an SSH key

Run the command:

```
ssh-keygen
```

And press Enter whenever prompted to enter anything. You can enter a passphrase —just a fancy name for a password—if you want. This will be required whenever you use the SSH key (e.g. pushing or pulling from GitHub).

Now, copy the contents of your newly-created SSH key:

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

Copy the output of that command, and let's go to GitHub!

Adding an SSH to GitHub

Log in to GitHub, and go to your profile by clicking on the top right.

Press the “Edit Profile” button, and then go to the section on SSH keys.

Add your new key. You will need to choose a title, but it is not important. The content must exactly reflect what you copied from the last command executed on the server. Then, you can attempt to clone the repository again!

Creating the log directory

Once you have cloned the directory, we need to do a last couple of things to set everything up.

uWSGI will need a directory in which to write what happens while the system is running. We must create this directory:

```
mkdir /var/www/html/pricing-service/log
```

Creating the virtual environment

uWSGI will try to run from a virtual environment which should be located at /var/www/html/pricing-service/venv. Lets create that now:

```
cd /var/www/html/pricing-service
virtualenv --python=python3.5 venv
./venv/bin/pip install -r requirements.txt
./venv/bin/pip install uwsgi
```

The commands above will create the virtual environment, install the requirements.txt as per the application, and then install uwsgi as well. This is a Python package that is required to run the uWSGI server.

Adding the config properties

Next, we need to create the configuration file for the project:

```
vi src/config.py
```

Place the following content in it:

```
DEBUG = False
ADMINS = frozenset(["yourname@yourdomain.com"])
```

Then, add your MailGun API details in the src/models/alerts/constants.py file:

```
vi src/models/alerts/constants.py
```

And make sure that the URL, API_KEY, and FROM properties are set to your MailGun properties.

Installing MongoDB

Execute the following command to create the MongoDB repository file. This is necessary so that the package manager yum knows where to download MongoDB from:

```
sudo vi /etc/yum.repos.d/mongodb-org-3.2.repo
```

And put the following content in it:

```
[mongodb-org-3.2]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/
mongodb-org/3.2/x86_64/
gpgcheck=0
enabled=1
```

Install MongoDB using yum:

```
sudo yum install -y mongodb-org
```

Make sure MongoDB restarts whenever you restart the server:

```
sudo chkconfig mongod on
```

Start MongoDB:

```
sudo service mongod start
```

Setting up uWSGI

uWSGI is a server that wraps around the Flask application and can serve the application to nginx. nginx can then give the content to the users.

Create the uWSGI configuration file:

```
sudo vi /etc/init/uwsgi_pricing_service.conf
```

And put the following content in it:

```
description "uWSGI_pricing_service"
start on runlevel [2345]
stop on runlevel [06]
respawn

env UWSGI_ALIVE=/var/www/html/pricing-service/venv/bin/uwsgi
env LOGTO_ALIVE=/var/www/html/pricing-service/log/emperor.log

exec $UWSGI_ALIVE --master --emperor /var/www/html/pricing-
service/uwsgi.ini --die-on-term --uid jose --gid jose --logto
$LOGTO_ALIVE
```

Remember to change --uid jose and --gid jose to be your user name and group name (they both tend to be equal to your user name).

At any point, you can start the uWSGI service by running sudo start uwsgi_pricing_service and stop it with sudo stop uwsgi_pricing_service.

In the /var/www/html/pricing-service/ folder, create a file uwsgi.ini:

```
cd /var/www/html/pricing-service/
vi uwsgi.ini
```

And put the following content in it:

```
[uwsgi]
#application's base folder
base = /var/www/html/pricing-service

#python module to import
app = src.app
module = %(app)

home = %(base)/venv
pythonpath = %(base)

#socket file's location
socket = /var/www/html/pricing-service/socket.sock

#permissions for the socket file
chmod-socket = 777

#add more processes
processes = 8

#add more threads
threads = 8

#kill worker if timeout > 15 seconds
harakiri = 15

#the variable that holds a flask application inside the
module imported at line #6
callable = app

#location of log files
logto = /var/www/html/pricing-service/log/%n.log
```

Running the app

To run the application, we just have to execute the following:

```
sudo service nginx start
sudo start uwsgi_pricing_service
```

Then, you can access the IP address of your server, and the application should load!

It is possible that it doesn't. In that case, maybe you need to disable SELinux. This is not something I recommend lightly, because it might incur some security problems later on down the line.

To disable SELinux:

```
sudo setenforce 0
```

It is possible to add nginx as an exception in SELinux, but these lower level security concerns are outside the scope of this course.

Setting up a cron job

In order to make sure that the alerting system is working, and checking for alerts every 15 minutes, we can set up a cron job.

The cron job just runs a program we specify every 15 minutes. We could just run the following program:

```
/var/www/html/pricing-service/venv/bin/python /var/www/html/
pricing-service/src/alert_updater.py
```

But running that program by itself causes an error, because PYTHONPATH has not been set, and thus the python file cannot find the Database class.

We must create a program that sets the PYTHONPATH, and then executes the script. First, create the program file:

```
vi /var/www/html/pricing-service/run_alert_updater.sh
```

Then, place the contents in it to export the PYTHONPATH and run the updater:

```
export PYTHONPATH=/var/www/html/pricing-service
/var/www/html/pricing-service/venv/bin/python /var/www/html/
pricing-service/src/alert_updater.py
```

Make sure the program is runnable. This means it must have "executable" permissions. We can add executable permissions to a program by running the following command:

```
sudo chmod ugo+x /var/www/html/pricing-service/
run_alert_updater.sh
```

Finally, modify the crontab file where we will put our program and set it to run every 15 minutes:

```
sudo vi /etc/crontab
```

Add the following line at the end to start running our program:

```
*/15 * * * * jose /var/www/html/pricing-service/
run_alert_updater.sh
```