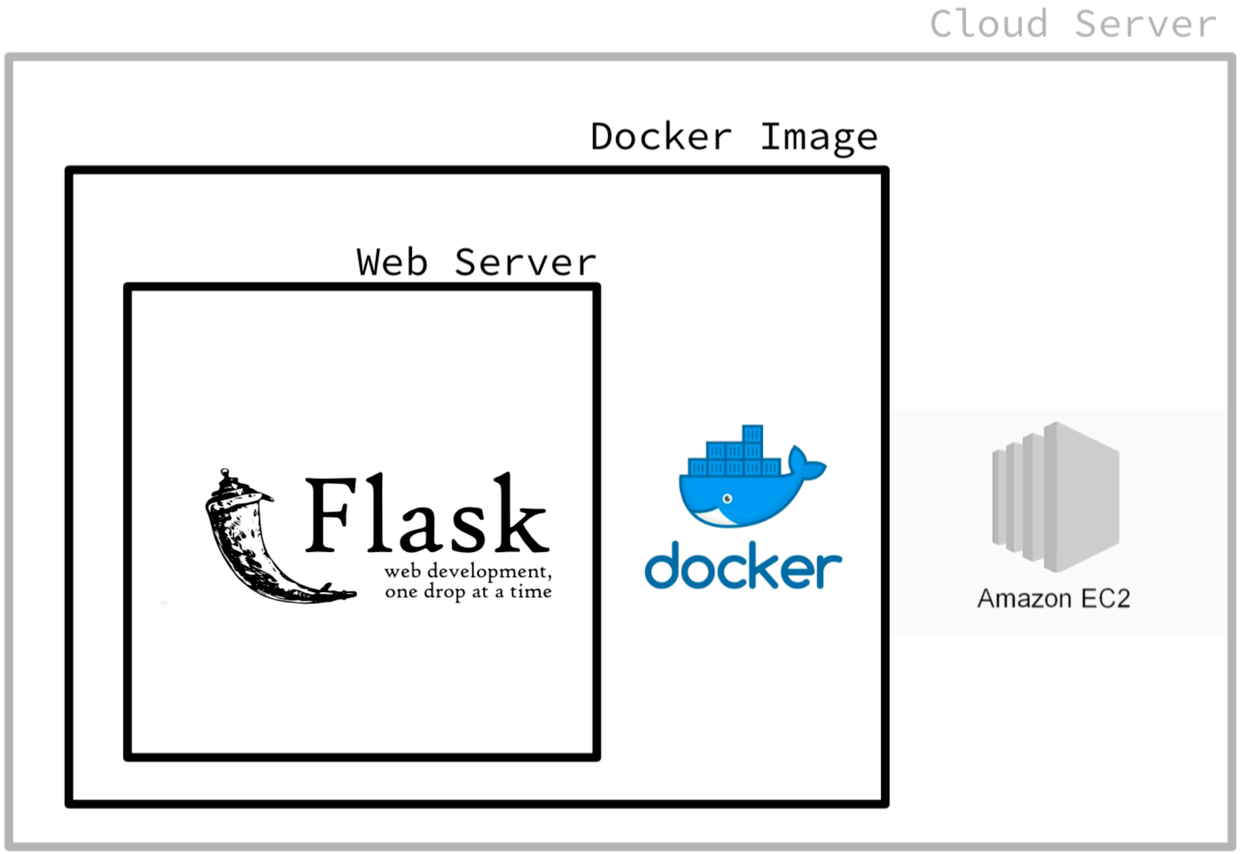
What are we doing here?

Created a machine learning model…now I want someone to use it!

Lets start to think about how we can deploy our model so someone else can access it (without knowing much of what's going on underneath)

Expose a web API (locally for today)

What are we going to build?



What is an API anyways?

Technically, API stands for **Application Programming Interface**. At some point or another, most large companies have built APIs for their customers, or for internal use.

When you type www.facebook.com into your browser, a request goes out to Facebook's remote server. Once your browser receives the response, it interprets the code and displays the page.

An API isn't the same as the remote server—rather it is the part of the server that receives requests and sends responses.

When a company offers an API to their customers, it just means that they've built a set of dedicated URLs that return pure data responses—meaning the responses won't contain the kind of presentational overhead that you would expect in a graphical user interface like a website. (thanks https://medium.freecodecamp.org/what-is-an-api-in-english-please-b880a3214a82)

REST API

A REST API defines a set of functions which developers can perform requests and receive responses via HTTP protocol such as GET and POST.

REST stands for Representational state transfer which essentially refers to a style of web architecture that has many underlying characteristics and governs the behavior of clients and servers.

An API can be considered “RESTful” if it has the following features (main features of R):

Client–server – The client handles the front end the server handles the backend and can both be replaced independently of each other.

Stateless – No client data is stored on the server between requests and session state is stored on the client.

Cacheable – Clients can cache response (just like browsers caching static elements of a web page) to improve performance

What is Flask?

A microframework for Python, meaning it has little to no dependencies to external libraries.

It really is a web framework providing tools, libraries and technologies to build web applications. (in our case an API)

Flask can create a REST API that allows you to send data, and receive a prediction as a response.

What are we doing with Flask?

We will create routes for our api that receives different requests (GET and POST), specifically a POSTing json data

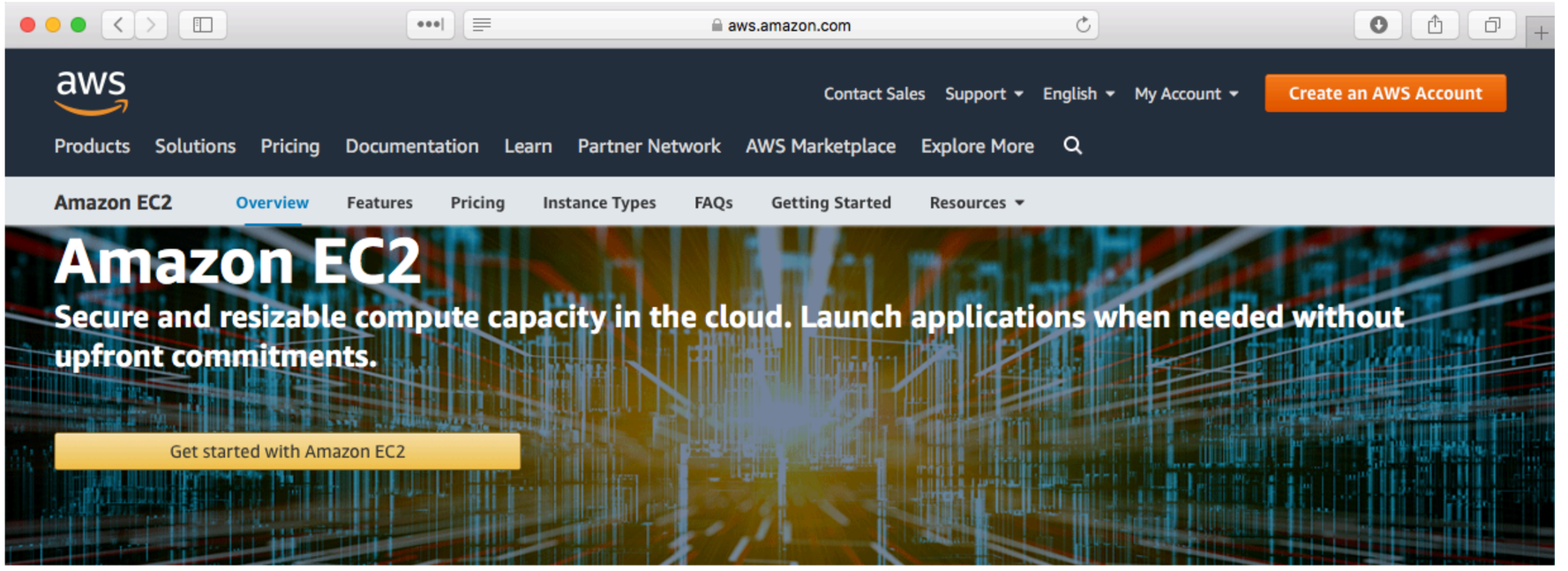
Once our flask server receives json data it will apply a function and then return some value also in json format

What are we doing with Flask?

Last week we created a locally deployed API to help us understand the process through which we could Our local API was a Flask app contained in a docker container…the portability of the docker container will come in handy here.

Now let's move this idea to a cloud based service, an Amazon Web Service ec2 instance (we get plenty of free resources!), where people outside our local machines can access our API

It should be noted that this is still simply to give us an idea about the deployment process, but we will be still falling a bit short of full-on productionalization. (no load balancing, no security, no testing, etc.)



Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.

Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate them from common failure scenarios.

Try Amazon EC2 for Free

AWS Free Tier includes 750 hours of Linux and Windows t2.micro instances each month for one year. To stay within the Free Tier, use only EC2 Micro instances.

View AWS Free Tier details >>

(https://aws.amazon.com/ec2/)

You will need an AWS account to get access to the necessary resources.

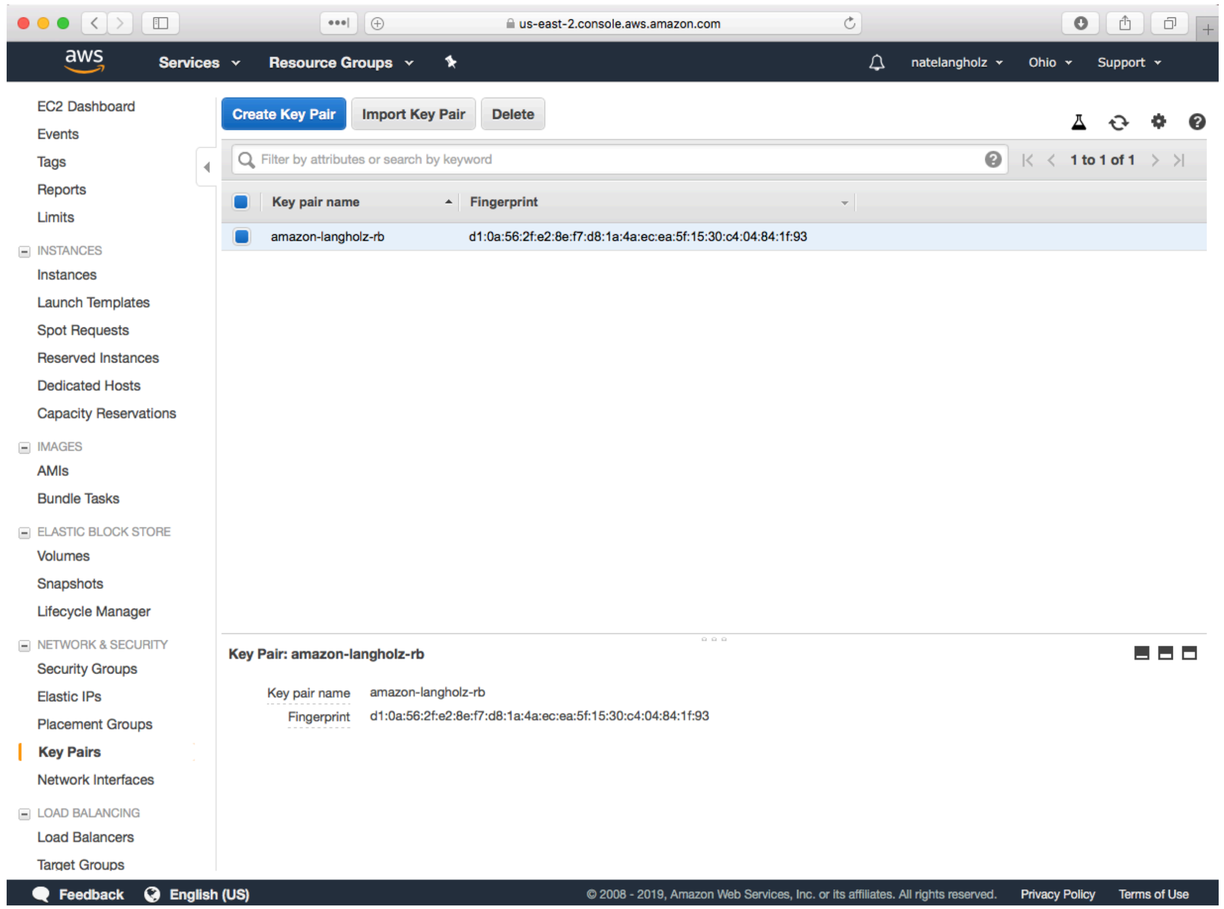
Create one now; make it a personal account.

You will need to verify your email (and maybe enter a credit card; won’t be billed)

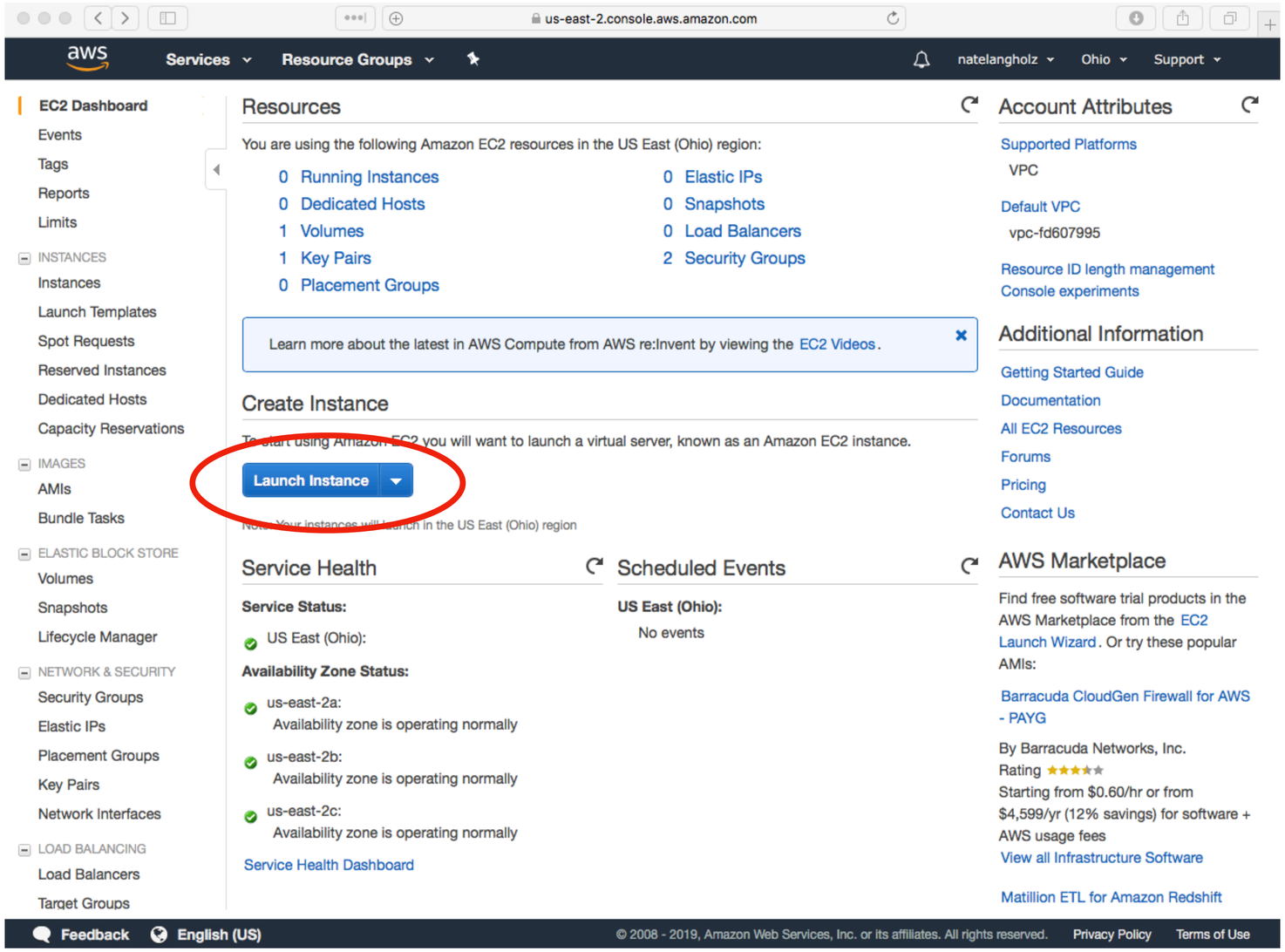
Once you have an account let's start the process. Login.

Look for the 'Key Pairs' section under Network & Security. Create one with a simple name which will download a `.pem` file. This is your key.

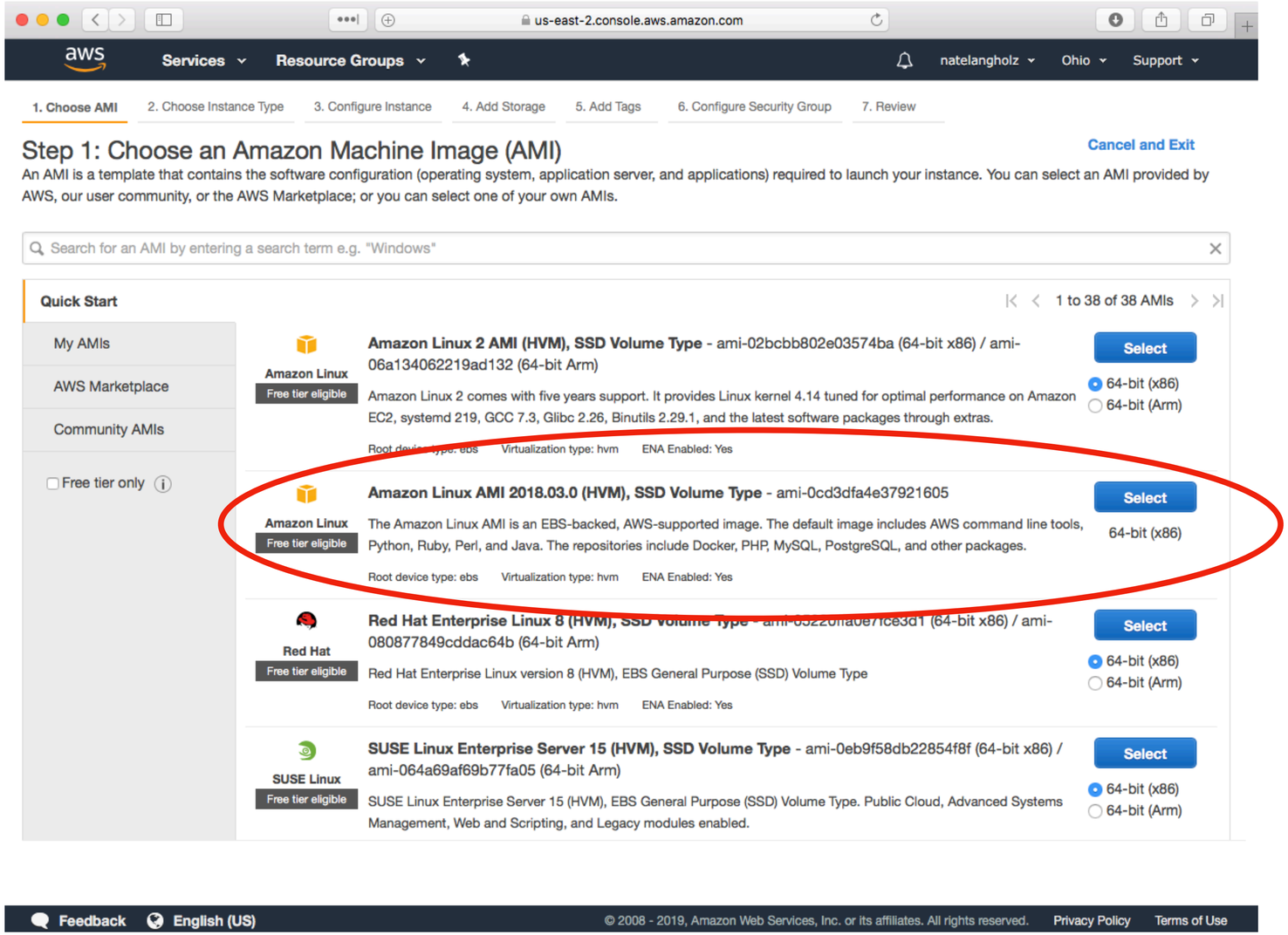
Save it somewhere with an easy path. For now I've put mine under my week-7 directory.



`Launch` an instance

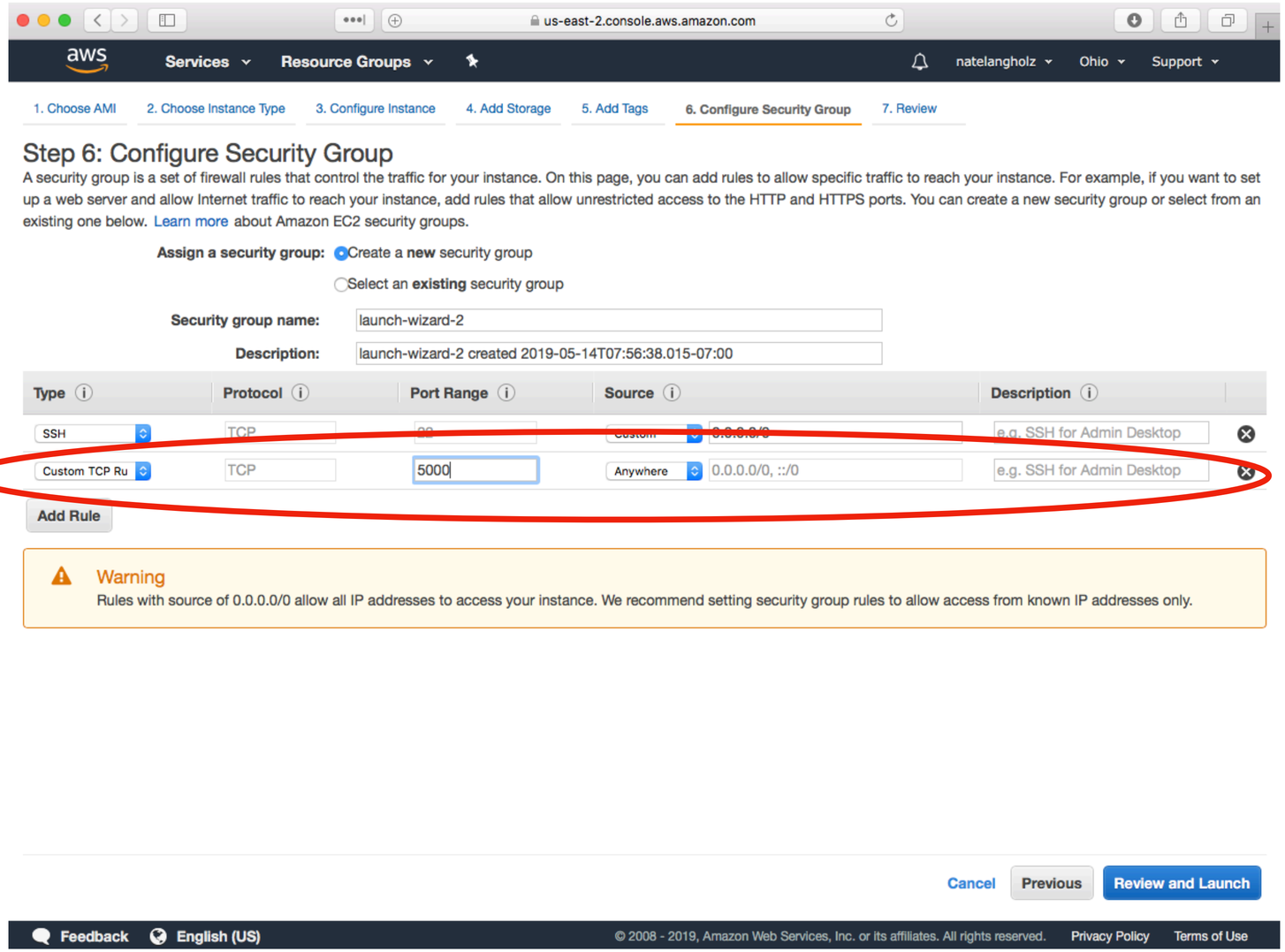


Choose an Amazon Machine Instance from the options.



Using the Amazon Linux AMI because its Free tier eligible and the repos include Docker. (I believe the Linux 2 AMI would also work)

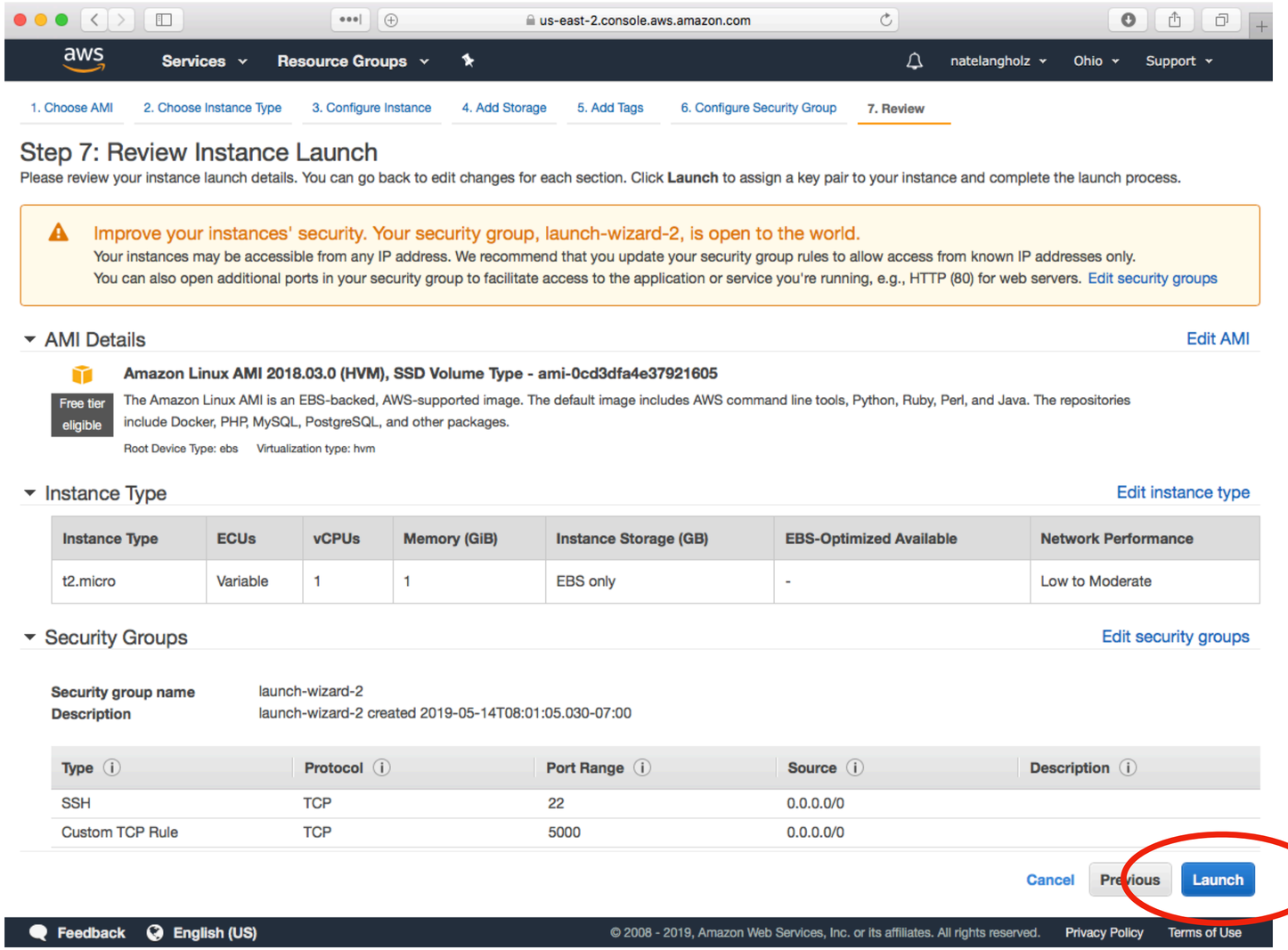
Configure Security Group; click `add rule`; the new rule will allow inbound requests



Pick `Custom TCP Rule`
Enter 5000 into the Port Range (match whatever port you have used in your local app)
Change Source to `Anywhere`

Review and Launch

You'll notice the warning about improving security. For what we're doing this should be okay not to worry about.



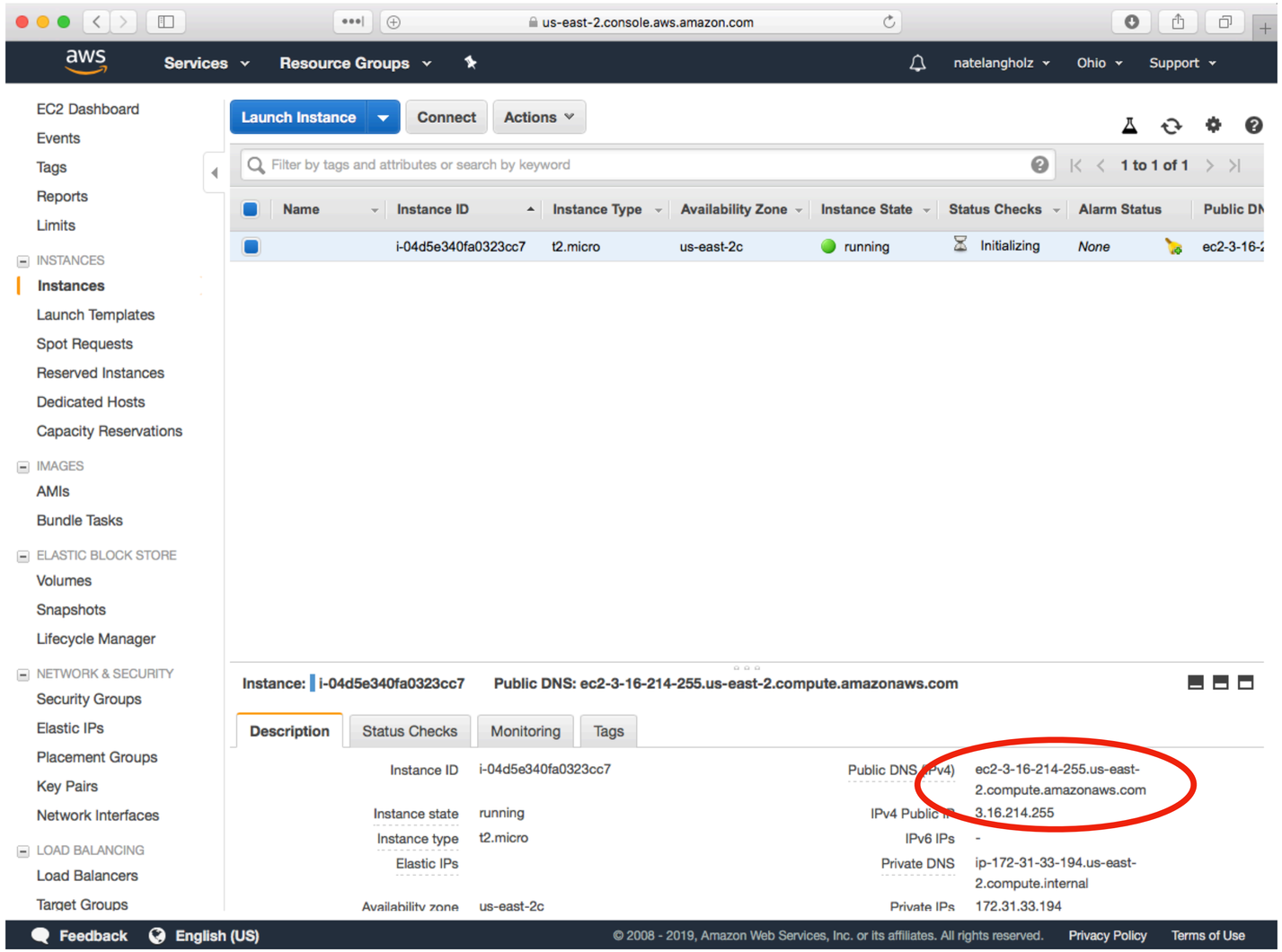
Should be okay to Launch at this point

After clicking launch you should see a pop-up looking for confirmation of your `key-pair`

Use the name of your key-pair that we generated earlier

Now launch the AMI and view instances

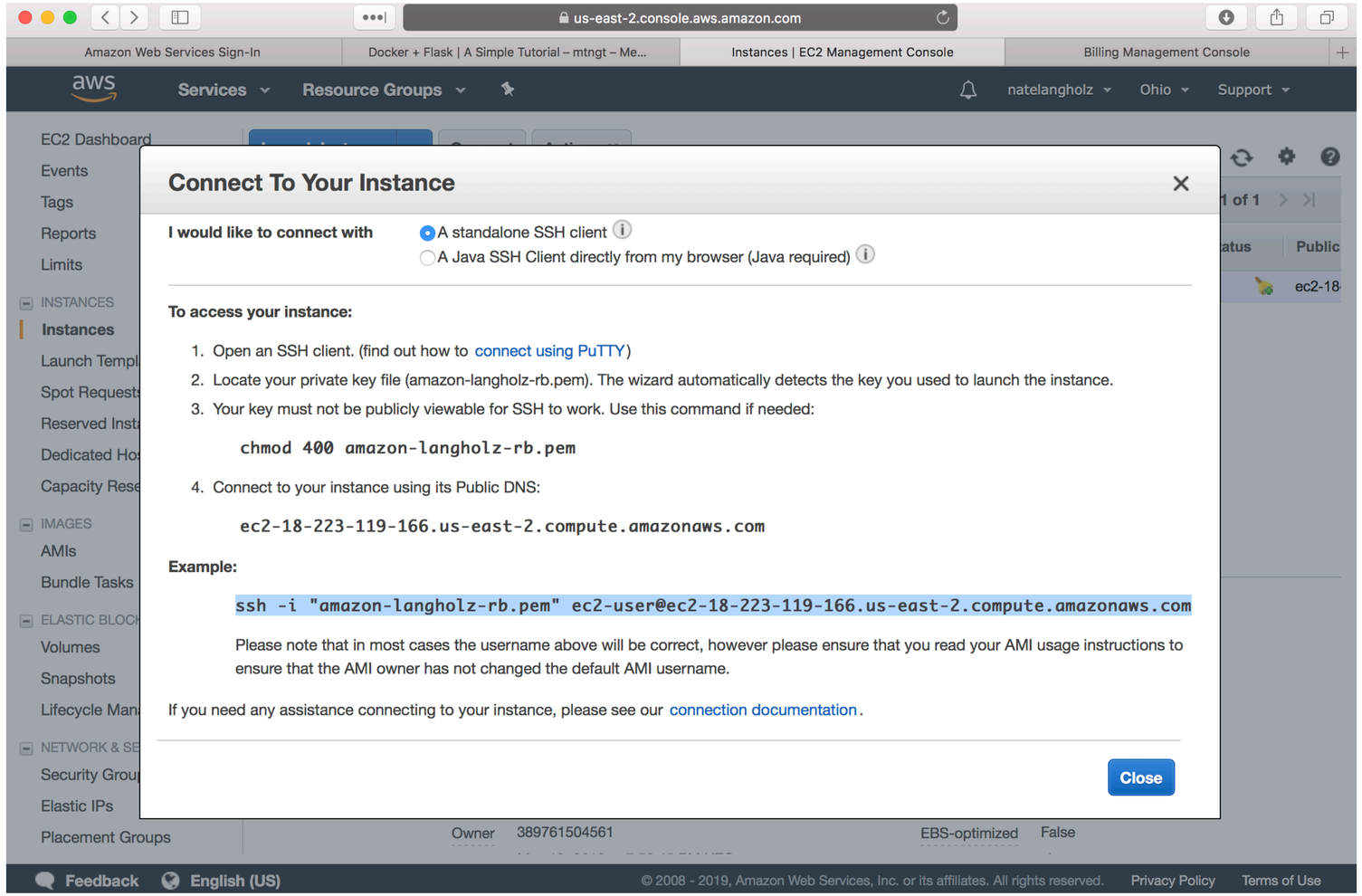
ec2 Dashboard



Observe that your Instance state is running

Also note your Public DNS, as this is where you will access this instance

Connect to your instance; this is an overview of the instructions. I'll try to detail them a bit more. (See these by clicking `Connect` in your dashboard)



Remember our Key?

PEM or Privacy Enhanced Mail is a Base64 encoded DER certificate. PEM certificates are frequently used for web servers as they can easily be translated into readable data using a simple text editor. Generally when a PEM encoded file is opened in a text editor, it contains very distinct headers and footers. (https://support.quovadisglobal.com/kb/a37/what-is-pem-format.aspx)

We need to to change the permissions on the key to private

For Mac and Linux users

```
chmod 400 key-file.pem
```

For Windows (if above doesn't work)…

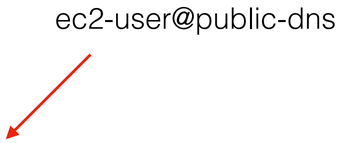
Windows solutions to chmod and ssh and scp (see on slack as well)

Possible solution 1: Download and install Gnu on windows (GOW) from https://github.com/bmatzelle/gow/wiki

Possible solution 2: (Re) install git for windows which gives you the option to add unix tools (chmod and ssh included) https://hackernoon.com/install-git-on-windows-9acf2a1944f0

Possible solution 3: connect using PuTTY as described in step 1 of connecting to your instance in the Amazon instructions https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html

Let’s ssh into our AMI




```
ssh -i "amazon-langholz-rb.pem" ec2-user@ec2-18-223-119-166.us-east-2.compute.amazonaws.com
```

Secure Shell (**SSH**) is a cryptographic network protocol for operating network services securely over an unsecured network. Typical applications include remote command-line login and remote command execution, but any network service can be secured with SSH. (https://en.wikipedia.org/wiki/Secure_Shell)

For windows if you are having trouble with the ssh command here are resources to help with connecting (https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html)

What does it look like if correctly connected?

week-7 — ec2-user@ip-172-31-33-194:~ — ssh -i amazon-langholz-rb.pem ec2-user@ec2-3-16-214-255.us-east-2.compute.amazonaws.com — 167×49

```
NA-nlanghol-A02:week-7 nlangholz$ ssh -i "amazon-langholz-rb.pem" ec2-user@ec2-3-16-214-255.us-east-2.compute.amazonaws.com
The authenticity of host 'ec2-3-16-214-255.us-east-2.compute.amazonaws.com (3.16.214.255)' can't be established.
ECDSA key fingerprint is SHA256:flBanriZO952jcSUdPhOF8SIRser92GnVJrYBb6P8u0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-3-16-214-255.us-east-2.compute.amazonaws.com,3.16.214.255' (ECDSA) to the list of known hosts.
Last login: Tue May 14 03:38:27 2019 from cpe-172-91-122-53.socal.res.rr.com

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
[ec2-user@ip-172-31-33-194 ~]$ ls
flask-app-small
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$ ls -al
total 36
drwx------ 4 ec2-user ec2-user 4096 May 14 04:46 .
drwxr-xr-x 3 root     root     4096 May 13 16:49 ..
-rw------- 1 ec2-user ec2-user 2197 May 14 04:59 .bash_history
-rw-r--r-- 1 ec2-user ec2-user   18 Aug 30  2017 .bash_logout
-rw-r--r-- 1 ec2-user ec2-user  193 Aug 30  2017 .bash_profile
-rw-r--r-- 1 ec2-user ec2-user  124 Aug 30  2017 .bashrc
drwxrwxr-x 4 ec2-user ec2-user 4096 May 14 04:17 flask-app-small
drwx------ 2 ec2-user ec2-user 4096 May 13 16:49 .ssh
-rw-rw-r-- 1 ec2-user ec2-user  209 May 14 04:23 .wget-hsts
[ec2-user@ip-172-31-33-194 ~]$
[ec2-user@ip-172-31-33-194 ~]$
```

lic-dns

Once inside, this is a Linux machine so everyone should be operating under the same build

Time to install Docker and Docker-compose

Update the environment installed packages and cache

```
sudo yum update -y
```

Install Docker

```
sudo yum install -y docker
```

Add the ec2-user to the docker group so you can execute Docker commands without sudo

```
sudo usermod -a -G docker ec2-user
```

this pulls docker-compose from GitHub

```
sudo curl -L https://github.com/docker/compose/releases/download/1.21.0/docker-compose-
`uname -s`-`uname -m` | sudo tee /usr/local/bin/docker-compose > /dev/null
```

```
sudo chmod +x /usr/local/bin/docker-compose
```

(honestly not sure if below is needed; try anyways)

```
ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

Install docker-compose

```
docker-compose
```

Start the docker service

```
sudo service docker start
sudo chkconfig docker on
```

Check docker-compose version

```
docker-compose --version
```

To leave your AMI, simply type `exit`

Let’s add some files from our local machine through secure copy

```
scp -i "amazon-langholz-rb.pem" -r flask-api-small/ ec2-
user@ec2-3-14-13-135.us-east-2.compute.amazonaws.com:/
home/ec2-user
```

SSH back into your AMI instance and run ls. You should see your directory; proceed as usual with docker.

Change to the docker directory, run `docker-compose up -d`

Then test the connection there in your instance using a curl command

To stop your API, within your AMI stop your docker container as usual.

Either

```
Docker-compose stop
```

or

```
Docker container ls

Docker container kill <container-name>
```

Then exit your AMI by simply typing `exit`

Another easy way to create an easy deployment is through GitHub.

Remember this is a linux environment with bash setup so we can use these commands.

We have a nice completed example of homework 3 in a standalone repository. We can get a zip file through

```
wget https://github.com/tannerkoscinski/MTCars-Flask-API/archive/master.zip
```

Unzip using

```
unzip *.zip
```

And then remove the zip file as we no longer need it

```
rm master.zip
```

Let’s take a look at making predictions on my api from your machines

Lastly to remove everything from that repository

```
sudo rm -rf MTCars-Flask-API-master/
```

STOP your instance from continuing to run when not in use!!!