CSCI 466: Networks

Cloud Computing, Deploying using AWS

Reese Pearsall

Fall 2024

Announcements

Workday on 11/22 and 12/2 (no class)

PA4 due 11/24

Flask App

```python
from flask import Flask,request

app = Flask(__name__)

@app.route('/')
def example():
    return "Hello World!"

@app.route('/reese')
def reese():
    return "I am reese!"

@app.route('/raven')
def raven():

    name = request.headers['name']

    file = open("raven.txt", "r", encoding="utf8")
    filetext = file.read()


    return "Hello there, " + name + "\n \n" + filetext

if __name__ == '__main__':
    app.run()
```

Sending HTTP requests to API

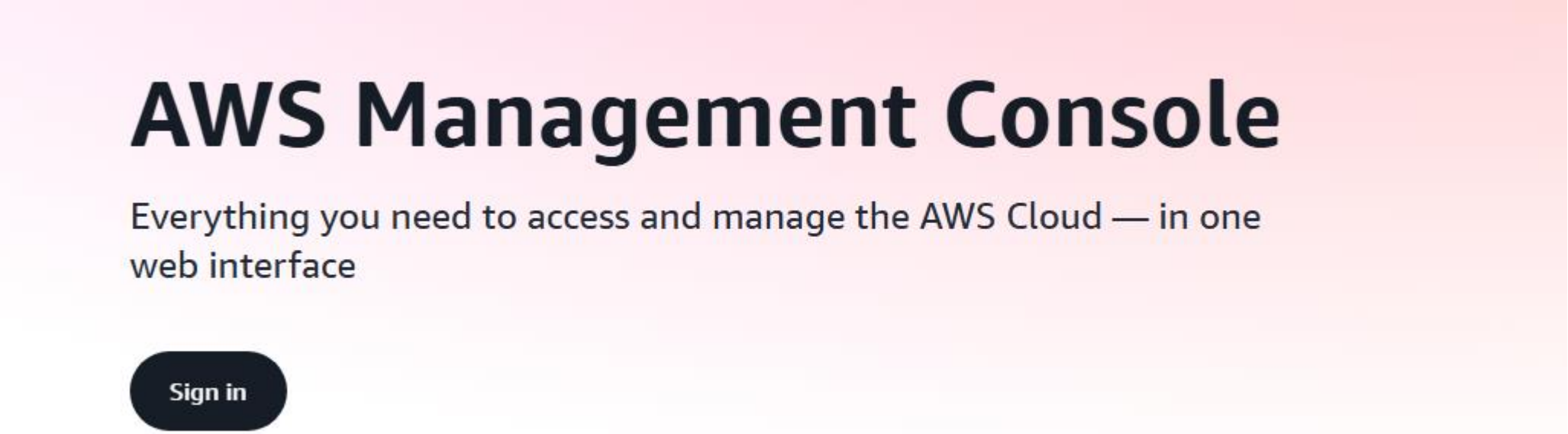
```
C:\Users\reese>curl --header "name: reese" 127.0.0.1:5000
Hello, reese

Once upon a midnight dreary, while I pondered, weak and weary,
Over many a quaint and curious volume of forgotten lore—
    While I nodded, nearly napping, suddenly there came a tapping,
As of some one gently rapping, rapping at my chamber door.
“’Tis some visitor,” I muttered, “tapping at my chamber door—
```

We have built our API, and we have tested it locally. We are ready to deploy it to the cloud!

Amazon AWS

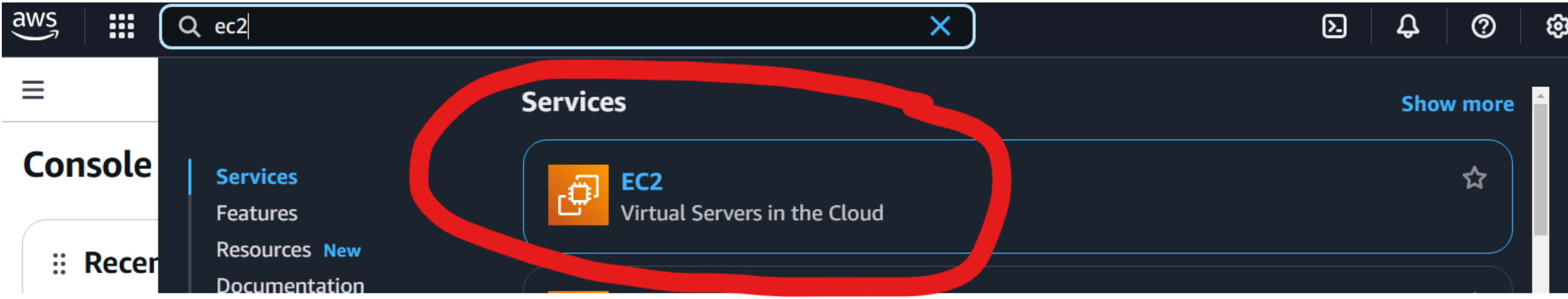
https://aws.amazon.com/console/

First, we will need to make an account, and then log into the AWS console

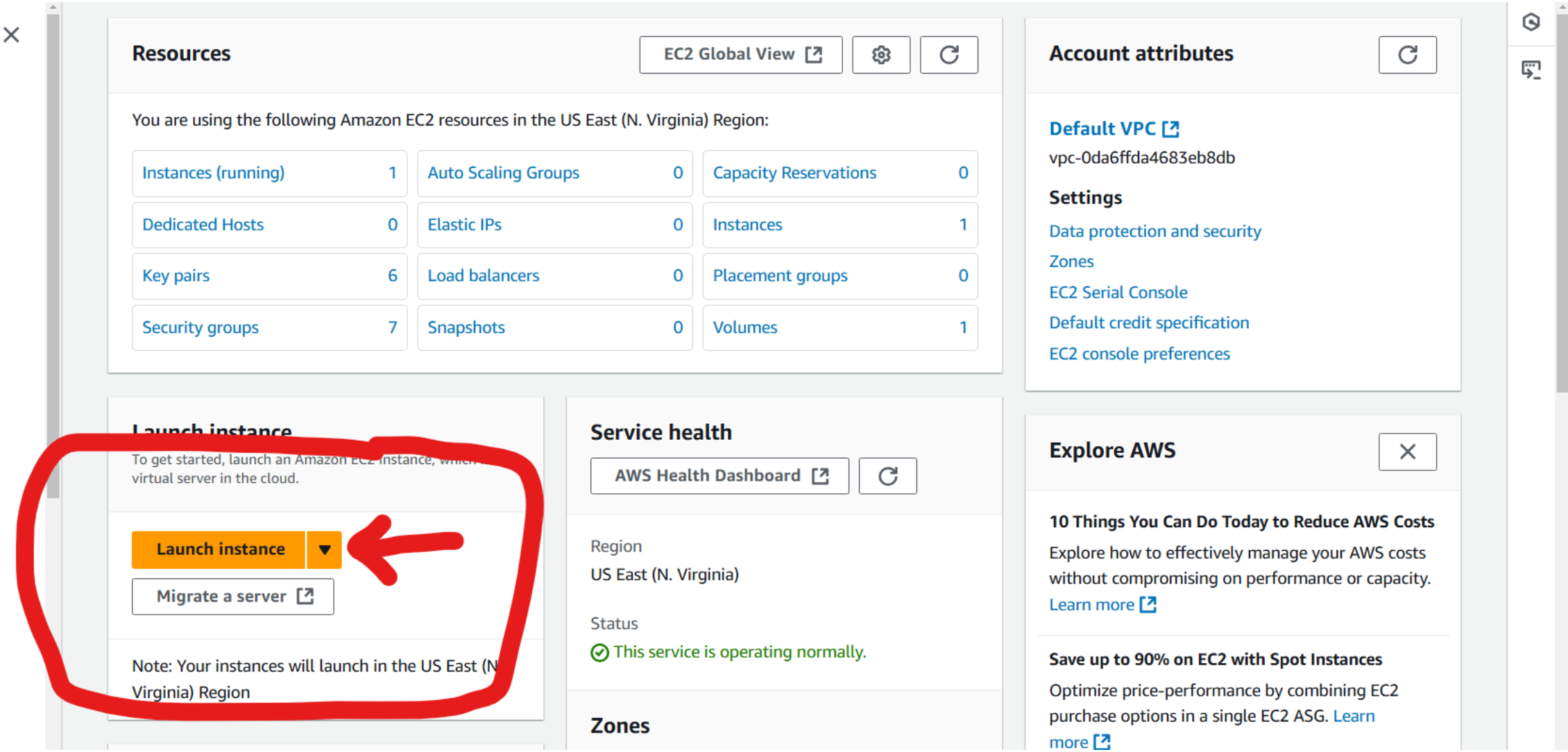


Amazon AWS

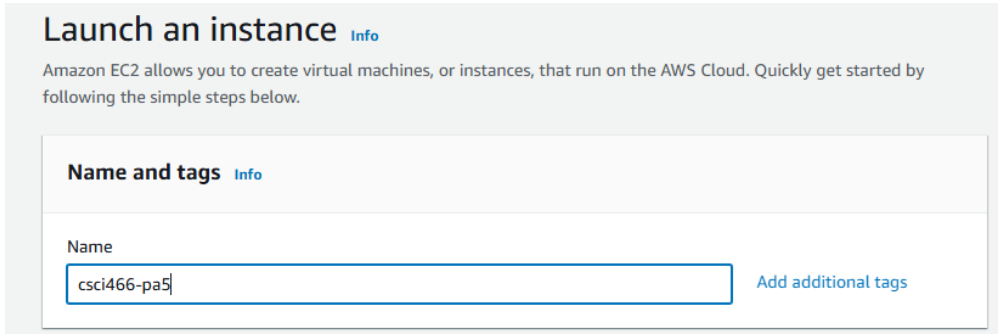
We will put our application on an **EC2** instance (a virtual server in the cloud)



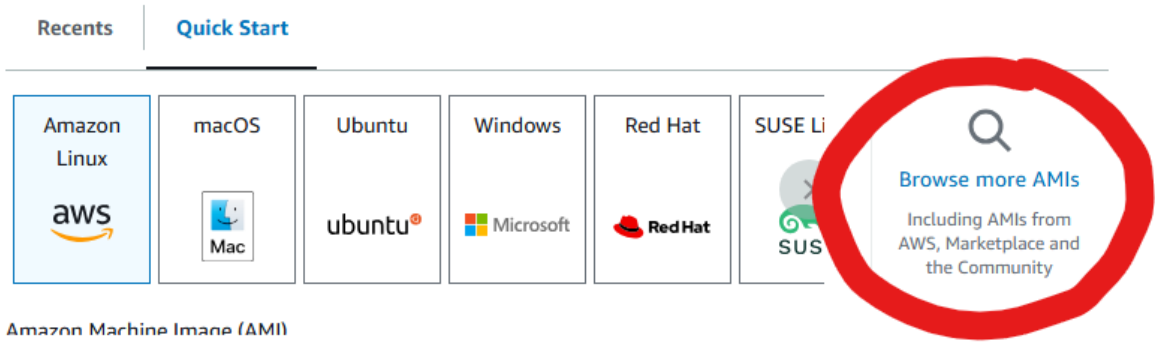
Deploying with AWS



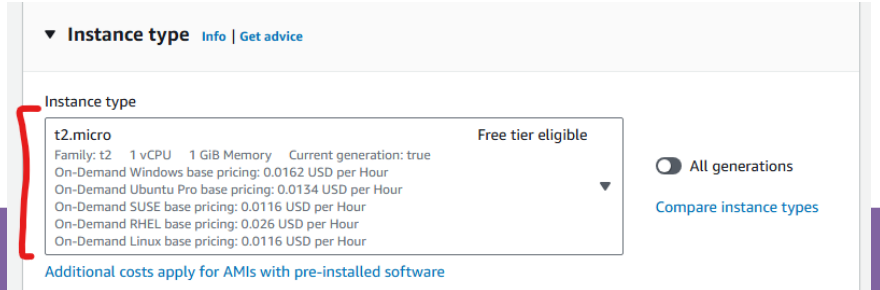
Deploying with AWS



Give your EC2 a name

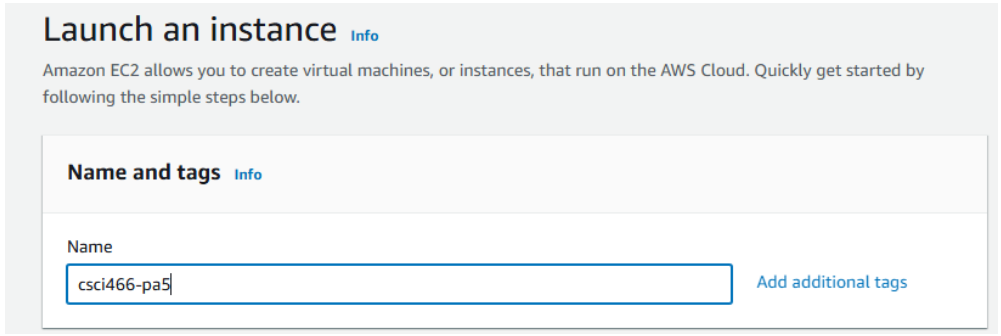


Our cloud machine will be an Ubuntu 22.04 web server

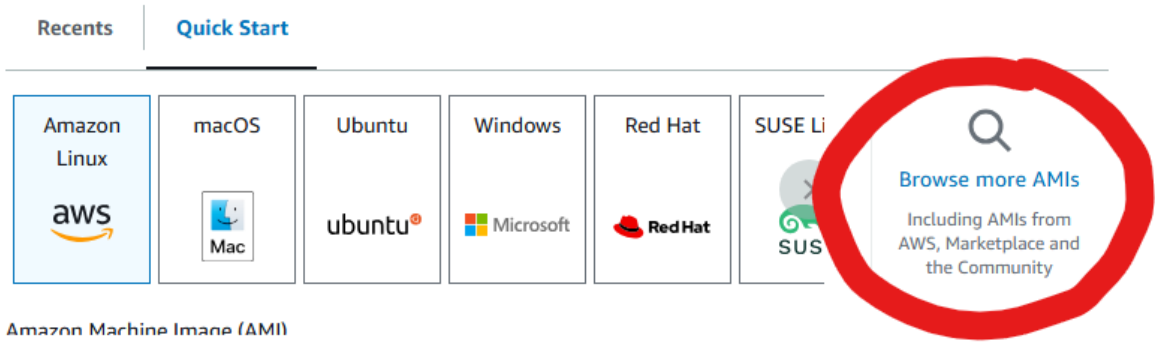


Use the t2.micro instance type

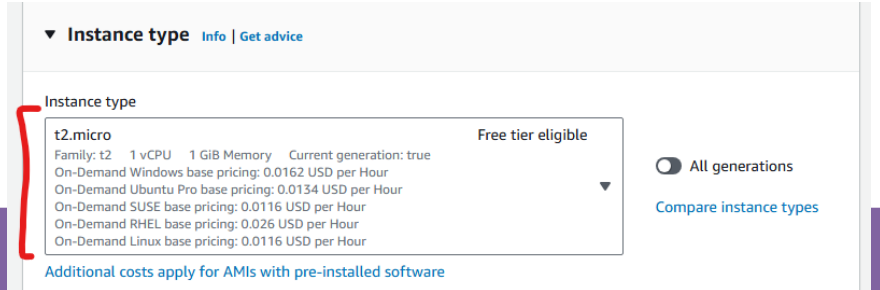
Deploying with AWS



Give your EC2 a name



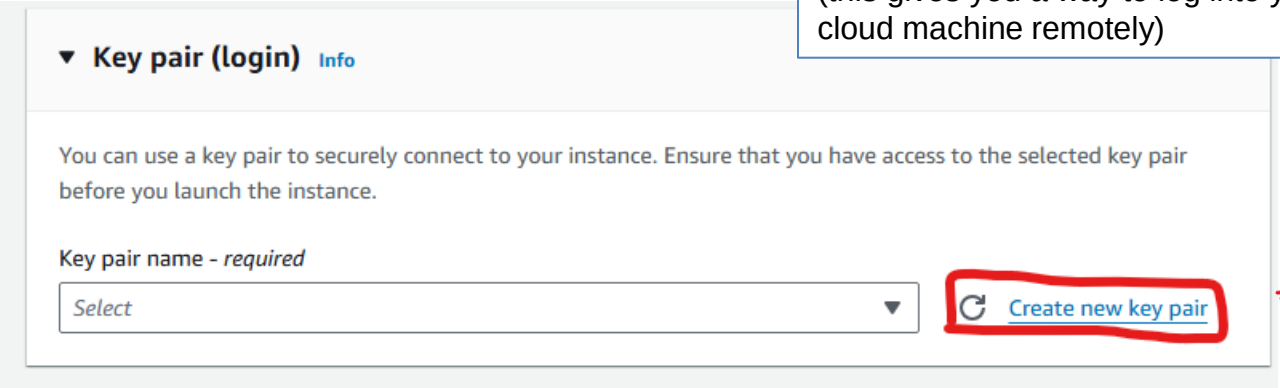
Our cloud machine will be an Ubuntu 22.04 web server

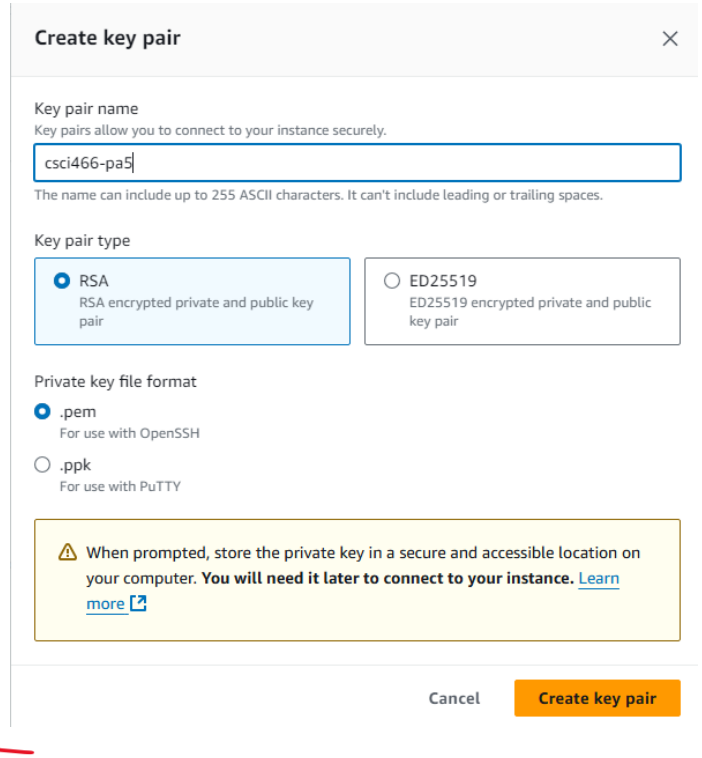


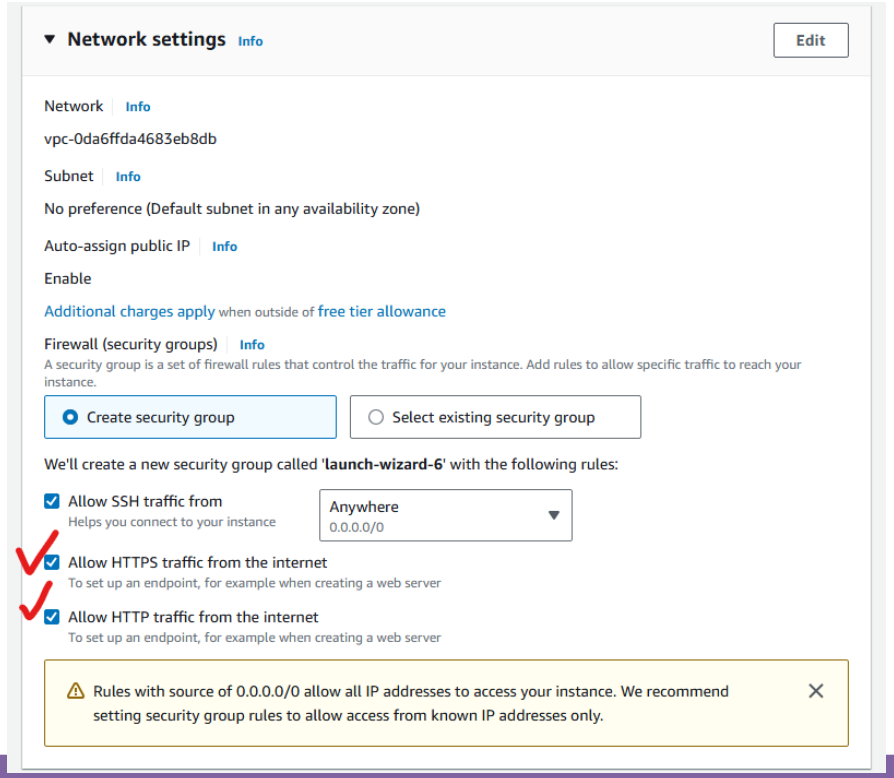
Use the t2.micro instance type

Deploying with AWS

You will need to generate a key pair (this gives you a way to log into your cloud machine remotely)

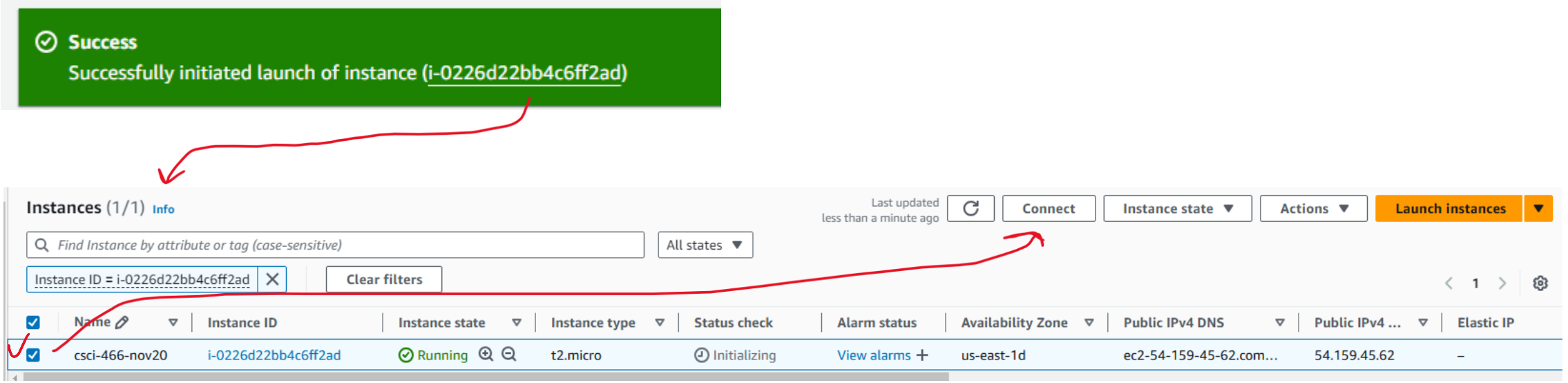


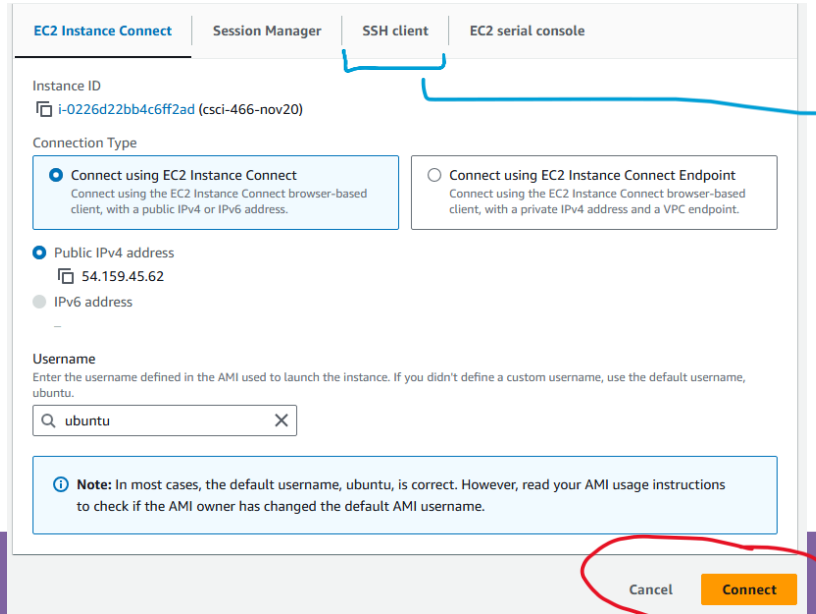




Enable port 80 and 443 to allow for HTTP requests

Deploying with AWS





If you want to connect to your machine with SSH, you can follow the instructions there

Setting up our server

aws | Services | Search [Alt+S]

```
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-1015-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Wed Nov 20 09:17:04 UTC 2024

  System load:  0.21              Processes:             108
  Usage of /:   21.1% of 7.57GB   Users logged in:       0
  Memory usage: 22%               IPv4 address for eth0: 172.31.23.249
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-23-249:~$
```

We are now connected to our cloud server !

Setting up our server

We need to run some commands to configure our machine:

// get python set up

```
sudo apt-get update
sudo apt-get install python3-venv
```

//create folder for our files and activate python

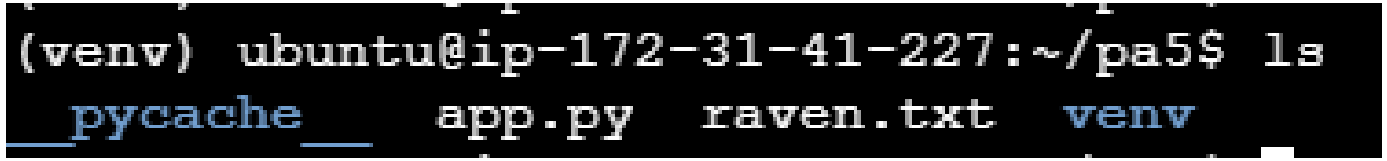
```
mkdir pa5
cd pa5
python3 -m venv venv
source venv/bin/activate
```

//install flask

```
pip install Flask
```

Setting up our server

Now, lets copy and paste over out `app.py` and our `raven.txt`



Gunicorn

We will use gunicorn to manage our Flask application and forward requests to it

```
pip install gunicorn
```

We want to make sure gunicorn is always running, so we will make a background process for it

```
sudo nano /etc/systemd/system/pa5.service
```

```
[Unit]
Description=Gunicorn instance for a simple api
After=network.target
[Service]
User=ubuntu
Group=www-data
WorkingDirectory=/home/ubuntu/pa5
ExecStart=/home/ubuntu/pa5/venv/bin/gunicorn -b localhost:8000 app:app
Restart=always
[Install]
WantedBy=multi-user.target
```

This file will make sure that on startup, it run the command

```
gunicorn localhost:8000 app:app
```

```
sudo systemctl daemon-reload
sudo systemctl start pa5
sudo systemctl enable pa5
```

Start the process

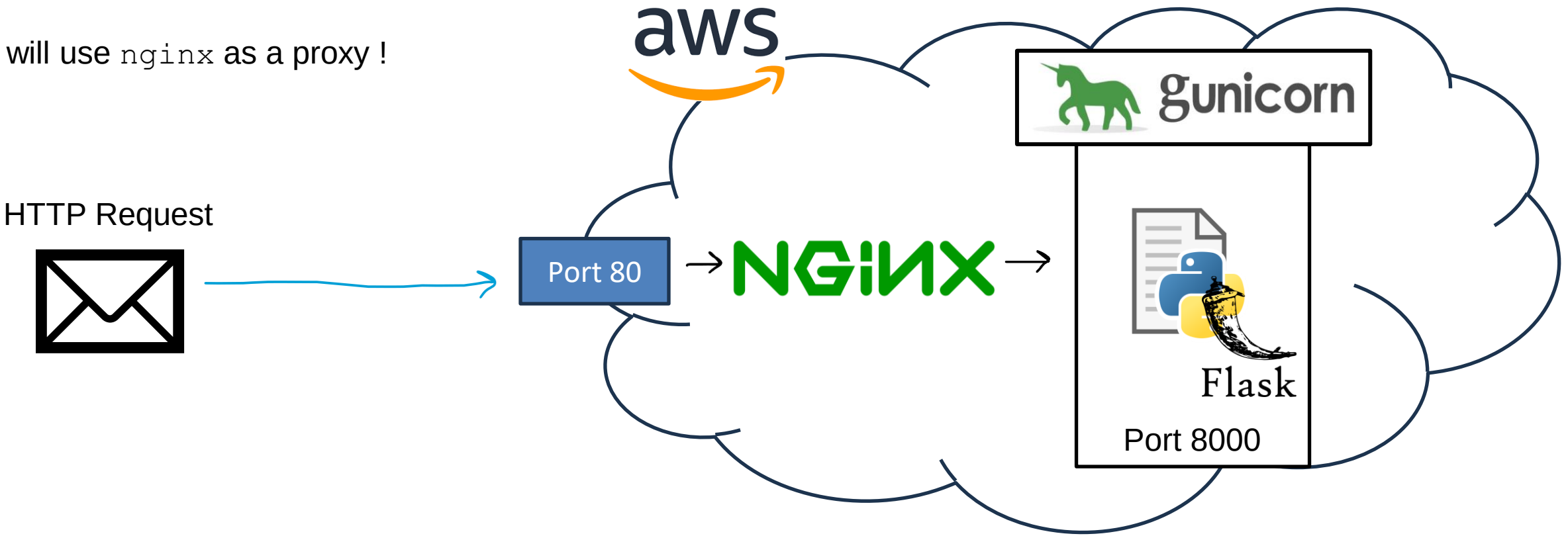
Our Flask API is running on port 8000 !

Nginx

Our app is listening on port 8000.

We want to take requests on port 80 (port for HTTP requests) and redirect them to gunicorn, which will redirect them to our Flask app!

We will use `nginx` as a proxy !



Nginx

Our app is listening on port 8000.

We want to take requests on port 80 (port for HTTP requests) and redirect them to gunicorn, which will redirect them to our Flask app!

We will use `nginx` as a proxy !

```
sudo apt-get install nginx

sudo systemctl enable nginx
```

Then we need to configure nginx to reroute traffic

```
sudo nano /etc/nginx/sites-available/default
```

Nginx

`sudo nano /etc/nginx/sites-available/default`

```
#
# In most cases, administrators will remove this file from sites-enabled/ and
# leave it as reference inside of sites-available where it will continue to be
# updated by the nginx packaging team.
#
# This file will automatically load configuration files provided by other
# applications, such as Drupal or Wordpress. These applications will be made
# available underneath a path with that package name, such as /drupal8.
#
# Please see /usr/share/doc/nginx-doc/examples/ for more detailed examples.
##

# Default server configuration


upstream flaskapp {
        server 127.0.0.1:8000;
}

server {
        listen 80 default_server;
        listen [::]:80 default_server;

        # SSL configuration
```

```
        # Add index.php to the list if you are using PHP
        index index.html index.htm index.nginx-debian.html;

        server_name _;

        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                proxy_pass http://flaskapp;
        }
```

Listen on port 80

Route requests to this function

Forward to port 8000

`sudo systemctl restart nginx`

Testing

We can now send curl requests to the public API of our cloud machine!

```
reese@Reese-PC MINGW64 ~
$ curl 34.238.139.130
Hello World!
reese@Reese-PC MINGW64 ~
$ curl --header "name: reese" 34.238.139.130/raven
Hello there, reese

Once upon a midnight dreary, while I pondered, weak and we
ary,
Over many a quaint and curious volume of forgotten lore-
```

Our cloud application is now online 24/7, and can accept requests!

PA5