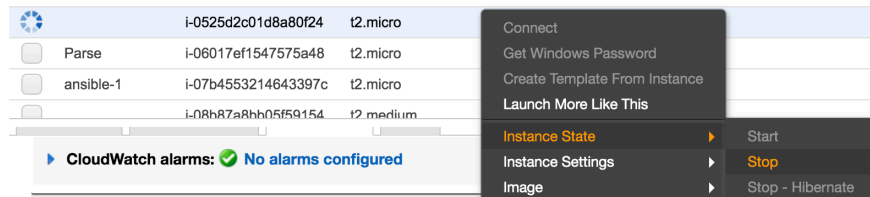


This walkthrough is created so that groups who are attempting to run very large datasets on their own local MacBooks can experiment with running their Python programs on the cloud. **Please make sure to shut down your instance after you are done.**

You can shut down your instance by **right clicking it in the EC2 console**, hovering over **Instance State**, and clicking on Stop:



Creating Your Server

1. After you have created your AWS account, log in:

Root user sign in

Email

ychnenay@gmail.com

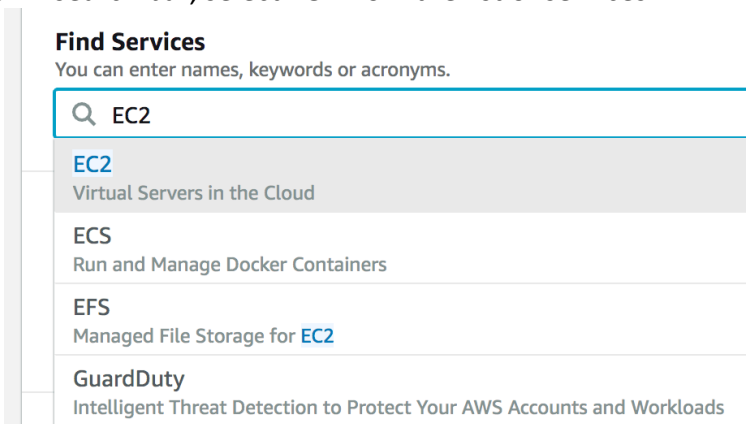
Password

Sign In

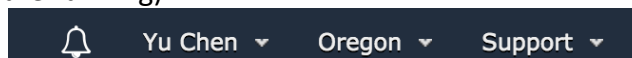
[Sign in to a different account](#)

[Forgot your password?](#)

2. Then, from the dropdown search bar, select **EC2** from the list of services:



3. Take a note of what **region** you are in. If you switch to a different region, you will not see the same instances (servers that are running):



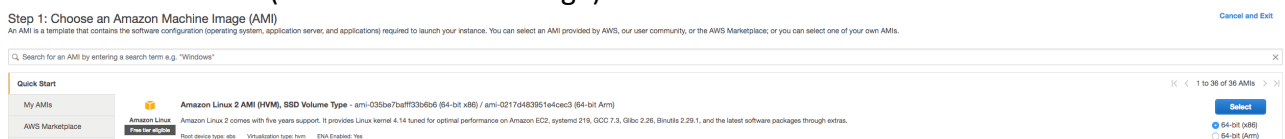
4. Click on **Launch Instance**:

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

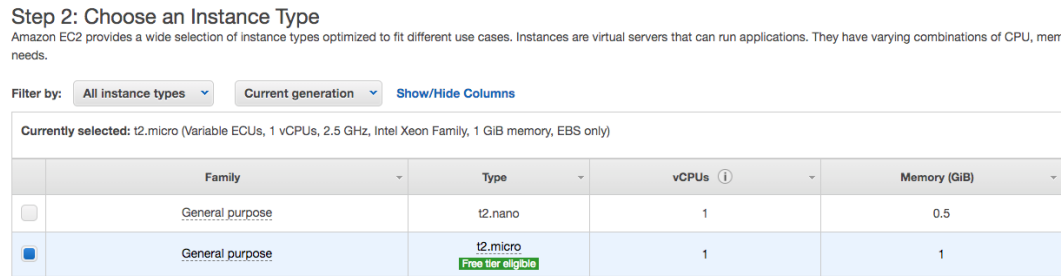
Launch Instance

5. Select the default AMI (Amazon Machine Image):



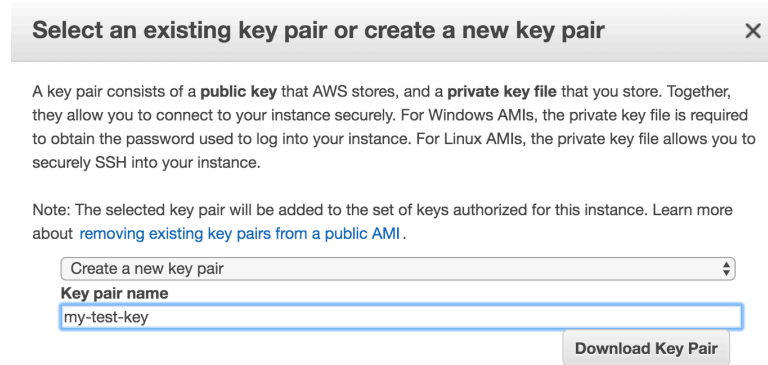
- Pick whatever instance type (size of the server) you need. Keep in mind that a standard MacBook Pro is around **8GB memory**, so you may want to pick a larger instance type. However, **please make sure to shut off your instance when you are done, or you will rack up a very, very large bill if you forget.**

In this example, I am just going to pick the small **t2.micro** (probably too small to do actual machine learning on) just to avoid paying anything:



After you have picked your instance type, click **Review and Launch**.

- In the next screen, click the blue Launch button. The following popup will appear:



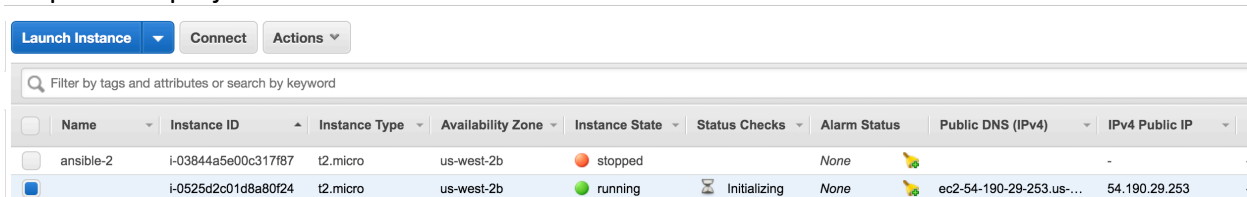
Create a new key pair. This key pair is what will be used to SSH into your instance (essentially log in). Do not lose this key pair, or give it to anyone via an unencrypted channel (ie. email). Then **Download Key Pair**.

Then click **Launch Instances**. This means that AWS is creating a server for you in which region of the world you selected (in my case, Oregon).

Logging Into Your Server

Note: if you are using Windows, the following instructions will not work for you. It is similar, but you'll have to install a tool called Putty. Please watch [this video for a good walkthrough](#).

- Click **View Instances**, and you'll be taken to a screen that lists all the server instances you have running. In my screenshot, I'll likely have a few more instances listed than you since I use AWS fairly regularly for personal projects:



Find your instance, and take note of its **IPv4 Public IP address**. This is the “address” on the internet you will go to find your server. Right click on the instance, and click **Connect**. The following pop up will appear:

Connect To Your Instance

I would like to connect with

☒ A standalone SSH client ⓘ
 ☐ A Java SSH Client directly from my browser (Java required) ⓘ

To access your instance:

1. Open an SSH client. (find out how to [connect using PuTTY](#))
2. Locate your private key file (my-test-key.pem). The wizard automatically detects the key you used to launch the instance.
3. Your key must not be publicly viewable for SSH to work. Use this command if needed:


```
chmod 400 my-test-key.pem
```
4. Connect to your instance using its Public DNS:


```
ec2-54-190-29-253.us-west-2.compute.amazonaws.com
```

Example:

```
ssh -i "my-test-key.pem" ec2-user@ec2-54-190-29-253.us-west-2.compute.amazonaws.com
```

Please note that in most cases the username above will be correct, however please ensure that you read your AML usage instructions to ensure that the AML owner has not changed the default AML username.

If you need any assistance connecting to your instance, please see our [connection documentation](#).

Close

9. In your terminal, navigate to the folder where your **key pair .pem** file is stored (likely in your Downloads folder). Then, issue the following command: **chmod 400 my-test-key.pem**

```

→ Downloads chmod 400 my-test-key.pem
→ Downloads

```

This is a safety precaution by AWS – it will refuse to allow connections from key pairs that allow other users besides yourself read/write/execute this key pair file. This is why you need to change its permissions (using **chmod**).

Then, issue the command right underneath in the popup window to connect:

```

→ Downloads ssh -i "my-test-key.pem" ec2-user@ec2-54-190-29-253.us-west-2.compute.amazonaws.com
The authenticity of host 'ec2-54-190-29-253.us-west-2.compute.amazonaws.com (54.190.29.253)' can't be
established.
ECDSA key fingerprint is SHA256:BDAMW0v8YBIFeoiYr0guXCFxfxIxnLLovmLXgECtsQU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-54-190-29-253.us-west-2.compute.amazonaws.com,54.190.29.253' (ECDSA)
to the list of known hosts.

  _I_  _I_  )
 _I_ (    /  Amazon Linux 2 AMI
  _I_\_I_

https://aws.amazon.com/amazon-linux-2/
9 package(s) needed for security, out of 14 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-21-84 ~]$

```

When you are asked **Are you sure you want to continue connecting**, type **yes**.

Congrats! You're logged into your EC2 instance!

Setting Up Your Server

10. Amazon's EC2 instances, by default, come with Python 2.7, but not Python 3. We'll need to install Python 3 ourselves. You can do this by using the following command: **sudo yum -y update && sudo yum install -y python3**

After about a minute or two, you'll see something like this:

```
Total 30 MB/s | 12 MB 00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : python3-setuptools-38.4.0-3.amzn2.0.6.noarch 1/4
Installing : python3-libs-3.7.2-4.amzn2.0.1.x86_64 2/4
Installing : python3-pip-9.0.3-1.amzn2.0.1.noarch 3/4
Installing : python3-3.7.2-4.amzn2.0.1.x86_64 4/4
Verifying : python3-3.7.2-4.amzn2.0.1.x86_64 1/4
Verifying : python3-libs-3.7.2-4.amzn2.0.1.x86_64 2/4
Verifying : python3-setuptools-38.4.0-3.amzn2.0.6.noarch 3/4
Verifying : python3-pip-9.0.3-1.amzn2.0.1.noarch 4/4

Installed:
python3.x86_64 0:3.7.2-4.amzn2.0.1

Dependency Installed:
python3-libs.x86_64 0:3.7.2-4.amzn2.0.1 python3-pip.noarch 0:9.0.3-1.amzn2.0.1
python3-setuptools.noarch 0:38.4.0-3.amzn2.0.6

Complete!
[ec2-user@ip-172-31-21-84 ~]$
```

To verify that Python 3 is correctly installed, type **python3** in the command line, and you should enter Python's interactive prompt:

```
[ec2-user@ip-172-31-21-84 ~]$ python3
Python 3.7.2 (default, Feb 26 2019, 20:08:16)
[GCC 7.3.1 20180303 (Red Hat 7.3.1-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type **exit()** to exit out of Python 3.

Copying Files To Your Instance

After you've set up Python, you'll need to now copy over your data and files (your **.py Python files** and **.csv data files**, for instance). You can do so by using **scp**. Let's say I want to copy over a file called **bbc-text.csv** located in my Downloads folder. I can do by using

```
scp -i my-test-key.pem bbc-text.csv ec2-user@ec2-54-190-29-253.us-west-2.compute.amazonaws.com:~/
```

However, your command will be slightly different, depending on what file you want to copy and the address of your server.

The diagram illustrates the components of the `scp` command used to copy a file to an AWS EC2 instance. The command is shown in a terminal window: `Downloads scp -i my-test-key.pem bbc-text.csv ec2-user@ec2-54-190-29-253.us-west-2.compute.amazonaws.com:~/`. Annotations with arrows point to specific parts of the command:

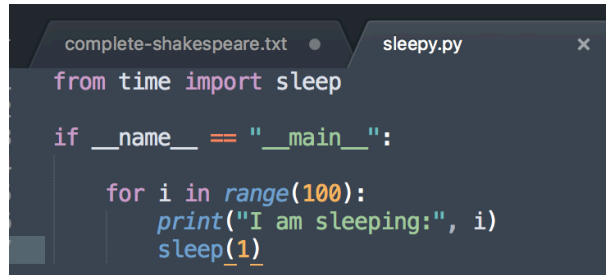
- Command to copy files over:** Points to the entire `scp` command.
- Use your .pem key to authenticate your identity:** Points to `-i my-test-key.pem`.
- File to copy over:** Points to `bbc-text.csv`.
- User name and address of the server (same as when you logged in using SSH):** Points to `ec2-user@ec2-54-190-29-253.us-west-2.compute.amazonaws.com`.
- Path on the server to copy to (the ~ means copy it to the ec2-user's home directory):** Points to `~/`.

Below the terminal window, a screenshot of the "Connect To Your Instance" dialog box is shown. It provides instructions on how to access the instance, including the command: `ssh -i my-test-key.pem ec2-user@ec2-54-190-29-253.us-west-2.compute.amazonaws.com`. The `my-test-key.pem` and the server address are highlighted with red boxes, corresponding to the annotations in the terminal window above.

Running A Python Program

Note: it's not recommended to use Jupyter notebooks on an EC2 instance. You should run your programs as Python scripts. Simply copy over all of your code into a **.py** file (including the import statements) and save it. You should be able to execute the same just by typing **python3 name_of_your_file.py**.

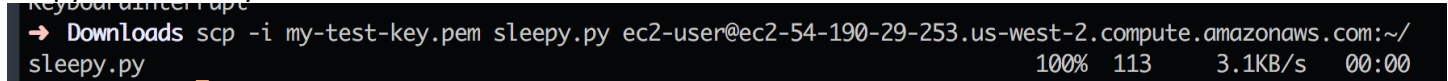
I am going to copy over a simple Python program called **sleepy.py**, which just prints out messages each second for 100 seconds:

A screenshot of a code editor with two tabs: 'complete-shakespeare.txt' and 'sleepy.py'. The 'sleepy.py' tab is active, showing the following Python code:

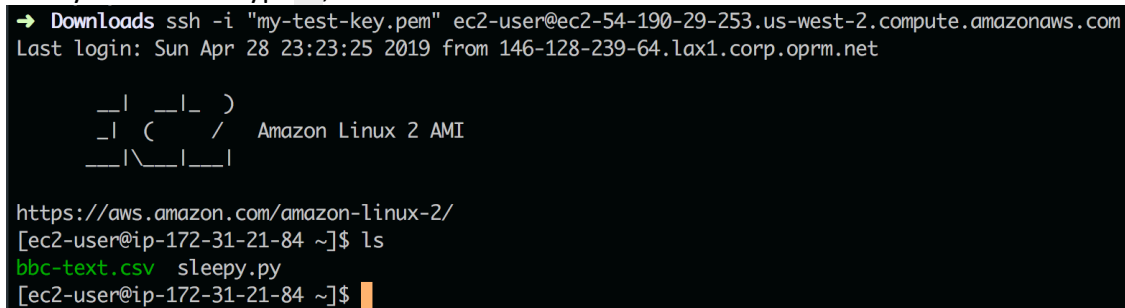
```
from time import sleep

if __name__ == "__main__":
    for i in range(100):
        print("I am sleeping:", i)
        sleep(1)
```

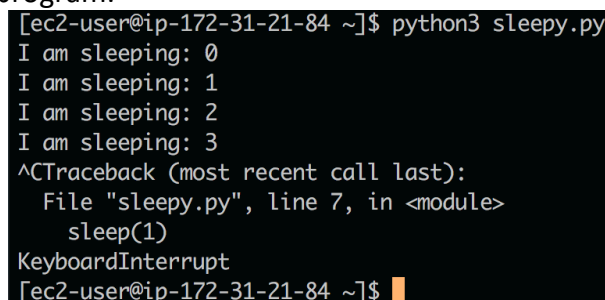
I copy it over to my server:

A terminal window showing a successful scp command. The prompt is 'Downloads'. The command is 'scp -i my-test-key.pem sleepy.py ec2-user@ec2-54-190-29-253.us-west-2.compute.amazonaws.com:~/sleepy.py'. The output shows '100% 113 3.1KB/s 00:00'.

Then I log into my server and type **ls**, which will list all the contents in that folder:

A terminal window showing an ssh login session. The prompt is 'Downloads'. The command is 'ssh -i "my-test-key.pem" ec2-user@ec2-54-190-29-253.us-west-2.compute.amazonaws.com'. The output shows the login banner for Amazon Linux 2 AMI, the URL 'https://aws.amazon.com/amazon-linux-2/', and the prompt '[ec2-user@ip-172-31-21-84 ~]\$'. The command 'ls' is entered, and the output shows 'bbc-text.csv sleepy.py'.

Notice the **bbc-text.csv** and **sleepy.py** files, both copied over from my local MacBook. I can run my **sleepy.py** program like any other Python program:

A terminal window showing the execution of the 'sleepy.py' program. The prompt is '[ec2-user@ip-172-31-21-84 ~]\$'. The command is 'python3 sleepy.py'. The output shows four lines of 'I am sleeping: 0', 'I am sleeping: 1', 'I am sleeping: 2', and 'I am sleeping: 3'. Then a '^C' (Ctrl+C) is pressed, followed by a traceback message: 'Traceback (most recent call last): File "sleepy.py", line 7, in <module> sleep(1) KeyboardInterrupt'. The prompt returns to '[ec2-user@ip-172-31-21-84 ~]\$'.

I ran the program using **python3 sleepy.py** (and then pressed CTRL + C to keep the process after about 4 seconds).

Note: you'll need to install the packages we've used in class like scikit-learn, pandas, etc. if your program uses those: **pip3 install -user scikit-learn pandas spacy genism**.

This can take a bit of time, depending on how large the packages you are installing – do not try installing everything on a **t2.micro** small instance- it can take quite a while to finish.

Running A Long-Running Python Program

Sometimes, the programs you want to run can be extremely long, or take hours. It's a pain to keep your computer open the entire time (ie. if your terminal window closes, the SSH connection is broken, and your Python program will hang up.)

A very useful command is **nohup**, which stands for no-hangup. This will make your Python program run in the background, regardless of whether you exit the shell, disconnect from SSH and reconnect later, etc.

You can use the command **nohup python3 -u name_of_file_to_run.py &**. This will run **name_of_file_to_run.py** in the background, allowing you to go to sleep, do other things, etc. and come back and check the status later:

```
[ec2-user@ip-172-31-21-84 ~]$ nohup python3 -u sleepy.py &
[2] 3343
[1] Done
[ec2-user@ip-172-31-21-84 ~]$ nohup: ignoring input and appending output to 'nohup.out'
^C
```

As the output states, it is no longer taking any commands from input, and any output is being written to a file called **nohup.out**. You can see that I immediately pressed **CTRL + C**, and exited out. You may even log out of your instance if you wish. The Python program will still run (as long as it doesn't encounter an error). You can see what **nohup.out** looks like by typing **cat nohup.out** to check on the progress of your program:

```
[ec2-user@ip-172-31-21-84 ~]$ cat nohup.out
I am sleeping: 0
I am sleeping: 1
I am sleeping: 2
I am sleeping: 3
I am sleeping: 4
I am sleeping: 5
I am sleeping: 6
I am sleeping: 7
I am sleeping: 8
I am sleeping: 9
I am sleeping: 10
I am sleeping: 11
I am sleeping: 12
I am sleeping: 13
I am sleeping: 14
I am sleeping: 15
I am sleeping: 16
I am sleeping: 17
I am sleeping: 18
I am sleeping: 19
I am sleeping: 20
I am sleeping: 21
I am sleeping: 22
I am sleeping: 23
I am sleeping: 24
I am sleeping: 25
I am sleeping: 26
I am sleeping: 27
```

If you really need to stop your program from running, you can do so by typing **kill -9 3343** (that was the process ID outputted when I used **nohup**, to let me know what process ID the Python program is running as in the background).