AWS EC2 Walk-through

Austin Privett, 8/23/18

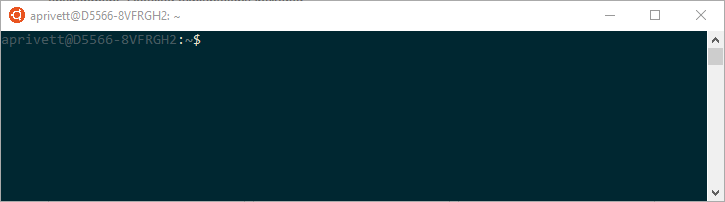
# Intended Audience

I’m writing this file as a how-to use Bhanu’s scripts to get the AWS EC2 instances going but giving enough instruction that people unfamiliar with the Linux command line can successfully follow (or at least be given the terms that are a good enough starting place for Googling). Note that if you open this Word Document offline, you can click on the triangles to the left of the section headers to open/close sections as needed.

# Pre-Requisites

## (Bash) Terminal

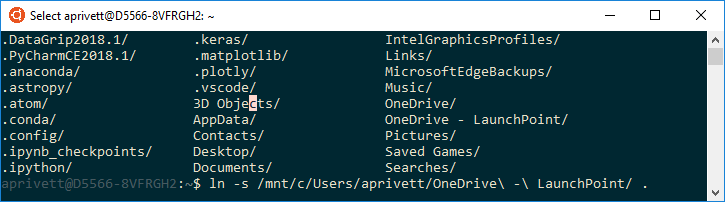
1. If you don’t already have the AWS files from Bhanu, [here](https://launchpointcorporation-my.sharepoint.com/:f:/g/personal/aprivett_discoveryhealthpartners_com/EgSCvhwhzVBNsxnEnaNb2r4BtFYs6w3cfjYGVeJXa-_0yA?e=yvrzxG) they are. (TODO: Get version-controlled repo going)
2. You’ll need some sort of \*nix-family console (at least the bash shell). Bhanu has used and tested [gitbash](https://gitforwindows.org/) (provided with Git for Windows), and I have used and tested the [Windows Subsystem for Linux](https://docs.microsoft.com/en-us/windows/wsl/install-win10) (WSL, requires administrative privileges on Windows 10 to unlock, I suggest the Ubuntu version). Below, I’ll call your choice here the “bash terminal”. I suggest you use the same username for this as you do on the EC2 instance to reduce typing.
3. Open this terminal. It will start in the “home directory”.



1. If you are new to the Linux command line, the following commands will be useful. Google them if you are unfamiliar: cd ls mv rm rmdir cp. Also, note that tab-completion will be a great asset in the terminal. I prefer zsh over bash to help with tab-completion.
2. If you use WSL, note that it creates a new (not virtual) subsystem that can access the Windows partition. The path to your Windows home directory is probably `/mnt/c/Users/<username>/`. You may want to link this (or some folder in it) to your Linux home directory via the terminal commands

cd # changes to your Linux home directory

ln -s /mnt/c/Users/<username>/ . # creates a symbolic link



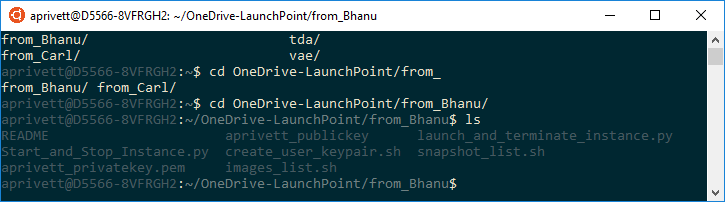
Above, you’ll see that I placed the OneDrive folder in my Linux home directory. Note that spaces are important, so you’ll be wise to not use them in your file/directory names (although Microsoft systems use them regularly).

## Python

1. Both of those should come with some sort of basic Python distribution, which you’ll need for the command-line tools.

# AWS Authentication

1. Open your Bash Terminal. Go to the location of the files from Bhanu. Note that he has wisely configured the scripts to give you the command-line options if you simply run the python scripts with no arguments (e.g., you may type `python launch\_and\_terminate\_instance.py` to see its options). The same goes for all python scripts in the directory.



1. Note that I’ve created a much briefer README file that contains the funny things I found when working through the steps. You can refer there for the short version of this file. Stay in that directory unless otherwise noted.
2. Install some non-standard Python packages required for running these scripts. Run

pip install boto3 docopt

1. We’ll soon begin running the bash and python scripts. Note that if you ever have some error that looks related to a new line, try the command

dos2unix <filename>

You may even want to apply this to all files in the directory via the bash wildcard

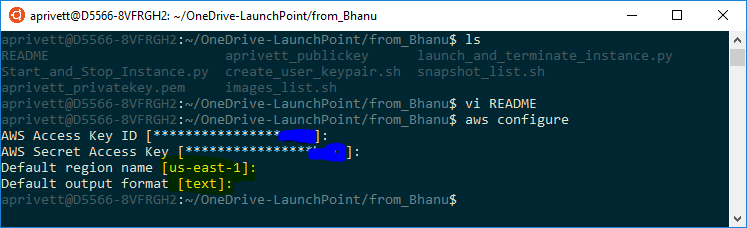
dos2unix ./\*.py ./\*.sh

1. Create your user keypair for AWS Authentication

bash create\_user\_keypair.sh <name>

1. This creates your private and public keys. Verify with `ls`.
2. Configure [AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html). When doing this, you’ll need a key pair as directed at the prior link (different from you public and private key just created) for your IAM user. Below you can see

aws configure



Note my selections highlighted in yellow above. Type/paste them in when prompted. This action creates some new files. Verify via `ls ~/.aws/`, and use, e.g., `cat` or `vi` to look at the contents of those files (e.g., `cat ~/.aws/config`); they should contain what you just entered.

# Instances

You’ll need to be familiar with the following AWS EC2 Instance terminology at the command line:

* Image: Bhanu has created these; they describe the software installed on some system (OS, python, etc.) and can exist on different hardware/instances.
* Instance: Specified hardware, attached to a specified Image. You pick the hardware you want. You’ll probably use [On-Demand Pricing](https://aws.amazon.com/ec2/pricing/on-demand/), so see that list of choices.
* Snapshot: A snapshot of the entire Instance, stored in Amazon S3. Stores both what is on the Instance disk *and* in RAM.
* Launch Instance: This should be done first and is akin to creating a machine with some software pre-installed and turning it ON.
* Terminate Instance: This should be done when you no longer need the machine. It kills and deletes the instance. You may save a snapshot first so that it can be restarted in the future.
* Start Instance: This is like WAKING the machine from sleep mode.
* Stop Instance: This is like putting a machine to SLEEP mode.

Better yet, read [Amazon’s documentation](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-lifecycle.html), partially pasted below:


        The instance lifecycle
      

|  |  |  |
| --- | --- | --- |
| **Instance state** | **Description** | **Instance usage billing** |
| pending | The instance is preparing to enter the running state. An instance enters the pending state when it launches for the first time, or when it is restarted after being in the stopped state. | Not billed |
| running | The instance is running and ready for use. | Billed |
| stopping | The instance is preparing to be stopped. | Not billed |
| stopped | The instance is shut down and cannot be used. The instance can be restarted at any time. | Not billed |
| shutting-down | The instance is preparing to be terminated. | Not billed |
| terminated | The instance has been permanently deleted and cannot be restarted. | Not billed |

# Rules of Thumb for Efficient Utilization of EC2 Resources

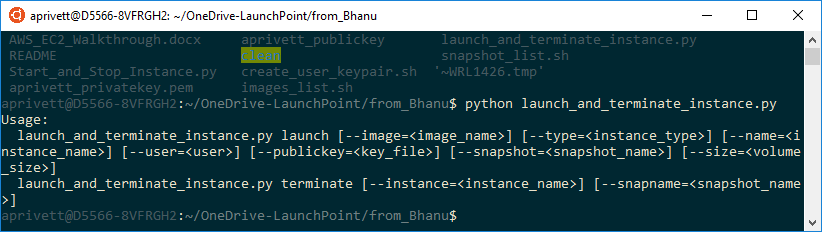
1. Choose an instance that is not larger than you need. [It IS possible to increase the size of your instance after you have created and used it.](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-resize.html) In general, you’ll need to stop the old instance prior to attaching to the new hardware.
2. Stop the instance when you are done computing (the only cost from the stopped instance is the storage cost, not the compute cost).

# Rules of Thumb for Naming

1. I suggest using the following naming convention (unless this is automated in the future):
   1. Prepend all Images, Snapshots, and Instances (anything you save) with a consistent user-identifier.
   2. Mine will be “aprivett.“, (dibs!). I suggest something unique and short, so that there is still ample space for explanatory text in the CLI table.
   3. For instance, I will probably name an instance: “aprivett.claimsopt” for claims optimization.
2. This accomplishes 2 goals:
   1. We can see who is running, on what hardware.
   2. Simplifies use of grep at command line if you are using AWS CLI (or some of Bhanu’s one-liners he has nicely shared using the AWS CLI).

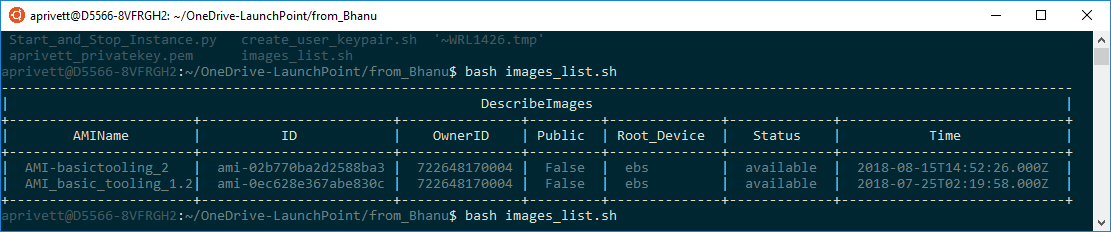
# Starting and Running your Command-Line AWS EC2 Instance

1. Now back to the step-by-step, after you have chosen the best EC2 instance type. We’ll choose the affordable t3.micro hardware for now (dual-core, 1 GB RAM, $0.0104 / hour = $0.25 / day).
2. Now, I know I need to launch an instance, but I have forgotten the commands. So I type

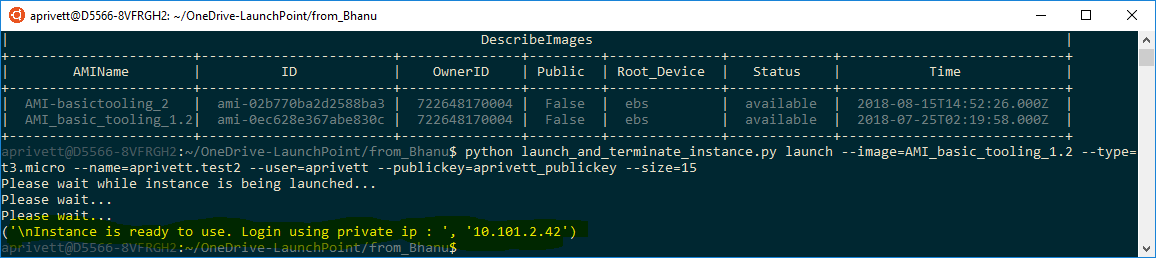
python launch\_and\_terminate\_instance.py

Note that this automatically tells me all the command-line arguments I can use with the “launch” command.

1. I now realize I need the following information:
   1. Image name (need to look up)
   2. Instance type (we’ll use t3.micro)
   3. Name (you can choose it)
      1. I suggest a shared best practice of prepending your name to the beginning of this chosen name, since all user’s instances will be viewable by others. For example, I’m naming mine “aprivett.test2”. This also makes it easy to grep for your instances via the AWS CLI.
   4. User (you can choose it)
   5. Public key (should be present in this directory already after the steps in “AWS Authentication”. Do not give the private key)
   6. Snapshot. Use this if you are launching the instance from a prior snapshot (we are not).
   7. Size (you can choose it, units are GB). This effects pricing; the cost is for EBS storage. It needs to be big enough to store all the OS + data you have on that node. You can calculate how much you will need based on the size of the data you import from the Redshift database (give instructions/code later). For the first try, let’s go with 15 GB.
2. At this point, the only remaining information we need is the image name, so let’s take a look. Run:

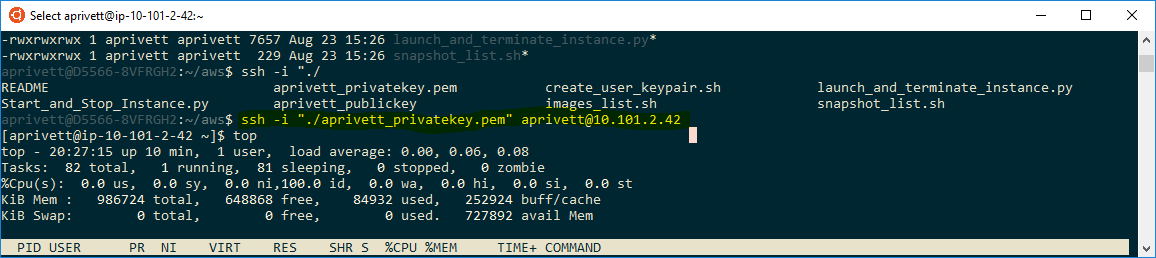
bash images\_list.sh

1. Choose ~~the more recent one, “AMI-basic\_tooling\_2”~~ “AMI\_basic\_tooling\_1.2” (the newer one didn’t work during my testing). We now have all the information we need to start an instance.

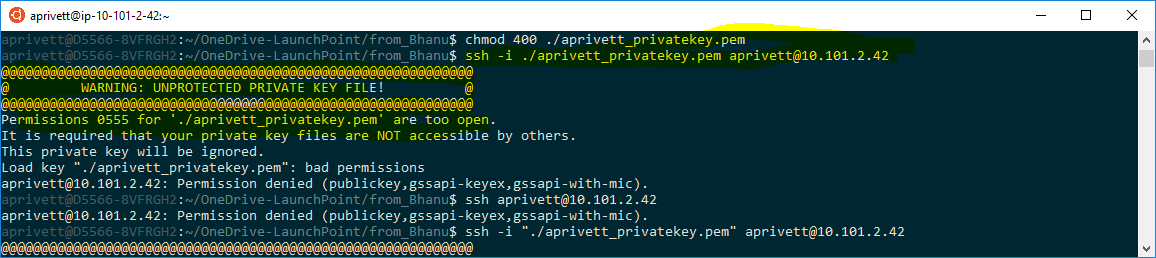


1. Once the instance is ready (can take 5-15 minutes), you may try to log in to it via the command:

ssh -i <privatekey.pem> <aws-user>@<given-ip-address>

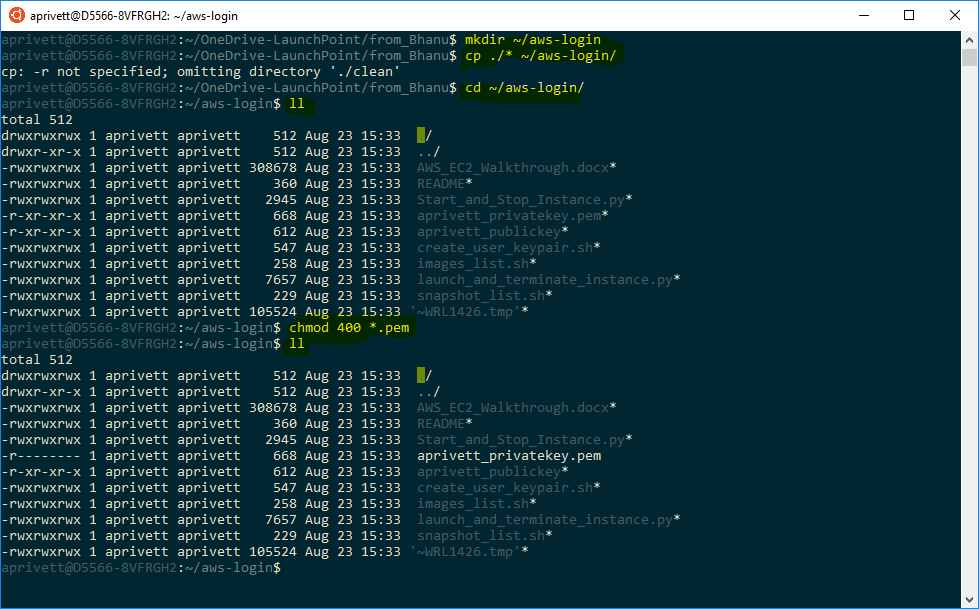


1. Congrats! You now are logged into an AWS EC2 instance! If you have any problems (it will take some time for the instance to initialize), make sure you are connected over the VPN and can log in to the AWS console.
2. *Troubleshooting Tip*: The first time I tried this, AWS told warned me about open permissions:



This happened because WSL gives open permissions to everything on the mounted Windows directory. To solve this, I copied the directory to my home Linux directory and then changed the permissions on the private key file with:

chmod 400 aprivett\_privatekey.pem



1. Also note that you can check on your instances via the AWS console (<https://console.aws.amazon.com>). You won’t be able to log on until the instance’s status is “running” as shown at the AWS console. You don’t need to go to the AWS console, just keep trying to log in until it works.

# Already-Installed Packages

1. Now that you have access to hardware with power anywhere from a small laptop to a personal supercomputer, you’ll want to run some packages. You may already run “python” and “R” from the command line.

# Installing New/Updating Packages (TODO: Decide what should be pre-installed on AMIs if not everything)

1. If installing some new package, note that any software that uses HTTP will NOT work (e.g., pip and conda), as the EC2 instances cannot see the internet. The ONLY allowed communication is over SSH (and thus SCP). So, you’ll need to use SCP to copy files/directories from your laptop to the instance via (for some folder in the cwd on your laptop):

scp -r <foldername-laptop> -i <privatekey> <aws-user>@<given-ip-address>/<foldername-ec2>

1. Until pip install works, use the following protocol to upload packages to AWS via scp.
2. Use `[pip download](https://pip.pypa.io/en/stable/reference/pip_download/)` to get your specified packages onto your local machine. This is easiest if the machine you are downloading the wheels files matches the EC2 instance. So, I used my local 64-bit Ubuntu with python 2 as the machine default to grab those corresponding wheels files.

pip download scikit-learn numpy scipy torch torchvision cvxpy plotly

1. Then, you can copy these to the cloud via scp. Ideally, we will later be able to use pip directly on the EC2 instances. This will be important for, e.g., Pytorch to take advantage of CUDA.
2. There may be some speedup available from the MKL-compiled versions from Anaconda, too.
3. In Production: Due to some package-compatibility problems I have noticed in the past, I would suggest that we **manage the package selection via the AMIs** rather than use the command-line scripts to do this installation, as versions can change and combinations should be tested prior to production use.

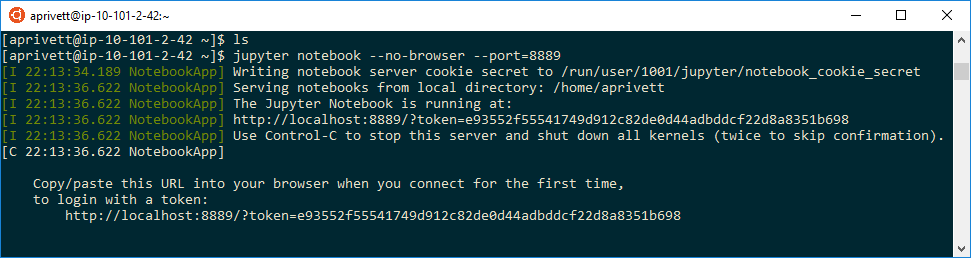
# Installing New Packages on Public Access Image

1. Use the file launch\_and\_terminate\_instance\_publicAccess.py (available from Austin or Bhanu) and follow the create instance instructions.
2. Then, install everything you need using pip, etc.
3. Ask Bhanu for the next step.

# Jupyter Notebook on EC2 Instance

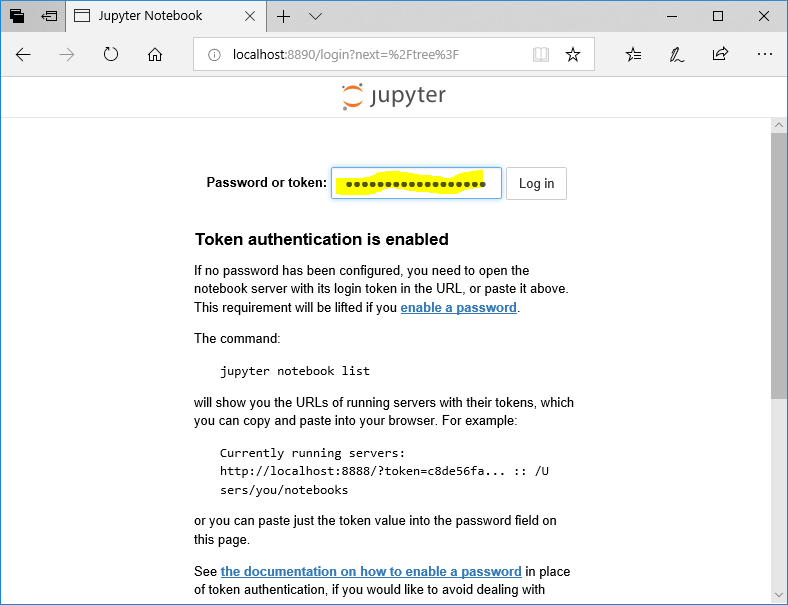
1. In addition to running scripts from your laptop command-line, you may want to open up a Jupyter notebook for rapid code iteration, data exploration, or visualization. Bhanu has opened up a single port (8889) for these notebooks. I’ll write a tailored explanation below, but [feel free to look here](https://amber-md.github.io/pytraj/latest/tutorials/remote_jupyter_notebook) to find the site that inspired this text.
2. Launch the Jupyter server on the instance:

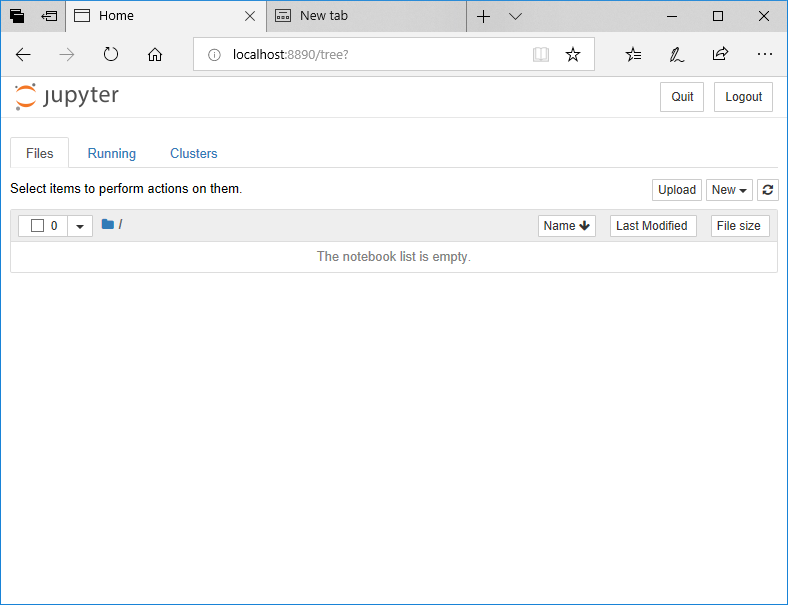
jupyter notebook --no-browser --port=8889



1. Note the password after “token=” above. On your local machine, connect to that Jupyter server via the following command, where the blue indicates the local machine and orange indicates the EC2 instance (the ports should accordingly match).

ssh -i <privatekey> -N -f -L localhost:8888:localhost:8889 <aws-user>@<aws-ip-address>

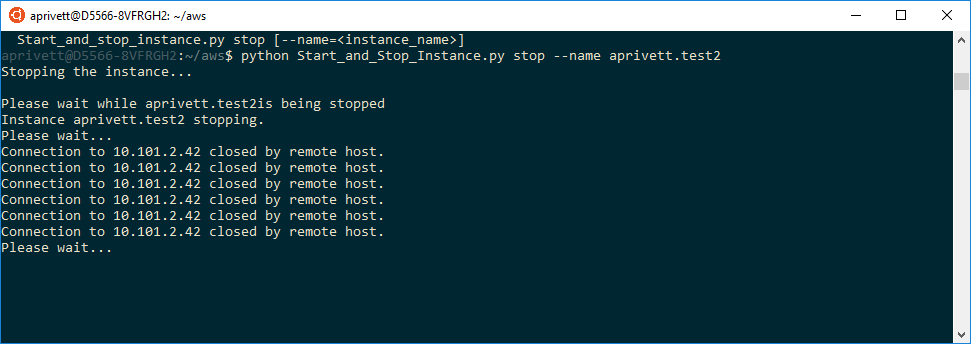
1. Open a browser to the local address (<http://localhost:8888>), and enter in your previously-copied token.
2. You’re in!



1. Ideally, this process will be almost completely automated within the near future.

# Time is Money

1. Don’t forget to stop your instance as shown below (don’t waste money on unused CPU cycles!) when finished working:



1. Note: In the future, once we have models that no longer need interactive development but only computation, I intend (unless somebody else wants to do it) to produce a script that manages starting an EC2 instance, submitting the job, and closing the instance when finished, so that this entire process is automated and resource use is optimized.

# Snapshots: Important!

1. While trying to get some EC2 instance running, I had one instance unrecoverably “break” (likely due to root changes that are only occurring during AWS system development but not later during production), so I strongly suggest snapshots of important work.
2. For this task, to see existing snapshots, use: `snapshot\_list.sh`. Then, to save a new snapshot, use `Start\_and\_Stop\_Instance.py` (NOTE: this doesn’t work yet) and give an intelligently-named snapshot name (e.g., prefix it with your username).
3. This action will save the data volume (user’s home directory), NOT what is installed on root. New installations on the root drive are saved as a new AMI (image).

# Terminate Instance

1. When you are ready to delete an instance, use `launch\_and\_terminate\_instance.py`. You can save a snapshot of your work in order to regenerate the machine. You can even launch the instance on different hardware.

# Grab Data from Redshift

1. Awaiting the enabling of pip install via http… (finished end of August).

# Future Possibilities (TODO)

1. Using the AWS CLI: I have successfully been using the [aws-shell](https://github.com/awslabs/aws-shell) to more quickly learn the AWS CLI, as it offers auto-complete otherwise not available..