Amazon Web Service (AWS) Elastic Compute Cloud (EC2)

An EC2 Instance is a virtual server for running applications on the Amazon Web Services infrastructure. AWS is a cloud computing platform, and an EC2 is a service that allows subscribers to run application programs in the computing environment.

This section of Recitation 0 will review the following;

- Creating and connecting an AWS account with the AWS Command Line Interface
- Getting started with a demo on an EC2 Instance

Part 1/2:
Creating and Connecting an AWS account with the AWS Command Line Interface

1. Set up an AWS account (create account)
    a. Provide your contact information
    b. Provide your credit card information
        i. FYI: Students are only charged in the case that they use up all of the coupons for payment provided through the class.
    c. Choose the Free / Basic account

2. Connect your AWS account with the Command Line Interface
    a. Install CLI version 2 for either Linux, macOS, or Windows (formal instructions)
        i. Confirm by the CLI version installed

```
$ which aws
/usr/local/bin/aws
$ aws --version
aws-cli/2.0.23 Python/3.7.4 Darwin/18.7.0 botocore/2.0.0
```

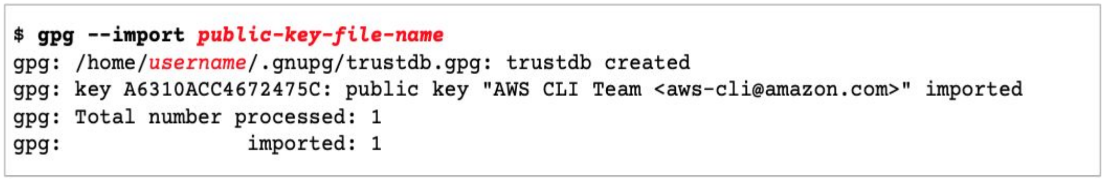
    b. Verify Authenticity (formal instructions)
        i. Download and install the gpg command (GnuPG website)
        ii. Create a text file and paste the following to create the public key file:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBF2Cr7UBEADJZHcgusOJl7ENSyumXh85z0TRV0xJorM2B/JL0kHOyigQluUG
ZMLhENaG0bYatdrKP+3H91lvK050pXwnO/R7fB/FSTouki4ciIx5OuLlnJZIxSzx
PqGl0mkxImLNbGWoi6Lto0LYxqHN2iQtzlwTVmq9733zd3XfcXrZ3+LblHAgEt5G
TfNxEKJ8soPLyWmwDH6HWCnjZ/aIQRBTIQ05uVeEoYxSh6wOai7ss/KveoSNBbYz
gbdzoqI2Y8cgH2nbfgp3DSasaLZEdCSsIsK1u05CinE7k2qZ7KgKAUIcT/cR/grk
C6VwsnDU0OUCideXcQ8WeHutqvgZH1JgKDbznoIzeQHJD238GEu+eKhRHcz8/jeG
94zkcgJOz3KbZGYMiTh277Fvj9zzvZsbMBCedV1BTg3TqgvdX4bdkhf5cH+7NtWO
lrFj6UwAsGukBTAOxC0l/dnSmZhJ7Z1KmEWilro/gOrjtOxqRQutlIqG22TaqoPG
fYVN+en3Zwbt97kcgZDwqbuykNt64oZWc4XKCa3mprEGC3IbJTBFqglXmZ7l9ywG
EEUJYOlb2XrSuPWml39beWdKM8kzr1OjnlOm6+lpTRCBfo0wa9F8YZRhHPAkwKkX
XDeOGpWRj4ohOx0d2GWkyV5xyN14p2tQOCdOODmz80yUTgRpPVQUtOEhXQARAQAB
tCFBV1MgQ0xJIFRlYW0gPGF3cy1jbGlAYW1hem9uLmNvbT6JAlQEEwEIAD4WIQT7
```

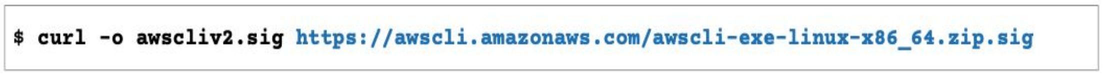
Xbd/1cEYuAURraimMQrMRnJHXAUCXYKvtQIbAwUJB4TOAAULCQgHAgYVCgkICwIE
FgIDAQIeAQIXgAAKCRCmMQrMRnJHXJIXEAChLUIkg80uPUkGjE3jejvQSA1aWuAM
yzy6fdpdlRUz6M6nmsUhOExjVIvibEJpzK5mhuSZ4lb0vJ2ZUPgCv4zs2nBd7BGJ
MxKiWgBReGvTdqZ0SzyYH4PYCJSE732x/Fw9hfnh1dMTXNcrQXzwOmmFNNegG0Ox
au+VnpcR5Kz3smiTrIwZbRudo1ijhCYPQ7t5CMp9kjC6bObvy1hSIg2xNbMAN/Do
ikebAl36uA6Y/Uczjj3GxZW4ZWeFirMidKbtqvUz2y0UFszobjiBSqZZHCreC34B
hw9bFNpuWC/0SrXgohdsc6vK50pDGdV5kM2qo9tMQ/izsAwTh/d/GzZv8H4lV9eO
tEis+EpR497PaxKKh9tJf0N6Q1YLRHof5xePZtOIlS3gfvsH5hXA3HJ9yIxb8T0H
QYmVr3aIUes20i6meI3fuV36VFupwfrTKaL7VXnsrK2fq5cRvyJLNzXucg0WAjPF
RrAGLzY7nP1xeg1a0aeP+pdsqjqlPJom8OCWc1+6DWbg0jsC74WoesAqgBItODMB
rsal1y/q+bPzpsnWjzHV8+1/EtZmSc8ZUGSJOPkfC7hObnfkl18h+1QtKTjZme4d
H17gsBJr+opwJw/Zio2LMjQBOqlm3K1A4zFTh7wBC7He6KPQea1p2XAMgtvATtNe
YLZATHZKTJyiqA==
=vYOk
-----END PGP PUBLIC KEY BLOCK-----

iii. Import the AWS CLI public key, substituting in the file name of the public key created in step ii.



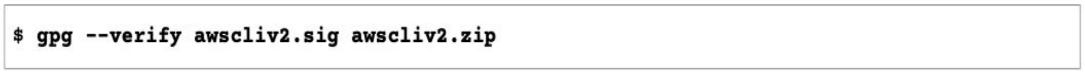
```
$ gpg --import public-key-file-name
gpg: /home/username/.gnupg/trustdb.gpg: trustdb created
gpg: key A6310ACC4672475C: public key "AWS CLI Team <aws-cli@amazon.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

iv. Download the AWS CLI signature for the package you downloaded, which has the extension ".sig"



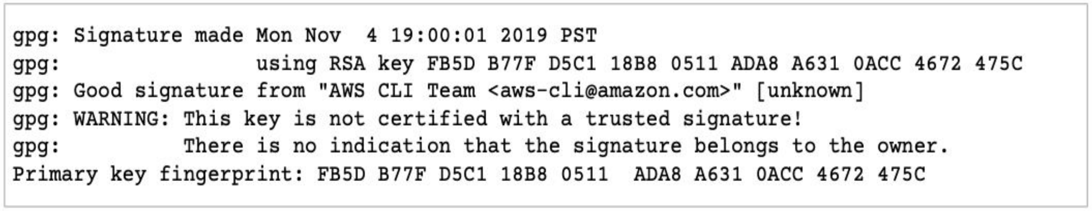
```
$ curl -o awscliv2.sig https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip.sig
```

v. Verify the signature by passing both the ".sig" and ".zip" file names as parameters



```
$ gpg --verify awscliv2.sig awscliv2.zip
```

vi. Check to ensure that the output looks like this:



```
gpg: Signature made Mon Nov  4 19:00:01 2019 PST
gpg:                using RSA key FB5D B77F D5C1 18B8 0511 ADA8 A631 0ACC 4672 475C
gpg: Good signature from "AWS CLI Team <aws-cli@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: FB5D B77F D5C1 18B8 0511  ADA8 A631 0ACC 4672 475C
```

c. Configure with Admin Privileges (formal instructions)
   i. Sign in to the IAM console as the "root user" with your AWS account
   ii. Go to "My Account" and scroll down to "IAM User and Role Access to Billing Information"
      1. `Click edit, check the box to "Activate IAM Access", update, and return to the IAM dashboard.
   iii. In the navigation pane, chose "Users", and then "Add User"

1. Name your user “Administrator”
2. Check the box to give “AWS Management Console access”
3.

Part 2/2:
Getting Started with a demo on an EC2 Instance

1. Launch an Instance
2. Shut Down an Instance
3. Switch from GPU to non-GPU

****

1)
    a) David’s idea on how to set up a volume (non-temp) to hook up with spot instances so as to save model files, in case a spot instance dies

****

Remote Environment Setting

- Jupyter notebook Remote environment setting.
- Pure python mode environment setting.
- Kaggle auto submission.

We provide two working environment settings. The first one is jupyter notebook based. And the second one is python source code based. We recommend using Jupyter Notebook Remote Server + AWS EC2 to do the homeworks because it’s easy to debug and interactively understand your code.

1) Jupyter notebook + AWS EC2.

a) Download the key pair of your EC2, `deeplearning.pem`.
b) Run the following code to set the password so that your local machine could access it using the password.

```
jupyter notebook password
```

c) On EC2, run the following code to open a jupyter notebook server listening at port 8889. If you close the terminal, then this server would be killed. (trick: You can add nohup to prefix this command line code to let it run in the background. So the server would not be terminated if you close the terminal.)

```
(nohup) jupyter notebook --no-browser --port=8889
```

d) On local machine, run the following command to connect 8888 port number on your machine to 8889 port on remote machine.

```
ssh -i deeplearning.pem -N -f -L localhost:8888:localhost:8889
                      ubuntu@ec2instance
```

e) Open your browser and type in “localhost:8888”, you can then input the password and access the remote jupyter server like you running a jupyter notebook on a local machine.

f) **Trick:** Sometimes you could close the terminal and run your model overnight. The second day you wake up and connect to the server. You will find that the output disappears. This happens because each terminal is a process and when you close it, it would be killed and the next time you connect to it, it’s a new process. The best way to keep track of your output is to using logging package in python.(https://docs.python.org/3/howto/logging-cookbook.html). You can add two handlers so that your output would be printed and also be written to the log file.

2) Pure python mode + AWS EC2

One problem of coding on a remote machine is that you cannot use user-friendly IDE or text edit like you are coding on your macbook. SFTP could help you. SFTP is a file transmission protocol that could automatically synchronize your code with the remote machine. We recommend that you use VScode or Pycharm because it provides such plugins.

a) If you are using VScode, you should install SFTP plugin first
   i) Type shift+cmd+p to see the pop-up window and choose “SFTP-config”.
   ii) In the configuration file, input the following code

```
{
   "name": "Anything You Like",
   "host": "ec2-instance-public-ip",
   "protocol": "sftp",
   "port": 22,
   "username": "ubuntu",
   "privateKeyPath": "deeplearning.pem",
   "remotePath": "/the-path-of-your-code-on-remote-machine",
   "uploadOnSave": true
}
```

   iii) Now you can edit your code on your local machine and it would automatically populate to the remote machine.

b) If you are using Pycharm.
   i) Preference -> Deployment -> Add, choose SFTP as the option. Configure it just like what you need to do if you are using VScode.

3) Kaggle

1) Download your kaggle.json file at Kaggle My Profile page.

2) Install Kaggle package

```
pip install kaggle
```

3) Copy your kaggle.json to .kaggle/ folder so that kaggle package could know who you are.

```
scp -i deep_learning.pem ./kaggle.json ubuntu@instance:~/.kaggle/
```

4) You could easily download the data using the following command. competition-name could be found on the 'Data' section in the competition main webpage.

```
kaggle competitions download -c competition-name
```

5) You can submit using the following code

```
kaggle competitions submit -c competition-name -f your-submission-path -m
"Message"
```