BDSP AWS EC2 onboarding documentation

Setting up the EC2 instance:

0. Setting up the email to Junior, and then Junior will send a password to you, you need to click on this link, and then enter your email, and the password, and then reset your password. https://broad-brain-data-science-project.signin.aws.amazon.com/console

- Note: **Select North Virginia** (AWS Region), and select instance (us-east-1) at the top, close to the your name (right corner)

1. Select the name with format: bdsp-dst-{custom_name}
2. To the right, select Tag
    a. Make a tag type 'Owner", and then the value is abc@somewhere, where this value must be IDENTICAL to your login email address from AWS. **If you don't do this, you will have problems controlling your instance.**
3. Select the machine: Ubuntu**. ** (see below for choosing a GPU set up Amazon image file instead)**
4. Select the type of machine: let's start with the free one and simple one
5. Select the instance type [different machine that you can select]: t2.micro, you can change it afterwards
6. Create key pair, pick the name, for example, in my case bdsp-dst-demo, select **RSA**, and **.pem**, and create the key pair, save it somewhere which is safe -
    a. Note: need to have it if you need to connect to VM
    b. *Note: it seems you must make a new one for each instance
7. Network setting: change VPC setting: change it to (**bdsp_rds_mysql)**
    a. Don't change subnet (**** note: if your subnet is not East 1a, then DO change your subnet, because otherwise you might not be able to access shared file resources!**)
    b. enable the auto assign public IP
    c. select existing security group, then select (launch-wizard-7 sg-...)
8. configure storage (SSD storage): the volume that your need (select 30GB if you don't know… though some image files will automatically required a higher number), then select gp3 (root volume)
9. [optional] In user data, enter some startup scripts
10. Launch

When you first launch the machine:

1. Connect to instance should now be possible. If you get error: reach out to Junior for privileges.
2. [Important] Tag your instance (**this should have automatically happened): click your instance → Actions → Instance settings → Manage tags:

Add a new tag with Key=Name, Value=<your instance name>
Then click Save
Otherwise you won't be able to start and stop your instance!

3. Move/save the .pem file to any location on your local computer.
4. Open terminal in that folder where the .pem file is.
5. Type chmod 400 [[name of key-pair file name]] <- in terminal; this changes the permissions of the file
    a. For windows:
    icacls "C:\path\to\your\file" /inheritance:r /T
    icacls "C:\path\to\your\file" /grant:r "%username%:R"
    (Here %username% is a system variable that should automatically fill with the name of the current user. The :R grants read-only access. replace "C:\path\to\your\file" with the actual path to your file.)

Type pwd (get path of directory)

6. Got to visual studio code
7. Go to the Extension tab and install "remote - SSH" extension.

8. Click on green '><' symbol on left-down corner
    a. "Connect to Host"
    b. Add this to the config file:

```
Host arbitrary_name
   HostName Your public IPv4 address (e.g. 3.87.68.0)
   User ubuntu
   IdentityFile path to your .pem file (e.g.
/home/username/aws/aws-bdsp-username.pem
```

    **** in some Instance images, the user is not ubuntu but instead ec2-user. Be sure to put the correct user that you want here. ****
    c. Click on green symbol again. Connect to the host configured above.
    d. Click "continue" to connect. (+for Windows: select Linux as host configuration)

9. type this in the remote terminal:

cd /
sudo mkdir bdsp
cd bdsp
sudo mkdir opendata
sudo mkdir staging

    a. got to top -> open the folder just created

sudo apt-get update
sudo apt-get install s3fs

    b. (this is the package to mount the s3 bucket)

10. File → Add Folder to Workspace → “/home/ubuntu”
11. Terminal /home/ubuntu:

cd /home/ubuntu/

echo 'AKIASV4LKNY27HIOMHMW:Ckjqqw5DBGtk47tyctNs0IrqgWvDYJzw5mkaWlsg' > .passwd-s3fs
echo 'AKIASV4LKNY2YEZJ6BDP:Kgd+1u4rostffLY0AaUzc0yeq9WlbhZrDOx4bXxu' > .passwd-s3fs-staging
sudo chmod 600 .passwd-s3fs
sudo chmod 600 .passwd-s3fs-staging

a. **NOTE: the next two steps need to be re-run if an instance was stopped and started again:**

sudo s3fs -o use_cache=/tmp/s3_cache -o allow_other bdsp-opendata-staging /bdsp/staging -o passwd_file=~/.passwd-s3fs-staging -o umask=000
sudo s3fs -o use_cache=/tmp/s3_cache -o allow_other bdsp-opendata-repository /bdsp/opendata -o passwd_file=~/.passwd-s3fs -o umask=222
→ you have now mounted /bdsp/staging and /bdsp/opendata

b. (Maybe necessary:
To see the folders in the visual studio code, click
File → Open Folder → “/bdsp/”

AWSKeys ← dropbox folder has important keys that allow us to connect to our storage areas
Copy those three files to the ubuntu “folder” in VS.

If you want to automate the setup of a virtual machine (e.g. for installing packages etc. when creating multiple virtual machines), you can use cloud-init to do so. You just need to enter the configuration data for cloud-init user data in the “User Data” textbox in the “Advanced” section. For example, here is an example of cloud-init user data that will set up the bdsp s3 bucket file system mounts (replacing steps 12 and 14 above) in a way that remounts the filesystem automatically whenever the machine restarts (so you don’t have to re-run 14(f) and 14(g) after every restart):

```
#cloud-config

# store the s3 key secrets
write_files:
 - path: /root/.passwd-s3fs
   permissions: 0o600
   content: AKIASV4LKNY27HIOMHMW:Ckjqqw5DBGtk47tyctNs0IrqgWvDYJzw5mkaWlsg
 - path: /root/.passwd-s3fs-staging
   permissions: 0o600
   content: AKIASV4LKNY2YEZJ6BDP:Kgd+1u4rostffLY0AaUzc0yeq9WlbhZrDOx4bXxu

# install the s3fs package
packages:
 - s3fs
package_update: true
```

```
# create the required directories for mounting the bdsp data
bootcmd:
 - mkdir -p /bdsp/opendata
 - mkdir -p /bdsp/staging

# mount the BDSP s3 buckets automatically on startup
mounts:
 - [
     "bdsp-opendata-repository",
     "/bdsp/opendata",
     "fuse.s3fs",
     "_netdev,allow_other,passwd_file=/root/.passwd-s3fs,umask=0222,nofail",
     "0",
     "0"
   ]
 - [
     "bdsp-opendata-staging",
     "/bdsp/staging",
     "fuse.s3fs",
     "_netdev,allow_other,passwd_file=/root/.passwd-s3fs-staging,umask=0000,nofail",
     "0",
     "0"
   ]

# reload the filesystem mounts now that s3fs is installed
runcmd:
 - mount -a
```

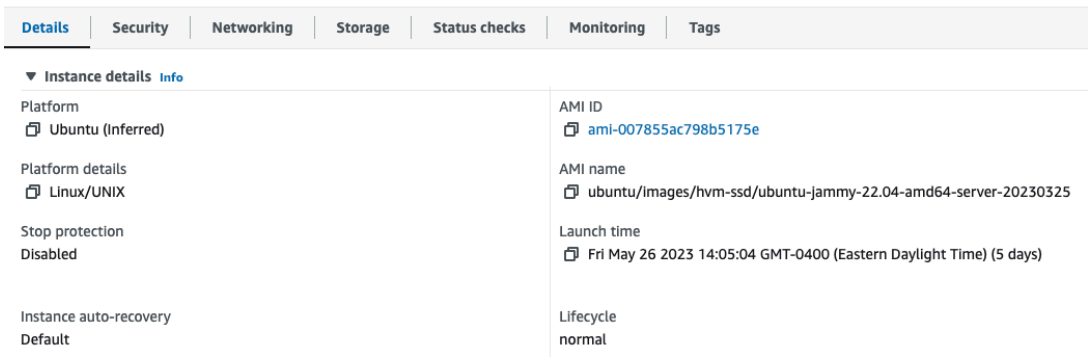
There are many other things you can do with cloud-init, such as creating users or installing packages, and running arbitrary commands at various stages in the boot process. For more details, see the documentation at https://cloudinit.readthedocs.io/en/latest/.

Add EC2 Serial Console - easier to debug booting up -- shows linux text as startup occurs

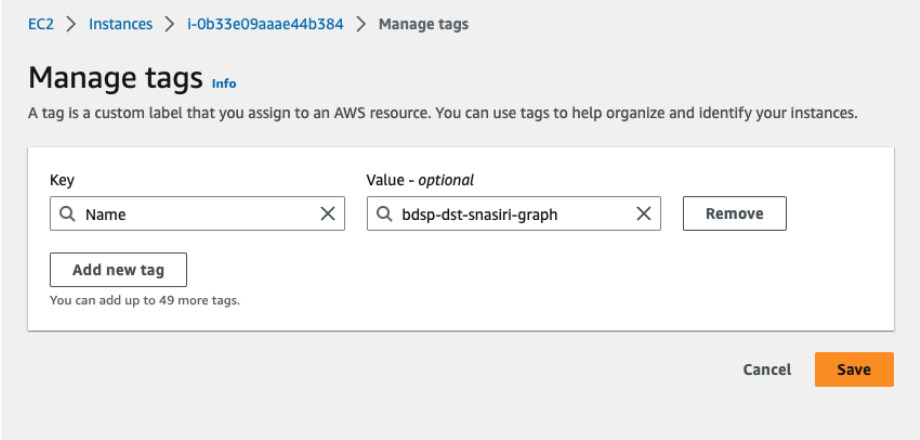
Add tag to your EC2 instance

You **need** to add a tag to your EC2 instance, it prevents people from terminating your EC2 instance accidently. **** also if not done, you will not have control over your own instances.**

Please click on your instance, then go to tag tab (last tab in the following figure)



Then click on manage tags, and then add new tag



Key: Owner (this is case-sensitive!)
Value: your_username (e.g.,: snasiri@mgh.harvard.edu)

Elastic IP address

If you stop your instance and want to restart it, the Public IPv4 address changes every time. As a result, you need to update your configuration file (~/.ssh/config) with the new address. To avoid this hassle, you can create an Elastic IP address and allocate it to your instance. After each stop and restart, you won't need to change the Public IPv4 address anymore.

AWS EC2: Install Anaconda on Linux Instance

Setting up coding environment:
wget https://repo.anaconda.com/archive/Anaconda3-2023.03-Linux-x86_64.sh
bash Anaconda3-2023.03-Linux-x86_64.sh

Monitor memory usage on AWS EC2

Command to check CPU/memory usage live: htop
Read more here

Insufficient instance capacity

According to the documentation, getting an `InsufficientInstanceCapacity` error when launching or restarting an

instance means that Amazon does not have enough capacity to serve your request. There are a few options:

- Waiting for a while and trying again
- Launching an instance without specifying an availability zone
- Changing the instance type (Samaneh: I did this, and the issue was resolved)

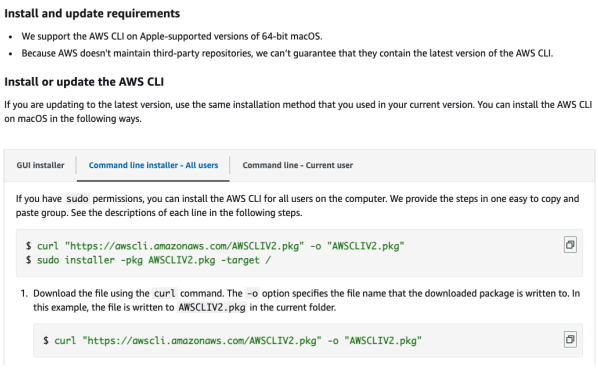
Pricing

If you want to check the price for your instance:
https://ec2pricing.net/
c5a.4xlarge

Copying Files from Local to S3

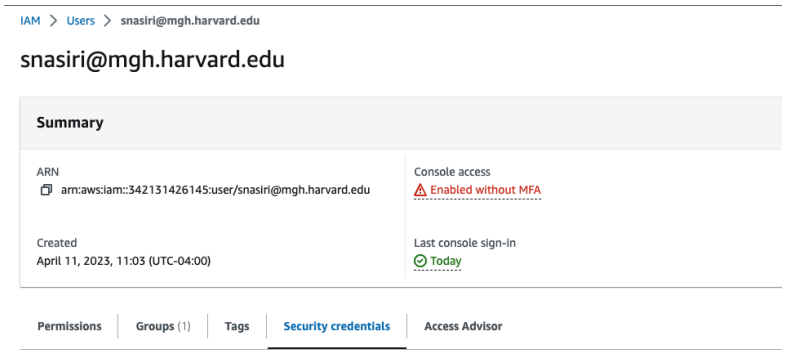
If you want to upload a folder on S3 Bucket:
https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

1. Need to install AWS CLI
    a. Select your operating system.
    b. Go to command line installer - All users, and follow each step in your local terminal

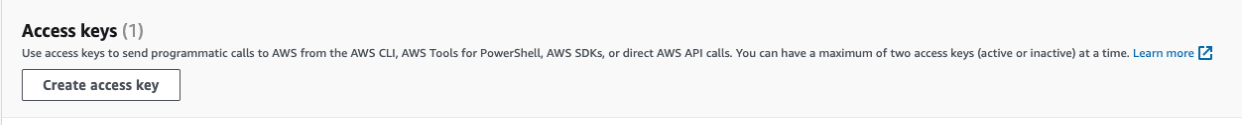


2. AWS CLI Configuration
https://docs.aws.amazon.com/cli/latest/reference/configure/
3. AWS Keys:
https://docs.aws.amazon.com/powershell/latest/userguide/pstools-appendix-sign-up.html

IAM dashboard - users, select your username, and then
https://us-east-1.console.aws.amazon.com/iamv2/home?region=us-east-1



Go to the security credentials;
Then scroll down, to find the following item “Access keys”, and create access key



To copy:
aws s3 sync location_path destination_path

Docker*:*

Docker is a platform that allows developers to package their software applications into containers. Containers are self-contained, isolated environments that have everything a piece of software needs to run, including its dependencies, libraries, and configuration files.

In the context of Python and ML, developers use Docker to create consistent and portable environments for their code to run in. This is especially useful in ML because the software dependencies for different ML models can be complex and specific. By using Docker, developers can ensure that their code runs the same way on any machine, regardless of its operating system or environment.

If you need to set up the docker, please follow this document.

If you lost your .pem file, you can follow the following instructions to access to your old-EC2 instance:
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/TroubleshootingInstancesConnecting.html#step-6-add-new-public-key-to-authorized_keys
This video might help as well: https://www.youtube.com/watch?v=UlnVl95OHKU

Fast EFS share ("The idea is to use this share only while you are processing the data. You will need to remove all files when you finish your tasks.")

How to mount the EFS share ***(don't try to do this on Amazon Linux… apt-get not installed there, but it is there in ubuntu)**
Install amazon-efs-utils (only need to be installed once) – for ubuntu/deb only

```
sudo apt-get update
sudo apt-get -y install git binutils
cd /tmp
git clone https://github.com/aws/efs-utils
cd /tmp/efs-utils/
./build-deb.sh
sudo apt-get -y install ./build/amazon-efs-utils*deb
```

Mount the file system – need to be mounted every time you restart the instance:

cd /bdsp
sudo mkdir efs-dst
sudo mount -t efs -o tls fs-0137c4caf24d7dc05:/ efs-dst
df -h (to check if the efs storage was mounted)
aws/efs-utils'

Screen (keep code running when ssh connection is closed)
`screen` is a command-line tool used for terminal multiplexing, allowing you to run multiple terminal sessions and switch between them within a single window. More importantly, it allows processes to keep running even after closing the terminal, with the ability to reattach to them later, making it invaluable for long-running tasks or remote session management.

Firstly, you need to make sure `screen` is installed on your remote server. You can check by typing `screen` in your terminal. If it's not installed, you can install it using a package manager:

On Ubuntu/Debian:
```bash
sudo apt-get install screen
```

Here's how to use `screen`:

1. Start a new screen session:
   ```bash
   screen -S my_session
   ```
   Here, `my_session` is a name you give to the session. It can be anything you like.

2. Run your program. For example, if you're running a Python script:
   ```bash
   python my_script.py
   ```

3. Detach from the screen session by pressing `Ctrl+A` followed by `D`. Your program will continue to run in the background, even if you close the SSH connection.

4. You can then close your terminal or SSH session, and your process will continue to run in the background.

To reattach to the session:

1. List the running screen sessions:
   ```bash
   ```

```
screen -ls
```

You should see a list of running sessions, including the one named `my_session` that you started earlier.

2. Reattach to a session using its name:

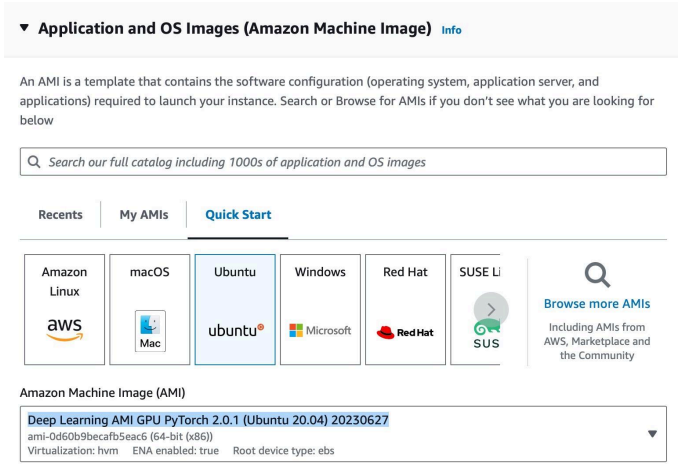
```bash
screen -r my_session
```

Now you're back in the `screen` session and can see the output of your running program.

3. If you want to terminate the session, you can simply reattach to it and then close the terminal in the session by typing `exit` or pressing `Ctrl+D`.

Installing pytorch for GPU

When installing pytorch, it is important to make sure that the CUDA (GPU capable) version is installed.
One easy way do so is to launch a new instance with pytorch 2.0 preinstalled.
To do so run Launch an instance -> Application and OS Images -> Quick Start -> Ubuntu -> Amazon Machine Image -> Deep Learning AMI GPU PyTorch 2.0.1 (Ubuntu 20.04) 20230627
****Be careful. If you use this one listed, you will be good. If you use the one which is Amazon Linux instead, you will have lots of problems. Make sure to get the one with Ubuntu, unless you know what you are doing.**
And configure the rest as described at the very top of this document



Setting 'alarms' to turn automatically turn off your instances

This tutorial shows you how to set up an alarm that turns off your instance after the CPU usage has been <1% for 1 hour

https://successengineer.medium.com/how-to-automatically-turn-off-your-ec2-instance-in-2021-b73374e51090