MIT GSL
GLOBAL STARTUP LABS

AWS API for GSL-Uruguay

Julian Alverio, Ryan Sander, Evan Pu, Madhav Kumar

GitHub Repo for this Project

Guidelines for Using this API

Overview

For the GSL-Uruguay program, we implemented an AWS API using Python (`boto3`) and Bash to set up AWS EC2 instances for our students. We created these instances because we (1) were able to acquire AWS credits for this program, and (2) to provide computing power and appropriate computing environments for our students (to make sure that all package and software requirements for this class are compatible across different operating systems and software versions). We outline the basics of this API below, and have shared links to our GitHub repository and custom Amazon Machine Image (AMI).

Installation & Preparation

To use this system, make sure you have made the following preparations:

1. Clone the GitHub repo here.
2. Make sure you have a Python interpreter set up on your computer.
3. Make sure you have an active AWS account (ideally with credits, to avoid receiving personal charges), and make sure you have a secret `.pem` key associated with this account. See this guide for installing `aws_cli` for configuring your key files.
4. Make sure you have the libraries `boto3` and `pandas` installed - these can both be installed via pip or conda. We will need these packages when configuring instances through the Python API.

Instance Management Guide

After completing the installation requirements above, you can follow the steps below to set up custom AMI instances for your students:

1. Make sure your secret key `.pem` file lies in the same directory as your cloned `machine_learning_aws` repository. This is important when starting/terminating

instances using the command line and `boto3`. If you need to set up your AWS account that you plan to link for this program, see the section below: "Tips for setting up machines like these on another AWS account". Please remember to NEVER push this `.pem` file to GitHub.

2. If you need to make any modifications to the Amazon Machine Image (AMI) (the virtual image that is created with each new instance), make these modifications in MAKE_AMI.txt (you'll copy and paste the contents of this file to an Ubuntu instance). From here, you can create a new AMI based off of the original AMI by following instructions 3-7 from the section below "Tips for setting up machines like these on another AWS account".
3. Once you're ready to start instances using our custom AMI or your own custom AMI (our custom AMI id: `ami-0c505f47bff8ab1b2`), you can do so by running the following command on a command line:

   ```
   python aws_api.py --start
   ```

   This will create instances according to the information in `users.csv`, which should not be pushed to GitHub for user's privacy but should be stored on the local machines that are starting/stopping these instances. For formatting the `users.csv` file, make the first column the user's name (what will be emailed to each student when they receive their credentials), and the second column the user's email. Note: The number of instances created will be the number of rows (users) present in `users.csv`.

4. When you're ready to terminate the students' instances (we did this every day to have the machines automatically pull new lessons from GitHub), you can do so by running the following command on a command line:

   ```
   python aws_api.py --stop
   ```

NOTE: The file `aws_api.py` also has other optional flags. Run `aws_api.py --help` to view these different flags and their explanations. By default, when running this script, exactly one option of `--start`, `--stop`, `--backup`, `--info`, or `--email` must be selected.

Tips for setting up machines like these on another AWS account

1. Load AWS credits onto someone's account.
2. Generate a .pem key. Keep this VERY secure and NEVER push this to github. Automated scrapers will steal your key and spent your money.

3. Start up an ubuntu AMI (AMI id: `ami-00a208c7cdba991ea`).
4. Look up its ip address on the EC2 console. ssh into the machine using:

```
ssh -i /path/to/pem ubuntu@<IP_ADDRESS>
```

5. Paste the commands one at a time from `MAKE_AMI.txt`. See `MAKE_AMI.txt` to see what each line does. It will:
    - clone the repo
    - install conda and initialize
    - build a conda env
    - set a cronjob to run at boot time
    - allow for ssh-ing with passwords.
6. You don't have to do anything for this step. At boot time, `setup.sh` will run. It will set the password, pull the latest version of the repo, and set up a jupyter notebook server. To interact with the jupyter notebook, set up port forwarding as follows:

```
ssh -o "StrictHostKeyChecking no" -NfL 5005:localhost:8888
ubuntu@<IP_ADDRESS>
```

7. Once all of this is set up you can go to the EC2 console, select your instance, click action → images → create image. Follow the prompts to create a new AMI. You can now start up a machine that will be identical to the one from before. Any datasets you loaded onto the machine to start will also be copies to subsequent copies.

Security

This whole process is very insecure, and is only usable for teaching purposes. DO NOT use this for anything intellectual property-sensitive.

1. The way we set up these instances, the jupyter notebooks are unencrypted and do not require a password. Anyone can intercept your data packets and see what you're doing.
2. The password is hard-coded for all the machines in `setup.sh`, and is publically available. Anyone who knows the IP address of your machines can log into them.

Backing Up Student Code

This repository also has built-in capabilities to create backups of the students' work from the class for the students to view. This backup process can be run by calling the class

method `backup_machines()` within `aws_api.py`. When this class method is called, students' code is backed up to `student_code/USERNAME`, where the USERNAME is everything that comes before the "@" in the student's email address, with any non-alphanumeric characters deleted, and all remaining characters lowercased. For example, éstuianté!@example.com would find their code under `student_copes/stdudiant`.

Instructors' Code

The instructor's version of the code can be found under `instructor_code/`. These files can be modified accordingly - we recommend that if you don't plan on running Jupyter notebooks on an AWS machine while you are presenting, that you keep separate Jupyter notebook files for both AWS/non-AWS machines to avoid path conflicts.

Jupyter Notebooks for Machine Learning

One of the primary purposes of this repository is to maintain a persistent state for our interactive teaching code base, which includes both template and student-edited Jupyter notebooks. Instructions for accessing template Jupyter notebooks, student Jupyter notebooks, and data are each discussed individually below:

1. Jupyter Notebook under `instructor_code`: These are templates for the student to modify.
2. Jupyter Notebook Files for Students: We will copy the template notebooks from `instructor_code` to a location where the students can use it every day. For now, the location is `machine_learning_aws/daily_user`, though this location may change as we continue the development of this repository.
3. Data for Students: Data for this course can be found on the EC2 instances under the `machine_learning_aws/data/` sub-directory in the `machine_learning_aws/` repository, or through the absolute path `/home/ubuntu/machine_learning_aws/data/`. Note that this data already exists on our custom AMI.

Installing and Configuring Conda Environment on EC2 Instances: Ubuntu AMI

We designed the login and student setup process for these machines to be as straightforward and efficient as possible. Upon startup of the EC2 instances (when the

instances are initialized), a `tmux` session is opened, the conda environment `conda_env` is activated, a `jupyter notebook` command with port assignment is set up, and then the `tmux` session automatically detaches. This ensures that the Jupyter notebook will continue to run remotely even if there is a loss in ssh connection.

Therefore, the only steps that need to be taken for login are the following (students receive these via email whenever new instances are created, which ideally should happen whenever new/updated code is released for the students to use):

1. ssh port forwarding (to listen to the remote EC2 Jupyter notebook):

```
ssh -o "StrictHostKeyChecking no" -NfL 5005:localhost:8888 ubuntu@<IP_ADDRESS>
```

2. In your web browser (e.g. Google Chrome, Safari, Internet Explorer), go to the following:

```
localhost:5005
```

You should now be able to see the Jupyter notebook interface! Changes made here are made on the remote EC2 instance, but our team has also built-in functionality in `aws_api.py backup_machines()` for backing up all student work to this GitHub repository, and if/when students are assigned to new instances, work from their user-specific folders is automatically downloaded onto their new assigned EC2 instances.

NOTE: If step 1 does not work because "port 5005 is already in use", try killing the port:

a. Mac/Linux users: Type the command `pkill -f 5005`, and repeat step 1.

b. Windows users: Restart/reboot your computer.

Installing and Configuring Conda Environment on Local Machines:

To provide our students with local access and practice to the machine learning resources/concepts taught in this course, we have also created a framework for our students to work on this code in their own Anaconda environments. Our team created a distributable Anaconda environment that contains all the necessary Python packages needed for this course called `local_requirements.txt`, which can be accessed through

this repository. Users can set up this conda environment and add it to their Jupyter notebook through the following steps:

1. After downloading Anaconda, open an Anaconda prompt (Windows) or a command line (Max/Linux), and change your working directory to the local version of this repository. From here, follow the OS-agnostic commands below:
2. Create the Anaconda environment in the command line:

```
conda env create -f local_env_requirements.txt -n local_env python=3.7
```

3. Activate the conda environment:

```
conda activate local_env
```

4. Install other packages the user may want to have in this Anaconda environment:

```
conda install <package_name>
```

OR

```
pip install <package_name>
```

OR

```
pip3 install <package_name>
```

5. In order for users to use the packages in their local environment in a local Jupyter notebook, add the kernel for this environment to `ipython`:

```
ipython kernel install --name local_env --user
```

The user should already have the packages `ipykernel` and `ipython` installed, but in case running the above command creates errors, make sure these packages are installed via pip:

```
pip install ipykernel
```

```
pip install ipython
```

6. Next, open a Jupyter notebook:

```
jupyter notebook
```

Select any `.ipynb` file, and open it. Once the notebook is loaded, find the top menu bar, scroll over "kernel". Find "Change kernel" at the bottom, and the user should then be

able to see and select "local_env". Click on "local_env"; the user's packages from `local_env` should now be import-compatible.

From here, the user can edit/create/delete files in Jupyter. For future login and use of this Anaconda environment, the user will simply need to open a Jupyter notebook and ensure that the kernel `local_env` is selected.

Installing TensorFlow 2.1 on Your Local Machines

To install TensorFlow 2.1, following these instructions (inside your conda environment):

1. Activate your conda environment:

```
conda activate local_env
```

2. Uninstall any existing installations of TensorFlow:

```
conda uninstall tensorflow
```

3. Upgrade pip, another Python package manager:

```
pip install --upgrade pip
```

4. Install TensorFlow 2.1 using pip:

```
pip install tensorflow
```

Credits

Thank you to the MIT GSL and Amazon AWS Educate teams for providing us with Amazon computing resources for this course.