Introduction to AWS

Tim Book

GENERAL ASSEMBLY

What is Cloud Computing?

Yet another buzzword in our field. Luckily this one is simple:

Cloud computing is computing on a server that is maintained by someone else.

Why Cloud Computing?

There are two major reasons why you would do this:

1. **Powerful computers are expensive.** It is much cheaper and more efficient to rent them hourly from someone else.

Cloud computing services such as AWS give us access to servers that would cost tens of thousands of dollars if we bought them ourselves. You could even execute code on one of these servers remotely from your laptop!

Why Cloud Computing?

There are two major reasons why you would do this:

2. **Maintaining servers is cumbersome** (and expensive), even for medium-sized companies.

For data scientists, compute times can range from minutes to days. What if the power goes out? What ensures your data won't be lost? Worse: if you maintain a website, your server needs to be **running at all times!** That's an expensive electric bill.

What is a Server?

The word **server** is just another word for **computer** or **computer program** that communicates with, or provides functionality for, a **client**.

Your computer, right now, is a server. Your computer might also be running another server: **Jupyter**.

For us in this lesson, we'll mostly be concerned with **computing servers** supplied by AWS.

 © 2018 General Assembly

Who Supplies Them?

There are several companies that provide cloud computing servers as a service. The biggest three are:

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure

 © 2018 General Assembly

Where are these servers?

Amazon has a few server farms throughout the country. Amazon likes to keep their exact locations as secret as possible for security purposes.

On the east coast, we'll be connecting to **northern Virginia**. There are multiple on the west coast.

Ok, cool. How do we do this?

We're going to go through several steps to get up and running with AWS:

1. Set up our keys for security.
2. "Spin up" a server.
3. Connect to the server.
4. Install Anaconda
5. Push data to the server.
6. Execute code on the server (either via the command line or Jupyter).
7. Pull results down from the server.

 © 2018 General Assembly

Step 1: Security

Since we'll be hosting potentially sensitive/proprietary data on the web, it's important that no one else can hack into our server!

In order to access our own servers, we'll need two things:

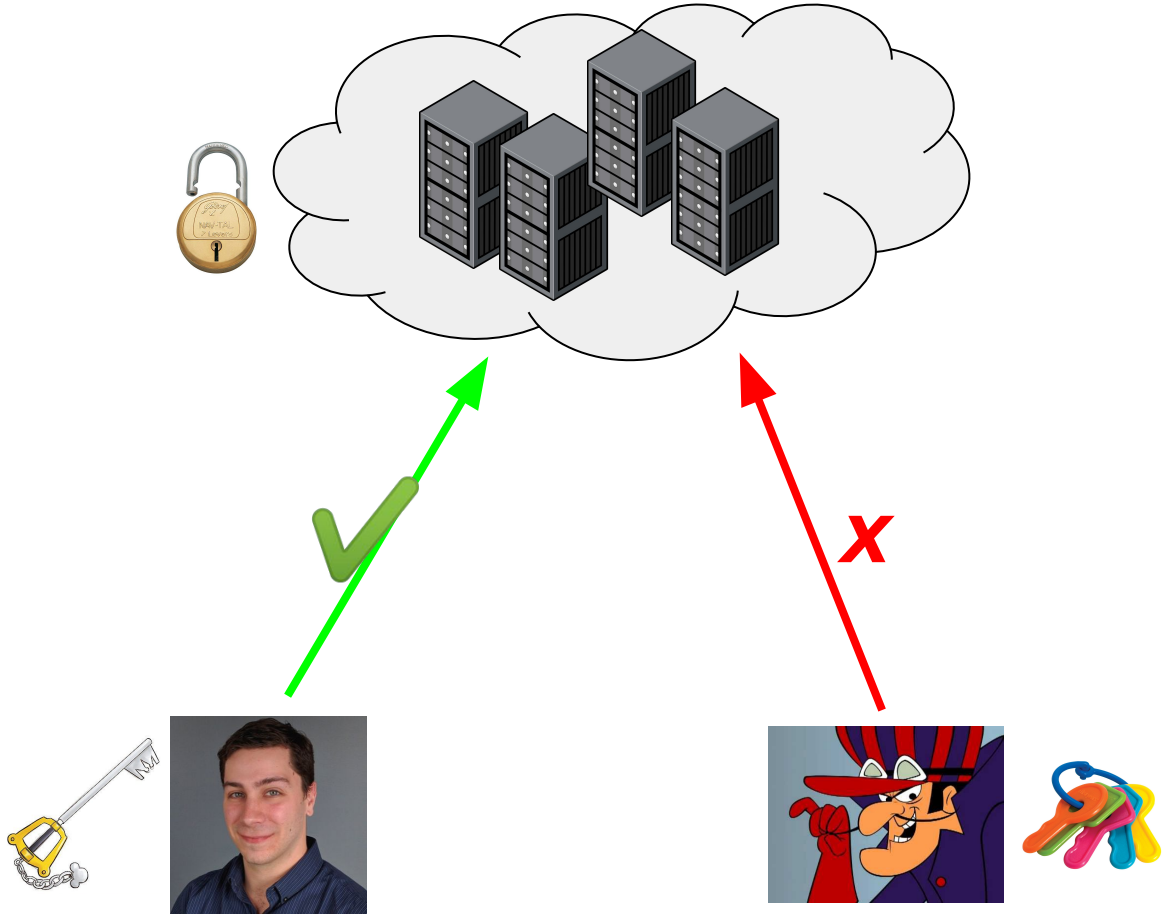
1. A security key (a .pem file)
2. Permissions from the admin (you)

Step 1: Security

You'll need a key to get into your own server.

The **private version** of your **key pair** will be a .pem file.

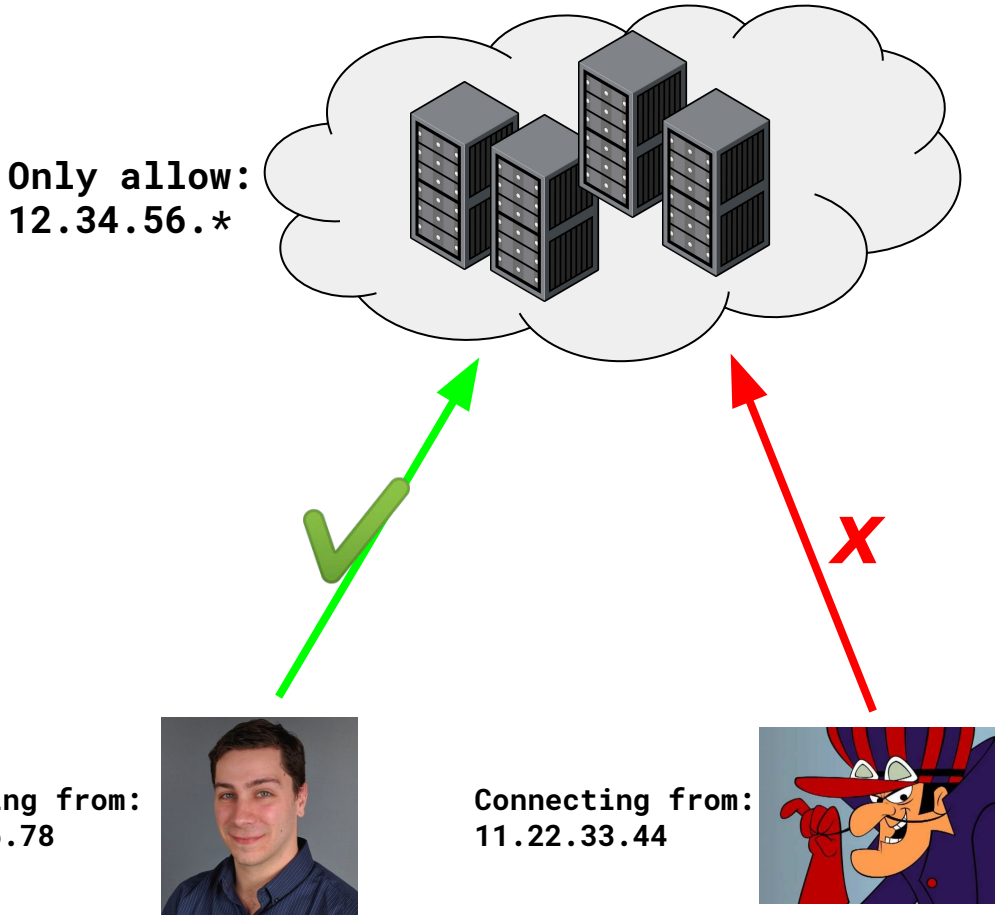
(Side note: AWS will demand your .pem file has certain security settings. You may need to chmod it.)



Step 1: Security

You'll also need to be allowed by permissions of a **security group**.

In most cases, this will be a range of allowable IP addresses you may connect from.



 © 2018 General Assembly

Step 1: Security

Let’s configure this now!

Step 2: Spinning up a server

You'll have a seemingly overwhelming amount of choices to make here. Let's break it down one at a time:

Choice 1: Operating System

AWS offers several flavors of Linux, as well as some versions of Windows.

We're choosing Linux. Never choose windows! This is advice I give in all instances, not just cloud computing. But especially in cloud computing.

The most beginner- and user-friendly flavor Linux is **Ubuntu**. It's also the most widely used, which makes things easier.

 © 2018 General Assembly

Choice 2: Instance Type

You'll also be presented with a buffet of instance choices. Your decision will largely be based on two things:

1. How much memory do you need?
    - The amount of RAM you have determines how much data you can hold in memory at once. Larger data = more RAM needed.
2. How many cores do you need?
    - The amount of cores you have determines the number of threads you may parallelize with (i.e., **`n_jobs`**). More threads = faster compute time.

 © 2018 General Assembly

Choice 2: Instance Type

Bigger instances are more expensive. Don't fall into the following pitfalls:

- **Don't choose an instance weaker than your own personal machine.** This is a waste of money for no benefit! If you don't need AWS in the first place, don't use it!
- **Don't choose a high-powered instance if your model is not parallelizable.** You probably won't see any improvement, and you'll just be wasting money.

Here is AWS's EC2 pricing guide!

Step 2: Spinning up a server

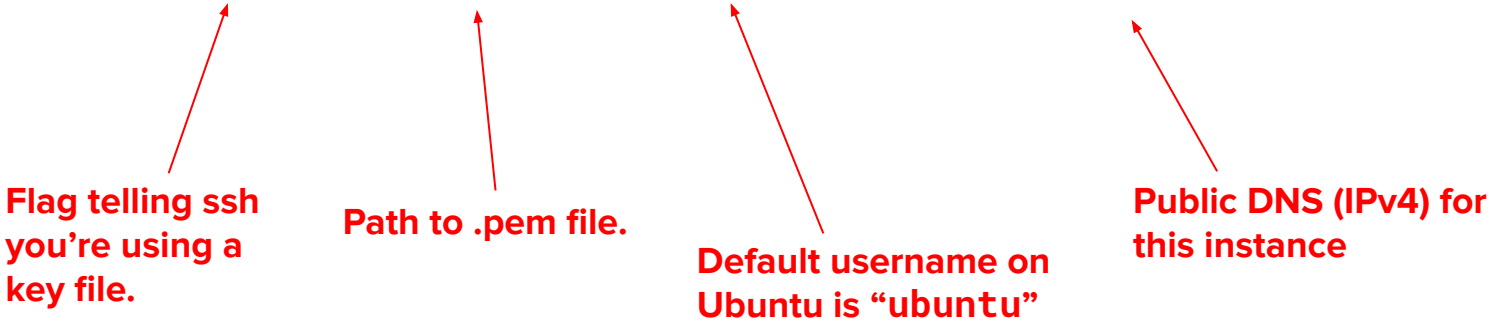
Finally, you'll need to tell AWS your security settings. In the tab labeled **6. Configure Security Group**, you'll specify your security group. After you hit "Launch" from **7. Review**, you'll be prompted for the key you want to use.

Let's do all of this now!

Step 3: Connect to the Server

Using our command line, we'll connect to our server using a new protocol: **SSH**, the **s**ecure **sh**ell protocol. As long as you've properly **chmod**ed your key, this is easy:

```
ssh -i /path/to/key.pem ubuntu@ec2-ip-address.amazonaws.com
```



 © 2018 General Assembly

Step 4: Install Anaconda

This is as simple as a few CL commands:

(Pull Anaconda install from internet)

```
wget http://repo.continuum.io/archive/Anaconda3-4.1.1-Linux-x86_64.sh
```

(Execute installer - make sure to say “yes” - add to PATH)

```
bash Anaconda3-4.1.1-Linux-x86_64.sh
```

(Restart Bash configuration)

```
source ~/.bashrc
```

(Check to see if it all worked!)

```
which python3
which conda
```

 © 2018 General Assembly

Step 4½: Something something Jupyter Notebooks

Yes, the rumors are true, you can actually run a Jupyter Notebook in your EC2 instance and interact with it via your local web browser. Setting this up requires an additional setting in your security group, and the manual editing of a few config files in your AWS instance. I'm not going to do this here because it's very error-prone and is just a few more directions to follow, but here are two very good tutorials:

Tutorial from the legendary Chris Albon

Another tutorial from Alex Sanchez

Step 5: Uploading files to the instance

In order to save time, money, and energy, **you should be building your code locally and then uploading**. You'll also need to upload your data. You can transfer files back and forth with yet another protocol: **scp** (secure copy protocol).

 © 2018 General Assembly

Step 5: SCP

Easy mode: Using your mouse

There are GUI-based SCP clients that make this very easy:

For Windows: **WinSCP** is best

For Everyone: **Filezilla** is popular

Step 5: SCP

Expert mode: Using the command line. (Reminder: you are an expert)

(Copy file.txt from local to remote)

```
scp -i key.pem file.txt ubuntu@ip-address:~/target_directory
```

Note!

(Copy my_folder directory from local to remote)

```
scp -i key.pem -r my_folder ubuntu@ip-address:~/target_directory
```

Note!

(Copy file.txt from remote to local)

```
scp -i key.pem ubuntu@ip-address:~/data/file.txt ~/local_dir
```

(General structure)

```
scp -i key.pem from to
```

 © 2018 General Assembly

Step 5: SCP

Let's bring **`model.py`** into our instance!

Step 6: Execute code!

Hopefully, the easiest part! Let's run:

```
python3 model.py
```

Note that because of Linux, we'll be executing Python scripts with **`python3`** instead of **`python`**.

This is a miniature example, but:

When I run a RandomForestClassifier on an arbitrary classification with:

$n = 300{,}000$

$p = 20$

$C = 5$

Location	Cores	Time
Local	1	13m17s
Local	8	3m47s
c5.9xlarge	36	???

Step 7: Pulling the results back down

Remember to use your favorite **scp** tool!

Let’s bring our output.csv down...

 © 2018 General Assembly

Recap

Wow that probably felt like a lot! Remember what the goal here was:

1. Instantiate powerful server
2. Put code and data on that server
3. Run complex code on server
4. Pull down results

We just had a lot of red tape to get through to get here. A great way to get this down is to go through it several times. Practice spinning up, SSHing in, and terminating server. Being comfortable with the command line is always a plus, too!