Introduction to AWS

Tim Book & Jeff Hale

GENERAL ASSEMBLY

What is Cloud Computing?

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

Why Cloud Computing?

1. Powerful computers are expensive or unavailable.
2. Change compute and storage to a variable expense.
3. Maintaining servers is a pain.
4. Reliability redundancy benefits.

What are Clients and Servers?

A **server** is a computer or computer program that communicates with another computer or device called a **client**.

Clients and Servers

You computer can be both a client and a server. Running Jupyter locally makes it a server.

When interacting with Jupyter in the browser it is functioning as a client.

We'll use **computing servers** supplied by AWS.

Who Supplies Them?

Many companies provide cloud computing servers as a service. 3 biggest:

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

Where is the cloud?

The real question is where are the servers?

All over the world. 🌍

Choose a server near your client.

Popular AWS Services

EC2 = Elastic Compute Cloud: processor with memory

S3 = Simple Storage Service: blob storage (throw anything in)

RDS = Relational Database Service: (e.g. PostgreSQL)

Ok, cool. How do we do this?

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.
8. SHUT DOWN YOUR RUNNING INSTANCE

Step 1: Security

It’s important that no one else can hack into our server!

To access our own AWS servers we need two things:

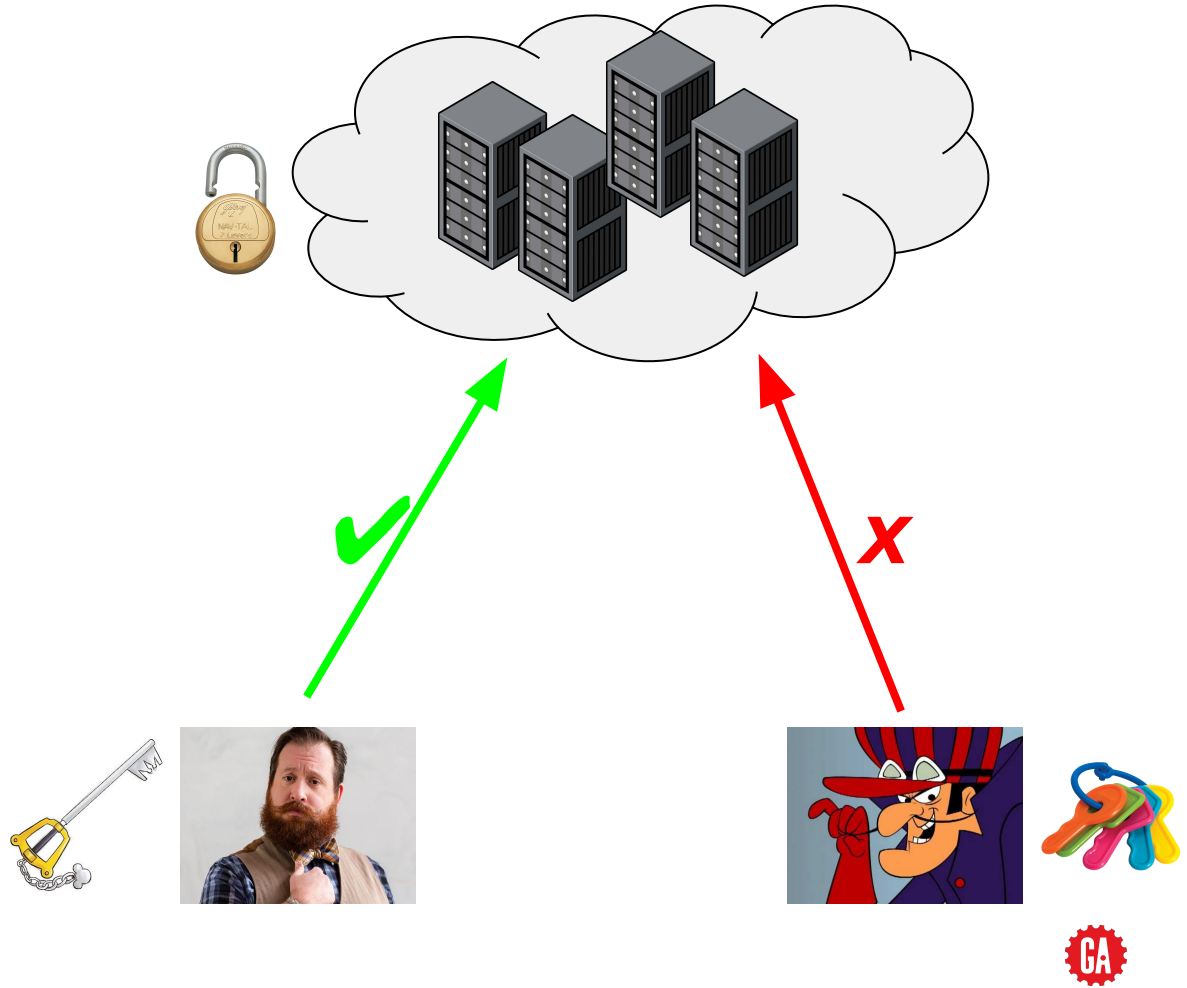
1. A security key (a .pem file)
2. Permissions from the admin (here that’s you)

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.

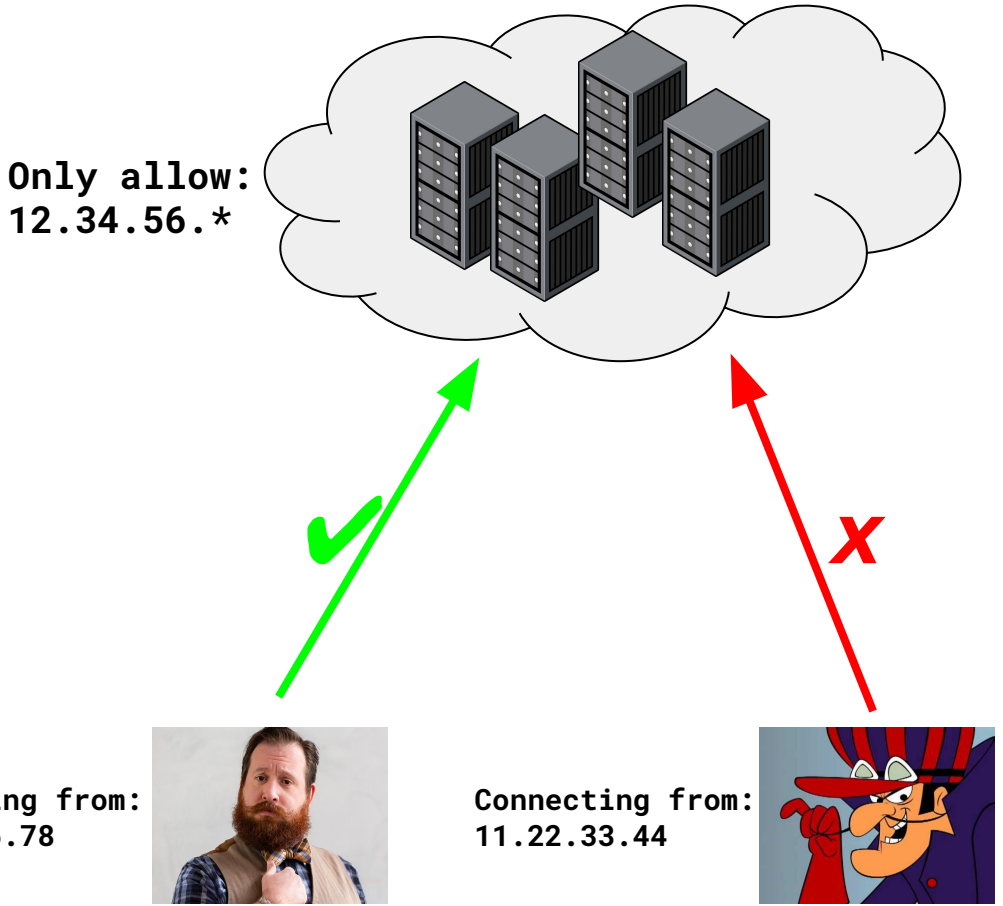
(Side note: AWS will demand your .pem file has certain security settings. You may need to chmod it.)



Step 1: Security

You'll also set the permissions of a **security group**.

Generally, this will be a range of allowable IP addresses you may connect from.



Step 1: Security

Let’s configure this now!

Step 2: Spinning up a server

EC2

You'll have a lot of choices to make here. Let's break them down.

Choice 1: Operating System

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

We're choosing Linux. Never choose windows!

Most popular Linux flavor is **Ubuntu**.

Choice 2: EC2 Instance Type

You have many instance choices.

Decision factors:

1. How much **memory** do you need? More RAM = more data you can hold in memory at once.
2. How many **cores** do you need? More cores = more threads you can parallelize (i.e., **`n_jobs`**) = faster compute time.
3. **GPU** needed?
4. **$$$** AWS's EC2 pricing guide - there is a free tier and other free trials

Step 2: Spinning up a server

Finally, tell AWS your security settings.

In the tab labeled **6. Configure Security Group**, 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

From your command line, connect to your server using **SSH**, the **s**ecure **sh**ell protocol. Must have properly **chmod**ed your key.

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

Flag telling ssh you're using a key file.

Path to .pem file.

Default username on Ubuntu is "ubuntu"

Public DNS (IPv4) for this instance

Step 4: Install Miniconda

CL commands:

(Fetch Anaconda install from internet)

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

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

```
bash Miniconda3-latest-Linux-x86_64.sh
```

(Restart Bash configuration)

```
source ~/.bashrc
```

(Check to see if it all worked!)

```
which python
which conda
```

Step 4½: Jupyter Notebooks in EC2

You can run a Jupyter notebook in your EC2 instance and interact with it via your local web browser.

Setting this up requires several more steps. We aren't doing this now. Instructions:

https://docs.aws.amazon.com/dlami/latest/devguide/setup-jupyter.html

Step 5: Uploading files to the instance

To save time, money, and energy, **you can build your code locally and then upload it**.

You'll also need to upload your data. You can transfer files back and forth with another protocol: **scp** (secure copy protocol).

Step 5: SCP

Option 1: point and click with a GUI-based SCP client.

FileZilla is popular.

Step 5: SCP

Option 2: the command line

(General structure)

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

(Copy file.txt from local to remote)

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

(Copy my_folder directory from local to remote)

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

(Copy file.txt from remote to local)

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


Step 5: SCP

Let's transfer **`model.py`** to our EC2 instance!

Step 6: Execute code!

Hopefully, the easiest part! Let's run:

```
python model.py
```

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

Step 7: Pull the results back down

Use your favorite **scp** tool!

Let’s bring our output.csv down...

Results from a little example:

When Tim ran a RandomForestClassifier on a basic classification problem with:

$n = 300{,}000$

$p = 20$

$C = 5$

Computer	Cores	Time	Improvement
Local	1	13m17s	1x
c5.2xlarge	8	2m37s	5x
c5.9xlarge	36	40s	20x

CliffsNotes

1. Have security features ready (security key, security group)
2. Instantiate EC2 instance SSH into instance (Slide 18)
3. Install Miniconda
4. SCP relevant files to instance
5. Execute code
6. SCP results back down
7. **TERMINATE INSTANCE!**

Summary of Important Things!

1. Don’t forget to terminate instances when you’re done!
2. Keep your security keys secure!
3. Understand if AWS is really what you need!

Lastly……

TERMINATE YOUR INSTANCE