

# AWS + High-Performance Python

---

Erich Wellinger

February 5, 2017

# Morning Goals

- Overview of Cloud Computing
- Amazon Web Services (AWS)
  - Identity & Access Management (IAM) **NEED TO ADD**
  - Elastic Compute Cloud (EC2)
  - SSH Config **NEED TO ADD**
  - Simple Storage Service (S3)
- EC2/S3 Overview and Work Flow

# Overview

- Cloud computing allows us to run programs remotely on distant computers (In the cloud!, a.k.a. a massive server farm located somewhere on land)
- In this scenario, our computer would be the “client” sending jobs to the “server” (e.g. a AWS EC2 instance)
- Advantages
  - Accessibility from anywhere
  - Dynamic Scaling
  - Storage
  - Maintenance

## Disadvantages

- Security
  - A third party owns the server
- Cost
  - In the long term, cloud service and cost a lot

# Why Amazon?

- Flexibility of many instance types
- A lot of traction over the years
- Libraries and tools built around AWS
- Behemoth in the cloud computing sector
  - Amazon Web Services owns ~30% of the cloud computing market

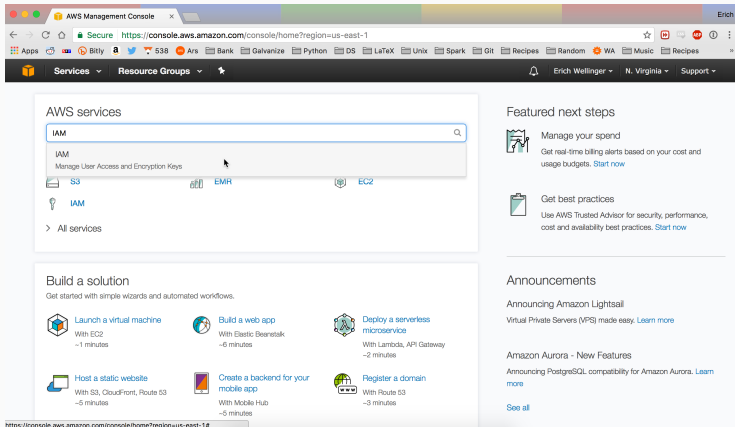
# Identity & Access Management

AWS Identity and Access Management (IAM) allows you to configure what permissions users have in your account. But why does IAM matter if you are the only one using your account?

This matters because anyone that has access to your root account (e.g. through AWS Access Keys) also has access to your billing information which includes CC info. Thus we should set up a Admin user account that sandboxed from accessing billing information.

# IAM (Continued)

## 1. First we need to access IAM after logging into our account



# IAM (Continued)

- Next you should customize your sign-in link. This is the link you will use to sign into the AWS console—bookmark it!

The screenshot shows the AWS IAM Management Console interface. The browser address bar displays the URL `https://console.aws.amazon.com/iam/home?region=us-east-1#/home`. The page title is "Welcome to Identity and Access Management".

**IAM users sign-in link:**  
`https://ewellinger.signin.aws.amazon.com/console`  
A [Customize](#) button is visible next to the link, and a button labeled "Click here to remove the alias" is also present.

**IAM Resources**

Resource Type	Count
Users	4
Groups	0
Roles	4
Identity Providers	0

Customer Managed Policies: 0

**Security Status** 3 out of 5 complete.

Task	Status
Delete your root access keys	Completed (Green checkmark)
Activate MFA on your root account	Completed (Green checkmark)
Create individual IAM users	Completed (Green checkmark)
Use groups to assign permissions	In Progress (Yellow warning icon)
Apply an IAM password policy	In Progress (Yellow warning icon)

**Feature Spotlight**  
Introduction to AWS IAM

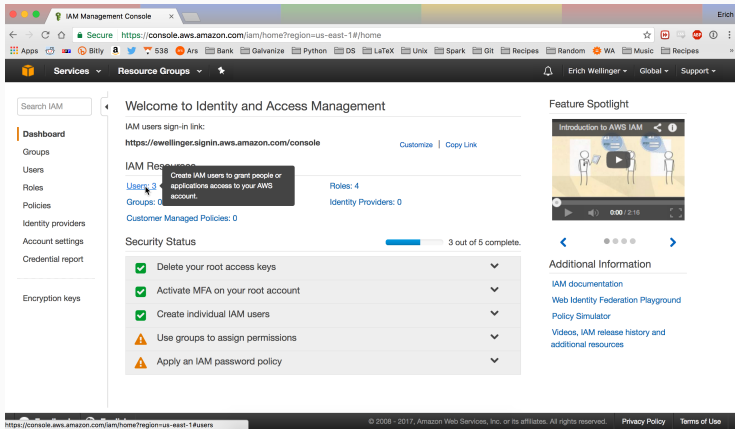
**Additional Information**

- [IAM documentation](#)
- [Web Identity Federation Playground](#)
- [Policy Simulator](#)
- [Videos, IAM release history and additional resources](#)



# IAM (Continued)

## 3. Now it's time to create a new user! Click on Users under IAM Resources



The screenshot shows the AWS IAM Management Console interface. The left sidebar contains a navigation menu with the following items: Search IAM, Dashboard, Groups, Users, Roles, Policies, Identity providers, Account settings, Credential report, and Encryption keys. The main content area is titled 'Welcome to Identity and Access Management' and displays the following information:

- IAM users sign-in link:** <https://ewellinger.signin.aws.amazon.com/console> (with 'Customize' and 'Copy Link' options)
- IAM Resources:** A section with a tooltip that reads 'Create IAM users to grant people or applications access to your AWS account.' The 'Users' link is highlighted with a blue circle and the number '3'. Other links include 'Groups: 0', 'Roles: 4', and 'Identity Providers: 0'.
- Customer Managed Policies:** 0
- Security Status:** A progress bar showing '3 out of 5 complete' with the following items:
  - ✓ Delete your root access keys
  - ✓ Activate MFA on your root account
  - ✓ Create individual IAM users
  - ⚠ Use groups to assign permissions
  - ⚠ Apply an IAM password policy

On the right side, there is a 'Feature Spotlight' section with a video titled 'Introduction to AWS IAM' and an 'Additional Information' section with links to 'IAM documentation', 'Web Identity Federation Playground', 'Policy Simulator', and 'Videos, IAM release history and additional resources'.

# IAM (Continued)

4. Create a user name and select Programmatic access and AWS Management Console access—create a password for this user

AWS Management Console | IAM Management Console | Additional dummy / plain text | Erich

Secure [https://console.aws.amazon.com/iam/home?region=us-east-1#/users\\$new?step=details](https://console.aws.amazon.com/iam/home?region=us-east-1#/users$new?step=details)

Services | Resource Groups | Erich Wellinger | Global | Support

### Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name\*

[Add another user](#)

### Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Access type\* ☒ **Programmatic access**  
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

☒ **AWS Management Console access**  
Enables a **password** that allows users to sign-in to the AWS Management Console.

Console password\* ☐ Autogenerated password  
☒ Custom password

☐ Show password

Feedback | English | © 2009 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. | Privacy Policy | Terms of Use

# IAM (Continued)

- Click on **Attach existing policies directly** and attach the **AdministratorAccess** policy type

The screenshot shows the AWS IAM Management Console interface. The browser address bar indicates the URL: `https://console.aws.amazon.com/iam/home?region=us-east-1#/users/New?step=permissions&passwordReset&accessKey&passwordTyp...`. The page title is "Set permissions for NewUser".

Three options are presented in a row:

- Add user to group (represented by an icon of three people)
- Copy permissions from existing user (represented by an icon of one person pointing to another)
- Attach existing policies directly** (represented by an icon of a document with a checkmark, highlighted with a blue background)

Below these options, a message states: "Attach one or more existing policies directly to the user or create a new policy. [Learn more](#)".

There are two buttons: "Create policy" and "Refresh".

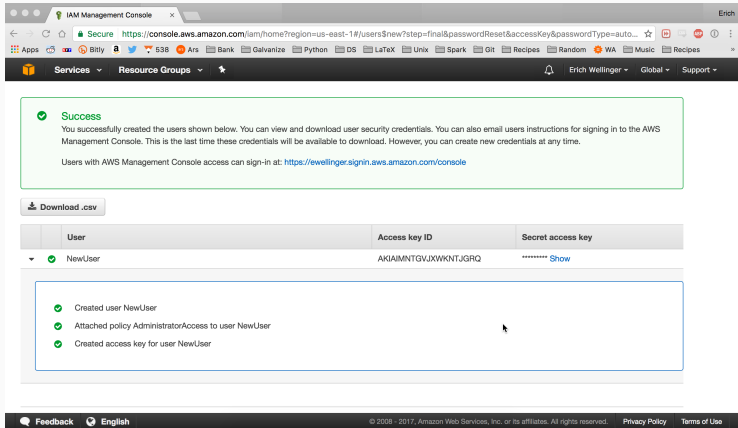
A search bar is present with the filter "Policy type" and the search term "AdministratorAccess". It indicates "Showing 1 result".

	Policy name	Type	Attachments	Description
<input checked="" type="checkbox"/>	AdministratorAccess	Job function	2	Provides full access to AWS services and resources.

At the bottom of the console, there is a footer with "Feedback", "English", and copyright information: "© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved." along with links for "Privacy Policy" and "Terms of Use".

# IAM (Continued)

6. You should now see a success message and an opportunity to download the associated Access Keys



The screenshot shows the AWS IAM Management Console interface. At the top, there's a navigation bar with 'Services' and 'Resource Groups' tabs. Below this, a green success message box states: 'Success. You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time. Users with AWS Management Console access can sign-in at: <https://ewellinger.signin.aws.amazon.com/console>'.

Below the message is a 'Download .csv' button. Underneath is a table with three columns: 'User', 'Access key ID', and 'Secret access key'. The table contains one row for a user named 'NewUser'.

User	Access key ID	Secret access key
NewUser	AKIAIMNTGVJXWKNTJGRQ	***** <a href="#">Show</a>

Below the table, a summary box lists the actions performed:

- Created user NewUser
- Attached policy AdministratorAccess to user NewUser
- Created access key for user NewUser

The footer of the console includes a 'Feedback' button, a language selector set to 'English', and copyright information: '© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.' along with links to 'Privacy Policy' and 'Terms of Use'.

# Elastic Compute Cloud (EC2)

- Virtual Machine in the cloud
- Can choose what Machine to launch
  - Ubuntu (Linux)
  - RedHat (Linux)
  - Many, many, others
- Use local Terminal as the client to access remote machine

## Step 1: Machine Image

This is where we will choose what type of image we will start up. You can either choose one of the default AMIs (Amazon Machine Image), which starts up a fresh copy of your desired OS similar to booting up a new computer, or we can choose from a plethora of community AMIs.

For our purposes we will be using the DSI-Template AMI located in the N. Virginia region.

**NOTE:** AMIs are region specific, so if you can't find an AMI you are looking for you might be in the wrong region!

## Step 2: Specifications

This is where we will choose what kind of resources our machine will have including the number of CPUs, GPUs, and Memory. Each instance will have a varying hourly cost associated with it that fluctuates with current demand.

As a rough guideline, a `m3.xlarge` instance which has 4 CPUs and 15GB of memory costs approximately \$0.25 - \$0.30 per hour.

## Step 3: Configuration

- Spot instances are available to reduce price starting from `m3.medium`
- Bid the machine with the price you set (as max)
- Takes longer to start
- Cannot stop and restart instance
- Much cheaper in general
- Otherwise expensive to use larger instances



## Step 4: Storage

- By default we will have a root EBS volume (more on this later) for storing OS and other files
- We can choose to up this or add on other hard drives

## Step 5: Tagging

- First Key is by default “Name”
- This controls the name that shows up in the Instances section of your EC2 dashboard.
- All this does is help you keep track of which machine is which

## Step 6: Security Group

- Here we will configure the security settings of our instance
- For example we could configure our instance to only accept `ssh` connections from our home or work IP Addresses
- It's also possible for our instance to run a Jupyter server that we can access remotely, but there are some important settings that need to be configured for this to work (ask me if you're interested in doing this)

## Create pem file

In order to access our machine through `ssh` we need to have a pem file which is essentially a cryptographic key.

**CAUTION:** Make sure you have the `.pem` file locally, otherwise create a new key pair

## Change pem permissions

- Once we have downloaded the pem file we need to make sure that it has the proper permission settings
- Run this line in the terminal:

```
chmod 400 my-key-pair.pem
```

- Once we do this, you should move it into your `~/ .ssh` folder for ease of access

## Logging into EC2

```
ssh -i keypair.pem User@Domain
```

- User: ubuntu (if you chose a different OS this might be different!)
- Domain: IP Address associated with your instance

## Copying files to EC2

```
scp -i keypair.pem File.txt User@Domain:/path/where/file/should/go  
scp -i keypair.pem -r Folder User@Domain:/path/to/folder
```

When running scripts on a AWS EC2 instance (or any remote client for that matter), we should be using `tmux`. But what is `tmux`?

- `tmux` is a terminal multiplexer which allows us to multiplex several virtual consoles
- This enables us to create terminal “sessions” that we can then detach from without stopping any running processes
- This works because a `tmux` session’s clients aren’t bound to a specific physical or virtual console



# tmux Session Management

tmux sessions are useful for completely separating work environments.

- `tmux new -s session_name`
  - Creates a new tmux session named `session_name`
- `tmux attach -t session_name`
  - Attaches to an existing tmux session named `session_name`
- `tmux list-sessions` or `tmux ls`
  - Lists all existing tmux sessions
- `tmux detach` or prefix + `d`
  - Detaches from the currently attached session
- `tmux kill-session -t session_name`
  - Kills session `session_name`

When inside a particular session, we can open up new tabs, called “Windows”

- `tmux new-window` or `prefix + c`
  - Create a new window
- `prefix + w`
  - List Windows
- `prefix + n`
  - Next Window
- `prefix + p`
  - Previous Window

## Useful `tmux` commands

By default we can access shortcuts using the `ctrl-b` prefix plus the given shortcut. Below is a short list of useful commands...

- `?` - Lists available commands
- `:` - Brings up prompt
- `d` - Detach
- `c` - Creates a new window
- `w` - Lists windows
- `[` - Enter Copy Mode
- `&` - Kill window
- `x` - Kill pane

This is far from a comprehensive guide to using `tmux`. For more information see <http://bit.ly/2dZzTI3>.

# Stopping Instances

- Make sure to Stop/Terminate your instance if you are not running tasks
- Stop Vs. Terminate
  - Stopping simply shuts the computer down (e.g. you turn off your laptop/desktop). Everything will still be there when you restart the instance
  - Terminating your instance deletes it (e.g. you throw your laptop/desktop in the trash)
- NOTE: If you start and then stop an instance immediately, you will still be charged for a full hour!

# SSH Configuration

- Setting up an SSH configuration streamlines the process of accessing our instance (or any remote host) by aliasing all the details for a particular host into a name of our choosing
- The configuration file is located at `~/.ssh/config` (create that folder and file if it doesn't exist already)

## SSH Configuration Example

The following is an example of an entry in our configuration file. This is specifying a Host located at 54.174.93.43, the user ubuntu, communication over port 22, and points to an identity file located at ~/.ssh/ewellinger-aws.pem. Using this configuration we could access our instance by simply typing `ssh MyNewInstance`

```
Host MyNewInstance
    Hostname 54.174.93.43
    User ubuntu
    Port 22
    IdentityFile ~/.ssh/ewellinger-aws.pem
```

There's just one problem. . . When we shut down our instance, the next time we start it back up we don't know what IP Address it will be assigned. Thus the details we outlined in our config file will no longer be correct.

While we could go in and change our config file every time, it's not exactly the most elegant solution. For that we turn to Elastic IP Addresses!

Elastic IP Addresses allow us to reserve a particular IP Address for our instance so that, when we start it back up, it will always be associated with the same address!

# AWS Elastic IP Address Setup

1. Locate the Elastic IPs button under the NETWORK & SECURITY drop down on the left hand side of the EC2 Management Console

The screenshot shows the AWS EC2 Management Console interface. On the left sidebar, under the 'NETWORK & SECURITY' section, the 'Elastic IPs' option is highlighted with a red circle. The main content area displays a table of instances, with one instance 'MyNewInstance' (ID: i-e1f5157d) selected. Below the table, the 'Description' tab is active, showing details for the selected instance. The instance is in a 'running' state, using the 't2.micro' instance type, and is located in the 'us-east-1b' availability zone. The public DNS is 'ec2-54-174-93-43.compute-1.amazonaws.com' and the public IP is '54.174.93.43'. The elastic IP is associated with the 'us-east-1b' availability zone. The instance is using the 'DSI-Template (ami-4be61326)' AMI and the 'ec2-instance-profile' IAM role. The launch time is 'May 23, 2016 at 9:14:56 AM UTC-6'.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
MyNewInstance	i-e1f5157d	t2.micro	us-east-1b	running	Initializing	None	ec2-54-174-93-43.compute-1.amazonaws.com

Instance: **i-e1f5157d (MyNewInstance)** Public DNS: **ec2-54-174-93-43.compute-1.amazonaws.com**

**Description** | Status Checks | Monitoring | Tags

Instance ID	Public DNS
i-e1f5157d	ec2-54-174-93-43.compute-1.amazonaws.com

Instance state	running
Instance type	t2.micro
Private DNS	ip-172-31-59-68.ec2.internal
Private IPs	172.31.59.68
Secondary private IPs	
VPC ID	vpc-87e376e3
Subnet ID	subnet-945045bf
Network interfaces	eth0
Source/dest. check	True
FRS-optimized	False

Public IP	54.174.93.43
Elastic IP	-
Availability zone	us-east-1b
Security groups	launch-wizard-10, view rules
Scheduled events	No scheduled events
AMI ID	DSI-Template (ami-4be61326)
Platform	-
IAM role	-
Key pair name	ewellinger-aws
Owner	160918117828
Launch time	May 23, 2016 at 9:14:56 AM UTC-6



# AWS Elastic IP Address Setup

## 2. Click on Allocate New Address to reserve a new IP Address

The screenshot shows the AWS Management Console interface for Elastic IP addresses. The 'Allocate New Address' button is highlighted with a red circle. The page displays a table of existing Elastic IP addresses and a sidebar with navigation options.

**Navigation Sidebar:**

- Reserved Instances
- Scheduled Instances
- Dedicated Hosts
- IMAGES
  - AMIs
  - Bundle Tasks
- ELASTIC BLOCK STORE
  - Volumes
  - Snapshots
- NETWORK & SECURITY
  - Security Groups
  - Elastic IPs**
  - Placement Groups
  - Key Pairs
  - Network Interfaces
- LOAD BALANCING
  - Load Balancers
- AUTO SCALING
  - Launch Configurations
  - Auto Scaling Groups

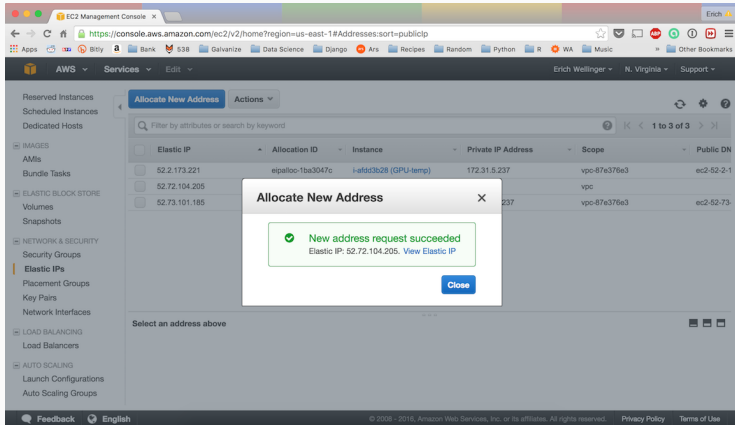
**Table of Elastic IP Addresses:**

<input type="checkbox"/>	Elastic IP	Allocation ID	Instance	Private IP Address	Scope	Public DN
<input type="checkbox"/>	52.2.173.221	eipalloc-1ba3047c	i-afdd3b28 (GPU-temp)	172.31.5.237	vpc-87e376e3	ec2-52-2-1
<input type="checkbox"/>	52.73.101.185	eipalloc-bc26f9db	i-9904af03 (DSI-Template)	172.31.27.237	vpc-87e376e3	ec2-52-73-

Select an address above

# AWS Elastic IP Address Setup

3. Click on Yes, Allocate and you should see a success message telling you what this new address is



# AWS Elastic IP Address Setup

- Next we need to associate this address with our instance by right clicking on the address and clicking Associate Address

The screenshot shows the AWS Management Console interface for Elastic IP addresses. The left sidebar contains navigation links for various AWS services. The main content area displays a table of Elastic IP addresses. One address is selected, and a context menu is open over it, showing options like 'Allocate New Address', 'Release Addresses', 'Associate Address' (which is highlighted with a red circle), and 'Disassociate Address'.

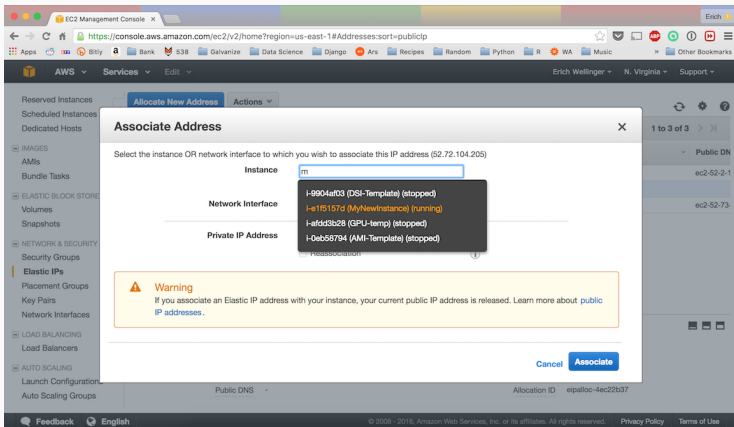
	Elastic IP	Allocation ID	Instance	Private IP Address	Scope	Public DN
<input type="checkbox"/>	52.2.173.221	eipalloc-1ba3047c	i-afdd3b28 (GPU-temp)	172.31.5.237	vpc-87e376e3	ec2-52-2-1
<input checked="" type="checkbox"/>	52.72.104.205	eipalloc-4ec22b37			vpc	
<input type="checkbox"/>	52.72.104.205	eipalloc-bc26f8db	i-9904af03 (DSI-Template)	172.31.27.237	vpc-87e376e3	ec2-52-73

Address: 52.72.104.205

	Elastic IP	52.72.104.205		Network interface ID	-
	Instance			Private IP address	-
	Scope	vpc		Network interface owner	-
	Public DNS	-		Allocation ID	eipalloc-4ec22b37

# AWS Elastic IP Address Setup

5. In the **Instance** field we can start typing in the name we gave our instance earlier and we should see it pop up as the only one that is running



# AWS Elastic IP Address Setup

6. Click Associate
7. Now when we look back at our instance details we can see that the Public IP address has changed to our Elastic IP Address!

The screenshot shows the AWS Management Console interface. The left sidebar contains navigation links for EC2 Dashboard, Events, Tags, Reports, Limits, INSTANCES, IMAGES, ELASTIC BLOCK STORE, and NETWORK & SECURITY. The main content area displays the details for the instance 'MyNewInstance' (ID: i-e1f5157d). The instance is running in the us-east-1b Availability Zone. The Public IP address is 52.72.104.205, which is circled in red. The Elastic IP address is also 52.72.104.205. The instance is associated with the subnet subnet-945045bf and the VPC vpc-87e376e3. The instance is running on the t2.micro instance type.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
MyNewInstance	i-e1f5157d	t2.micro	us-east-1b	running	2/2 checks ...	None	ec2-52-72-104-205

Instance: **i-e1f5157d (MyNewInstance)** Elastic IP: 52.72.104.205

Description		Status Checks	Monitoring	Tags
Instance ID	i-e1f5157d			
Instance state	running			
Instance type	t2.micro			
Private DNS	ip-172-31-59-68.ec2.internal			
Private IPs	172.31.59.68			
Secondary private IPs				
VPC ID	vpc-87e376e3			
Subnet ID	subnet-945045bf			
Network interfaces	eth0			
Source/dest. check	True			
FRS-optimized	False			

Public DNS	Public IP	Elastic IP
ec2-52-72-104-205.compute-1.amazonaws.com	52.72.104.205	52.72.104.205

Availability zone	Security groups	Scheduled events
us-east-1b	launch-wizard-10. view rules	No scheduled events

AMI ID	Platform	IAM role
DSI-Template (ami-4be61326)	-	-

Key pair name	Owner	Launch time
ewellinger-aws	16091817828	May 23, 2016 at 9:14:56 AM UTC-6

## Simple Storage Service (S3)

- Storage space for big data
- S3 Storage → Permanent and big data
- EBS (Elastic Block) Storage → Temporary and small data

# S3 Operations

- Upload
- Download
- Read
- Write

- Extremely cheap
- First 1 TB per month is approximately \$0.03 per GB



- An S3 Bucket is where your data will live
- Can contain folders and files
- Permissions can be configured to dictate access

## Create Bucket

- Bucket name must be unique (i.e. no one has ever used it before)
- Must be all lower case
- Must not have underscores

- We can add a permission dictating that everyone can read the contents of your bucket
- **WARNING:** Don't give everyone the ability to write or modify your bucket

# Bucket Permissions

- Below is an example bucket policy

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "AllowPublicRead",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::<your bucket name>/*"
    }
  ]
}
```

- We can do this through AWS's GUI
- This kind of sucks though, Python for the win!

Pandas has an easy API to read from S3

```
1 import pandas as pd
2 df = pd.read_csv('https://s3.amazonaws.com/bucket/samplefile.csv')
```

We can also specify a chunk size if the data is very large...

```
1 data_chunks = pd.read_csv(url, chunksize=100)
2 df_chunk = data_chunks.get_chunk()
```

# Writing, Deleting, and Managing buckets

- Pandas is great but doesn't allow us to do these other import tasks
- For this we will use Boto, a package for interacting with AWS in Python

```
1 import boto
2
3 # Connect to S3
4 conn = boto.connect_s3(access_key, access_secret_key)
5
6 # List all the buckets
7 all_buckets = [b.name for b in conn.get_all_buckets()]
8 print all_buckets
```

## Create a new bucket

```
1 bucket_name = 'testbucket'
2
3 if conn.lookup(bucket_name) is None:
4     b = conn.create_bucket(bucket_name, policy='public-read')
5 else:
6     b = conn.get_bucket(bucket_name)
```



# Read / Write Files

```
1  # Write new file
2  file_object = b.new_key('sample.txt')
3  file_object.set_contents_from_string("Stuff 'N' Things",
4                                     policy='public-read')
5
6  # Read from file
7  print file_object.get_contents_as_string()
```

## List / Delete Files

```
1  # Print all the files in the bucket
2  filenames = [f.name for f in b.list()]
3  print filenames
4
5  # Delete a file
6  a = b.new_key('somefilename')
7  a.delete()
```

# Delete Bucket

```
1  # Delete Bucket  
2  # Must delete all files in bucket before deleting bucket  
3  conn.delete_bucket('galvanizebucket')
```

# Usefulness of Boto

- Can read/write anything to S3 from anywhere
- Either from EC2 or locally
- Almost unlimited storage

- When data is too big to fit locally
- Scripts that take a long time to run
- Have to run continuously (Hosting Web Apps)

1. Upload/Access data on S3
2. Use pandas/boto to pull a subset down to local
3. Develop script locally
4. Upload script to EC2
5. Run script on EC2 on full set
6. Write results to S3

## Afternoon Goals

- Motivation
- Intro to Computing Resources
- Multi-core Processing
  - Parallelism
- Threading
  - Concurrency

# Motivation

Let's say we want to count the words in a set of documents. To do this in a normal Python script we might set a `count` variable to 0, read in each file in turn, and loop through each line counting the words and adding that value to our `count` variable.

While this works fine for a reasonably sized number of documents, what if we wanted to do this for thousands or even millions of documents? Clearly this approach would be suboptimal.



# Motivation

Instead we could count the words of each document in parallel. This works by calling our counting function (e.g. `count_words(file)`) for each file in parallel where the result is the word count for that particular file.

To then get our total word count we would simply add all these values up!

- Process biggish data ( $\geq 5\text{GB}$  depending on the task)
- Saves time which allows for more coffee breaks
- More efficient use of CPU resources

- **Central Processing Unit (CPU)**
  - Unit: GHz (instructions/second)
- **Random Access Memory (RAM/Memory)**
  - Unit: GB (Gigabytes)

- A CPU can hold multiple cores
- Each core is a self-contained processor that can execute programs
- Internal (OS) Programs are almost always configured to take advantage of these multiple cores by distributing the job to each of your cores.
- External Programs (e.g. a Python script you've written) will generally only use a single core unless specified otherwise

# What happens when a program runs?

- When we tell our program to run it translates our Python code into instructions for the CPU by breaking it up into **processes**.  
A process consists of:
  - Process ID (PID)
  - Program Code
  - CPU Scheduling
  - Memory
- We can take a look at these things by looking at the Activity Monitor on a Mac or running `top` or `htop` (if you have it installed)

# Multicore Processing

---

Now that we understand the CPU, Memory, and Processes fit together, how do we go about writing code that will leverage the multiple cores our computer has?

For this we will use the built-in `multiprocessing` package for specifying how the processes should be split up.

# multiprocessing Sample Code

```
1 from multiprocessing import Pool, cpu_count
2 import os
3
4 # Count the number of words in a file
5 def word_count(f):
6     with open(f):
7         count = sum(len(l.split()) for l in f)
8     return count
9
10 def sequential_word_count(lst_of_files):
11     return sum(word_count(f) for f in lst_of_files)
```



## multiprocessing Sample Code

```
12 # Use multiple cores to count words in files
13 def parallel_word_count(lst_of_files):
14     # Set to use all available cores
15     pool = Pool(processes=cpu_count())
16     results = pool.map(word_count, lst_of_files)
17     return sum(results)
```

# Concurrency and Threading

- A *Thread* is the smallest sequence of programmed instructions that can be managed by a scheduler
- Generally speaking a thread is component of a process which means. . .
  - Multiple threads can exist within a single process
  - They will execute *concurrently*
  - They will share resources such as memory (in contrast, separate processes do not share resources)
- When being run on a single processor, threads are implemented using *time slicing*
  - The CPU switches between different *software threads*
  - Happens quickly enough that users perceive the threads or tasks as running in parallel
- Threads within *the same process* are **not** parallel, instead the CPU will rapidly switch between multiple tasks

# Sharing between Threads

- All threads in the process share:
  - Memory allocated to the process
  - Code of the process
  - Context (i.e. variables)

## Parallelism vs. Concurrency Analogy

---

- Problems that are CPU-intensive
- Tasks that use heavy computations with no wait time in between lend themselves well to running in parallel

# Concurrency Use Case

- I/O bound problems
  - Read/Writing of files
  - Making Web Requests
- When reading, writing, and making web requests, we can thread these processes so as soon as there is downtime we switch to a new thread

# threading Sample Code

```
1 import threading
2
3 threads = []
4
5 # Configure target_function to save result
6 def thread_target_function(arg1, arg2):
7     self = threading.current_thread()
8     self.result = target_function(arg1, arg2)
```

# threading Sample Code

```
1  # Initiate Threads
2  for i in range(num_threads):
3      t = threading.Thread(target=thread_target_function,
4                          args=(arg1, arg2))
5      threads.append(t)
6
7  # Start Threads
8  for thread in threads: thread.start()
9  # join to make sure each thread is finished
10 for thread in threads: thread.join()
11 # Access the result, if any
12 results = [t.result for t in threads]
```



# Summary

- Parallelism on multiple cores
- Threading within a core
- multiprocessing for parallelism
- threading for concurrency
- Parallelism → CPU-bound problems
- Concurrency → I/O-bound problems