

Lab 2

Saif Ali Athyaab - 23810756

Lab 2 covers the creation of an actual EC2 instance. EC2 stands for Elastic Cloud Compute is an IaaS offering by AWS.

What this means is that we have the ability to pay-per-use for a machine of any configuration that we would like. Several options include

- Storage options ranging from a couple GBs to TBs
- RAM, Graphics Card
- Operating System

The technical process of EC2 creation involves

- security groups - which help to maintain/restrict user-level access
- creation of the instance with required config
- running the instance
- getting info about instance like IP address
- terminating the Instance

The last exercise involves Docker. Docker is an application which enables easy shipping of code between machines and instances. The concept of Containerisation jells well with the OOP principles of encapsulation.

Create an EC2 instance using awscli

[1] Create a security group

- A security group controls incoming and outgoing traffic for your EC2 instances by acting as a virtual firewall.
- Inbound rules govern incoming traffic to your instance, whereas outbound rules govern outgoing traffic from your instance.
- When Amazon EC2 chooses whether or not to allow traffic to reach an instance, it considers all of the rules from all of the security groups connected with the instance.

In this case below we can see that we have passed on 2 arguments

- group-name
- description

 create security group

[2] Authorise inbound traffic for ssh

The authorisation function allows to specify the below as parameters

- Protocols like TCP
- IP address ranges

- Port Numbers



[3] Create a key pair that will allow you to ssh to the EC2 instance

The `create-key-pair` command enables to create an encrypted key pair, allowing remote access through SSH. The condition is the `.pem` file should be provided. I had already created earlier, so couldn't include the screenshot. Key-pair I am including the file contents below. Basically, it just specifies the **encryption** algorithm - RSA and the required Hash functions that are used - SHA256, SHA1.

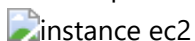
pem file Changing permissions Here is what the `chmod` command does. The owner gets only **READ** permission `chmod` info Credits: <https://chmodcommand.com/chmod-400/>

[4] Create the instance and note the instance id

The **`run-instances`** command is used to launch an instance using the AMI selected. We specify

- the security group
- the count of instances to be launched
- instance type as per our need/budget
- key name

The last part of the code slices through and gets the 0th (latest) instance creates and **returns the instance id**.



Tagging my instance

When an existing tag key is specified, the value is overwritten with the new value. Each resource can have up to 50 tags. Each tag has a key and an optional value. Tag keys must be distinct for each resource.

tagging my instance As you can see, the tag is visible on Console Tag in console

[5] Get the public IP address

The **`describe-instances`** command specifies details of the instances that are called out. There are more than 20 properties for each instance.

- Instance Type
- Architecture
- VpcId
- Availability-Zone

For this reason it is recommended to use `--query` option on the *Reservations* object. There is also a **pagination** option on the main command to allow for simultaneous API calls.

In our case we are only interested in our instance's **IP address**. IP address

[6] Connect to the instance

The **`SSH`** command is a native command in the laptop(Ubuntu), taking in 2 arguments

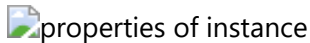
1. The pem file to access the instance
2. The public IP address to connect to



It can now be observed that my CLI prompt on my laptop now displays the EC2 instance new prompt

[7] Look at the instance using the AWS console

Upon filtering, filtering ec2 we can now navigate to our instance and view the properties. Notice that the **Private ipV4 address is same as our local CLI prompt** Instance on Console There are tabs to view all the properties we'd like to examine. Notice the **name of the Key-pair** matching the pem file we created.



Create an EC2 instance with Python Boto script

[1] Python Script to launch an EC2 instance

First we start off by importing boto3 and setting the client as ec2. Next we create a security group using the **create_security_group()** function passing in below parameters.

```
import boto3

ec2=boto3.client('ec2')
sg = ec2.create_security_group(

    Description='Security group from boto3',
    GroupName='23810756boto3-sg'
)
security_group_id = sg['GroupId']
print('Security Group ID %s' % security_group_id)
```



Next we provide the authorisation as required using **authorize_security_group_ingress()**.

```
data = ec2.authorize_security_group_ingress(
    GroupId=security_group_id,
    IpPermissions=[
        {'IpProtocol': 'tcp',
         'FromPort': 22,
         'ToPort': 22,
         'IpRanges': [{'CidrIp': '0.0.0.0/0'}]}
    ])
print('Ingress Successfully Set %s' % data)
```



Next we run the instance using the **run_instances()** function passing in the **SecurityGroupId** and **KeyName** I haven't sliced the response so I am showcasing the instance ID from the output.

```
inst = ec2.run_instances(  
    ImageId='ami-d38a4ab1',  
    SecurityGroupIds=[  
        security_group_id  
    ],  
    InstanceType='t2.micro',  
    KeyName='23810756-key',  
    MaxCount=1,  
    MinCount=1  
)  
  
instance_id='i-0126317b417e358e1'
```

 Instance ID

Finally we can get the Public IP address using the **describe_instances()** function. Here also, I'm manually extracting the output.

```
describe = ec2.describe_instances(  
    InstanceIds=[  
        instance_id  
    ]  
)  
res=describe['Reservations']  
print(res)
```

 Public IP

From AWS Console we can see  Boto instance from AWS Instances used have been terminated

 Terminated

Using Docker


[1] Install Docker

Docker is an open platform for app development, shipping, and running.


Docker enables us to decouple your apps from your infrastructure, allowing you to release software more quickly. Docker allows you to manage your infrastructure in the same manner that you control your applications.

We may drastically minimize the time between writing code and executing it in production by utilizing Docker's methodology for fast shipping, testing, and deploying code.

 docker install


The start and enable commands initialize the docker process.  start enable

[2] Check the version

By simply passing the **--version** argument, we can see the version number  dock version `

[3] Build and run an httpd container


Create a directory called html.

1. Change to the appropriate directory using **cd**
2. Create directory using **mkdir** and change to html directory 
3. Create a file called **index.html**
4. Type in the contents, adding in the tags and *Hello World!*

 html.index

[4] Create a file called "Dockerfile" outside the html directory with the following content:

A **Dockerfile** is a helper document allowing Docker to automatically create the image by following the instructions specified. Our file enables

1. FROM command to enable HTTP requests
2. COPY command to copy our html file contents to the htdocs path 

```
FROM httpd:2.4
COPY ./html/ /usr/local/apache2/htdocs/
```

Include the code and output with descriptions in your report.

[5] Build the docker image


Docker images are created using a Dockerfile and a "context" using the docker build command. The context of a build is the collection of files in the supplied PATH or URL. Any of the files in the context can be referenced by the build process.

In our case the **-t** argument creates a tag called **my-apache2**. The **'.'** indicates current directory

 Docker build


[6] Run the image

The basic intention of the **run** function is to grab the image and attach it to a new container.

- **-p** allows to publish the port number 80:80 (in our case)
- **--name** allows to name our container (my-app my-apache2) 

[7] Open a browser and access address http://localhost or http://127.0.0.1 Confirm you get Hello World!

It is seen that the app is up and running and the browser has rendered our html code from the index.html file

Hello world



[8] Other commands

To check what is running, we run the **ps** command. The **-a** option allows to show non-running containers as well (but we do not have any).

We can see the

- container id
- image name
- command
- time
- Status
- ports

docker ps

To stop and remove the container (self-explanatory) docker stop docker rm