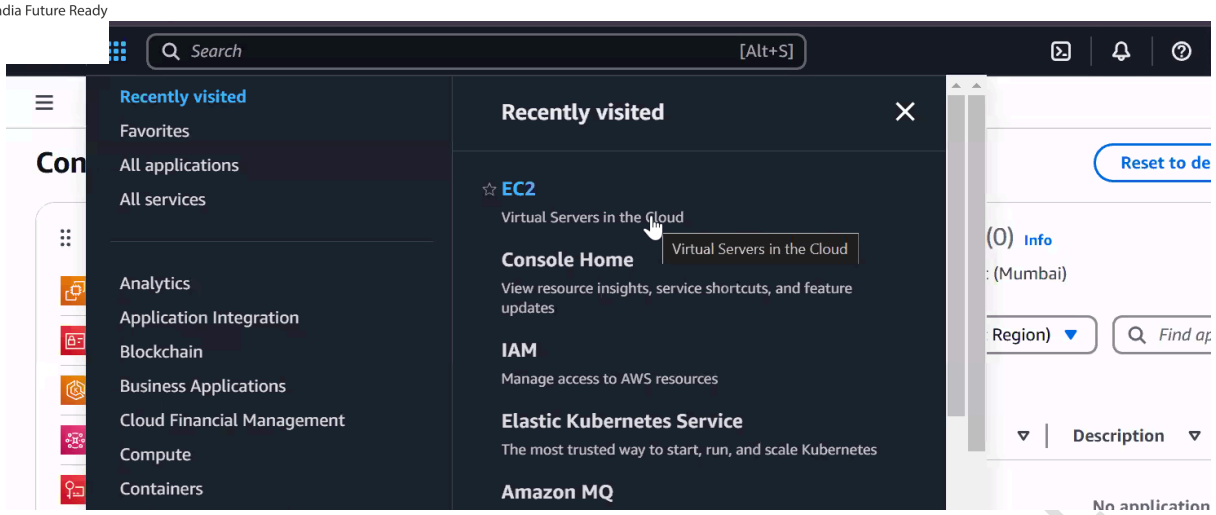
## 7 Days DevOps Deloitte Session 1

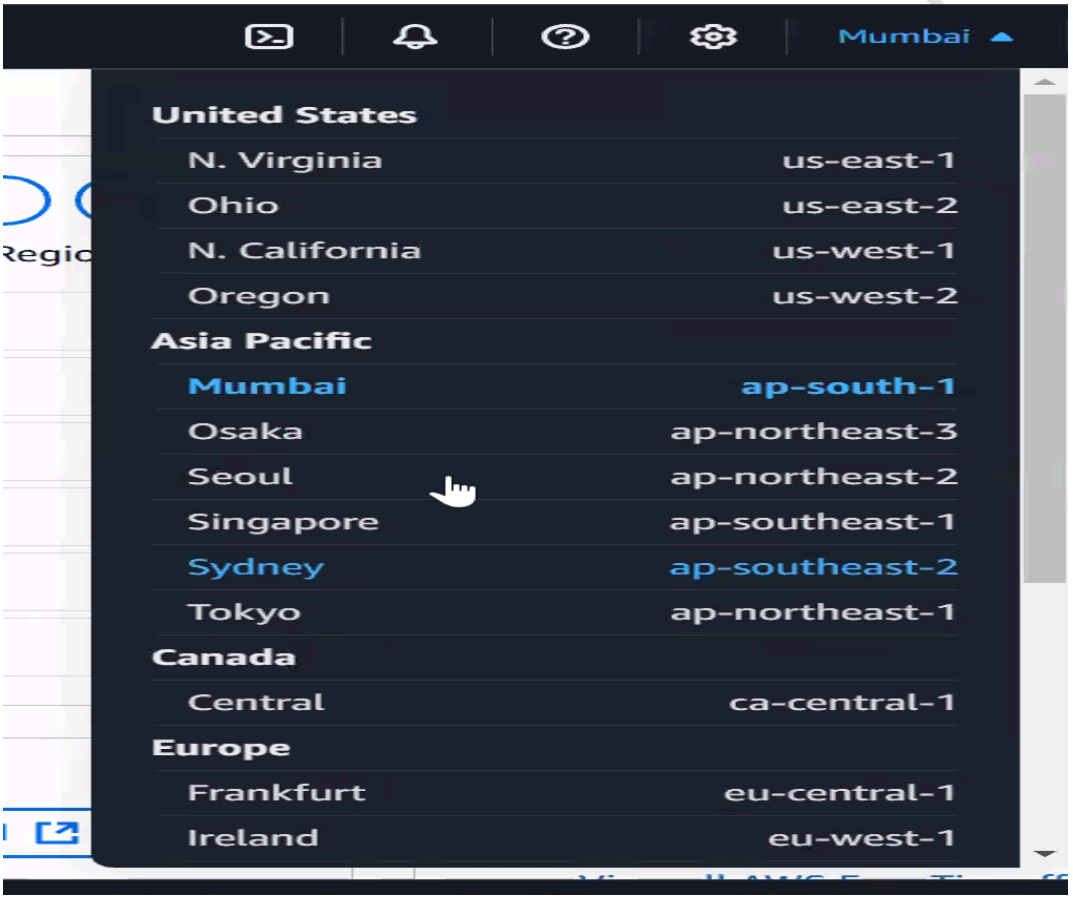
## Summary 02-01-2025

- When you want to run an app or program, you need an operating system (OS). Deploying an app on an OS is known as **deployment**.
- Most projects start with an idea, which leads to creating an application or web app. Developers are hired to build the app using various coding languages. To make the app accessible, it needs to be deployed or hosted on an operating system.
- When users access the app via the internet, the number of users connected at the same time is referred to as **concurrent users**. Physical hardware, such as RAM, CPU, and OS, has specific limits. For example, at any given point, a system may only handle hundreds or thousands of users. If the traffic exceeds these limits, the site may go down.
- During a live cricket match between India and New Zealand, Hotstar handled a record 25 million concurrent users without any lag or downtime. This achievement is attributed to **DevOps principles**, particularly **automation**.
- On the first day of the match, within the first minute, 2 million users joined. The number briefly dropped to 1 million, but as the match started, it surged to 12 million within minutes. Every minute saw an increase of 1 million users. The traffic later dropped from 14 million to 2 million.
- On the second day, traffic started with 1-2 million users, increased to 10 million, and then dropped again. This fluctuating traffic is challenging to handle. Behind the scenes, Hotstar managed thousands of servers to accommodate the surge, a phenomenon known as the **tsunami effect**.
- Any downtime or errors during such events could damage Hotstar's reputation. For example, when India wasn’t performing well, traffic dropped from 11 million to 2 million. However, when interest surged, traffic skyrocketed to 25 million within minutes, setting a world record.
- Hotstar also managed a sudden switch in traffic when users minimized the app during the match, redirecting millions to the home screen. Despite this, Hotstar's systems handled the load seamlessly. Currently, 70% of India's internet traffic flows to Hotstar during such events.

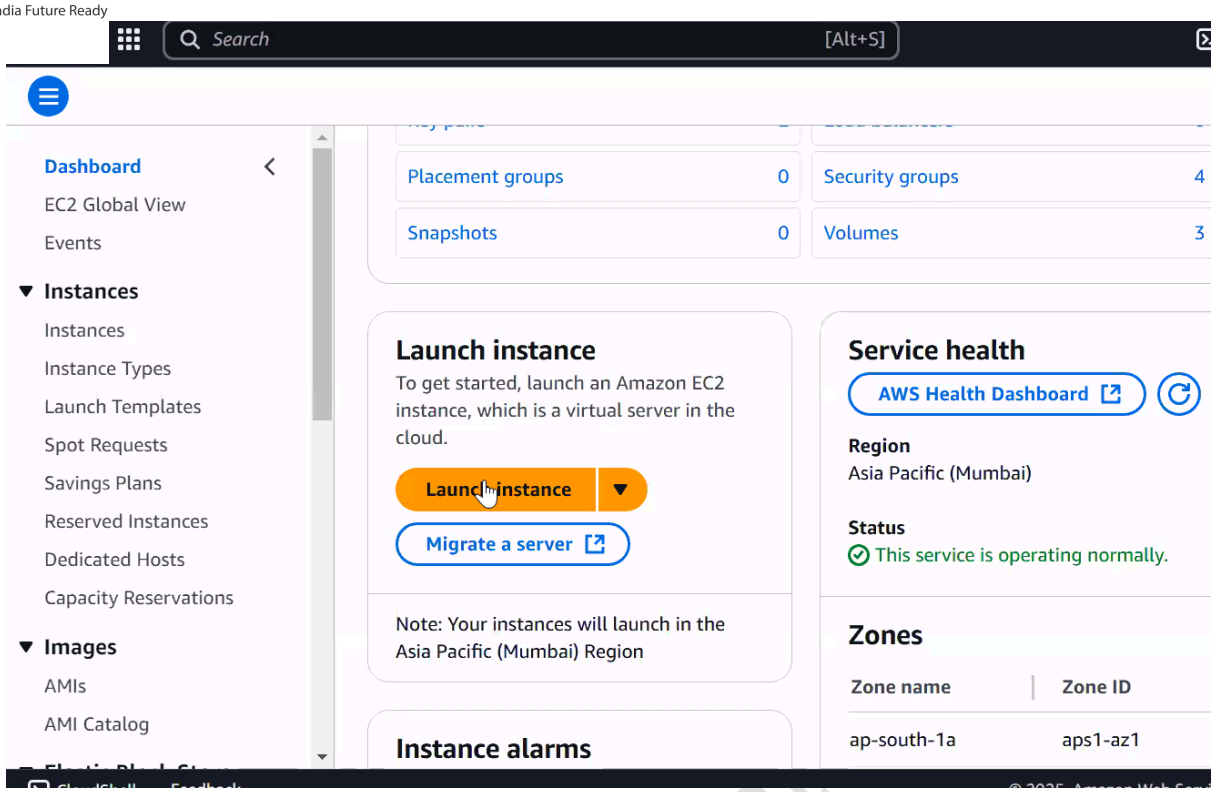
- There are four ways to install an operating system:
- **Bare Metal**: Installed directly on hardware for high performance.
- **Virtual**: Installed in a virtual environment, such as VirtualBox or a hypervisor.
- **Container**: Uses container technology like Docker.
- **Cloud**: Leverages cloud computing.
- The choice of installation method depends on use cases. **Bare metal** is ideal for high performance but requires owning hardware, which can be expensive and inflexible. In contrast, **cloud computing** allows renting hardware on demand, making it cost-effective and scalable.
- Cloud computing operates on a **pay-as-you-go** model. AWS (Amazon Web Services) is a leading provider, with services available in regions worldwide. For disaster recovery, AWS offers **Availability Zones (AZs)**—data centers within a region, spaced over 100 km apart. India has two regions: Mumbai and Hyderabad, with three AZs in Mumbai.
- AWS serves clients like Netflix, Hotstar, NASA, banks, and hospitals. These clients rely on multiple AZs to ensure uptime and disaster recovery.
- **Latency** and **cost** influence the choice of region. Applications hosted closer to users experience lower latency. Conversely, hosting in a region with lower costs can save money if latency is not critical.
- **Instances**: Virtual servers in the cloud, managed remotely.
- **Multi-Cloud**: Using services from multiple cloud providers to optimize cost and performance.
- **Hybrid Cloud**: Combining private and public clouds for sensitive data.
- Tools like **Terraform** automate multi-cloud management, while **Kubernetes** helps manage applications across clouds. AWS offers over 200 services, including EC2 for virtual servers, enabling remote management of resources like RAM and CPU (computing devices).



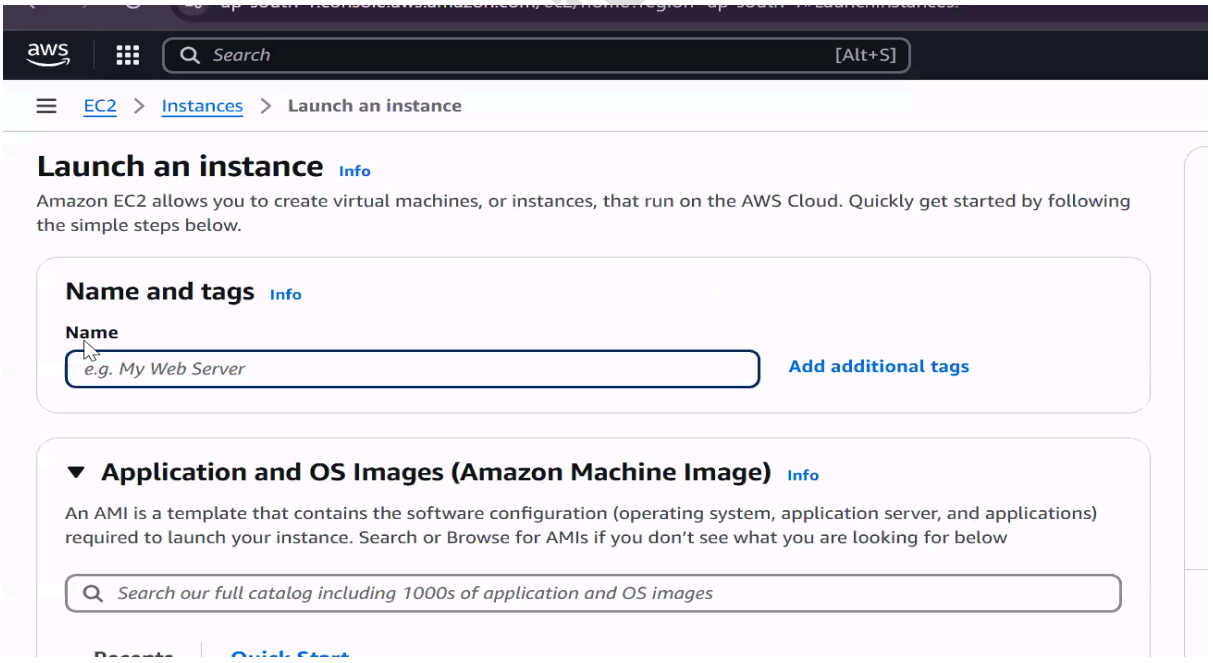

- You are using the service, but you must always specify the region in which you want to use it.



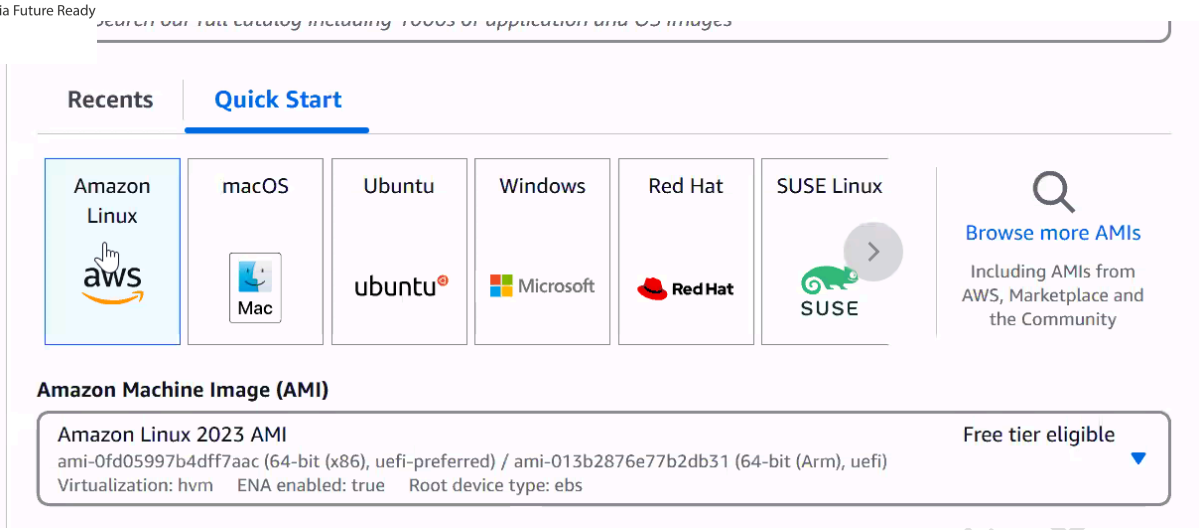

- Each region has its own ID, such as **ap-south-1** for Mumbai.
- Click on 'Launch Instance' to start your own server, which means setting up an operating system in the cloud.



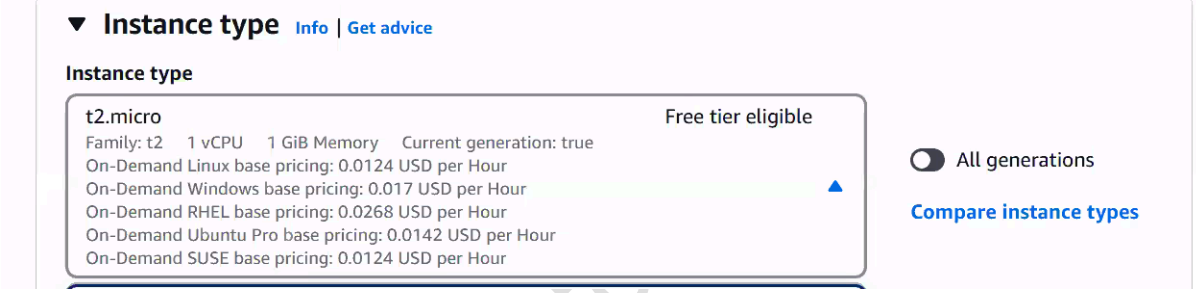

- They also give you the way throw which we can increase the resources or decrease the resources.
- Give os name here.



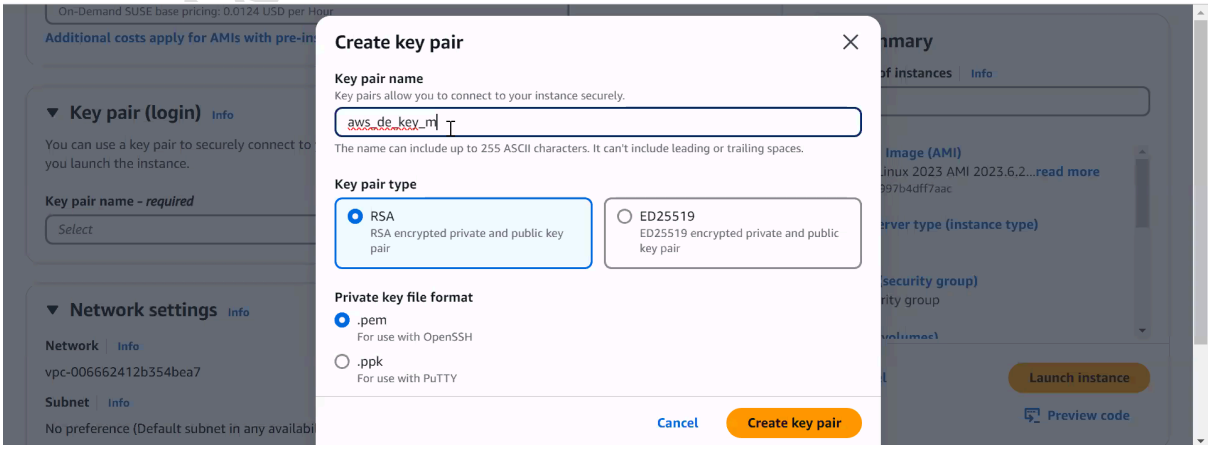

- In AWS the give you the catalog of os like we have bootable dvd or iso. but in aws this are known as AMI (amazon machine image)
- Select the iso which you need to install.




- They give you instance type in which you select how much ram and cpu you need.

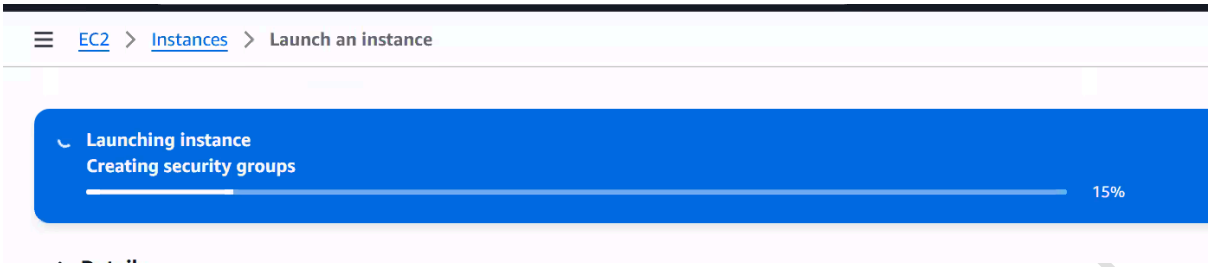



- And for every different hardware they have the own differnent codes like we have t2.micro instance type.
- For login we need to password so for that we have key pairs so create a keypair and select it.

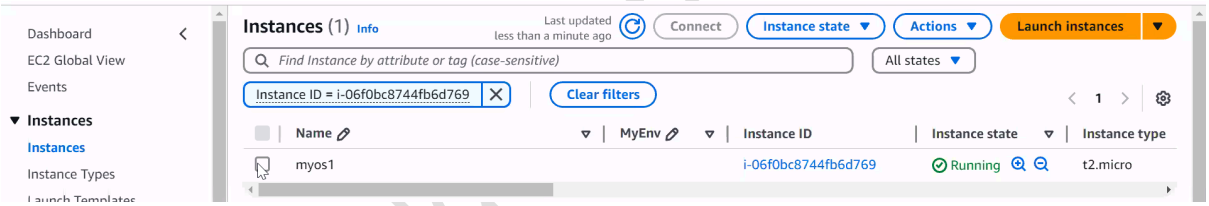



- This password are known as private key.

- Then click on launch instance.




- The way we set up this process involves manual steps. For example, on a Hotstar server, a large number of people join every minute, but we humans are relatively slow.
- To address this, we need to automate the process, and for automation, we use programming languages. Currently, we are using Python.
- Every operating system in AWS provides you with an instance ID. To connect to your instance, there are multiple ways. One common method is to select the instance and click on 'Connect'




- And then you will be the connected and you can run the linux command and do all the things there.

- In this scenario, we launch manually, but in the real world, automation is essential for managing the cloud. The principle of DevOps emphasizes agility and speed, but humans can be slow and prone to delays.
- In today's fast-paced world, having an idea is not enough—you also need to be agile. The company that can launch its idea the fastest gains a significant advantage. This focus on rapid deployment is known as "time to market," which is achievable through the DevOps approach, relying heavily on automation.
- Another reason for using cloud platforms is that they eliminate the need to manage infrastructure manually. This allows companies to concentrate on new ideas, innovations, and their customers instead of infrastructure management.
- For automation, our instance will be launched in EC2. There are multiple automation methods available in the market. For instance:

1. **Imperative Languages**: In these languages, you define every step explicitly—what to do and how to do it. Examples include Python and Shell scripting.
2. **Declarative Languages**: In these languages, you define what you want to achieve without specifying how to achieve it. Examples include Ansible (for managing Linux systems) and Terraform (for automating cloud infrastructure).

- To begin, we will use Python. To work with Python, you need an IDE.
- This will open the Jupyter Notebook IDE.



```
C:\Users\Vimal Daga>cd Documents

C:\Users\Vimal Daga\Documents>mkdir free_deloitte_training

C:\Users\Vimal Daga\Documents>cd free_deloitte_training

C:\Users\Vimal Daga\Documents\free_deloitte_training>jupyter notebook
```

```
C:\Users\Vimal Daga\Documents\free_deloitte_training>jupyter notebook
[I 2025-01-02 17:23:22.893 ServerApp] Extension package jupyter_lsp took 0.4524s to import
[W 2025-01-02 17:23:22.893 ServerApp] A `_jupyter_server_extension_points` function was not found
in jupyter_lsp. Instead, a `_jupyter_server_extension_paths` function was found and will be used f
or now. This function name will be deprecated in future releases of Jupyter Server.
[I 2025-01-02 17:23:23.131 ServerApp] Extension package jupyter_server_terminals took 0.2365s to i
mport
[W 2025-01-02 17:23:27.341 ServerApp] A `_jupyter_server_extension_points` function was not found
in notebook_shim. Instead, a `_jupyter_server_extension_paths` function was found and will be used
 for now. This function name will be deprecated in future releases of Jupyter Server.
```

- In Python if you want to print something then we have print function.

```
[1]: print("Vimal Daga")

     Vimal Daga
```

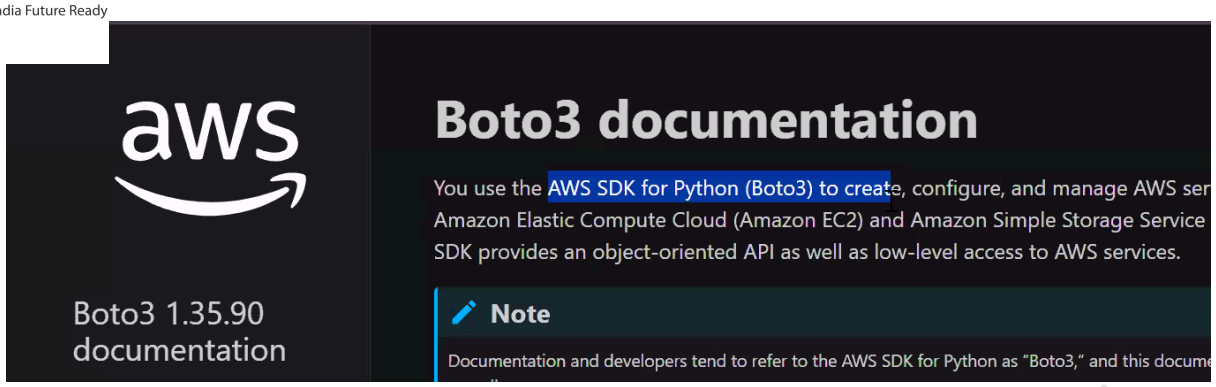
- If you want to use any data again and again you can create variable and print the value of variable enter variable name without double quotes.

```
[2]: x = 5

[4]: print(x)

     5
```

- when you launch the instance we always select a region so we are using the Mumbai region.
- We click on launch instance then we select the AMI from which AMI we have to launch the os.
- Then we select the types and tell how much instance you want.
- Now for convert this into python code

```
auto python to launch Linux os/ server in aws cloud - ec2
-----------------------------------------------------------

region = Mumbai

launvh instance function

os image / AMI = amazon Linux

instance type : t2.micro

count = 1
```

- To automate the aws cloud using python then we have boto3 library available.




- To install this library we need to open the cmd and run the pip install boto3 command.

```
C:\Users\Vimal Daga>pip install boto3
```

- pip command help you to install Python library and boto3 helps you to automate the aws cloud with Python.
- If you want to do same work again and again for that we can create function and function come from module and for launch the instance we have function and its come from boto3 module so we have to import boto3.

```
[5]: import boto3
```

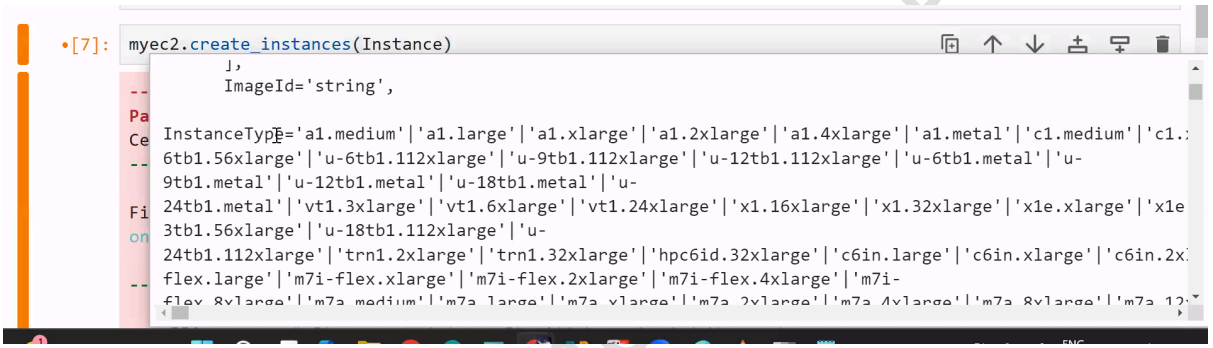
- One of the function is resources and you can store that function in variable.

```
[5]: import boto3

[6]: myec2 = boto3.resource("ec2")
```

- If you want to launch the instance the you can use variable name and enter . and press tab 2 times so it will print all the function stored in that variable and on of the function is create instance.

```
[5]: import boto3

[6]: myec2 = boto3.resource("ec2")

[ ]: myec2.create_instances()
```

- If you run this line, it will fail because you need to specify the instance and provide all the required information. When you pass the information to the function, these are known as arguments. In Jupyter, you can use **Shift + Tab** to open the help menu for the function and see all the parameters it supports




- One of the keywords is **instance type**, and you can specify the instance type you want to use. For example, if you want to use `t2.micro`, you will need the corresponding AMI. For that, we use the **image ID** keyword.
- In the cloud world, names are primarily used for identification, but internally, IDs are used. Therefore, we need the ID of the AMI.

```
[7]: myec2.create_instances(
        InstanceType="t2.micro",
        ImageId="ami-0fd05997b4dff7aac")
```

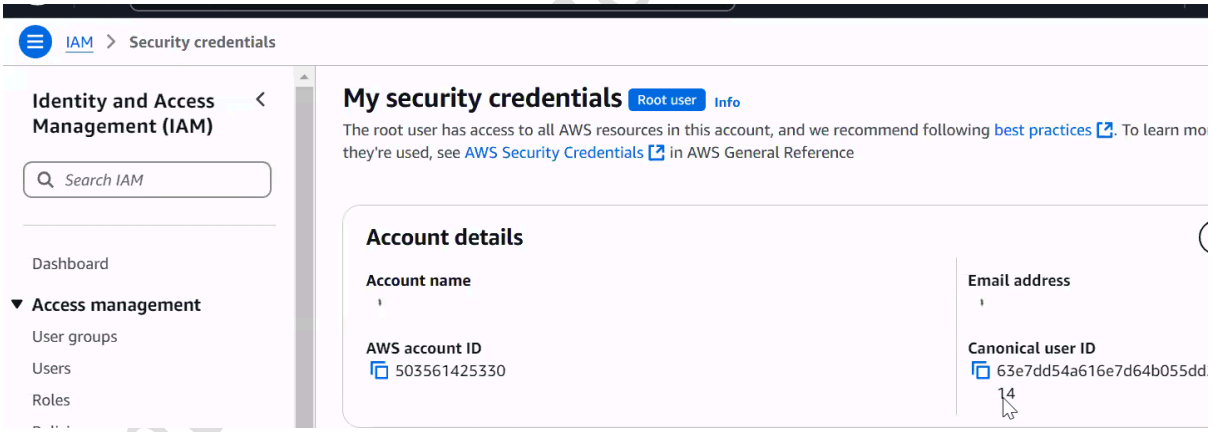
- We also have to put the count how much count you want so you have to tell minimum count and maximum count.

```
[8]: myec2.create_instances(
        InstanceType="t2.micro",
        ImageId="ami-0fd05997b4dff7aac",
        MinCount=1,
        MaxCount=1
)
```

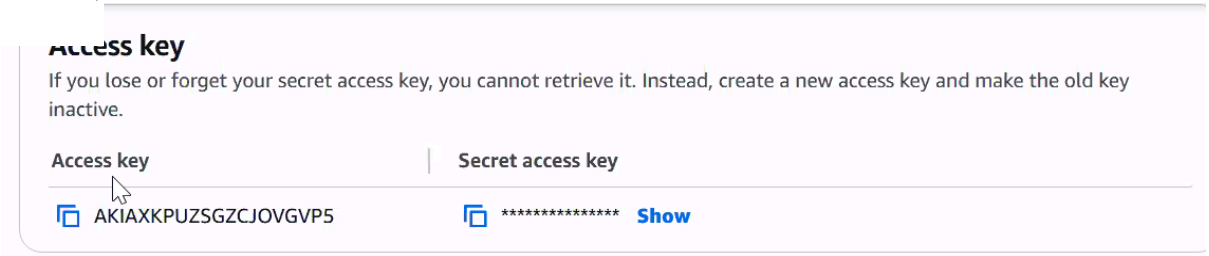
- Now if you run this it will fail because of authentication.

```
1025     return parsed_response

ClientError: An error occurred (AuthFailure) when calling the RunInstances operation: AWS was not able
to validate the provided access credentials
```

- The minimum requirement is to log in to the cloud.
- If a user accesses the platform, they use a username and password. However, if a program needs to access the account, a key must be provided.
- To create a key, open your AWS cloud account and navigate to the **Security Credentials** section




- Here you find the tab of access keys and create a keys.




- Then pass the key in the function.

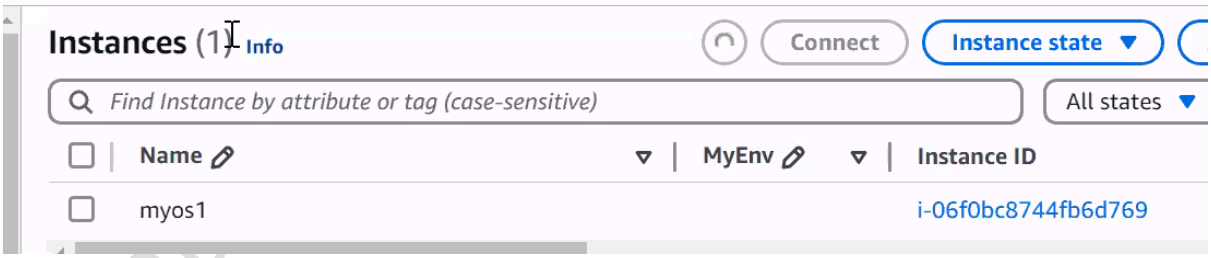
```
[6]: myec2 = boto3.resource("ec2", "AKIAXKPUZSGZCJOVGVP5", "3LMiKMUEfrCgCre2MEQ2hY2TtaH1tTdtWj47Q7iK")
```

- But it will again fail because they are not able to understand which is the secret key and which is the access key so you have to pass the keywords there and we also have to pass the name of the region and we need to use the ID of that region.

```
[10]: myec2 = boto3.resource("ec2" region_name="ap-south-1" , aws_access_key_id="AKIAXKPUZSGZCJOVGVP5", aws_s
```

```
[10]: myec2 = boto3.resource(
            service_name="ec2",
            region_name="ap-south-1" ,
            aws_access_key_id="AKIAXKPUZSGZCJOVGVP5",
            aws_secret_access_key="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"
)
```

- Now if you check in ec2 we have only one instance is running.




- But when you run the Python code it will launch the instance in AWS cloud.

```
[13]: myec2.create_instances(
            InstanceType="t2.micro",
            ImageId="ami-0fd05997b4dff7aac",
            MinCount=1,
            MaxCount=1
)
```

```
[13]: [ec2.Instance(id='i-0caa5027501f3b14c')]
```

- Now you see in aws you see the instance is launched in ec2.
- And mostly in programming is a good practice to create a function in python we use the def function and then the function name and column and for calling we use a name to call the function and colon.

```
[14]: def lw():
          print("hi")

[15]: lw()

      hi
```

- In Python we use equal space to tell this code is a block of code of this function.
- And in the function after its run it give some output means return something so for that we use return keyword.
- And whenever you run this function it will launch the instance and return the instance id.

```
[16]: def lwoslaunch():
          myid = myec2.create_instances(
                      InstanceType="t2.micro",
                      ImageId="ami-0fd05997b4dff7aac",
                      MinCount=1,
                      MaxCount=1
          )

          return myid

[18]: lwoslaunch()

[18]: [ec2.Instance(id='i-05e5f917530357880')]
```

- We are running this function manually. However, we can automate it using a monitoring tool like Prometheus. For visualization, we can use Grafana to create graphs and dashboards. Whenever the server load increases, this function will automatically execute.

- If you only want to retrieve the ID of the operating system, you can store the function's output in a variable.

```
[20]: myos = lwoslaunch()
```

- Once the output is stored in a variable, we need to retrieve the ID from it. Since it is a list, we can use index 0 and then access the ID property. This will print the ID.

```
[20]: myos = lwoslaunch()

[23]: myos[0].id

[23]: 'i-0bb94364e3379f7fb'
```

- Then we can create a list and use the .append funtion to append the id into that list.

```
[23]: myos[0].id

[23]: 'i-0bb94364e3379f7fb'

[25]: myoslist = []

[26]: myoslist.append( myos[0].id )
```

- So when every you launch the instance the information is gone save in list.

```
[28]: myos = lwoslaunch()

[29]: myoslist.append( myos[0].id )

[30]: myoslist

[30]: ['i-0bb94364e3379f7fb', 'i-0d259e8c9a5a2ef70']
```

- We can use length function to check how many value are stored there.

```
[31]: myoslist

[31]: ['i-0bb94364e3379f7fb', 'i-0d259e8c9a5a2ef70']

[32]: len(myoslist)

[32]: 2
```

- As our traffic increases, we add more operating systems. This is known as **scale-out**, or **horizontal scaling**. Conversely, if we decrease the number of operating systems, it is called **scale-in**.
- If you want to delete or terminate an instance, you need to use **services** in Boto3. We use **resources**, but there is also another function available called **client**, similar to resources.

```
[37]: myec1 = boto3.client("ec2")
```

- If you want to delete an instance, there is a **terminate_instance** function available in the **client** function.

```
[ ]: myec1.terminate_instances()
```

- And this function take the instance id in list format.

```
[38]: myec1.terminate_instances(InstanceIds=["i-06f0bc8744fb6d769"])
```

- It will fail because in client function we also have to pass the keys.

```
myec1.terminate_instances(InstanceIds=["i-06f0bc8744fb6d769"])
```

```
{'TerminatingInstances': [{'InstanceId': 'i-06f0bc8744fb6d769',
   'CurrentState': {'Code': 32, 'Name': 'shutting-down'},
   'PreviousState': {'Code': 16, 'Name': 'running'}}],
 'ResponseMetadata': {'RequestId': '2377c935-3921-40e1-a56d-63b3289f7199',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '2377c935-3921-40e1-a56d-63b3289f7199',
   'cache-control': 'no-cache, no-store',
   'strict-transport-security': 'max-age=31536000; includeSubDomains',
   'content-type': 'text/xml;charset=UTF-8',
   'content-length': '426',
```

- Then you see the instance will get shut down and you can create the function for the shutdown the os we can pass the id of the instance in input and terminate the instance.

```
[42]: def lwosterminate(n):
          myec1.terminate_instances(InstanceIds=[n])
```

- We are going to use the list for taking the instance id and what we want from any list we want to use the id and remove it then we can use pop function.

```
[48]: los = myoslist.pop()

[49]: los

[49]: 'i-0dbf91881816102dd'
```

- And store the id in variable and pass this variable in terminate function.