Create An AWS EC2 Instance With Python

## Install AWS CLI and Python Boto3 Library

Before we can get started, you'll need to install **Boto3 library in Python**and the **AWS Command Line Interface (CLI)tool**using **'pip'**which is a package management system written in Python used to install and manage packages that can contain code libraries and dependent files.

Boto3 is the AWS SDK for Python, which provides Object-based APIs and low-level direct access to AWS services like EC2. AWS CLI is a command line tool written in Python that introduces efficient use cases to manage AWS services with a set of very simple commands.

Using 'pip' run the following command to install the AWS CLI and Python's Boto3 library on your machine:

pip install awscli boto3

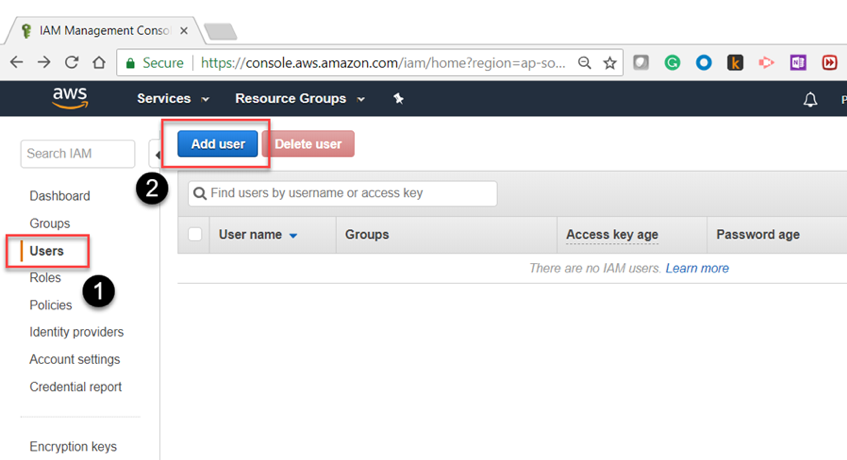
## prateek-pip-install-awscli-boto3Create a User and get AWS Access ID and Secret Key

Now that we've installed the AWS CLI and Boto3, its time to create your user credentials on the AWS console, so that AWS services can be access programmatically. Follow these steps to create your user credentials:

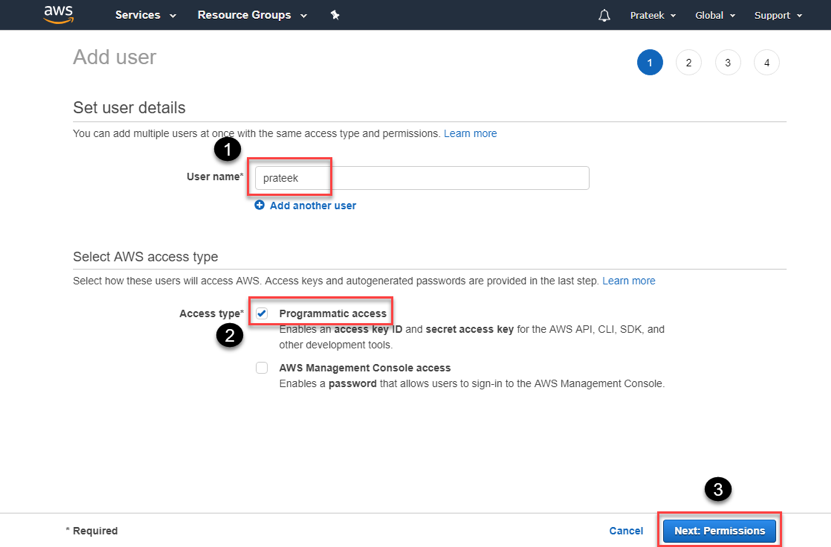
1. Launch the Identity and Access Management console (IAM) in AWS.

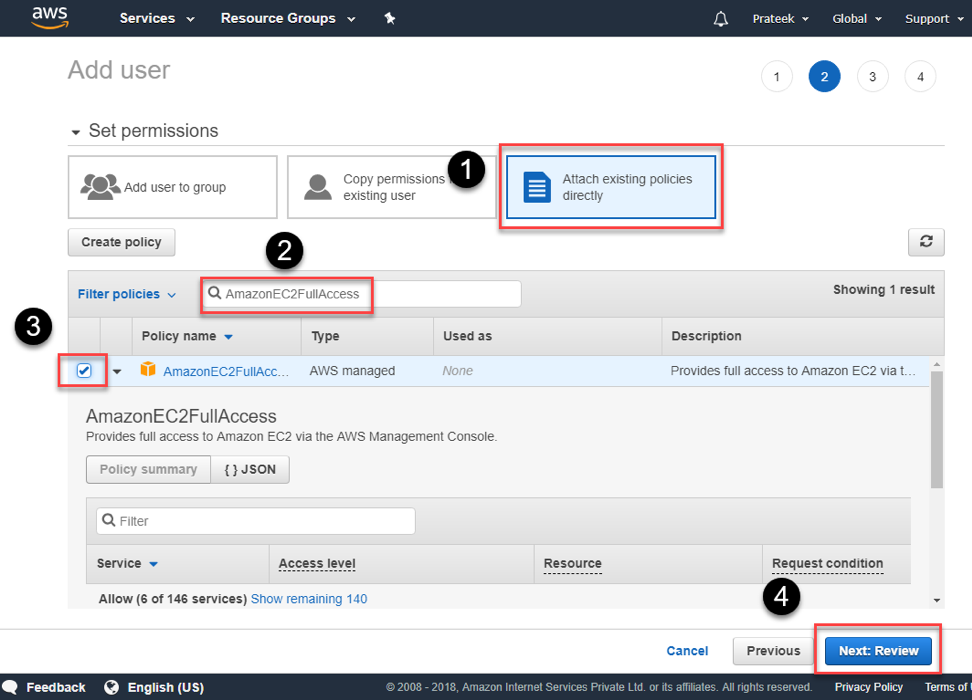
2. Click **Users**on the navigation menu on the left of the screen.

3. In the popup window, click on **Add User.**

****

4. In the new window, provide a user name and choose the 'Programmatic Access' access type, then click next.

5. to set the permissions, choose **'Attach Existing Policies Directly'**and in the Policy Filter type **'AmazonEC2FullAccess'**, you can choose any permission level, but in this example I'll click on the checkbox next to **'AmazonEC2FullAccess'**and then click the 'next' button.

6. Finally, review the user and permission levels, and click on the **'Create User'**button.

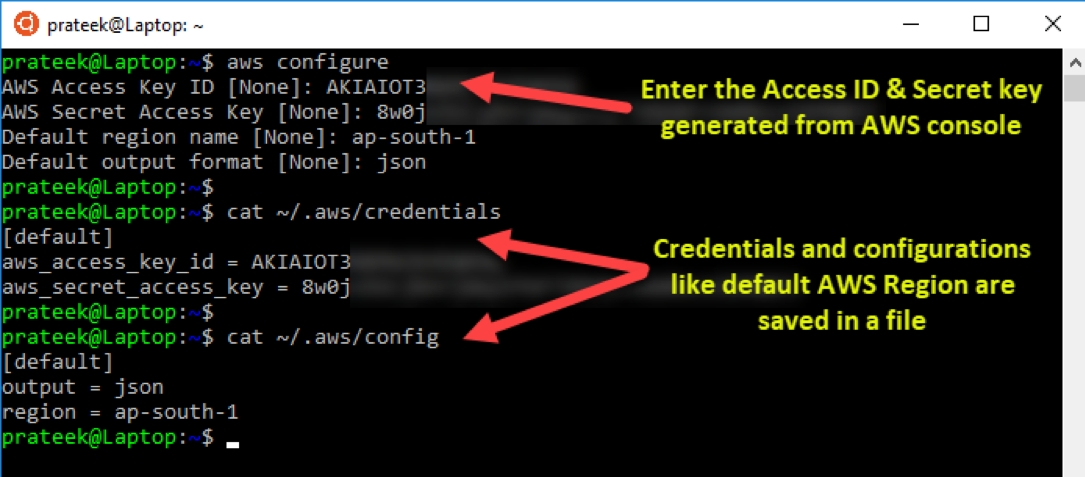
7. The next page will show your keys, as shown below. These are only available once, so it its a good idea to download and save then safely in a secure location.

## prateek-add-user-aws-iam-4Configure AWS Credentials Locally

After creating the user and obtaining the credentials (Access ID and Secret key), we can now configure our Python scripting environment with this credential in order to manage EC2. Use the AWS CI tool to configure these credentials by running the following command from a Bash terminal:

aws configure

It will prompt you to provide the Access Key ID, Secret Key, Default AWS region, and output format. Once those are provided, credentials are saved in a local file at path **~/.aws/credentials**and other configurations like region are stored in **~/.aws/config**file as demonstrated in the following example.

Now that we've configured our credentials, let's test if these credentials work well with AWS CLI tools. To do that, run the following command from a Bash shell:

aws ec2 describe-instances

This should return details of any EC2 instance running on AWS in JSON format if the credentials are good. Otherwise, an error is thrown, which means the credentials do not work.

## Create Key Pair for EC2 Instance

Before we can jump into how to create EC2 instances, it's important to understand how to create a keypair for EC2 instances, so that they can be accessed later, once the virtual machines are launched programmatically using Python.

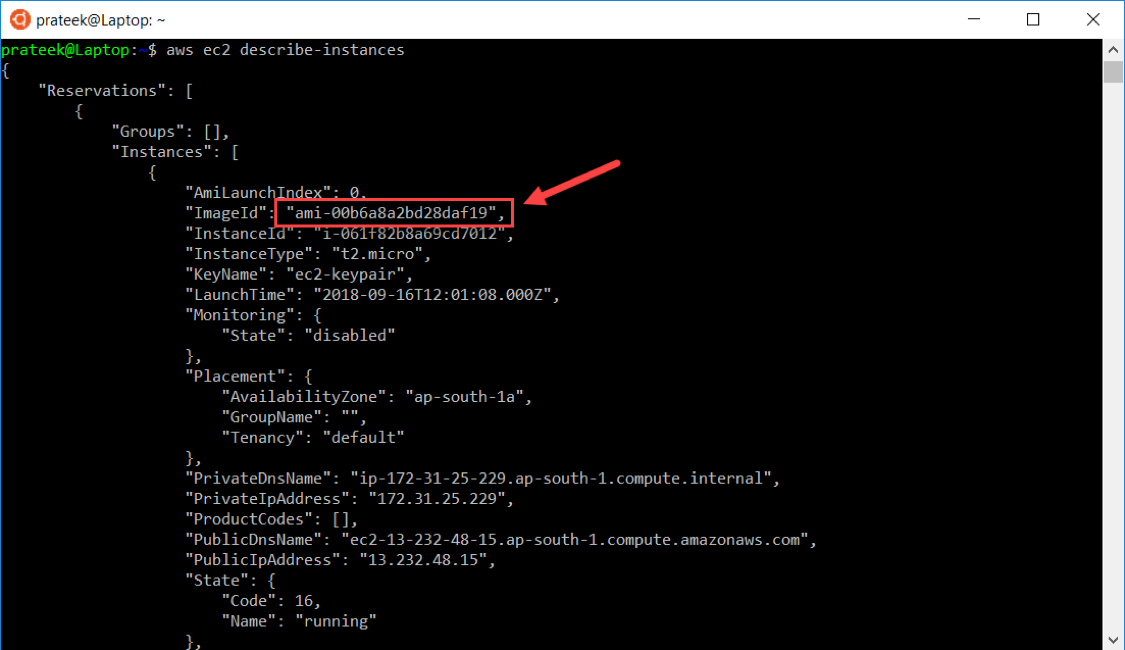
**import** boto3  
ec2 = boto3.resource('ec2')  
  
# create a file to store the key locally  
outfile = open(**'ec2-keypair.pem'**,**'w'**)  
# call the boto ec2 function to create a key pairkey\_pair = ec2.create\_key\_pair(KeyName=**'ec2-keypair'**)# capture the key and store it in a fileKeyPairOut = str(key\_pair.key\_material)print(KeyPairOut)outfile.write(KeyPairOut)

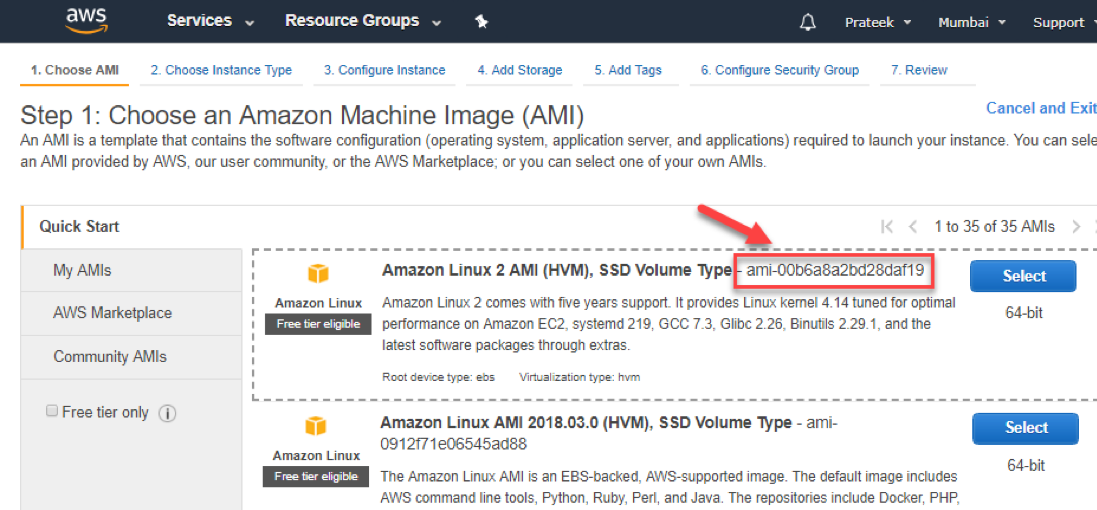
 The above program not only creates a key pair in AWS, it also captures and stores it on your local machine. You can use this key pair to SSH into the virtual machines later. Please make sure to change the mode of the key pair file to read-only using the following command in bash terminal, otherwise it will be denied access.

chmod 400 ec2-keypair.pem

## prateek-configure-aws-credentials-locally-2Create a New EC2 Instance

In one of our previous examples, when we listed the EC2 instances, you could also see the Amazon Machine Image (AMI) ID, which looks like **ami-00b6a8a2bd28daf19**, this is important information, and is required to create a new instance programmatically using Python.

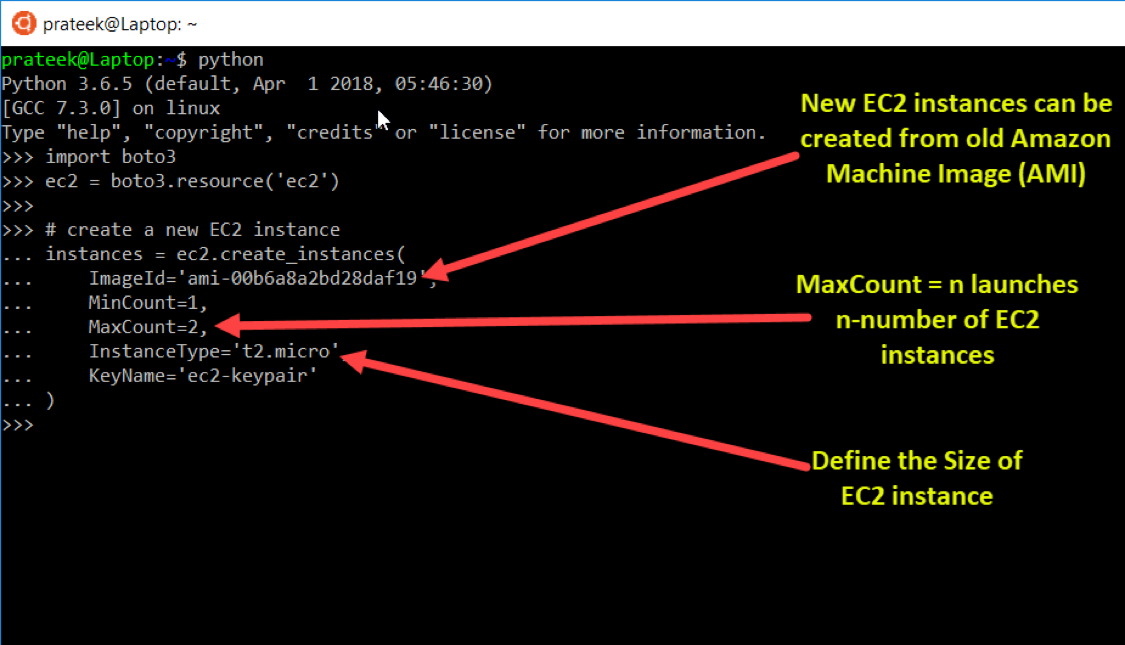
You can also obtain the AMI ID from the AWS console in your browser when you launch and instance.

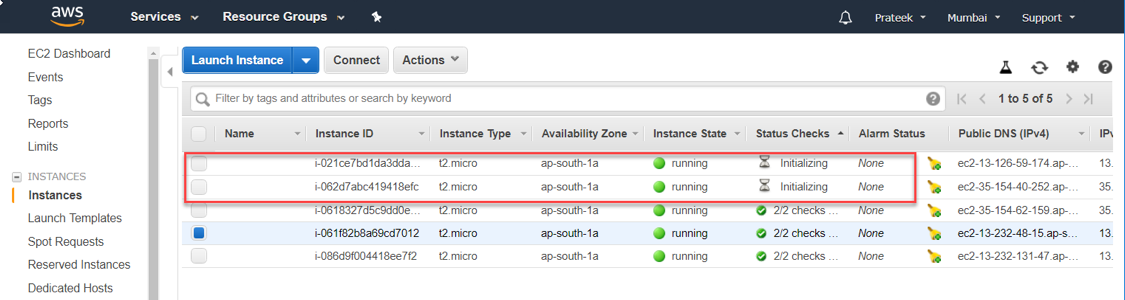
Once we have this information, it's pretty straight-forward to script this in Python.

**import** boto3  
ec2 = boto3.resource(**'ec2'**)  
  
# create a new EC2 instance  
instances = ec2.create\_instances(  
     ImageId=**'ami-00b6a8a2bd28daf19'**,  
     MinCount=1,  
     MaxCount=2,  
     InstanceType=**'t2.micro'**,  
     KeyName=**'ec2-keypair'** )

In the above code sample:

* **ImageID** specifies the **Amazon Machine Image (AMI) ID** of the instance we want to create. In this case we've chose an image of a Windows Server 2016.
* **MinCount** and **MaxCount** are used to define the number of EC2 instances to launch. That means if MinCount=1 and MaxCount=3, then 3 instances will be launched.
* **InstanceType** is the size of the instance, like **t2.micro**, **t2.small**, or **m5.large**.
* **KeyName**defines the name of the key pair that will allow access to the instance. In our case we'll use the **'ec2-keypair'**we've created in AWS, and also have a local copy.

After running the above script, now when you go to your EC2 dashboard in AWS console, you'll observe new EC2 instances are being provisioned and are in initialization state, which is expected to complete in a few minutes. Once that is complete, your virtual machines are ready to be used.

So, to wrap things up, in this article we covered how to instal AWS CLI Tools and Python's Boto3 library, then created credentials to programmatically access AWS services, which was later configured in our local Python scripting environment. Once everything was setup, we created Python scripts to launch new EC2 instances with specific configurations.

import logging

import boto3

from botocore.exceptions import ClientError

def create\_ec2\_instance(image\_id, instance\_type, keypair\_name):

"""Provision and launch an EC2 instance

The method returns without waiting for the instance to reach

a running state.

:param image\_id: ID of AMI to launch, such as 'ami-XXXX'

:param instance\_type: string, such as 't2.micro'

:param keypair\_name: string, name of the key pair

:return Dictionary containing information about the instance. If error,

returns None.

"""

*# Provision and launch the EC2 instance*

ec2\_client = boto3.client('ec2')

try:

response = ec2\_client.run\_instances(ImageId=image\_id,

InstanceType=instance\_type,

KeyName=keypair\_name,

MinCount=1,

MaxCount=1)

except ClientError as e:

logging.error(e)

return None

return response['Instances'][0]

def main():

"""Exercise create\_ec2\_instance()"""

*# Assign these values before running the program*

image\_id = 'AMI\_IMAGE\_ID'

instance\_type = 'INSTANCE\_TYPE'

keypair\_name = 'KEYPAIR\_NAME'

*# Set up logging*

logging.basicConfig(level=logging.DEBUG,

format='%(levelname)s: %(asctime)s: %(message)s')

*# Provision and launch the EC2 instance*

instance\_info = create\_ec2\_instance(image\_id, instance\_type, keypair\_name)

if instance\_info is not None:

logging.info(f'Launched EC2 Instance {instance\_info["InstanceId"]}')

logging.info(f' VPC ID: {instance\_info["VpcId"]}')

logging.info(f' Private IP Address: {instance\_info["PrivateIpAddress"]}')

logging.info(f' Current State: {instance\_info["State"]["Name"]}')

if \_\_name\_\_ == '\_\_main\_\_':

main()