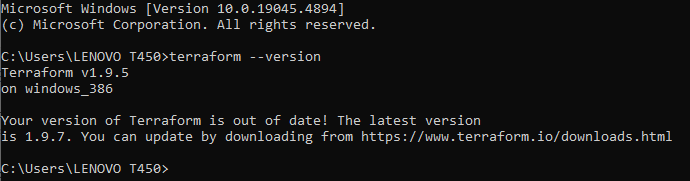
**DevOps project to automate infrastructure on AWS using Terraform and Lambda Function**

**Prerequisites:**

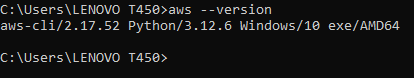
1. AWS account creation

<https://aws.amazon.com/console/>

1. Terraform installed



1. AWS CLI installed



1. Code Editor (VS Code)

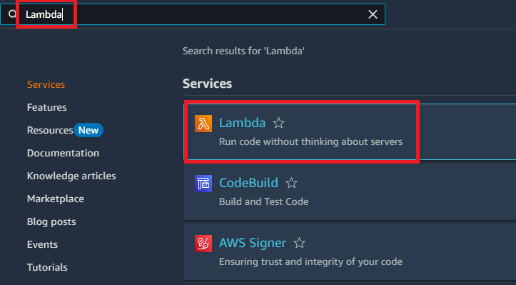
https://code.visualstudio.com/docs/?dv=win64user

**AWS Lambda:** Lambda is a serverless compute service provided by AWS. It allows you to run code without provisioning or managing servers.

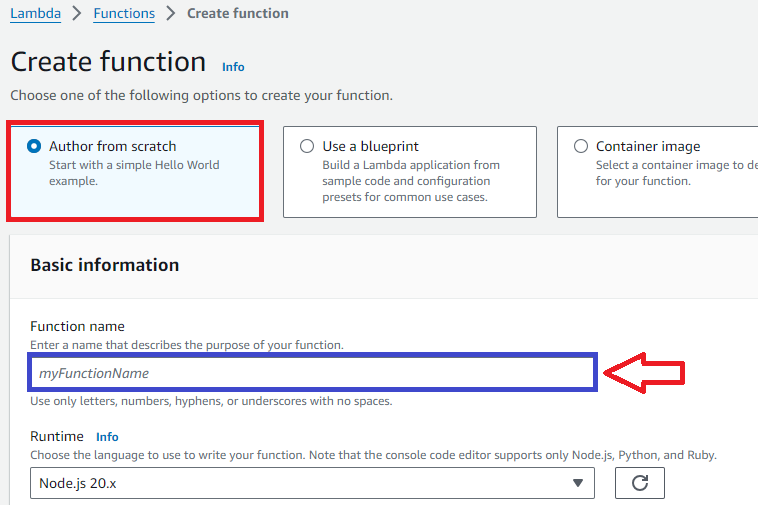
**How to create a Lambda Function Using AWS console?**

Here we can create lambda function in AWS console

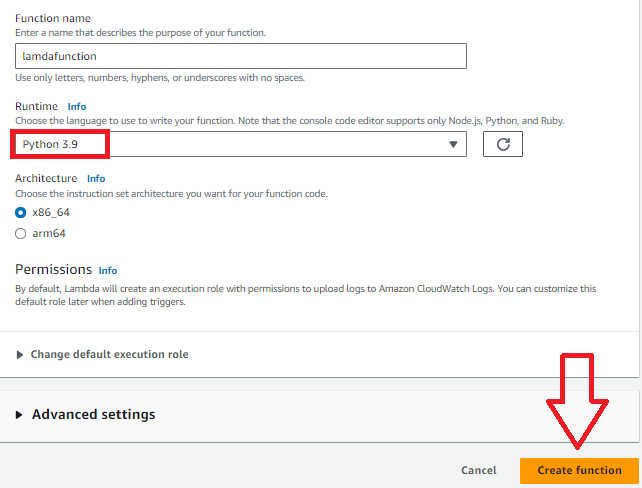
**Step 1:**  First we have to sign in to the console page and in search bar type lambda and search



Click on the Lambda then the lambda dashboard will be opened. Here we can see **“create function”.**  Click on the Create function, the below page will be opened.

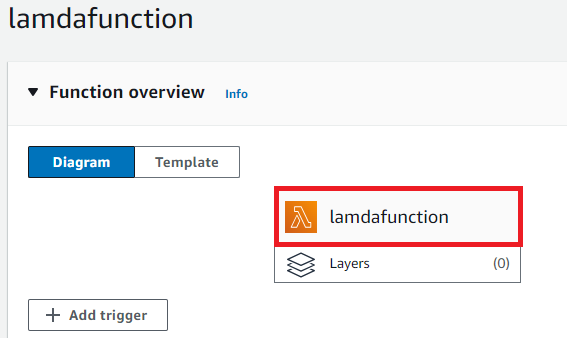


In above image firstly, we can select **Author from scratch**, after that give one name as function name. Ex: lambdafunction

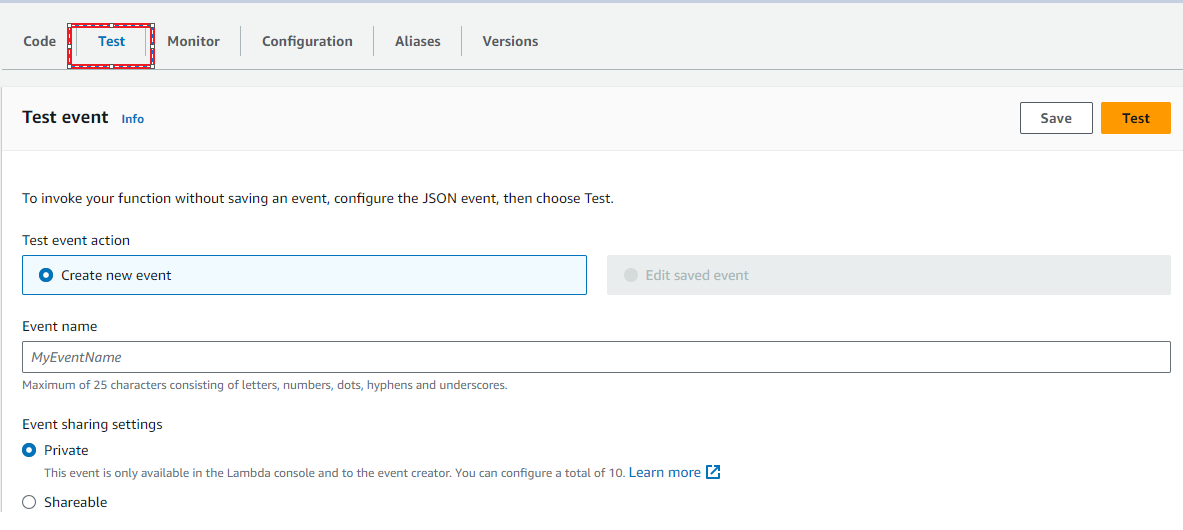


Here we can select runtime as **python 3.9 version** and click on the **Create function** button.

Here will see one function i.e. created by us like lambdafunction. See below image



In lambdafunction page below test option is there, click on that and here we can add event name and set private/shareable as event sharing settings and click on the save button.

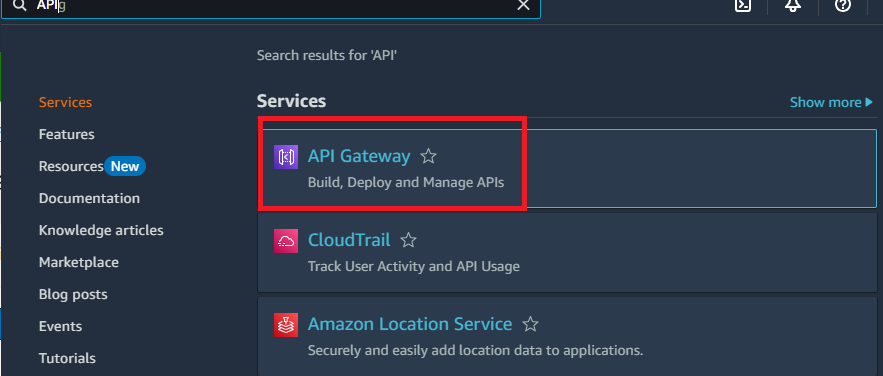


After saving the information click on the test button, here we can receive a message like below image.

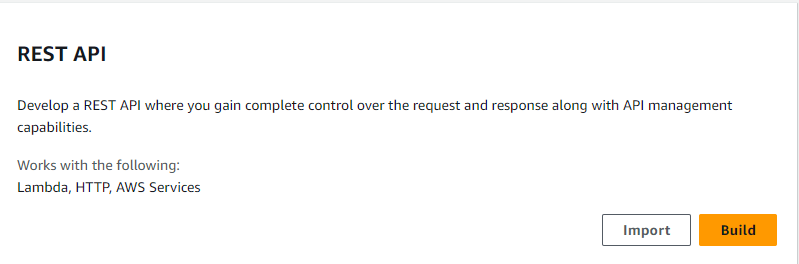


**API:** API Stands for **Application Programming Interface**. APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols.

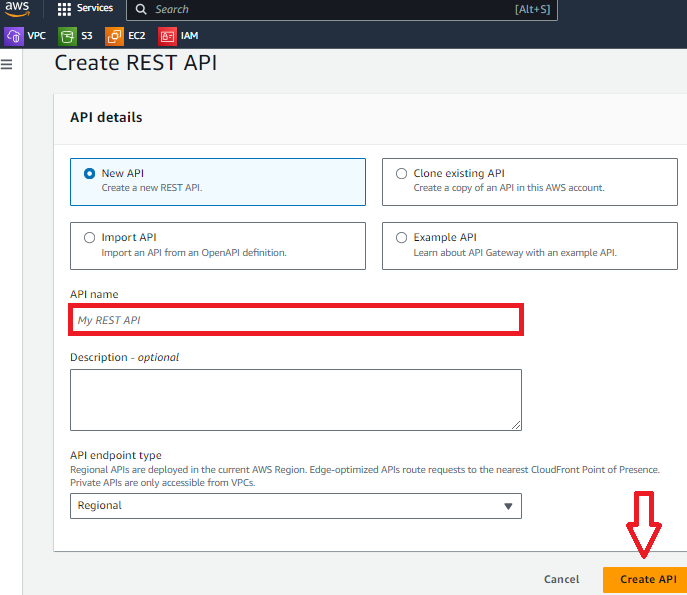
**Step 2:** Here we go to create an API using AWS, in search bar search “**API Gateway**” and click on that.



After click on the API Gateway here we can see some of the API, in that we can choose **Rest API** and click on the **build** button



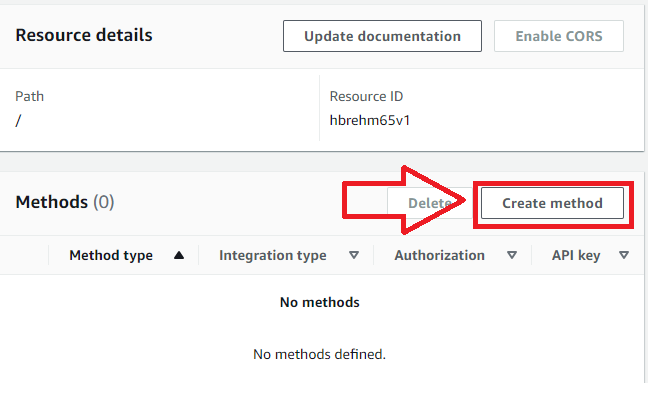
After click on the build, we choose New API in API details and here we can write a name for API name and click on **“Create API”** button



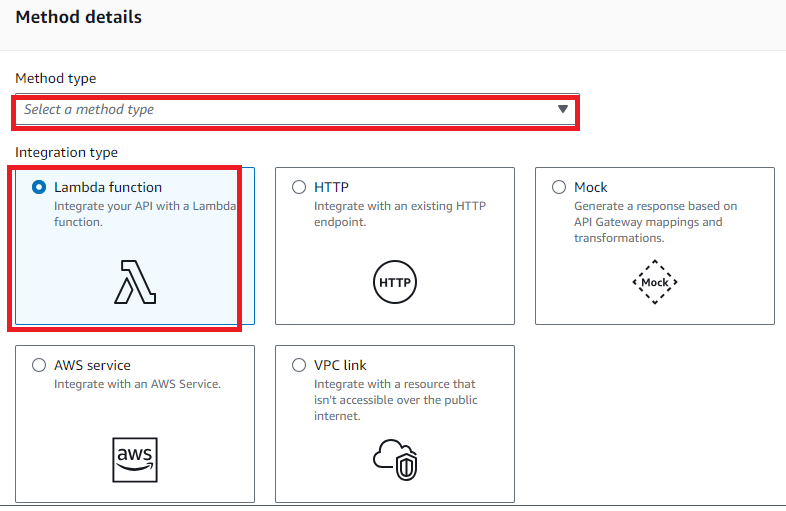
After click on Create API we have received a message like below image



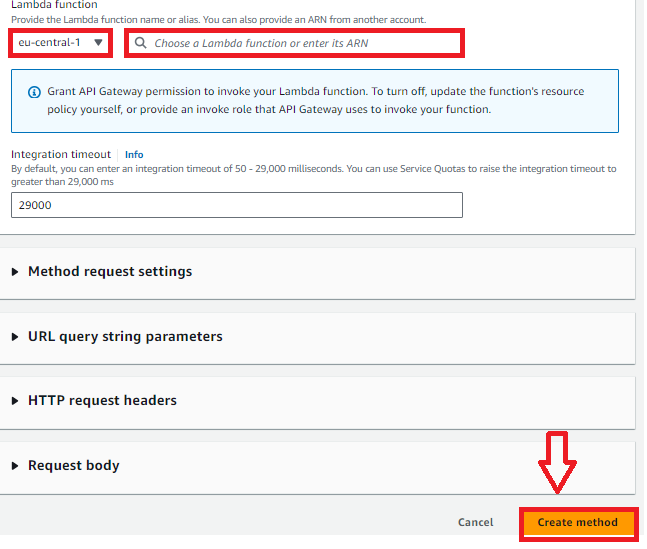
After create API we can create a new method so that click on **Create method** button



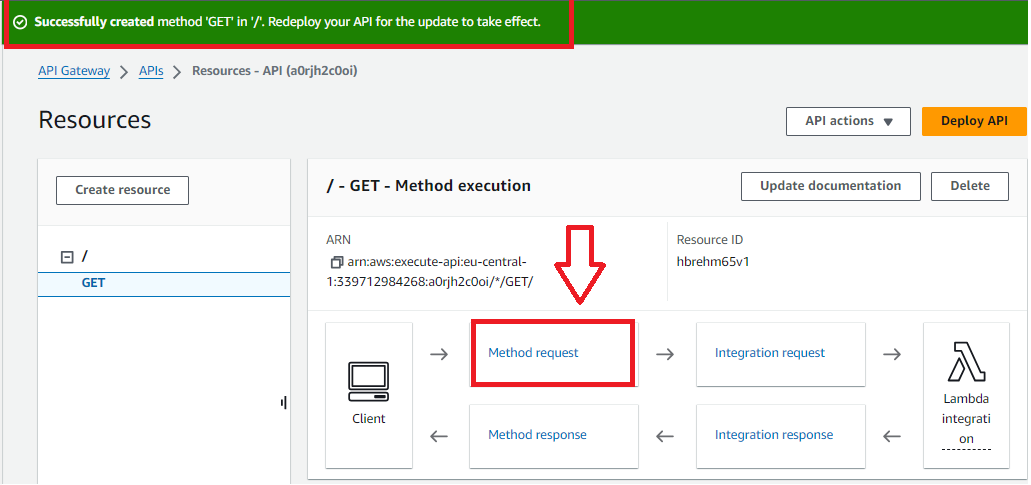
After click on Create method we can select “**GET”** as method type from dropdown and select lambda function as integration type



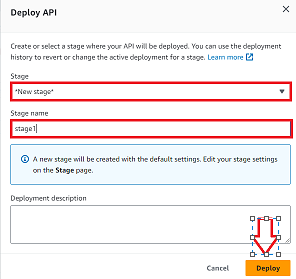
And in lambda function column you can select any region and click on choose a lambda function column, then click on **Create method** button**.**



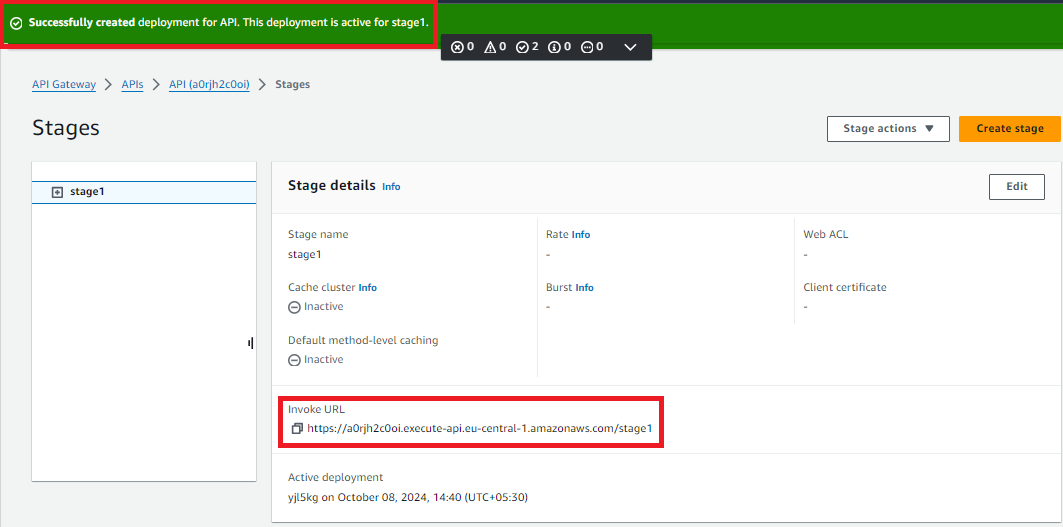
After click on create method button we received a message like below image and here click on **Method request**



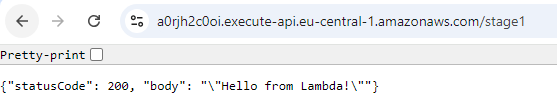
In this page we can see **Deploy API** button on that we can fill some information like new stage as stage column and stage name after that click on **Deploy** button



After deploy we can receive a message and we can get invoke URL as shown in below image



Copy that URL and paste it in new tab, then will get output like below image



Here is the output of lambda and creation of lambda function by using AWS console.

**Terraform:** Terraform is an open-source Infrastructure as a code (IAC) tool developed by HashiCorp that allows you to define and provision infrastructure using a declarative configuration language. When working with AWS, terraform helps you automate the process of setting up and managing your cloud resources.

**How to create a Lambda Function Using Terraform?**

We will be using terraform to provision the lambda function. You can find the Terraform code for the lambda function here

**Step 3:** Firstly, we have to create one folder and that folder can add in VS code.

In that folder we will also create a new file like main.tf. In main.tf file we can add our terraform code like below

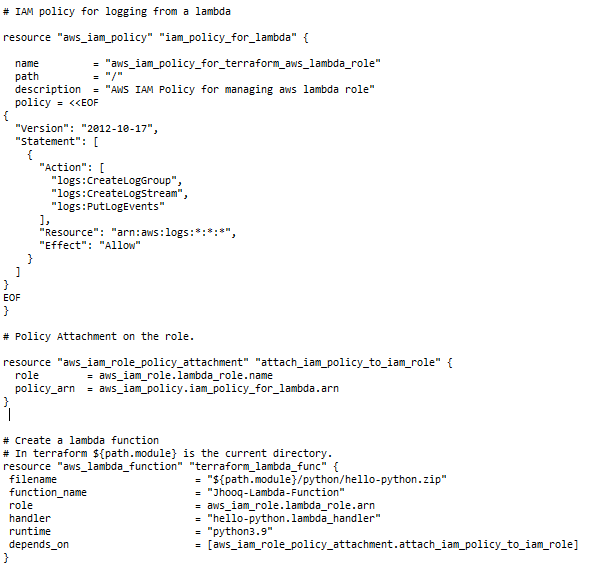
**IAM role:** An IAM role in AWS Identity and Access Management is an entity that allows trusted identities to perform actions in AWS. IAM roles are a security best practice because they provide temporary credentials that don’t need to be rotated

* IAM role



**IAM Policy:** IAM policies define permissions for an action regardless of the method that you use to perform the operation

* IAM policy attachment that will enable Lambda function to “**aws\_iam\_policy\_for\_terraform\_aws\_lambda\_role”.** This will allow the lambda function to list the access key details and send policies using logs.

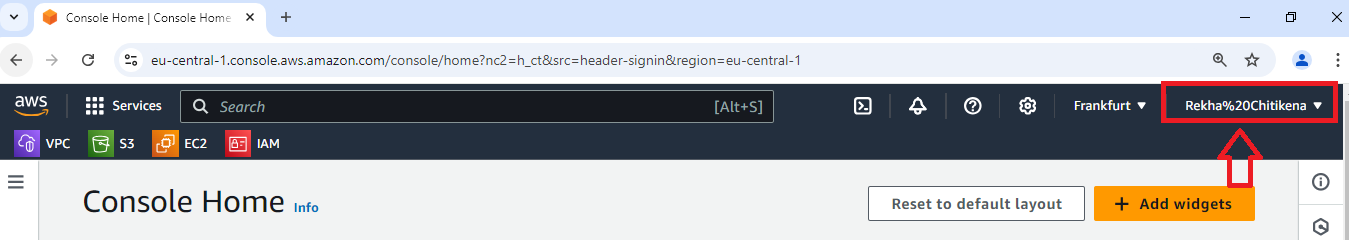


After write the terraform code we need to check whether the code is working fine or not, let’s run terraform commands. Make sure to connect aws with terraform using **“aws configure”** command

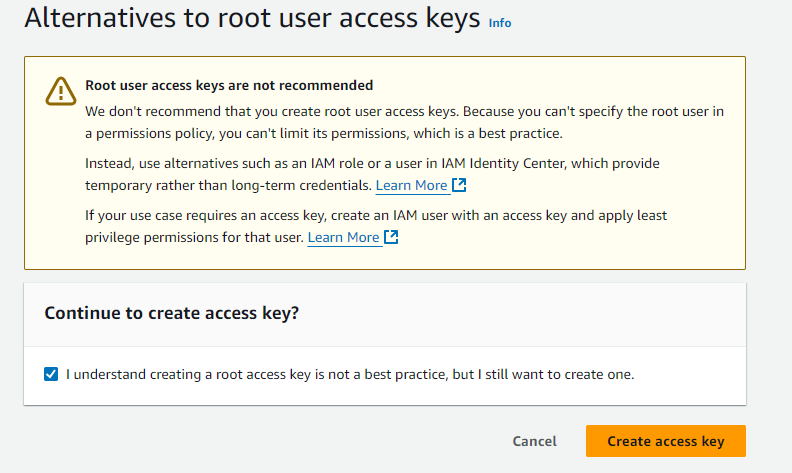
**aws configure:** aws configure command lists the profile, access key, secret key, and region configuration information used for the specified profile.

**How to create access key and secret key?**

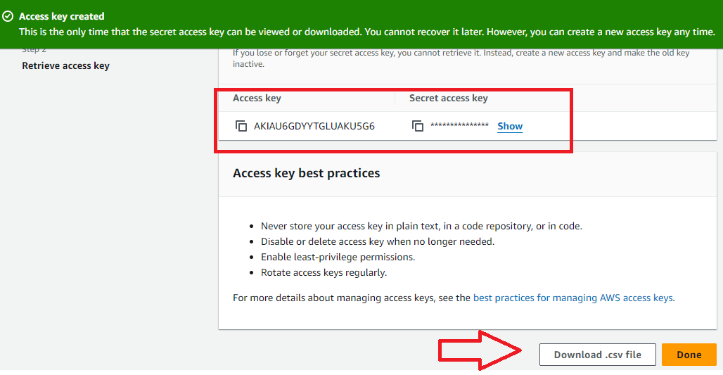
* Firstly, we go to AWS console, in that click on profile button as shown like below image.

****

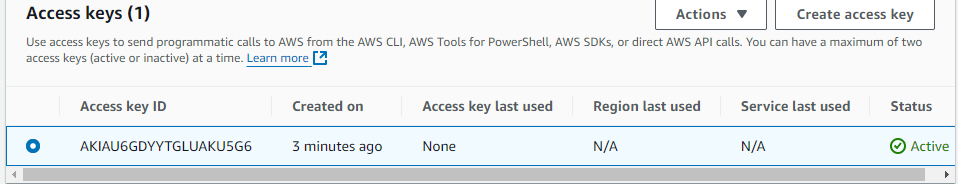
In that profile we can select security credentials, in that we can see **create access key** button, click on that. Here tick the checkbox to continue to create the access key.



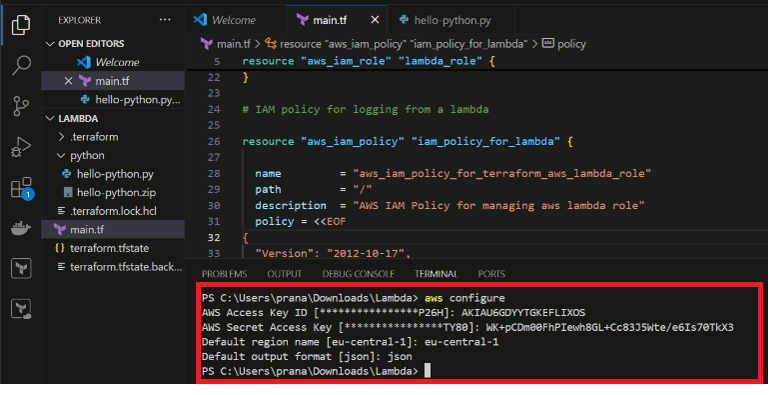
And click on Create access key, after click on the button we can see access key and secret key. And also download **.csv file**



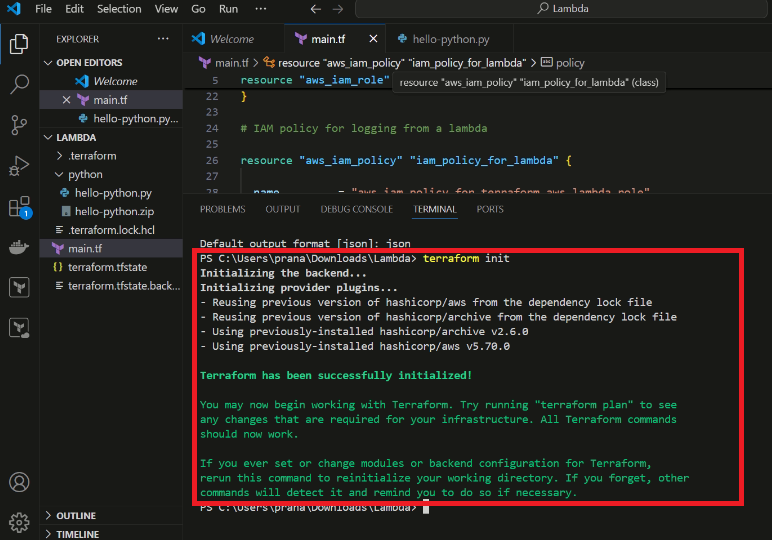
Here is the Access key



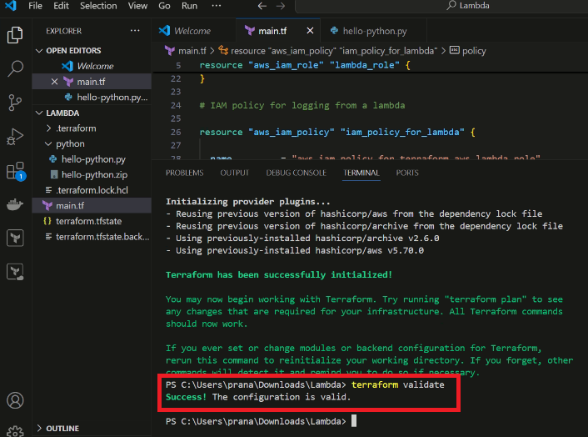
Here we connect terraform with aws, see below image



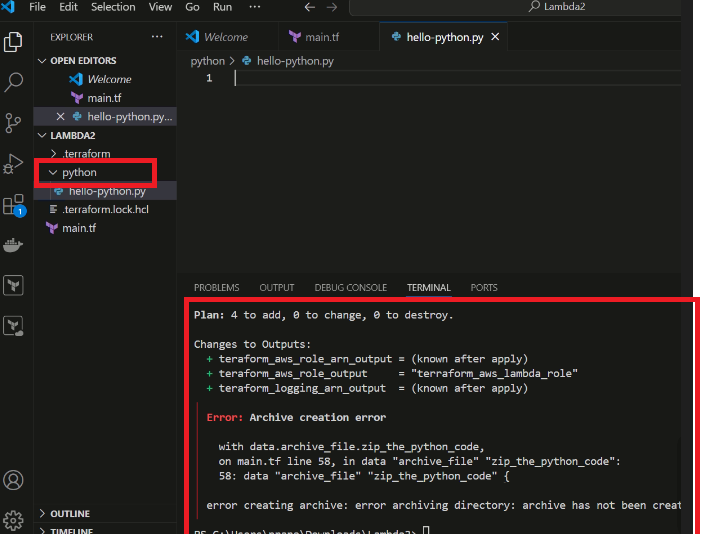
Now, we have to initialize terraform, use **“terraform init”** command which setups everything necessary for terraform to manage your infrastructure such as modules, plugins, backend config etc., as defined in your configuration files.



To check if your code is valid or not, use **“terraform validate”** command.

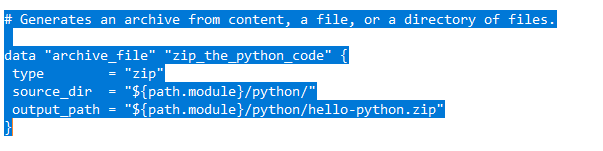


To run the code, we want to use **“terraform plan”** command is used to create an execution plan to see what changes terraform will make to your infrastructure without actually applying those changes.

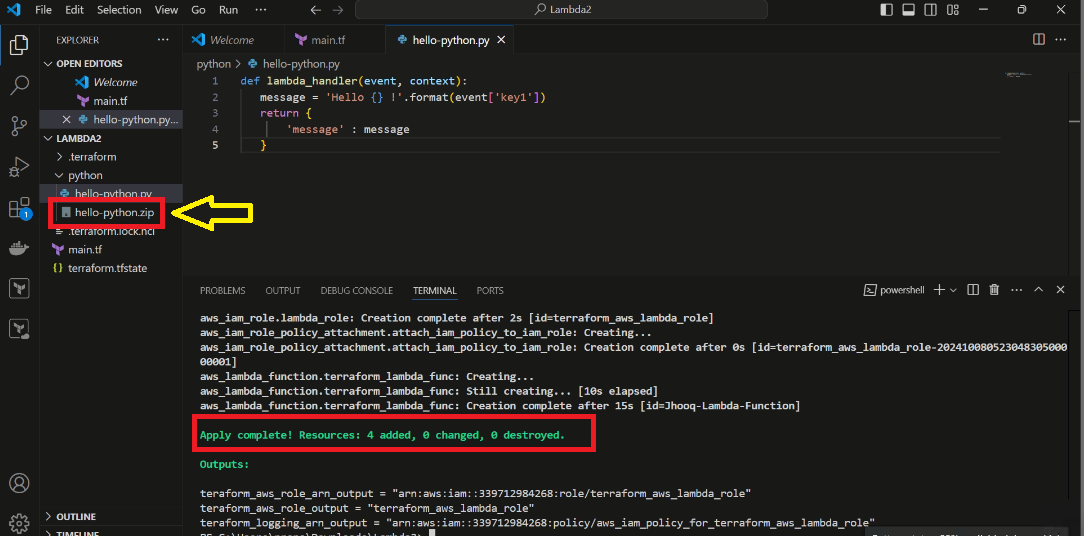


After give **terraform plan** automatically the python folder will be added and error will coming as shown in above image.

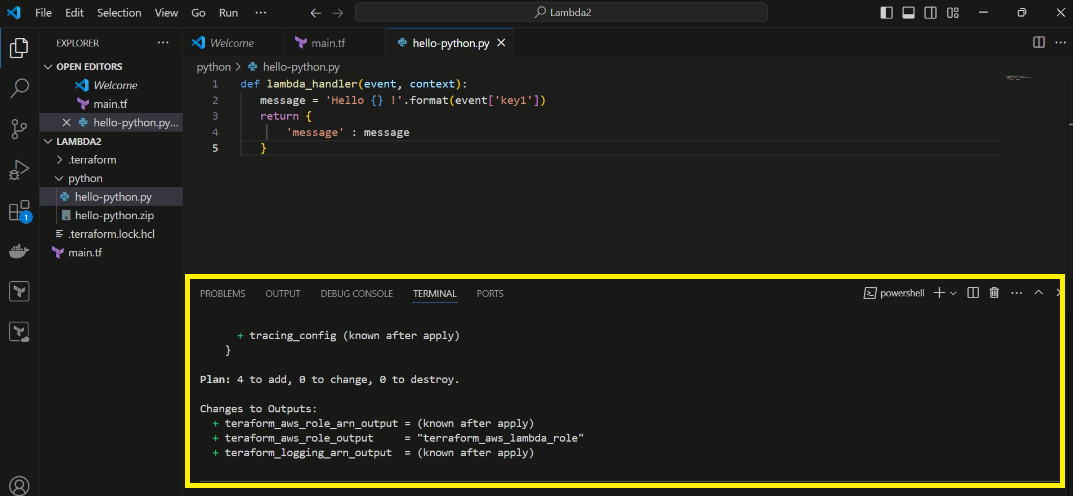
If we get that type of error in that python folder we can add a python file name as **“hello-pyton.py”.** Let’s write the python code for our lambda function.



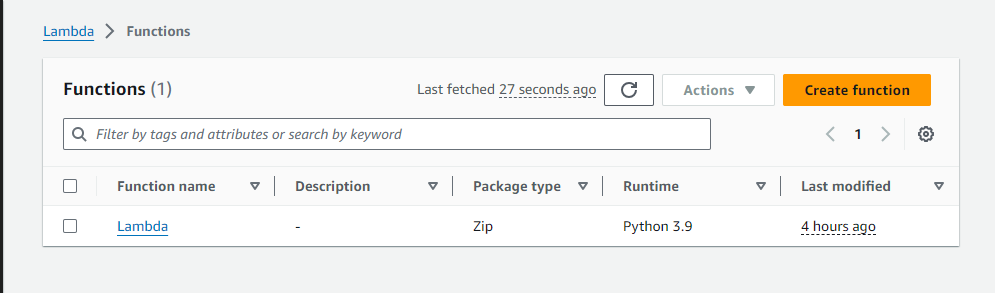
In time to add python code we again run code with the help of **“terraform plan”.**  After we have to give terraform plan we have to see changes like x added, x changed, x destroyed (here x indicates some number) and automatically added a python zip file as shown below.



After run the code we have to give **“terraform apply”** command, this command is a critical part of the terraform workflow that executes the planned actions, creating, updating, or deleting infrastructure resources to match the new state outlined in your IAC (Infrastructure As a Code)



After terraform apply it will reflect on AWS console page for Lambda function service, here below is the reference image.



This is the final output of automate infrastructure on AWS using Terraform and Lambda Function.

The total code is given below:

|  |
| --- |
| er provid "aws" {  region = "eu-central-1"  }    resource "aws\_iam\_role" "lambda\_role" {  name = "terraform\_aws\_lambda\_role"  assume\_role\_policy = <<EOF  {  "Version": "2012-10-17",  "Statement": [  {  "Action": "sts:AssumeRole",  "Principal": {  "Service": "lambda.amazonaws.com"  },  "Effect": "Allow",  "Sid": ""  }  ]  }  EOF  }    # IAM policy for logging from a lambda    resource "aws\_iam\_policy" "iam\_policy\_for\_lambda" {    name = "aws\_iam\_policy\_for\_terraform\_aws\_lambda\_role"  path = "/"  description = "AWS IAM Policy for managing aws lambda role"  policy = <<EOF  {  "Version": "2012-10-17",  "Statement": [  {  "Action": [  "logs:CreateLogGroup",  "logs:CreateLogStream",  "logs:PutLogEvents"  ], |

|  |
| --- |
| "Resource": "arn:aws:logs:\*:\*:\*",  "Effect": "Allow"  }  ]  }  EOF  }    # Policy Attachment on the role.    resource "aws\_iam\_role\_policy\_attachment" "attach\_iam\_policy\_to\_iam\_role" {  role = aws\_iam\_role.lambda\_role.name  policy\_arn = aws\_iam\_policy.iam\_policy\_for\_lambda.arn  }    # Generates an archive from content, a file, or a directory of files.    data "archive\_file" "zip\_the\_python\_code" {  type = "zip"  source\_dir = "${path.module}/python/"  output\_path = "${path.module}/python/hello-python.zip"  }    # Create a lambda function  # In terraform ${path.module} is the current directory.  resource "aws\_lambda\_function" "terraform\_lambda\_func" {  filename = "${path.module}/python/hello-python.zip"  function\_name = "Jhooq-Lambda-Function"  role = aws\_iam\_role.lambda\_role.arn  handler = "hello-python.lambda\_handler"  runtime = "python3.9"  depends\_on = [aws\_iam\_role\_policy\_attachment.attach\_iam\_policy\_to\_iam\_role]  }      output "teraform\_aws\_role\_output" {  value = aws\_iam\_role.lambda\_role.name  }    output "teraform\_aws\_role\_arn\_output" {  value = aws\_iam\_role.lambda\_role.arn  }    output "teraform\_logging\_arn\_output" {  value = aws\_iam\_policy.iam\_policy\_for\_lambda.arn  } |