

**Cloud Computing Architecture Lab**

**Name:** Pratham Kandari

**Sap Id:** 500097663

**Batch:** 7

**Experiment -10**

Creating, Starting and Stopping EC2 instances using AWS Lambda

**Q:1.** What is AWS Lambda? Explain the working and benefits of AWS lambda in detail.

**Ans:** AWS Lambda is a serverless compute service offered by Amazon Web Services (AWS) that allows developers to run code without having to manage servers or infrastructure. It is a popular choice for building event-driven applications and microservices because it can automatically scale up or down to handle changes in application traffic.

The basic working of AWS Lambda is that it allows developers to write code in a variety of programming languages, including Node.js, Python, Java, C#, and Go, and then upload that code to AWS Lambda. The developer can then specify when and how that code should be executed by creating a trigger for the Lambda function. Triggers can be anything from changes to an S3 bucket, a new message in an SQS queue, or a new event in an Amazon EventBridge event bus. Once the trigger condition is met, AWS Lambda executes the code and returns the results to the calling application.

One of the key benefits of AWS Lambda is its scalability. Because Lambda automatically scales up or down to handle changes in application traffic, developers don't need to worry about provisioning and managing servers. This makes it easier to build and deploy event-driven applications and microservices that can quickly and seamlessly respond to changes in application usage.

Another benefit of AWS Lambda is its cost-effectiveness. With traditional server-based architecture, developers have to pay for servers whether they are being used or not. With AWS Lambda, developers only pay for the compute time that their code actually uses. This means that they can build and run applications at a much lower cost, which makes it more accessible for small businesses and startups.

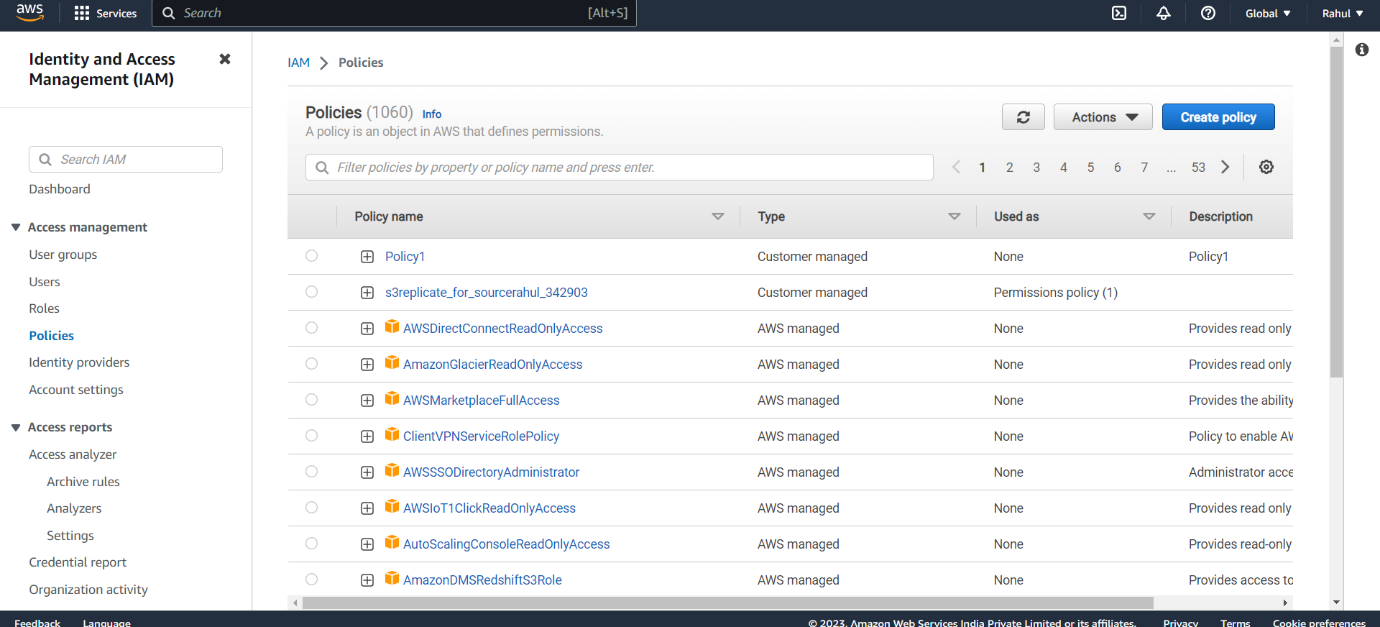
Additionally, AWS Lambda supports a wide variety of integrations with other AWS services, such as Amazon S3, Amazon DynamoDB, and Amazon EventBridge. This makes it easier to build complex, event-driven applications that can leverage the power of AWS's entire suite of cloud services.

**Q:2.** Explain in detail the use cases of AWS Lambda.

**Ans:** AWS Lambda is a serverless compute service that offers a wide range of use cases for developers. Here are some of the most common use cases for AWS Lambda:

1. Event-driven applications: AWS Lambda is designed for building event-driven applications that respond to changes in data or user behavior. For example, developers can use Lambda to automatically process new data uploaded to an S3 bucket, to trigger a notification when a new record is added to a DynamoDB table, or to analyze log data in real-time.
2. Microservices: AWS Lambda is well-suited for building microservices that can be used in a variety of different applications. Developers can build small, independent functions that can be reused across multiple applications, which makes it easier to maintain and update code over time.
3. Data processing: AWS Lambda can be used for processing large amounts of data, such as image or video files. Developers can write code to process data in parallel, which can help reduce processing time and improve performance.
4. Chatbots: AWS Lambda is commonly used for building chatbots that can be integrated with messaging platforms like Slack or Facebook Messenger. Developers can write code that listens for new messages, processes the text input, and returns a response.
5. IoT applications: AWS Lambda is well-suited for building IoT applications that can process data from sensors and other devices. Developers can write code that listens for new data from devices, processes that data, and triggers actions based on the results.
6. Mobile backends: AWS Lambda can be used to build mobile backends that can process data from mobile devices and trigger actions based on that data. For example, developers can write code that processes location data from a mobile device and triggers notifications based on that data.
7. Batch processing: AWS Lambda can be used for batch processing of large data sets. Developers can write code to process data in batches, which can help reduce processing time and improve performance.

Step 1) Go to IAM and Create a policy .



Step 2) Under Json paste the following script.

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents"

],

"Resource": "arn:aws:logs:\*:\*:\*"

},

{

"Action": [

"ec2:RunInstances"

],

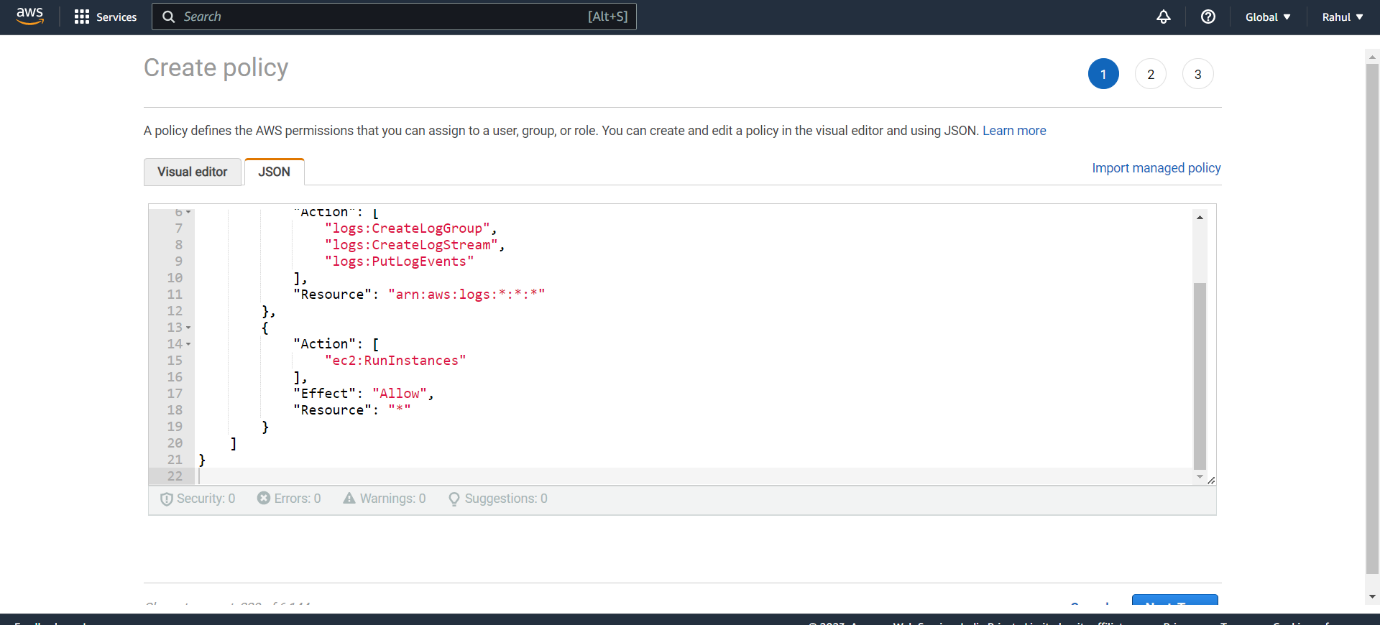
"Effect": "Allow",

"Resource": "\*"

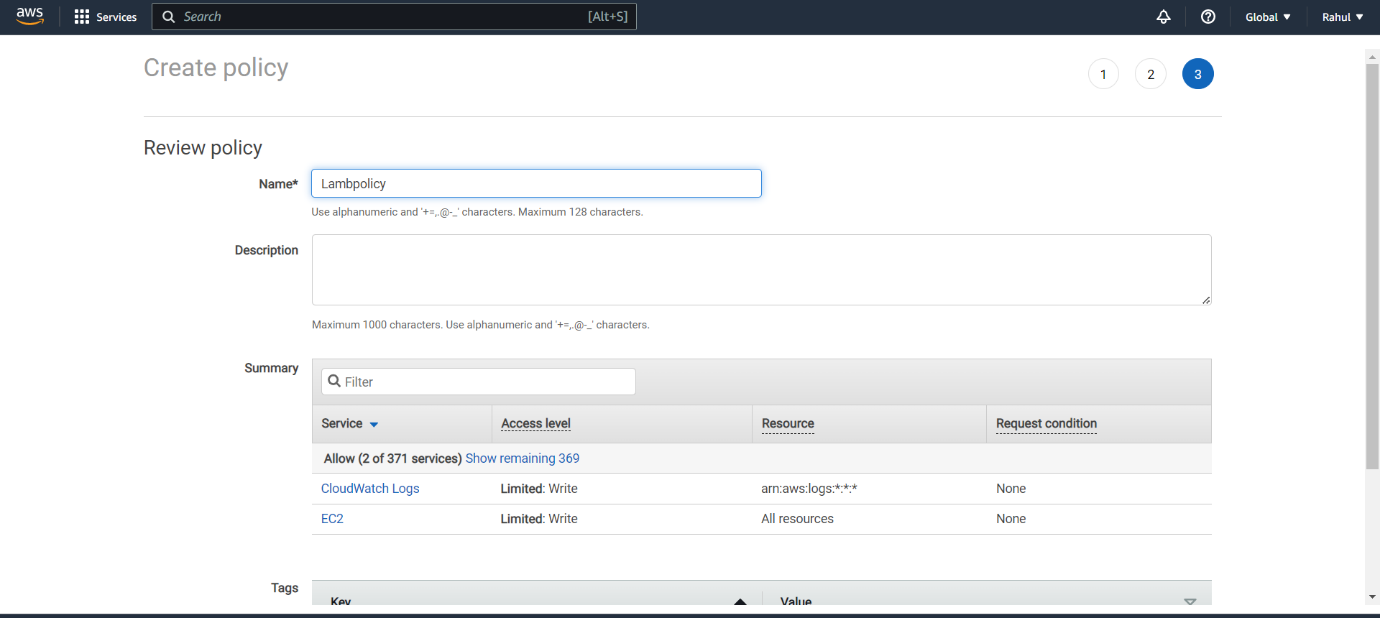
}

]

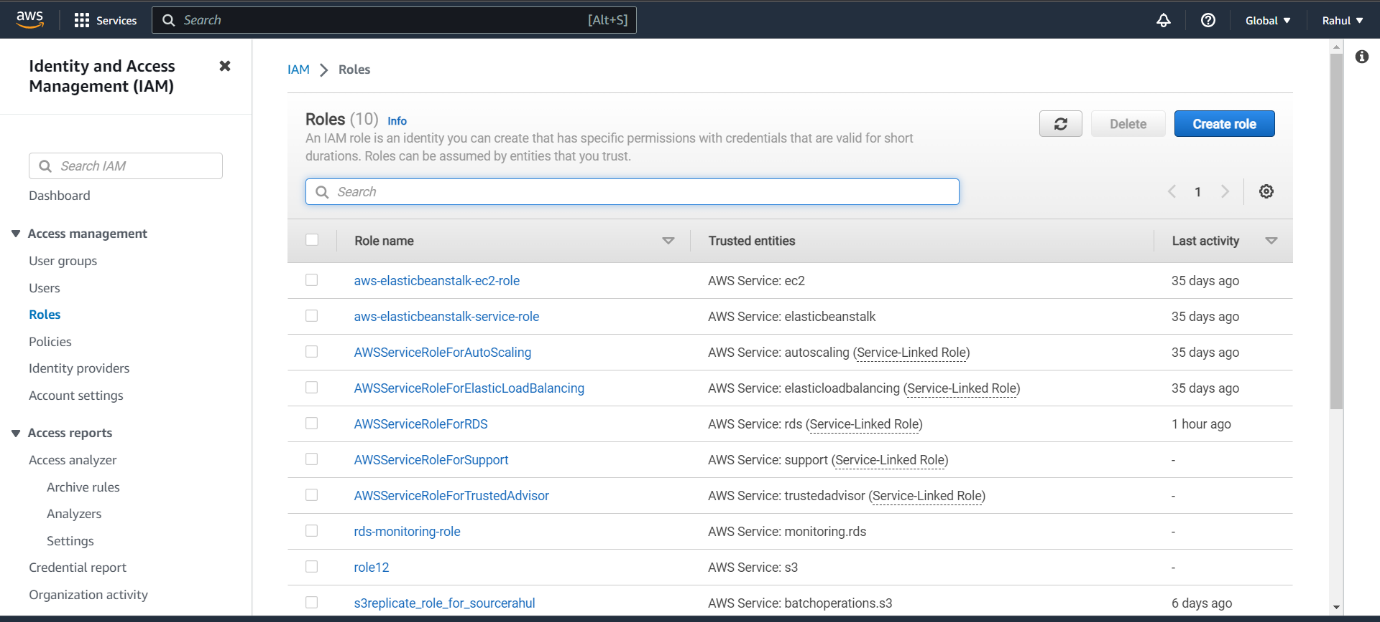
}



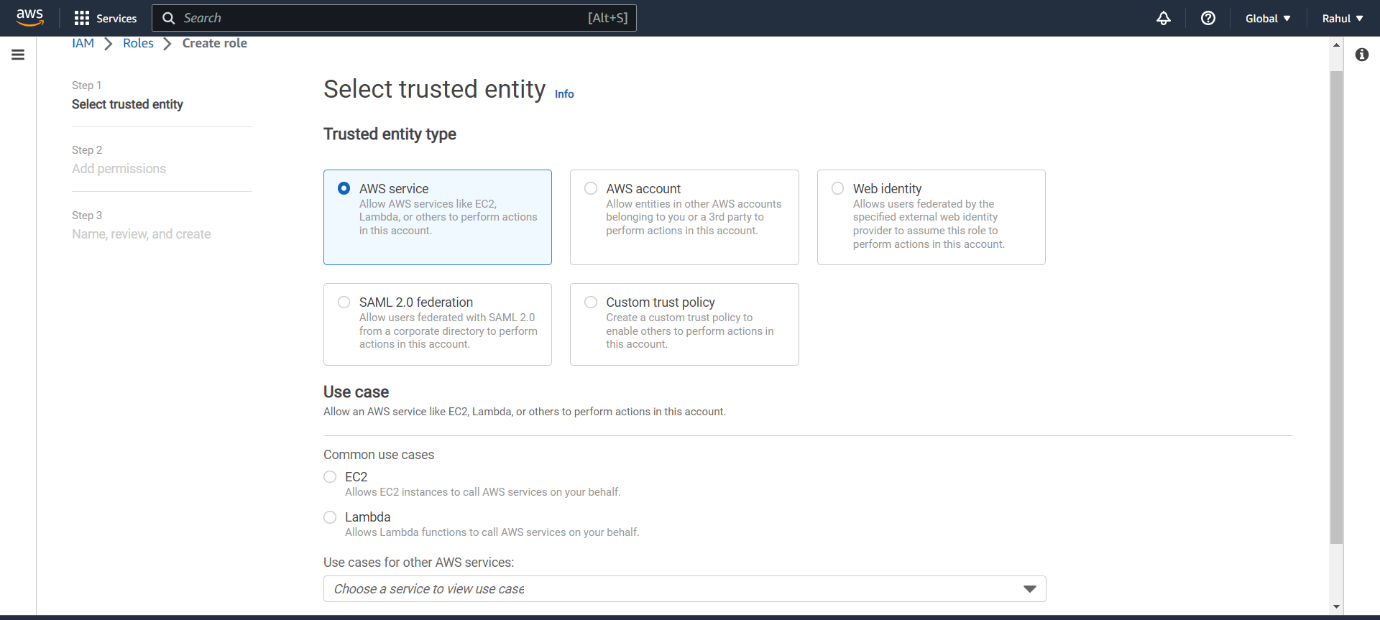
Step 3) And Create the Policy

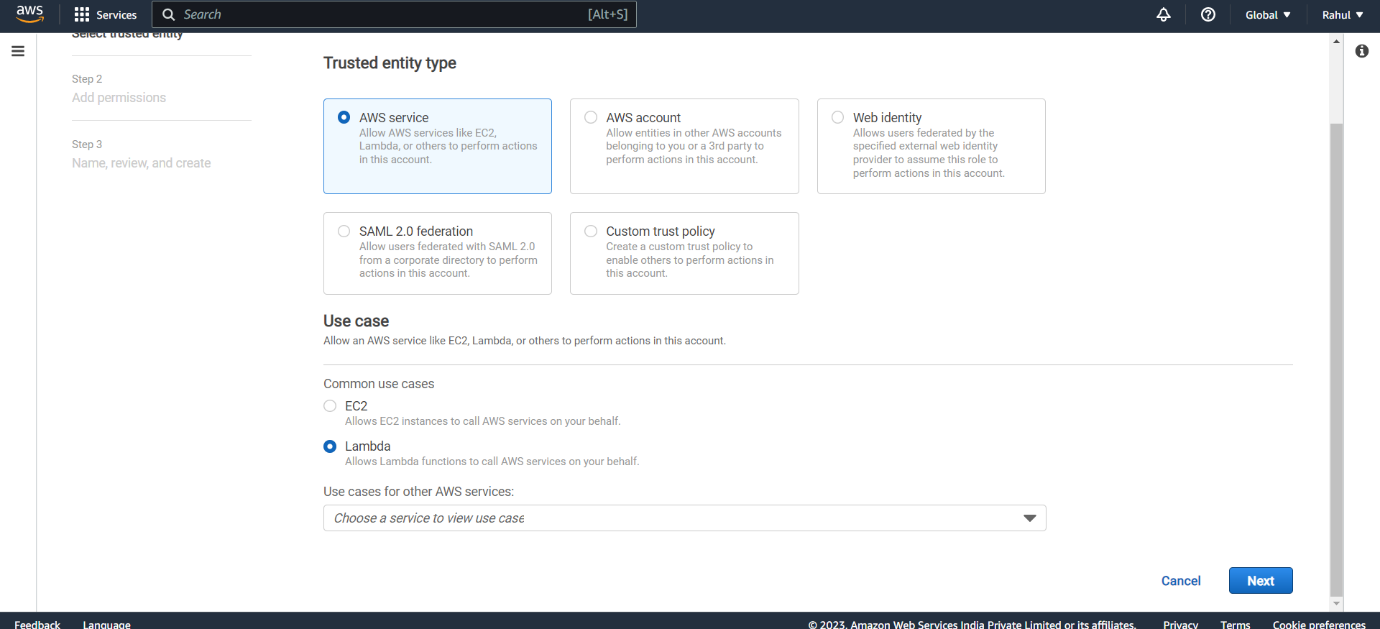


Step 4) Go to roles -> create Role.

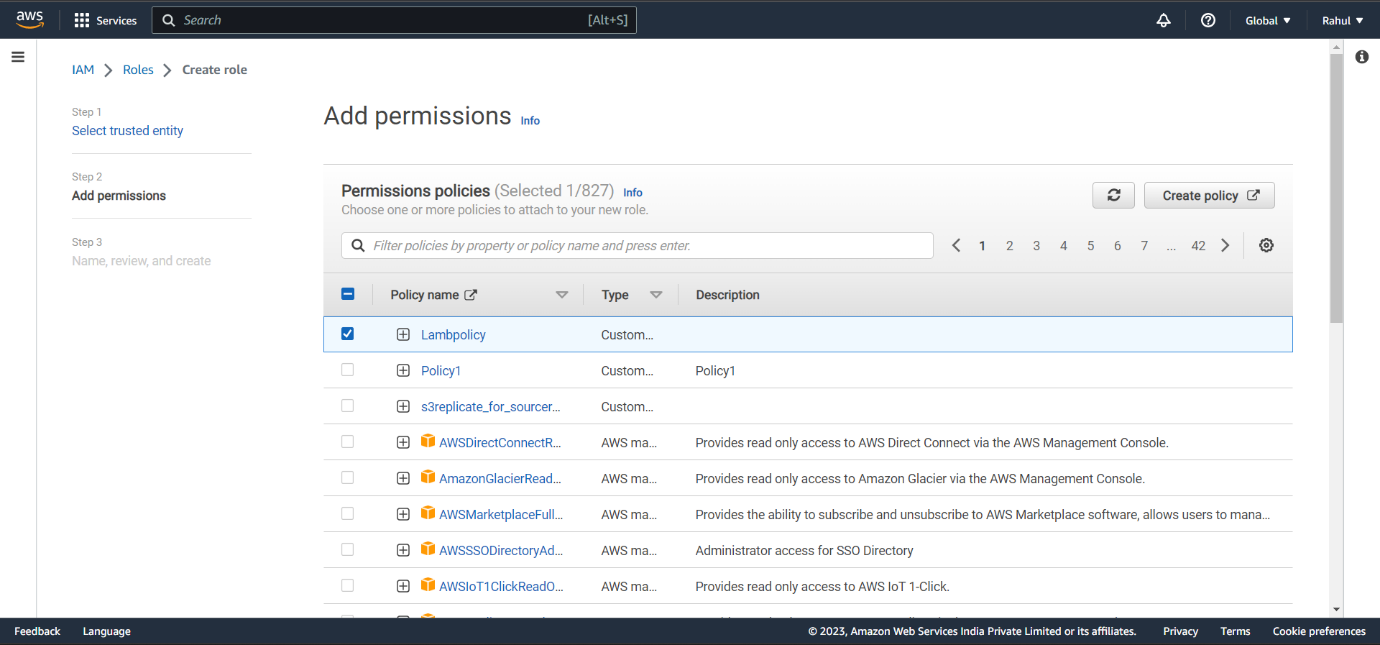


Step 5) Entity type -> AWS Service.



Step 6) Use case -> Lambda.

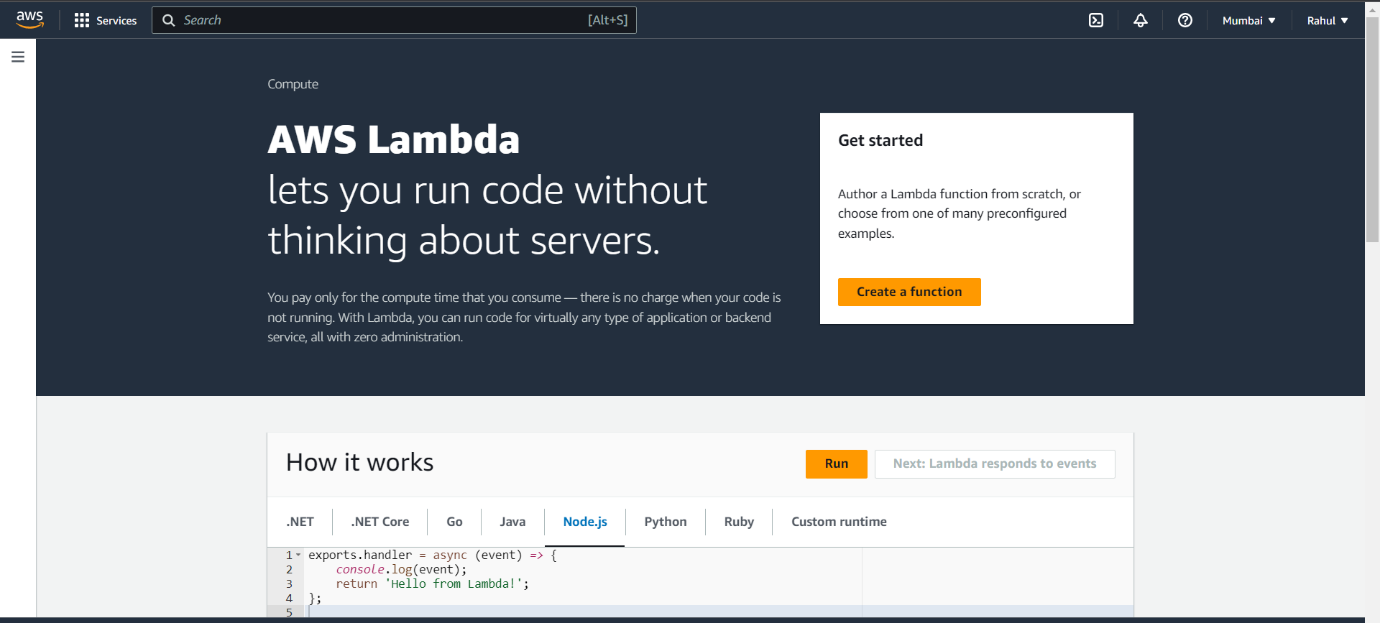
Step 7) Attach the policy created before.



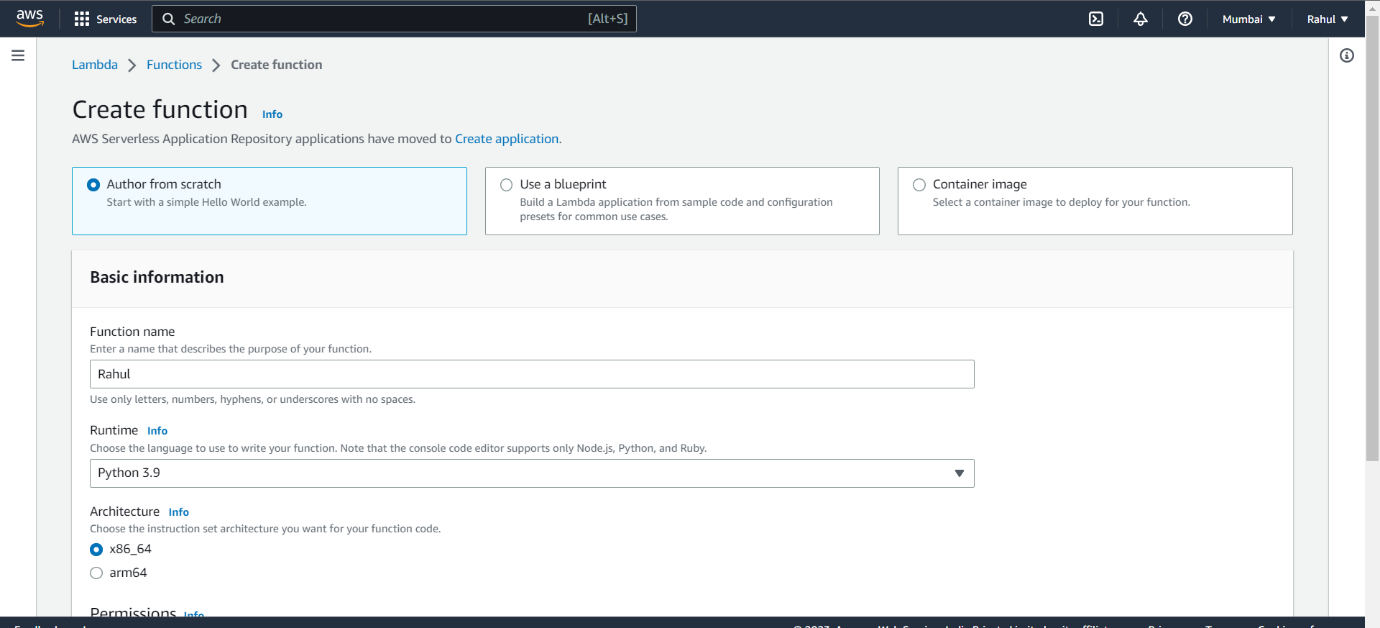
Step 8) Give it a name and create Role.

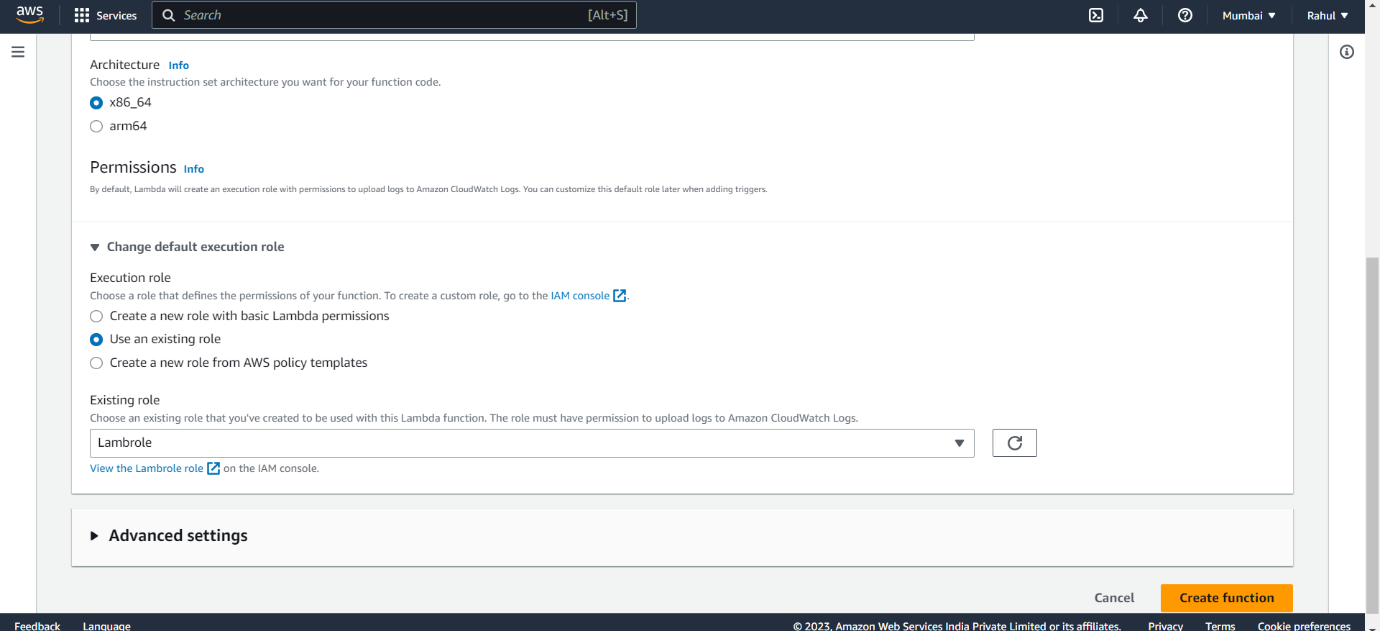


Step 9) Go to AWS Lambda -> Create a function.

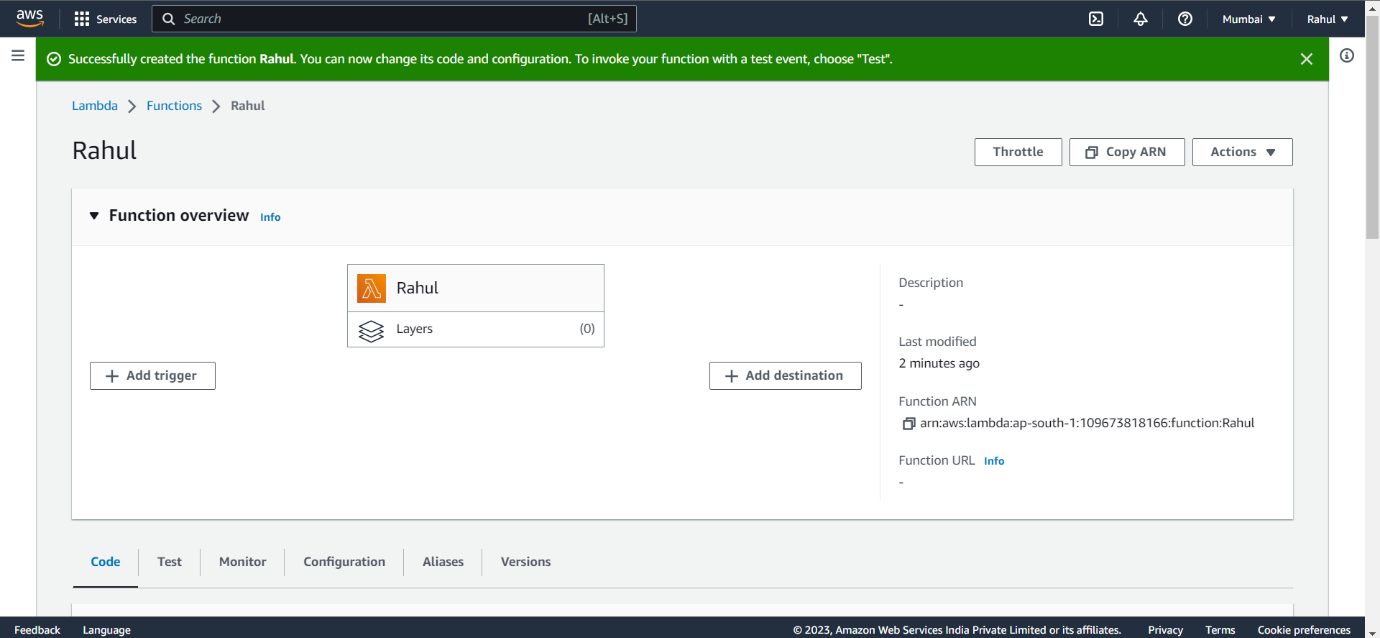


Step 10) Let everything be default , give it a name, Runtime -> Python.



Step 11) Under permissions select the role you created and create the function.

Step 12) Function created.



Step 13) Go to CODE and paste the following code.

import os

import boto3

AMI = os.environ['AMI']

INSTANCE\_TYPE = os.environ['INSTANCE\_TYPE']

KEY\_NAME = os.environ['KEY\_NAME']

SUBNET\_ID=os.environ['SUBNET\_ID']

REGION=os.environ['REGION']

ec2 = boto3.resource('ec2',region\_name=REGION)

def lambda\_handler(event, context):

instance = ec2.create\_instances(

ImageId=AMI,

InstanceType=INSTANCE\_TYPE,

KeyName=KEY\_NAME,

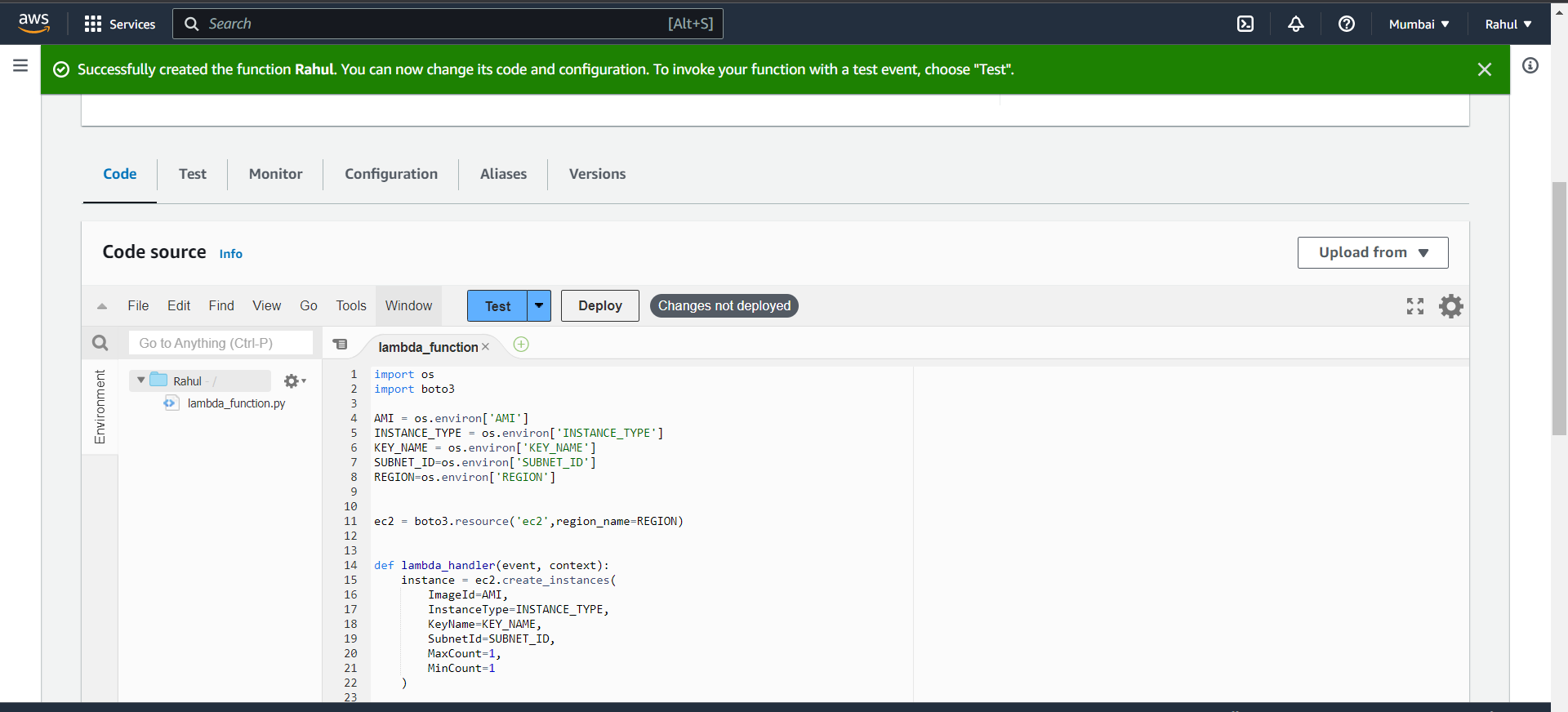
SubnetId=SUBNET\_ID,

MaxCount=1,

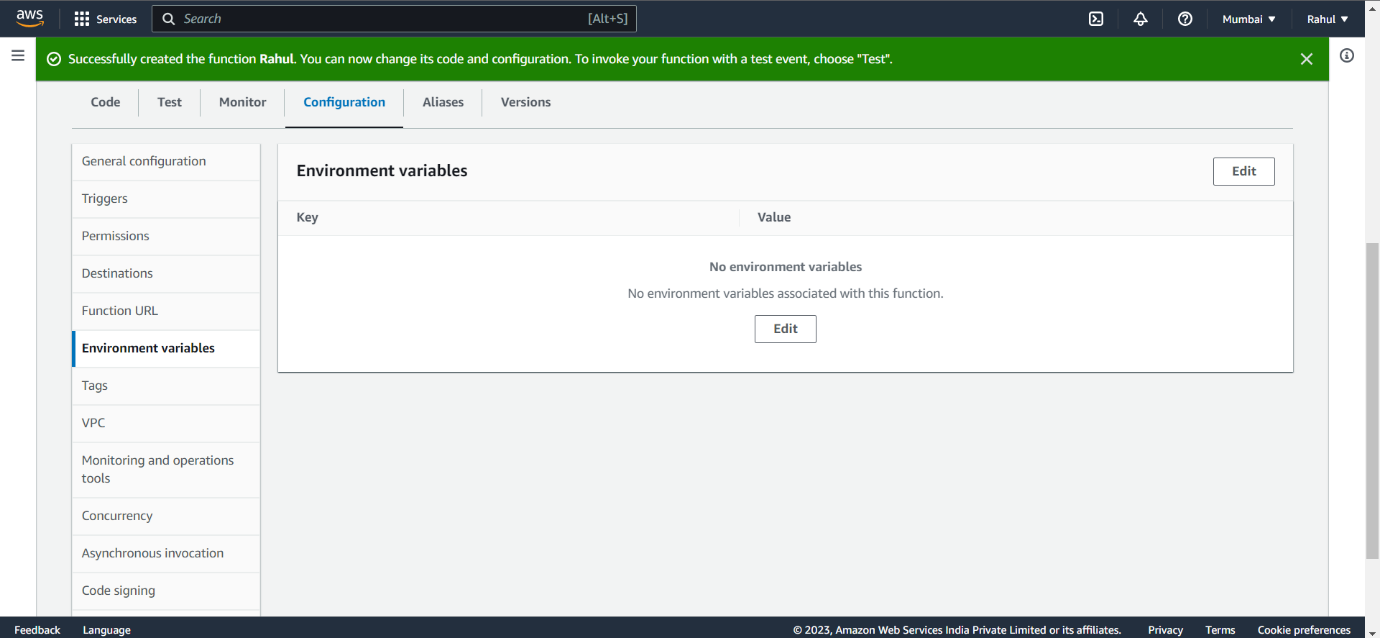
MinCount=1

)

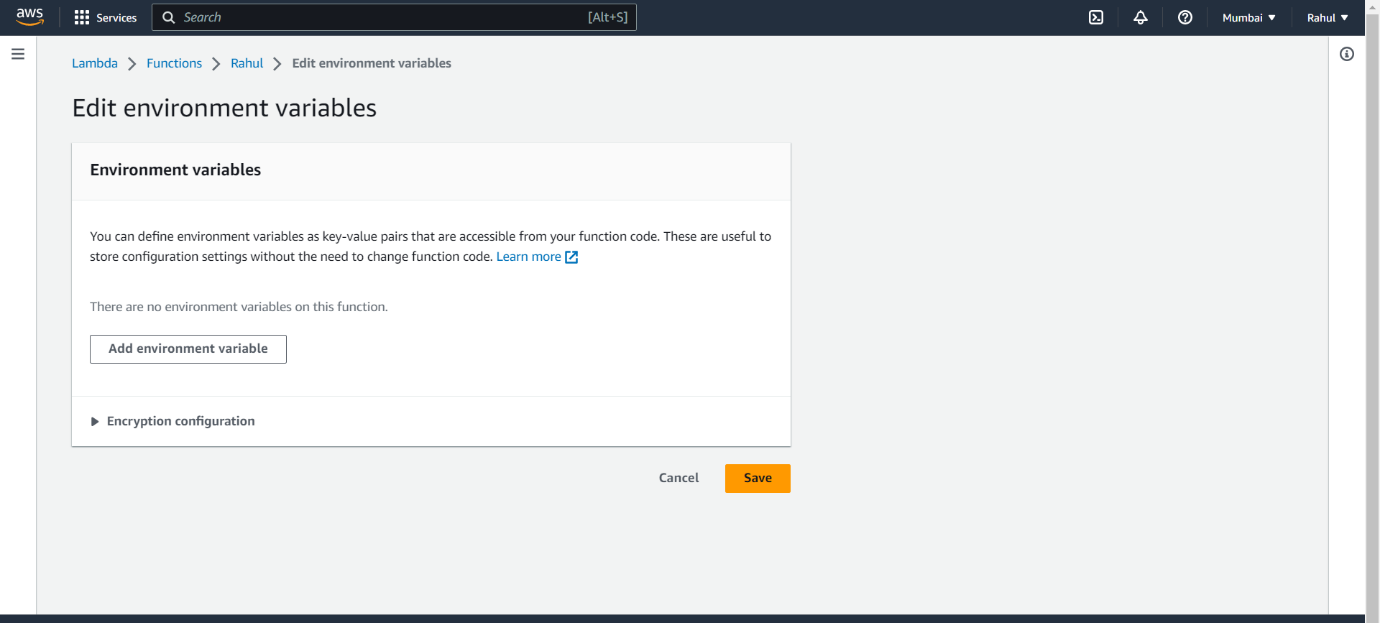
print("New instance created:", instance[0].id)



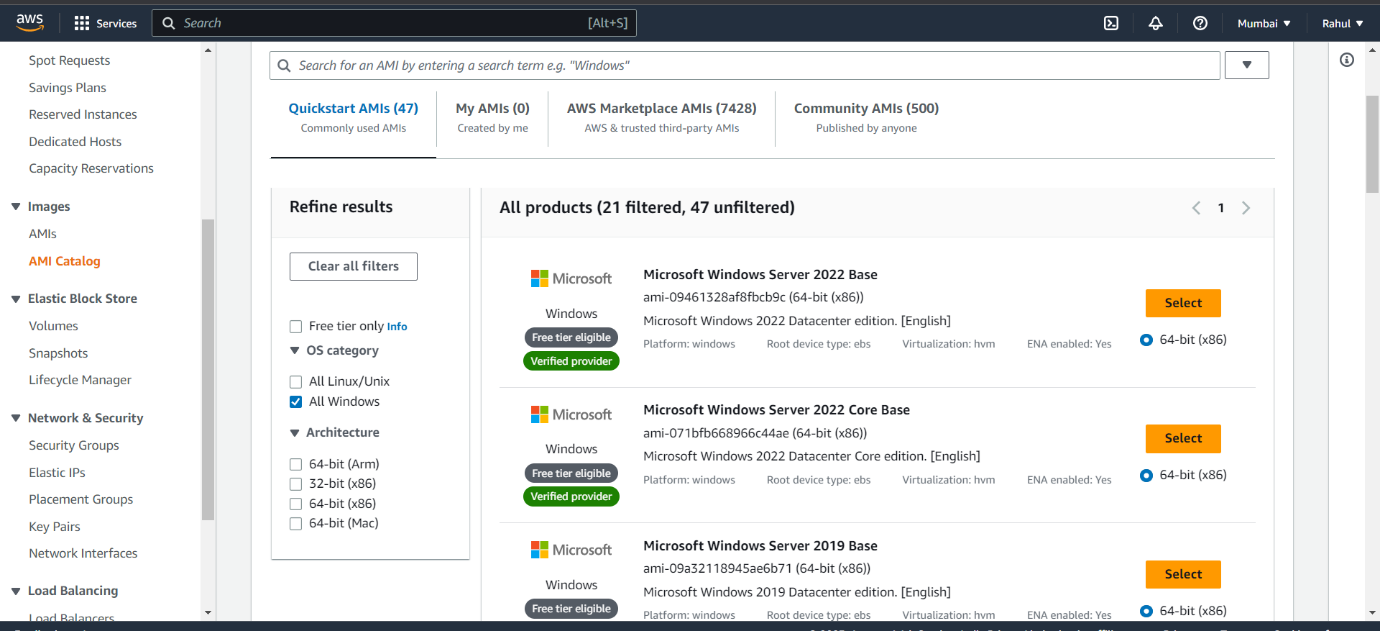
Step 14) Go to configuration -> Environment variables -> edit.



Step 15) Add environment variables.



Step 16) Copy a Windows AMI ID from AMI Catalog.



Step 17) For subnet ID, go to VPC-> Subnet and select the vpc in your region.

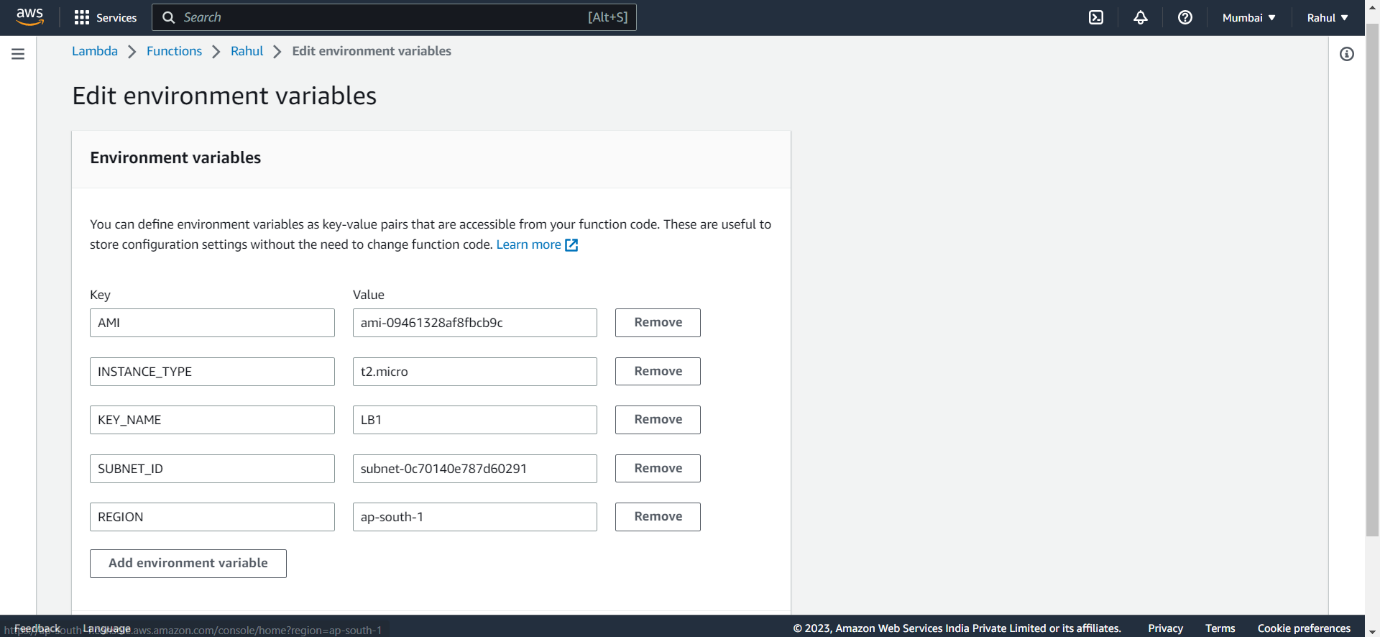
Graphical user interface, application

Description automatically generated

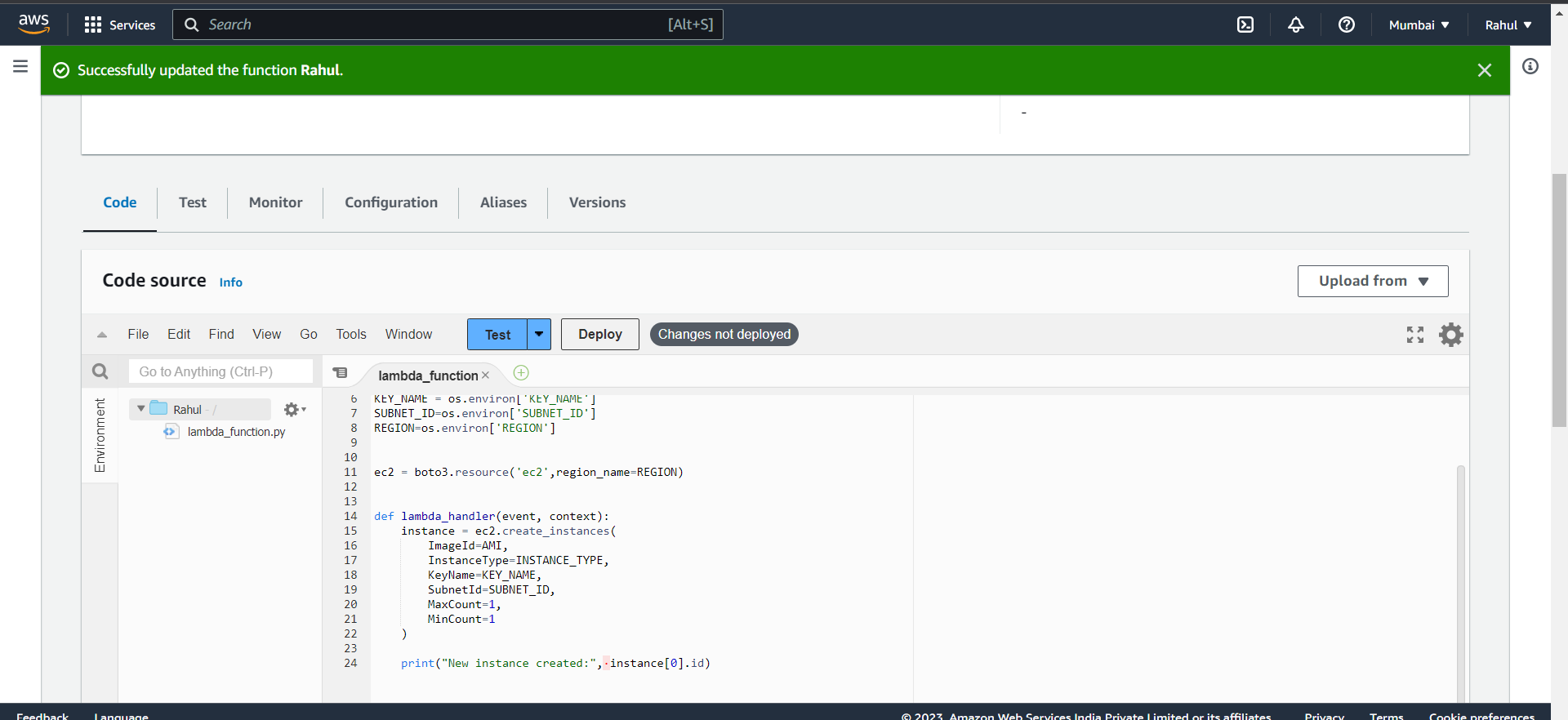
Step 18) Add the following environment variables

* AMI - ami-09461328af8fbcb9c
* INSTANCE\_TYPE - t2.micro
* KEY\_NAME - (Any key you created)
* SUBNET\_ID – (Subnet copied from VPC->Subnets)
* REGION – (The region you’re in)

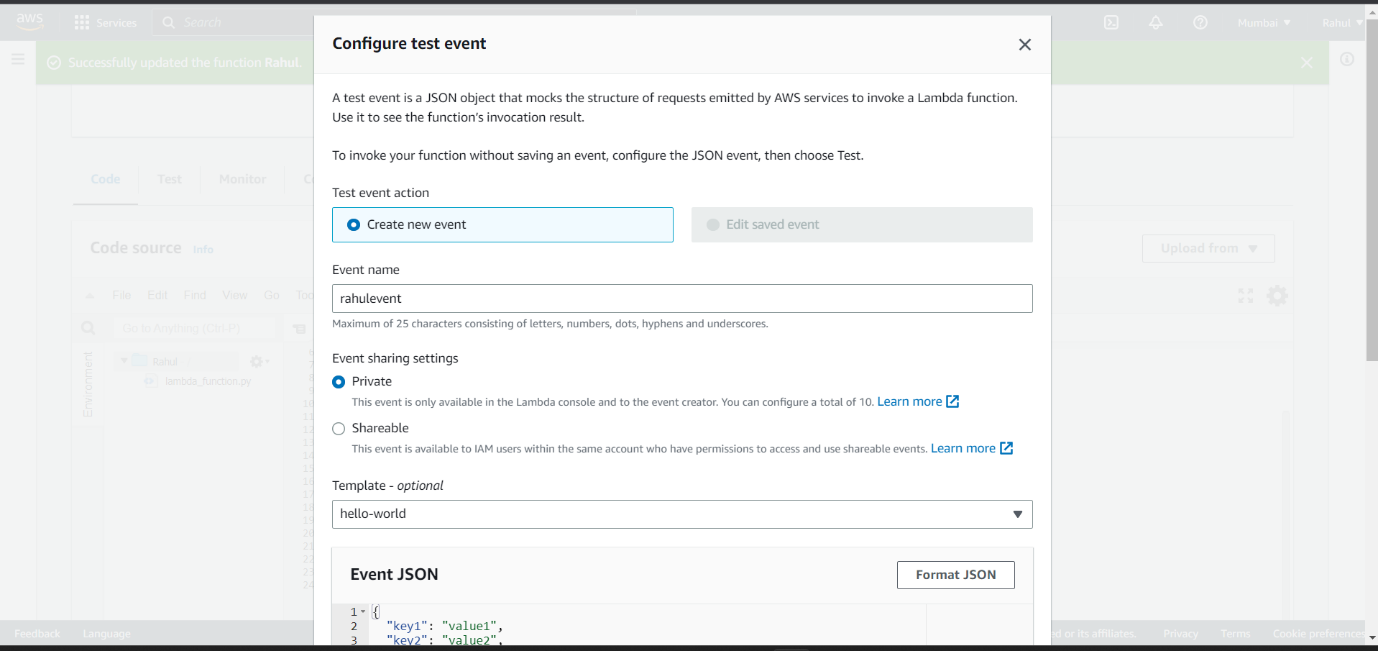
And save it.



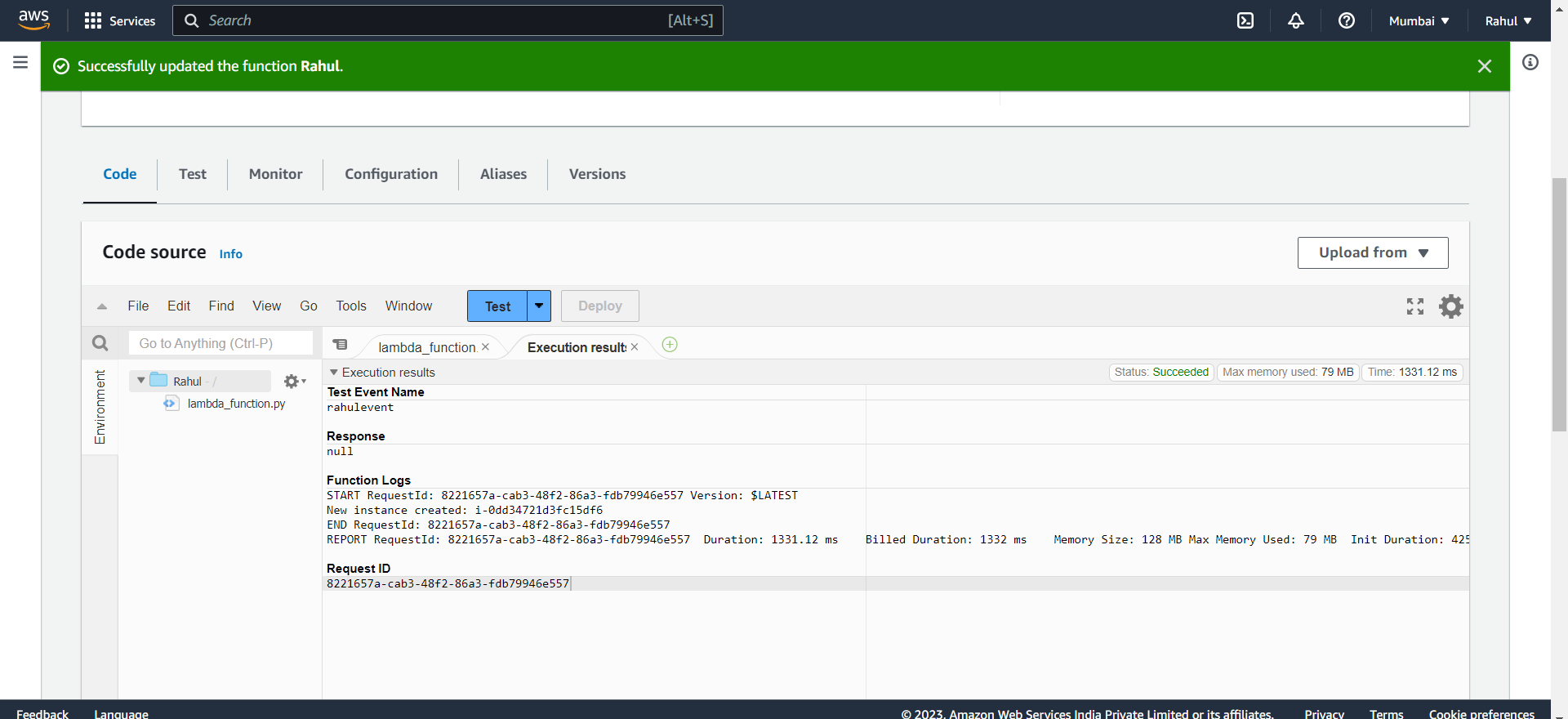
Step 19) Go back to CODE and click on Deploy.



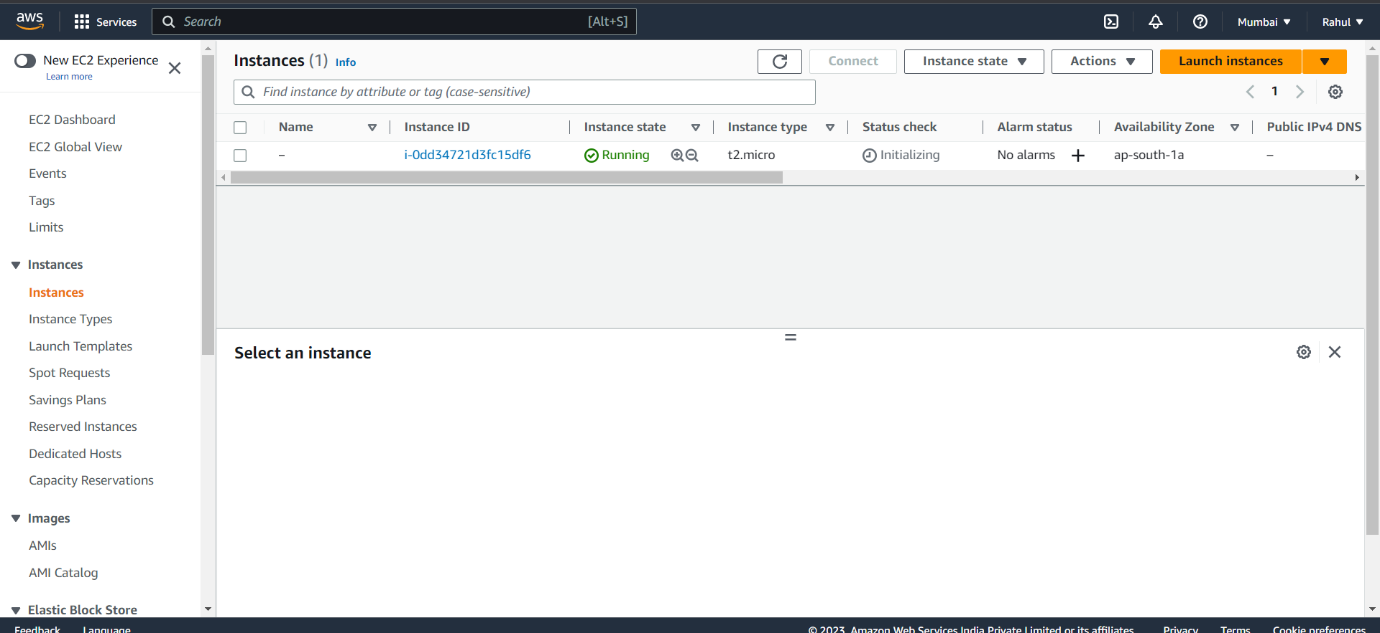
Step 20) Now after deploying click on test, give it a name, and select sharable.



Step 21) Successfully executed.

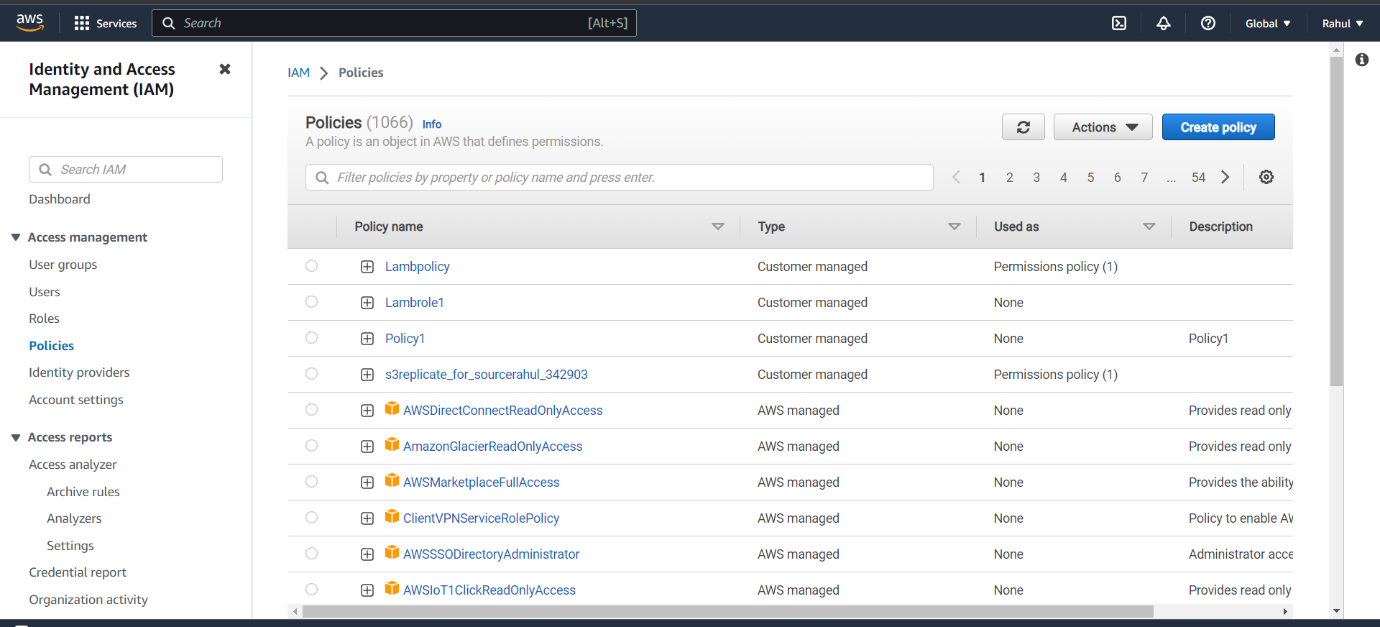


Step 22) As you can see instance created.



**To stop the instance**

Step 23) Now go back to IAM -> policies -> Create policy.



Step 24) And under JSON Paste the following script.

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents"

],

"Resource": "arn:aws:logs:\*:\*:\*"

},

{

"Effect": "Allow",

"Action": [

"ec2:Start\*",

"ec2:Stop\*"

],

"Resource": "\*"

}

]

}

Graphical user interface, text, application

Description automatically generated

Step 25) Give it a name and save.

A screenshot of a computer

Description automatically generated

Step 26) Go to roles -> Create role.

Graphical user interface, text, application, email

Description automatically generated

Step 27) Entity type -> AWS service, use case -> Lambda.

Graphical user interface, text, application

Description automatically generated

Step 28) Attach the stop policy created before.

A screenshot of a computer

Description automatically generated

Step 29) Give it a name and create role.

A screenshot of a computer

Description automatically generated

Step 30) Go to Lambda- >Create function,

Give the function a name, runtime - > Python.

Graphical user interface, text, application

Description automatically generated

Step 31) Select the stop role created and create function.

Graphical user interface, text, application, email

Description automatically generated

Step 32) After function is created, go to CODE and paste the following script.

import boto3

region = 'ap-south-1'

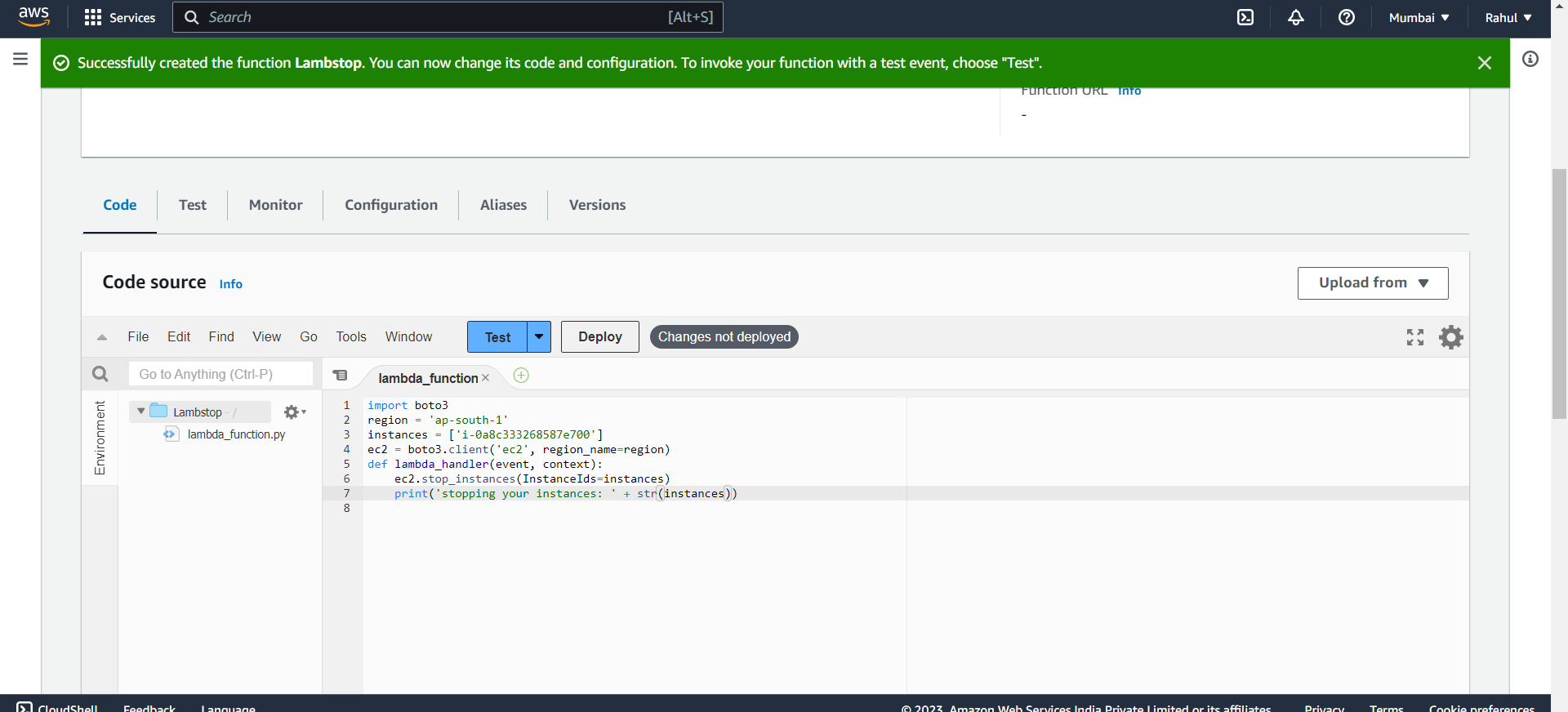
instances = ['i-0f89ee6c8705f2e61']

ec2 = boto3.client('ec2', region\_name=region)

def lambda\_handler(event, context):

ec2.stop\_instances(InstanceIds=instances)

print('stopping your instances: ' + str(instances))

Step 33) Deploy and test the code.

Graphical user interface, text, application, email, Teams

Description automatically generatedStep 34) Code successfully tested.

A screenshot of a computer

Description automatically generated

Step 35) As you can see instance has stopped.

Graphical user interface, text, application, Word

Description automatically generated