CLOUD COMPUTING ARCHITECTURE LAB

NAME- SHUBHI DIXIT

BATCH- 05

SAP ID- 500094571

EXPERIMENT- 10

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

AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS). It allows developers to run their code without managing servers, which can greatly reduce the operational overhead of deploying and scaling applications.

AWS Lambda works by allowing developers to upload their code to AWS Lambda and then configuring an event source, such as an API Gateway or S3 bucket, to trigger the execution of the code. When an event occurs, AWS Lambda automatically runs the code in response to the event, and then shuts down the execution environment once the code has completed running.

This means that you only pay for the time your code is actually running, and there is no need to pay for idle time or to provision and manage servers.

One of the primary benefits of AWS Lambda is its scalability. As more events occur, AWS Lambda will automatically scale the execution environment to handle the increased load, ensuring that your code can handle any amount of traffic without needing to manually scale your infrastructure. Additionally, AWS Lambda integrates with a wide variety of AWS services, such as Amazon S3, DynamoDB, and API Gateway, making it easy to build serverless applications that can handle a wide range of use cases.

Another benefit of AWS Lambda is its cost-effectiveness. Because you only pay for the time your code is actually running, AWS Lambda can be much cheaper than traditional server-based computing models, especially for applications with variable or sporadic traffic patterns. Additionally, AWS Lambda offers a free tier that includes 1 million free requests per month and 400,000 GB-seconds of compute time per month, making it easy to get started with the service without incurring any costs.

Overall, AWS Lambda is a powerful tool that can greatly simplify the process of building and deploying applications in the cloud. Its scalability, flexibility, and cost-effectiveness make it an attractive choice for a wide range of use cases, from simple data processing tasks to complex, multi-tier applications.

2) Explain in detail the use cases of AWS Lambda.

AWS Lambda is a serverless computing platform that provides a wide range of use cases across various industries. Here are some examples of how AWS Lambda can be used:

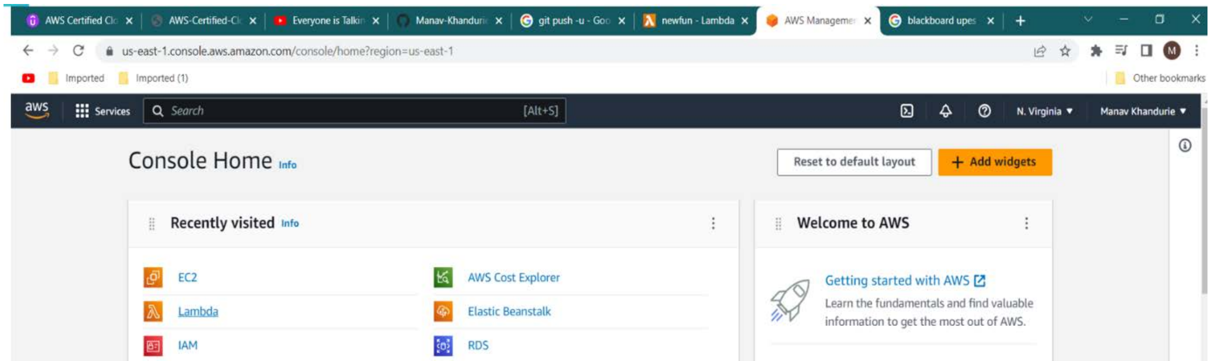
- Event-driven computing: AWS Lambda can be used to create event-driven architectures that can respond to real-time events. This makes it ideal for building chatbots, real-time data processing applications, and event-driven automation.
- Web applications: AWS Lambda can be used to build backend services for web applications. For example, it can be used to process form data, authenticate users, or perform other backend tasks for web applications.

- Data processing: AWS Lambda can be used to process data from various sources. For example, it can be used to process log files, create thumbnails for images, or perform data transformation tasks.
- Internet of Things (IoT): AWS Lambda can be used to create serverless applications that interact with IoT devices. This makes it ideal for building smart homes, industrial automation, and other IoT applications.
- Mobile applications: AWS Lambda can be used to build backend services for mobile applications. For example, it can be used to process user data, perform authentication, or push notifications.
- Machine learning: AWS Lambda can be used to build machine learning models and perform real-time predictions. It can also be used to trigger training jobs and perform model evaluation tasks.
- Batch processing: AWS Lambda can be used to perform batch processing tasks that require large-scale data processing. For example, it can be used to process data for

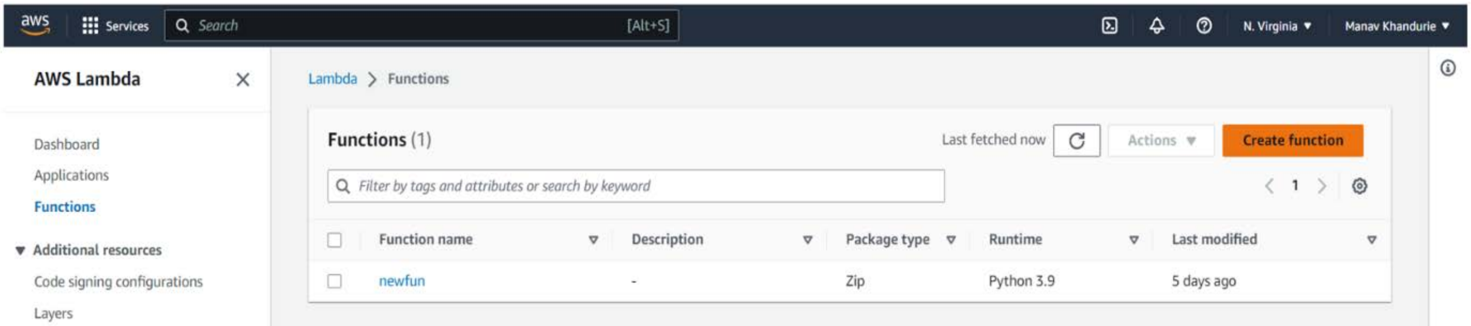
financial reports, inventory management, or customer analytics.

- Serverless architectures: AWS Lambda can be used to build serverless architectures that do not require the management of servers. This makes it ideal for building applications that require automatic scaling, high availability, and cost-effectiveness.

Step 1: Login to AWS Console and goto Lambda

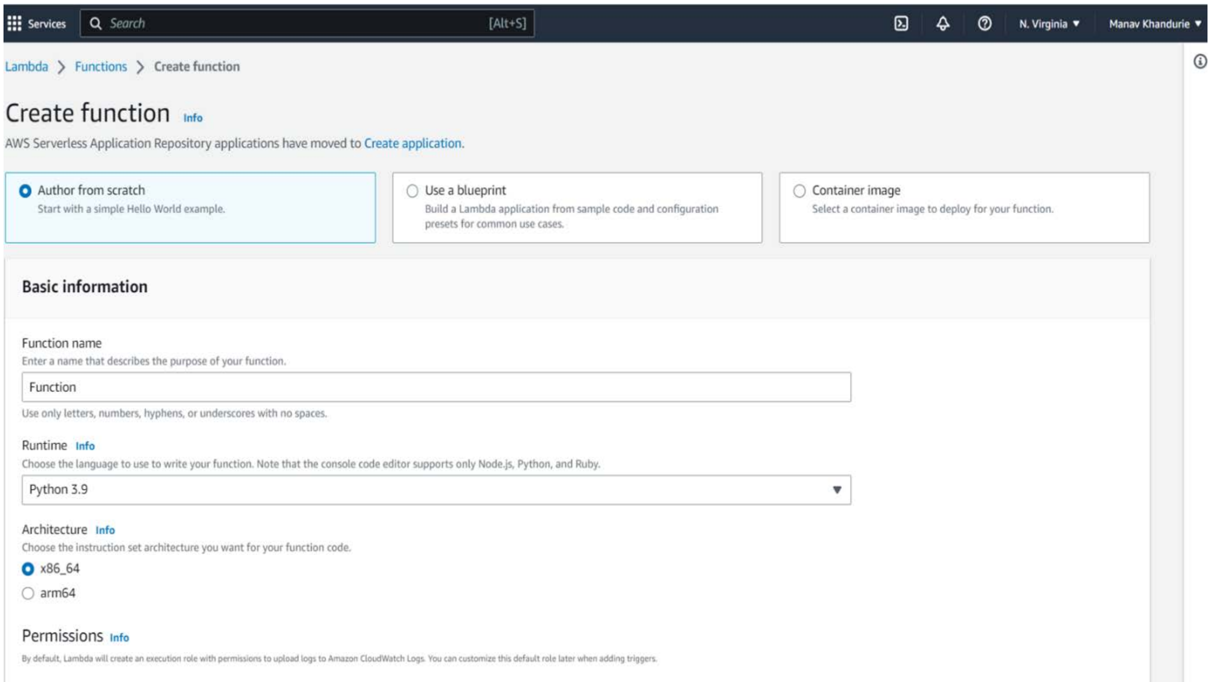


Step 2: In Lambda create a new function



□	Function name	Description	Package type	Runtime	Last modified
□	newfun	-	Zip	Python 3.9	5 days ago

Step 3: Give a name to the function & assign a name, runtime to the function



Lambda > Functions > Create function

Create function Info

AWS Serverless Application Repository applications have moved to Create application.

● Author from scratch
Start with a simple Hello World example.

○ Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

○ Container image
Select a container image to deploy for your function.

Basic information

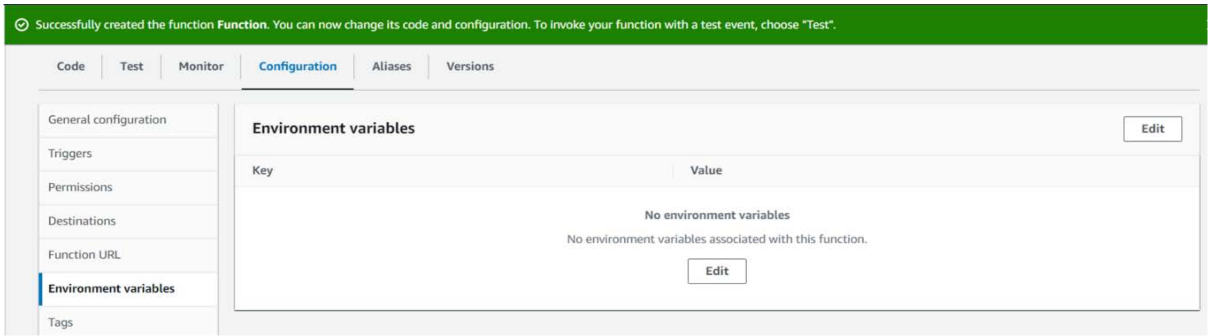
Function name
Enter a name that describes the purpose of your function.
Function
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.9 ▾

Architecture Info
Choose the instruction set architecture you want for your function code.
● x86_64
○ arm64

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Step 4: Click on create function and goto configration


Successfully created the function **Function**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

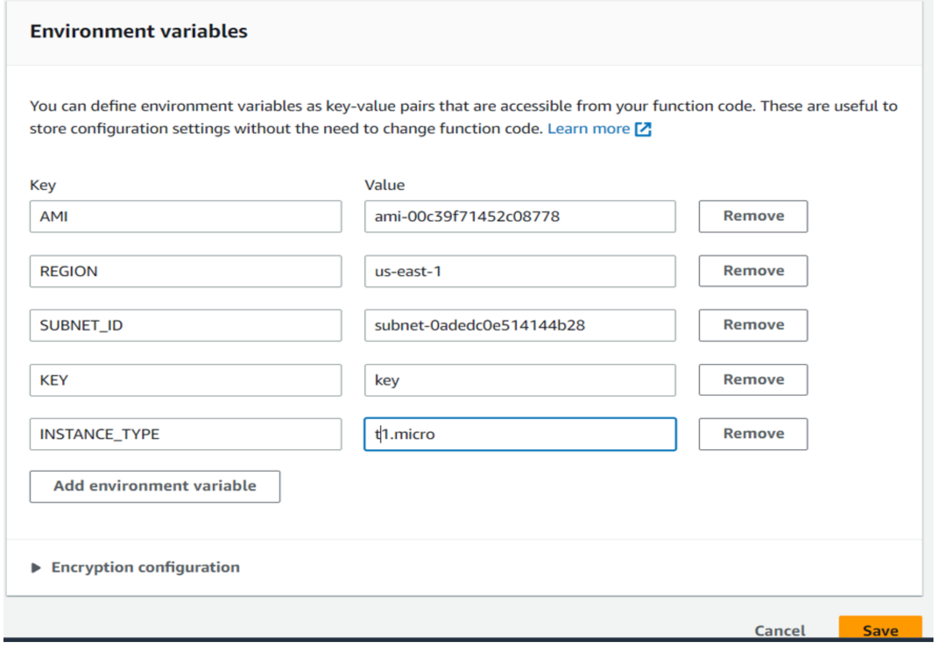
Environment variables

Key	Value

No environment variables

No environment variables associated with this function.

Step 5: Add environment variable by providing AMI,InstanceType,Region,key

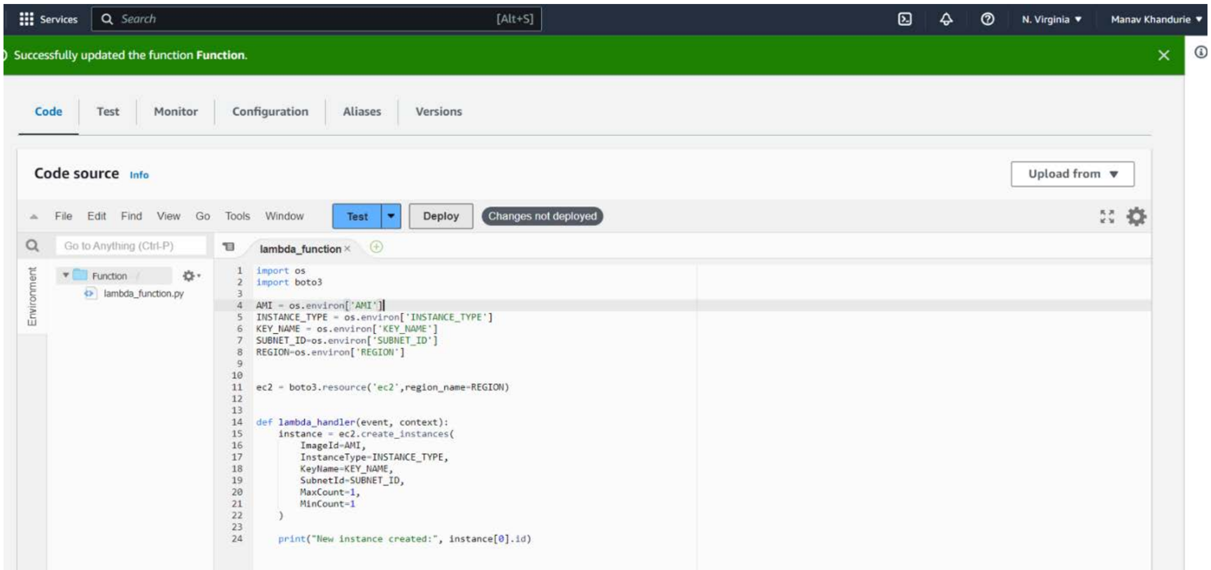


Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more

Key	Value
AMI	ami-00c39f71452c08778
REGION	us-east-1
SUBNET_ID	subnet-0adedc0e514144b28
KEY	key
INSTANCE_TYPE	t1.micro

Step 6: In the code section write a python script that creates a ec2 instance



Successfully updated the function **Function**.

```python
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)
```

#Python Script to create an ec2 instance

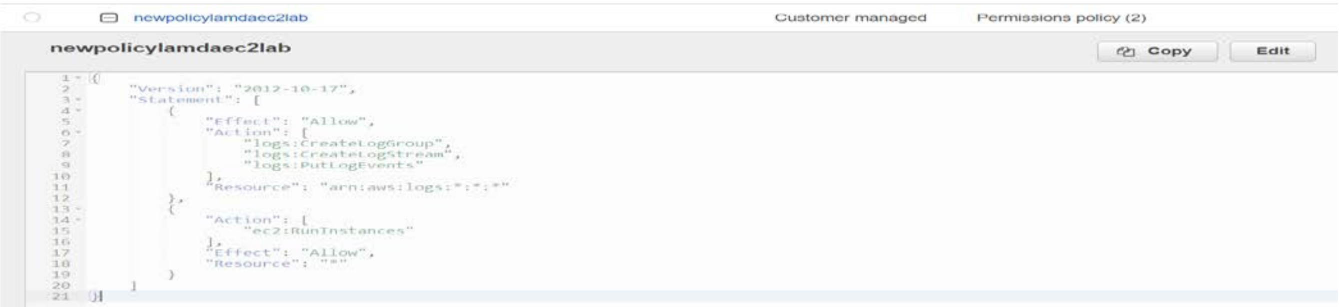
```
import os
import boto3

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 7: Now goto IAM and create a custom policy with the JSON Script-

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Action": [
                "ec2:RunInstances"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

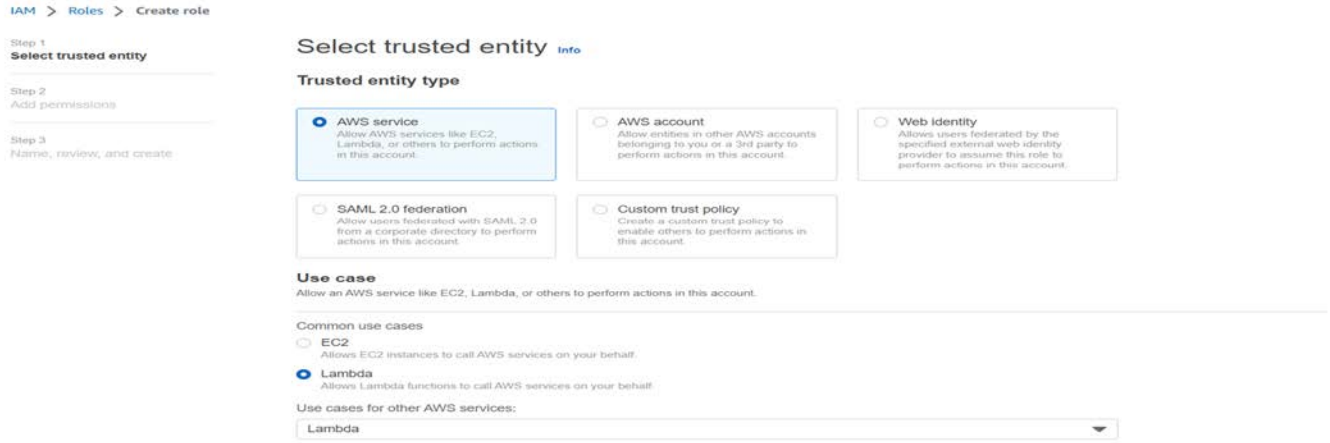


○ ⊟ newpolicylamdaec2lab — Customer managed — Permissions policy (2)

newpolicylamdaec2lab

```
{
    "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 8: Create an IAM role for Lambda service and attach the above policy to it



Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Select trusted entity

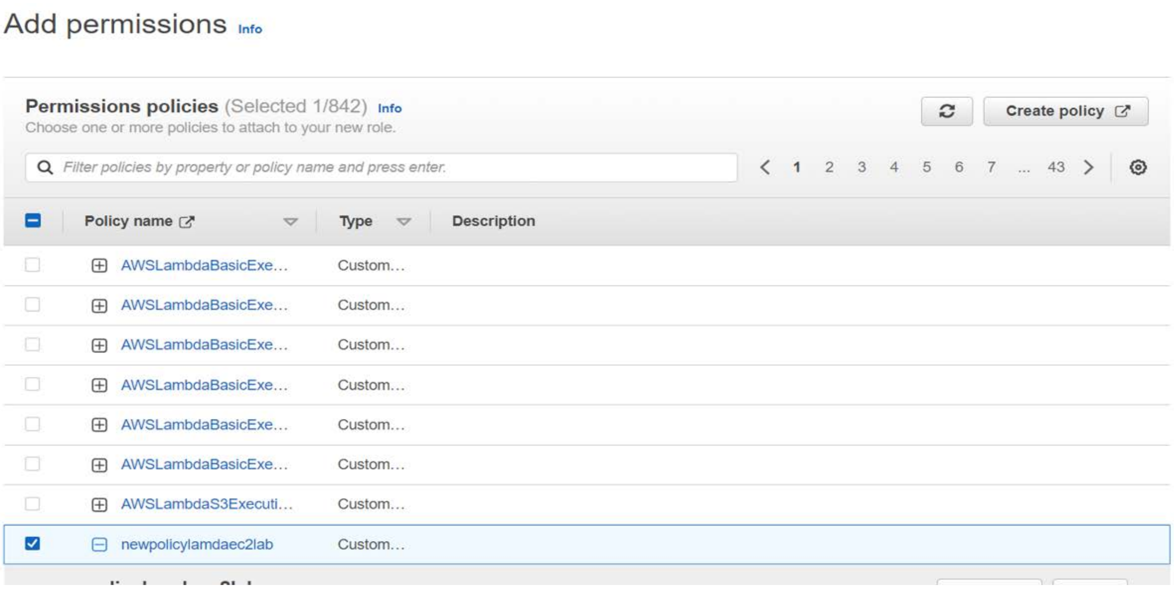
Trusted entity type

- ● AWS service — Allow AWS services like EC2, Lambda, or others to perform actions in this account.
- ○ AWS account — Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- ○ Web identity — Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.
- ○ SAML 2.0 federation — Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- ○ Custom trust policy — Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Common use cases
- ○ EC2 — Allows EC2 instances to call AWS services on your behalf.
- ● Lambda — Allows Lambda functions to call AWS services on your behalf.

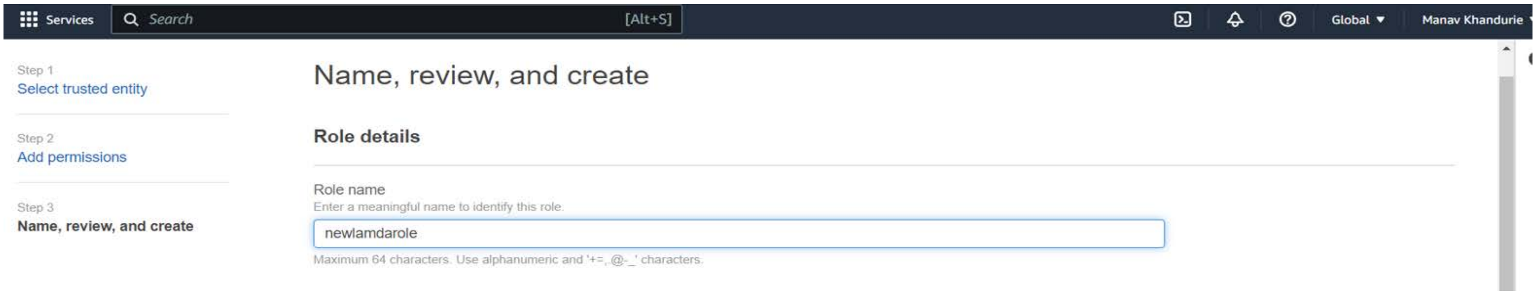
Use cases for other AWS services:
Lambda



Add permissions

Permissions policies (Selected 1/842)
Choose one or more policies to attach to your new role.

	Policy name	Type	Description
□	⊞ AWSLambdaBasicExe...	Custom...	
□	⊞ AWSLambdaBasicExe...	Custom...	
□	⊞ AWSLambdaBasicExe...	Custom...	
□	⊞ AWSLambdaBasicExe...	Custom...	
□	⊞ AWSLambdaBasicExe...	Custom...	
□	⊞ AWSLambdaBasicExe...	Custom...	
□	⊞ AWSLambdaS3Executi...	Custom...	
☑	⊟ newpolicylamdaec2lab	Custom...	



Step 1
Select trusted entity

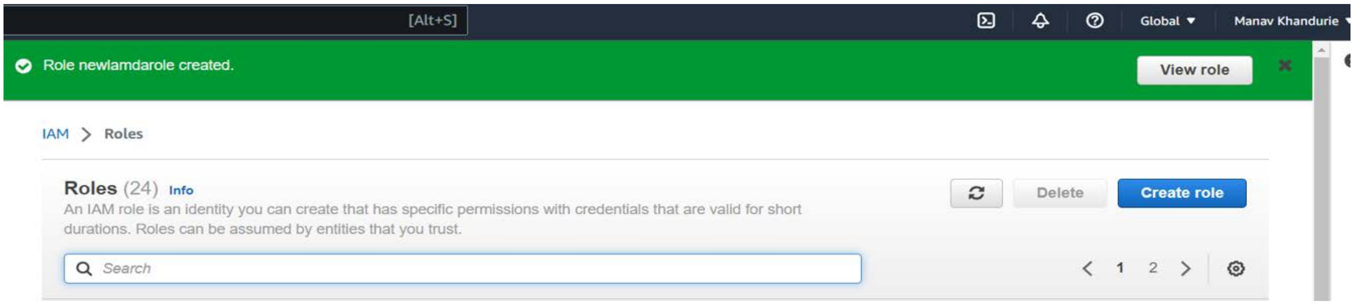
Step 2
Add permissions

Step 3
Name, review, and create

Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.
newlamdarole
Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

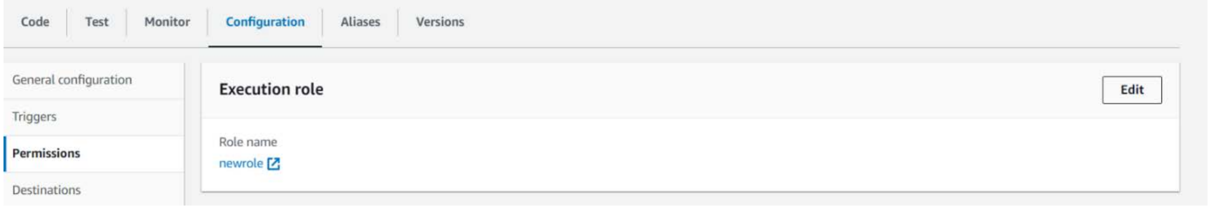


Role newlamdarole created.

IAM > Roles

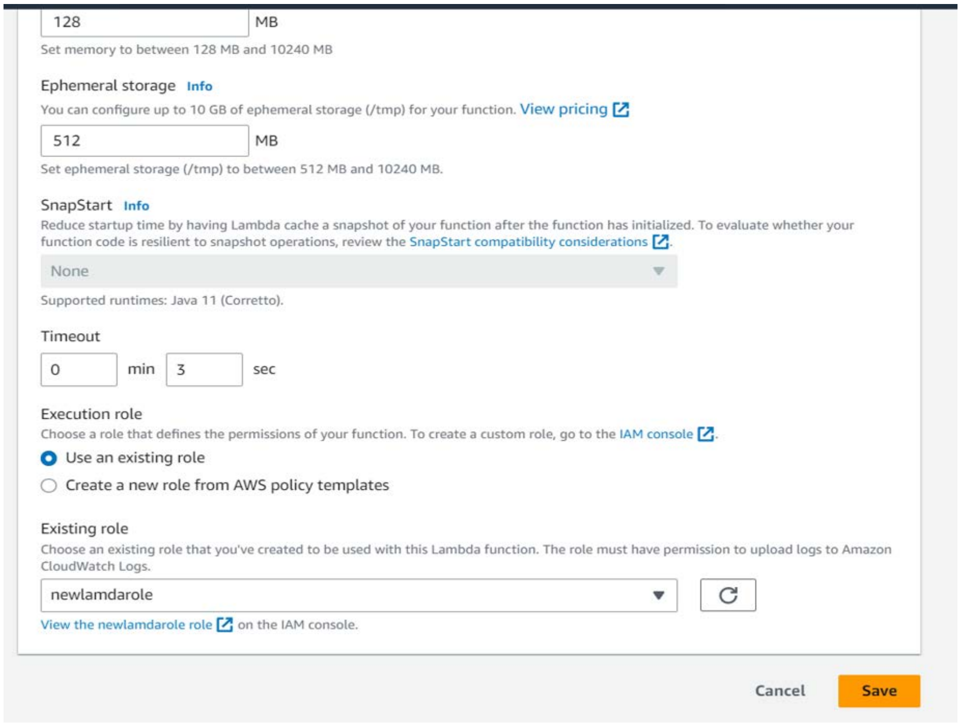
Roles (24) Info
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Step 9: Goto Lambda-> Function-> Configration->Exection Role & add created role



Execution role

Role name
newrole



128 MB

Set memory to between 128 MB and 10240 MB

Ephemeral storage Info

You can configure up to 10 GB of ephemeral storage (/tmp) for your function. View pricing

512 MB

Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.

SnapStart Info

Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is resilient to snapshot operations, review the SnapStart compatibility considerations.

None

Supported runtimes: Java 11 (Corretto).

Timeout

0 min 3 sec

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

● Use an existing role

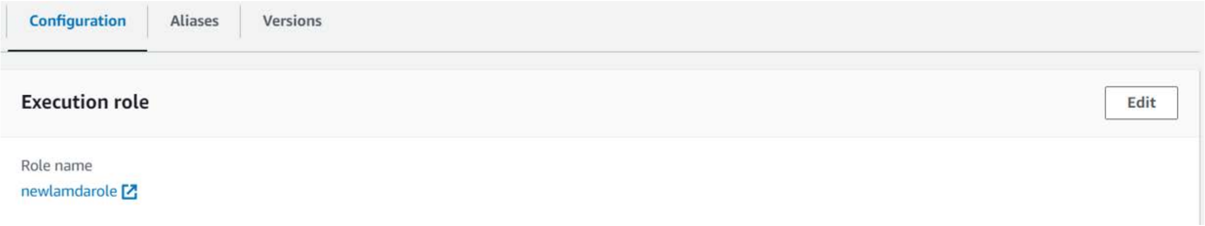
○ Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

newlamdarole

View the newlamdarole role on the IAM console.

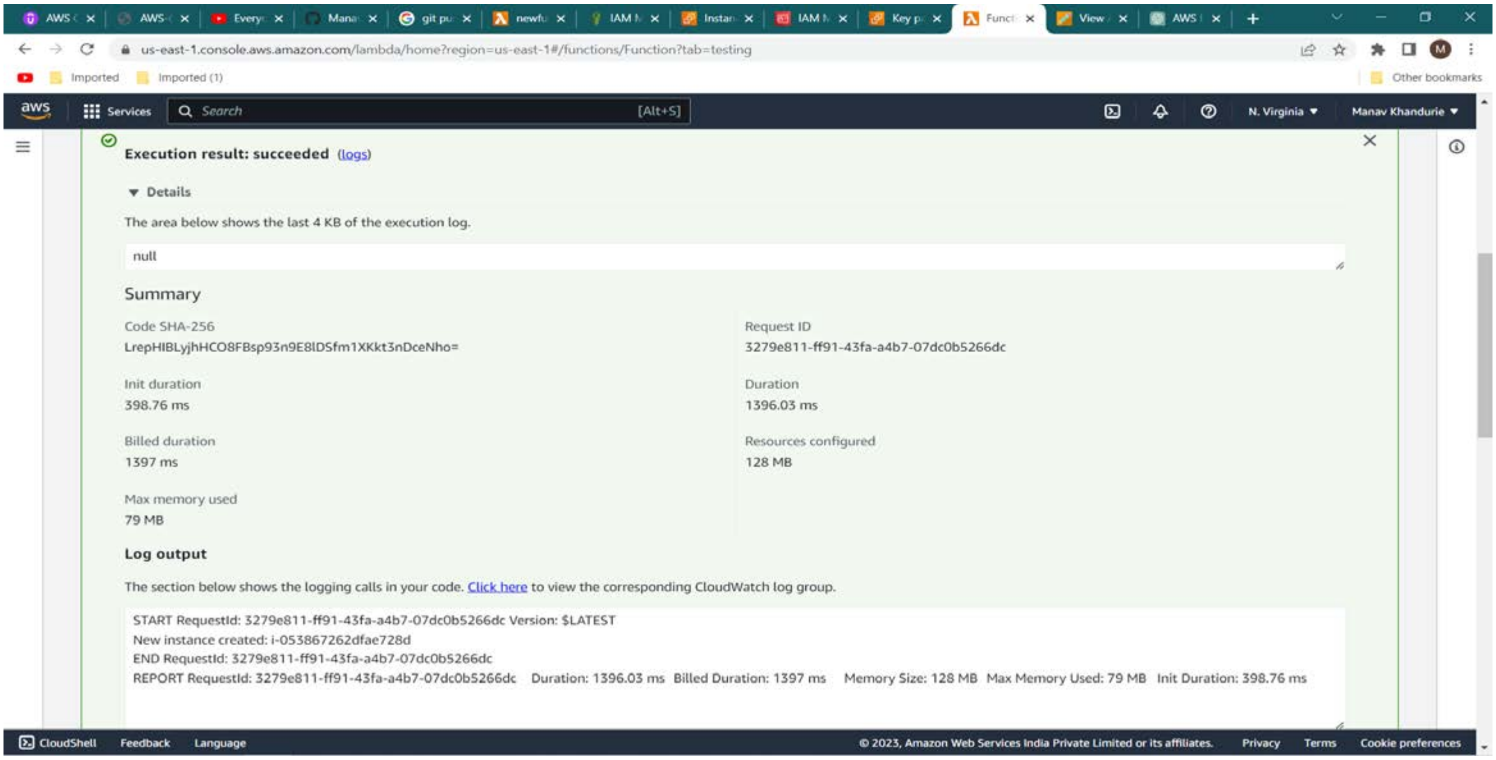


Execution role

Role name

newlamdarole

Step 10: Now goto Function -> Test



us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/Function?tab=testing

Execution result: succeeded (logs)

▼ Details

The area below shows the last 4 KB of the execution log.

```
null
```

Summary

Code SHA-256 LrepHIBLyjhHCO8FBsp93n9E8lDSfm1XKkt3nDceNho=	Request ID 3279e811-ff91-43fa-a4b7-07dc0b5266dc
Init duration 398.76 ms	Duration 1396.03 ms
Billed duration 1397 ms	Resources configured 128 MB
Max memory used 79 MB	

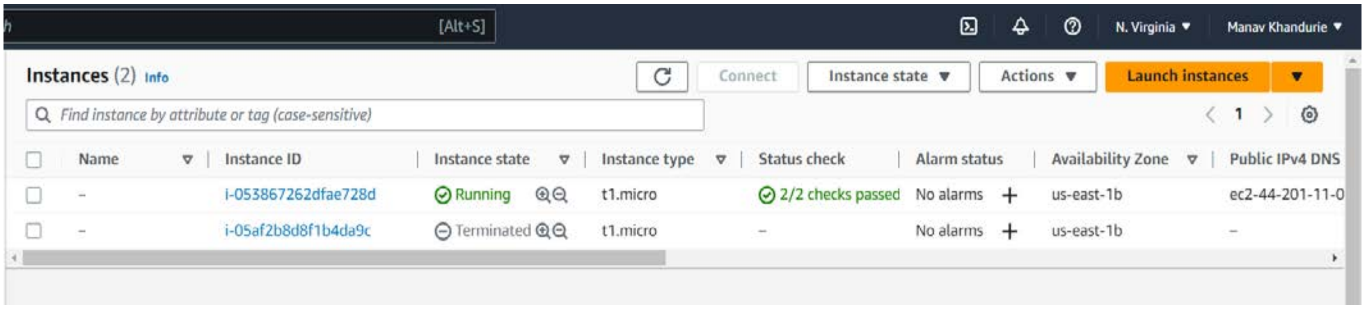
Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
START RequestId: 3279e811-ff91-43fa-a4b7-07dc0b5266dc Version: $LATEST
New instance created: i-053867262dfae728d
END RequestId: 3279e811-ff91-43fa-a4b7-07dc0b5266dc
REPORT RequestId: 3279e811-ff91-43fa-a4b7-07dc0b5266dc	Duration: 1396.03 ms	Billed Duration: 1397 ms	Memory Size: 128 MB	Max Memory Used: 79 MB	Init Duration: 398.76 ms
```

CloudShell
Feedback
Language
© 2023, Amazon Web Services India Private Limited or its affiliates.
Privacy
Terms
Cookie preferences

Step 11: Goto EC2 instances and the new instance would be present



Instances (2) Info

Find instance by attribute or tag (case-sensitive)

☐	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
☐	–	i-053867262dfae728d	✓ Running	t1.micro	✓ 2/2 checks passed	No alarms	us-east-1b	ec2-44-201-11-0
☐	–	i-05af2b8d8f1b4da9c	⊖ Terminated	t1.micro	–	No alarms	us-east-1b	–