Soham Satam Roll no. 109

Assignment 1

Aim : To make an EC2 machine on AWS.

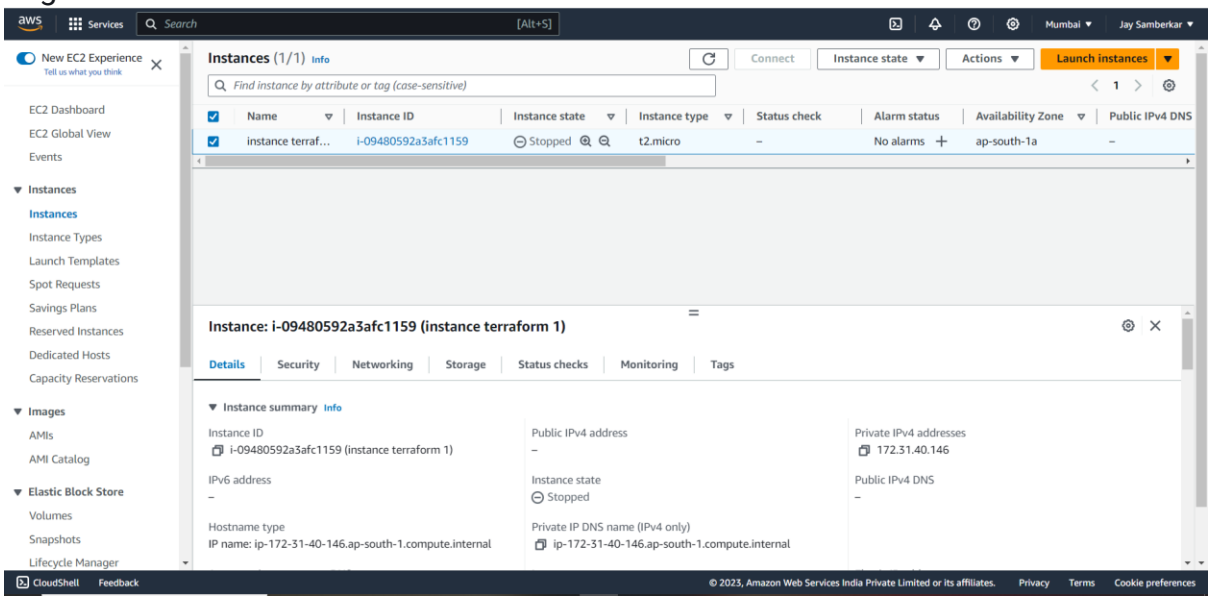
Theory :

Amazon Web Services is a widely used cloud computing platform provided by Amazon. AWS offers a vast array of cloud-based services, including computing power, storage solutions, networking, databases, machine learning, analytics, Internet of Things (IoT) and security.

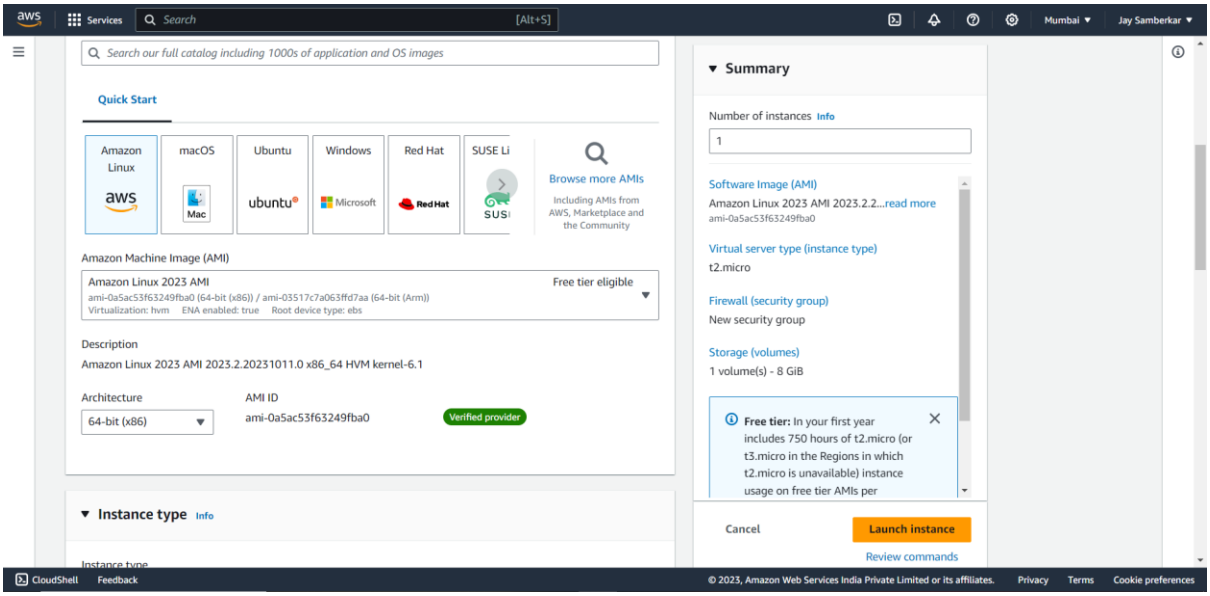
Amazon Elastic Compute Cloud (Amazon EC2) provides on-demand, scalable computing capacity in the AWS Cloud. Using Amazon EC2 reduces hardware costs so you can develop and deploy applications faster. We can use Amazon EC2 to launch virtual servers according to our needs. We can add capacity (scale up) to handle compute-heavy tasks or reduce capacity based on website traffic demands.

Steps :

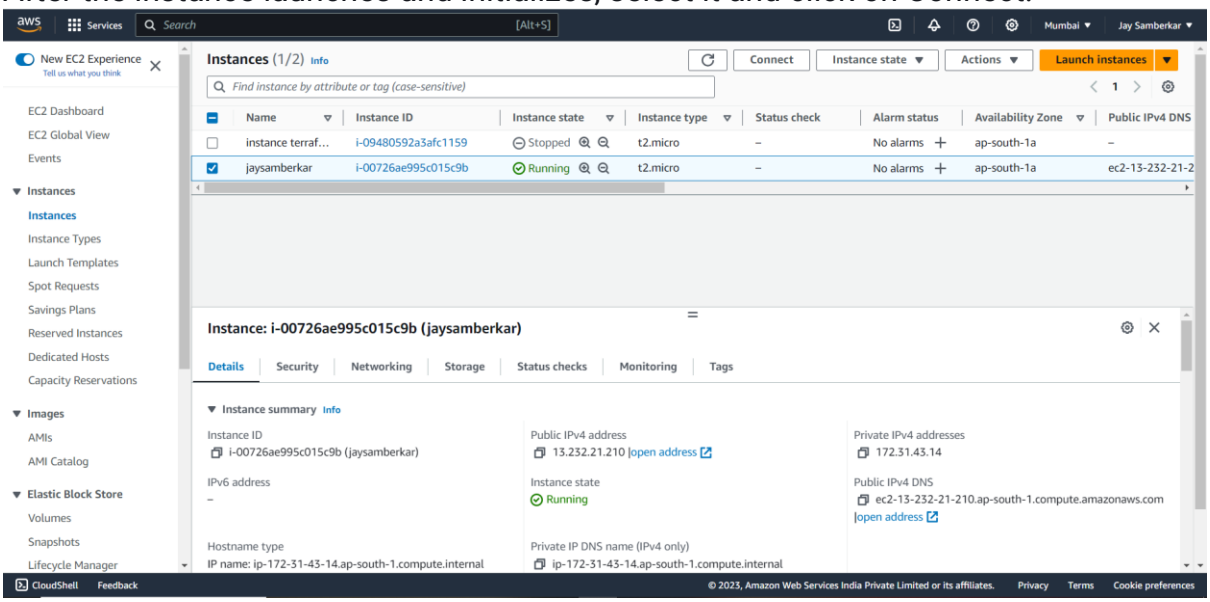
1. Login and Search EC2 and scroll down to Launch an Instance.



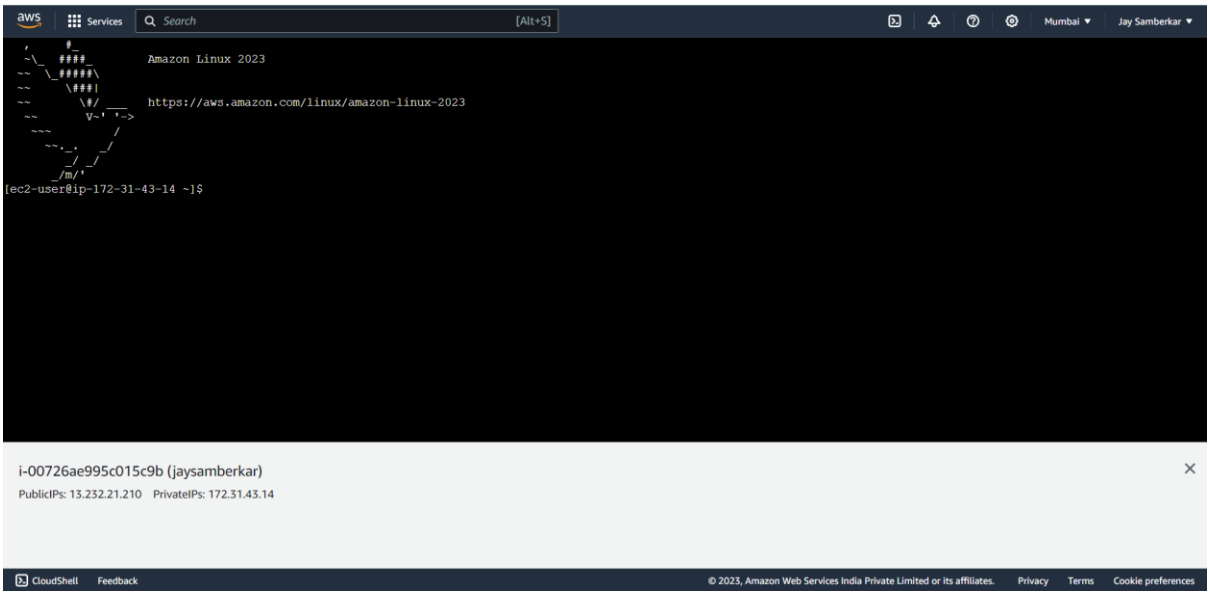
2. Click on launch instance and select the required machine and instance type (Free Tier Eligible) and then Launch the instance.



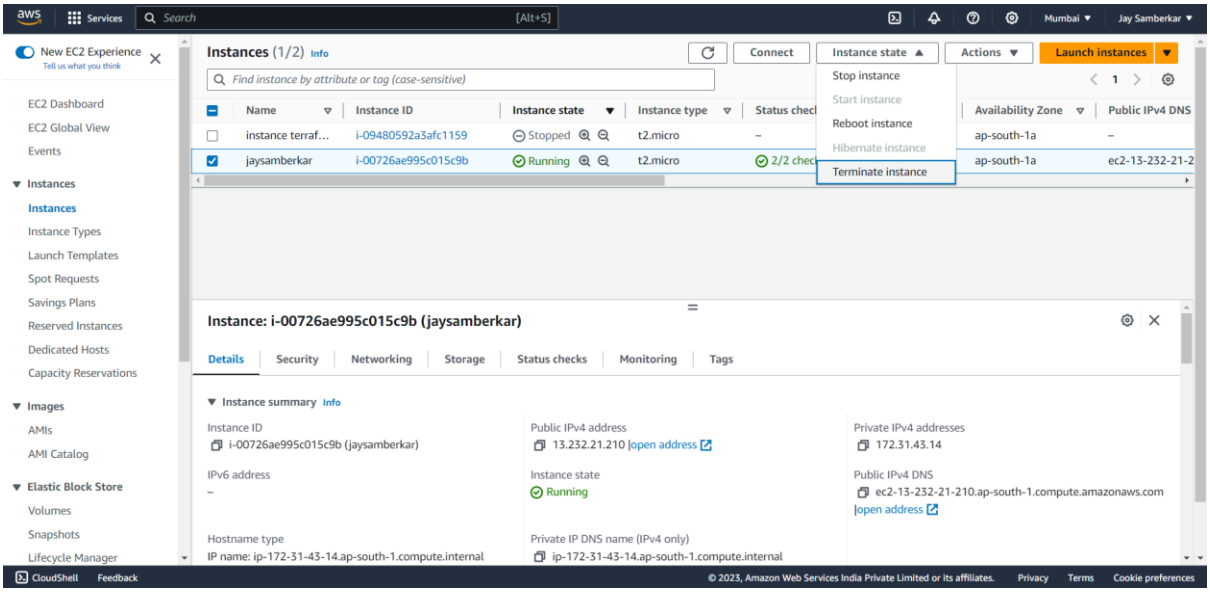
3. After the instance launches and initializes, select it and click on Connect.



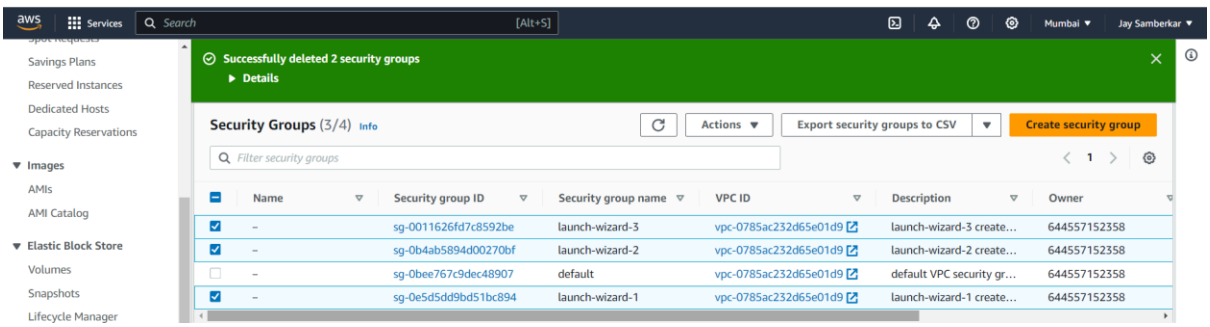
4. The machine will boot up to the console.



5. For shutting down the instance, go back and terminate the instance from the instance state menu.



6. Head to Security Groups and delete the non-default group from the Actions Menu.



Lab Outcome:

LO1: To understand the fundamentals of cloud computing and be fully proficient with Cloud based DevOps solution deployment options to meet your business requirements.

Conclusion :
Hence learned and implemented the steps to create an EC2 Machine

Assignment 2

Aim : To make and launch AWS Cloud9 IDE.

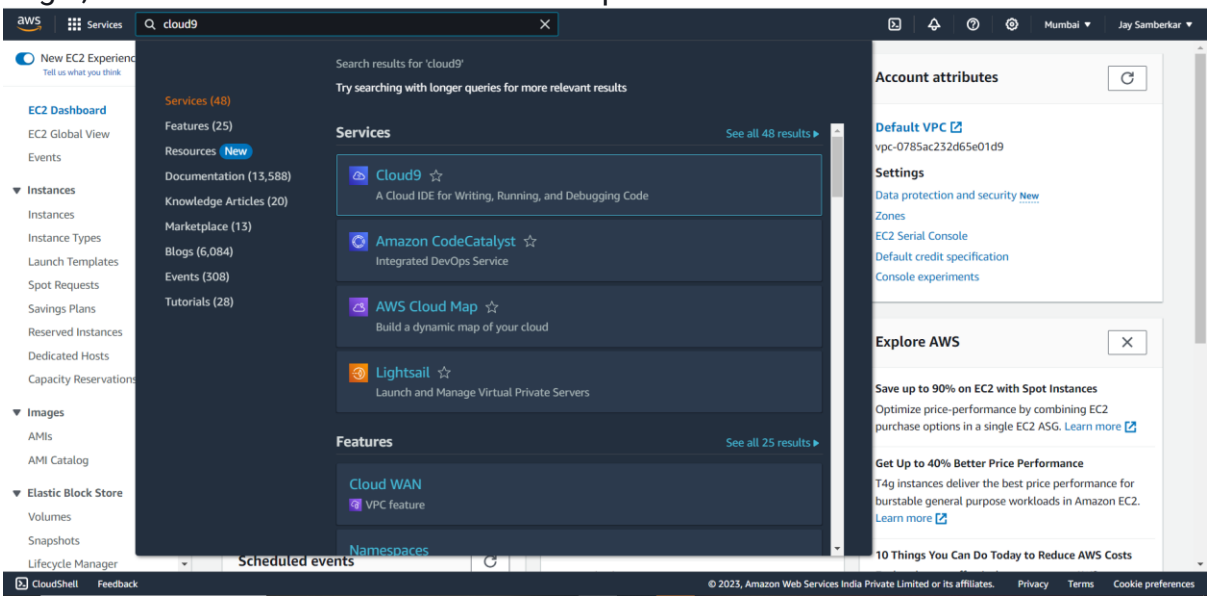
Theory :

AWS Cloud9, originally Cloud9 IDE before its acquisition by Amazon Web Services, revolutionized the development landscape by providing developers with a cloud-based integrated development environment (IDE) that facilitated coding, debugging, and running code with remarkable ease and efficiency.

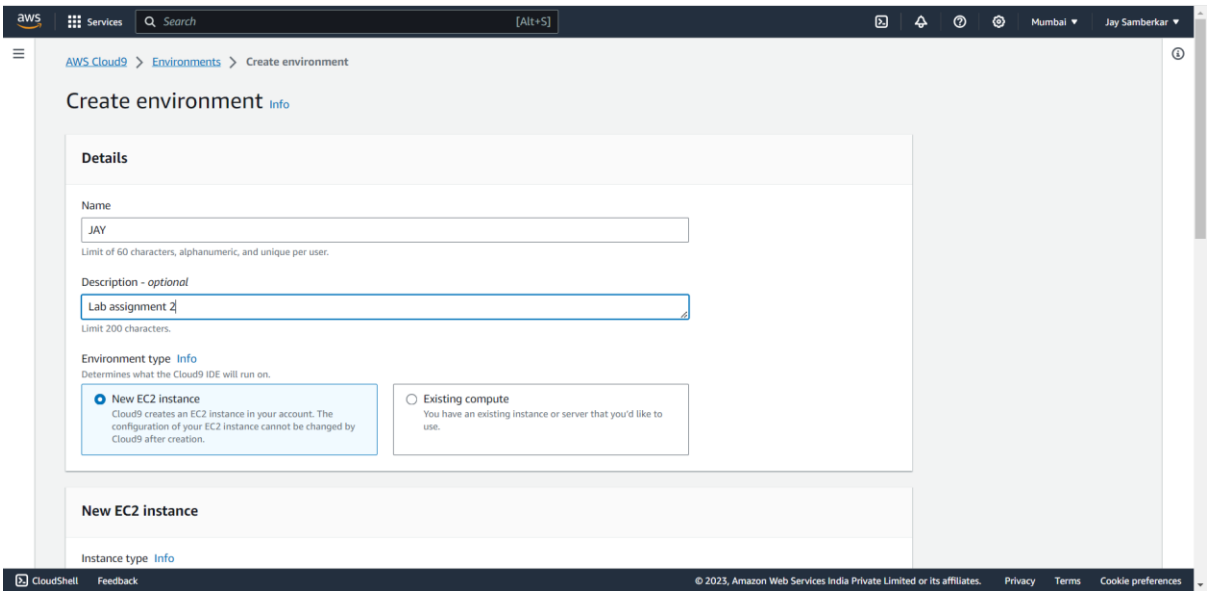
The inclusion of a terminal within the IDE allowed developers to execute commands and run server-side code directly from the cloud. Beyond its accessibility and flexibility, AWS Cloud9 prioritized data security and scalability, ensuring a reliable and safe environment for businesses and individual developers seeking cutting-edge cloud-based development solutions.

Steps :

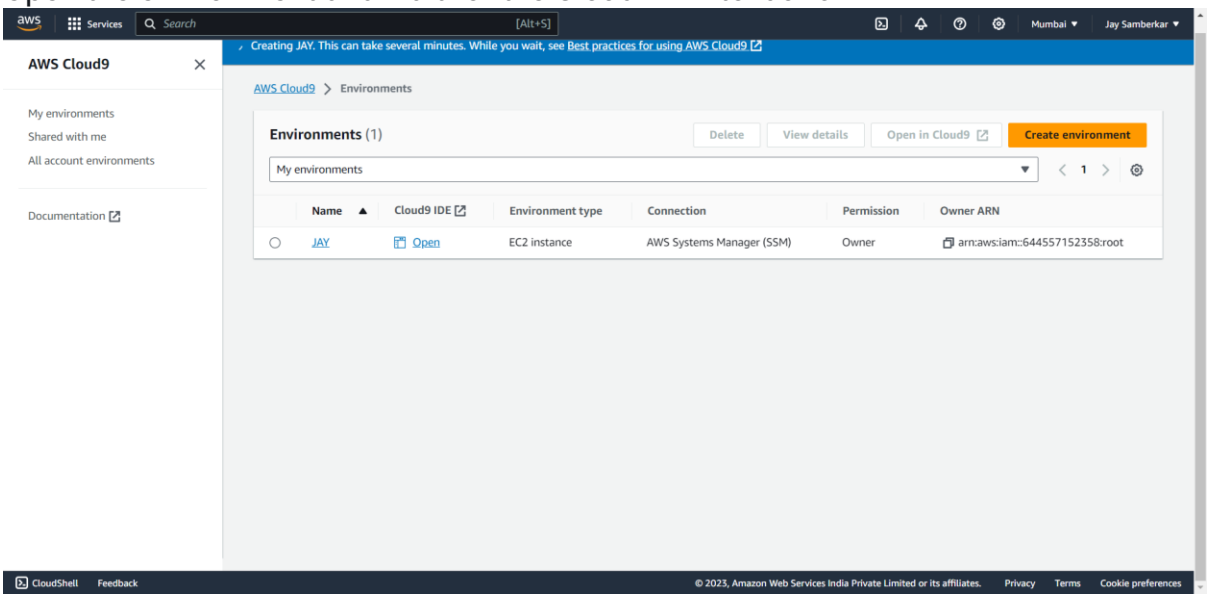
1. Login, Search Cloud9 and select the first option.



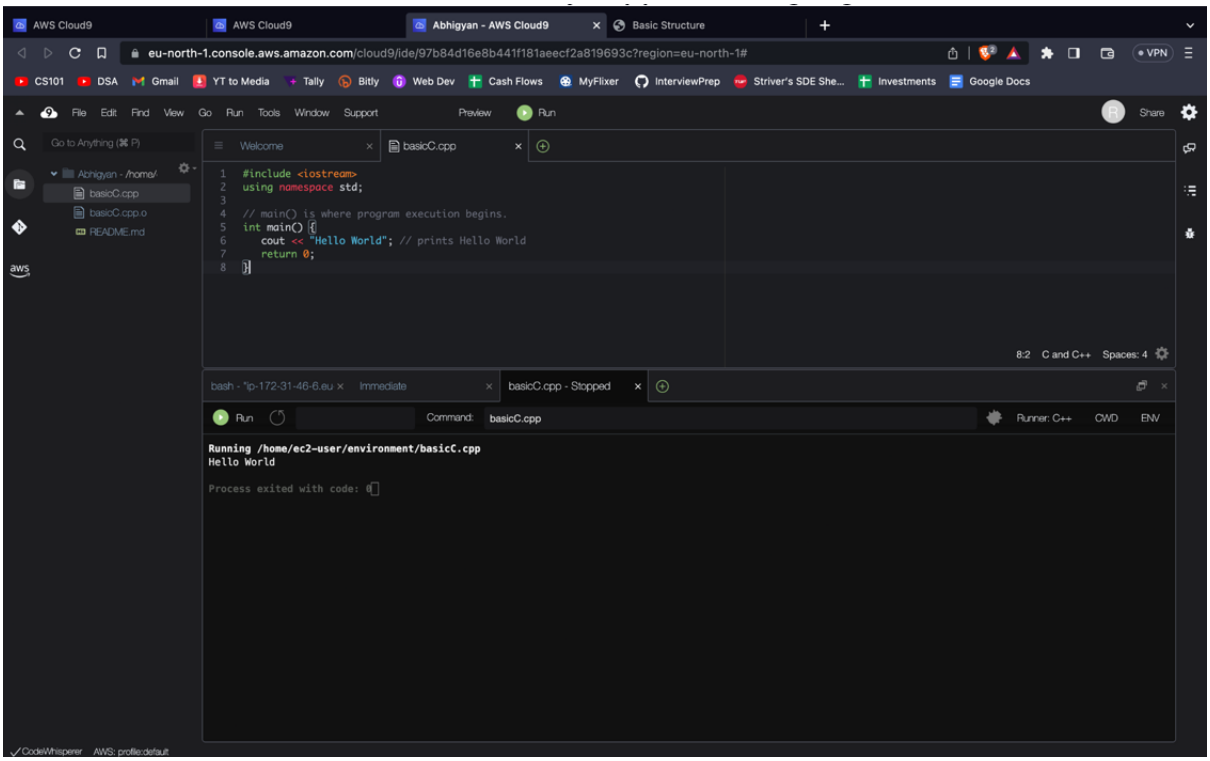
2. Fill in the name and description. Then scroll down and click on Create an Environment.



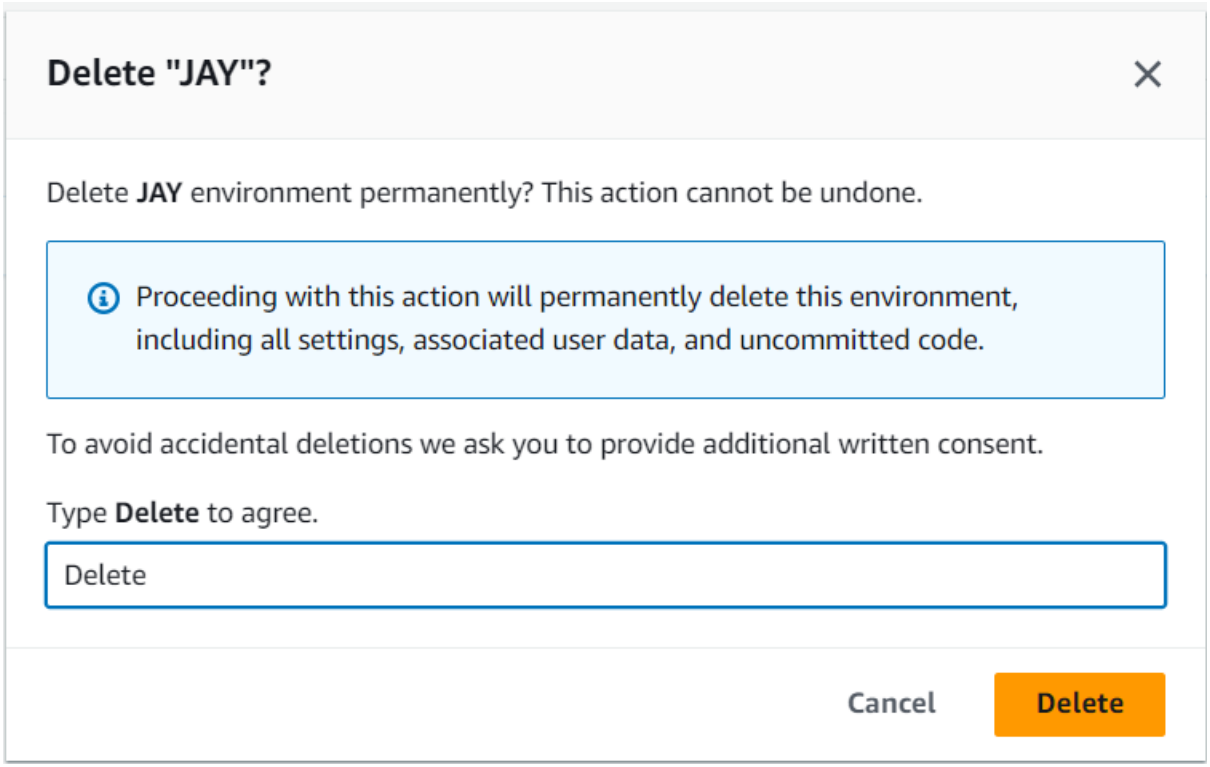
3. Open the environment and wait for the Cloud9 IDE to launch



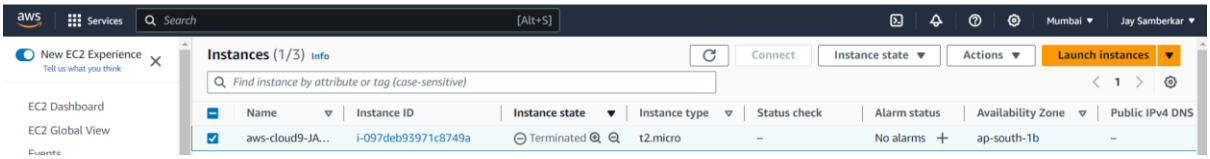
4. Create a new file, write some code in any supported language and execute it.



5.Go back to Dashboard,then click on the delete button, type in Delete and Delete the environment.



6.Lastly, head over to your EC2 instances and terminate the instance related to your cloud9 environment.



Lab Outcome:
LO1, LO2 mapped.

Conclusion :
We understood the benefits of Cloud Infrastructure and Setup AWS Cloud9 IDE, Launched AWS Cloud9 IDE and performed Collaboration Demonstration.

Soham Satam Roll N0. 109

ASSIGNMENT No. – 3

Aim: To build your application using AWS CodeBuild and Deploy on S3/SEBS using AWS CodePipeline.

LO Mapped: LO1, LO2

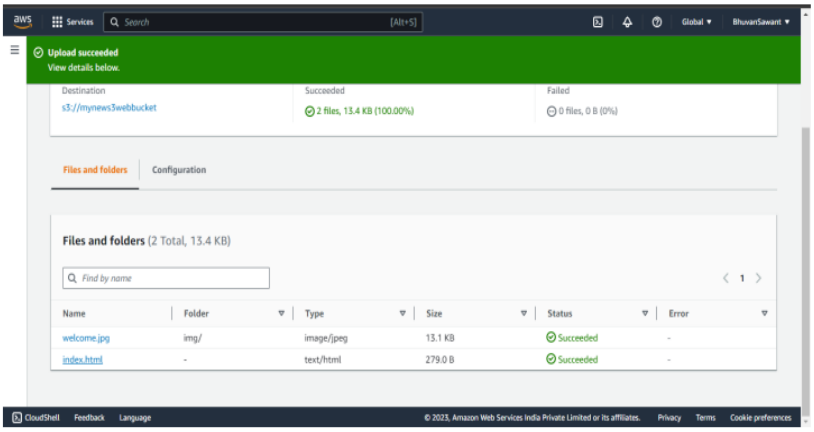
Theory:

Amazon Simple Storage Service (Amazon S3) is an obiect storage service offering industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can store and protect any amount of data for virtually any use case, such as data lakes, cloud-native applications, and mobile apps. With cost-effective storage classes and easy-to-use management features, you can optimize costs, organize data, and configure fine-tuned access controls to meet specific business, organizational, and compliance requirements.

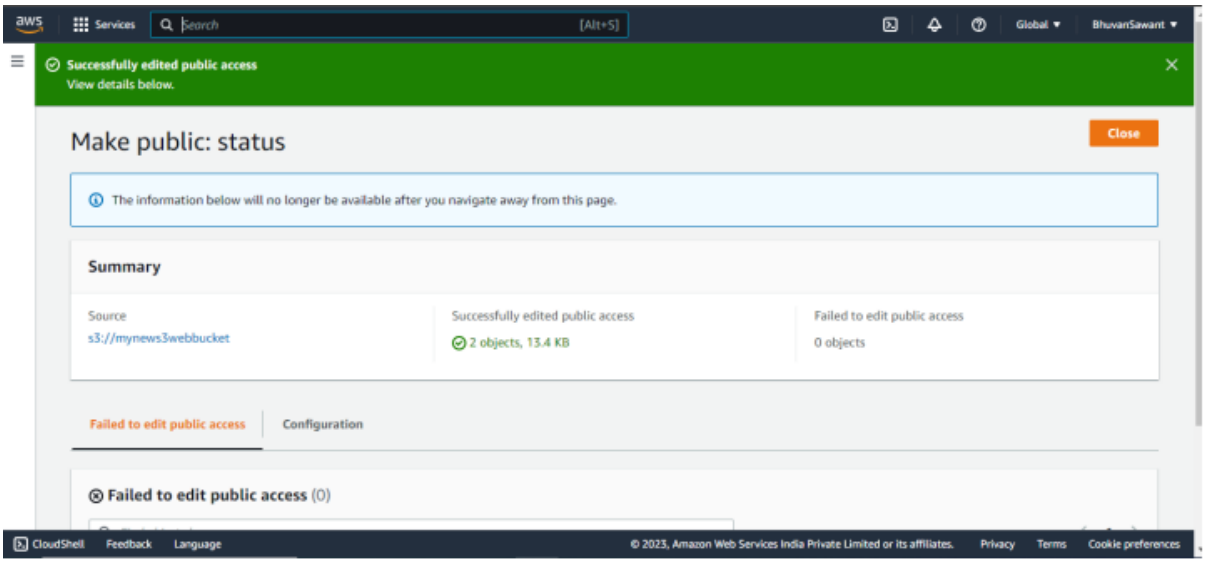
Some of the benefits of AWS S3 are:

- Durability: S3 provides 99.999999999 percent durability.
- Low cost: S3 lets you store data in a range of "storage classes." These classes are based on the frequency and immediacy you require in accessing files.
- Scalability: S3 charges you only for what resources you actually use, and there are no hidden fees or overage charges. You can scale your storage resources to easily meet your organization's ever-changing demands.
- Availability: S3 offers 99.99 percent availability of objects
- Security: S3 offers an impressive range of access management tools and encry'ption features that provide top-notch security.
- Flexibility: S3 is ideal for a wide range of uses like data storage, data backup, software delivery, data archiving, disaster recovery, website hosting, mobile applications, IoT devices, and much more.
- Simple data transfer: You don't have to be an IT genius to execute data transfers on S3. The service revolves around simplicity and ease of use.

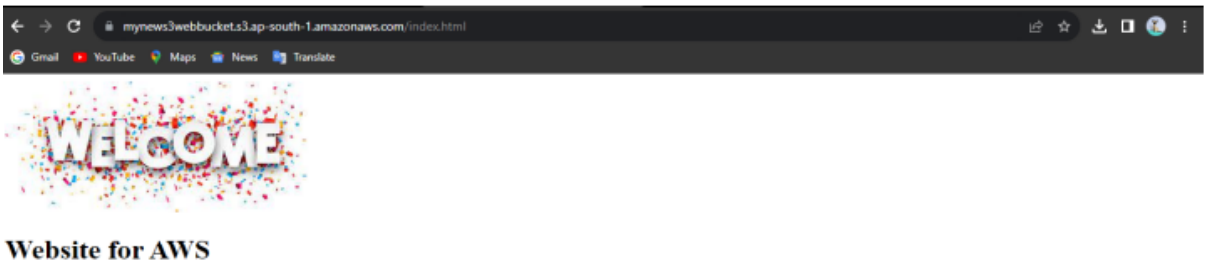
First login to your AWS account.



Then create a new S3 bucket.



After that host a static web page using the S3 bucket.



Conclusion: In this assignment we learnt about AWS S3 bucket and hosted a static web page using the S3 bucket.

Assignment Number: 4

Name: Soham Satam Branch: IT/V Roll No.:109 Date:10/09/2023

Aim: To study AWS Code Pipeline and deploy web application using Code Pipeline.

LO mapped: LO1, LO2

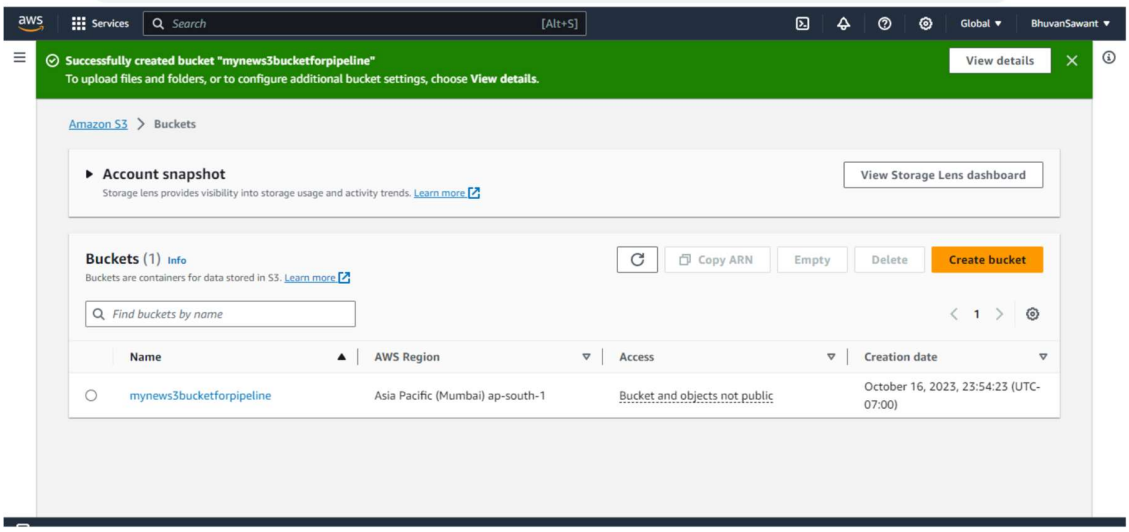
Theory:

AWS CodePipeline is a continuous integration and continuous delivery (CI/CD) service provided by Amazon Web Services (AWS).

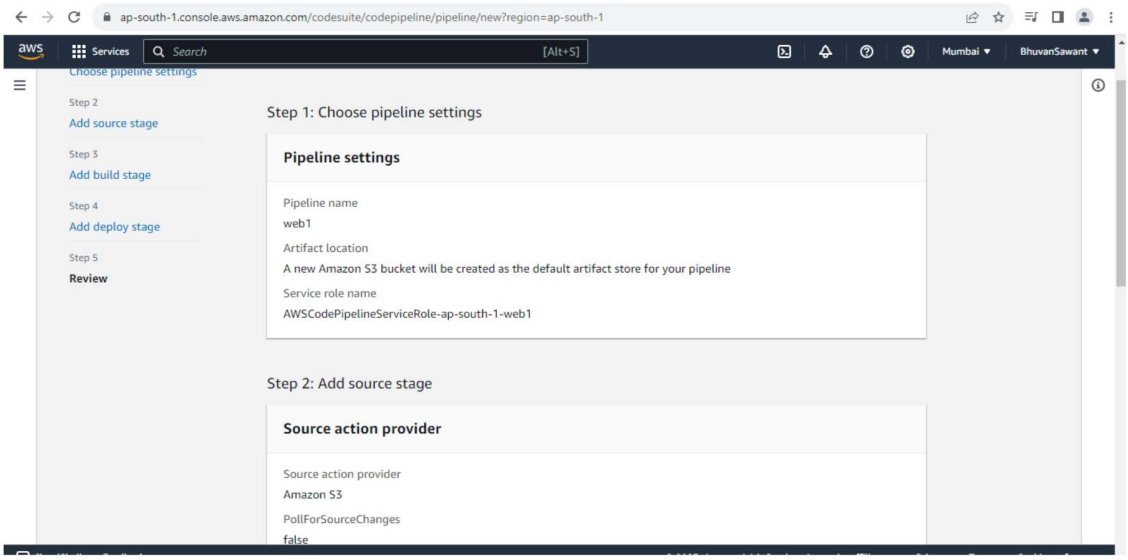
The key aspects of AWS CodePipeline:

Overview:

1. **CI/CD Workflow:**
    - AWS CodePipeline facilitates the automation of the build, test, and deployment phases of the release process. It allows you to define a series of stages, each of which can represent a phase in your release pipeline.

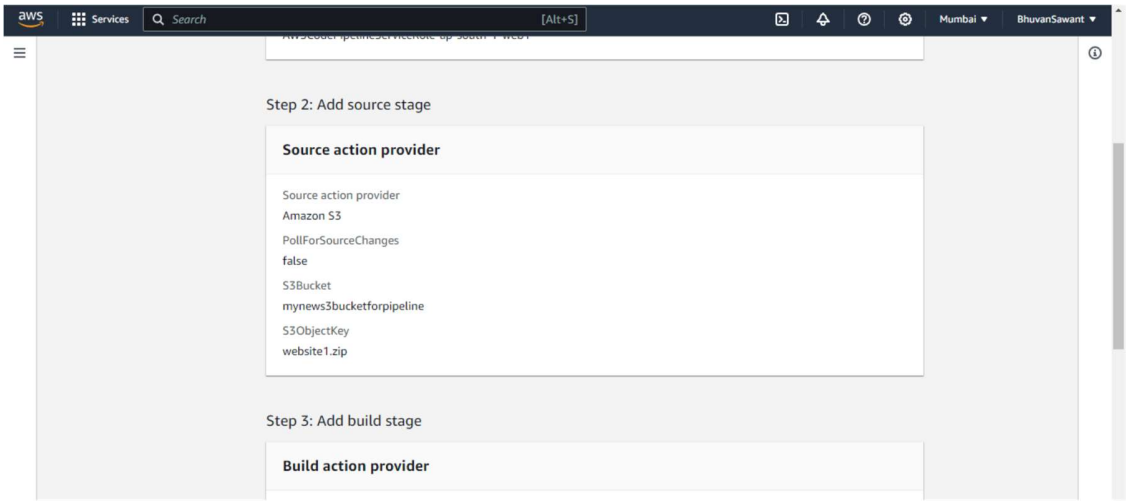


2. **Integration with Other AWS Services:**
    - CodePipeline integrates with various AWS services, such as AWS CodeBuild for building applications, AWS CodeDeploy for automating deployments, and AWS Lambda for running custom actions.

- 

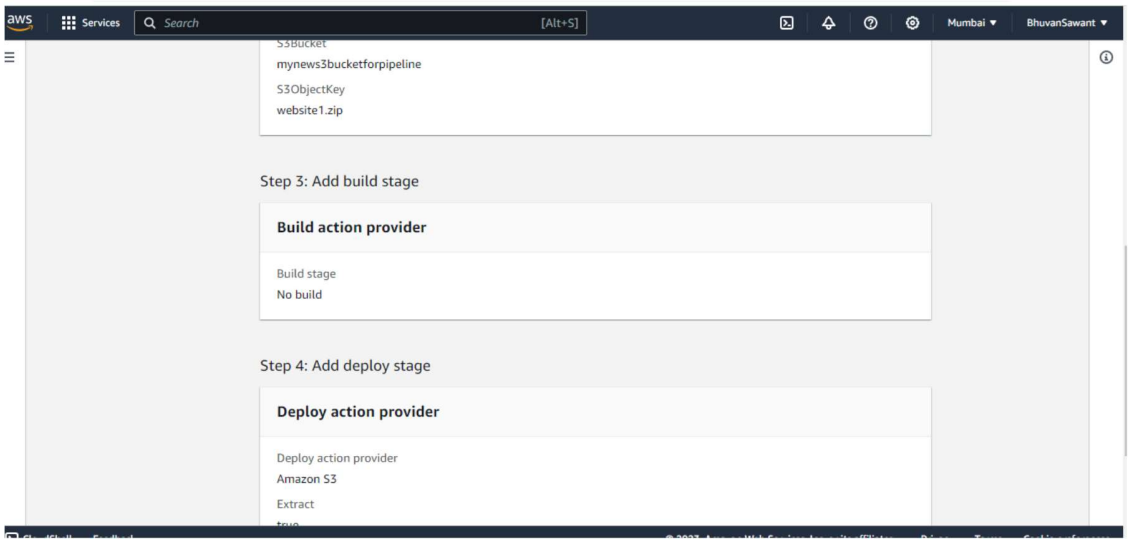
3. **Pipeline Execution:**

    - Pipelines consist of a series of stages, and each stage can have one or more actions. Actions represent a task, such as source code retrieval or deployment to a specific environment.



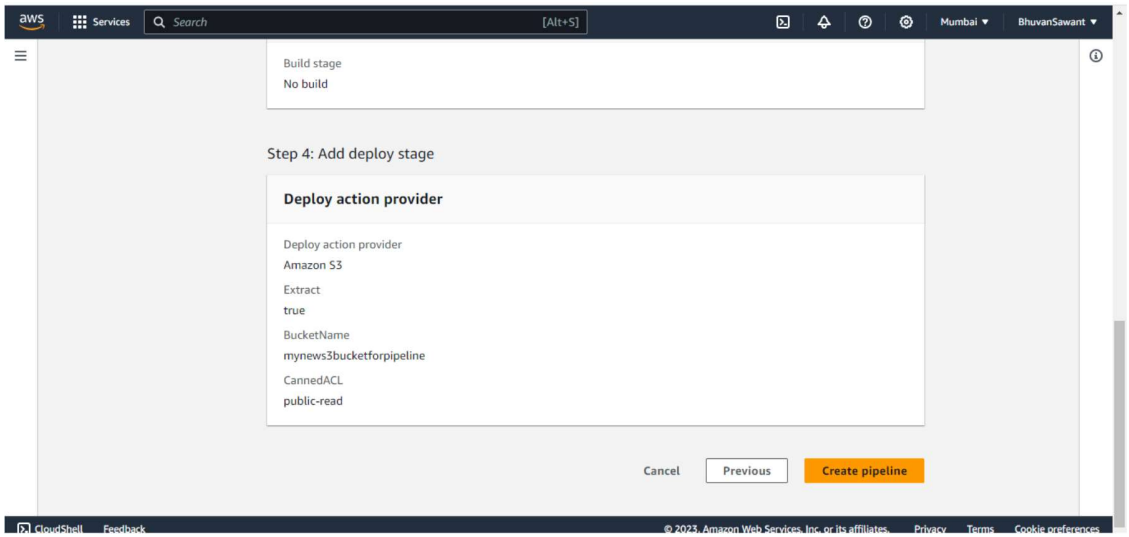
4. **Source Providers:**

    - CodePipeline supports integration with various source code repositories, including AWS CodeCommit, GitHub, and Amazon S3.

- 

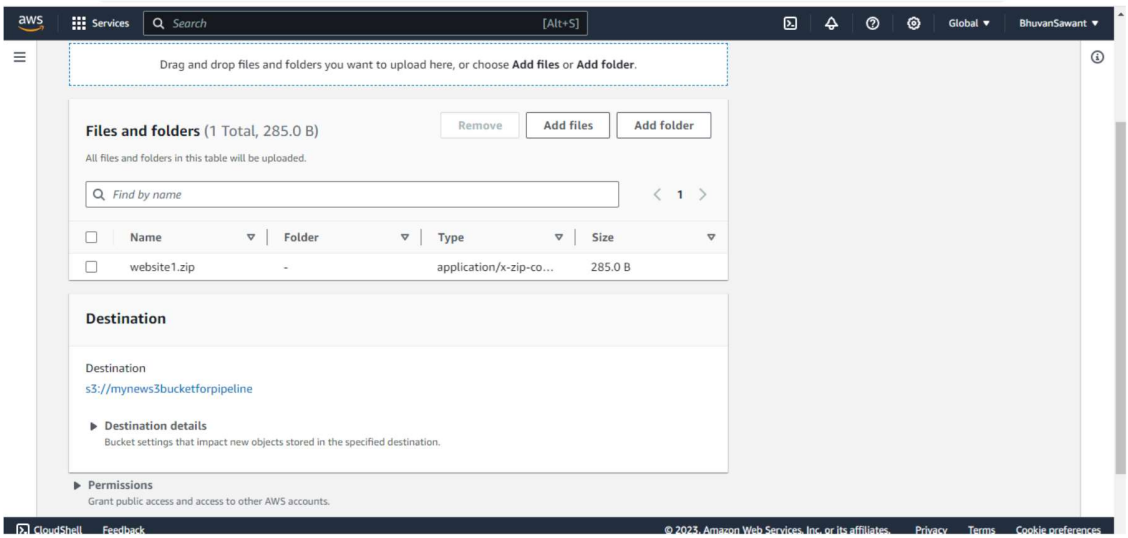
5. **Artifact Management:**

   - CodePipeline uses artifacts to store the files and data needed for each action in a pipeline. Artifacts can be passed between stages to ensure consistency in the deployment process.

- 

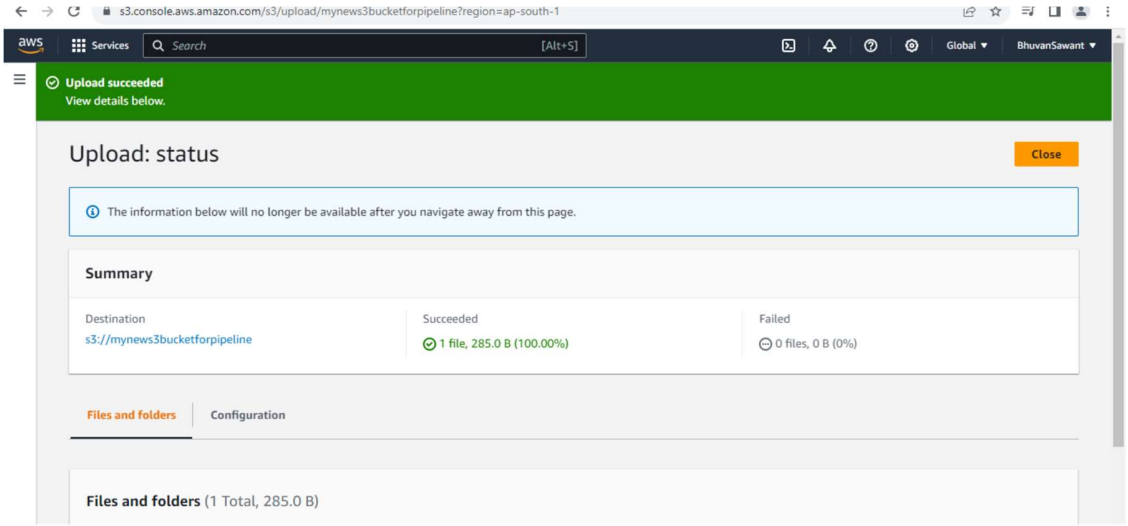
6. **Integration with Third-Party Tools:**

   - Besides AWS services, CodePipeline supports integration with third-party tools. This is achieved through custom actions, which allow you to use external tools and scripts in your pipeline.

- 

7. **Pipeline Visualizations:**

   - CodePipeline provides a visual representation of your release process, making it easy to understand and monitor the status of each stage and action.

- 

Key Concepts:

1. **Pipeline:**

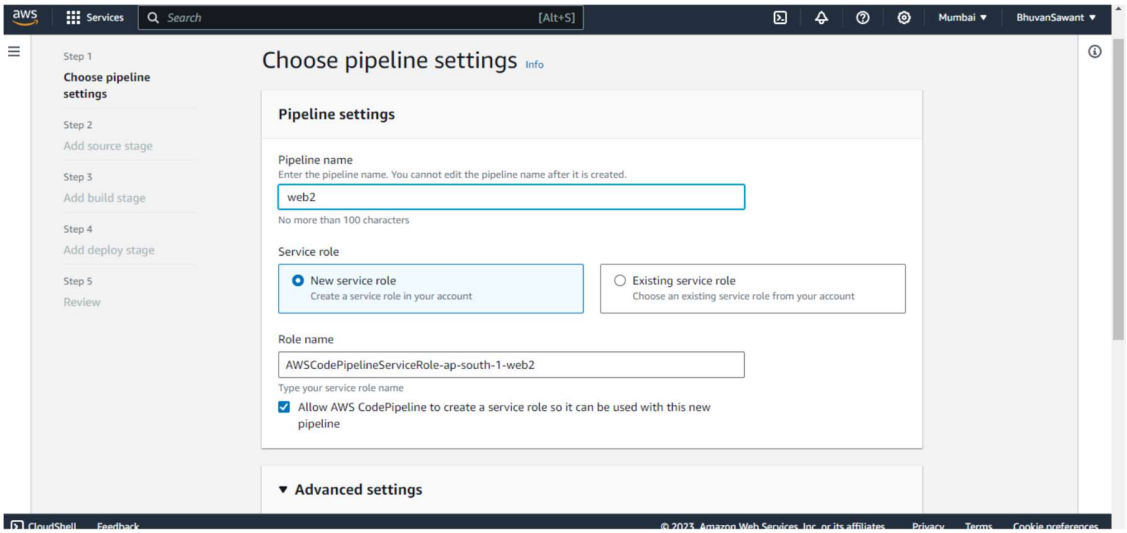
   - A pipeline is a series of stages that represents your release process. Each stage can contain one or more actions.

2. **Stage:**

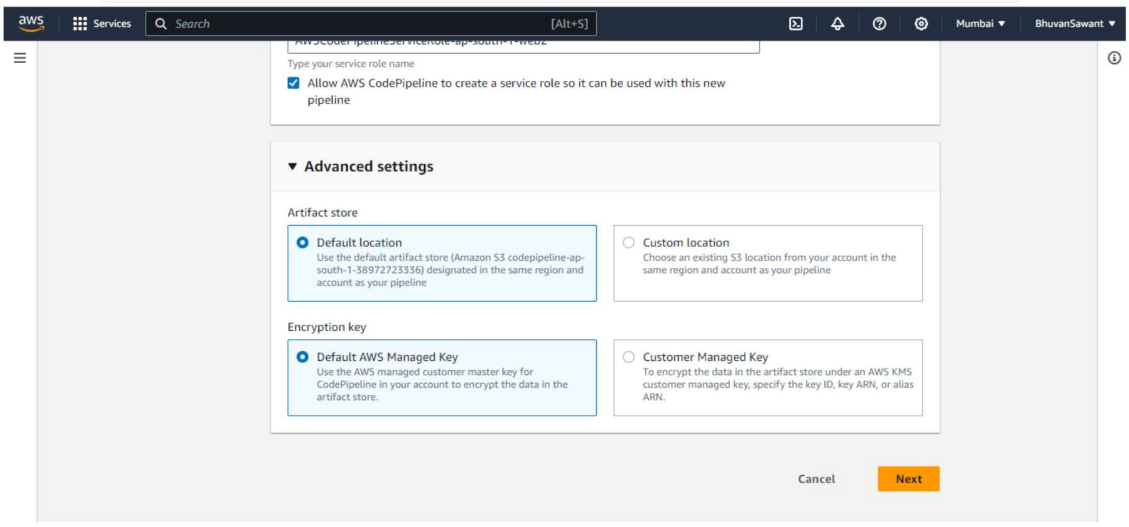
   - A stage is a logical unit in a pipeline, representing a phase in the release process. Stages are executed sequentially.

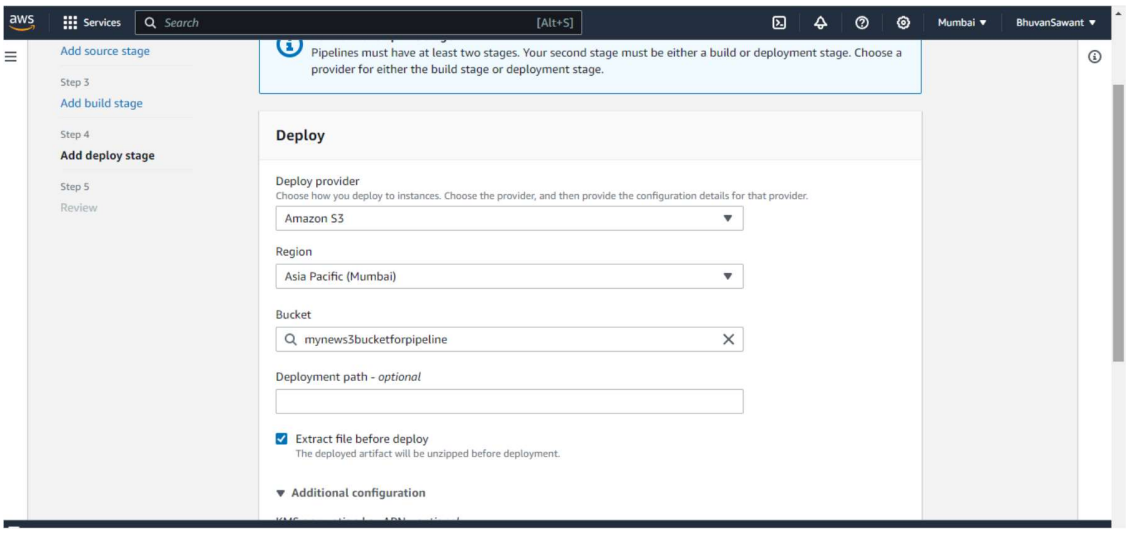
3. **Action:**
    - An action represents a task within a stage. Actions can include tasks such as building code, deploying to a test environment, or running tests.
4. **Artifact:**
    - Artifacts are the files and data that are produced as a result of an action. They are used to pass information between stages in a pipeline.

To deploy web application using CodePipeline here are the following steps to be followed:

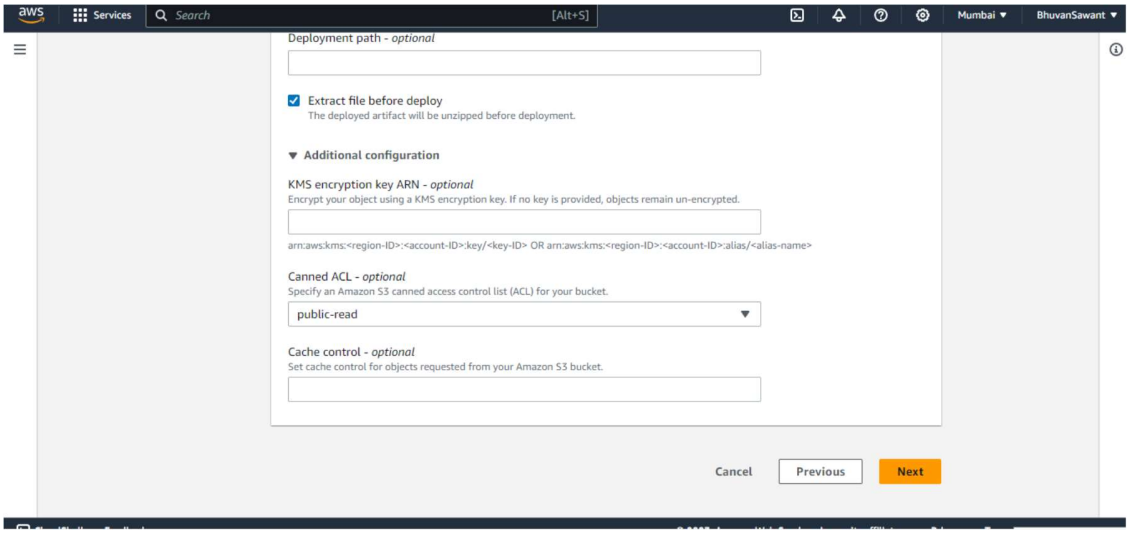
1. **Set Up Source Stage:**
    - Configure a source stage in AWS CodePipeline, linking to your version control system (e.g., CodeCommit, GitHub).
    - 

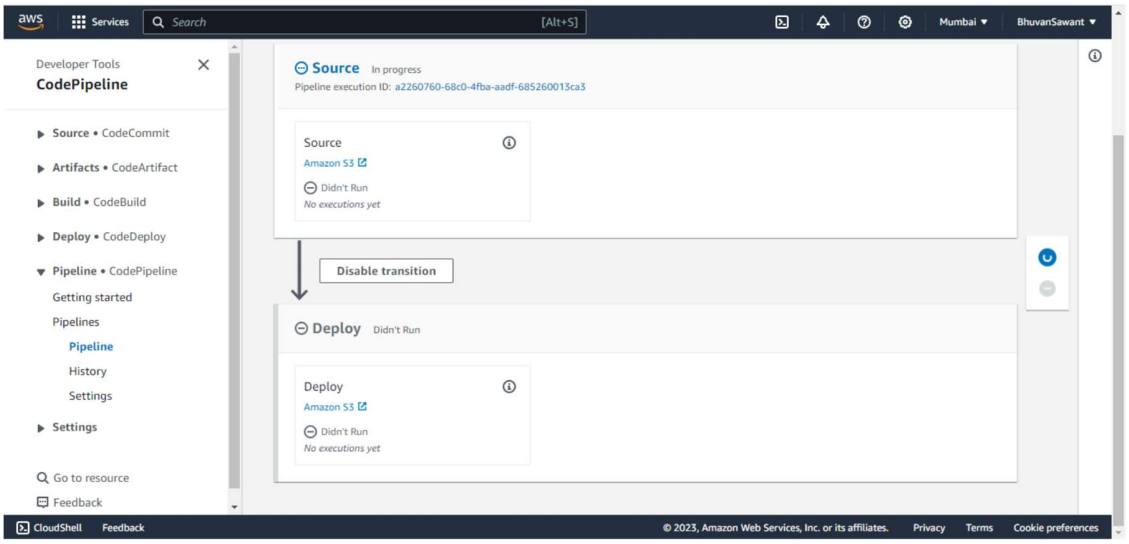
2. **Configure Build Stage:**
    - Set up a build stage using AWS CodeBuild to compile, test, and package your web application.

- 

3. **Define Deployment Stage:**
   - Create a deployment stage using AWS CodeDeploy or another deployment provider to deploy your application to target environments.
   - 

4. **Configure Approval (Optional):**
   - Optionally, add a manual approval stage to review and approve deployments before proceeding to the next stage.

- 

5. **Artifact Passing:**
    - Ensure proper passing of artifacts between stages to maintain consistency in the deployment process.
    - 

6. **Add Monitoring (Optional):**
    - Integrate monitoring tools (e.g., AWS CloudWatch) to track the performance and health of your application during and after deployment.
7. **Configure Notifications (Optional):**
    - Set up notifications using AWS SNS or other services to receive alerts about pipeline events and status changes.
8. **Test and Validate:**

- Test the pipeline by triggering a build, ensuring that each stage executes successfully, and the application deploys as expected.

9. **Modify Pipeline as Needed:**
    - Make adjustments to the pipeline configuration based on the specific requirements of your web application and deployment process.
10. **Continuous Improvement:**
    - Implement continuous improvement practices, such as monitoring feedback, optimizing build and deployment scripts, and iterating on the pipeline structure.

Conclusion: By this assignment we learned how to host static web page through codepipeline.

ASSIGNMENT 5

Aim – To learn basic idea of Kubernetes.

Theory-

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

The name Kubernetes originates from Greek, meaning helmsman or pilot. K8s as an abbreviation results from counting the eight letters between the "K" and the "s". Google open-sourced the Kubernetes project in 2014. Kubernetes combines over 15 years of Google's experience running production workloads at scale with best-of-breed ideas and practices from the community.

Let's take a look at why Kubernetes is so useful by going back in time.

Traditional deployment era: Early on, organizations ran applications on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues. For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform. A solution for this would be to run each application on a different physical server. But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers.

Virtualized deployment era: As a solution, virtualization was introduced. It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.

Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more. With virtualization you can present a set of physical resources as a cluster of disposable virtual machines. Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.

Container deployment era: Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. Similar to a VM,

a container has its own filesystem, share of CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

Why you need Kubernetes and what it can do

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behavior was handled by a system?

That's how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system. Kubernetes provides you with:

- **Service discovery and load balancing** Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
- **Storage orchestration** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
- **Automated rollouts and rollbacks** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
- **Automatic bin packing** You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.
- **Self-healing** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- **Secret and configuration management** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

Understanding Kubernetes objects

Kubernetes objects are persistent entities in the Kubernetes system. Kubernetes uses these entities to represent the state of your cluster. Specifically, they can describe:

What containerized applications are running (and on which nodes)

The resources available to those applications

The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance

A Kubernetes object is a "record of intent"--once you create the object, the Kubernetes system will constantly work to ensure that object exists. By creating an object, you're effectively telling the Kubernetes system what you want your cluster's workload to look like; this is your cluster's desired state.

To work with Kubernetes objects--whether to create, modify, or delete them--you'll need to use the Kubernetes API. When you use the kubectl command-line interface, for example, the CLI makes the necessary Kubernetes API calls for you. You can also use the Kubernetes API directly in your own programs using one of the Client Libraries.

Conclusion – Hence learned what is Kubernetes.

Assignment 6

Aim : To understand terraform lifecycle, core concepts/ terminologies and install it.

Theory :

Terraform, developed by HashiCorp, is an open-source infrastructure as code (IaC) software tool that allows users to define and provision data center infrastructure using a declarative configuration language. In simpler terms, it lets you codify your infrastructure, enabling consistent and reproducible deployments. It is used for defining, provisioning, and managing cloud infrastructure using a declarative language.

Core Concepts:

Providers: Terraform uses providers to interact with cloud services. Examples include AWS, Azure, Google Cloud, and many others. Each provider offers resource types that can be managed.

Resources: These are the primary components in Terraform. A resource might be a physical component such as an EC2 instance in AWS or a database in Azure.
State: Terraform maintains a state file that maps real-world resources to your configuration. This state is used to determine what Terraform will do on the next apply.

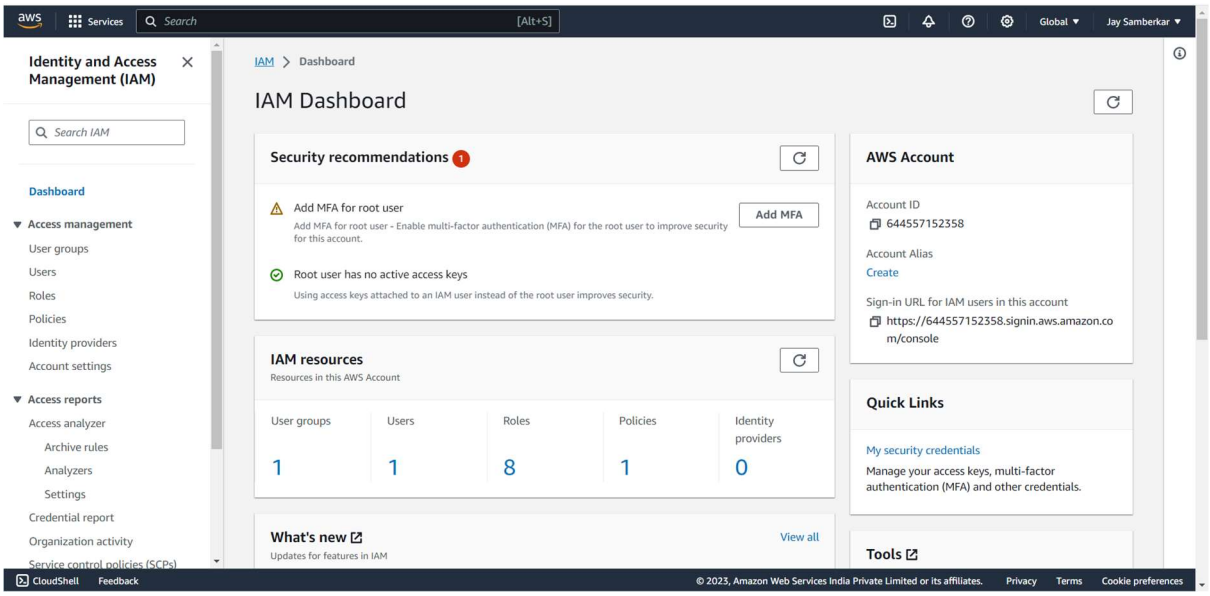
Modules: These are containers for multiple resources that are used together. They provide a way to group resources, and they can be used to create reusable infrastructure components.

Variables and Outputs: Variables allow for parameterization of the Terraform configuration, making it more dynamic and flexible. Outputs are a way to get information about the infrastructure, like IP addresses or DNS names.

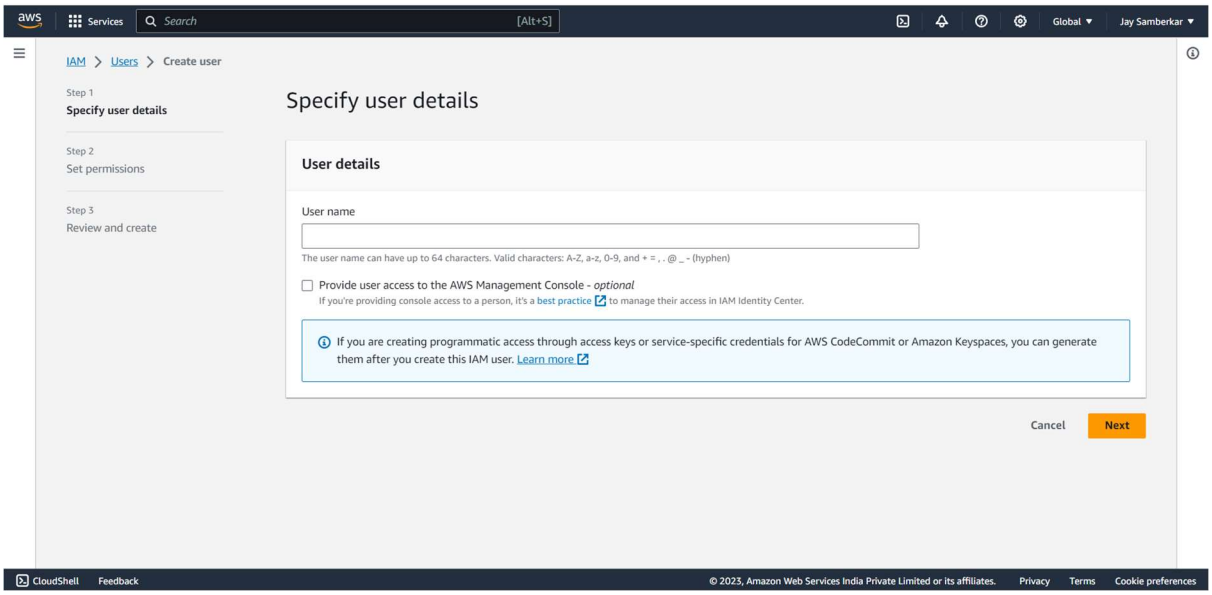
Provisioners: While not always recommended due to their imperative nature, provisioners can be used to model specific actions on the local machine or on a remote machine, like executing scripts.

Steps :

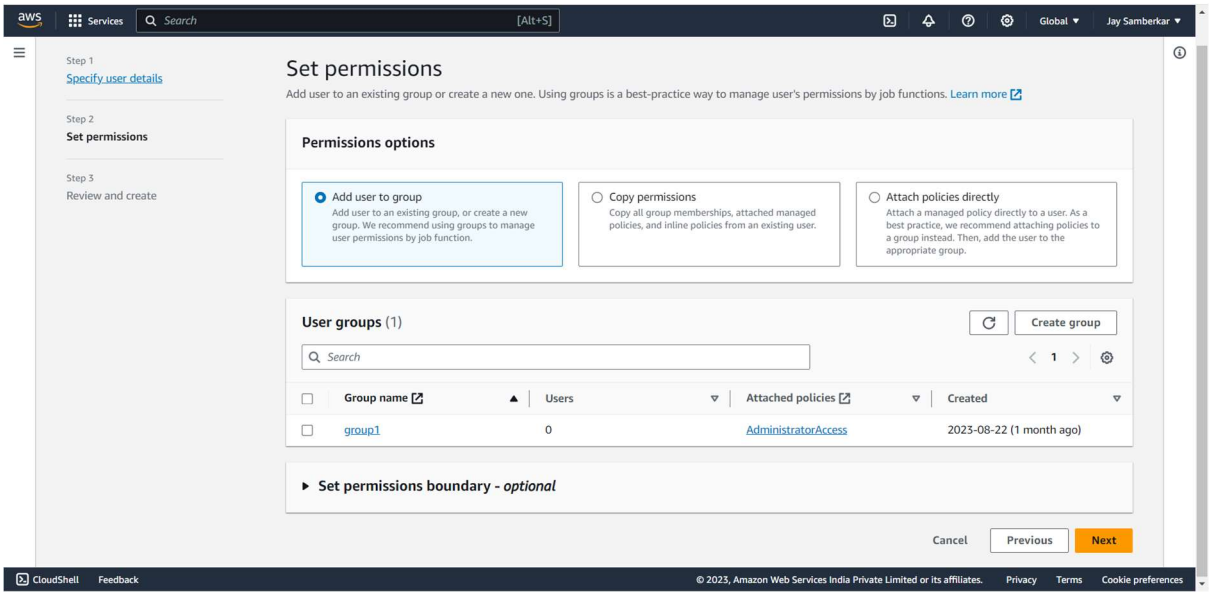
1. Login and Search IAM on AWS Console and select the first option.



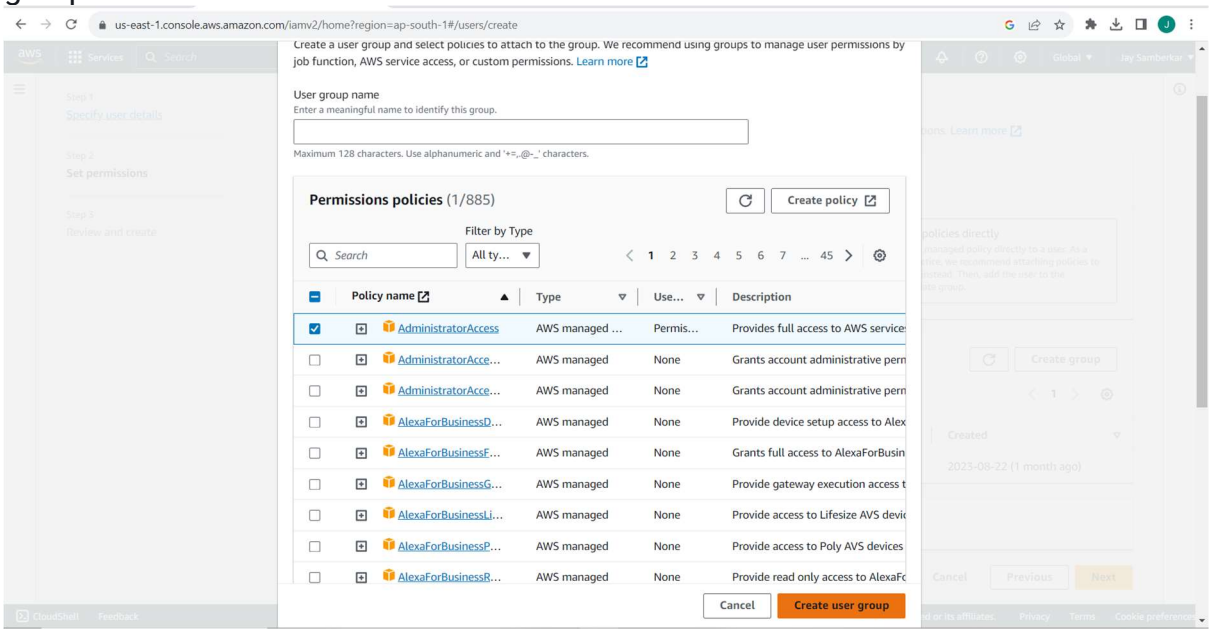
2. Head to users and Click on Create User



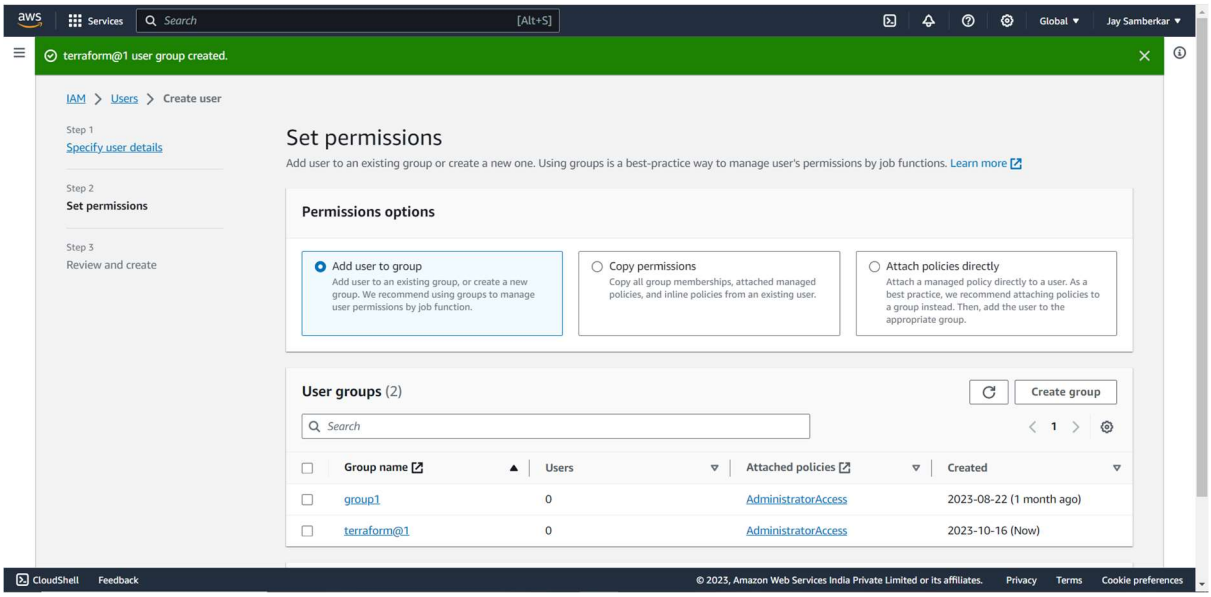
3. Enter a new username and click on next
4. Click on create group



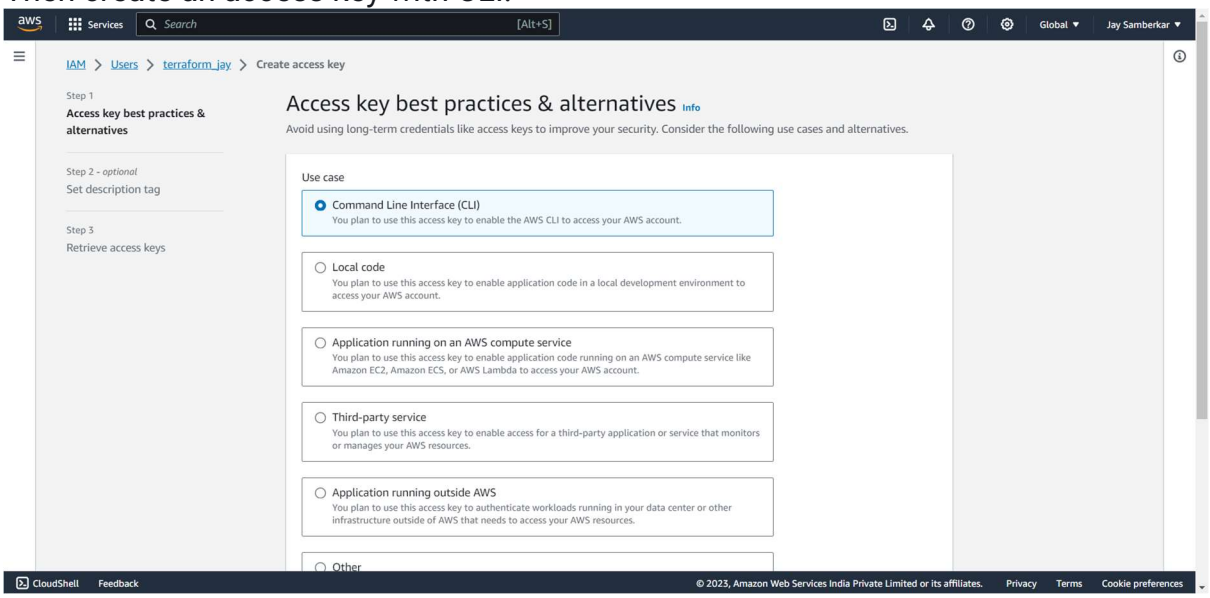
5. Select the first Policy will full access and enter a group name and create the group.



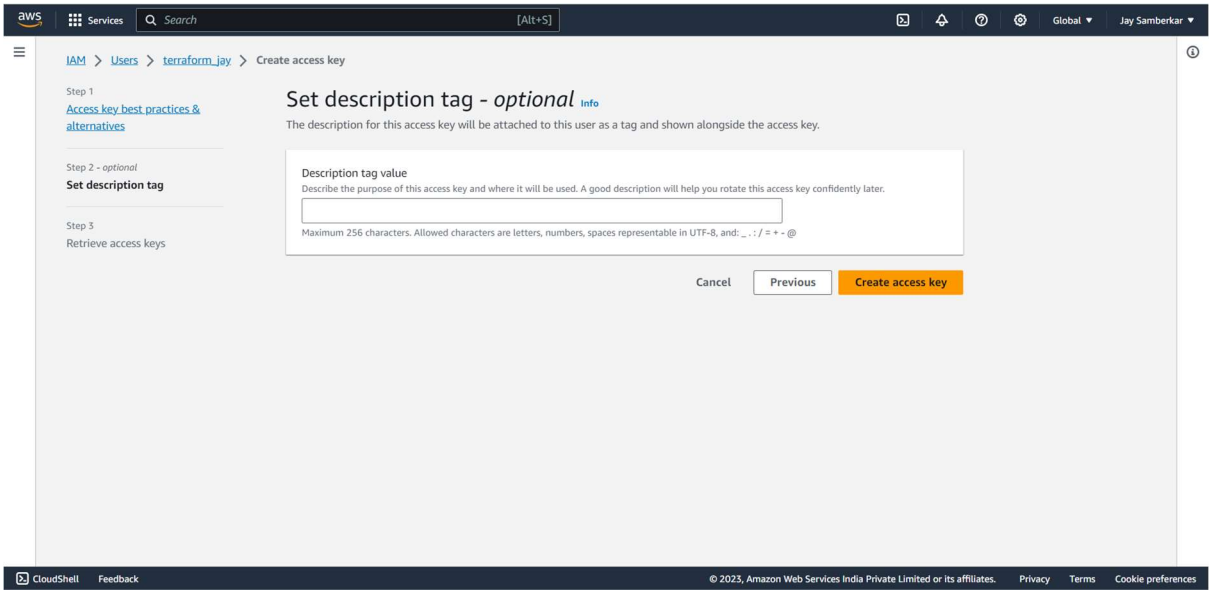
6. Then click on next, select the policy and create user.



7. Then create an access key with CLI.



8. Leave the description field empty and create a key. Download the access key in the .csv file.



9. Install terraform.

```
Last login: Sat Aug 19 17:44:56 on ttys000
❯ brew tap hashicorp/tap
Running `brew update --auto-update`...
Installing from the API is now the default behaviour!
You can save space and time by running:
  brew untap homebrew/core
  brew untap homebrew/cask
==> Auto-updated Homebrew!
Updated 3 taps (mongodb/brew, homebrew/core and homebrew/cask).
==> New Formulae
arm-none-eabi-binutils                  medusa
arm-none-eabi-gcc                       mjml
arm-none-eabi-gdb                       mongodb/brew/mongodb-community@6.0
asnmap                                  mongodb/brew/mongodb-enterprise@6.0
cargo-auditable                         mongodb/brew/mongodb-mongocryptd@6.0
cdi                                     mysql-client@8.0
cloudlist                               mysql@8.0
coder                                   ollama
ctpv                                    proxify
czkawka                                 python-certifi
dnsrobocert                             riff
dolphie                                 riscv64-elf-binutils
ebook2cw                                riscv64-elf-gcc
go@1.20                                 riscv64-elf-gdb
img2pdf                                 rpmspectool
```

10. Create the terraform config file main.tf with required configurations.

```
❯ nano main.tf
❯ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 4.16"...
- Installing hashicorp/aws v4.67.0...
- Installed hashicorp/aws v4.67.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
~/terraform_scripts                                                  ✔ 25s
```

11. Run the command terraform plan and enter yes

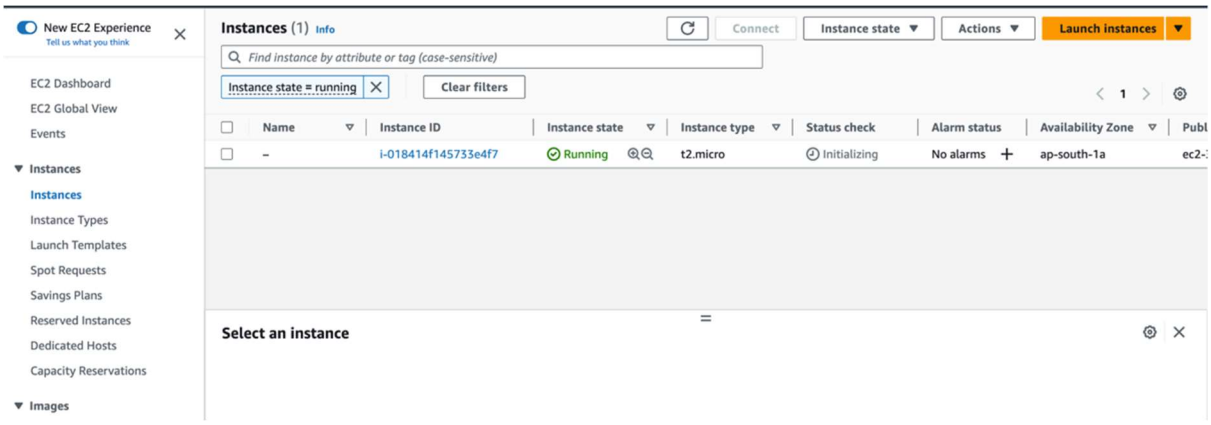
```
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
❯ terraform plan

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.Ubuntu will be created
  + resource "aws_instance" "Ubuntu" {
      + ami                                  = "ami-0f5ee92e2d63afc18"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
```

12. Check EC2 instances on AWS, you'll find an instance running started by terraform.



13. Now run terraform destroy to terminate the instance.

```
          - enable_resource_name_dns_a_record    = false -> null
          - enable_resource_name_dns_aaaa_record = false -> null
          - hostname_type                        = "ip-name" -> null
        }

      - root_block_device {
          - delete_on_termination = true -> null
          - device_name           = "/dev/sda1" -> null
          - encrypted             = false -> null
          - iops                  = 100 -> null
          - tags                  = {} -> null
          - throughput            = 0 -> null
          - volume_id             = "vol-07a64df7d00e4e7a1" -> null
          - volume_size           = 8 -> null
          - volume_type           = "gp2" -> null
        }
    }

Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

Lab Outcome:
LO1, LO6 mapped.

Conclusion :
We understood the workings of Terraform to create and manage EC2 instances in AWS which provides a highly automated and reproducible infrastructure-as-code solution. The ability to effortlessly spin up and tear down instances not only enhances operational agility but also ensures optimal resource utilization.

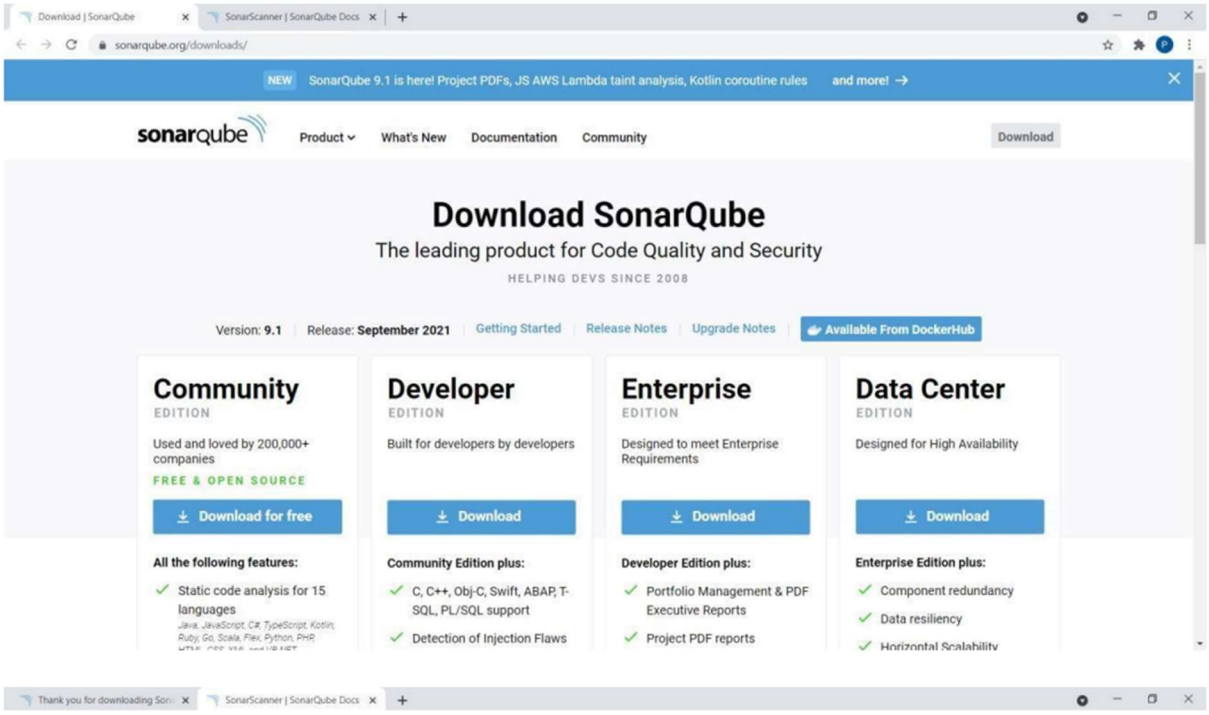
Assignment 7

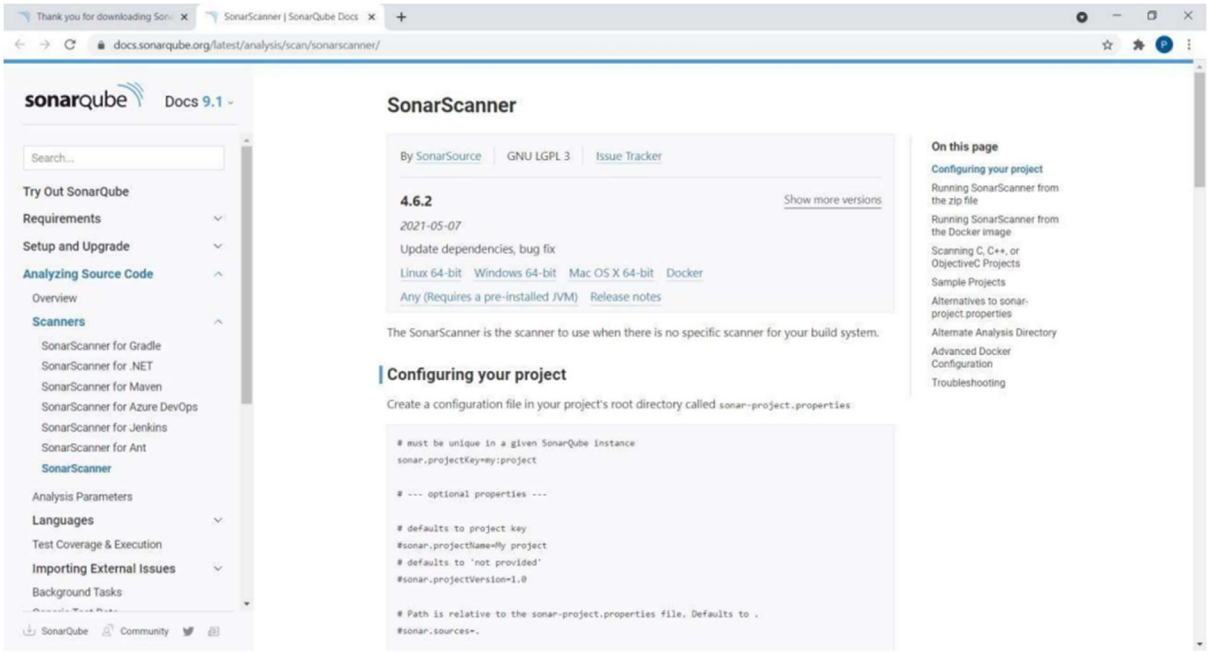
Aim: TO perform static analysis on python programs using SonarCube SAST processes

LO Mapped: L04

Theory:

Download and Sonar Scanner

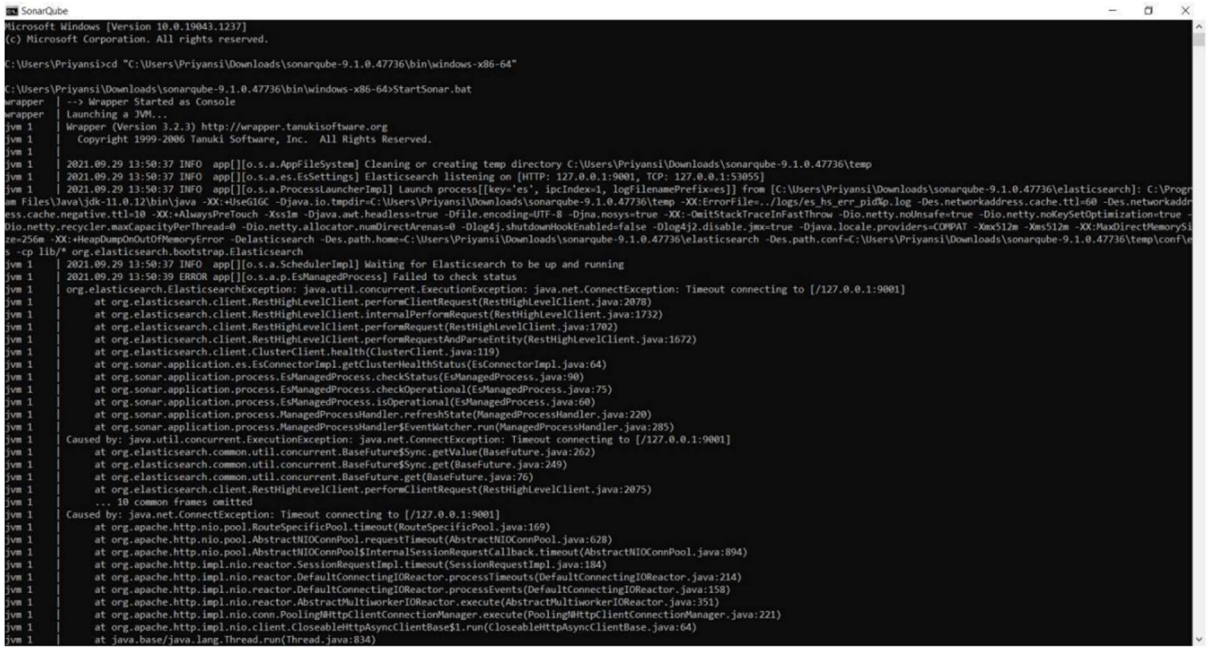


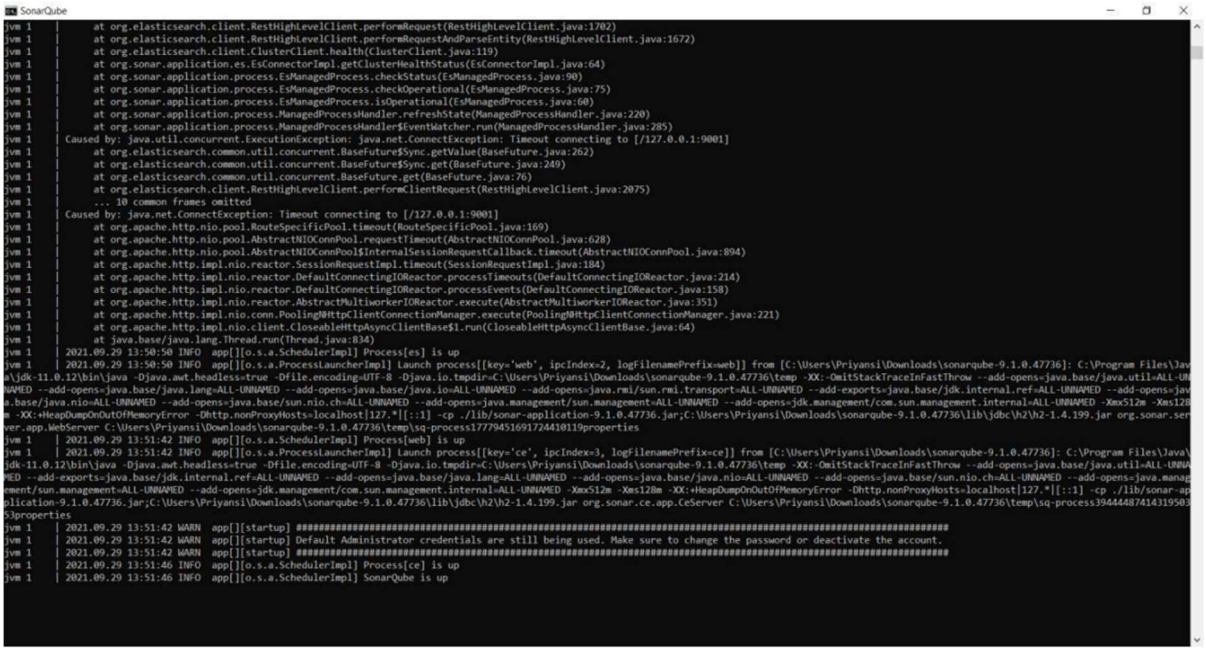


After downloading, set Environment Variables. Add "sonarqube-9.1.0.47736\bin" to Path.

Open command prompt. Run commands:

- cd "sonarqube-9.1.0.47736\bin\windows-x86-64"
    - StartSonar.bat

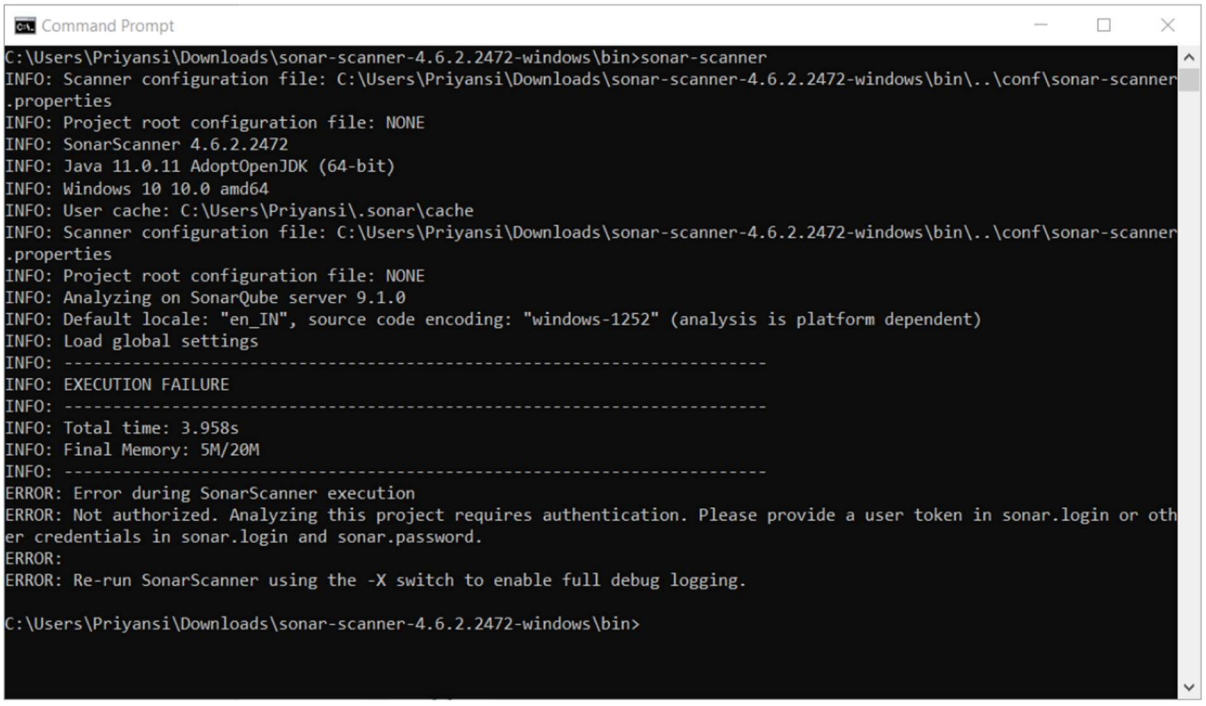




Open another command prompt. Run command:

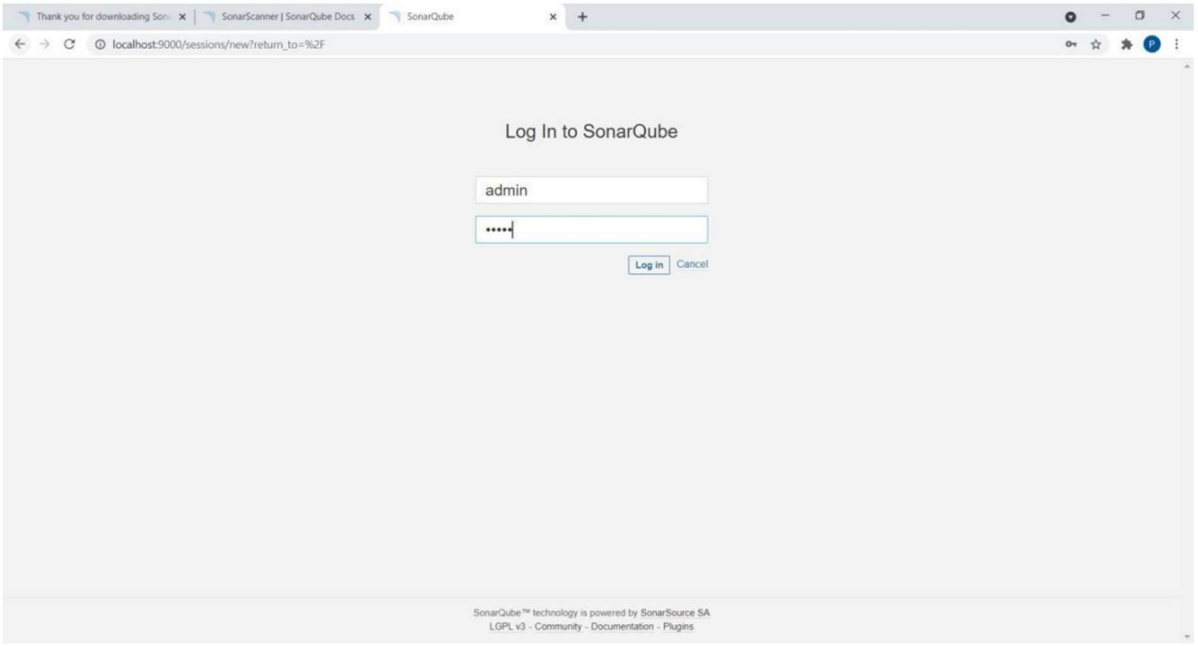
e cd "sonar-scanner-4.6.2.2472-windows\bin"

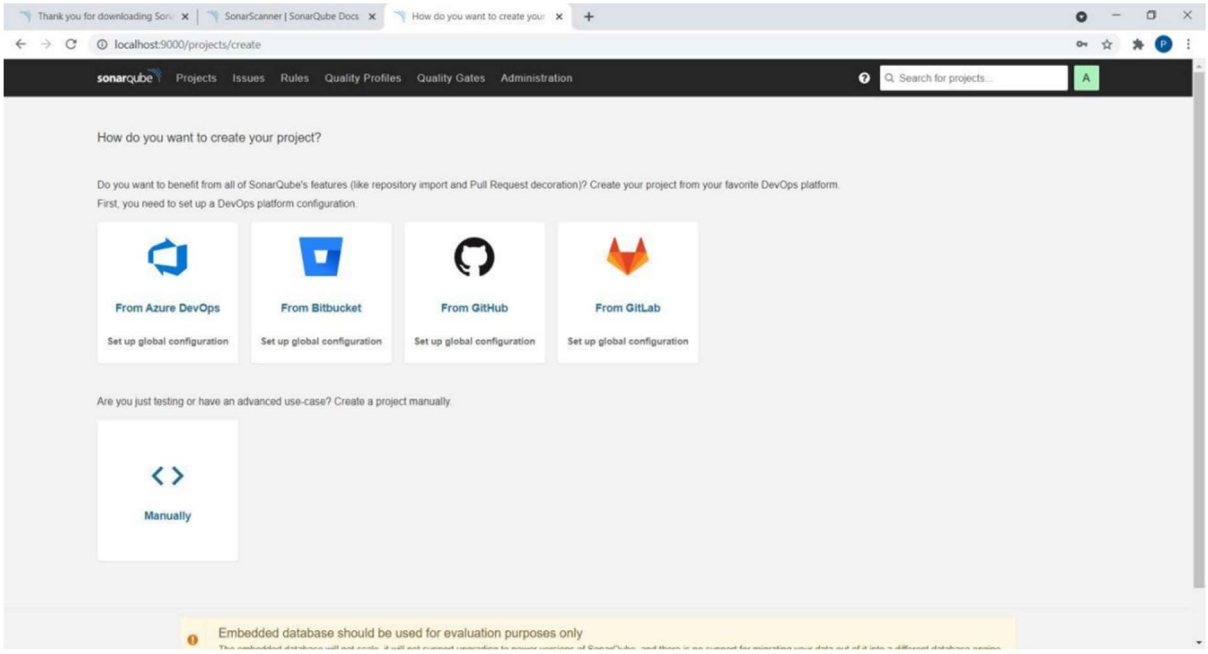
sonar-scanner



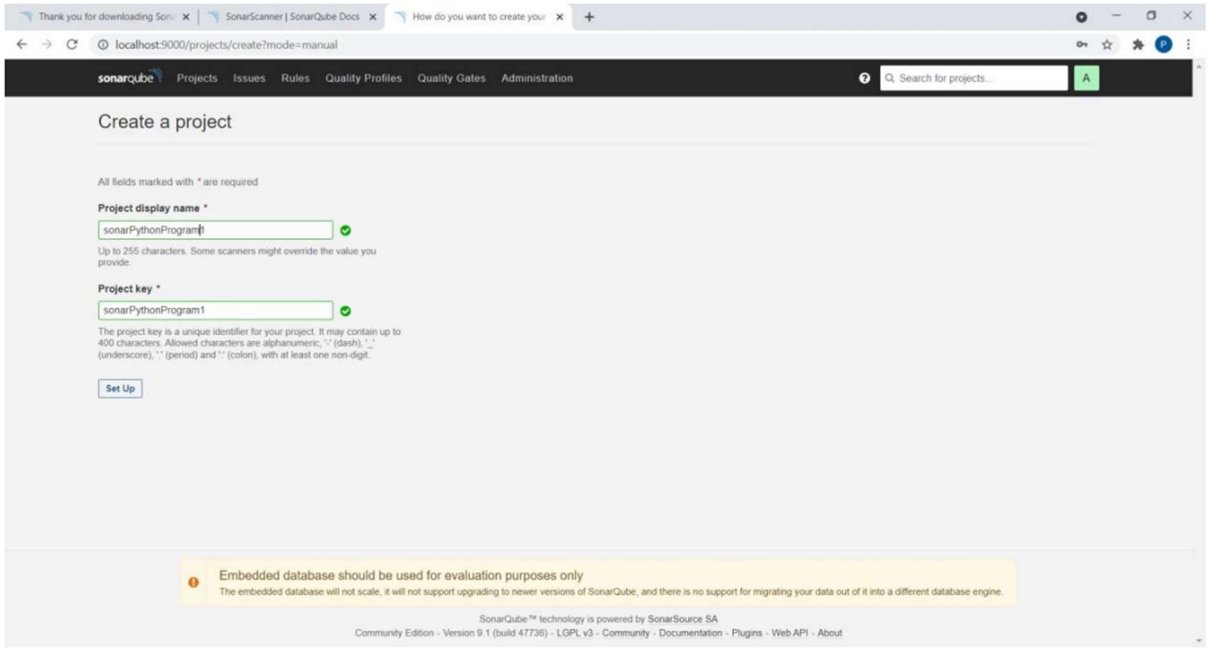
Server up and running on localhost:9000

Login using credentials as User: admin and Password: admin and Set a new password

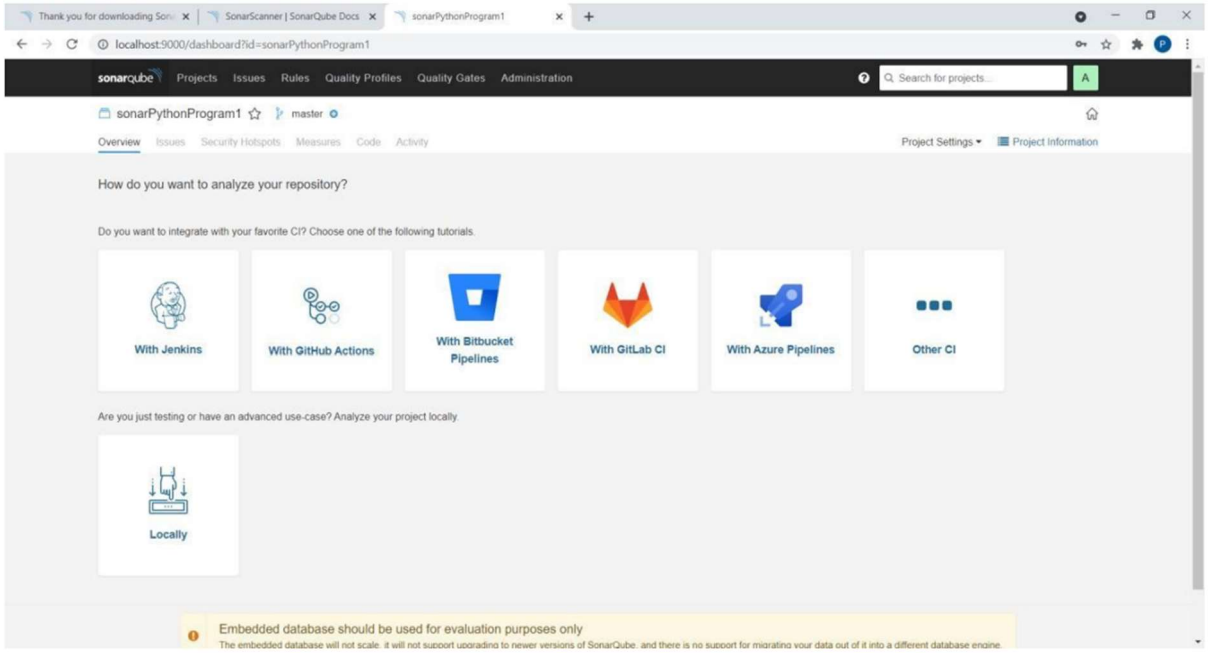




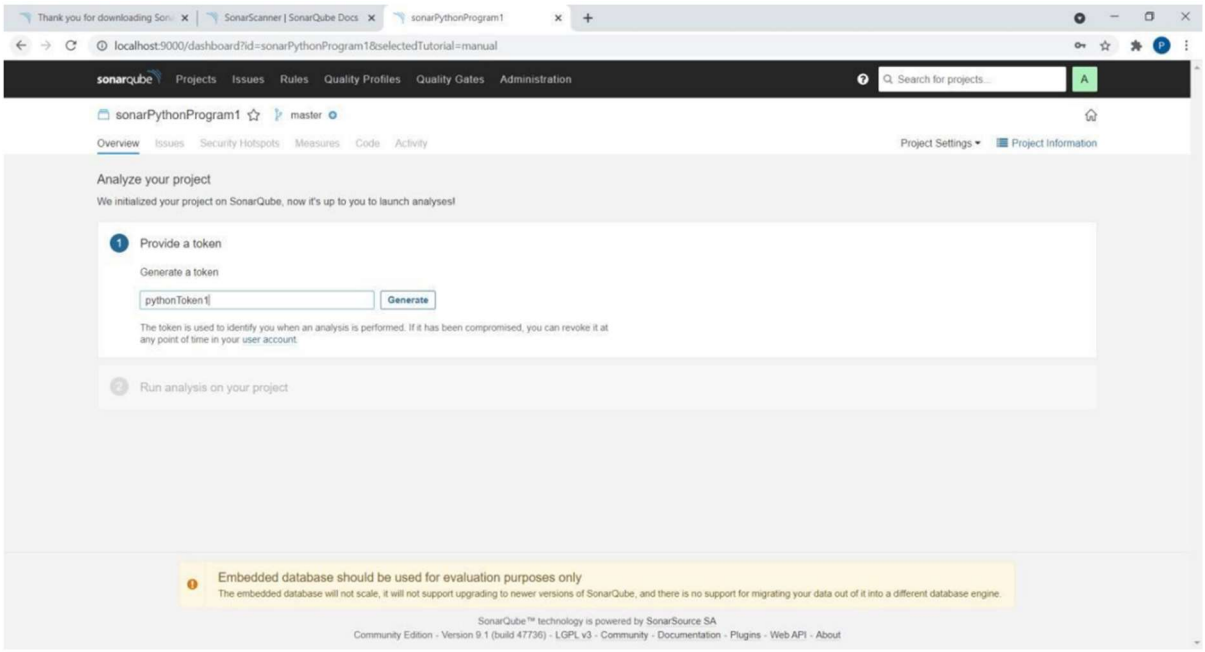
Click on Create a project Manually.



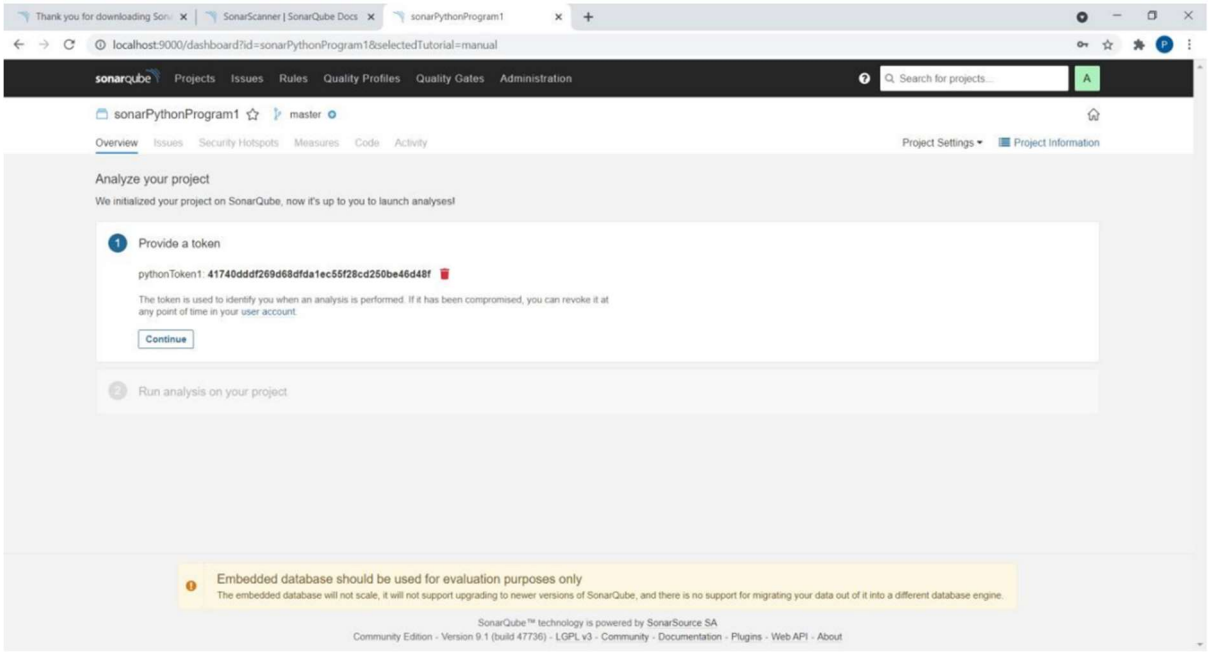
Give any Project display name.



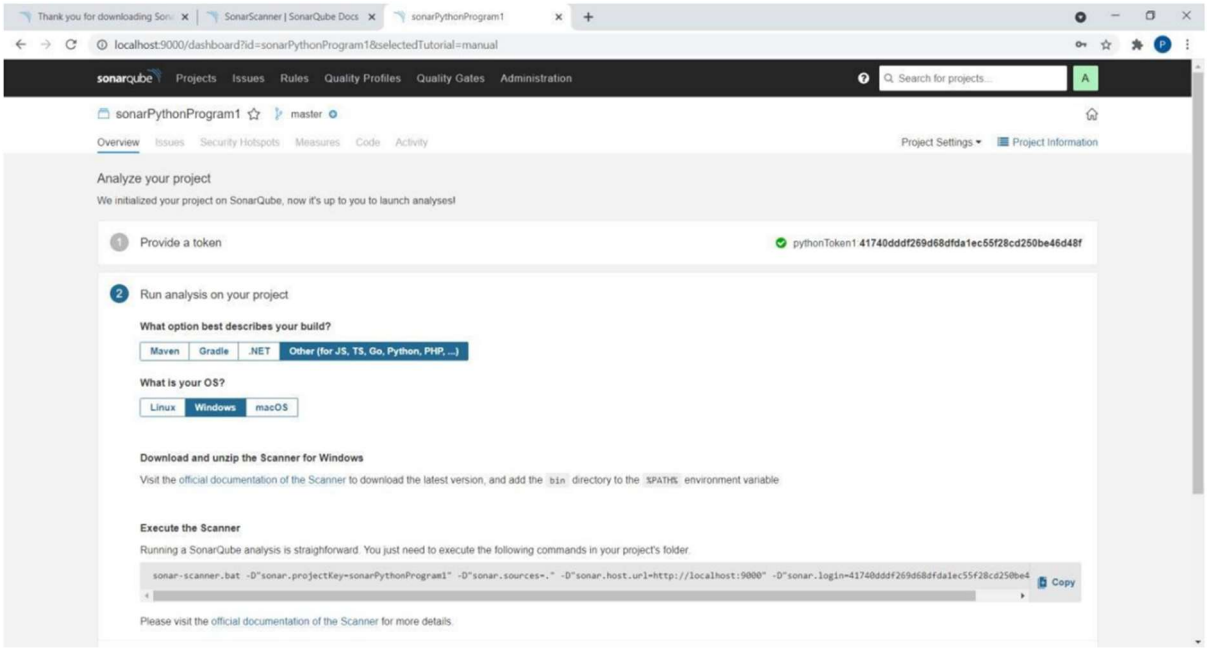
Click on Locally.



Give any name to token and click on Generate.



Click on Continue.



Save a Python program in a folder.

```
class Solution(object):
def romanToInt(self, s):
roman =
num =
while i < len(s):
if i+l<len(s) and s[i:i+2] in roman:
num+=roman[s[i:i+2]]
else:
#print(i)
num+=roman[s[i]]
return num
obl = Solution()
print(obl l"))
```

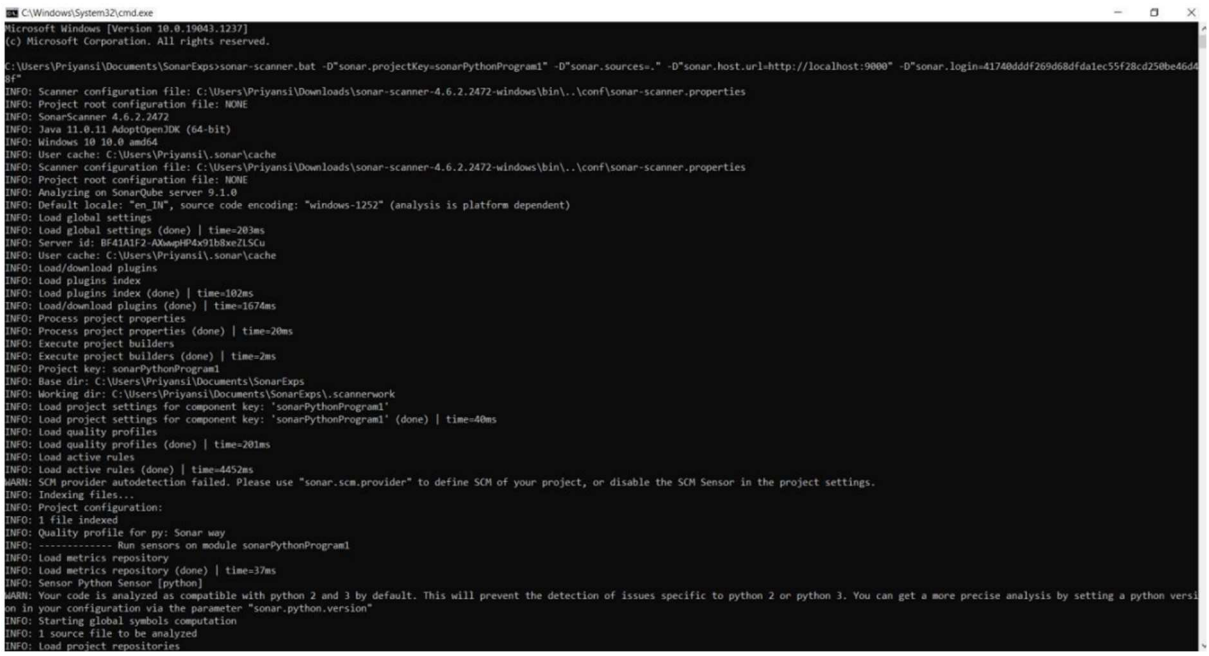
print(obl . l")

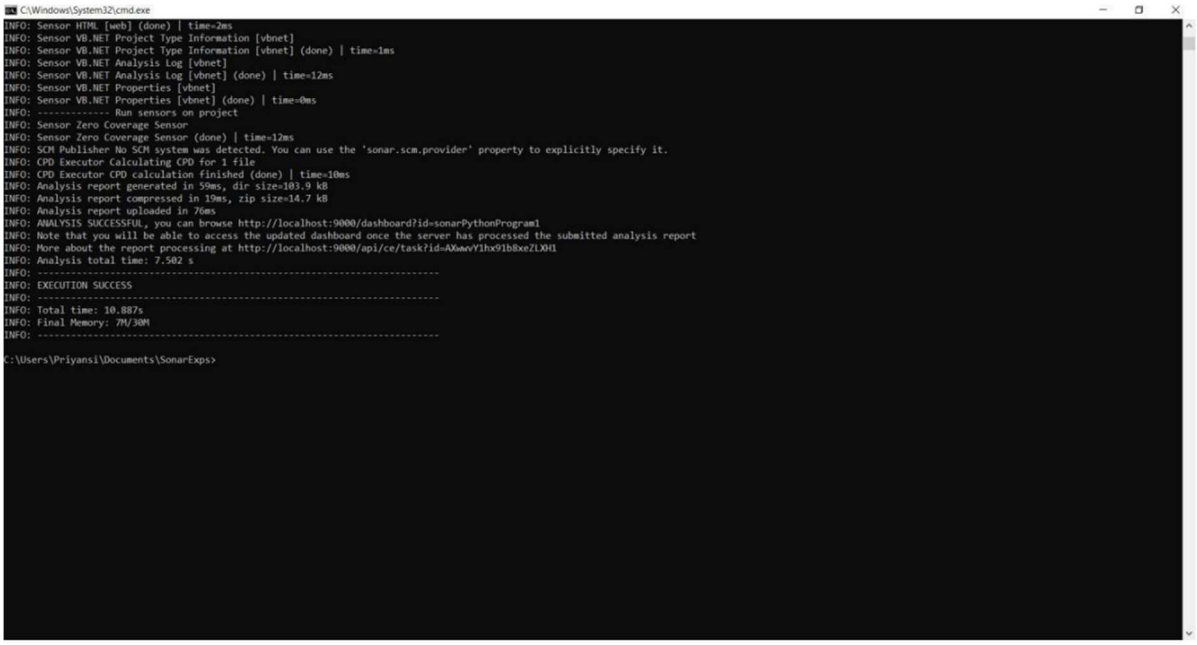
Open command prompt in this folder and Run program using copied command.

sonar-scanner.bat -D"sonar.projectKey=<YourDisplayName>" -

D"sonar.sources=." -D"sonar.host.url=http://localhost:9000" -

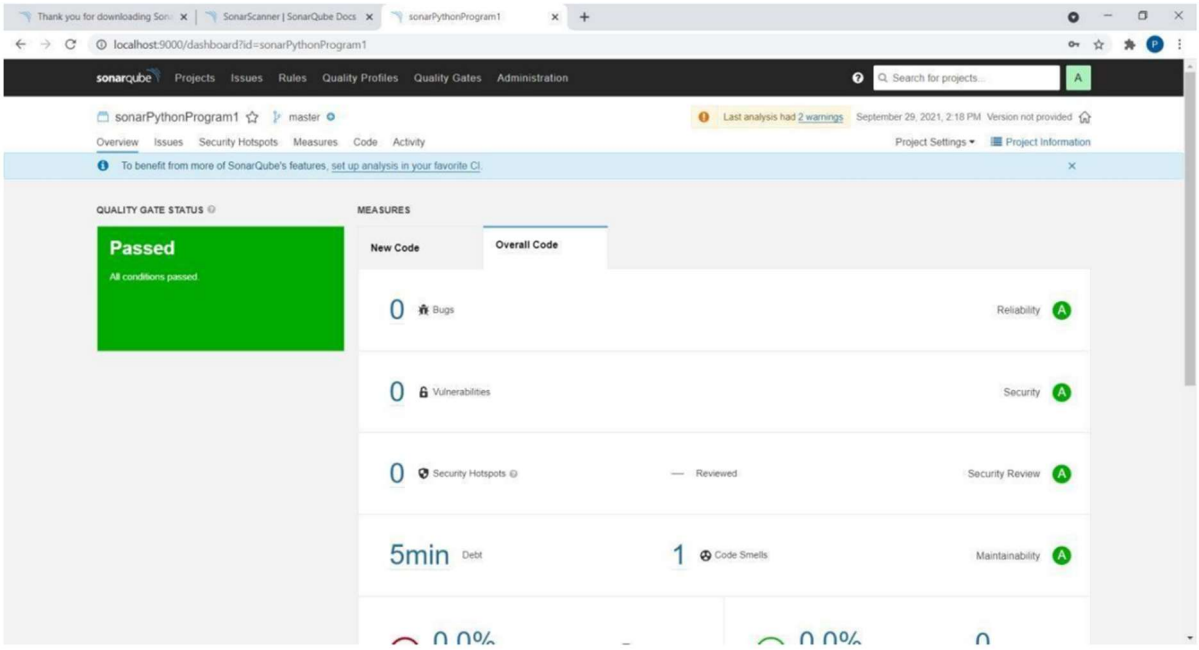
D"sonar.login=<YourTokenGeneratedID>'

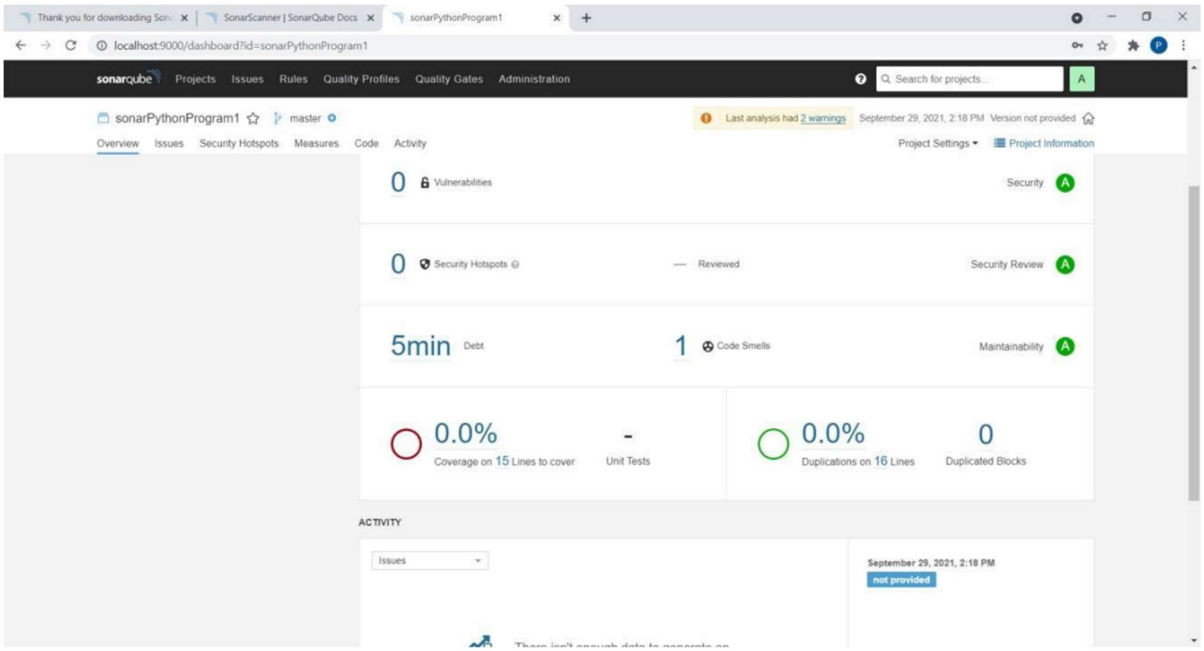


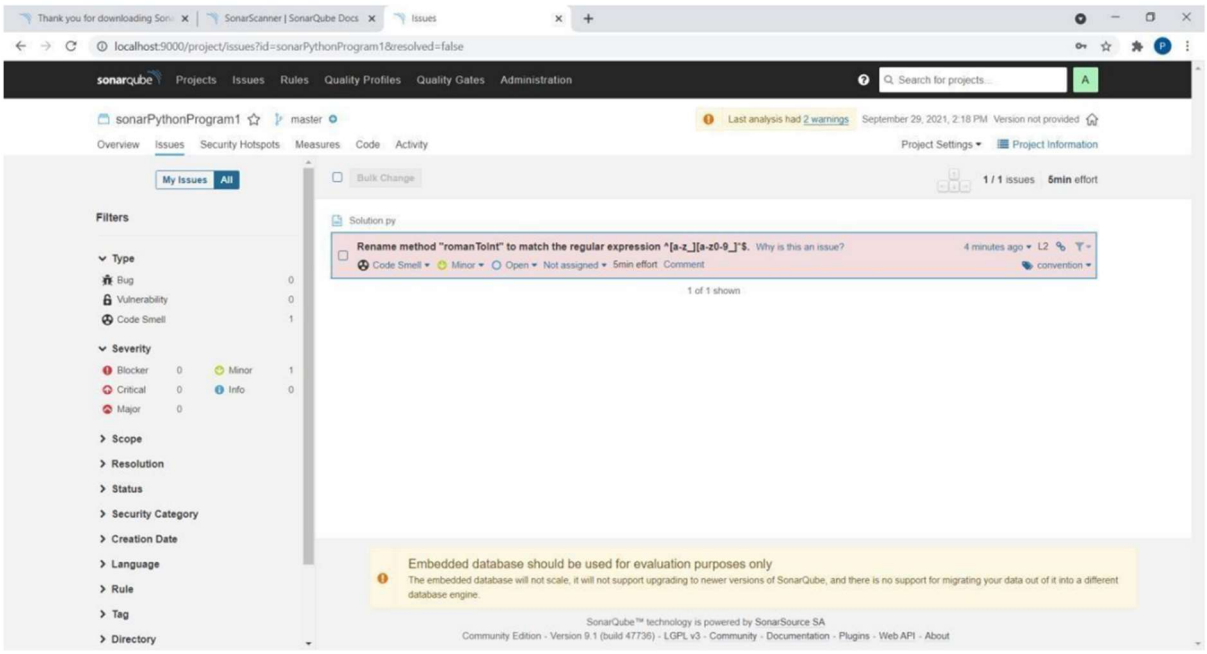


Given below is the inspection of code quality to perform automatic reviews with static analysis of code

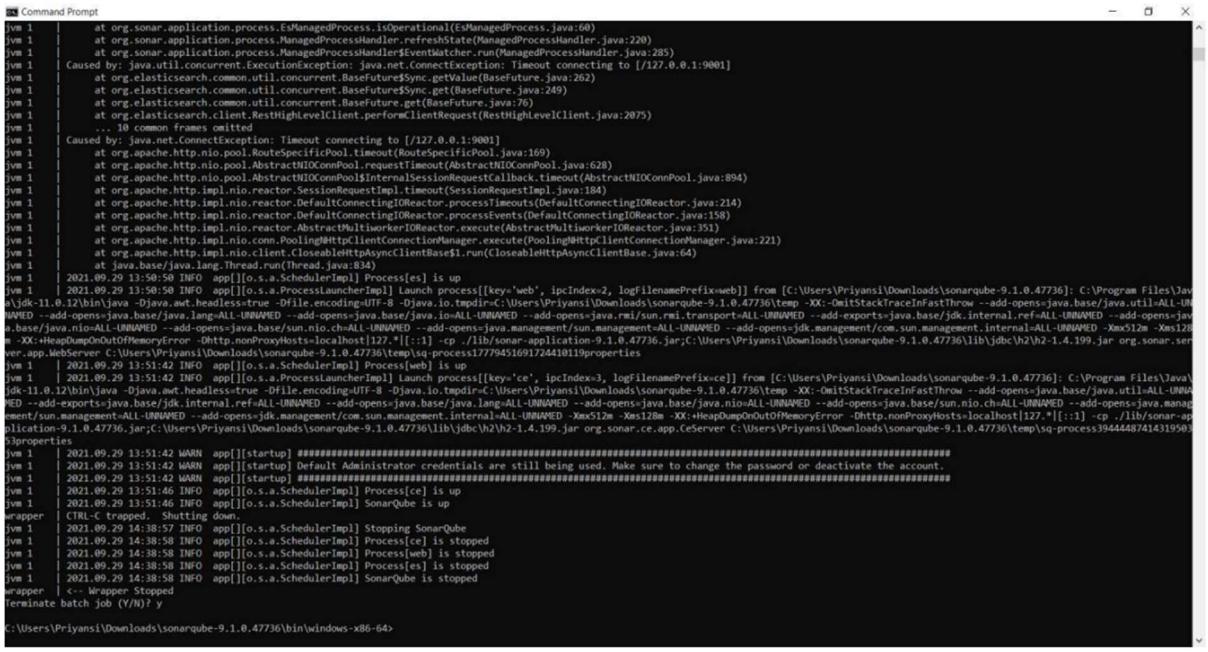
to detect bugs, code smells, and security vulnerabilities.







Press "Ctrl + C" to stop the server.



Conclusion:

In this assignment we implemented static analysis on python programs using SonarCube

SAST processes.

Assignment 8

Name Soham Satam Roll No.: 109 Branch: IT/T2 Date:17/10/2023

Aim: To understand Continuous Monitoring using Nagios.

LO Mapped: LO5

Theory:

Login to Aws.

Create Ec2 instances on Aws account of any linux os

Then Run the following command in SS

```
ubuntu@ip-172-31-13-219:~$ sudo apt update
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:4 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
Get:5 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 Packages [8628 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [909 kB]
Get:7 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal/universe Translation-en [5124 kB]
```

cd /usr/src/

sudo wget https://github.com/NagiosEnterprises/nagioscore/archive/nagios-4.4.2.tar.gz

```
ubuntu@ip-172-31-13-219:/usr/src$ sudo wget https://github.com/NagiosEnterprises/nagioscore/archive/nagios-4.4.2.tar.gz
--2021-10-05 10:24:05--  https://github.com/NagiosEnterprises/nagioscore/archive/nagios-4.4.2.tar.gz
Resolving github.com (github.com)... 13.234.176.102
Connecting to github.com (github.com)|13.234.176.102|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://codeload.github.com/NagiosEnterprises/nagioscore/tar.gz/nagios-4.4.2 [following]
--2021-10-05 10:24:05--  https://codeload.github.com/NagiosEnterprises/nagioscore/tar.gz/nagios-4.4.2
Resolving codeload.github.com (codeload.github.com)... 13.233.43.20
Connecting to codeload.github.com (codeload.github.com)|13.233.43.20|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/x-gzip]
Saving to: 'nagios-4.4.2.tar.gz'

nagios-4.4.2.tar.gz                     [    <=>                                                 ]  10.78M  16.9MB/s    in 0.6s

2021-10-05 10:24:06 (16.9 MB/s) - 'nagios-4.4.2.tar.gz' saved [11301457]
```

sudo tar zxf nagios-*.tar.gz

cd nagioscore-nagios-*/

```
ubuntu@ip-172-31-13-219:/usr/src$ cd nagioscore-nagios-*/
ubuntu@ip-172-31-13-219:/usr/src/nagioscore-nagios-4.4.2$
```

Now finally run the following command

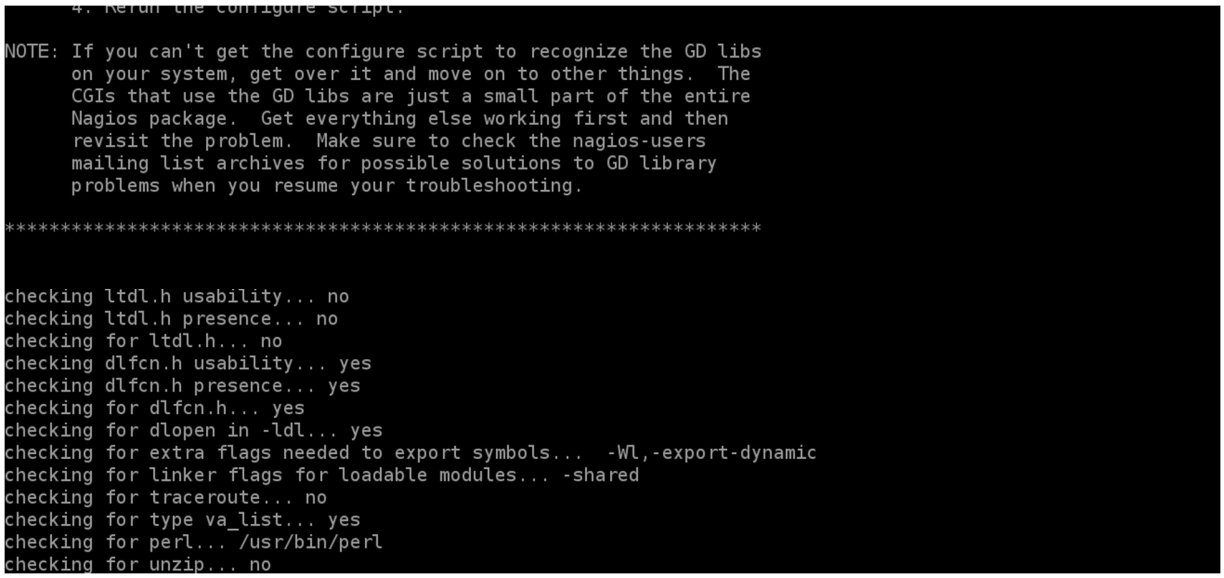
sudo ./configure --with-httpd-conf=/etc/apache2/sites-enabled

if error comes install c compiler on the linux

by following this link

https://linuxize.com/post/how-to-install-gcc-compiler-on-ubuntu-18-04/

Finally-



Now let us install plugins by

sudo wget -O nagios-plugins.tar.gz https://github.com/nagios-plugins/nagios-plugins/archive/release-2.2.1.tar.g

then

sudo tar zxf nagios-plugins.tar.gz

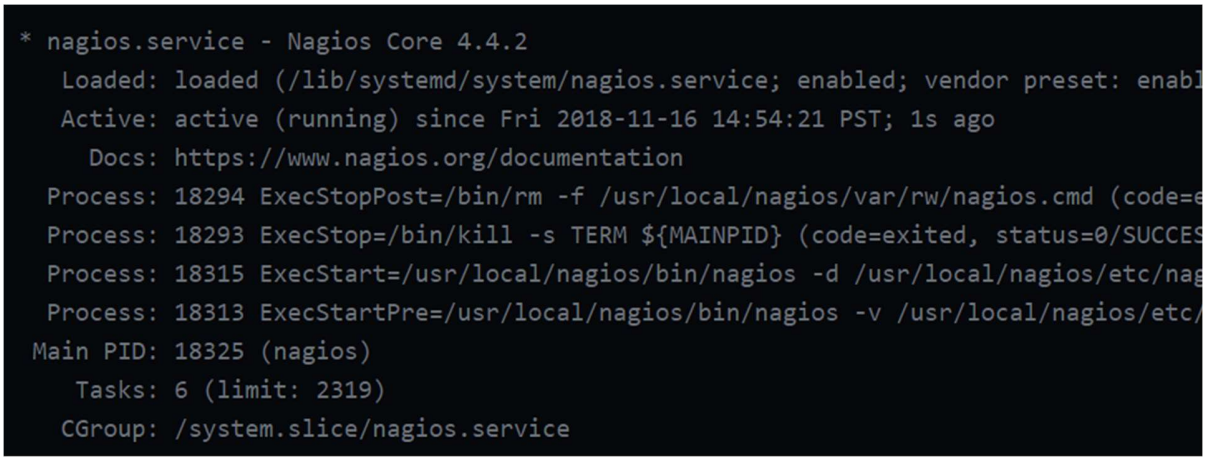
then

cd nagios-plugins-release-2.2.1

finally start and then check status of nagi os

sudo systemctl start nagios

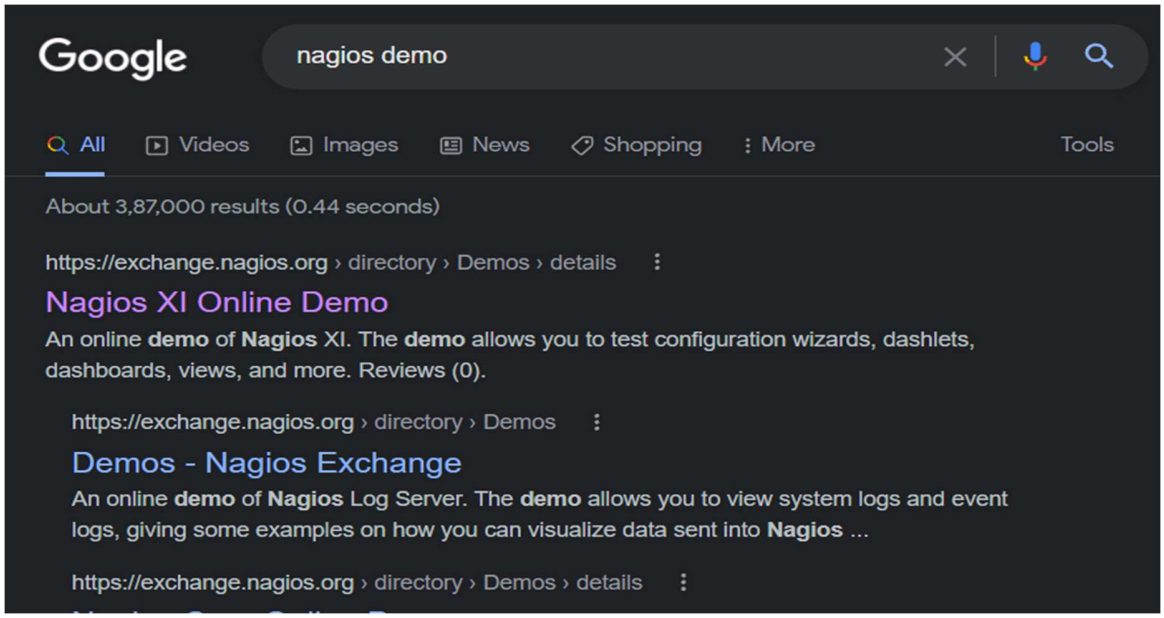
sudo systemctl status nagios



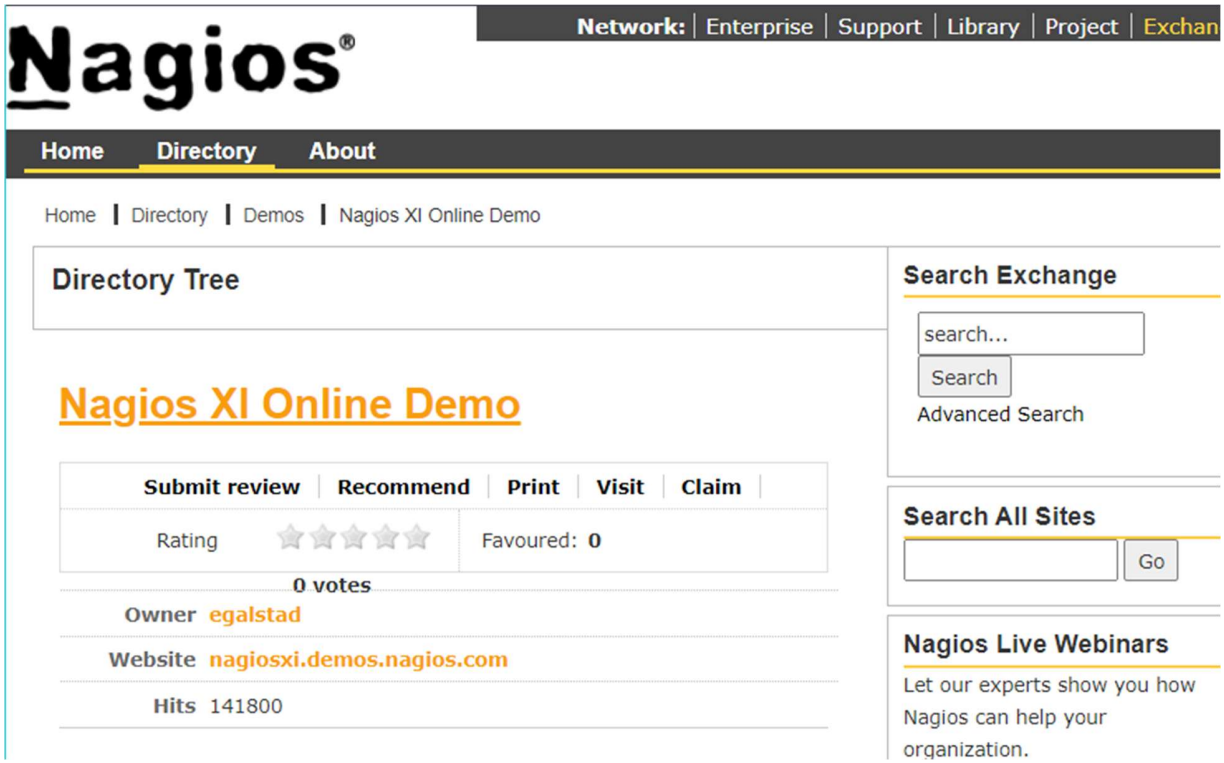
Steps-

Go to google.com, Search NagiOs Demo

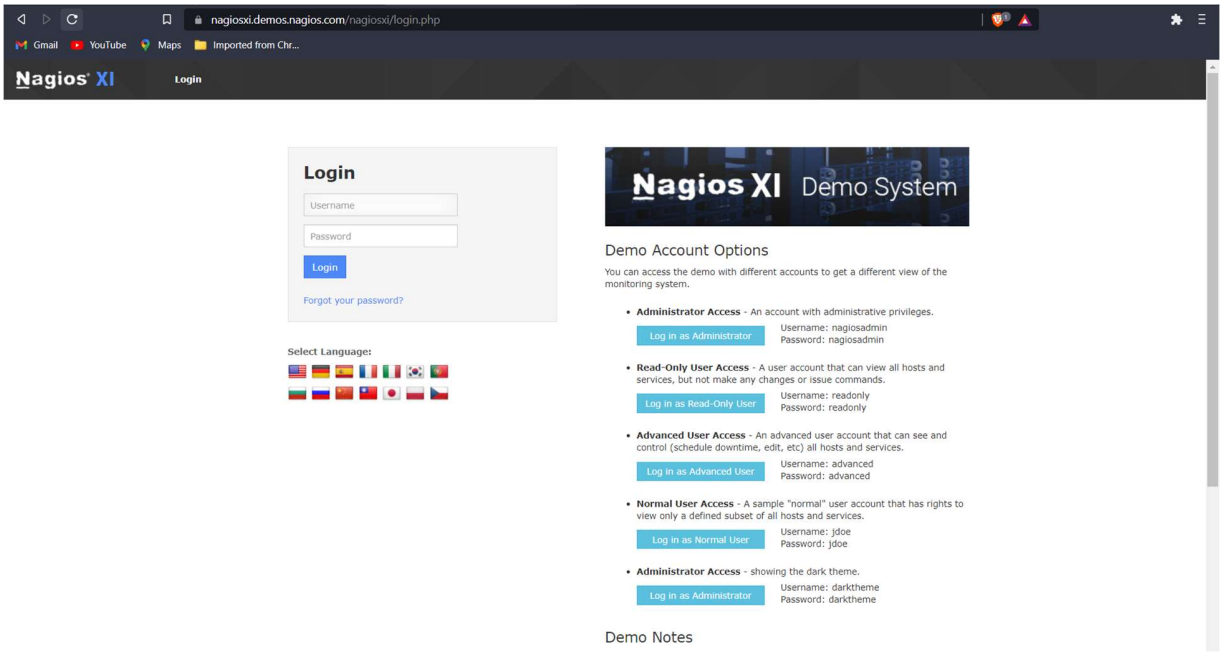
Click on the first link shown below



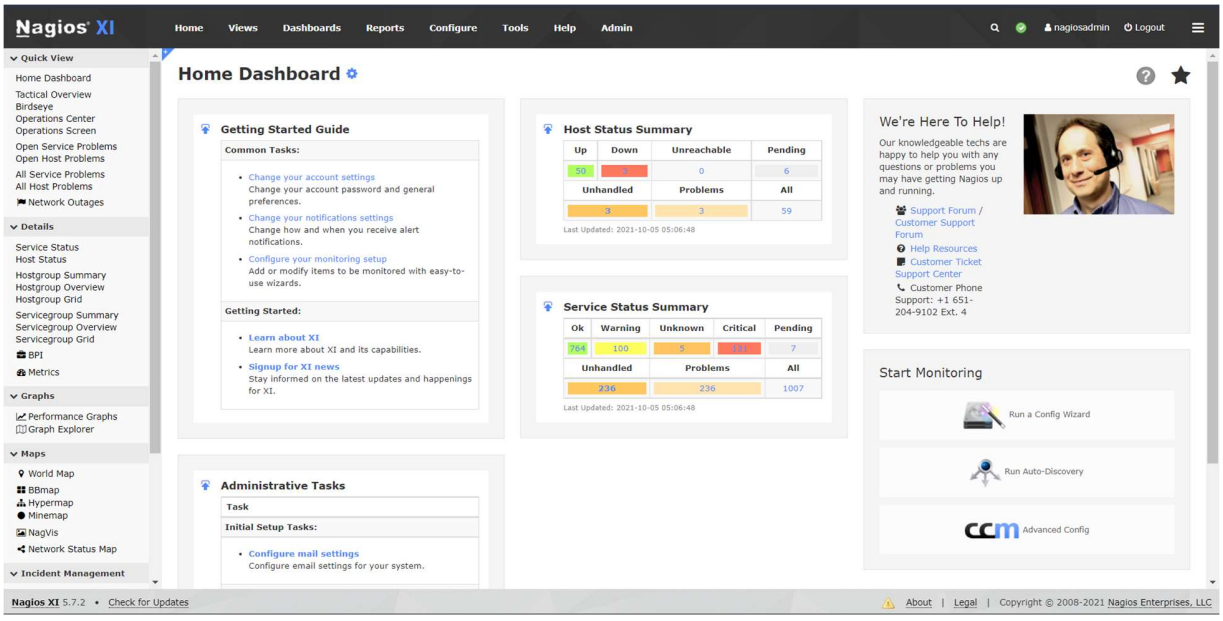
Now click on the website-



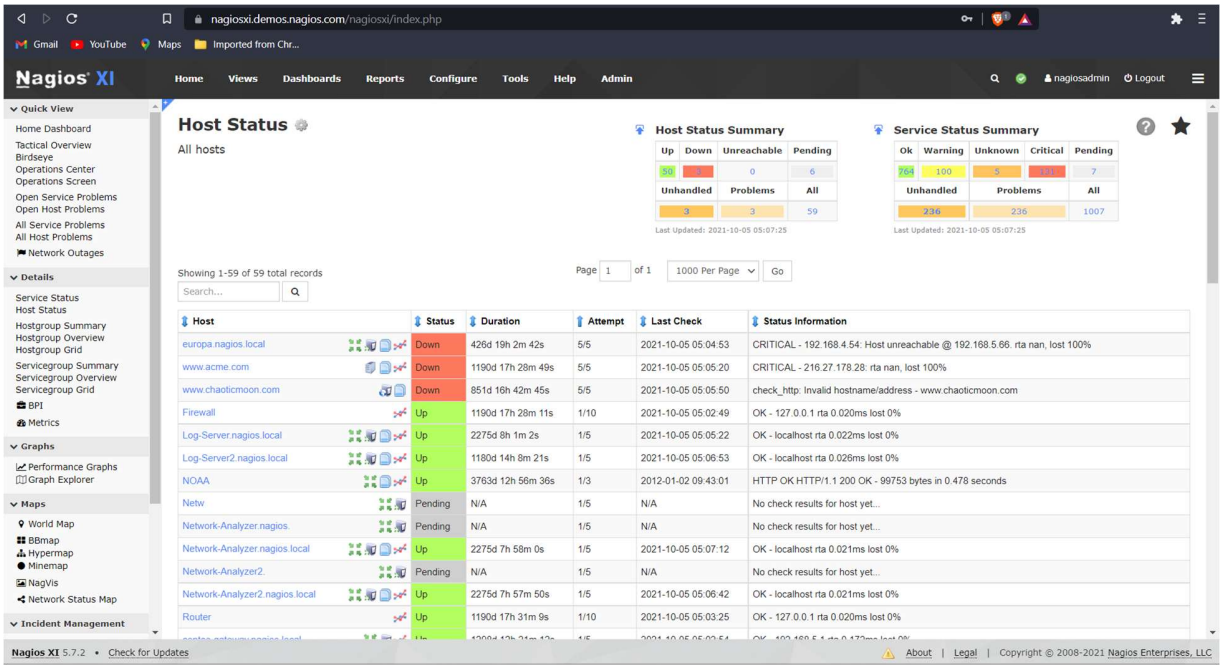
Now click on login as administrator



It will have interface like this

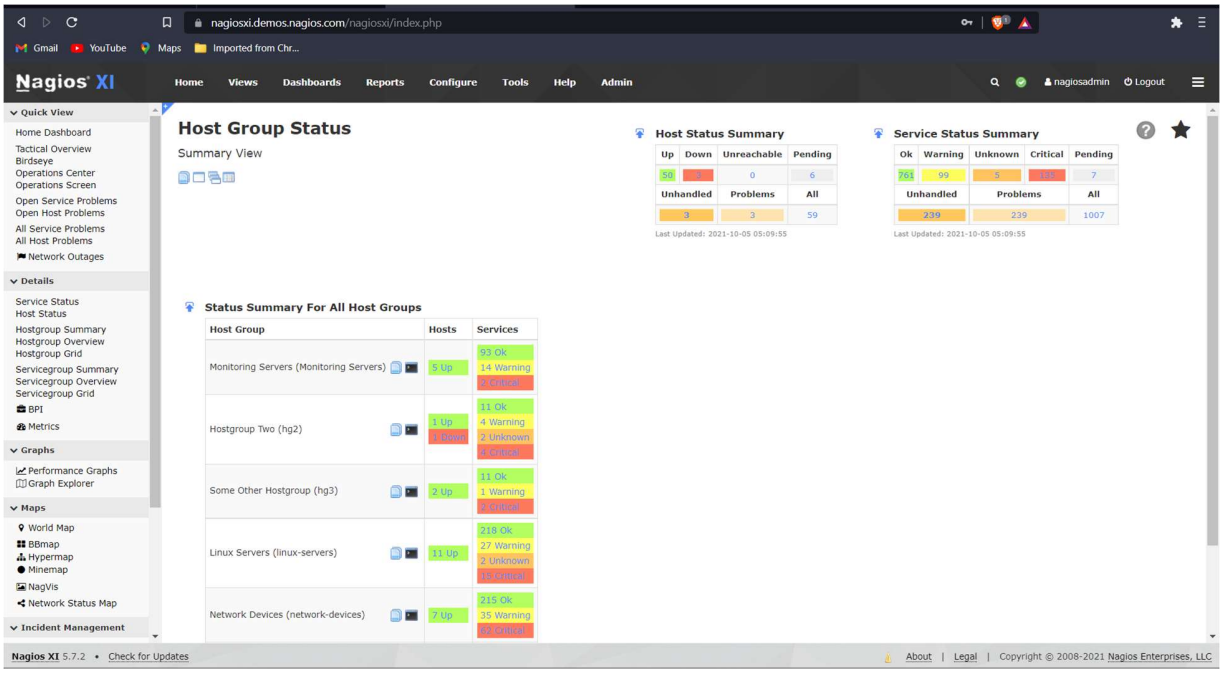


Now click on Host status-



In the above image one can see Host Status Summary and Service Status Summary also how many host are up, down and also errors in detail

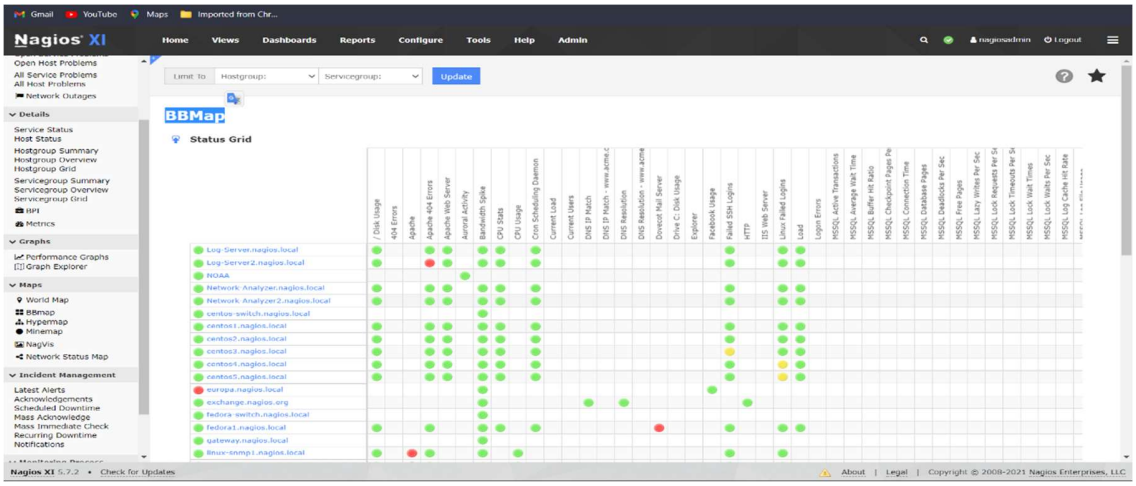
Now click on Host Group Status.



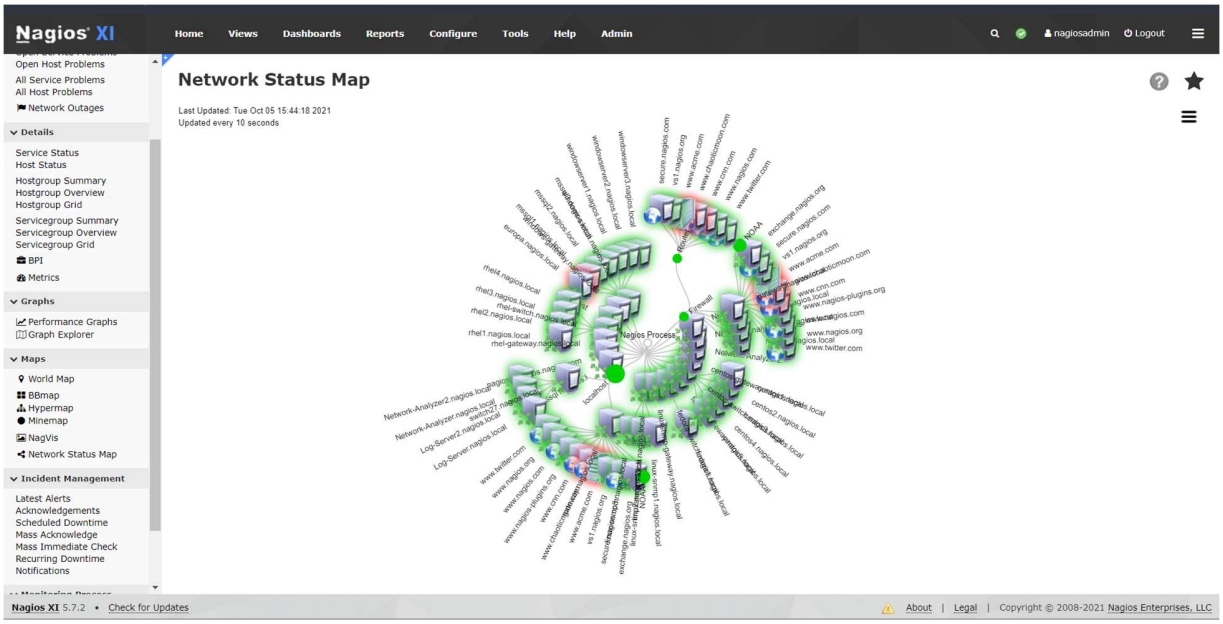
Here we can see Status Summary For All Host Groups

Now we click on BBMap

In this we can see status of following stuff in each host-



Now we have Network status map which is graphical representation of the network status



Conclusion: In this assignment we learned about how to continuously monitor Nagios commands

ASSIGNMENT 9

AIM: To understand Lambda Function and create a Lambda function which will log "An Image has been added" once you add an object to a specific bucket in S3.

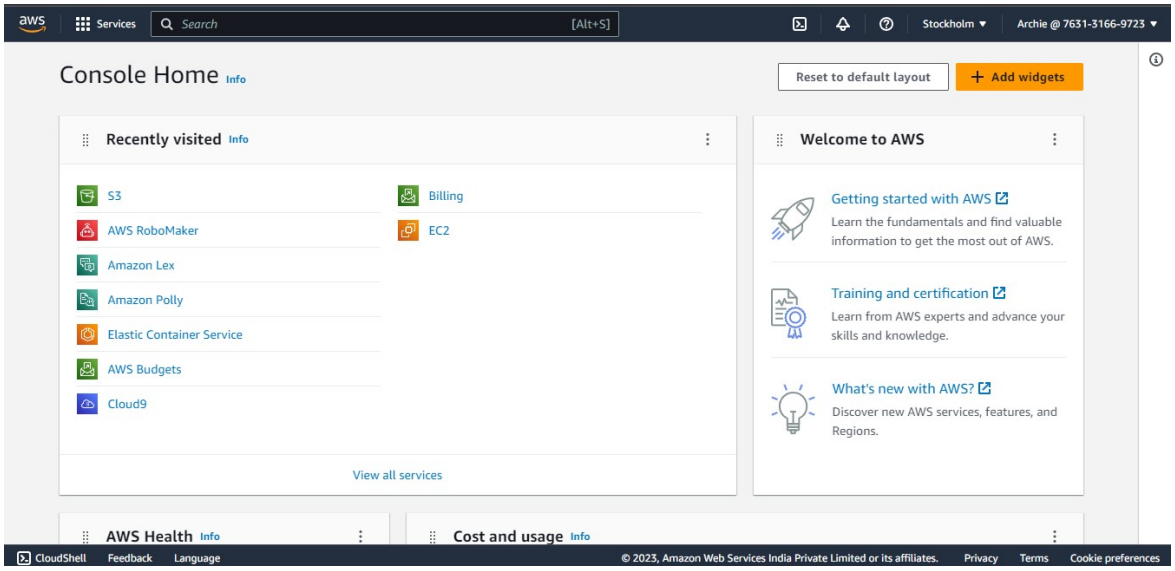
LO MAPPED: LO1, LO5

THEORY:

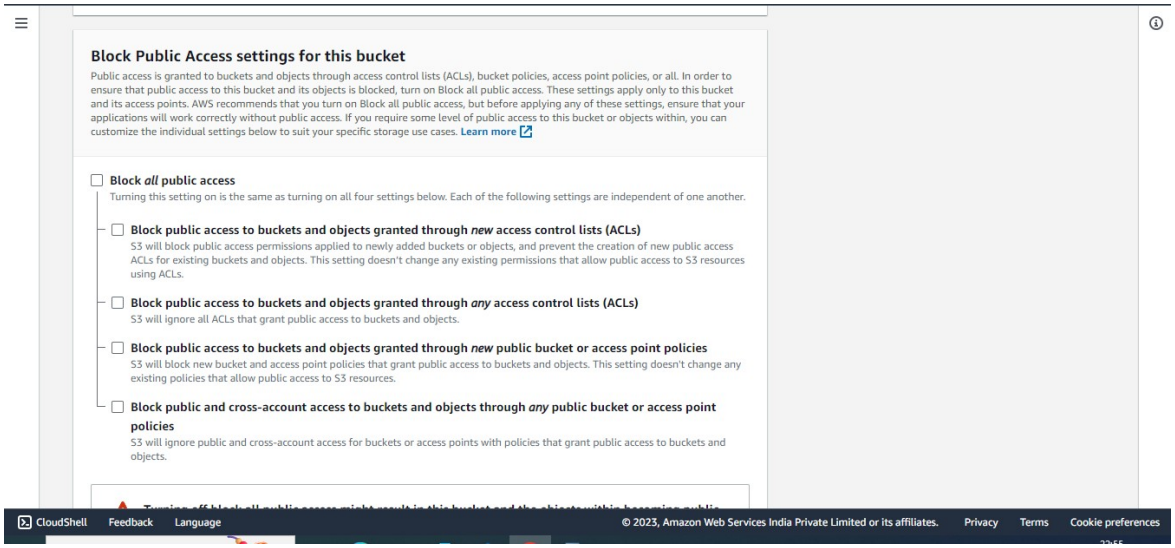
You can use Lambda to process event notifications from Amazon Simple Storage Service. Amazon S3 can send an event to a Lambda function when an object is created or deleted. You configure notification settings on a bucket, and grant Amazon S3 permission to invoke a function on the function's resource-based permissions policy.

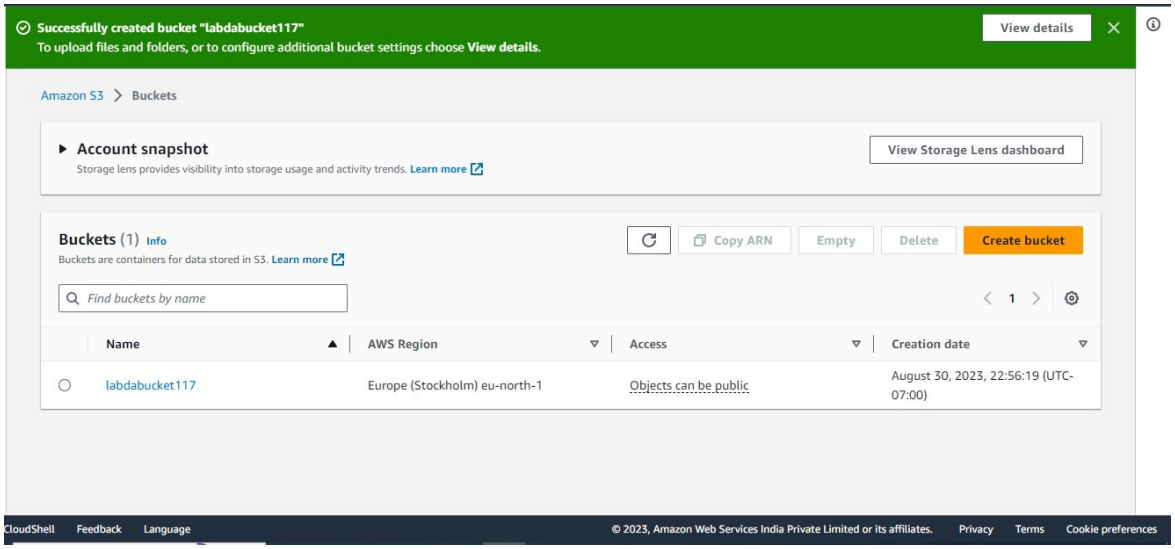
STEPS TO FOLLOW:

1. Log in as IAM User

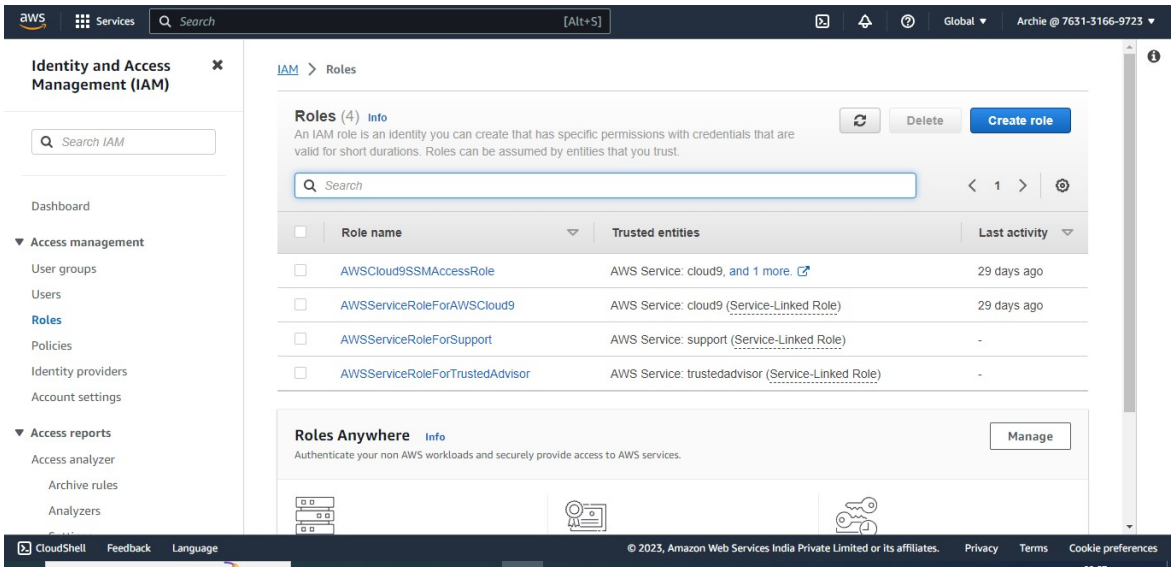


2. Create a S3 bucket and Enable the "Block all public access"

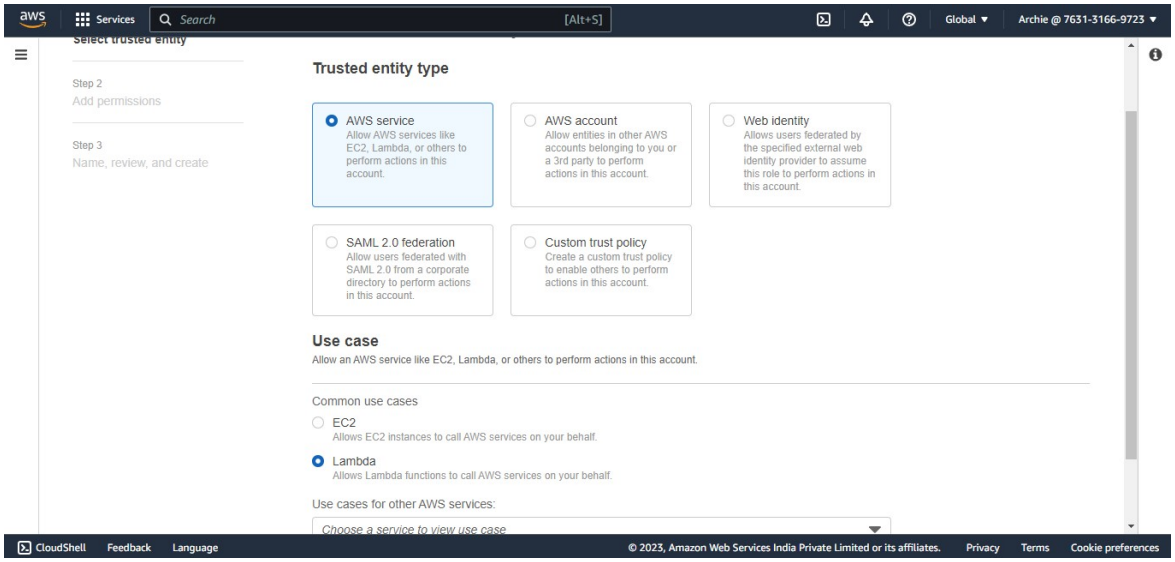




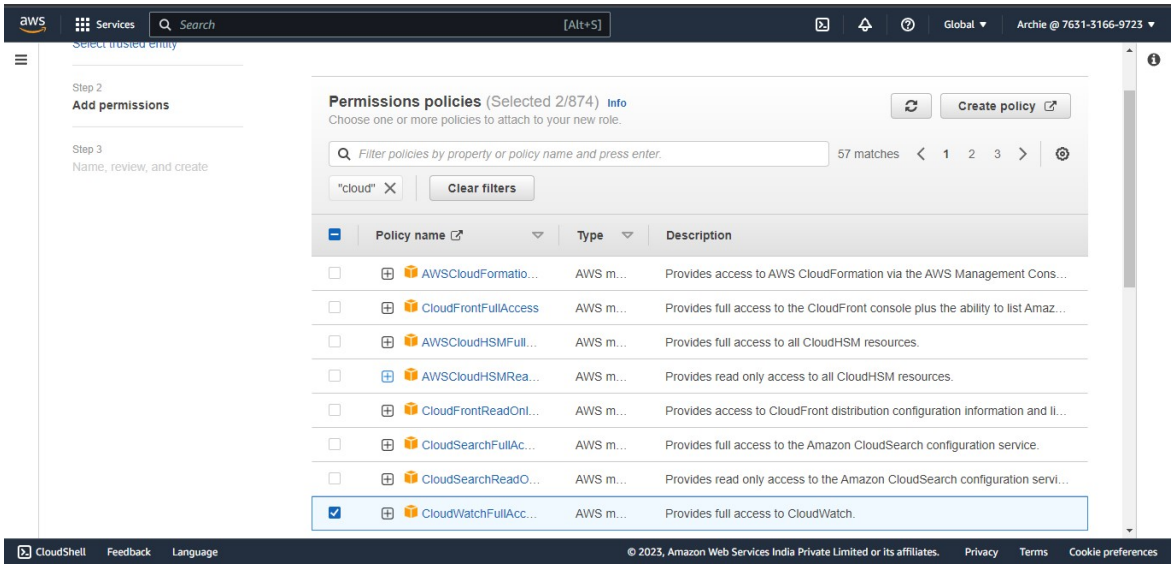
3. Search IAM on the console and go to "Roles". Click on Create Roles

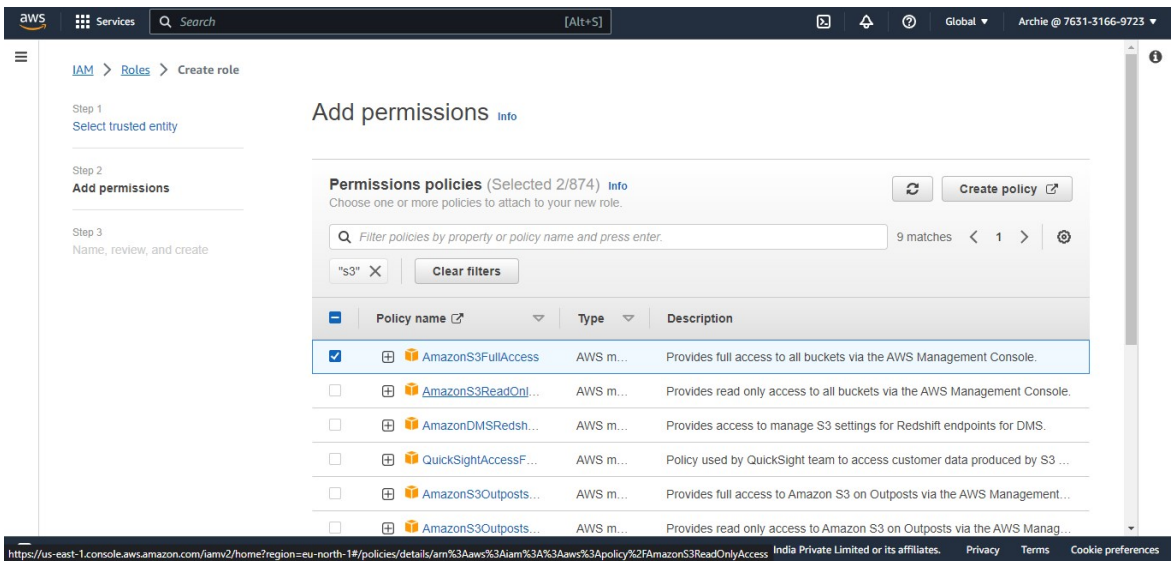


4. Select the options of "AWS service" and "lambda"

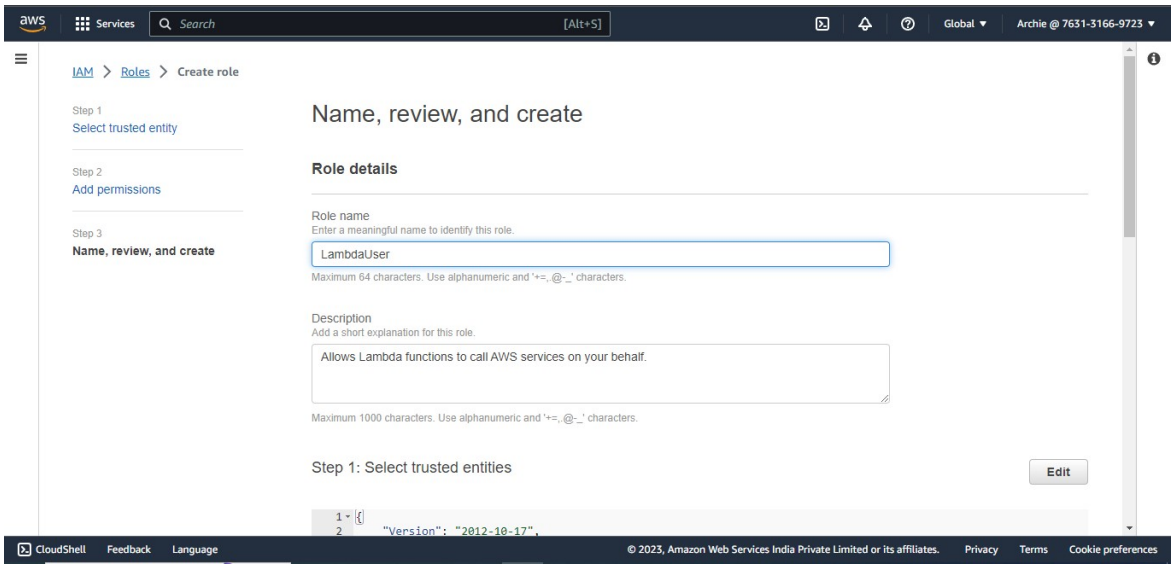


5. Enable the "CloudWatchFullAccess" and "AmazonS3FullAccess"

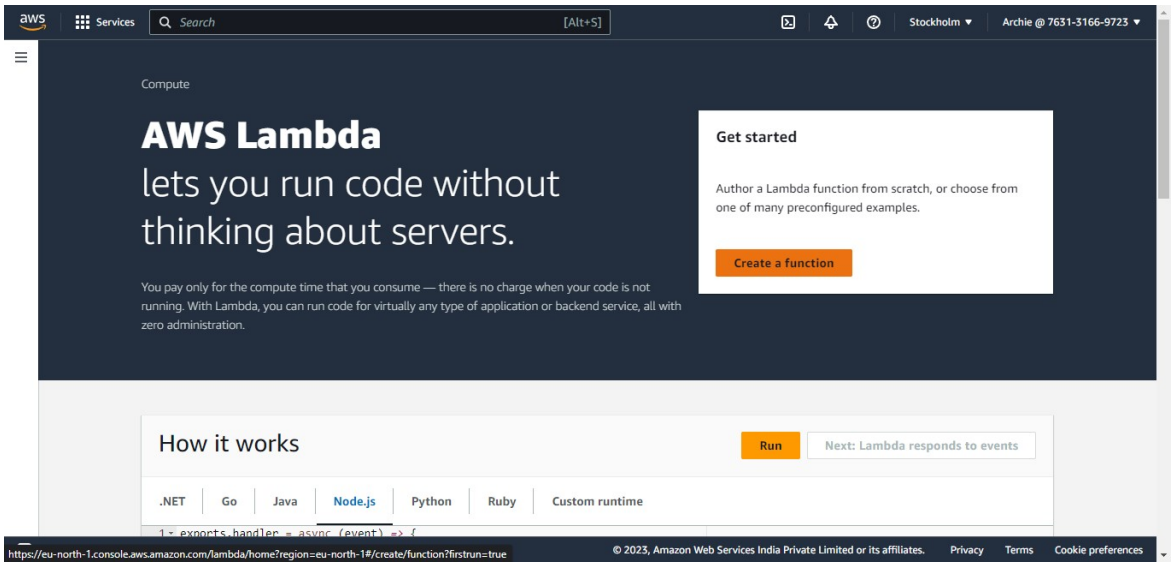




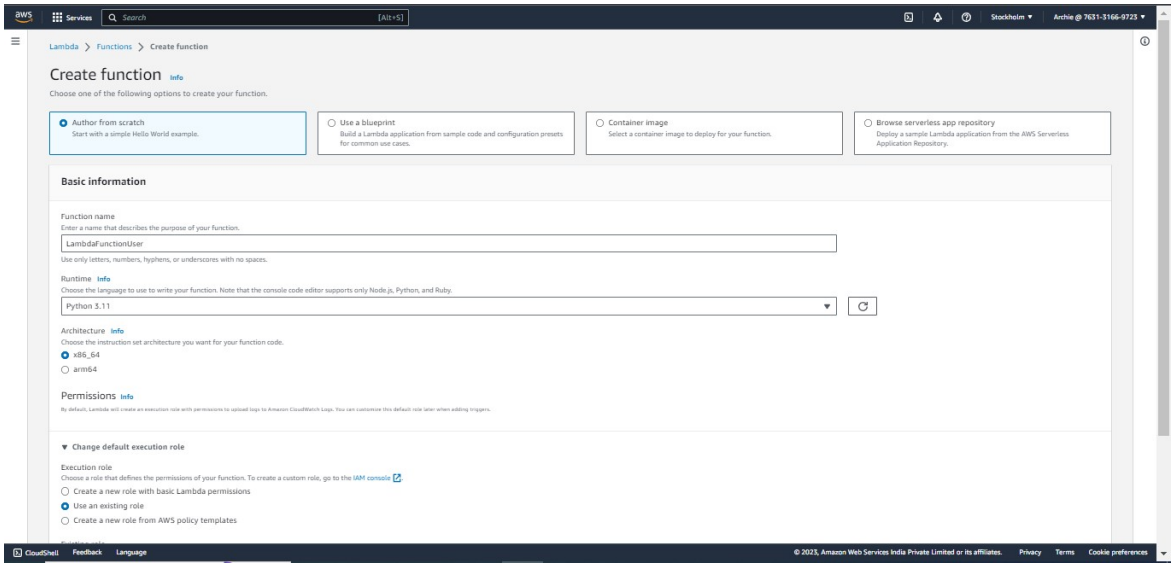
6. Click on Create Role and you will be redirected to this dashboard. Give the Role a Name and Click on Done.



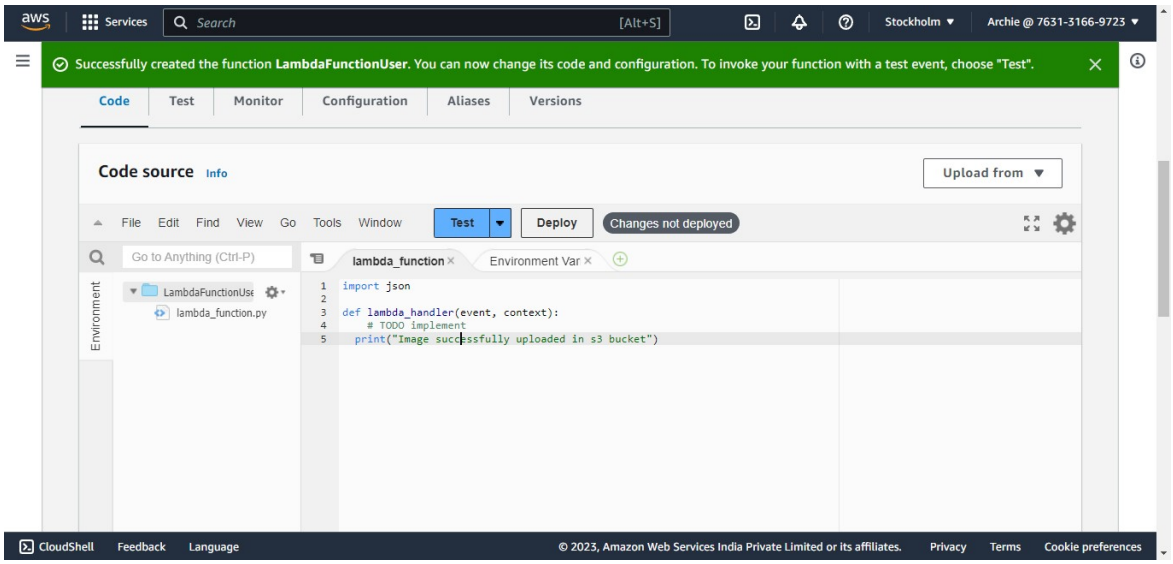
7. Search for Lambda in the console and click on Create Function.



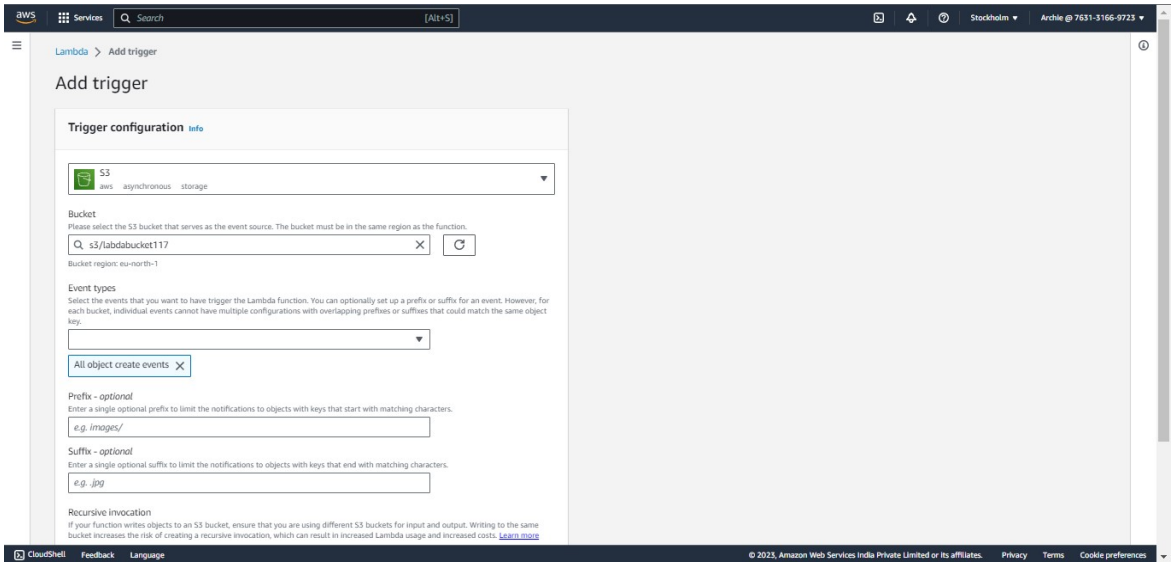
8. Change the settings of the Lambda Function.



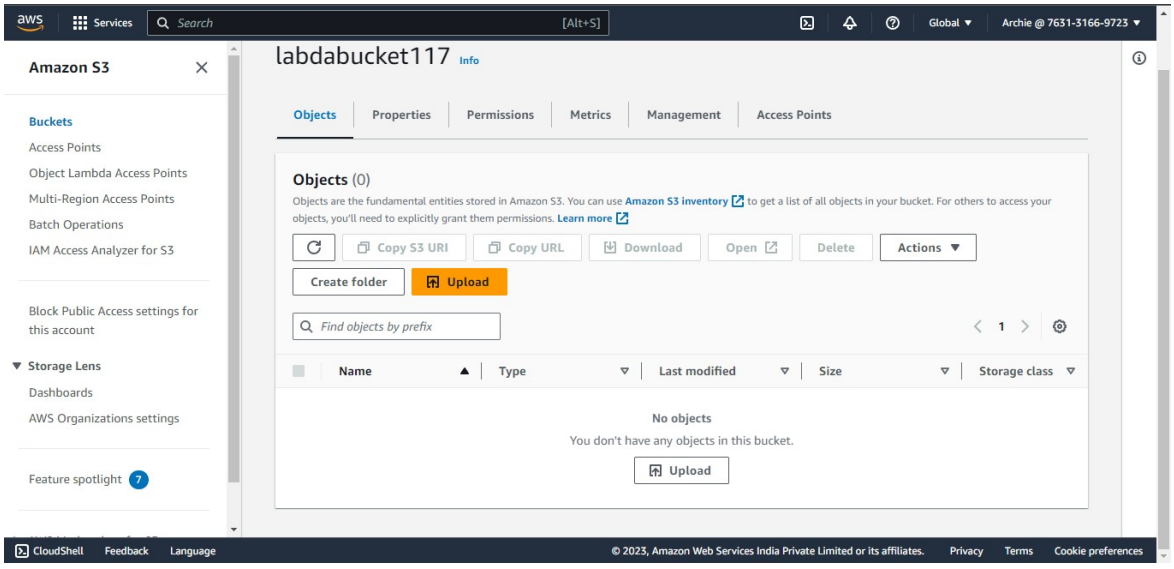
9. Add the python code and click on deploy to save the changes.

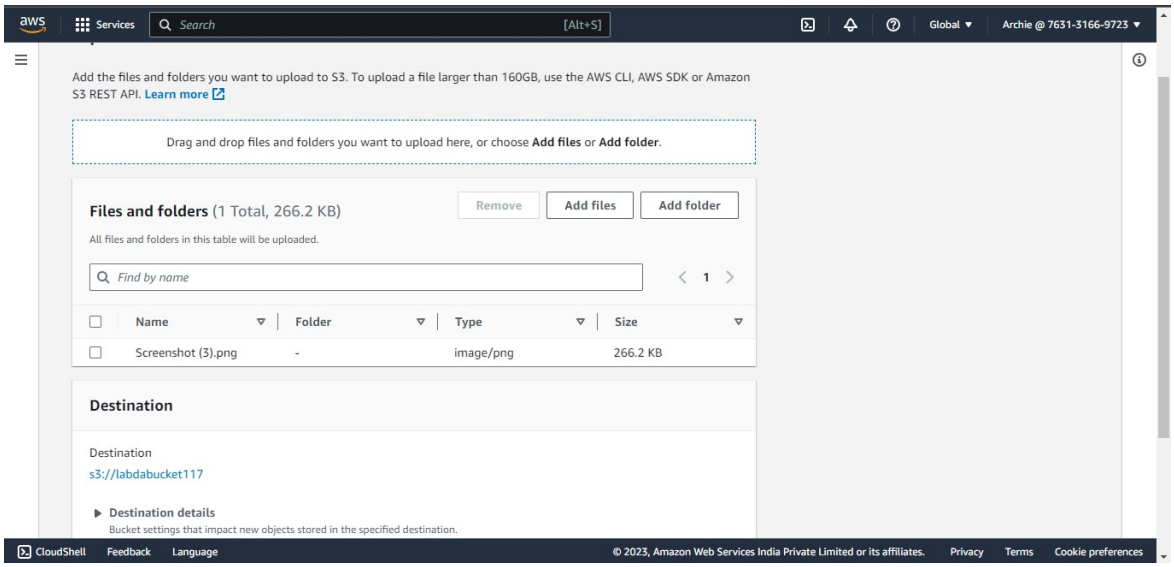


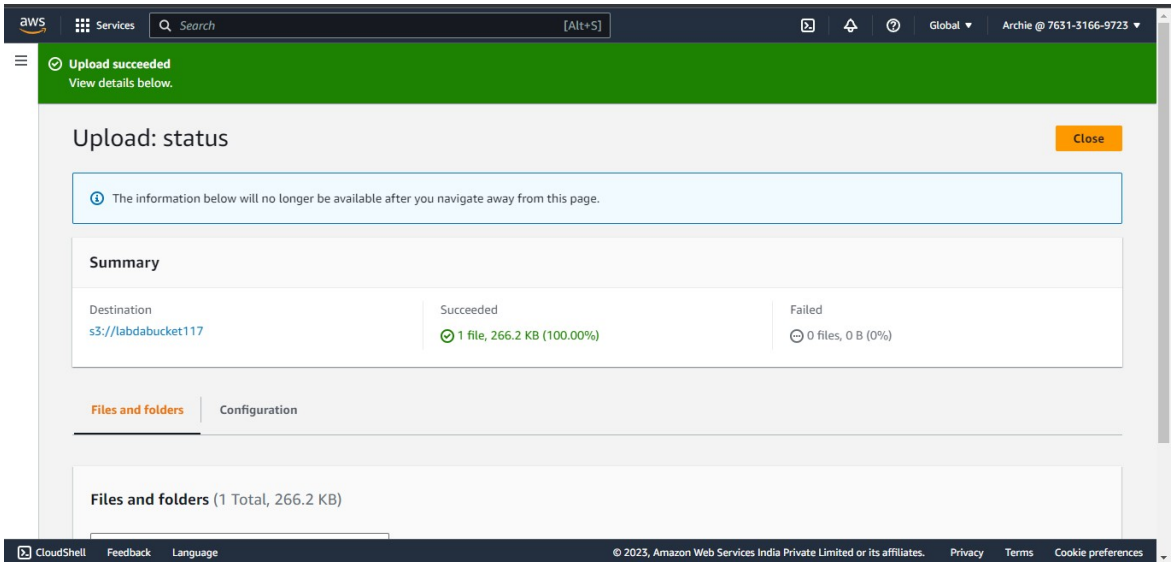
10. Scroll above and click on ADD TRIGGER. Select the following options and click on Done.



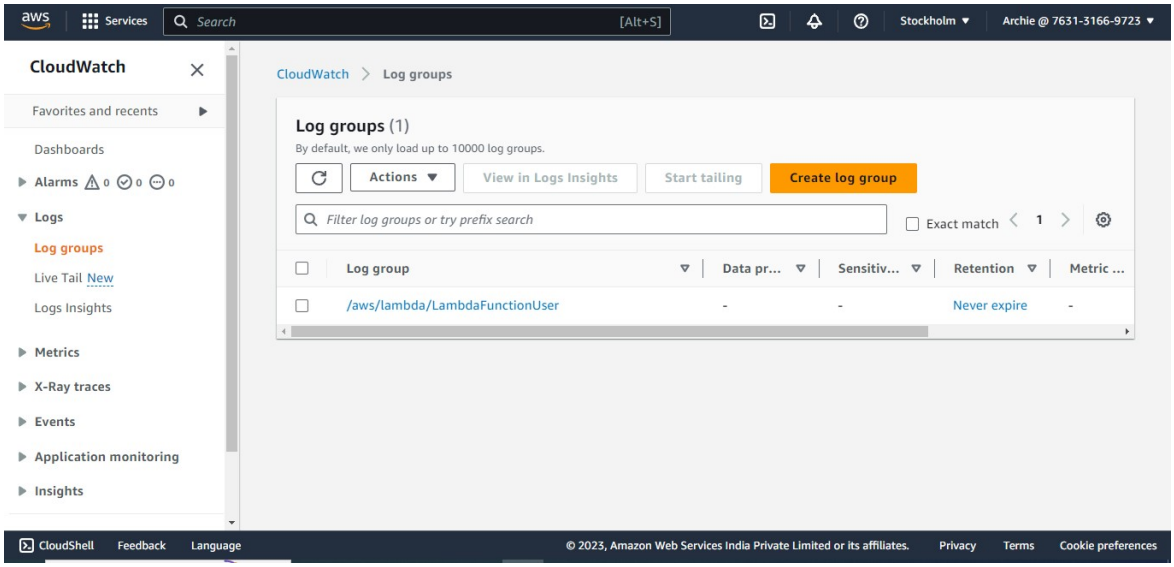
11. Go to S3 bucket and click on Add files. Select a image and click on Upload.



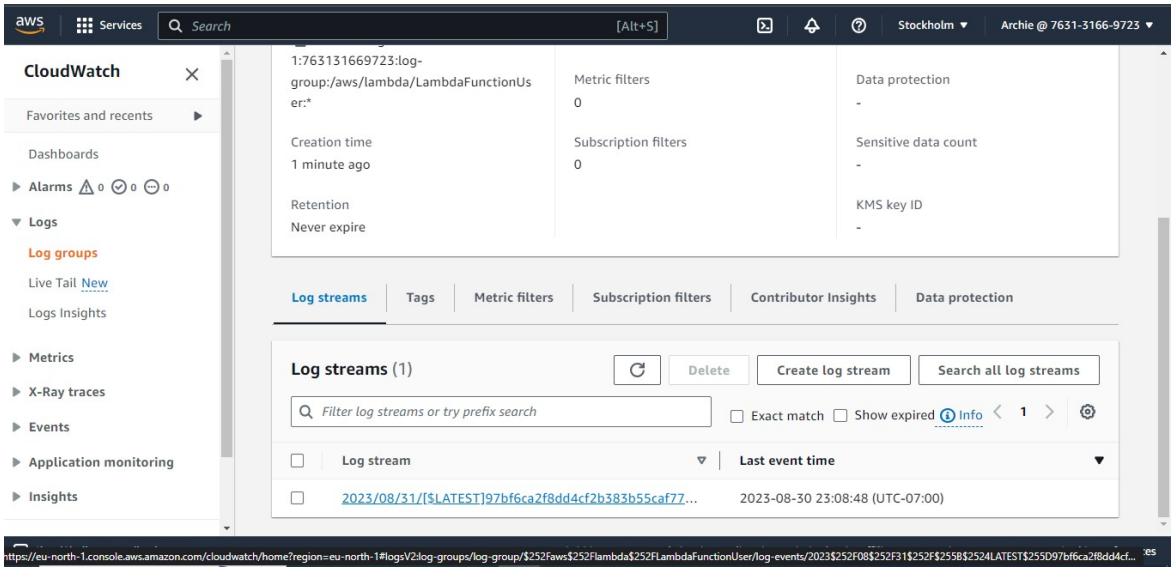


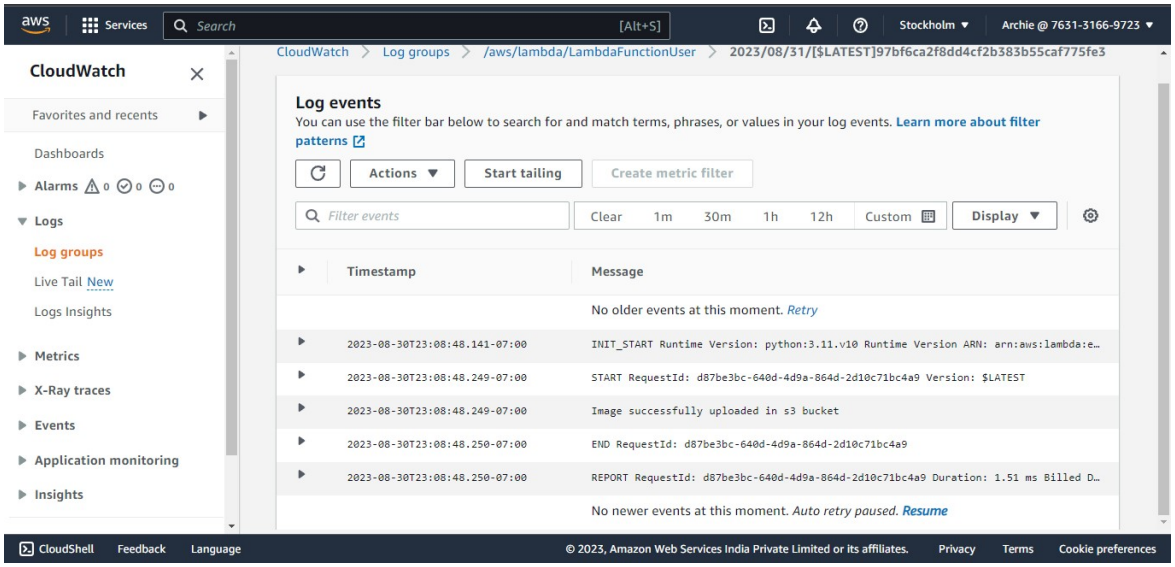


12. Search CloudWatch and go to Log groups. Select the existing Log Group.



13. Click on the link provided and then you will see the message displayed.





CONCLUSION: In this assignment, we learnt how to create a Lambda function which will log "An Image has been added" once you add an object to a specific bucket in S3.

Assignment Number 10

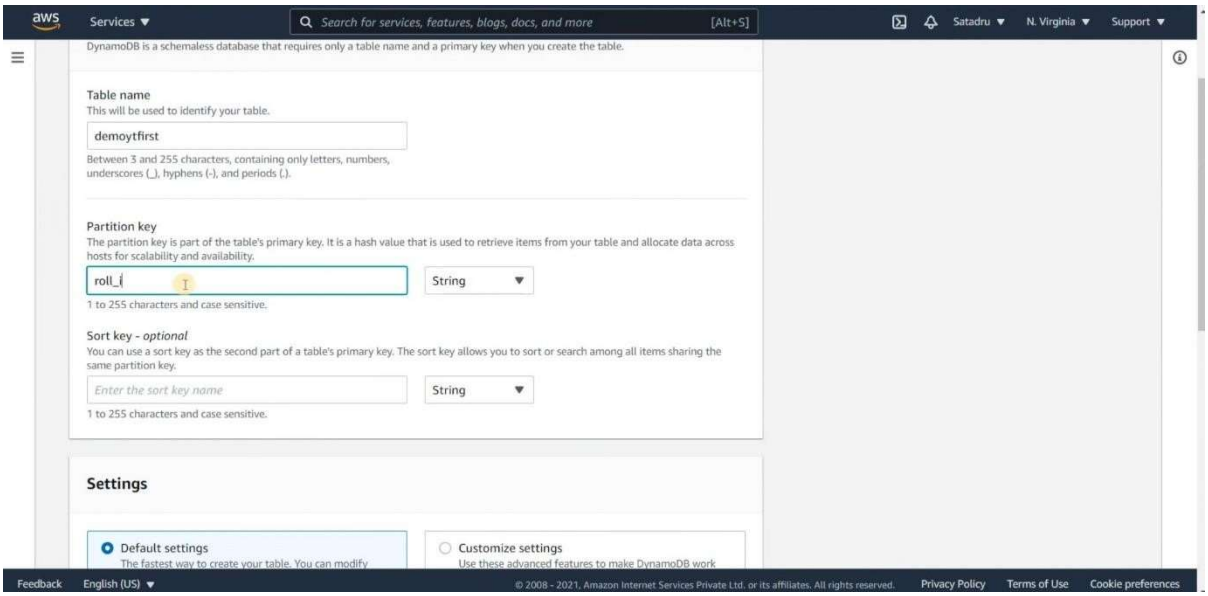
Name: Soham Satam ; Branch: I.T (T.E.); Roll Number: 109 17/10/2023

Aim: To create a Lambda function using Python for adding data to Dynamo DB database. LO mapped: LO6

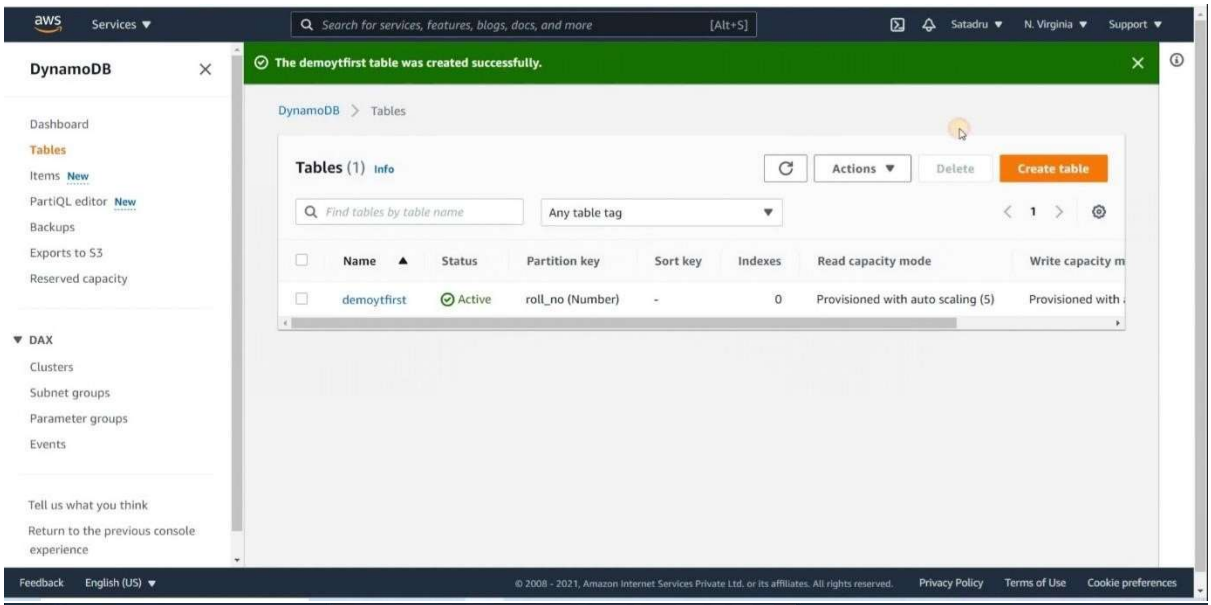
Theory:

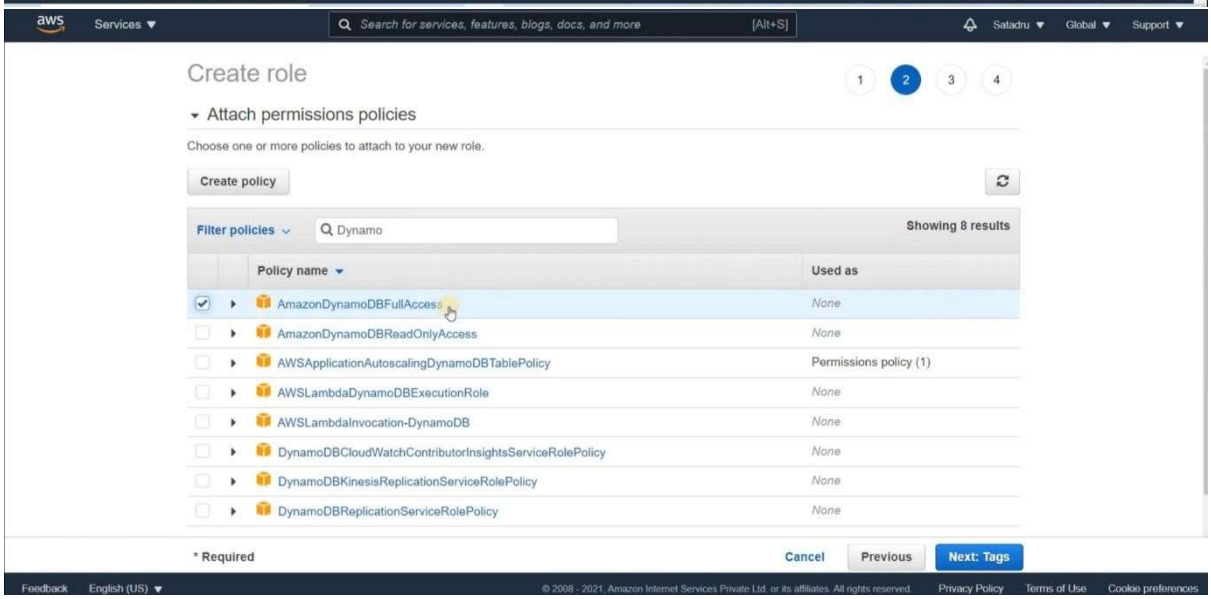
Step 1: Set Up AWS Environment

1. **Create an AWS Account:** If you don't have an AWS account, sign up for one at AWS Console.
2. **Access AWS Lambda Console:**



   - Go to the AWS Lambda Console.
   - Click on "Create function."
3. **Create a New Lambda Function**

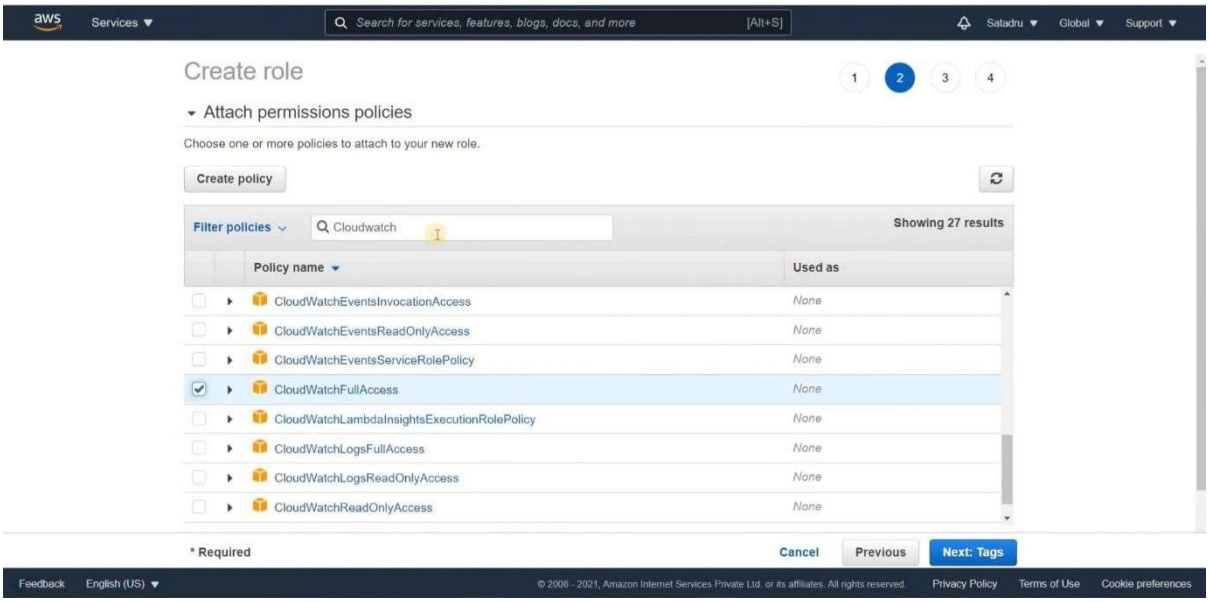




:

- Choose "Author from scratch."
- Enter a name for your function (e.g., **AddToDynamoDBFunction**).
- Choose "Python" as the runtime.

4. **Configure Execution Role**

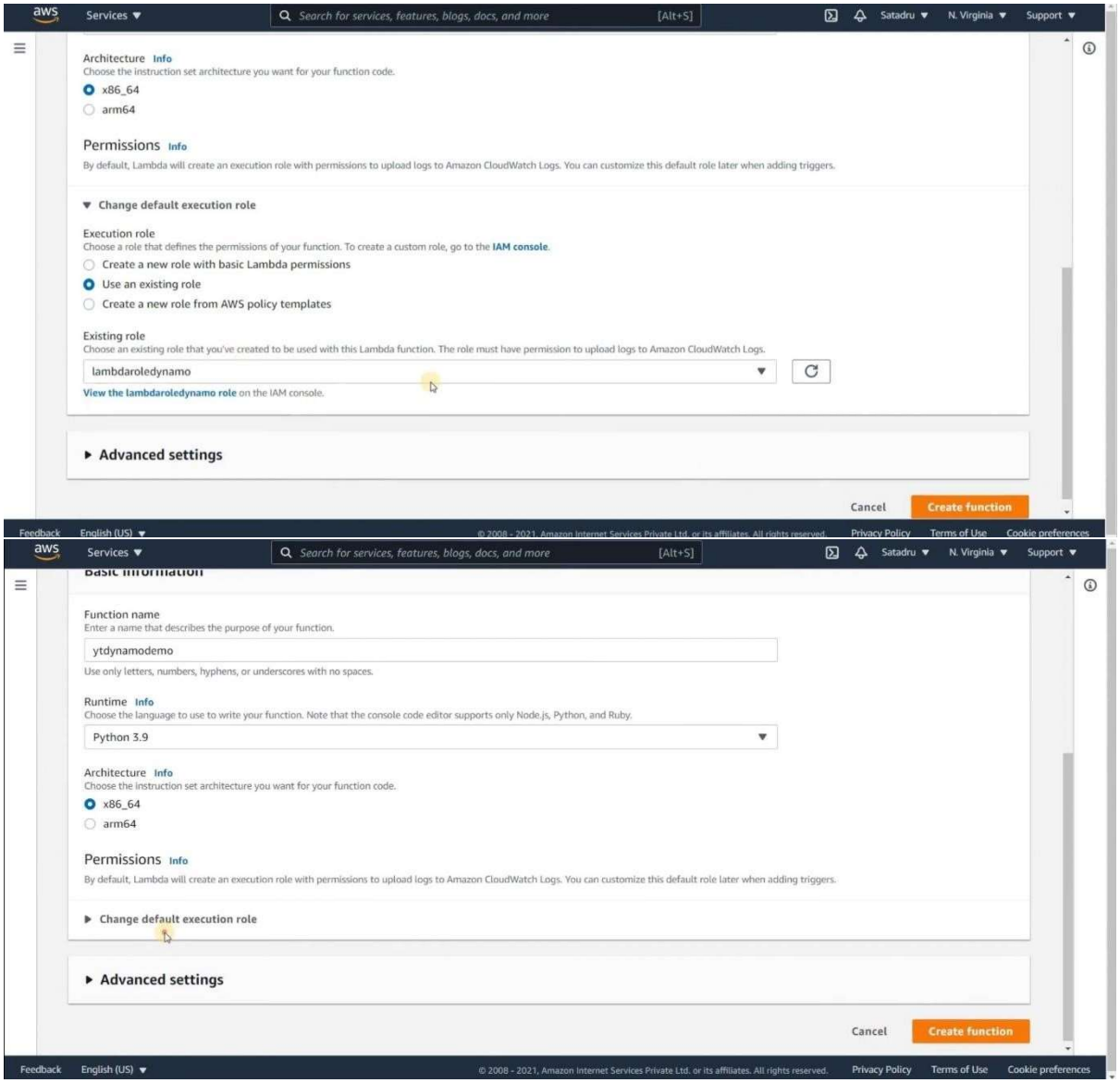


:

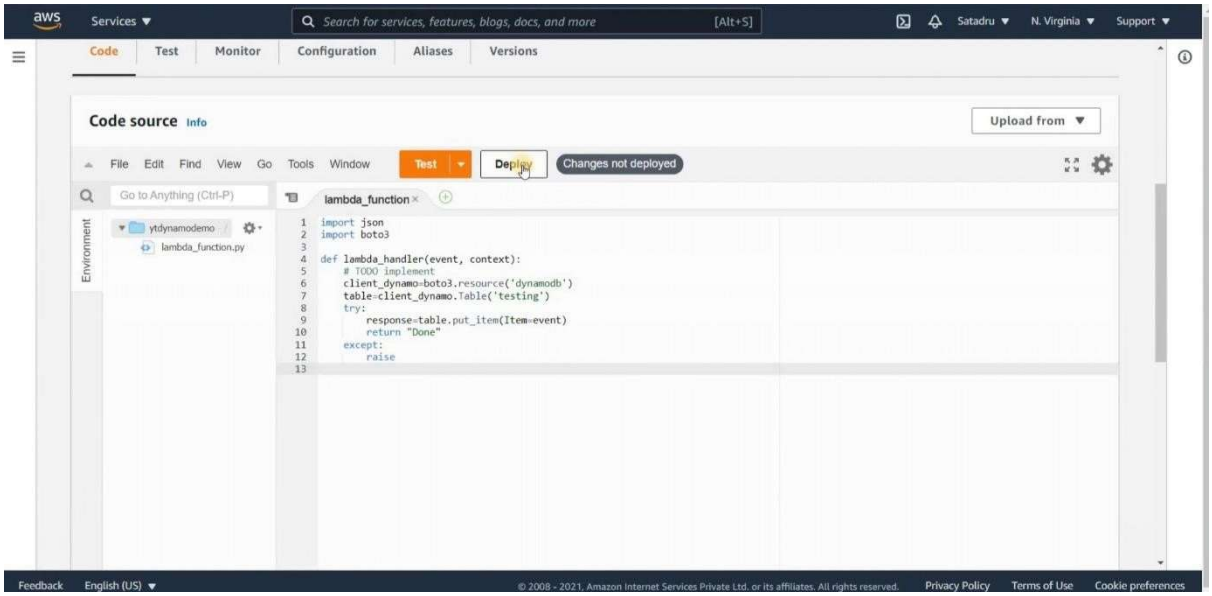
- Create a new role with basic Lambda permissions and additional permissions to interact with DynamoDB.
- Attach the role to your Lambda function.

Step 2: Set Up DynamoDB

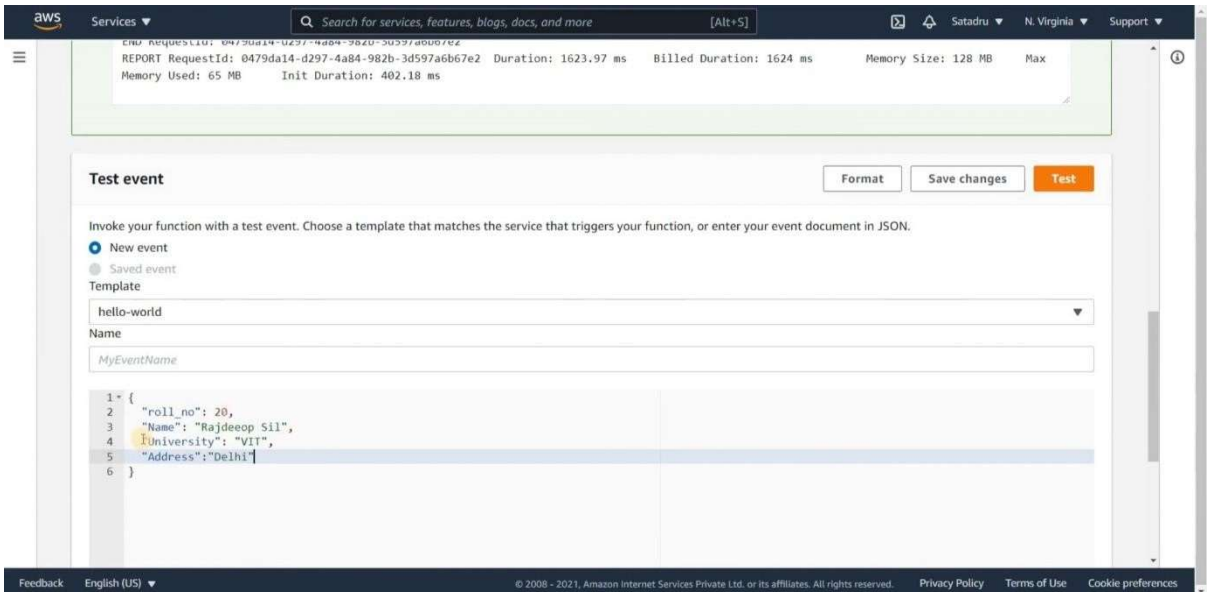
1. **Access DynamoDB Console:**
   - Go to the AWS DynamoDB Console.



2. **Create a DynamoDB Table:**
    - Click on "Create table."
    - Enter a table name and a primary key (e.g., **ID**).
    - Configure other settings as needed and create the table.



Step 3: Write Lambda Function Code



1. Ensure to replace **'YourDynamoDBTableName'** with the actual name of your DynamoDB table.

2. **Configure Lambda Handler:**

    - In the Lambda function configuration, set the handler to **filename.lambda_handler** (e.g., **yourfilename.lambda_handler**).

Step 4: Configure Environment Variables

1. **Add DynamoDB Table Name:**

    - In the Lambda function configuration, add an environment variable (e.g., **DYNAMODB_TABLE**) with the value set to your DynamoDB table name.

Step 5: Test Locally

1. **Test Lambda Function Locally:**
    - Create a test event with sample data.
    - Test the Lambda function using the "Test" button in the Lambda console.

Step 6: Deploy Lambda Function

1. **Deploy the Lambda Function:**
    - Click on the "Deploy" button in the Lambda console.

Step 7: Test in AWS Lambda Console

1. **Test in AWS Lambda Console:**
    - Use the Lambda console to test the function by creating a test event with sample data.
    - Verify that the function executes successfully.

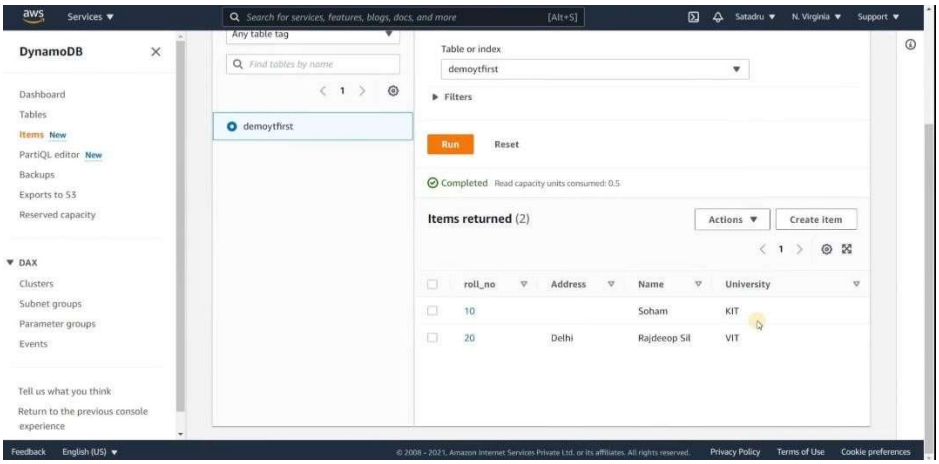
Step 8: Monitor and Troubleshoot

1. **Check CloudWatch Logs:**
    - Use CloudWatch Logs to check logs generated by your Lambda function for troubleshooting.

Step 9: Documentation

1. **Create Documentation:**
    - Document the purpose, input parameters, and expected output of your Lambda function.
    - Explain any challenges faced during development and how they were resolved.

By following these steps, you can create a Lambda function using Python to add data to a DynamoDB database in the AWS environment.



Conclusion: By this assignment we learn how to create a Lambda function using Python for adding data to Dynamo DB database.

Adv. DevOps Written Assignment : 01

1. what security measures can be taken while using Kubernetes?

1. Role-Based Access Control (RBAC): RBAC restricts who can perform actions within a Kubernetes cluster. It defines roles and role bindings to specify what resources and operations users or service accounts can access. This prevents unauthorized access and actions within the cluster.

2. Regular Updates: Keeping Kubernetes and its components up to date is crucial. New releases often include security patches. Regular updates help mitigate known vulnerabilities and ensure your cluster remains secure.

3. Network Policies: Network policies allow you to define rules for communication between pods. By specifying which pods can communicate with each other, you can limit the attack surface and prevent unauthorized access.

4. Container Security Tools: Employ container security tools like vulnerability scanners to assess the security of container images. These tools can identify and remediate vulnerabilities in the containerized applications before they are deployed.

5. Monitoring and Audit: Implement monitoring and auditing solutions to track cluster activity. This helps detect and respond to suspicious or unauthorized behavior. Tools like Prometheus and Grafana can be used for monitoring, while audit logs provide insights into cluster activity.

6. Secrets Management: Sensitive data like API keys, passwords, and certificates should be stored securely using Kubernetes secrets or external vaults. This prevents sensitive information from being exposed within containers or configuration files.

7. PodSecurityPolicies (PSP): PSP is a Kubernetes feature that enforces security policies at the

pod level. It allows you to define restrictions on privilege escalation, host access, and other security-sensitive configurations for pods.

8. Namespaces: Use Kubernetes namespaces to logically isolate workloads. This provides a level of separation between different applications or teams, reducing the risk of unauthorized access or interference between them.

9. Admission Controllers: Admission controllers are webhook plugins that intercept and validate requests to the Kubernetes API server. You can use them to enforce custom policies and ensure that only compliant resources are admitted to the cluster.

10. Container Runtime Security: Implement container runtime security solutions like Docker Security Scanning or container runtime protection tools. These tools monitor containers at runtime for abnormal behavior, helping to detect and respond to potential threats.

Combining these measures into a comprehensive security strategy is essential for safeguarding your Kubernetes cluster and the applications running within it. It's important to stay informed about best practices and evolving security threats in the Kubernetes ecosystem.

2. What are the three security techniques that can be used to protect data?

Three security techniques commonly used to protect data are:

1. Encryption: Encryption is the process of converting data into a secure format that can only be read by someone with the decryption key. It ensures that even if unauthorized parties access the data, they cannot understand it without the correct key. Two common types of encryption are:

- Data-at-rest Encryption: Protects data when it's stored on disk or in a database.
- Data-in-transit Encryption: Secures data as it's transmitted between systems over networks.

2. Access Control: Access control mechanisms regulate who can access data and what actions they can perform on it. This involves setting permissions, roles, and policies to ensure that only authorized users or applications can access and manipulate data. Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) are commonly used access control models.

3. Data Masking/Redaction: Data masking or redaction involves obscuring or replacing sensitive data with fictitious or scrambled values. This is often used in non-production environments or when sharing data with third parties. It ensures that even if someone gains access to the data, they cannot see the actual sensitive information.

These techniques are often used in combination to create a layered approach to data security, providing multiple levels of protection to safeguard sensitive information from unauthorized access and disclosure.

3. How do you expose a service using ingress in Kubernetes?

To expose a service using Ingress in Kubernetes, you need to follow these steps:

1. Set up Kubernetes: Ensure you have a Kubernetes cluster up and running, and you have the `kubectl` command-line tool configured to communicate with the cluster.

2. Deploy Your Application: Deploy your application as a Kubernetes Deployment or a Pod, and create a Kubernetes Service to expose it internally within the cluster. This Service will be the target for the Ingress.

3. Install an Ingress Controller: You need to have an Ingress controller installed in your cluster. Some popular options include Nginx Ingress Controller, Traefik, or HAProxy Ingress. The controller will manage the Ingress resources and configure the load balancer.

For example, to install the Nginx Ingress Controller, you can use:

```bash
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.0.0/deploy/static/provider/cloud/deploy.yaml
```

4. Create an Ingress Resource: Define an Ingress resource that specifies the rules for routing traffic to your service. Here's an example Ingress resource manifest:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

```
  name: my-ingress
spec:
  rules:
    - host: example.com
      http:
        paths:
          - path: /path
            pathType: Prefix
            backend:
              service:
                name: your-service
                port:
                  number: 80
```

In this example, traffic for `example.com/path` will be routed to `your-service`.

5. Apply the Ingress Resource: Use `kubectl apply` to create the Ingress resource in your cluster:

```bash
kubectl apply -f your-ingress.yaml
```

6. Configure DNS: Ensure that the DNS records for the specified hostname (e.g., `example.com`) point to the external IP address of your Ingress controller.

7. Access Your Service: After DNS propagation, you should be able to access your service externally via the hostname and path you defined in the Ingress resource.

4. Which service protocols does Kubernetes ingress expose?

Kubernetes Ingress is primarily designed to expose HTTP and HTTPS services, making it suitable for routing and load balancing web traffic. However, with the evolution of Kubernetes and Ingress controllers, it has expanded to support additional protocols and features:

1. HTTP: Ingress is commonly used to expose HTTP services. You can define routing rules based on URL paths, hostnames, and other HTTP attributes.

2. HTTPS: Secure HTTP services can be exposed through Ingress by configuring TLS certificates. This allows you to terminate SSL/TLS encryption at the Ingress controller and route decrypted traffic to your services.

3. TCP: Some Ingress controllers, like Nginx Ingress, support TCP services. This enables you to expose non-HTTP services such as databases or custom protocols. TCP-based routing typically relies on port numbers.

4. UDP: While less common, some Ingress controllers support UDP services. UDP is a connectionless protocol used for various purposes, including DNS and VoIP. Exposing UDP services may require specific controller support.

5. gRPC: If your services use the gRPC protocol, you can configure Ingress resources to handle gRPC traffic. gRPC is a high-performance RPC (Remote Procedure Call) framework often used for communication between microservices.

6. WebSocket: Ingress controllers can be configured to support WebSocket connections. WebSocket is a protocol that enables full-duplex communication over a single TCP connection and is used for real-time applications.

7. Custom Protocols: In some cases, you may need to expose services using custom or proprietary protocols. Depending on your Ingress controller and its capabilities, you might be able to configure it to handle these custom protocols.

Additionally, Ingress controllers often evolve, so it's essential to refer to the documentation and features of the specific controller you plan to use to ensure compatibility with your service protocols.

WRITTEN ASSIGNMENT 2

Q1. How to deploy lambda function on AWS?

Deploying a Lambda function on AWS involves several steps. Lambda is a serverless compute service that lets you run code without provisioning or managing servers. Here's a step-by-step guide on how to deploy a Lambda function:

Step 1: Create a Lambda Function

1. Log in to the AWS Management Console.

2. Navigate to the Lambda service by searching for it in the AWS Services section.

3. Click the "Create function" button.

Step 2: Configure Your Function

4. Choose "Author from scratch."

5. Fill in the basic information for your function, including the name, runtime, and execution role.

6. For "Execution role," you can create a new role from a template or use an existing role if you have one with the necessary permissions.

7. Click the "Create function" button.

Step 3: Upload Your Code

8. In the "Function code" section, you can upload your code either directly (ZIP file or folder) or by specifying a repository from AWS CodeCommit, Amazon S3, or other options.

9. Configure the handler if needed. The handler is the method that AWS Lambda invokes.

10. Set environment variables if your code requires them.

11. Adjust the runtime settings if necessary.

12. Click the "Deploy" button to upload your code.

Step 4: Define the Trigger

13. In the "Add triggers" section, you can specify event sources that will trigger your Lambda function. This can be an API Gateway, an S3 bucket, an SNS topic, etc.

14. Configure the trigger according to your needs and grant permissions as required.

Step 5: Test Your Function

15. You can test your Lambda function by creating a test event or using a sample test event provided by AWS.

16. Click the "Test" button to run your function and see the results.

Step 6: Monitor and Troubleshoot (Optional)

17. AWS Lambda provides various monitoring and logging options. You can configure CloudWatch alarms, inspect logs, and set up notifications to keep an eye on your function's performance.

Step 7: Save and Deploy the Function

18. After you've configured everything to your satisfaction, click the "Save" button to save your changes.

19. Click the "Deploy" button to make your function live and ready to respond to triggers.

Step 8: Invocation and Scaling

Your Lambda function is now deployed and ready to be invoked by the trigger you configured. AWS Lambda handles the scaling of resources based on the incoming workload automatically.

Q2. What are the deployment options for AWS Lambda?

AWS Lambda offers several deployment options:

1. Code Upload: You can directly upload your function code as a ZIP file or a deployment package when creating or updating your Lambda function.

2. AWS Lambda Layers: You can use Lambda Layers to separate your function code from its dependencies. This allows you to manage and version common libraries independently and share them across multiple functions.

3. **AWS SAM (Serverless Application Model):** AWS SAM is a framework for building serverless applications. You can define your Lambda functions and their associated resources in a SAM template file, then deploy the entire application using AWS SAM CLI.

4. AWS CloudFormation: You can use AWS CloudFormation templates to define and deploy Lambda functions along with other AWS resources. This enables infrastructure as code (IaC) for your serverless applications.

5. AWS Serverless Application Repository: You can publish and share serverless applications and Lambda functions using the AWS Serverless Application Repository. Users can deploy your applications directly from the repository.

6. AWS CodePipeline: You can integrate AWS Lambda deployment into a continuous integration/continuous deployment (CI/CD) pipeline using AWS CodePipeline. This automates the building, testing, and deployment of your Lambda functions.

7. Container Image Support: AWS Lambda now supports deploying functions as container images, allowing you to package your function and dependencies in a Docker container and deploy them as a Lambda function.

These deployment options provide flexibility and enable you to choose the one that best fits your application architecture and development workflow.

Q3 What are the 3 full deployment modes that can be used for AWS?

In the context of AWS Lambda, there are three primary deployment modes:

1. Automatic Deployment: AWS Lambda provides automatic deployment when you directly upload your function code as a ZIP file or specify a deployment package. This is the most common and straightforward deployment method, and it's suitable for most use cases.

2. Infrastructure as Code (IaC) with CloudFormation: AWS CloudFormation is a service that allows you to define and provision AWS infrastructure and resources in a template. You can use CloudFormation to define your Lambda functions, their associated resources, and any other AWS resources your application needs. When you create or update the CloudFormation stack, it deploys the Lambda functions and other resources defined in the template. This approach is more suitable for complex serverless applications and infrastructure orchestration.

3. Serverless Application Model (AWS SAM): AWS SAM is an open-source framework for building serverless applications. It extends AWS CloudFormation to provide a simplified way to define serverless resources, including Lambda functions. You can use AWS SAM to define your functions and related resources in a SAM template, and then deploy the entire application using AWS SAM CLI. This approach is a balance between the simplicity of direct deployment and the power of CloudFormation for more complex applications.

These deployment modes offer different levels of control and abstraction, allowing you to choose the one that best matches your deployment needs and development workflow.

Q4. What are the 3 components of AWS Lambda?

AWS Lambda consists of three primary components:

1. Function Code: This is the core component of AWS Lambda. It's the code that you want to run in response to events. You can write your code in various supported programming languages (e.g., Node.js, Python, Java, C#, etc.). You can package your code along with its dependencies into a deployment package, which you upload to Lambda.

2. Event Sources: Event sources are triggers that invoke your Lambda function. AWS Lambda supports various event sources, such as Amazon S3, Amazon DynamoDB, Amazon SNS, AWS API Gateway, and more. When an event occurs in one of these services, it triggers the execution of your Lambda function.

3. Execution Environment: AWS Lambda manages the execution environment for your function. It automatically scales and provisions the necessary infrastructure to run your code. You don't need to worry about server provisioning or resource management. AWS Lambda also provides runtime support for various programming languages, so your code

runs in an isolated environment with the necessary resources to execute.