Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

CERTIFICATE

Certify that Mr./Miss Bhuvan Pravin Sawant

of ~~Information technology~~ Department, Semester V with

Roll No. 110 has completed a course of the necessary

experiments in the subject Advance devops under my

supervision in the **Thadomal Shahani Engineering College**

Laboratory in the year 2023 - 2024

Agarwal

Teacher In-Charge

Head of the Department

Date 20/10/23

Principal

TSEC

CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1.	To study & perform the setup of AWS EC2 service and launch an EC2 instance		18/7/23	
2.	To study & perform the setup of AWS Cloud9 service & launch a Python program in Cloud9		25/7/23	
3.	To study AWS S3 service & create a bucket for hosting static web application		1/8/23	
4.	To study ~~AWS S3 service & create~~ codepipeline deploy a web application using Codepipline		2/8/23	
5.	To understand the kubernetes cluster Architecture		12/9/23	
6.	To understand the terraform life cycle & to build change and destroy AWS infrastructure using Terraform		22/8/23	
7.	To perform static analysis on python programs using Sonar Qube SAST process		29/8/23	Signed 20/10/23
8.	To understand continuous monitoring using Nagios		28/8/23	
9.	To understand AWS lambda function & create a lambda function using python to log "An image has been added"		31/8/23	
10.	To create AWS lambda function using python for adding data to Dynamo DB database		5/9/23	
11.	Written Assignment 1		23/9/23	
12.	Written Assignment 2		5/10/23	

ASSIGNMENT - 1

AIM: To create an EC2 machine/instance using AWS

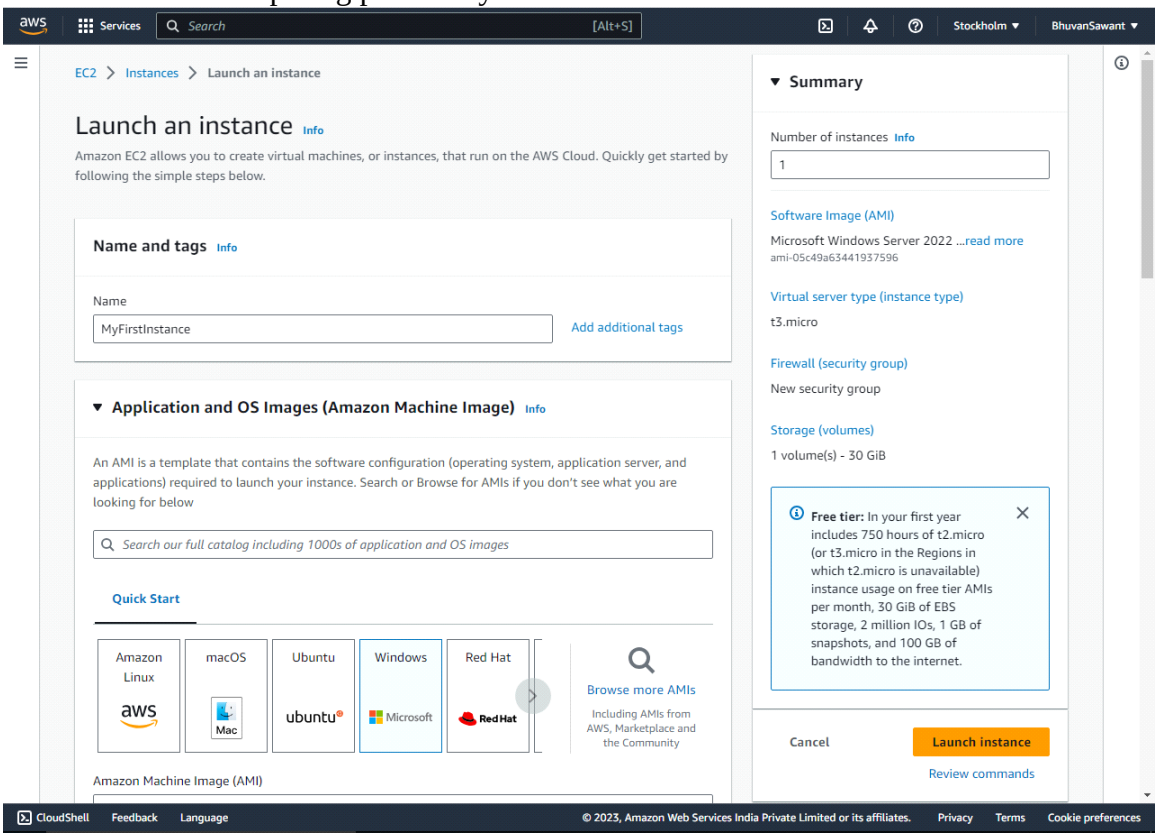
LO Mapped: LO1

Theory:
Amazon Elastic Compute Cloud is a part of Amazon.com's cloud-computing platform, Amazon Web Services, that allows users to rent virtual computers on which to run their own computer applications. It is designed to make web-scale cloud computing easier for developers. Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction.

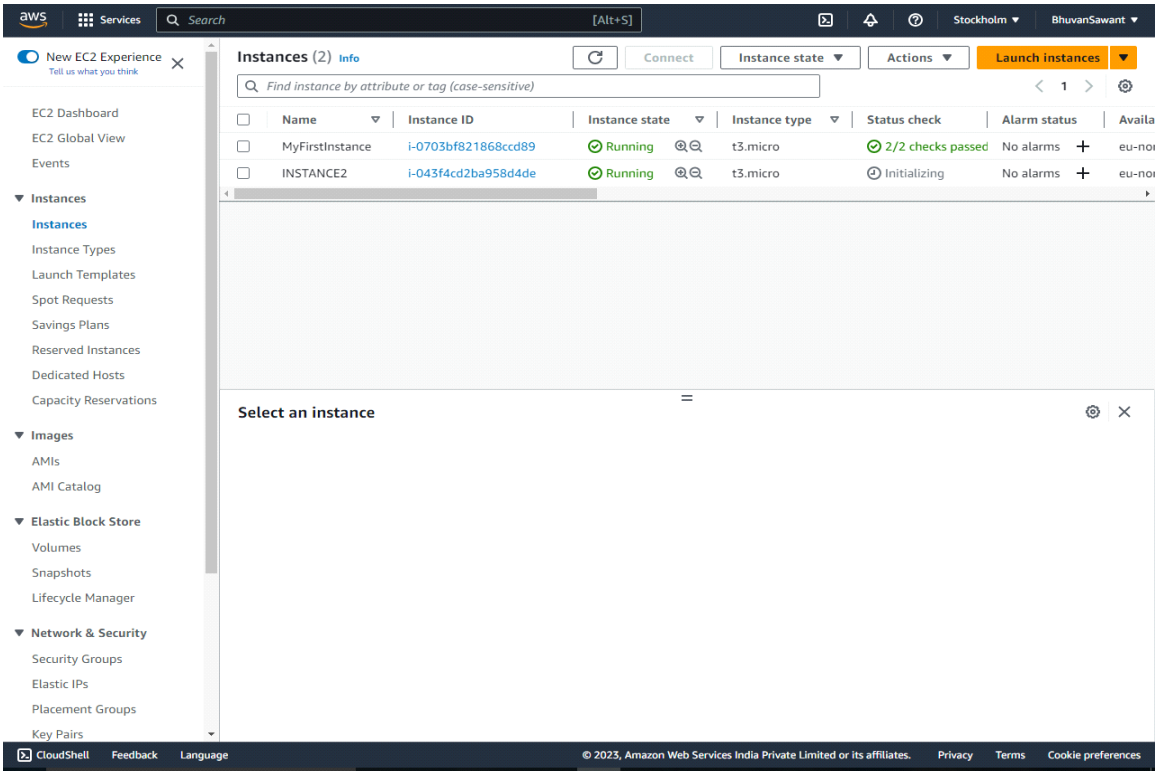
The following are the features of AWS:

- Flexibility
- Cost-effective
- Scalable and elastic
- Secure
- Experienced

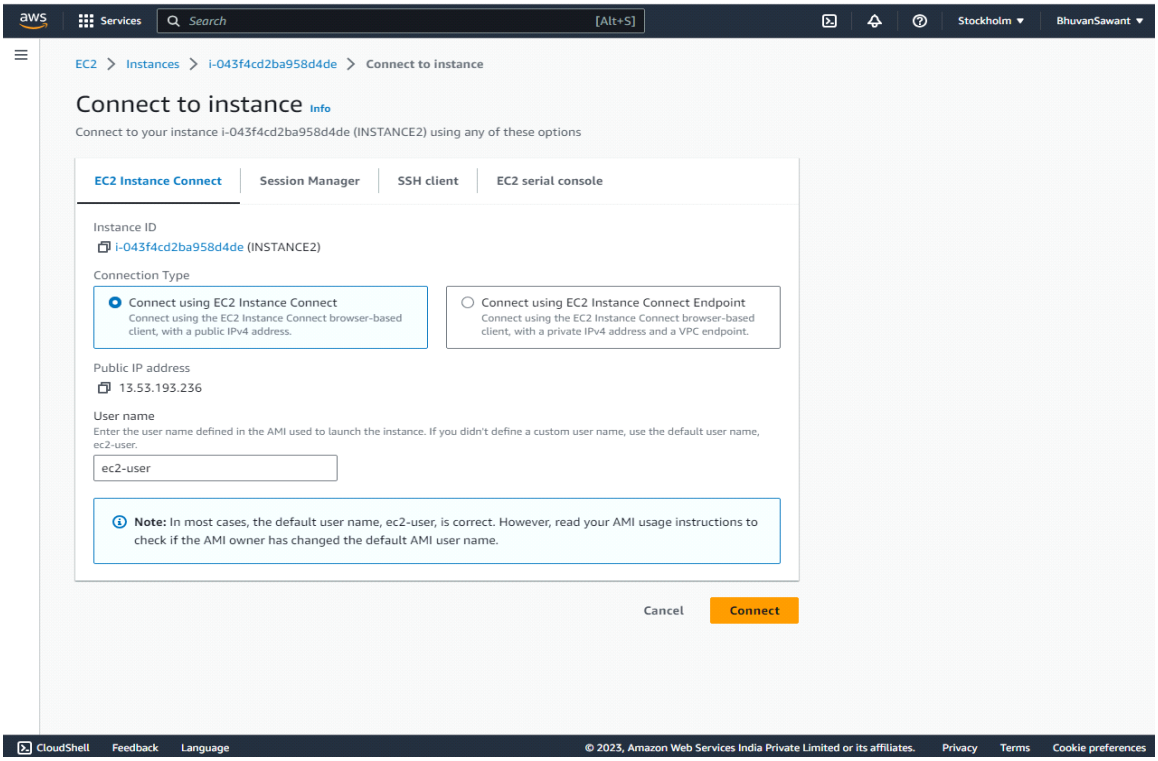
Steps: Login to AWS account, then search EC2.
Select the cloud computing platform you want to work on



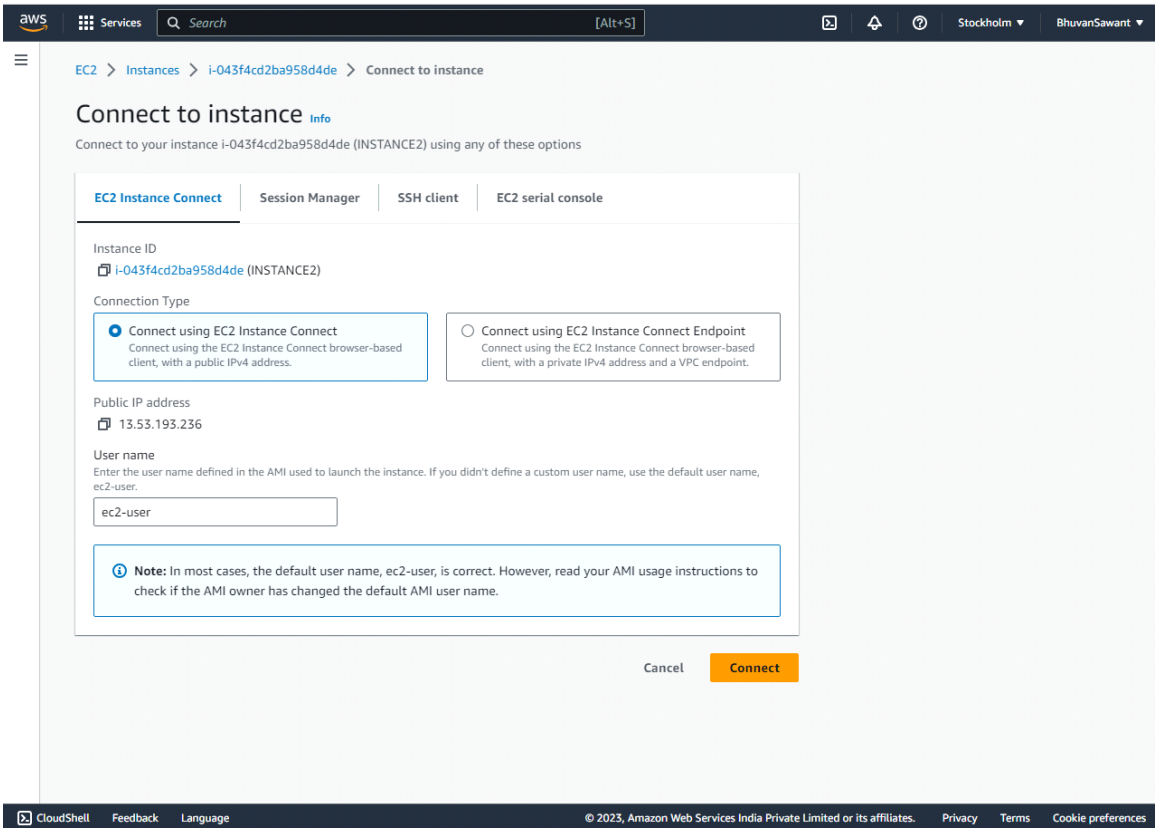
Now wait till the status check is 2/2 and instance is running.

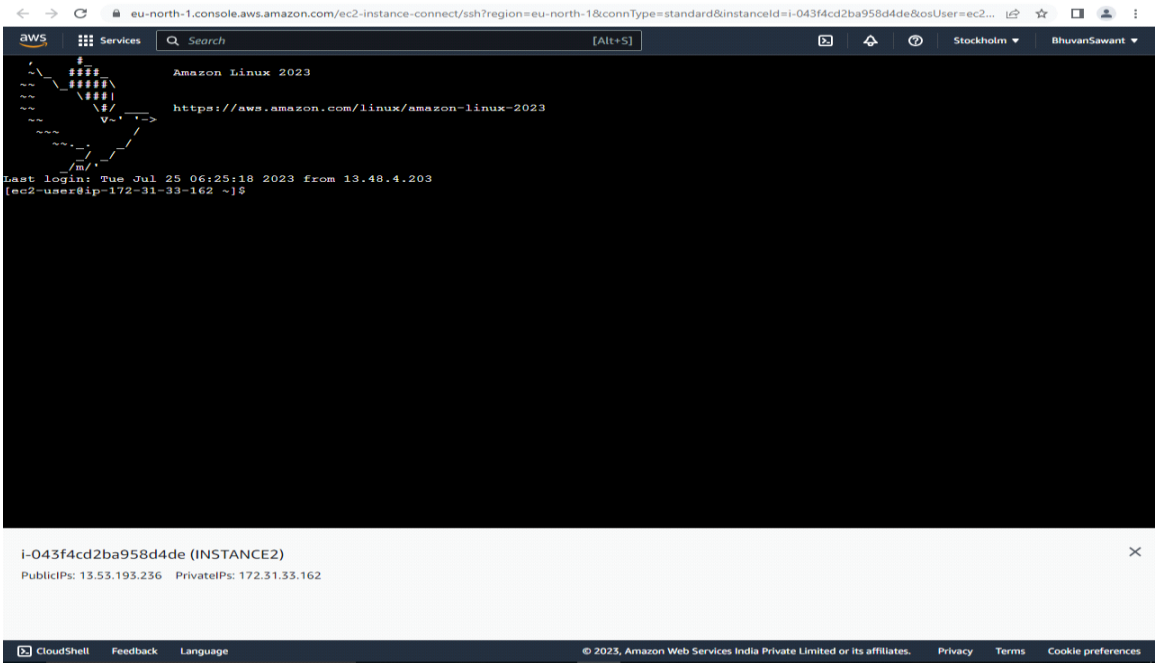


Once Check is complete click on launch instances.

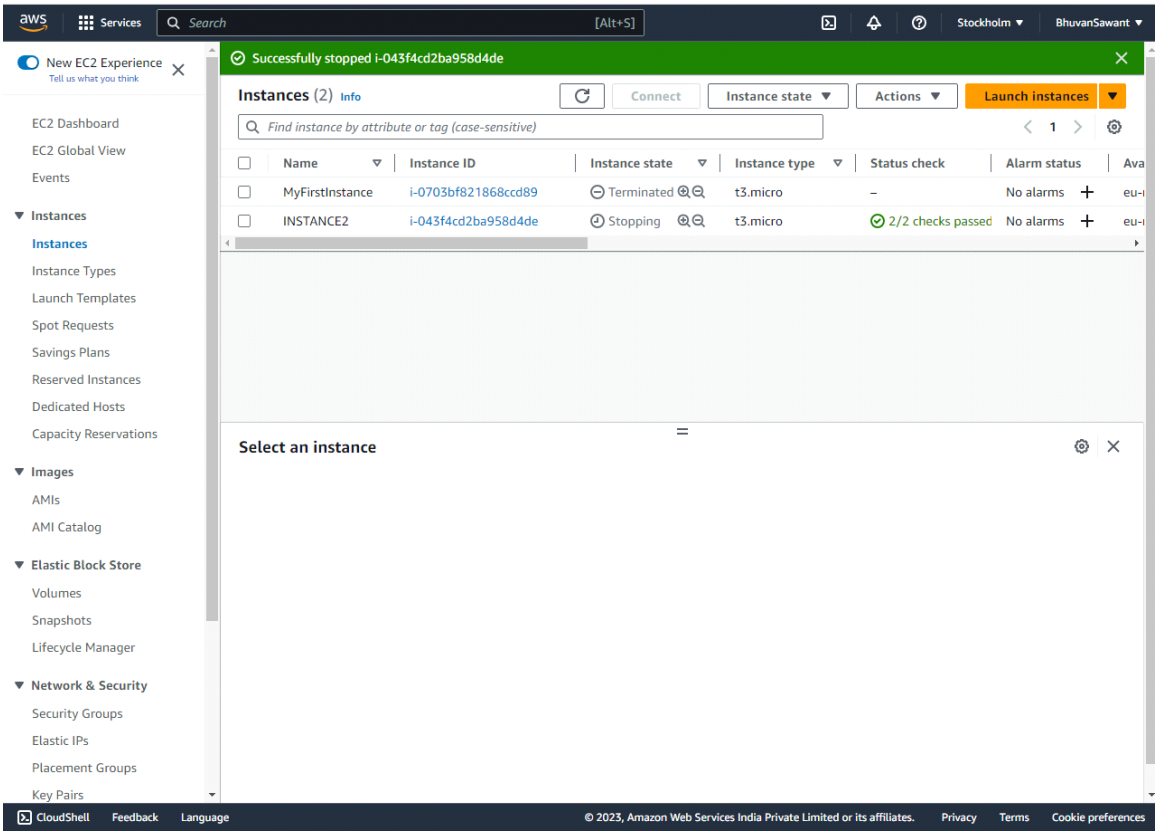


After connecting to instance successfully the portal is ready





Select the instances that you created and terminate them



CONCLUSION: Understanding the concept of EC2 and to create an EC2 machine/instance using AWS was performed successfully.

Assignment - 2

Aim: To understand the benefits of Cloud infrastructure and setup AWS Cloud9 IDE, Launch AWS Cloud9 IDE and Perform Collaboration Demonstration.

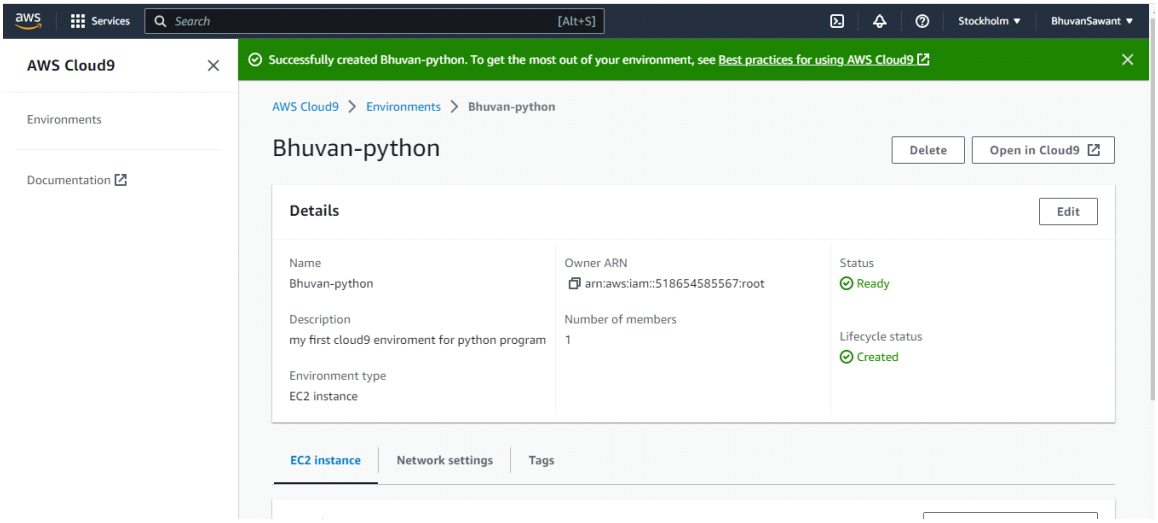
LO Mapped: LO1

Theory:

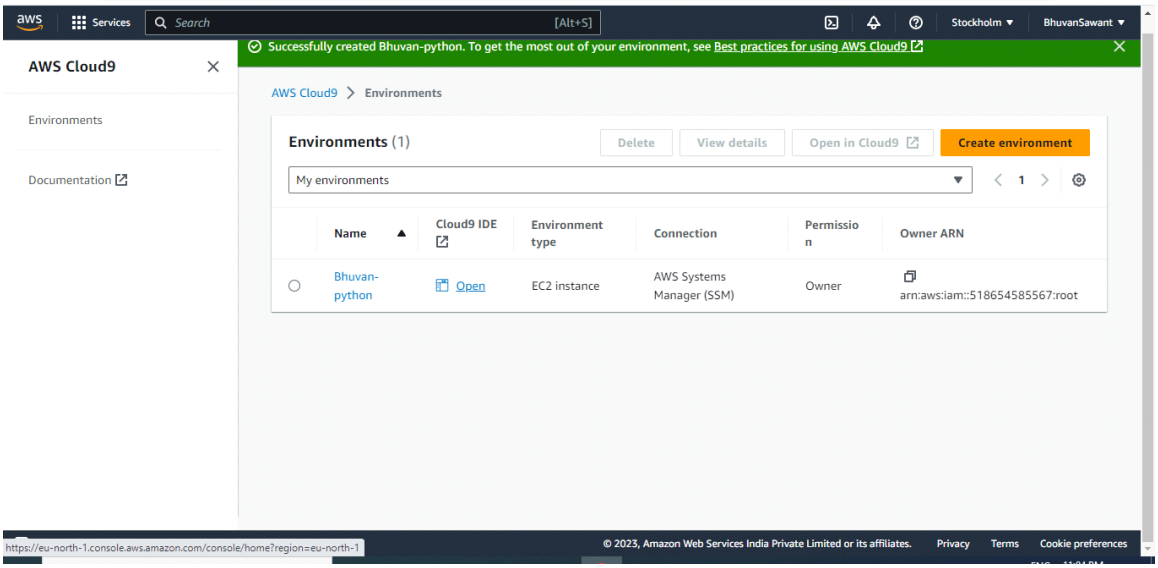
An integrated development environment (IDE) for the cloud called AWS Cloud9 enables you to write, run, and debug code using just a web browser. It has a terminal, debugger, and code editor. You don't need to install files or set up your development workstation to start new projects because Cloud9 comes preconfigured with necessary tools for popular programming languages like JavaScript, Python, PHP, and more. Since your Cloud9 IDE is cloud-based, you may use any internet-connected device to work on your projects from your home, office, or anywhere else. Additionally, Cloud9 offers a fluid development environment for serverless apps that makes it simple to declare resources, debug, and switch between local and remote execution.You can easily pair program and keep track of each other's inputs in real time with Cloud9, which enables you to swiftly share your working environment with your colleagues.

Output:

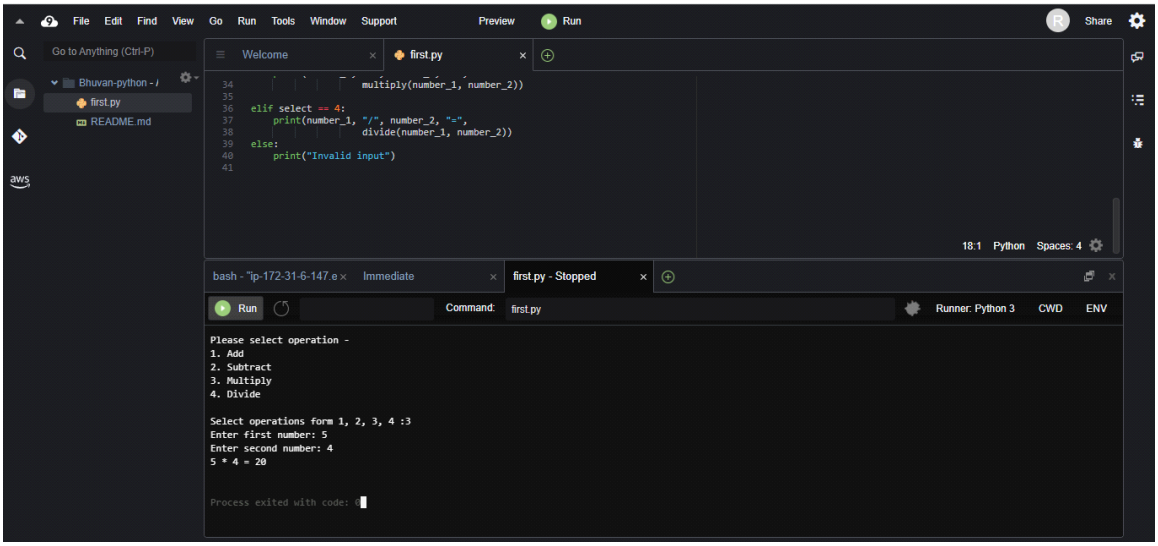
first login to your AWS account. Search for cloud9 in the search bar.
Then click on create environment. Fill the detail of your environment. After filling all details click on create button



After clicking create button you will redirect to following page. You can open the environment after clicking on Open.



In file section we created a python file and wrote a python program for calculator. After saving your program you can run program on clicking Run green button at top



Conclusion:

In this assignment we learnt and understood the steps create an Cloud9 environment.

Assignment No. - 3

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

LO Mapped: LO1, LO2

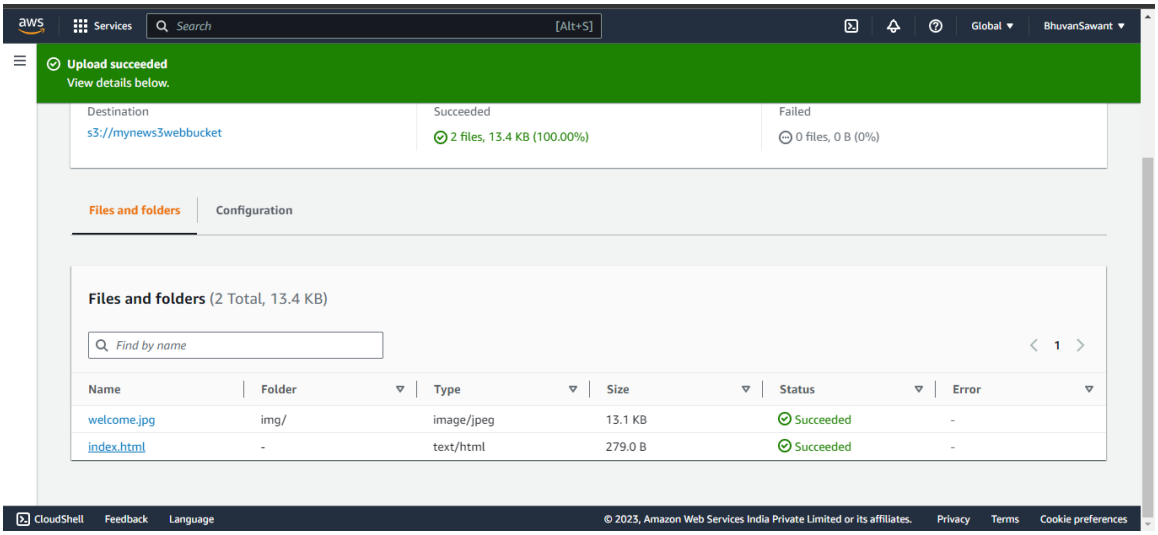
Theory:

Amazon Simple Storage Service (Amazon S3) is an object 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.

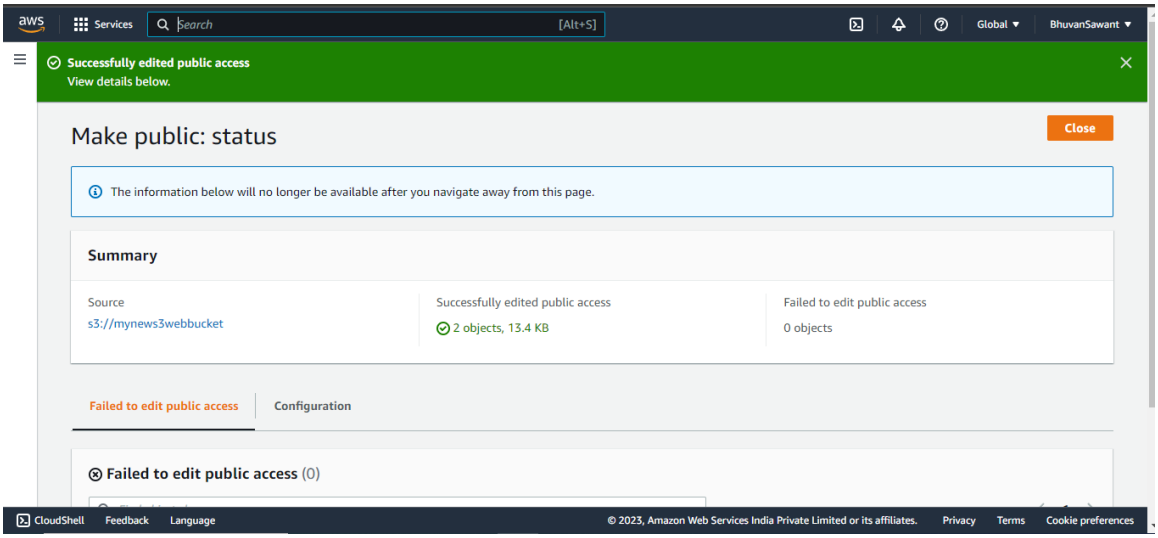
Some of the benefits of AWS S3 are:

- 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 encryption 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.

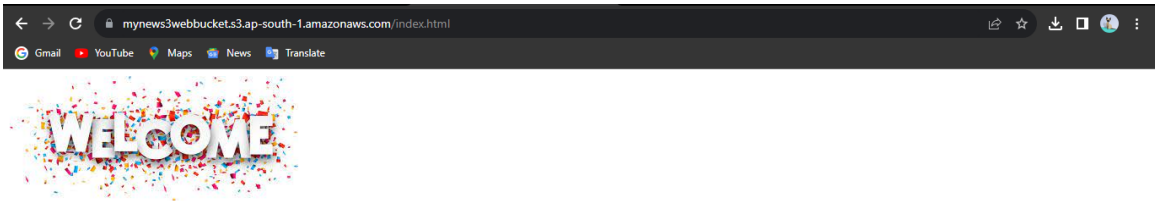
First login to your AWS account.



Then create a new S3 bucket.



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



Website for AWS

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

Assignment Number: 4

Name: bhuvan sawant Branch: IT/V Roll No.:110 Date:10/09/2023

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

LO mapped: LO1, LO2

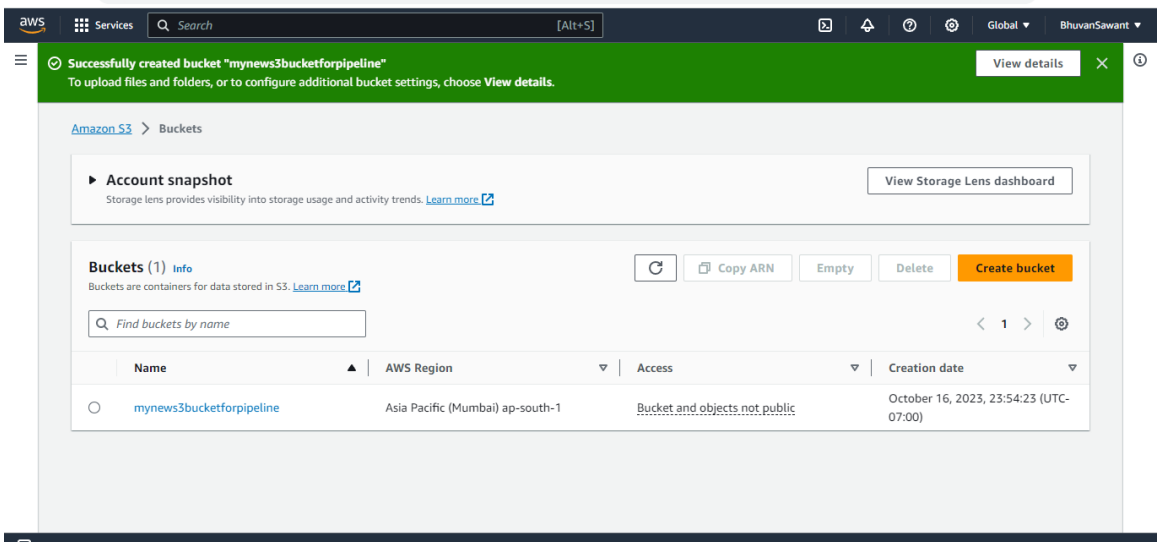
Theory:

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

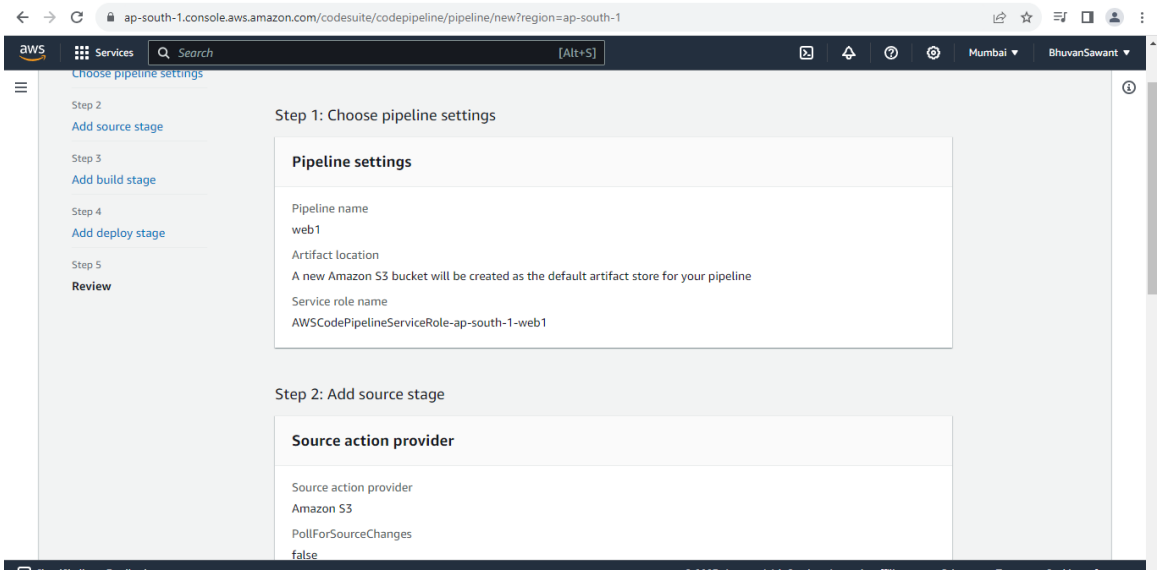
The key aspects of AWS CodePipeline:

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.

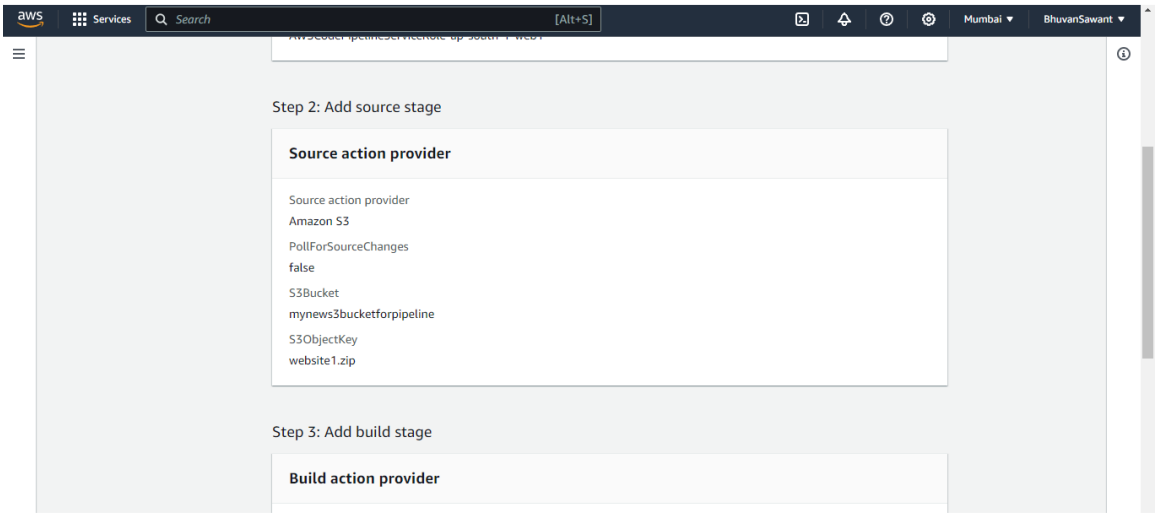


2. **Integration with Other AWS Services:**
    - CodePipeline integrates with various AWS services, such as AWS CodeBuild for building applications, AWS CodeDeploy for automating deployments, and AWS Lambda for running custom actions.

- 

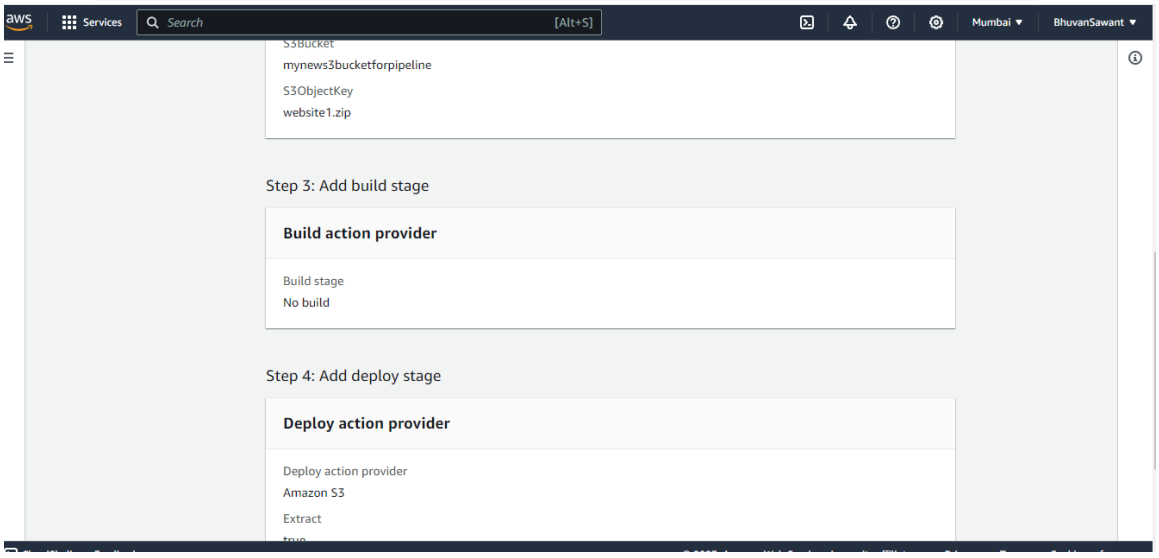
3. **Pipeline Execution:**

    - 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.



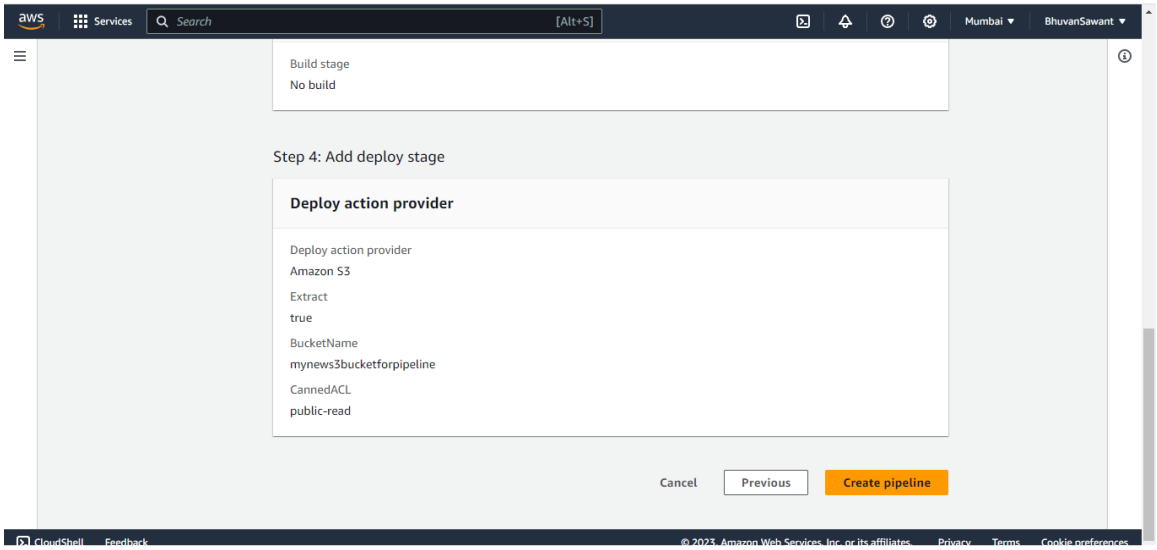
4. **Source Providers:**

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

- 

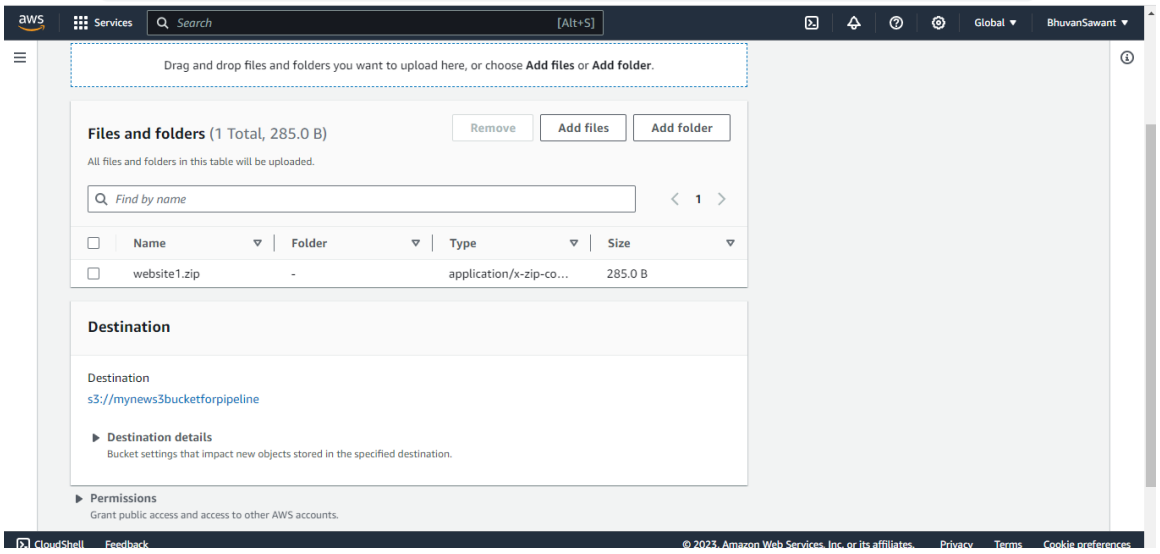
5. **Artifact Management:**

   - 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.

- 

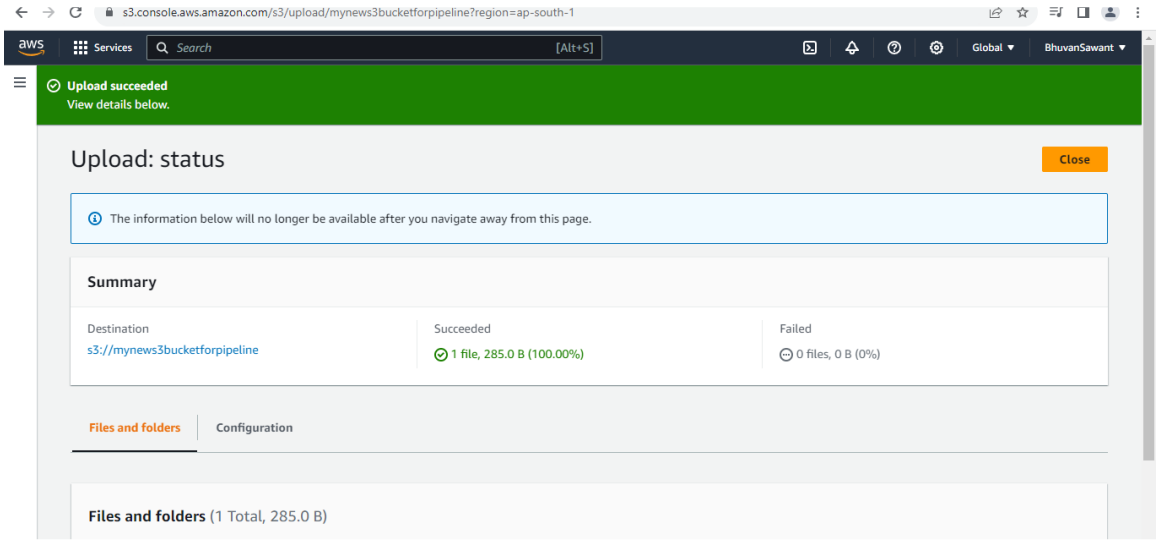
6. **Integration with Third-Party Tools:**

   - 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:**

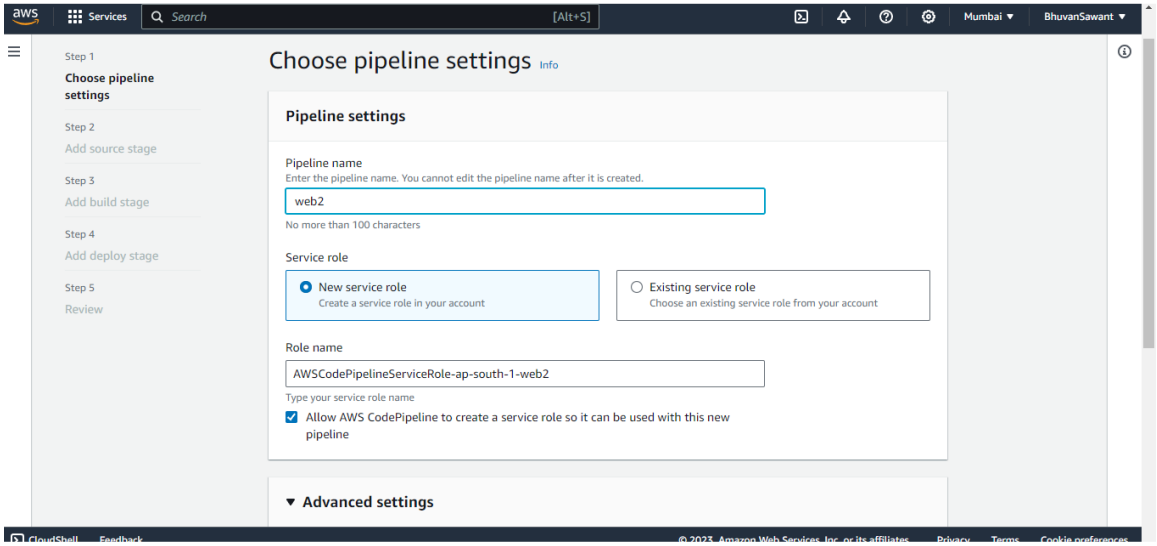
    - A pipeline is a series of stages that represents your release process. Each stage can contain one or more actions.

2. **Stage:**

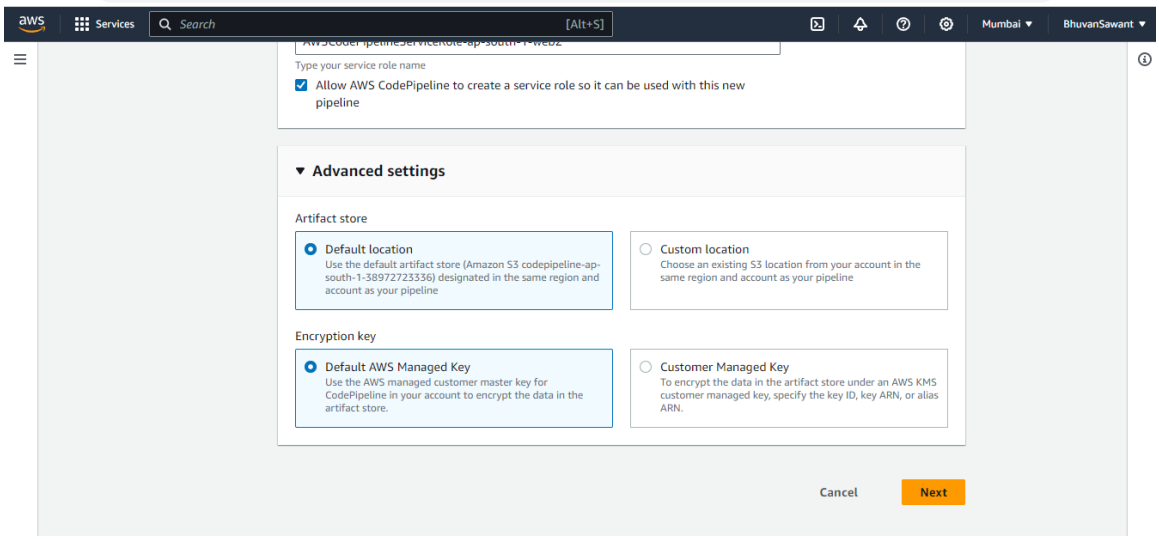
    - A stage is a logical unit in a pipeline, representing a phase in the release process. Stages are executed sequentially.

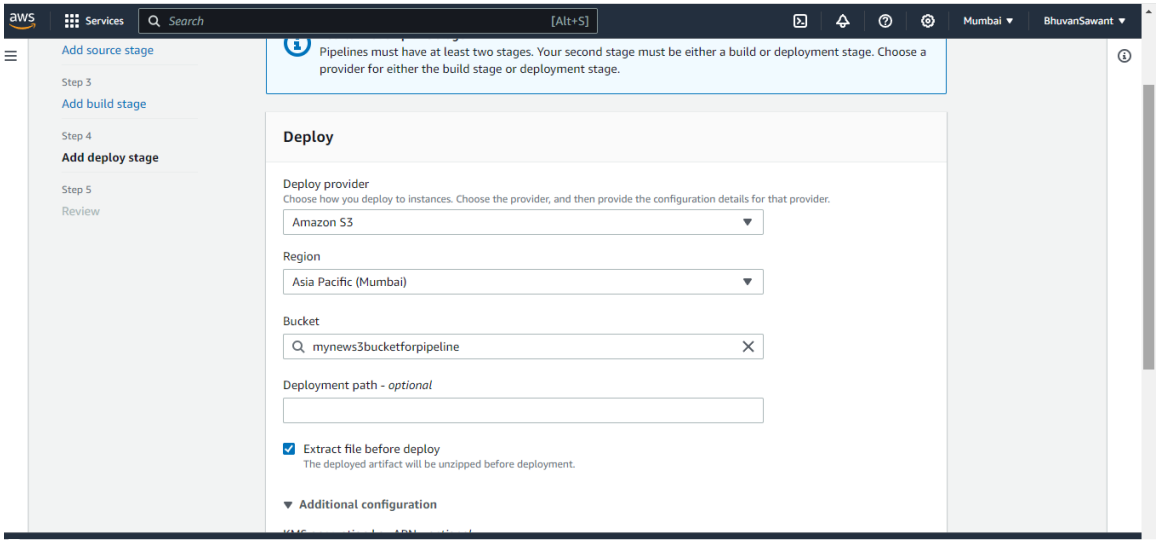
3. **Action:**
    - An action represents a task within a stage. Actions can include tasks such as building code, deploying to a test environment, or running tests.
4. **Artifact:**
    - Artifacts are the files and data that are produced as a result of an action. They are used to pass information between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

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

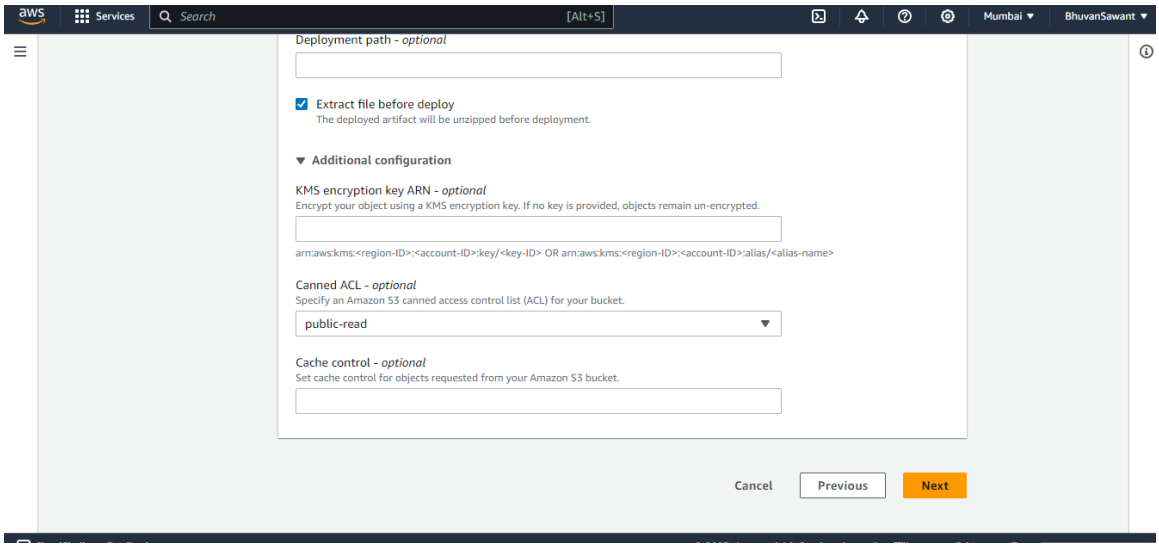
1. **Set Up Source Stage:**
    - Configure a source stage in AWS CodePipeline, linking to your version control system (e.g., CodeCommit, GitHub).
    - 

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

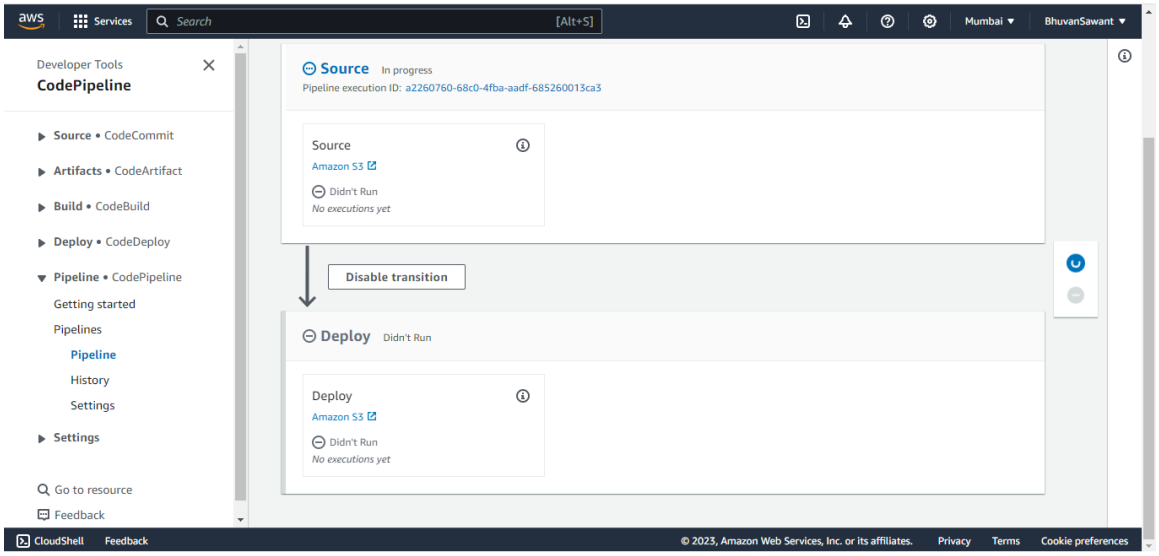
- 

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.

Bhuvan Sawant

110

ASSIGNMENT 5

Aim: To understand Kubernetes Cluster Architecture

Theory:

1. What are the various Kubernetes sevices running on nodes? Describe the role of each service.

In a Kubernetes cluster, several services and components run on each node to manage various aspects of container orchestration, networking, and communication. Here are some of the key services and their roles:

Kubelet: The Kubelet is an agent that runs on each node and communicates with the Kubernetes Control Plane (Master) to ensure that containers are running in a Pod. It starts and stops containers, monitors their health, and reports the current node's status to the control plane.

Kube Proxy: Kube Proxy is responsible for network connectivity and load balancing. It maintains network rules on the node and forwards traffic to the appropriate container. It helps implement services of type NodePort and LoadBalancer.

Container Runtime: This is not a Kubernetes-specific service, but it's a crucial component. It's the software responsible for running containers. Docker, containerd, and CRI-O are common container runtimes used with Kubernetes.

Node Controller: The Node Controller monitors the health of nodes and manages their lifecycle. If a node becomes unresponsive or fails, it is the Node Controller's responsibility to handle the situation.

Kube DNS: Kube DNS is a cluster DNS service that provides DNS for services within the cluster. It ensures that services and Pods can communicate with each other using their DNS names.

cAdvisor: Container Advisor (cAdvisor) is a tool that collects, aggregates, processes, and exports information about running containers. It is primarily used for monitoring and performance analysis.

Ingress Controller: An Ingress Controller manages external access to services in the cluster. It is responsible for routing external traffic to the appropriate services based on Ingress resources defined in the cluster.

Heapster/Metrics Server: Heapster was used for cluster-wide monitoring and resource usage analysis. In newer versions of Kubernetes, Heapster has been deprecated in favor of the Metrics Server, which provides resource usage metrics to various parts of Kubernetes.

Container Network Interface (CNI): The CNI is not a single service but a framework that allows different networking solutions to be used with Kubernetes. Services like Flannel, Calico, and Weave provide network plugins to implement container networking.

Kubelet Garbage Collection: This service helps clean up old and unused resources, such as unused images and containers, to reclaim disk space.

Node Problem Detector: This service detects and reports hardware and operating system problems on nodes, helping to improve cluster stability.

Kubelet Plug-ins: Kubelet supports dynamic configuration through plug-ins. These can be used for device plugins, resource management, and custom extensions to the Kubelet's functionality.

2. What is Pod Disruption Budget?

A Pod Disruption Budget (PDB) is a Kubernetes resource that allows you to define policies for how disruptions to your Pods can be controlled during events like voluntary evictions (e.g., due to scaling down) or involuntary disruptions (e.g., node failures). PDBs provide a way to ensure the availability of your applications during maintenance or scaling operations while also allowing for safe disruptions.

Here are some key aspects of Pod Disruption Budgets:

Availability Control: PDBs allow you to set constraints on how many Pods of a particular application can be disrupted simultaneously. This helps maintain a minimum level of availability for your application during various operational procedures, such as rolling updates.

Selector-Based Policy: PDBs are associated with one or more Pods through label selectors. You can specify which Pods the PDB applies to by selecting them based on labels.

MaxUnavailable: The most common constraint specified in a PDB is maxUnavailable, which defines the maximum number or percentage of Pods that are allowed to be unavailable during disruptions. For example, you might specify maxUnavailable: 1 to ensure that at least one instance of your application is available at all times.

MinAvailable: Alternatively, you can use minAvailable to specify the minimum number or percentage of Pods that must remain available. This can be useful in scenarios where you want to ensure a certain level of redundancy.

Scope: PDBs can be scoped at the namespace level, allowing you to set different disruption budgets for different applications or components within a cluster.

3. What is the role of Load Balance in Kubernetes?

In Kubernetes, load balancing plays a crucial role in distributing network traffic across multiple Pods or replicas of an application to ensure high availability, optimal resource

utilization, and the efficient use of the cluster's resources. The primary role of load balancing in Kubernetes is to:

Ensure High Availability: Load balancing helps maintain the availability of your applications by distributing incoming traffic across multiple instances of your application (Pods) running on different nodes. If one of the Pods or nodes fails, the load balancer can route traffic to healthy instances.

Optimize Resource Utilization: By evenly distributing traffic, load balancing helps prevent overloading of specific Pods or nodes. This results in better resource utilization, improved performance, and prevents any single point of failure.

Scaling and Horizontal Pod Autoscaling: Load balancing works seamlessly with scaling mechanisms like Horizontal Pod Autoscaling (HPA). When your application experiences increased traffic, HPA can dynamically create more replicas of your Pods, and the load balancer will automatically start routing traffic to the new instances.

Service Discovery: Load balancers often integrate with DNS, making it easier for clients to discover services. Kubernetes provides a built-in DNS service that resolves service names to their associated Pod IP addresses.

Traffic Management: Kubernetes offers different types of services, such as ClusterIP, NodePort, and LoadBalancer services. LoadBalancer services expose applications to external traffic, and the cloud provider's load balancer, in this case, performs the load balancing.

Ingress: Ingress controllers provide a way to manage external access to services within the cluster. They often include load balancing capabilities to distribute incoming traffic to different services based on hostnames or paths.

Session Affinity: Some load balancers can be configured to use session affinity (sticky sessions), which ensures that a client's requests are consistently directed to the same backend Pod. This can be useful for stateful applications.

Security: Load balancers can also enhance security by providing features like SSL termination, which allows them to handle encryption and decryption of traffic.

Global Load Balancing: In multi-region or multi-cloud setups, global load balancing can distribute traffic across different clusters or data centers, ensuring low-latency access for users.

Conclusion: In this assignment we understood the Kubernetes Cluster Architecture.

Assignment 6

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

Lab Outcome:
LO1, LO6 mapped

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.

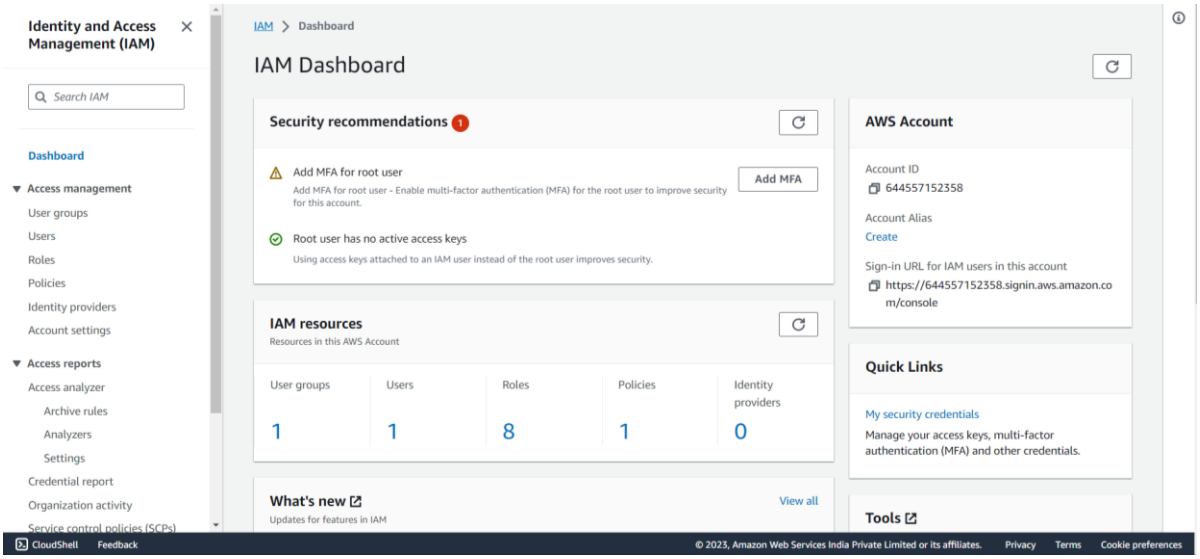
Modules: These are containers for multiple resources that are used together. They provide a way to group resources, and they can be used to create reusable infrastructure components.

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.

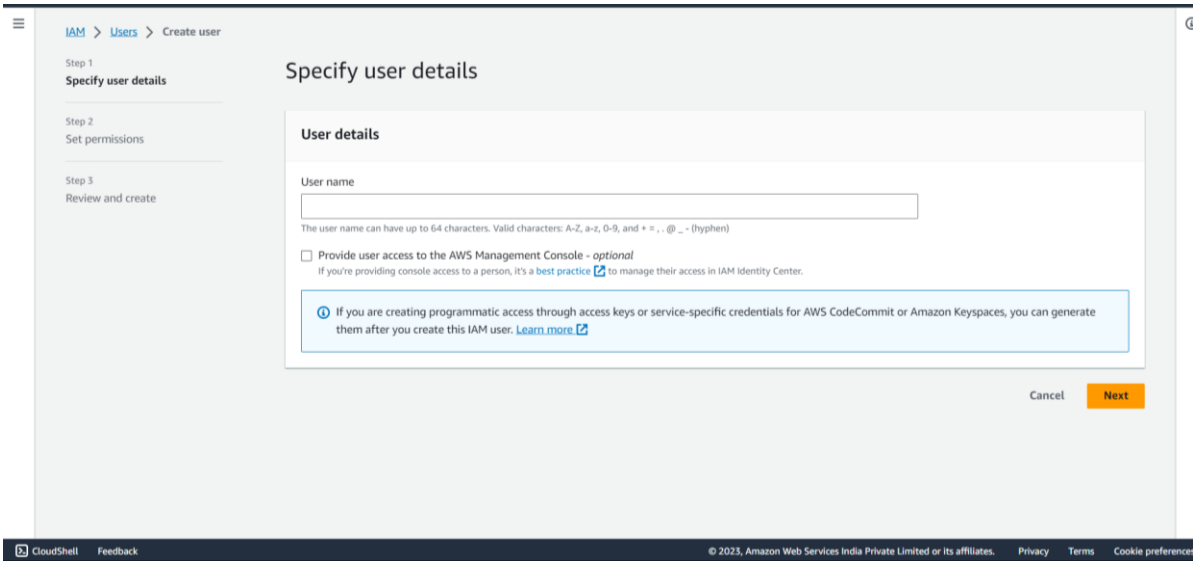
Provisioners: While not always recommended due to their imperative nature, provisioners can be used to model specific actions on the local machine or on a remote machine, like executing scripts.

Steps :

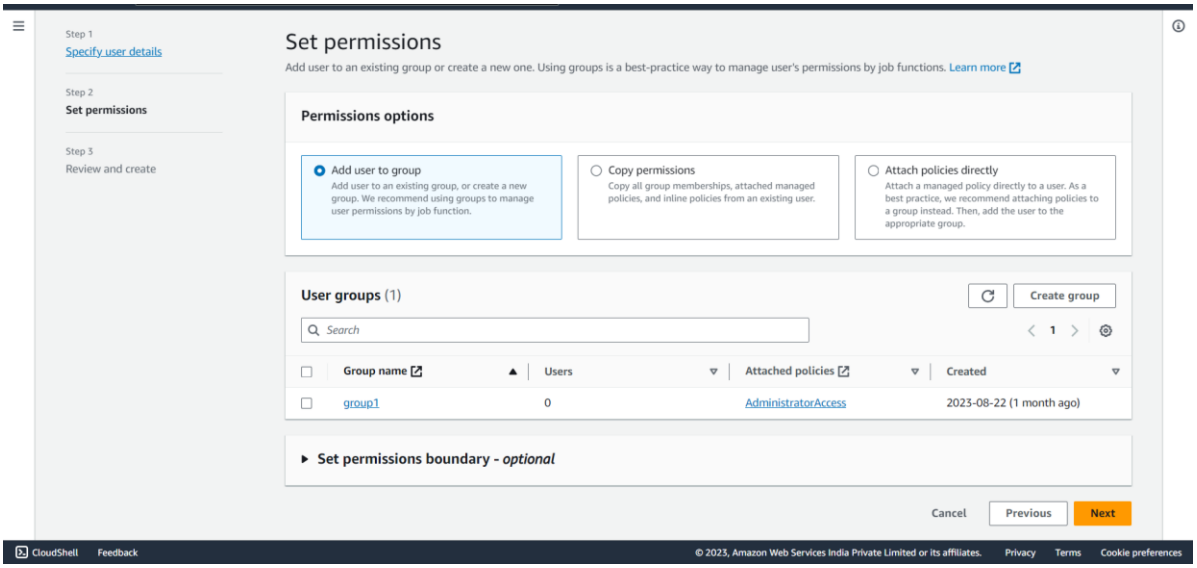
1. Login and Search IAM on AWS Console and select the first option.



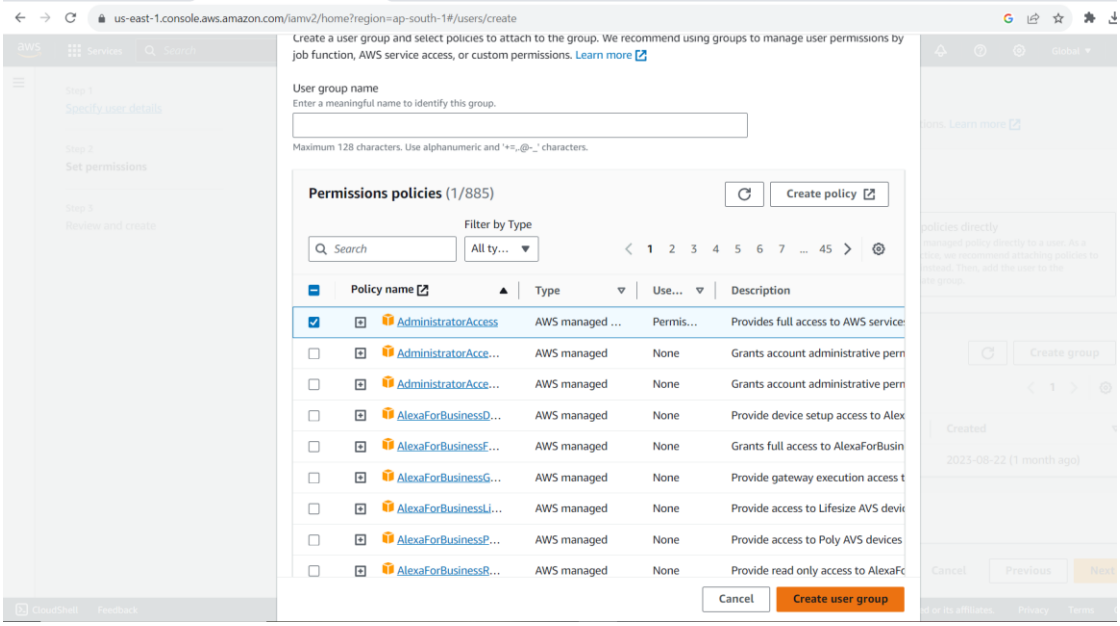
2. Head to users and Click on Create User



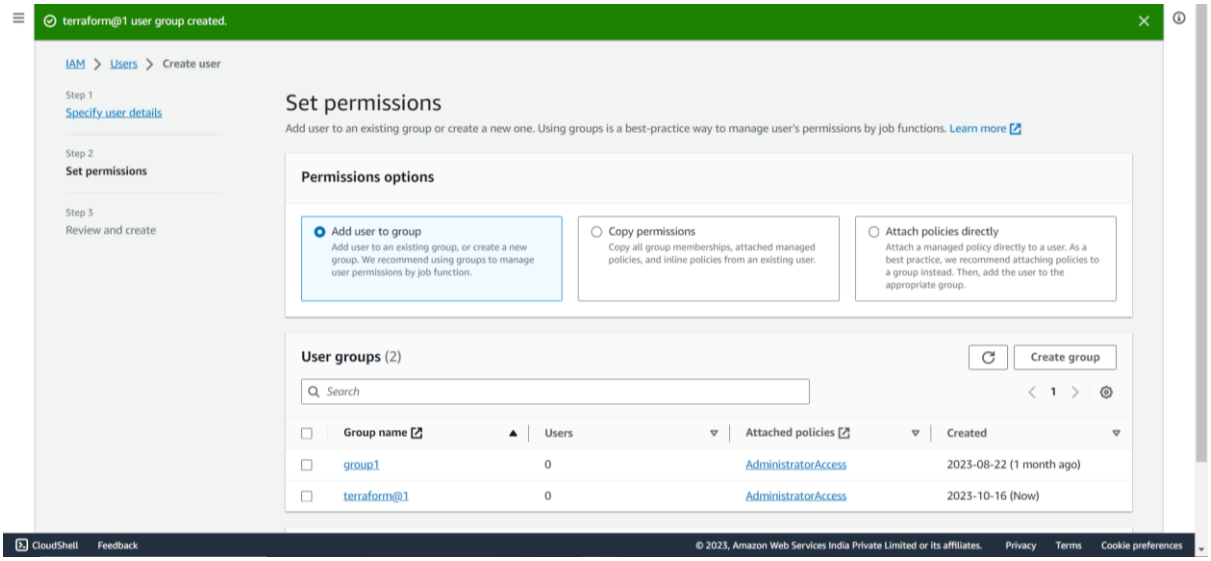
3. Enter a new username and click on next
4. Click on create group



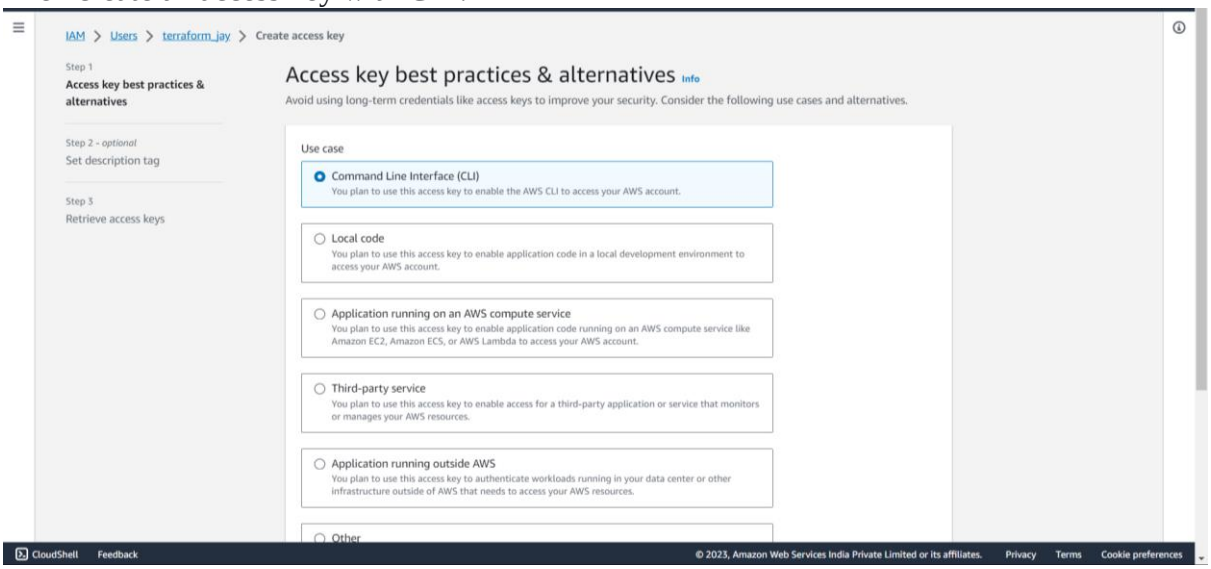
5. Select the first Policy will full access and enter a group name and create the group.



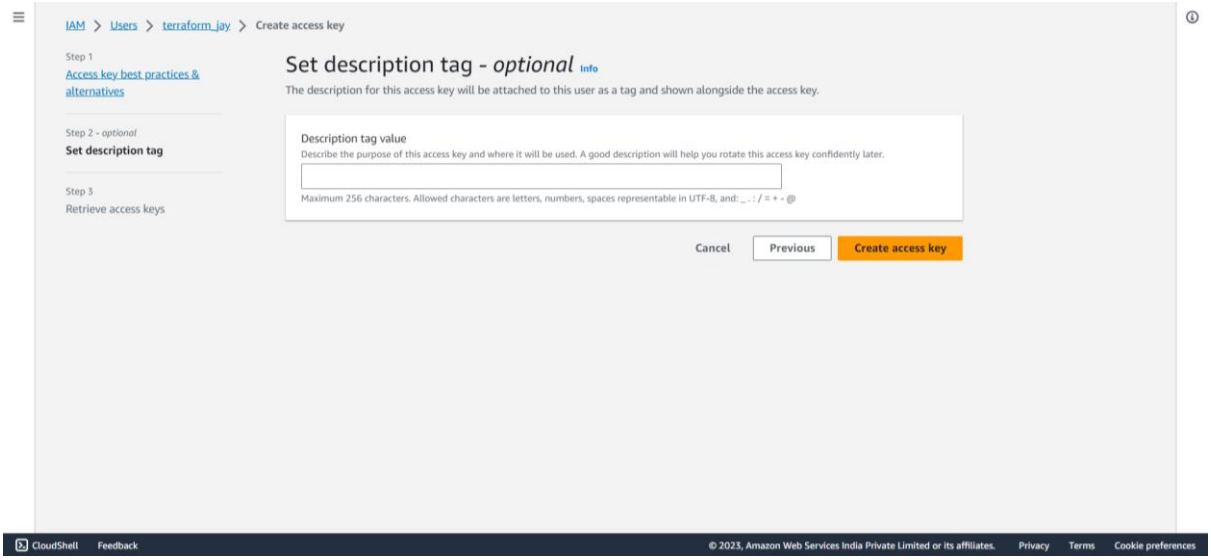
6. Then click on next, select the policy and create user.



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
```

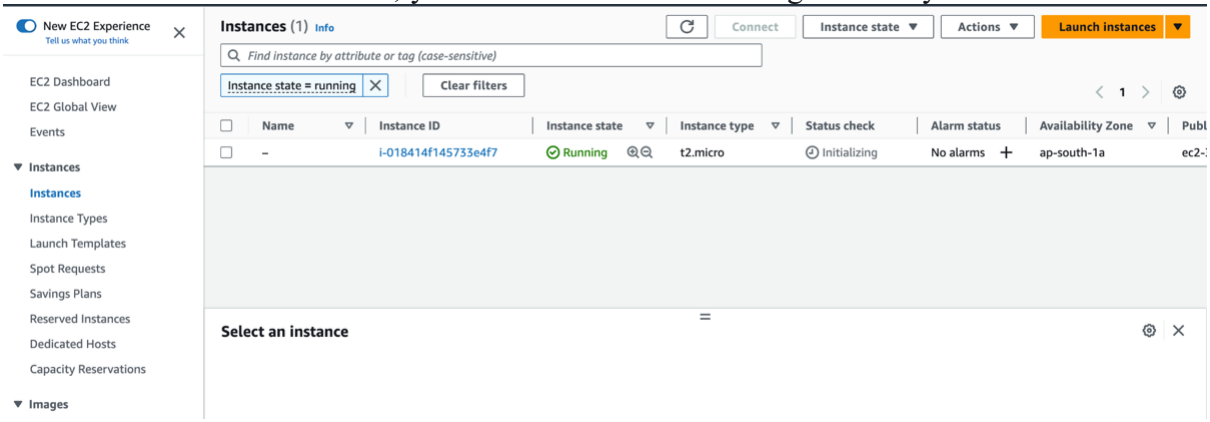
10. Create the terraform config file main.tf with required configurations.

```
❯ 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
```

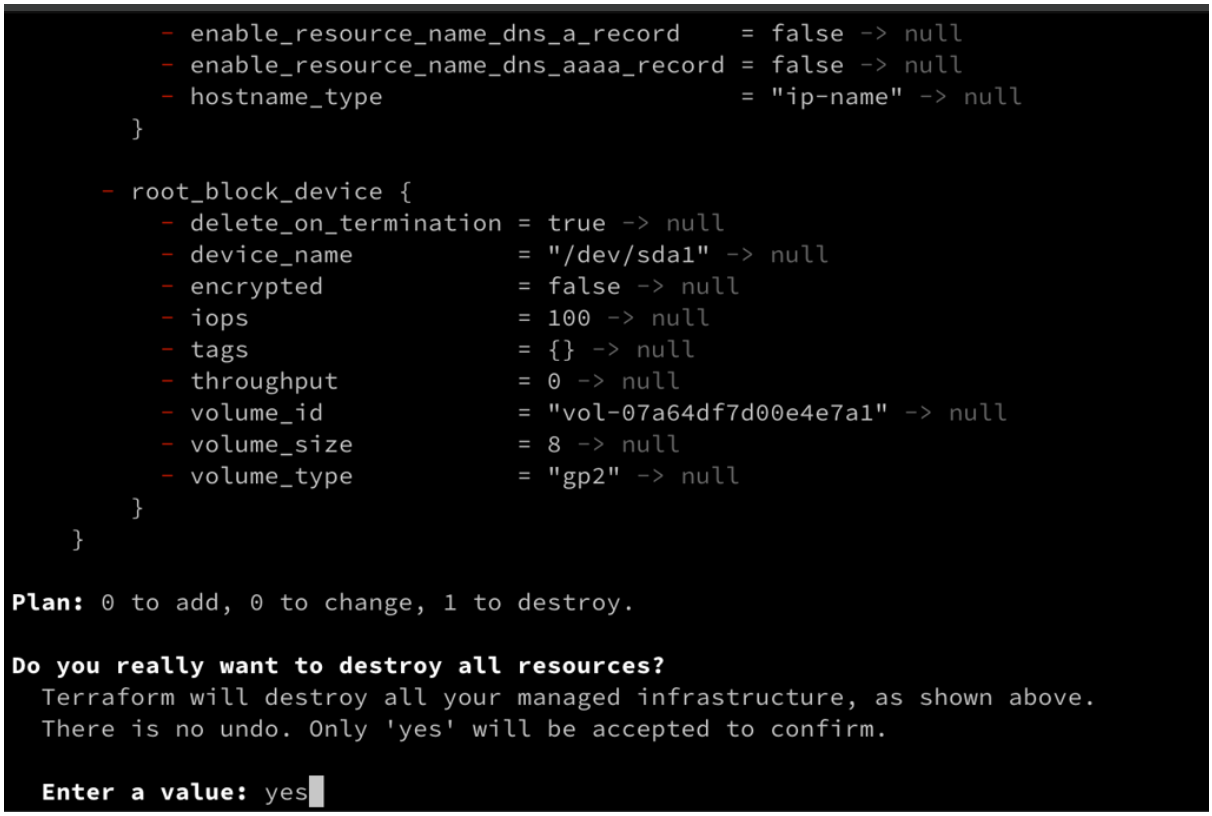
11. Run the command terraform plan and enter yes

```
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.



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.

Bhuvan Sawant T22 Roll No: 110

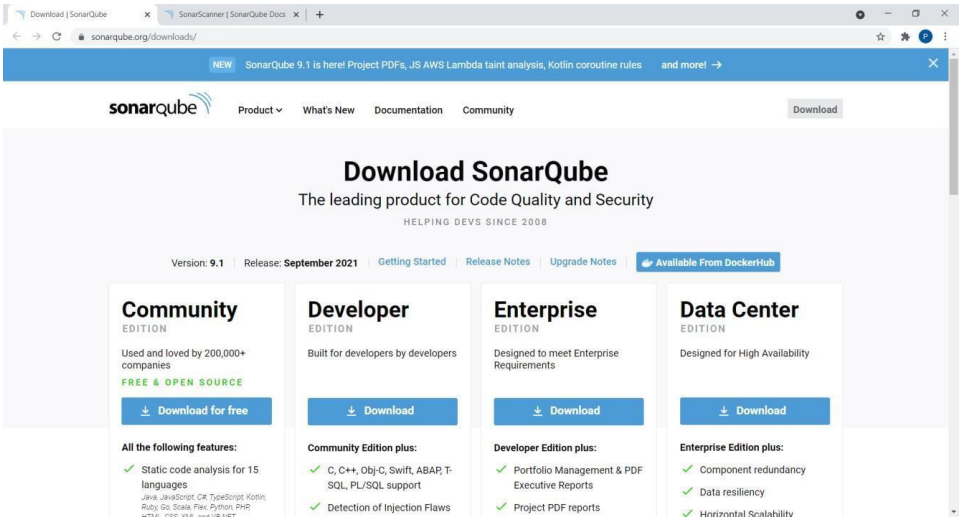
AdvDevOps Assignment 7

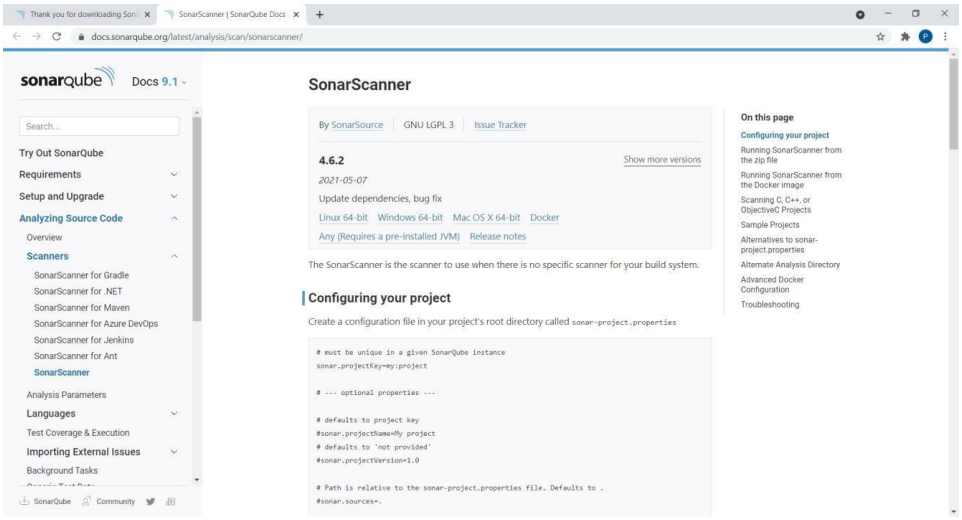
Aim: To perform static analysis on python programs using SonarCube SAST processes

LO Mapped: LO4

Theory:

Download SonarQube and Sonar Scanner

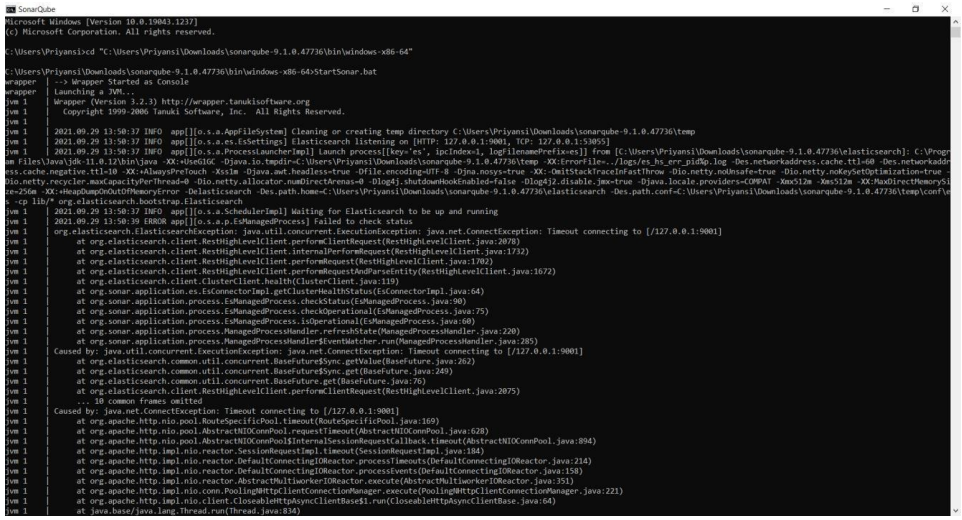


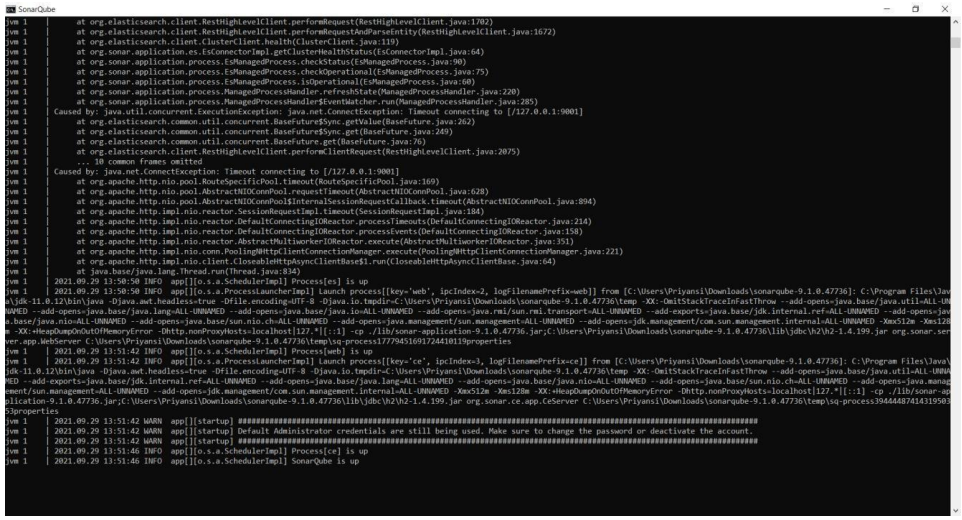


After downloading, set Environment Variables. Add “sonarqube-9.1.0.47736\bin” to Path.

Open command prompt. Run commands:

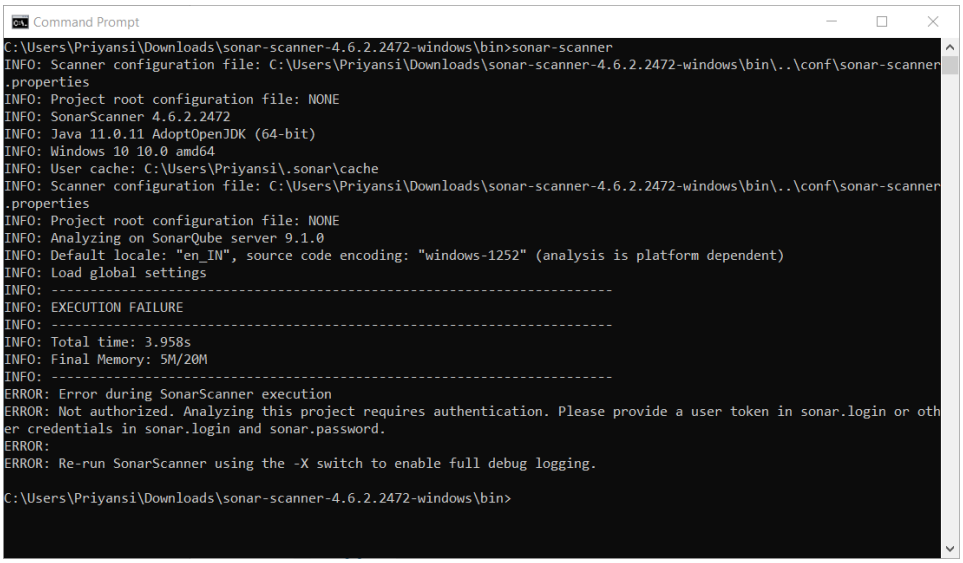
- cd “sonarqube-9.1.0.47736\bin\windows-x86-64”
- StartSonar.bat



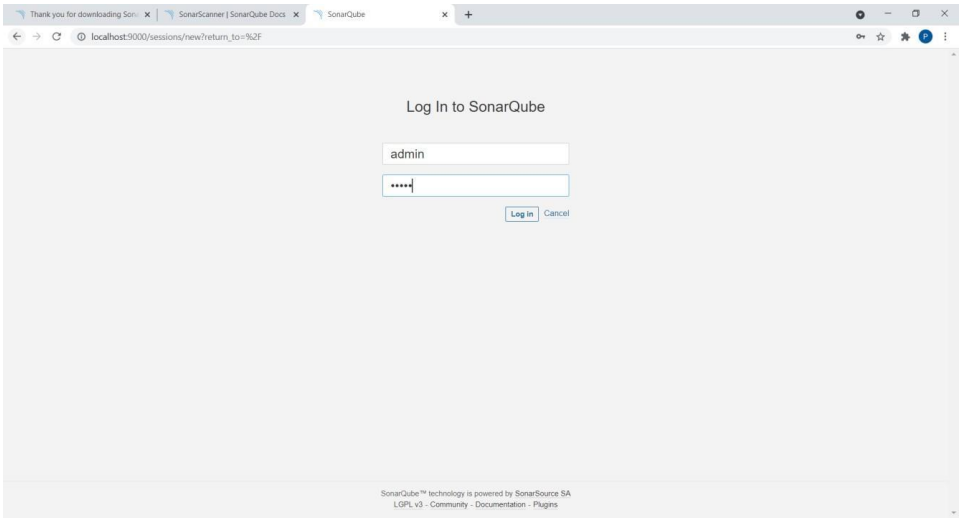


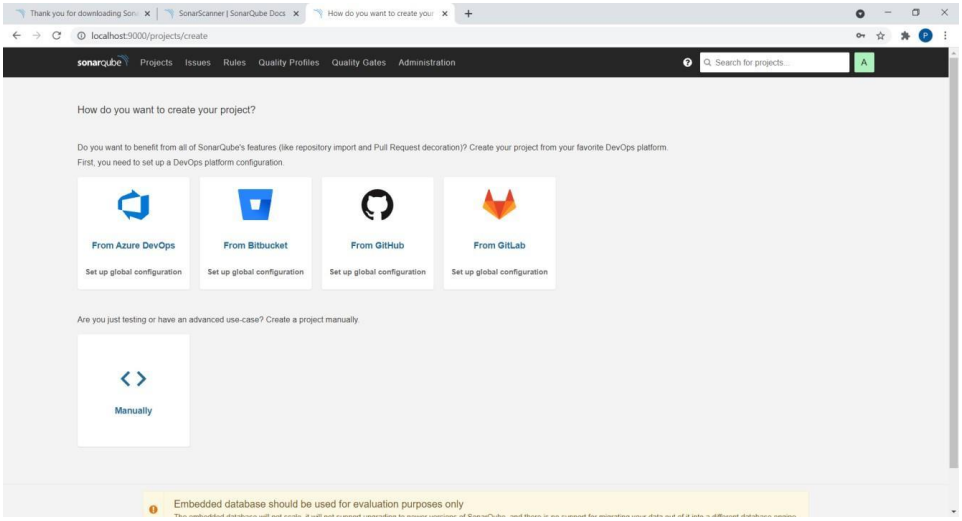
Open another command prompt. Run command:

- cd “sonar-scanner-4.6.2.2472-windows\bin”
- sonar-scanner

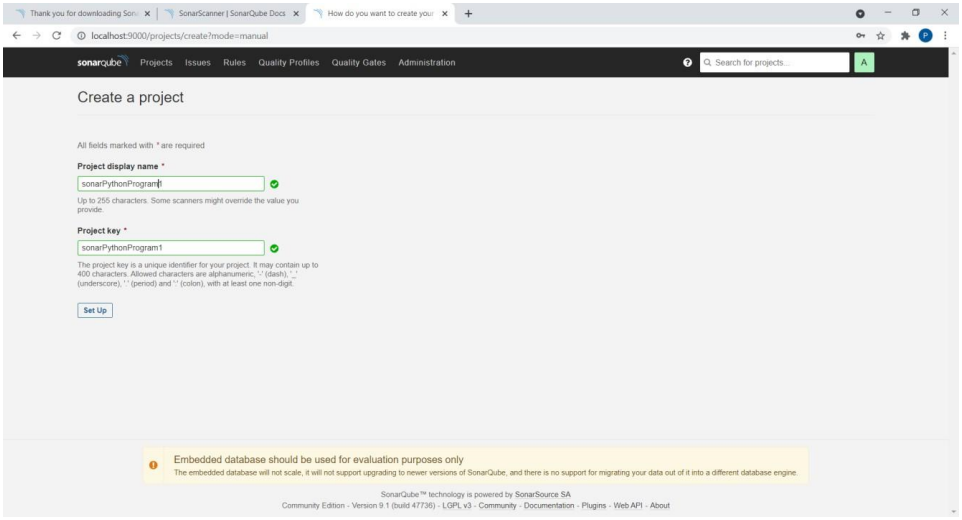


Server up and running on **localhost:9000**
Login using credentials as User: admin and Password: admin and Set a new password

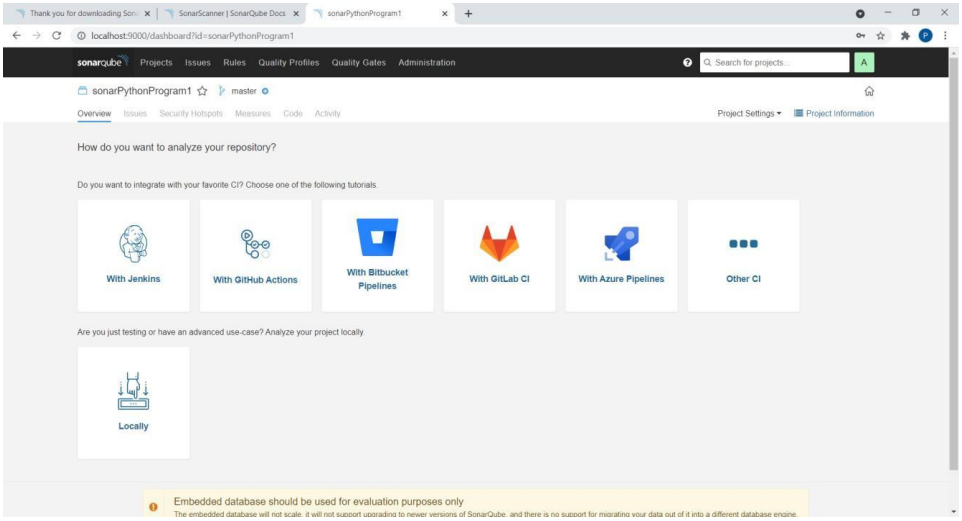




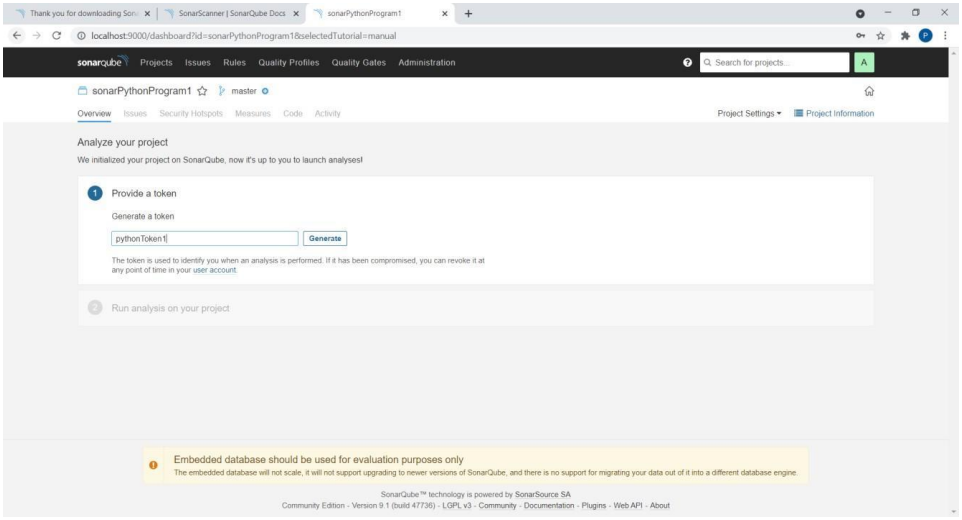
Click on Create a project **Manually.**



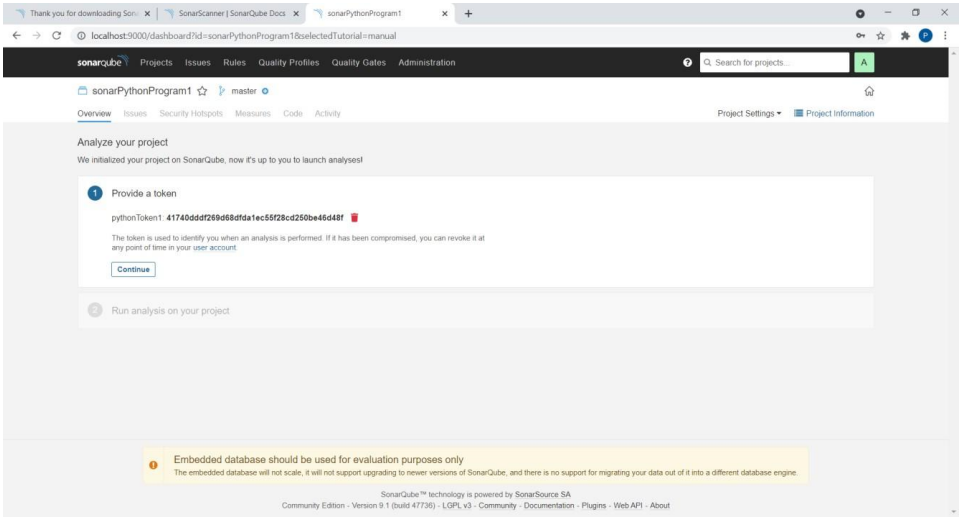
Give any Project display name.



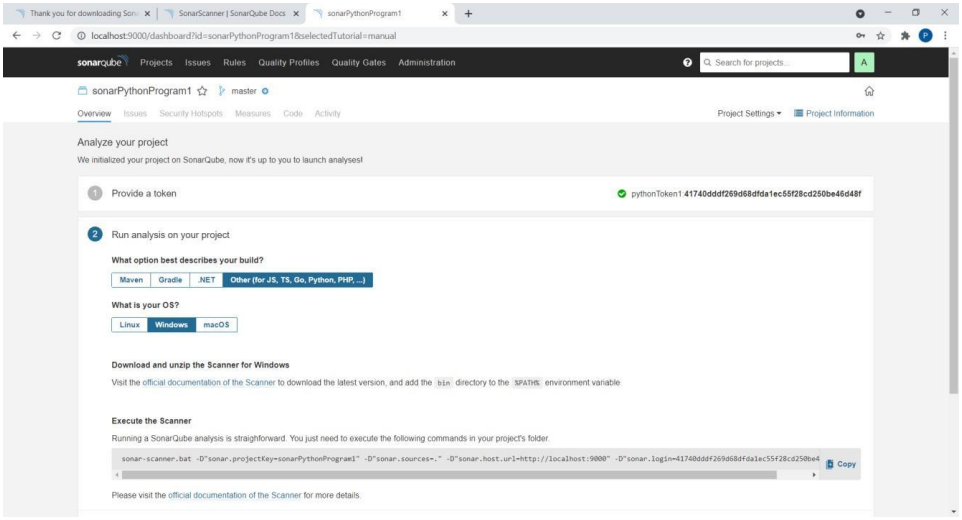
Click on **Locally.**



Give any name to token and click on **Generate.**



Click on **Continue.**

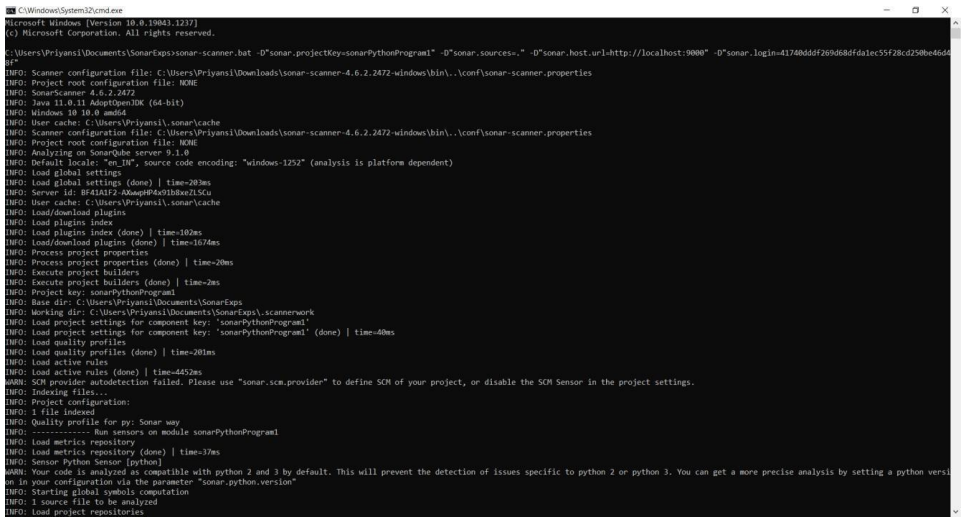


Save a Python program in a folder.

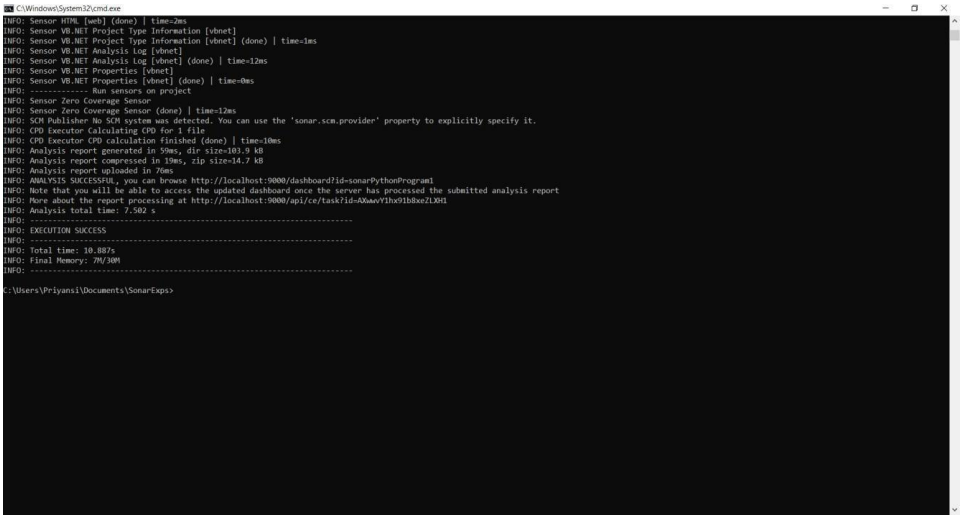
```
class Solution(object):
   def romanToInt(self, s):
```

```
    roman =
{'I':1,'V':5,'X':10,'L':50,'C':100,'D':500,'M':1000,'IV':4,'IX':9,'XL':40,'XC':90,'CD':400,'CM':900}
    i = 0
    num = " "
    while i < len(s):
      if i+1<len(s) and s[i:i+2] in roman:
        num+=roman[s[i:i+2]]
        i+=2
      else:
        #print(i)
        num+=roman[s[i]]
        i+=1
    return num
ob1 = Solution()
print(ob1.romanToInt("III"))
print(ob1.romanToInt("CDXLIII")
```

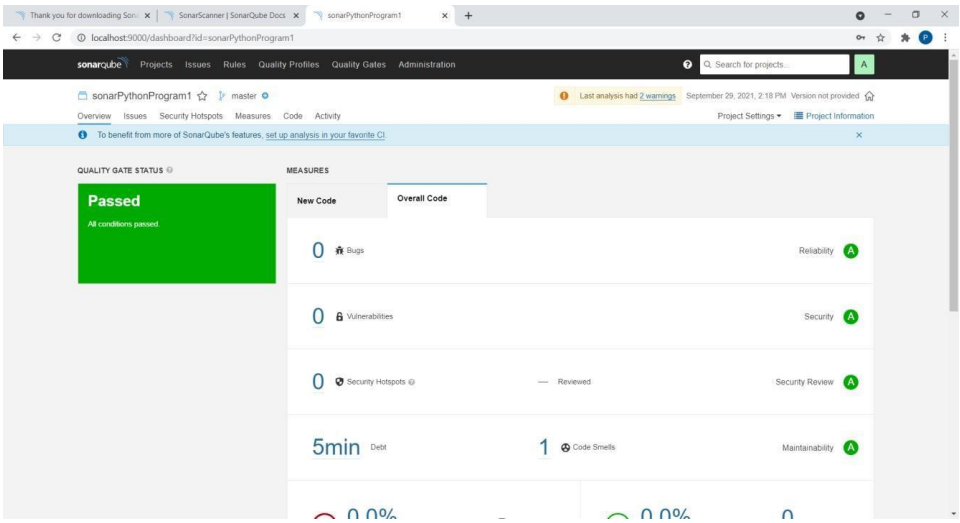
Open command prompt in this folder and Run program using copied command.
sonar-scanner.bat -D"sonar.projectKey=<YourDisplayName>" -
D"sonar.sources=." -D"sonar.host.url=http://localhost:9000" -
D"sonar.login=<YourTokenGeneratedID>"

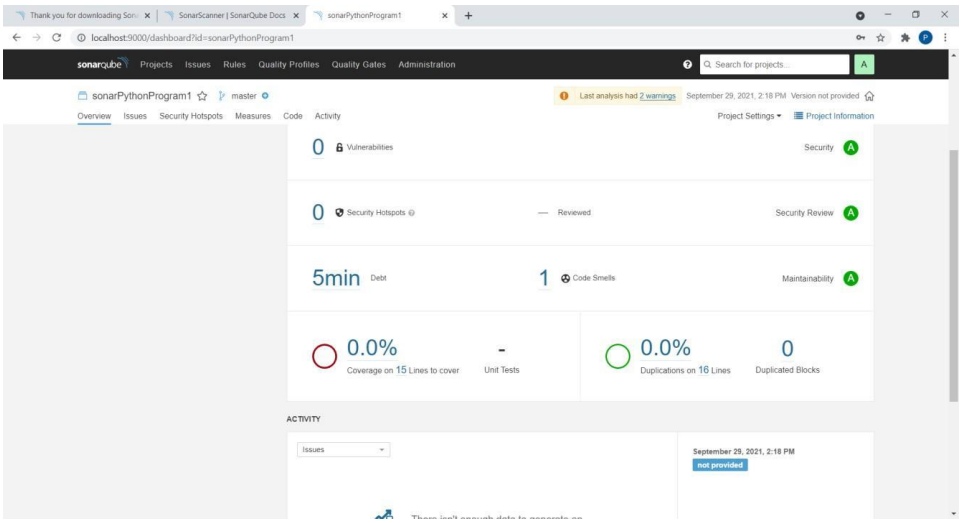


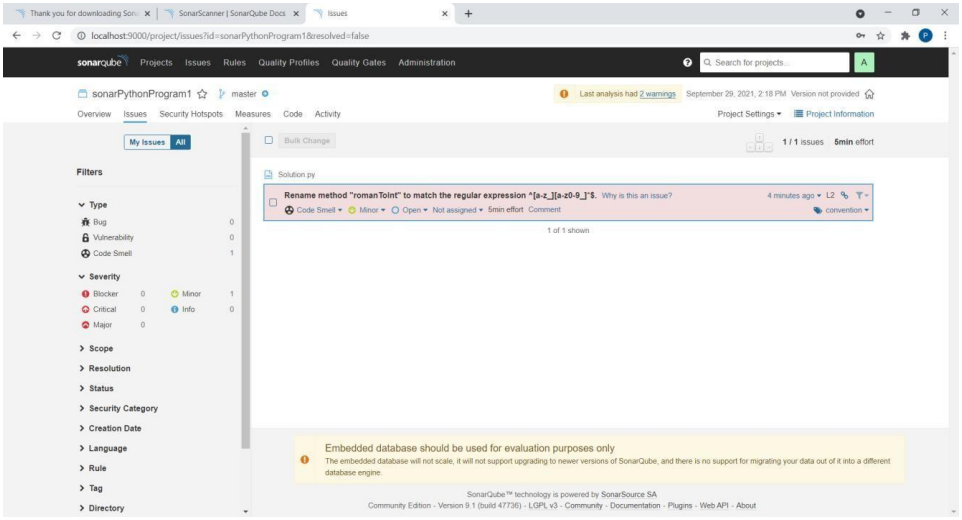
Commented [DS1]: sonar-scanner.bat -D"sonar.projectKey=<YourDisplayName>" -D"sonar.sources=." -D"sonar.host.url=http://localhost:9000" -D"sonar.login=<YourTokenGeneratedID>" Page 7 command



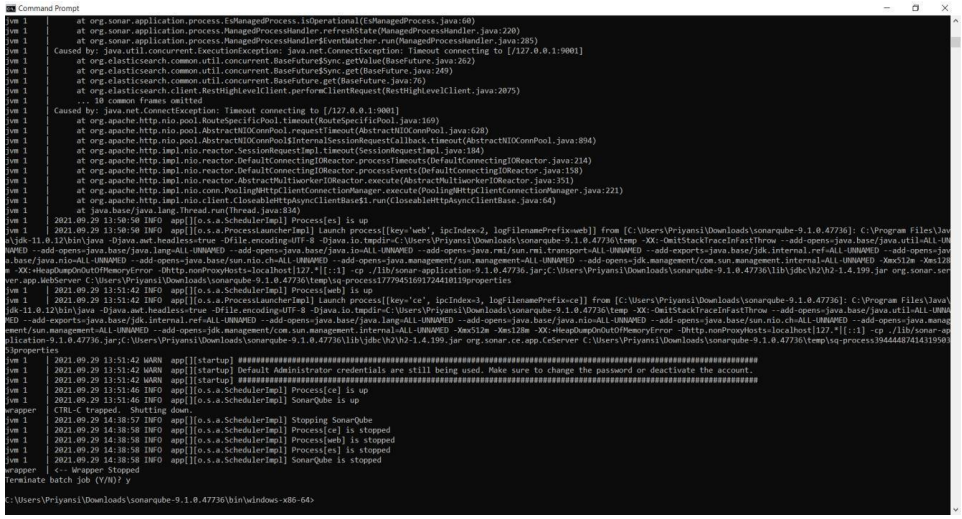
Given below is the inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities.







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: Bhuvan Sawant Roll No.- 110 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

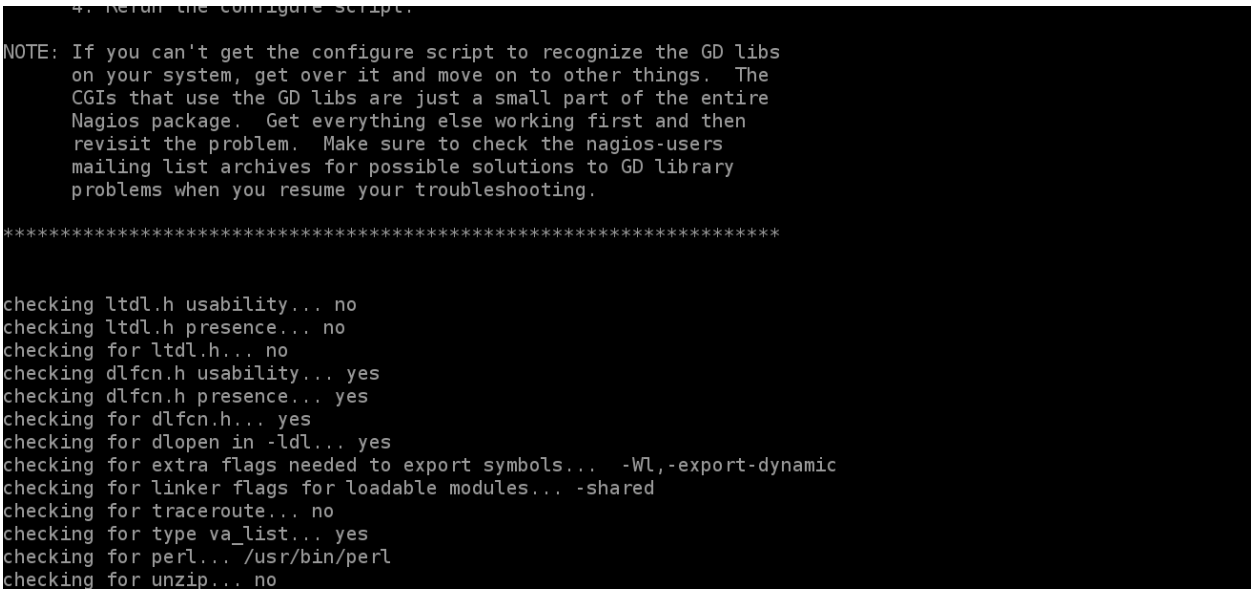
sudo ./configure --with-httpd-conf=/etc/apache2/sites-enabled

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

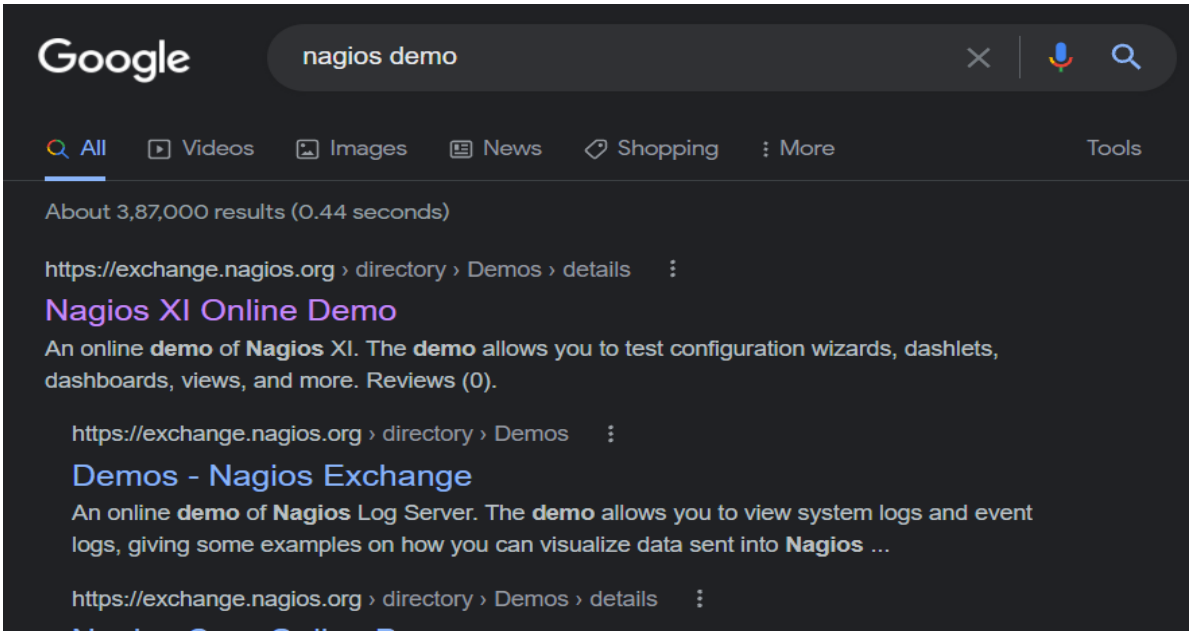
sudo systemctl status nagios

```
* nagios.service - Nagios Core 4.4.2
   Loaded: loaded (/lib/systemd/system/nagios.service; enabled; vendor preset: enabl
   Active: active (running) since Fri 2018-11-16 14:54:21 PST; 1s ago
     Docs: https://www.nagios.org/documentation
  Process: 18294 ExecStopPost=/bin/rm -f /usr/local/nagios/var/rw/nagios.cmd (code=e
  Process: 18293 ExecStop=/bin/kill -s TERM ${MAINPID} (code=exited, status=0/SUCCES
  Process: 18315 ExecStart=/usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nag
  Process: 18313 ExecStartPre=/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/
 Main PID: 18325 (nagios)
    Tasks: 6 (limit: 2319)
   CGroup: /system.slice/nagios.service
```

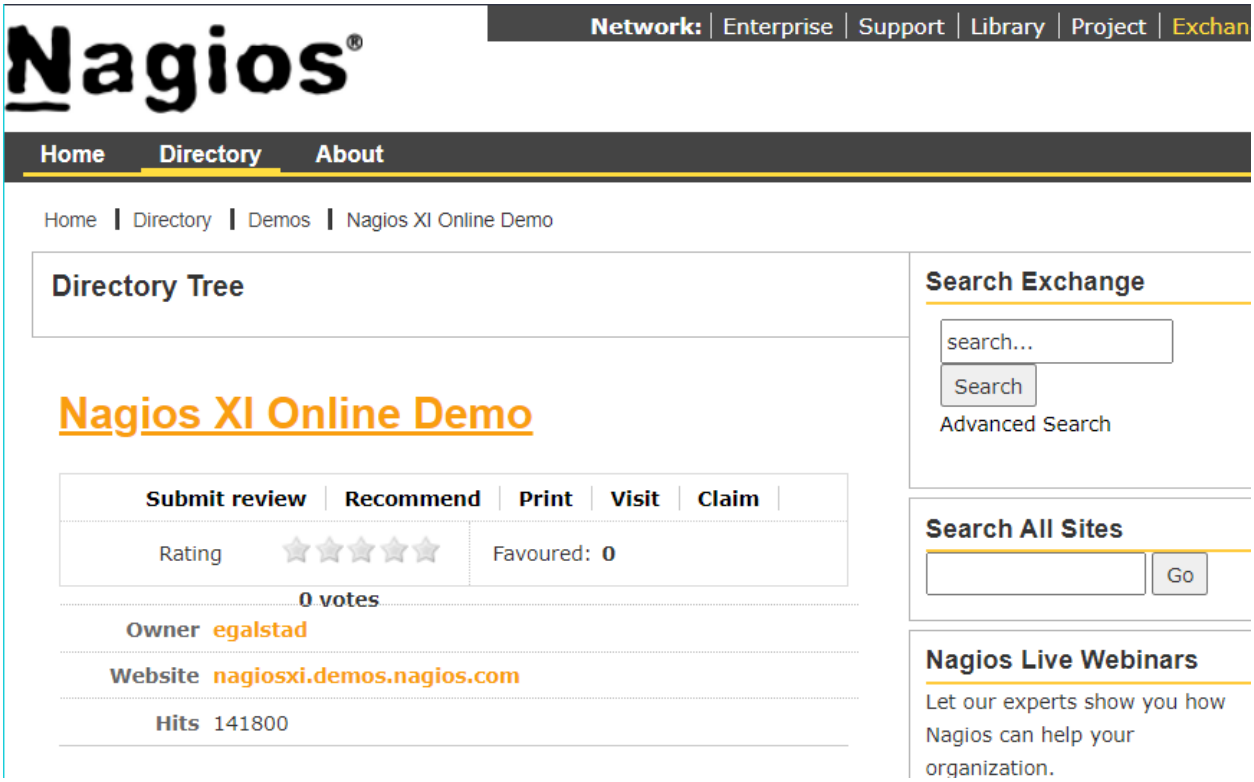
Steps-

Go to google.com, Search NagiOs Demo

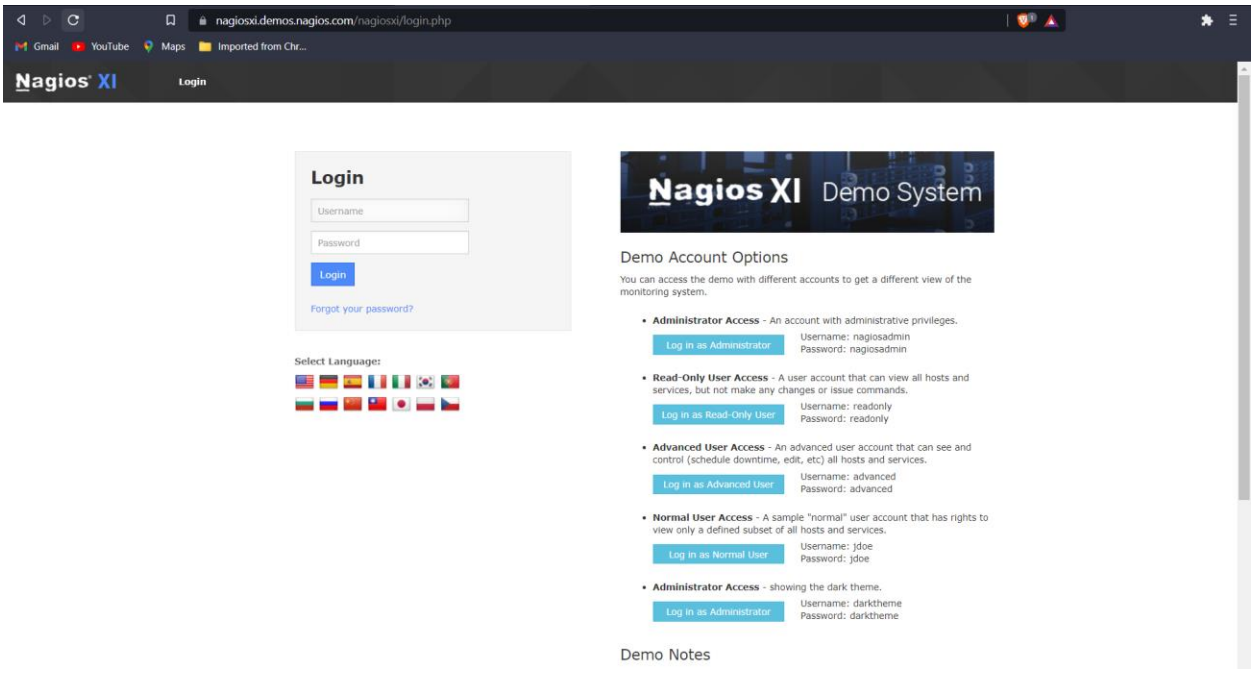
Click on the first link shown below



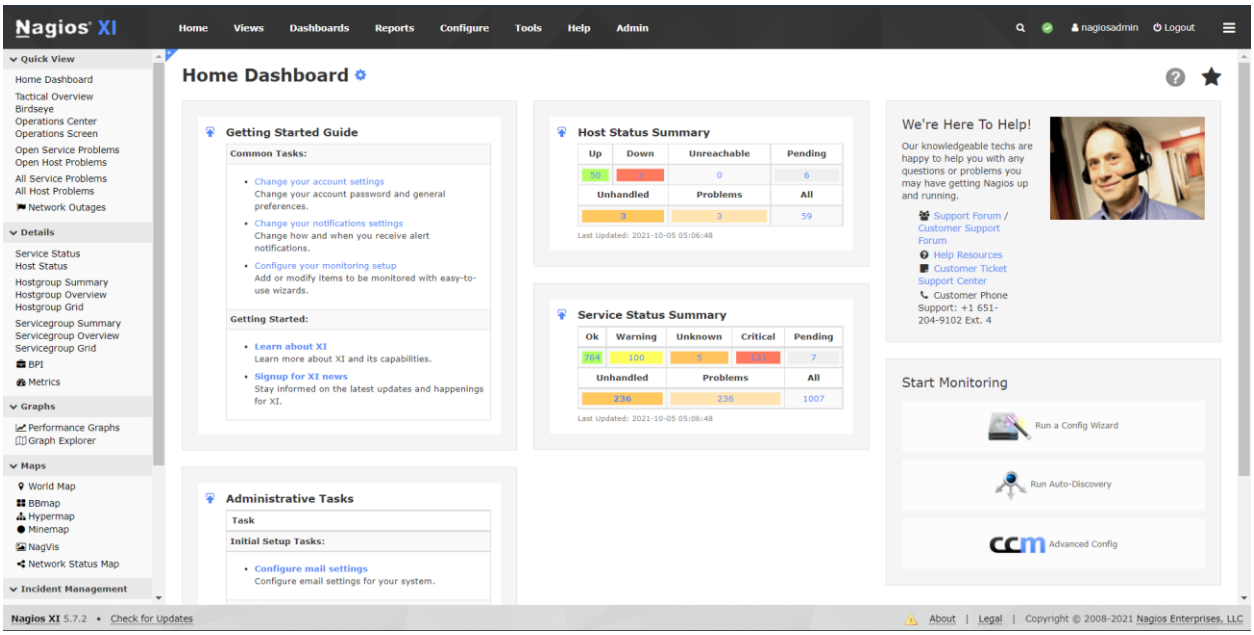
Now click on the website-



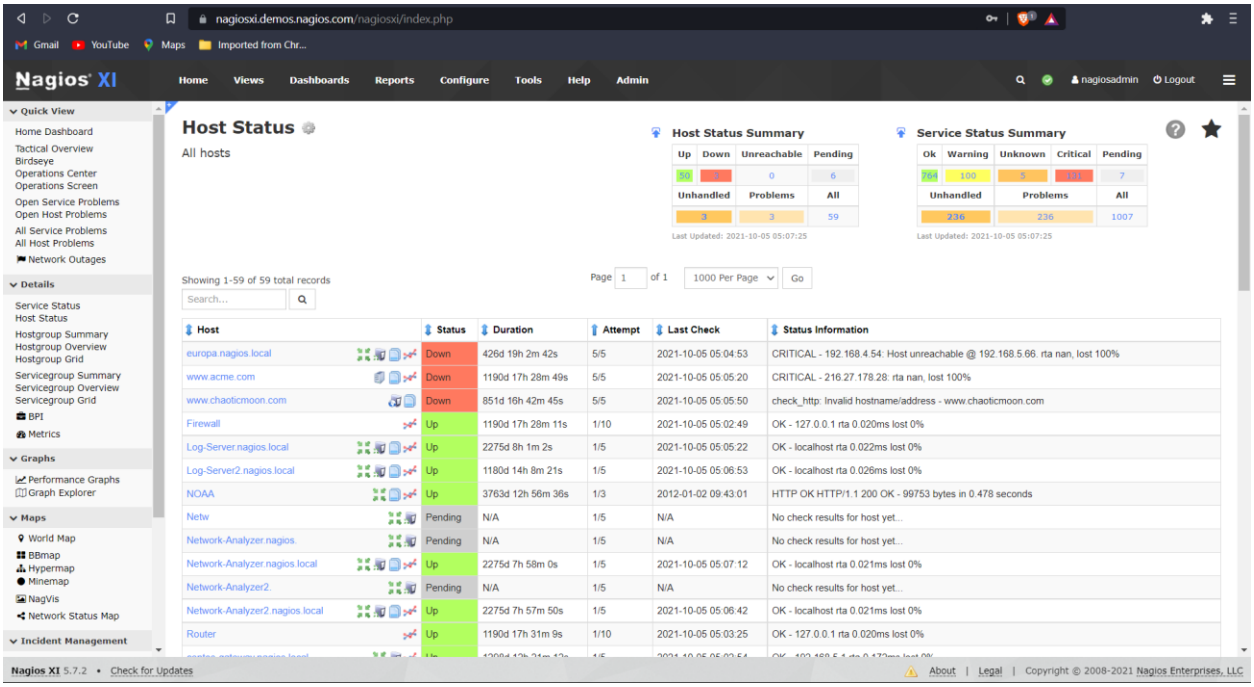
Now click on login as administrator



It will have interface like this

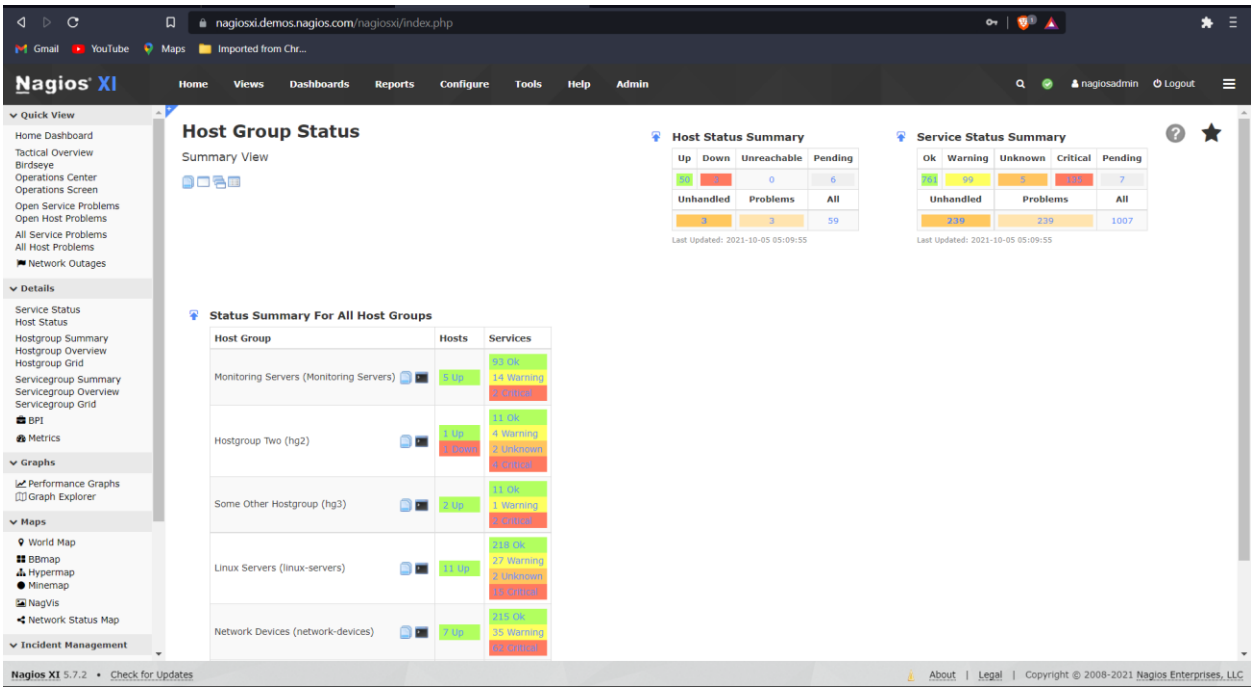


Now click on Host status-



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

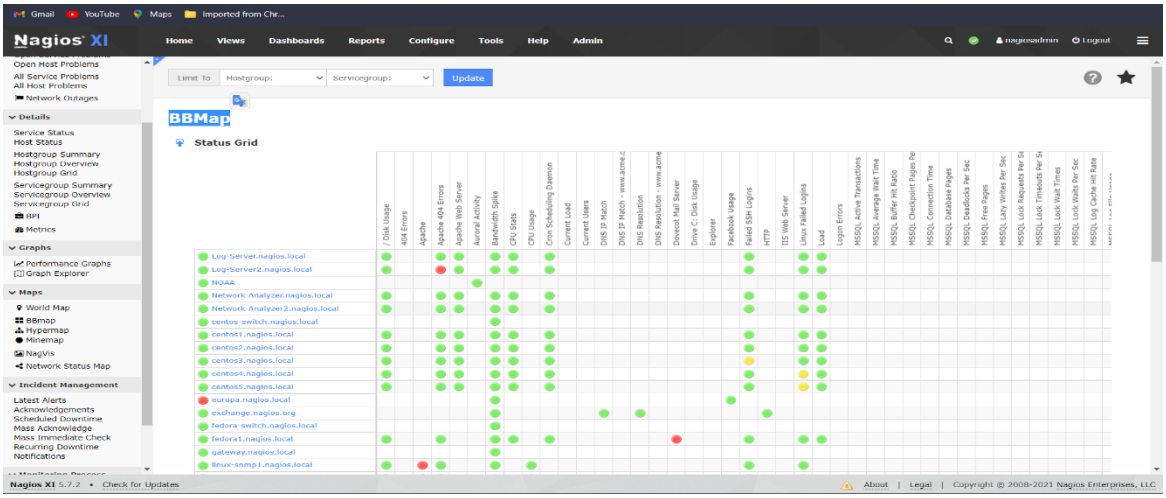
Now click on Host Group Status.



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.

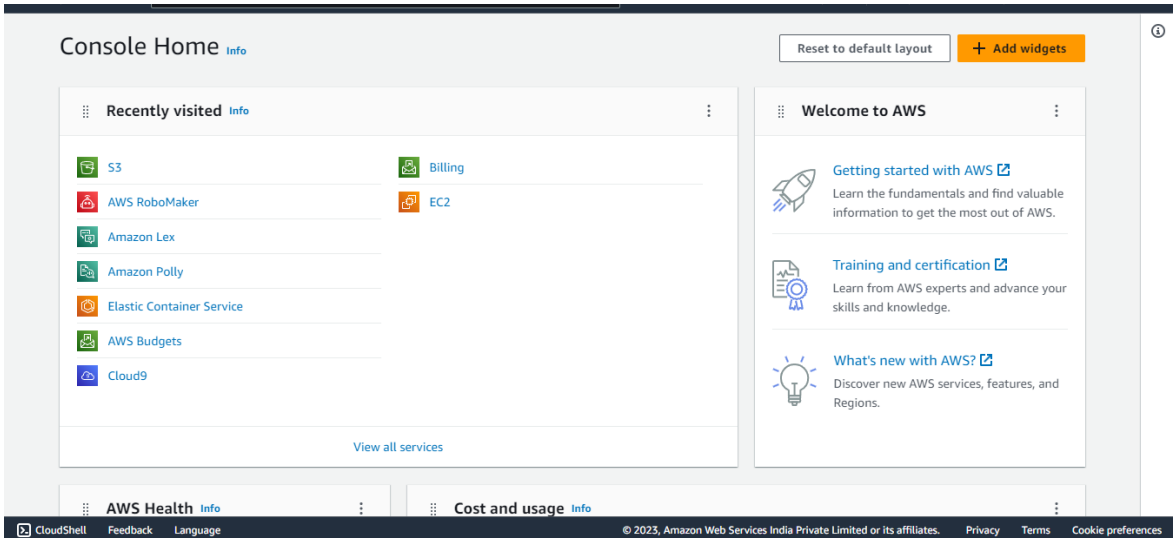
LO MAPPED: LO1, LO5

THEORY:

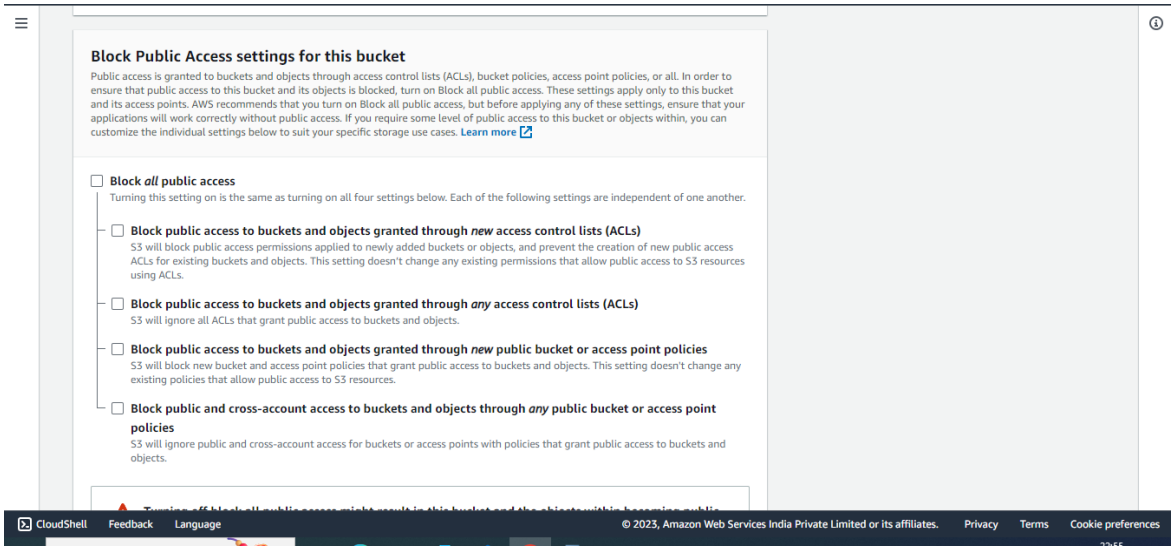
You can use Lambda to process event notifications from Amazon Simple Storage Service. Amazon S3 can send an event to a Lambda function when an object is created or deleted. You configure notification settings on a bucket, and grant Amazon S3 permission to invoke a function on the function's resource-based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

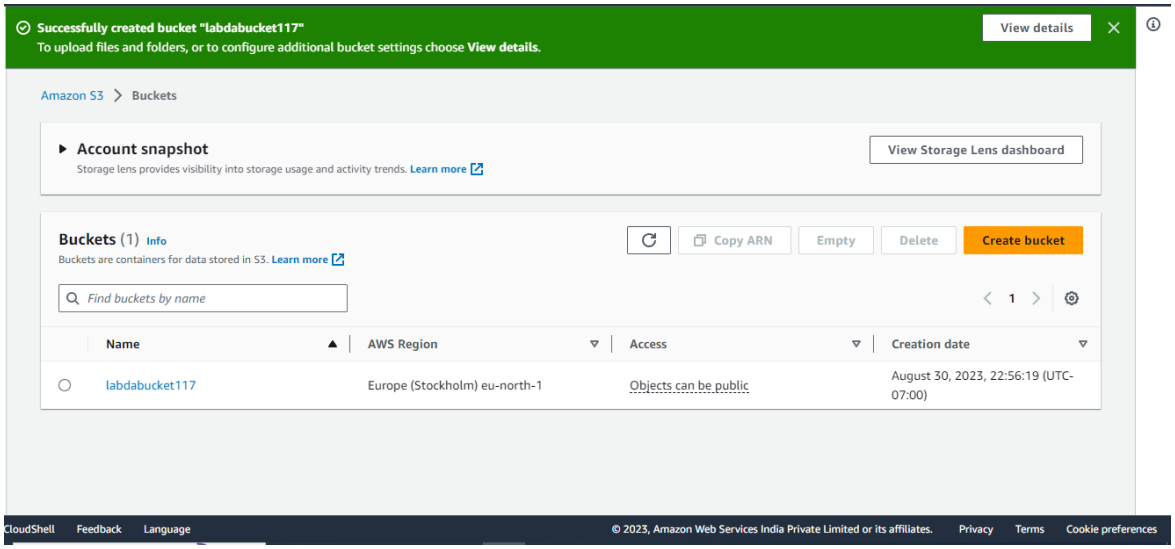
STEPS TO FOLLOW:

1. Log in as IAM User

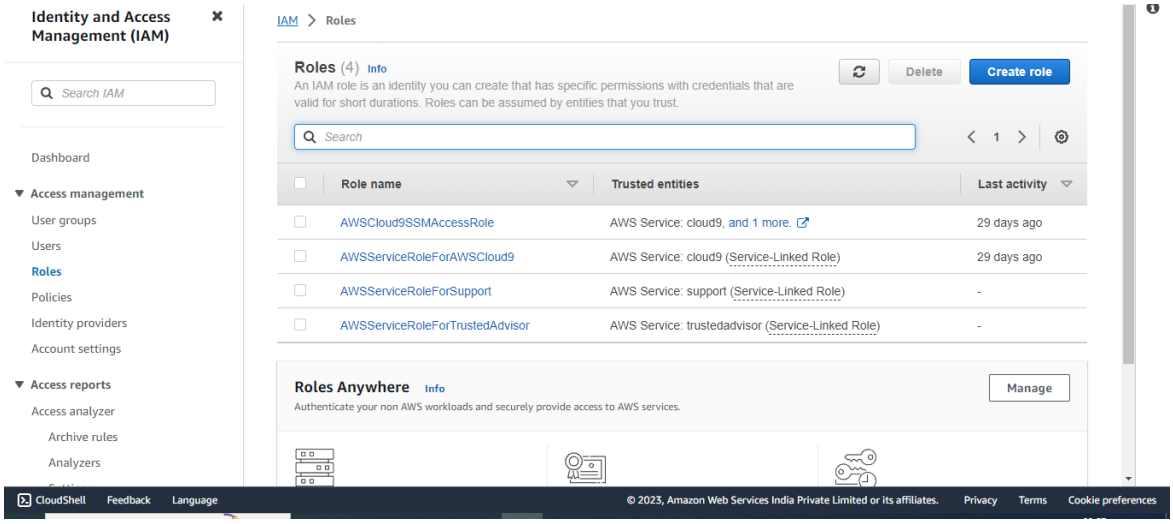


2. Create a S3 bucket and Enable the "Block all public access"

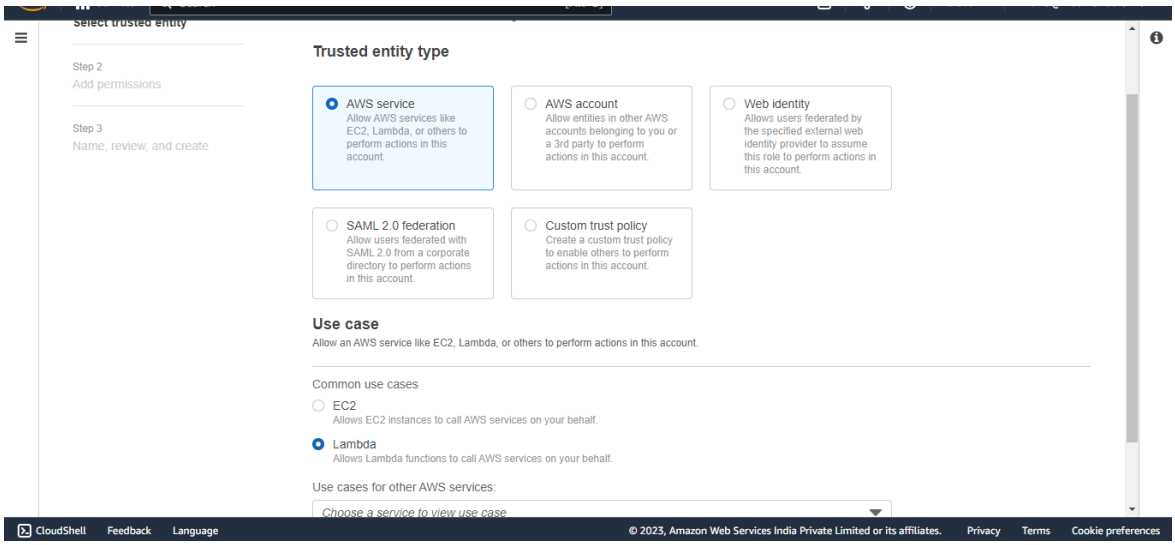




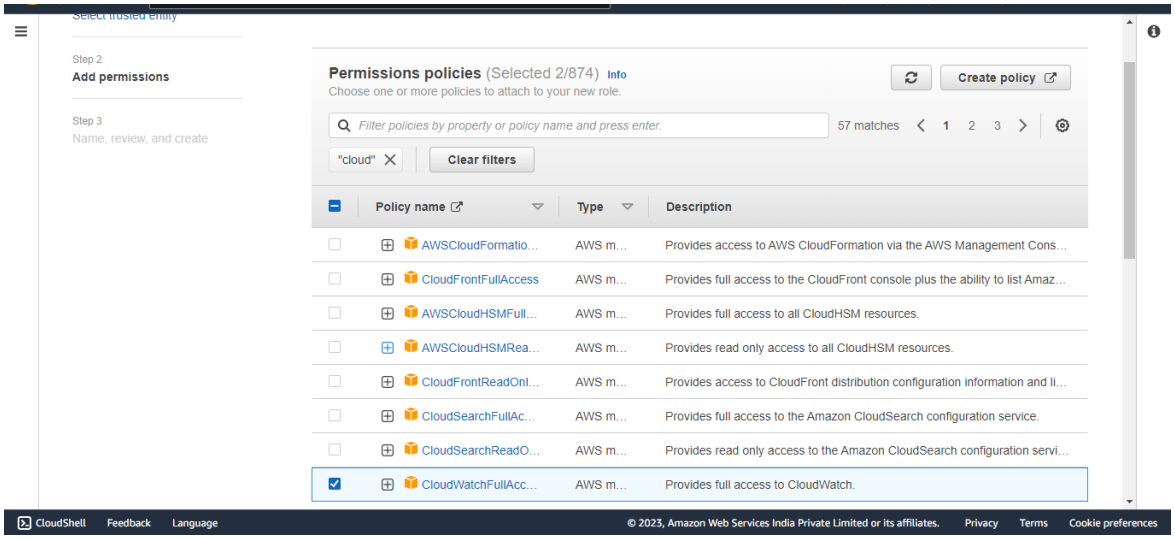
3. Search IAM on the console and go to "Roles". Click on Create Roles

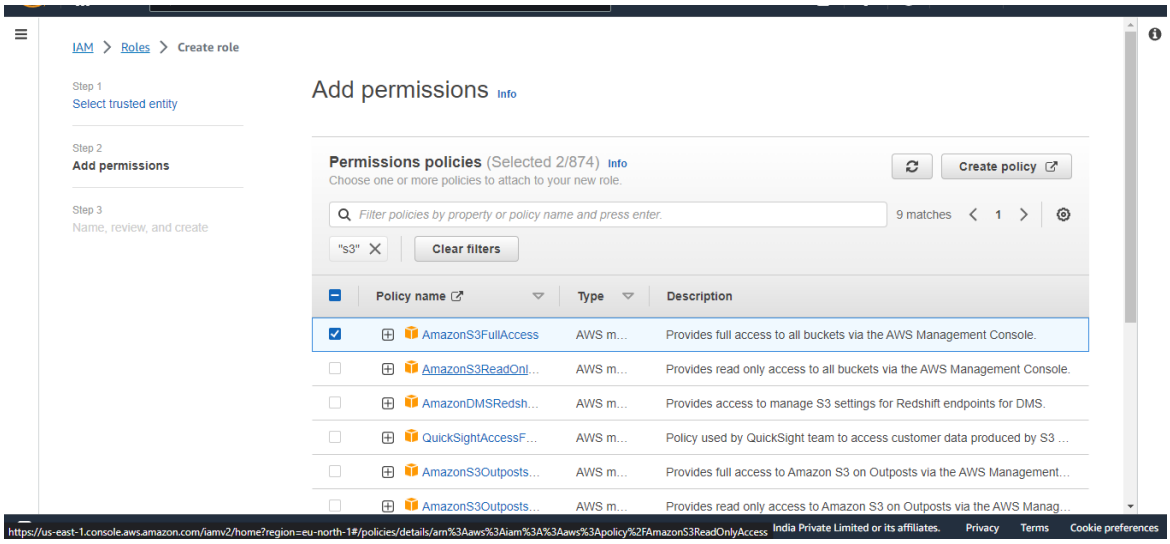


4. Select the options of "AWS service" and "lambda"

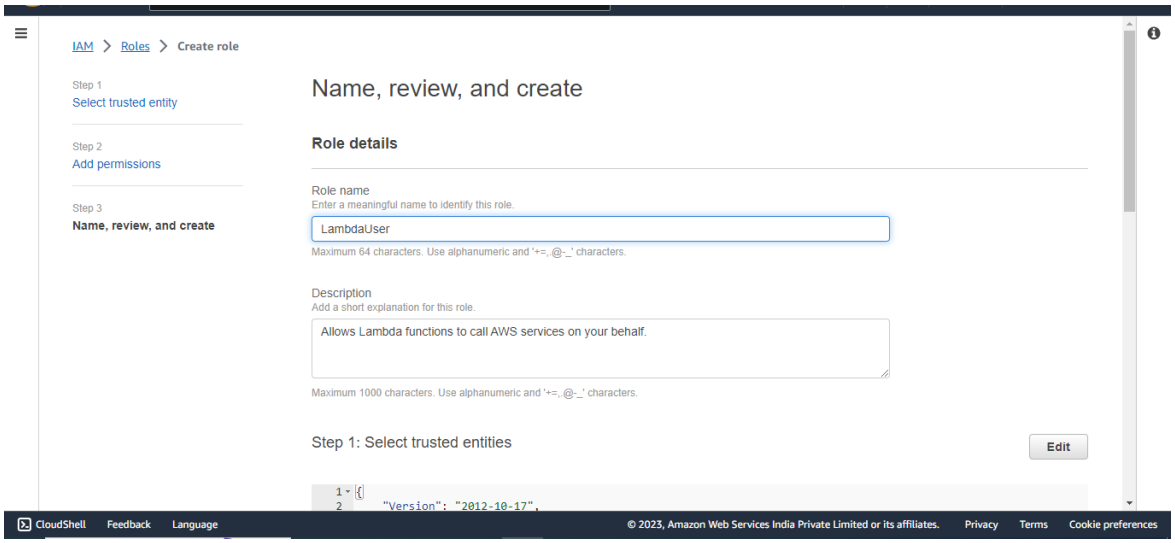


5. Enable the "CloudWatchFullAccess" and "AmazonS3FullAccess"

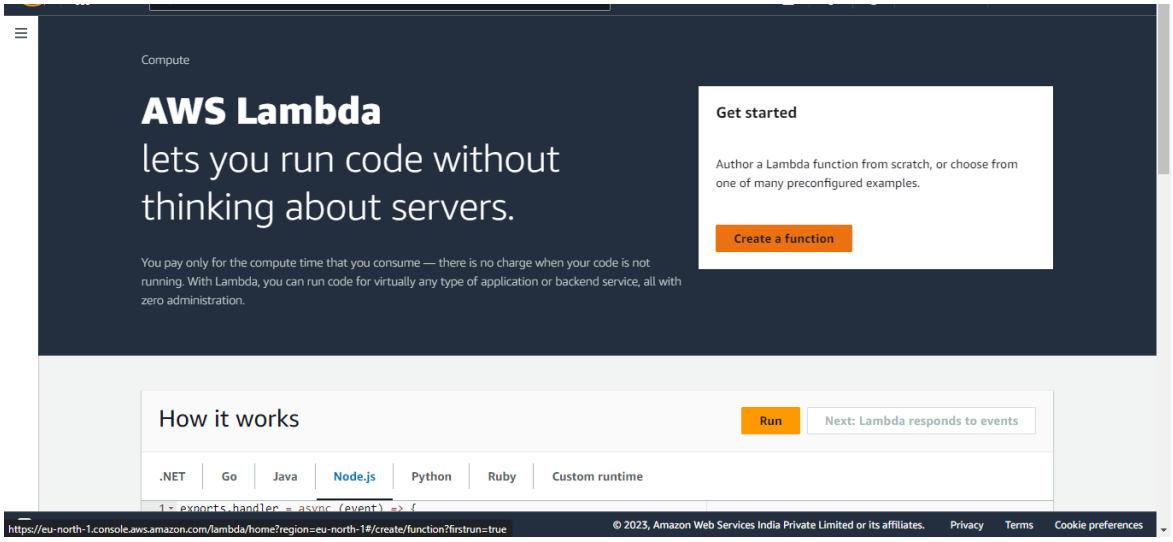




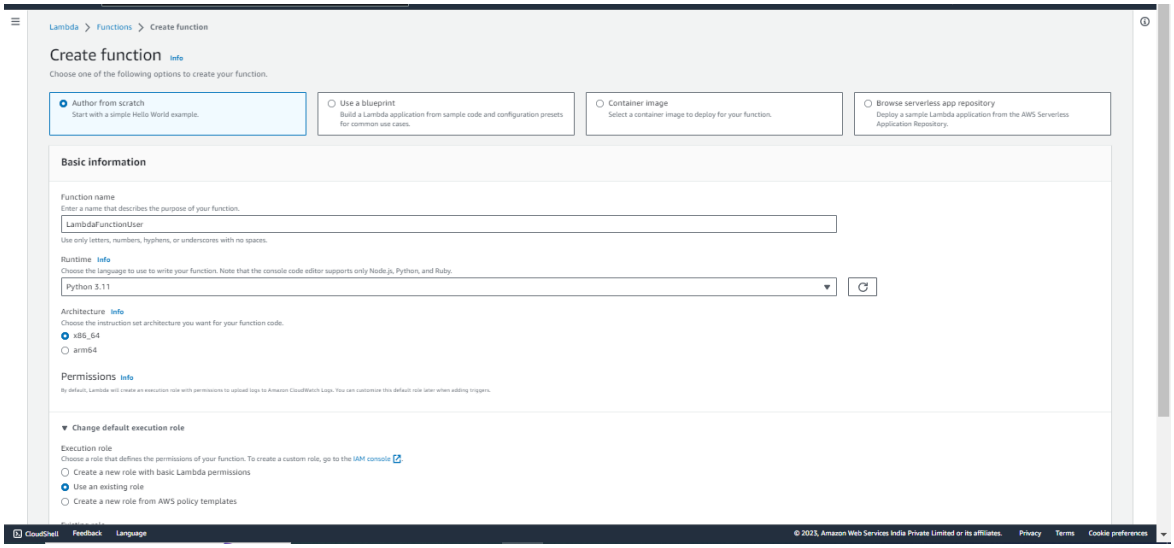
6. Click on Create Role and you will be redirected to this dashboard. Give the Role a Name and Click on Done.



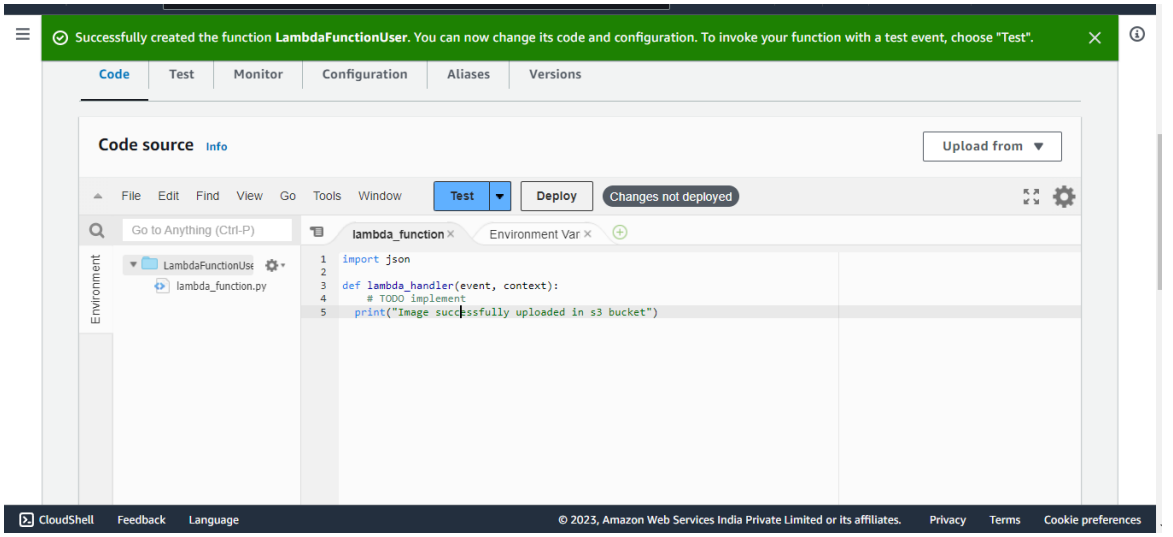
7. Search for Lambda in the console and click on Create Function.



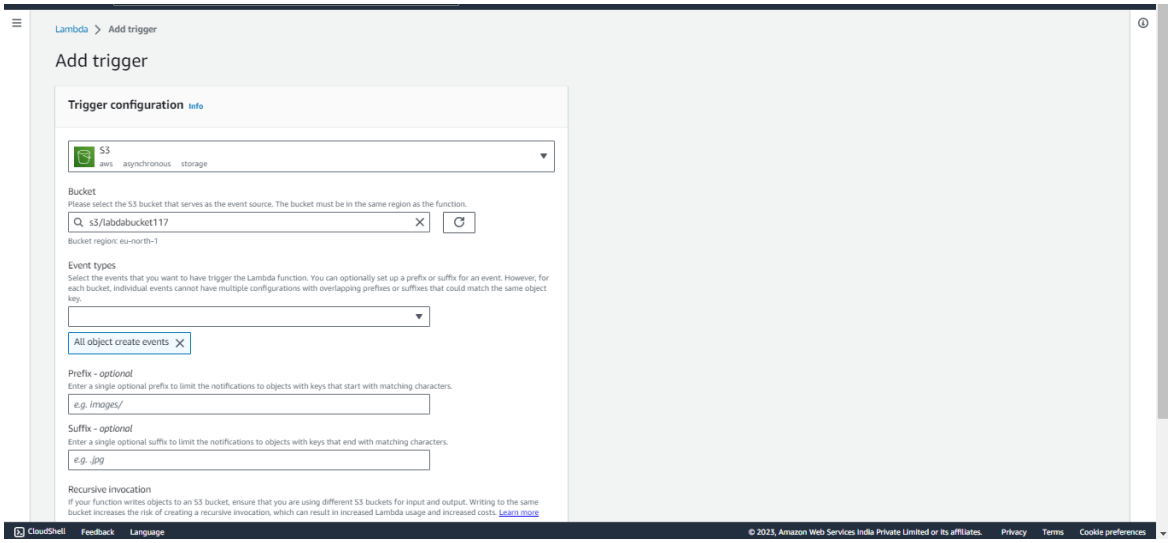
8. Change the settings of the Lambda Function.



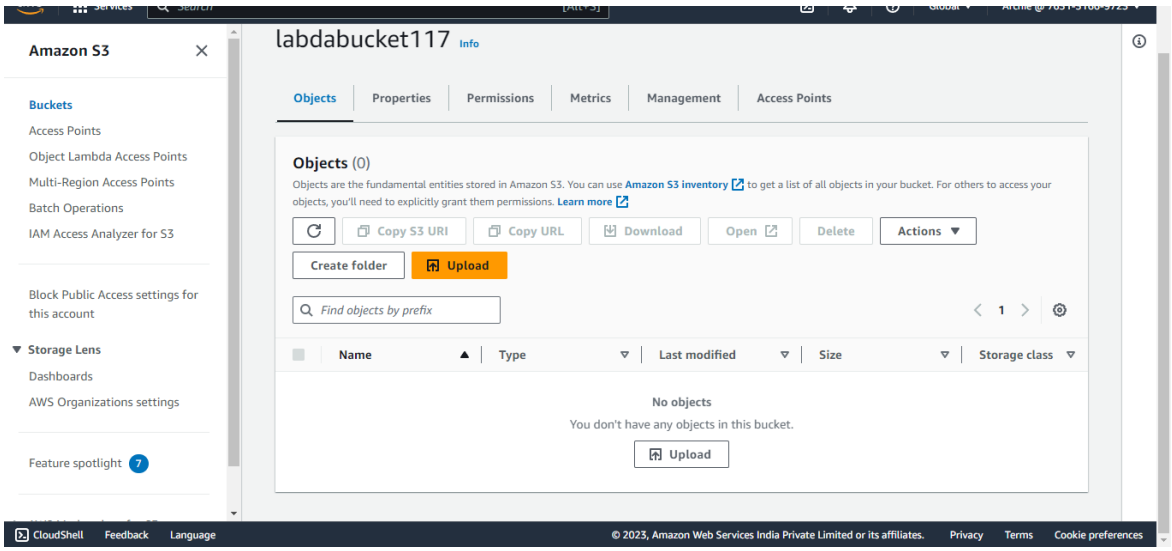
9. Add the python code and click on deploy to save the changes.

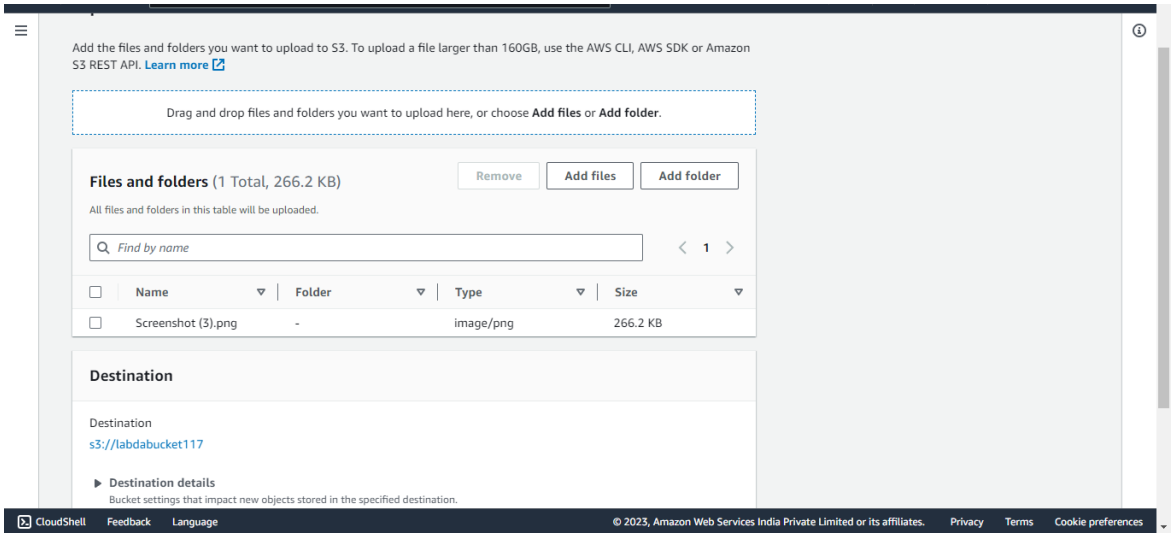


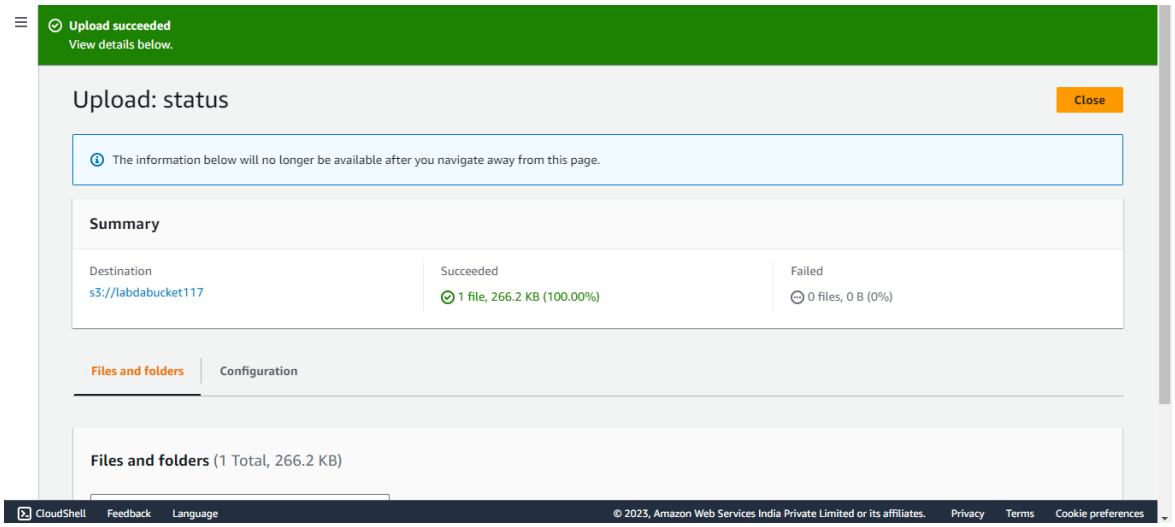
10. Scroll above and click on ADD TRIGGER. Select the following options and click on Done.



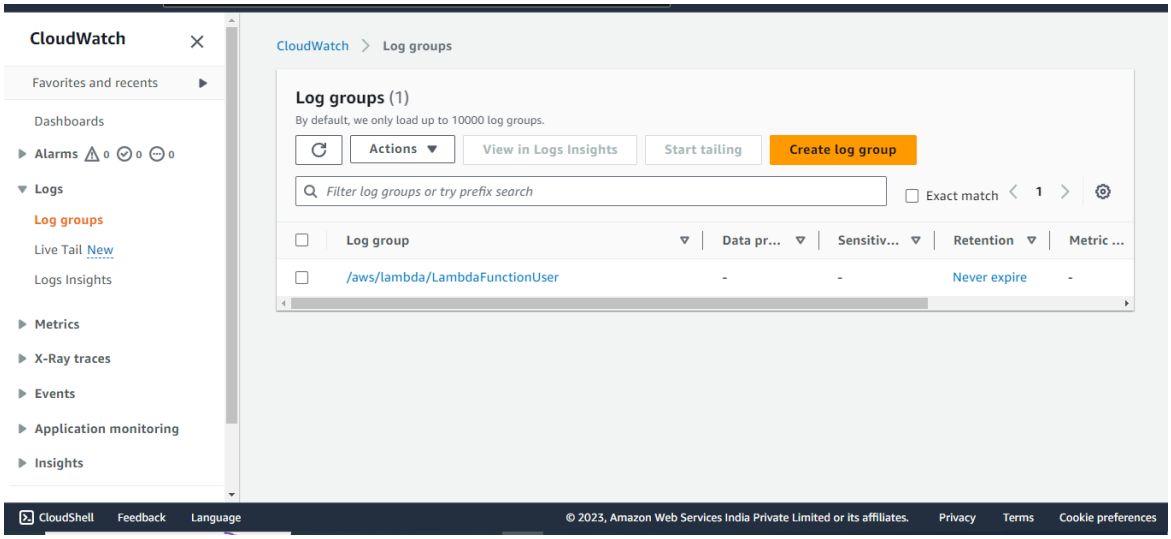
11. Go to S3 bucket and click on Add files. Select a image and click on Upload.



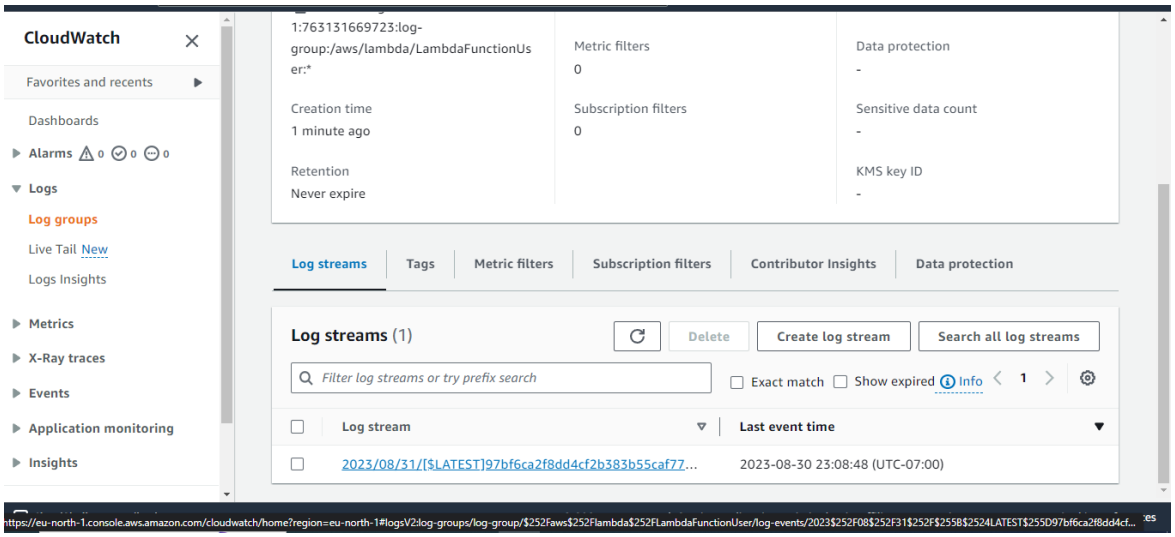


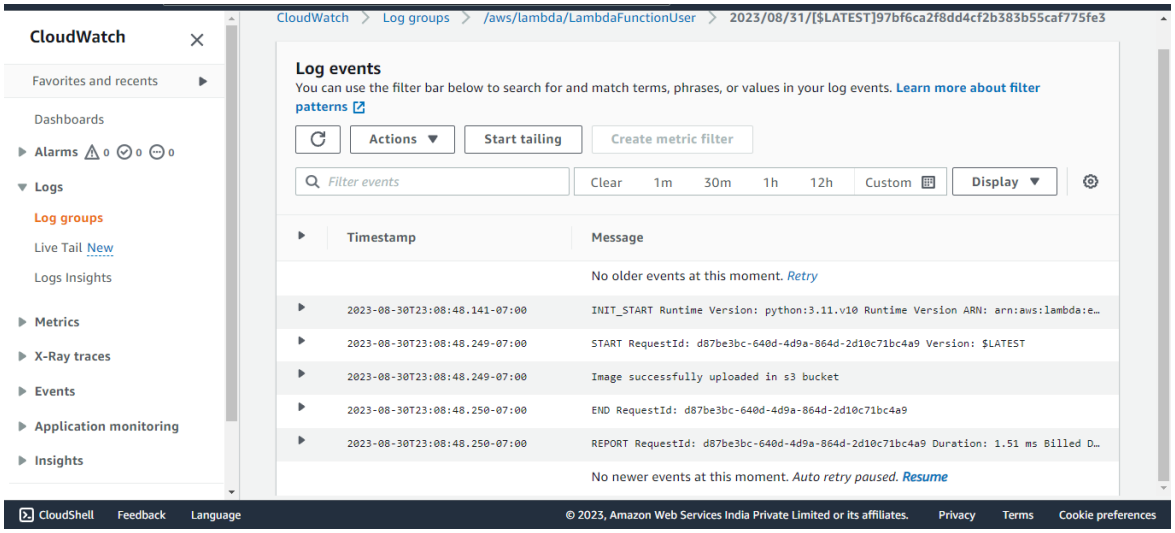


12. Search CloudWatch and go to Log groups. Select the existing Log Group.



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: Bhuvan Sawant; Branch: I.T (T.E.); Roll Number: 110 17/10/2023

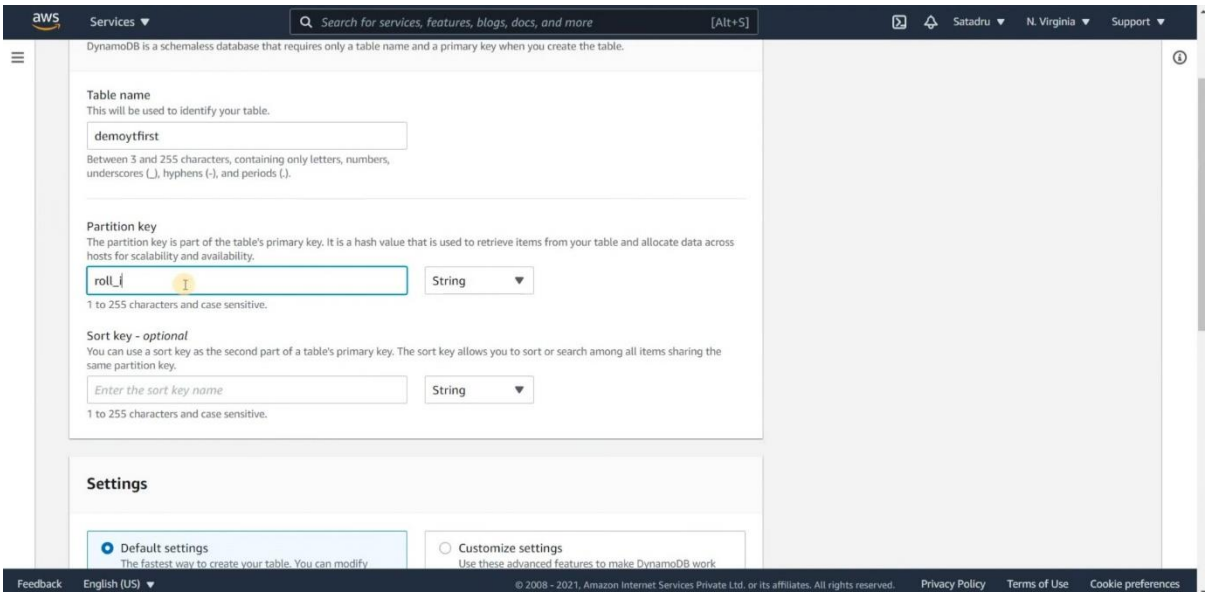
Aim: To create a Lambda function using Python for adding data to Dynamo DB database.

LO mapped: LO6

Theory:

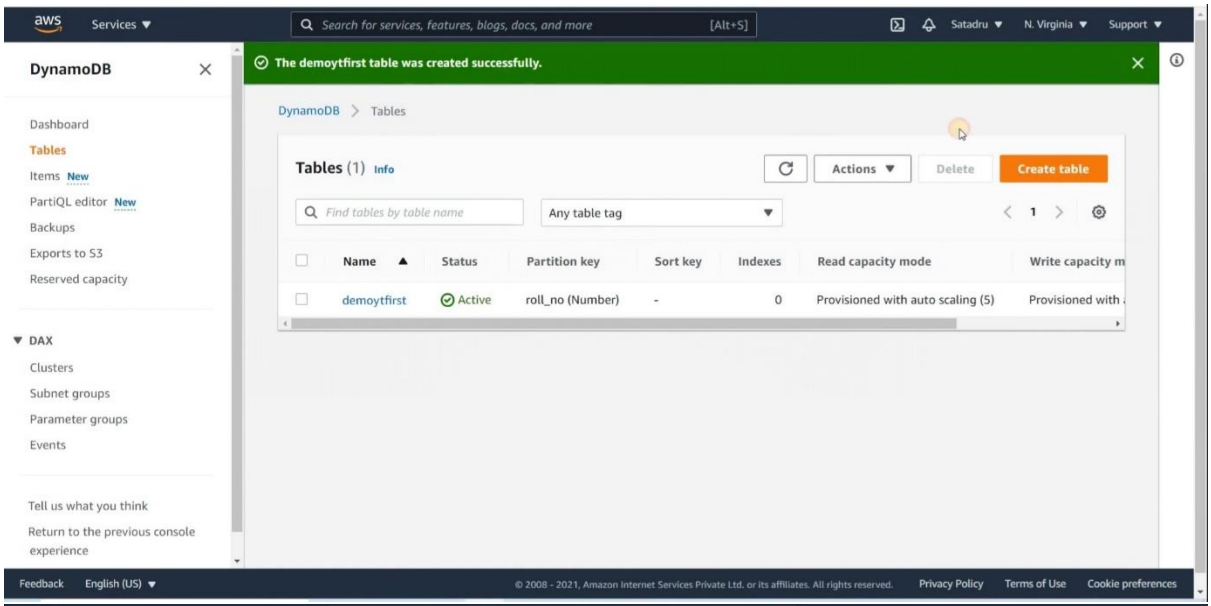
Step 1: Set Up AWS Environment

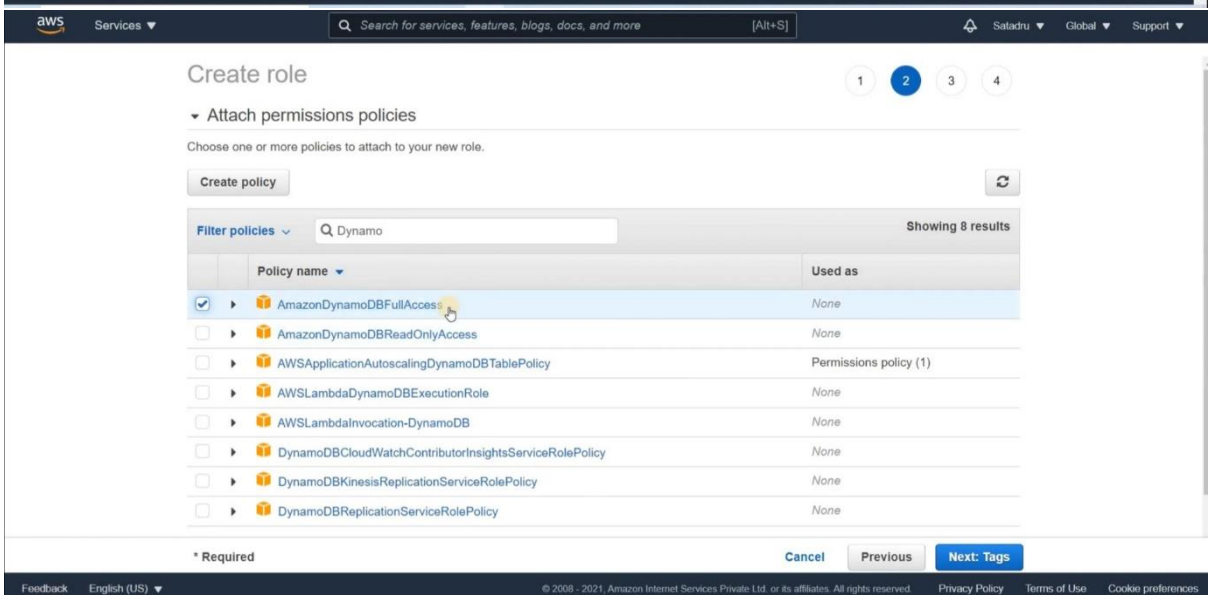
1. **Create an AWS Account:** If you don't have an AWS account, sign up for one at AWS Console.
2. **Access AWS Lambda Console:**



- Go to the AWS Lambda Console.
- Click on "Create function."

3. **Create a New Lambda Function**

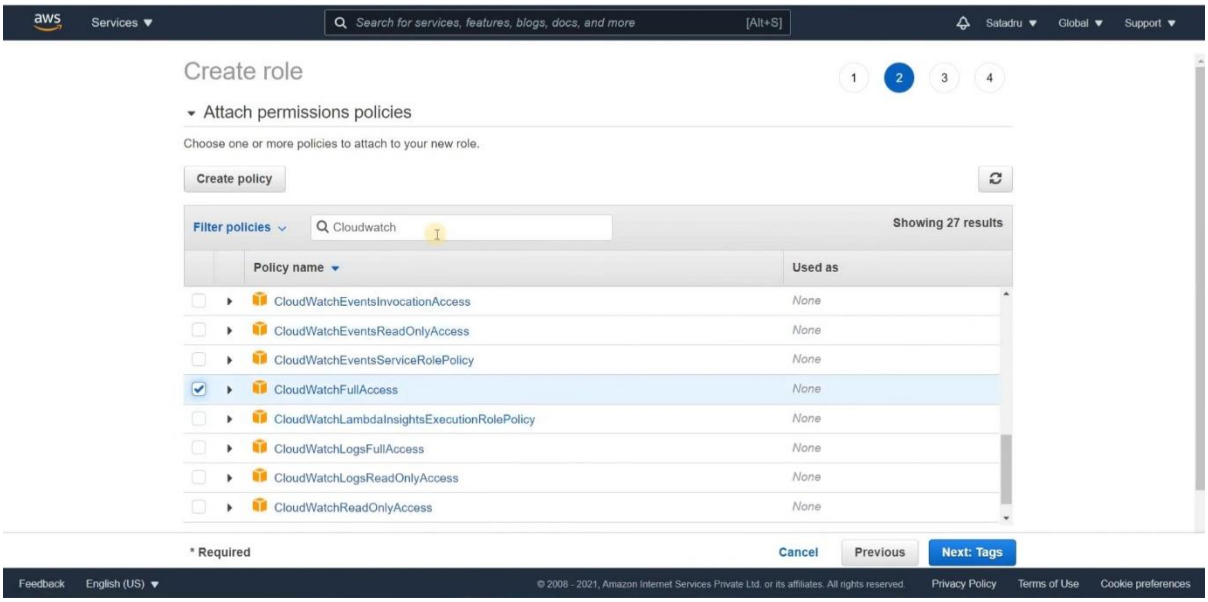




:

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

4. **Configure Execution Role**

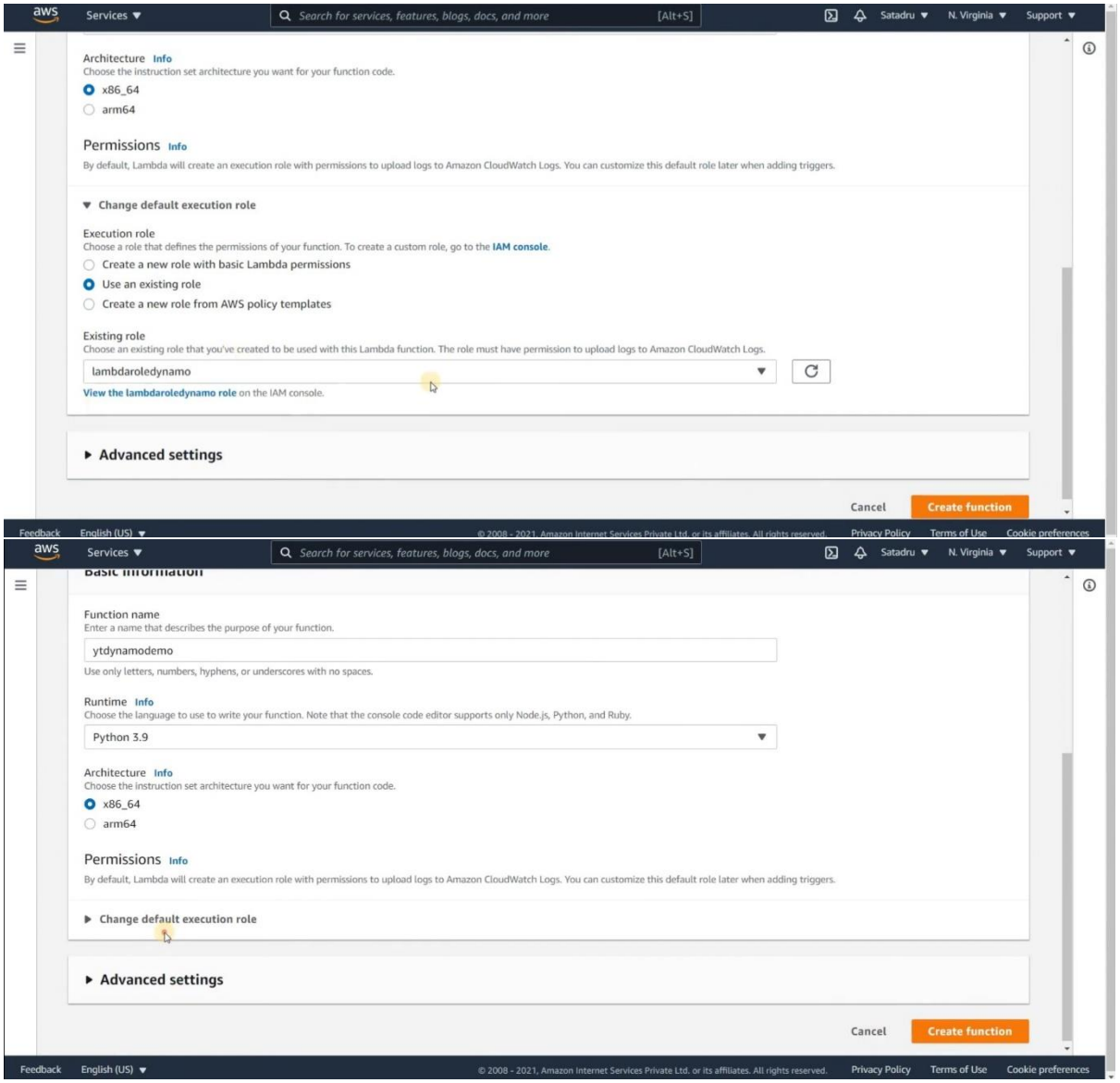


:

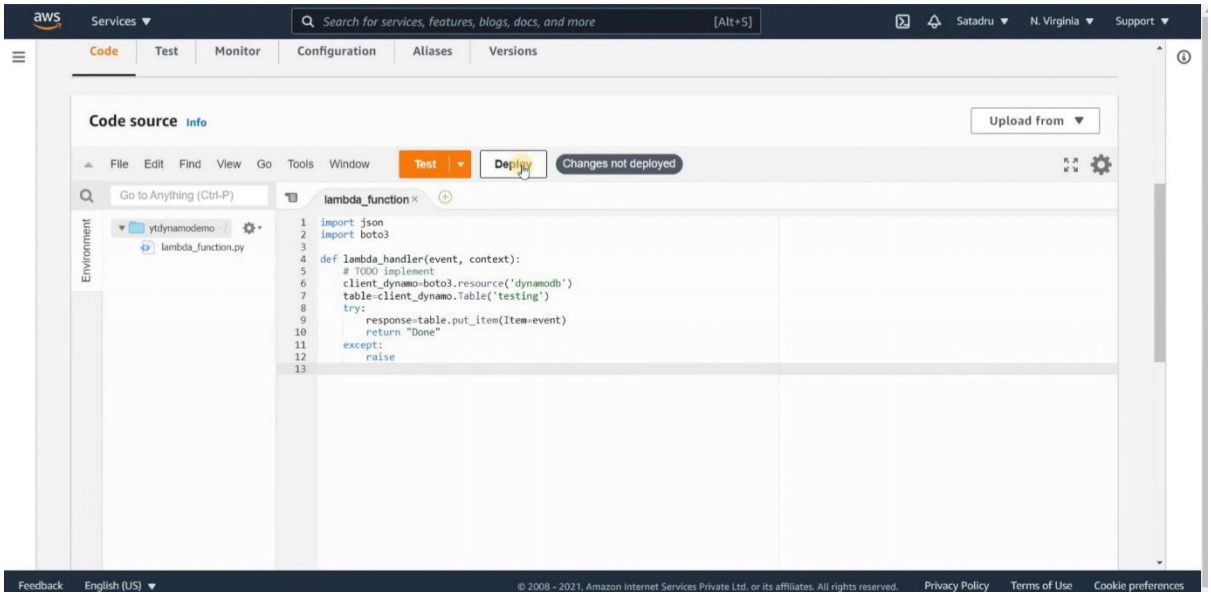
- Create a new role with basic Lambda permissions and additional permissions to interact with DynamoDB.
- Attach the role to your Lambda function.

Step 2: Set Up DynamoDB

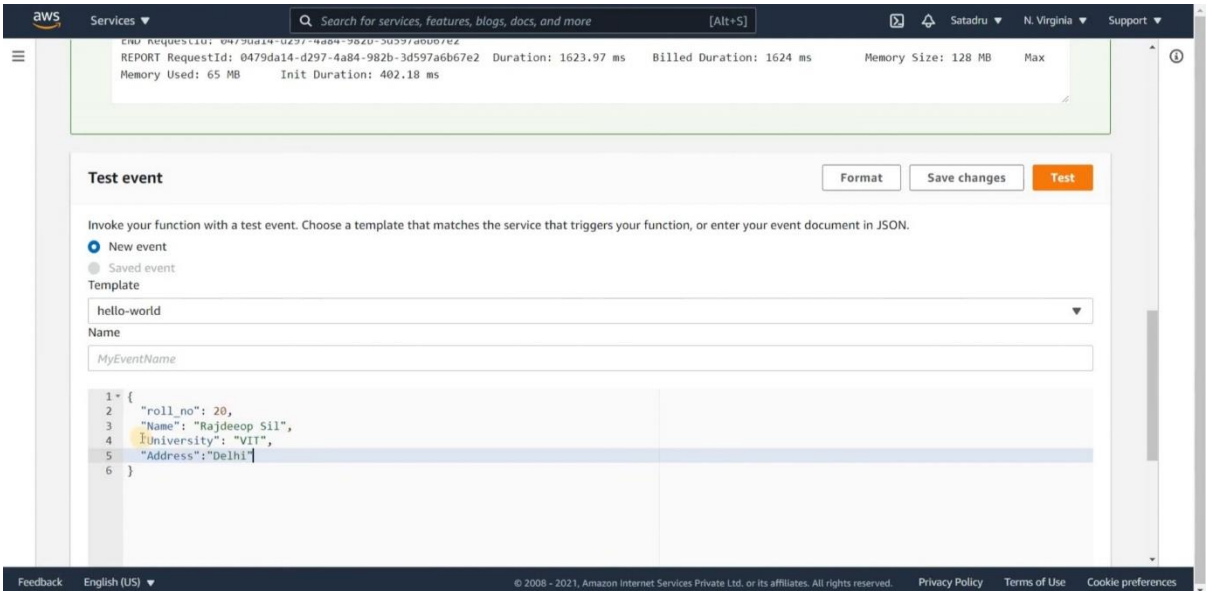
1. **Access DynamoDB Console:**
   - Go to the AWS DynamoDB Console.



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.

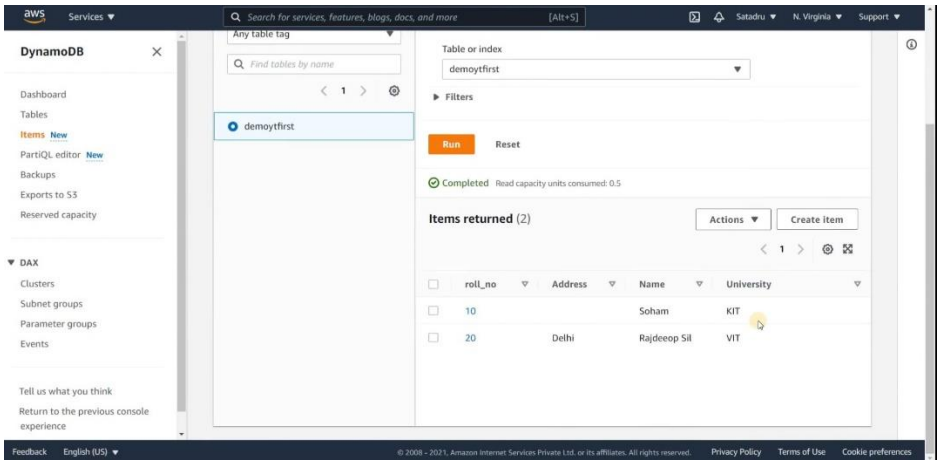
Step 8: Monitor and Troubleshoot

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: 1

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.