Academic Year: 2023-24
Semester: V

Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof. Manjusha Kashilkar

EXPERIMENT NO. 01

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

Theory:

AWS Cloud9 is a cloud-based integrated development environment (IDE) that lets you write, run, and debug your code with just a browser. It includes a code editor, debugger, and terminal. Cloud9 comes prepackaged with essential tools for popular programming languages, including JavaScript, Python, PHP, and more, so you don't need to install files or configure your development machine to start new projects. Since your Cloud9 IDE is cloud-based, you can work on your projects from your office, home, or anywhere using an internet-connected machine. Cloud9 also provides a seamless experience for developing serverless applications enabling you to easily define resources, debug, and switch between local and remote execution of serverless applications. With Cloud9, you can quickly share your development environment with your team, enabling you to pair program and track each other's inputs in real time.

Benifits:

CODE WITH JUST A BROWSER

AWS Cloud9 gives you the flexibility to run your development environment on a managed Amazon EC2 instance or any existing Linux server that supports SSH. This means that you can write, run, and debug applications with just a browser, without needing to install or maintain a local IDE. The Cloud9 code editor and integrated debugger include helpful, time-saving features such as code hinting, code completion, and step-through debugging. The Cloud9 terminal provides a browser- based shell experience enabling you to install additional software, do a git push, or enter commands.

CODE TOGETHER IN REAL TIME

AWS Cloud9 makes collaborating on code easy. You can share your development environment with your team in just a few clicks and pair program together. While collaborating, your team members can see each other type in real time, and instantly chat with one another from within the IDE.

BUILD SERVERLESS APPLICATIONS WITH EASE

AWS Cloud9 makes it easy to write, run, and debug serverless applications. It preconfigures the development environment with all the SDKs, libraries, and plug-ins needed for serverless development. Cloud9 also provides an environment for locally testing and debugging AWS Lambda functions. This allows you to iterate on your code directly, saving you time and improving the quality of your code.

DIRECT TERMINAL ACCESS TO AWS

AWS Cloud9 comes with a terminal that includes sudo privileges to the managed Amazon EC2 instance that is hosting your development environment and a preauthenticated AWS Command Line Interface. This makes it easy for you to quickly run commands and directly access AWS services

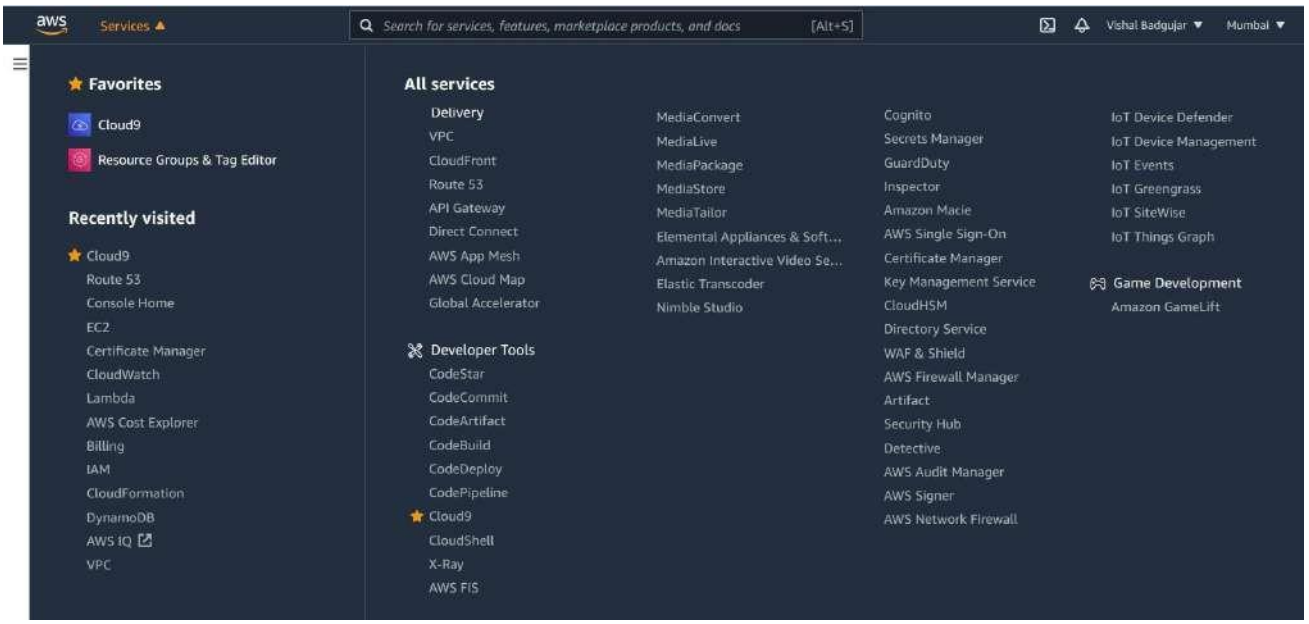
START NEW PROJECTS QUICKLY

AWS Cloud9 makes it easy for you to start new projects. Cloud9's development environment comes prepackaged with tooling for over 40 programming languages, including Node.js, JavaScript, Python, PHP, Ruby, Go, and C++. This enables you to start writing code for popular application stacks within minutes by eliminating the need to install or configure files, SDKs, and plug-ins for your development machine. Because Cloud9 is cloud-based, you can easily maintain multiple development environments to isolate your project's resources.

Steps:

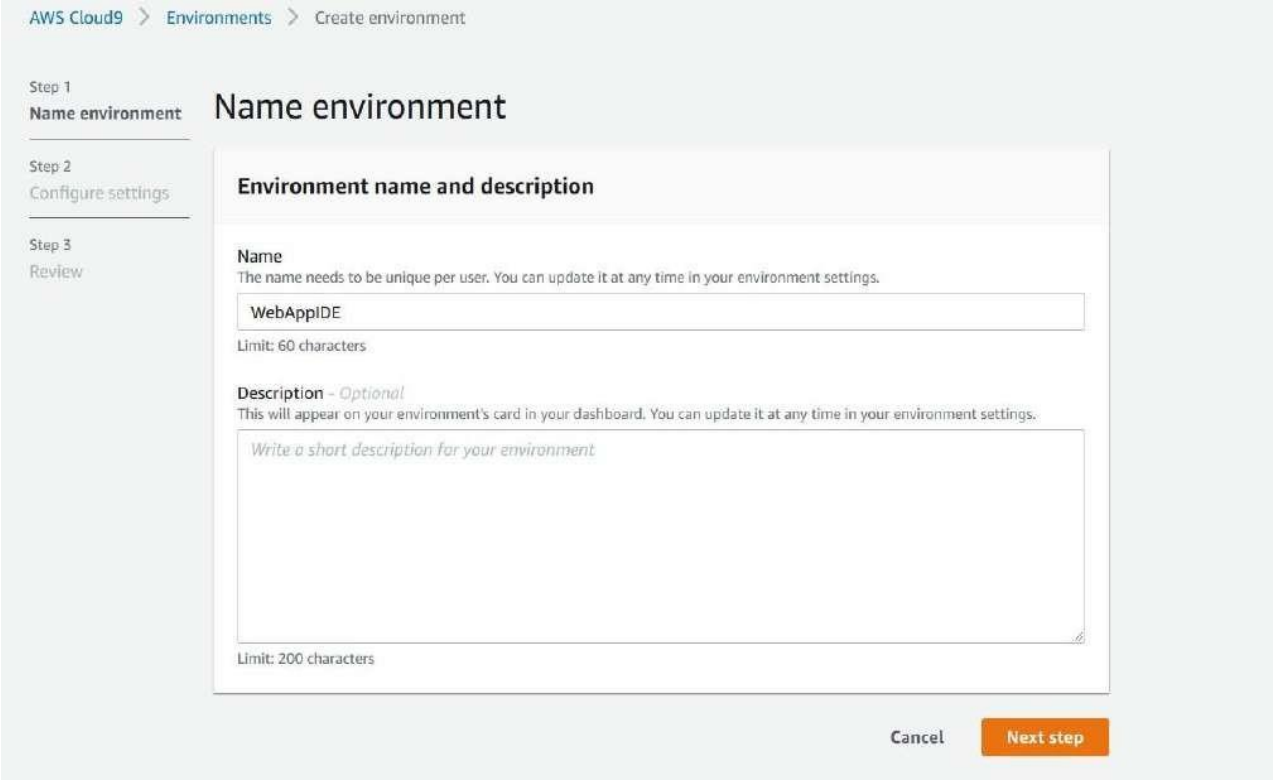
1. Login with your AWS account.

2. Navigate to Cloud 9 service from Developer tools section as below:

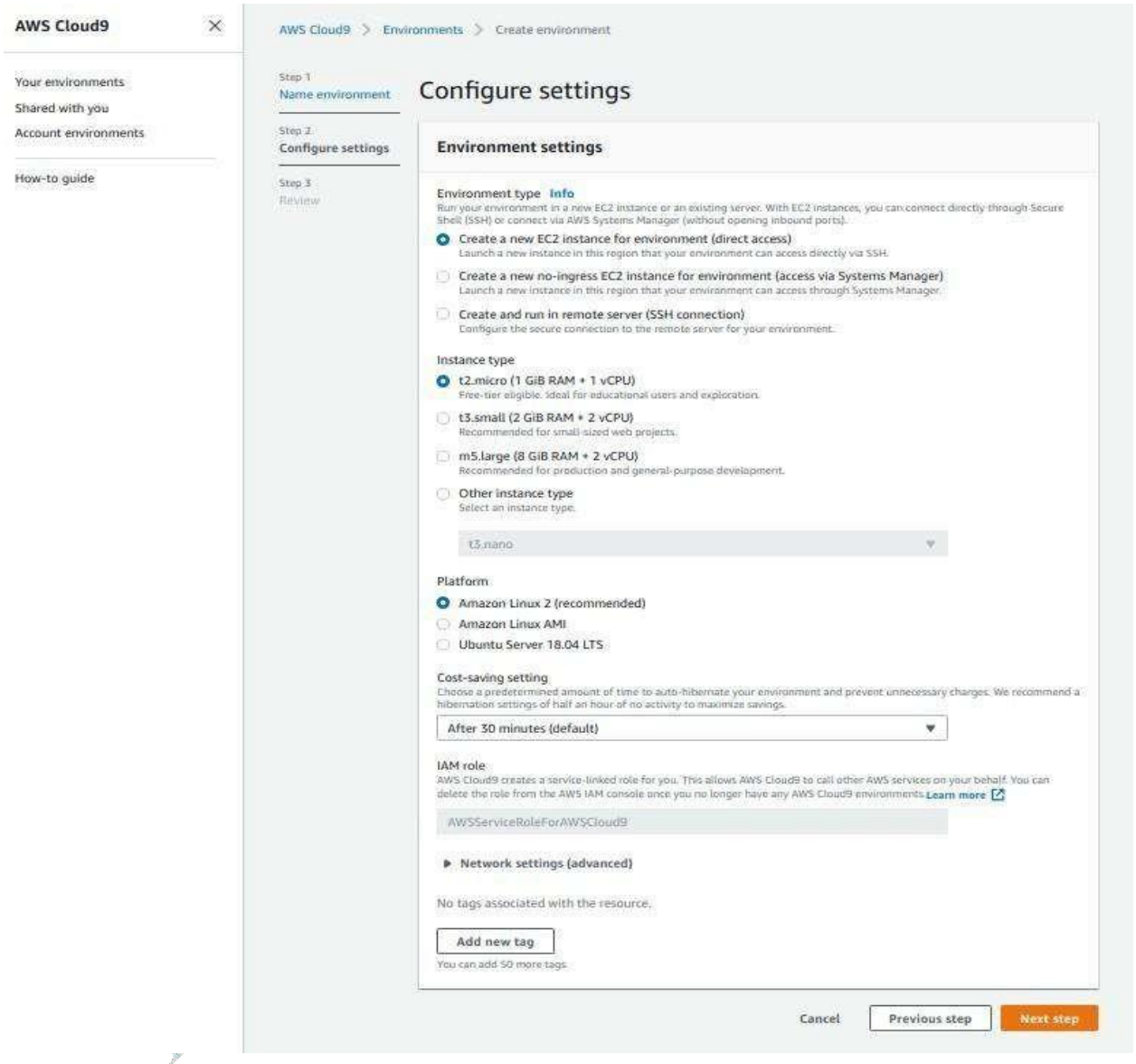


3. Click on Create Enviornment :

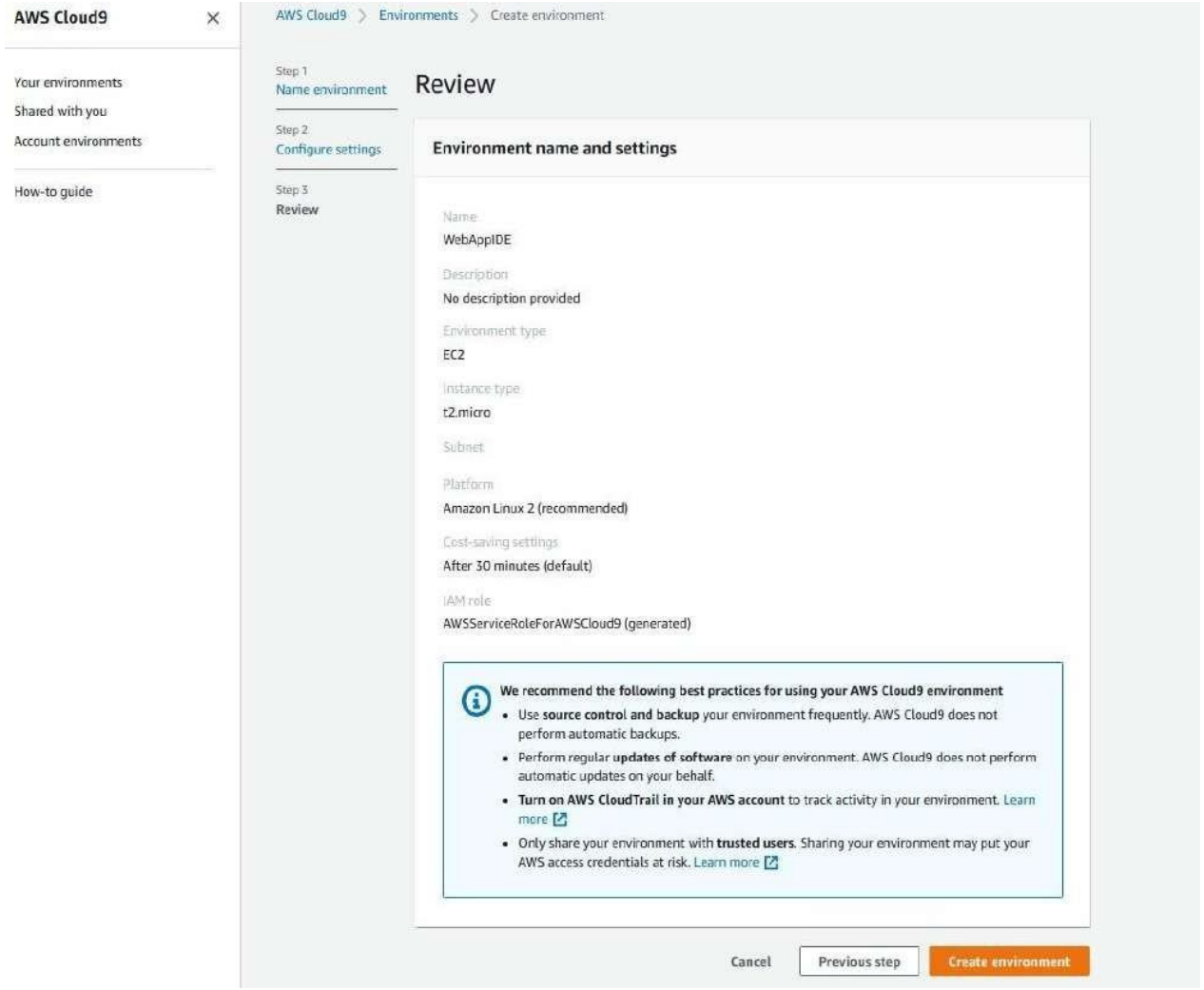
1. Provide name for the Enviornment (WebAppIDE) and click on next.

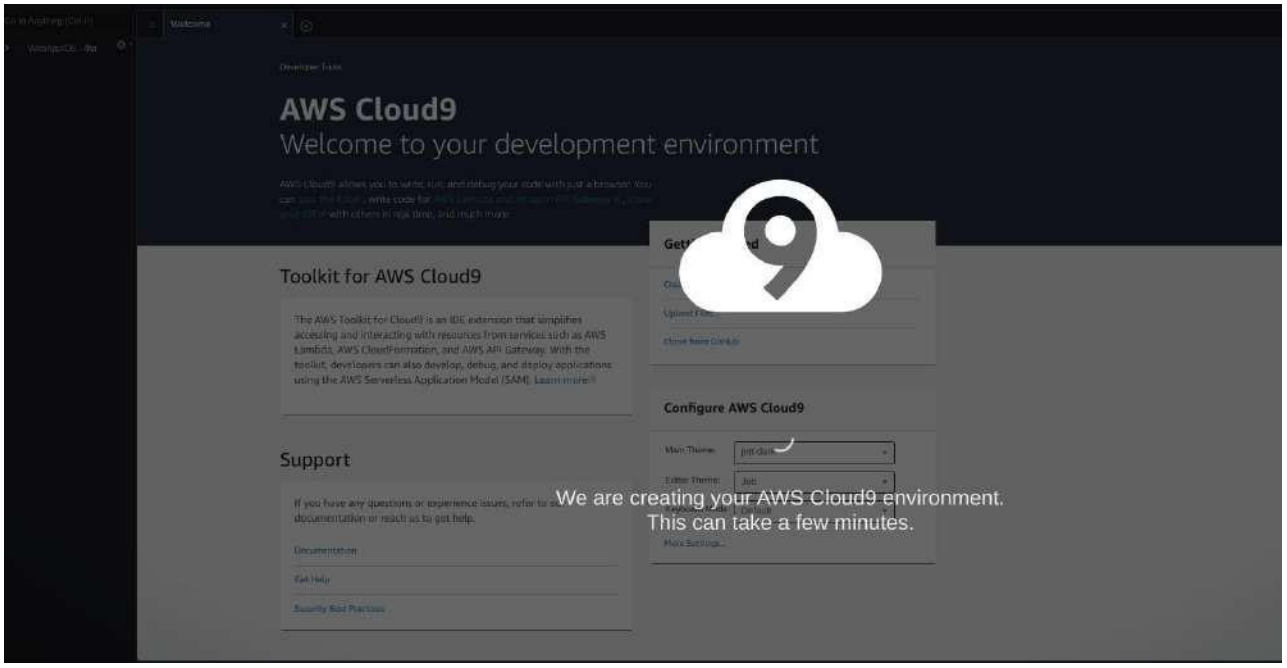


2. Keep all the Default seetings as shown in below:



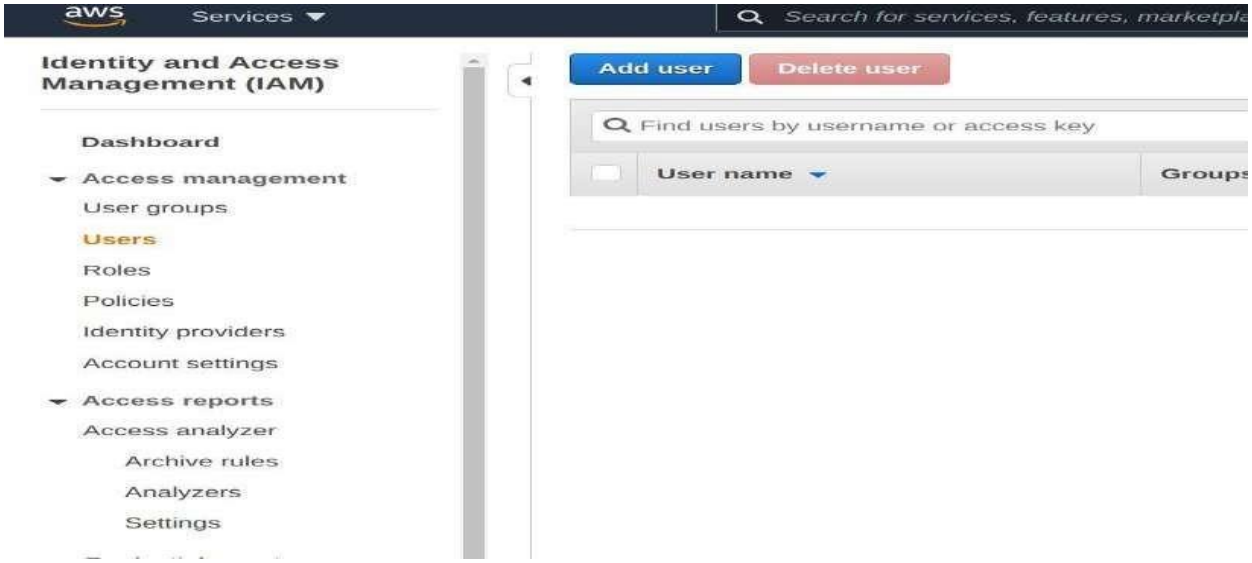
3. Review the Enviornment name and Settings and click on Create Enviornment:



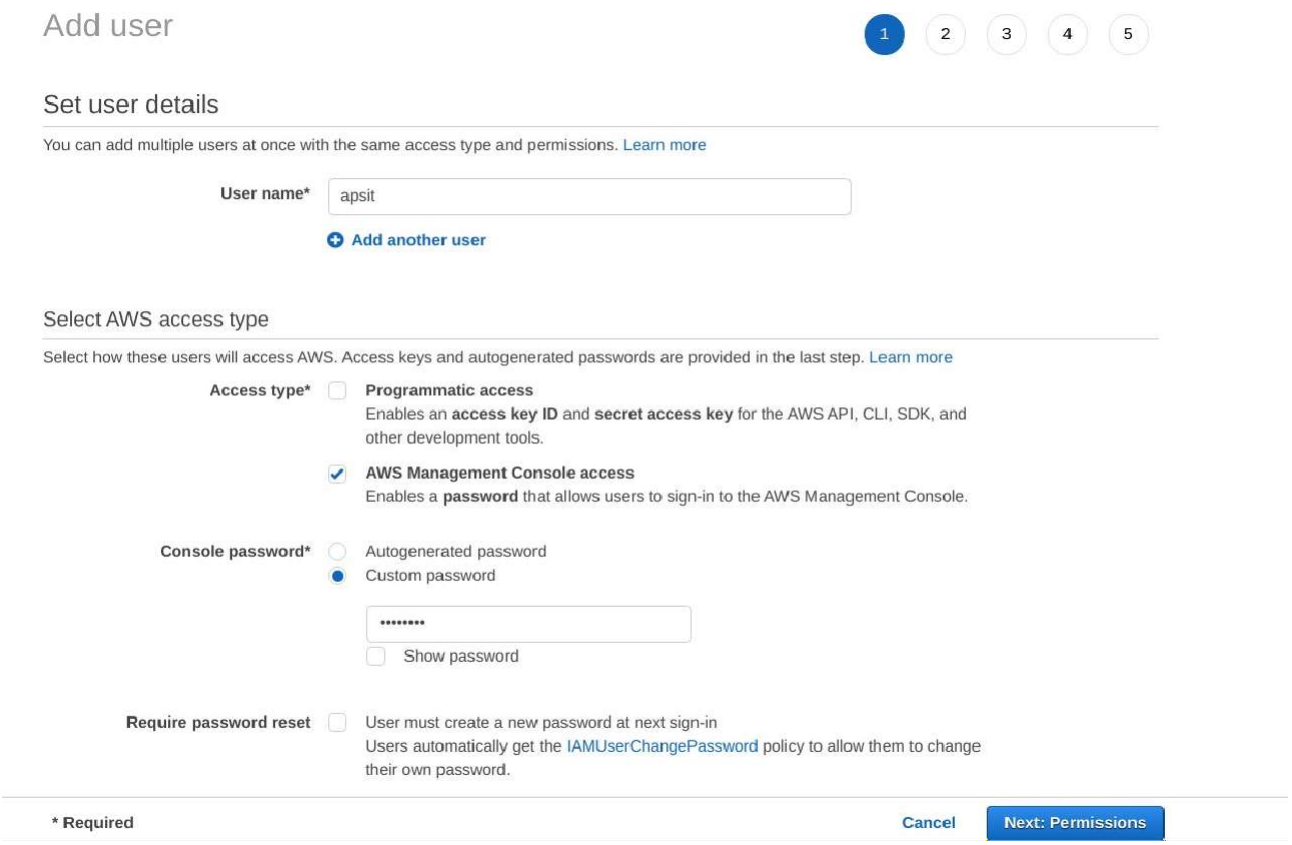


It will take few minutes to create aws instance for your Cloud 9 Enviornment.

4. Till that time open IAM Identity and Access Management in order to Add user In other tab.

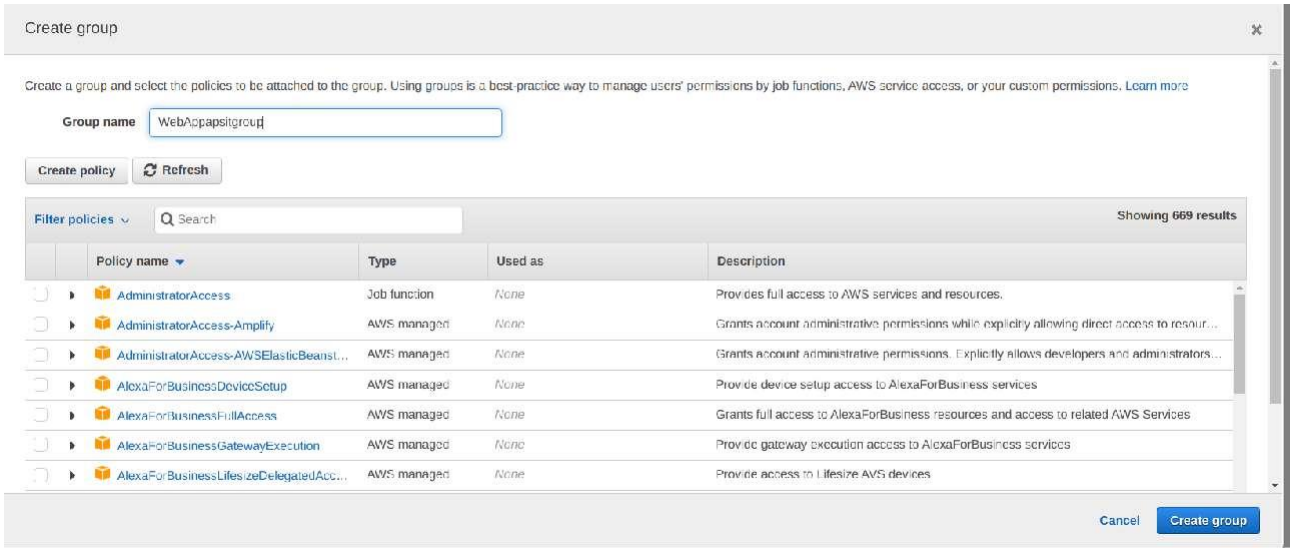


5. Add user provide manual password if you want and click on Next permission tab.

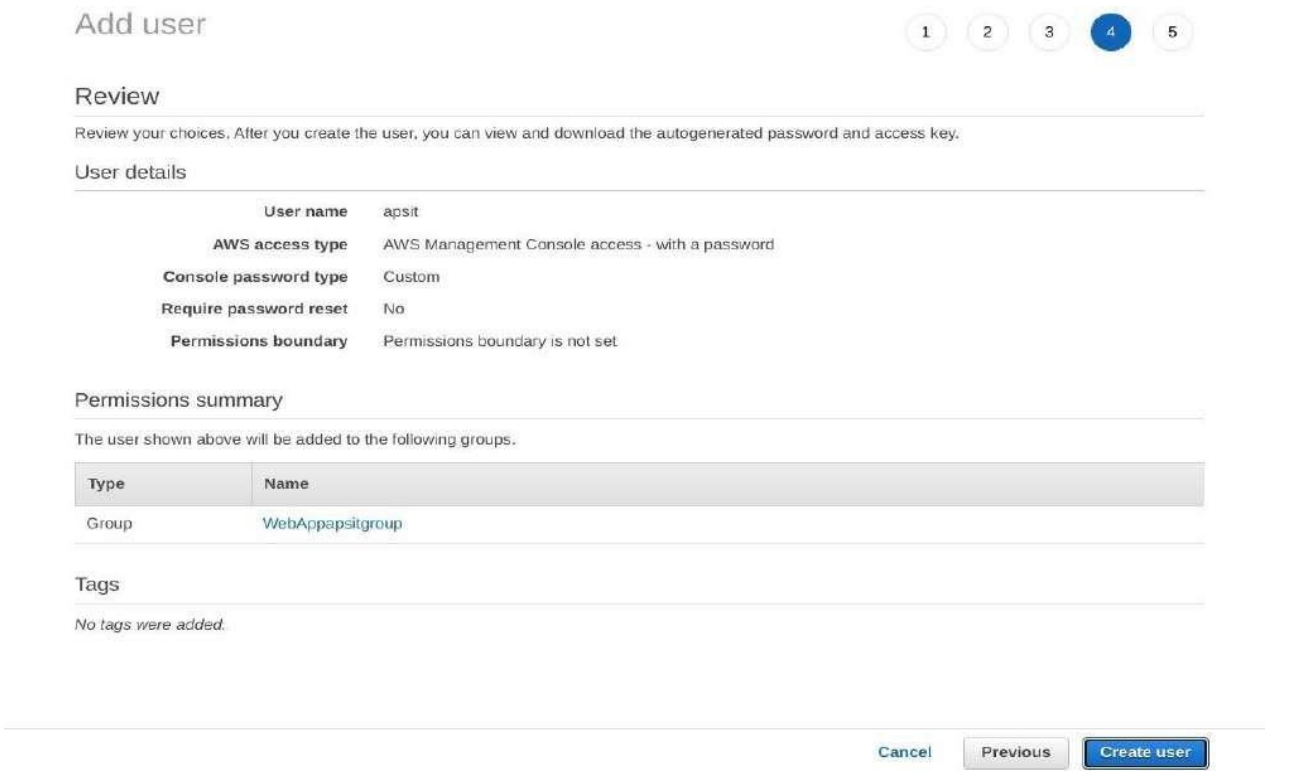


6. Click on Create group

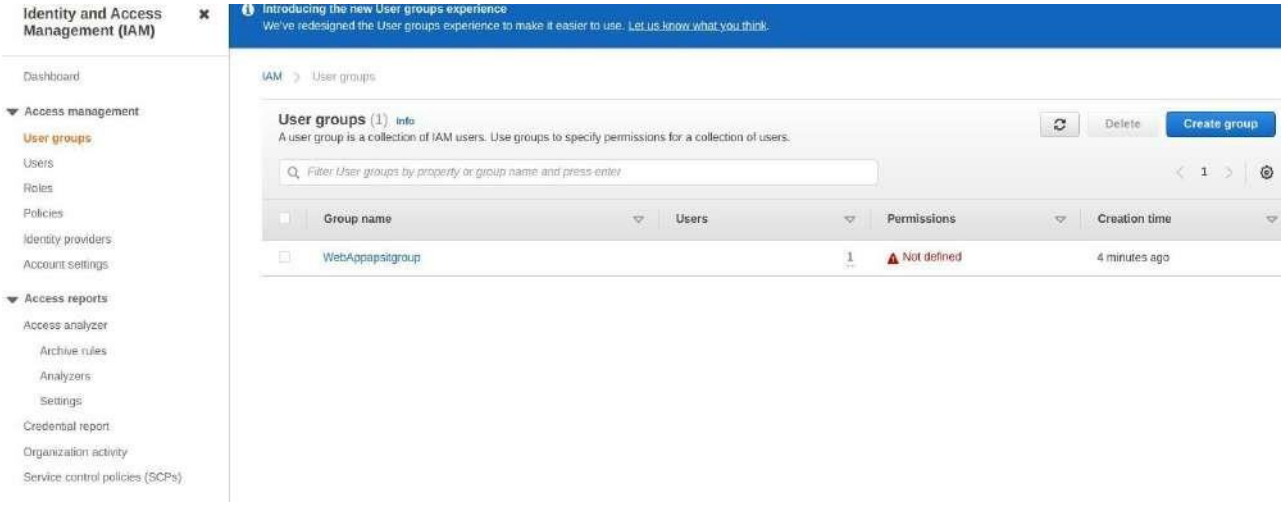
7. Provide group name and click on create group.



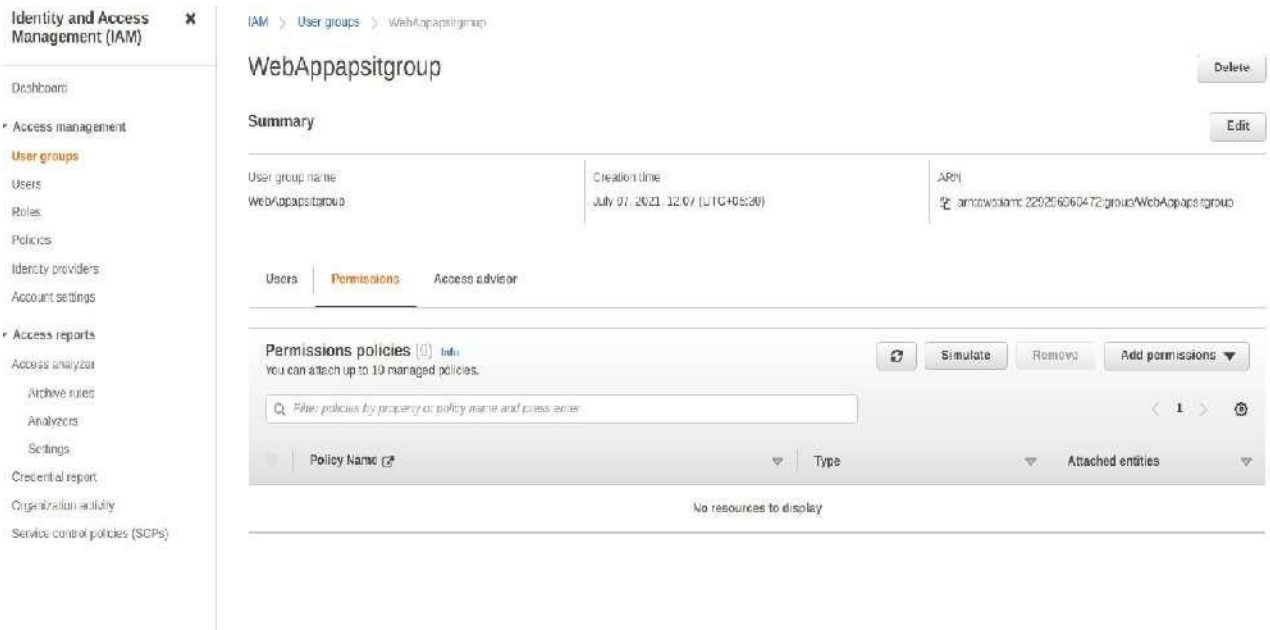
8. After that group is created click on next if u want to provide tag else click on Review for user settings and click on create user as shown in fig.



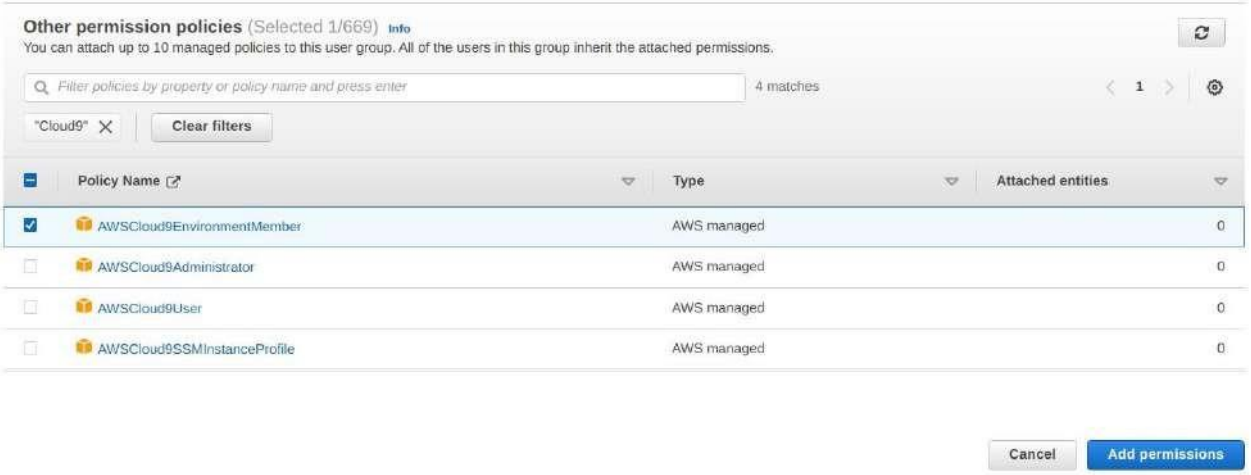
9. Now close that window and Navigate to user Groups from left pane in IAM.



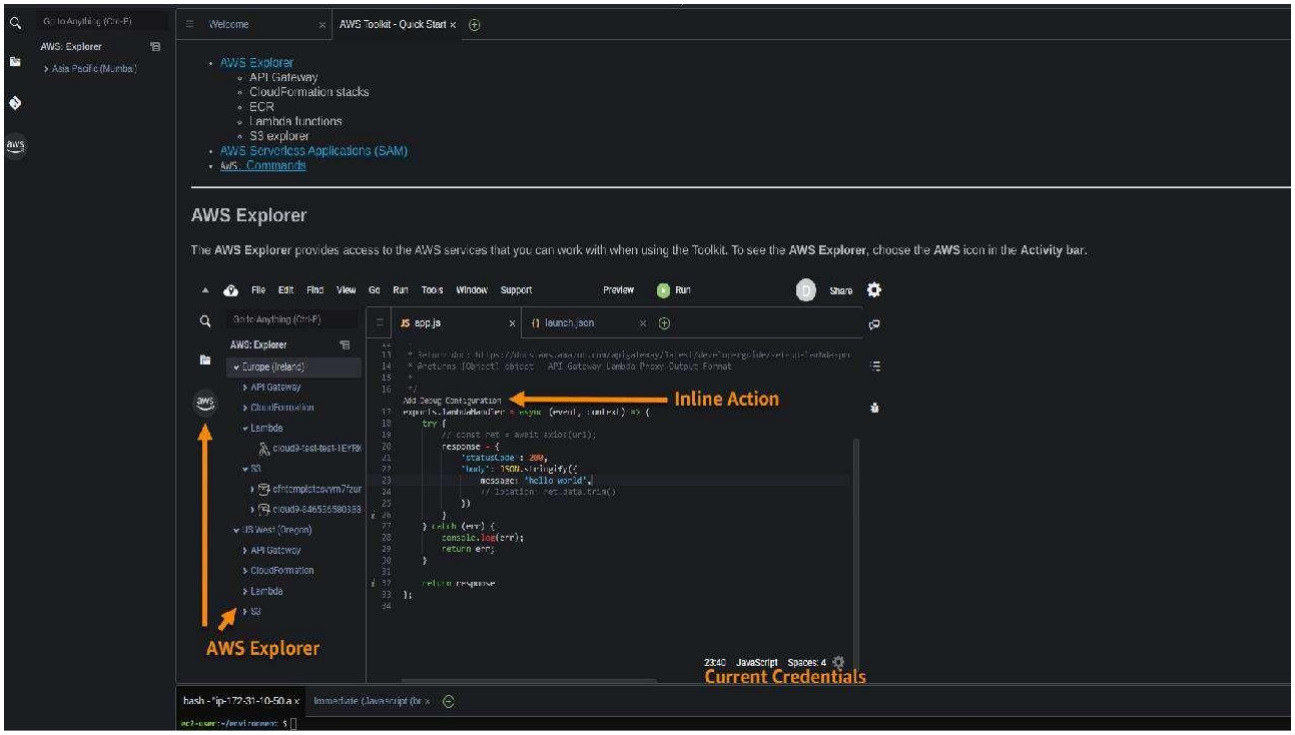
10. click on your group name which you have created and nevigate to permission tab as shown:



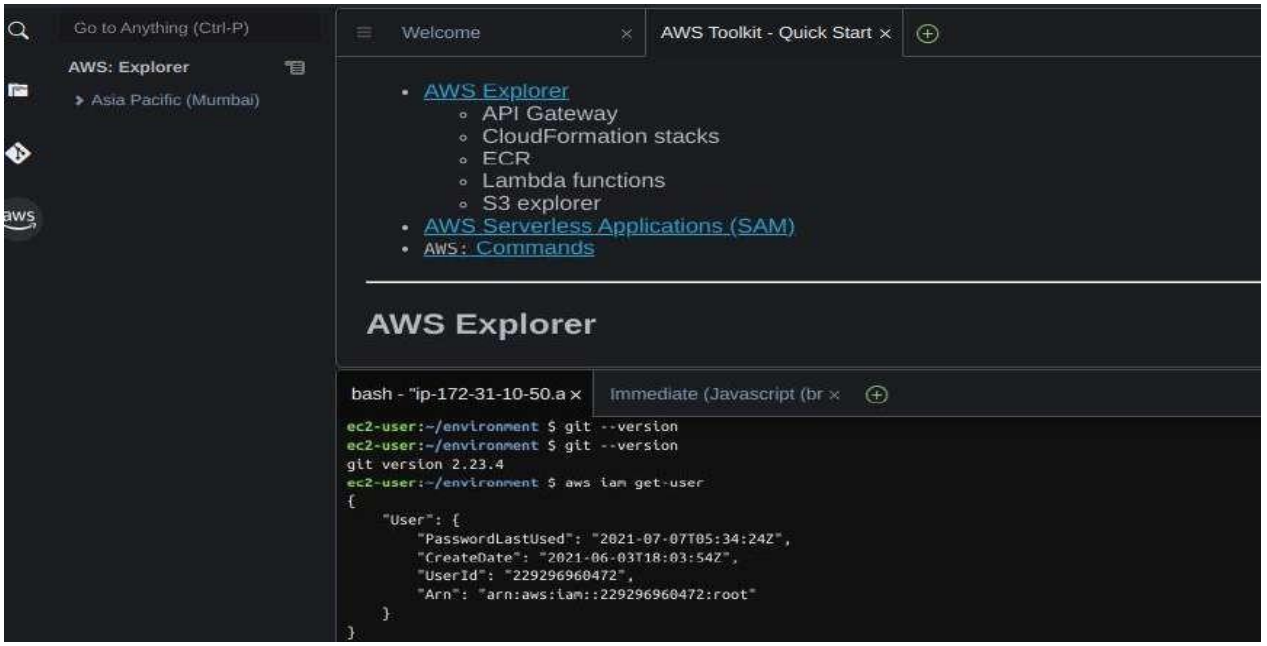
11. Now click on Add permission and select Attach Policy after that search for Cloud9 related policy and select Awscloud9EnviornmentMember policy and add it.



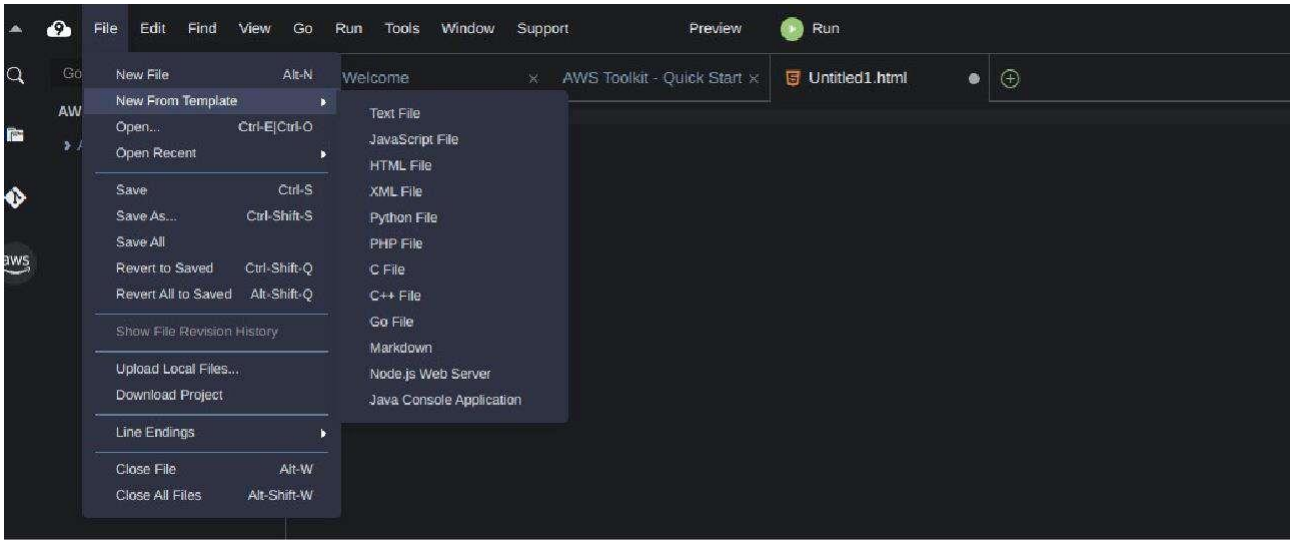
12. now we move towards our cloud9 IDE Enviornment tab it shows as shown :



13. If you check at bottom side Cloud9 IDE also giving you and aws CLI for command operations: as we here checked git version, iam user details and so on...

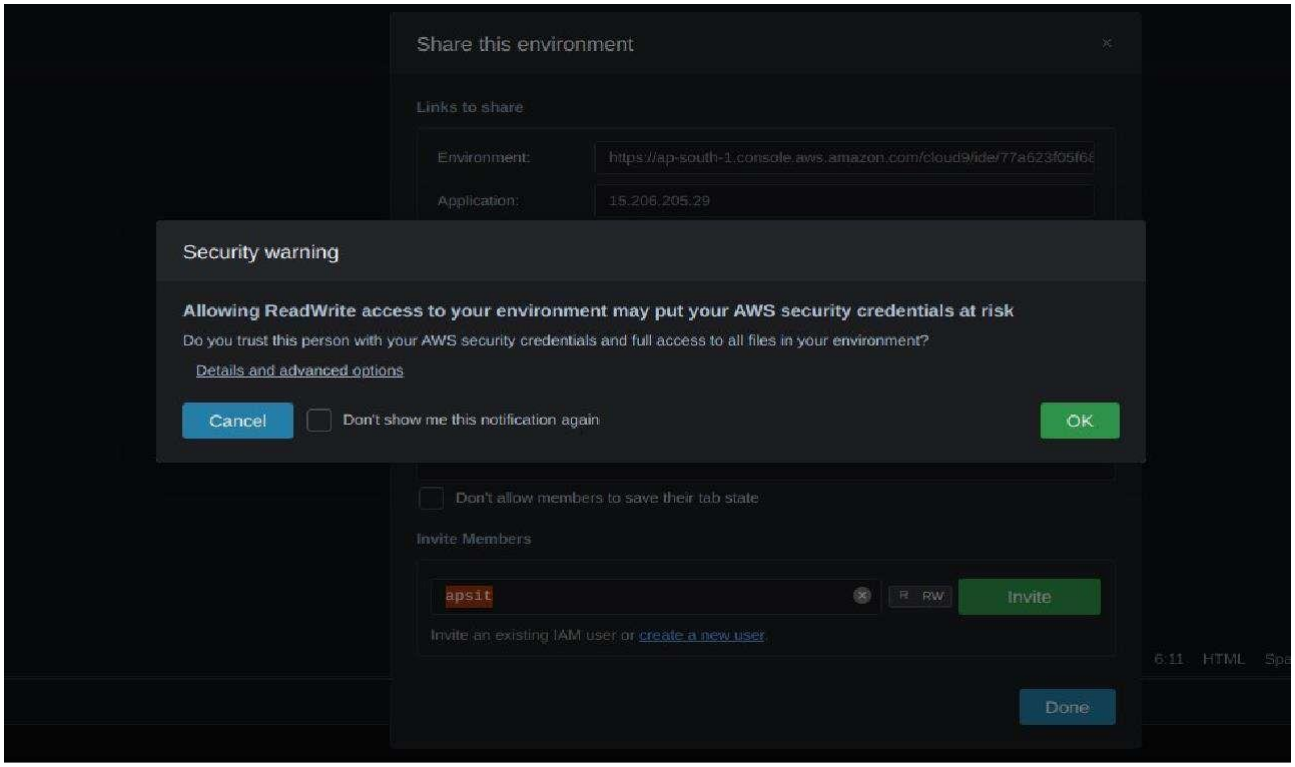


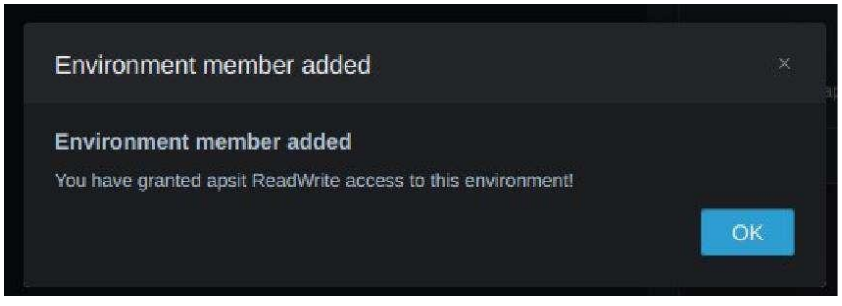
14. Now we will setup collaborative enviornment Click on File you can create new file or choose from template, here m opting html file to collaborate.



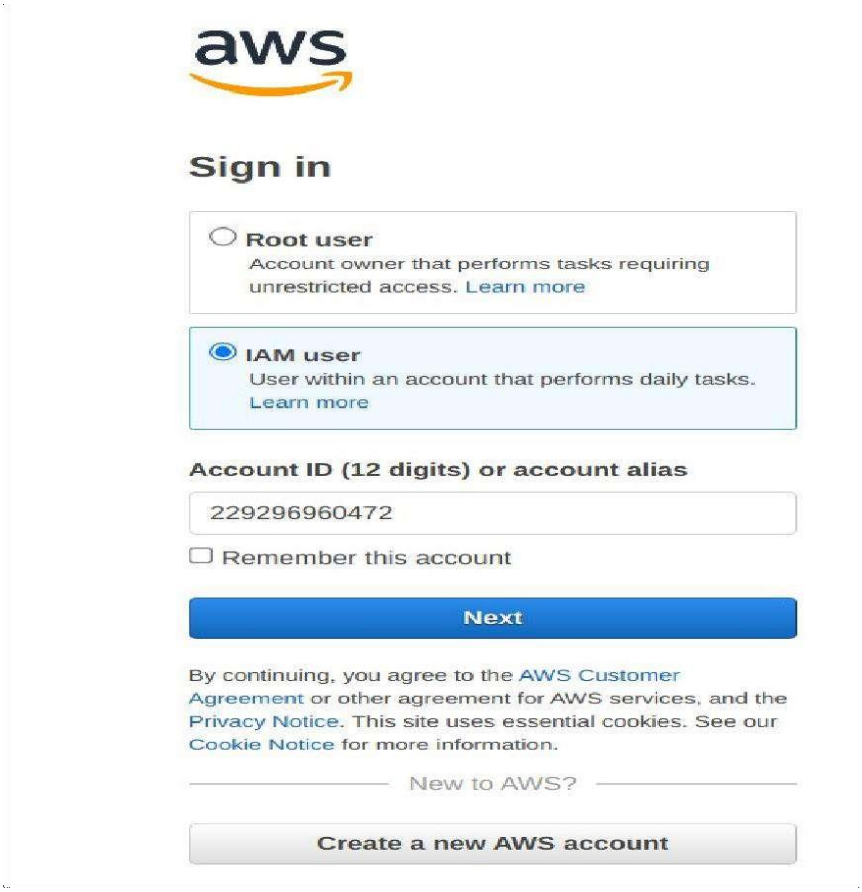
15. Edit html file and save it

16. now in order to share this file to collaborate with other members of your team click on Share option on Roght Pane and username which you created in IAM before into Invite members and enable persmission as RW (Read and Write) and click on Done. Click OK for Security warning.

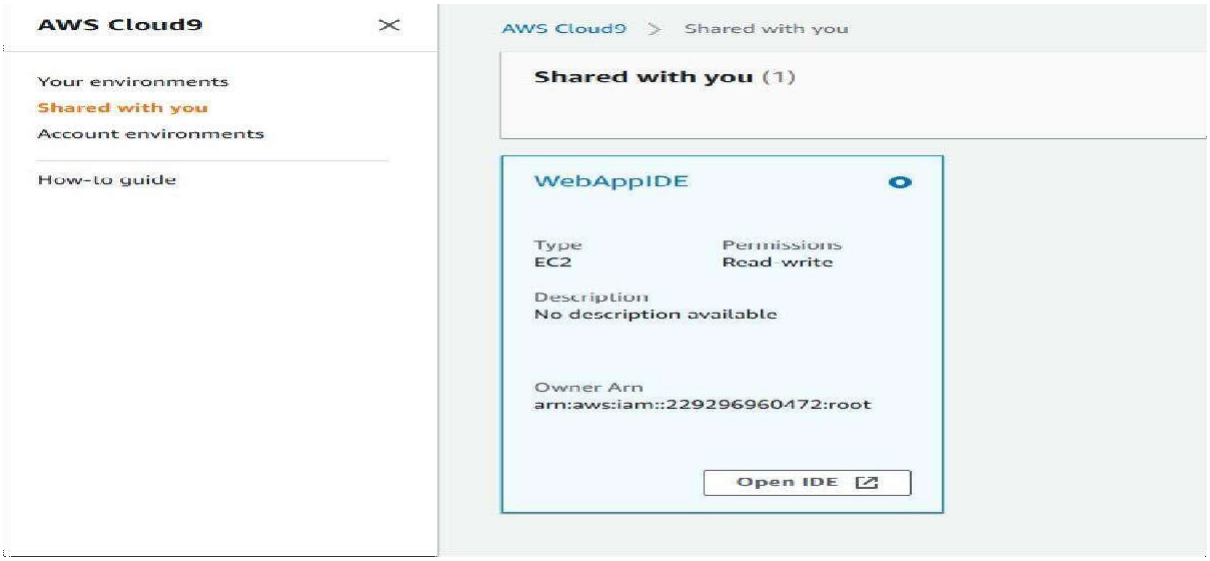




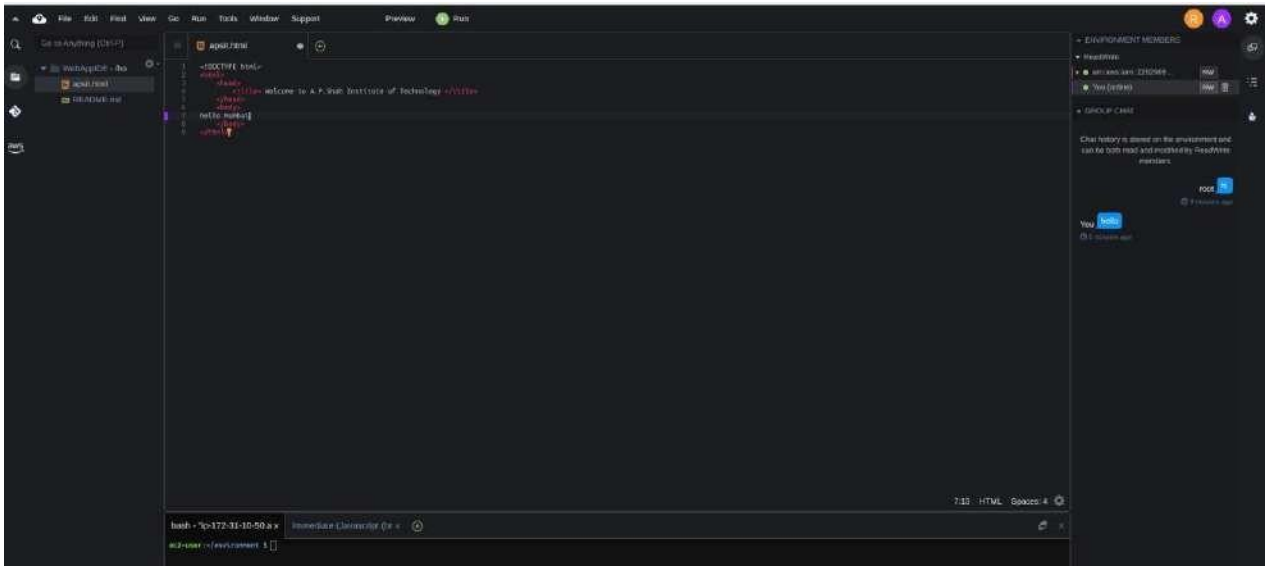
17. Now Open your Browsers Incognito Window and login with IAM user which you configured before.

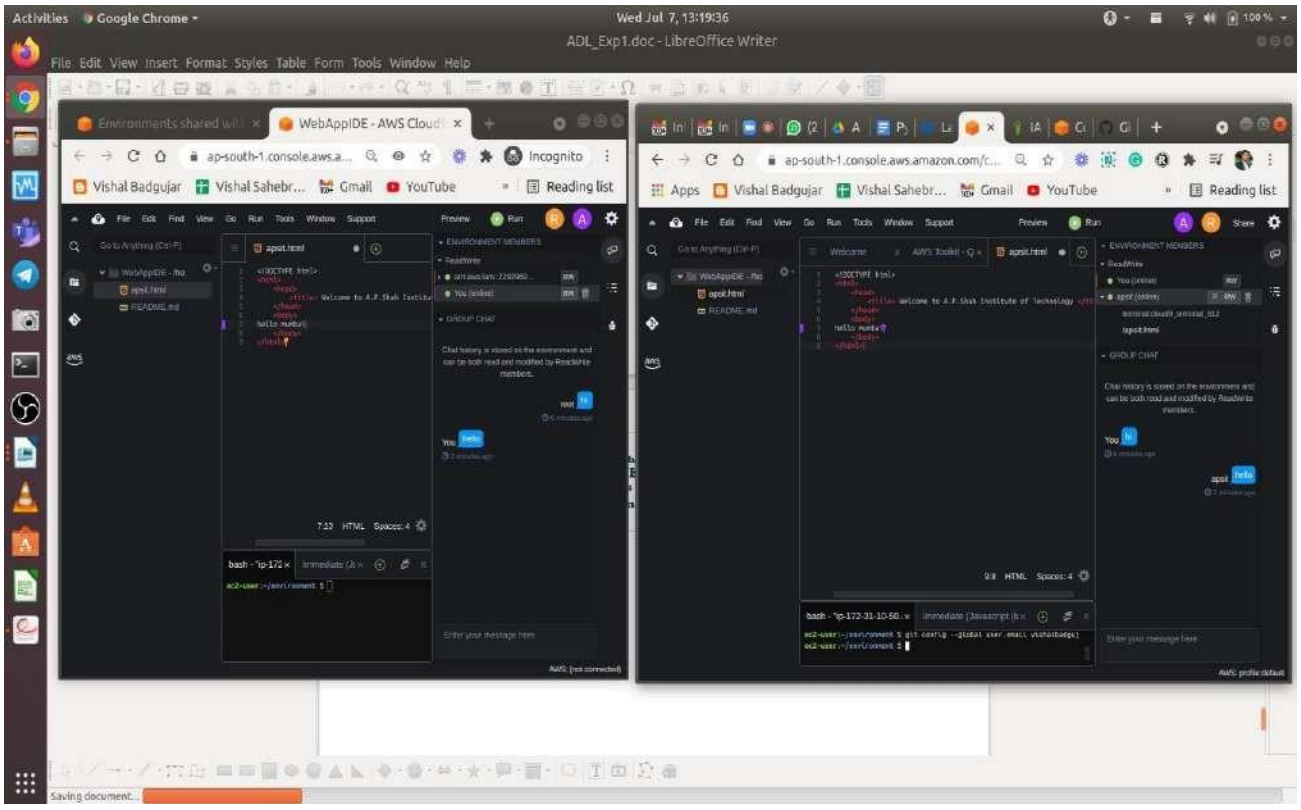


18. After Successful login with IAM user open Cloud9 service from dashboard services and click on shared with you enviornment to collaborate.

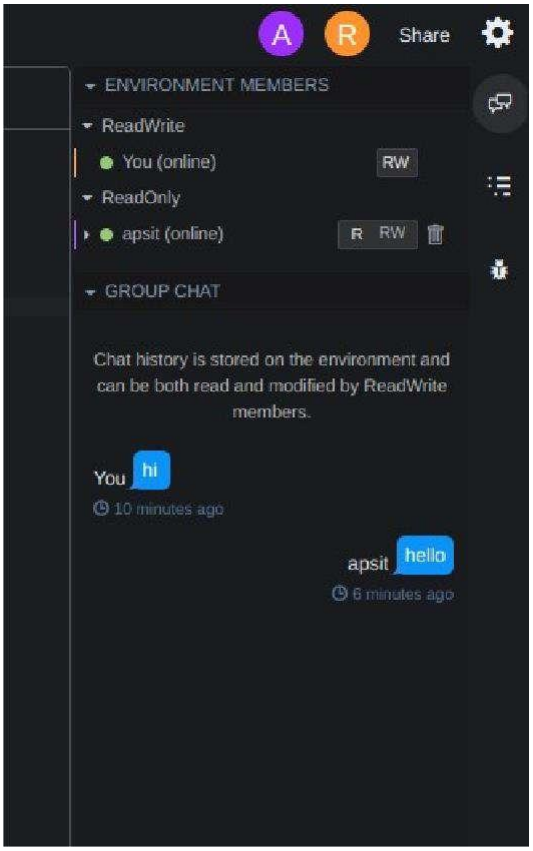


19. Click on Open IDE you will same interface as your other member have to collaborate in real time, also you all within team can do group chats as shown below:




Activities
Google Chrome
Wed Jul 7, 13:19:36
ADL_Exp1.doc - LibreOffice Writer
File Edit View Insert Format Styles Table Form Tools Window Help
WebAppIDE - AWS Cloud
Incognito
Vishal Badgujar
Vishal Sahebr...
Gmail
YouTube
Reading list
Apps
apost.html
README.md
GROUP CHAT
Saving document...

24. you can also explore settings where you can update permissions of your temmates as fromRW to R only or you can remove user too.



For more info related to AWS-Cloud 9 you all can refer following Docs.

https://docs.aws.amazon.com/cloud9/latest/user-guide/aws-cloud9-ug.pdf

Conclusion: Write your own findings.

Academic Year: 2023-24 **Semester: V**
Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof. Manjusha Kashilkar/Prof.Sonal Jain/Prof.Yaminee Patil

EXPERIMENT NO. 02

Aim: To Build Your Application using AWS CodeBuild and Deploy on S3 / SEBS using AWS CodePipeline, deploy Sample Application on EC2 instance using AWS CodeDeploy.

Theory:

Continuous deployment allows you to deploy revisions to a production environment automatically without explicit approval from a developer, making the entire software release process automated.

You will create the pipeline using AWS CodePipeline, a service that builds, tests, and deploys your code every time there is a code change. You will use your GitHub account, an Amazon Simple Storage Service (S3) bucket, or an AWS CodeCommit repository as the source location for the sample app's code. You will also use AWS Elastic Beanstalk as the deployment target for the sample app. Your completed pipeline will be able to detect changes made to the source repository containing the sample app and then automatically update your live sample app.

Step1: Create a deployment environment

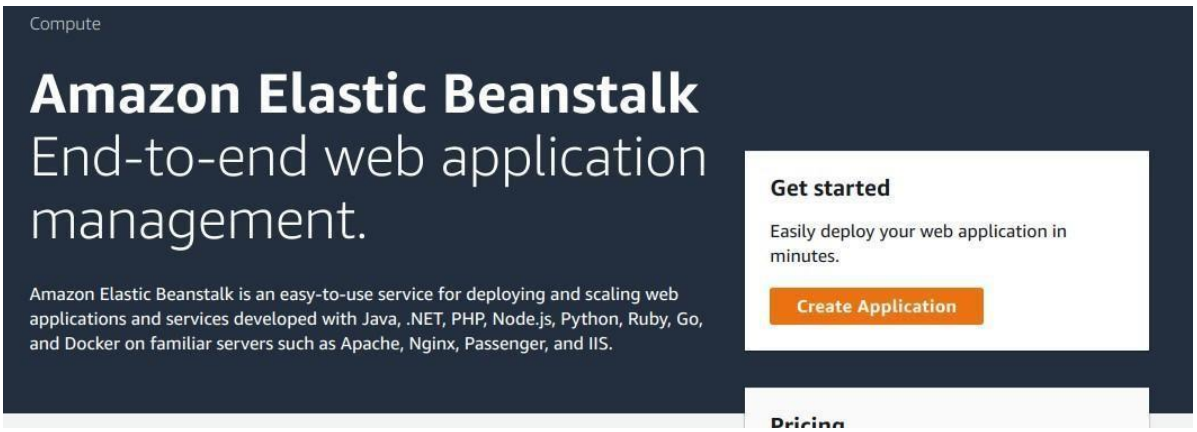
Your continuous deployment pipeline will need a target environment containing virtual servers, or Amazon EC2 instances, where it will deploy sample code. You will prepare this environment before creating the pipeline.

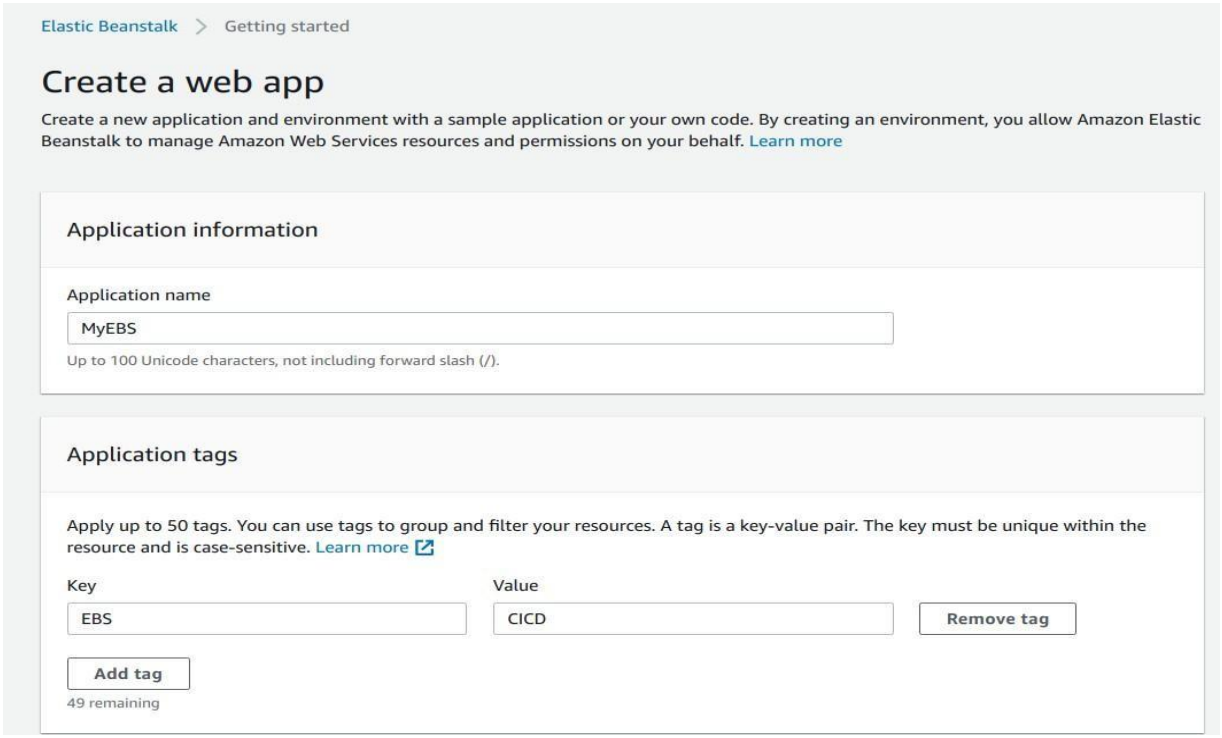
1. To simplify the process of setting up and configuring EC2 instances for this tutorial, you will spin up a sample environment using AWS Elastic Beanstalk. Elastic Beanstalk lets you easily host web applications without needing to launch, configure, or operate virtual servers on your own. It automatically provisions and operates the infrastructure (e.g. virtual servers, load balancers, etc.) and provides the application stack (e.g. OS, language and framework, web and application

server, etc.) for you.

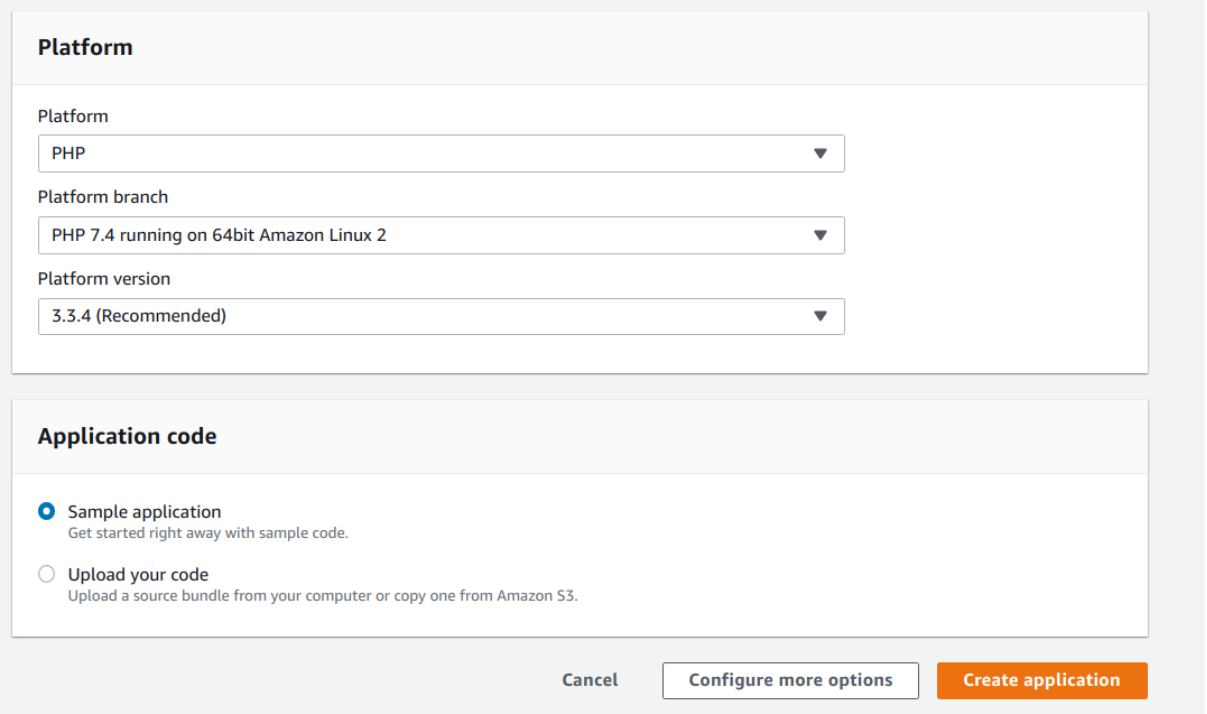
Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE
(All Programs Accredited by NBA)
Department of Information Technology

2. Name your web app and choose PHP from the drop-down menu(or any other language you are interested in) and then click Create Application.





Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE
(All Programs Accredited by NBA)
Department of Information Technology



3. Elastic Beanstalk will begin creating a sample environment for you to deploy your application to. It will create an Amazon EC2 instance, a security group, an Auto Scaling group, an Amazon S3 bucket, Amazon CloudWatch alarms, and a domain name for your application.

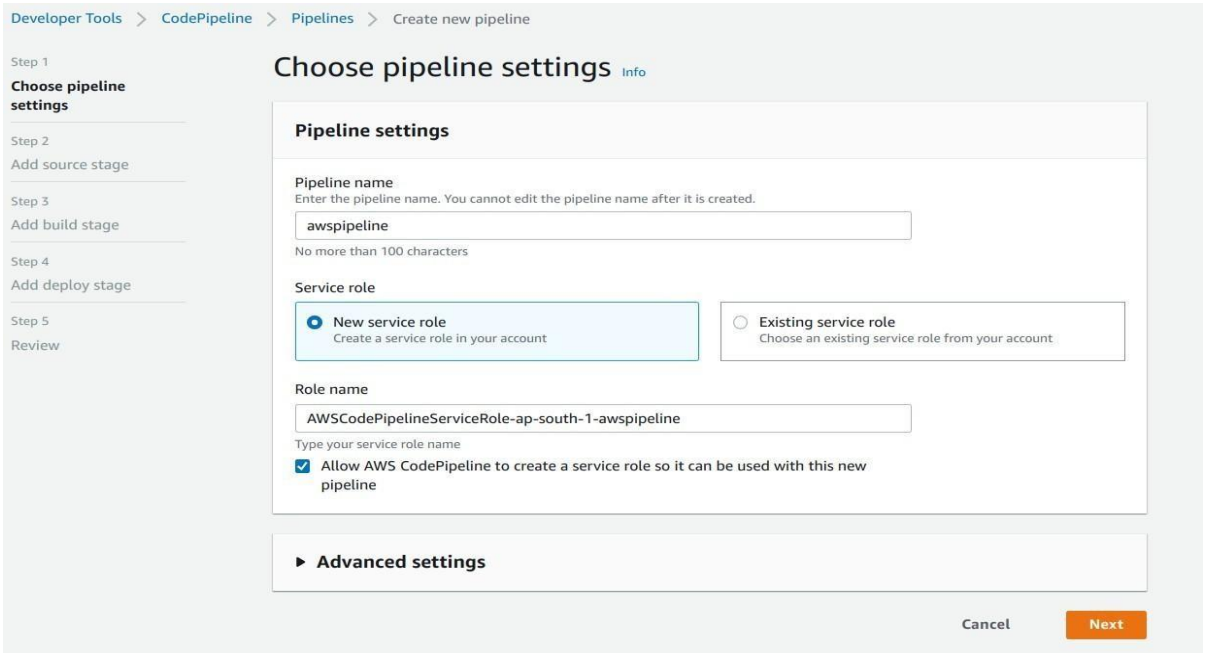
Note: This will take several minutes to complete.

Step2: Get a copy of the sample code

In this step, you will retrieve a copy of the sample app's code and choose a source to host the code. The pipeline takes code from the source and then performs actions on it.

You can use one of three options as your source: a GitHub repository, an Amazon S3 bucket, or an AWS CodeCommit repository. Select your preference and follow the steps

below:



a. If you plan to use Amazon S3 as your source, you will retrieve the sample code from the AWS GitHub repository, save it to your computer, and upload it to an Amazon S3 bucket.

- Visit our GitHub repository containing the sample code at

  https://github.com/imoisharma/aws-codepipeline-s3-codedeploy-linux-2.0

- Click the dist folder.

b. Save the source files to your computer:

- Click the file named aws-codepipeline-s3-aws-codedeploy_linux.zip
- Click View Raw.
- Save the sample file to your local computer.

c. open the Amazon S3 console and create your Amazon S3 bucket:

- Click Create Bucket
- Bucket Name: type a unique name for your bucket, such as awscodepipeline-demobucket- variables. All bucket names in Amazon S3 must be unique, so use one of your own, not one with the name shown in the example.
- Region: In the drop-down, select the region where you will create your pipeline, such as ap- South-1
- Click Create.

d. The console displays the newly created bucket, which is empty.

- Click Properties.

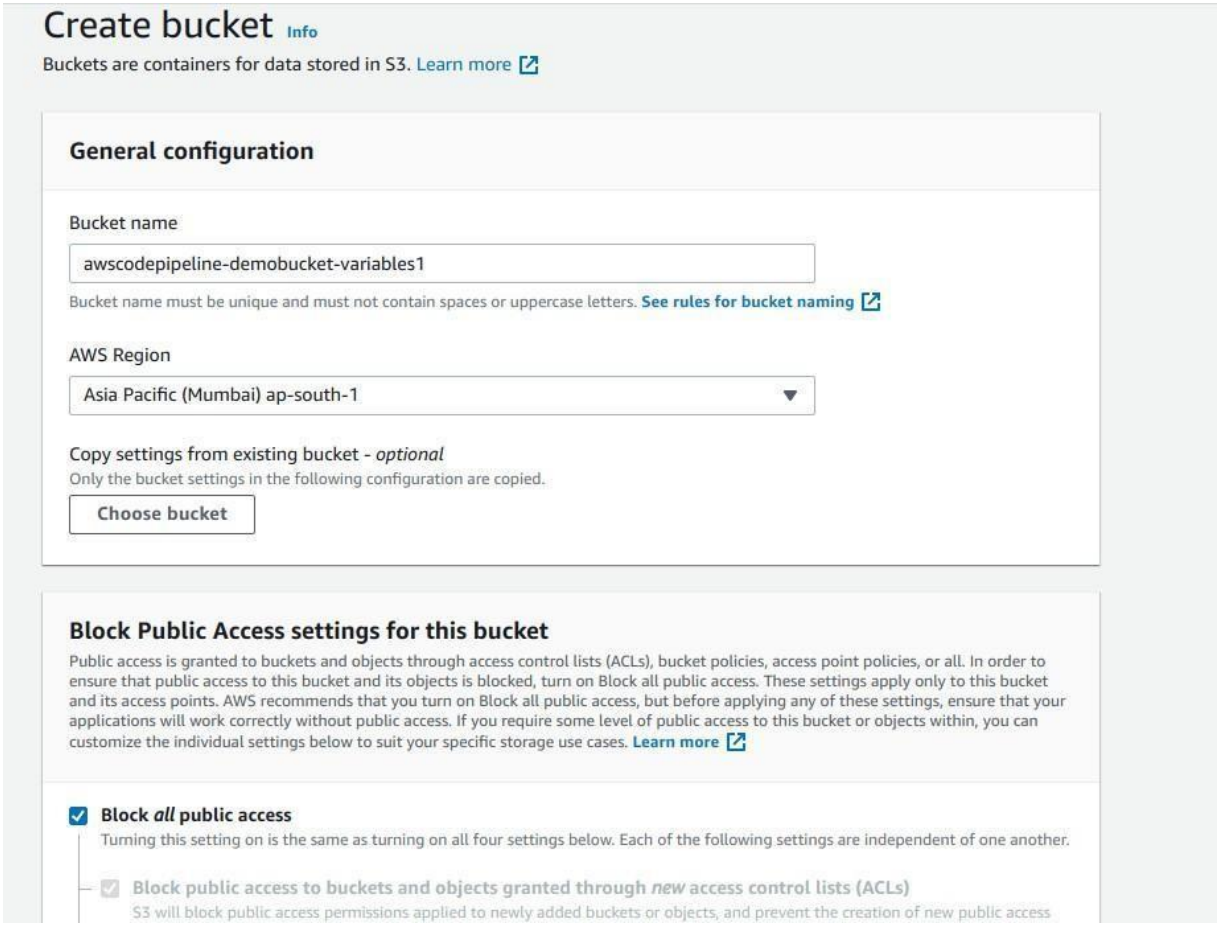
- Expand Versioning and select Enable Versioning. When versioning is enabled, Amazon S3 saves every version of every object in the bucket.

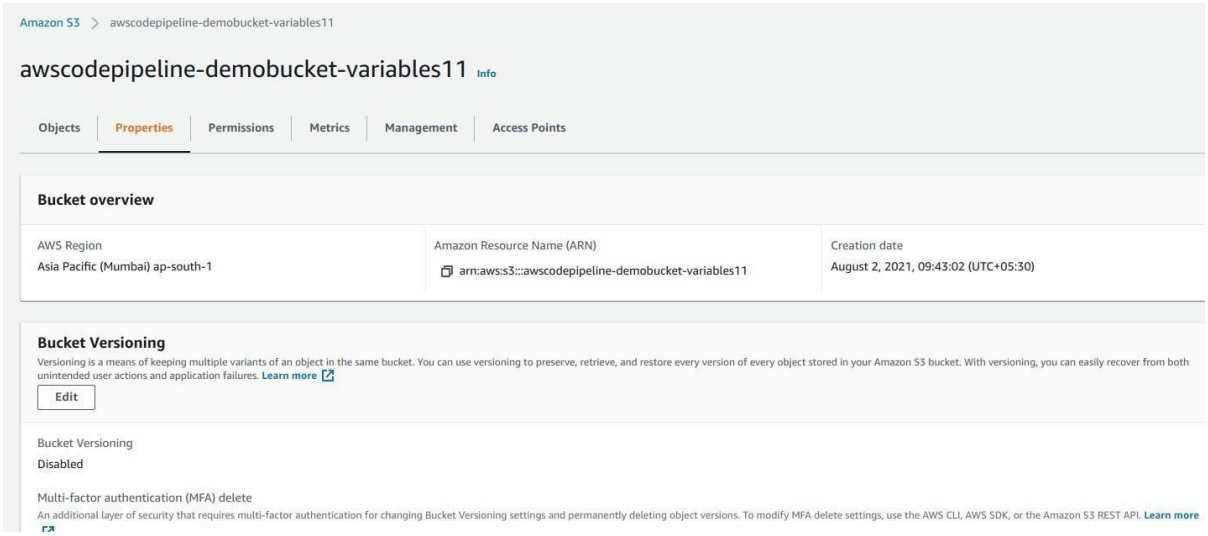
e. You will now upload the sample code to the Amazon S3 bucket:

- Click Upload.

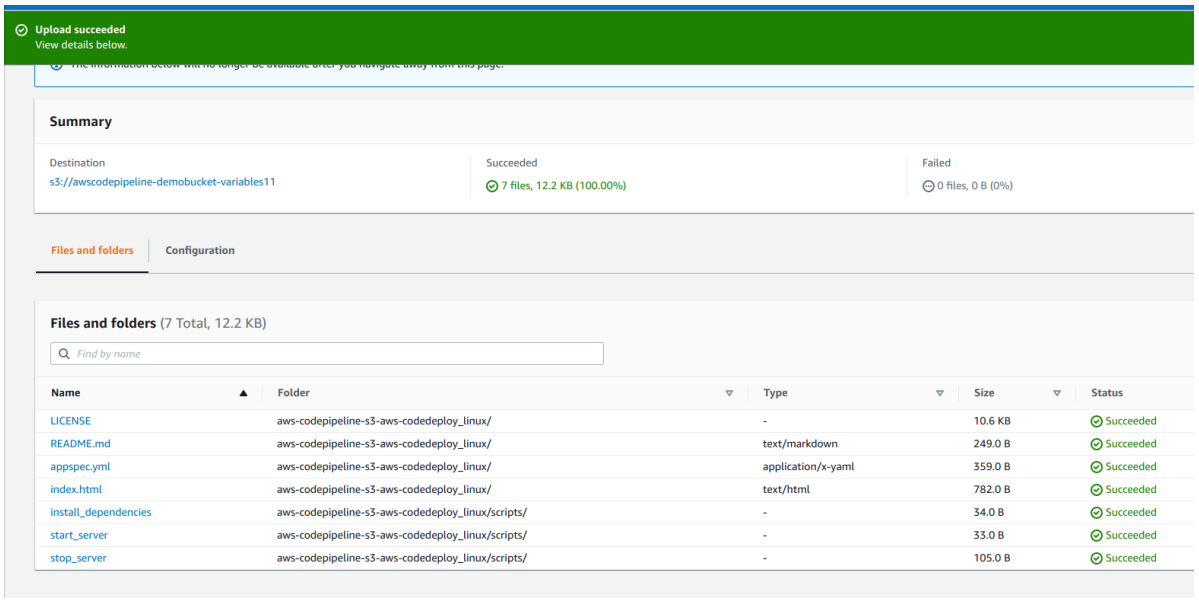
- Follow the on-screen directions to upload the .zip file containing the sample code you downloaded from GitHub.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE
(All Programs Accredited by NBA)
Department of Information Technology


Create bucket Info
Buckets are containers for data stored in S3. Learn more
General configuration
Bucket name
awscodepipeline-demobucket-variables1
Bucket name must be unique and must not contain spaces or uppercase letters. See rules for bucket naming
AWS Region
Asia Pacific (Mumbai) ap-south-1
Copy settings from existing bucket - optional
Only the bucket settings in the following configuration are copied.
Choose bucket
Block Public Access settings for this bucket
Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more
Block all public access
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.
Block public access to buckets and objects granted through new access control lists (ACLs)
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access



you can upload directly zip file here from https://github.com/imoisharma/aws-codepipeline- s3-codedeploy-linux-2.0

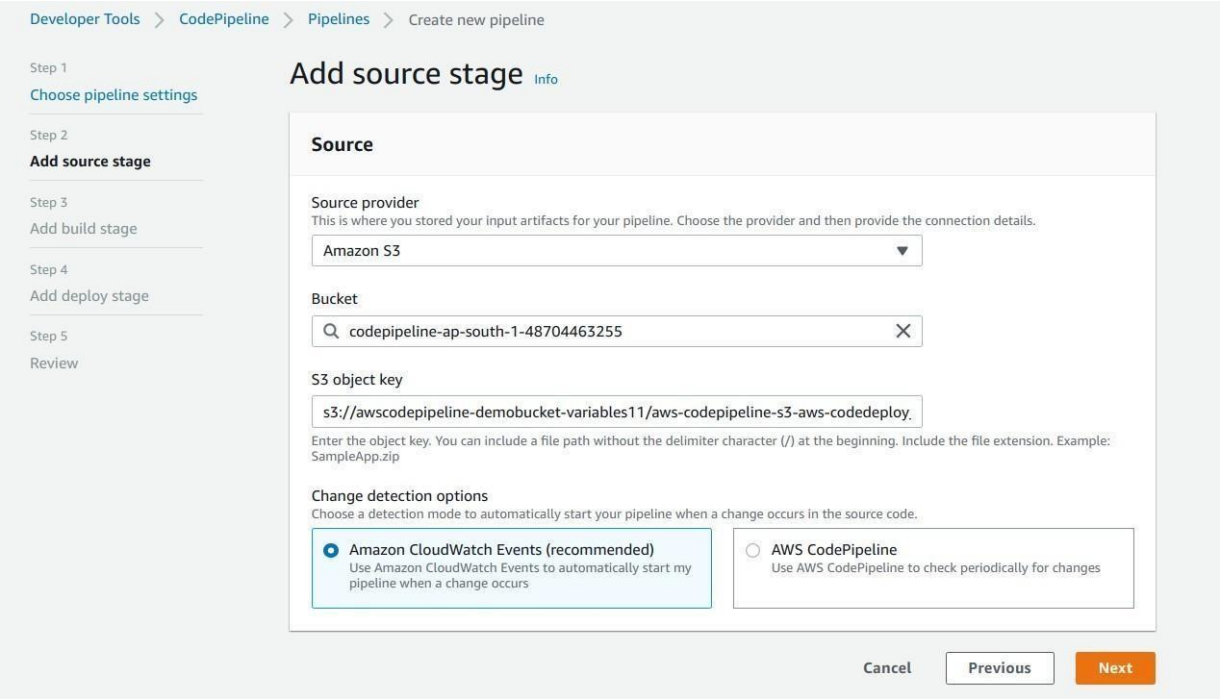


Step3: Create your Pipeline

In this step, you will create and configure a simple pipeline with two actions: source and deploy. You will provide CodePipeline with the locations of your source repository and deployment environment.

A true continuous deployment pipeline requires a build stage, where code is compiled and unit tested. CodePipeline lets you plug your preferred build provider into your pipeline. However, in this we will skip the build stage.

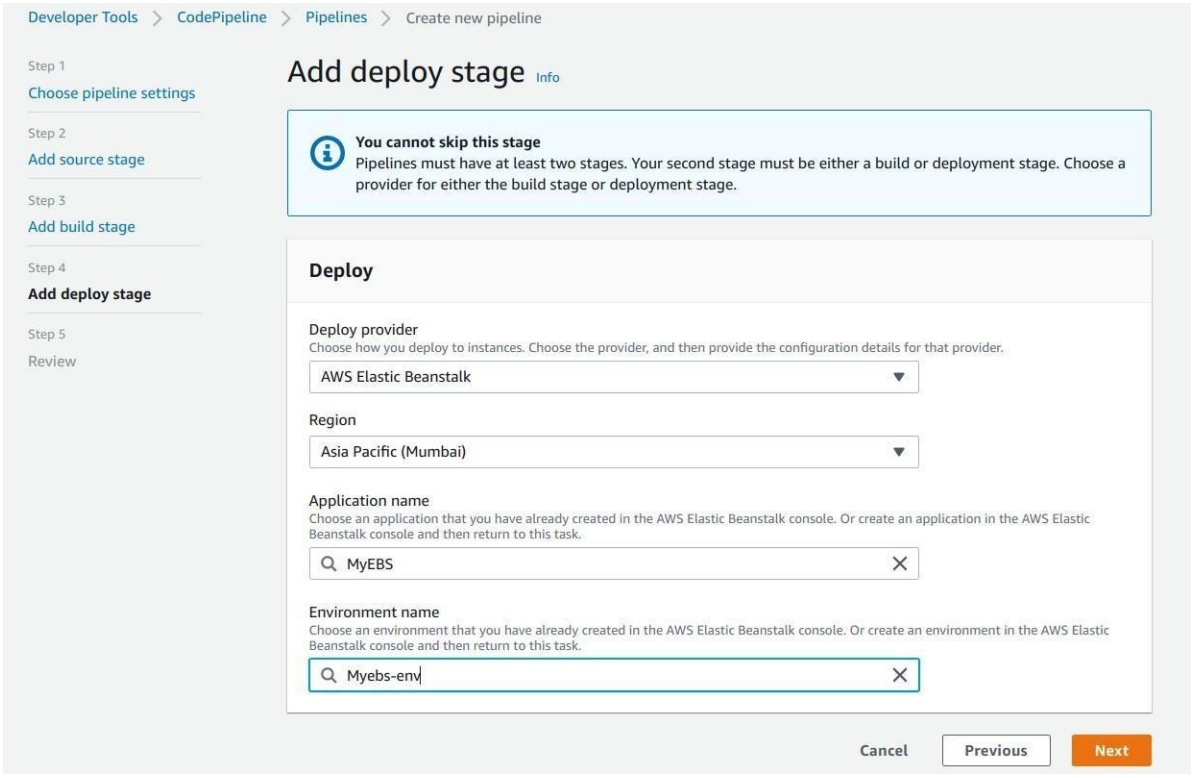
Goto Pipeline again and create it

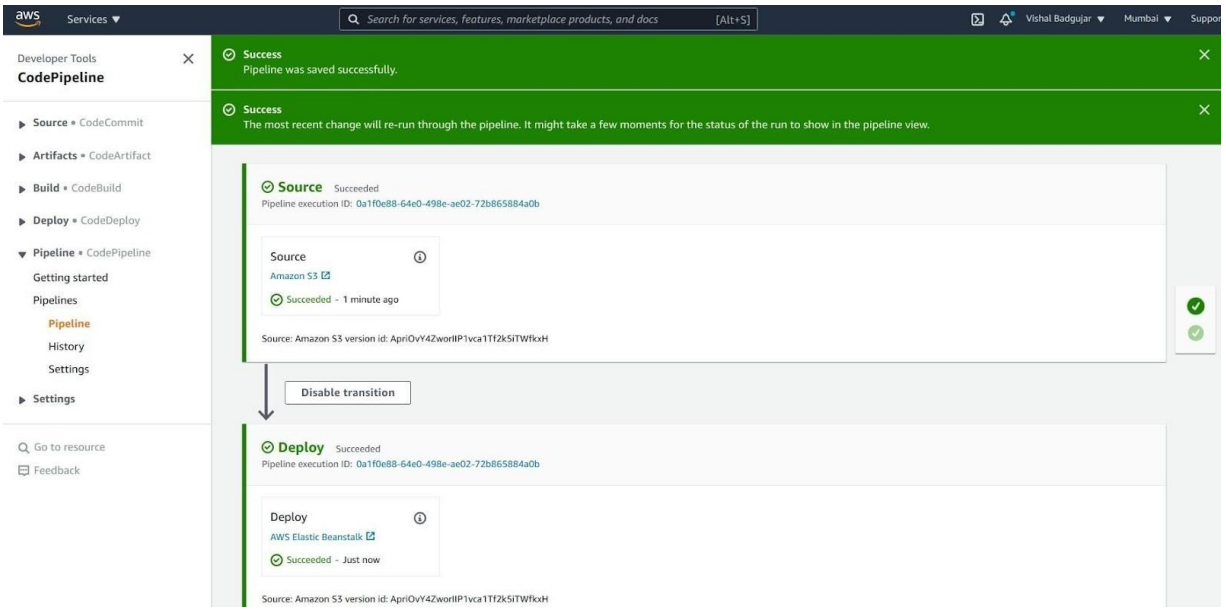


In above you can give zip file name in S3 object Key and choose bucket name which you created

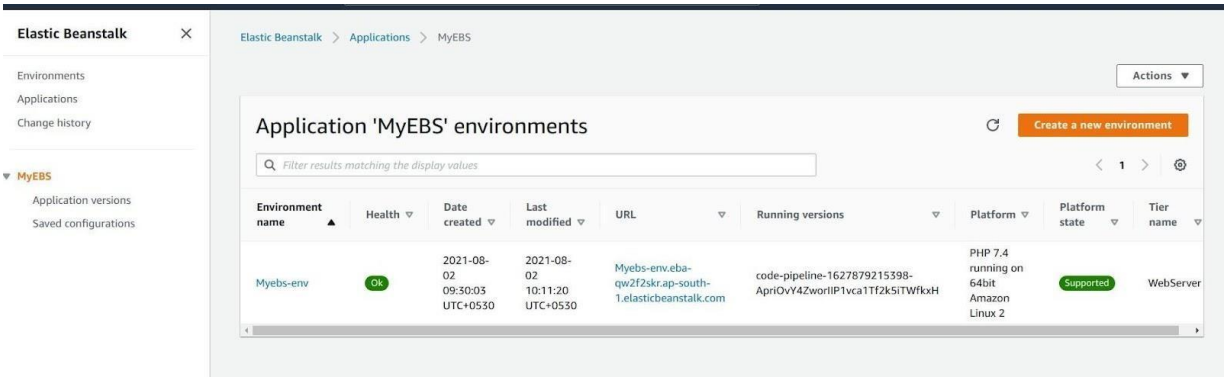
In Step 4: Deploy Stage:

- Deployment provider: Click AWS Elastic Beanstalk.
- Application name: MYEBS.
- Environment name: Click Myebs-env.


Developer Tools > CodePipeline > Pipelines > Create new pipeline
Step 1
Choose pipeline settings
Step 2
Add source stage
Step 3
Add build stage
Step 4
Add deploy stage
Step 5
Review
Add deploy stage Info
You cannot skip this stage
Pipelines must have at least two stages. Your second stage must be either a build or deployment stage. Choose a provider for either the build stage or deployment stage.
Deploy
Deploy provider
Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.
AWS Elastic Beanstalk
Region
Asia Pacific (Mumbai)
Application name
Choose an application that you have already created in the AWS Elastic Beanstalk console. Or create an application in the AWS Elastic Beanstalk console and then return to this task.
MyEBS
Environment name
Choose an environment that you have already created in the AWS Elastic Beanstalk console. Or create an environment in the AWS Elastic Beanstalk console and then return to this task.
Myebs-env
Cancel
Previous
Next

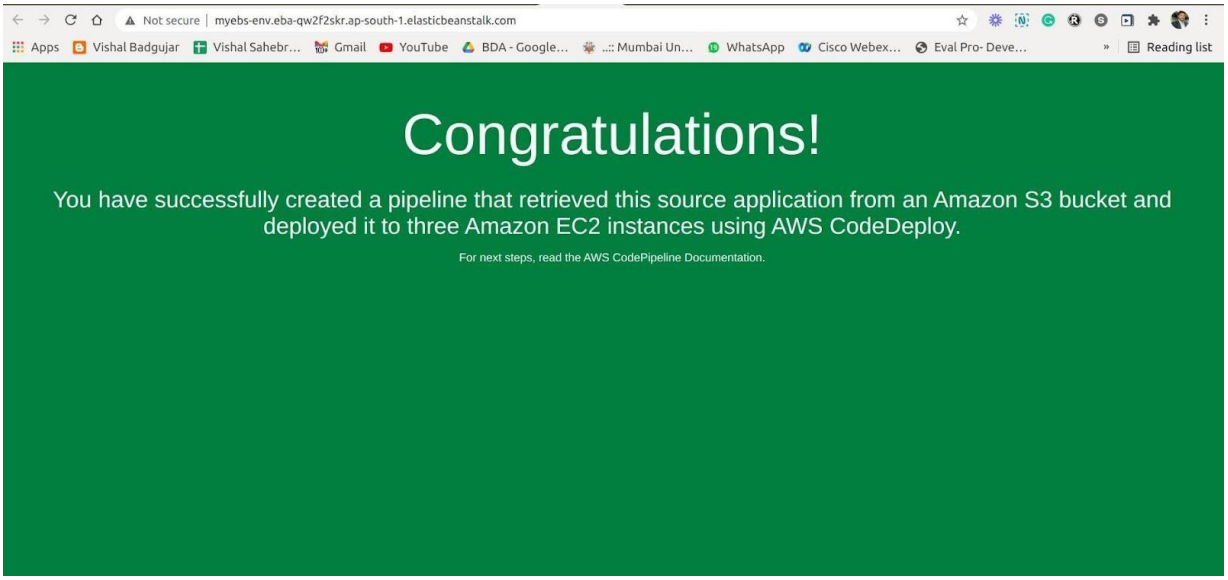


Now go to your EBS environment and click on the URL to view the sample website you deployed.



You have successfully created an automated software release pipeline using AWS CodePipeline!

Using CodePipeline, you created a pipeline that uses GitHub, Amazon S3, or AWS CodeCommit as the source location for application code and then deploys the code to an Amazon EC2 instance managed by AWS Elastic Beanstalk.



Step 5: Commit a change and then update your app

In this step, you will revise the sample code and commit the change to your repository. CodePipeline will detect your updated sample code and then automatically initiate deploying it to your EC2 instance via Elastic Beanstalk.

Note that the sample web page you deployed refers to AWS CodeDeploy, a service that automates code deployments. In CodePipeline, CodeDeploy is an alternative to using Elastic Beanstalk for deployment actions. Let's update the sample code so that it correctly states that you deployed the sample using Elastic Beanstalk.

a. Visit your own copy of the repository that you forked in GitHub.

- Open index.html

- Select the Edit icon

b. Update the webpage by copying and pasting the following text on line 30:

c. Commit the change to your repository.

d. Return to your pipeline in the CodePipeline console. In a few minutes, you should see the Source change to blue, indicating that the pipeline has detected the changes you made to your source repository. Once this occurs, it will automatically move the updated code to Elastic Beanstalk.

- After the pipeline status displays Succeeded, in the status area for the Beta stage, click AWS Elastic Beanstalk.

e. The AWS Elastic Beanstalk console opens with the details of the deployment. Select the environment you created earlier. And click the URL again from EBS environment again.

Step 6: Clean up your resources

To avoid future charges, you will delete all the resources you launched throughout this tutorial, which includes the pipeline, the Elastic Beanstalk application, and the source you set up to host the code.

a. First, you will delete your pipeline:

- In the pipeline view, click Edit.
- Click Delete.
- Type in the name of your pipeline and click Delete.

b. Second, delete your Elastic Beanstalk application:

- Visit the Elastic Beanstalk console.

- Click Actions.

- Then click Terminate Environment.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE
(All Programs Accredited by NBA)
Department of Information Technology

You have successfully created an automated software release pipeline using AWS CodePipeline! Using CodePipeline, you created a pipeline that uses GitHub, Amazon S3, or AWS CodeCommit as the source location for application code and then deploys the code to an Amazon EC2 instance managed by AWS Elastic Beanstalk. Your pipeline will automatically deploy your code every time there is a code change.

Conclusion: Write your own findings.

Academic Year: 2023-24 **Semester: V**
Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof. Manjusha Kashilkar/Prof.Sonal Jain/Prof.Yaminee Patil

EXPERIMENT NO. 3

Aim: To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.

Theory:

Prerequisites

Launch two or more Linux servers running Ubuntu 18.04 /20.04 on Virtual box OR Launch two or more EC2instances of Ubuntu 20.04 AMI free tier.

If using EC2 instances, connect to all instances using PUTTY (on Windows) or usingSSH (on Linux/Ubuntu).

Assign Unique Hostname for Each Server Node

$ sudo hostnamectl set-hostname master-node

Next, set a worker node hostname by entering the following on the worker server:

$ sudo hostnamectl set-hostname worker-node-01

Steps to Install Kubernetes on UbuntuSet up Docker

Step 1 : Install Docker

Kubernetes requires an existing Docker installation. If you already have Docker installed, skip ahead to Step 2. If you do not have Kubernetes, install it by following these steps: Update the package list with the command:

$ sudo apt-get update

Next, install Docker with the command:

$ sudo apt-get install docker.io

Repeat the process on each server that will act as

a node. Check the installation (and version) by

entering the following:

$ docker —version

Step 2 : Start and Enable Docker

Set Docker to launch at boot by entering the following:

$ sudo systemctl enable docker

$ sudo systemctl status docker

$ sudo systemctl start docker

Install Kubernetes

Step 3 : Add Kubernetes Signing Key

(https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/)

(Please refer above link)

$ sudo apt-get update

$ sudo apt-get install -y apt-transport-https ca-certificates curl

$ sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg

$ echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

$ sudo apt-get update

$ sudo apt-get install -y kubelet kubeadm kubectl

$ sudo apt-mark hold kubelet kubeadm kubectl

Allow the process to complete. Verify the installation with:
$ kubeadm version

Then repeat the previous command to install the signing keys. Repeat for each server node.Kubernetes Deployment

Step 4 : Begin Kubernetes Deployment

$ sudo swapoff –a

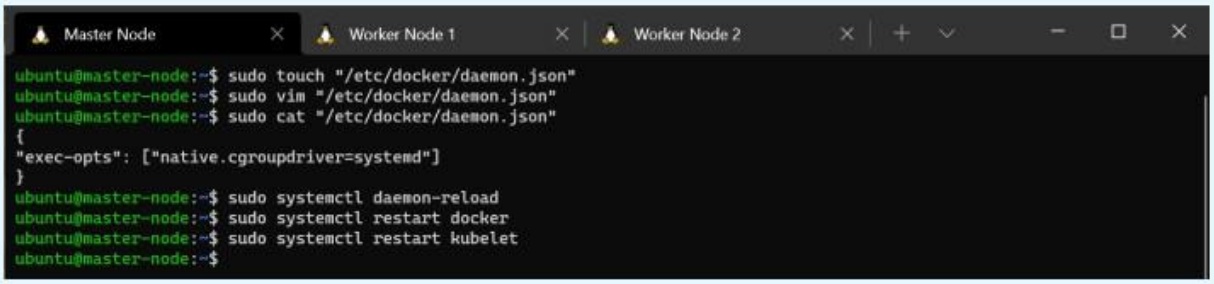
Step 5 : Initialize Kubernetes on Master Node

$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16--ignore-preflight-errors=all

If the kubeadm init command ran without error then ignore this part. If you receive this error "kubelet isn't running or healthy", then do the following.
Create file daemon.json in /etc/docker/ and add following lines in the file.

```
{
   "exec-opts": ["native.cgroupdriver=systemd"]
}
```

And run the following commands.
Do this on both master and worker nodes.



After this run sudo kubeadm reset command and then the init or join command.

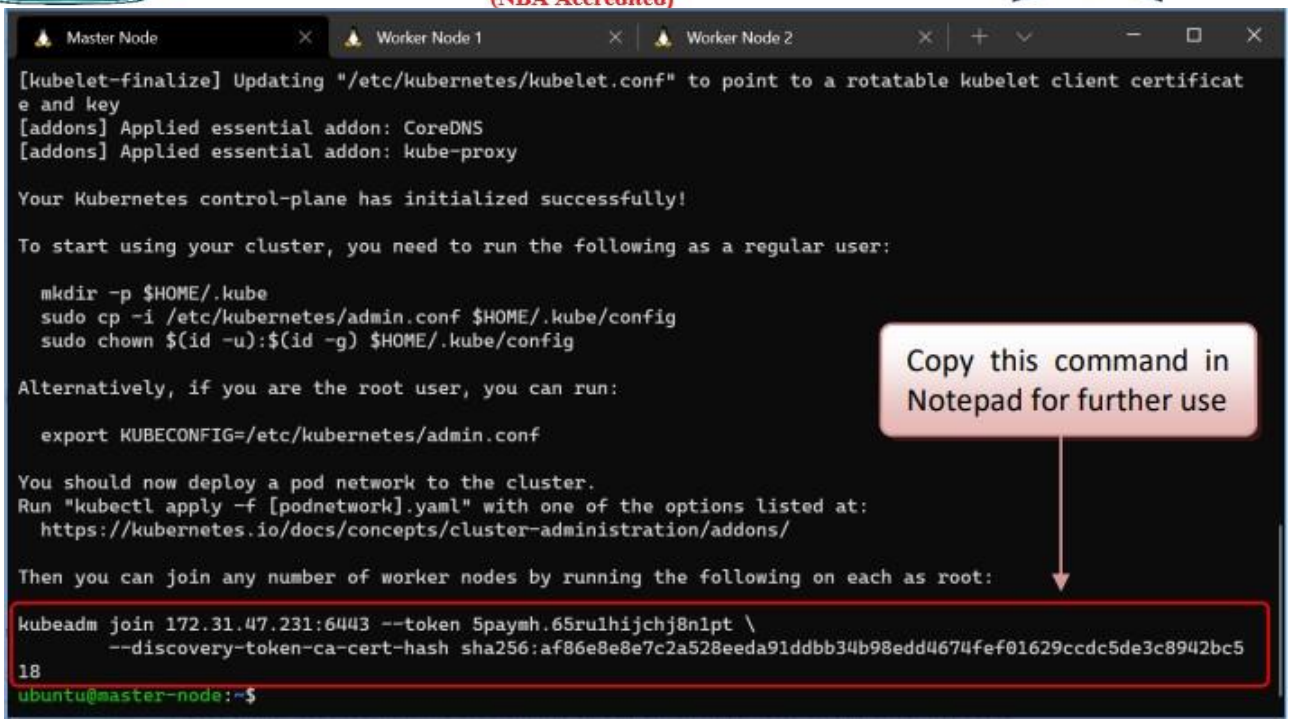
$ sudo touch "/etc/docker/daemon.json"
$ sudo nano "/etc/docker/daemon.json"
$ sudo cat "/etc/docker/daemon.json"

```
{
  "exec-opts": ["native.cgroupdriver=systemd"]
}
```

$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
$ sudo systemctl restart kubelet
$ sudo kubeadm reset

$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16--ignore-preflight-errors=all

Once this command finishes, it will display a kubeadm join message at the end. Make a note of the whole entry. This will be used to join the worker nodes to the cluster.



Next, enter the following to create a directory for the cluster:

kubernetes-master $ mkdir -p $HOME/.kube

kubernetes-master $ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

kubernetes-master $ sudo chown $(id -u):$(id -g) $HOME/.kube/config

Step 6 : Deploy Pod Network to Cluster

A Pod Network is a way to allow communication between different nodes in the cluster.This tutorial uses the flannel virtual network.

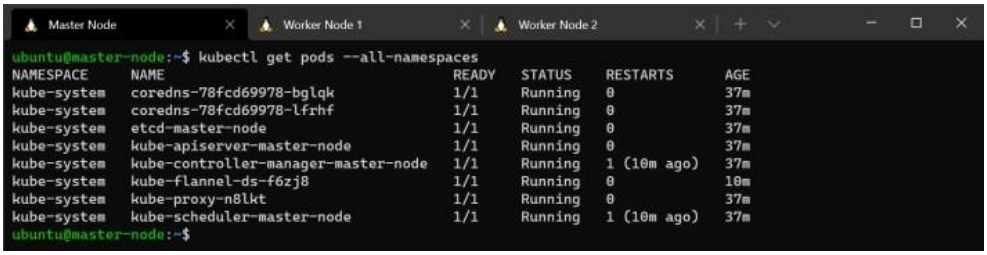
Enter the following:

$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-

flannel.ymlAllow the process to complete.

Verify that everything is running and communicating:

$ kubectl get pods --all-namespaces



Step 7 : Join Worker Node to Cluster

Switch to the worker-node-01 system and enter the command you noted from Step 5: $ kubeadm join -- discovery-token abcdef.1234567890abcdef --discoverytoken-ca-cert-hash sha256:1234..cdef 1.2.3.4:6443Replace the alphanumeric codes with those from your master server.

Repeat for each worker node on the cluster.

Wait a few minutes; then you can check the status of the nodes.

If you are trying to run this on EC2 you'll get an error message saying less CPU and memory to overridethe error run the above command with

--ignore-preflight-errors=all

Switch to the master server, and enter:

$ kubectl get nodes

The system should display the worker nodes that you joined to the cluster.

Conclusion: Write your own findings.

Academic Year: 2023-24 **Semester: V**
Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof. Manjusha Kashilkar

EXPERIMENT NO.4

Aim :To install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy Your First Kubernetes Application.

Theory:

Prerequisites
Launch two or more Linux servers running Ubuntu 18.04 /20.04 on Virtual box OR Launch two or more EC2instances of Ubuntu 20.04 AMI free tier.

If using EC2 instances, connect to all instances using PUTTY (on Windows) or usingSSH (on Linux/Ubuntu).

Assign Unique Hostname for Each Server Node

$ sudo hostnamectl set-hostname master-node

Next, set a worker node hostname by entering the following on the worker server:

$ sudo hostnamectl set-hostname worker-node-01

Steps to Install Kubernetes on Ubuntu
Set up Docker
Step 1 : Install Docker
Kubernetes requires an existing Docker installation. If you already have Docker installed, skip ahead to Step 2. If you do not have Kubernetes, install it by following these steps: Update the package list with the command:

$ sudo apt-get update

Next, install Docker with the command:

$ sudo apt-get install docker.io

Repeat the process on each server that will act as a node. Check

the installation (and version) by entering the following:

$ docker —version

Step 2 : Start and Enable Docker

Set Docker to launch at boot by entering the following:

$ sudo systemctl enable docker

$ sudo systemctl status docker

$ sudo systemctl start docker

Install Kubernetes

Step 3 : Add Kubernetes Signing Key

(https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/)

$ sudo apt-get update

$ sudo apt-get install -y apt-transport-https ca-certificates curl

$ sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg

$ echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

$ sudo apt-get update

$ sudo apt-get install -y kubelet kubeadm kubectl

$ sudo apt-mark hold kubelet kubeadm kubectl

Allow the process to complete. Verify the installation with:

$ kubeadm version

$ echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

$ sudo apt-get update

$ sudo apt-get install -y kubelet kubeadm kubectl

$ sudo apt-mark hold kubelet kubeadm kubectl

Allow the process to complete. Verify the installation with:
$ kubeadm version

Then repeat the previous command to install the signing keys. Repeat for each server node.

Kubernetes Deployment

Step 4 : Begin Kubernetes Deployment

$ sudo swapoff –a

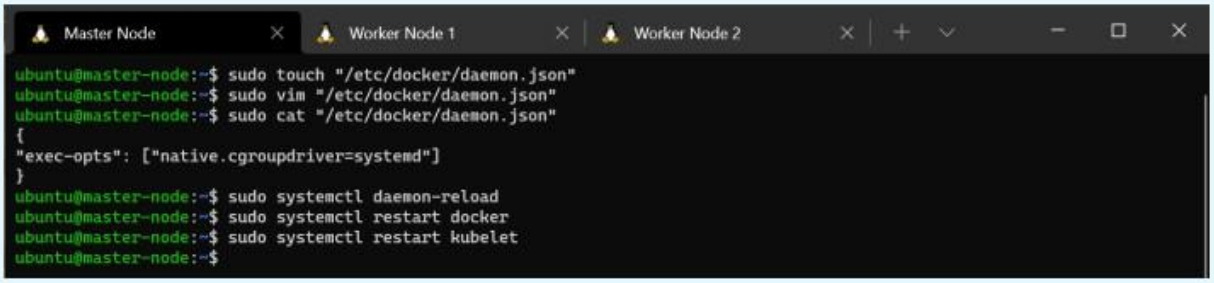
Step 5 : Initialize Kubernetes on Master Node

$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16--ignore-preflight-errors=all

If the kubeadm init command ran without error then ignore this part. If you receive this error "kubelet isn't running or healthy", then do the following.
Create file daemon.json in /etc/docker/ and add following lines in the file.

```
{
   "exec-opts": ["native.cgroupdriver=systemd"]
}
```

And run the following commands.
Do this on both master and worker nodes.

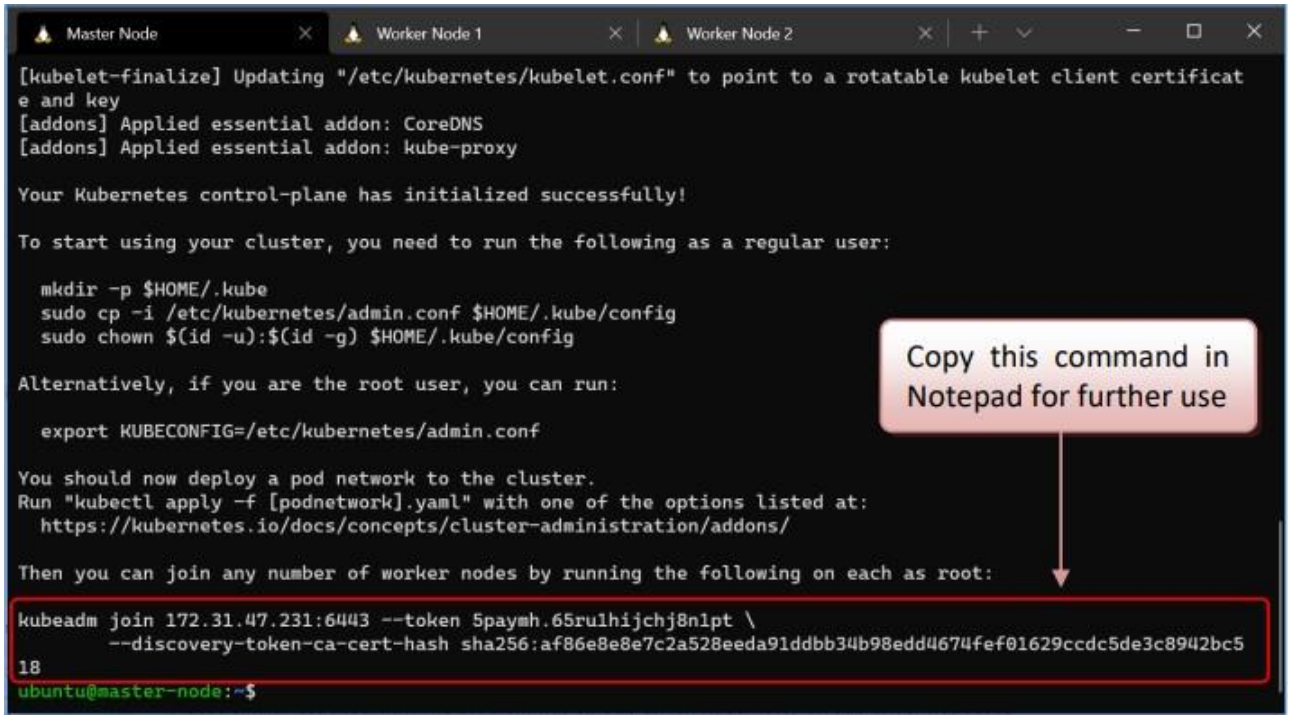


After this run sudo kubeadm reset command and then the init or join command.

```
$ sudo touch "/etc/docker/daemon.json"
 $ sudo nano "/etc/docker/daemon.json"
$ sudo cat "/etc/docker/daemon.json"
{
  "exec-opts": ["native.cgroupdriver=systemd"]
}
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
$ sudo systemctl restart kubelet
$ sudo kubeadm reset
```

```
$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16--ignore-preflight-errors=all
```

Once this command finishes, it will display a kubeadm join message at the end. Make a note of the whole entry. This will be used to join the worker nodes to the cluster.



Next, enter the following to create a directory for the cluster:

kubernetes-master $ mkdir -p $HOME/.kube

kubernetes-master $ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

kubernetes-master $ sudo chown $(id -u):$(id -g) $HOME/.kube/config

Step 6 : Deploy Pod Network to Cluster

A Pod Network is a way to allow communication between different nodes in the cluster.This tutorial uses the flannel virtual network.

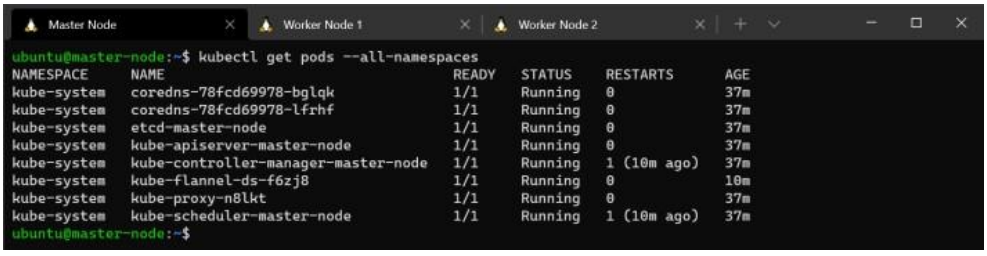
Enter the following:

$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-

flannel.ymlAllow the process to complete.

Verify that everything is running and communicating:

$ kubectl get pods --all-namespaces



Step 7 : Join Worker Node to Cluster

Switch to the worker-node-01 system and enter the command you noted from
Step 5: $ kubeadm join -- discovery-token abcdef.1234567890abcdef --discoverytoken-ca-cert-hash sha256:1234..cdef 1.2.3.4:6443Replace the alphanumeric codes with those from your master server.
Repeat for each worker node on the cluster.
Wait a few minutes; then you can check the status of the nodes.
If you are trying to run this on EC2 you'll get an error message saying less CPU and memory to override the error run the above command with
--ignore-preflight-errors=all

Switch to the master server, and enter:

$ kubectl get nodes

The system should display the worker nodes that you joined to the cluster.

Conclusion -Hence we have studied how to install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy Your First Kubernetes Application.

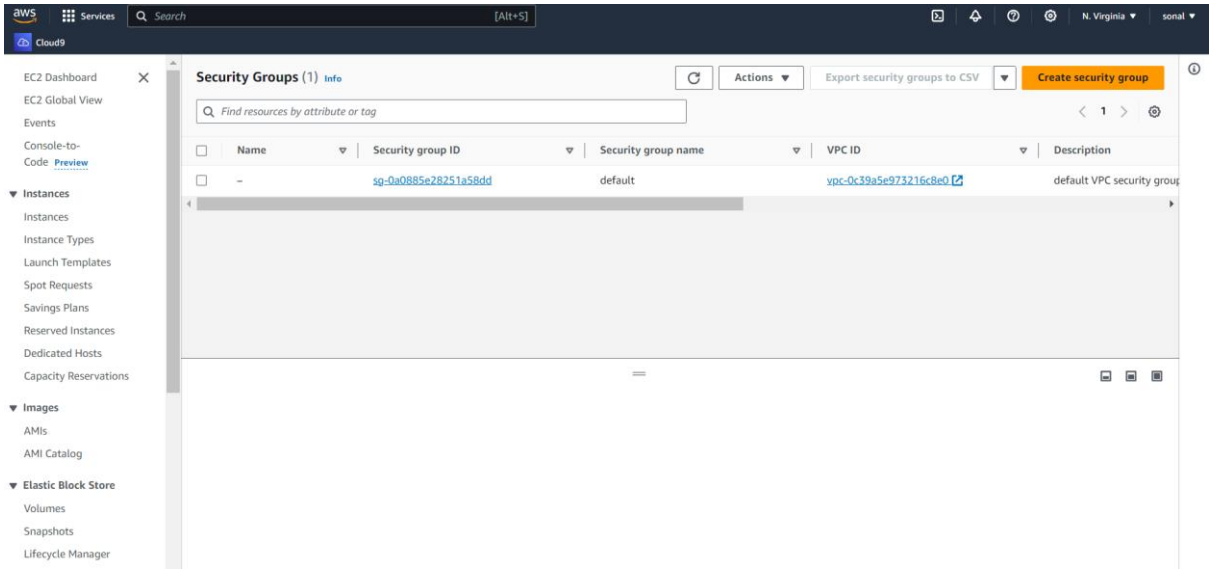
EXPERIMENT NO. 03

Aim: To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.

Aim: To install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy YourFirst Kubernetes

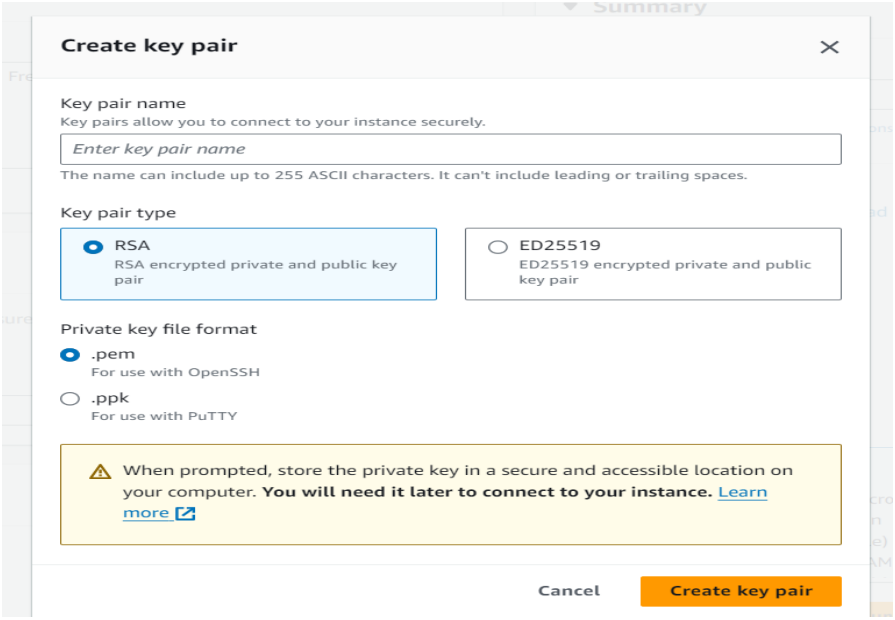
STEP 1:

Check security group,delete all SG only keep default

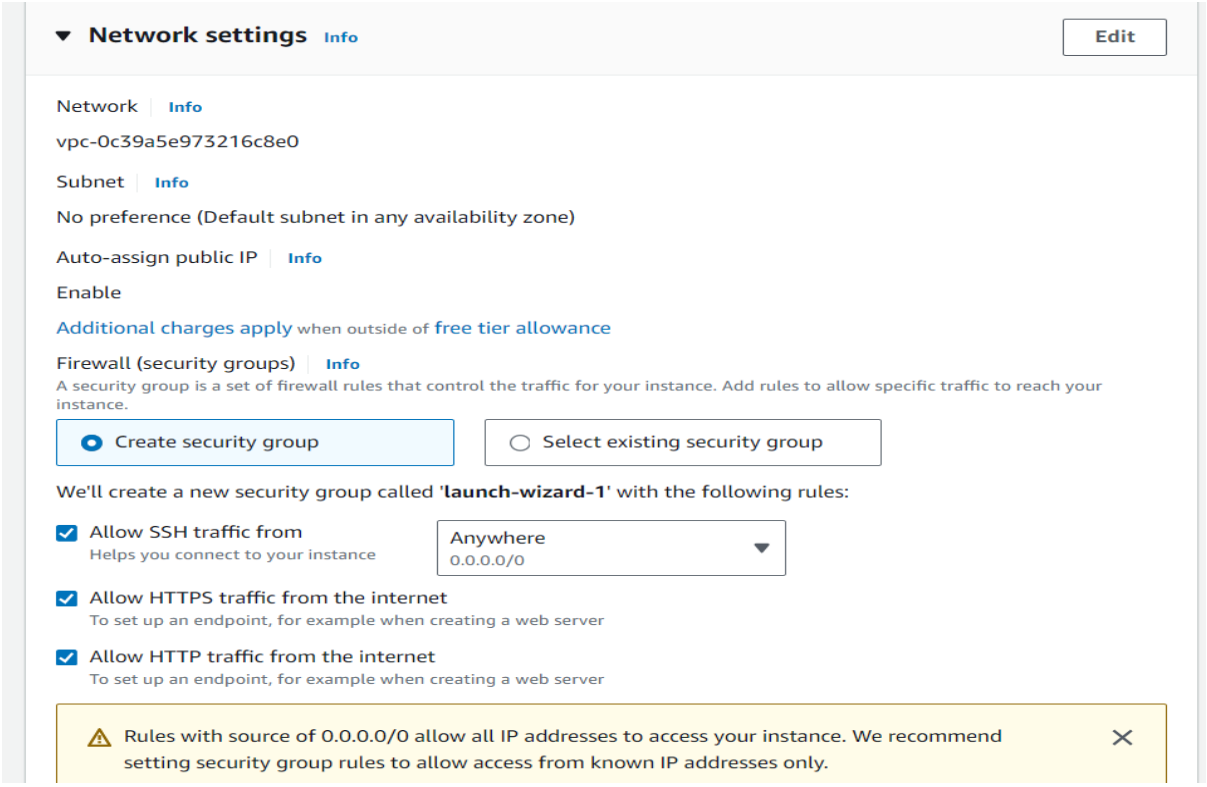


Create 2 instance

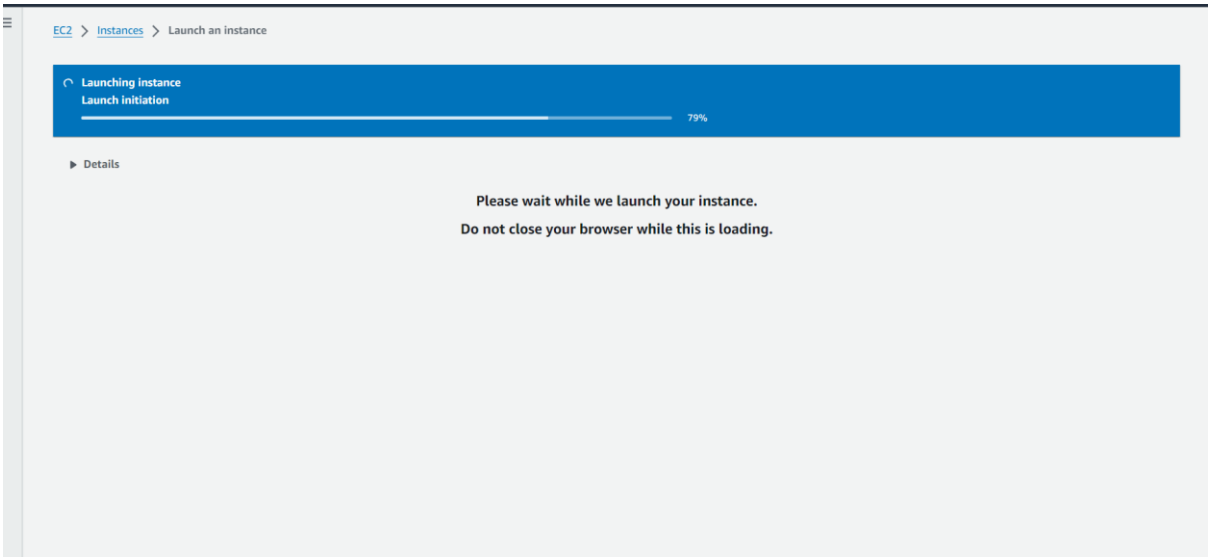
Create key pair



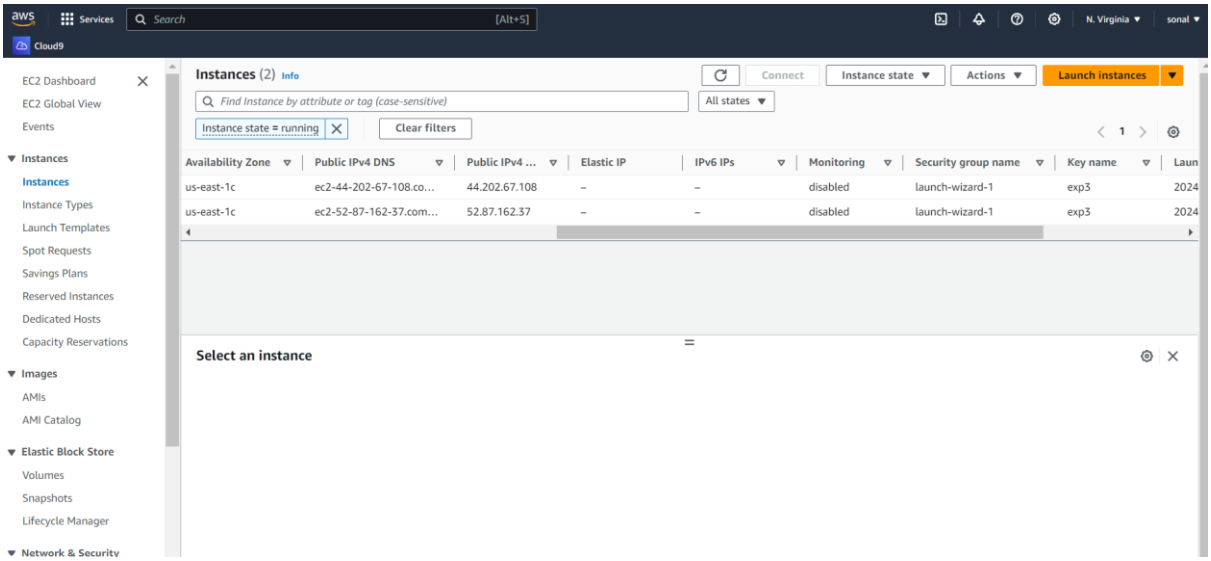
Create security group and allow traffic



Launch Instance



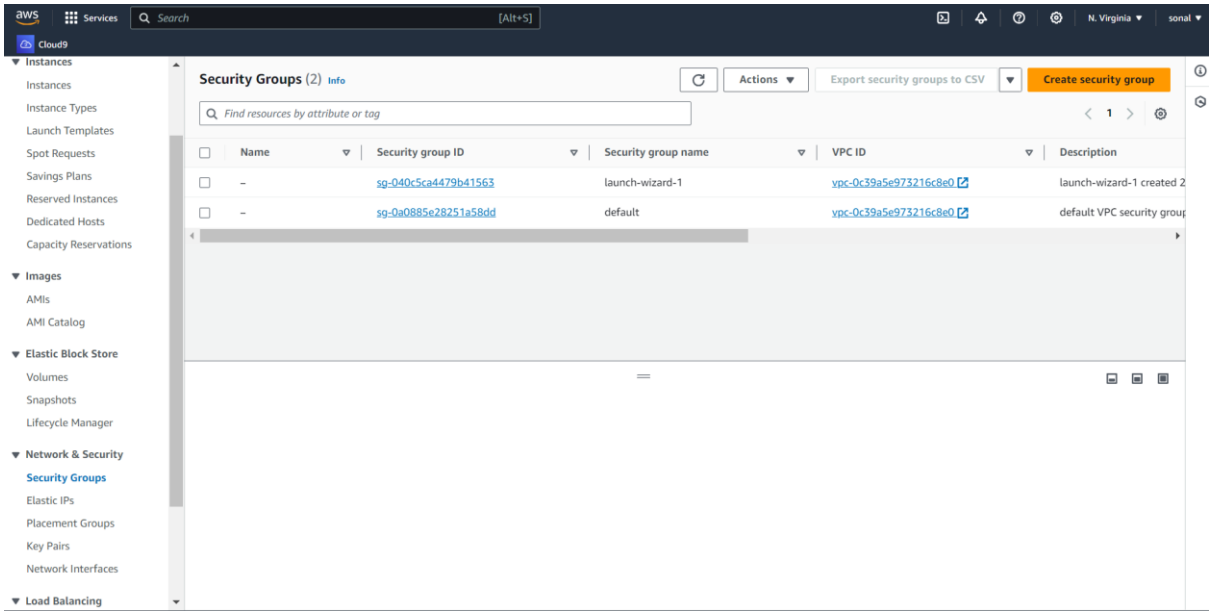
Check security group of both instances

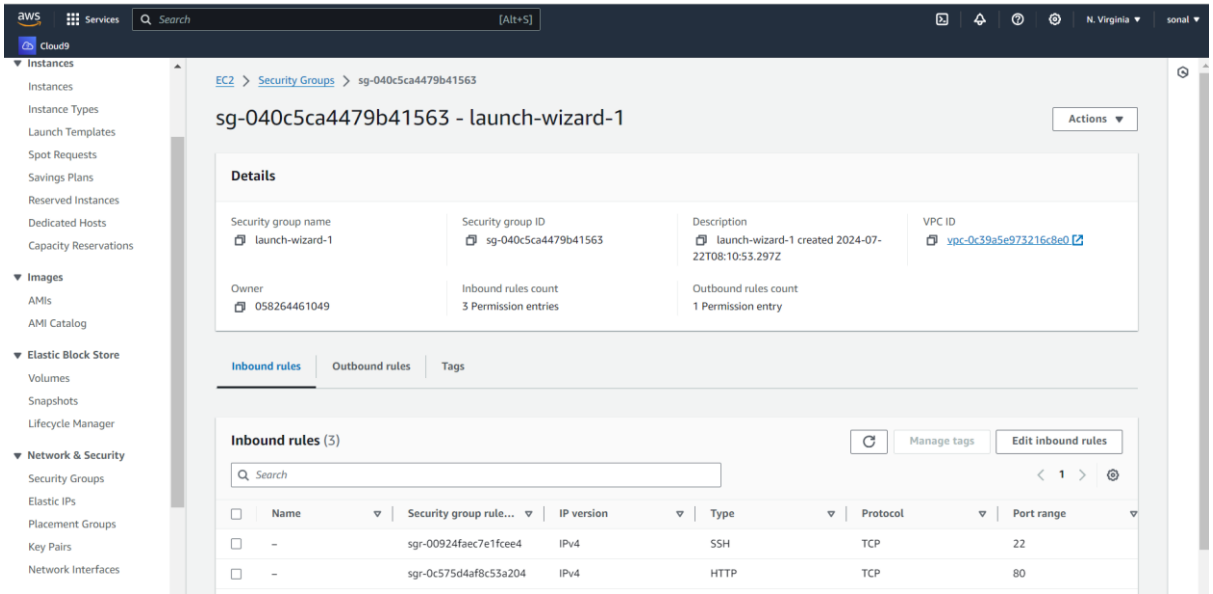


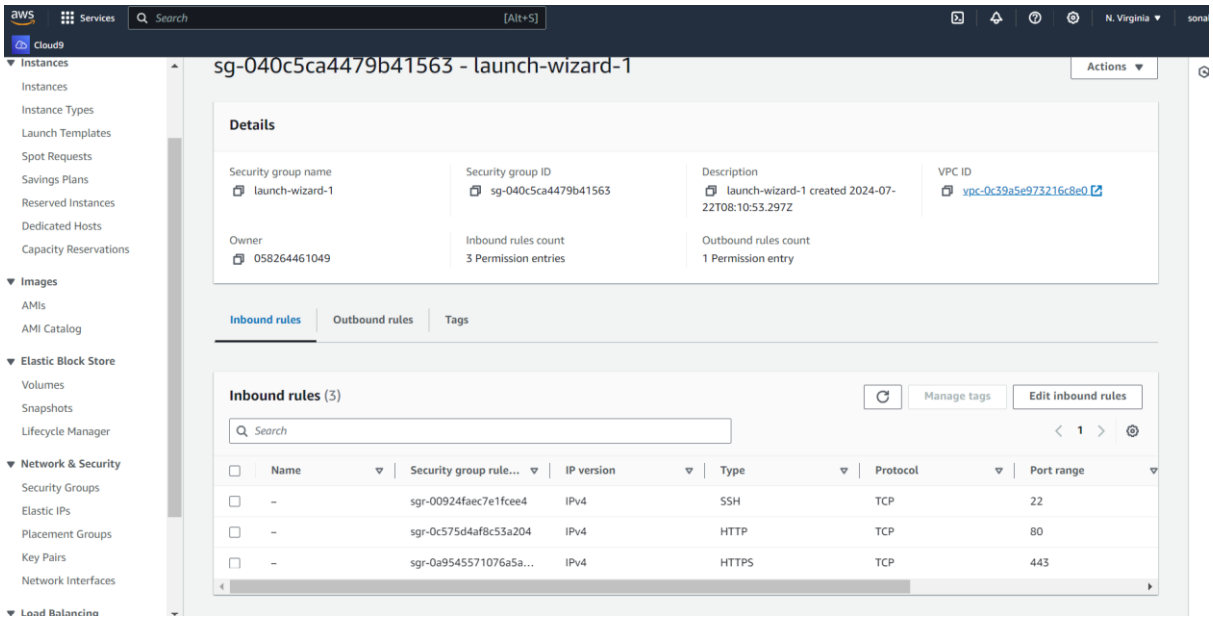
Here security group is launch wizard-1

Now go to security group from left pane

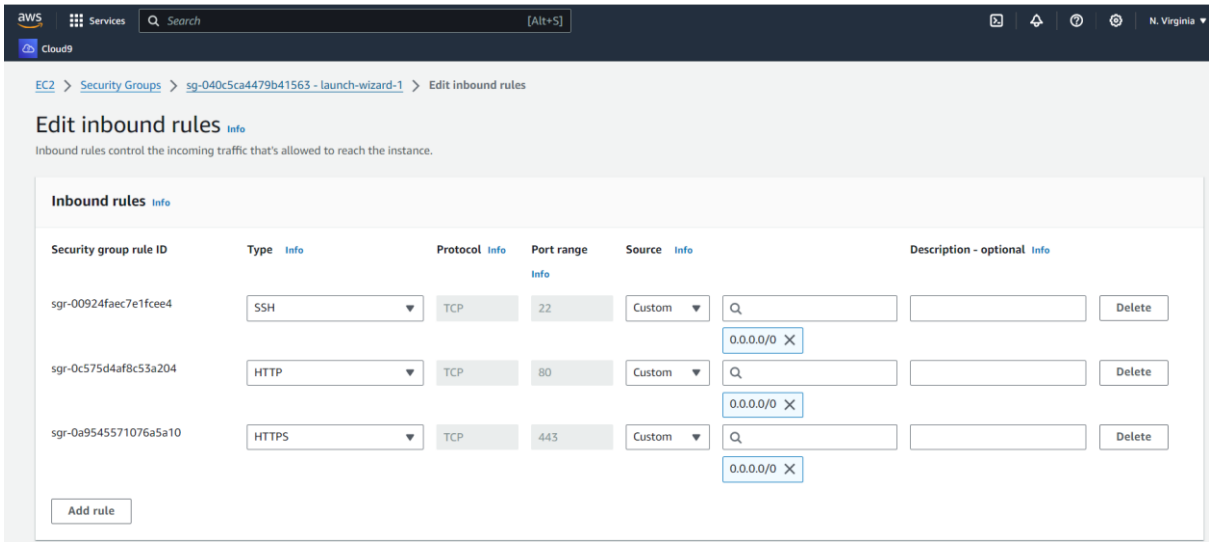
Click on Security group id of Launch wizard-1



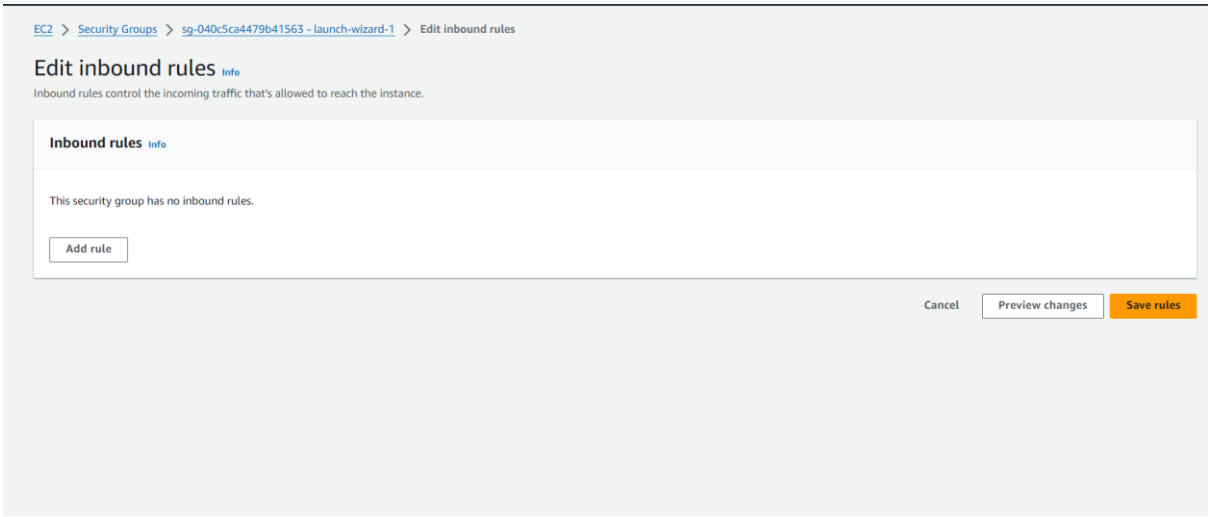




Edit inbound rule



Delete all rules

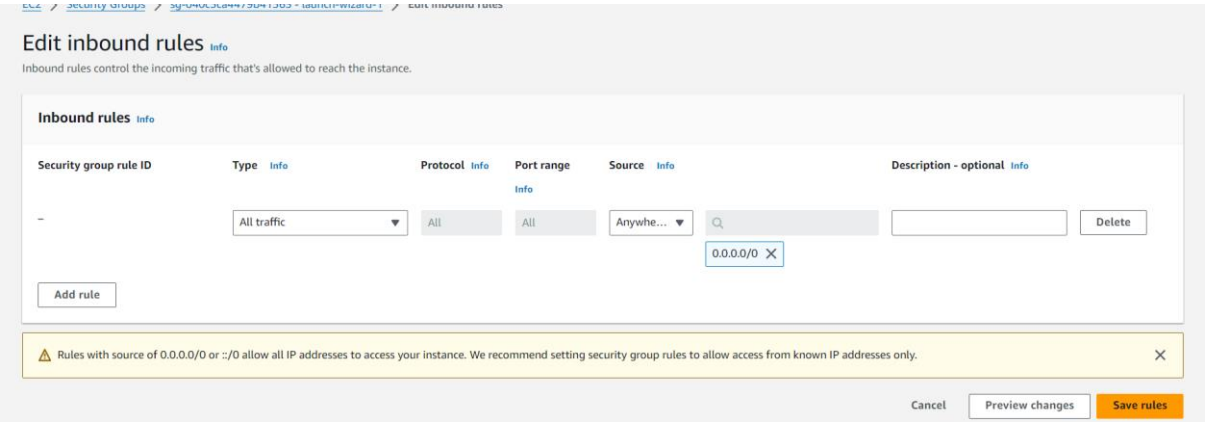


Add new rule

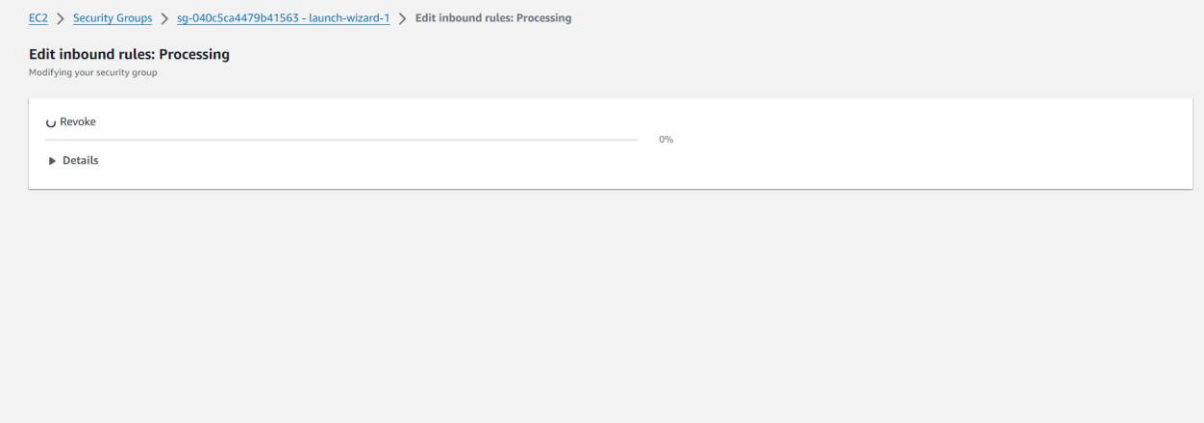
Select

ALL traffic

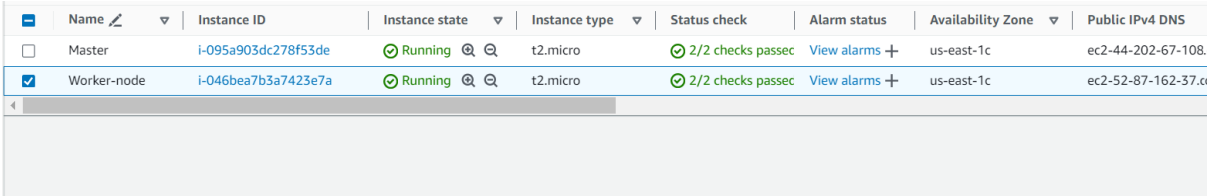
Anywhere IPV4



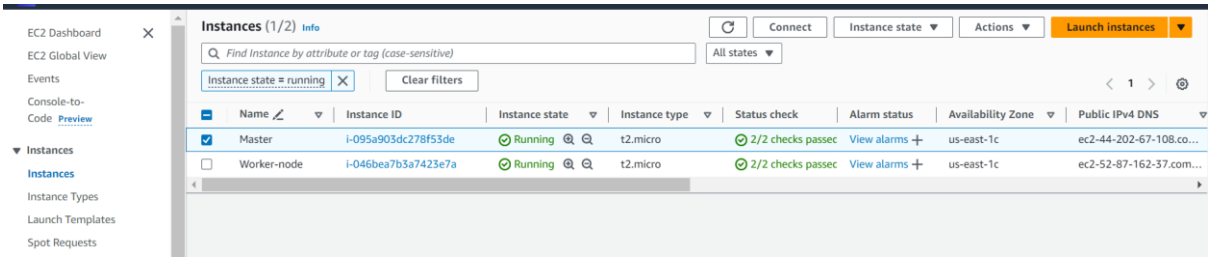
Now save rules



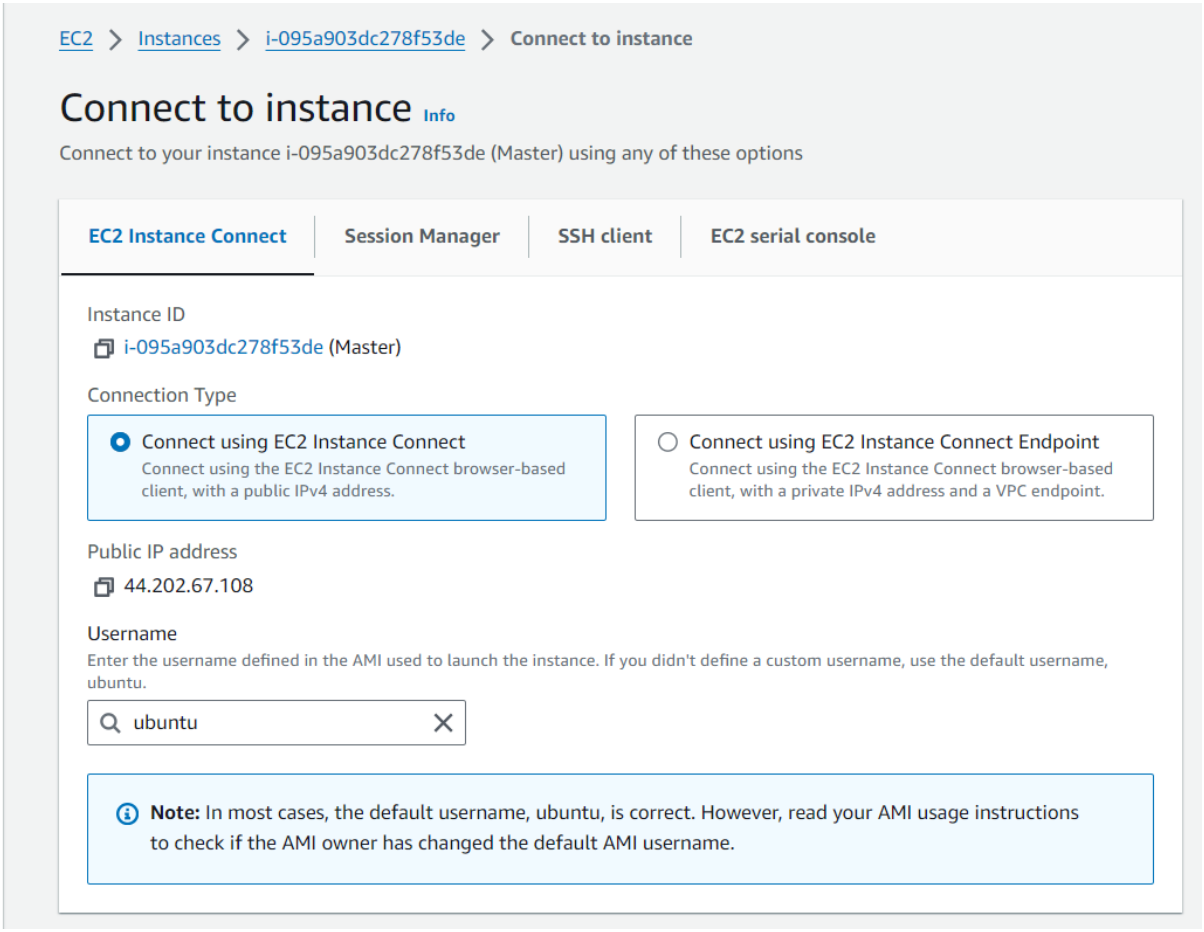
Name the instances



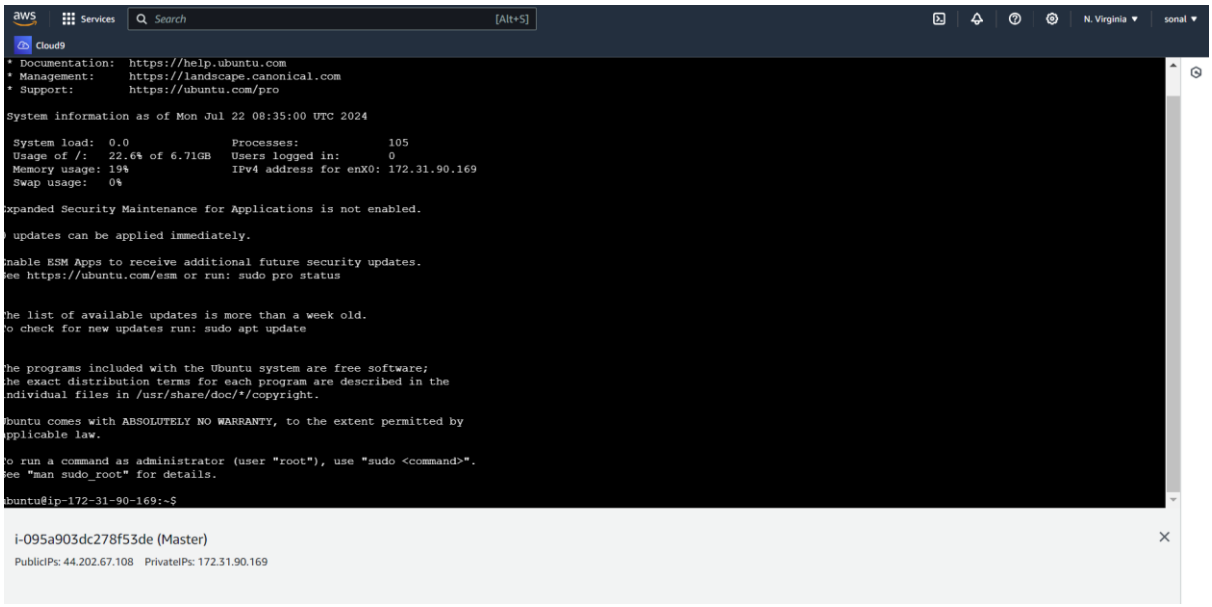
Select Master and connect



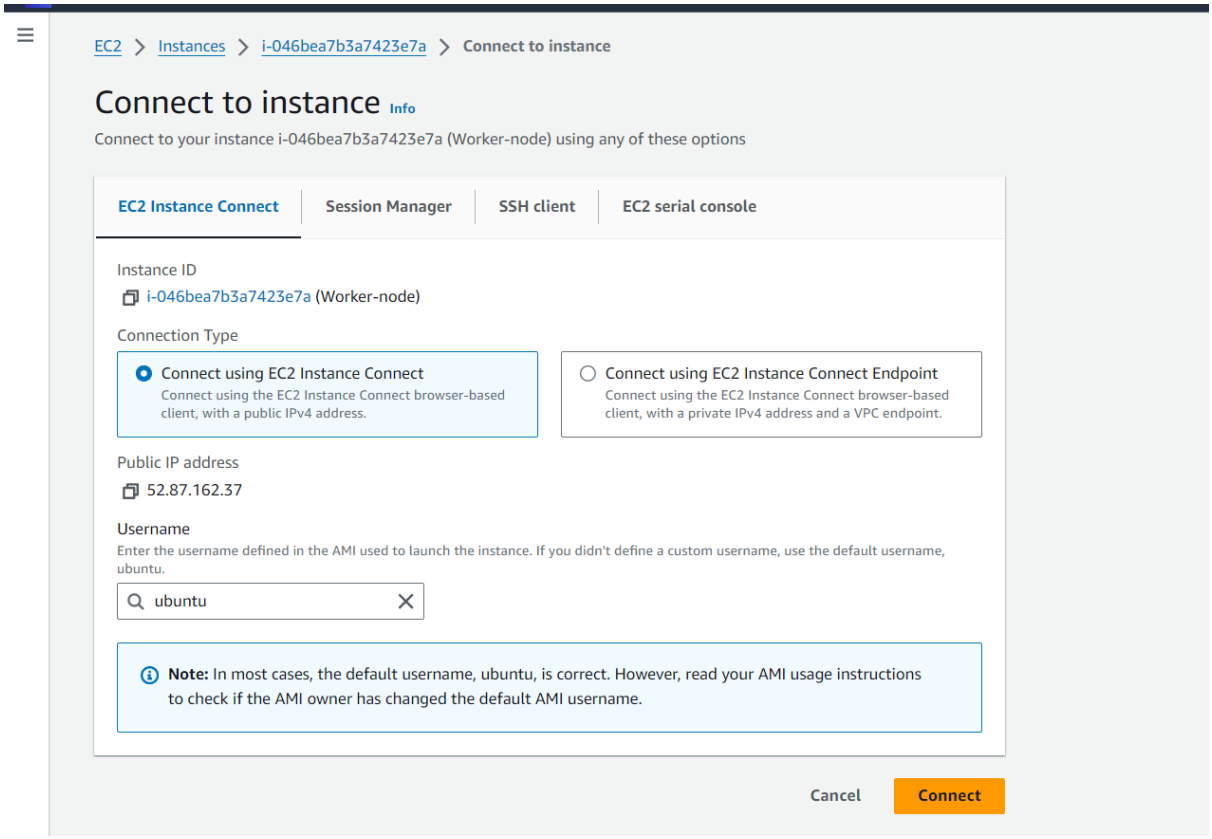
Click on connect



After Connecting



Sameway connect to worker-node



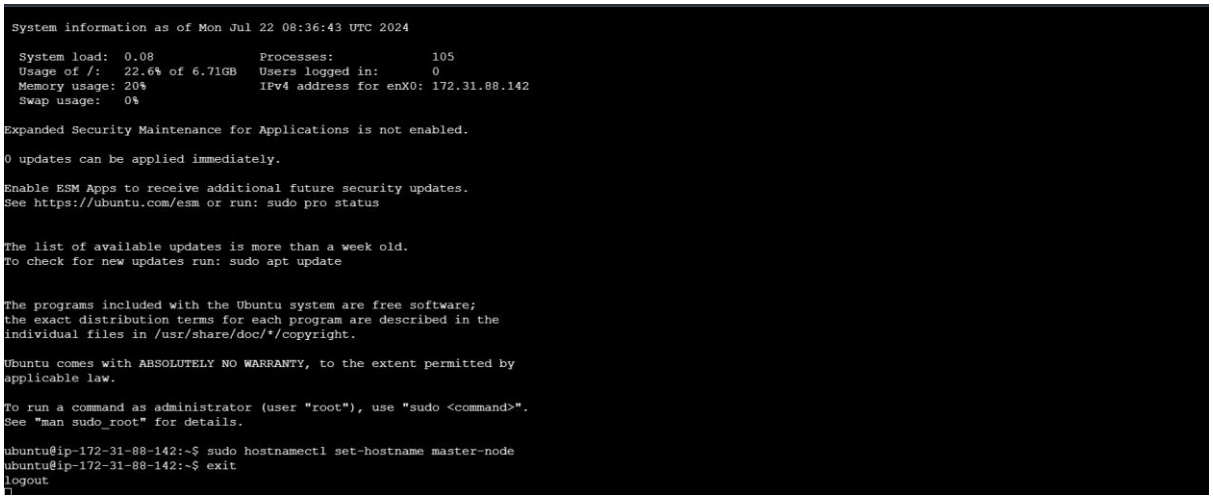
Step 2:

Assign Unique Hostname for Each Server Node

$ sudo hostnamectl set-hostname master-node

Than exit

Refresh



Next, set a worker node hostname by entering the following on the worker server:

$ sudo hostnamectl set-hostname worker1

STEP 3:

On both master and worker1

$ sudo apt-get update

STEP 4:

On both master and worker1

Install docker

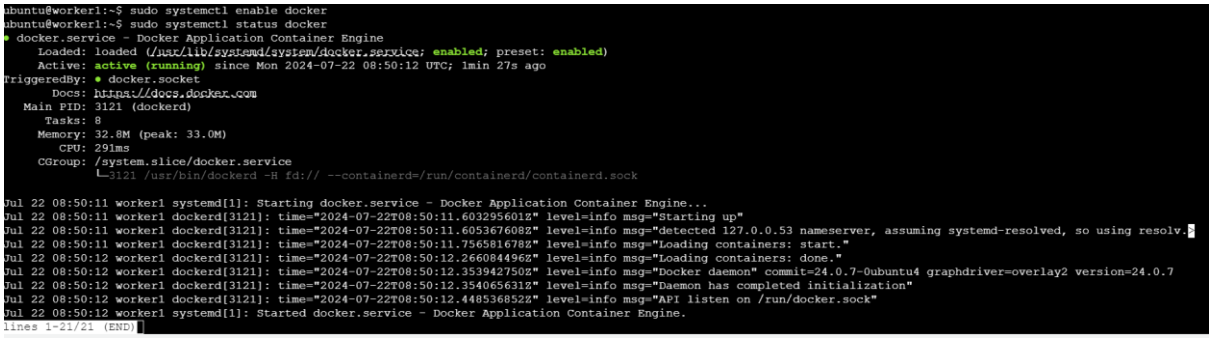
sudo apt-get install docker.io

STEP 5 : Start and Enable Docker

Set Docker to launch at boot by entering the following:

$ sudo systemctl enable docker

$ sudo systemctl status docker



Ctrl+c

Clear

sudo systemctl start docker

STEP 6 Install Kubernetes

https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20240203).
ca-certificates set to manually installed.
curl is already the newest version (8.5.0-2ubuntu10.1).
curl set to manually installed.
gpg is already the newest version (2.4.4-2ubuntu17).
gpg set to manually installed.
The following NEW packages will be installed:
  apt-transport-https
0 upgraded, 1 newly installed, 0 to remove and 22 not upgraded.
Need to get 3974 B of archives.
After this operation, 35.8 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 apt-transport-https all 2.7.14build2 [3974 B]
Fetched 3974 B in 0s (268 kB/s)
Selecting previously unselected package apt-transport-https.
(Reading database ... 68106 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_2.7.14build2_all.deb ...
Unpacking apt-transport-https (2.7.14build2) ...
Setting up apt-transport-https (2.7.14build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host
```

Signining key

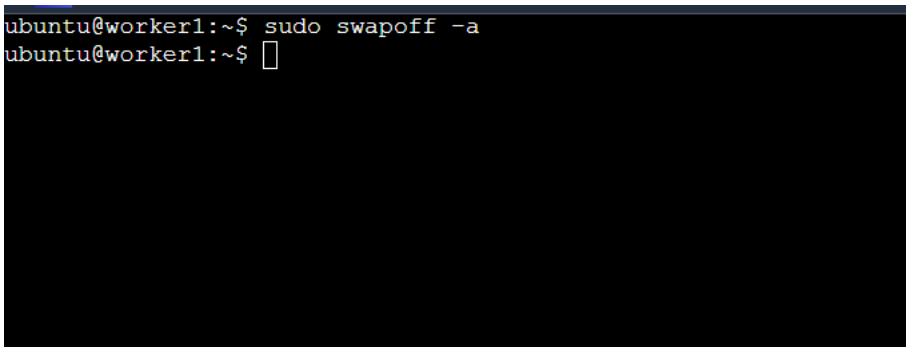
```
ubuntu@worker1:~$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
ubuntu@worker1:~$
```

Cloud9

```
Preparing to unpack .../5-kubernetes-cni_1.4.0-1.1_amd64.deb ...
Unpacking kubernetes-cni (1.4.0-1.1) ...
Selecting previously unselected package socat.
Preparing to unpack .../6-socat_1.8.0.0-4build3_amd64.deb ...
Unpacking socat (1.8.0.0-4build3) ...
Selecting previously unselected package kubelet.
Preparing to unpack .../7-kubelet_1.30.3-1.1_amd64.deb ...
Unpacking kubelet (1.30.3-1.1) ...
Setting up conntrack (1:1.4.8-1ubuntu1) ...
Setting up kubectl (1.30.3-1.1) ...
Setting up ebtables (2.0.11-6build1) ...
Setting up socat (1.8.0.0-4build3) ...
Setting up cri-tools (1.30.0-1.1) ...
Setting up kubernetes-cni (1.4.0-1.1) ...
Setting up kubeadm (1.30.3-1.1) ...
Setting up kubelet (1.30.3-1.1) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@worker1:~$ sudo apt-mark hold kubelet kubeadm kubectl
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
```

Step 7 : Begin Kubernetes Deployment

$ sudo swapoff –a



STEP 8:

Initialize Kubernetes on Master Node

$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all

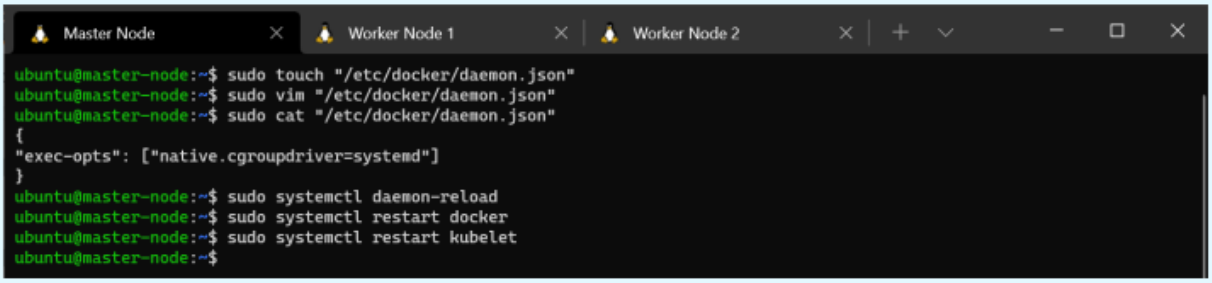
If getting error

Run below code on both

If the kubeadm init command ran without error then ignore this part. If you receive this error "kubelet isn't running or healthy", then do the following.
Create file daemon.json in /etc/docker/ and add following lines in the file.

```
{
   "exec-opts": ["native.cgroupdriver=systemd"]
}
```

And run the following commands.
Do this on both master and worker nodes.



After this run sudo kubeadm reset command and then the init or join command.

$ sudo touch "/etc/docker/daemon.json"

$ sudo nano "/etc/docker/daemon.json"

ADD code

CTRL+ O enter CTRL X

$ sudo cat "/etc/docker/daemon.json"

{

"exec-opts": ["native.cgroupdriver=systemd"]

}

$ sudo systemctl daemon-reload

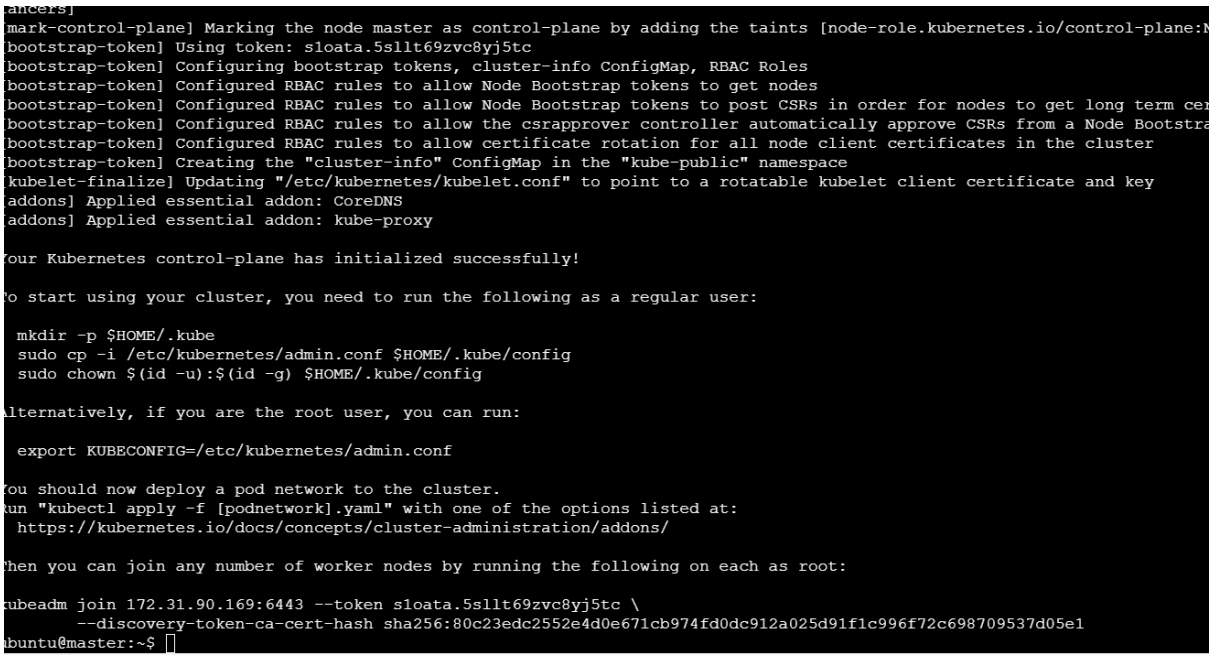
$ sudo systemctl restart docker

$ sudo systemctl restart kubelet

$ sudo kubeadm reset

STEP 9 on master node

sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all



Next, enter the following to create a directory for the cluster: (Master)

kubernetes-master $ mkdir -p $HOME/.kube

kubernetes-master $ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

kubernetes-master $ sudo chown $(id -u):$(id -g) $HOME/.kube/config

copy

**kubeadm join 172.31.90.169:6443 --token s1oata.5sllt69zvc8yj5tc **

--discovery-token-ca-cert-hash sha256:80c23edc2552e4d0e671cb974fd0dc912a025d91f1c996f72c698709537d05e1

Makeit as and copy in worker after flannel is created on master(after step 10)

kubeadm join 172.31.90.169:6443 --token s1oata.5sllt69zvc8yj5tc --discovery-token-ca-cert-hash sha256:80c23edc2552e4d0e671cb974fd0dc912a025d91f1c996f72c698709537d05e1

it will give error

sudo kubeadm join 172.31.90.169:6443 --token s1oata.5sllt69zvc8yj5tc --discovery-token-ca-cert-hash sha256:80c23edc2552e4d0e671cb974fd0dc912a025d91f1c996f72c698709537d05e1 --ignore-preflight-errors=all

---

STEP 10 Copy weblink from masternode

https://kubernetes.io/docs/concepts/cluster-administration/addons/

goto flannel

copy this command and paste on master

For Kubernetes v1.17+

Deploying Flannel with kubectl

```
kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
```

If you use custom `podCIDR` (not `10.244.0.0/16`) you first need to download the above manifest and modify the network to match your one.

Flannel created

```
Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.90.169:6443 --token s1oata.5sllt69zvc8yj5tc \
        --discovery-token-ca-cert-hash sha256:80c23edc2552e4d0e671cb974fd0dc912a025d91f1c996f72c698709537d05e1
ubuntu@master:~$ mkdir -p $HOME/.kube
ubuntu@master:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
ubuntu@master:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@master:~$ ^C
ubuntu@master:~$ ^C
ubuntu@master:~$ kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
namespace/kube-flannel created
serviceaccount/flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
ubuntu@master:~$
```

Step 11:

AFTER

```
r endpoint \"unix:///var/run/containerd/containerd.sock\": rpc error: code = Unavailable desc = connection error: desc = \"transport: Error while dialing: dia
/run/containerd/containerd.sock: connect: permission denied\""
, error: exit status 1
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
error execution phase kubelet-start: couldn't save bootstrap-kubelet.conf to disk: open /etc/kubernetes/bootstrap-kubelet.conf: permission denied
To see the stack trace of this error execute with --v=5 or higher
ubuntu@worker1:~$ sudo ^[[200~kubeadm join 172.31.90.169:6443 --token s1oata.5sllt69zvc8yj5tc --discovery-token-ca-cert-hash sha256:80c23edc2552e4d0e671cb974f
d91f1c996f72c698709537d05e1 --ignore-preflight-errors=all
sudo: kubeadm: command not found
ubuntu@worker1:~$ sudo kubeadm join 172.31.90.169:6443 --token s1oata.5sllt69zvc8yj5tc --discovery-token-ca-cert-hash sha256:80c23edc2552e4d0e671cb974fd0dc912
96f72c698709537d05e1 --ignore-preflight-errors=all
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.004306297s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

ubuntu@worker1:~$
```

Step 12:

Reboot both instances

```
Last login: Mon Jul 22 08:44:23 2024 from 18.206.107.29
ubuntu@master:~$ kubectl get pods --all-namespaces
NAMESPACE      NAME                              READY   STATUS             RESTARTS       AGE
kube-flannel   kube-flannel-ds-qtfbw             1/1     Running            2 (66s ago)    16m
kube-flannel   kube-flannel-ds-t9v2g             1/1     Running            2 (65s ago)    6m29s
kube-system    coredns-7db6d8ff4d-2h5b4          1/1     Running            1 (119s ago)   30m
kube-system    coredns-7db6d8ff4d-sfzs8          1/1     Running            1 (119s ago)   30m
kube-system    etcd-master                       1/1     Running            1 (119s ago)   31m
kube-system    kube-apiserver-master             1/1     Running            1 (119s ago)   31m
kube-system    kube-controller-manager-master    1/1     Running            1 (119s ago)   31m
kube-system    kube-proxy-8kd97                  0/1     CrashLoopBackOff   5 (3s ago)     6m29s
kube-system    kube-proxy-9x78m                  0/1     CrashLoopBackOff   10 (21s ago)   30m
kube-system    kube-scheduler-master             1/1     Running            1 (119s ago)   31m
ubuntu@master:~$ kubectl get pods --all-namespaces
NAMESPACE      NAME                              READY   STATUS    RESTARTS        AGE
kube-flannel   kube-flannel-ds-qtfbw             1/1     Running   2 (118s ago)    17m
kube-flannel   kube-flannel-ds-t9v2g             1/1     Running   2 (117s ago)    7m21s
kube-system    coredns-7db6d8ff4d-2h5b4          1/1     Running   1 (2m51s ago)   31m
kube-system    coredns-7db6d8ff4d-sfzs8          1/1     Running   1 (2m51s ago)   31m
kube-system    etcd-master                       1/1     Running   1 (2m51s ago)   31m
kube-system    kube-apiserver-master             1/1     Running   1 (2m51s ago)   31m
kube-system    kube-controller-manager-master    1/1     Running   1 (2m51s ago)   31m
kube-system    kube-proxy-8kd97                  1/1     Running   6 (55s ago)     7m21s
kube-system    kube-proxy-9x78m                  1/1     Running   11 (73s ago)    31m
kube-system    kube-scheduler-master             1/1     Running   1 (2m51s ago)   31m
ubuntu@master:~$
```

```
ubuntu@master:~$ kubectl get nodes
NAME      STATUS   ROLES           AGE   VERSION
master    Ready    control-plane   44m   v1.30.3
worker1   Ready    <none>          19m   v1.30.3
ubuntu@master:~$
```

Deploy service

On browser search for ngnix deployment yaml

ubuntu@master:~$ sudo nano deploy.yaml

ubuntu@master:~$ sudo cat deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

ubuntu@master:~$ kubectl create -f deploy.yaml

```
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
ubuntu@master:~$ kubectl create -f deploy.yaml
deployment.apps/nginx-deployment created
ubuntu@master:~$ ^C
ubuntu@master:~$
```

ubuntu@master:~$ kubectl get deploy

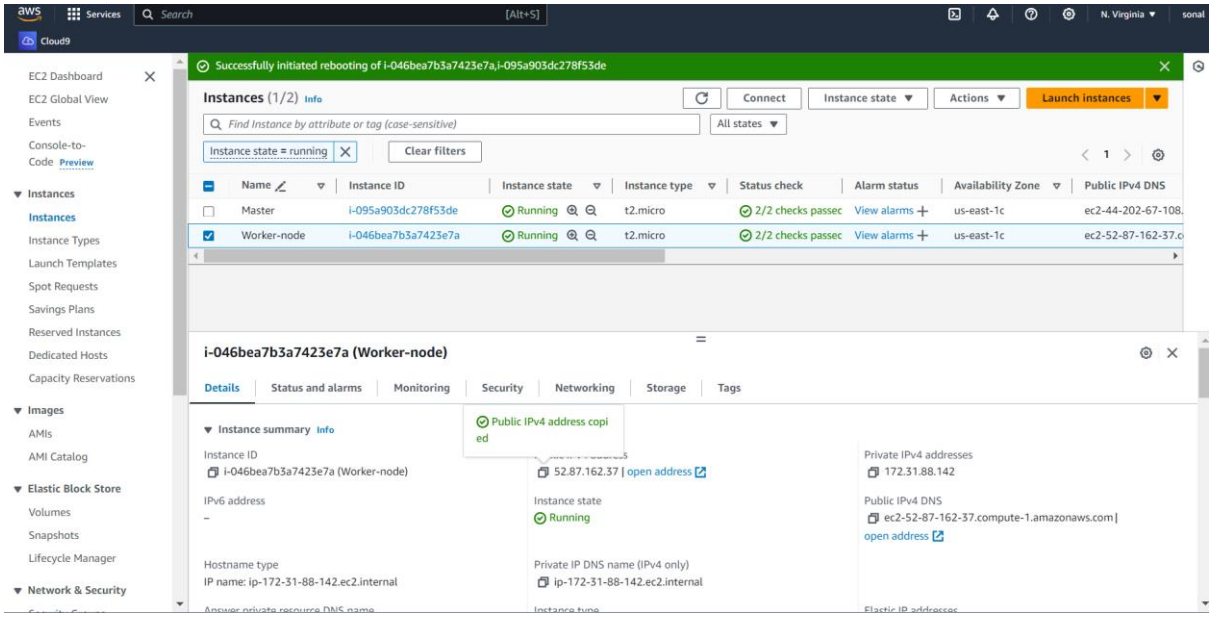
```
ubuntu@master:~$ kubectl get deploy
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   2/2     2            2           2m32s
ubuntu@master:~$ ^C
ubuntu@master:~$ ^C
ubuntu@master:~$
```

kubectl expose deployment.apps/nginix-deployment --type="LoadBalancer"

ubuntu@master:~$ kubectl get svc

```
        - containerPort: 80
ubuntu@master:~$ kubectl create -f deploy.yaml
deployment.apps/nginx-deployment created
ubuntu@master:~$ ^C
ubuntu@master:~$ kubectl get deploy
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   2/2     2            2           2m32s
ubuntu@master:~$ ^C
ubuntu@master:~$ ^C
ubuntu@master:~$ kubectl expose deployment.apps/nginix-deployment --type="LoadBalance"
Error from server (NotFound): deployments.apps "nginix-deployment" not found
ubuntu@master:~$ ^C
ubuntu@master:~$ kubectl expose deployment.apps/nginix-deployment --type="LoadBalancer"
Error from server (NotFound): deployments.apps "nginix-deployment" not found
ubuntu@master:~$ kubectl expose deployment.apps/nginx-deployment --type="LoadBalancer"
service/nginx-deployment exposed
ubuntu@master:~$ kubectl get svc
NAME               TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes         ClusterIP      10.96.0.1       <none>        443/TCP        57m
nginx-deployment   LoadBalancer   10.96.174.79    <pending>     80:31825/TCP   15s
ubuntu@master:~$ ^C
ubuntu@master:~$ ^C
ubuntu@master:~$
```

Go to instance,master select public ipv4



Go to brower

Ipv4:portnumber

Academic Year: 2023-24 **Semester: V**
Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof. Manjusha Kashilkar/Prof.Sonal Jain/Prof.Yaminee Patil

---

EXPERIMENT NO. 05

Aim: To understand terraform lifecycle, core concepts/terminologies and install it on a Linux Machine.

Theory:

Terraform

Terraform is an infrastructure as code (IaC) tool that allows you to build, change, and version infrastructure safely and efficiently. This includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc. Terraform can manage both existing service providers and custom in-house solutions.

Key Features

Infrastructure as Code:

You describe your infrastructure using Terraform's high-level configuration language in human-readable, declarative configuration files. This allows you to create a blueprint that you can version, share, and reuse.

Resource Graph

Terraform builds a resource graph and creates or modifies non-dependent resources in parallel. This allows Terraform to build resources as efficiently as possible and gives you greater insight into your infrastructure.

Change Automation

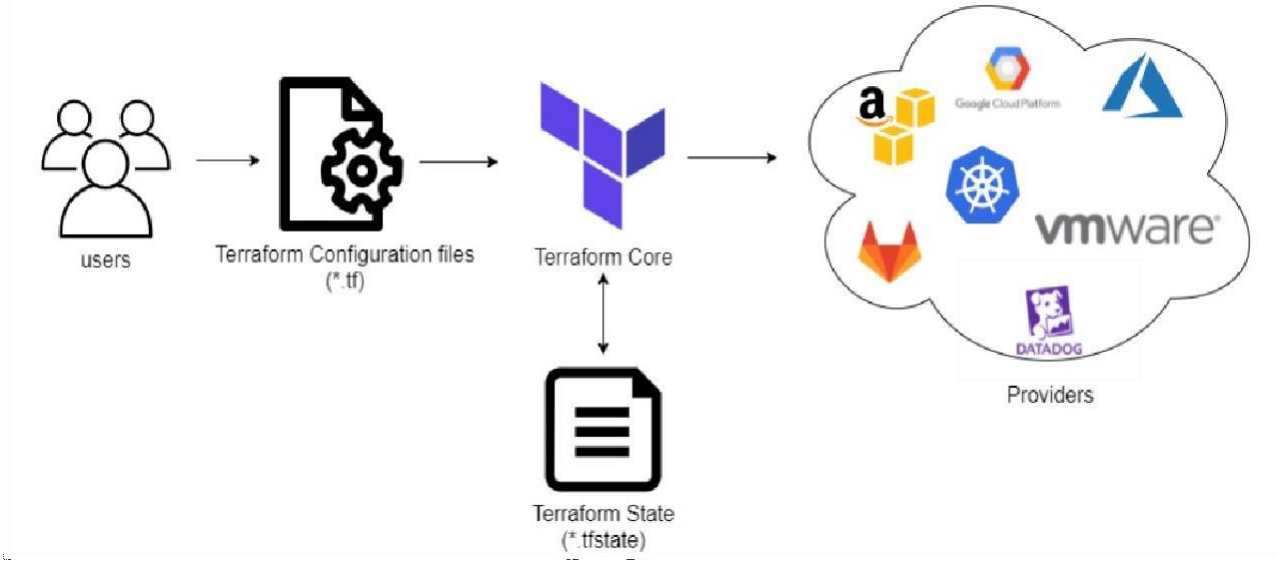
Terraform can apply complex changesets to your infrastructure with minimal human interaction. When you update configuration files, Terraform determines what changed and creates incremental execution plans that respect dependencies.

How does Terraform work?

Terraform actually works, there's sort of two major components:

one is the **terraform core:** it takes the terraform configuration which is being provided by the user and then takes the terraform state which is managed by terraform itself. As such, this gets fed into the core that is responsible for figuring out what is that graph of our different resources for exemple how these different pieces relate to each other or what needs to be created/updated/destroyed, it does all the essential lifecycle management.

On the backside, terraform supports many different **providers**, such as: cloud providers (AWS,GCP,AZURE) and they also could be on-premise infrastructure (VMware,OpenStack.) But this support is not restricted or limited only to Infrastructure As A Service , terraform can also manage higher level like Platform As A Service(Kubernetes,Lambdas..)or even Software As A Service (DataDog,GitHub..)



All of these are important pieces of the infrastructure, they are all part of the logical end-to-end delivery.

Terraform has over a hundred providers for different technologies, and each provider gives terraform users access to their resources. It also gives you the ability to create infrastructure at different levels.

Terraform Core Concepts:

Below are the core concepts/terminologies used in Terraform:

· **Variables:** Also used as input-variables, it is a key-value pair used by Terraform modules to allow customization.

· **Provider:** It is a plugin to interact with APIs of service and access its related resources.

· **Module:** It is a folder with Terraform templates where all the configurations are defined

· **State:** It consists of cached information about the infrastructure managed by Terraform and its related configurations.

· **Resources:** It refers to a block of one or more infrastructure objects (compute instances, virtual networks, etc.), which are used in configuring and managing the infrastructure.

· **Data Source:** It is implemented by providers to return information on external objects to terraform.

· **Output Values:** These are return values of a terraform module that can be used by other configurations.

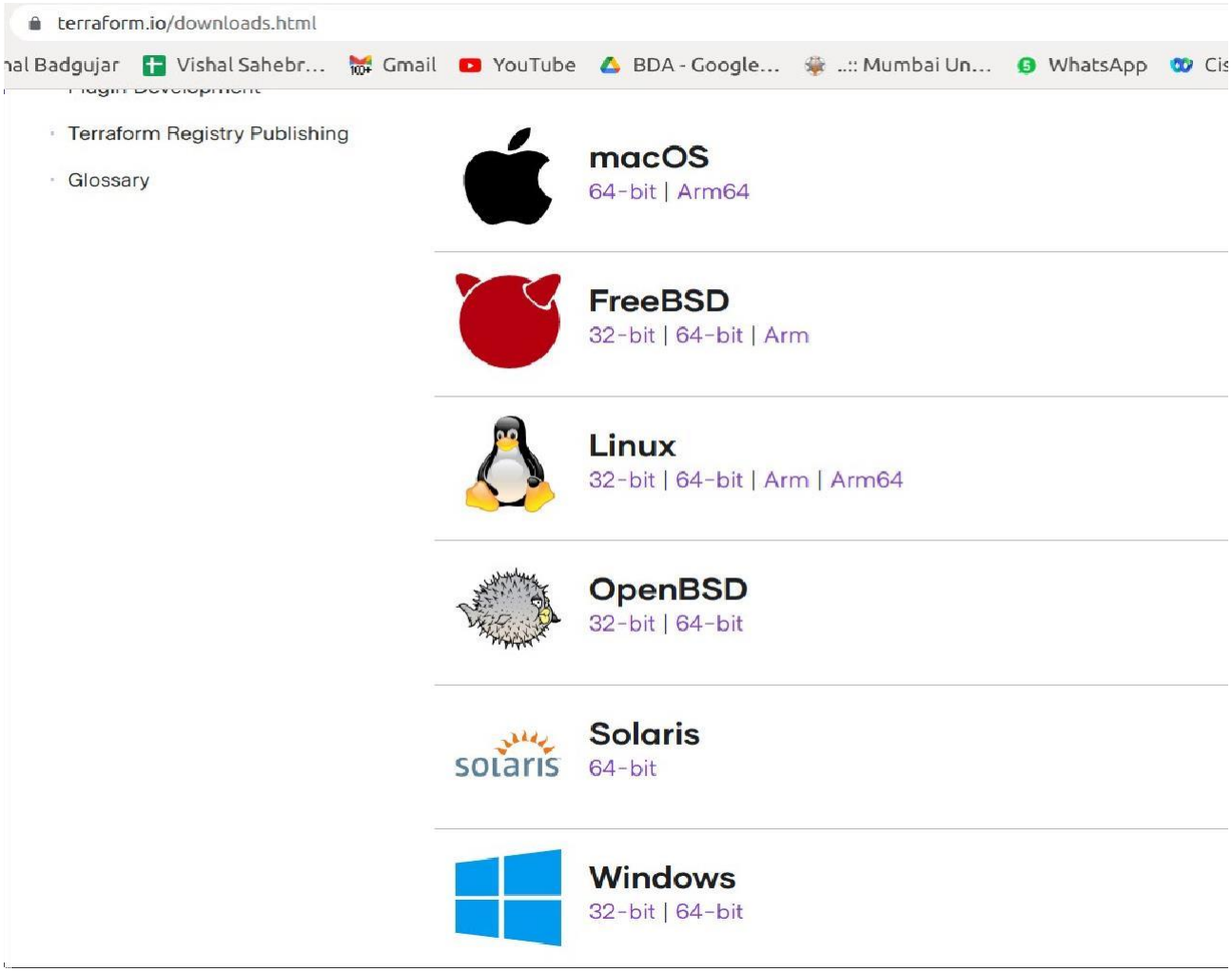
· **Plan:** It is one of the stages where it determines what needs to be created, updated, or destroyed to move from the real/current state of the infrastructure to the desired state.

· **Apply:** It is one of the stages where it applies the changes in the real/current state of the infrastructure in order to move to the desired state.

Terraform Installation Steps on Ubunu18.04

Step: 1 Terraform uses HashiCorp Configuration Language (HCL) to manage environments of Operators and Infrastructure teams. To download go to site https://www.terraform.io/downloads.html

Select the appropriate package for your operating system and architecture.



Step:2 unzip the archive by using below command

```
vishal@master:~$ unzip terraform_1.0.3_linux_amd64.zip
```

The archive will extract a single binary called **terraform**.

Step 3: Change the directory to unzipped folder

```
vishal@master:~$ cd terraform_1.0.3_linux_amd64/
```

and Move the terraform binary to a directory included in your system's PATH in my case */usr/*local/bin/

```
vishal@master:~$ sudo mv terraform /usr/local/bin/
```

Step 4: To check whether Terraform is installed, run:

```
vishal@master:~$ terraform -v
Terraform v1.0.3
on linux_amd64
vishal@master:~$
```

Conclusion: Write your own findings.

Academic Year: 2022-23 **Semester: V**
Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof. Manjusha K.

EXPERIMENT NO. 06

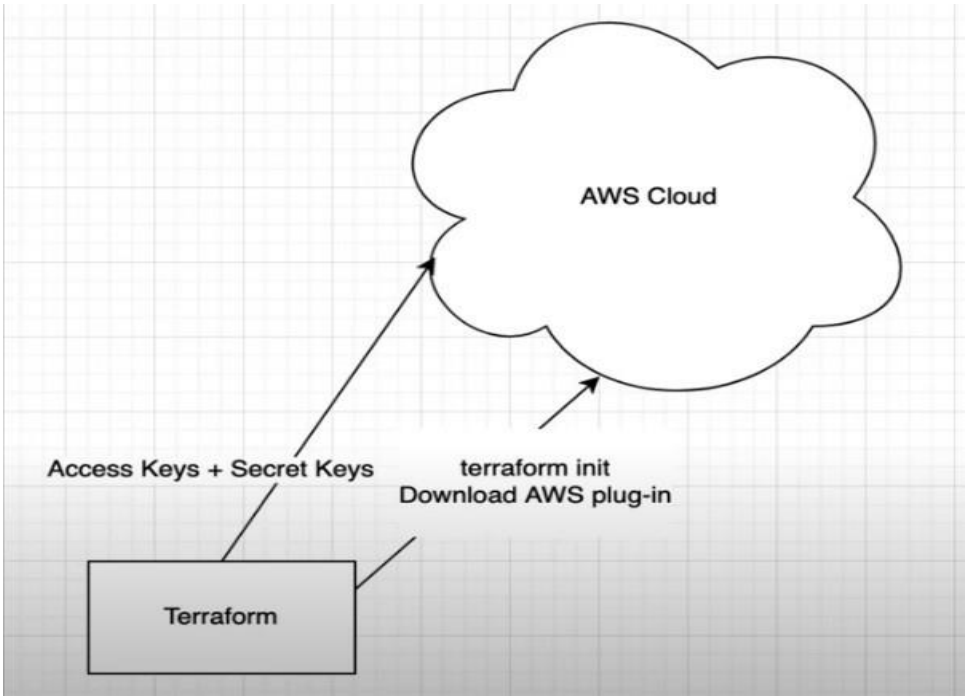
Aim: To Build, change, and destroy AWS infrastructure Using Terraform.

Theory:

Hashicorp's Terraform is an open-source tool for provisioning and managing cloud infrastructure. Terraform can provision resources on any cloud platform.

Terraform allows you to create infrastructure in configuration files(**tf files**) that describe the topology of cloud resources. These resources include virtual machines, storage accounts, and networking interfaces.

We will see how you can use Terraform to provision EC2 instance. Please do the below steps for provisioning EC2 instances on AWS:



Pre-requistes:

1. Install the AWS CLI version 2 on Linux

Follow these steps from the command line to install the AWS CLI on Linux.

Install curl on linux

```
vishal@apsit:~$ sudo apt-get install curl
```

vishal@apsit:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

```
vishal@apsit:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 41.8M  100 41.8M    0     0  2529k      0  0:00:16  0:00:16 --:--:-- 2555k
```

vishal@apsit:~$ sudo apt install unzip

```
vishal@apsit:~$ sudo apt install unzip
```

vishal@apsit:~$ sudo unzip awscliv2.zip

```
vishal@apsit:~$ sudo unzip awscliv2.zip
```

vishal@apsit:~$ sudo ./aws/install

```
vishal@apsit:~$ sudo ./aws/install
You can now run: /usr/local/bin/aws --version
```

vishal@apsit:~$ aws --version

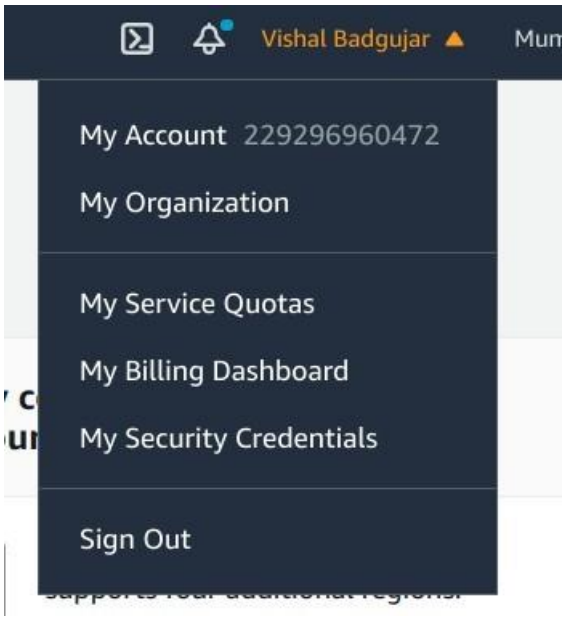
it should display the below outout.

aws-cli/2.1.29 Python/3.8.8 Linux/5.4.0-1038-aws exe/x86_64.ubuntu.18 prompt/off

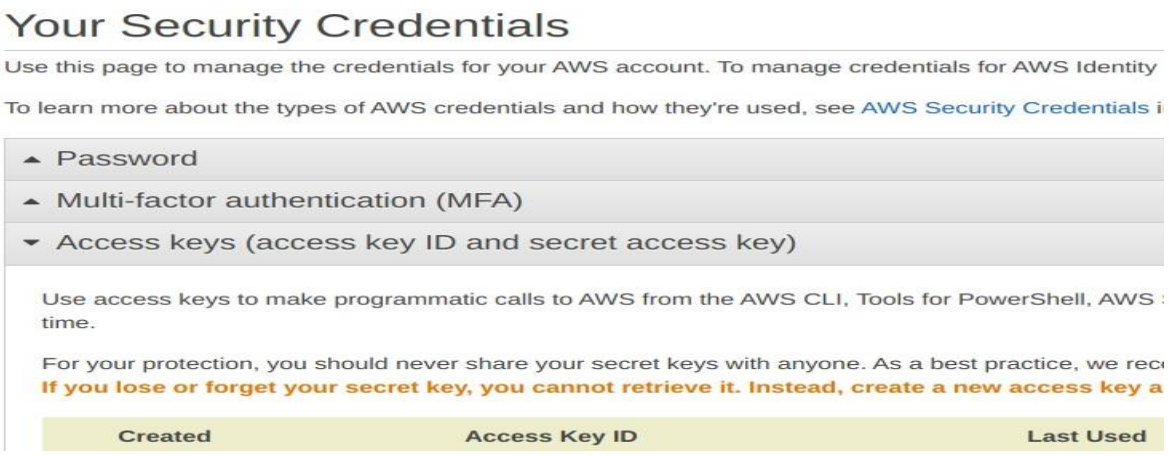
```
vishal@apsit:~$ aws --version
aws-cli/2.2.25 Python/3.8.8 Linux/5.4.0-80-generic exe/x86_64.ubuntu.18 prompt/off
```

2. Create a new access key if you don't have one. Make sure you download the keys in your local machine.

Login to AWS console, click on username and go to My security credentials.



Continue on security credentials, click on access keys



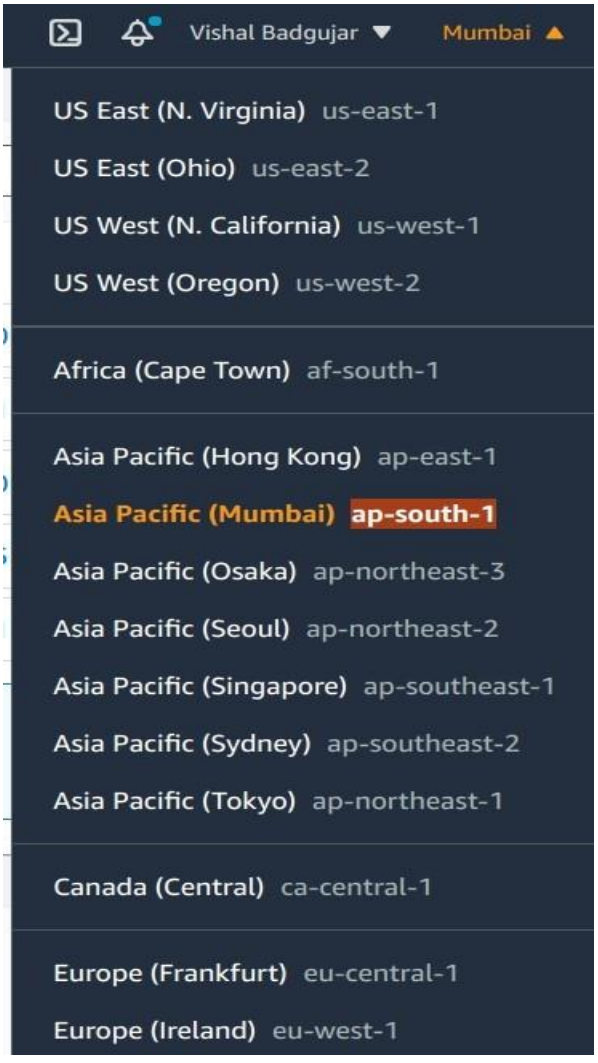
Perform below commands in Linux where you have installed Terraform

First setup your access keys, secret keys and region code locally.

vishal@apsit:~$aws configure

Created	Access Key ID	Last Used	Last Used Region	Last Used Service	Status
Jun 4th 2021	AKIATKYZJ6PMCN2VF436	2021-07-04 21:26 UTC+0530	us-east-1	sts	Active
Aug 1st 2021	AKIATKYZJ6PMFLTCGGPV	N/A	N/A	N/A	Active

You can check region as shown in below image :



```
vishal@apsit:~$ aws configure
AWS Access Key ID [None]: AKIATKYZJ6PMFLTCGGPV
AWS Secret Access Key [None]: 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
Default region name [None]: ap-south-1
Default output format [None]:
vishal@apsit:~$
```

Create one Directory for Terraform project in which all files of terraform we can save

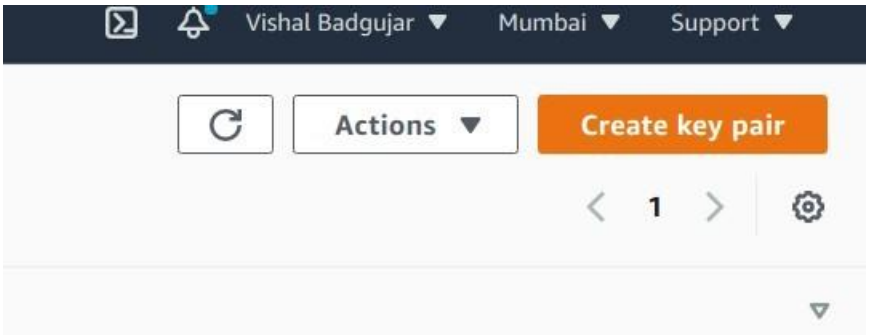
vishal@apsit:~$ cd ~
vishal@apsit:~$ mkdir project-terraform
vishal@apsit:~$ cd project-terraform

```
vishal@apsit:~$ mkdir project-terraform
vishal@apsit:~$ cd project-terraform/
vishal@apsit:~/project-terraform$
```

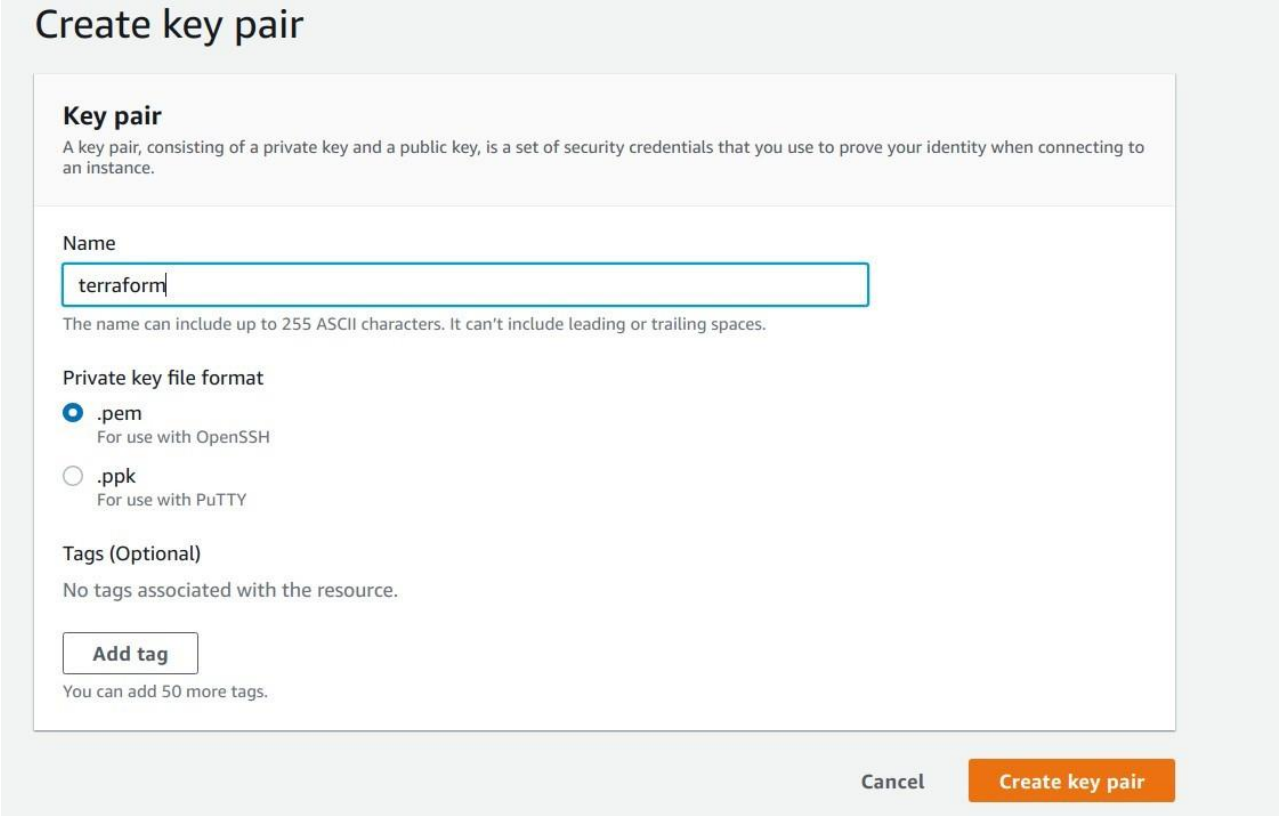
Create Terraform Files

vishal@apsit:~$ sudo nano variables.tf

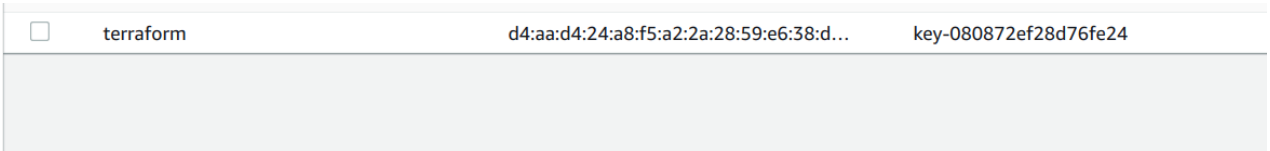
In order to provide key name in variables first create key pair as shown:



Give name to key pair file as **terraform**



Key pair is generated



Use your Region and Key name in variable.tf as shown and provide instance type which you want to create.

```
File  Edit  View  Search  Terminal  Help
  GNU nano 2.9.3                        variables.tf                          Modified

variable "aws_region" {
  description = "The AWS region to create things in."
  default     = "ap-south-1"
}

variable "key_name" {
  description = " SSH keys to connect to ec2 instance"
  default     =  "terraform"
}

variable "instance_type" {
  description = "instance type for ec2"
  default     =  "t2.micro"
}

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line  M-E Redo
```

After creating variable terraform file note down the AMI ID of instance which u want to create which we will use to configure our instance in main.tf file.

Now create main.tf file:

```
vishal@apsit:~/project-terraform$ sudo nano main.tf
```

```
provider "aws" {
  region = var.aws_region
}

#Create security group with firewall rules
resource "aws_security_group" "security_jenkins_port" {
  name        = "security_jenkins_port"
  description = "security group for jenkins"

  ingress {
    from_port  = 8080
    to_port    = 8080
    protocol   = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

 ingress {
    from_port  = 22
    to_port    = 22
    protocol   = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
# outbound from jenkis server
```

```
  egress {
    from_port  = 0
    to_port     = 65535
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags= {
    Name = "security_jenkins_port"
  }
}

resource "aws_instance" "myFirstInstance" {
  ami           = "ami-0b9064170e32bde34"
  key_name = var.key_name
  instance_type = var.instance_type
  security_groups= [ "security_jenkins_port"]
  tags= {
    Name = "jenkins_instance"
  }
}

# Create Elastic IP address
resource "aws_eip" "myFirstInstance" {
  vpc      = true
  instance = aws_instance.myFirstInstance.id
```

```
tags= {
   Name = "jenkins_elstic_ip"
  }
}
```

Put AMI-ID in above highlighted space and Now execute the below command:

```
vishal@apsit:~/project-terraform$ terraform init
```

you should see like below screenshot.

```
vishal@apsit:~/project-terraform$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v3.52.0...
- Installed hashicorp/aws v3.52.0 (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.
```

Execute the below command

```
vishal@apsit:~/project-terraform$ terraform plan
```

the above command will show how many resources will be added.
Plan: 3 to add, 0 to change, 0 to destroy.

```
vishal@apsit:~/project-terraform$ 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_eip.myFirstInstance will be created
  + resource "aws_eip" "myFirstInstance" {
      + allocation_id        = (known after apply)
      + association_id       = (known after apply)
      + carrier_ip           = (known after apply)
```

```
Plan: 3 to add, 0 to change, 0 to destroy.
```

Execute the below command

```
vishal@apsit:~/project-terraform$ terraform apply
```

Provide the value as Yes for applying terraform

```
Plan: 3 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

Plan: 3 to add, 0 to change, 0 to destroy.

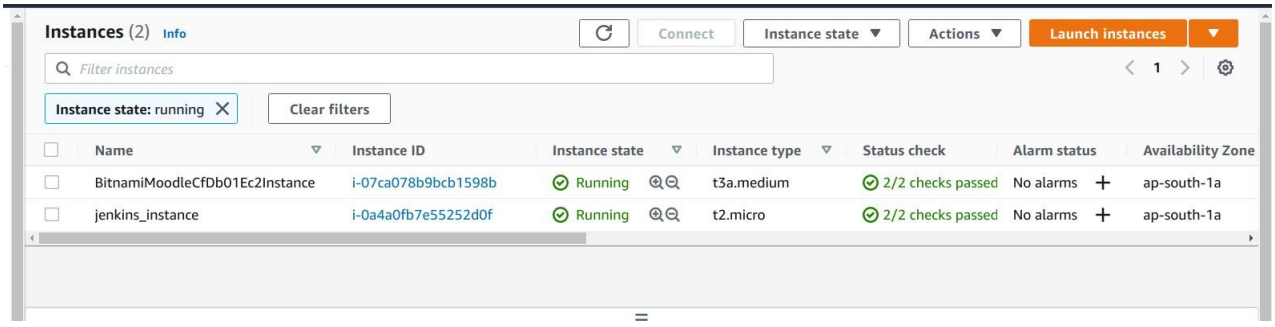
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

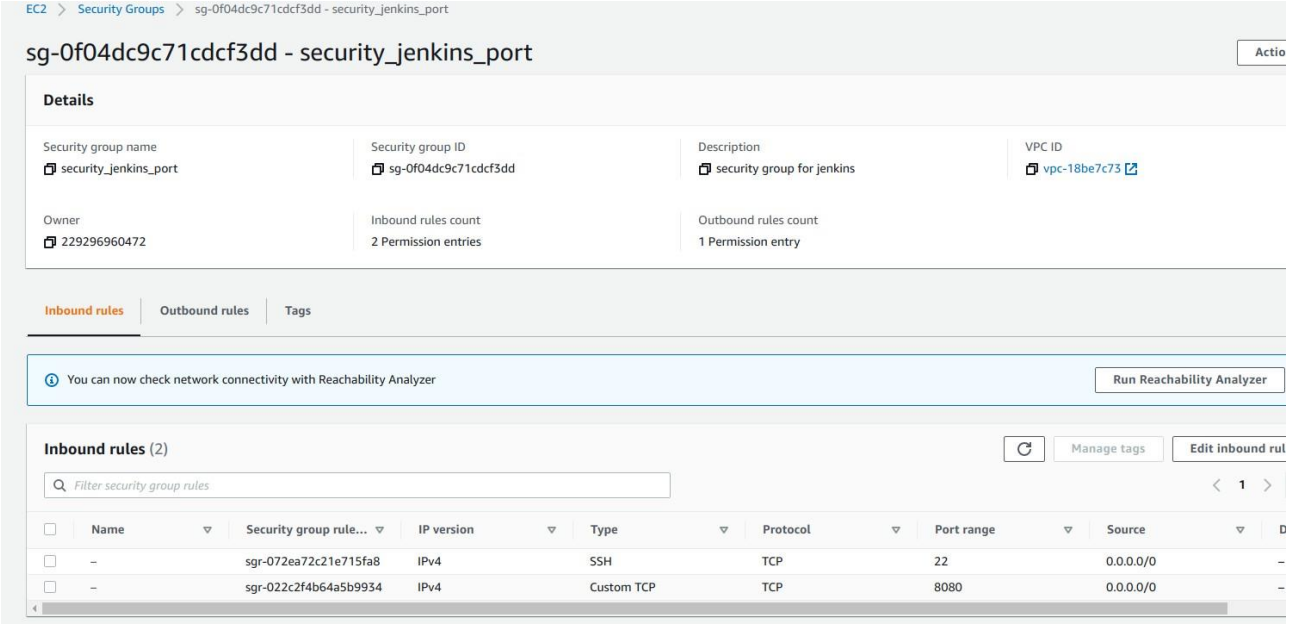
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

```
Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.myFirstInstance: Creating...
aws_instance.myFirstInstance: Still creating... [10s elapsed]
aws_instance.myFirstInstance: Still creating... [20s elapsed]
aws_instance.myFirstInstance: Still creating... [30s elapsed]
aws_instance.myFirstInstance: Creation complete after 32s [id=i-0a4a0fb7e55252d0f]
aws_eip.myFirstInstance: Creating...
aws_eip.myFirstInstance: Creation complete after 1s [id=eipalloc-0fd8f60524b10fc93]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

Now login to EC2 console, to see the new instances up and running, you can see Jenkins_instance is up and running which we deploy from terraform.



You can also check the security group resource details which you created from terraform :



Terraform destroy

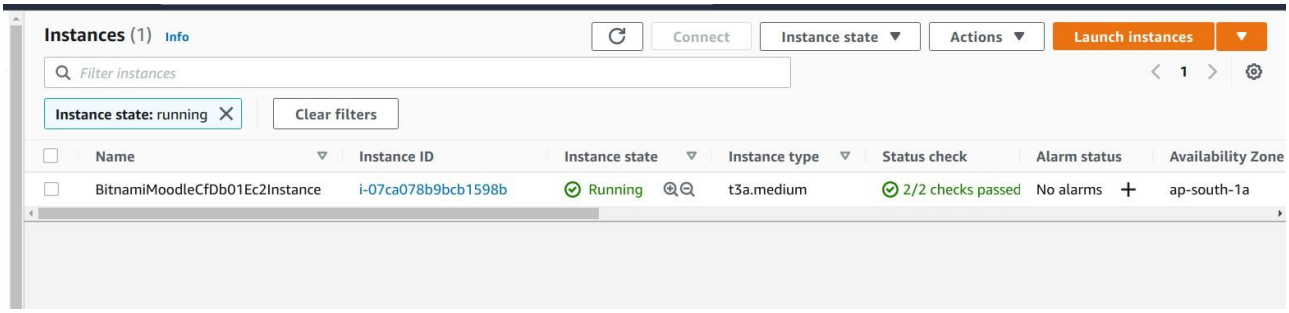
you can also destroy or delete your instance by using terraform destroy command :

```
vishal@apsit:~/project-terraform$ terraform destroy
```

```
Plan: 0 to add, 0 to change, 3 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
```

```
  Enter a value: yes

aws_eip.myFirstInstance: Destroying... [id=eipalloc-0fd8f60524b10fc93]
aws_security_group.security_jenkins_port: Destroying... [id=sg-0f04dc9c71cdcf3dd]
aws_eip.myFirstInstance: Destruction complete after 2s
aws_instance.myFirstInstance: Destroying... [id=i-0a4a0fb7e55252d0f]
aws_security_group.security_jenkins_port: Still destroying... [id=sg-0f04dc9c71cdcf3dd, 10s elapsed]
aws_instance.myFirstInstance: Still destroying... [id=i-0a4a0fb7e55252d0f, 10s elapsed]
aws_security_group.security_jenkins_port: Still destroying... [id=sg-0f04dc9c71cdcf3dd, 20s elapsed]
aws_instance.myFirstInstance: Still destroying... [id=i-0a4a0fb7e55252d0f, 20s elapsed]
aws_security_group.security_jenkins_port: Still destroying... [id=sg-0f04dc9c71cdcf3dd, 30s elapsed]
aws_instance.myFirstInstance: Still destroying... [id=i-0a4a0fb7e55252d0f, 30s elapsed]
aws_security_group.security_jenkins_port: Destruction complete after 38s
aws_instance.myFirstInstance: Still destroying... [id=i-0a4a0fb7e55252d0f, 40s elapsed]
aws_instance.myFirstInstance: Destruction complete after 40s

Destroy complete! Resources: 3 destroyed.
```

Now you can see instance which you created by using terraform is deleted successfully from aws console also you can check it will removed successfully:



All the Resources including Security groups, EC2 instances using terraform will be deleted. In this way we can automate infrastructure set up using terrform in aws cloud.

Conclusion: Write your own findings.

Academic Year: 2022-23 **Semester: V**
Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof. Manjusha Kashilkar

EXPERIMENT NO. 07

Aim: To understand Static Analysis SAST process and learn to integrate Jenkins SAST to SonarQube/GitLab.

Theory:

Static application security testing (SAST) is a way to perform automated testing and analysis of a program's source code without executing it to catch security vulnerabilities early on in the software development cycle. Also referred to as static code analysis, SAST is the process of parsing through the code looking at how it was written and checking for security vulnerabilities and safety concerns.

Because static application security testing tools don't need a running application to perform an analysis, they can be used early and often in the implementation phase of the software development life cycle (SDLC). As a developer is writing code, SAST can analyze it in real-time to inform the user of any rule violations, so you can immediately deal with issues and deliver higher quality applications out of the box while preventing issues at the end of the development process.

Additionally, as SAST helps you audit code and triage issues during implementation, test automation tools can also easily integrate into development ecosystems where continuous integration/continuous delivery (CI/CD) are part of the workflow that helps assure secure, safe, and reliable code during integration, and before it's delivered.

What's the Difference Between SAST and DAST?

While SAST analyses every line of code without running the application, dynamic application security testing (DAST) simulates malicious attacks and other external behaviors by searching for ways to exploit security vulnerabilities during runtime or black box testing.

DAST is particularly useful when catching unexpected vulnerabilities that development teams simply didn't think of. This additional level of insight that DAST brings offers a broad array of security testing to find flaws and prevent attacks like SQL injections, cross-site scripting (XSS), and

other exploits. Remember the 2014 Sony Pictures hack? That could have been prevented with DAST.

Comparing SAST against DAST, each is more effective than the other during different stages of the SDLC. SAST represents the developer's point of view to make sure that all coding procedures follow the appropriate safety standards to ensure the security of an application from the start. DAST, on the other hand, mimics the hacker approach to identify possible user behavior towards the end of development.

Steps:

1) Install and configure a Jenkins and SonarQube CICD environment using Docker containers.

2) Configure Jenkins with the SonarQube Scanner plugin for automated static code analysis.

1) Install and configure a Jenkins and SonarQube CICD environment using Docker containers.

Installation of Jenkins

The version of Jenkins included with the default Ubuntu packages is often behind the latest available version from the project itself. To take advantage of the latest fixes and features, you can use the project-maintained packages to install Jenkins.

```
manjusha@apsit:~$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key
| sudo apt-key add -
```

When the key is added, the system will return OK. Next, append the Debian package repository address to the server's sources.list:

```
manjusha@apsit:~$ sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
```

When both of these are in place, run `update` so that `apt`will use the new repository:

```
manjusha@apsit:~$ sudo apt update
```

Finally, install Jenkins and its dependencies:

```
manjusha@apsit:~$sudo apt install jenkins
```

Let's start Jenkins using systemctl:

manjusha@apsit:~$sudo systemctl start jenkins

Since systemctl doesn't display output, you can use its status command to verify that Jenkins started successfully:

manjusha@apsit:~$sudo systemctl status jenkins

If everything went well, the beginning of the output should show that the service is active and configured to start at boot:

Now that Jenkins is running, let's adjust our firewall rules so that we can reach it from a web browser to complete the initial setup.

Opening the Firewall

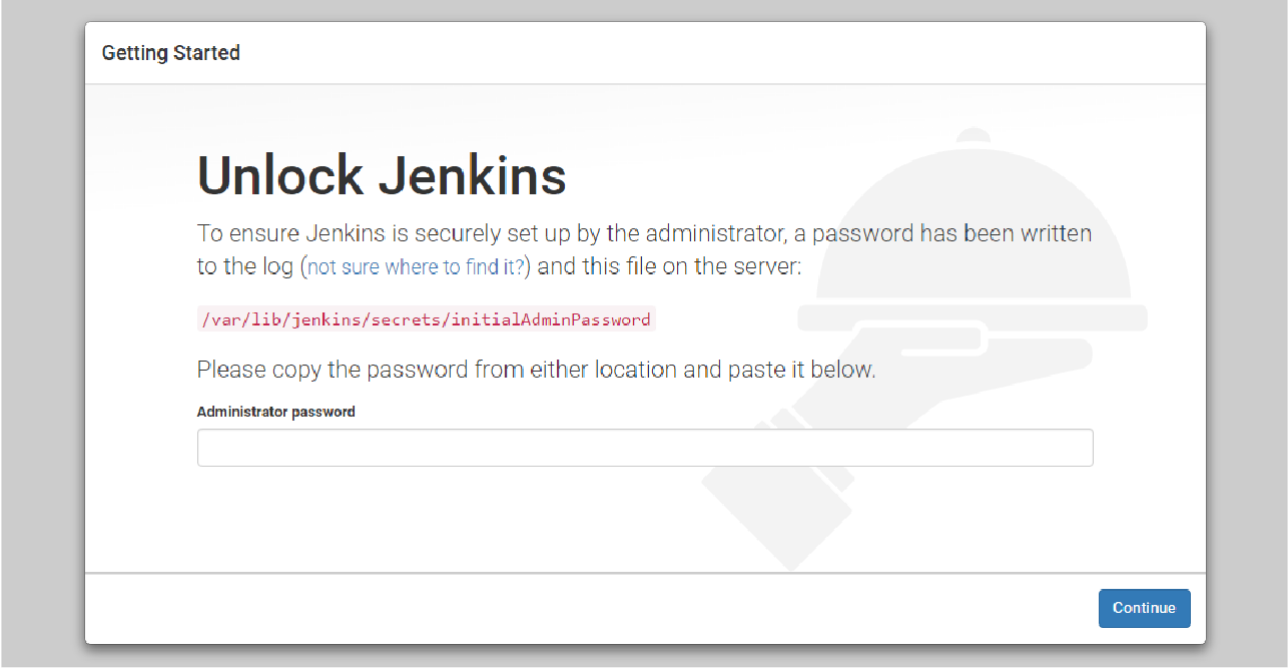
By default, Jenkins runs on port 8080, so let's open that port using ufw:

manjusha@apsit:~$sudo ufw allow

8080Setting Up Jenkins

To set up your installation, visit Jenkins on its default port, 8080, using your server domain name or IP address: **http://your_server_ip_or_domain:8080**

You should see the Unlock Jenkins screen, which displays the location of the initial password:



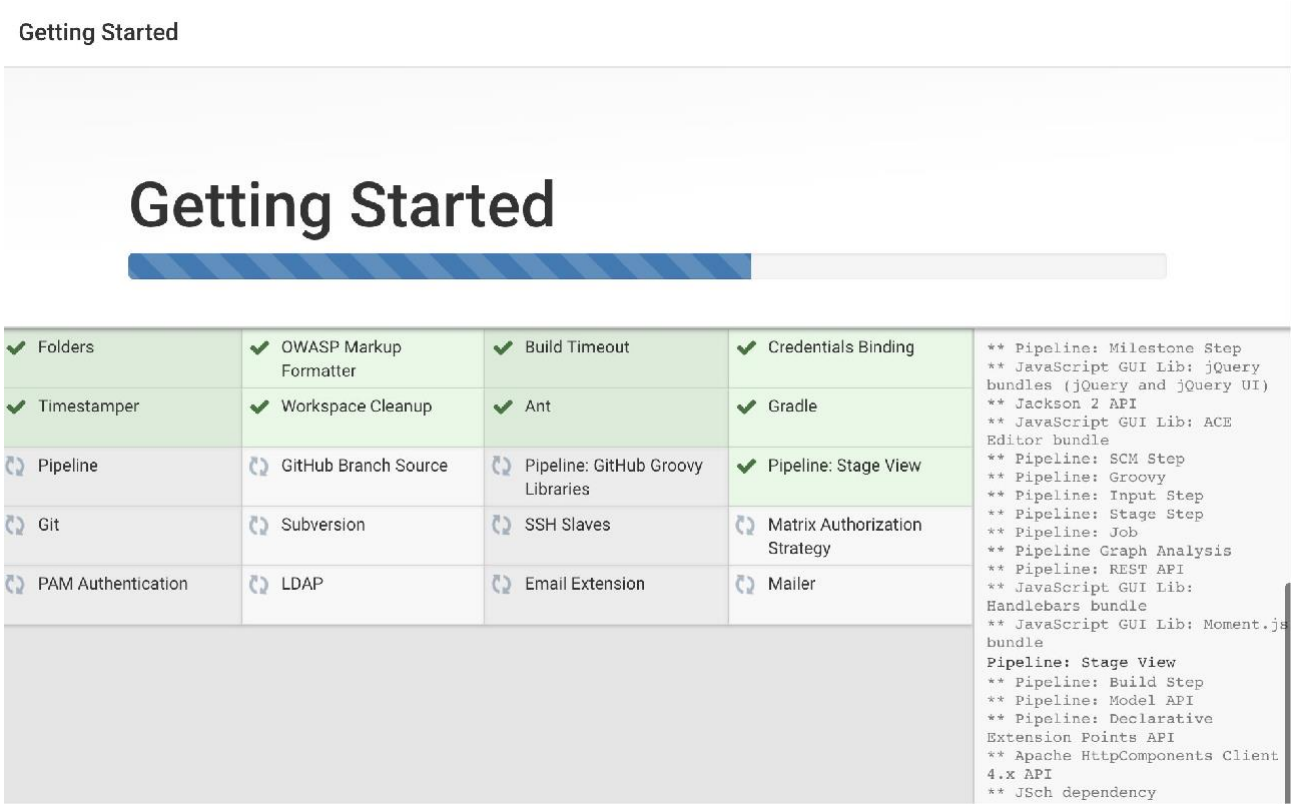
In the terminal window, use the cat command to display the password:

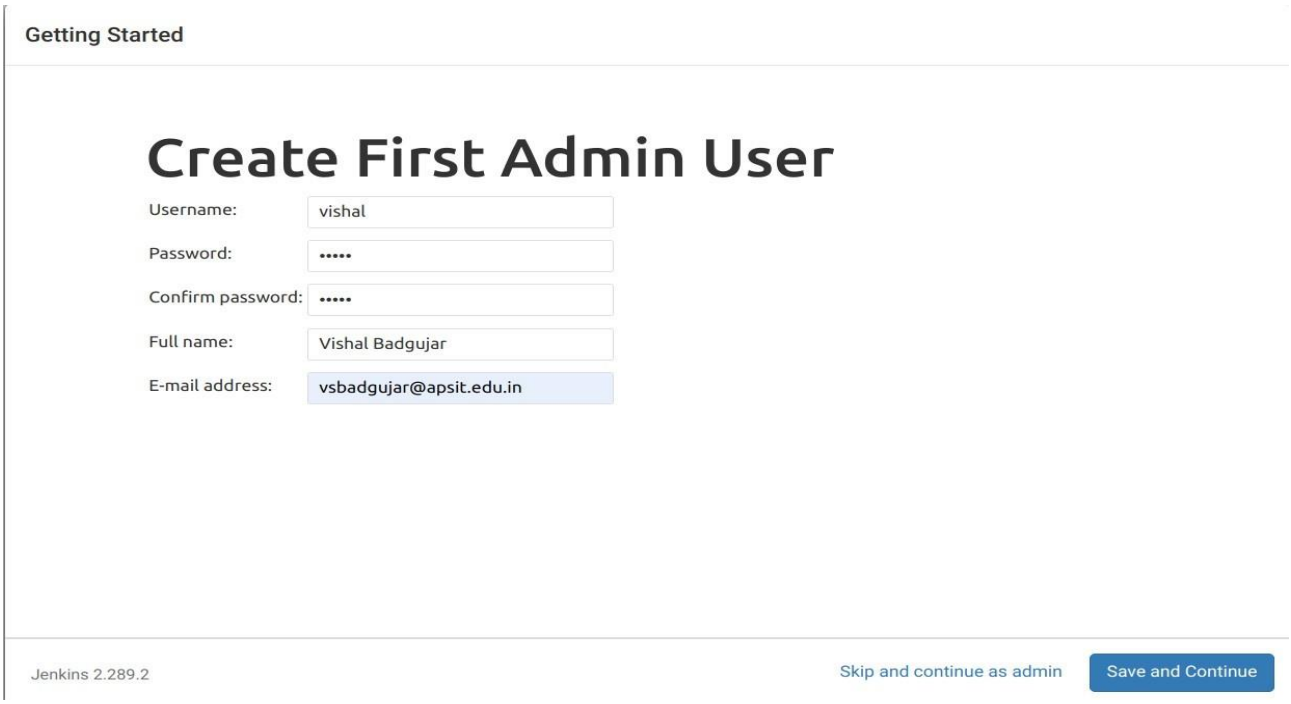
manjusha@apsit:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Copy the 32-character alphanumeric password from the terminal and paste it into the Administrator password field, then click Continue.

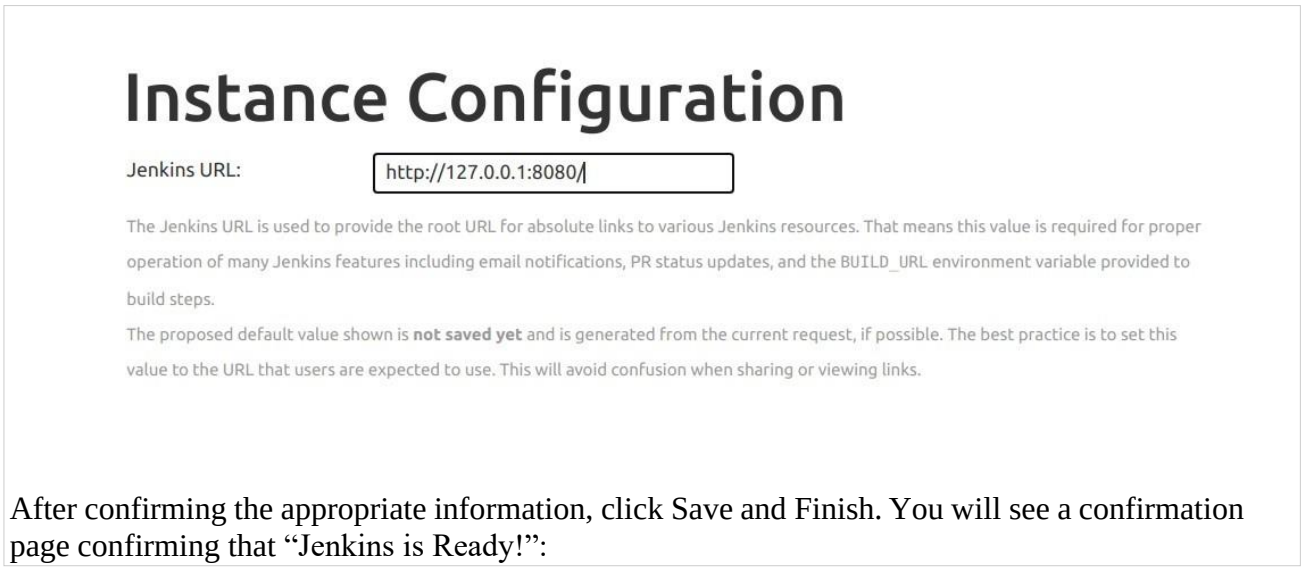
The next screen presents the option of installing suggested plugins or selecting specific plugins:

We'll click the Install suggested plugins option, which will immediately begin the installation process:

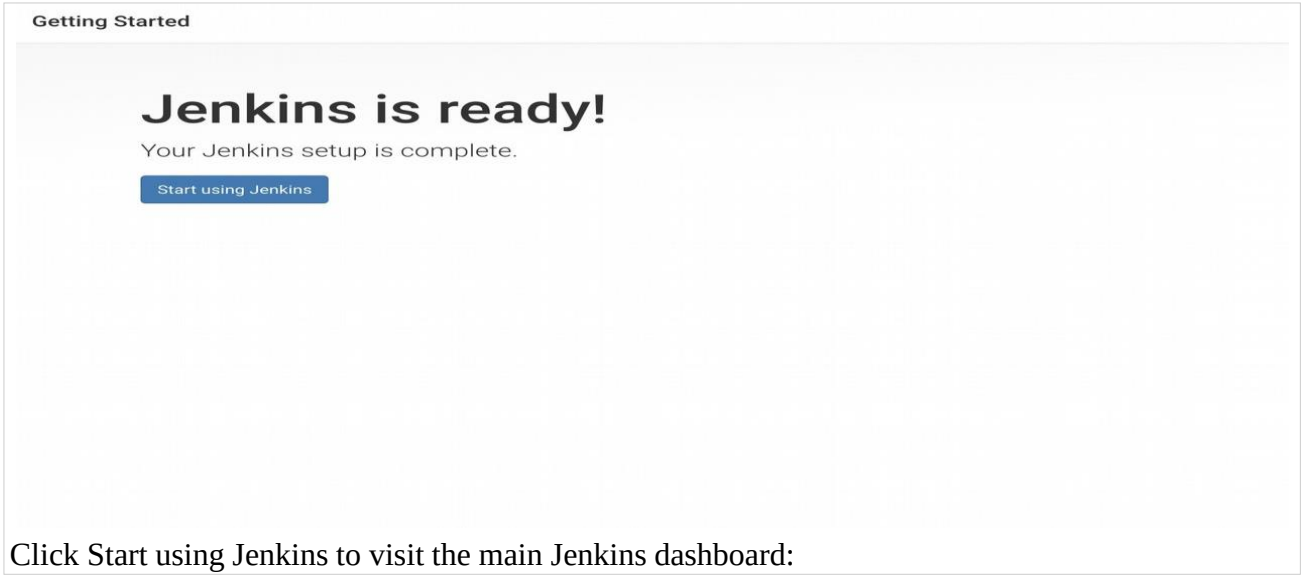




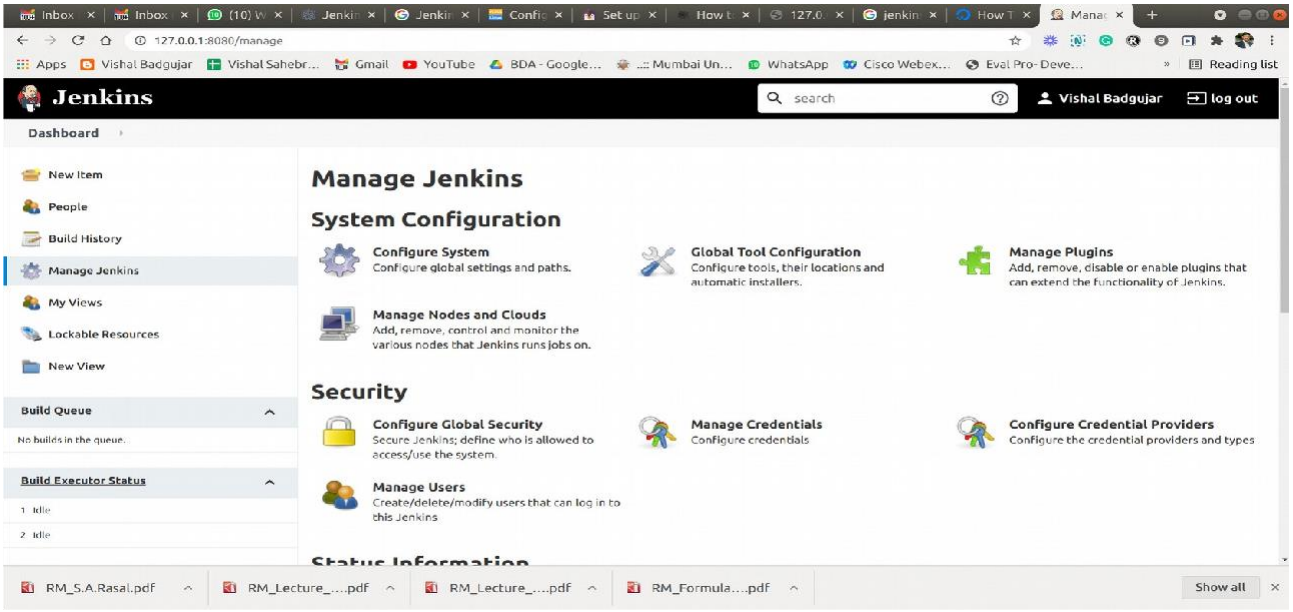
When the installation is complete, you will be prompted to set up the first administrative user. It's possible to skip this step and continue as admin using the initial password we used above, but we'll take a moment to create the user.



After confirming the appropriate information, click Save and Finish. You will see a confirmation page confirming that "Jenkins is Ready!":



Click Start using Jenkins to visit the main Jenkins dashboard:



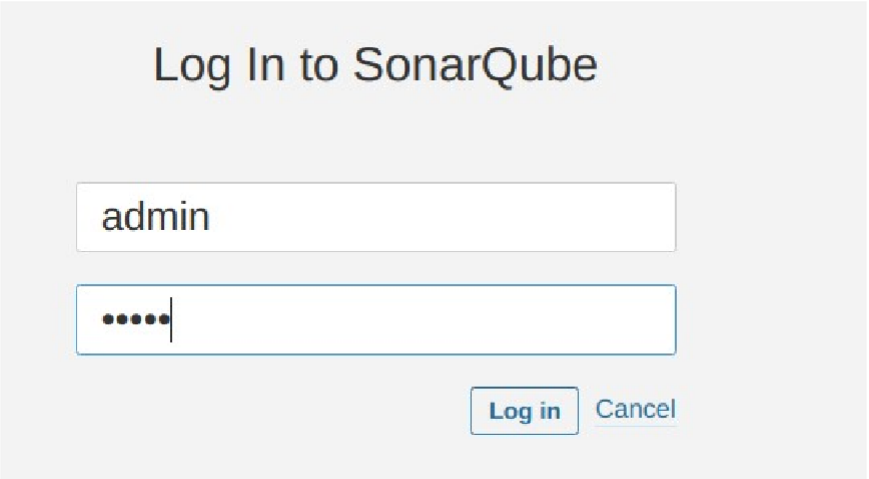
SonarQube Setup

Before proceeding with the integration, we will setup SonarQube Instance. we are using SonarQube Docker Container.

manjusha@apsit:~$docker run -d -p 9000:9000 sonarqube

```
vishal@apsit:~$ sudo docker run -d -p 9000:9000 sonarqube
Unable to find image 'sonarqube:latest' locally
latest: Pulling from library/sonarqube
5843afab3874: Pull complete
a131164fad71: Pull complete
d77763c1bc70: Pull complete
572e2a545fb3: Pull complete
f32e9b0d93df: Pull complete
Digest: sha256:d1f18c804d8bdcea0a90d13d93f6ec9af9012d48747fcb63dff
b7c8f06b5666f
Status: Downloaded newer image for sonarqube:latest
cf5325b4e2e80064d0d8faf76c8600ddd13cc26a5892074cac09674185f72fdc
vishal@apsit:~$
```

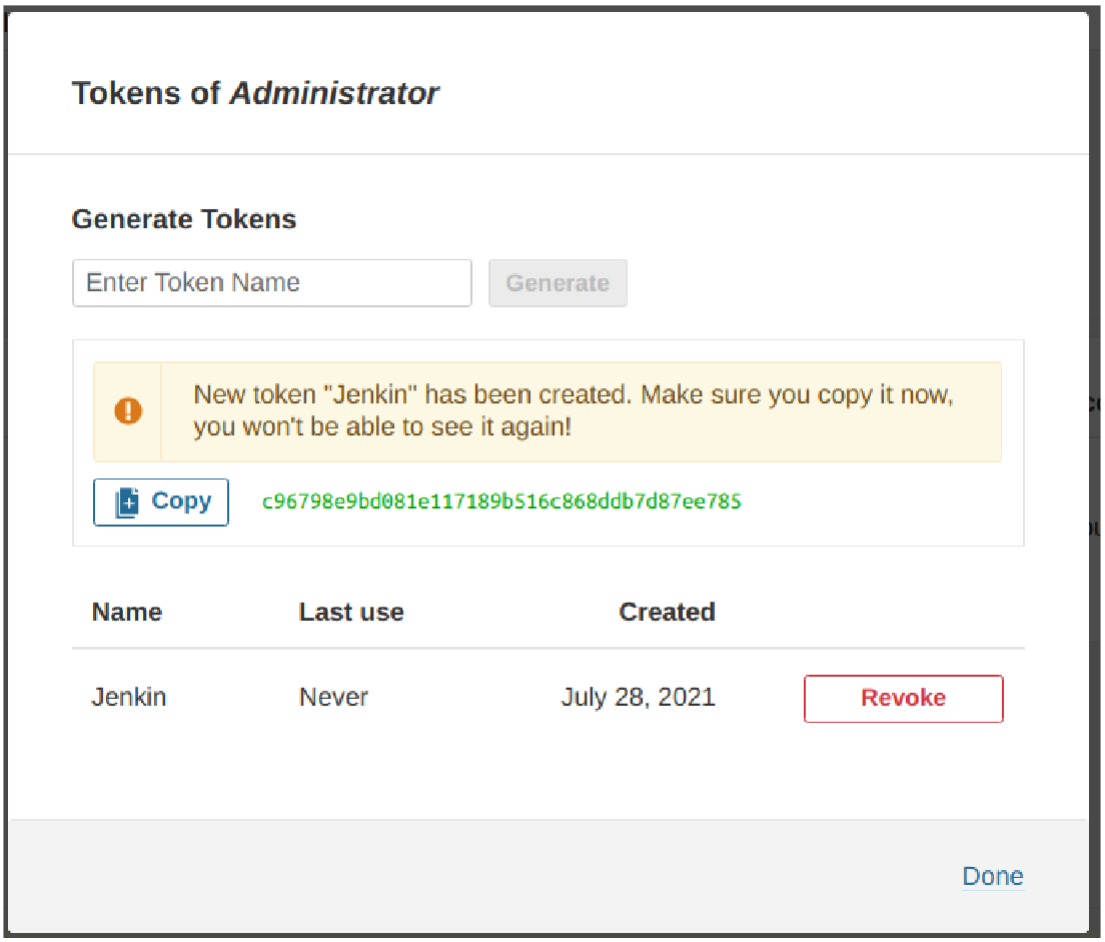
In the above command, we are forwarding port 9000 of the container to the port 9000 of the host machine as SonarQube is will run on port 9000. Then, from the browser, enter http://localhost:9000. After That, you will see the SonarQube is running. Then, login using default credentials (admin:admin).



Generate User Token

Now, we need to get the SonarQube user token to make connection between Jenkins and SonarQube. For the same, go to **Administration> User > My Account > Security** and then, from the bottom of the page you can create new tokens by clicking the Generate Button. Copy the Token and keep it safe.

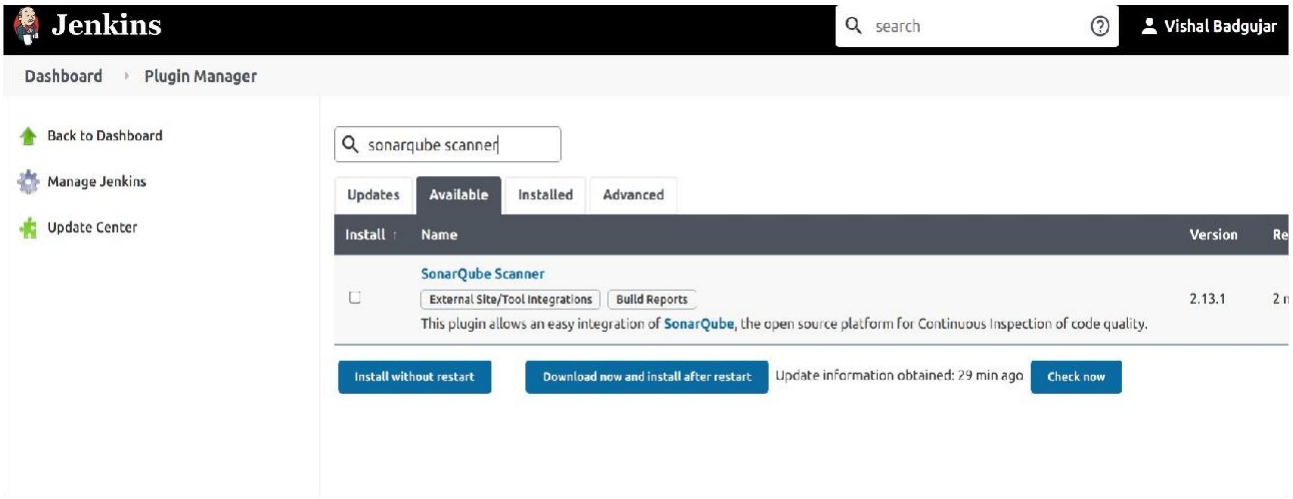
C96798e9bd081e117189b516c868ddb7d87ee785 SonarQube



2) Configure Jenkins with the SonarQube Scanner plugin for automated static code analysis.

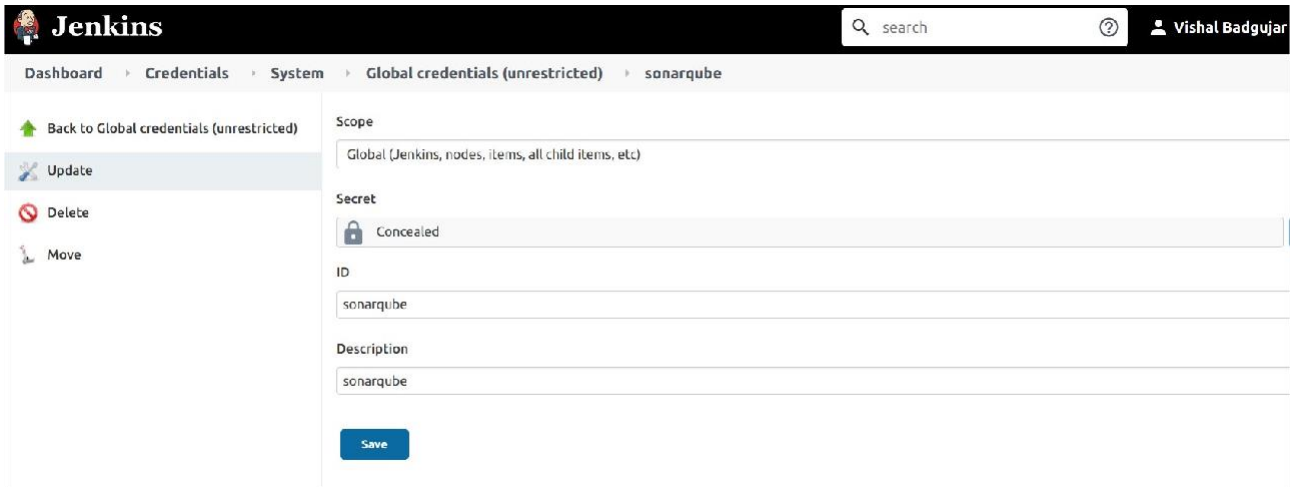
Jenkins Setup for SonarQube

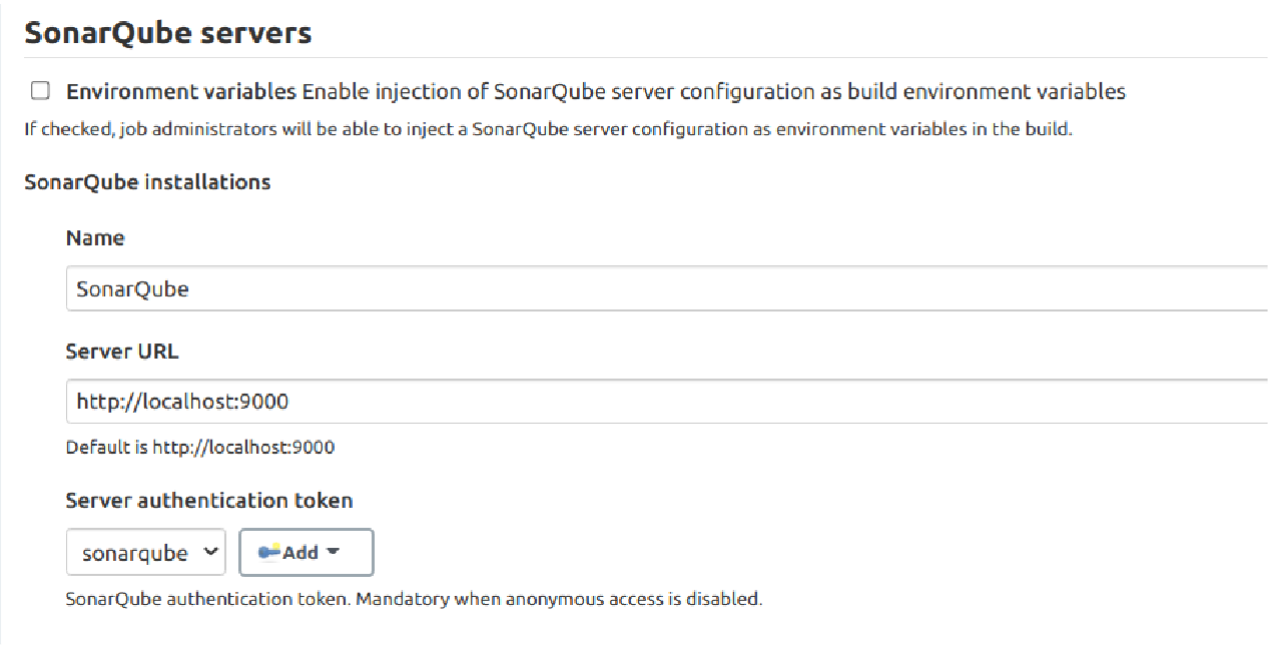
Before all, we need to install the SonarQube Scanner plugin in Jenkins. For the same, go to **Manage Jenkins > Plugin Manager > Available.** From here, type SonarQube Scanner then select and install.



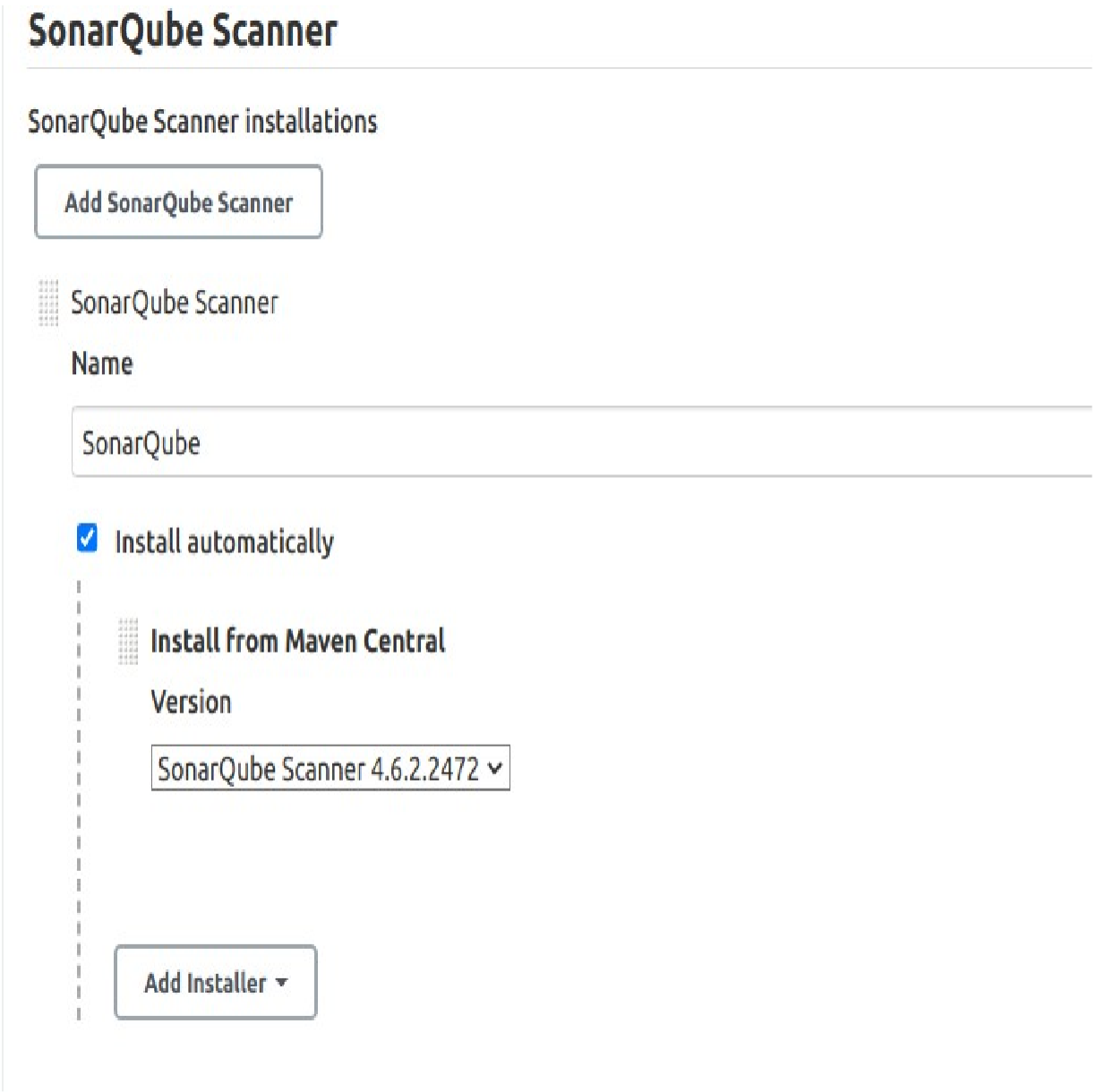
Tool Configuration SonarQube Scanner

Now, we need to configure the Jenkins plugin for SonarQube Scanner to make a connection with the SonarQube Instance. For that, got to **Manage Jenkins > Configure System > SonarQube Server.** Then, Add SonarQube. In this, give the Installation Name, Server URL then Add the Authentication token in the Jenkins Credential Manager and select the same in the configuration.



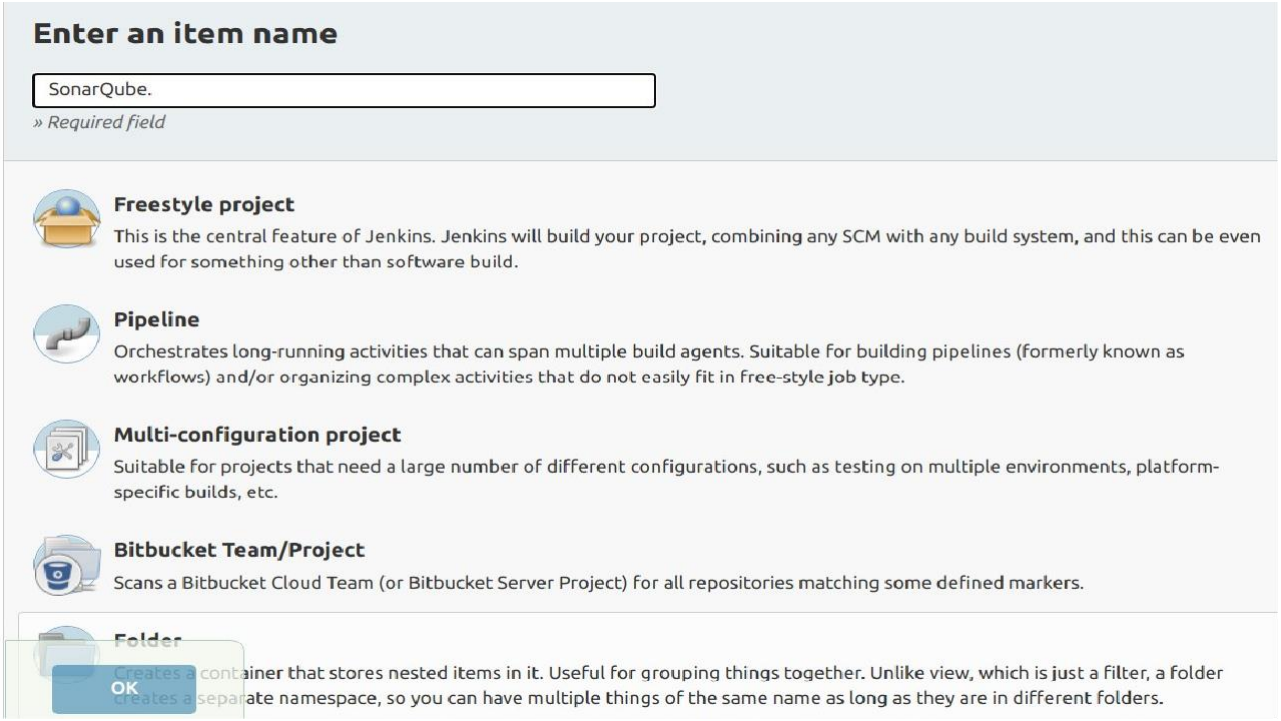


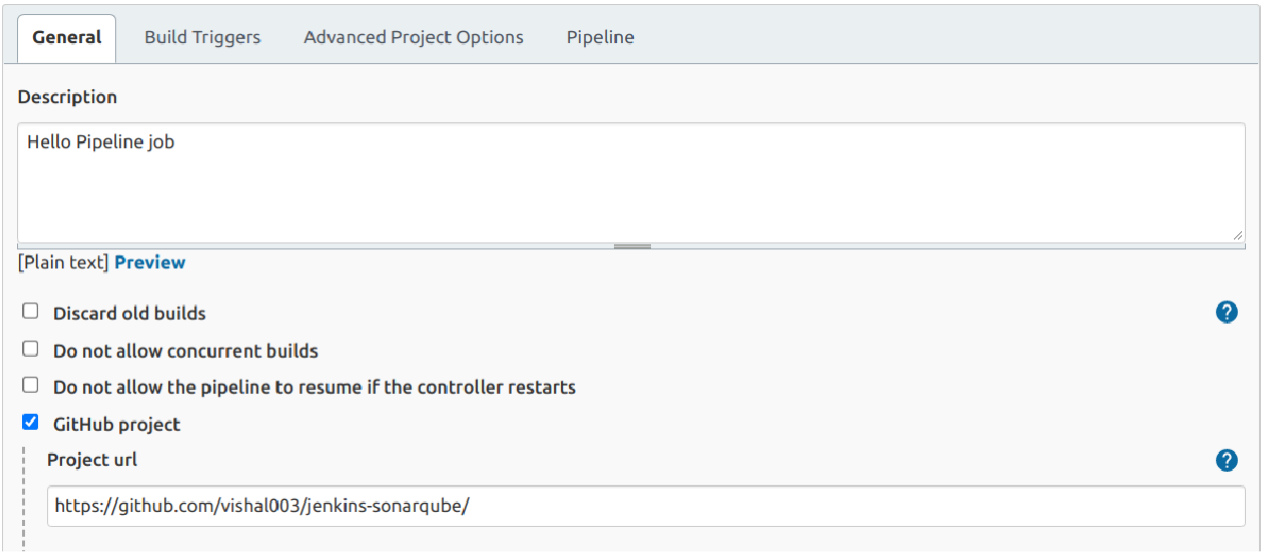
Then, we need to set-up the SonarQube Scanner to scan the source code in the various stage. For the same, go to **Manage Jenkins > Global Tool Configuration > SonarQube Scanner**. Then, Click **Add SonarQube Scanner Button**. From there, give some name of the scanner type and **Add Installer** of your choice. In this case, I have selected SonarQube Scanner from Maven Central.


SonarQube Scanner
SonarQube Scanner installations
Add SonarQube Scanner
SonarQube Scanner
Name
SonarQube
Install automatically
Install from Maven Central
Version
SonarQube Scanner 4.6.2.2472
Add Installer

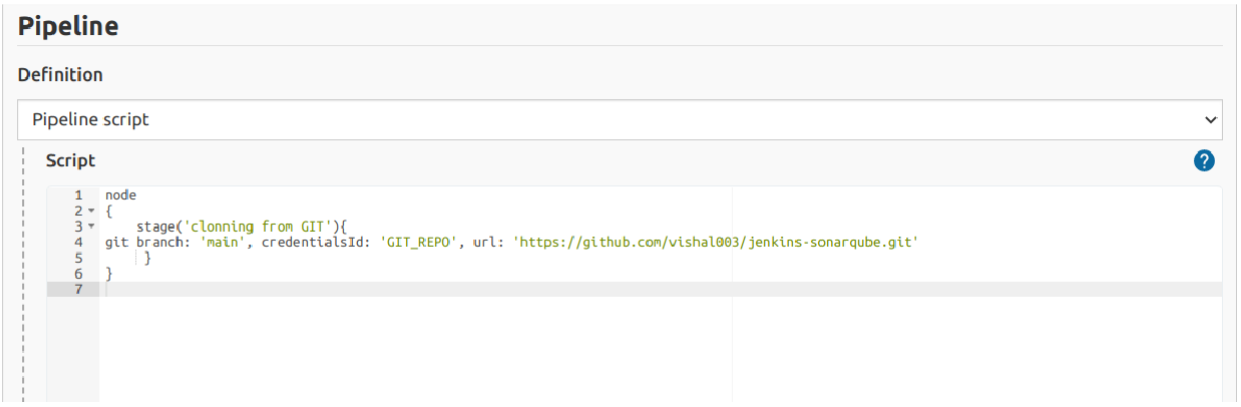
SonarQube Scanner in Jenkins Pipeline

Now, It's time to integrate the SonarQube Scanner in the Jenkins Pipeline. For the same, we are going to add one more stage in the Jenkinsfile called SonarQube and inside that, I am adding the following settings and code.

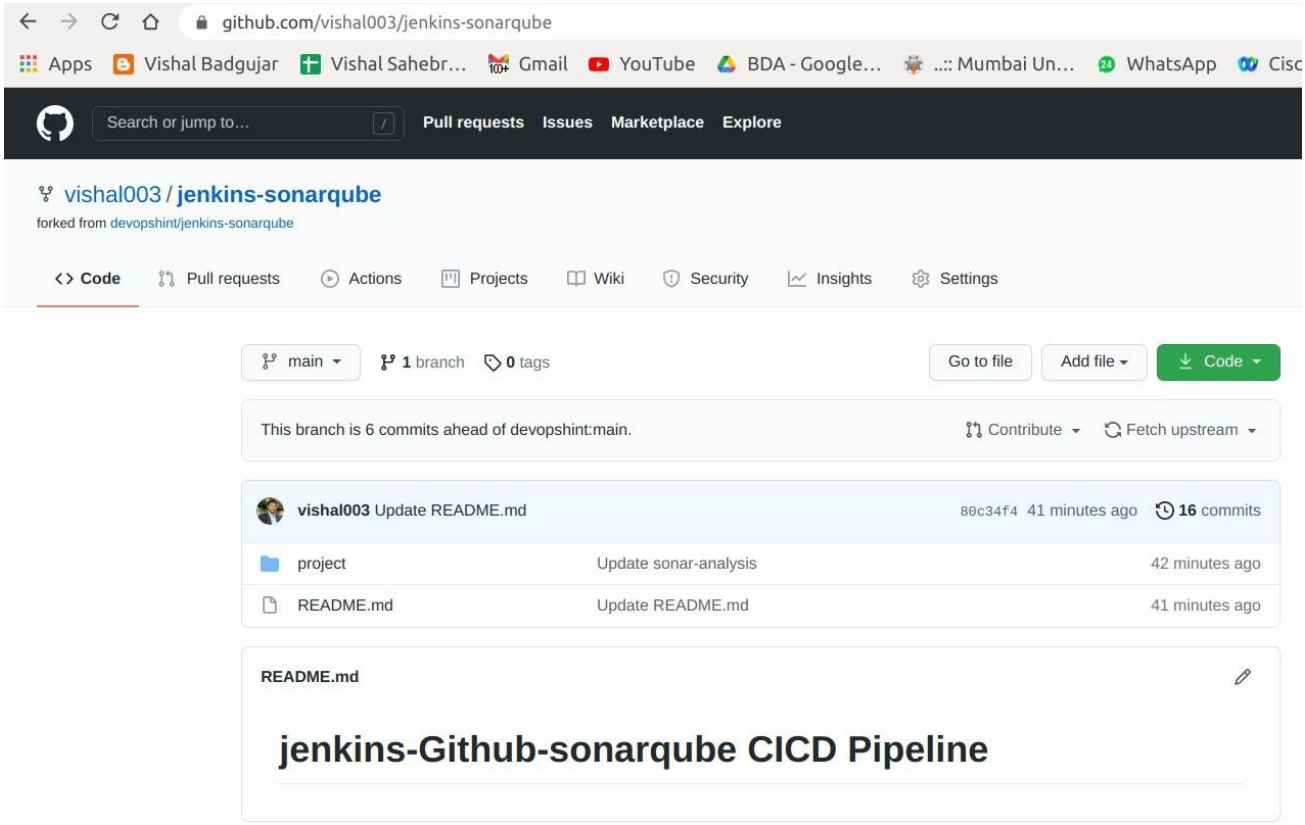




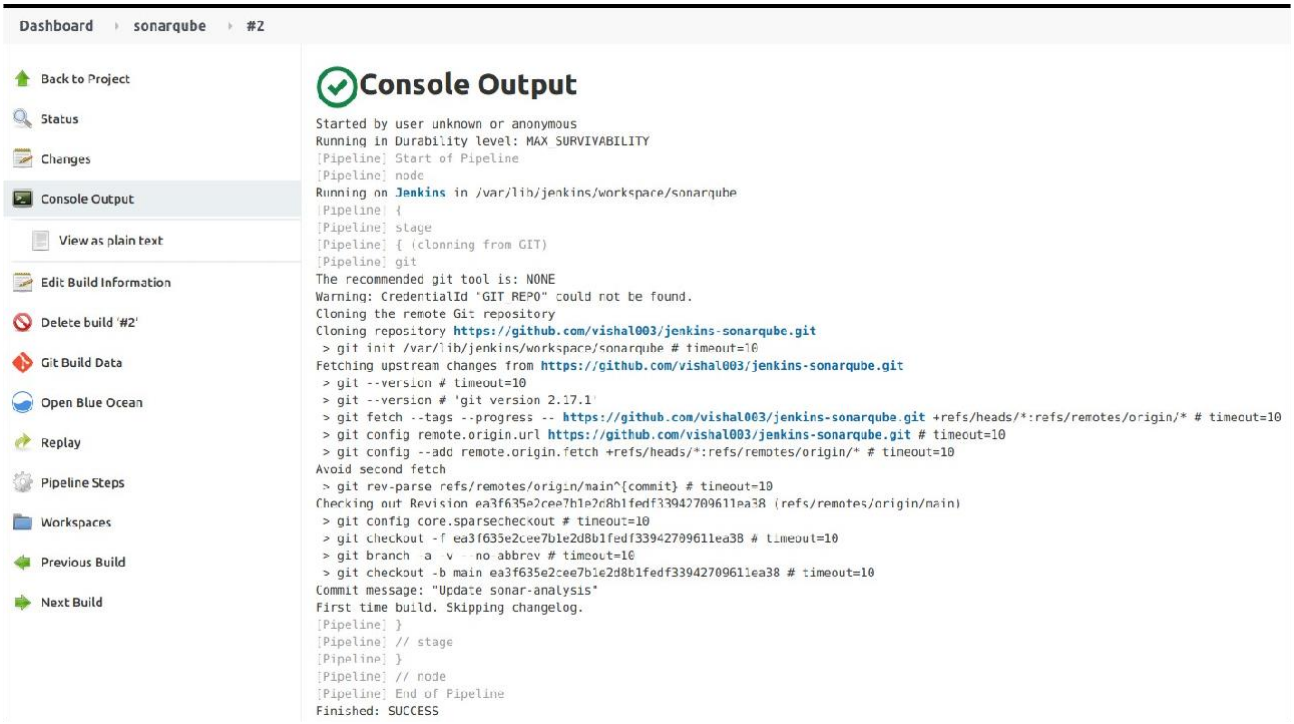
Github Configuration in Jenkins Pipeline



Git Clonning into Jenkins



Github Repository Contents



Successfully Build Github Repository in Jenkins

Pre-requiste required for Integration settings of Jenkins SAST with SonarQube we have done here successfully, now in order to Integrate of Jenkins CICD with SonarQube with the help of sample JAVA program we will implement in next experiment.

Conclusion: Write your own findings.

Academic Year: 2022-23 **Semester: V**
Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof. Manjusha K.

EXPERIMENT NO. 08

Aim: Create a Jenkins CICD Pipeline with SonarQube / GitLab Integration to perform a static analysis of the code to detect bugs, code smells, and security vulnerabilities on a sample Java application.

Theory:

Integrating Jenkins with SonarQube provides you with an automated platform for performing continuous inspection of code for quality and security assurance.

Everyday enhancement simplifies developers' tasks. Let's assume a situation where Developers have committed their codes to the repository, and then they want to know the project source code quality, code smells, any bugs, vulnerabilities, code analysis, etc. So it is extremely challenging for them to know all this information. So what if they want all these source codes and information beforehand? For such cases, Jenkins is the best fit. If a Software developer starts to build any new project, then the source code is automatically or manually saved while using Jenkins and their daily commit operation is not needed every time.

For this purpose, we can go for CI/CD i.e. Continuous Integration &Continuous Deployment of the code using SonarQube-Jenkins Integration.

SonarQube :

SonarQube is an open-source platform, which is used for continuous analysis of source code quality by performing analysis on your code to detect duplications, bugs, security vulnerabilities and code smells on programming languages.

Jenkins :

Jenkins an open-source automation tool is created using Java programming language. For the initial setup, it facilitates users with CI/CD(continuous integration (CI) or continuous delivery) technique that simplifies the use and management of processes. It is fundamentally focused on continuously

building and testing software projects for developers and to implement changes in real-time. In addition, it also allows users to plan a new build whenever the need arises.

Steps:

1) Install and configure a Jenkins and SonarQube CICD environment using Docker containers.

2) Configure Jenkins with the SonarQube Scanner plugin for automated static code analysis.

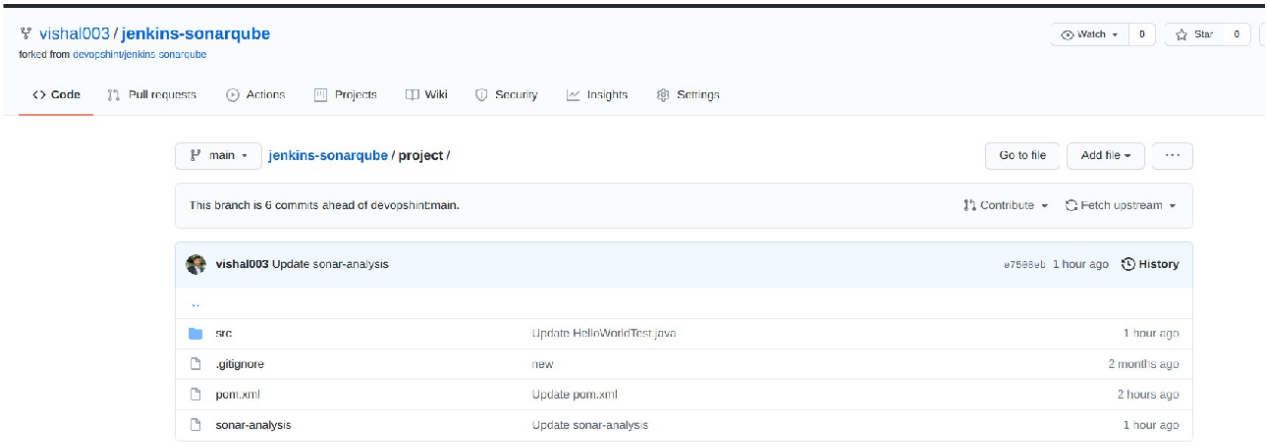
3) Create and set up a Jenkins build pipeline using a Jenkinsfile stored within a GitHub repo.

4) Use the SonarQube web application to examine and review the generated static analysis report.

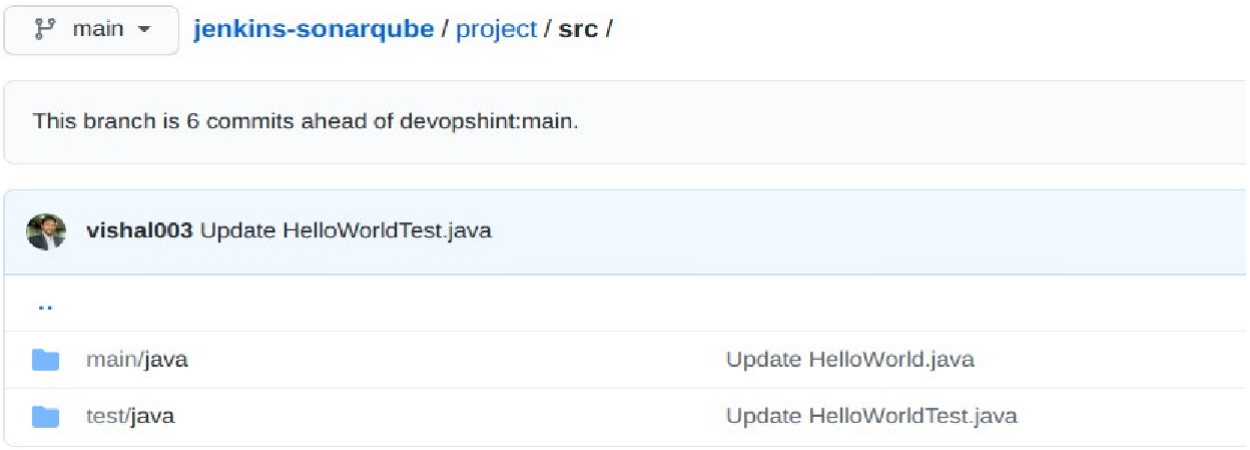
5) Use the Blue Ocean Plugin to review Pipeline Steps.

Note: From Step 1 and 2 we have already done in Expt. 7 as a Pre-requiste required for Integration settings of Jenkins SAST with SonarQube so in this Experiment we will continue from 3rd Step.

Check the contents of jenkins-sonarqube repository which we are using for Pipeline Project.



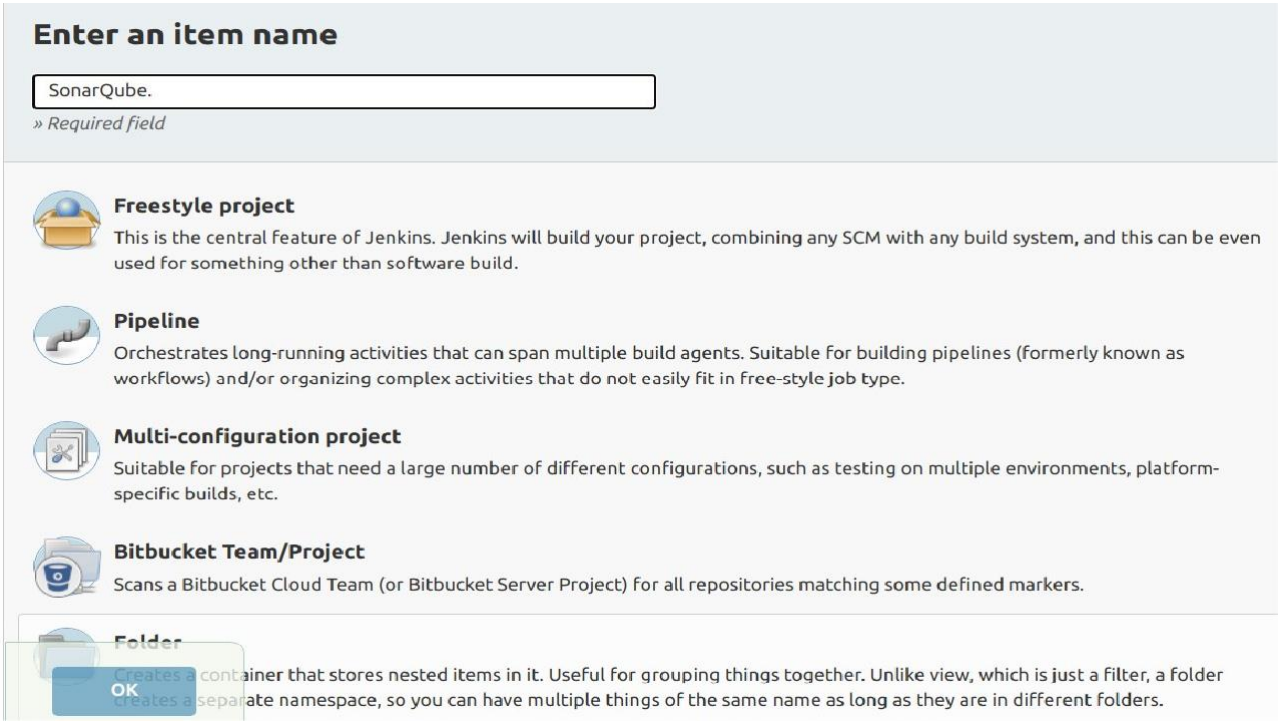
Check path for the Source and Test Java Programs from repository



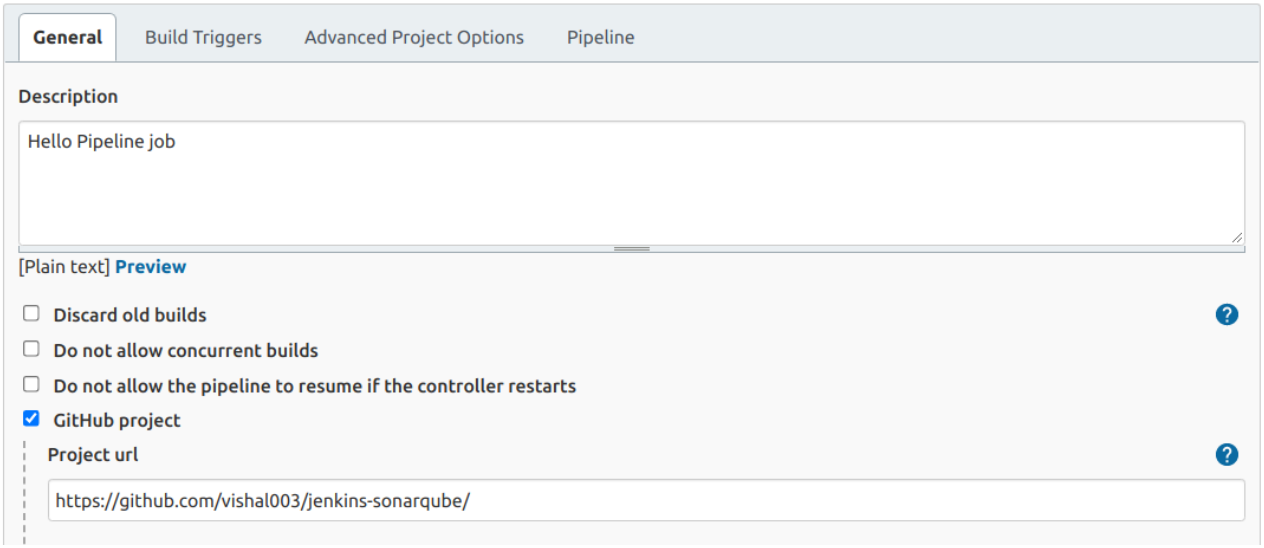
Provide sonar host as http://127.0.0.1:9000 in POM.xml which is available in Project on Github.

```
    <activation>
        <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
        <!-- Optional URL to server. Default value is http://localhost:9000 -->
        <sonar.host.url>
            http://127.0.0.1:9000/
        </sonar.host.url>
    </properties>
</profile>
<profile>
    <id>coverage</id>
```

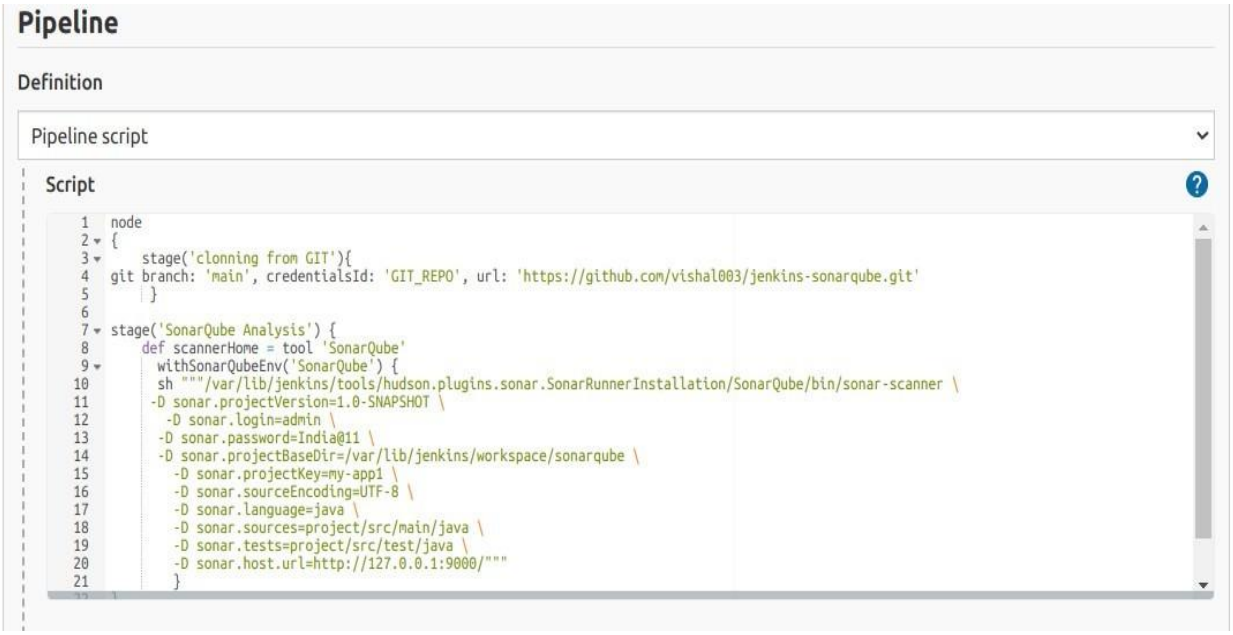
To integrate the SonarQube Scanner in the Jenkins Pipeline. For the same, we are going to add one more stage in the Jenkinsfile called SonarQube and inside that, I am adding the following settings and code.



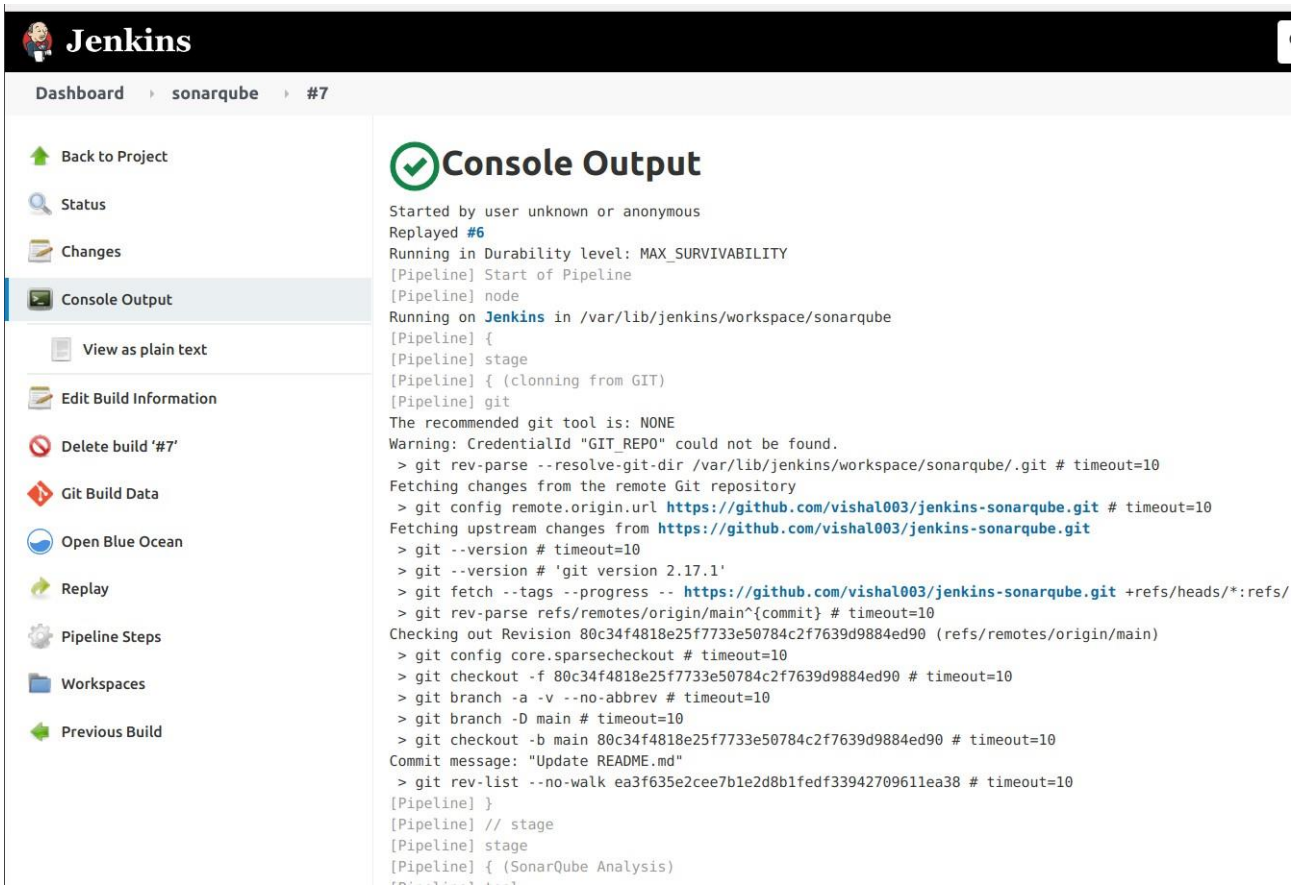
Github Repository Configuration in Jenkins Pipeline Project



Pipeline Script where stages are written along with scanner tool, repository path for source and test Java sample program, SonarQube Credential for integration, Application name on sonarqube, code language,etc.

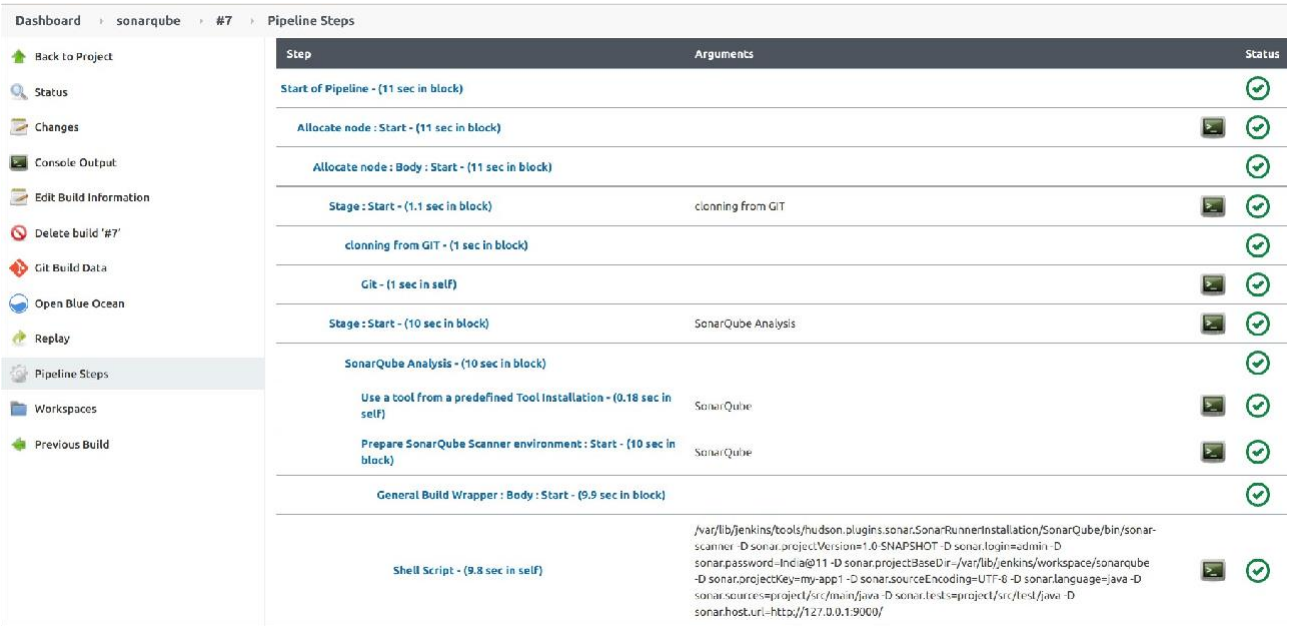


After creating a Pipeline Script Build it in Jenkins , Click on save and then Click on Build Now

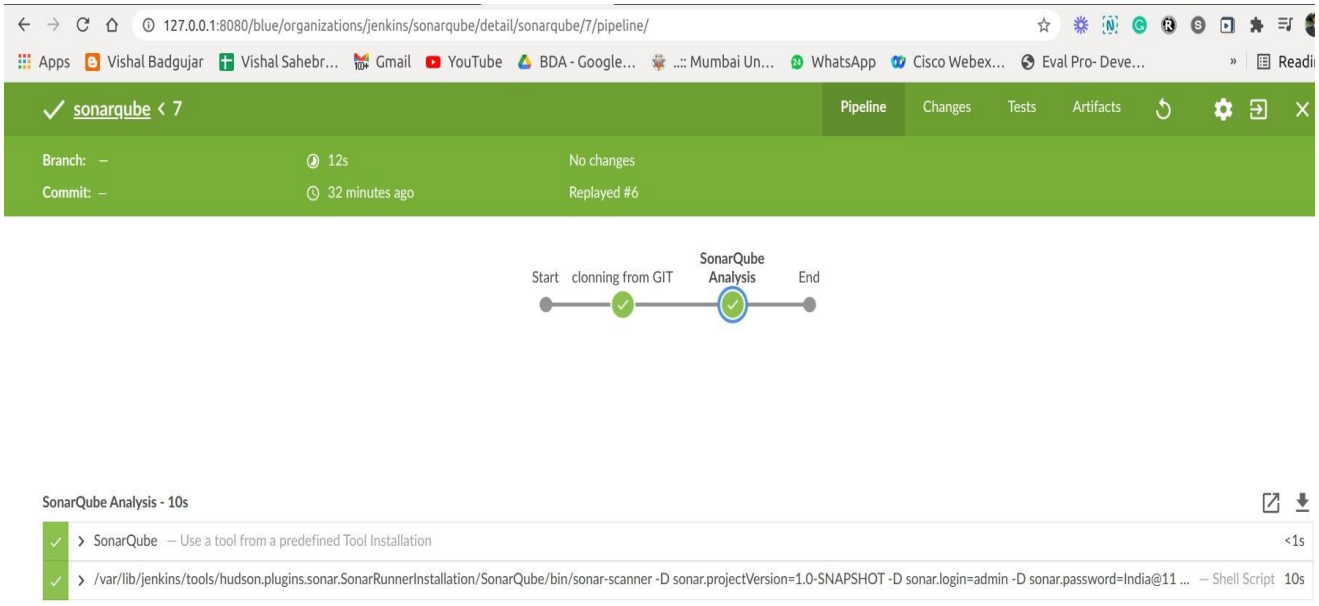


Click on Console Output to check output whether build is successful or not.

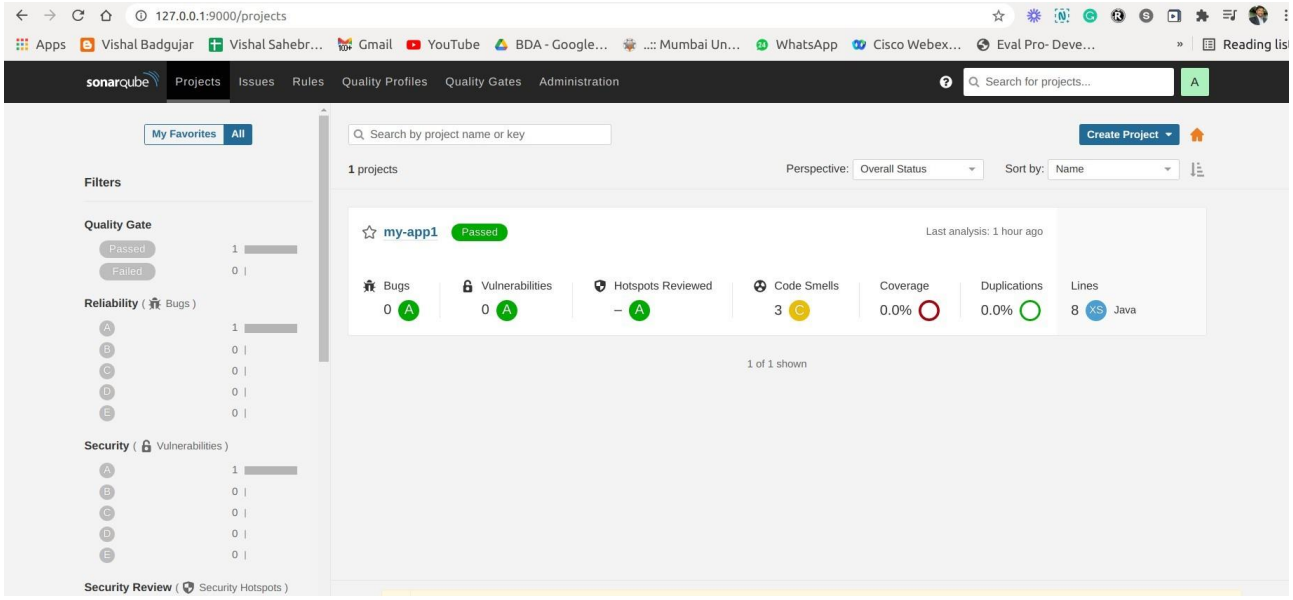
Click on Pipeline Steps to check Sequence of events during building of pipeline.



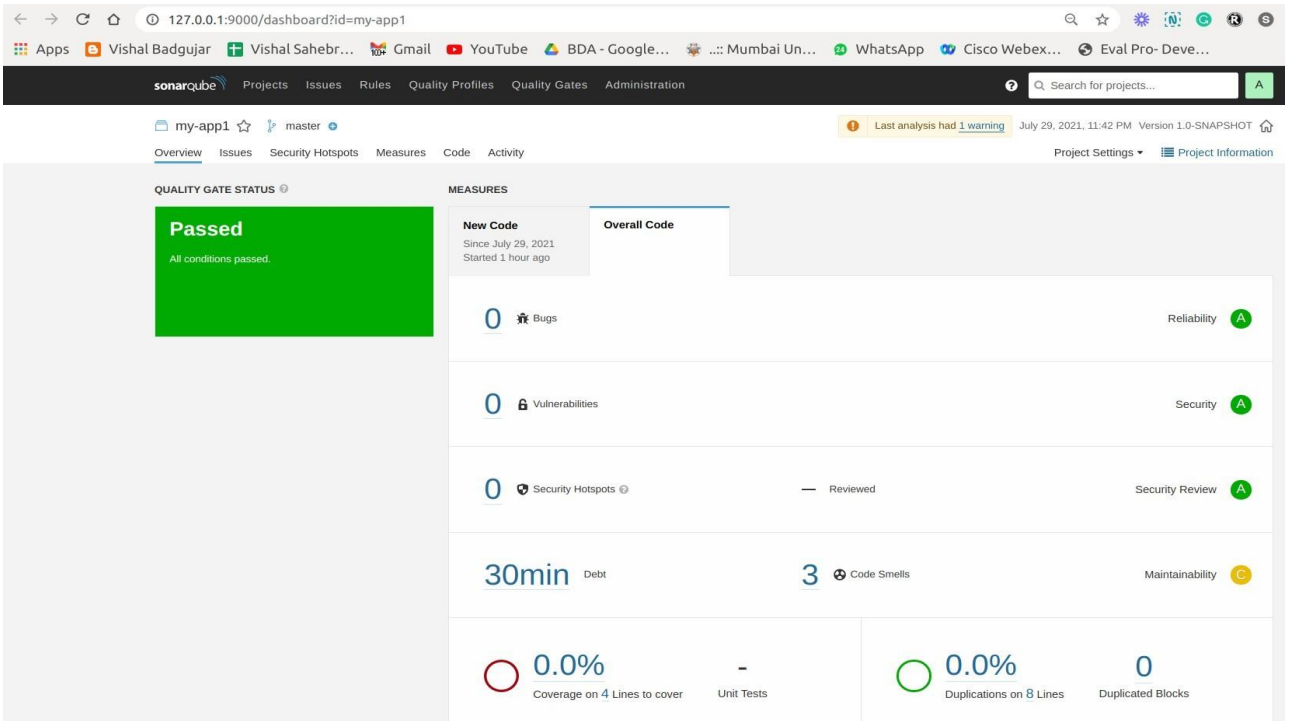
Also you can use Blue Ocean to check Pipeline execution stage by stage and log of the pipeline too.



If you login to the SonarQube and visit the Dashboard, you will see the Analysis of the project there.



For Detailed Report for code analysis you can go to application overview and check for all Bugs, Vulnerabilities, code smells and all parameters as shown in below image.



Since we have both Jenkins and SonarQube in the Enterprise standard, we have a lot of features including the alert system. Where we can configure the Email, or Instance message Notification system for the findings in the SonarQube or Jenkins. In the best case, we can auto convert certain bugs or findings as ticket and assign to the respective developer as a one option.

Conclusion: Write your own findings.

Academic Year: 2021-22 Semester: V
Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof.Manjusha Kashilkar

EXPERIMENT NO. 09
Aim: To Understand Continuous monitoring and Installation and configuration of Nagios Core, Nagios Plugins and NRPE (Nagios Remote Plugin Executor) on Linux Machine.

Theory:

Continuous monitoring is a process to detect, report, respond all the attacks which occur in its infrastructure. Once the application is deployed into the server, the role of continuous monitoring comes in to play. The entire process is all about taking care of the company's infrastructure and respond appropriately.

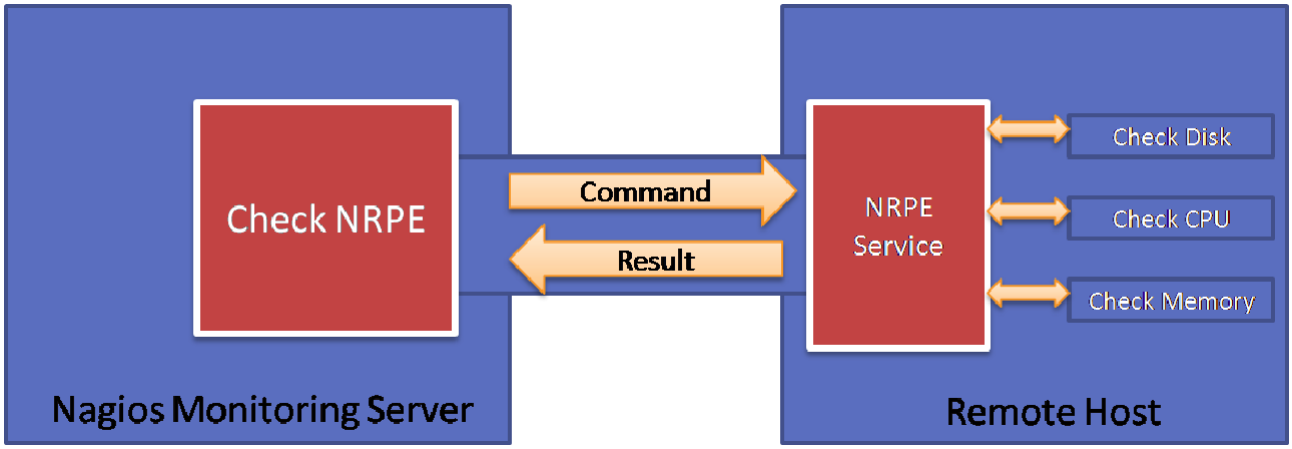
Why We Need Nagios tool?
Here, are the important reasons to use Nagios monitoring tool:

- Detects all types of network or server issues
- Helps you to find the root cause of the problem which allows you to get the permanent solution to the problem
- Active monitoring of your entire infrastructure and business processes
- Allows you to monitors and troubleshoot server performance issues
- Helps you to plan for infrastructure upgrades before outdated systems create failures
- You can maintain the security and availability of the service
- Automatically fix problems in a panic situation

- Nagios is the most popular, open source, powerful monitoring system for any kind of infrastructure. It enables organizations to identify and resolve IT infrastructure problems before they affect critical business processes. Nagios has the capability of monitoring application, services, entire IT infrastructure.

- NRPE is known as **Nagios Remote Plugin Executor**. The NRPE add-on is designed to execute plugins on remote Nix systems. In this setup, NRPE daemon is installed on the remote system to which services need to monitor through Nagios server.
- NRPE runs as a daemon on remote systems and waits for Nagios requests. When Nagios server needs to check the status of any resources or applications to that remote host, sends

and commands signal, which command definition is stored on NRPE service. NRPE takes Nagios server request and execute the command on the local system and sends the result back to Nagios.



1 - Pre-requisite

First requirement is to install Apache and PHP first. Use the following commands to complete it. And use commands to install required packages for Nagios.

manjusha@apsit:~$ sudo apt-get update
manjusha@apsit:~$ sudo apt-get install wget build-essential unzip openssl libssl- dev
manjusha@apsit:~$ sudo apt-get install apache2 php libapache2-mod-php php-gd libgd-dev

2 – Create Nagios User

Create a new user account for Nagios in your system and assign a password.
manjusha@apsit:~$ sudo adduser nagios

Now create a group for Nagios setup “nagcmd” and add nagios user to this group. Also, add nagios user in the Apache group.

manjusha@apsit:~$sudo groupadd nagcmd
manjusha@apsit:~$sudo usermod -a -G nagcmd nagios
manjusha@apsit:~$sudo usermod -a -G nagcmd www-data

Step 3 – Install Nagios Core Service

After installing required dependencies and adding user accounts and Nagios core installation. Download latest Nagios core service from the official site.

manjusha@apsit:~$cd /opt/

manjusha@apsit:~$sudo wget https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.4.3.tar.gz
manjusha@apsit:~$sudo tar xzf nagios-4.4.3.tar.gz

After extracting naviate to nagios source directory and install using make command.

manjusha@apsit:~$cd nagios-4.4.3

manjusha@apsit:~$sudo ./configure --with-command-group=nagcmd
manjusha@apsit:~$sudo make all
manjusha@apsit:~$sudo make install
manjusha@apsit:~$sudo make install-init
manjusha@apsit:~$sudo make install-daemoninit

manjusha@apsit:~$sudo make install-config
manjusha@apsit:~$sudo make install-commandmode
manjusha@apsit:~$sudo make install-exfoliation

Now copy event handlers scripts under libexec directory. These binaries provides multiple events triggers for your Nagios web interface.

manjusha@apsit:~$sudo cp -R contrib/eventhandlers/ /usr/local/nagios/libexec/

manjusha@apsit:~$sudo chown -R nagios:nagios/usr/local/nagios/libexec/eventhandlers

Step 4 – Setup Apache with Authentication

Now create an Apache configuration file for your Nagios server as below:

manjusha@apsit:~$sudo nano /etc/apache2/conf-available/nagios.conf

Add below lines to nagios.conf file.

ScriptAlias /nagios/cgi-bin "/usr/local/nagios/sbin"
<Directory "/usr/local/nagios/sbin"> Options ExecCGI
AllowOverride None

Order allow,deny Allow from all
AuthName "Restricted Area" AuthType Basic
AuthUserFile /usr/local/nagios/etc/htpasswd.users Require valid-user
</Directory>

Alias /nagios "/usr/local/nagios/share"

<Directory "/usr/local/nagios/share"> Options None
AllowOverride None Order allow,deny Allow from all
AuthName "Restricted Area" AuthType Basic
AuthUserFile /usr/local/nagios/etc/htpasswd.users Require valid-user
</Directory>

To setup apache authentication for user **nagiosadmin**

manjusha@apsit:~$sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin

Enable Apache configuration and restart Apache service to make the new settings take effect.

manjusha@apsit:~$sudo a2enconf nagios manjusha@apsit:~$sudo a2enmod cgi rewrite
manjusha@apsit:~$sudo service apache2 restart

Step 5 – Installing Nagios Plugins
After installing and configuring Nagios core service, Download latest nagios-plugins source and install using following commands.
manjusha@apsit:~$cd /opt
manjusha@apsit:~$sudo wget http://www.nagios-plugins.org/download/nagios-plugins- 2.2.1.tar.gz
manjusha@apsit:~$sudo tar xzf nagios-plugins-2.2.1.tar.gz manjusha@apsit:~$cd nagios-plugins-2.2.1

Now compile and install Nagios plugins
manjusha@apsit:~$sudo ./configure --with-nagios-user=nagios --with-nagios- group=nagios --with-openssl
manjusha@apsit:~$sudo make
manjusha@apsit:~$sudo make install

Step 6 – Verify Settings
Use the Nagios commands to verify the Nagios installation and configuration file. After successfully verify start the Nagios core service.
manjusha@apsit:~$/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
manjusha@apsit:~$ sudo service nagios start

Also configure Nagios to auto start on system boot.

Step 7 – Access Nagios Web Interface
Access your nagios setup by access nagios server using hostname or ip address followed by /nagios.
http://127.0.0.1/nagios/
Prompting for Apache Authentication Password –
username: nagiosadmin
Password : 123456 (which you enter while configuration)

Nagios After login screen –

We have successfully installed and configured Nagios Monitoring Server core service in our system now we need to install NRPE on all remote Linux systems to monitor with Nagios.

Conclusion: Write your own findings.

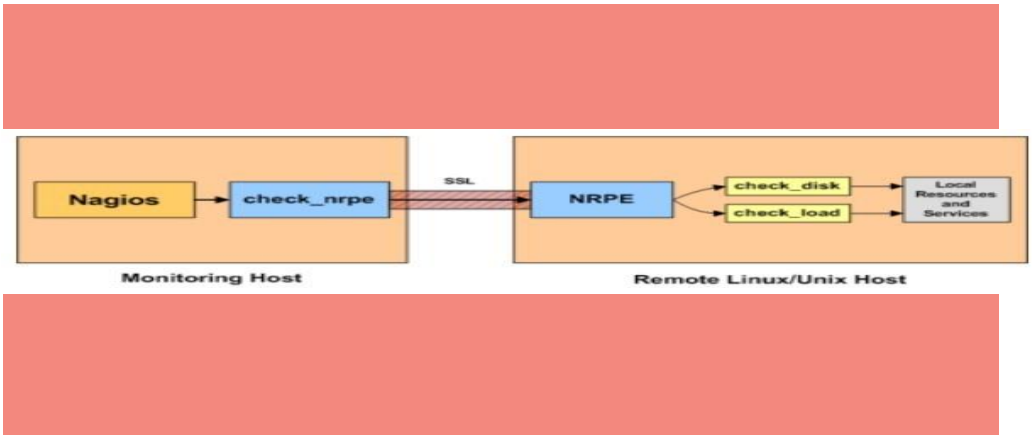
Academic Year: 2022-23 **Semester: V**
Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof.Manjusha K.

EXPERIMENT NO. 10

Aim: To perform Port, Service monitoring, Linux server monitoring using Nagios.

Theory:

Monitoring remote Linux/Unix hosts is to use the NRPE addon. NRPE allows you to execute plugins on remote Linux/Unix hosts. This is useful if you need to monitor local resources/attributes like disk usage, CPU load, memory usage, etc. on a remote host.



Note: To perform this experiment Experiment 9 is pre-requisite where we have configured Nagios on Linux System. Here In this Experiment we will Add a Linux Host to Nagios for Monitoring purpose.

Step 1 – Configure NRPE on Linux Host

Follow the below steps to install and configure NRPE on client machine and check connectivity with Nagios server.

Step 1.1 – Install NRPE

```
vishal@apsit:~$  sudo apt-get install nagios-nrpe-server nagios-plugins
```

Step 1.2 – Configure NRPE

After successfully installing NRPE service, Edit nrpe configuration file /etc/nagios/nrpe.cfg in your favorite editor and add your nagios service ip in allowed hosts.

```
vishal@apsit:~$ sudo nano /etc/nagios/nrpe.cfg
```

```
allowed_hosts=127.0.0.1, 192.168.64.3, 192.168.1.100
```

Where **192.168.1.100** is your Nagios server ip address.

After making above changes in nrpe configuration file, Lets restart NRPE service as per your system

```
vishal@apsit:~$ sudo /etc/init.d/nagios-nrpe-server restart
```

Step 1.3 – Verify Connectivity from Nagios

Now run the below command from Nagios server to make sure your nagios is able to connect nrpe client on remote Linux system. Here **192.168.64.3** is your remote Linux system ip.

```
vishal@apsit:~$ /usr/local/nagios/libexec/check_nrpe -H 192.168.64.3
NRPE v2.15
```

Step 2 – Add Linux Host in Nagios

First create a configuration file using below values. for example you Linux hosts ip is . We also need to define a service with host. So add a ping check service, which will continuously check that host is up or not.

```
vishal@apsit:~$  sudo nano /usr/local/nagios/etc/servers/MyLinuxHost001.cfg
```

```
define host {
        use                             linux-server
        host_name                       Linux_Host_001
        alias                           Linux Host 001
        address                         192.168.64.3
        register                        1
}
define service{
      host_name                          Linux_Host_001
      service_description                PING
      check_command                      check_ping!100.0,20%!500.0,60%
      max_check_attempts                 2
      check_interval                     2
```

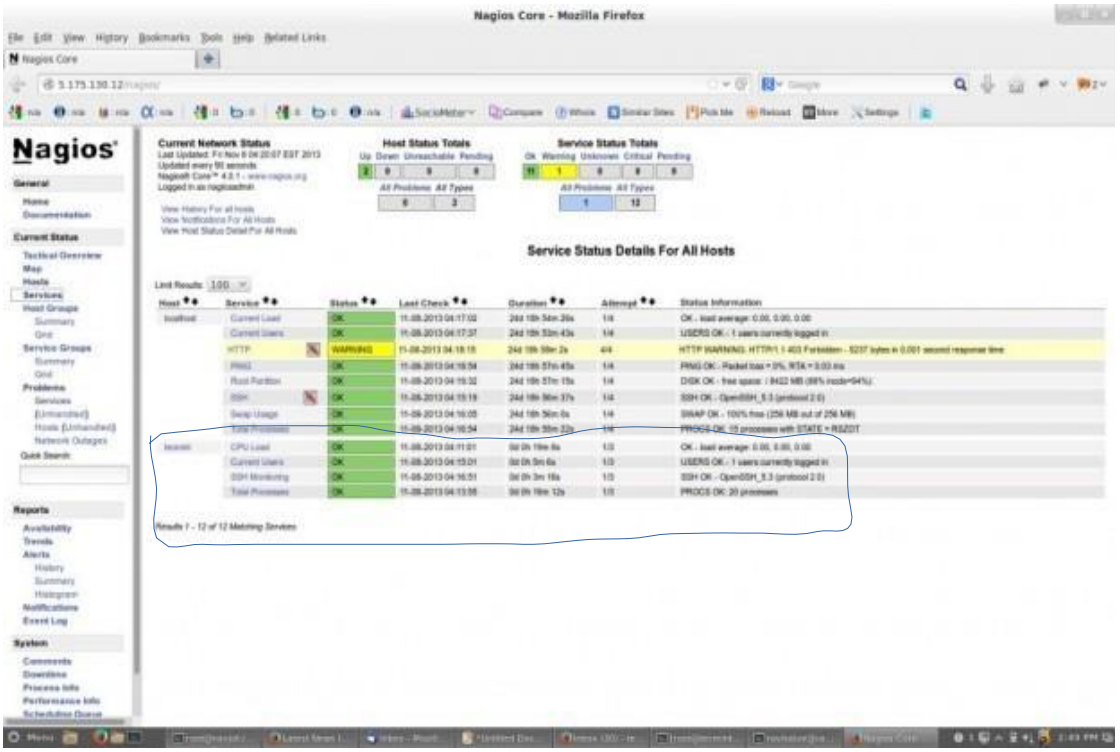
```
        retry_interval                  2
        check_period                    24x7
        check_freshness                 1
        contact_groups                  admins
        notification_interval           2
        notification_period             24x7
        notifications_enabled           1
        register                        1
}
```

Now verify configuration files using following command. If there are no errors found in configuration, restart nagios service.

```
vishal@apsit:~$ sudo nagios -v /usr/local/nagios/etc/nagios.cfg
vishal@apsit:~$ sudo service nagios restart
```

Step 3 – Check Host in Nagios Web Interface

Open your Nagios web interface and check for new Linux hosts added in Nagios core service.



Conclusion: Write your own findings.

Academic Year: 2023-24 **Semester: V**
Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof. Manjusha Kashilkar

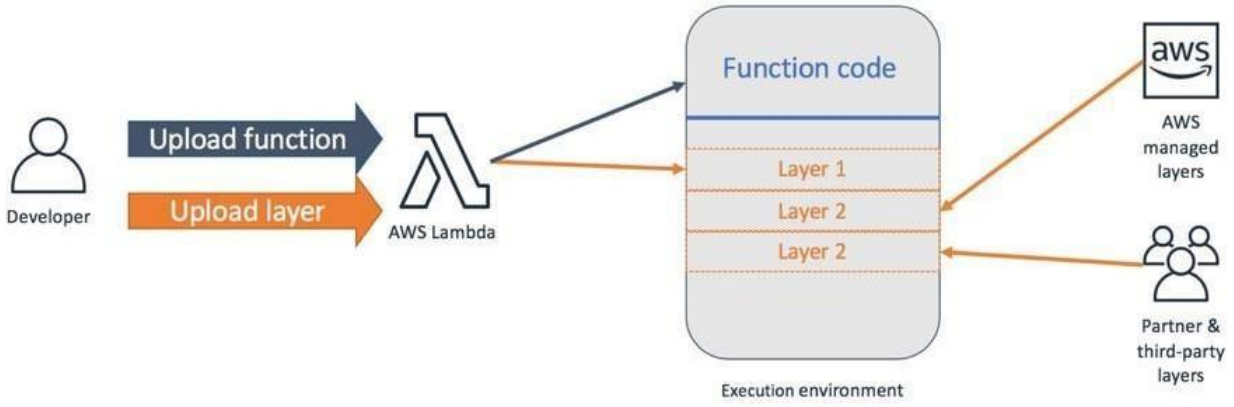
EXPERIMENT NO. 11

Aim: To understand AWS Lambda, its workflow, various functions and create your first Lambda functions using Python / Java / Nodejs.

Theory:

Lambda is a compute service that lets you run code without provisioning or managing servers. Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. With Lambda, you can run code for virtually any type of application or backend service. All you need to do is supply your code in one of the languages that Lambda supports.

You organize your code into Lambda functions. Lambda runs your function only when needed and scales automatically, from a few requests per day to thousands per second. You pay only for the compute time that you consume—there is no charge when your code is not running.



You can invoke your Lambda functions using the Lambda API, or Lambda can run your functions in response to events from other AWS services. For example, you can use Lambda to:

- Build data-processing triggers for AWS services such as Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB.
- Process streaming data stored in Amazon Kinesis.
- Create your own backend that operates at AWS scale, performance, and security.

What is a Lambda function?

The code you run on AWS Lambda is called a "Lambda function." After you create your Lambda function, it is always ready to run as soon as it is triggered, similar to a formula in a spreadsheet. Each function includes your code as well as some associated configuration information, including the function name and resource requirements. Lambda functions are "stateless", with no affinity to the underlying infrastructure, so that Lambda can rapidly launch as many copies of the function as needed to scale to the rate of incoming events.

After you upload your code to AWS Lambda, you can associate your function with specific AWS resources, such as a particular Amazon S3 bucket, Amazon DynamoDB table, Amazon Kinesis stream, or Amazon SNS notification. Then, when the resource changes, Lambda will execute your function and manage the compute resources as needed to keep up with incoming requests.

AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers, creating workload-aware cluster scaling logic, maintaining event integrations, or managing runtimes. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just upload your code as a ZIP file or container image, and Lambda automatically and precisely allocates compute execution power and runs your code based on the incoming request or event, for any scale of traffic. You can set up your code to automatically trigger from over 200 AWS services and SaaS applications or call it directly from any web or mobile app. You can write Lambda functions in your favorite language (Node.js, Python,

Go, Java, and more) and use both serverless and container tools, such as AWS SAM or Docker CLI, to build, test, and deploy your functions.

Benefits

1. No servers to manage

AWS Lambda automatically runs your code without requiring you to provision or manage infrastructure. Just write the code and upload it to Lambda either as a ZIP file or container image.

2. Continuous scaling

AWS Lambda automatically scales your application by running code in response to each event. Your code runs in parallel and processes each trigger individually, scaling precisely with the size of the workload, from a few requests per day, to hundreds of thousands per second.

3. Cost optimized with millisecond metering

With AWS Lambda, you only pay for the compute time you consume, so you're never paying for over-provisioned infrastructure. You are charged for every millisecond your code executes and the number of times your code is triggered. With Compute Savings Plan, you can additionally save up to 17%.

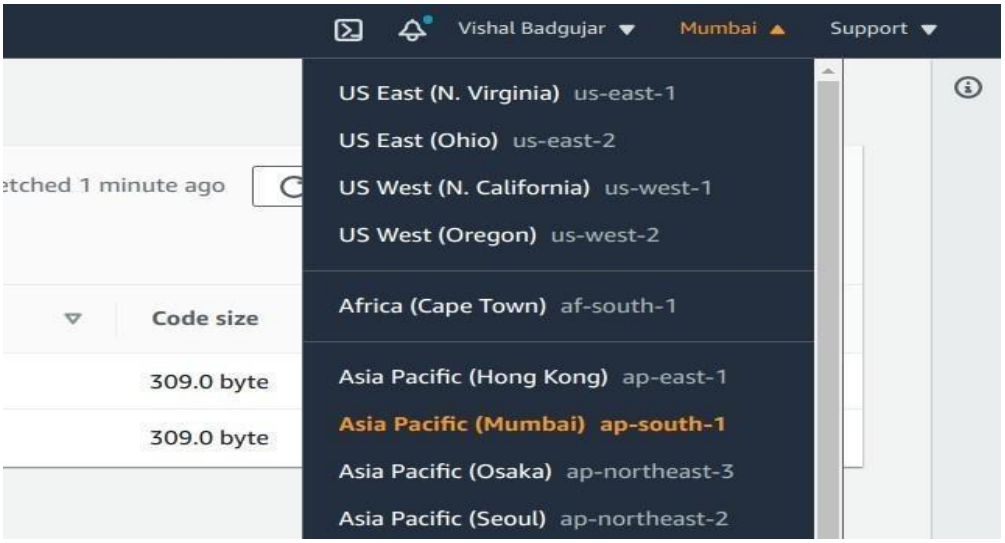
4. Consistent performance at any scale

With AWS Lambda, you can optimize your code execution time by choosing the right memory size for your function. You can also keep your functions initialized and hyper-ready to respond within double digit milliseconds by enabling Provisioned Concurrency.

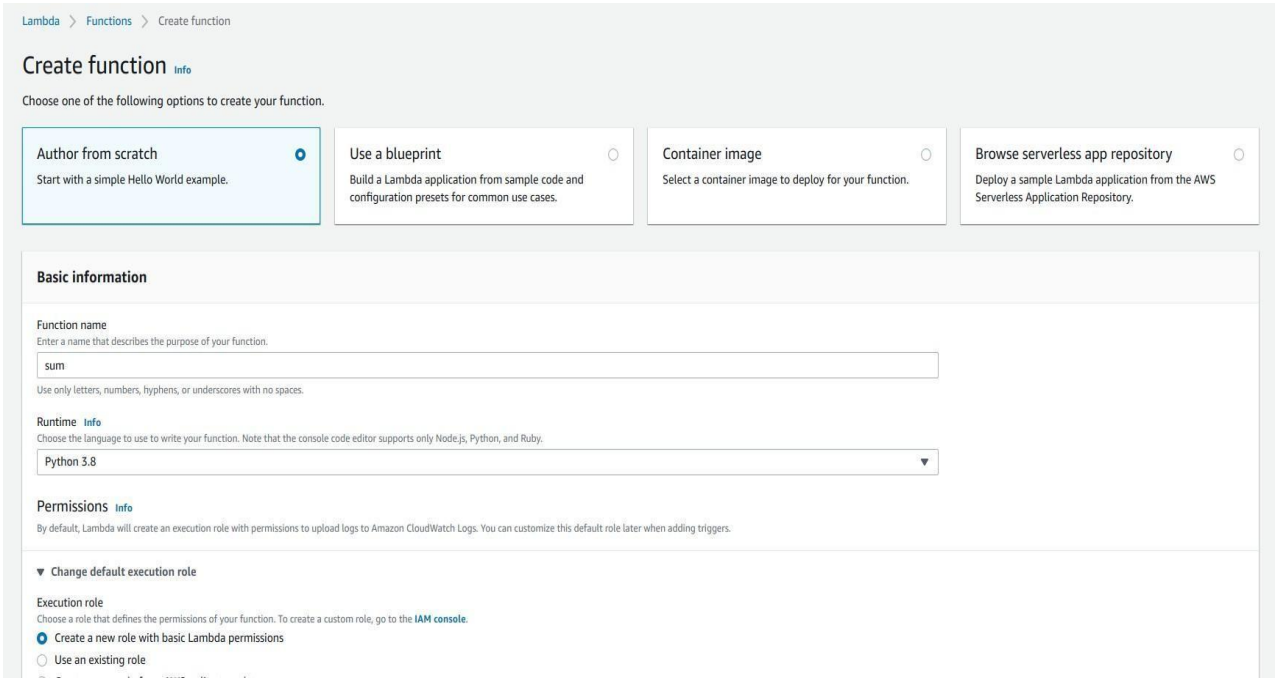
Steps: First Lambda functions using Python

1. Open Aws Console and search for Lambda Service and open home screen of Lambda.

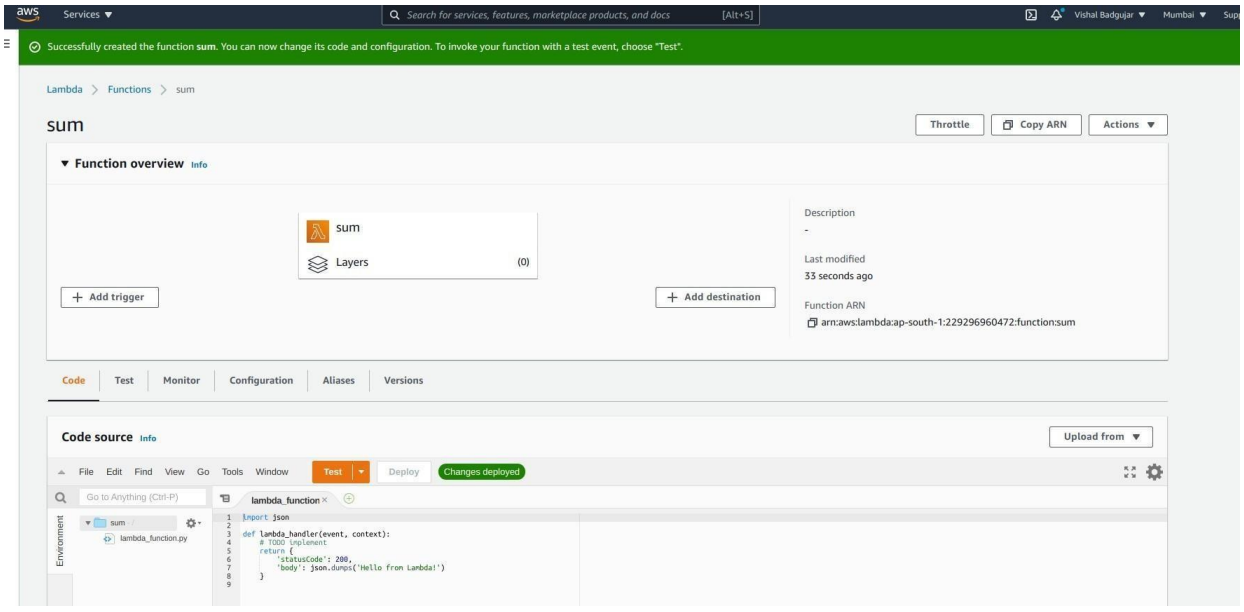
2. Choose region in which you need to create Lambda function as it is region specific.



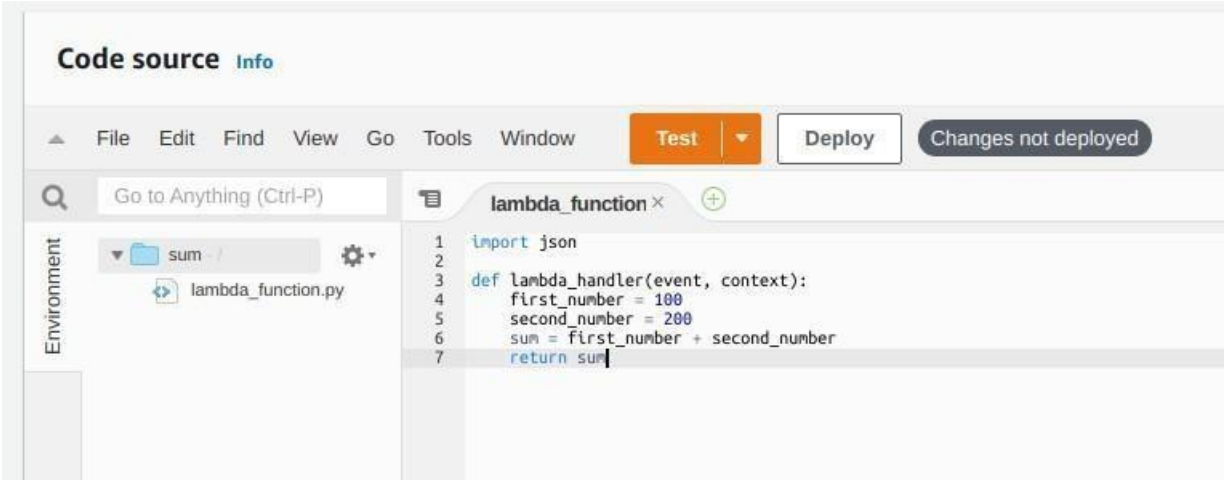
3. Create sum as a Lambda Function in Python Language so select latest version of Python and choose role with basic Lambda Permission to allow cloudwatch for monitoring.



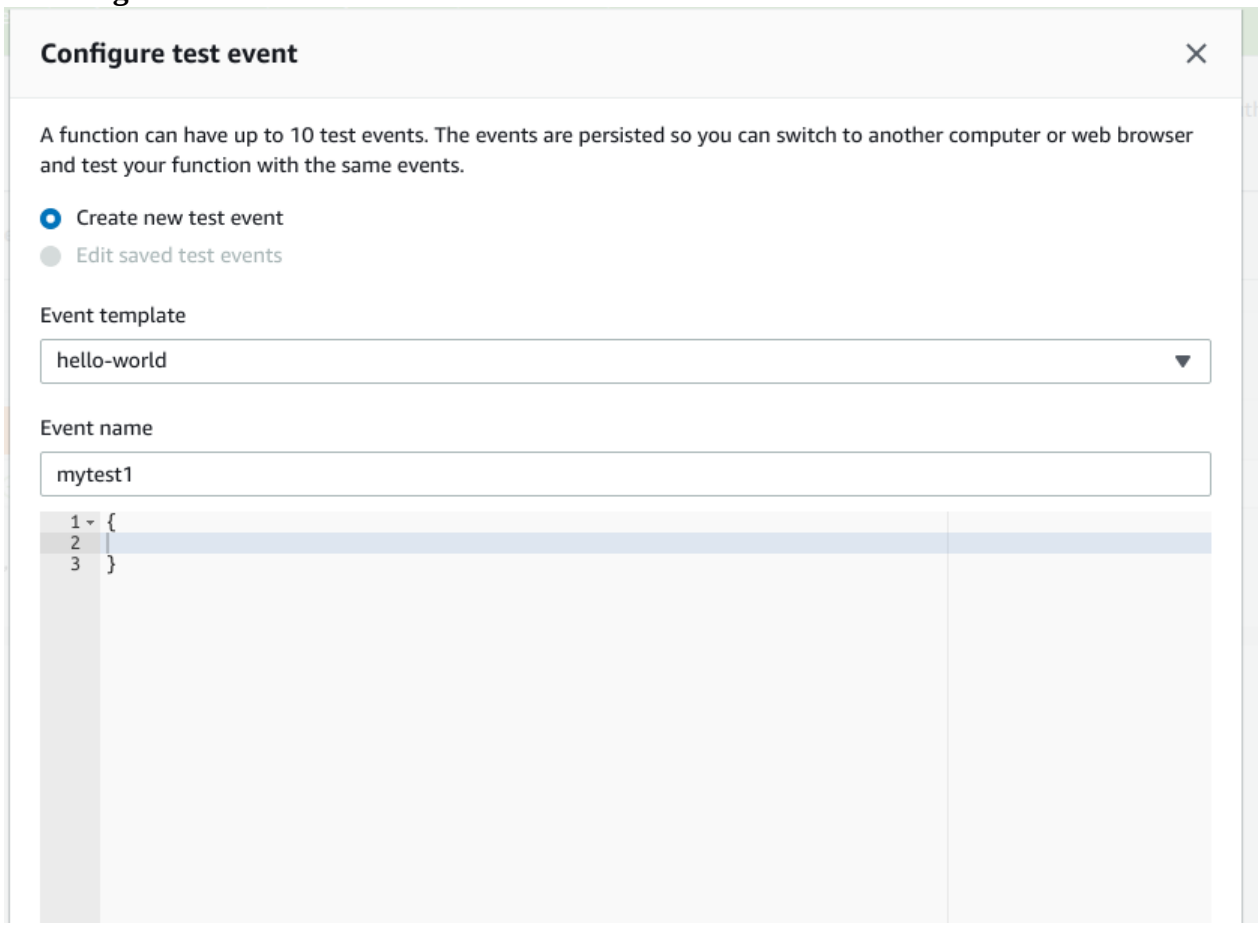
4. Lambda sum function is created successfully

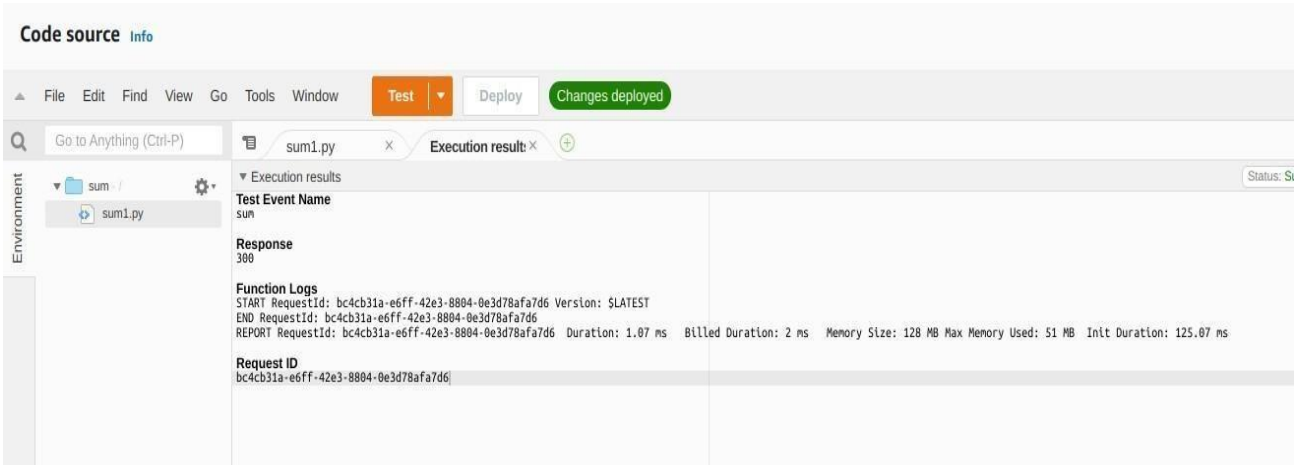


5. Write a sample python code for sum of two numbers:

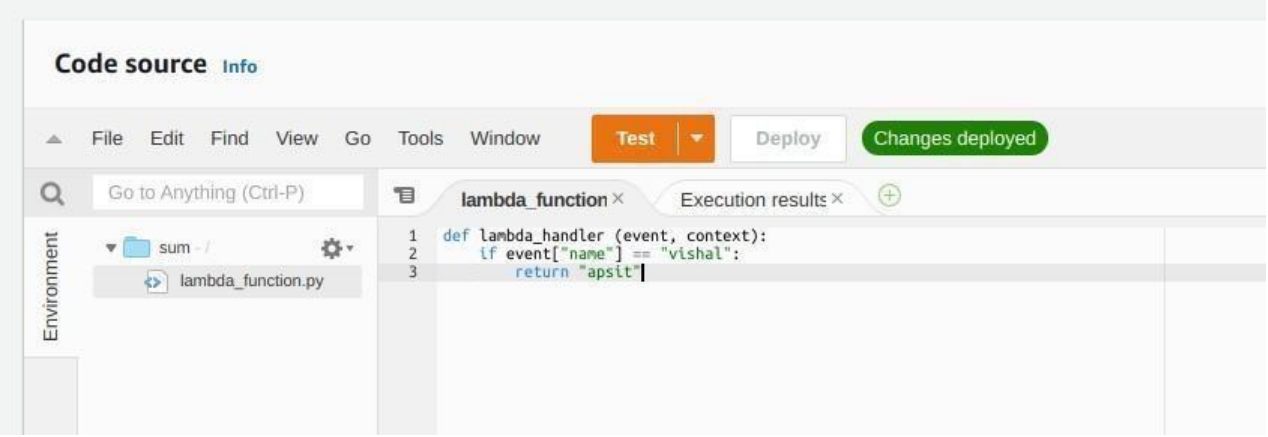


6. Configure Test Event in Json Format

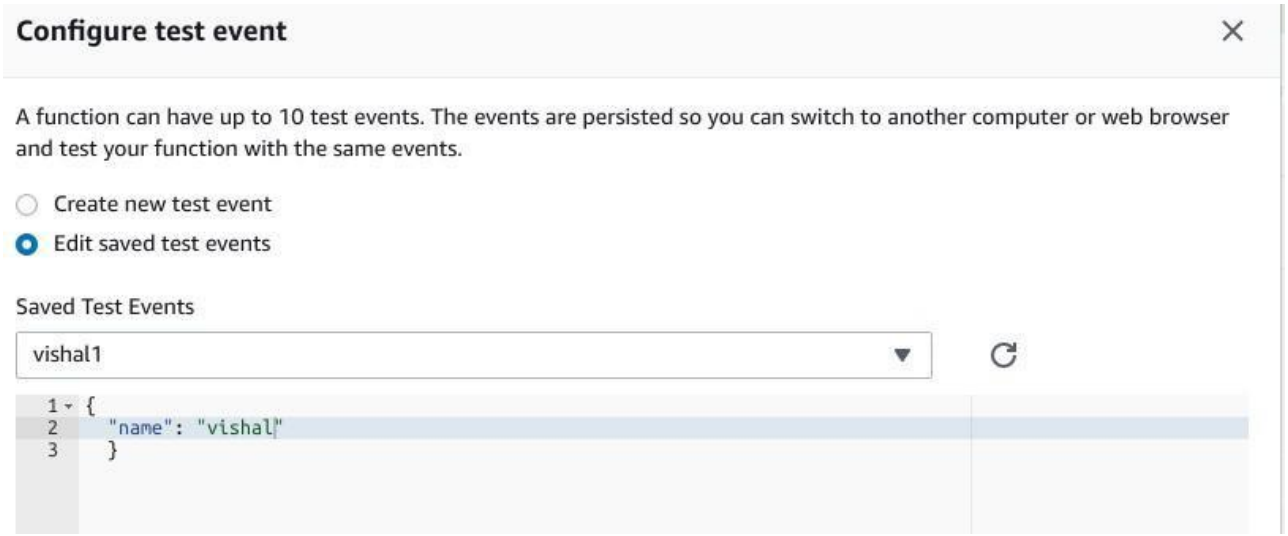




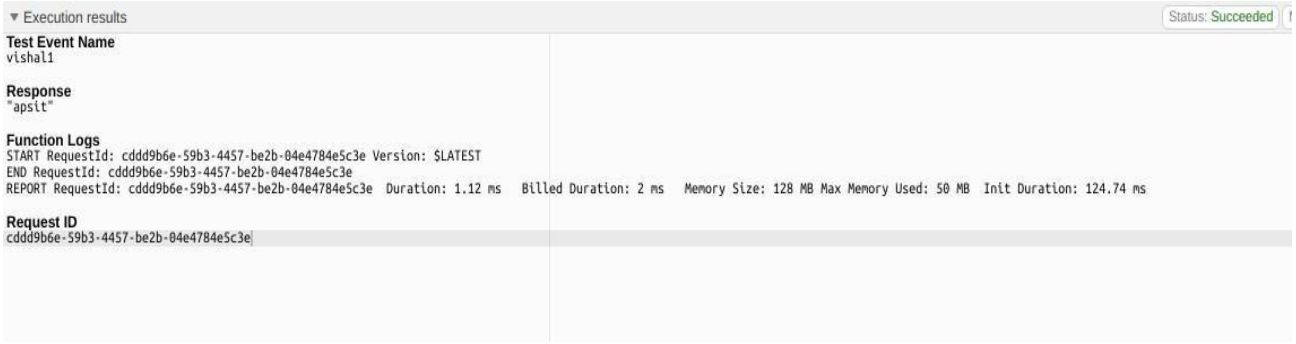
Write a sample Second sample python Code:



Configure Test Event



If condition met returns a value as apsit



Conclusion: Write your own findings.

Academic Year: 2022-23 **Semester: V**
Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof.Manjusha K.

EXPERIMENT NO. 12

Aim: To create a Lambda function which will log "An Image has been added" once you addan object to a specific bucket in S3.

Theory:

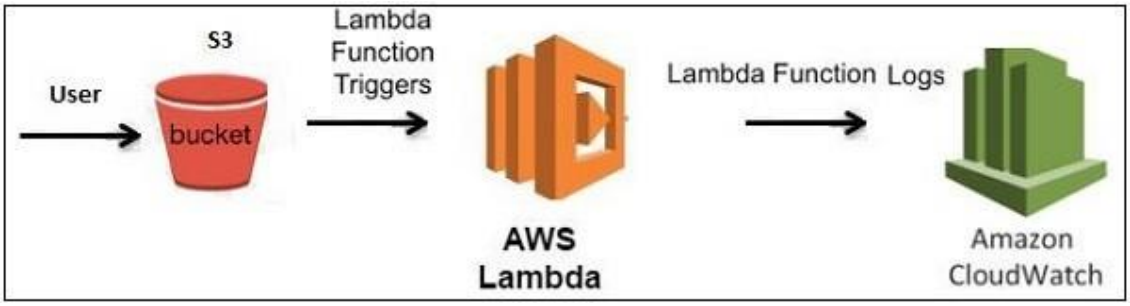
Using Lambda Function with Amazon S3

Amazon S3 service is used for file storage, where you can upload or remove files. We can trigger AWS Lambda on S3 when there are any file uploads in S3 buckets. AWS Lambda has a handler function which acts as a start point for AWS Lambda function. The handler has the details of the events. In this chapter, let us see how to use AWS S3 to trigger AWS Lambda function when we upload files in S3 bucket.

Steps for Using AWS Lambda Function with Amazon S3

To start using AWS Lambda with Amazon S3, we need the following −

- Create S3 Bucket
- Create role which has permission to work with s3 and lambda
- Create lambda function and add s3 as the trigger.



Let us see these steps with the help of an example which shows the basic interaction between Amazon S3 and AWS Lambda

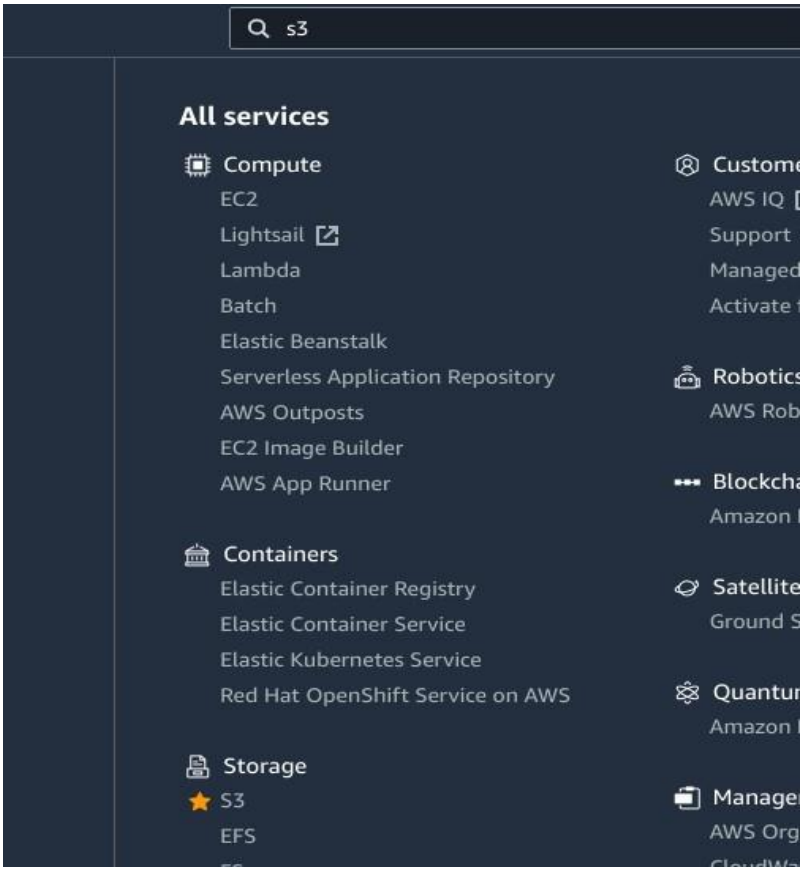
- User will upload a file in Amazon S3 bucket
- Once the file is uploaded, it will trigger AWS Lambda function in the background which will display an output in the form of a console message that the file is uploaded.
- The user will be able to see the message in Cloudwatch logs once the file is uploaded.

Creating S3 Bucket

Let us start first by creating a s3 bucket in AWS console using the steps given below −

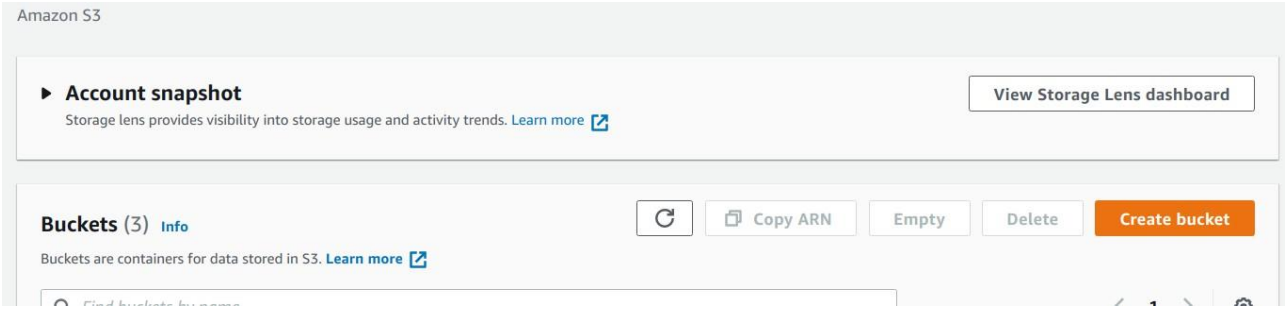
Step 1

Go to Amazon services and click **S3** in storage section as highlighted in the image given below −



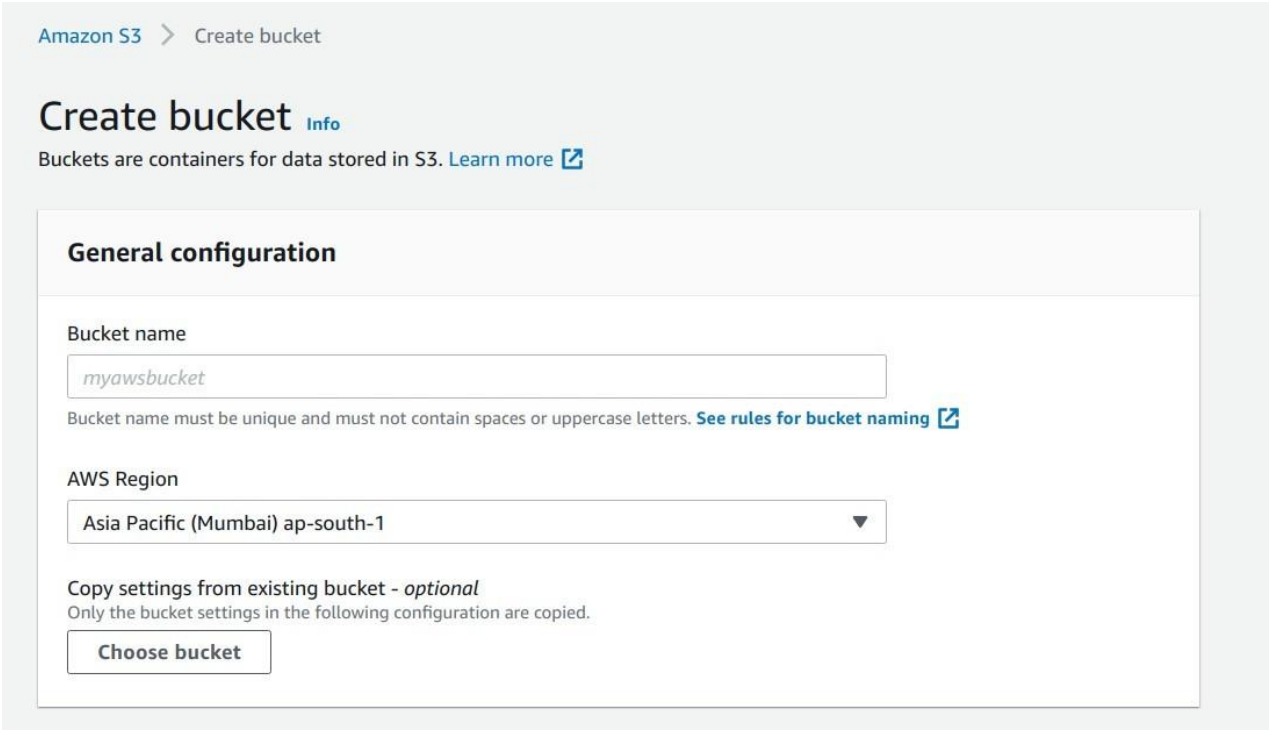
Step 2

Click **S3** storage and **Create bucket** which will store the files uploaded.



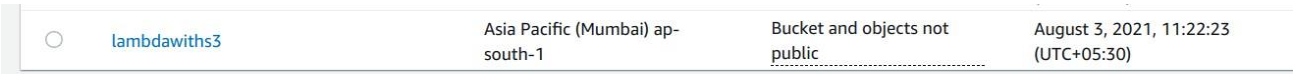
Step 3

Once you click Create bucket button, you can see a screen as follows −



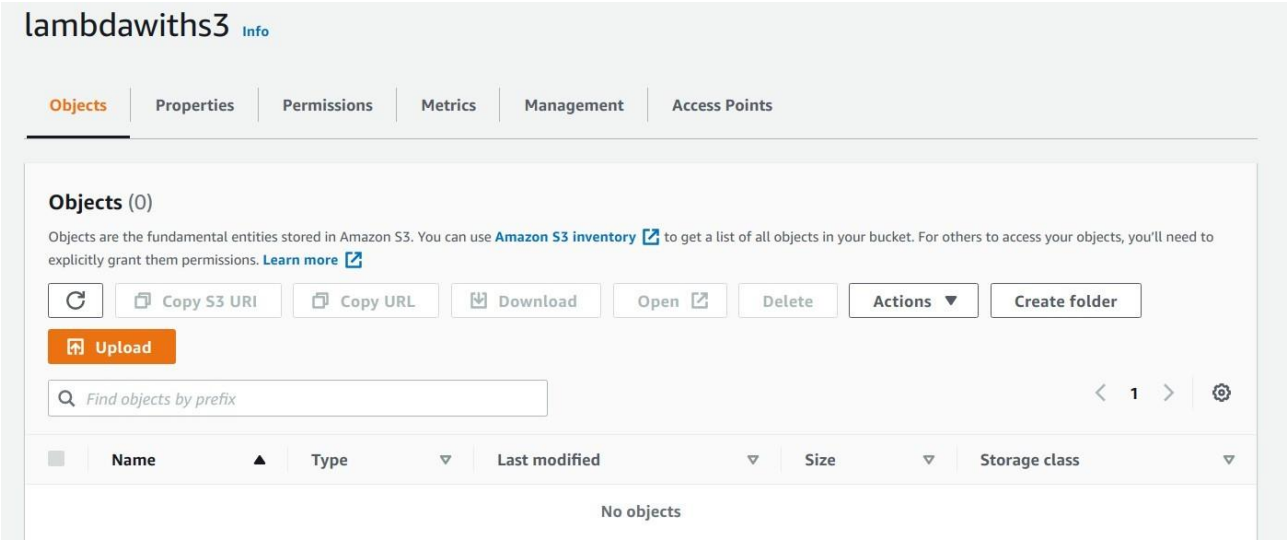
Step 4

Enter the details Bucket name, Select the Region and click Create button at the bottom left side. Thus, we have created bucket with name :



Step 5

Now, click the bucket name and it will ask you to upload files as shown below −



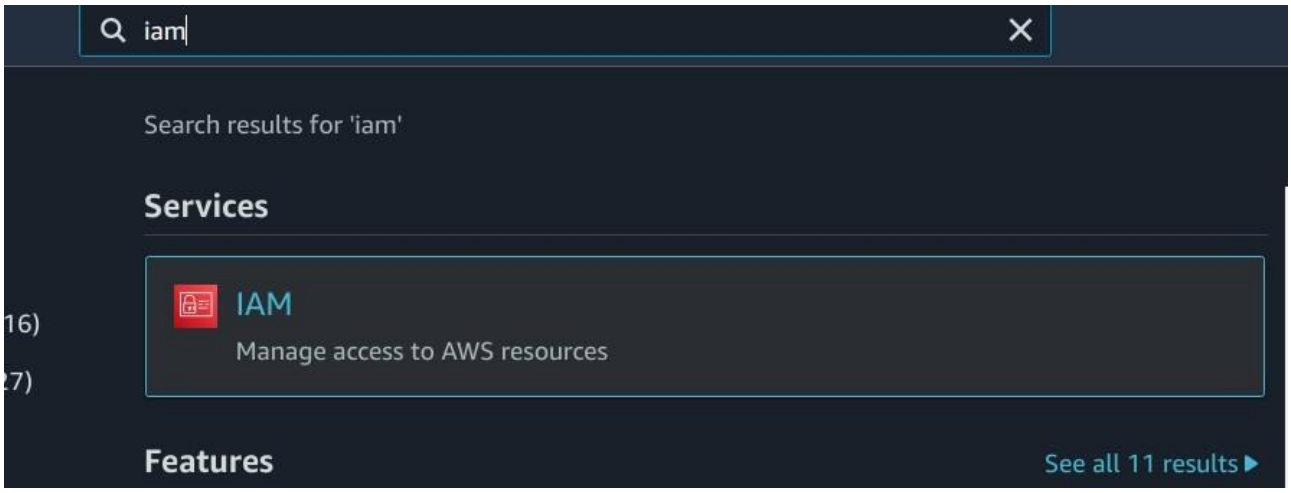
Thus, we are done with bucket creation in S3.

Create Role that Works with S3 and Lambda

To create role that works with S3 and Lambda, please follow the Steps given below −

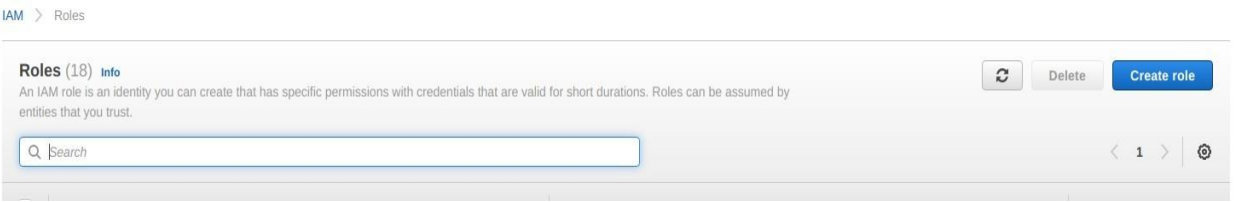
Step 1

Go to AWS services and select IAM as shown below −



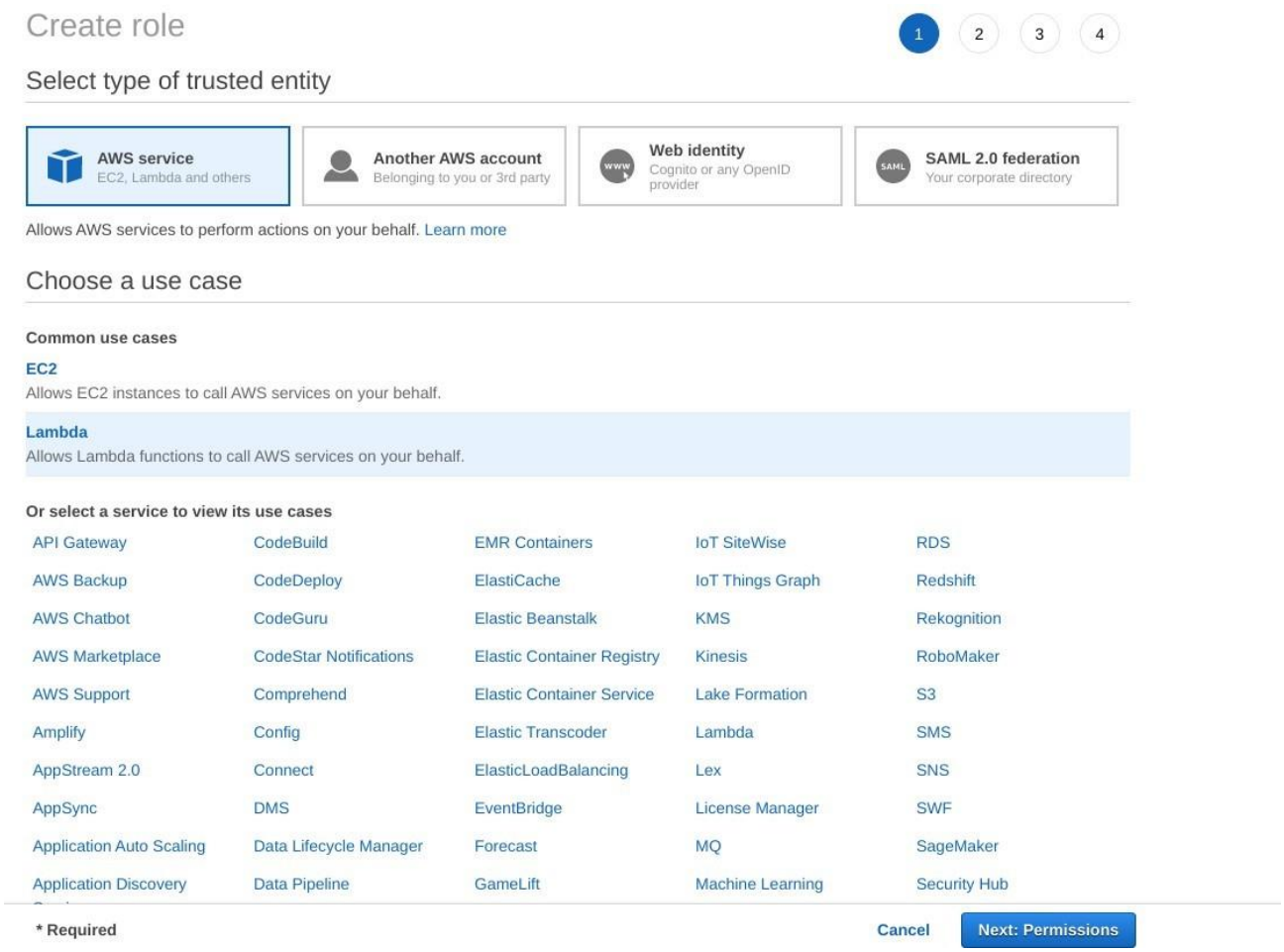
Step 2

Now, click **IAM -> Roles** as shown below −



Step 3

Now, click **Create role** and choose the services that will use this role. Select Lambda and click **Permission** button.



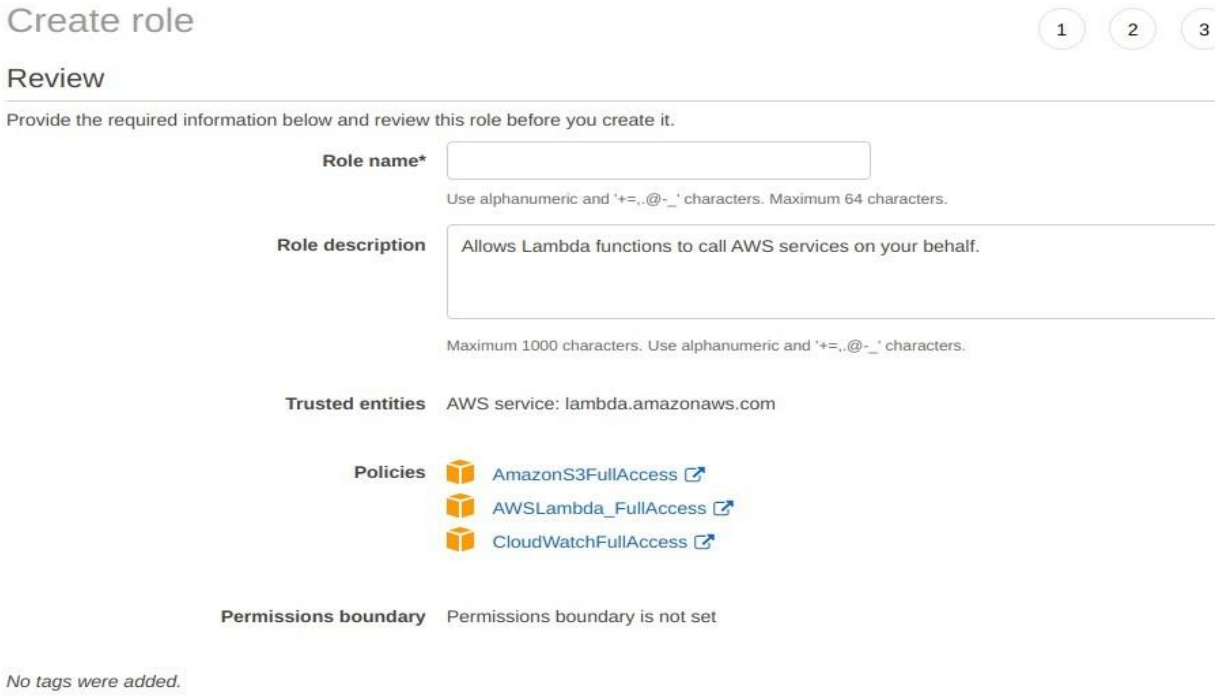
Step 4

Add the permission from below and click Review.

AmazonS3FullAccess, AWSLambdaFullAccess and CloudWatchFullAccess.

Step 5

Observe that we have chosen the following permissions −



Observe that the Policies that we have selected are **AmazonS3FullAccess, AWSLambdaFullAccess and CloudWatchFullAccess.**

Step 6

Now, enter the Role name, Role description and click Create Role button at the bottom.

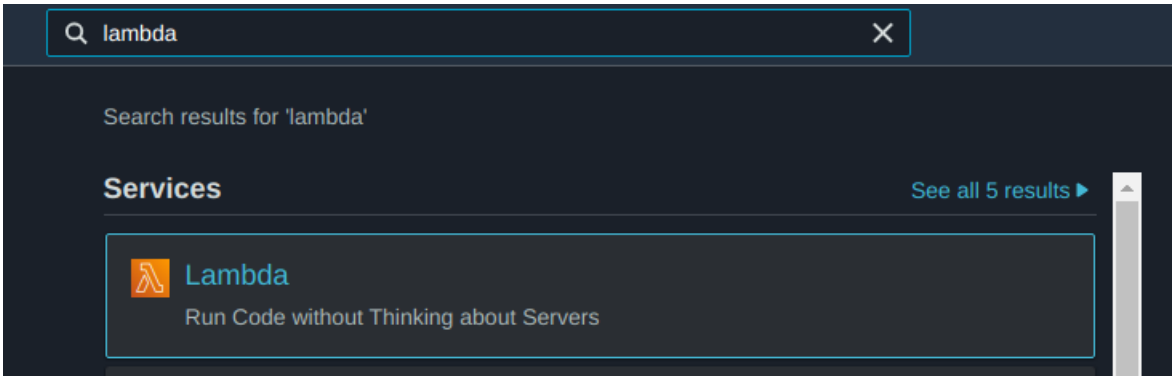
Thus, our role named lambdawiths3service is created.

Create Lambda function and Add S3 Trigger

In this section, let us see how to create a Lambda function and add a S3 trigger to it. For this purpose, you will have to follow th Steps given below −

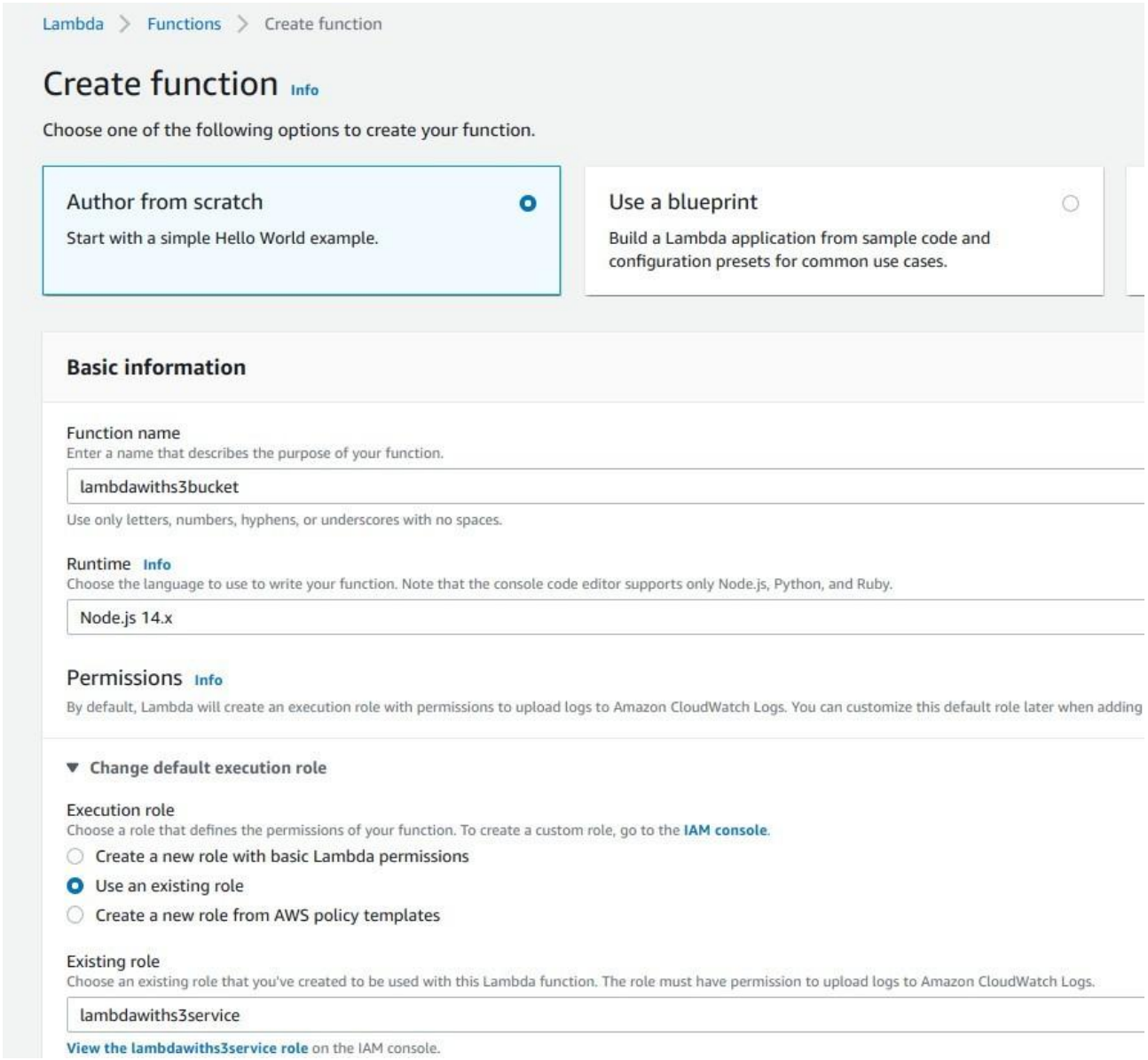
Step 1

Go to AWS Services and select Lambda as shown below −



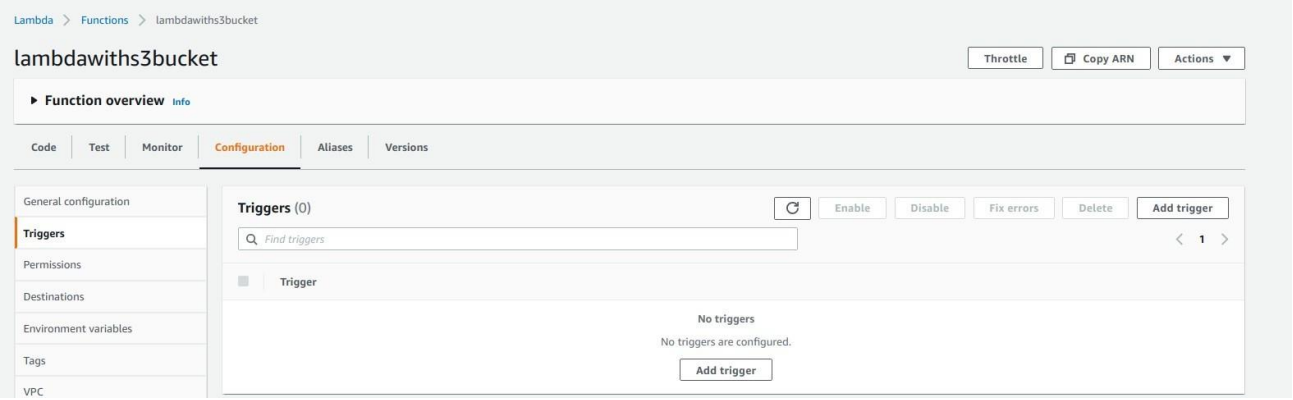
Step 2

Click **Lambda** and follow the process for adding **Name**. Choose the **Runtime, Role** etc. and createthe function. The Lambda function that we have created is shown in the screenshot below −



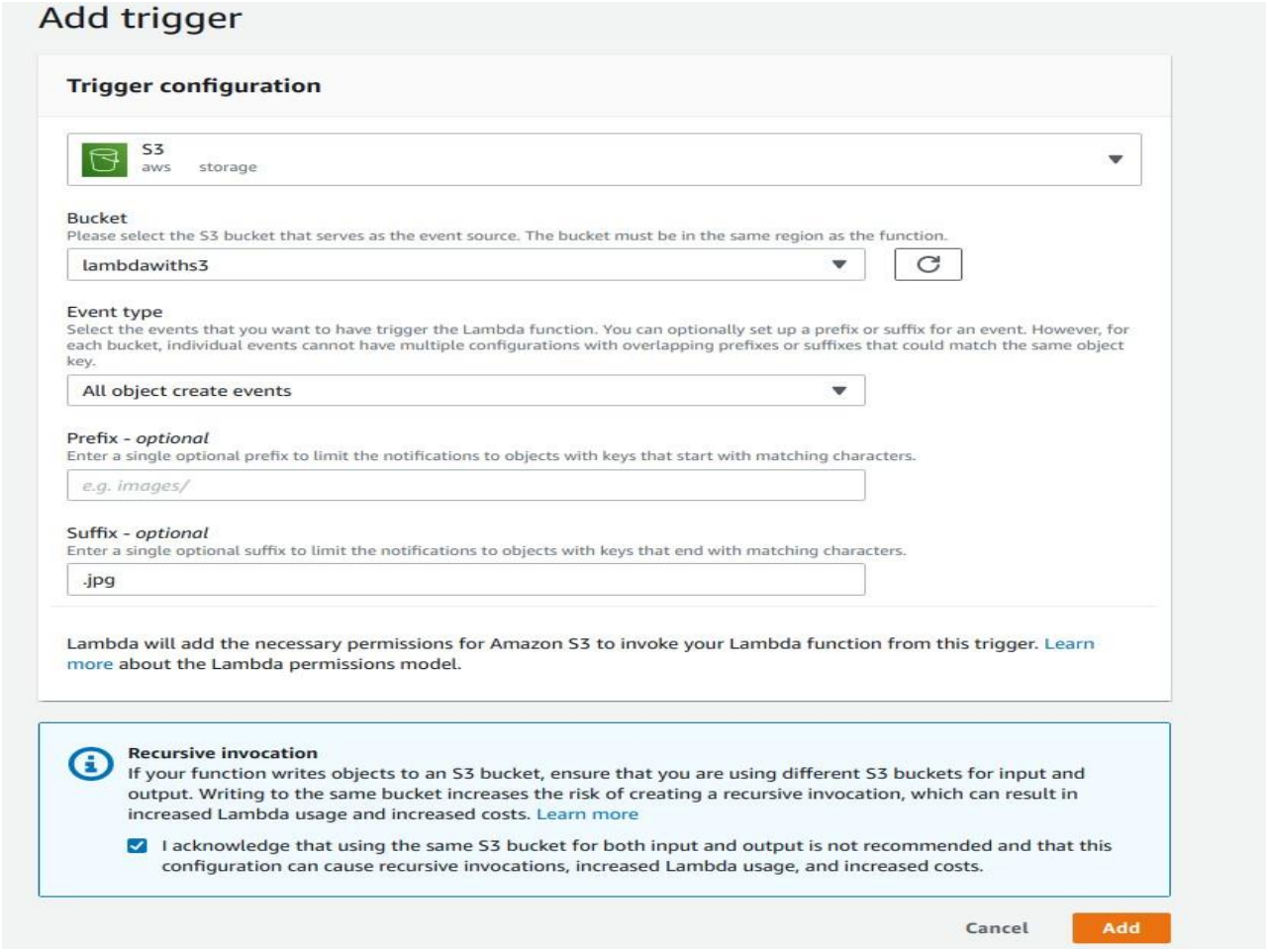
Step 3

Now let us add the S3 trigger.



Step 4

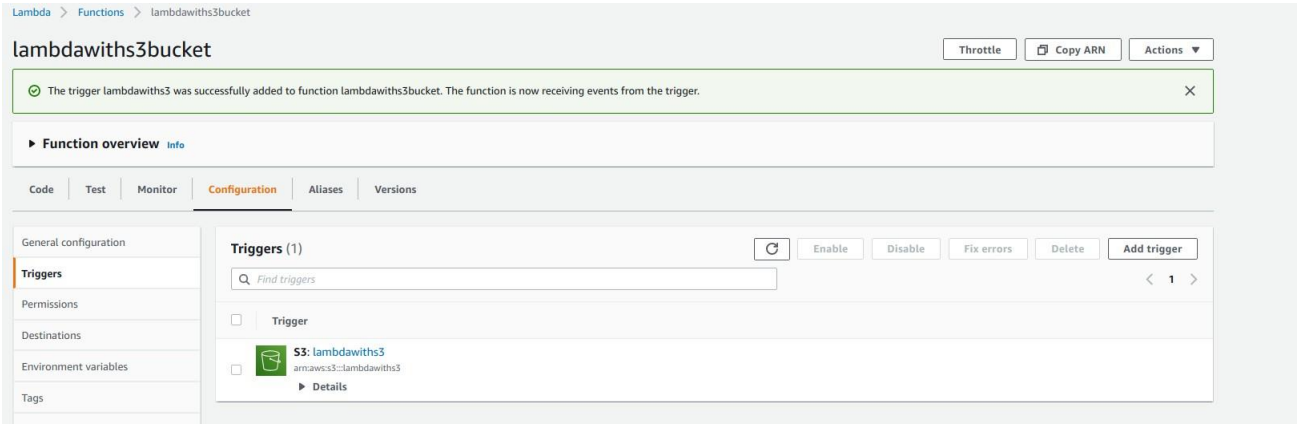
Choose the trigger from above and add the details as shown below −



You can add Prefix and File pattern which are used to filter the files added. For Example, to trigger lambda only for .jpg images. as we need to trigger Lambda for all jpg image files uploaded. Click Add button to add the trigger.

Step 5

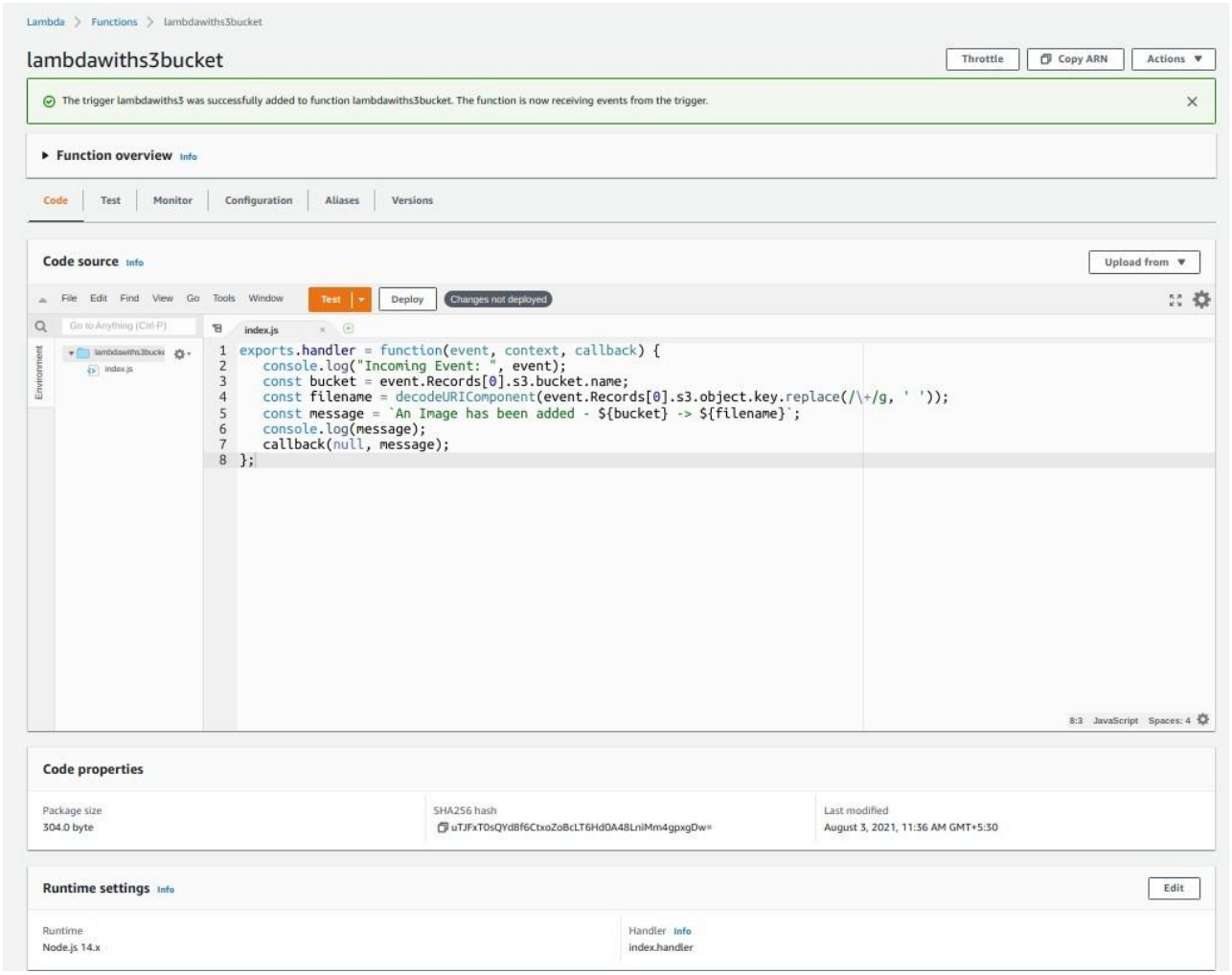
You can find the the trigger display for the Lambda function as shown below −



Step 6

Let’s add the details for the aws lambda function. Here, we will use the online editor to add ourcode and use nodejs as the runtime environment.

To trigger S3 with AWS Lambda, we will have to use S3 event in the code as shown below −



Step 7:

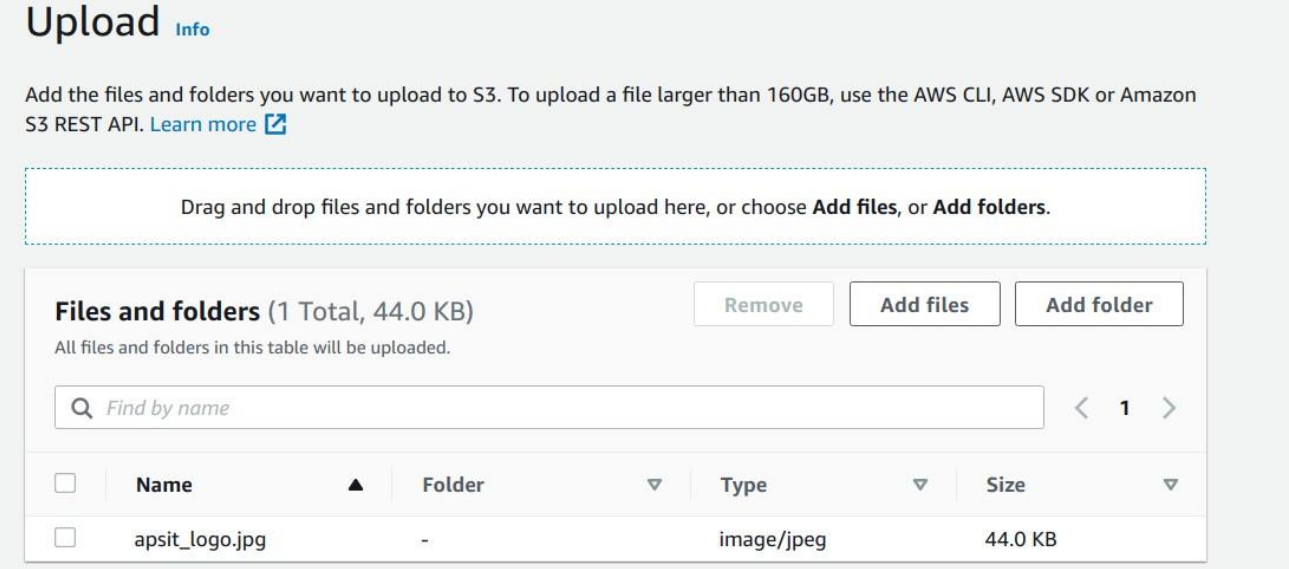
let us save the changes and test the lambda function with S3upload.

Step 8:

Now, save the Lambda function. Open S3 from Amazon services and open the bucket we createdearlier namely lambdawiths3.

Upload the image in it as shown below −

Click **Add files** to add files. You can also drag and drop the files. Now, click **Upload** button.

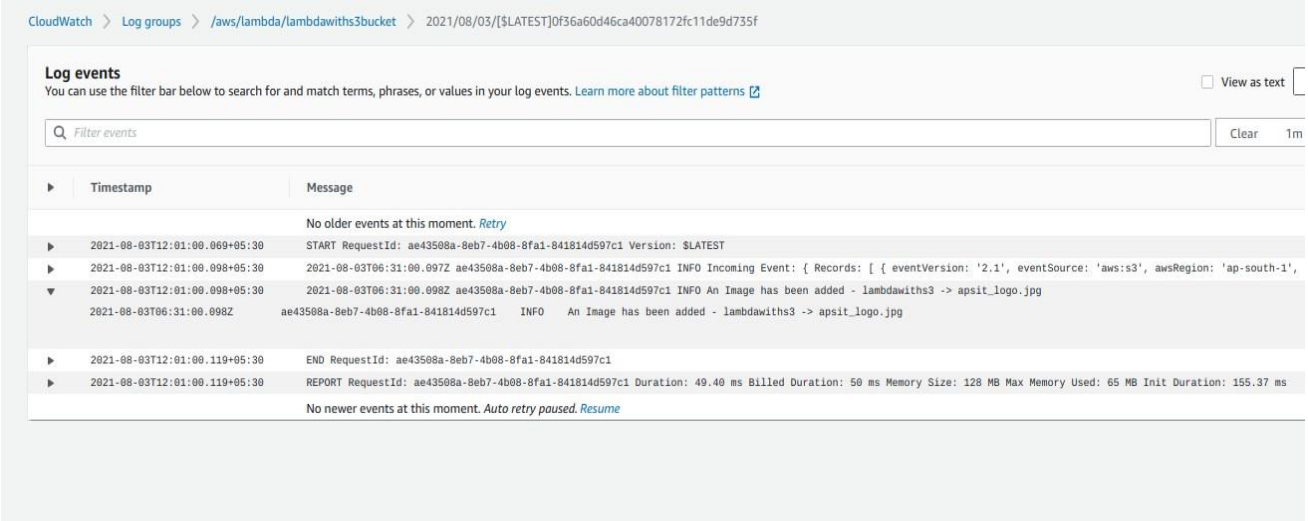


Thus, we have uploaded one image in our S3 bucket.

Step 9

To see the trigger details, go to AWS service and select CloudWatch. Open the logs for the Lambda

AWS Lambda function gets triggered when file is uploaded in S3 bucket and the details are loggedin Cloudwatch as shown below −



An image has been Added -> apsit_logo.jpg you can see in cloudwatch logs.

Conclusion: Write your own findings.

```
export const handler = async (event, context) => {
  console.log("Incoming Event:", event);

  if (!event.Records || event.Records.length === 0) {
    const errorMessage = "No records found in the event.";
    console.log(errorMessage);
    return errorMessage;
  }

  const bucket = event.Records[0].s3.bucket.name;
  const filename = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ''));
  const message = An Image has been added - ${bucket} -> ${filename};
  console.log(message);

  return message;
};
```

Semester: V
Academic Year: 2023-24
Year :TE IT
Subject: Advanced Devops Lab (ADL)
Name of Instructor:Prof. Manjusha K.

EXPERIMENT NO. 13

Aim: To demonstrate working of GAE launcher to launch the web applications.

Theory:

Prerequisites:

Installed Google Cloud SDK: Ensure that you have the Google Cloud SDK installed on your machine. If not, follow the installation guide provided by Google Cloud.
Lab Steps:
1. Create a Google Cloud Project:
a. Open the Google Cloud Console.
b. Create a new project or select an existing one.

2. Enable App Engine API:
a. Navigate to the "APIs & Services" > "Dashboard" section.
b. Click on the "+ ENABLE APIS AND SERVICES" button.
c. Search for "App Engine Admin API" and enable it.

3. Install GAE Launcher:
a. Download and install the Google App Engine Launcher for your operating system.

4. Configure GAE Launcher:
a. Open GAE Launcher and sign in with your Google Cloud account.
b. Click on "Create a new application" and enter a unique application ID.

5. Develop a Simple Web Application:
a. Create a new directory for your web application.
b. Inside the directory, create an 'app.yaml' file with the following content:
yaml runtime: python39 handlers: - url: /.* script: main.app c. Create a 'main.py' file with a simple Python web application.

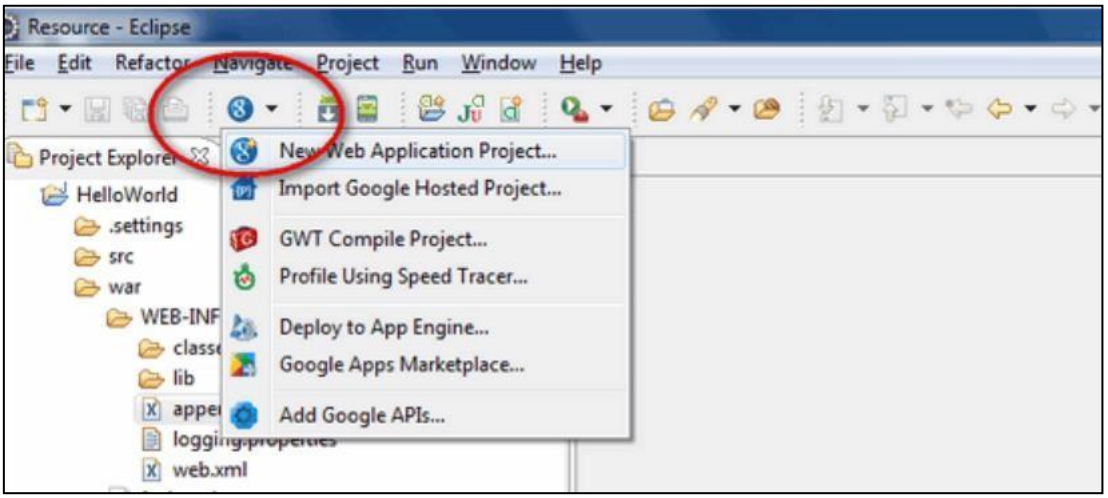
6. Run the Application Locally:
a. In GAE Launcher, click on the "Run" button next to your application.
b. Open a web browser and navigate to http://localhost:8080 to view your locally running application.

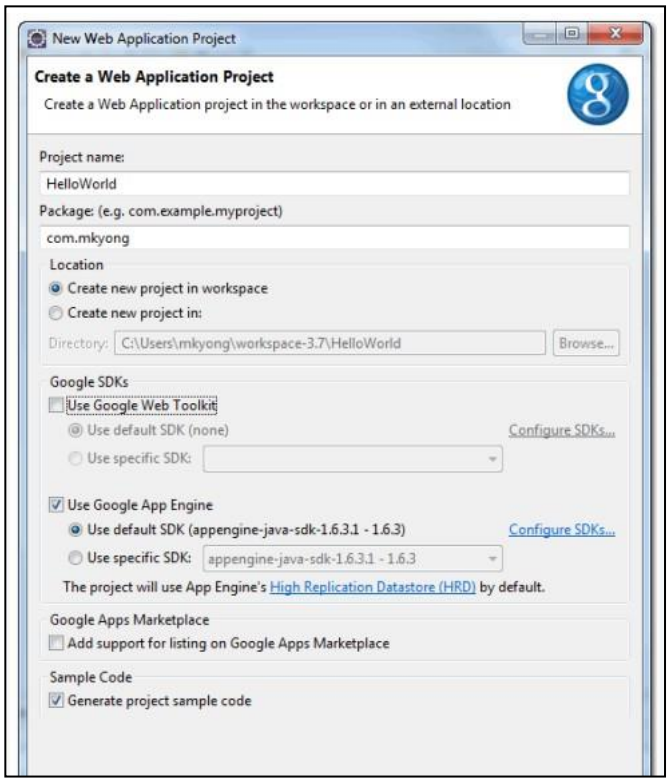
7. Deploy the Application to GAE:
a. Click on the "Deploy" button in GAE Launcher.
b. Follow the prompts to deploy your application to the Google Cloud Platform.

8. Access the Deployed Application:
a. Once deployed, open a web browser and navigate to https://[YOUR_PROJECT_ID].appspot.com.

How to install Plugins -

1. Install Google Plugin for Eclipse
2. Create New Web Application Project
   In Eclipse toolbar, click on the Google icon, and select "New Web Application Project…"

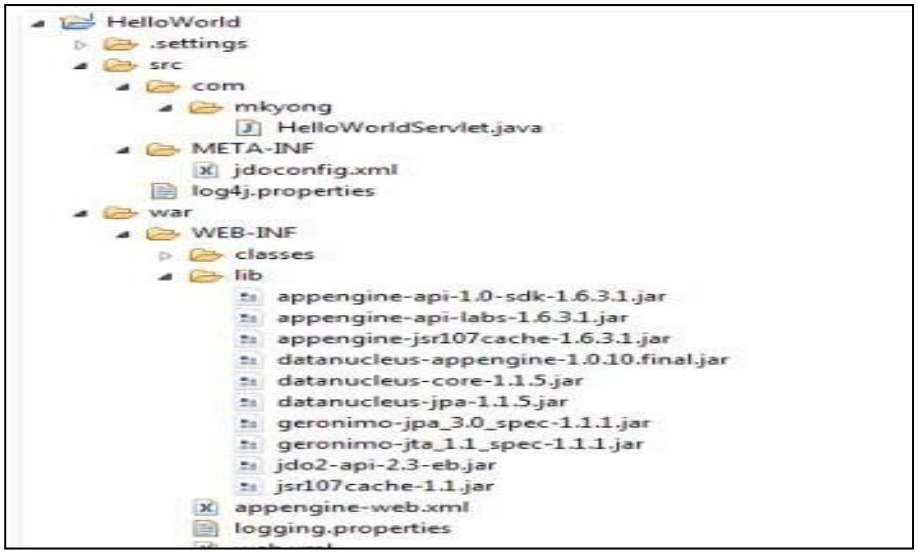




Click finished, Google Plugin for Eclipse will generate a sample project automatically.

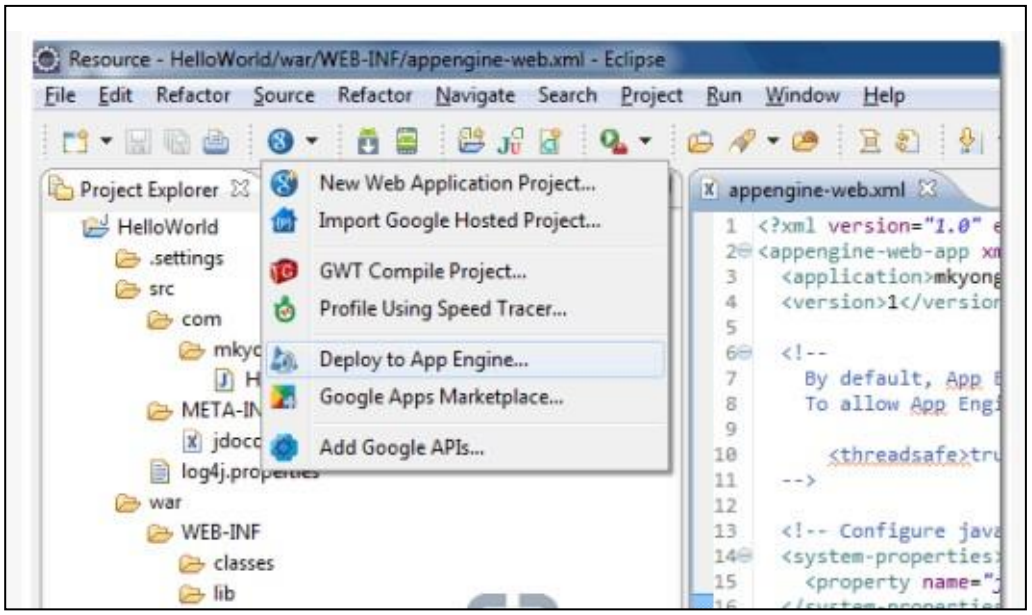
3. Hello World

Review the generated project directory



4. Run it local Right click on the project and run as "Web Application". Eclipse console : //... INFO: The server is running at http://localhost:8888/ 30 Mac 2012 11:13:01 PM com.google.appengine.tools.development.DevAppServerImpl start INFO: The admin console is running at http://localhost:8888/_ah/admin
Copy Access URL http://localhost:8888/, see output

5. Deploy to Google App Engine
Register an account on https://appengine.google.com/, and create an application ID for your web application.
In this demonstration, I created an application ID, named "mkyong123", and put it in appengineweb.xml.
File : appengine-web.xml



Sign in with your Google account and click on the Deploy button. If everything is fine, the hello world web application will be deployed to this URL – http://mkyong123.appspot.com/

Conclusion: Thus we have have learned how to use the Google App Engine Launcher to deploy a web application. This practical provides a foundation for deploying more complex applications on the Google App Engine platform.