Name: Shivpratik Hande Roll No.: 14 Class: D15C AY: 2024-25

Static Hosting [Exp 1(a)]

Aim: To develop a website and host it on:-

1. Localmachineorvirtualmachine.
2. AmazonS3bucket.

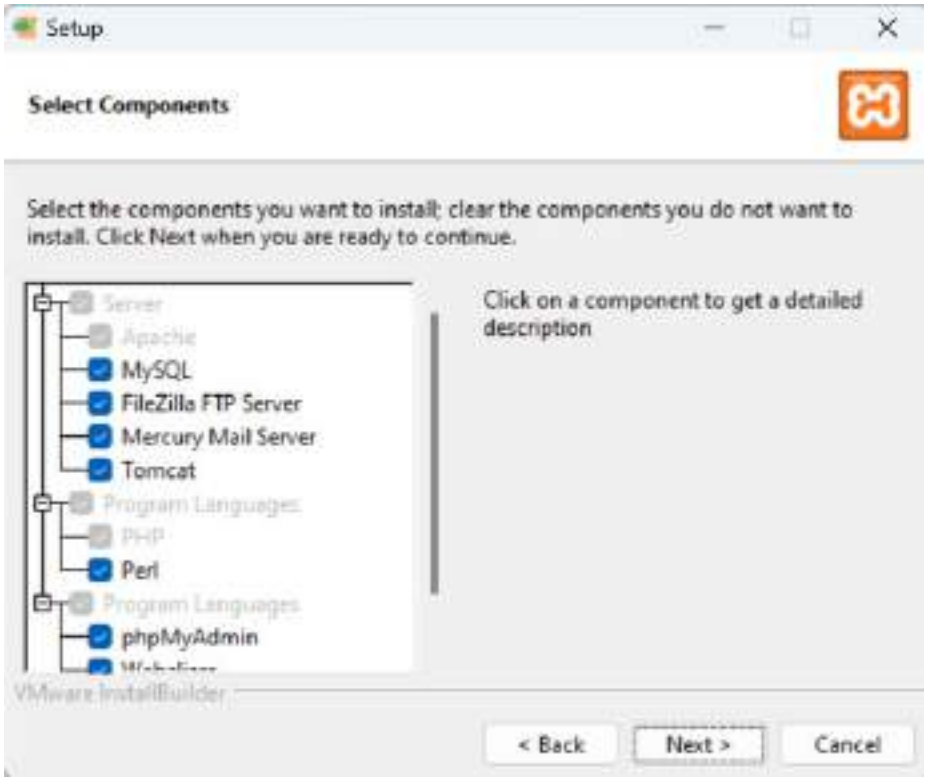
Steps:-

- OnlocalserverusingXAMPP:-

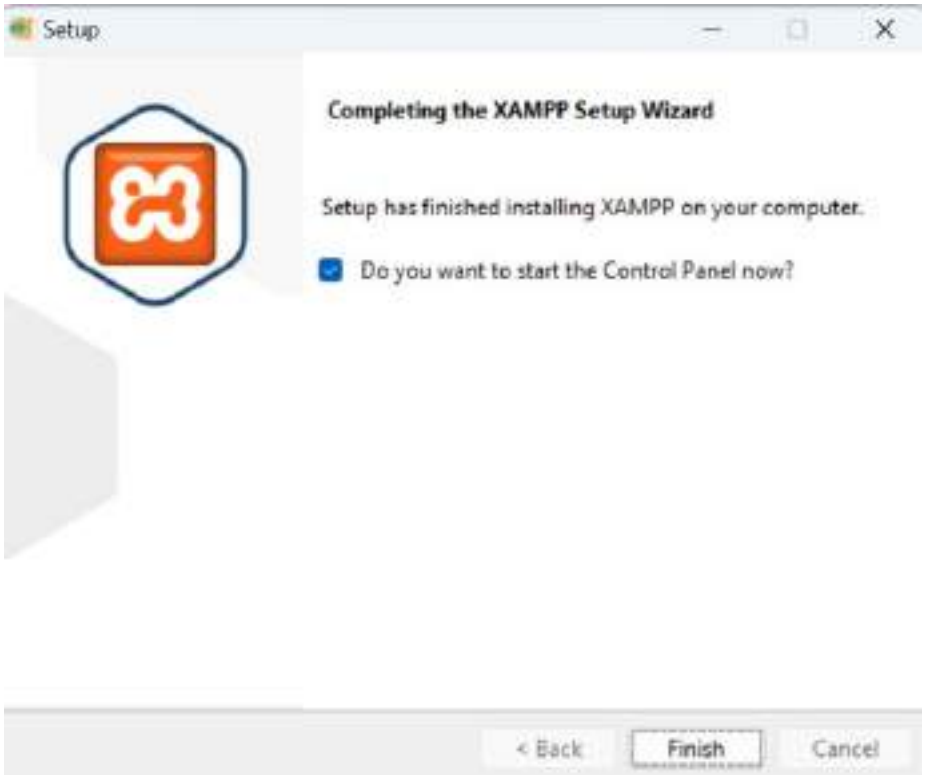
Step 1: Install XAMPP.

1. Beginthesetupprocess.

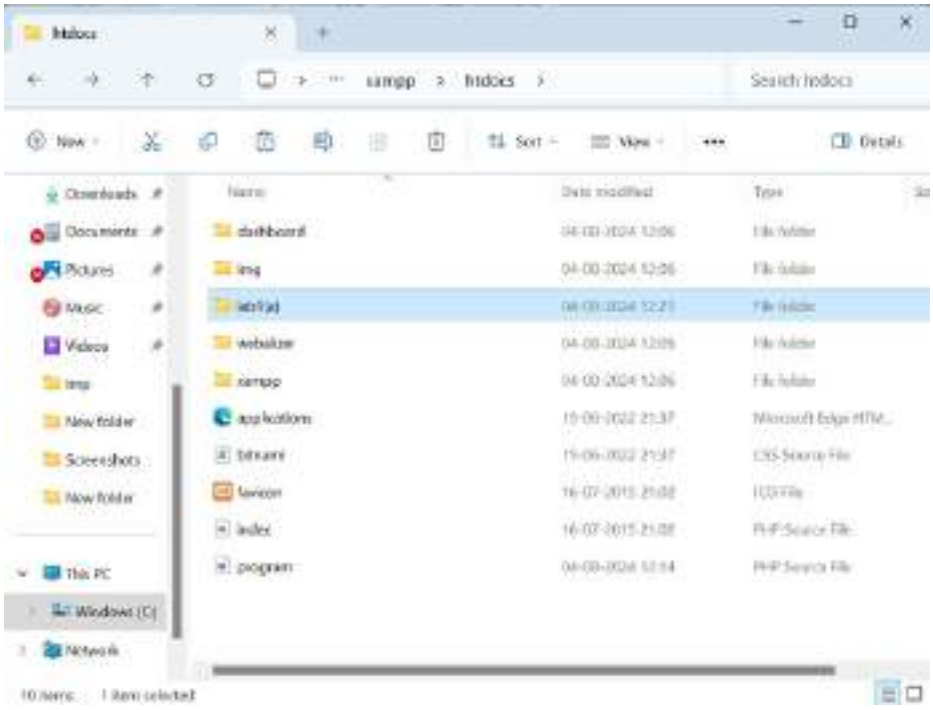
2. Select the components you want to install and click Next.



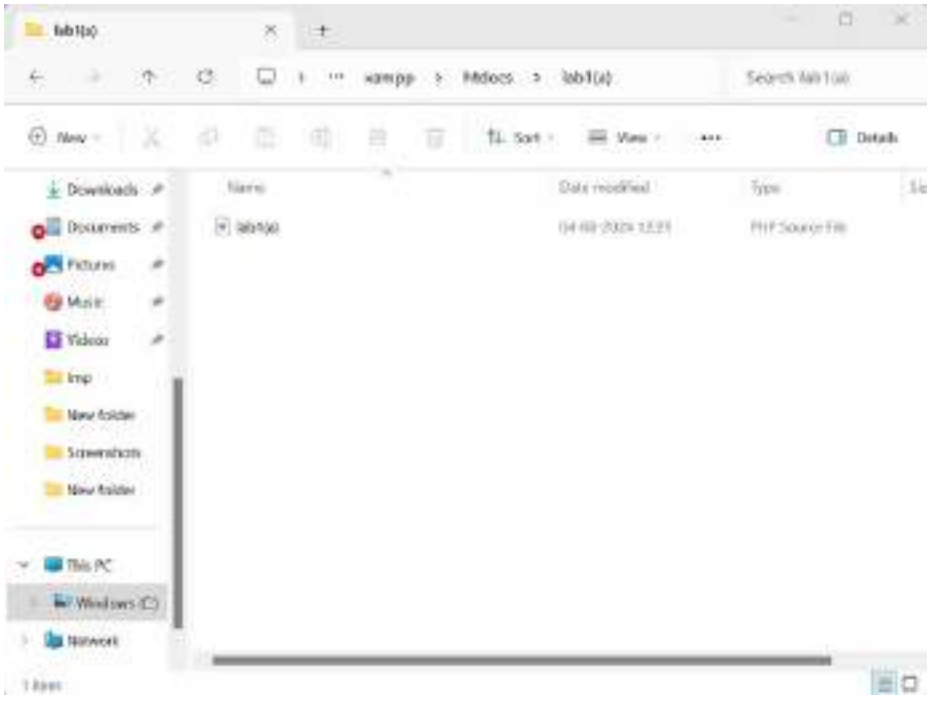
3. Go through the subsequent installation steps (keeping the default options) and finish the installation.



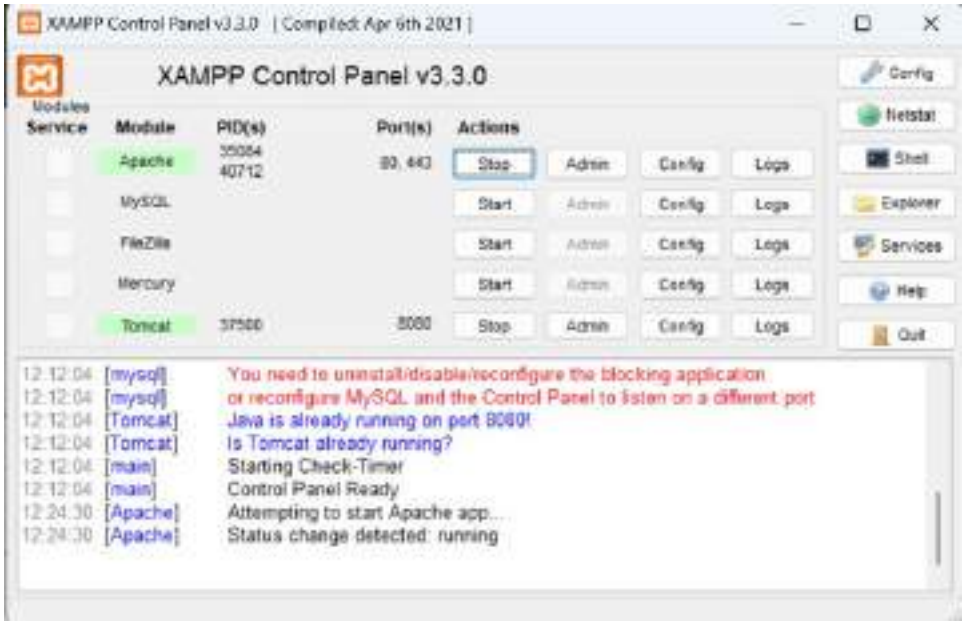
Step 2: Inside the htdocs folder inside of the xampp folder create a new folder.



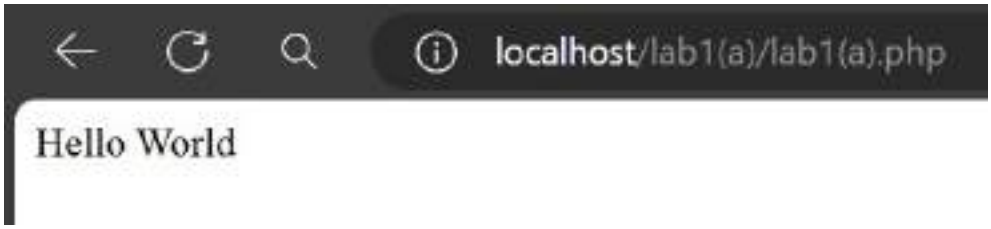
Step 3: Create a file (with extension: .php) that is to be hosted on the local server.



Step 4: Open the XAMPP Control Panel and start the Apache service.

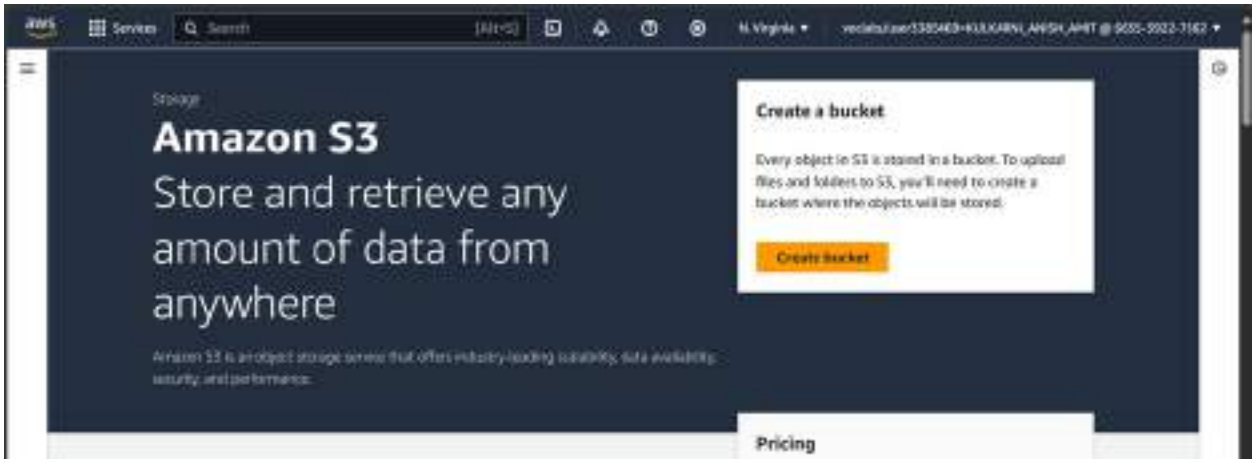


Step 5: Type ‘localhost/YOUR_FILENAME.php into your web browser. This will open your website on your browser.

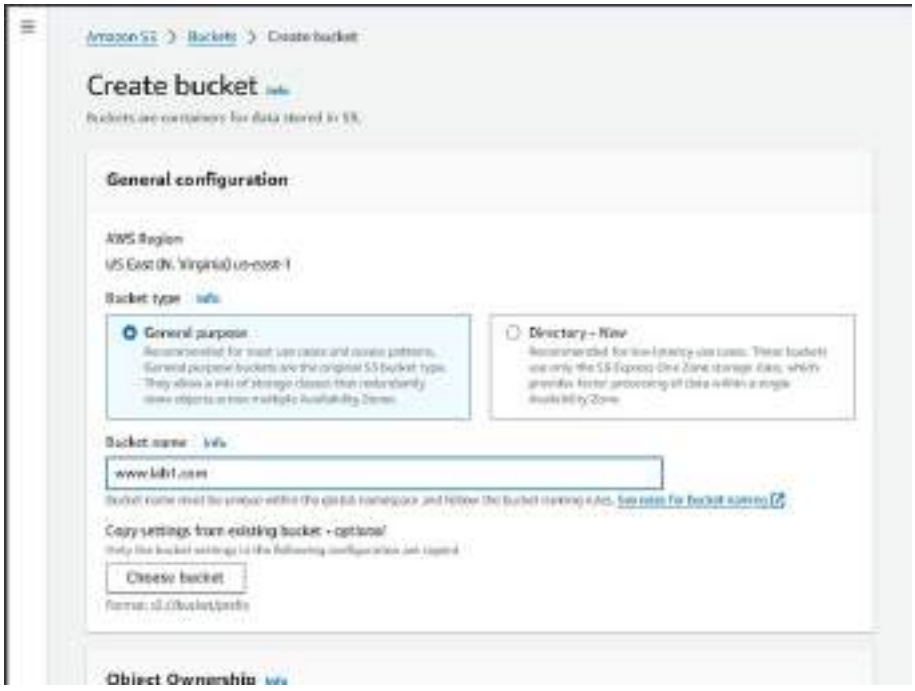


● OnAWSS3:-

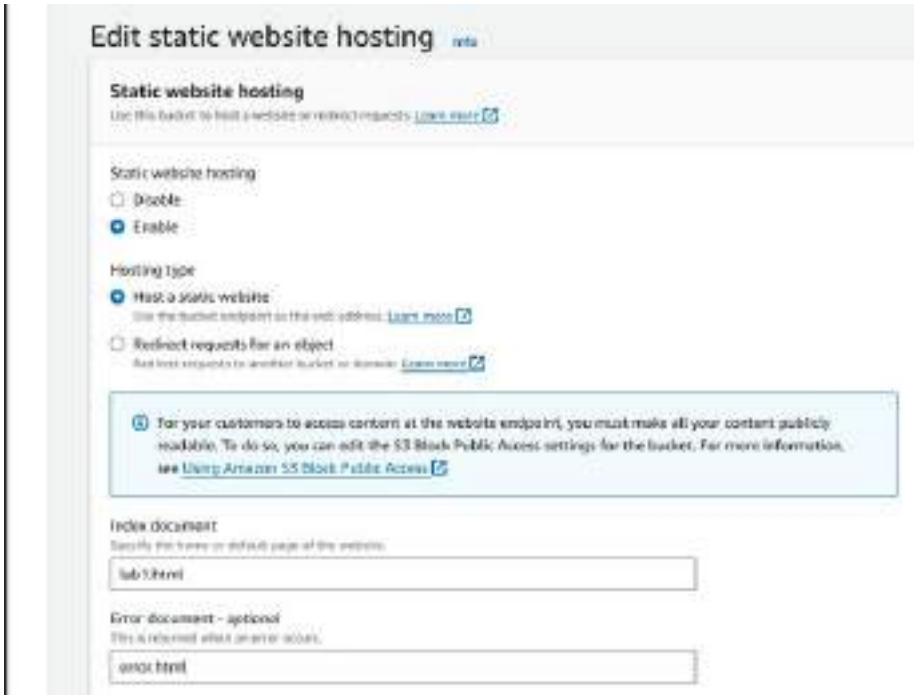
Step 1: Login into your AWS account, choose the S3 option in the ‘Services’ tab and click on ‘Create Bucket’.



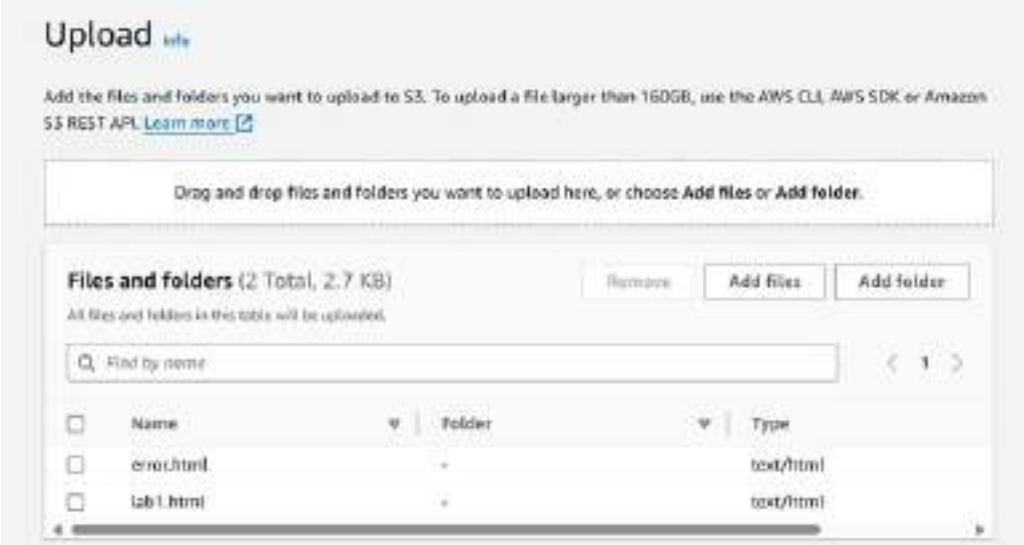
Step 2: Give a name to your bucket and keeping the others options default, click on ‘Create Bucket’.



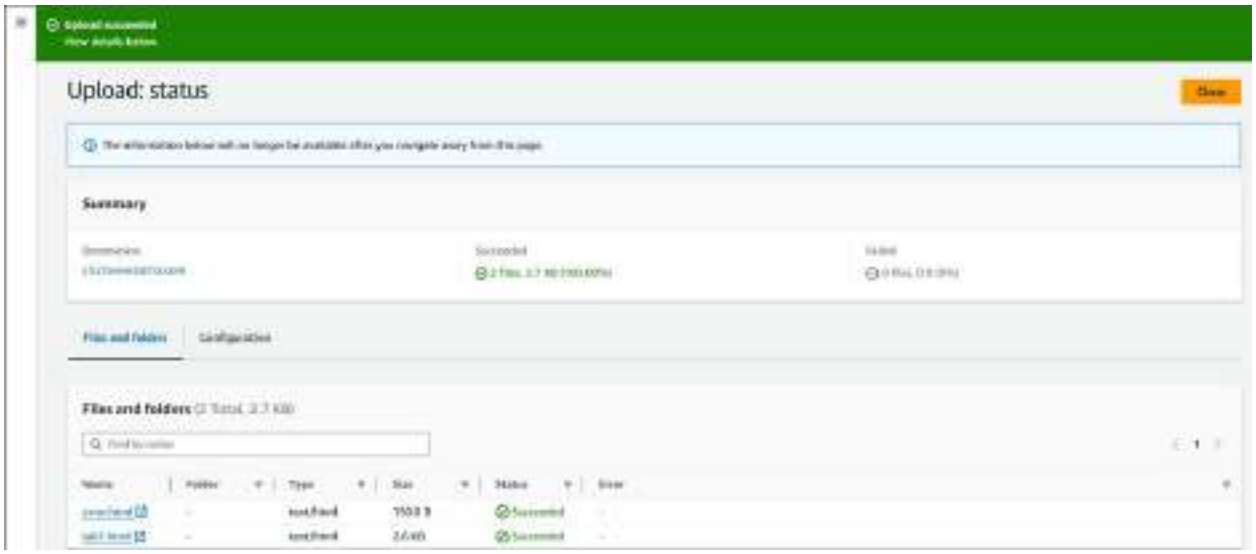
Step 3: In the properties of your bucket, navigate to the ‘Edit static website hosting’ tab and click on ‘Enable’. Also, in the ‘Index document’ and ‘Error document’ boxes, enter the names of your index and error websites respectively.



Step 4: In your bucket, click on ‘Upload’ option, click on ‘Add files’ option and select the websites that you want to upload (index and error websites). Then, click on ‘Upload’.

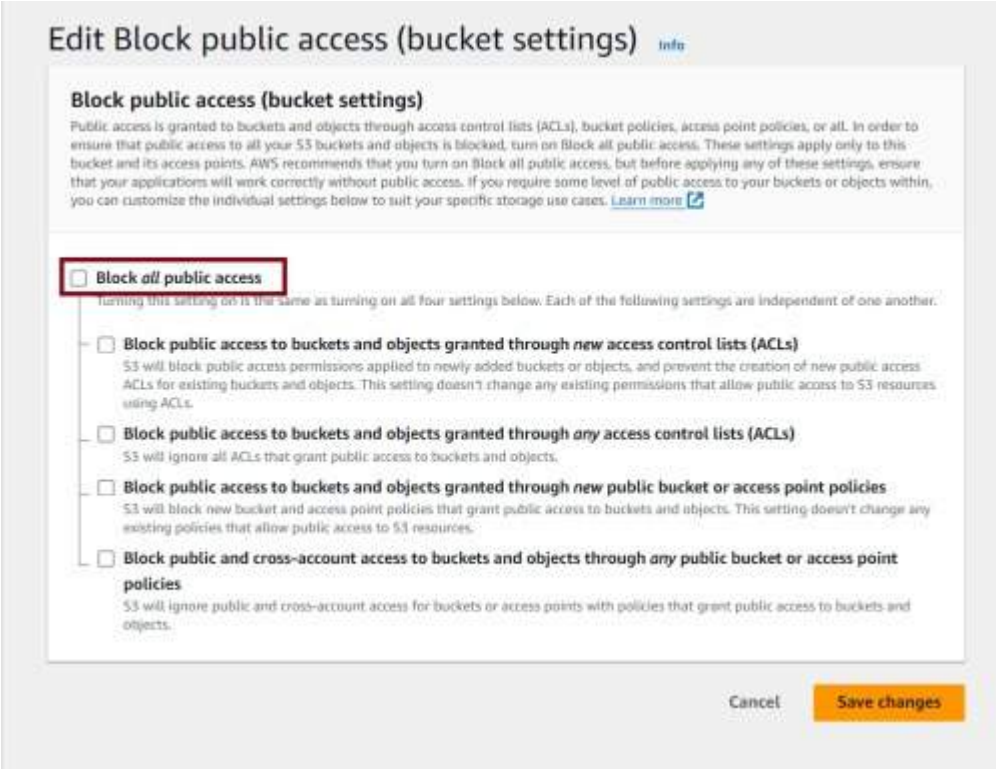


Step 5: Your files are uploaded.



Now, as the contents of the bucket are not yet available to the public, the website does not get displayed and an error message is displayed.

Step 6: To make it public, go to permissions tab, go to ‘Block public access’ and click on edit. Uncheck the ‘Block all public access’ checkbox and click on save changes.



Step 7: Go to ‘Bucket policy’ tab and click edit. Paste the following code snippet into the text box (replace‘YOUR-BUCKET-NAME-HERE’withthenameofyourbucketandsavechanges.

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "PublicReadGetObject",
"Effect": "Allow",
"Principal": {
"AWS": "*"
},
"Action": "s3:GetObject",
"Resource": "arn:aws:s3:::YOUR-BUCKET-NAME-HERE/*"
}
]
}
```

Step 8: Click on ‘Bucket website endpoint’ in the ‘Static website hosting’ tab in your bucket. Your website is visible.

Conclusion: In this experiment, we learned how to host a static website both on a local server using XAMPP and on AWS S3. By setting up XAMPP, we were able to deploy a local web server and access the hosted site through the browser using localhost. Additionally, we configured an S3 bucket for static website hosting on AWS, where we uploaded files, enabled public access, and applied the necessary bucket policies to

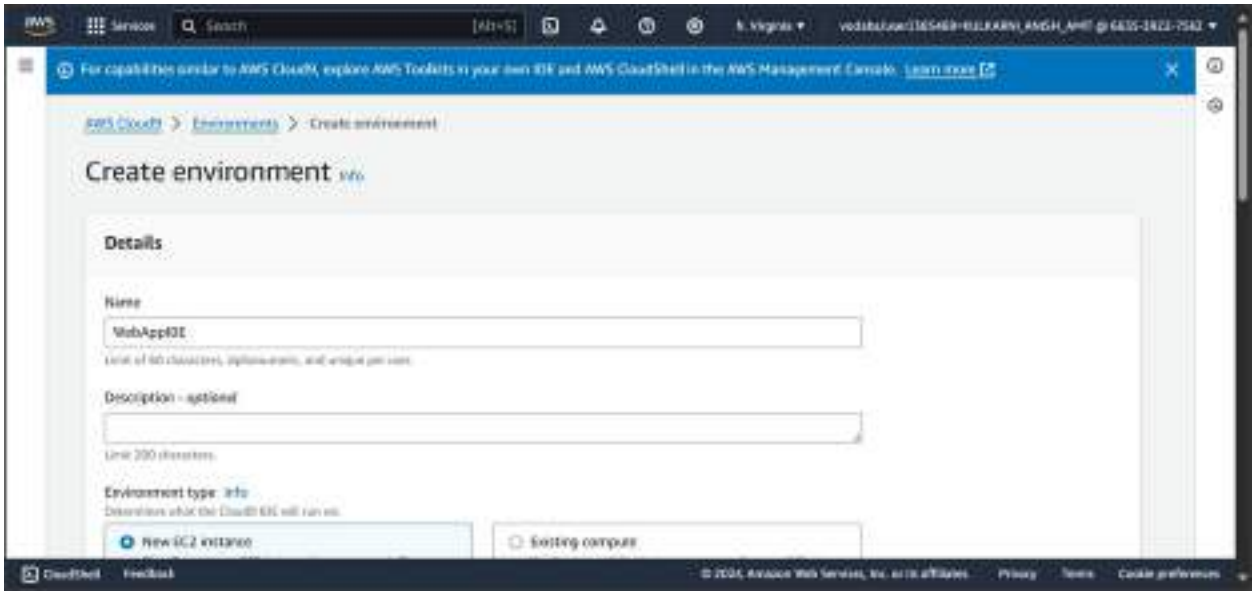
make the website accessible. This hands-on process illustrated the differences between local and cloud hosting, emphasizing the ease of deployment and configuration flexibility that each method offers.

Cloud9 setup, launch and collaboration [Exp 1(b)]

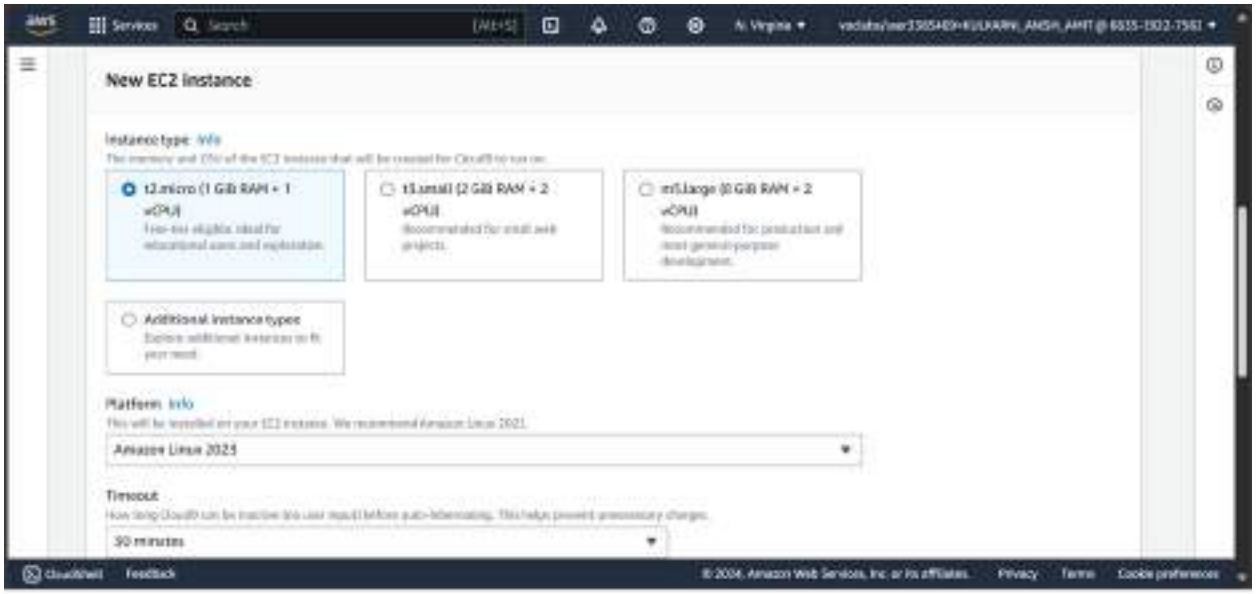
Aim: To understand the benefits of Cloud infrastructure and setup AWS Cloud9 IDE, launch AWS Cloud9 IDE and perform collaboration demonstration.

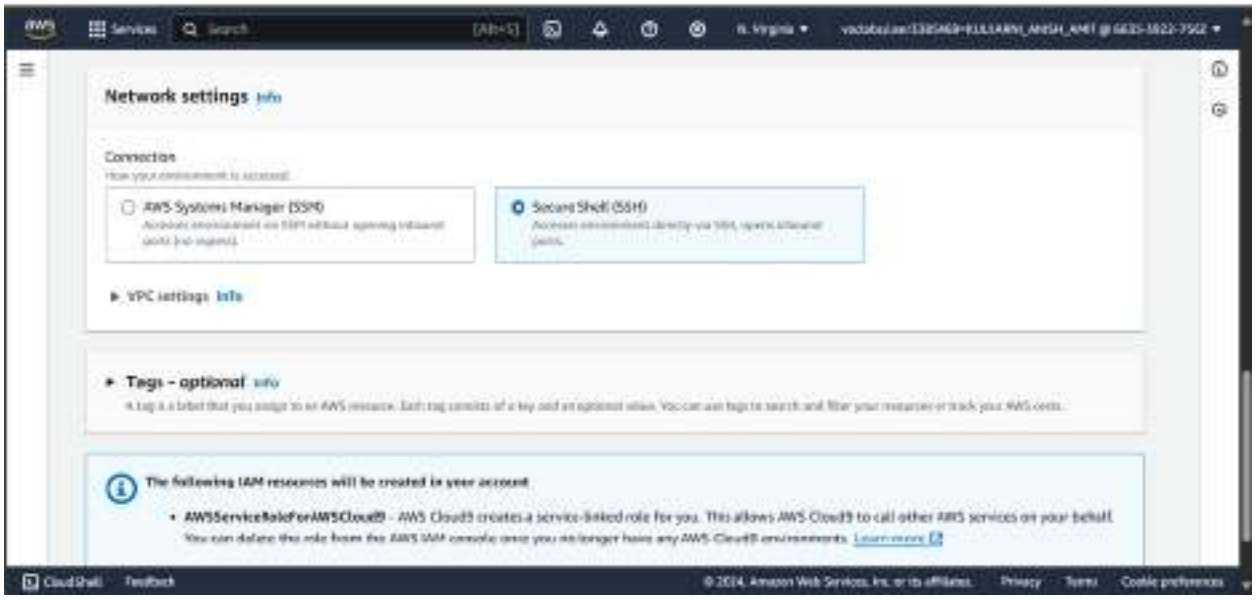
Steps:-

Step 1: Log into your AWS account and navigate to Cloud9 and click on ‘Create environment’ option. Give your environment a name.

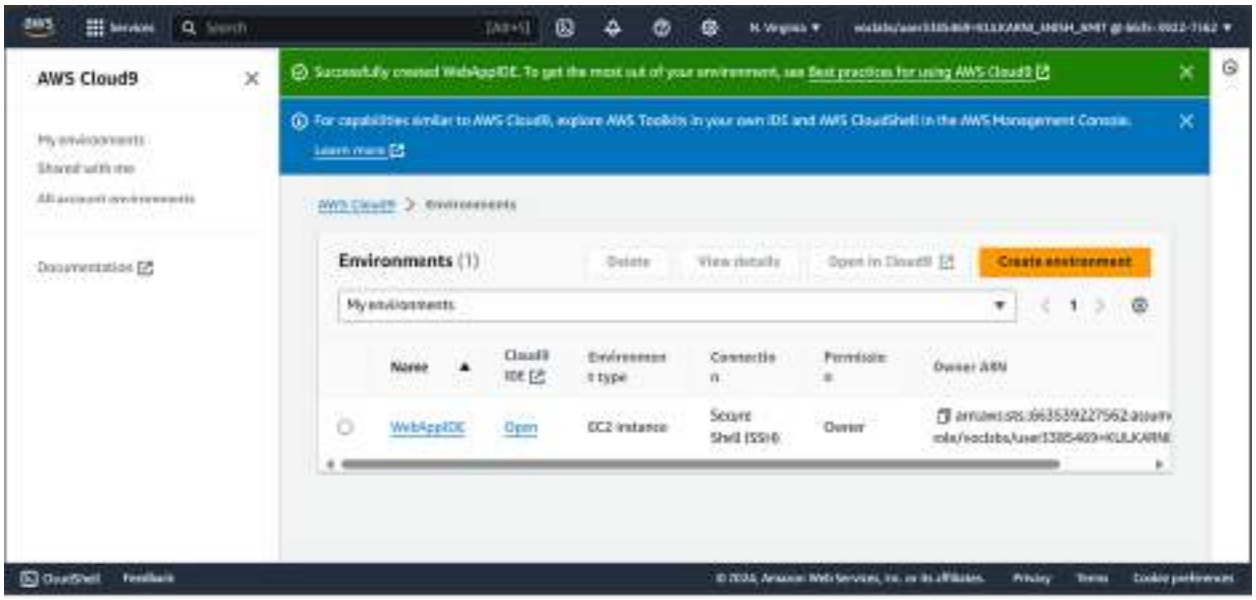


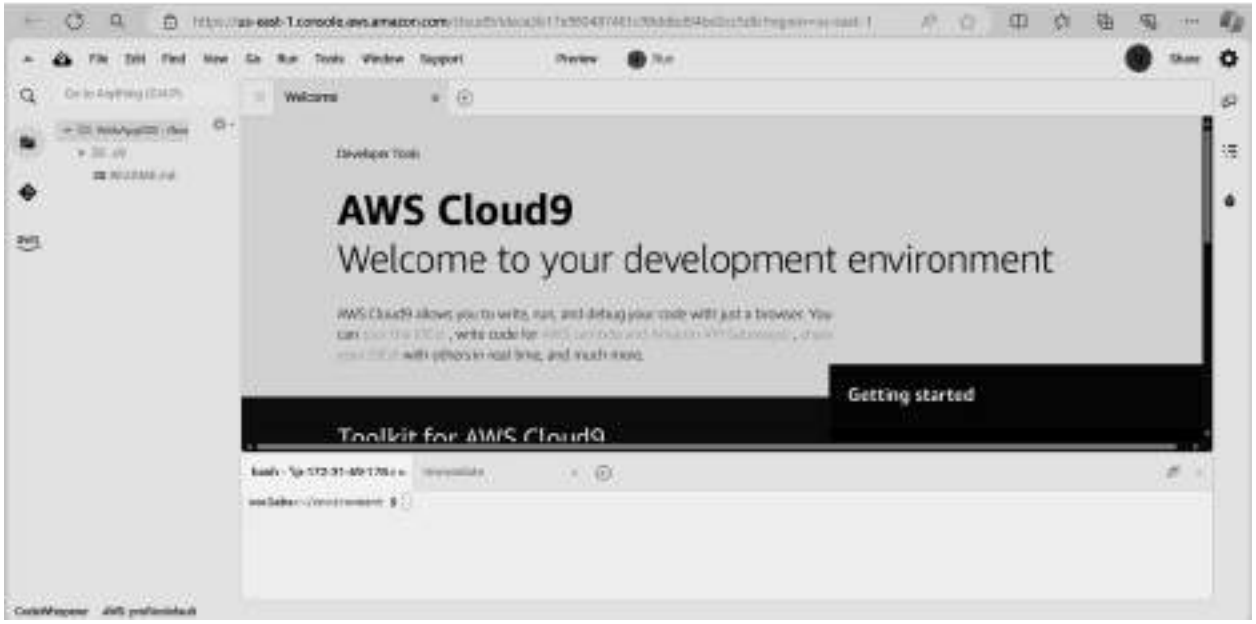
Step 2: Keep all settings as default and click on Next.





Step 3: Review your environment options and click on ‘Create environment’. Your environment is created.

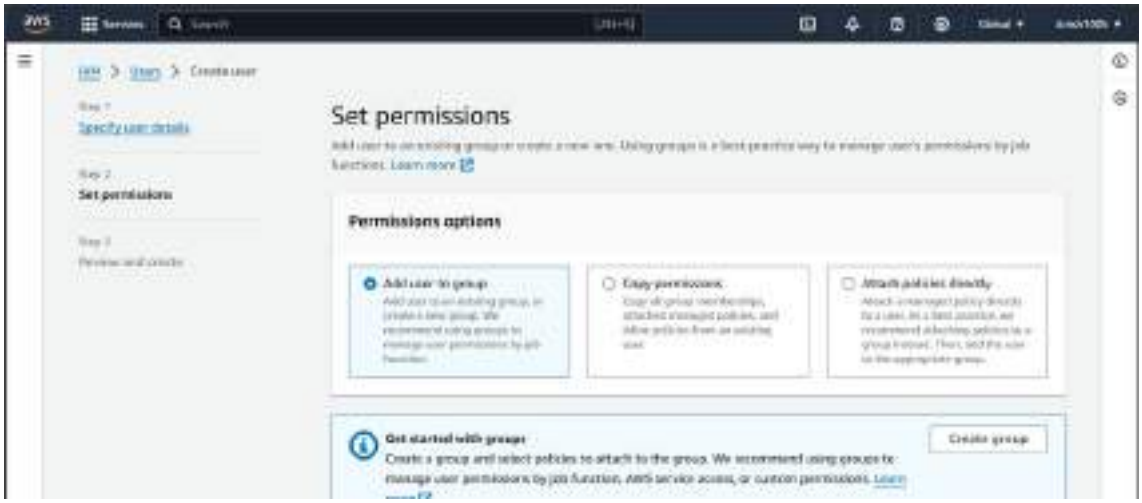




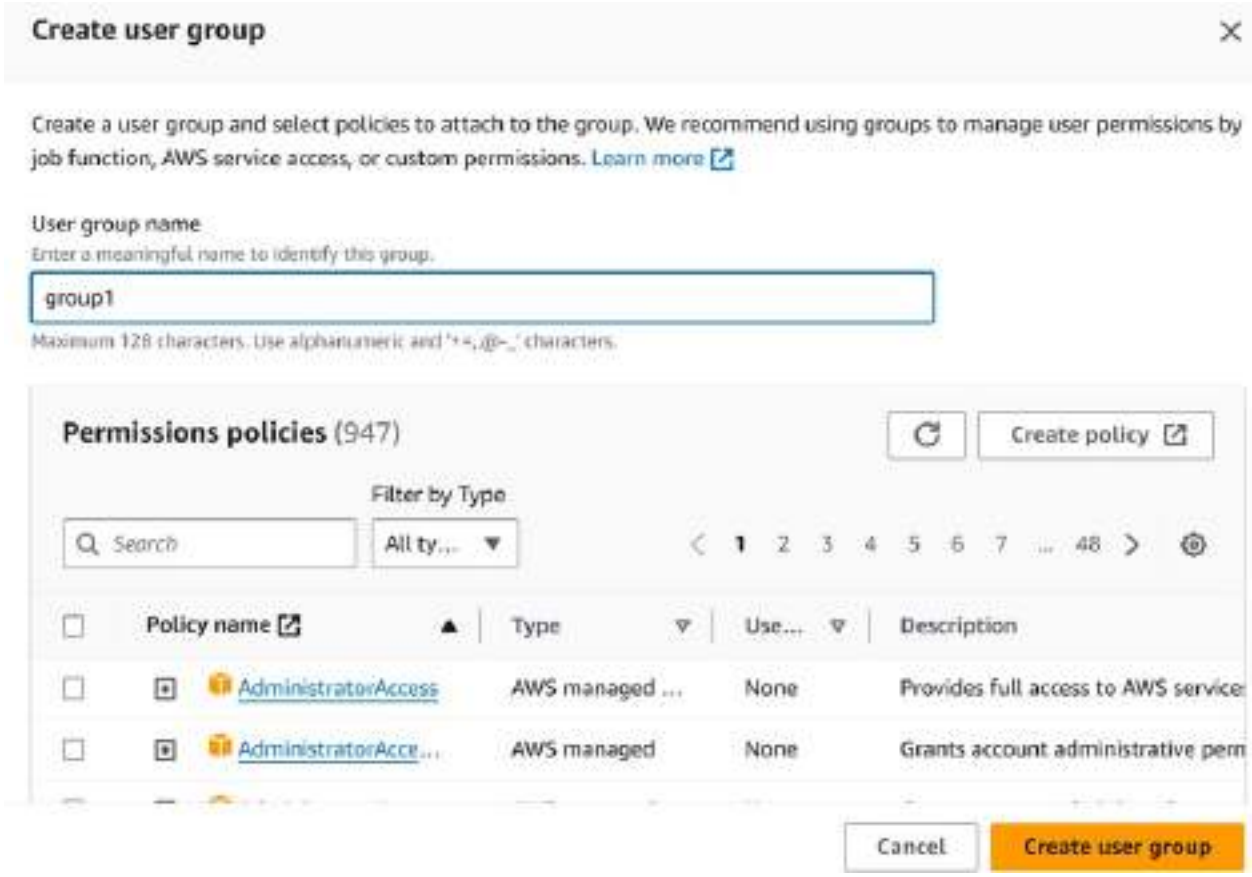
Step 4: Navigate to IAM (Identity and Access Management), click on ‘Users’ tab and click on ‘Create User’.

Step 5: Give a name to your user and click on ‘Next’.

Step 6: Set permissions for your user and click on ‘Create group’ option.

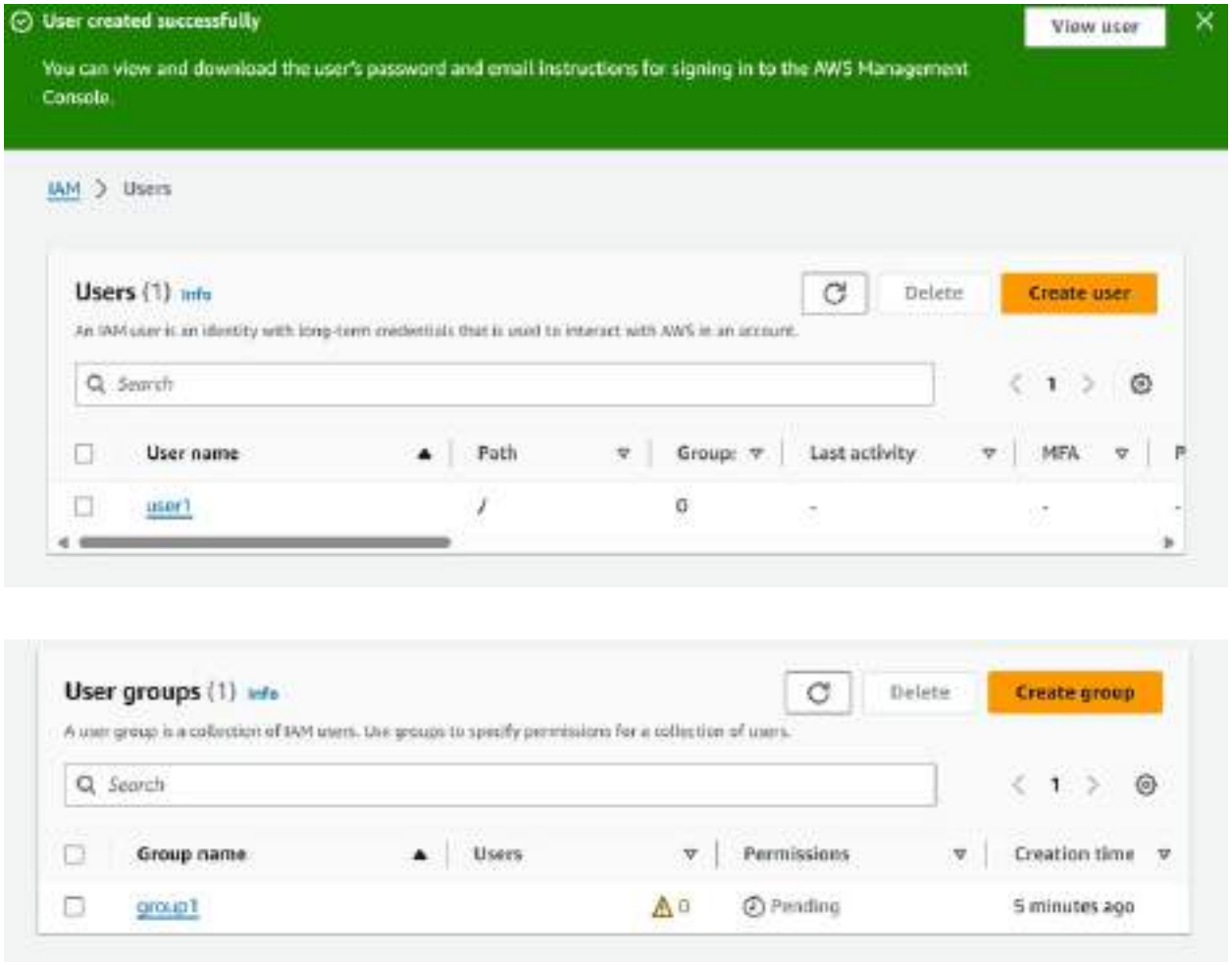


Step 7: Give a name to your user group and click on ‘Create user group’’.

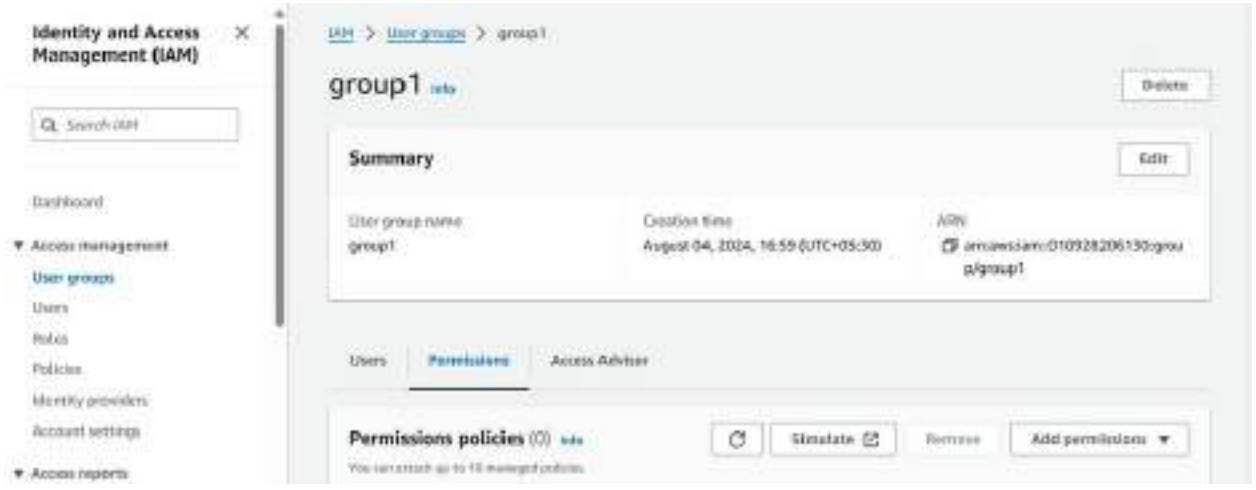


Step 8: Review your user options and click on ‘Create user’.

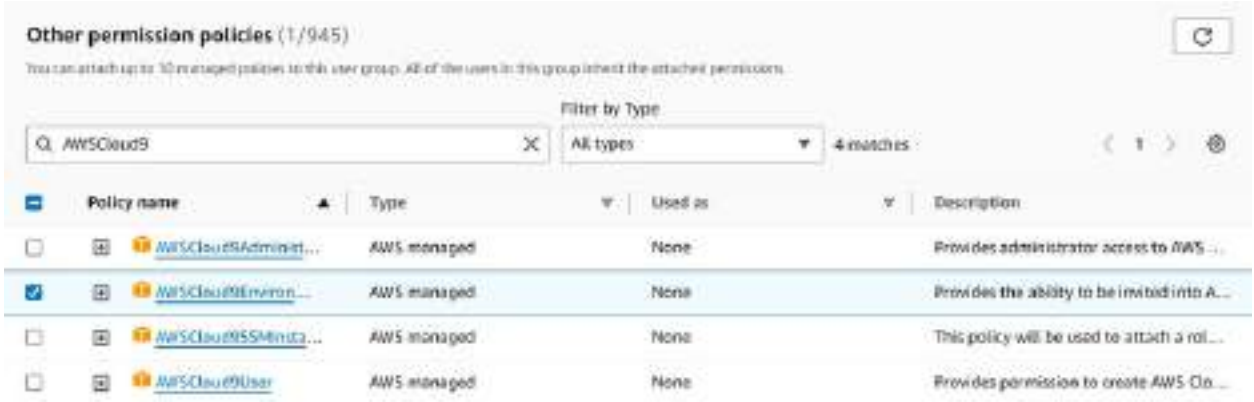
Step 9: User and User group are created successfully.



Step 10: Open the 'User groups' tab and click on the name of your group.



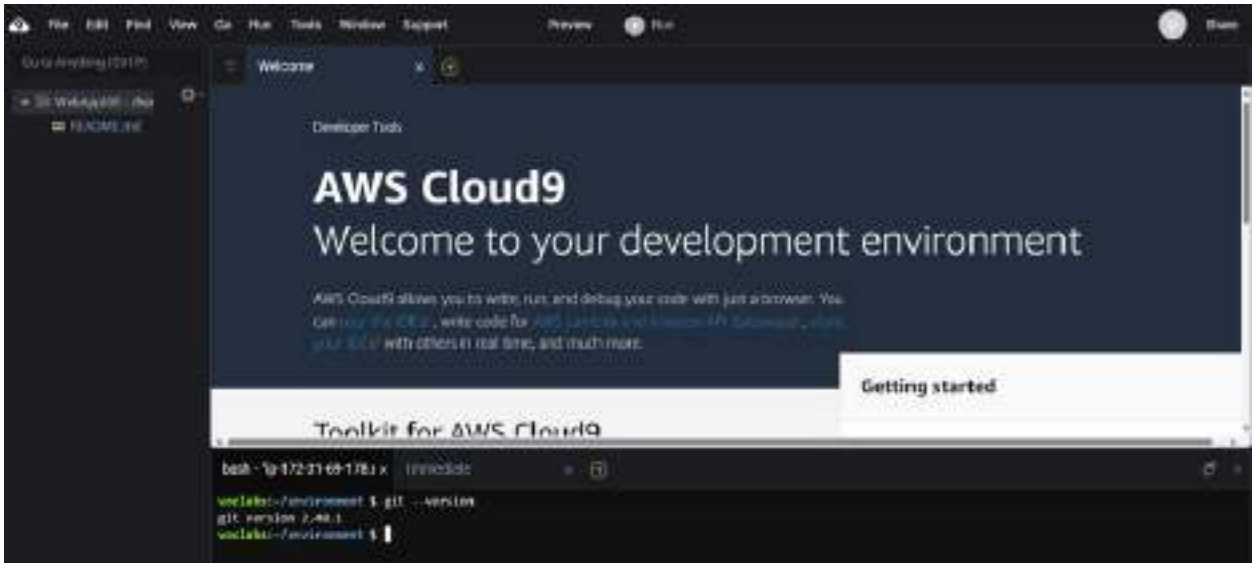
Step 11: Go to permissions and click on ‘Add permissions’. Then, click on ‘Attach policies’ and attach any policies as per your requirement.



This attaches the policies to your user group.

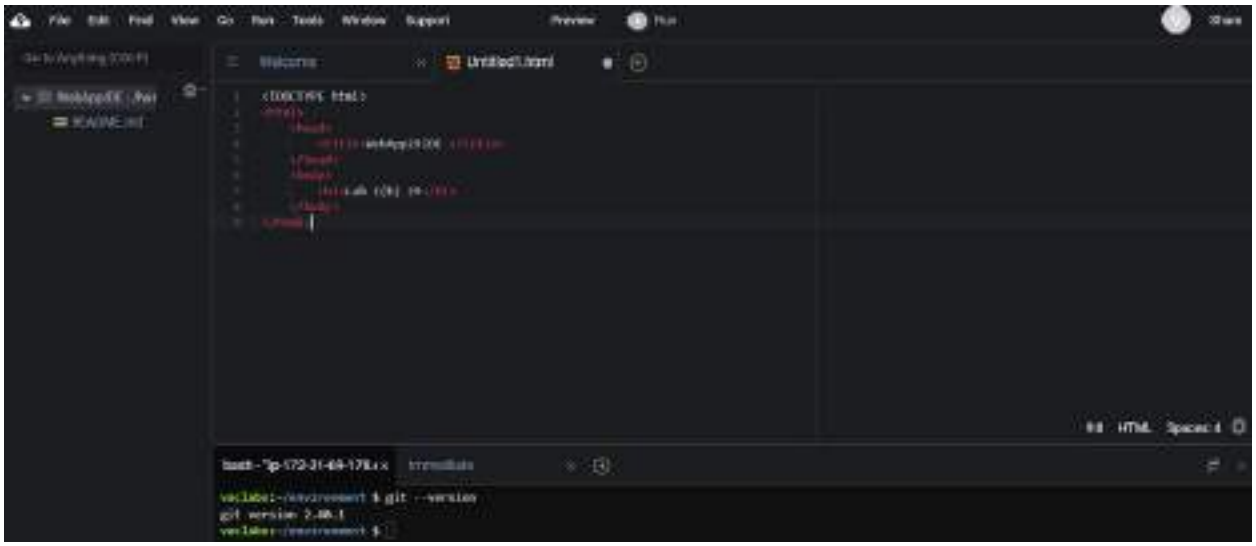
Step 12: To work on the Cloud9 IDE interface, enter commands into the command console which occupies the bottom one-third of the screen.

For example, entering the command ‘git –version’ produces the following output:-

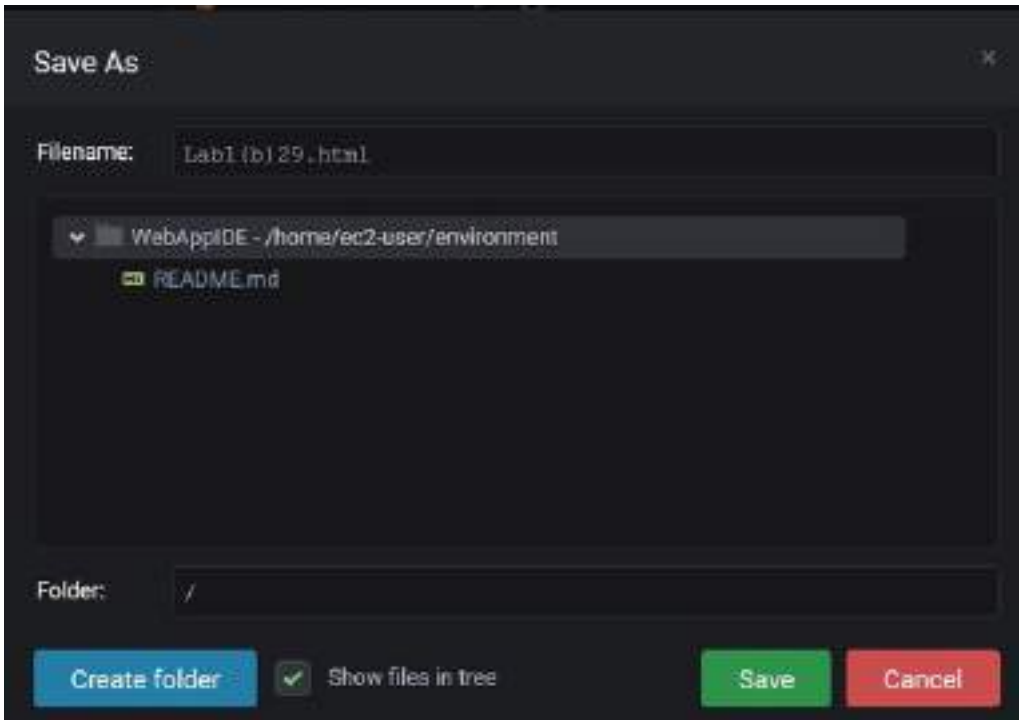


Step13:Toaddafile,clickon‘File’inthetopleftcorner,thenclickon‘NewFromTemplate’and choose the type of file you want to create.

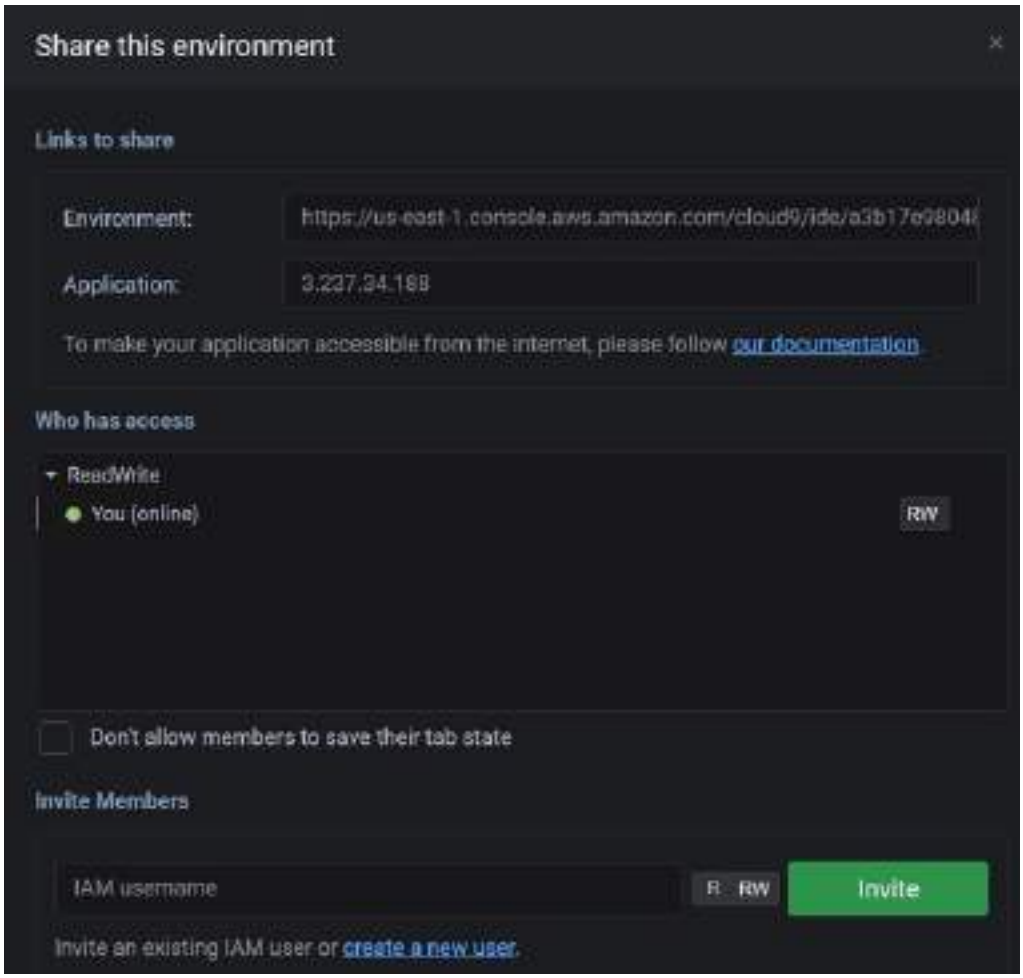
For this example, we create an HTML file. This gives a basic HTML code template on the coding IDE.



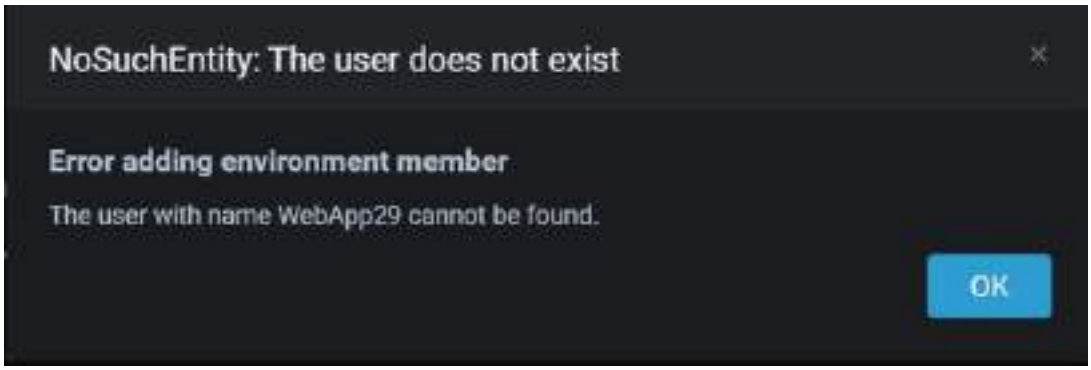
Step 14: After creating your file, click on ‘File’ in the top left corner and click on ‘Save’ from the drop-down menu.



Step 15: After sharing, click on ‘Share’ in the top-right corner. Then, enter the username that you had entered during creating your IAM user.



Doing the above prompts the following error message to pop-up. This error occurs because while the above Cloud9 activities have been performed on AWS academy, it doesn’t allow IAM users to be created on the academy account. On the other hand, while IAM users can be created on your personal account, Cloud9 services have been made inaccessible on personal accounts. Thus, integration of Cloud9 and IAM is not possible currently.



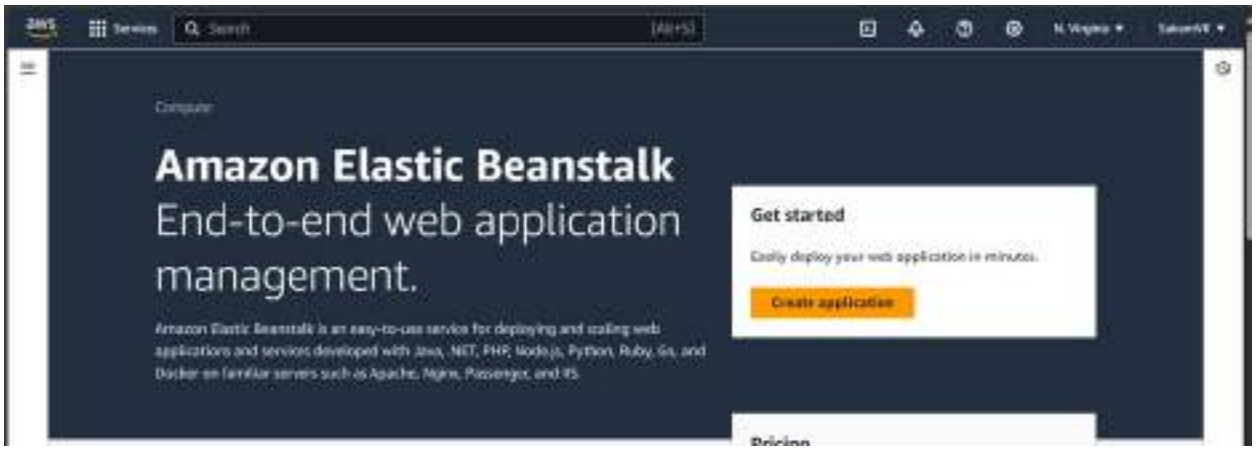
Conclusion: This experiment highlights how to set up and work with AWS Cloud9, although account limitations on AWS Academy prevent full collaboration demonstration. By understanding the steps and limitations, users can better leverage cloud infrastructure for development and teamwork when fully functional accounts are available.

Experiment No. 2

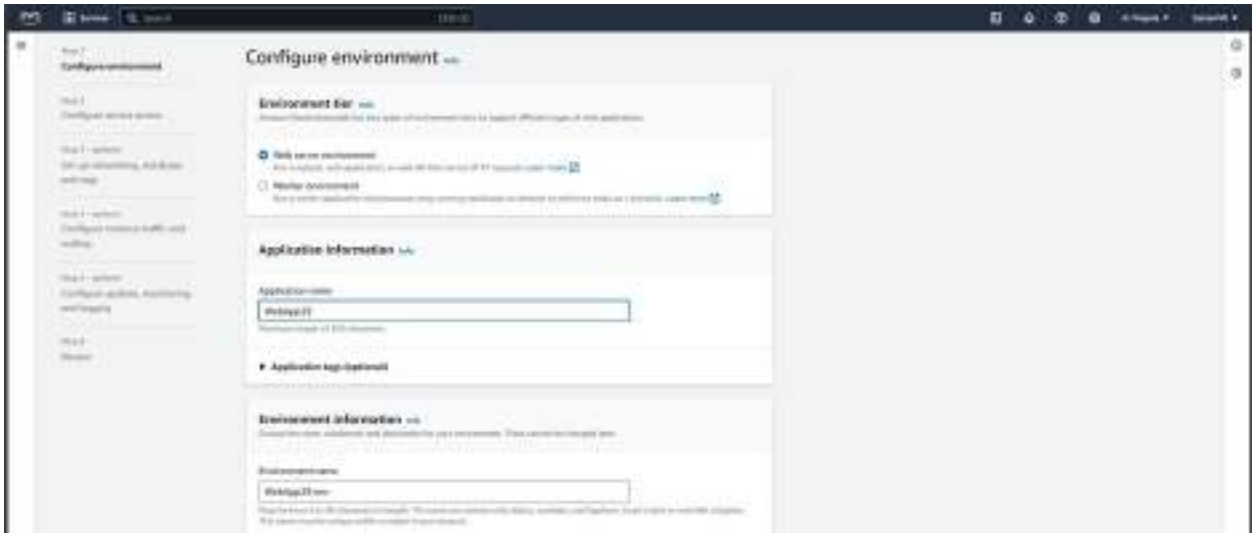
Aim: To build your application using AWS Codebuild and deploy on S3/SEBS using AWS CodePipeline, deploy a sample application on an EC2 instance using AWS CodeDeploy.

Steps:-

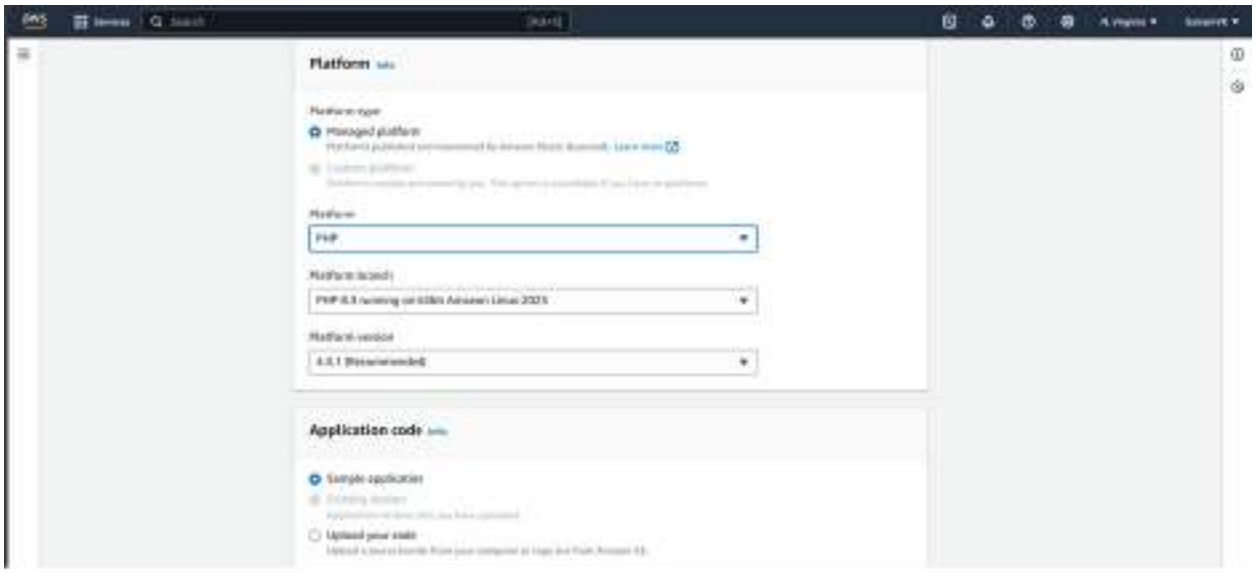
Step 1: Log into AWS and navigate to ‘Elastic Beanstalk’. Then, click on ‘Create application’.

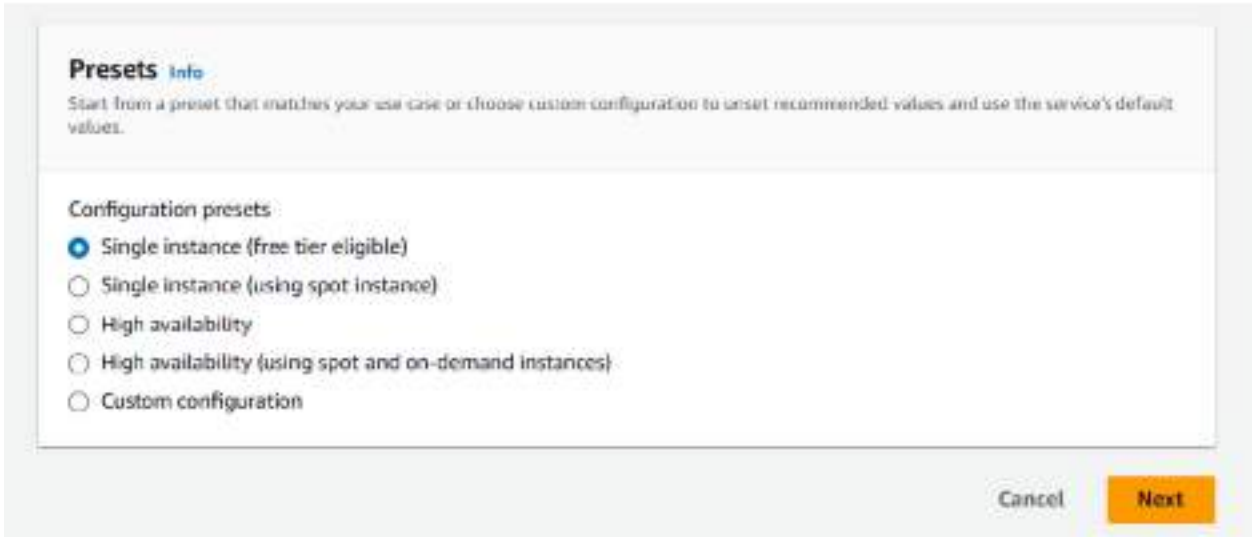


Step 2: Give your environment a name.

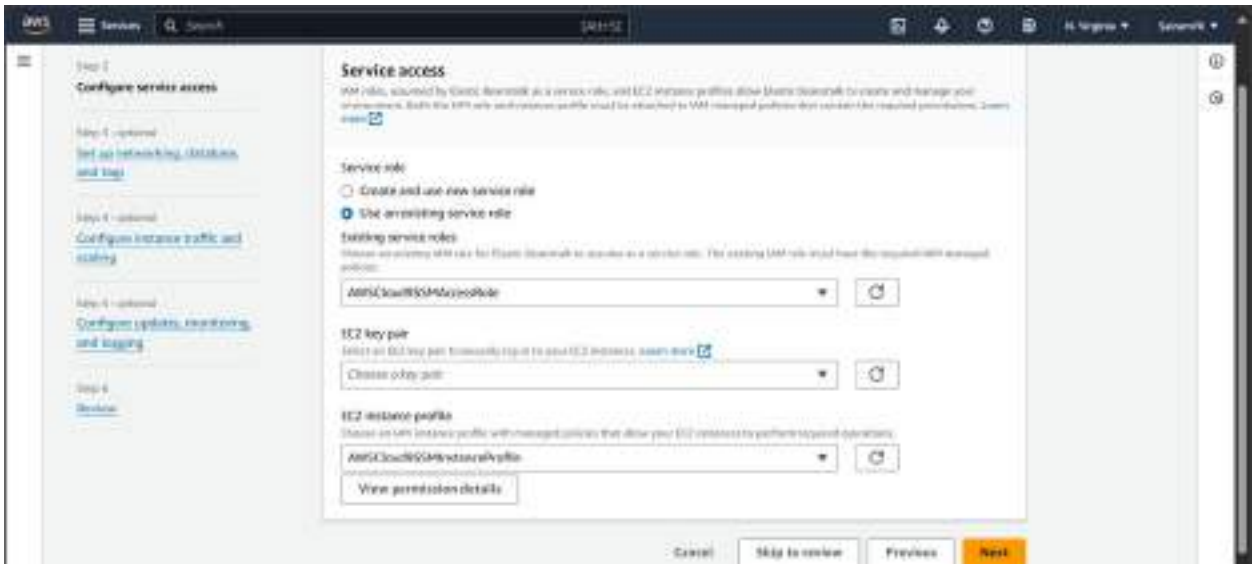


Step 3: In the ‘Platform’ drop-down box, choose PHP. Keep all other settings as default and click on ‘Next’.

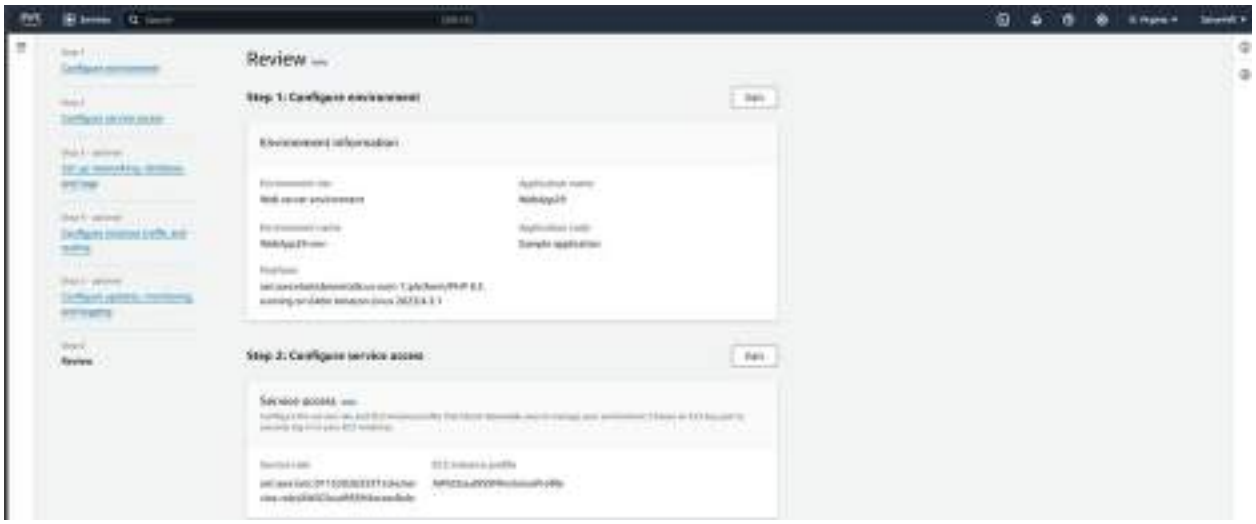


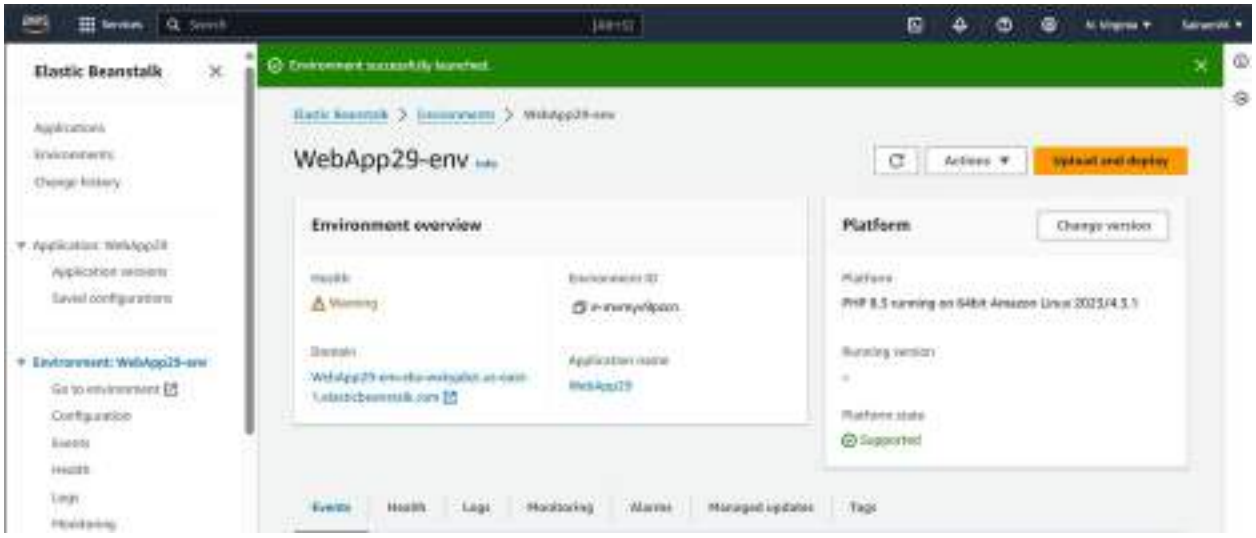


Step 4: In the ‘Service access’ section, choose ‘AWSCloud9AAMAccessRole’ in ‘Existing service roles’ and ‘AWSCloud9SSMInstanceProfile’ in ‘EC2 instance profile’.



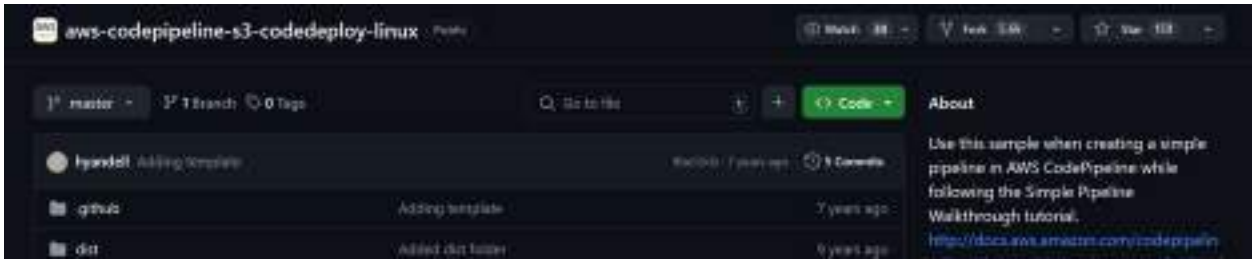
Step 5: Review all the environment settings and click on ‘Submit’.

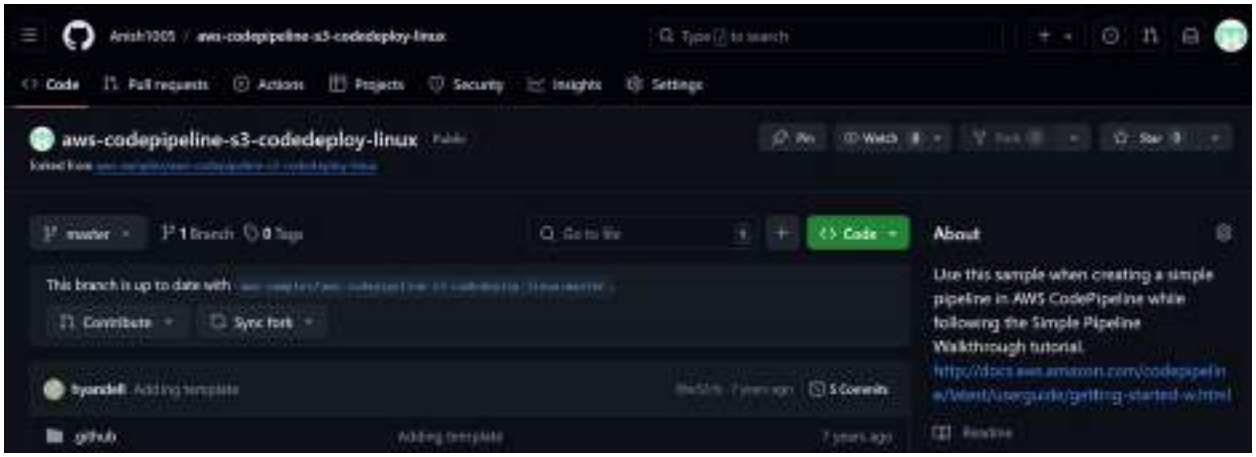




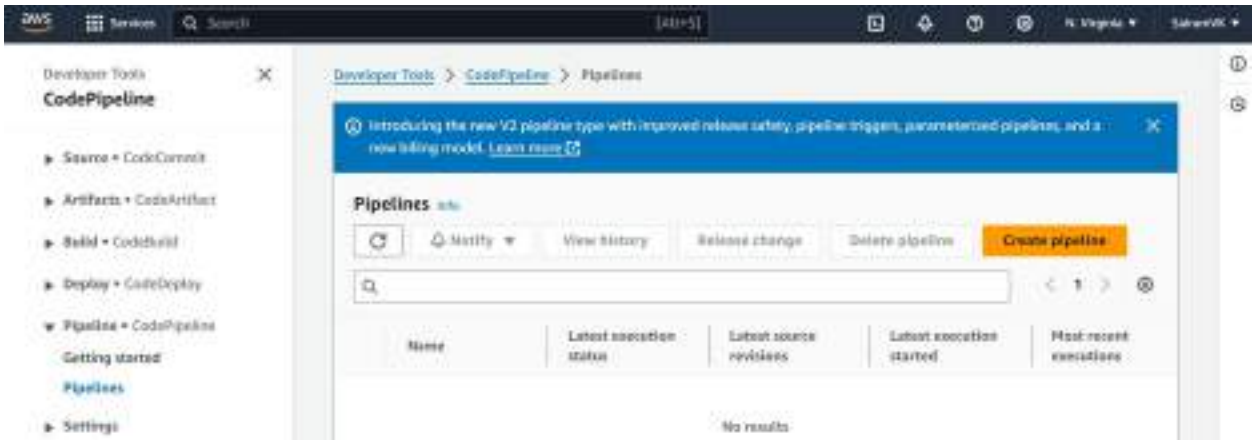
The environment gets created.

Step 6: Go to the github link below and fork the repository into your personal github in order to get the sample code for deploying a file on AWS CodePipeline.
https://github.com/aws-samples/aws-codepipeline-s3-codedeploy-linux

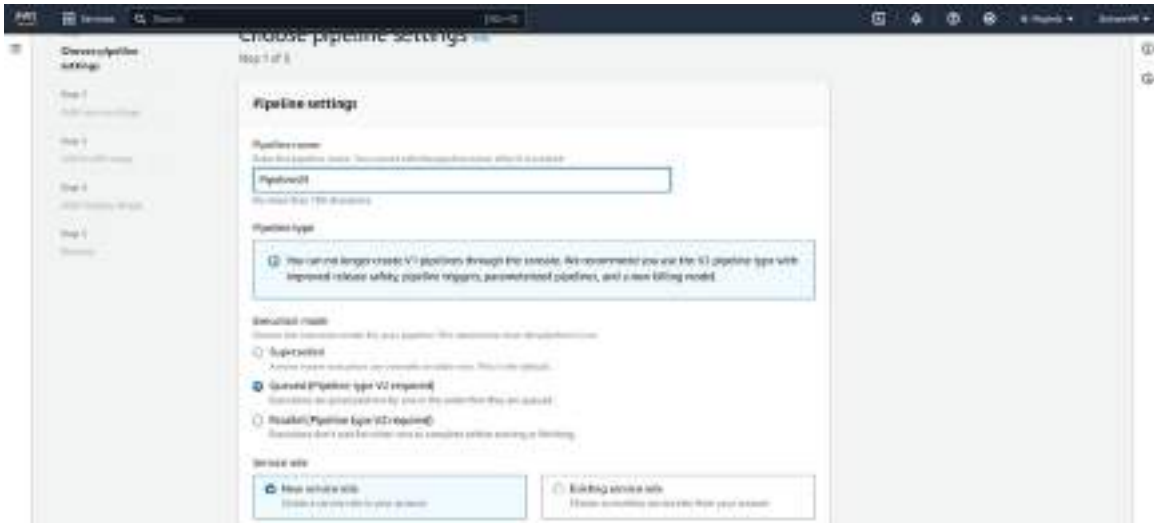




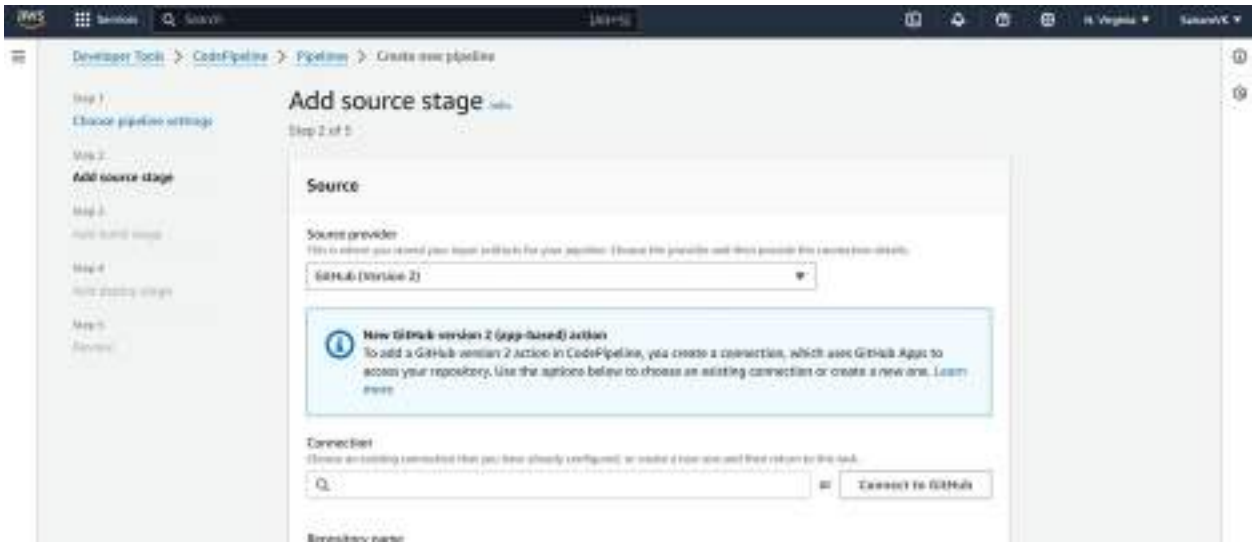
Step 7: Navigate to CodePipeline and click on ‘Create Pipeline’.



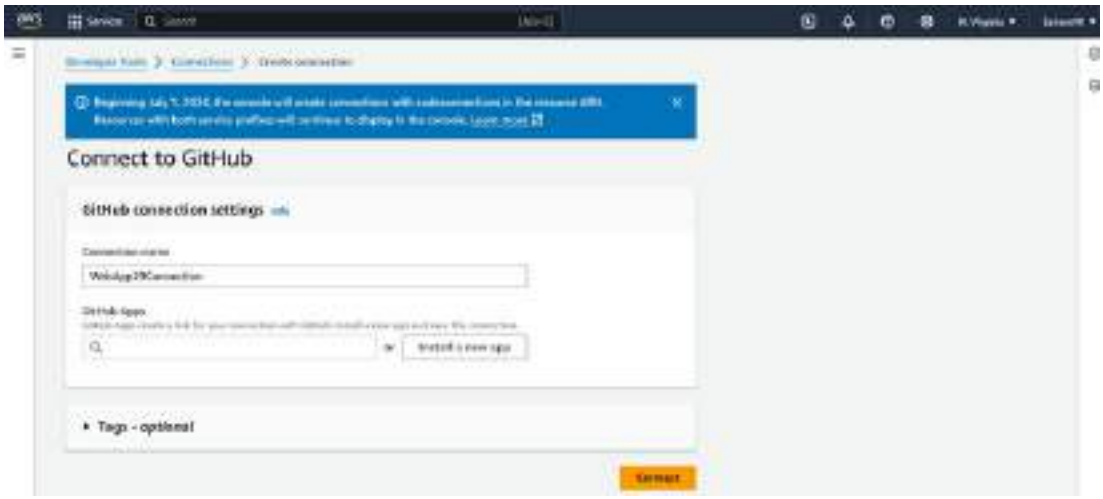
Step 8: Give your pipeline a name. A new service role is also created with the name of the pipeline.



Step 9: Select Github (Version 2) as source provider and click on ‘Connect to Github’ to connect the pipeline to your Github.

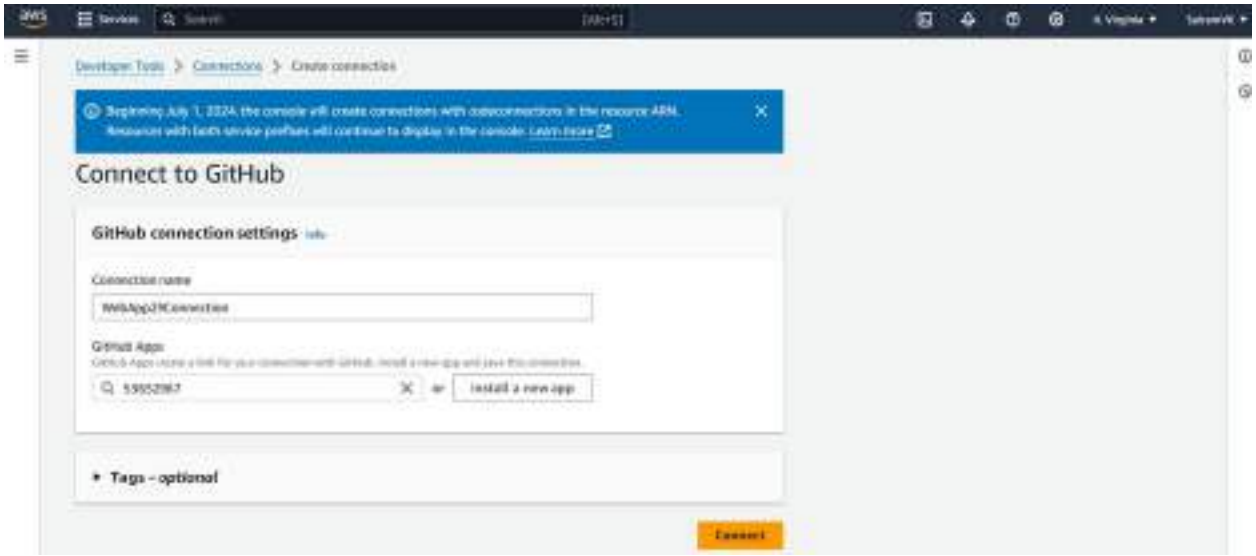


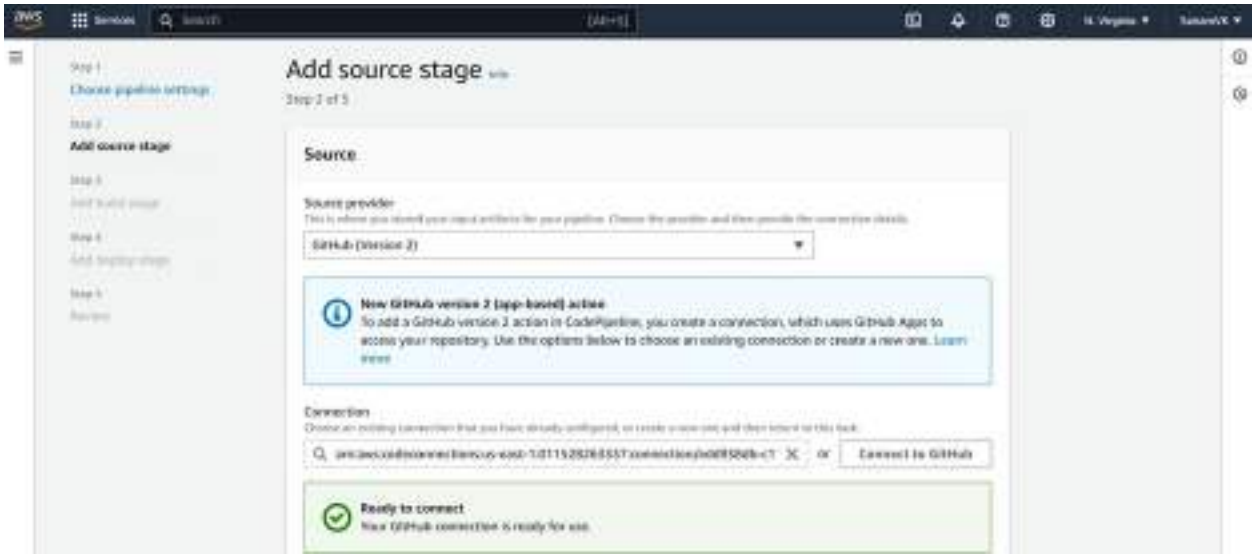
Step 10: Give your connection a name and click on ‘Connect to Github’. Then, either provide a link for connection or install the app if it is your first time.



Step 11: Install AWS Connector for Github.

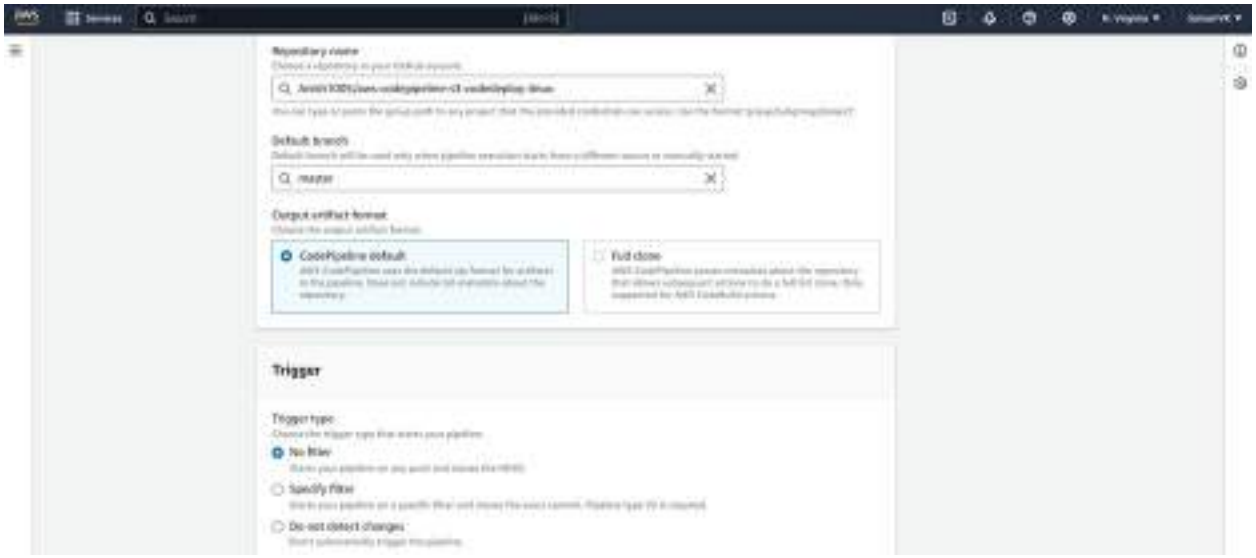
Step 12: Once installation is done, the text field is filled automatically. Click on ‘Connect’.



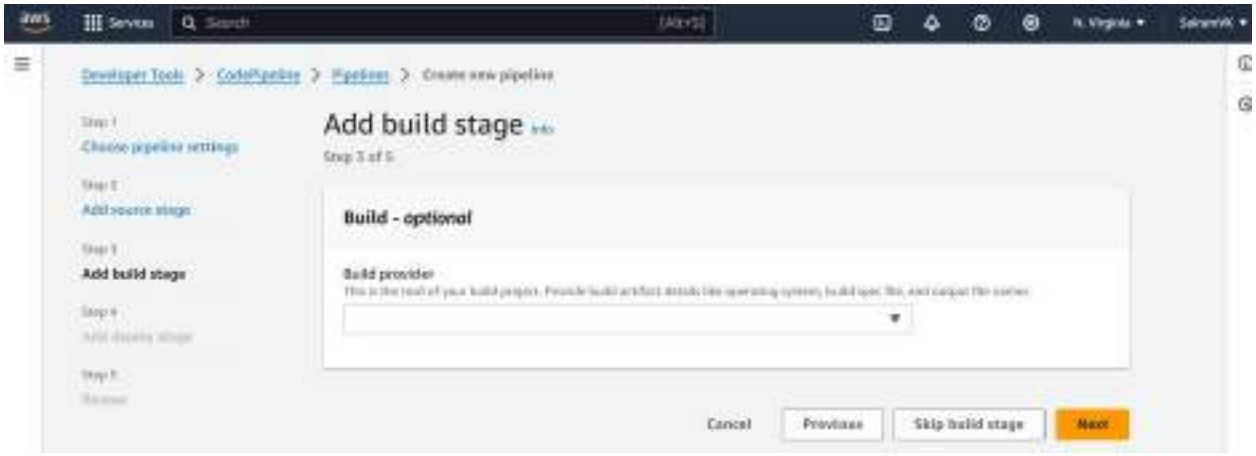


AWS shows that the Github connection is ready to use.

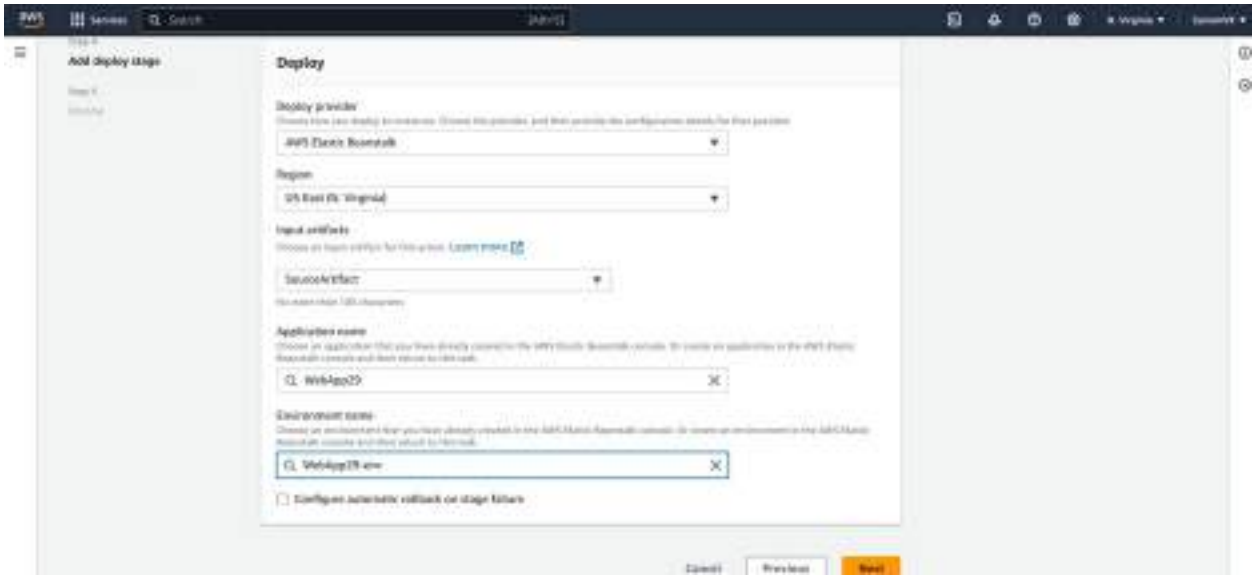
Step 13: Select the repository that you forked the sample code to be deployed to and choose the branch on which the files are present (‘master’ is set as default). Also set the trigger type as ‘no filter’.



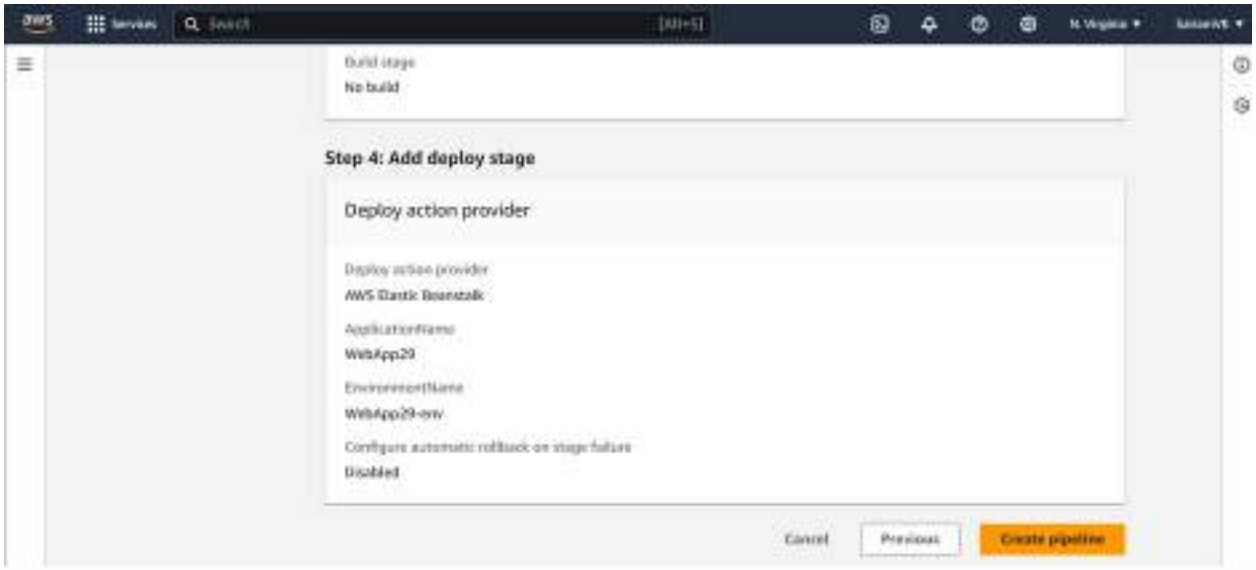
Step 14: Skip the build stage and go to deploy stage.

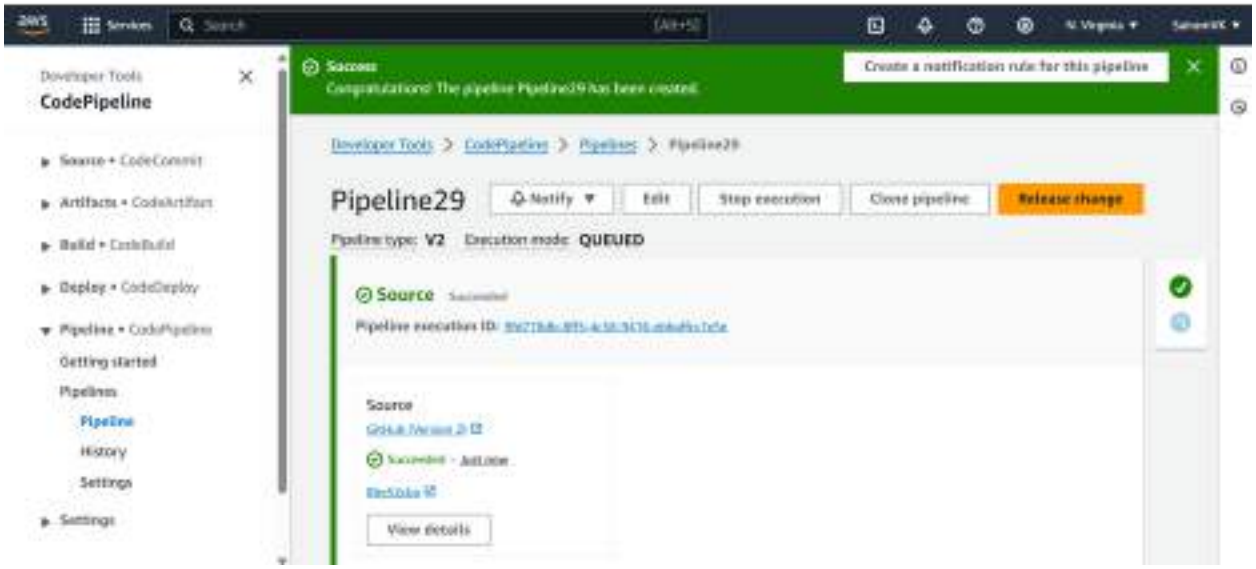


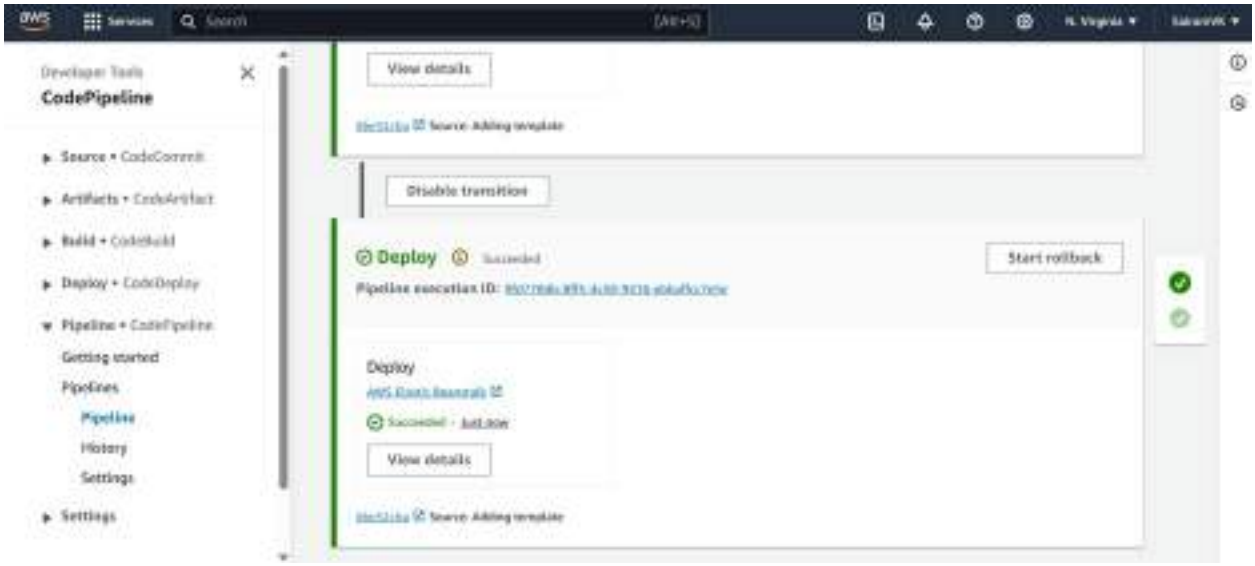
Step 15: Choose ‘AWS Elastic Beanstalk’ as deploy provider and input artifacts as ‘SourceArtifact’. Then, enter the names of your application and environment.



Step 16: Review all pipeline information and click on ‘Create pipeline’.

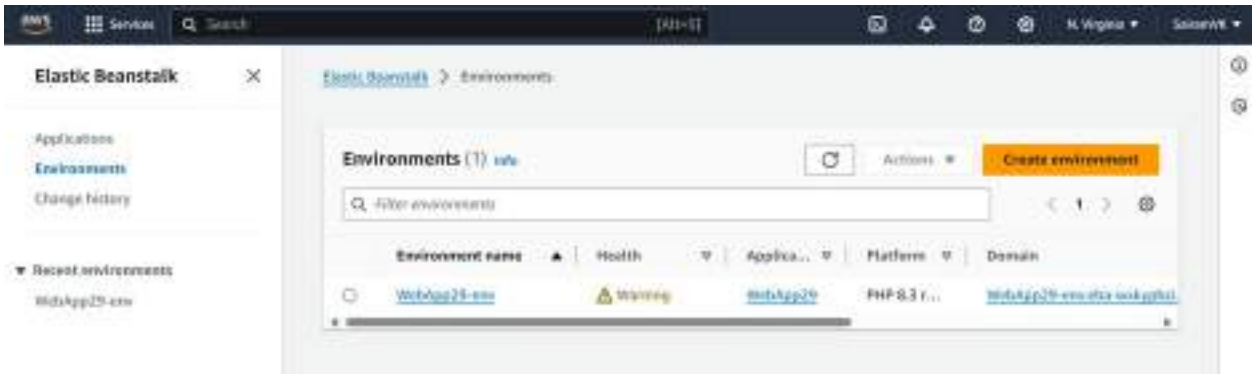




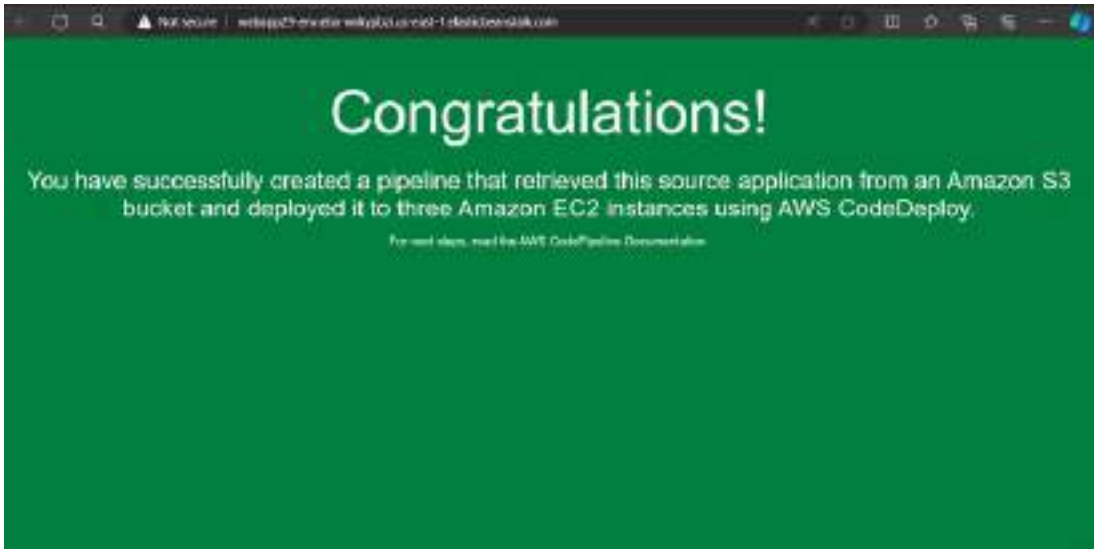


The pipeline is created and deployment is complete.

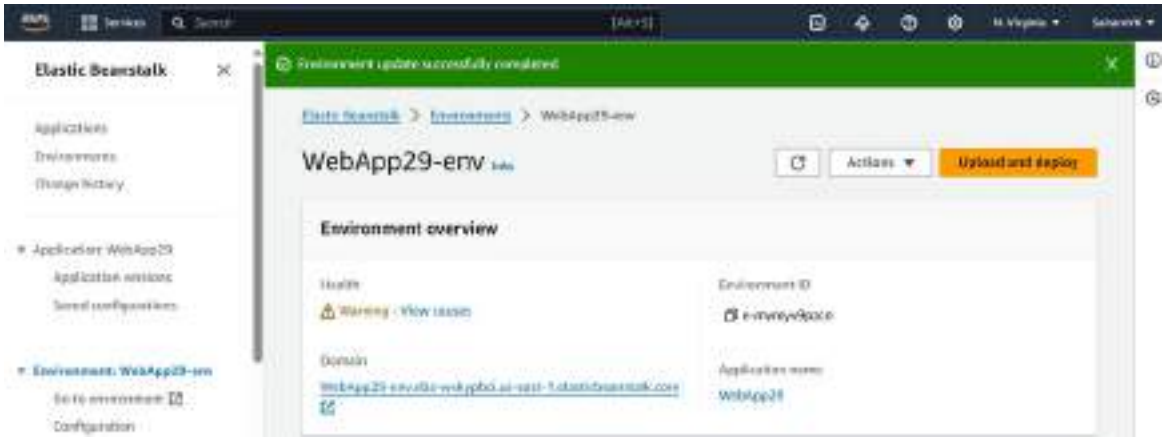
Step 17: Clicking on ‘AWS Elastic Beanstalk’ under ‘Deploy’ redirects you to the environments screen of Elastic Beanstalk. Click on the link under ‘Domain’.



Step 18: The sample website is displayed showing that the website was successfully hosted.



Step 19: Making changes to the index file in the github will update the website and changes made to the website are visible.



Conclusion: This process showcases the ability to automate the entire lifecycle of building, deploying, and managing updates for an application using AWS services. By utilizing AWS CodeBuild, AWS CodePipeline, and Elastic Beanstalk, you streamline the deployment of a sample application and ensure continuous delivery. Additionally, by integrating GitHub, any changes to the source code are automatically deployed, making the process efficient and scalable. This approach simplifies application management, reduces manual intervention, and enhances collaboration, making it an ideal choice for modern cloud-based software development.

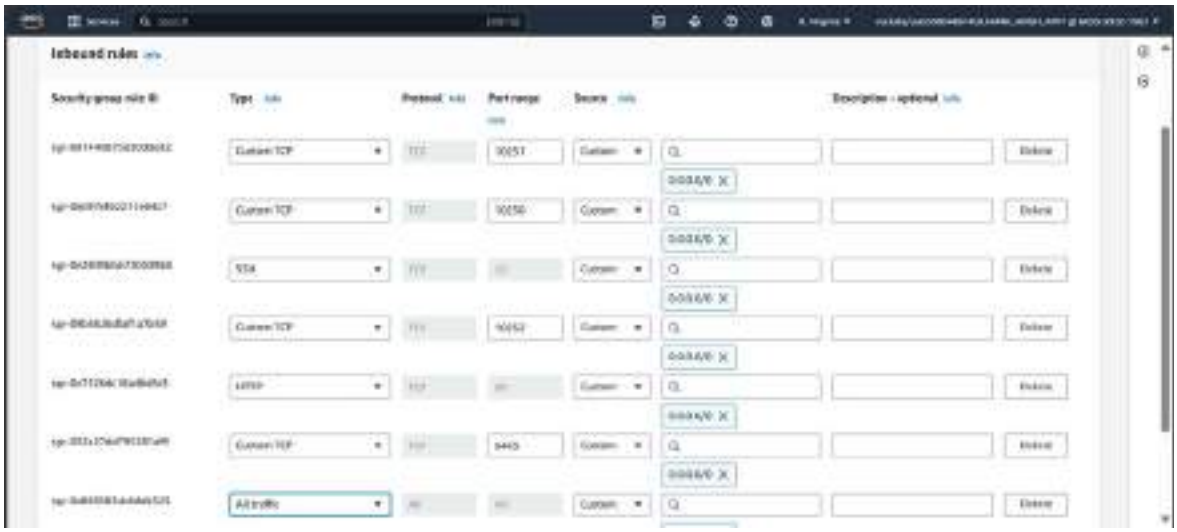
Experiment 3

Aim:TounderstandtheKubernetesClusterArchitecture,installandSpinUpa Kubernetes Cluster on Linux Machines/Cloud Platforms.

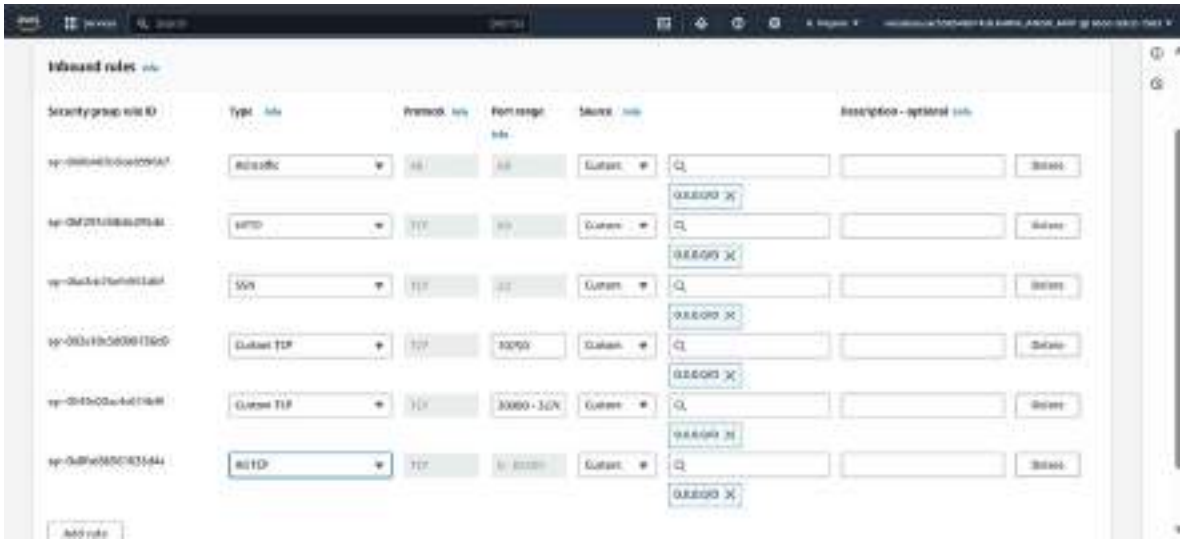
Steps:

Step 1: Create separate security groups for the master and worker nodes and add the following inbound rules. To do so, click on ‘Security groups’ in the left sidebar and click on ‘Edit inbound rules’.

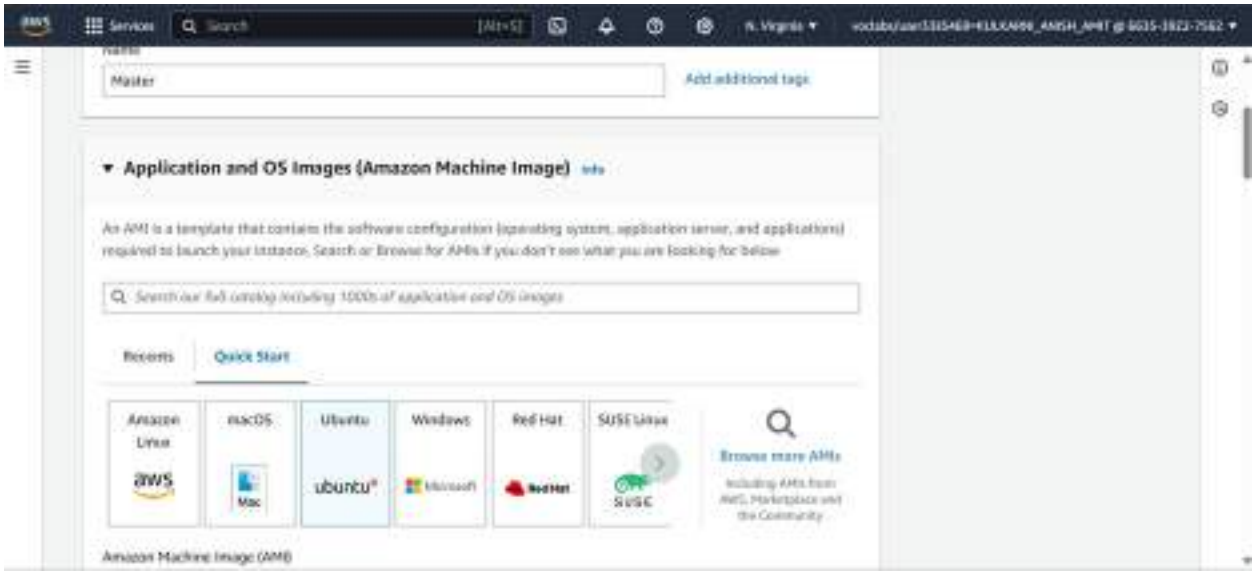
- Master:-



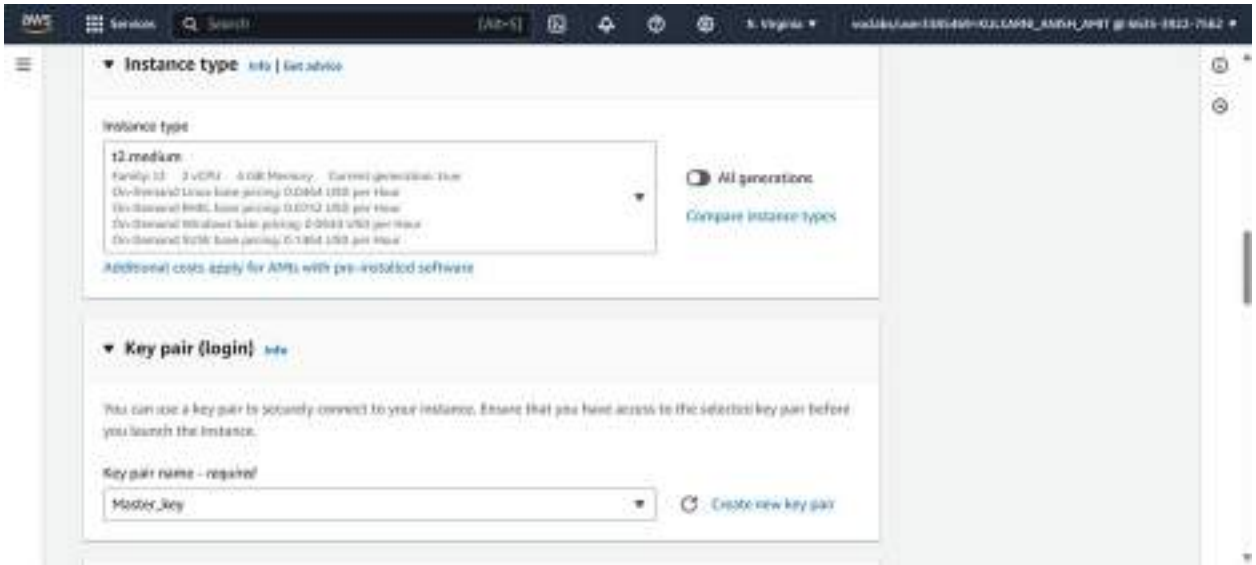
- Nodes:-



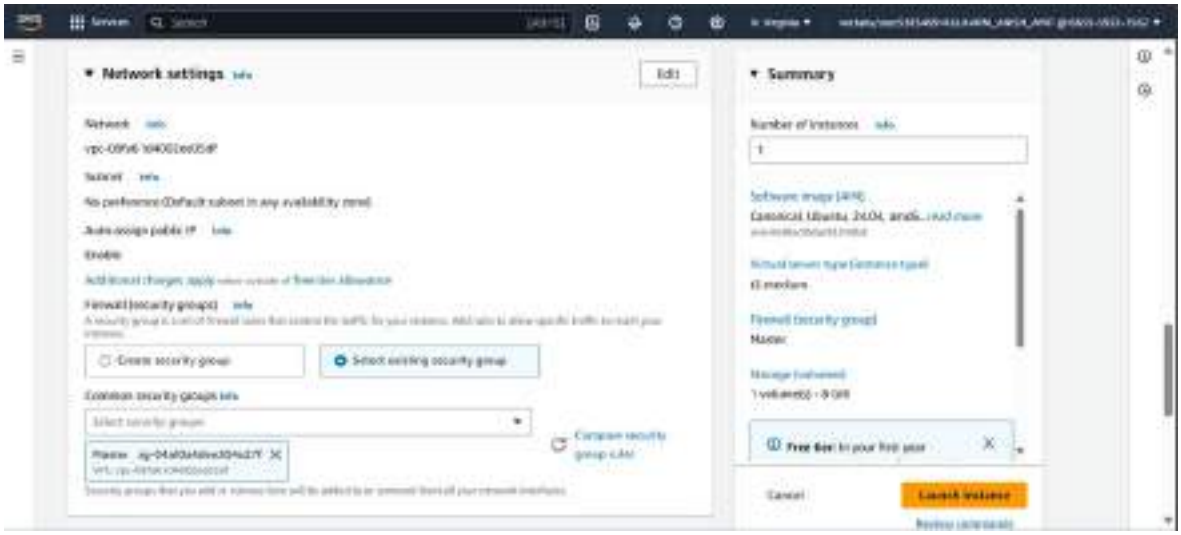
Step 2: Create 3 new EC2 instances in AWS (1 master and 2 nodes). Make sure that you choose Ubuntu as the instance type.



Under ‘Instance type’, choose t2.medium because the default (t2.micro) does not provide the sufficient resources required to execute this experiment.

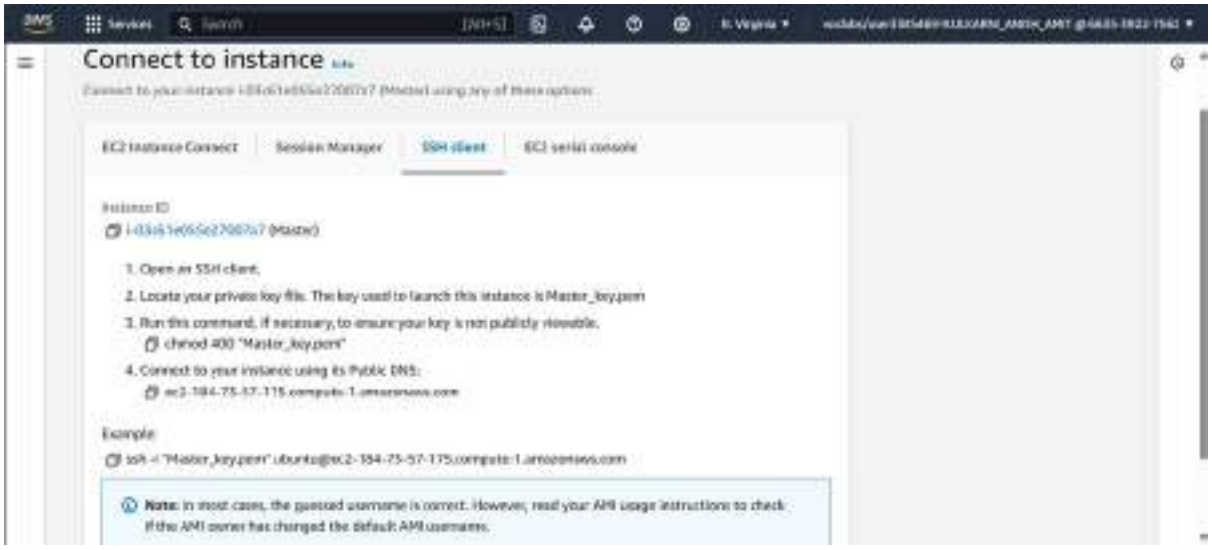


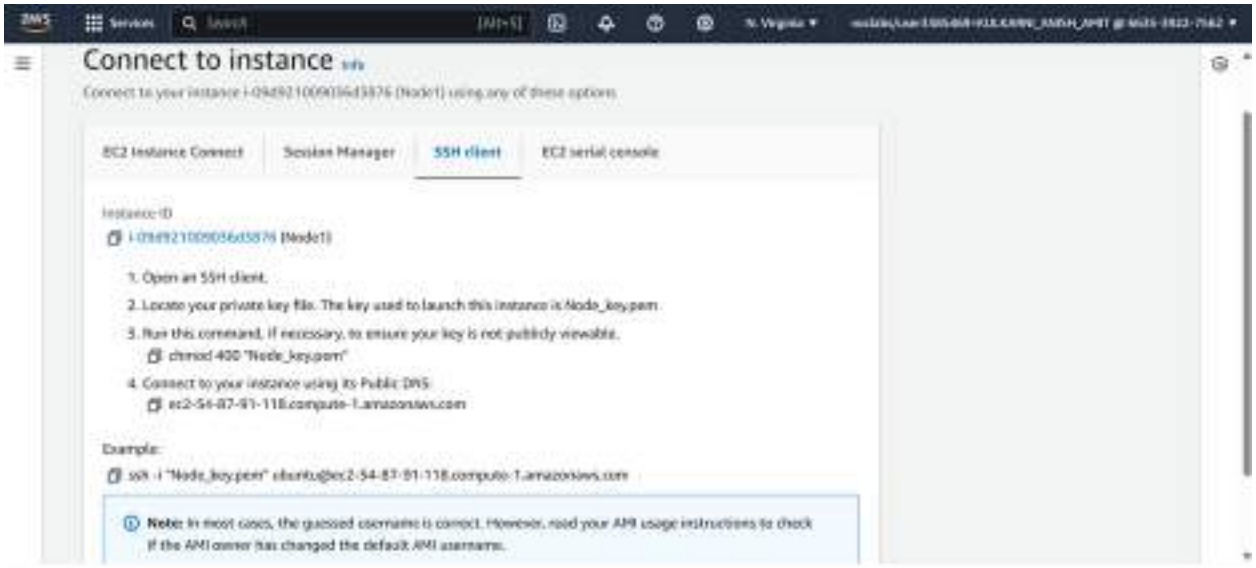
Create 2 separate key pairs (one for the master and one for the other 2 nodes). Under ‘Network settings’, click on ‘Select existing security group and for the master, choose the Master key-pair and for the other 2 nodes, choose the Node key-pair.

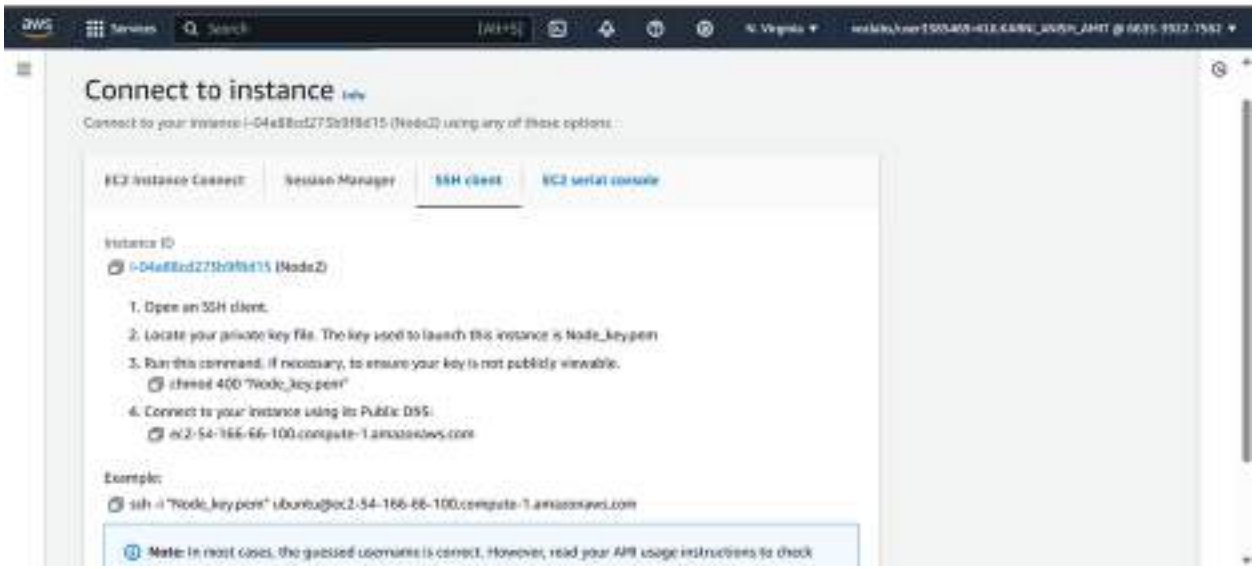


All the instances are created as above.

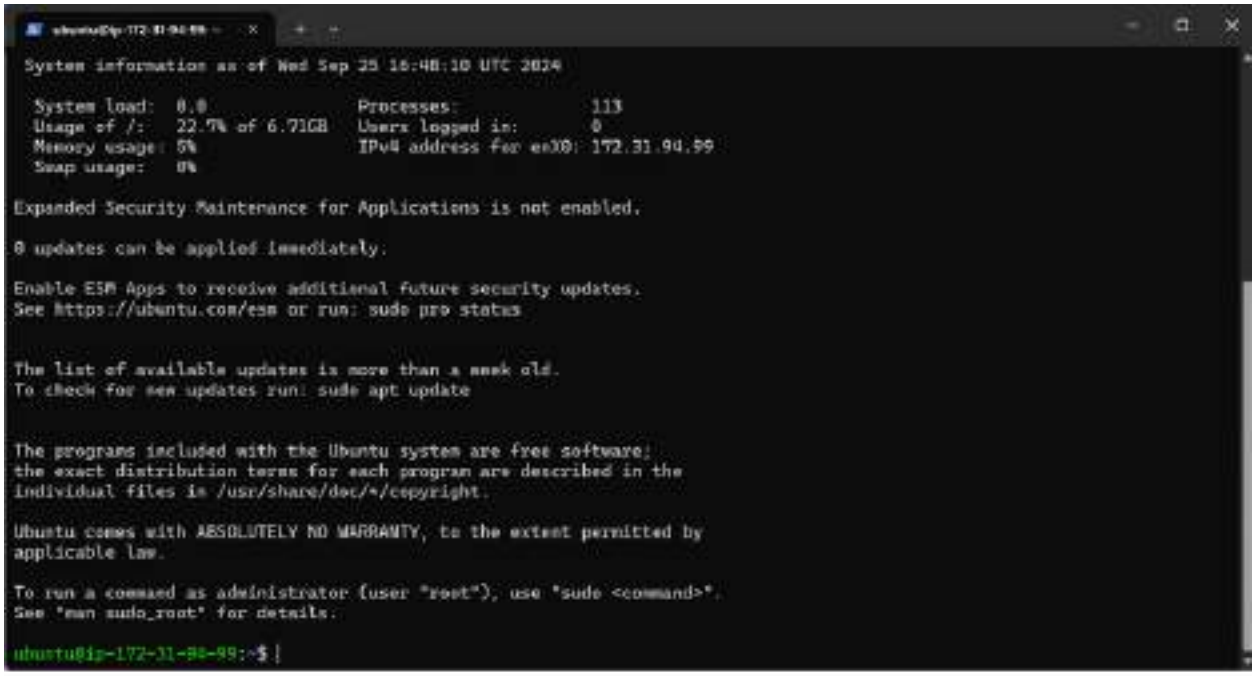
Step 3: Now, we must connect each instance to a SSH. To do so, click on an instance and click on ‘Connect’. Next, navigate to the ‘SSH client’ section and copy the command under the ‘Example’ section.

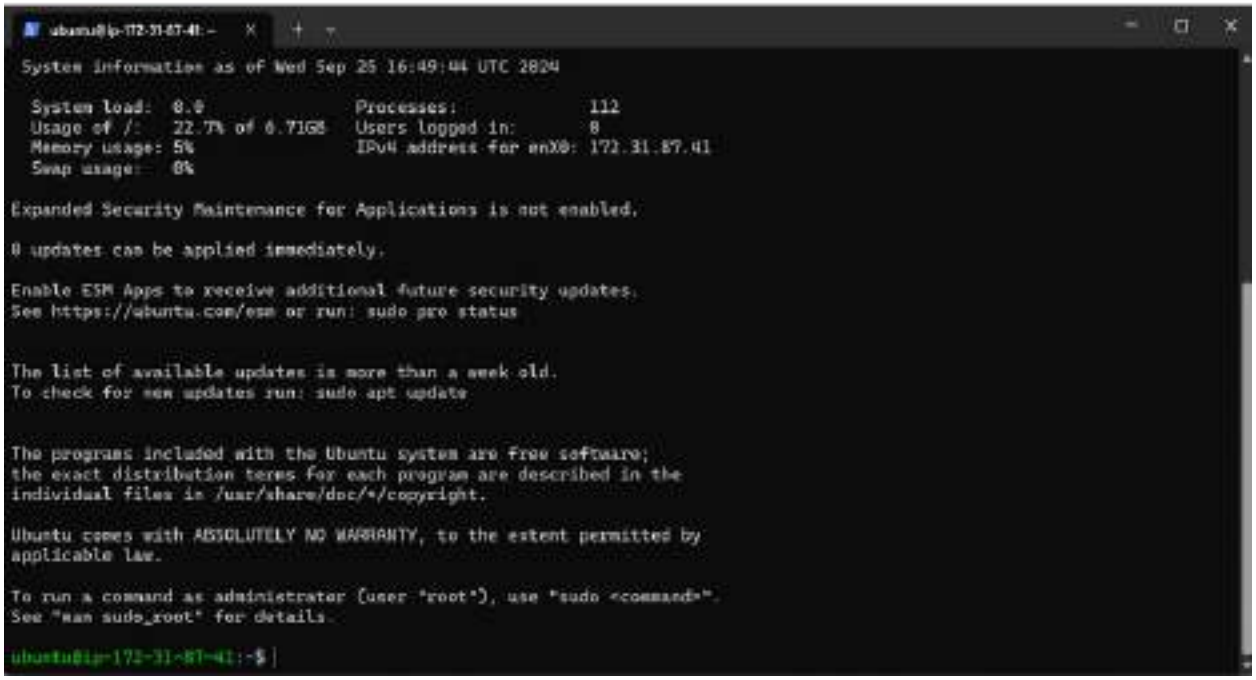


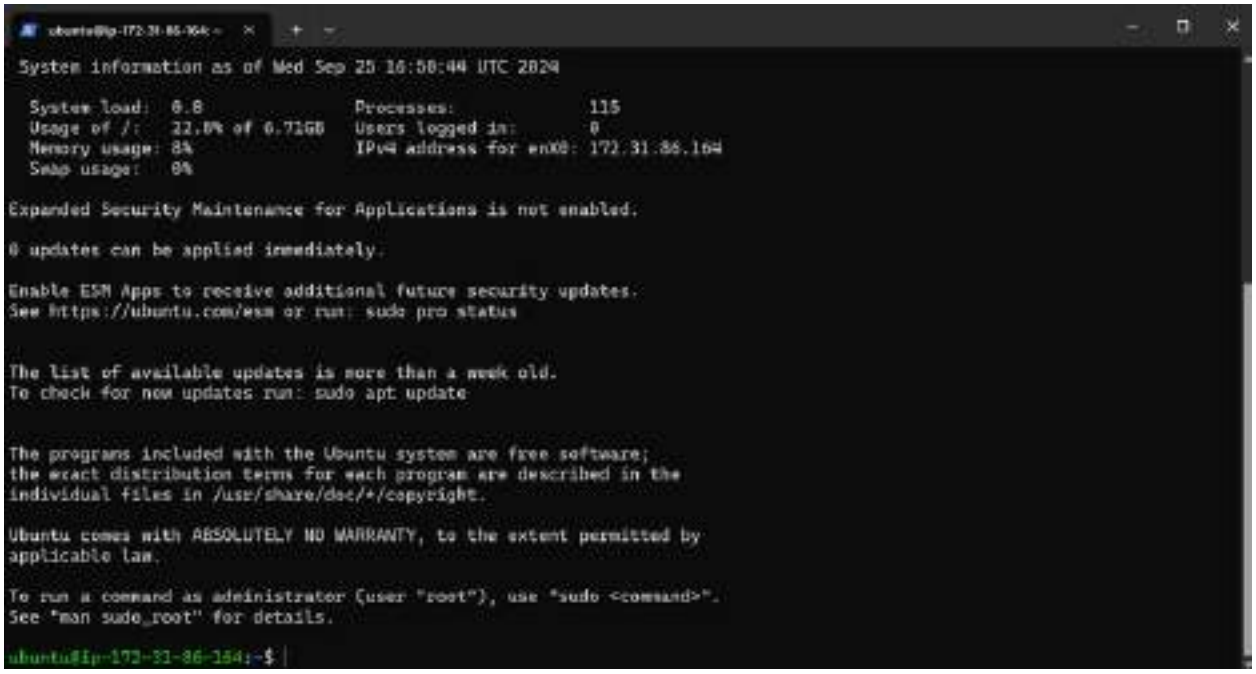

Services
Search
[Alt+S]
N. Virginia
Connect to instance
Connect to your instance i-0fd921009056d5876 (Node1) using any of these options
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Instance ID
i-0fd921009056d5876 (Node1)
1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is Node_key.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
chmod 400 "Node_key.pem"
4. Connect to your instance using its Public DNS:
ec2-54-87-91-118.compute-1.amazonaws.com
Example:
ssh -i "Node_key.pem" ubuntu@ec2-54-87-91-118.compute-1.amazonaws.com
Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.


Services
Search
[Alt+S]
N. Virginia
Connect to instance
Connect to your instance i-04e88cd275b9f8d15 (Node2) using any of these options
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Instance ID
i-04e88cd275b9f8d15 (Node2)
1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is Node_key.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
chmod 400 "Node_key.pem"
4. Connect to your instance using its Public DNS:
ec2-54-166-66-100.compute-1.amazonaws.com
Example:
ssh -i "Node_key.pem" ubuntu@ec2-54-166-66-100.compute-1.amazonaws.com
Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check

Step 4: Navigate to the folder where the .pem files for the key-pairs (master and nodes) have been downloaded and open the folder in terminal. Then paste the command copied in Step 3 in the terminal.

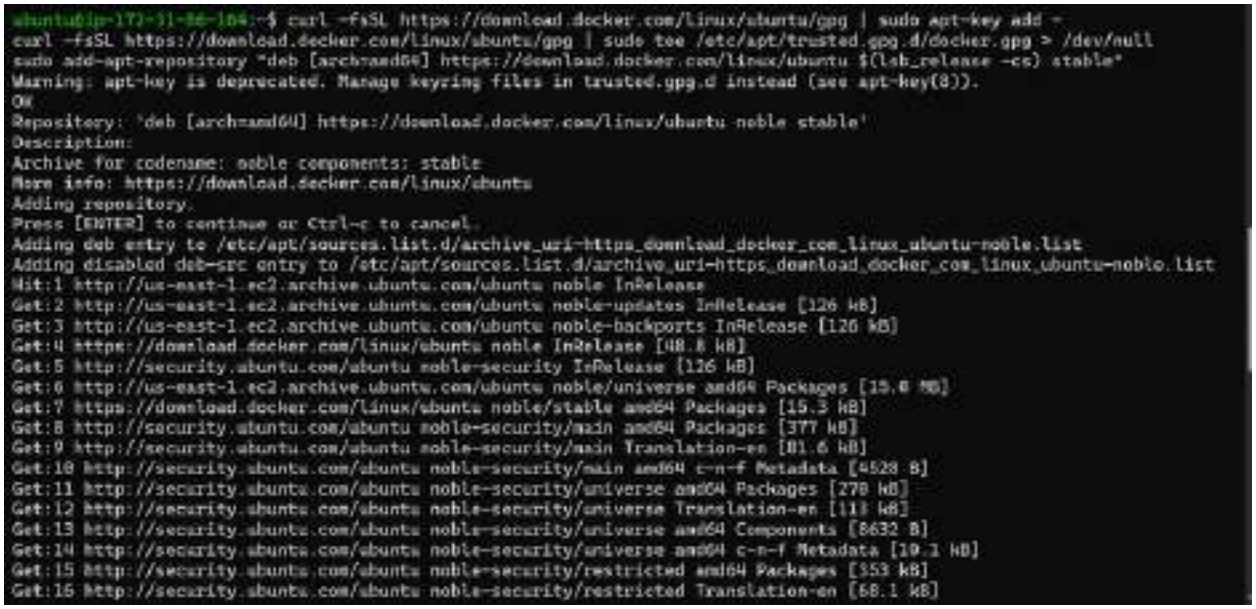






Thus, the master and 2 nodes are now connected to the SSH.

Step 5: Run the following command in Master, Node1 and Node2 to install and setup docker in all 3:-
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo tee /etc/apt/trusted.gpg.d/docker.gpg > /dev/null
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"



```
Get:36 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Packages [353 kB]
Get:37 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted Translation-en [68.1 kB]
Get:38 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 c-n-f Metadata [424 B]
Get:39 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Packages [14.4 kB]
Get:40 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse Translation-en [3668 B]
Get:41 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Components [212 B]
Get:42 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 c-n-f Metadata [532 B]
Get:43 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/main amd64 Components [208 B]
Get:44 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/main amd64 c-n-f Metadata [112 B]
Get:45 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/universe amd64 Packages [10.6 kB]
Get:46 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/universe Translation-en [10.8 kB]
Get:47 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/universe amd64 Components [17.6 kB]
Get:48 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/universe amd64 c-n-f Metadata [1104 B]
Get:49 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/restricted amd64 Components [216 B]
Get:50 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/restricted amd64 c-n-f Metadata [116 B]
Get:51 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 Components [212 B]
Get:52 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 c-n-f Metadata [116 B]
Fetched 29.1 MB in 4s (6952 kB/s)
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: The key(s) in the keyring /etc/apt/trusted.gpg.d/dock
er.gpg are ignored as the file has an unsupported filetype.
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt
/trusted.gpg), see the DEPRECATION section in apt-key(8) for details.
ubuntu@ip-172-31-86-164:~$
```

Step 6: Execute the following command:-
sudo apt-get update
sudo apt-get install -y docker-ce

```
ubuntu@ip-172-31-94-99:~$ sudo apt-get update
sudo apt-get install -y docker-ce
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 https://download.docker.com/linux/ubuntu noble InRelease
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: The key(s) in the keyring /etc/apt/trusted.gpg.d/dock
er.gpg are ignored as the file has an unsupported filetype.
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt
/trusted.gpg), see the DEPRECATION section in apt-key(8) for details.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  containerd.io docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libslirp0
  pigz slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7
  libslirp0 pigz slirp4netns
0 upgraded, 10 newly installed, 0 to remove and 142 not upgraded.
Need to get 123 MB of archives.
After this operation, 441 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 pigz amd64 2.8-1 [65.6 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 libltdl7 amd64 2.4.7-7build1 [40.3 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 libslirp0 amd64 4.7.0-1ubuntu3 [63.8 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 slirp4netns amd64 1.2.1-1build2 [34.9 kB]
```

```
Setting up docker-ce (5:27.3.1-1~ubuntu.24.04~noble) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.2) ...
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@ip-172-31-94-99:~$
```

Step 7: Execute the following command:-
sudo mkdir -p /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
EOF

```
ubuntu@ip-172-31-86-164:~$ sudo mkdir -p /etc/docker

# Use this block to create and write into the daemon.json file
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"]
}
```

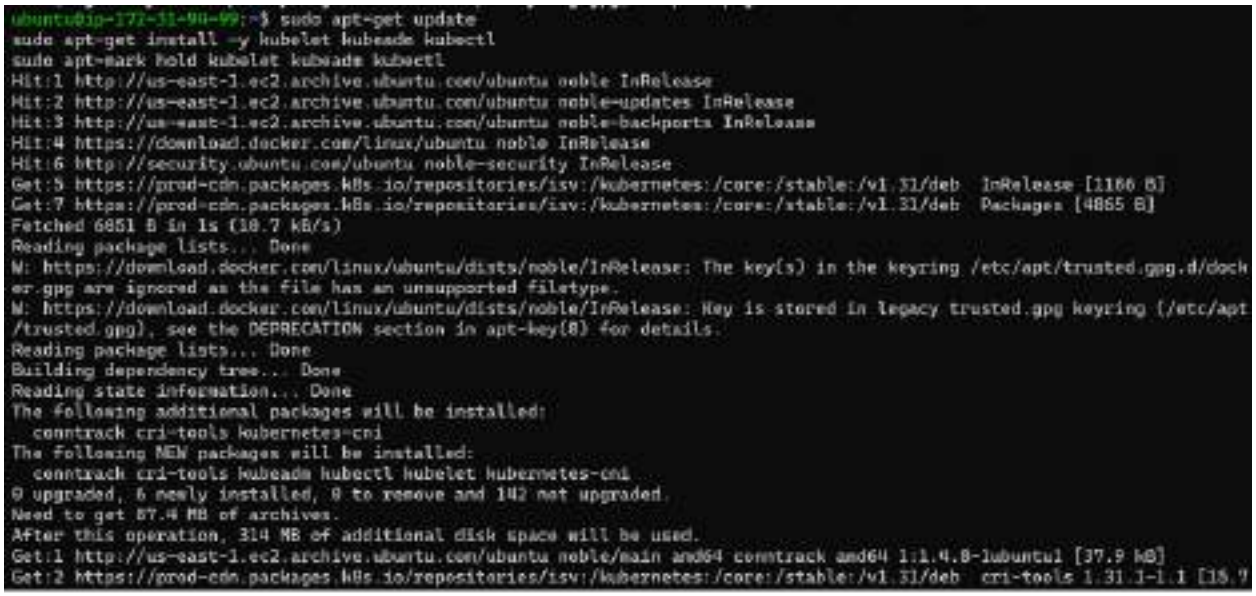
Step 8: Execute the following command:-
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker

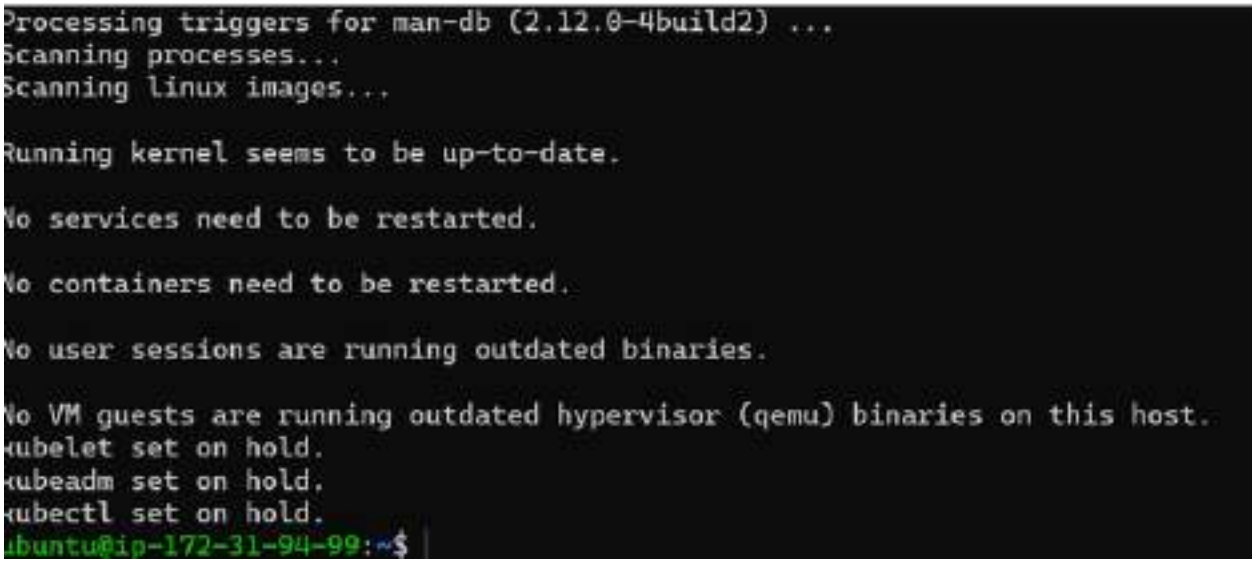
```
ubuntu@ip-172-31-90-99:~$ sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
Synchronizing state of docker.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable docker
```

Step 9: Execute the following command to install Kubernetes on all 3 machines:-
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

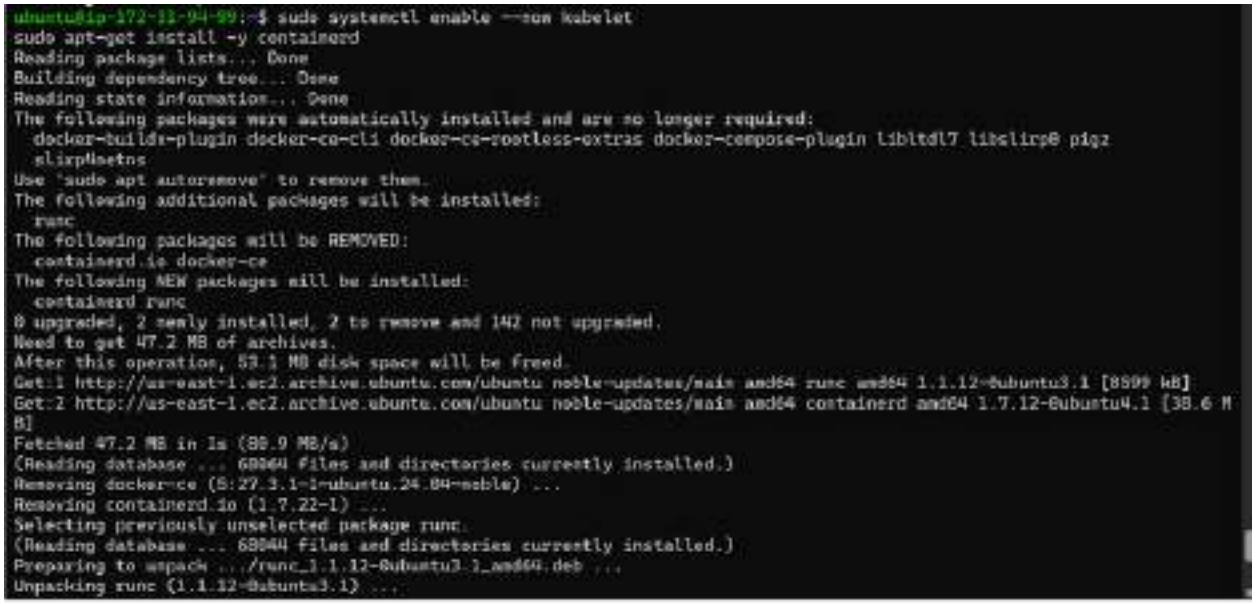
```
ubuntu@ip-172-31-87-41:~$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /et
c/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sud
o tee /etc/apt/sources.list.d/kubernetes.list
deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /
```

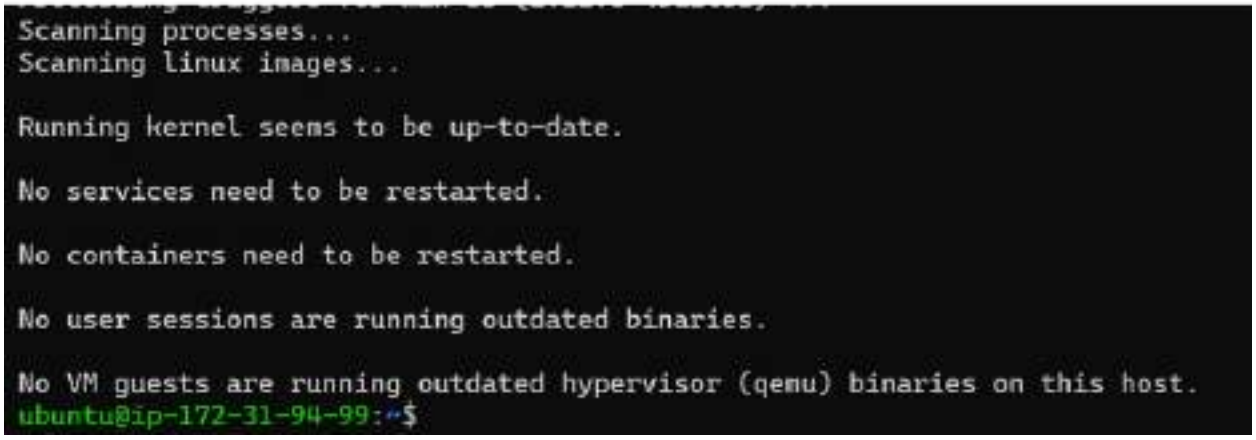
Step 10: Execute the following command:-
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl





Step 11: Execute the following command:-
sudo systemctl enable --now kubelet
sudo apt-get install -y containerd



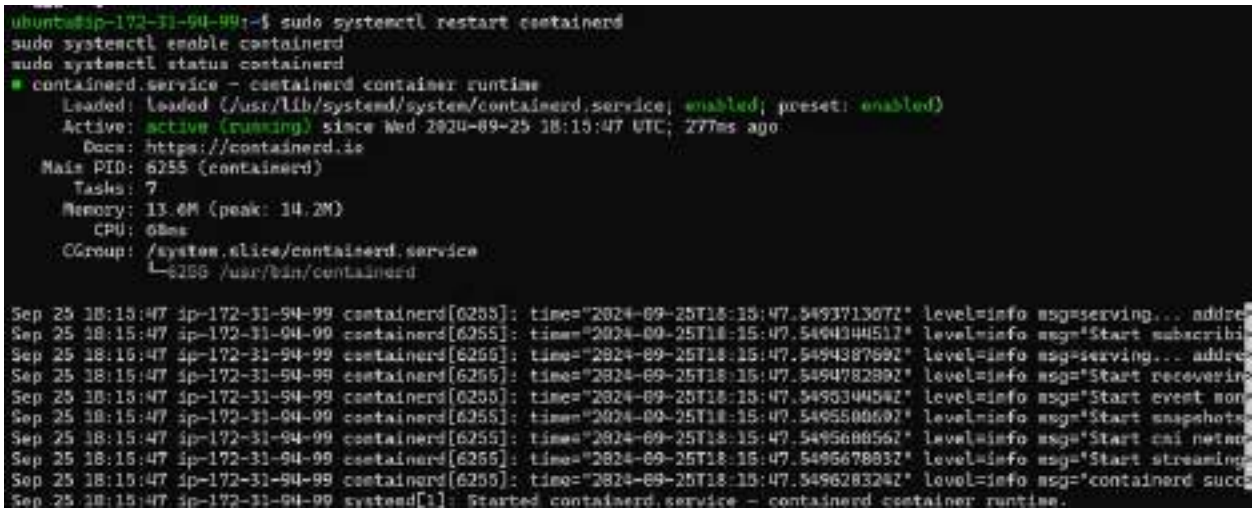


Step 12: Execute the following command:-
sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml

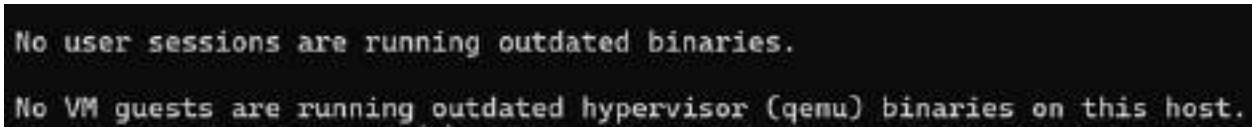
```
ubuntu@ip-172-31-94-99:~$ sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml
disabled_plugins = []
imports = []
oom_score = 0
plugin_dir = ""
required_plugins = []
root = "/var/lib/containerd"
state = "/run/containerd"
temp = ""
version = 2

[cgroup]
  path = ""

[debug]
  address = ""
  format = ""
  gid = 0
  level = ""
  uid = 0

[grpc]
  address = "/run/containerd/containerd.sock"
  gid = 0
  max_recv_message_size = 16777216
  max_send_message_size = 16777216
  tcp_address = ""
  tcp_tls_ca = ""
```

```
  [stream_processors."io.containerd.ocicrypt.decoder.v1.tar.gzip"]
    accepts = ["application/vnd.oci.image.layer.v1.tar+gzip+encrypted"]
    args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]
    env = ["OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider.conf"]
    path = "ctd-decoder"
    returns = "application/vnd.oci.image.layer.v1.tar+gzip"

[timeouts]
  "io.containerd.timeout.bolt.open" = "0s"
  "io.containerd.timeout.metrics.shimstats" = "2s"
  "io.containerd.timeout.shim.cleanup" = "5s"
  "io.containerd.timeout.shim.load" = "5s"
  "io.containerd.timeout.shim.shutdown" = "3s"
  "io.containerd.timeout.task.state" = "2s"

[ttrpc]
  address = ""
  gid = 0
  uid = 0
ubuntu@ip-172-31-94-99:~$
```

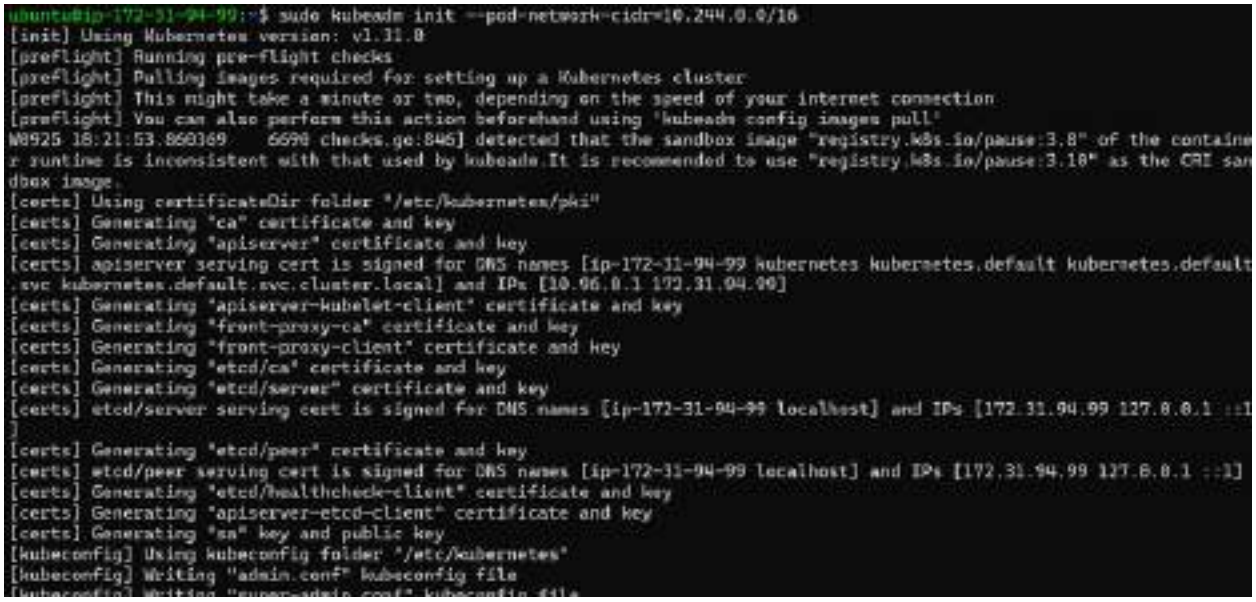
Step 13: Execute the following command:-
sudo systemctl restart containerd
sudo systemctl enable containerd
sudo systemctl status containerd



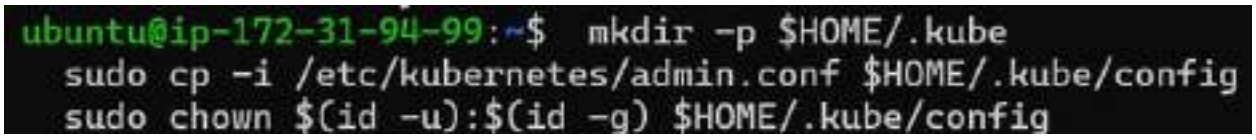
Step 14: Execute the following command:-
sudo apt-get install -y socat



Step 15: Now, we must initialize the kubernetes cluster. To do so, run the following command on the Master machine only:-
sudo kubeadm init --pod-network-cidr=10.244.0.0/16



Step 16: Copy the following part of the output of Step 15 and run it in the terminal:-
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
Also, copy the join command from the output of Step 15 for future steps.



Step 17: Check if the kubernetes cluster has been initialized correctly using the ‘kubectl get nodes’ command.

```
ubuntu@ip-172-31-94-99:~$ kubectl get nodes
NAME              STATUS     ROLES           AGE     VERSION
ip-172-31-94-99   NotReady   control-plane   2m53s   v1.31.1
```

We see that the cluster has been initialized without any issues and for now, only the Master machine is a part of the cluster.

Step 18: Run the join command that was previously copied only in the Node machines. This allows the Node machines to join the kubernetes cluster as well.

- Node1:-

```
ubuntu@ip-172-31-87-41:~$ sudo kubeadm join 172.31.94.99:6443 --token tc2id6.v5nr4lyvt5pgi3or \
    --discovery-token-ca-cert-hash sha256:5ef2566eele34d82e039485e51aaa179f58639b1d4fef2cb131f55be51826469
[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 at http://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.001283532s
[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.
```

- Node2:-

```
ubuntu@ip-172-31-86-164:~$ sudo kubeadm join 172.31.94.99:6443 --token tc2id6.v5nr4lyvt5pgi3or \
    --discovery-token-ca-cert-hash sha256:5ef2566eele34d82e039485e51aaa179f58639b1d4fef2cb131f55be51826469
[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 at http://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 508.72345ms
[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.
```

Step 19: Run the ‘kubectl get nodes’ command on the Master machine again to check if the Node machines have successfully joined the cluster.

```
ubuntu@ip-172-31-94-99:~$ kubectl get nodes
NAME              STATUS     ROLES           AGE     VERSION
ip-172-31-86-164  NotReady   <none>          34s     v1.31.1
ip-172-31-87-41   NotReady   <none>          73s     v1.31.1
ip-172-31-94-99   NotReady   control-plane   6m17s   v1.31.1
```

But, it is observed that the status of all the nodes is ‘NotReady’.

Step 20: We must add a network plugin to change the status of the nodes to 'Ready':-
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

```
ubuntu@ip-172-31-94-99:~$ kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
```

Step 21: Execute the following command:-
sudo systemctl status kubelet

```
ubuntu@ip-172-31-94-99:~$ sudo systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; preset: enabled)
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Wed 2024-09-25 18:22:14 UTC; 8min ago
       Docs: https://kubernetes.io/docs/
   Main PID: 7366 (kubelet)
      Tasks: 10 (limit: 4676)
     Memory: 33.6M (peak: 34.1M)
        CPU: 8.441s
     CGroup: /system.slice/kubelet.service
             └─7366 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/ku

Sep 25 18:30:29 ip-172-31-94-99 kubelet[7366]: I0925 18:30:29.856207    7366 pod_container_deletor.go:80] "Container no
Sep 25 18:30:29 ip-172-31-94-99 kubelet[7366]: I0925 18:30:29.856234    7366 scope.go:117] "RemoveContainer" containerI
Sep 25 18:30:29 ip-172-31-94-99 kubelet[7366]: I0925 18:30:29.857797    7366 scope.go:117] "RemoveContainer" containerI
Sep 25 18:30:29 ip-172-31-94-99 kubelet[7366]: I0925 18:30:29.869348    7366 scope.go:117] "RemoveContainer" containerI
Sep 25 18:30:29 ip-172-31-94-99 kubelet[7366]: I0925 18:30:29.876450    7366 scope.go:117] "RemoveContainer" containerI
Sep 25 18:30:33 ip-172-31-94-99 kubelet[7366]: I0925 18:30:33.872760    7366 scope.go:117] "RemoveContainer" containerI
Sep 25 18:30:33 ip-172-31-94-99 kubelet[7366]: E0925 18:30:33.872874    7366 pod_workers.go:1301] "Error syncing pod, s
Sep 25 18:30:39 ip-172-31-94-99 kubelet[7366]: I0925 18:30:39.167638    7366 scope.go:117] "RemoveContainer" containerI
Sep 25 18:30:39 ip-172-31-94-99 kubelet[7366]: E0925 18:30:39.167749    7366 pod_workers.go:1301] "Error syncing pod, s
Sep 25 18:30:39 ip-172-31-94-99 kubelet[7366]: I0925 18:30:39.882133    7366 scope.go:117] "RemoveContainer" containerI
ubuntu@ip-172-31-94-99:~$
```

Step 22: Run the 'kubectl get nodes' to check if the status of the nodes have been successfully updated or not.

```
ubuntu@ip-172-31-94-99:~$ kubectl get nodes
NAME               STATUS   ROLES           AGE     VERSION
ip-172-31-86-164   Ready    <none>          3m37s   v1.31.1
ip-172-31-87-41    Ready    <none>          4m16s   v1.31.1
ip-172-31-94-99    Ready    control-plane   9m20s   v1.31.1
```

Step 23: To rename the nodes to their actual names instead of their IP address, run the following on the Master machine:-
kubectl label node ip-172-31-87-41 kubernetes.io/role=Node1
kubectl label node ip-172-31-86-164 kubernetes.io/role=Node2

```
ubuntu@ip-172-31-94-99:~$ kubectl label node ip-172-31-87-41 kubernetes.io/role=Node1
node/ip-172-31-87-41 labeled
ubuntu@ip-172-31-94-99:~$ kubectl label node ip-172-31-86-164 kubernetes.io/role=Node2
node/ip-172-31-86-164 labeled
```

Step 24: Run the 'kubectl get nodes' to check if the 'Roles' of the nodes have been successfully updated or not.

```
ubuntu@ip-172-31-94-99:~$ kubectl get nodes
NAME               STATUS   ROLES           AGE     VERSION
ip-172-31-86-164   Ready    Node2           7m58s   v1.31.1
ip-172-31-87-41    Ready    Node1           8m37s   v1.31.1
ip-172-31-94-99    Ready    control-plane   13m     v1.31.1
```

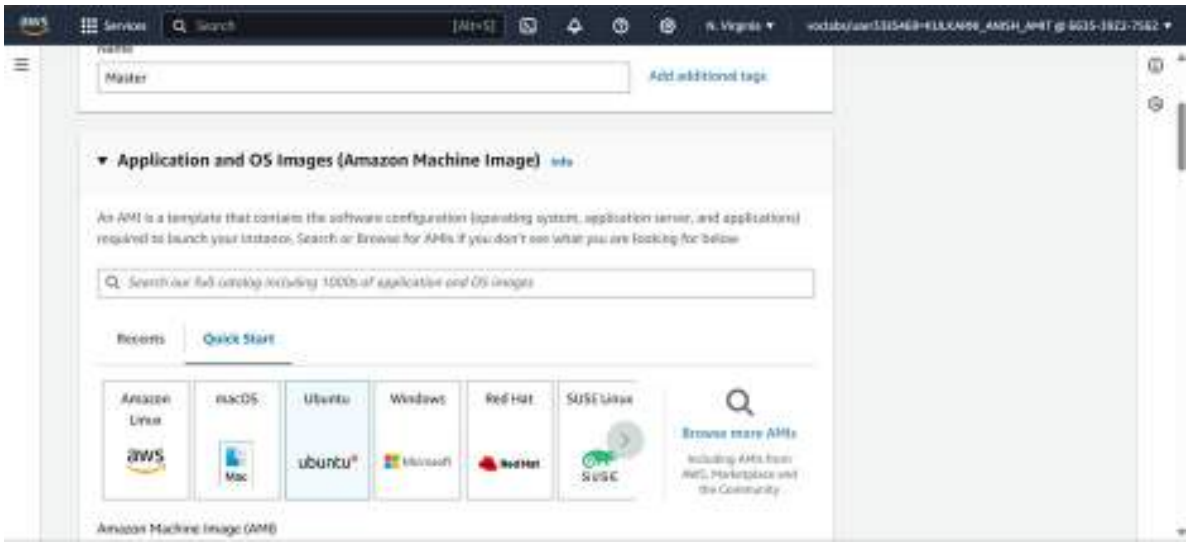
Conclusion: In this experiment, we learned about the Kubernetes cluster architecture and how to install and spin up a Kubernetes cluster on Linux Machines/Cloud Platforms. First, we created 3 EC2 Ubuntu instances on AWS (1 master and 2 worker nodes) and connected them to our local terminal using SSH. Then, we installed and configured docker and kubernetes on all machines and added the master node to a kubernetes cluster. Then, we used the 'join' command to add the worker nodes to the cluster as well. At the end, we have a kubernetes cluster with all 3 machines with 'Ready' status.

Experiment 4

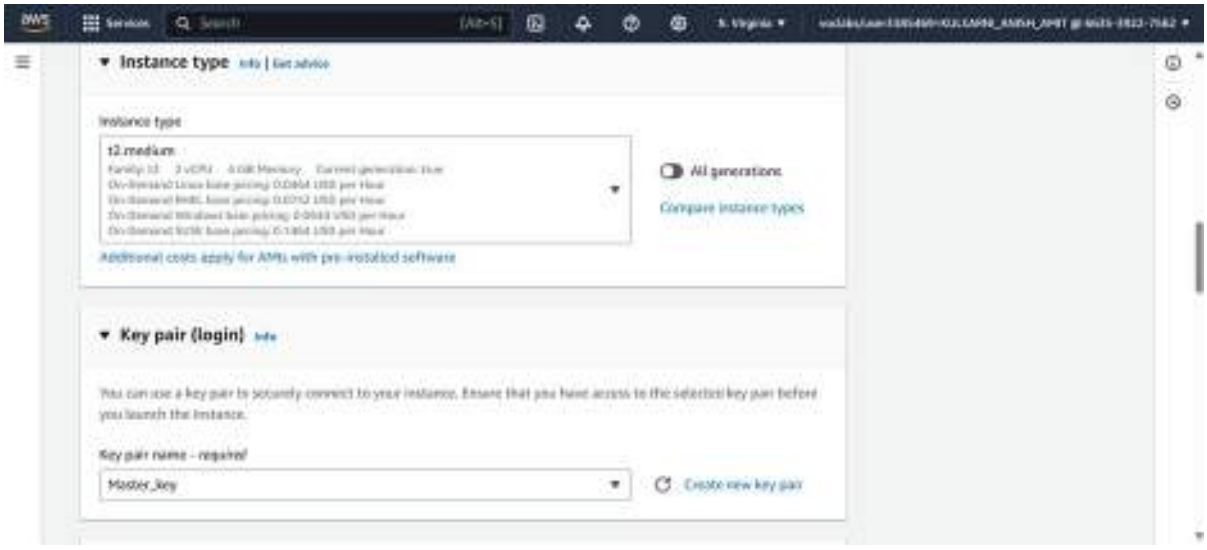
Aim:ToinstallKubectlandexecuteKubectlcommandstomanagetheKubernetes cluster and deploy Your First Kubernetes Application.

Steps:

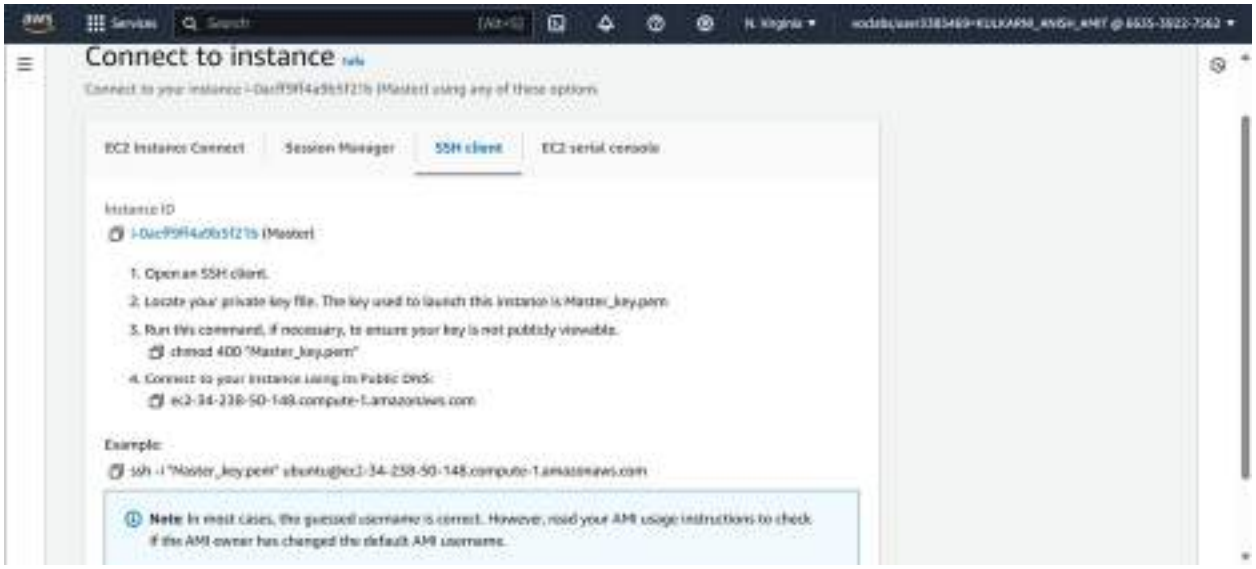
Step 1: Create a new EC2 instance in AWS. Make sure that you choose Ubuntu as the instance type.



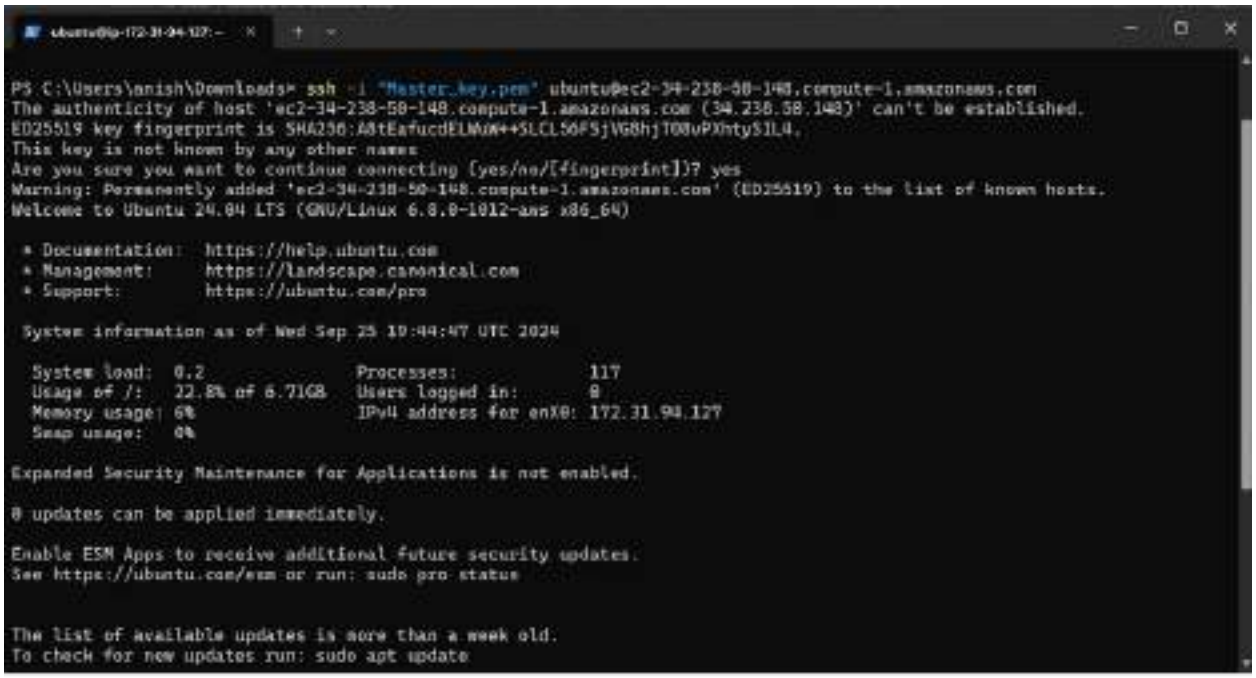
Under ‘Instance type’, choose t2.medium because the default (t2.micro) does not provide the sufficient resources required to execute this experiment.

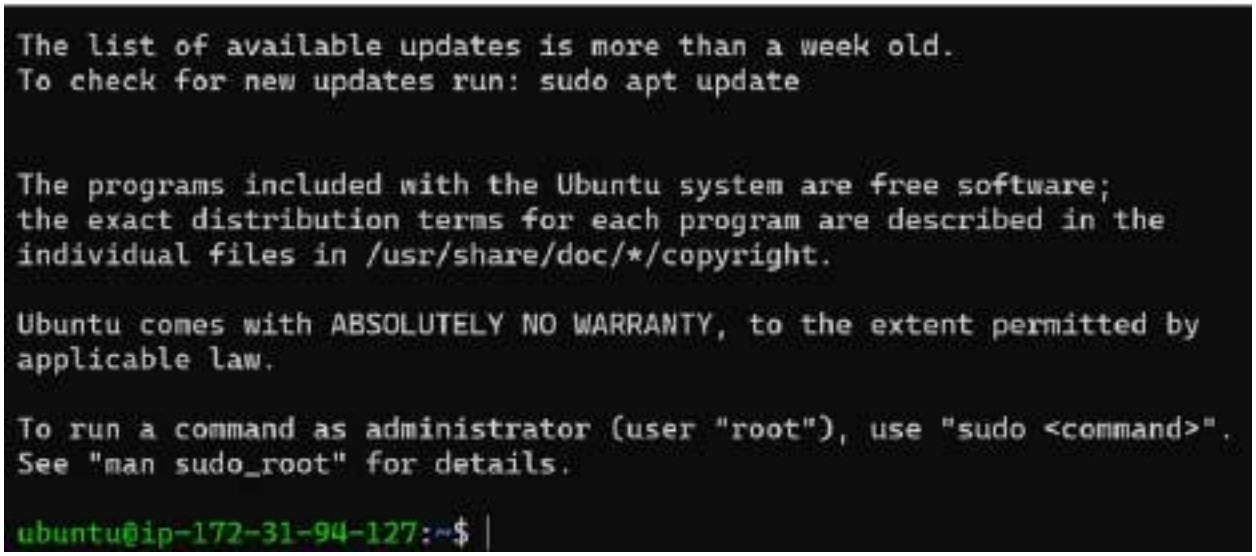


Step 2: Now, we must connect the instance to a SSH. To do so, click on the instance and click on ‘Connect’. Next, navigate to the ‘SSH client’ section and copy the command under the ‘Example’ section.

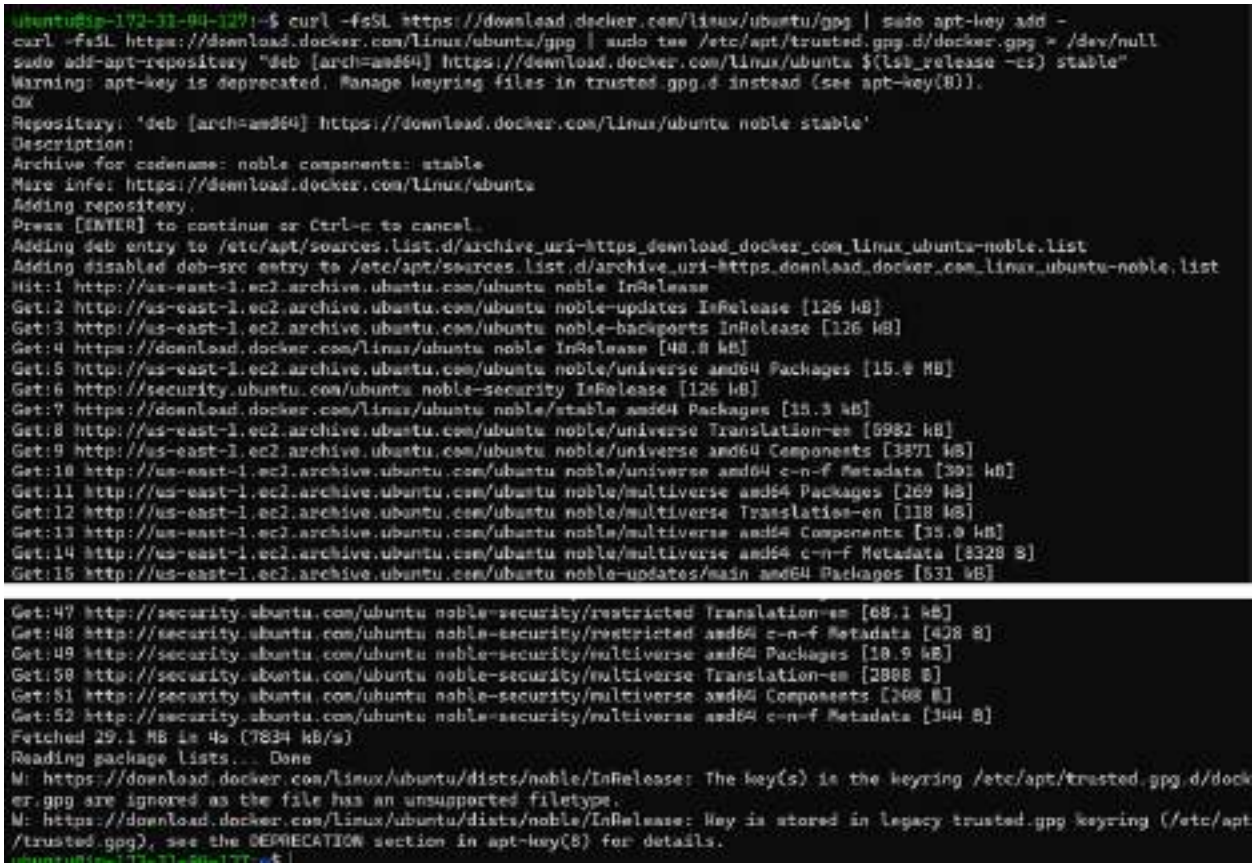


Step 3: Navigate to the folder where the .pem file for the Master key-pair has been downloaded and open the folder in terminal. Then paste the command copied in Step 2 in the terminal.

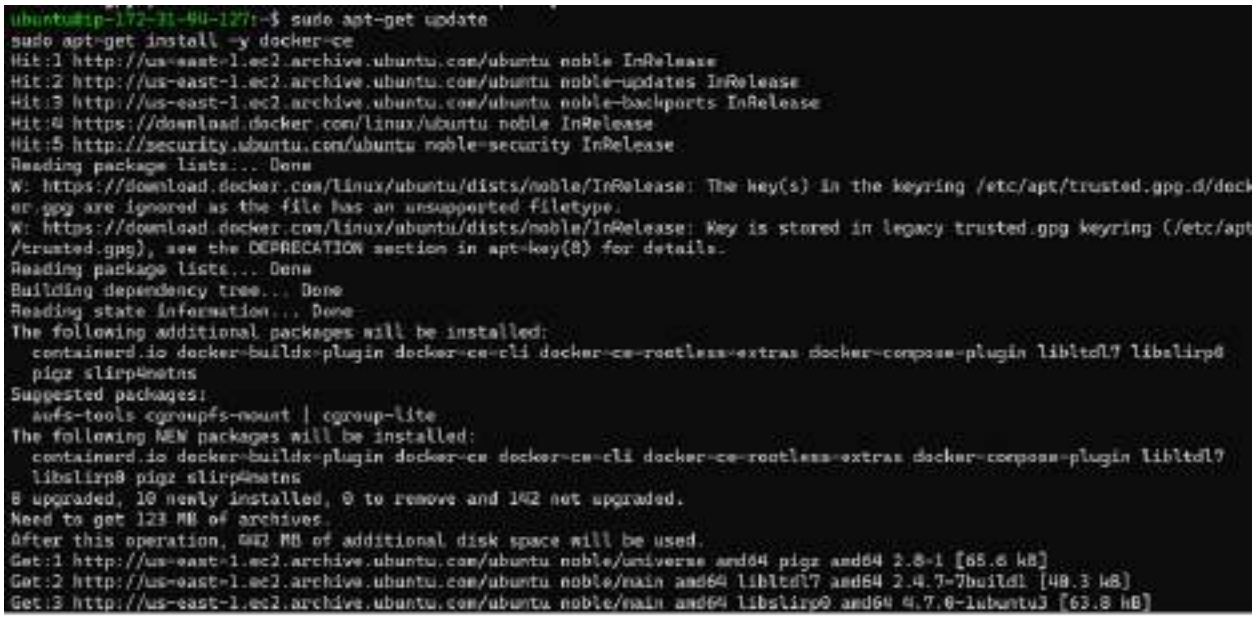


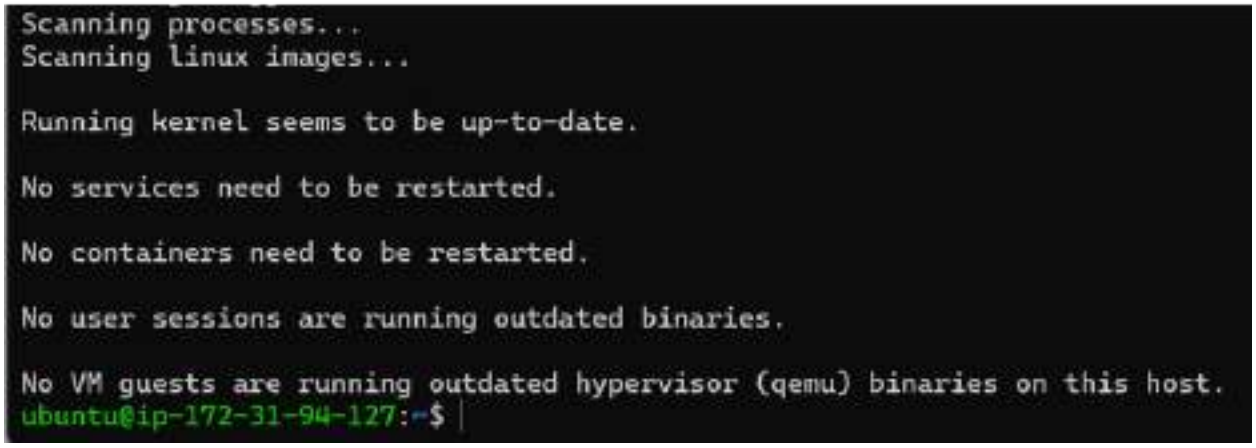


Step 4: Run the following command to install and setup docker:-
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo tee /etc/apt/trusted.gpg.d/docker.gpg > /dev/null
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

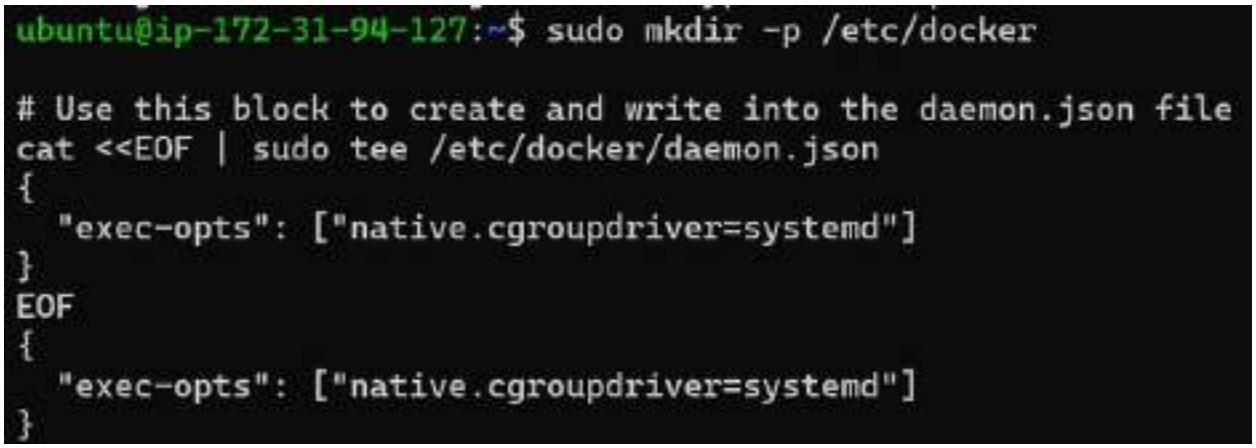


Step 5: Execute the following command:-
sudo apt-get update
sudo apt-get install -y docker-ce





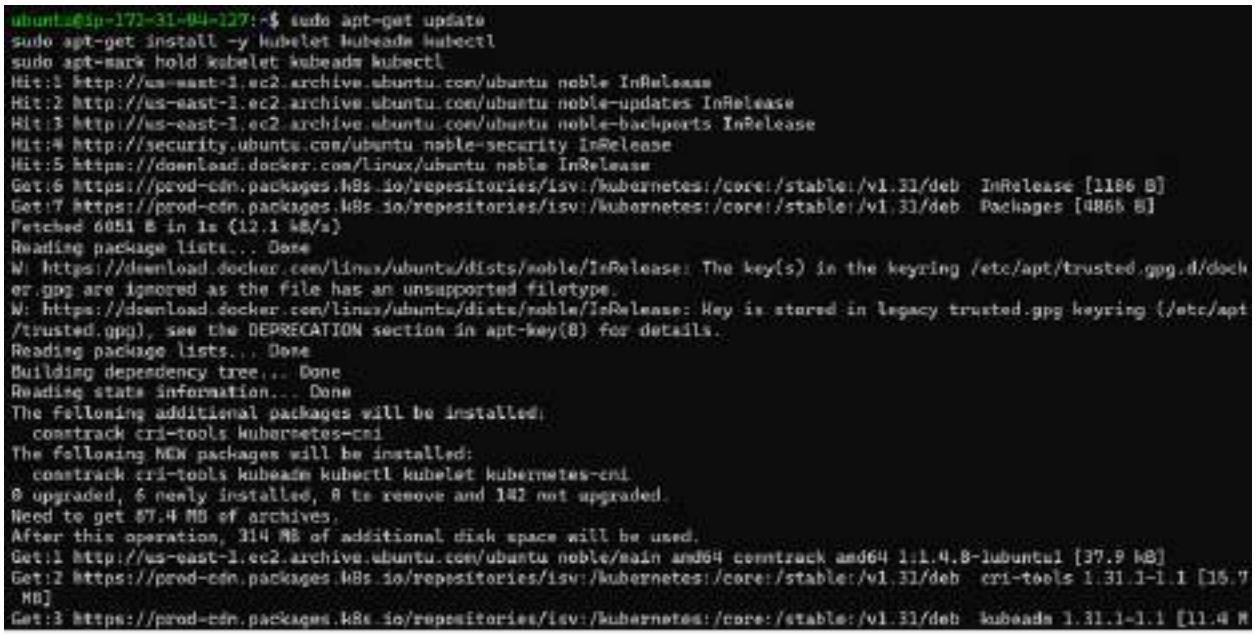
Step 6: Execute the following command:-
sudo mkdir -p /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
EOF

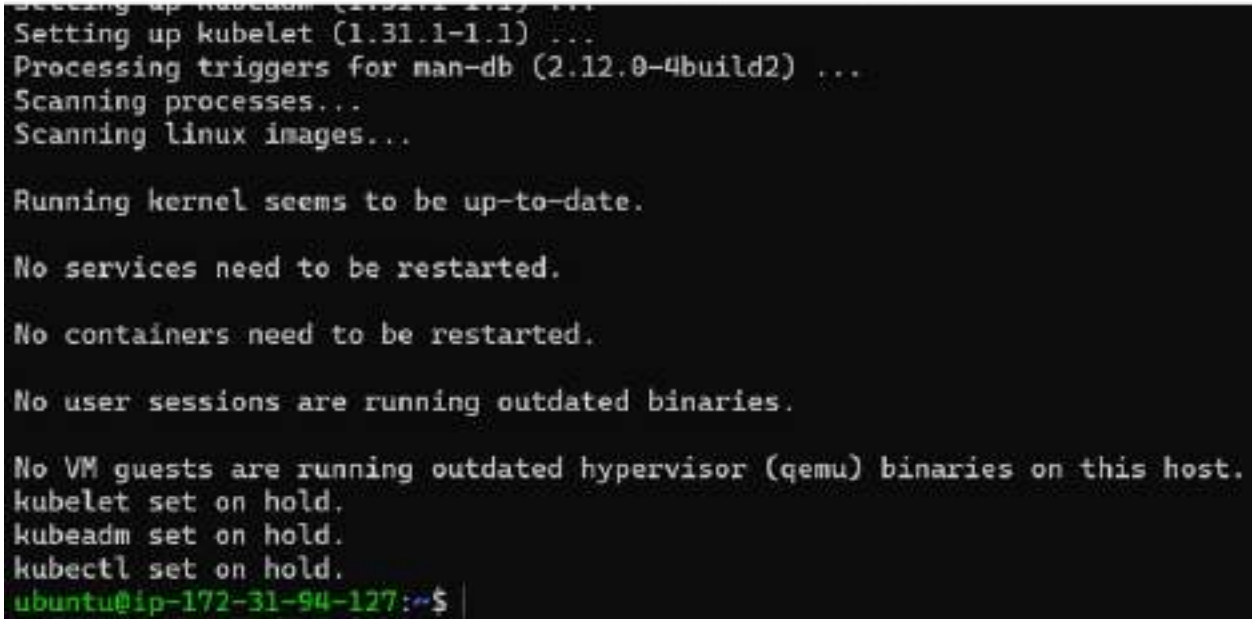


Step 7: Execute the following command:-
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker

Step 8: Run the following command to install and setup kubernetes:-
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

Step 9: Execute the following command:-
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl

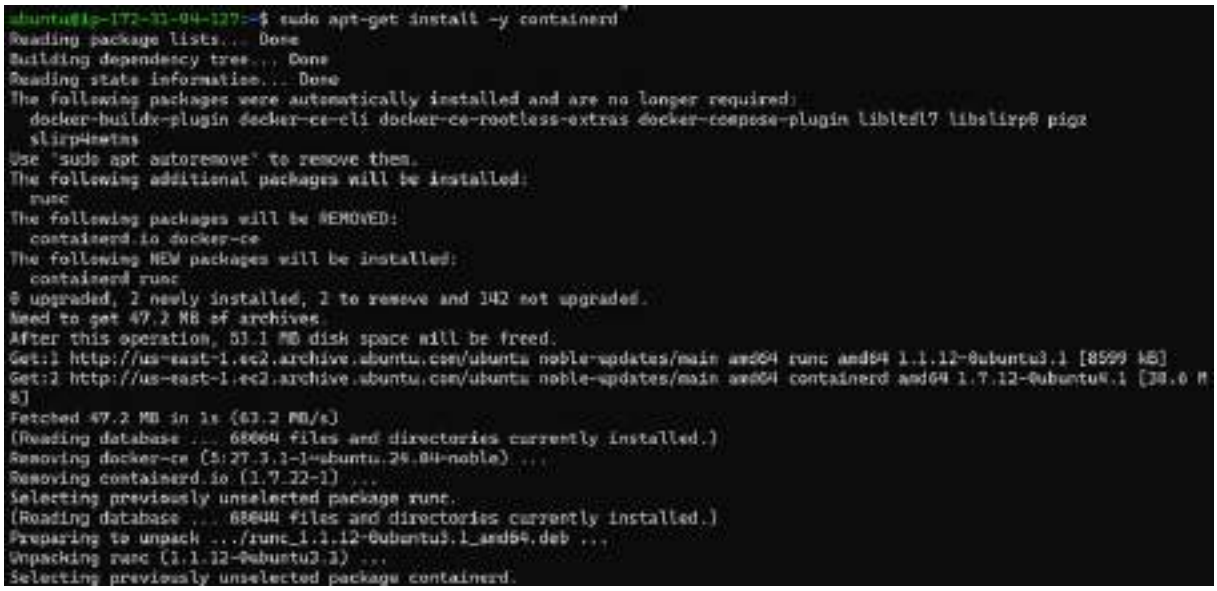


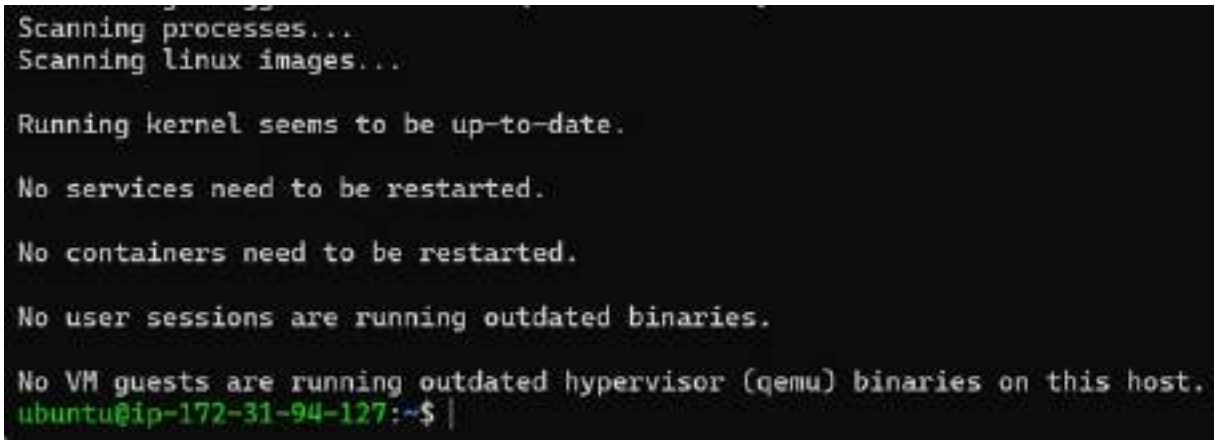


Step 10: Execute the following command:-
sudo systemctl enable --now kubelet
sudo kubeadm init --pod-network-cidr=10.244.0.0/16

But, we encounter the above error on running the commands. Thus, a few more commands need to be run in order to solve the error.

Step 11: Execute the following command:-
sudo apt-get install -y containerd



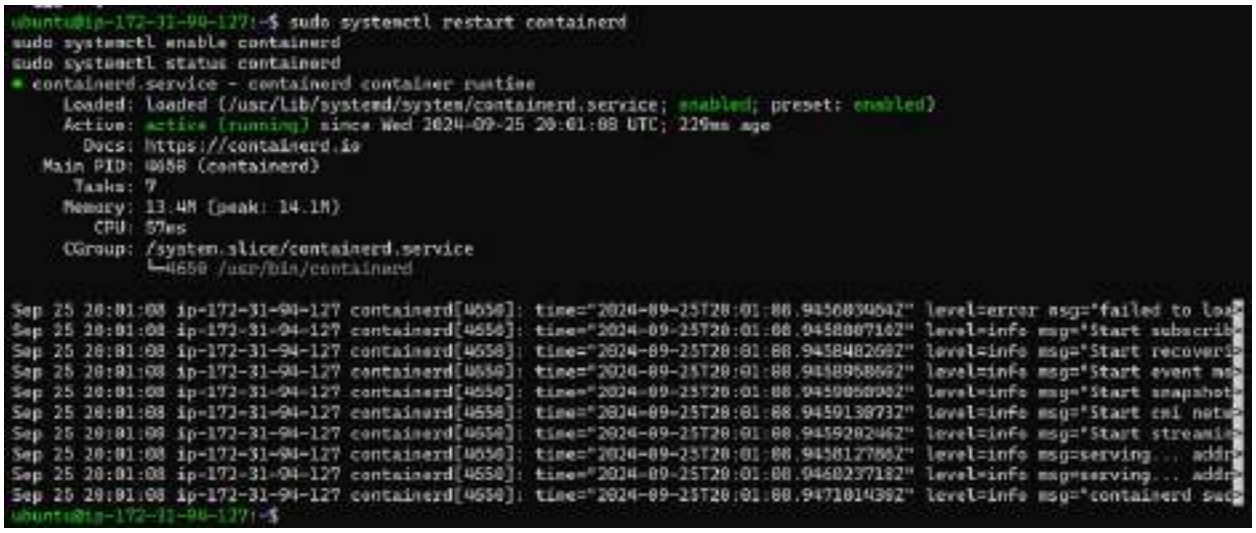


Step 12: Execute the following command:-
sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml

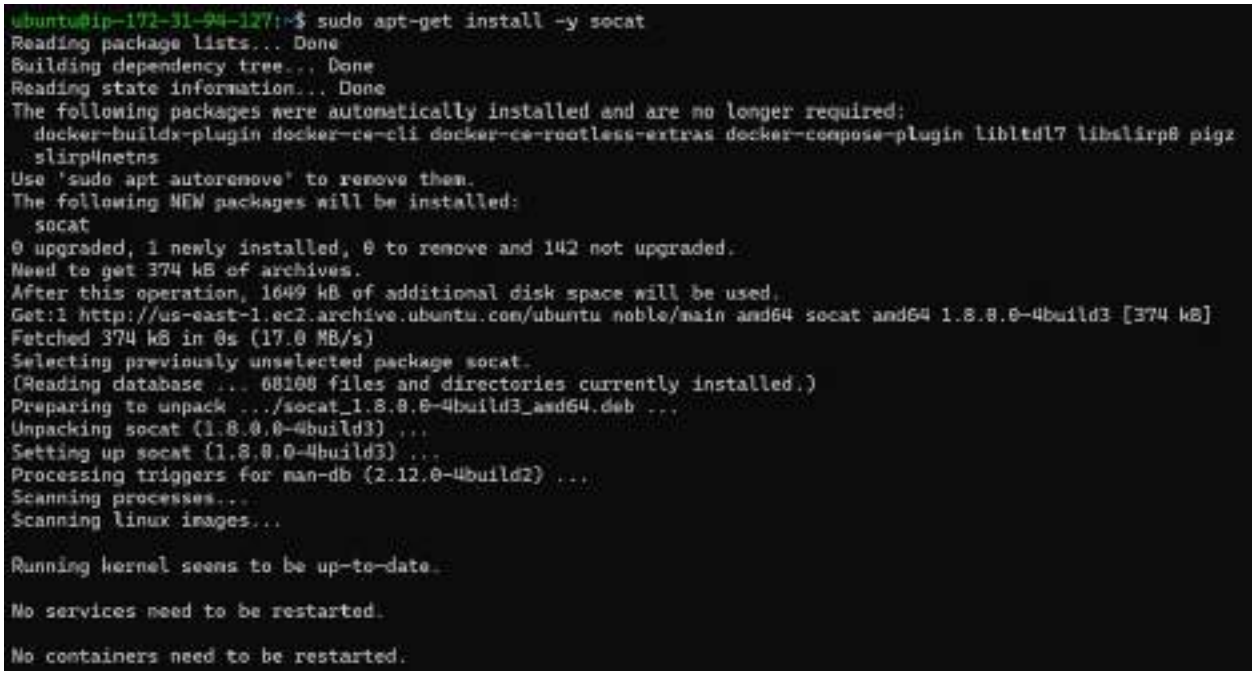
```
ubuntu@ip-172-31-94-127:~$ sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml
disabled_plugins = []
imports = []
oom_score = 0
plugin_dir = ""
required_plugins = []
root = "/var/lib/containerd"
state = "/run/containerd"
temp = ""
version = 2

[cgroup]
  path = ""

[debug]
  address = ""
  format = ""
  gid = 0
  level = ""
  uid = 0

[grpc]
  address = "/run/containerd/containerd.sock"
  gid = 0
  max_recv_message_size = 16777216
  max_send_message_size = 16777216
  tcp_address = ""
  tcp_tls_ca = ""
  tcp_tls_cert = ""
```

```
  [stream_processors."io.containerd.ocicrypt.decoder.v1.tar.gzip"]
    accepts = ["application/vnd.oci.image.layer.v1.tar+gzip+encrypted"]
    args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]
    env = ["OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider.conf"]
    path = "ctd-decoder"
    returns = "application/vnd.oci.image.layer.v1.tar+gzip"

[timeouts]
  "io.containerd.timeout.bolt.open" = "0s"
  "io.containerd.timeout.metrics.shimstats" = "2s"
  "io.containerd.timeout.shim.cleanup" = "5s"
  "io.containerd.timeout.shim.load" = "5s"
  "io.containerd.timeout.shim.shutdown" = "3s"
  "io.containerd.timeout.task.state" = "2s"

[ttrpc]
  address = ""
  gid = 0
  uid = 0
ubuntu@ip-172-31-94-127:~$
```

Step 13: Execute the following command:-
sudo systemctl restart containerd
sudo systemctl enable containerd
sudo systemctl status containerd



Step 14: Execute the following command:-
sudo apt-get install -y socat



Now, the error that was encountered in Step 10 should be fixed.

Step 15: Initialise the kubernetes cluster. To do so, execute the following command:-
sudo kubeadm init --pod-network-cidr=10.244.0.0/16

```
ubuntu@ip-172-31-94-127:~$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W0925 20:04:28.033570    4946 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of the containe
r runtime is inconsistent with that used by kubeadm.It is recommended to use "registry.k8s.io/pause:3.10" as the CRI san
dbox image.
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-172-31-94-127 kubernetes kubernetes.default kubernetes.defaul
t.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.94.127]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [ip-172-31-94-127 localhost] and IPs [172.31.94.127 127.0.0.1 :
:1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-94-127 localhost] and IPs [172.31.94.127 127.0.0.1 ::1
]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "super-admin.conf" kubeconfig file
```

```
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Boots
trap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

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

kubeadm join 172.31.94.127:6443 --token lu1lmh.9tuddvu59w0bzzrv4 \
        --discovery-token-ca-cert-hash sha256:82da7c027529ee859fa90156f1005921683bc3f1ac6501184759c6da8227ff1
ubuntu@ip-172-31-94-127:~$
```

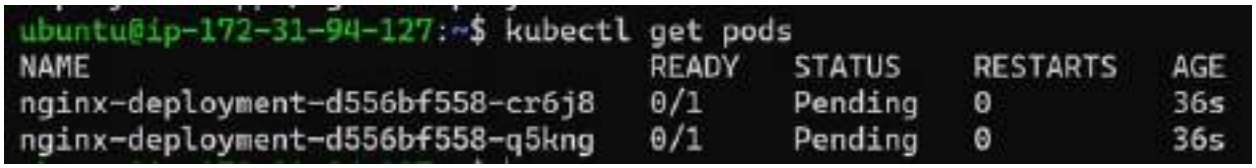
Step 16: Copy the following part of the output of Step 15 and run it in the terminal:-
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
Also, copy the join command from the output of Step 15 for future steps.

```
ubuntu@ip-172-31-94-127:~$   mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@ip-172-31-94-127:~$
```

Step 17: Add a common networking plugin called Flannel to the kubernetes cluster:-
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

Step 18: The kubernetes cluster has been created and initialized. Now, we can deploy our nginx server on the cluster. To apply this deployment file, use the following command to create a deployment:-
kubectl apply -f https://k8s.io/examples/application/deployment.yaml

Step 19: Check the status of the kubernetes cluster using 'kubectl get pods' command.



Step 20: Now, to access the kubernetes pod running the nginx application, run the following command:-
POD_NAME=$(kubectl get pods -l app=nginx -o jsonpath="{.items[0].metadata.name}")
kubectl port-forward $POD_NAME 8080:80

But, we see that an error is encountered because the status of our pod is 'Pending' and not 'Running'. To do so, we must untaint the tainted nodes.

Step 21: Execute the following command:-
kubectl taint nodes --all node-role.kubernetes.io/control-plane-

This changes the status of the pod to 'Running'.

Step 22: Run 'kubectl get nodes' to check on the status of the kubernetes cluster.

Step 23: Run the ‘kubectl get pods’ command again to check if the status of our pod has been updated to ‘Running’ or not.

```
ubuntu@ip-172-31-94-127:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d556bf558-cr6j8    1/1     Running   0          6m50s
nginx-deployment-d556bf558-q5kng    1/1     Running   0          6m50s
```

Step 24: Execute the following command again:-
POD_NAME=$(kubectl get pods -l app=nginx -o jsonpath="{.items[0].metadata.name}")
kubectl port-forward $POD_NAME 8080:80

```
ubuntu@ip-172-31-94-127:~$ POD_NAME=$(kubectl get pods -l app=nginx -o jsonpath="{.items[0].metadata.name}")
kubectl port-forward $POD_NAME 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
Handling connection for 8080
```

Step 25: To verify the deployment, open a new terminal (SSH) for your EC2 instance.

```
PS C:\Users\anish\Downloads> ssh -i "Master_key.pem" ubuntu@ec2-34-238-60-140.compute-1.amazonaws.com
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-1012-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Wed Sep 25 20:22:30 UTC 2024

  System load:  0.0               Processes:             154
  Usage of /:   55.4% of 6.71GB   Users logged in:       1
  Memory usage: 19%               IPv4 address for enX0: 172.31.94.127
  Swap usage:   0%


Expanded Security Maintenance for Applications is not enabled.

143 updates can be applied immediately.
41 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Wed Sep 25 19:44:48 2024 from 182.48.210.206
```

Step 26: Run the following command to check if the nginx server is running or not:-

```
ubuntu@ip-172-31-94-127:~$ curl --head http://127.0.0.1:8080
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Wed, 25 Sep 2024 20:23:09 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 04 Dec 2018 14:44:49 GMT
Connection: keep-alive
ETag: "5c0692e1-264"
Accept-Ranges: bytes
```

As the response says ‘200 OK’ along with the server name, we can conclude that we successfully deployed our nginx server on our EC2 instance.

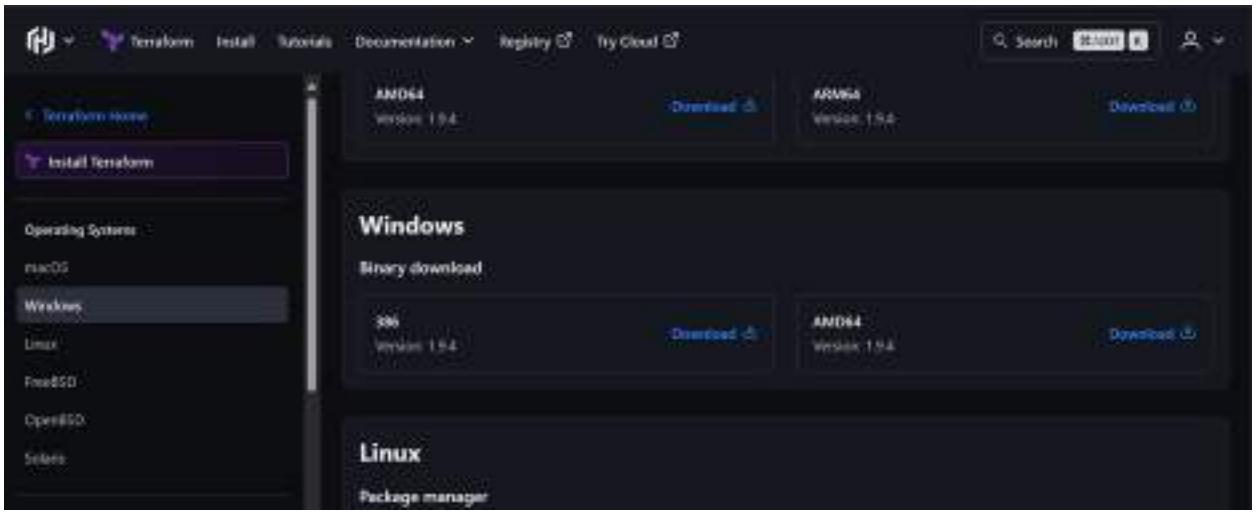
Conclusion: In this experiment, we learned how to install kubectl and execute kubectl commands to manage the Kubernetes cluster and deploy our first Kubernetes application. First, we created an EC2 Ubuntu instance on AWS and connected it to our local terminal using SSH. Then, we installed and configured Docker and Kubernetes on the machine and added the machine to the kubernetes cluster. Next, we deployed the Flannel networking plugin and nginx server on our cluster and forwarded port 8080 on our local machine to port 80 on the specified pod to deploy and run the nginx server on our EC2 instance.

Experiment No. 5

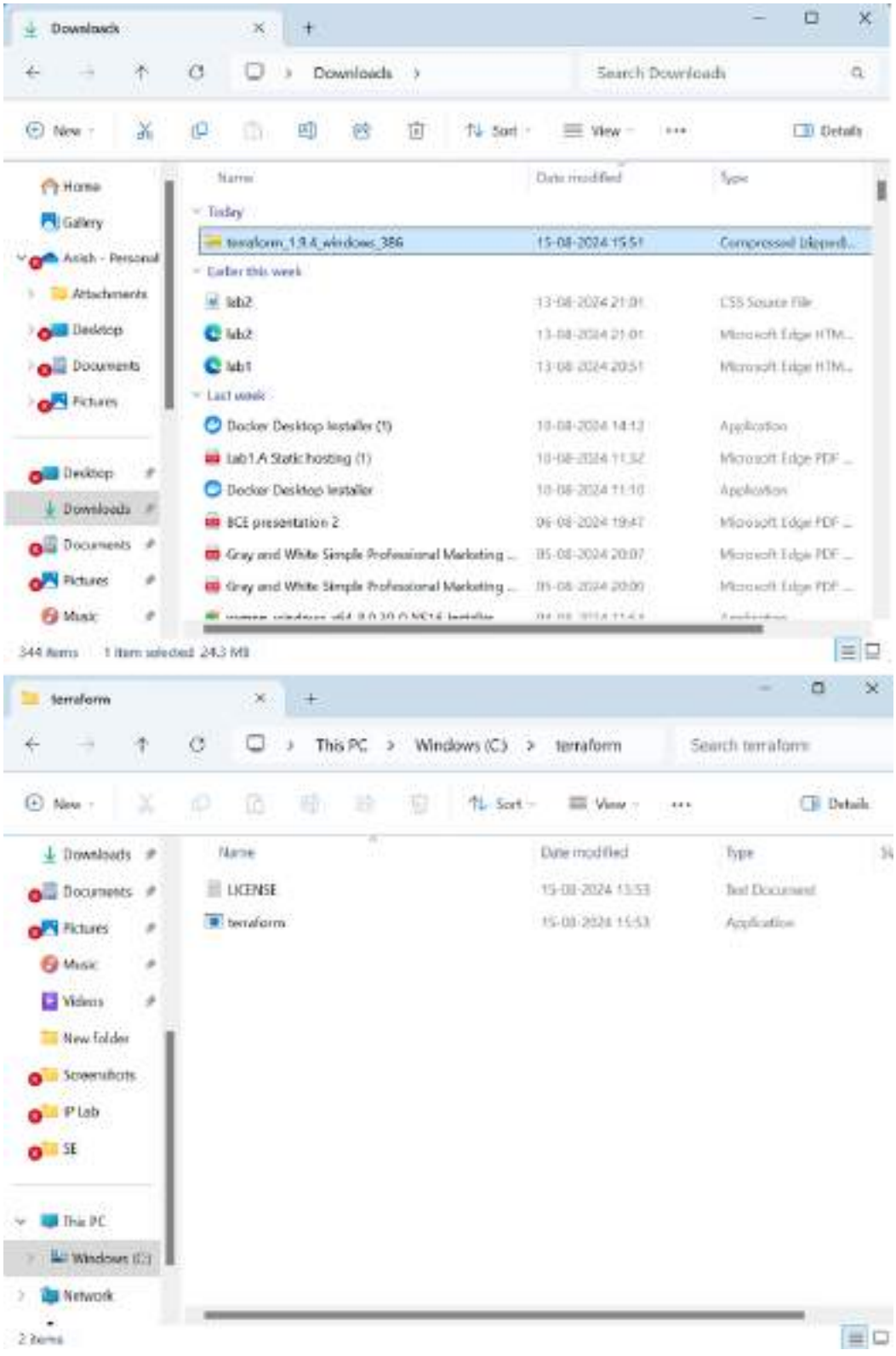
Aim: To understand Terraform lifecycle, core concepts/terminologies and install it on a Linux machine and Windows.

Steps:-

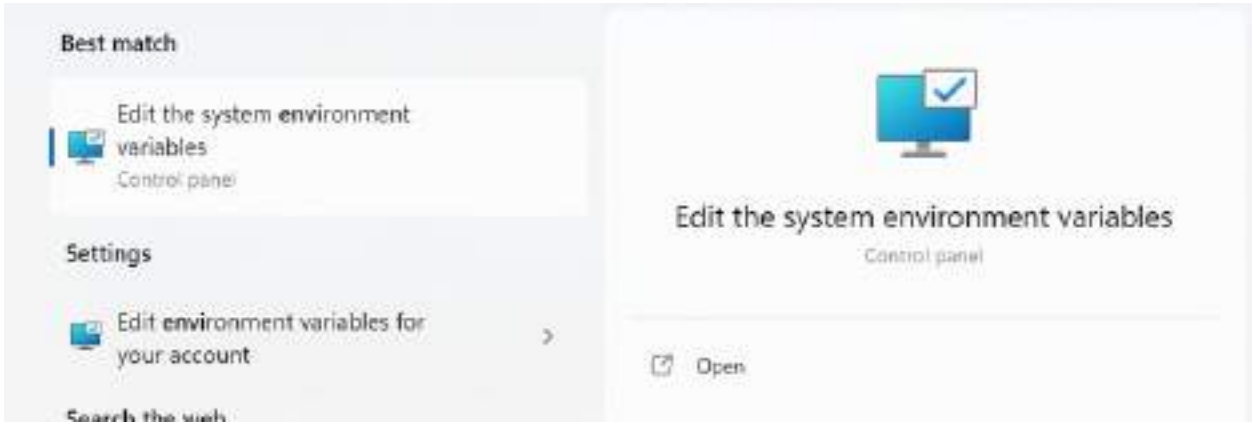
Step 1: Install Terraform from Terraform's official website:
https://www.terraform.io/downloads.html.
Select your operating system followed by either 32bit or 64bit depending on your OS type.

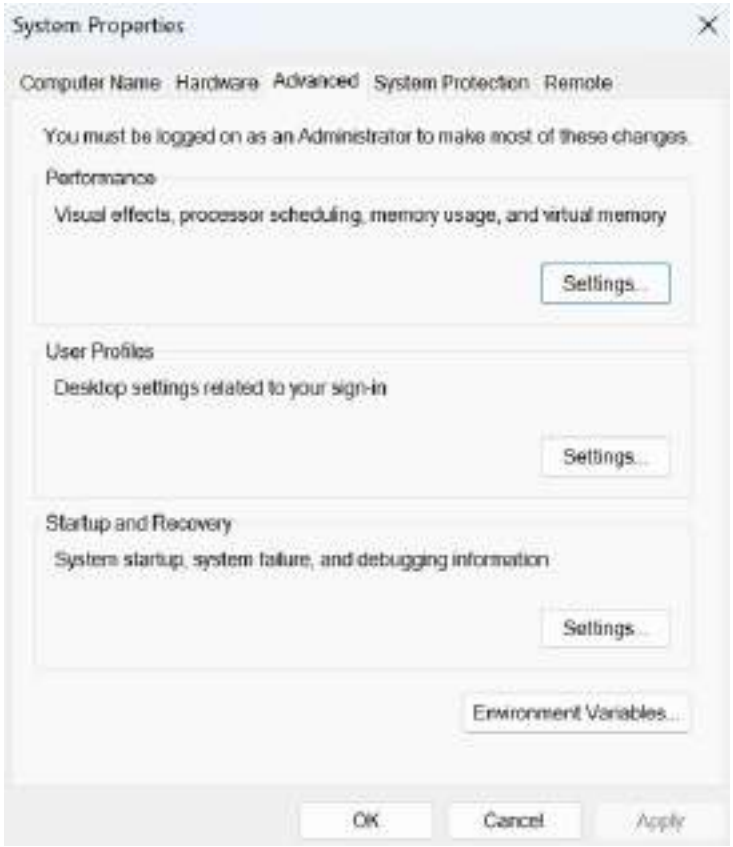


Step 2: Once the download is complete, navigate to the downloaded zipped terraform folder and extract the folder (including the setup file Terraform.exe) into ‘C:\Terraform’ directory.

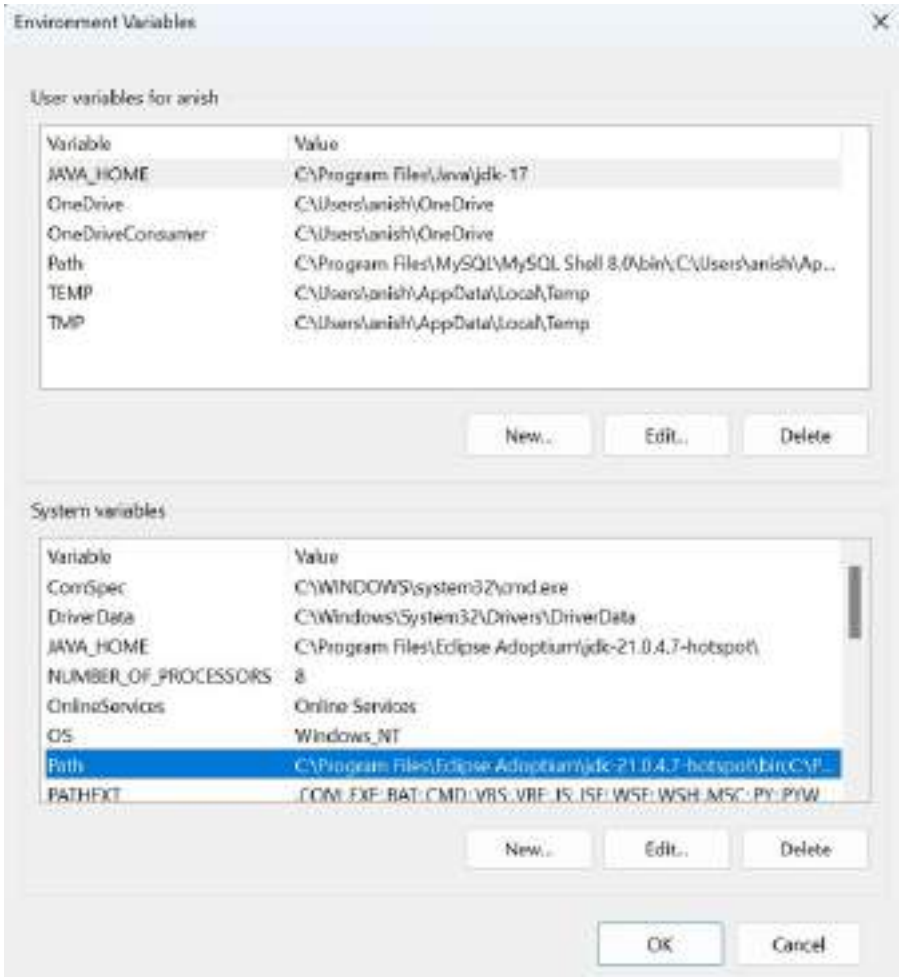


Step 3: Navigate to ‘Edit the system environment variables’ in your device and click on the ‘Environment variables’ button.

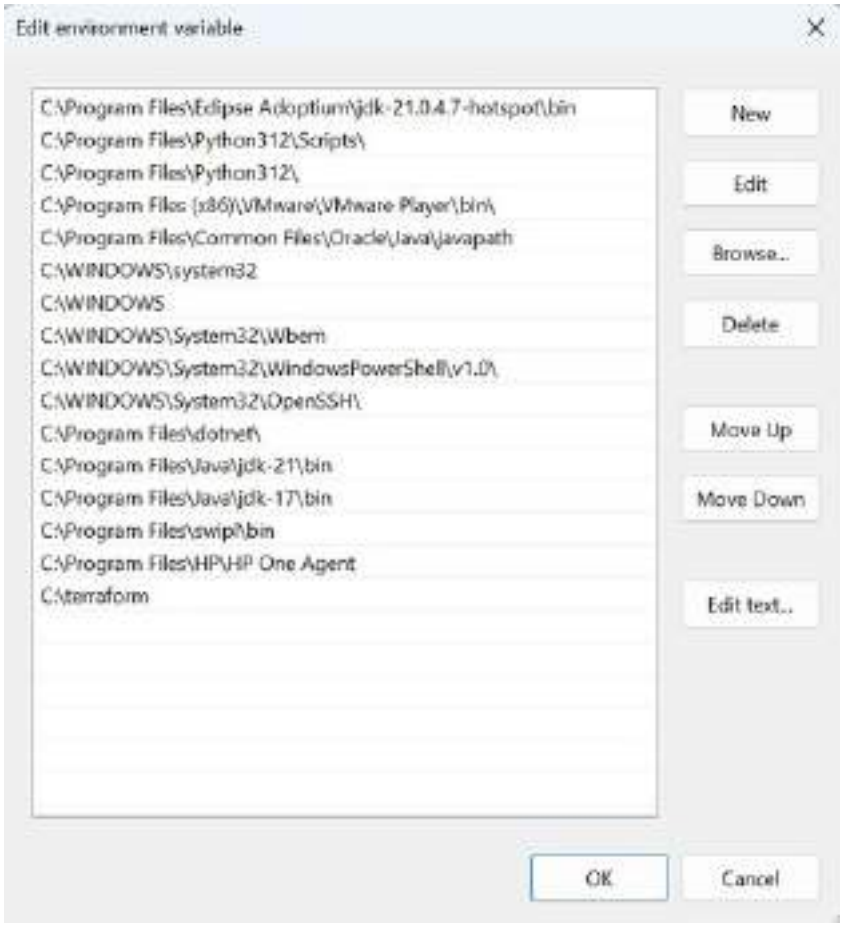




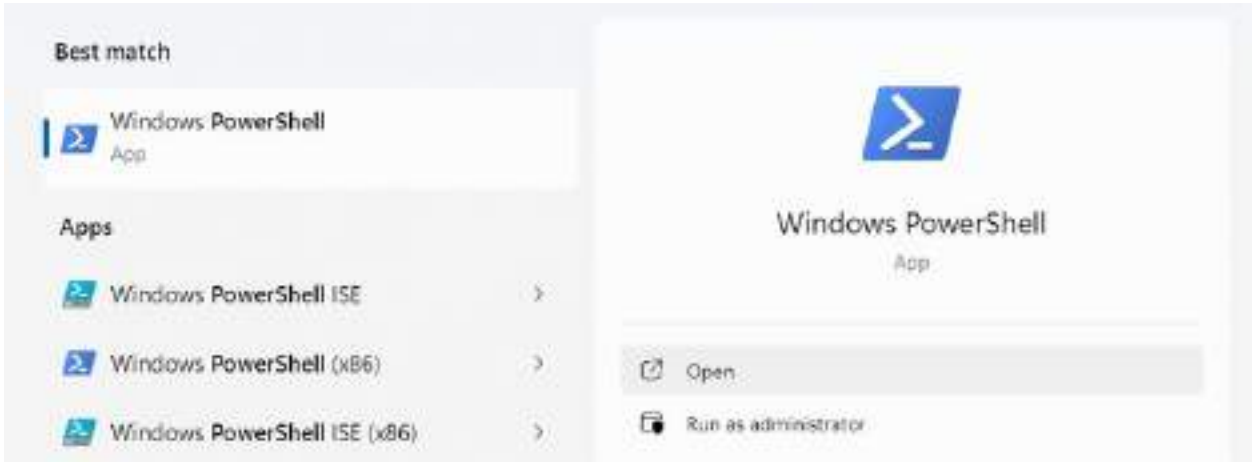
Step 4: In ‘System variables’, click on ‘Path’ and then click on the ‘Edit’ button.



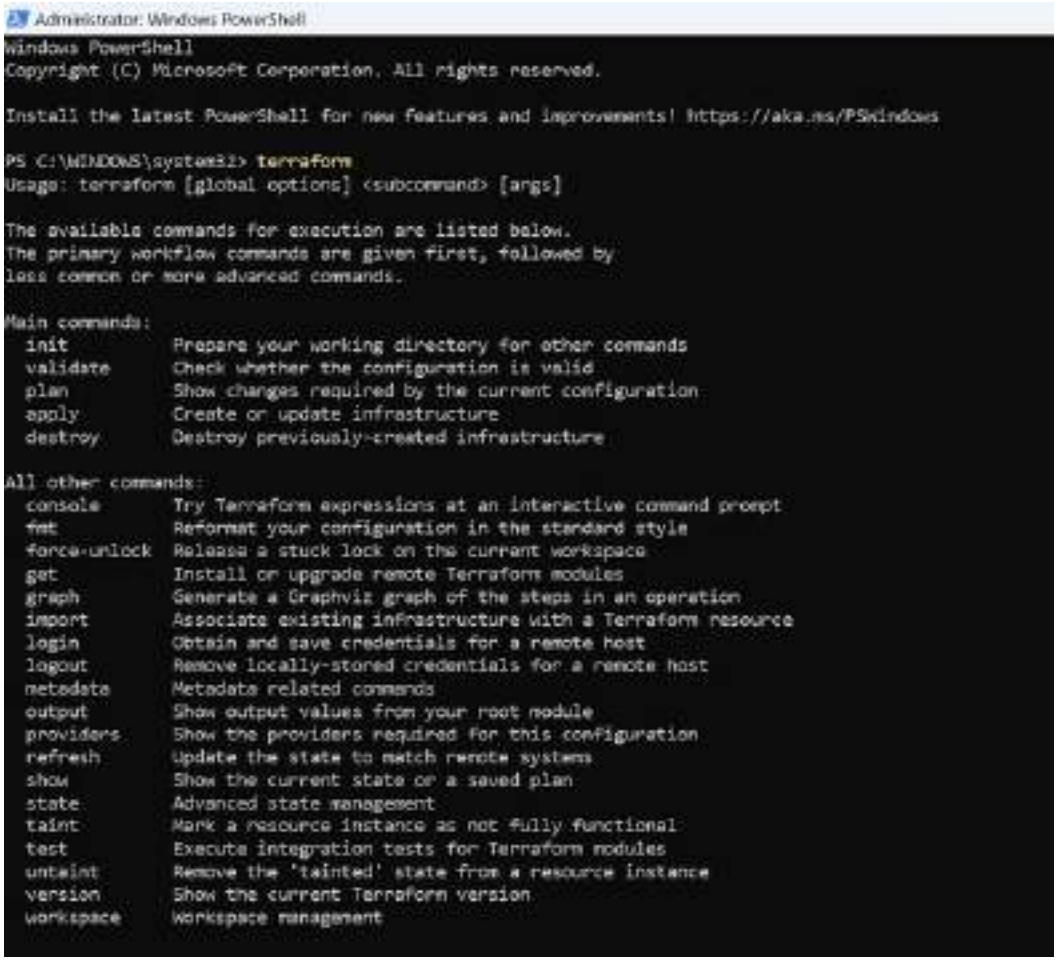
Step 5: Click on ‘New’ and enter ‘C:\terraform’ and click on ‘OK’.



Step 6: Navigate to ‘Windows PowerShell’ on your device and click on ‘Run as administrator’.



Step 7: Open Terraform in PowerShell. This displays all the main commands, other commands and global options. This being your output means that you have successfully installed Terraform on your system.



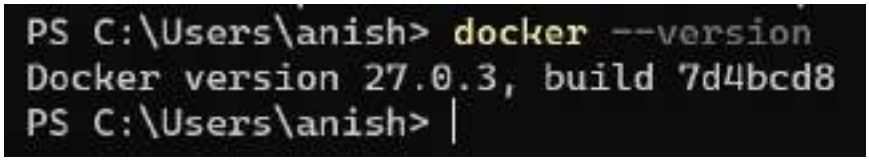
Conclusion: The installation of Terraform on a Windows machine involves downloading the appropriate version, setting up the Terraform folder on the system, and configuring the system's environment variables to recognize Terraform commands. By adding the folder to the system path and verifying the installation via PowerShell, users can effectively utilize Terraform for infrastructure automation. This setup is essential for integratingTerraformintothesystem'scommandline,enablingsmoothexecutionof Terraform commands for managing cloud infrastructure.

Experiment 6

Aim: To Build, change, and destroy AWS / GCP /Microsoft Azure/ DigitalOcean infrastructure Using Terraform.
(S3 bucket or Docker) fdp.

Steps:-

Step 1: Install Docker Desktop from its official website at https://www.docker.com/ and check docker's functionality by using the 'docker' and 'docker --version' commands in Powershell.



Step 2: Create a folder named ‘Terraform Scripts’. Create a folder named ‘Docker’ inside of the ‘Terraform Scripts’ folder and create a new file named docker.tf in this folder. Write the following in the ‘docker.tf’ file (creates a container):-

```
terraform {
  required_providers {
    docker = {
      source ="kreuzwerker/docker"
      version = "2.21.0"
    }
  }
}

provider "docker" {
  host = "npipe:////.//pipe//docker_engine"
}

resource "docker_image" "ubuntu" {
  name = "ubuntu:latest"
}

resource "docker_container" "foo" {
  image = docker_image.ubuntu.image_id
  name ="foo"
}
```

Welcome docker.tf

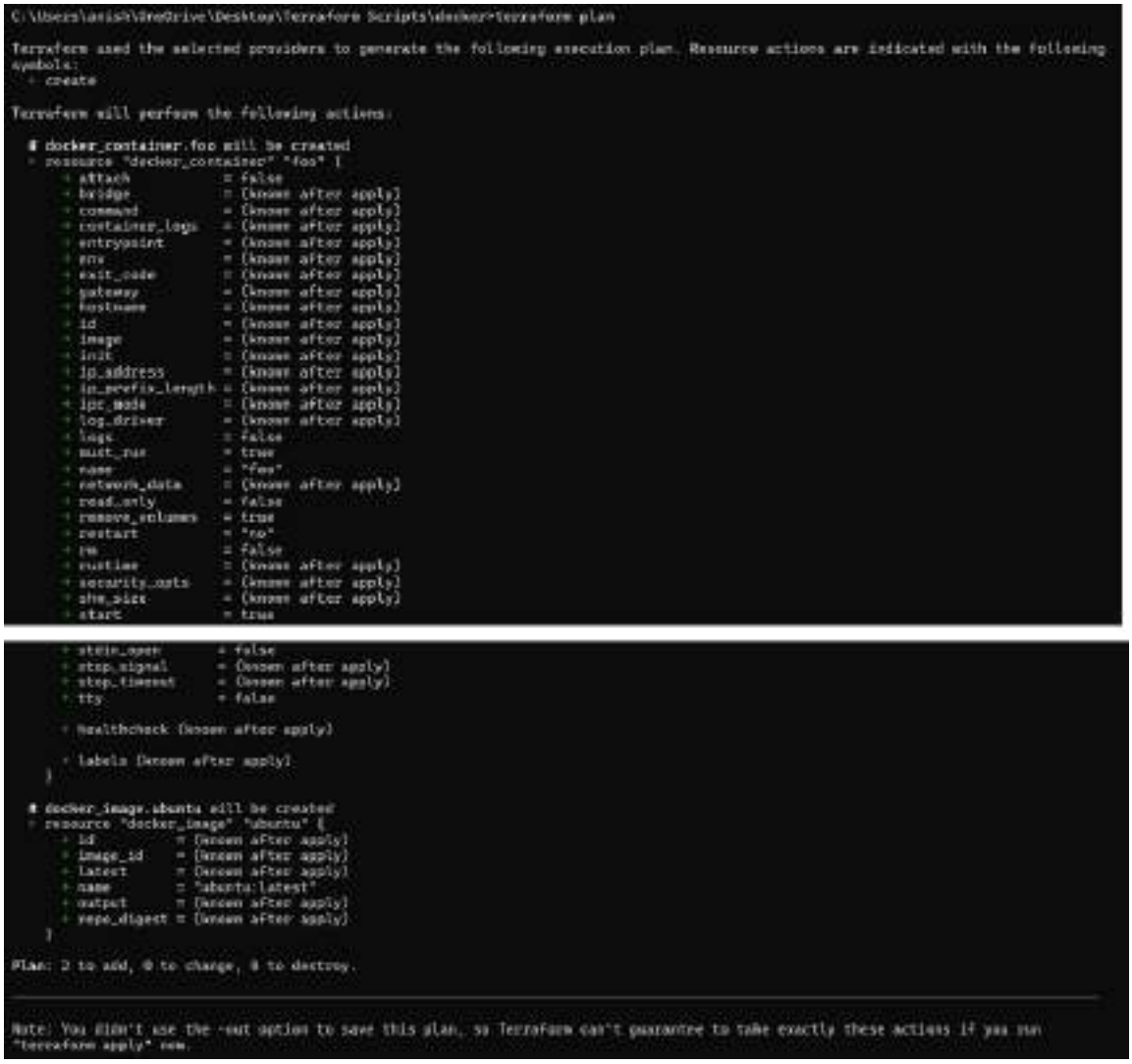
docker > docker.tf > resource "docker_container" "foo"

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "2.21.0"
    }
  }
}

provider "docker" {
  host = "npipe:////.//pipe//docker_engine"
}

# Pulls the image
resource "docker_image" "ubuntu" {
  name = "ubuntu:latest"
}

# Create a container
resource "docker_container" "foo" {
  image = docker_image.ubuntu.image_id
  name  = "foo"
}
```

Step 3: Execute ‘terraform init’ command in Powershell. This command initializes a Terraform working directory by downloading necessary plugins and setting up the backend for state management.

```
C:\Users\anish\OneDrive\Desktop\Terraform Scripts\docker>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding kreuzwerker/docker versions matching "2.21.0"...
- Installing kreuzwerker/docker v2.21.0...
- Installed kreuzwerker/docker v2.21.0 (self-signed, key ID BD080C4571C6104C)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html
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.

C:\Users\anish\OneDrive\Desktop\Terraform Scripts\docker>
```

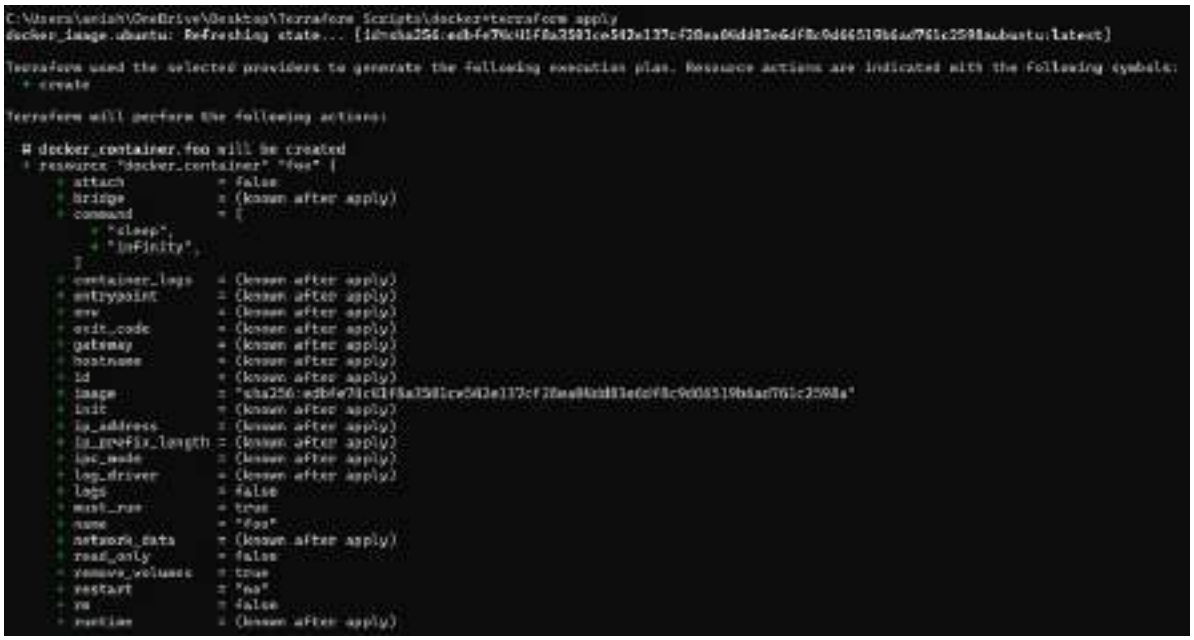
Step 4: Execute ‘terraform plan’ command to generate and display an execution plan, showing what actions Terraform will take to achieve the desired infrastructure state without making any actual changes.

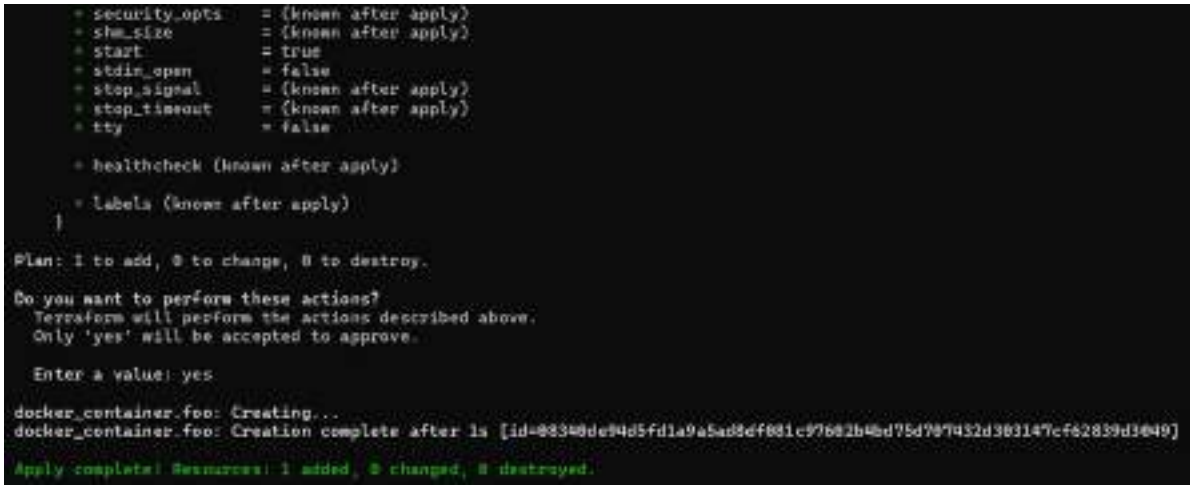


Step 5: Execute ‘terraform apply’ command to execute the changes outlined in the Terraform plan, creating, updating, or deleting resources in the infrastructure.
On executing the command we see that the following error occurs:-

This error occurs because the container's entry command or process gets completed too quickly, causing the container to stop running. To fix the error, add the following lines of code at the end of the docker.tf file.

Now, executing the ‘terraform apply’ command gives the following output:-



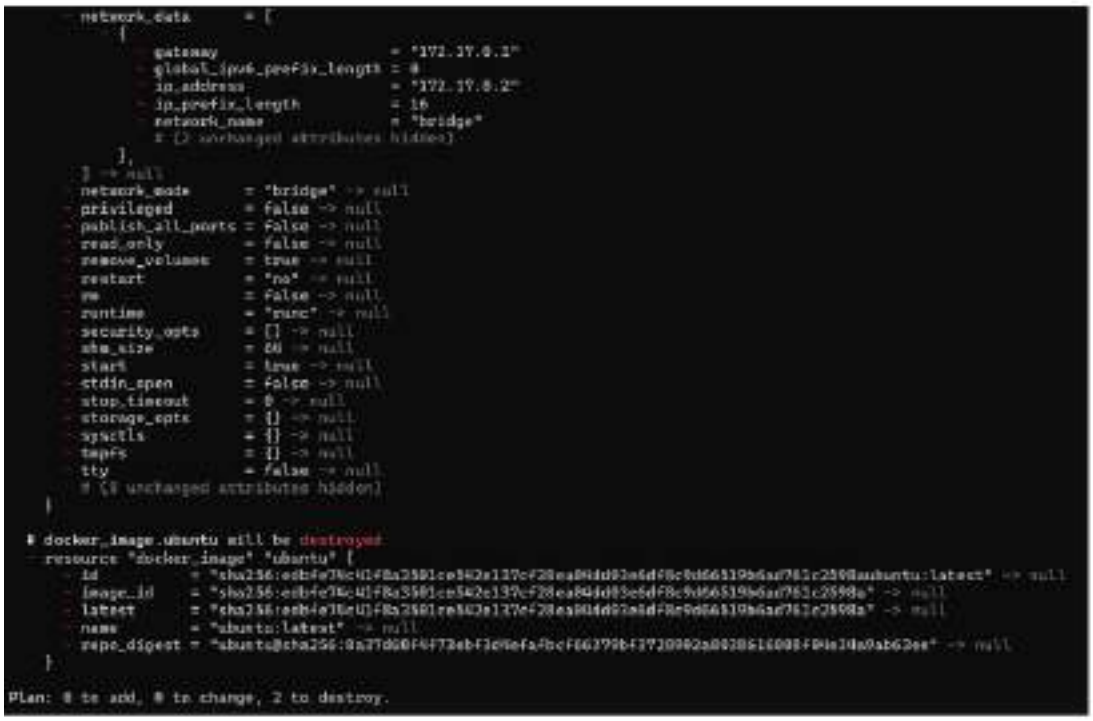


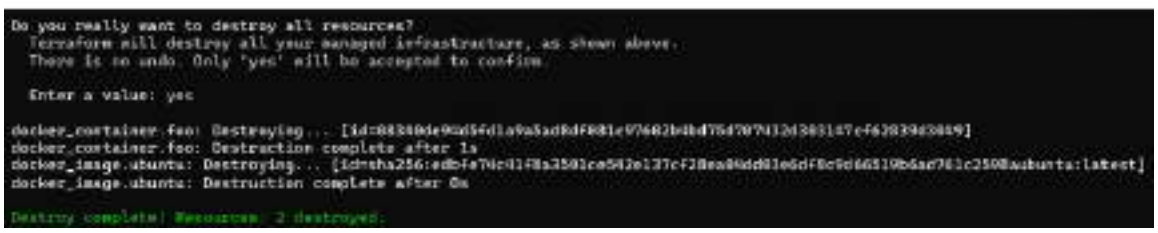
Executing the ‘terraform apply’ command executes the Terraform plan and creates a docker image. This can be seen as such:-

- Dockerimagesbeforeexecuting‘terraformapply’:-

- Dockerimagesafterexecuting‘terraformapply’:-

Step 6: Execute the ‘terraform destroy’ command to remove all the infrastructure resources that Terraform previously created, effectively tearing down the environment. This automatically deletes the docker image that was created in the previous step.





Docker images after executing the ‘terraform destroy’ command:-

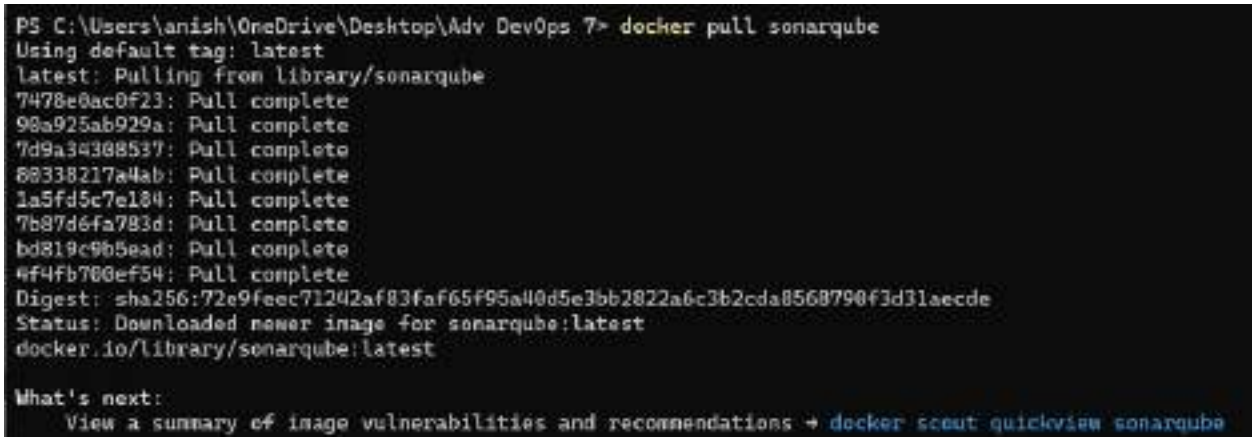
Conclusion: This experiment demonstrates how to build, modify, and destroy infrastructure using Terraform by interacting with Docker containers. By creating a Terraform configuration file (docker.tf), initializing Terraform in the directory, and applying changes, users can pull Docker images and manage containers efficiently. The key commands—terraform init, terraform plan, terraform apply, and terraform destroy—enable infrastructure automation. This setup showcases Terraform's capability to manage Docker containers, making it a versatile tool for multi-cloud and containerized infrastructure management.

Experiment 7

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

Steps:

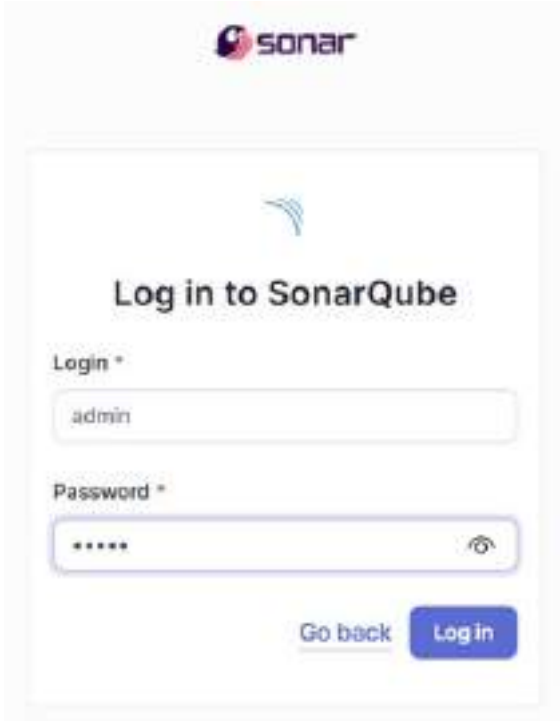
Step 1: Install a SonarQube image by running the ‘docker pull sonarqube’ command on your terminal. This allows for a Sonarqube image to be used on a local machine without having to install the SonarQube application.



Step 2: Execute the following command:
docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
This command will run the SonarQube image that was just installed using docker.

Step 3: Go to http://localhost:9000 on your browser and check if SonarQube is starting or not.

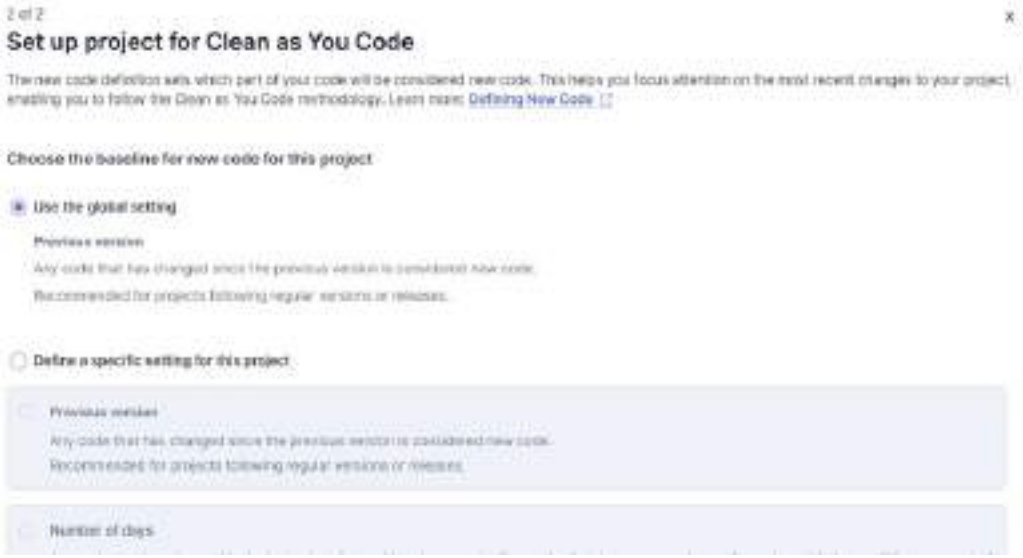
Step 4: On the login page, enter ‘Login’ as admin and ‘Password’ as admin to log in initially. It then asks you to change the password to a password of your choice. Do the same and proceed to the next step.



Step 5: On the SonarQube dashboard, click on ‘Create a local project’.

Step 6: Create a local project by entering the project name and key and click on ‘Next’.

Step 7: Set up your project and click on ‘Create project’.

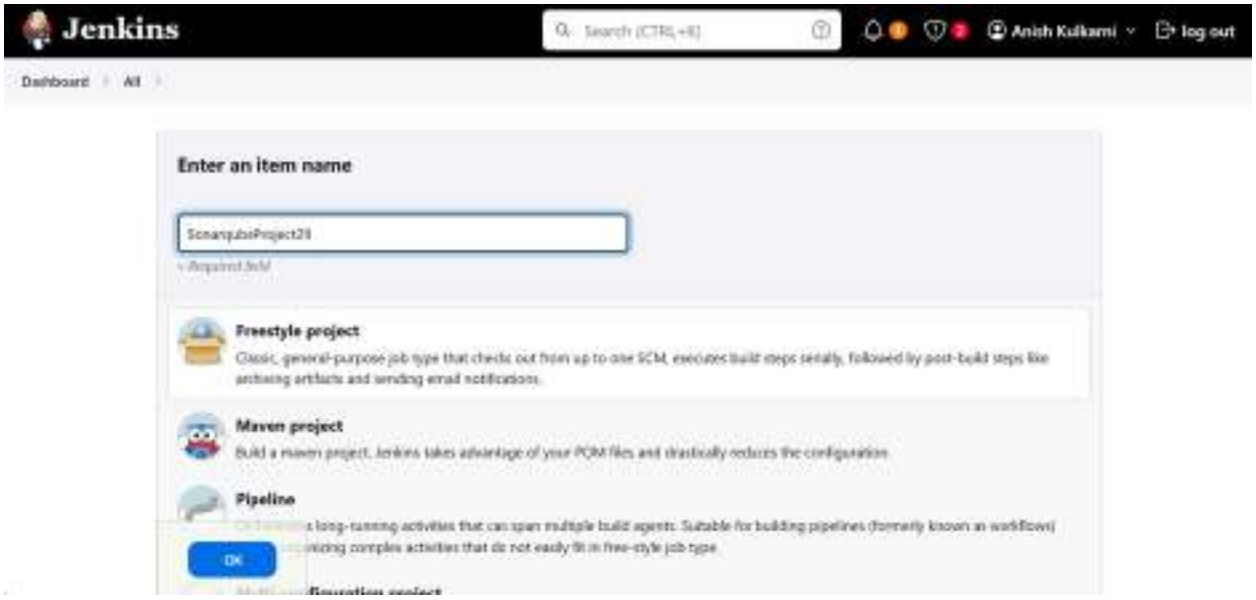


Step 8: Navigate to your Jenkins server (on whichever port it has been installed), click on ‘Manage Jenkins’, click on ‘Plugins’ and search for the ‘SonarQube Scanner’ plugin and install it.

Step 9: Under ‘Manage Jenkins’, click on System. Under the ‘Sonarqube installations’ section, add a server and add a server authentication token if needed.

Step 10: Under ‘Manage Jenkins’, click on ‘Tools’. Under the ‘SonarQube Scanner installations’ section, give your scanner a name, choose the latest version and click on ‘Install automatically’.

Step 11: Create a new Jenkins project by giving it a name and ensure that it is a freestyle project.



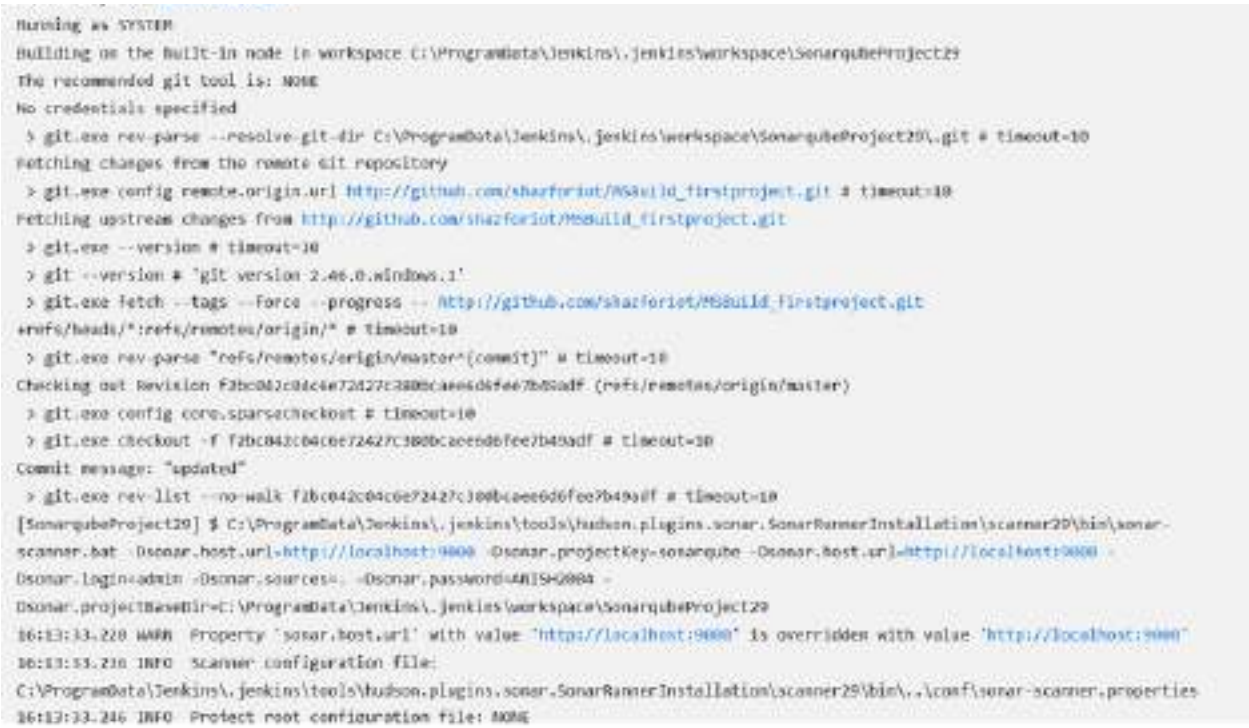
Step 12: In ‘Source Code Management’ section, choose ‘Git’ and enter the following repository URL:-
https://github.com/shazforiot/MSBuild_firstproject
The above is a sample hello-world project with no vulnerabilities.

Step 13: Under Build Steps, enter Sonarqube Scanner and enter these analysis properties. Mention the SonarQube Project Key, Login, Password, Source path and Host URL.

Step 14: On your browser, go to http://localhost:<port_number>/admin/permissions and check the ‘Execute Analysis’ checkbox for Administrator. This gives the required permissions to the user for the analysis stage on SonarQube.

Step 15: Navigate to your Jenkins project and click on ‘Build Now’.

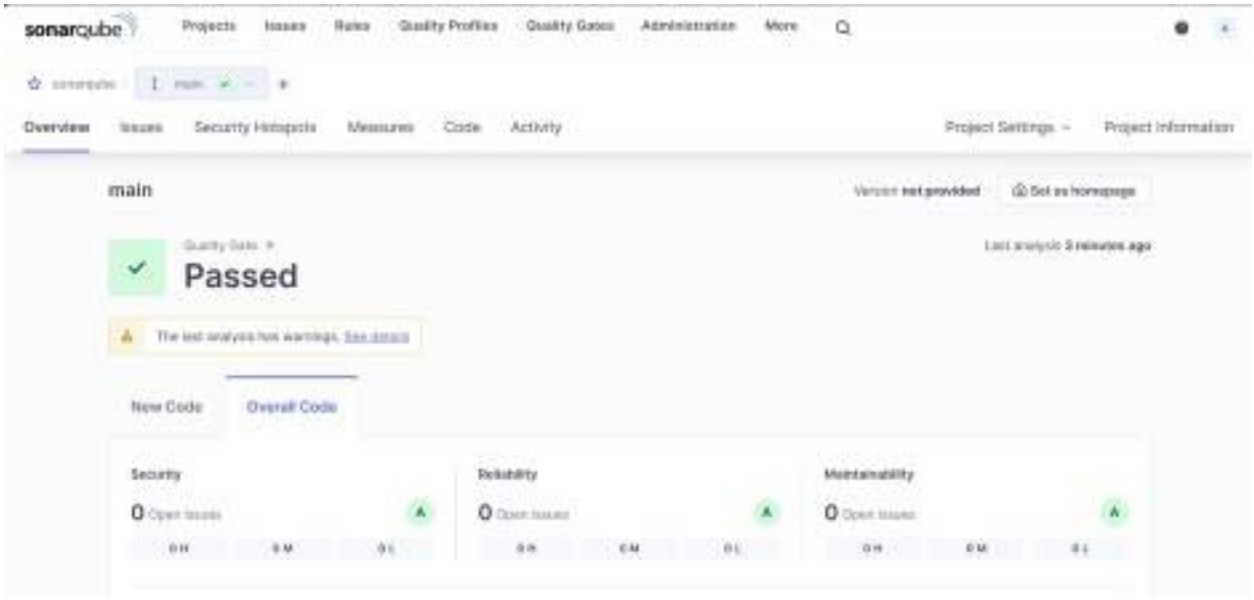
Step 16: Once the build is successful, check the console output.



```
Running as SYSTEM
Building on the built-in node in workspace C:\ProgramData\Jenkins\.jenkins\workspace\SonarqubeProject29
The recommended git tool is: NONE
No credentials specified
 > git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\.jenkins\workspace\SonarqubeProject29\.git # timeout=10
Fetching changes from the remote Git repository
 > git.exe config remote.origin.url http://github.com/shazforiot/MSBuild_firstproject.git # timeout=10
Fetching upstream changes from http://github.com/shazforiot/MSBuild_firstproject.git
 > git.exe --version # timeout=10
 > git --version # 'git version 2.46.0.windows.1'
 > git.exe fetch --tags --force --progress -- http://github.com/shazforiot/MSBuild_firstproject.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
 > git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision f2bc042c04c6e72427c300bcaee6d6fee7b49adf (refs/remotes/origin/master)
 > git.exe config core.sparsecheckout # timeout=10
 > git.exe checkout -f f2bc042c04c6e72427c300bcaee6d6fee7b49adf # timeout=10
Commit message: "updated"
 > git.exe rev-list --no-walk f2bc042c04c6e72427c300bcaee6d6fee7b49adf # timeout=10
[SonarqubeProject29] $ C:\ProgramData\Jenkins\.jenkins\tools\hudson.plugins.sonar.SonarRunnerInstallation\scanner29\bin\sonar-
scanner.bat -Dsonar.host.url=http://localhost:9000 -Dsonar.projectKey=sonarqube -Dsonar.host.url=http://localhost:9000 -
Dsonar.login=admin -Dsonar.sources=. -Dsonar.password=ARISHG004 -
Dsonar.projectBaseDir=C:\ProgramData\Jenkins\.jenkins\workspace\SonarqubeProject29
16:13:13.220 WARN  Property 'sonar.host.url' with value 'http://localhost:9000' is overridden with value 'http://localhost:9000'
16:13:13.236 INFO  Scanner configuration file:
C:\ProgramData\Jenkins\.jenkins\tools\hudson.plugins.sonar.SonarRunnerInstallation\scanner29\bin\..\conf\sonar-scanner.properties
16:13:13.246 INFO  Project root configuration file: NONE
```

```
16:14:12.549 INFO  Sensor C# [csharp] (done) | time=2ms
16:14:12.549 INFO  Sensor Analysis Warnings import [csharp]
16:14:12.549 INFO  Sensor Analysis Warnings import [csharp] (done) | time=0ms
16:14:12.549 INFO  Sensor C# File Caching Sensor [csharp]
16:14:12.549 WARN  Incremental PR analysis: Could not determine common base path, cache will not be computed. Consider setting
'sonar.projectBaseDir' property.
16:14:12.549 INFO  Sensor C# File Caching Sensor [csharp] (done) | time=0ms
16:14:12.549 INFO  Sensor Zero Coverage Sensor
16:14:12.555 INFO  Sensor Zero Coverage Sensor (done) | time=0ms
16:14:12.555 INFO  SCM Publisher SCM provider for this project is: git
16:14:12.567 INFO  SCM Publisher 4 source files to be analyzed
16:14:13.180 INFO  SCM Publisher 4/4 source files have been analyzed (done) | time=613ms
16:14:13.184 INFO  CPD Executor Calculating CPD for 0 files
16:14:13.184 INFO  CPD Executor CPD calculation finished (done) | time=0ms
16:14:13.191 INFO  SCM revision ID 'f2bc042c04c6e72427c300bcaee6d6fee7b49adf'
16:14:13.474 INFO  Analysis report generated in 117ms, dir size=201.0 kB
16:14:13.522 INFO  Analysis report compressed in 30ms, zip size=22.2 kB
16:14:13.750 INFO  Analysis report uploaded in 228ms
16:14:13.759 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=sonarqube
16:14:13.759 INFO  Note that you will be able to access the updated dashboard once the server has processed the submitted analysis
report
16:14:13.759 INFO  More about the report processing at http://localhost:9000/api/ce/task?id=fe00e0c2-e0e2-4116-ae21-10cce1b3c001
16:14:13.787 INFO  Analysis total time: 25.432 s
16:14:13.789 INFO  SonarScanner Engine completed successfully
16:14:13.862 INFO  EXECUTION SUCCESS
16:14:13.862 INFO  Total time: 40.606s
Finished: SUCCESS
```

Step 17: Go back to SonarQube and check your project.



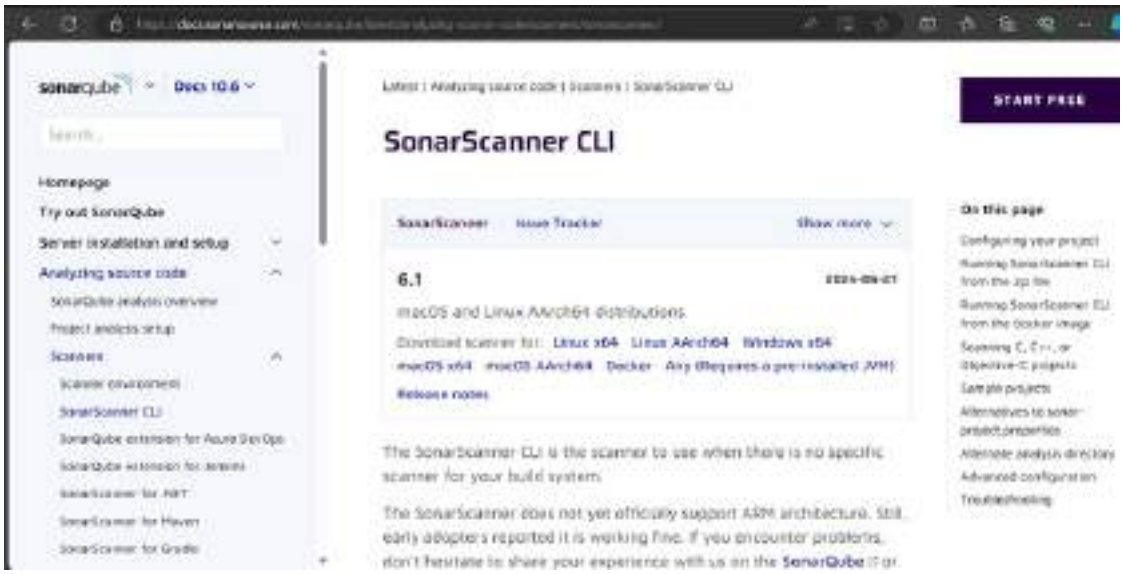
Conclusion: In this experiment, we learned how to integrate Jenkins SAST to SonarQube. We first used a docker image of SonarQube in order to avoid having to install it on our system. Next, SonarQube was configured and a SonarQube project was created. Then, required configurations were done on Jenkins and a Jenkins freestyle project was created which contained links to a code file from a Github repository and also our SonarQube project. When the Jenkins project was built, the SonarQube project displayed that there were no issues with the code in the Jenkins project.

Experiment 8

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 Web / Java / Python application.

Steps:

Step 1: Install the SonarScanner CLI from the following link:
https://docs.sonarsource.com/sonarqube/latest/analyzing-source-code/scanners/sonarscanner/



Once download is complete, extract the downloaded files into a folder.

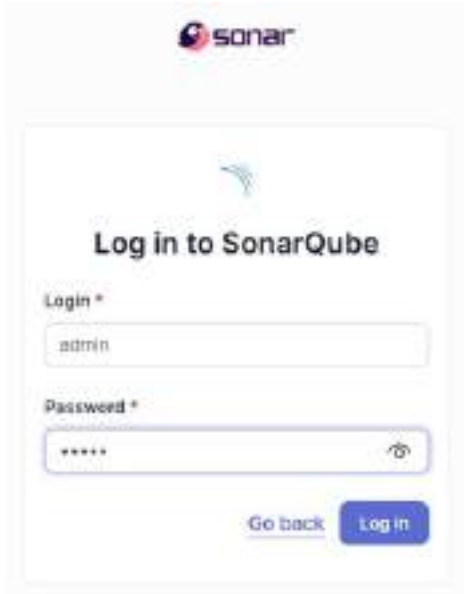
Step 2: Install a SonarQube image by running the 'docker pull sonarqube' command on your terminal. This allows for a Sonarqube image to be used on a local machine without having to install the SonarQube application.

```
PS C:\Users\anish\OneDrive\Desktop\Adv DevOps 7> docker pull sonarqube
Using default tag: latest
latest: Pulling from library/sonarqube
7478e8ac0f23: Pull complete
90a925ab929a: Pull complete
7d9a34368537: Pull complete
8033821a4ab: Pull complete
1a5fd5c7e184: Pull complete
7b87d0fa783d: Pull complete
bd819c9b5ead: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:72e9feec71242af83faf65f95a40d6e3bb1822a6c3b2cda8568790f3d31aecde
Status: Downloaded newer image for sonarqube:latest
docker.io/library/sonarqube:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview sonarqube
```

Step 3: Execute the following command:
docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
This command will run the SonarQube image that was just installed using docker.

Step 4: Go to http://localhost:9000 on your browser and check if SonarQube is starting or not.

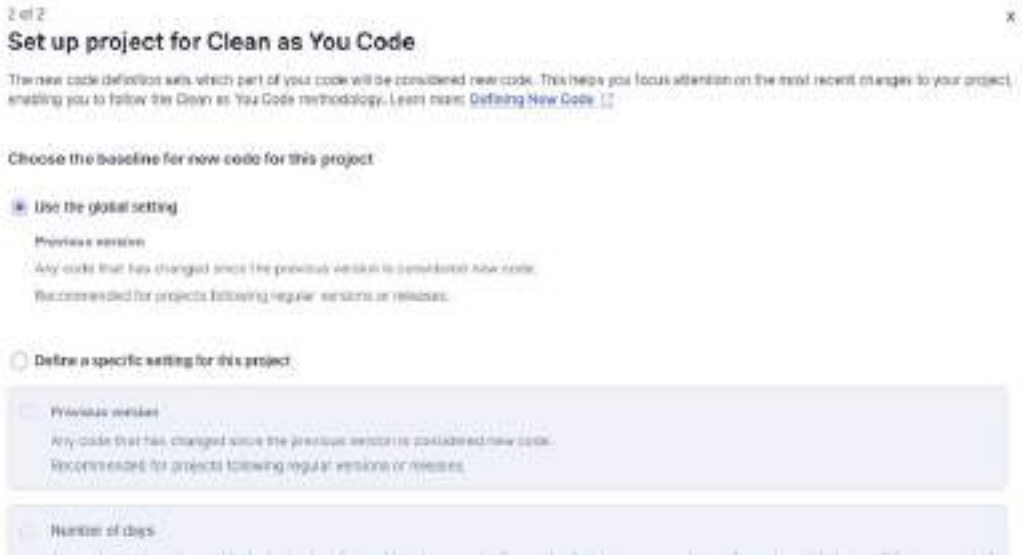
Step 5: On the login page, enter 'Login' as admin and 'Password' as admin to log in initially. It then asks you to change the password to a password of your choice. Do the same and proceed to the next step.



Step 6: On the SonarQube dashboard, click on ‘Create a local project’.

Step 7: Create a local project by entering the project name and key and click on ‘Next’.

Step 8: Set up your project and click on ‘Create project’.

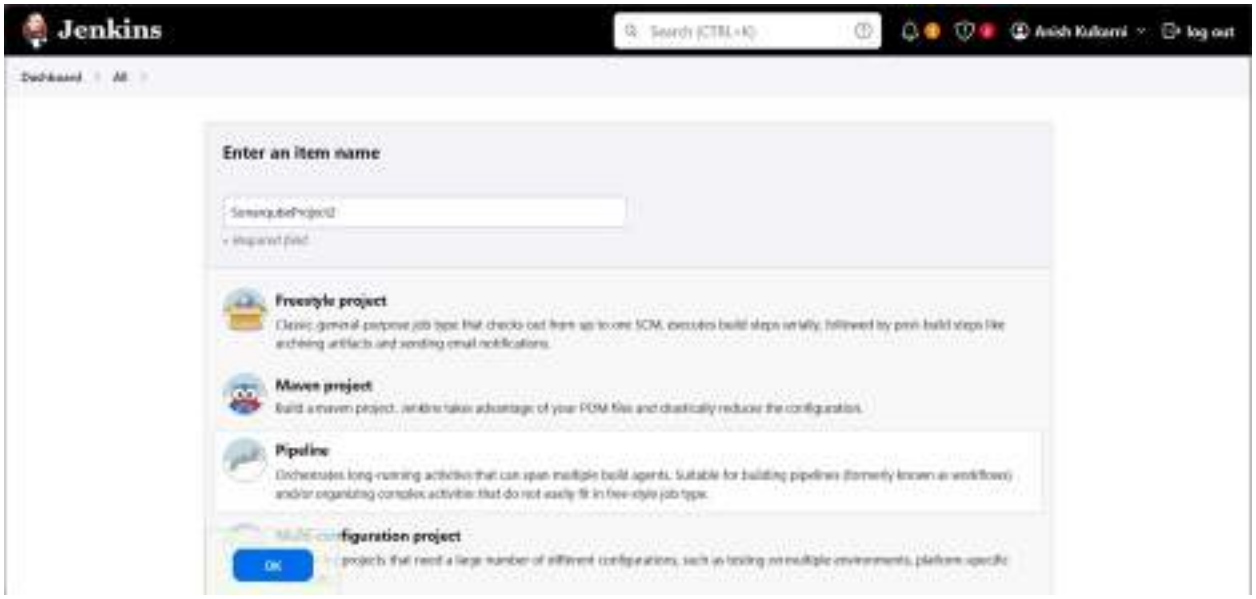


Step 9: Navigate to your Jenkins server (on whichever port it has been installed), click on ‘Manage Jenkins’, click on ‘Plugins’ and search for the ‘SonarQube Scanner’ plugin and install it.

Step 10: Under ‘Manage Jenkins’, click on System. Under the ‘Sonarqube installations’ section, add a server and add a server authentication token if needed.

Step 11: Under ‘Manage Jenkins’, click on ‘Tools’. Under the ‘SonarQube Scanner installations’ section, give your scanner a name, choose the latest version and click on ‘Install automatically’.

Step 12: Create a new Jenkins project by giving it a name and ensure that it is a pipeline project.



Step 13: Under the ‘Pipeline Script’ section, enter the following:-

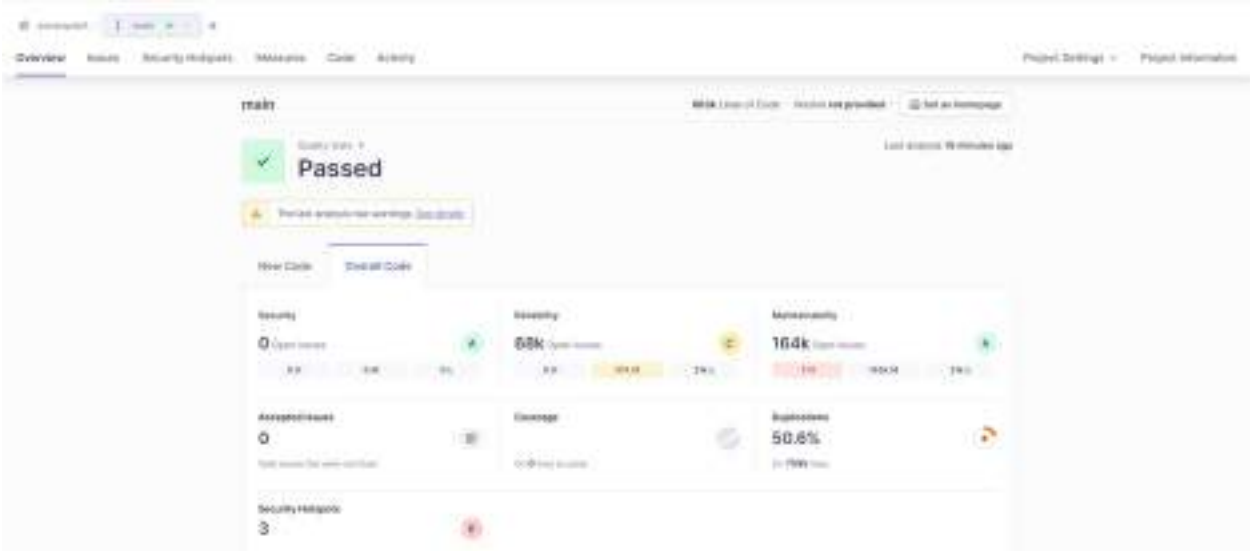
```
node {
        stage('Cloning the GitHub Repo') {
        git 'https://github.com/shazforiot/GOL.git'
        }

         stage('SonarQube analysis') {
                withSonarQubeEnv('sonarqube29') {
                        bat """
                  <PATH_TO_SONARSCANNER_FOLDER>\\bin\\sonar-scanner.bat^
                        -D sonar.login=<SONARQUBE_LOGIN> ^
                        -D sonar.password=<SONARQUBE_PASSWORD> ^
                        -D sonar.projectKey=<PROJECT_KEY> ^
                        -D sonar.exclusions=vendor/**,resources/**,**/*.java ^
                   -D sonar.host.url=http://localhost:9000/
                         """
        }
    }
}
```

The above is a java sample project which has a lot of repetitions and issues that will be detected by SonarQube.

Step 14: Go back to Jenkins, navigate to your Jenkins project and click on ‘Build Now’.

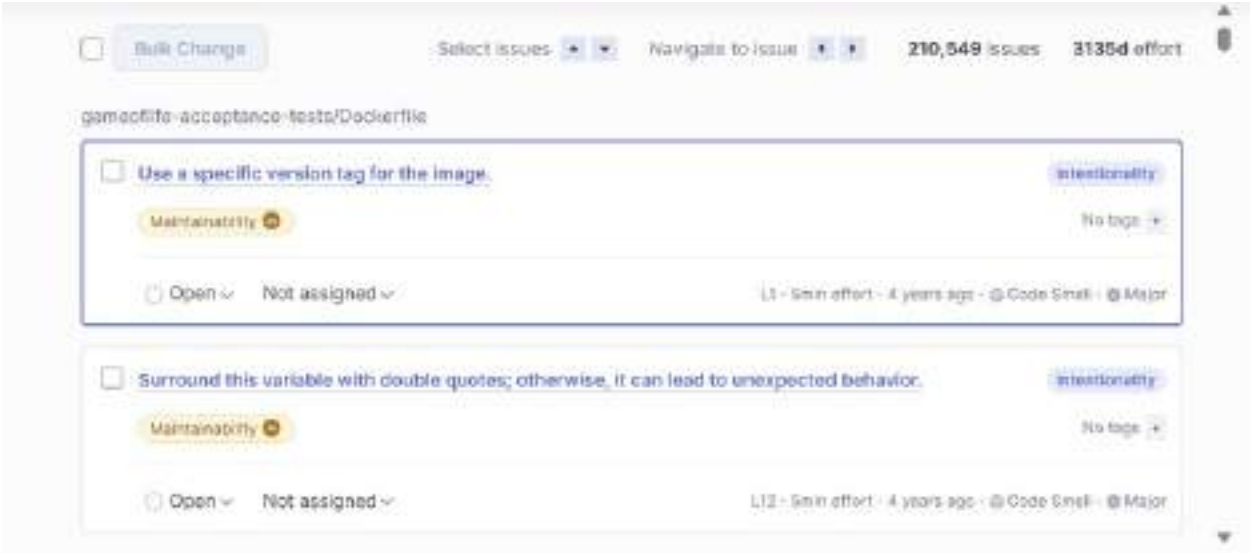
Step 15: Once build is successfully completed, check the console output.

Step 16: Go back to SonarQube and check your project.



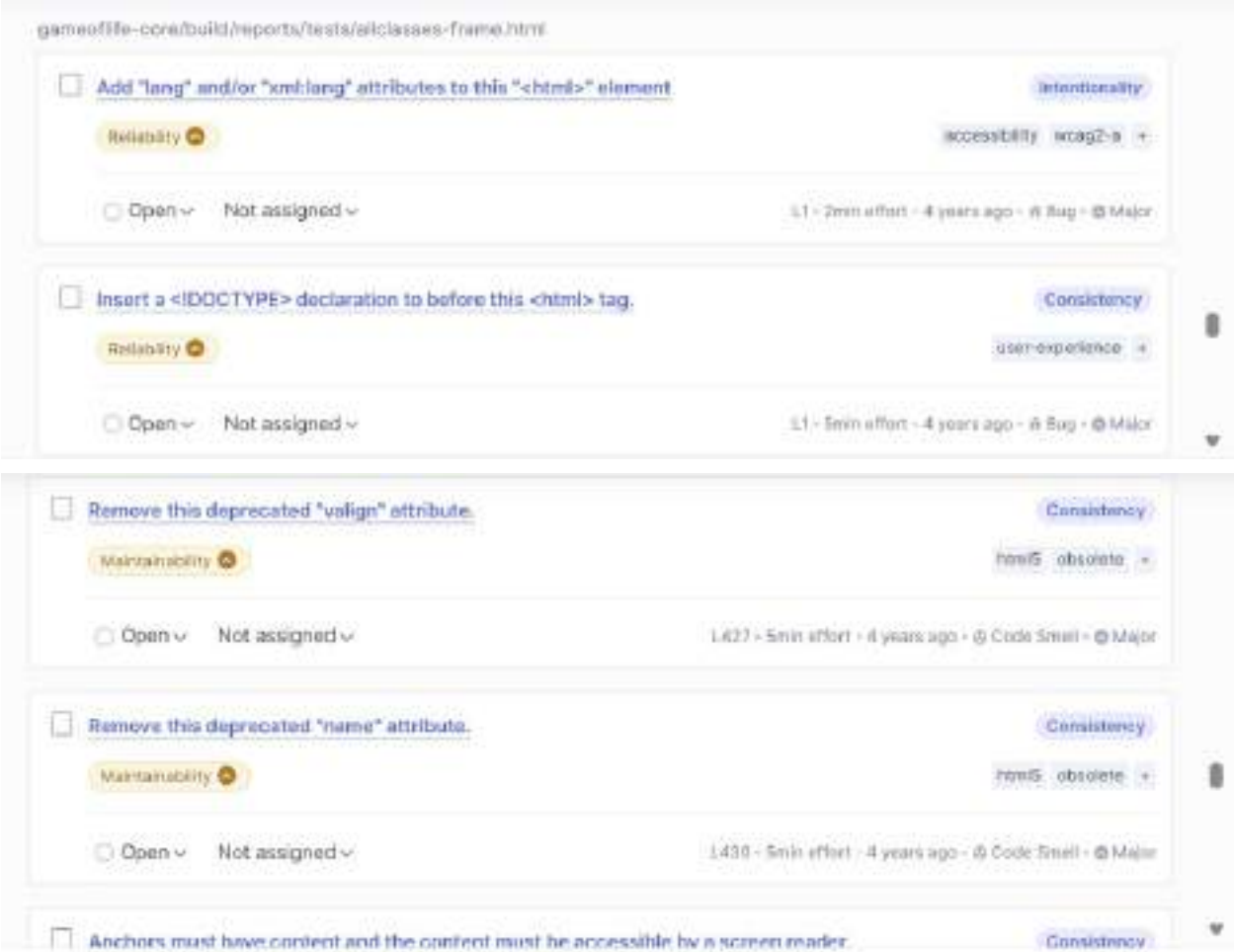
Step 17: Check the different types of issues with the code:-

- Codeproblems:-
  - Intentionality:-



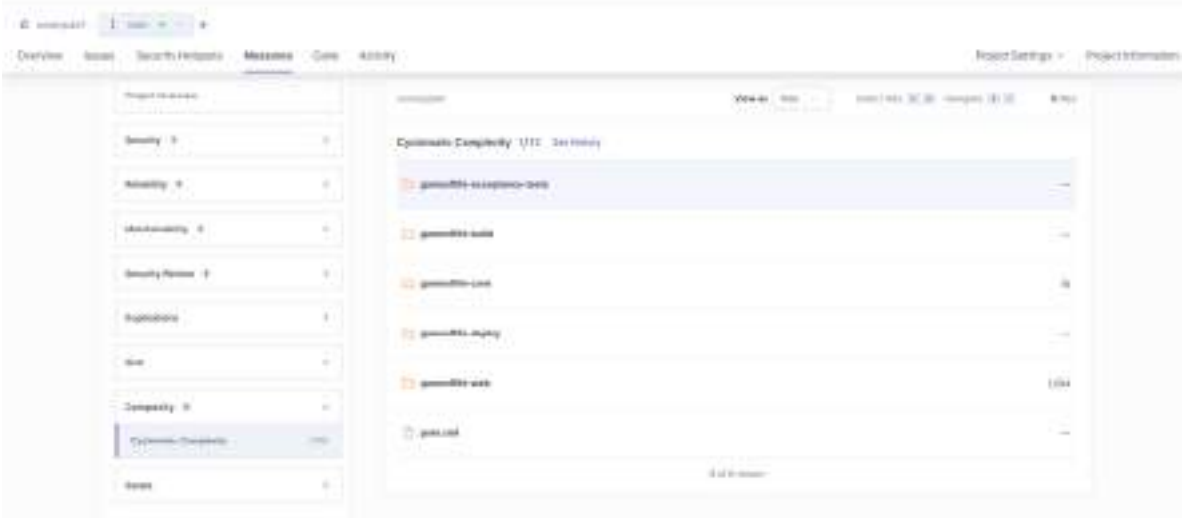
- Consistency:-

- Bugs and Code Smells:-



● Duplications:-

● Cyclomaticcomplexities:-



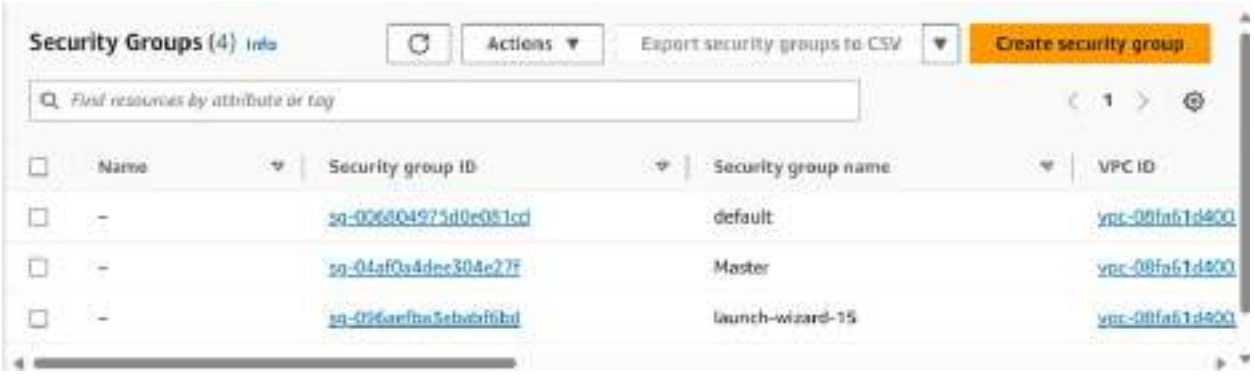
Conclusion: In this experiment, we learned how to create a Jenkins CI/CD Pipeline with SonarQube integration to perform a static analysis of the code to detect bugs, code smells, and security vulnerabilities on a sample Java application. A pipeline project is created in Jenkins and a pipeline script contains the link to the Java application on which the SonarQube analysis is to be done. Then the pipeline project is configured as per needs and built. The SonarQube project linked to the pipeline project then successfully does the SonarQube analysis and points out all the issues, bugs, duplications etc in the pipeline project.

Experiment 9

Aim:ToUnderstandContinuousmonitoringandInstallationandconfigurationofNagios Core, Nagios Plugins and NRPE (Nagios Remote Plugin Executor) on Linux Machine.

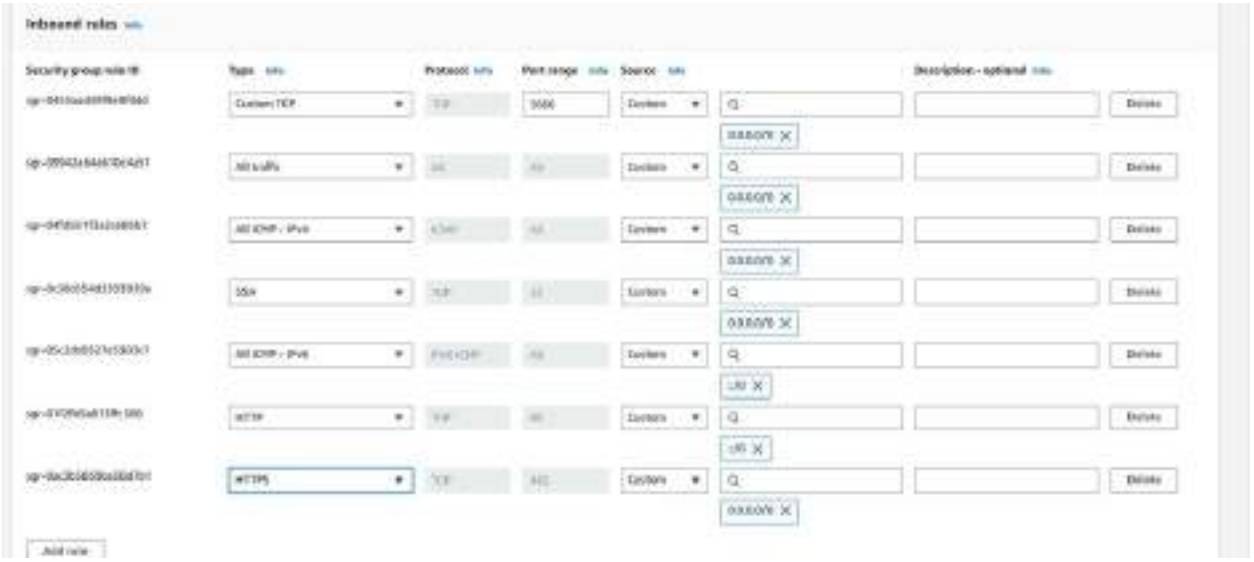
Steps:

Step 1: Navigate to the EC2 section on your AWS console using the ‘Services’ section. Then, from the options in the left-side panel, click on ‘Security groups’. Next, click on ‘Create security group’.

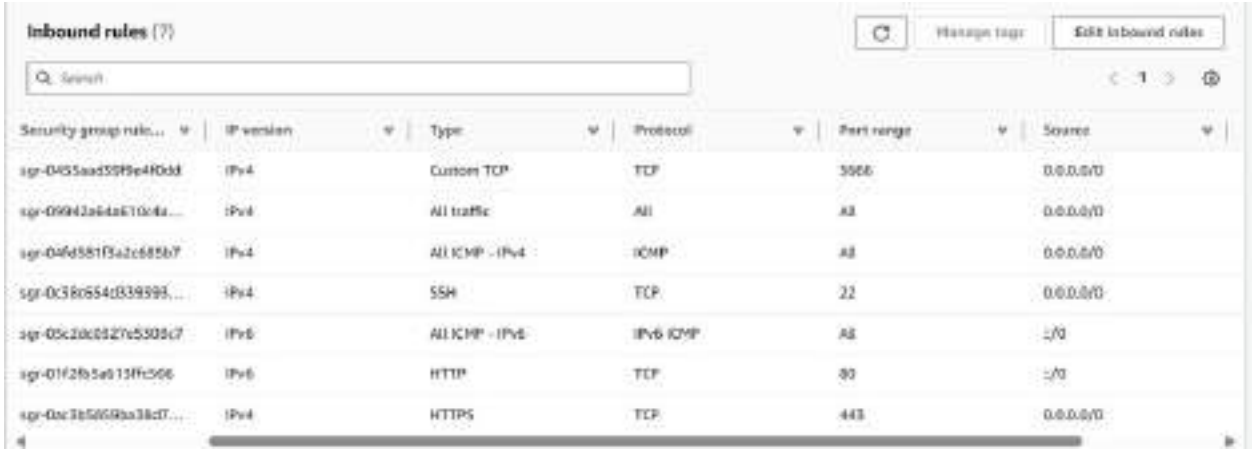


Give your security group a name (here, the name is launch-wizard-15) and then in the ‘Inbound rules’ section, click on ‘Edit’. Then, click on add rules, and add the rules for the following protocols:

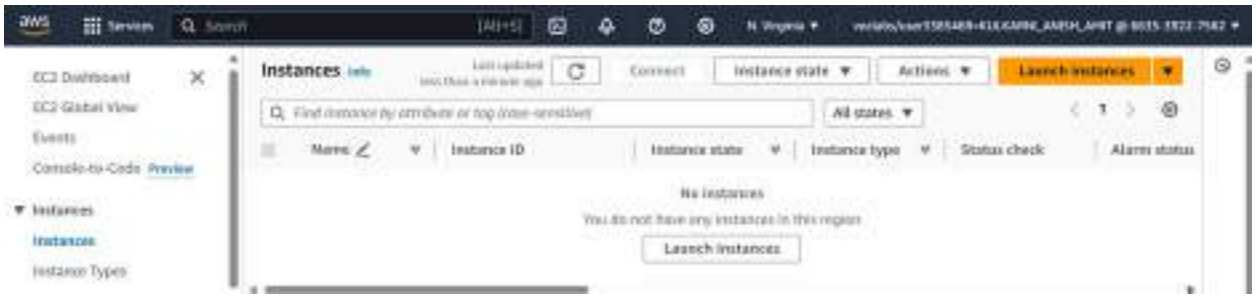
HTTP, All ICMP - IPv6, HTTPS, All traffic, Custom TCP (Port 5666), All ICMP - IPv4



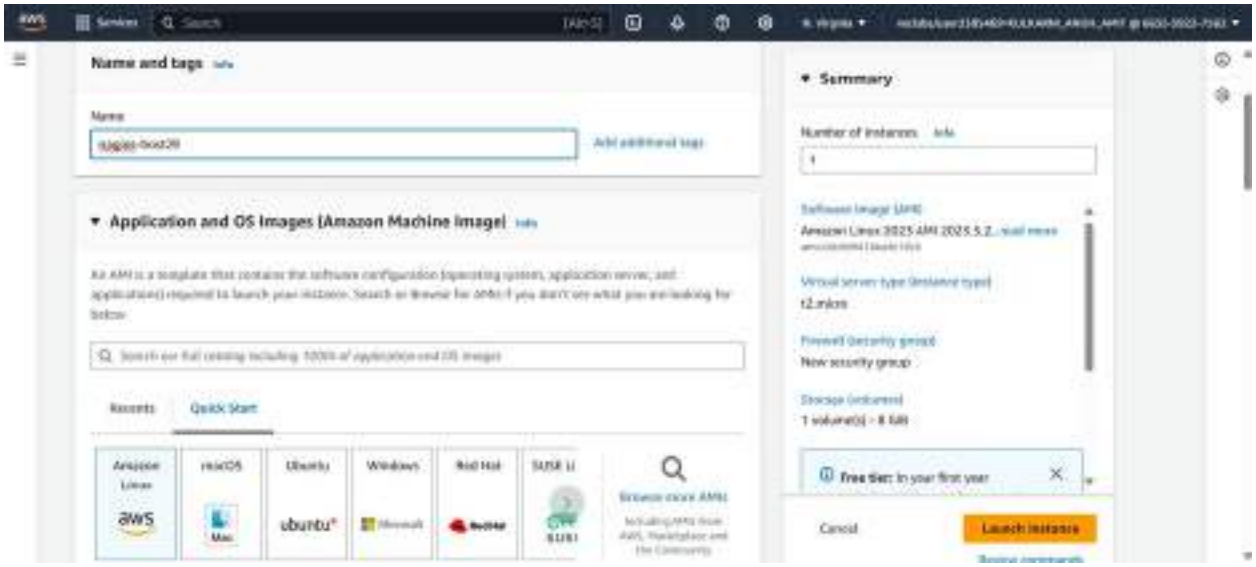
Your security group with the required inbound rules gets created as such:-

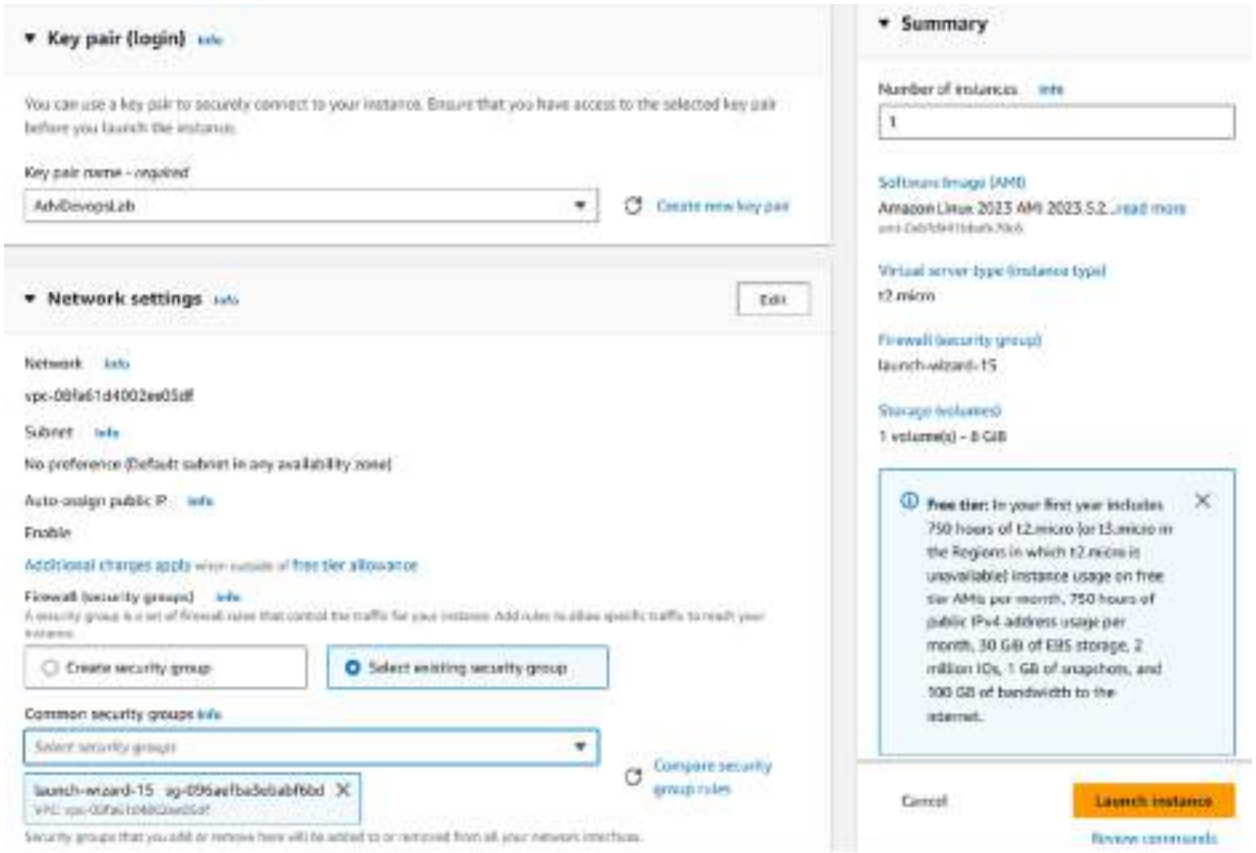


Step 2: Navigate to the EC2 section and click on ‘Launch instances’.



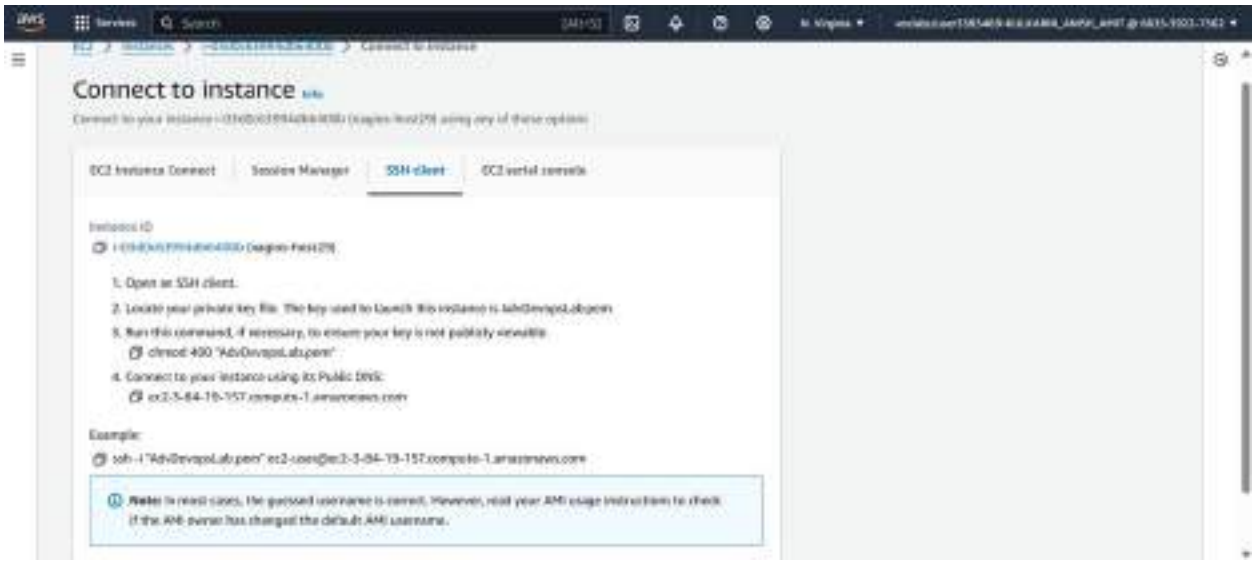
Give your instance a name, choose ‘Amazon Linux’ as the instance type, insert the key pair for which you have the .pem file available in the ‘Key pair’ section, choose the security group that you created in Step 1 in the ‘Network settings’ section, keep all other options as default and click on ‘Launch instance’.



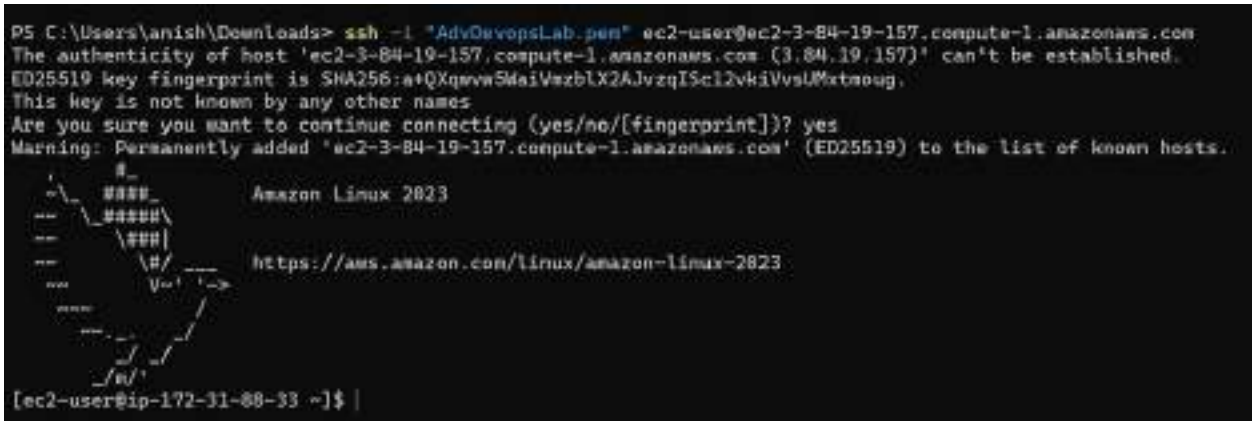


Your instance gets created.

Step 3: Click on the instance ID of your instance and click on ‘Connect’. Then, click on ‘SSH client’ and copy the command under ‘Example’.

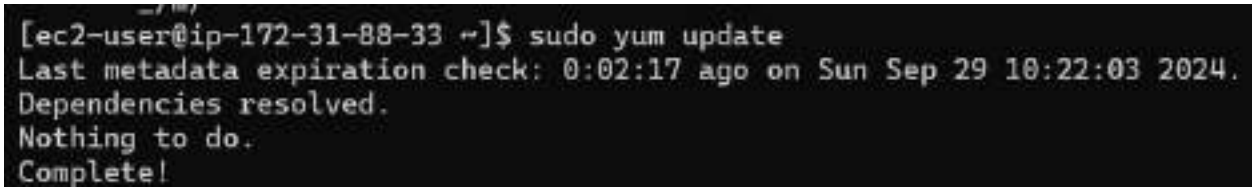


Step 4: Now, we need to connect our local terminal to the instance using SSH. To do so, open the terminal in the folder where the .pem file for your instance’s key pair is located and paste the SSH command that you copied in Step 3.

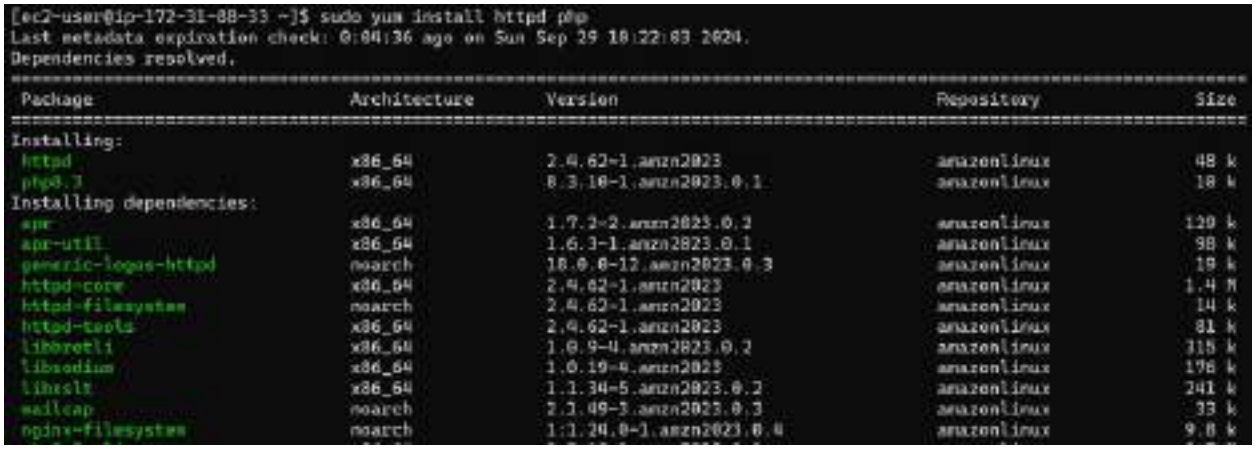


This connects your instance to your local terminal using SSH.

Step 5: First, run the following command:-
sudo yum update
This command will check for any updates for the YUM library.



Step 6: Run the command:
sudo yum install httpd php
This installs an Apache server and a PHP on your instance.



```
[ec2-user@ip-172-31-88-33 ~]$ sudo yum install httpd php
Last metadata expiration check: 0:04:36 ago on Sun Sep 29 10:22:03 2024.
Dependencies resolved.
================================================================================
 Package                 Architecture    Version                 Repository    Size
================================================================================
Installing:
 httpd                   x86_64          2.4.62-1.amzn2023       amazonlinux   48 k
 php8.3                  x86_64          8.3.10-1.amzn2023.0.1   amazonlinux   18 k
Installing dependencies:
 apr                     x86_64          1.7.2-2.amzn2023.0.2    amazonlinux  129 k
 apr-util                x86_64          1.6.3-1.amzn2023.0.1    amazonlinux   98 k
 generic-logos-httpd     noarch          18.0.0-12.amzn2023.0.3  amazonlinux   19 k
 httpd-core              x86_64          2.4.62-1.amzn2023       amazonlinux  1.4 M
 httpd-filesystem        noarch          2.4.62-1.amzn2023       amazonlinux   14 k
 httpd-tools             x86_64          2.4.62-1.amzn2023       amazonlinux   81 k
 libbrotli               x86_64          1.0.9-4.amzn2023.0.2    amazonlinux  315 k
 libsodium               x86_64          1.0.19-4.amzn2023       amazonlinux  176 k
 libxslt                 x86_64          1.1.34-5.amzn2023.0.2   amazonlinux  241 k
 mailcap                 noarch          2.1.49-3.amzn2023.0.3   amazonlinux   33 k
 nginx-filesystem        noarch          1:1.24.0-1.amzn2023.0.4 amazonlinux  9.8 k
```

```
Installed:
  apr-1.7.2-2.amzn2023.0.2.x86_64
  apr-util-1.6.3-1.amzn2023.0.1.x86_64
  apr-util-openssl-1.6.3-1.amzn2023.0.1.x86_64
  generic-logos-httpd-18.0.0-12.amzn2023.0.3.noarch
  httpd-2.4.62-1.amzn2023.x86_64
  httpd-core-2.4.62-1.amzn2023.x86_64
  httpd-filesystem-2.4.62-1.amzn2023.noarch
  httpd-tools-2.4.62-1.amzn2023.x86_64
  libbrotli-1.0.9-4.amzn2023.0.2.x86_64
  libsodium-1.0.19-4.amzn2023.x86_64
  libxslt-1.1.34-5.amzn2023.0.2.x86_64
  mailcap-2.1.49-3.amzn2023.0.3.noarch
  mod_http2-2.0.27-1.amzn2023.0.3.x86_64
  mod_lua-2.4.62-1.amzn2023.x86_64
  nginx-filesystem-1:1.24.0-1.amzn2023.0.4.noarch
  php8.3-8.3.10-1.amzn2023.0.1.x86_64
  php8.3-cli-8.3.10-1.amzn2023.0.1.x86_64
  php8.3-common-8.3.10-1.amzn2023.0.1.x86_64
  php8.3-fpm-8.3.10-1.amzn2023.0.1.x86_64
  php8.3-mbstring-8.3.10-1.amzn2023.0.1.x86_64
  php8.3-opcache-8.3.10-1.amzn2023.0.1.x86_64
  php8.3-pdo-8.3.10-1.amzn2023.0.1.x86_64
  php8.3-process-8.3.10-1.amzn2023.0.1.x86_64
  php8.3-sodium-8.3.10-1.amzn2023.0.1.x86_64
  php8.3-xml-8.3.10-1.amzn2023.0.1.x86_64

Complete!
```

Step 7: Run the command:
sudo yum install gcc glibc glibc-common
This installs the C/C++ compiler (GCC) along with the necessary C libraries required for compiling and running C programs.

```
[ec2-user@ip-172-31-88-33 ~]$ sudo yum install gcc glibc glibc-common
Last metadata expiration check: 0:05:41 ago on Sun Sep 29 10:22:03 2024.
Package glibc-2.34-52.amzn2023.0.11.x86_64 is already installed.
Package glibc-common-2.34-52.amzn2023.0.11.x86_64 is already installed.
Dependencies resolved.
================================================================================
 Package                 Architecture    Version                 Repository    Size
================================================================================
Installing:
 gcc                     x86_64          11.4.1-2.amzn2023.0.2   amazonlinux   32 M
Installing dependencies:
 annobin-docs            noarch          10.93-1.amzn2023.0.1    amazonlinux   92 k
 annobin-plugin-gcc      x86_64          10.93-1.amzn2023.0.1    amazonlinux  887 k
 cpp                     x86_64          11.4.1-2.amzn2023.0.2   amazonlinux   10 M
 gc                      x86_64          8.0.4-5.amzn2023.0.2    amazonlinux  105 k
 glibc-devel             x86_64          2.34-52.amzn2023.0.11   amazonlinux   27 k
 glibc-headers-x86       noarch          2.34-52.amzn2023.0.11   amazonlinux  427 k
 guile22                 x86_64          2.2.7-2.amzn2023.0.3    amazonlinux  6.4 M
 kernel-headers          x86_64          6.1.109-118.189.amzn2023 amazonlinux 1.4 M
 libmpc                  x86_64          1.2.1-2.amzn2023.0.2    amazonlinux   62 k
 libtool-ltdl            x86_64          2.4.7-1.amzn2023.0.3    amazonlinux   38 k
 libxcrypt-devel         x86_64          4.4.33-7.amzn2023       amazonlinux   32 k
 make                    x86_64          1:4.3-5.amzn2023.0.2    amazonlinux  534 k

Transaction Summary
================================================================================
Install  13 Packages
```

```
Installed:
  annobin-docs-10.93-1.amzn2023.0.1.noarch                annobin-plugin-gcc-10.93-1.amzn2023.0.1.x86_64
  cpp-11.4.1-2.amzn2023.0.2.x86_64                        gc-8.0.4-6.amzn2023.0.2.x86_64
  gcc-11.4.1-2.amzn2023.0.2.x86_64                        glibc-devel-2.34-52.amzn2023.0.11.x86_64
  glibc-headers-x86-2.34-52.amzn2023.0.11.noarch          guile22-2.2.7-2.amzn2023.0.3.x86_64
  kernel-headers-6.1.109-118.189.amzn2023.x86_64          libmpc-1.2.1-2.amzn2023.0.2.x86_64
  libtool-ltdl-2.4.7-1.amzn2023.0.3.x86_64                libxcrypt-devel-4.4.33-7.amzn2023.x86_64
  make-1:4.3-5.amzn2023.0.2.x86_64

Complete!
[ec2-user@ip-172-31-88-33 ~]$
```

Step 8: Run the command:
sudo yum install gd gd-devel

```
[ec2-user@ip-172-31-88-33 ~]$ sudo yum install gd gd-devel
Last metadata expiration check: 0:06:51 ago on Sun Sep 29 10:22:03 2024.
Dependencies resolved.
================================================================================
 Package                 Architecture    Version                  Repository    Size
================================================================================
Installing:
 gd                      x86_64          2.3.3-5.amzn2023.0.3     amazonlinux  139 k
 gd-devel                x86_64          2.3.3-5.amzn2023.0.3     amazonlinux   38 k
Installing dependencies:
 brotli                  x86_64          1.0.9-4.amzn2023.0.2     amazonlinux  314 k
 brotli-devel            x86_64          1.0.9-4.amzn2023.0.2     amazonlinux   31 k
 bzip2-devel             x86_64          1.0.8-6.amzn2023.0.2     amazonlinux  214 k
 cairo                   x86_64          1.17.6-2.amzn2023.0.1    amazonlinux  684 k
 cmake-filesystem        x86_64          3.22.2-1.amzn2023.0.4    amazonlinux   16 k
 fontconfig              x86_64          2.13.94-2.amzn2023.0.2   amazonlinux  273 k
 fontconfig-devel        x86_64          2.13.94-2.amzn2023.0.2   amazonlinux  128 k
 fonts-filesystem        noarch          1:2.0.5-12.amzn2023.0.2  amazonlinux  9.5 k
 freetype                x86_64          2.13.2-5.amzn2023.0.1    amazonlinux  423 k
 freetype-devel          x86_64          2.13.2-5.amzn2023.0.1    amazonlinux  912 k
```

```
  libffi-devel-3.4.4-1.amzn2023.0.1.x86_64                libicu-67.1-7.amzn2023.0.3.x86_64
  libicu-devel-67.1-7.amzn2023.0.3.x86_64                 libjpeg-turbo-2.1.4-2.amzn2023.0.5.x86_64
  libjpeg-turbo-devel-2.1.4-2.amzn2023.0.5.x86_64         libmount-devel-2.37.4-1.amzn2023.0.4.x86_64
  libpng-2:1.6.37-10.amzn2023.0.6.x86_64                  libpng-devel-2:1.6.37-10.amzn2023.0.6.x86_64
  libselinux-devel-3.4-5.amzn2023.0.2.x86_64              libsepol-devel-3.4-3.amzn2023.0.3.x86_64
  libtiff-4.4.0-4.amzn2023.0.18.x86_64                    libtiff-devel-4.4.0-4.amzn2023.0.18.x86_64
  libwebp-1.2.4-1.amzn2023.0.6.x86_64                     libwebp-devel-1.2.4-1.amzn2023.0.6.x86_64
  libxcb-1.13.1-7.amzn2023.0.2.x86_64                     libxcb-devel-1.13.1-7.amzn2023.0.2.x86_64
  libxml2-devel-2.10.4-1.amzn2023.0.6.x86_64              pcre2-devel-10.40-1.amzn2023.0.3.x86_64
  pcre2-utf16-10.40-1.amzn2023.0.3.x86_64                 pcre2-utf32-10.40-1.amzn2023.0.3.x86_64
  pixman-0.40.0-3.amzn2023.0.3.x86_64                     systemtap-sdt-devel-4.6.1-2.amzn2023.0.2.x86_64
  xml-common-0.6.3-56.amzn2023.0.2.noarch                 xorg-x11-proto-devel-2021.4-1.amzn2023.0.2.noarch
  xz-devel-5.2.5-9.amzn2023.0.2.x86_64                    zlib-devel-1.2.11-33.amzn2023.0.5.x86_64

Complete!
[ec2-user@ip-172-31-88-33 ~]$
```

Step 9: Run the commands:
sudo adduser -m nagios
sudo passwd nagios
This creates a user named ‘nagios’, ensures it has a home directory and sets up a password for it.

```
[ec2-user@ip-172-31-88-33 ~]$ sudo adduser -m nagios
sudo passwd nagios
Changing password for user nagios.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

Step 10: Create a user group named ‘nagcmd’ to execute nagios commands.
sudo groupadd nagcmd

```
[ec2-user@ip-172-31-88-33 ~]$ sudo groupadd nagcmd
[ec2-user@ip-172-31-88-33 ~]$
```

Step11:Addusersapacheandnagiostothisusergroup.
sudo usermod -a -G nagcmd nagios
sudo usermod -a -G nagcmd apache

```
[ec2-user@ip-172-31-88-33 ~]$ sudo usermod -a -G nagcmd nagios
sudo usermod -a -G nagcmd apache
[ec2-user@ip-172-31-88-33 ~]$
```

Step 12: mkdir ~/downloads
cd ~/downloads
This creates a directory named ‘downloads’, to store the files of the nagios server that are downloaded.

```
[ec2-user@ip-172-31-88-33 ~]$ mkdir ~/downloads
cd ~/downloads
[ec2-user@ip-172-31-88-33 downloads]$
```

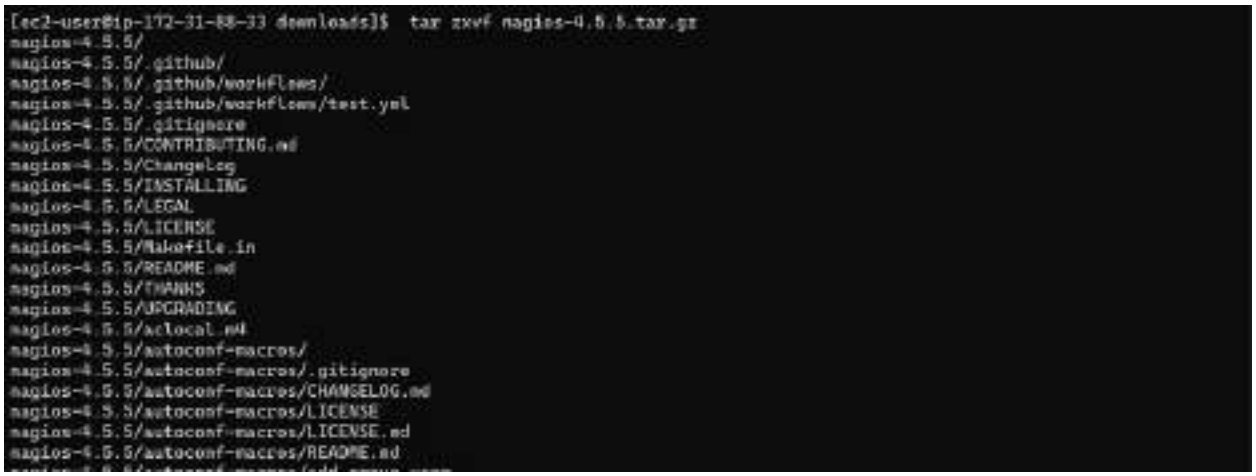
Step 13: wget https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.5.5.tar.gz
The above command installs the latest version of nagios-core.

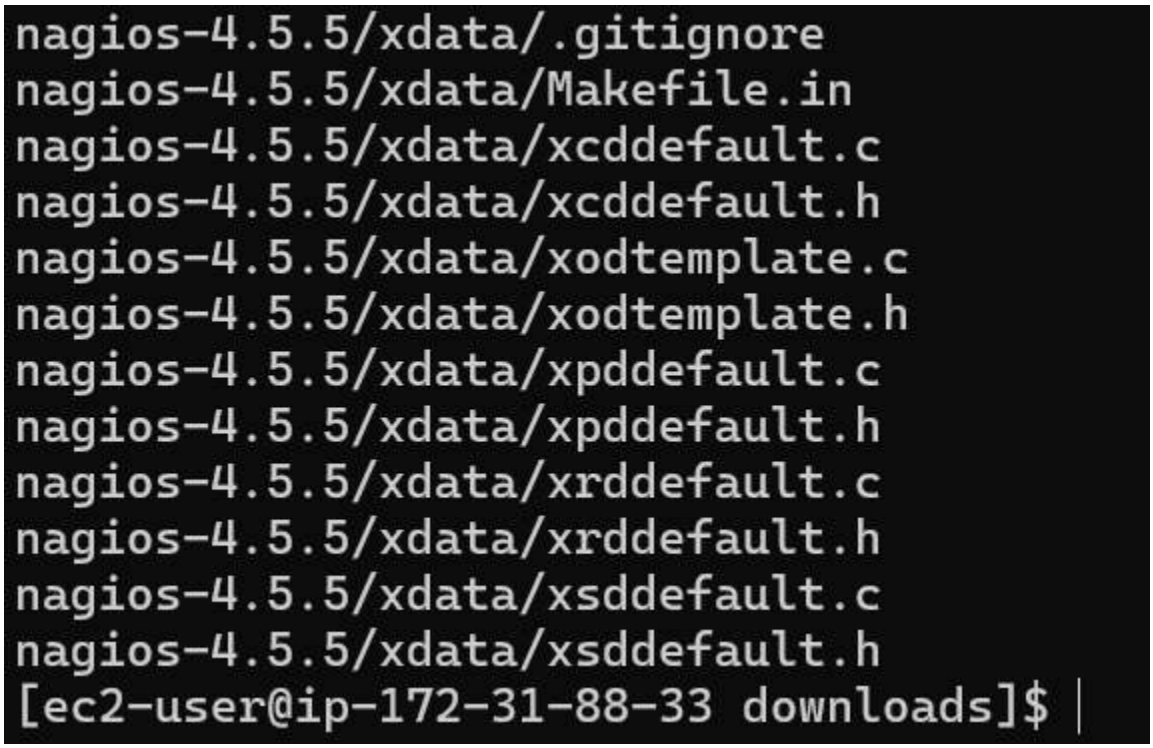
```
[ec2-user@ip-172-31-88-33 downloads]$ wget https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.5.5.tar.gz
--2024-09-29 10:33:56--  https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.5.5.tar.gz
Resolving assets.nagios.com (assets.nagios.com)... 45.79.49.128, 2600:3c00::f03c:92ff:fef7:45ce
Connecting to assets.nagios.com (assets.nagios.com)|45.79.49.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2065473 (2.0M) [application/x-gzip]
Saving to: 'nagios-4.5.5.tar.gz'

nagios-4.5.5.tar.gz          100%[=====================================================>]   1.97M  8.29MB/s    in 0.2s

2024-09-29 10:33:57 (8.29 MB/s) - 'nagios-4.5.5.tar.gz' saved [2065473/2065473]
```

Step 14: wget https://nagios-plugins.org/download/nagios-plugins-2.4.11.tar.gz
The above command installs the latest version of nagios-plugins.

```
[ec2-user@ip-172-31-88-33 downloads]$ wget https://nagios-plugins.org/download/nagios-plugins-2.4.11.tar.gz
--2024-09-29 10:34:29--  https://nagios-plugins.org/download/nagios-plugins-2.4.11.tar.gz
Resolving nagios-plugins.org (nagios-plugins.org)... 45.56.123.251
Connecting to nagios-plugins.org (nagios-plugins.org)|45.56.123.251|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2753049 (2.6M) [application/x-gzip]
Saving to: 'nagios-plugins-2.4.11.tar.gz'

nagios-plugins-2.4.11.tar.gz  100%[=====================================================>]   2.62M  9.99MB/s    in 0.3s

2024-09-29 10:34:30 (9.99 MB/s) - 'nagios-plugins-2.4.11.tar.gz' saved [2753049/2753049]
```

Step 15: tar zxvf nagios-4.5.5.tar.gz
This extracts the nagios-core files into the same directory using the tar command.





Step 16: ./configure --with-command-group=nagcmd
This command ensures that Nagios uses a specific group (in this case, nagcmd) for executing external commands.

But, we encounter an error as we weren't in the correct directory.

Use ‘ls’ command to find the correct directory.

```
[ec2-user@ip-172-31-88-33 downloads]$ ls
nagios-4.5.5  nagios-4.5.5.tar.gz  nagios-plugins-2.4.11.tar.gz
```

Use cd to change directory to the correct directory. Then, run the ‘./configure --with-command-group=nagcmd’ command again.

```
[ec2-user@ip-172-31-88-33 downloads]$ cd nagios-4.5.5
[ec2-user@ip-172-31-88-33 nagios-4.5.5]$ ./configure --with-command-group=nagcmd
checking for a BSD-compatible install... /usr/bin/install -c
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether the compiler supports GNU C... yes
checking whether gcc accepts -g... yes
checking for gcc option to enable C11 features... none needed
checking whether make sets $(MAKE)... yes
checking whether ln -s works... yes
checking for strip... /usr/bin/strip
checking for sys/wait.h that is POSIX.1 compatible... yes
checking for stdio.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for inttypes.h... yes
```

```
checking for unsetenv... yes
checking for type of socket size... size_t
checking for Kerberos include files... configure: WARNING: could not find include files
checking for pkg-config... pkg-config
checking for SSL headers... configure: error: Cannot find ssl headers
```

Another error occurs which says that ssl headers cannot be found.

To fix the above error, run the ‘sudo yum install openssl-devel’ command.

```
[ec2-user@ip-172-31-88-33 nagios-4.5.5]$ sudo yum install openssl-devel
Last metadata expiration check: 0:35:17 ago on Sun Sep 29 10:22:03 2024.
Dependencies resolved.
================================================================================
 Package              Architecture        Version                    Repository          Size
================================================================================
Installing:
 openssl-devel        x86_64              1:3.0.8-1.amzn2023.0.14    amazonlinux         3.0 M

Transaction Summary
================================================================================
Install  1 Package

Total download size: 3.0 M
Installed size: 4.7 M
Is this ok [y/N]: y
Downloading Packages:
openssl-devel-3.0.8-1.amzn2023.0.14.x86_64.rpm                  26 MB/s | 3.0 MB     00:00
--------------------------------------------------------------------------------
Total                                                           16 MB/s | 3.0 MB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : openssl-devel-1:3.0.8-1.amzn2023.0.14.x86_64           1/1
  Running scriptlet: openssl-devel-1:3.0.8-1.amzn2023.0.14.x86_64           1/1
  Verifying        : openssl-devel-1:3.0.8-1.amzn2023.0.14.x86_64           1/1
```

```
Installed:
  openssl-devel-1:3.0.8-1.amzn2023.0.14.x86_64

Complete!
```

Then, run the ‘./configure --with-command-group=nagcmd’ command again.

```
[ec2-user@ip-172-31-88-33 nagios-4.5.5]$ ./configure --with-command-group=nagcmd
checking for a BSD-compatible install... /usr/bin/install -c
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether the compiler supports GNU C... yes
checking whether gcc accepts -g... yes
checking for gcc option to enable C11 features... none needed
checking whether make sets $(MAKE)... yes
checking whether ln -s works... yes
checking for strip... /usr/bin/strip
checking for sys/wait.h that is POSIX.1 compatible... yes
checking for stdio.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
```

```
General Options:
-------------------------
        Nagios executable:  nagios
        Nagios user/group:  nagios,nagios
       Command user/group:  nagios,nagcmd
             Event Broker:  yes
        Install ${prefix}:  /usr/local/nagios
    Install ${includedir}:  /usr/local/nagios/include/nagios
                Lock file:  /run/nagios.lock
   Check result directory:  /usr/local/nagios/var/spool/checkresults
           Init directory:  /lib/systemd/system
  Apache conf.d directory:  /etc/httpd/conf.d
             Mail program:  /bin/mail
                  Host OS:  linux-gnu
          IOBroker Method:  epoll

Web Interface Options:
------------------------
                 HTML URL:  http://localhost/nagios/
                  CGI URL:  http://localhost/nagios/cgi-bin/
Traceroute (used by WAP):  /usr/bin/traceroute


Review the options above for accuracy.  If they look okay,
type 'make all' to compile the main program and CGIs.
```

Step 17: Next, we must compile all components of this software according to the instructions in the Makefile. To do so, use the following command:

make all

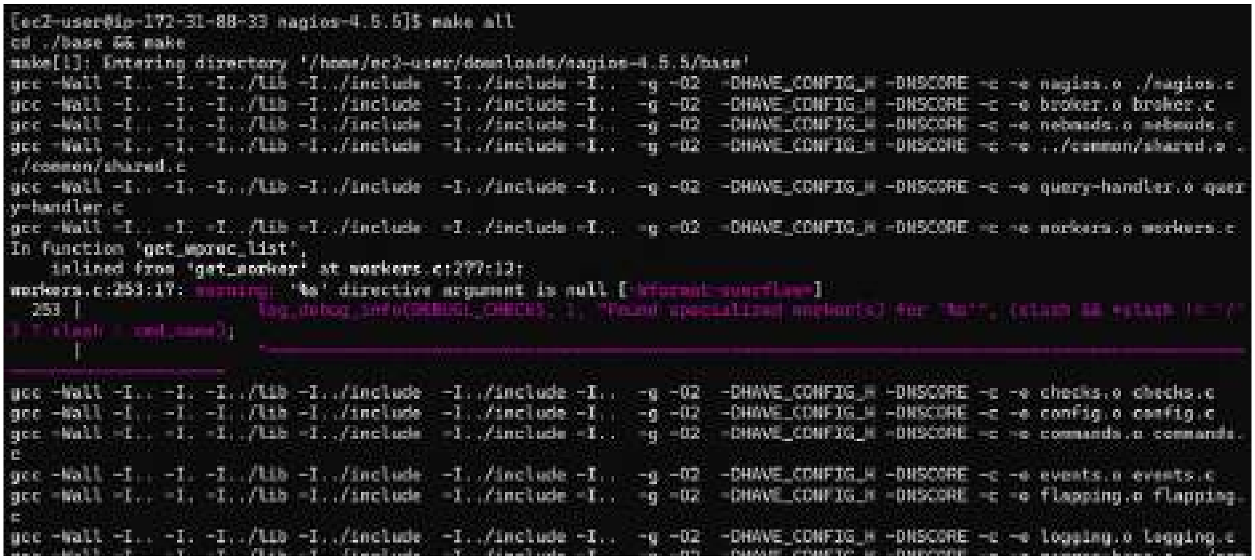
Then,

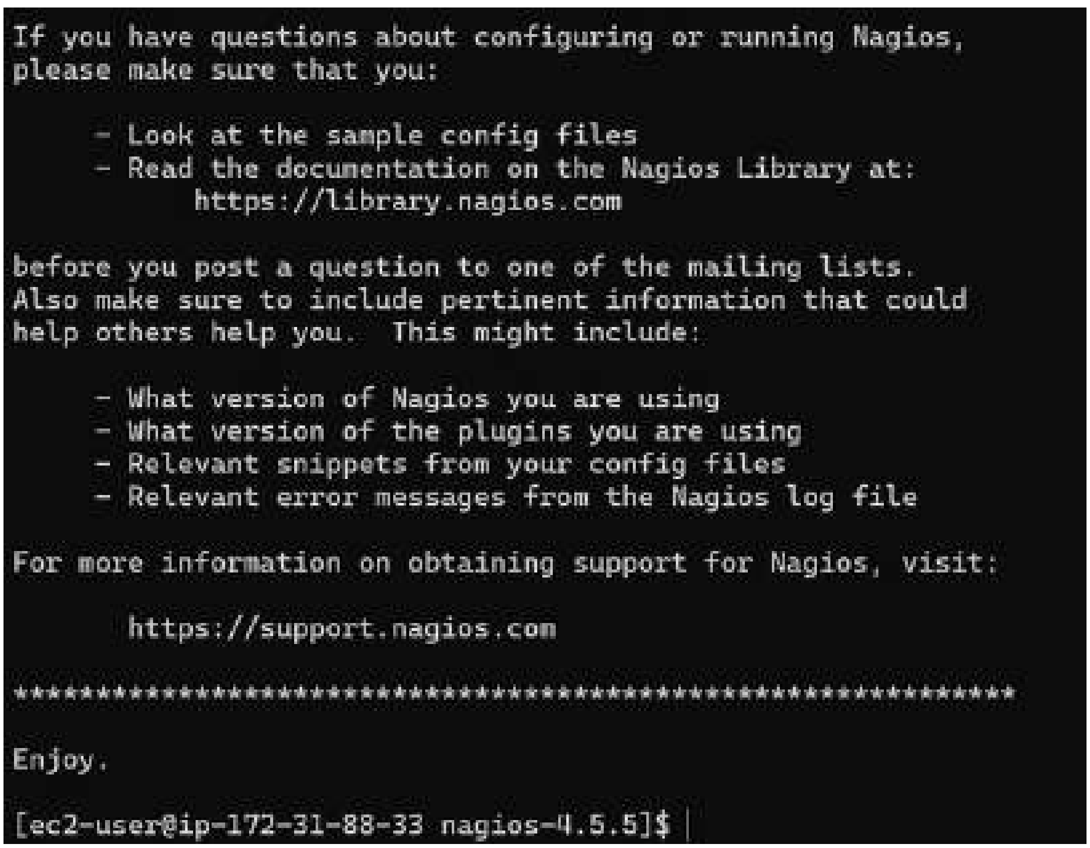
sudo make install

sudo make install-init

sudo make install-config

sudo make install-commandmode





```
[ec2-user@ip-172-31-88-33 nagios-4.5.5]$ sudo make install
sudo make install-init
sudo make install-config
sudo make install-commandmode
cd ./base && make install
make[1]: Entering directory '/home/ec2-user/downloads/nagios-4.5.5/base'
/usr/bin/install -c -m 775 -o nagios -g nagios -d /usr/local/nagios/bin
/usr/bin/install -c -s -m 774 -o nagios -g nagios nagios /usr/local/nagios/bin
/usr/bin/install -c -s -m 774 -o nagios -g nagios nagiostats /usr/local/nagios/bin
make[1]: Leaving directory '/home/ec2-user/downloads/nagios-4.5.5/base'
cd ./cgi && make install
make[1]: Entering directory '/home/ec2-user/downloads/nagios-4.5.5/cgi'
make install-basic
make[2]: Entering directory '/home/ec2-user/downloads/nagios-4.5.5/cgi'
/usr/bin/install -c -m 775 -o nagios -g nagios -d /usr/local/nagios/sbin
```

```
/usr/bin/install -c -b -m 664 -o nagios -g nagios sample-config/template-object/switch.cfg /usr/local/nagios/etc/objects
/switch.cfg

*** Config files installed ***

Remember, these are *SAMPLE* config files.  You'll need to read
the documentation for more information on how to actually define
services, hosts, etc. to fit your particular needs.

/usr/bin/install -c -m 775 -o nagios -g nagcmd -d /usr/local/nagios/var/rw
chmod g+s /usr/local/nagios/var/rw

*** External command directory configured ***
```

Step 18: We need to update the email linked with this server to our email for it to send notifications (if any needed).
sudo nano /usr/local/nagios/etc/objects/contacts.cfg

```
  GNU nano 5.8                    /usr/local/nagios/etc/objects/contacts.cfg                    Modified


###############################################################################
#
# CONTACTS
#
###############################################################################

# Just one contact defined by default - the Nagios admin (that's you)
# This contact definition inherits a lot of default values from the
# 'generic-contact' template which is defined elsewhere.

define contact {

    contact_name            nagiosadmin             ; Short name of user
    use                     generic-contact         ; Inherit default values from generic-contact template (defined abo
    alias                   Nagios Admin            ; Full name of user
    email                   2022.anish.kulkarni@ves.ac.in ; <<***** CHANGE THIS TO YOUR EMAIL ADDRESS ******
}



###############################################################################
#
# CONTACT GROUPS
#

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location  M-U Undo      M-A Set Mark
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line M-E Redo      M-6 Copy
```

In the email section, enter your email address. Then, ‘Write out’ your file and ‘Exit’.

Step 19: sudo make install-webconf
This installs the necessary configuration files for the Nagios web interface.

```
[ec2-user@ip-172-31-88-33 nagios-4.5.5]$ sudo make install-webconf
/usr/bin/install -c -m 644 sample-config/httpd.conf /etc/httpd/conf.d/nagios.conf
if [ 0 -eq 1 ]; then \
        ln -s /etc/httpd/conf.d/nagios.conf /etc/apache2/sites-enabled/nagios.conf; \
fi

*** Nagios/Apache conf file installed ***
```

Step 20: sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
This creates a user named ‘nagiosadmin’ to access the nagios web interface. Create a password and keep it in mind as it will be required in the future steps.

```
[ec2-user@ip-172-31-88-33 nagios-4.5.5]$ sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
New password:
Re-type new password:
Adding password for user nagiosadmin
```

Step 21: Restart the apache server to apply all the recent configurations.
sudo service httpd restart

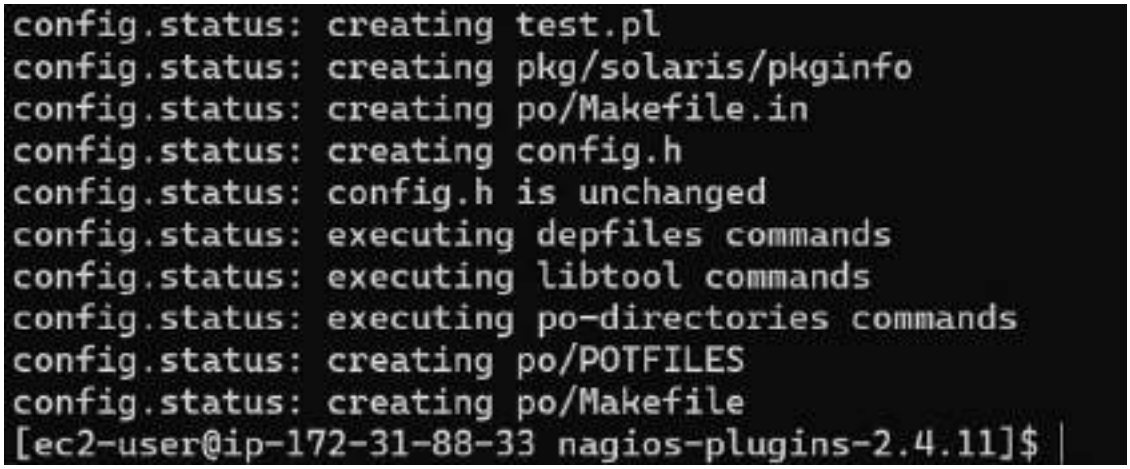
```
[ec2-user@ip-172-31-88-33 nagios-4.5.5]$ sudo service httpd restart
Redirecting to /bin/systemctl restart httpd.service
```

Step 22: cd ~/downloads
tar zxvf nagios-plugins-2.4.11.tar.gz
This changes the directory to the ‘downloads’ directory and extracts the files for nagios-plugins.

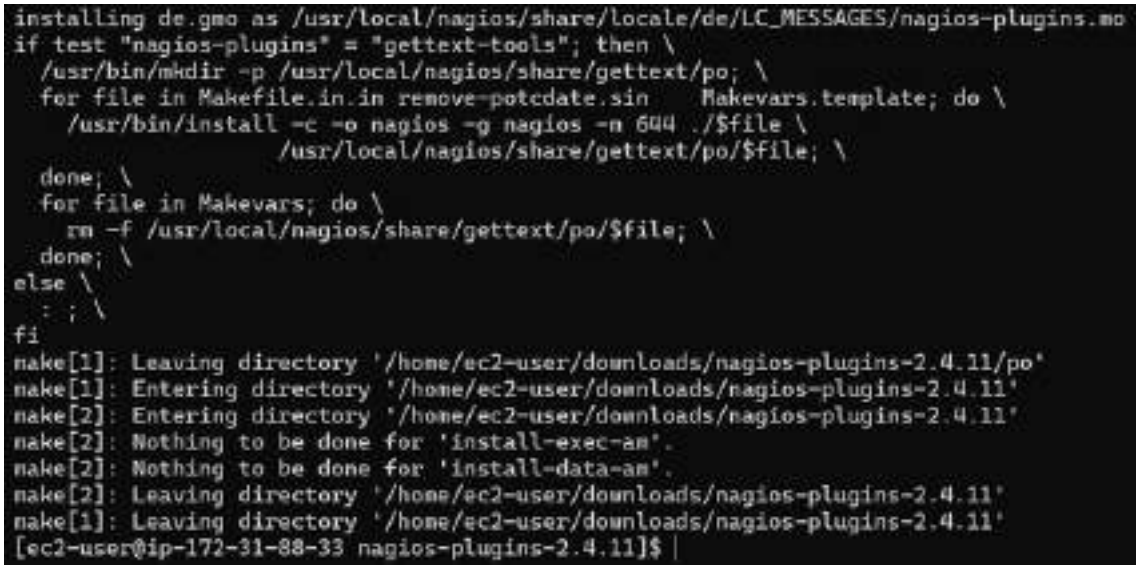
```
[ec2-user@ip-172-31-88-33 downloads]$ tar zxvf nagios-plugins-2.4.11.tar.gz
nagios-plugins-2.4.11/
nagios-plugins-2.4.11/build-aux/
nagios-plugins-2.4.11/build-aux/compile
nagios-plugins-2.4.11/build-aux/config.guess
nagios-plugins-2.4.11/build-aux/config.rpath
nagios-plugins-2.4.11/build-aux/config.sub
nagios-plugins-2.4.11/build-aux/install-sh
nagios-plugins-2.4.11/build-aux/ltmain.sh
nagios-plugins-2.4.11/build-aux/missing
nagios-plugins-2.4.11/build-aux/mkinstalldirs
nagios-plugins-2.4.11/build-aux/depcomp
nagios-plugins-2.4.11/build-aux/snippet/
nagios-plugins-2.4.11/build-aux/snippet/_Noreturn.h
nagios-plugins-2.4.11/build-aux/snippet/arg-nonnull.h
nagios-plugins-2.4.11/build-aux/snippet/c++defs.h
nagios-plugins-2.4.11/build-aux/snippet/warn-on-use.h
nagios-plugins-2.4.11/build-aux/test-driver
nagios-plugins-2.4.11/config_test/
nagios-plugins-2.4.11/config_test/Makefile
```

```
nagios-plugins-2.4.11/po/
nagios-plugins-2.4.11/po/Makefile.in.in
nagios-plugins-2.4.11/po/remove-potcdate.sin
nagios-plugins-2.4.11/po/Makevars
nagios-plugins-2.4.11/po/POTFILES.in
nagios-plugins-2.4.11/po/fr.po
nagios-plugins-2.4.11/po/de.po
nagios-plugins-2.4.11/po/fr.gmo
nagios-plugins-2.4.11/po/de.gmo
nagios-plugins-2.4.11/po/nagios-plugins.pot
nagios-plugins-2.4.11/po/stamp-po
nagios-plugins-2.4.11/po/ChangeLog
nagios-plugins-2.4.11/po/LINGUAS
nagios-plugins-2.4.11/release
```

Step 23: cd nagios-plugins-2.4.11
./configure --with-nagios-user=nagios --with-nagios-group=nagios
This installs the configurations for the nagios-plugins files.

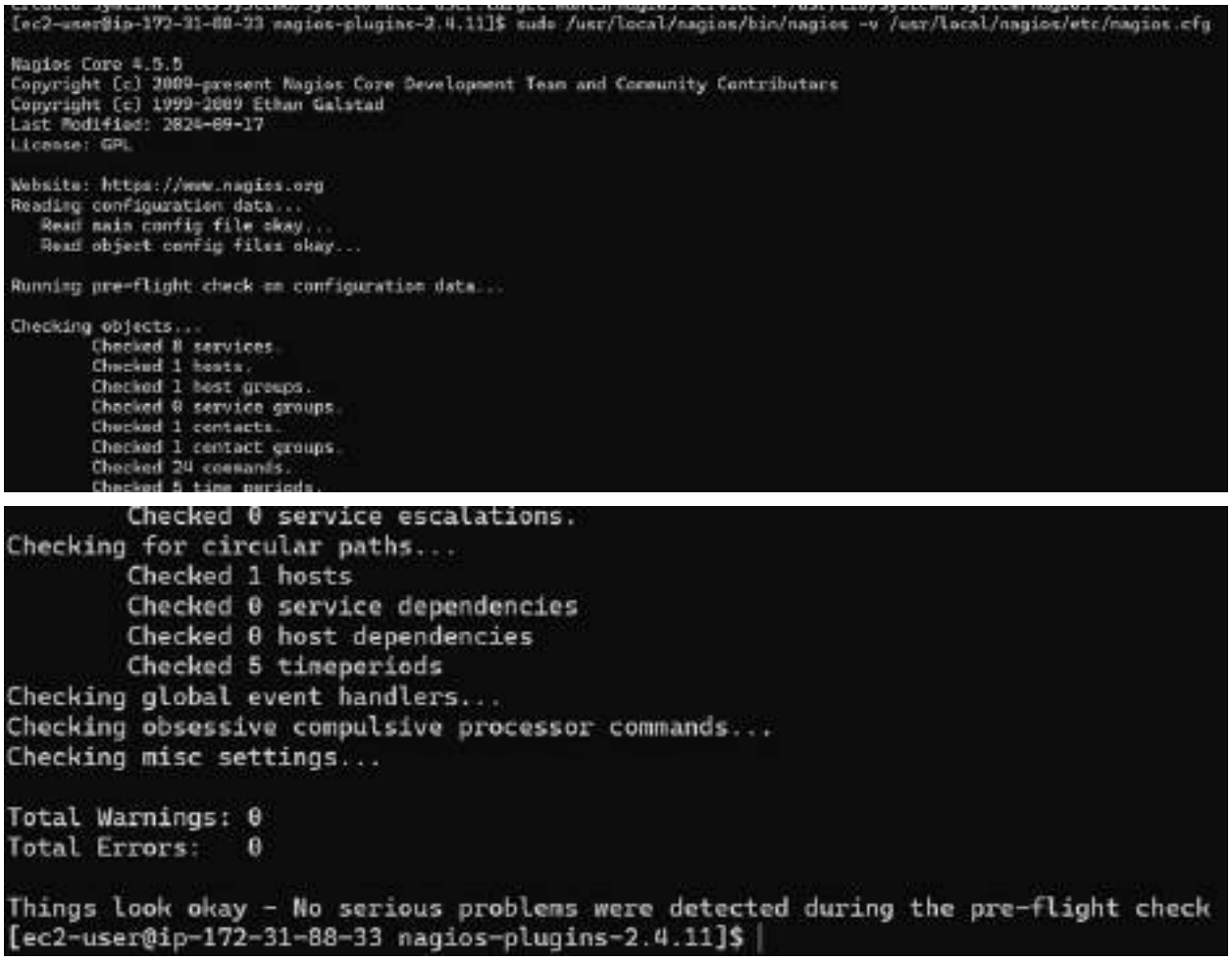


Step 24: Next, we must compile all components of this software according to the instructions in the Makefile. To do so, use the following commands:
make
sudo make install

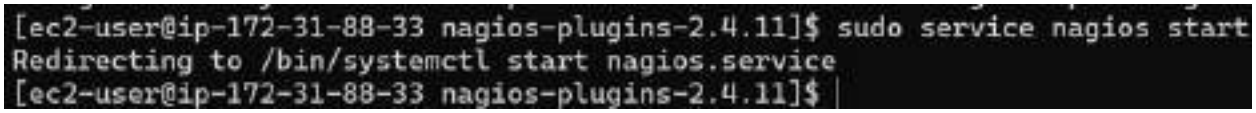


Step 25: sudo chkconfig --add nagios
sudo chkconfig nagios on
This registers the Nagios service with the system ensuring that it can manage the server status.

Step 26: sudo /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
This command checks and verifies that the sample configuration files has no errors.



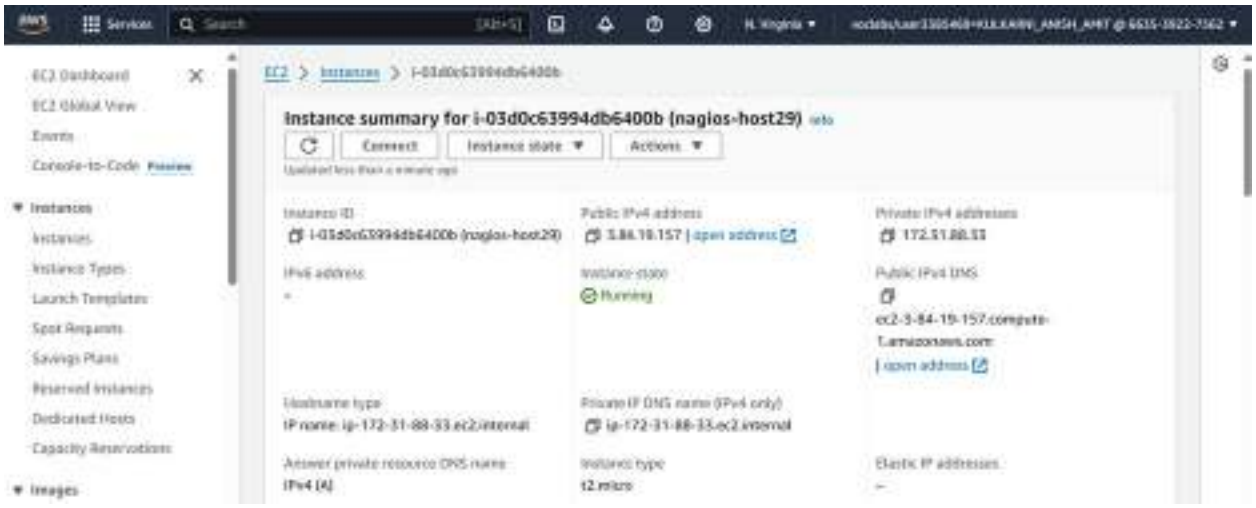
Step 27: sudo service nagios start
This starts the Nagios service.



Step 28: sudo systemctl status nagios
This checks the status of Nagios. Ensure that it is ‘active(running)’.

```
[ec2-user@ip-172-31-88-33 nagios-plugins-2.4.11]$ sudo systemctl status nagios
● nagios.service - Nagios Core 4.5.5
     Loaded: loaded (/usr/lib/systemd/system/nagios.service; enabled; preset: disabled)
     Active: active (running) since Sun 2024-09-29 11:25:40 UTC; 2min 3s ago
       Docs: https://www.nagios.org/documentation
    Process: 67487 ExecStartPre=/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg (code=exited, status=0
    Process: 67488 ExecStart=/usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg (code=exited, status=0/SU
   Main PID: 67489 (nagios)
      Tasks: 6 (limit: 1112)
     Memory: 6.1M
        CPU: 122ms
     CGroup: /system.slice/nagios.service
             ├─67489 /usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg
             ├─67490 /usr/local/nagios/bin/nagios --worker /usr/local/nagios/var/rw/nagios.qh
             ├─67491 /usr/local/nagios/bin/nagios --worker /usr/local/nagios/var/rw/nagios.qh
             ├─67492 /usr/local/nagios/bin/nagios --worker /usr/local/nagios/var/rw/nagios.qh
             ├─67493 /usr/local/nagios/bin/nagios --worker /usr/local/nagios/var/rw/nagios.qh
             └─67494 /usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg

Sep 29 11:25:40 ip-172-31-88-33.ec2.internal nagios[67489]: qh: core query handler registered
Sep 29 11:25:40 ip-172-31-88-33.ec2.internal nagios[67489]: qh: echo service query handler registered
Sep 29 11:25:40 ip-172-31-88-33.ec2.internal nagios[67489]: qh: help for the query handler registered
Sep 29 11:25:40 ip-172-31-88-33.ec2.internal nagios[67489]: wproc: Successfully registered manager as @wproc with query
Sep 29 11:25:40 ip-172-31-88-33.ec2.internal nagios[67489]: wproc: Registry request: name=Core Worker 67492;pid=67492
Sep 29 11:25:40 ip-172-31-88-33.ec2.internal nagios[67489]: wproc: Registry request: name=Core Worker 67493;pid=67493
Sep 29 11:25:40 ip-172-31-88-33.ec2.internal nagios[67489]: wproc: Registry request: name=Core Worker 67491;pid=67491
Sep 29 11:25:40 ip-172-31-88-33.ec2.internal nagios[67489]: wproc: Registry request: name=Core Worker 67490;pid=67490
Sep 29 11:25:41 ip-172-31-88-33.ec2.internal nagios[67489]: Successfully launched command file worker with pid 67494
Sep 29 11:27:32 ip-172-31-88-33.ec2.internal nagios[67489]: SERVICE ALERT: localhost;HTTP;WARNING;SOFT;1;HTTP WARNING: 
lines 1-28/28 (END)
```

Step 29: Navigate to your EC2 instance and copy the public IPv4 address.



Step 30: In the address bar, enter ‘http://<publicipaddress>/nagios’.

The above page is visible.

Conclusion: In the above experiment, we learned how to install and configure Nagios Core, Nagios Plugins and NRPE (Nagios Remote Plugin Executor) on Linux Machine. We created an EC2 Linux instance with the required security rules. Then, we installed the latest versions of nagios-core and nagios-plugins and configured them to ensure that they contained no errors. Once the setup was complete, we hosted the Nagios server and accessed the Nagios dashboard by pasting the public IPv4 address of our instance in the browser.

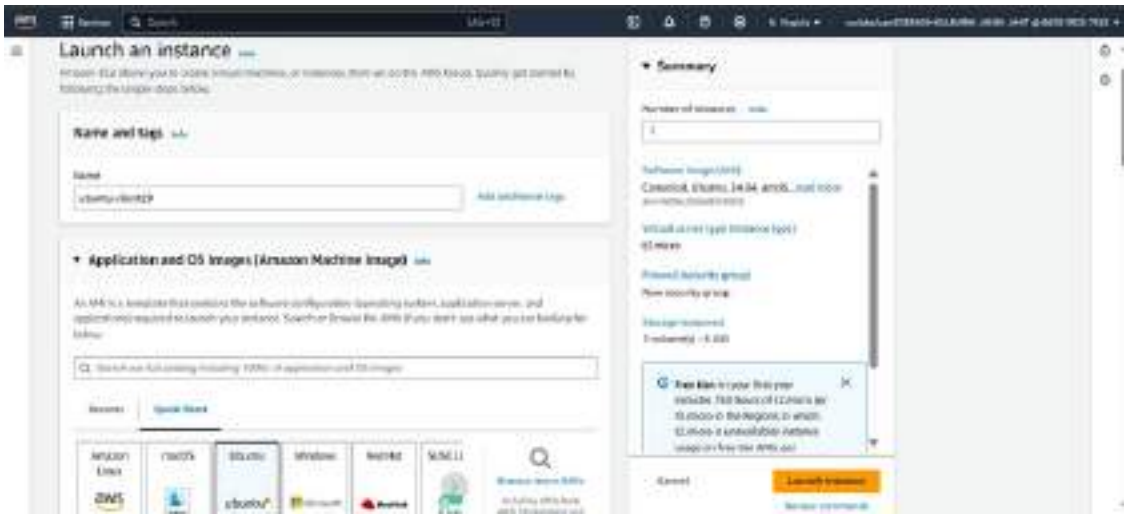
Experiment 10

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

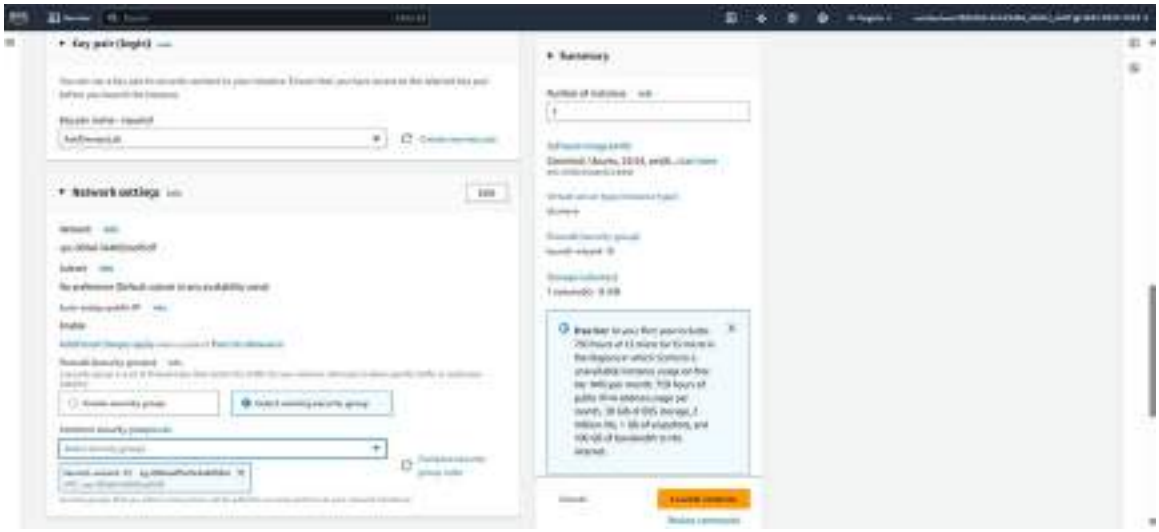
Prerequisites: An Amazon Linux instance with nagios (nagios-server) is already set up.

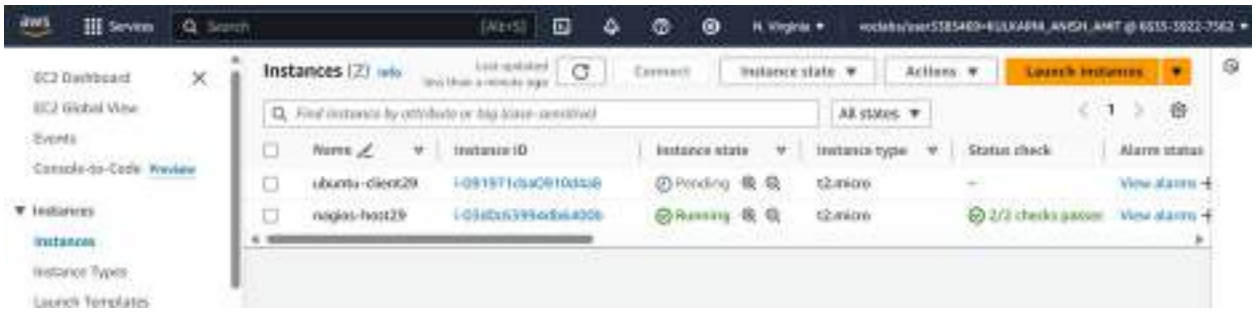
Steps:

Step 1: Navigate to EC2 on the AWS console using the ‘Services’ section and click on ‘Create instance’. Give your instance a name and choose ‘Ubuntu’ as the instance type.



Ensure that you choose the same key pair and security group for the Ubuntu client instance as you did for the Nagios host instance. Then, click on ‘Create instance’.





Your Ubuntu client instance gets created along with the Nagios host instance.

Step 2: Click on the instance ID of your nagios-server instance and click on ‘Connect’. Then, click on ‘SSH client’ and copy the command under ‘Example’. Then, open the terminal in the folder where the .pem file for your instance’s key pair is located and paste the SSH command that you just copied. This connects your instance to your local terminal using SSH.

Step 3: ps -ef | grep nagios
Run the above command on the nagios-host instance. This verifies whether the nagios service is running or not.

```
[root@ip-172-31-88-33 ec2-user]# ps -ef | grep nagios
nagios     67489       1  0 11:25 ?        00:00:00 /usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg
nagios     67490   67489  0 11:25 ?        00:00:00 /usr/local/nagios/bin/nagios --worker /usr/local/nagios/var/rw/nagio
s.qh
nagios     67491   67489  0 11:25 ?        00:00:00 /usr/local/nagios/bin/nagios --worker /usr/local/nagios/var/rw/nagio
s.qh
nagios     67492   67489  0 11:25 ?        00:00:00 /usr/local/nagios/bin/nagios --worker /usr/local/nagios/var/rw/nagio
s.qh
nagios     67493   67489  0 11:25 ?        00:00:00 /usr/local/nagios/bin/nagios --worker /usr/local/nagios/var/rw/nagio
s.qh
nagios     67494   67489  0 11:25 ?        00:00:00 /usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg
root       68807   68853  0 11:51 pts/1    00:00:00 grep --color=auto nagios
[root@ip-172-31-88-33 ec2-user]#
```

Step 4: sudo su
mkdir -p /usr/local/nagios/etc/objects/monitorhosts
mkdir -p /usr/local/nagios/etc/objects/monitorhosts/linuxhosts
This makes you the root user and creates two folders with the above paths.

```
[root@ip-172-31-88-33 ec2-user]# sudo su
mkdir /usr/local/nagios/etc/objects/monitorhosts
mkdir /usr/local/nagios/etc/objects/monitorhosts/linuxhosts
[root@ip-172-31-88-33 ec2-user]#
```

Step 5: We need to create a config file in this folder. So, copy the contents of the existing localhost config to the new file 'linuxserver.cfg'.
cp /usr/local/nagios/etc/objects/localhost.cfg
/usr/local/nagios/etc/objects/monitorhosts/linuxhosts/linuxserver.cfg

```
[root@ip-172-31-88-33 ec2-user]# cp /usr/local/nagios/etc/objects/localhost.cfg /usr/local/nagios/etc/objects/monitorhos
ts/linuxhosts/linuxserver.cfg
```

Step 6: We need to make some changes in this config file. Open it using nano editor:-
nano /usr/local/nagios/etc/objects/monitorhosts/linuxhosts/linuxserver.cfg

1. Changehostnameandaliasfrom'hostname'to'linuxserver'.
2. Change address to the public ip address of the ubuntu-client instance.

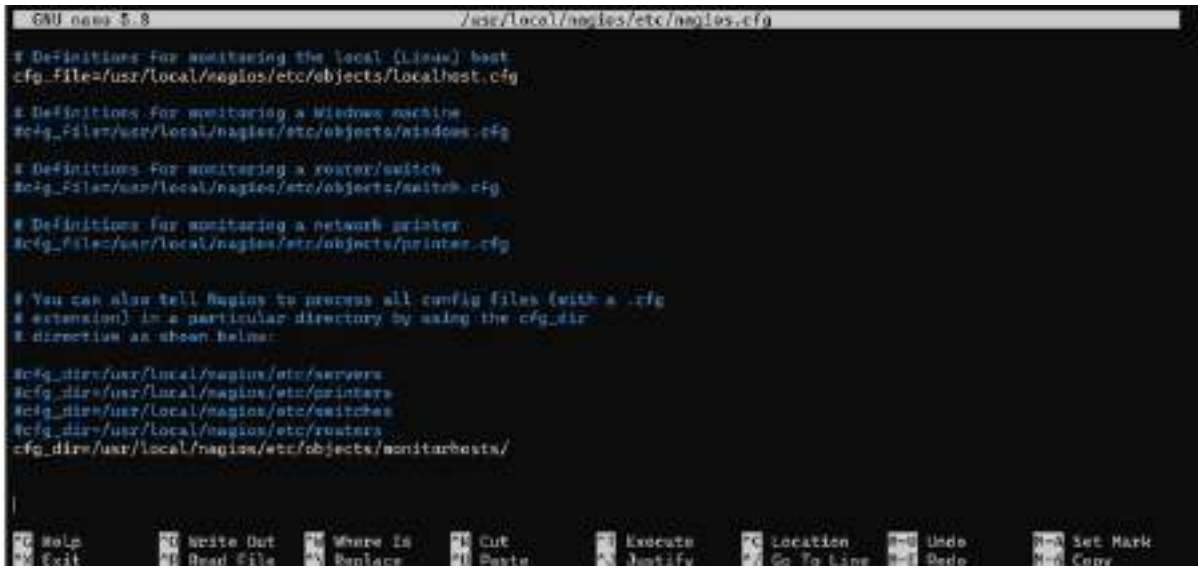
```
##############################################################################
#
# HOST DEFINITION
#
##############################################################################

# Define a host for the local machine

define host {

    use                     linux-server            ; Name of host template to use
                                                    ; This host definition will inherit all variables that are defined
                                                    ; in (or inherited by) the linux-server host template definition.
    host_name               linuxserver
    alias                   linuxserver
    address                 52.91.181.68
}


##############################################################################
#
```

Change hostgroup_name to 'linux-servers1'.

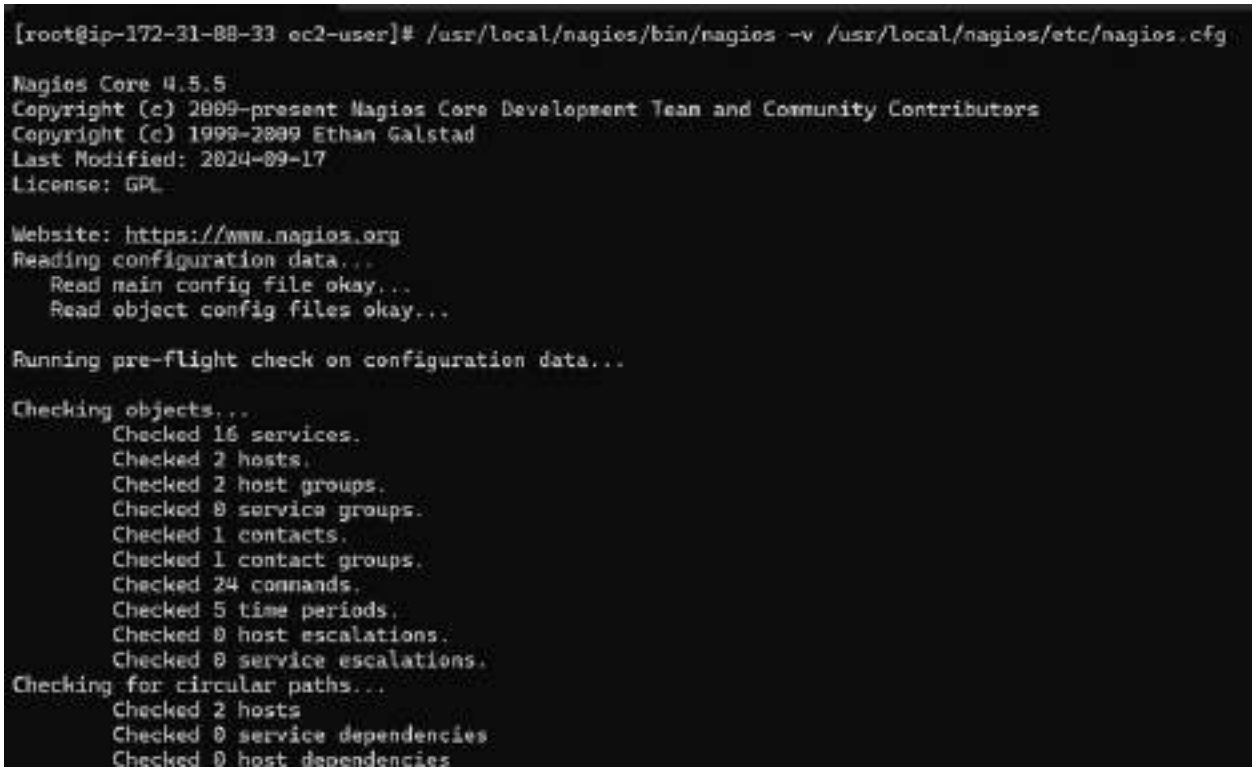
```
define hostgroup {

    hostgroup_name          linux-servers1          ; The name of the hostgroup
    alias                   Linux Servers           ; Long name of the group
    members                 linuxserver             ; Comma separated list of hosts that belong to this group
}
```

Change all the subsequent occurrences of hostname in the file from 'localhost' to linuxserver'.

Step 7: Open the Nagios config file using the following command:
nano /usr/local/nagios/etc/nagios.cfg
Then, add the following line to the config file:
cfg_dir=/usr/local/nagios/etc/objects/monitorhosts/



Step 8: Now we verify the configuration files and check that they contain no errors using the following command:
/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg



```
        Checked 0 host dependencies
        Checked 5 timeperiods
Checking global event handlers...
Checking obsessive compulsive processor commands...
Checking misc settings...

Total Warnings: 0
Total Errors:   0

Things look okay - No serious problems were detected during the pre-flight check
[root@ip-172-31-88-33 ec2-user]#
```

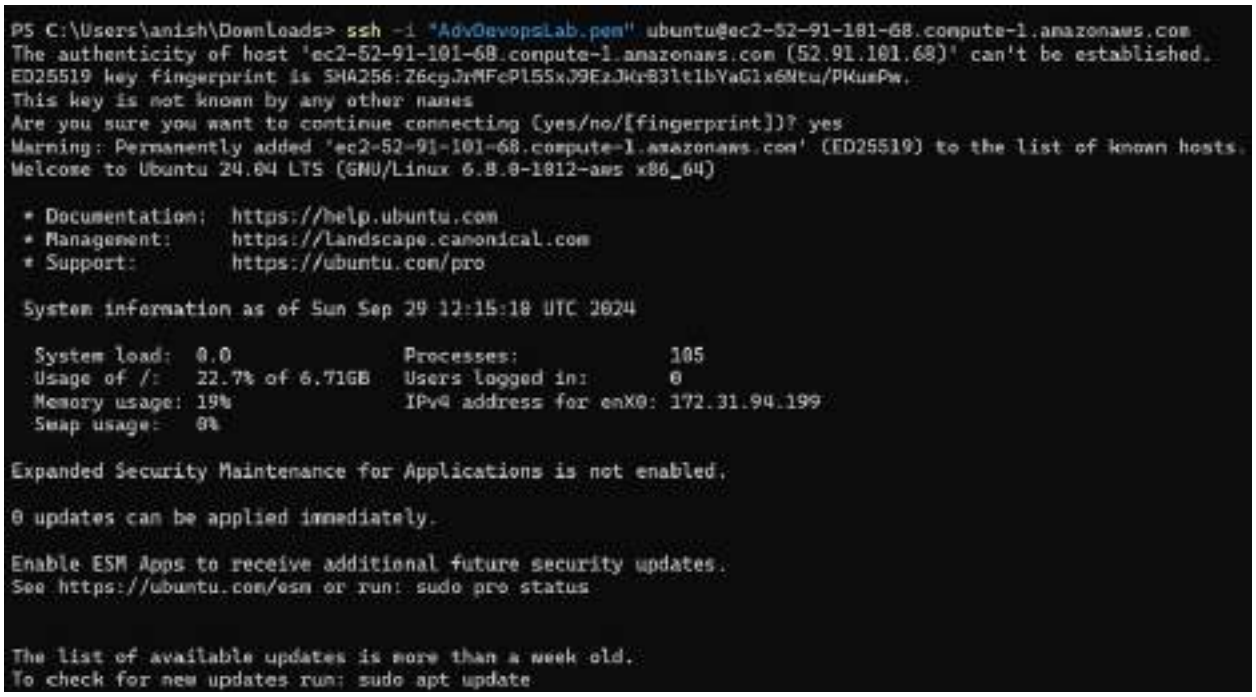
Step 9: Once the files are verified and it is confirmed that there are no errors, we must restart the server.
service nagios restart

```
[root@ip-172-31-88-33 ec2-user]# service nagios restart
Redirecting to /bin/systemctl restart nagios.service
```

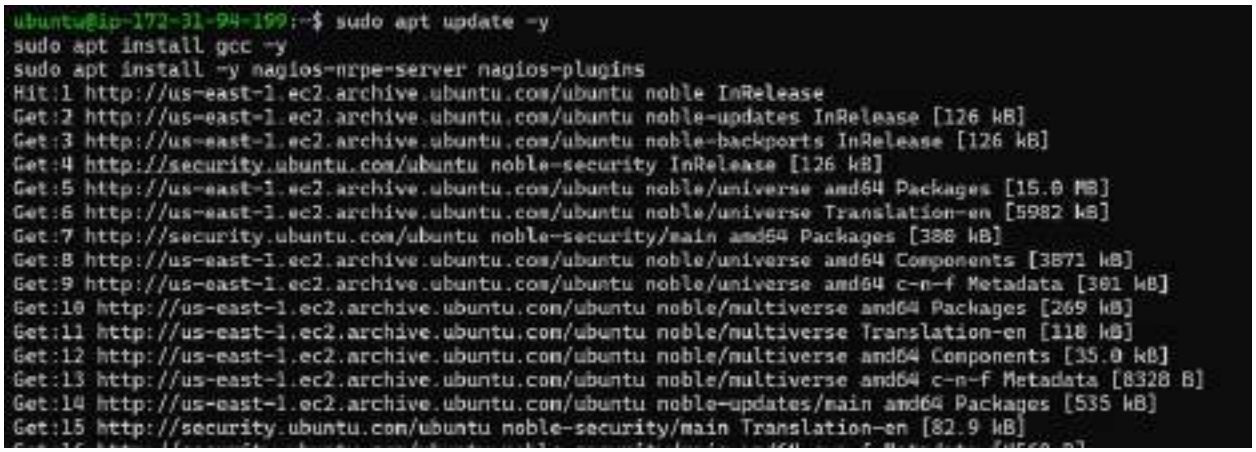
Step 10: systemctl status nagios
Using the above command, we check the status of the nagios server and ensure that it is active (running).

```
[root@ip-172-31-88-33 ec2-user]# systemctl status nagios
● nagios.service - Nagios Core 4.5.5
     Loaded: loaded (/usr/lib/systemd/system/nagios.service; enabled; preset: disabled)
     Active: active (running) since Sun 2024-09-29 12:11:40 UTC; 1min 12s ago
       Docs: https://www.nagios.org/documentation
    Process: 70244 ExecStartPre=/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg (code=exited, status=0
    Process: 70245 ExecStart=/usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg (code=exited, status=0/SU
   Main PID: 70246 (nagios)
      Tasks: 6 (limit: 1112)
     Memory: 4.0M
        CPU: 38ms
     CGroup: /system.slice/nagios.service
             ├─70246 /usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg
             ├─70247 /usr/local/nagios/bin/nagios --worker /usr/local/nagios/var/rw/nagios.qh
             ├─70248 /usr/local/nagios/bin/nagios --worker /usr/local/nagios/var/rw/nagios.qh
             ├─70249 /usr/local/nagios/bin/nagios --worker /usr/local/nagios/var/rw/nagios.qh
             ├─70250 /usr/local/nagios/bin/nagios --worker /usr/local/nagios/var/rw/nagios.qh
             └─70251 /usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg

Sep 29 12:11:40 ip-172-31-88-33.ec2.internal nagios[70246]: qh: Socket '/usr/local/nagios/var/rw/nagios.qh' successfull
Sep 29 12:11:40 ip-172-31-88-33.ec2.internal nagios[70246]: qh: core query handler registered
Sep 29 12:11:40 ip-172-31-88-33.ec2.internal nagios[70246]: qh: echo service query handler registered
Sep 29 12:11:40 ip-172-31-88-33.ec2.internal nagios[70246]: qh: help for the query handler registered
Sep 29 12:11:40 ip-172-31-88-33.ec2.internal nagios[70246]: wproc: Successfully registered manager as @wproc with query
Sep 29 12:11:40 ip-172-31-88-33.ec2.internal nagios[70246]: wproc: Registry request: name=Core Worker 70250;pid=70250
Sep 29 12:11:40 ip-172-31-88-33.ec2.internal nagios[70246]: wproc: Registry request: name=Core Worker 70249;pid=70249
Sep 29 12:11:40 ip-172-31-88-33.ec2.internal nagios[70246]: wproc: Registry request: name=Core Worker 70248;pid=70248
Sep 29 12:11:40 ip-172-31-88-33.ec2.internal nagios[70246]: wproc: Registry request: name=Core Worker 70247;pid=70247
Sep 29 12:11:40 ip-172-31-88-33.ec2.internal nagios[70246]: Successfully launched command file worker with pid 70251
```

Step 11: Connect your ubuntu-client instance to your local terminal using SSH in the same way as you connected the nagios-host instance to your local terminal using SSH (follow Step 2)



Step 12: On your ubuntu-client instance, run the following commands:-
sudo apt update -y
sudo apt install gcc -y
sudo apt install -y nagios-nrpe-server nagios-plugins
The above commands check for any new updates and then install gcc, Nagios NRPE server and Nagios plugins.



```
Scanning candidates...
Scanning linux images...

Running kernel seems to be up-to-date.

Restarting services...

Service restarts being deferred:
 /etc/needrestart/restart.d/dbus.service
 systemctl restart getty@tty1.service
 systemctl restart networkd-dispatcher.service
 systemctl restart serial-getty@ttyS0.service
 systemctl restart systemd-logind.service
 systemctl restart unattended-upgrades.service

No containers need to be restarted.

User sessions running outdated binaries:
 ubuntu @ session #4: sshd[1021,1132]
 ubuntu @ user manager service: systemd[1027]

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-94-199:~$
```

Step 13: Run the following command:
sudo nano /etc/nagios/nrpe.cfg
The above command opens the NRPE config file. Here, we need to add the public IP address of our host nagios-host instance to the NRPE configuration file.
Under allowed_hosts, add the nagios-host public IPv4 address.

```
#
# NOTE: This option is ignored if NRPE is running under either inetd or xinetd

nrpe_group=nagios



# ALLOWED HOST ADDRESSES
# This is an optional comma-delimited list of IP address or hostnames
# that are allowed to talk to the NRPE daemon. Network addresses with a bit mask
# (i.e. 192.168.1.0/24) are also supported. Hostname wildcards are not currently
# supported.
#
# Note: The daemon only does rudimentary checking of the client's IP
# address.  I would highly recommend adding entries in your /etc/hosts.allow
# file to allow only the specified host to connect to the port
# you are running this daemon on.
#
# NOTE: This option is ignored if NRPE is running under either inetd or xinetd

allowed_hosts=127.0.0.1,3.84.19.157
```

Step 14: Navigate to the Nagios dashboard. Click on ‘hosts’. We see that linuxserver has been added as a host.

Click on ‘linuxserver’. Here, we can access all information about the ‘linuxserver’ host.

Click on ‘Services’. Here, we can see all the services that are being monitored by ‘linuxserver’.

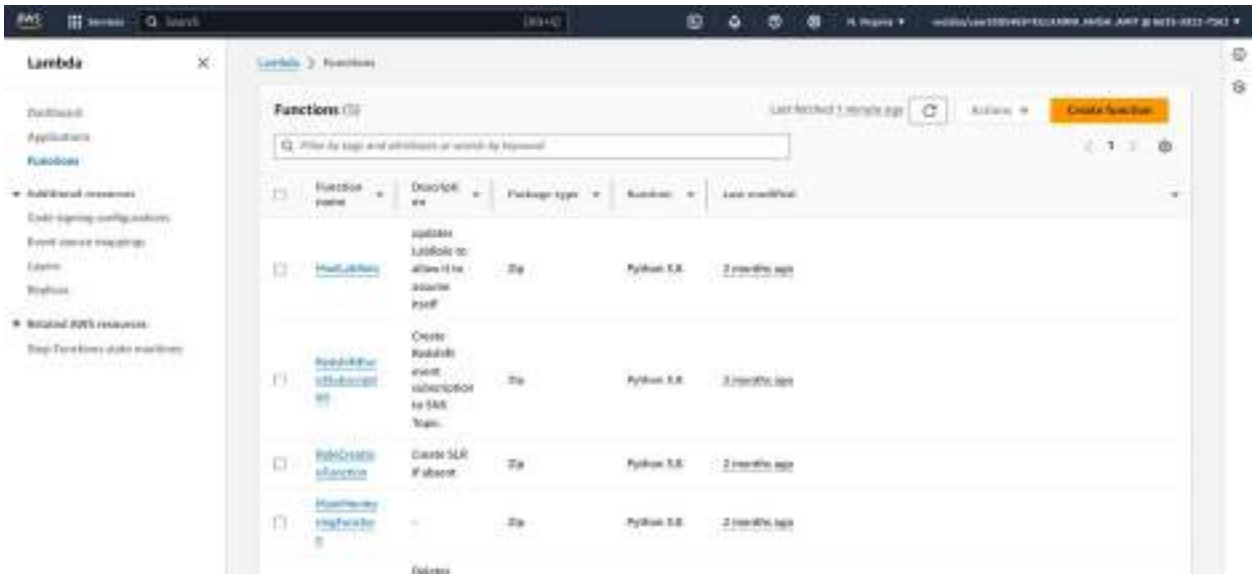
Conclusion: In this experiment, we learned how to perform port, service monitoring, Windows/Linux server monitoring using Nagios. To do so, we needed a nagios-host EC2 Linux instance which was used to host the Nagios server and dashboard. We created an Ubuntu client instance to connect to the host. We set up some configurations on the Linux instance and added the public IP address of the Ubuntu instance in it. We also set up some configurations on the Ubuntu client instance and added the IP address of the Linux server instance in it. Then, we made sure to add the Linux server instance as a ‘allowed host’ for the Ubuntu client instance. After restarting the NRPE server, we can see the ‘linuxserver’ host added.

Experiment 11

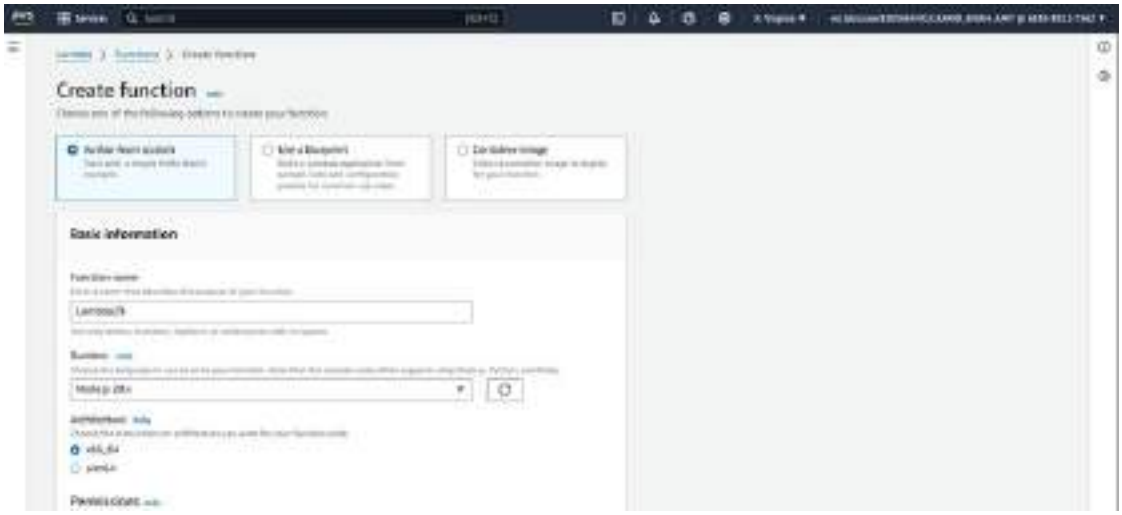
Aim: To understand AWS Lambda, its workflow, various functions and create your first Lambda functions using Python / Java / Nodejs.

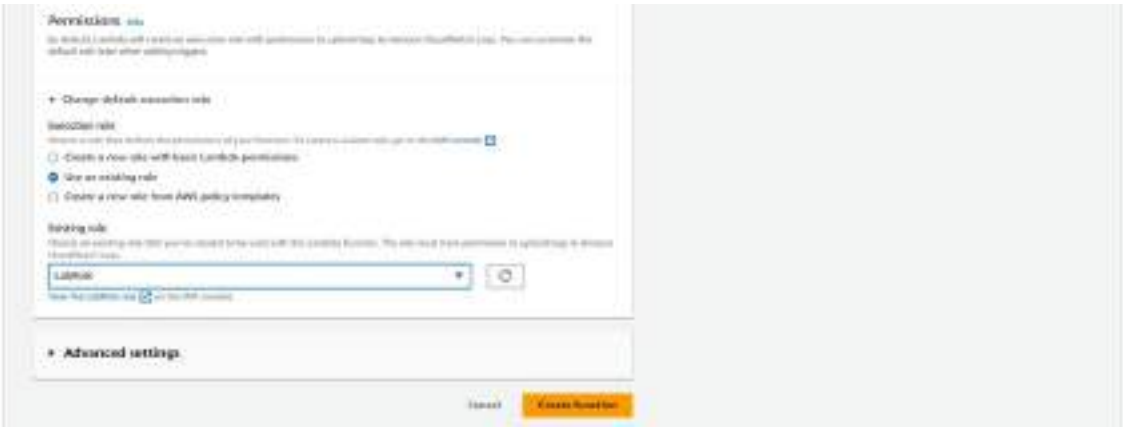
Steps:

Step 1: On your AWS console, click on ‘Lambda’ in the services section and click on ‘Create function’.



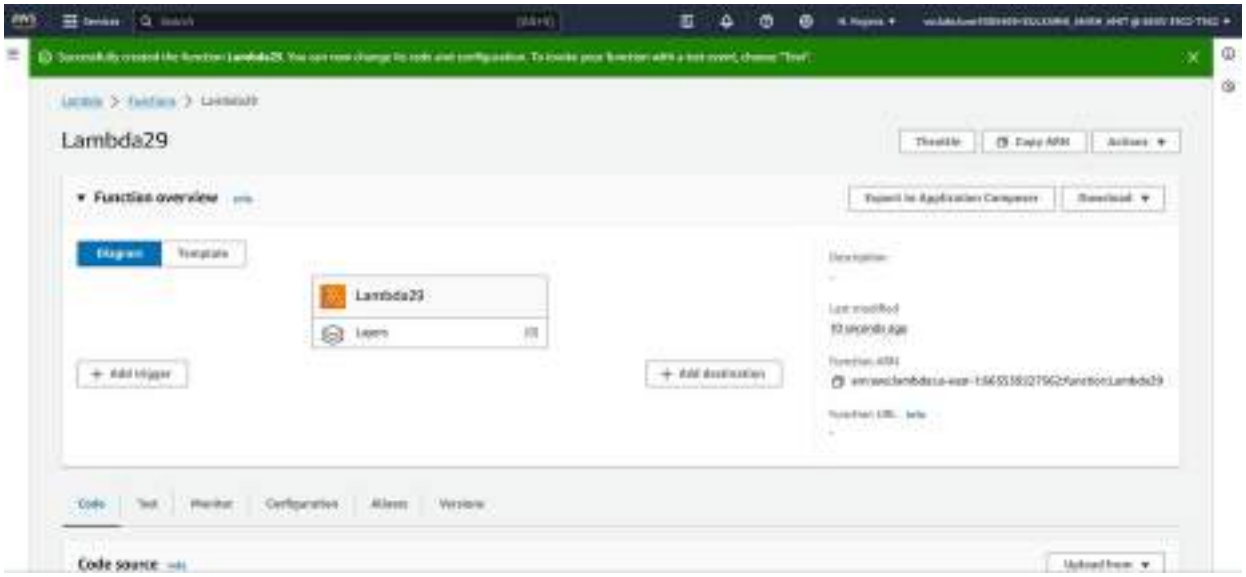
Step 2: Give your Lambda function a name. Select the language to use to write your function (Node.js is the default and what we will use in this experiment). Keep other options as default.

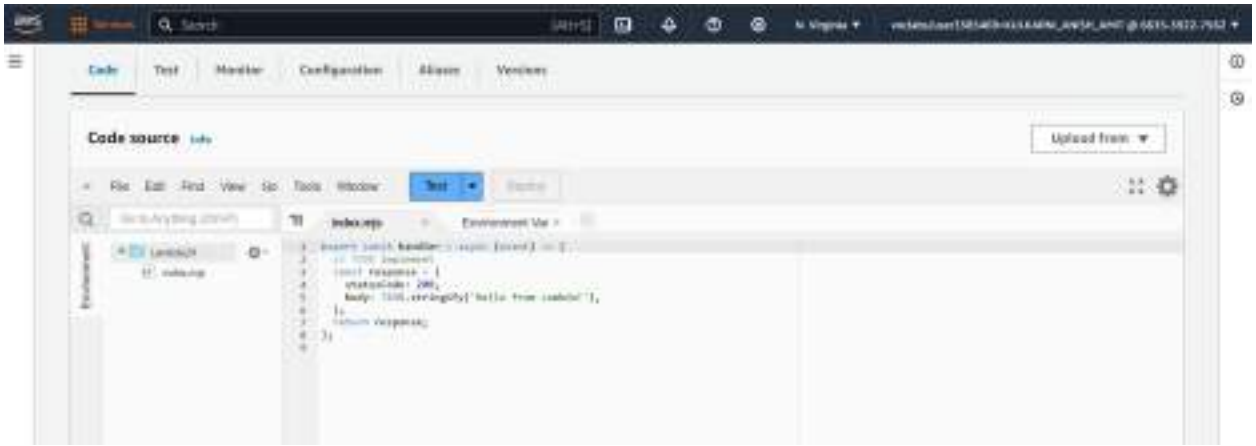




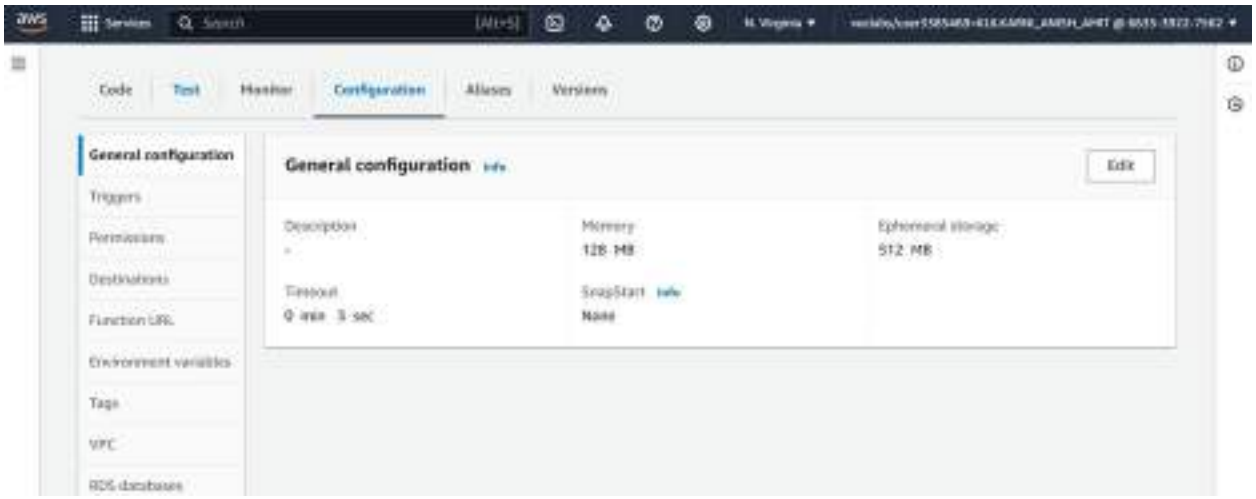
Under ‘Execution role’, choose ‘Use an existing role’ and then choose LabRole. Then, click on ‘Create function’.

Your Lambda function gets created.

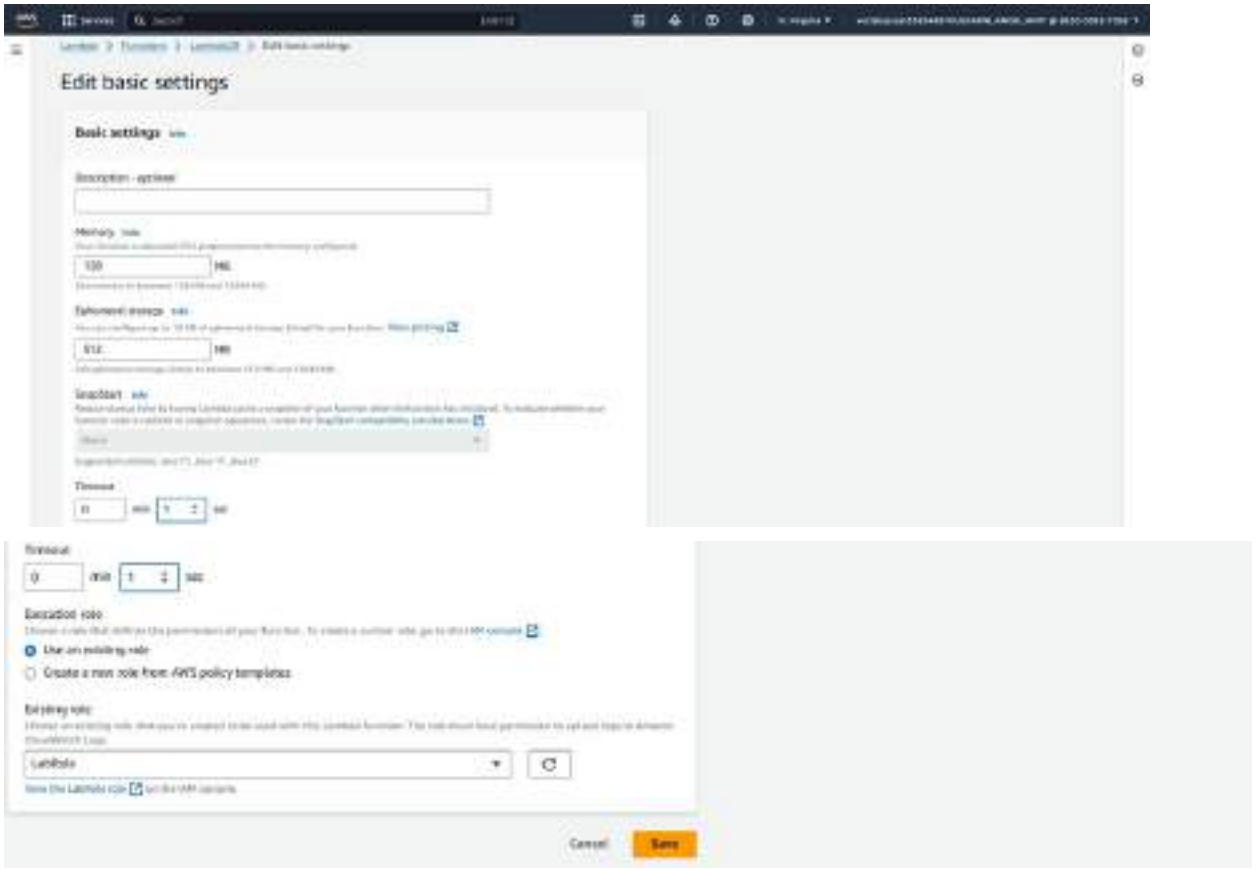




Step 3: The general configuration of the function is visible in the ‘Configuration’ tab. To change the configuration, click on ‘Edit’.

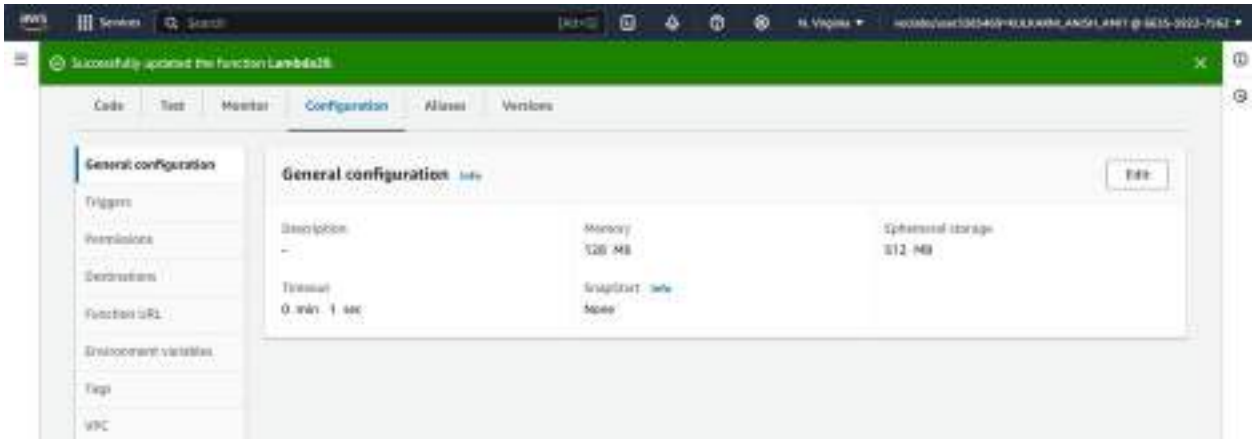


You can change the various parameters of the configuration as per your needs. Here, we can change the ‘Timeout’ period to 1 second as it’s sufficient for our function for now. ‘Timeout’ is the time for which a function can be running before it gets forcibly terminated.



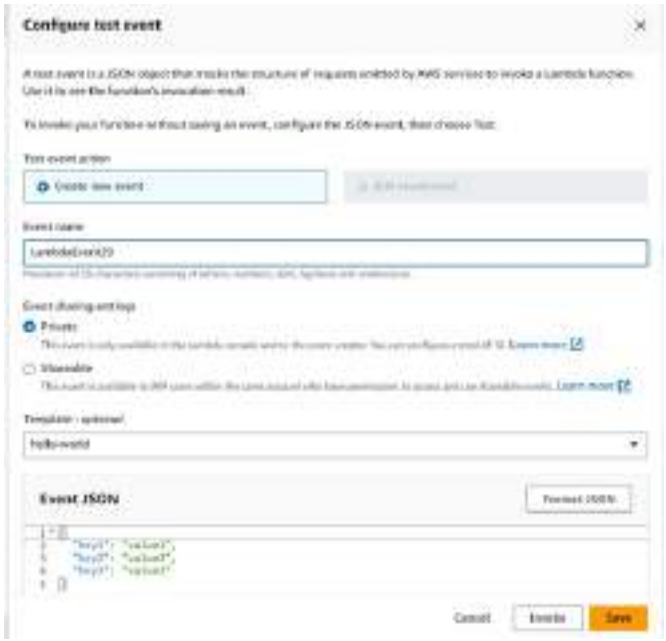
After making the required changes, click on ‘Save’.

The changes in the general configuration are visible in the function.

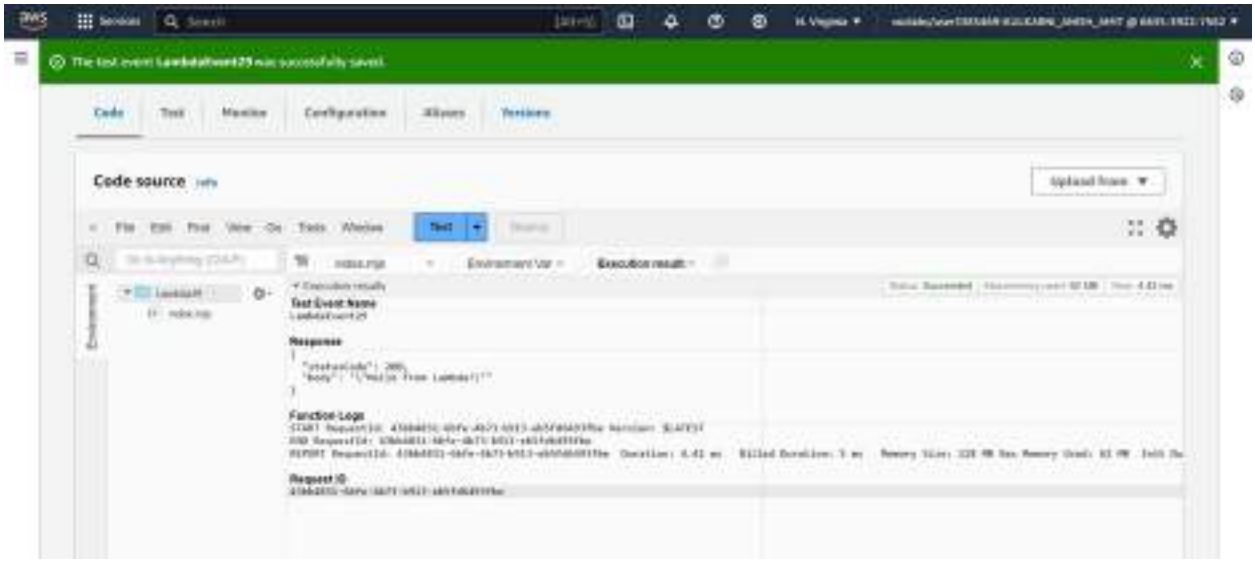


Step 4: In the ‘Code source’ section, click on the arrow next to the ‘Test’ button and click on ‘Configure test event’.

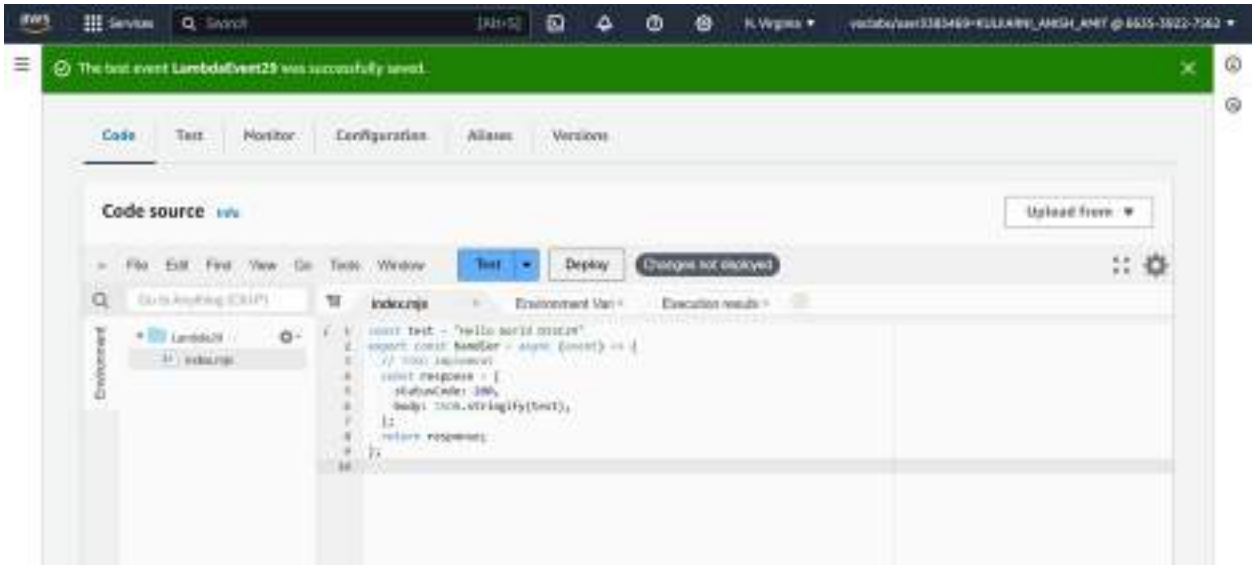
Step 5: Give your test event a name, keep all other options as default and click on ‘Save’.



Step 6: Click on the ‘Test’ button. The following output appears.

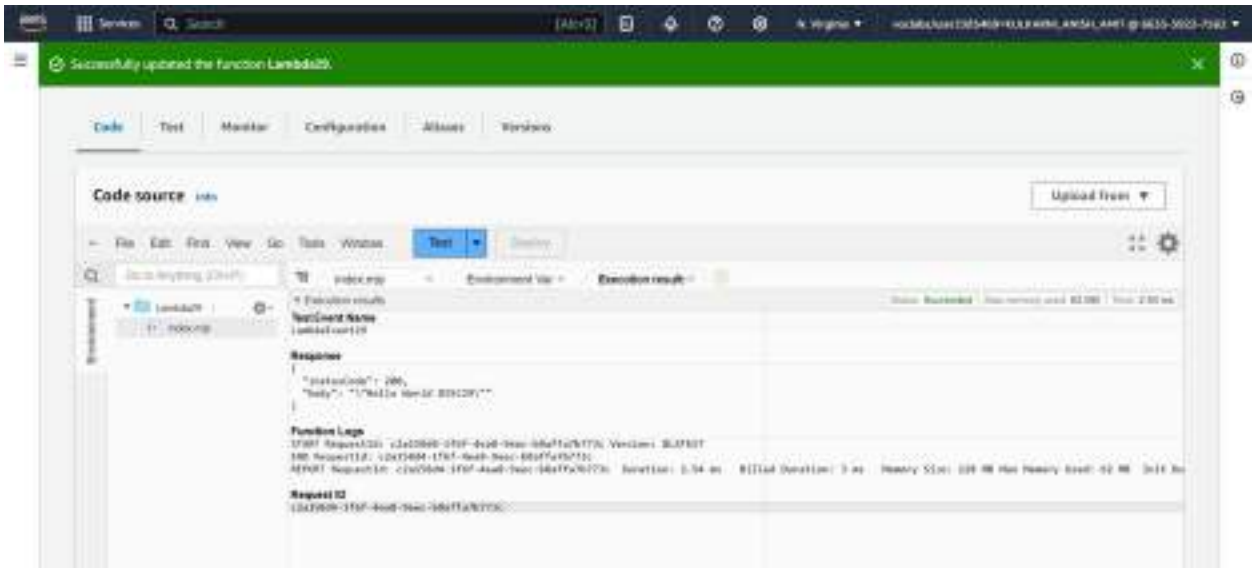


Step 7: You can make changes in the code to observe the difference in the output. Here, we change the code to display a different string as such:-



Once the changes are made, click on ‘Deploy’.

Step 8: Click on ‘Test’ and observe how the output after the changes differs from the output before the changes.



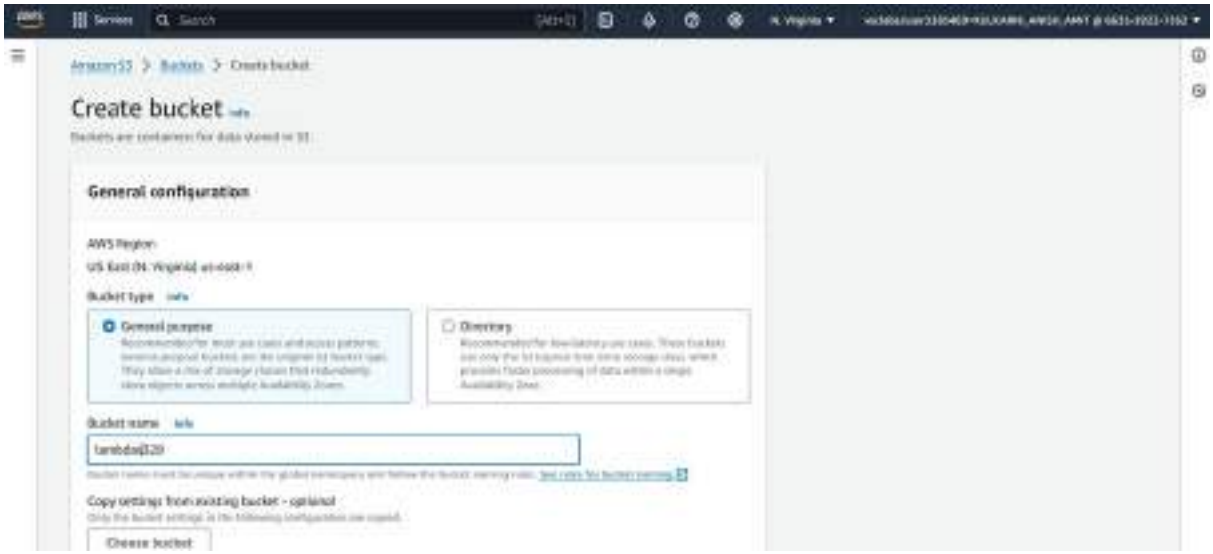
Conclusion: In this experiment, we understood the working of AWS Lambda service by creating and configuring a Lambda function using Node.js. We learned how to set up a Lambda function, change its configurations and test its functioning by creating test events for the function. From the output of the tests, we learn about the working of the Lambda function and how changes in its configuration affects its functionality and outputs.

Experiment 12

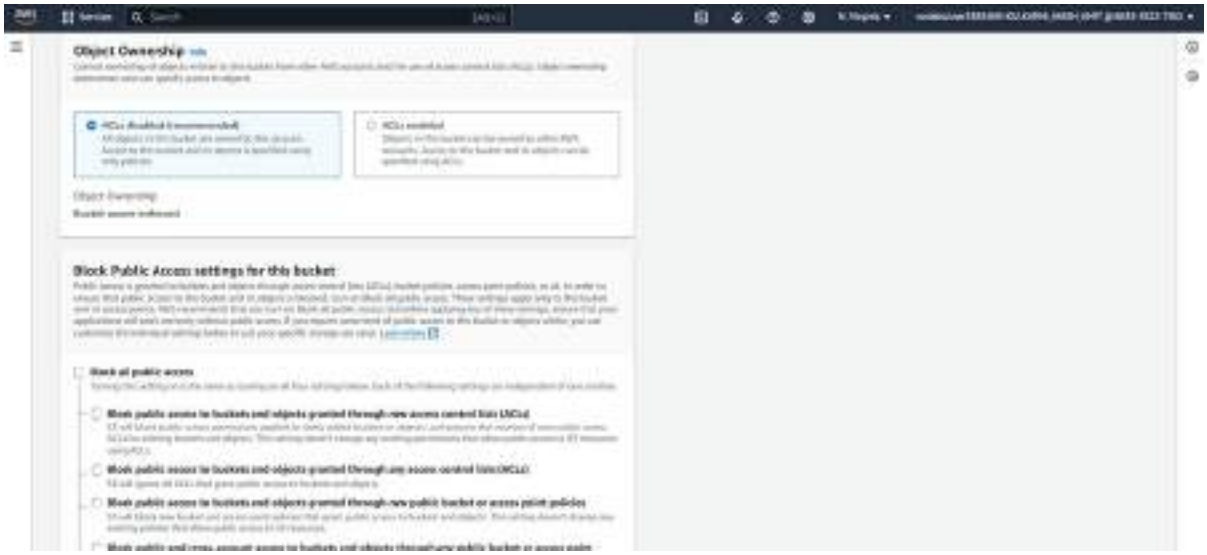
Aim: To create a Lambda function which will log “An Image has been added” once you add an object to a specific bucket in S3.

Steps:

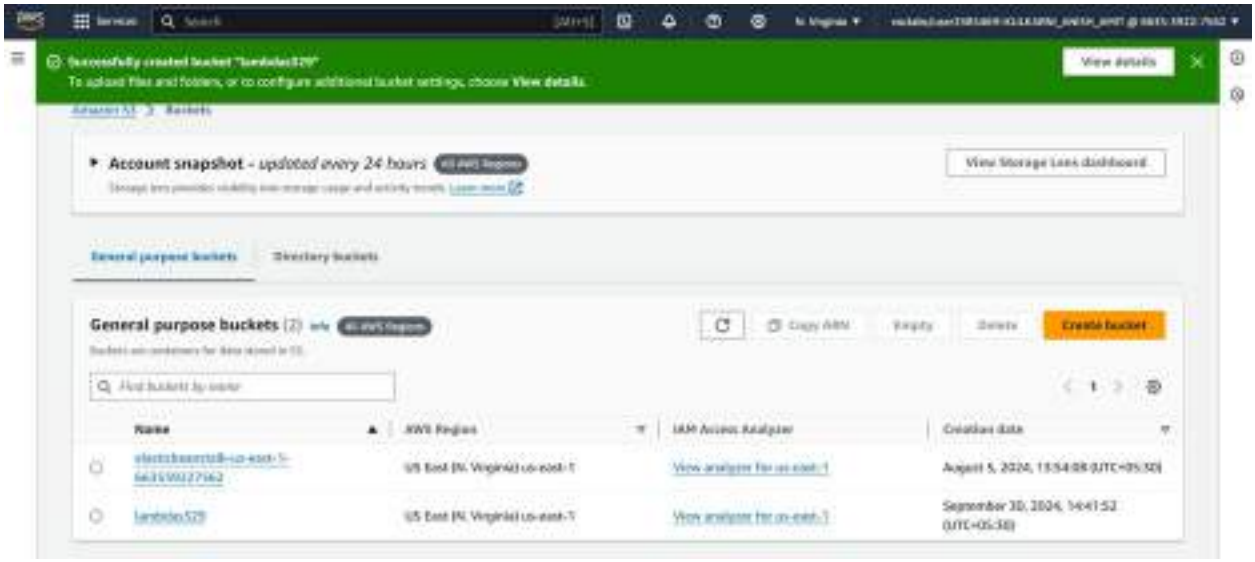
Step 1: On your AWS console, click on ‘S3’ in the services section and click on ‘Create bucket’. Give your bucket a name.



Uncheck the ‘Block all public access’ box.

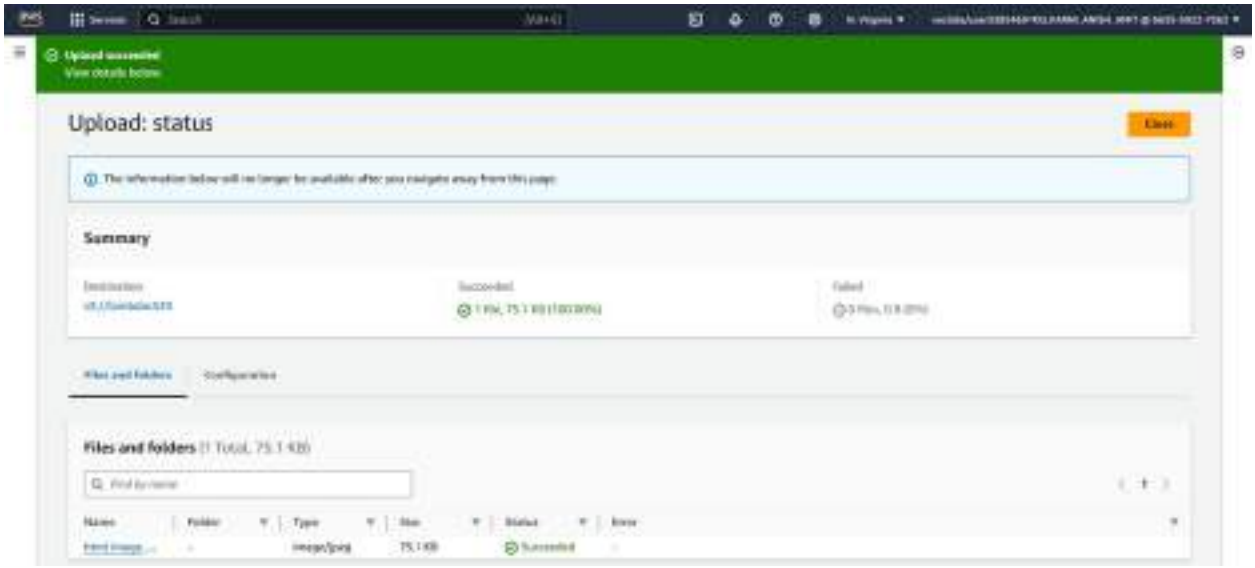


Keep all other options as default and click on ‘Create bucket’.



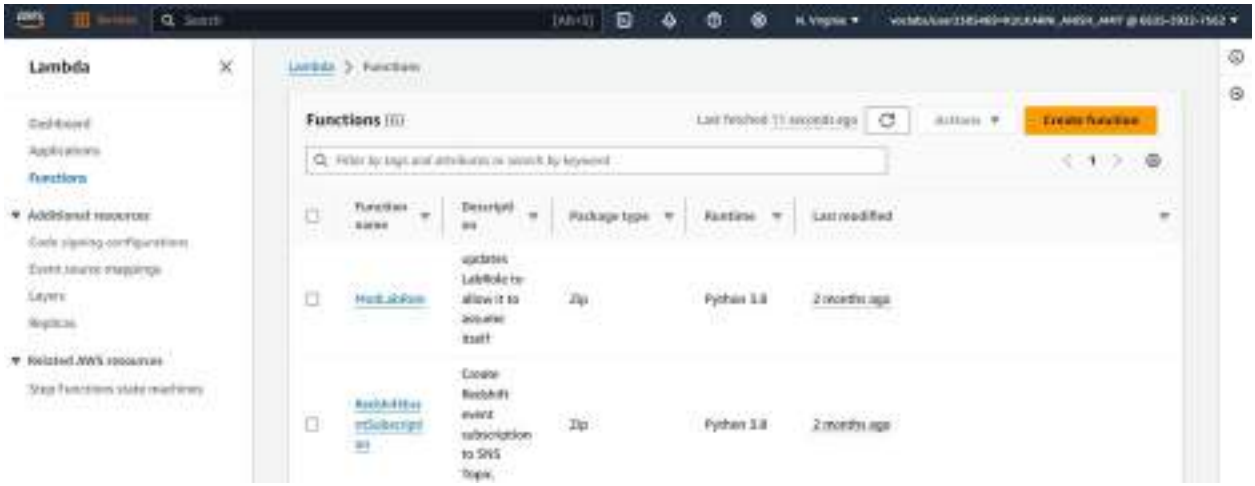
Your bucket is created.

Step 2: Upload an image onto your S3 bucket by clicking on your S3 bucket, clicking on ‘Upload’, clicking on ‘Add files’, navigating to your image and selecting it.

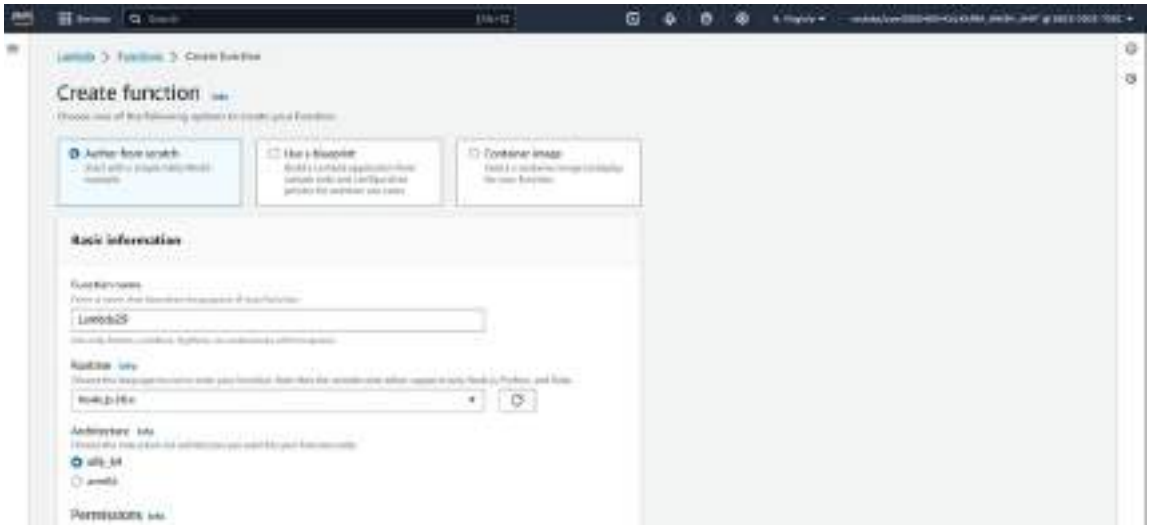


Your image gets uploaded onto the S3 bucket.

Step 3: Navigate to the AWS Lambda console using the ‘Services’ section. Click on ‘Create function’.

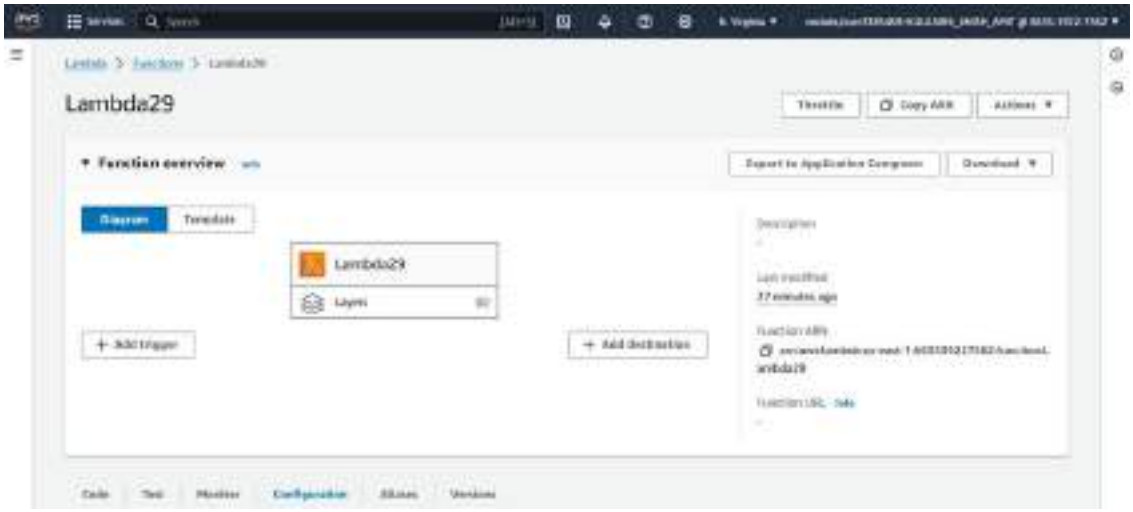


Step 4: Give your function a name and keep other settings as default.

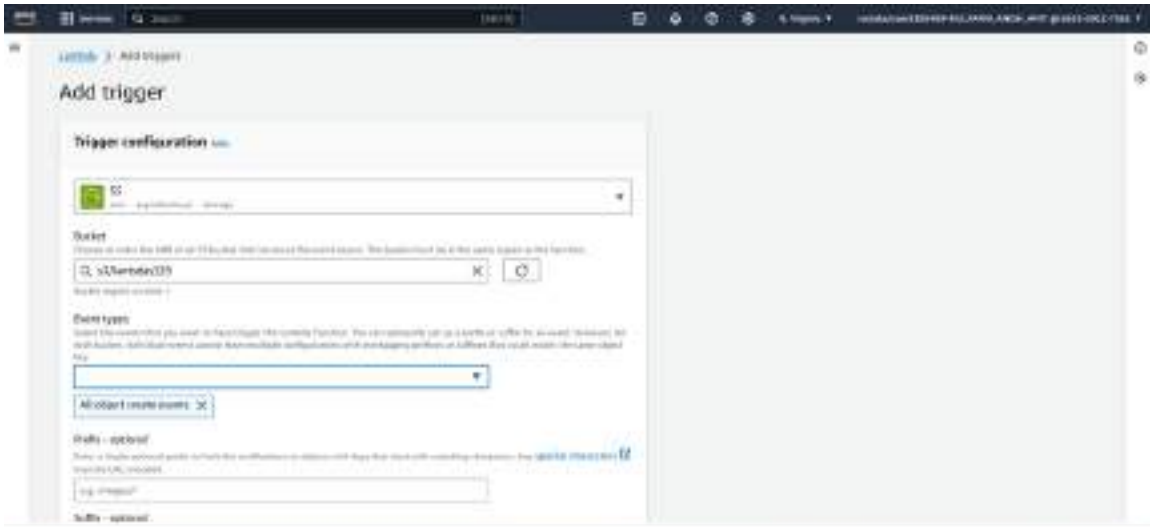


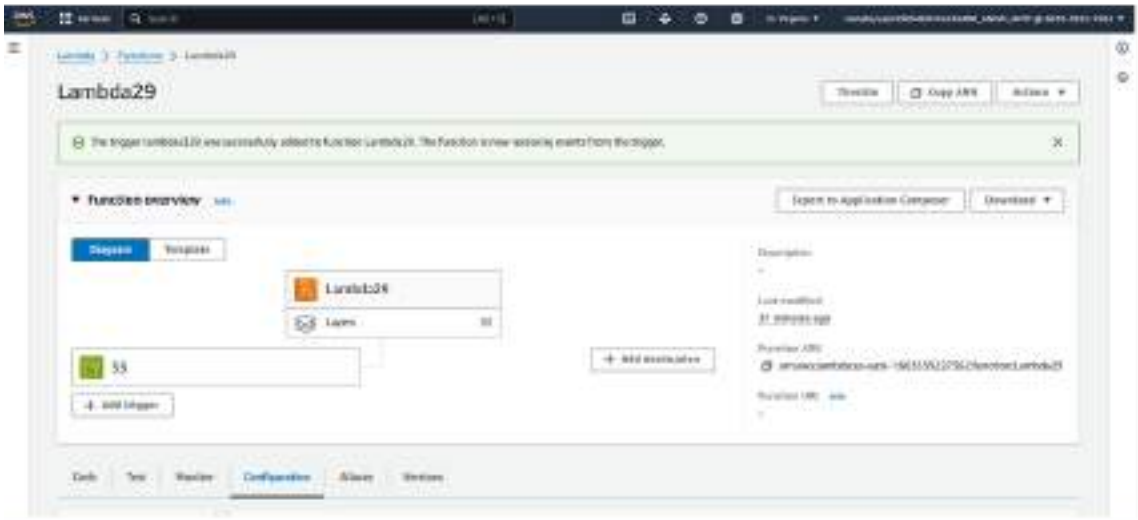
Under ‘Execution role’, choose ‘Use an existing role’ and in the dropdown box below, choose ‘LabRole’. Then, click on ‘Create function’. Your function gets created.

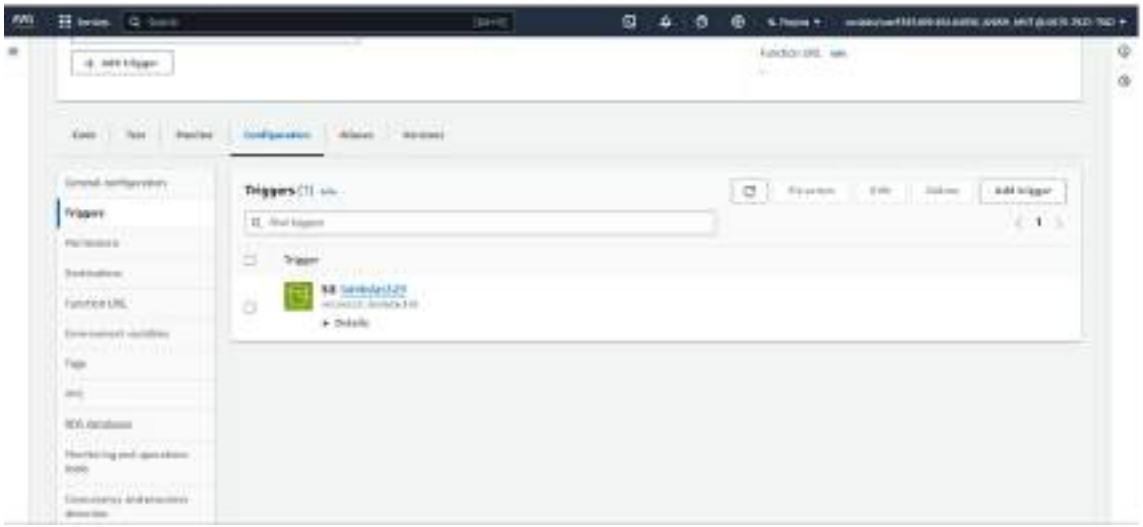
Step 5: On the page of the function you created, click on ‘Add trigger’.



Step 6: Choose ‘Trigger configuration’ as S3 and select the name of your bucket in the dropdown box below it. Keep other options as default and click on ‘Add’.



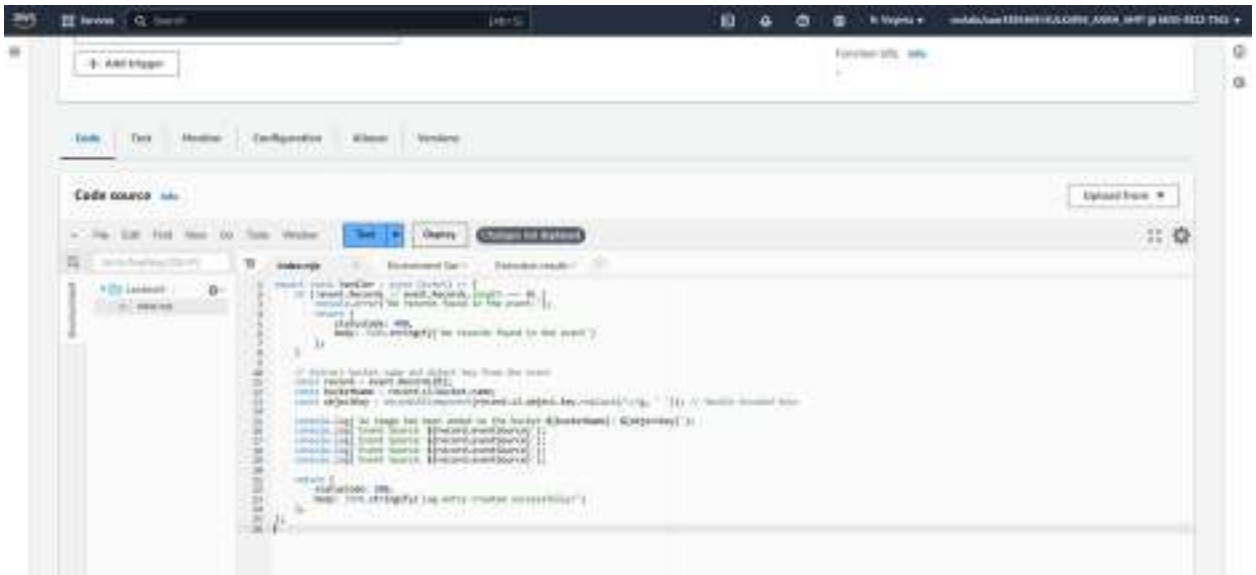




The trigger gets successfully added to your function.

Step 7: In the ‘Code source’ section of your function, paste the following javascript code instead of the existing code:-

```
export const handler = async (event) => {
    if (!event.Records || event.Records.length === 0) {
        console.error("No records found in the event.");
        return {
            statusCode: 400,
            body: JSON.stringify('No records found in the event')
        };
    }

    // Extract bucket name and object key from the event
    const record = event.Records[0];
    const bucketName = record.s3.bucket.name;
    const objectKey = decodeURIComponent(record.s3.object.key.replace(/\+/g, ' ')); // Handle encoded keys

    console.log(`An image has been added to the bucket ${bucketName}: ${objectKey}`);
    console.log(`Event Source: ${record.eventSource}`); console.log(`Event Source: ${record.eventSource}`); console.log(`Event Source: ${record.eventSource}`);
    console.log(`Event Source: ${record.eventSource}`);


    return {
        statusCode: 200,
        body: JSON.stringify('Log entry created successfully!')
    };
};
```

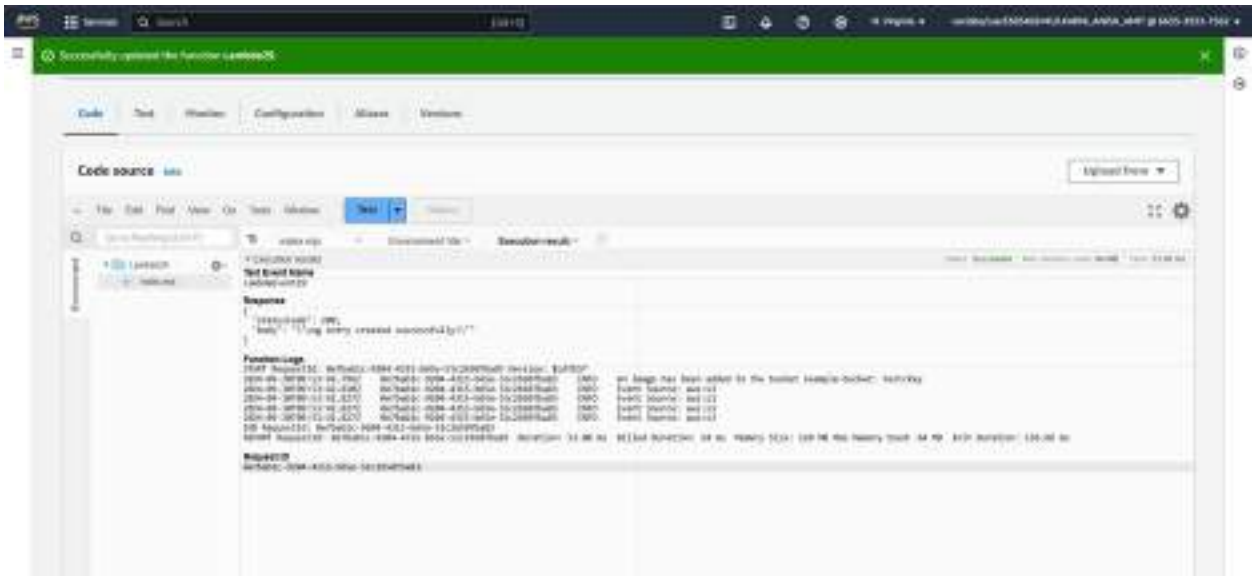


Step 8: Click on the arrow next to the ‘Test’ button and click on ‘Configure test event’. In the popup box that appears, if you have an existing event, enter the name of your event or create a new event and in the ‘Event JSON’ section, paste the following code:-

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2":
"EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "example-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::example-bucket"
        },
        "object": {
          "key": "test%2Fkey",
          "size": 1024,
          "eTag":"0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
  ]
}
```

Then, click on ‘Save’. Your function gets successfully updated.

Step 9: Click on ‘Deploy’ and after deployment is successful, click on ‘Test’.



Running the test gives the above output which displays that ‘An Image has been added to the bucket’ and that the log entry was successfully created.

Conclusion: In this experiment, we learned how to create a Lambda function which logs “An image has been added” once we add an image to our specific S3 bucket. We first created an S3 bucket and uploaded an image to it. We then created a Lambda function and added an S3 trigger to it and selected the S3 bucket we created. Then, we configured the ‘Code section’ of our Lambda function and a test event for our Lambda function. On running the test event, we observed that it logged important information about the event such as the bucket name and object key and also verified that an image had been added to the S3 bucket. Also, a log of our Lambda function was also created which confirmed that the image had been successfully added to the bucket.