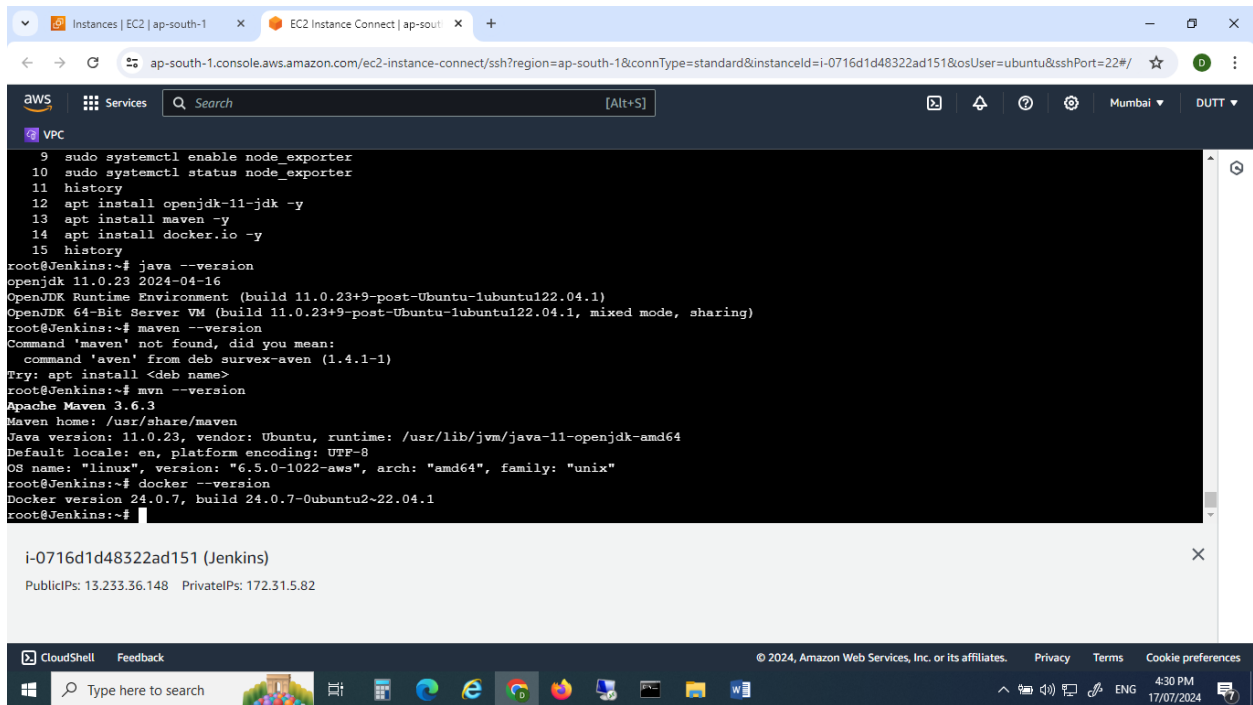


Installed jdk,maven,docker



The screenshot shows a terminal window within the AWS CloudShell interface. The terminal output displays the following commands and their results:

```
9 sudo systemctl enable node_exporter
10 sudo systemctl status node_exporter
11 history
12 apt install openjdk-11-jdk -y
13 apt install maven -y
14 apt install docker.io -y
15 history

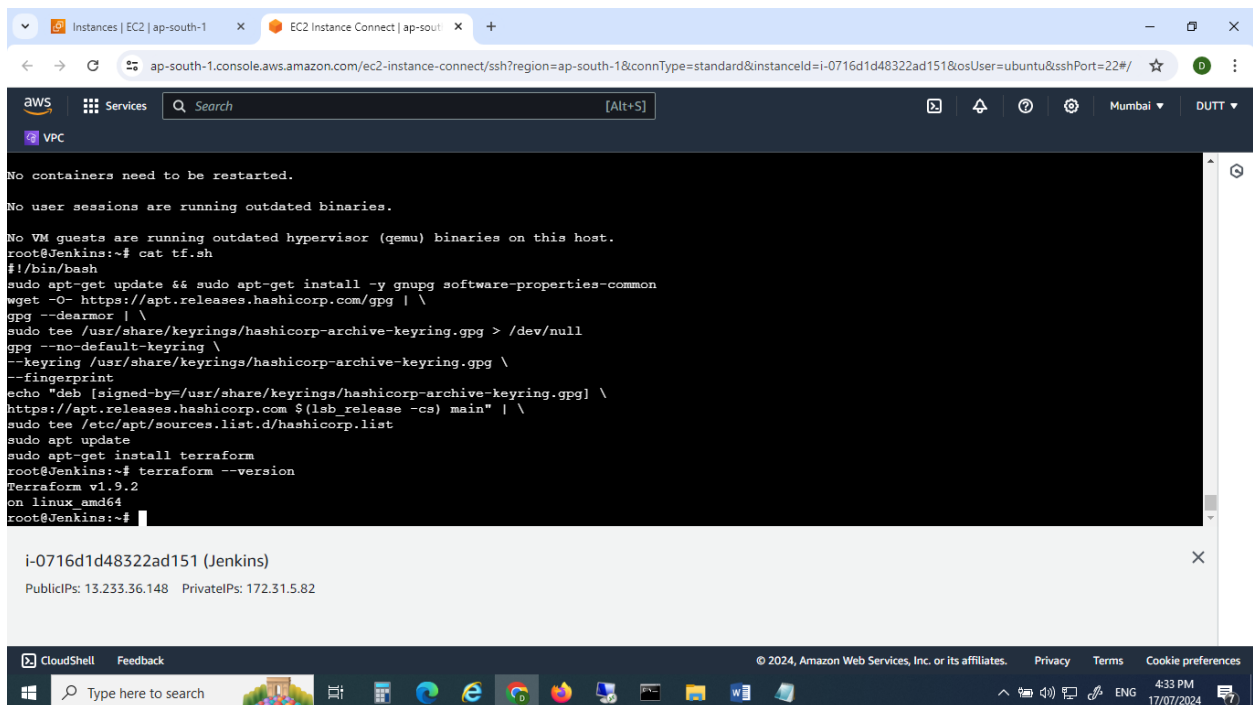
root@jenkins:~# java --version
openjdk 11.0.23 2024-04-16
OpenJDK Runtime Environment (build 11.0.23+9-post-Ubuntu-1ubuntu122.04.1)
OpenJDK 64-Bit Server VM (build 11.0.23+9-post-Ubuntu-1ubuntu122.04.1, mixed mode, sharing)
root@jenkins:~# maven --version
Command 'maven' not found, did you mean:
  command 'aven' from deb surverx-aven (1.4.1-1)
Try: apt install <deb name>
root@jenkins:~# mvn --version
Apache Maven 3.6.3
Maven home: /usr/share/maven
Java version: 11.0.23, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdk-amd64
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "6.5.0-1022-aws", arch: "amd64", family: "unix"
root@jenkins:~# docker --version
Docker version 24.0.7, build 24.0.7-0ubuntu2-22.04.1
root@jenkins:~#
```

Below the terminal output, the instance details are shown:

i-0716d1d48322ad151 (Jenkins)
PublicIPs: 13.233.36.148 PrivateIPs: 172.31.5.82

The bottom of the screenshot shows the AWS CloudShell interface with the search bar, feedback button, and copyright information: © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences. The date and time are 4:30 PM 17/07/2024.

Terraform Installed



The screenshot shows a terminal window within the AWS CloudShell interface. The terminal output displays the following commands and their results:

```
No containers need to be restarted.

No user sessions are running outdated binaries.

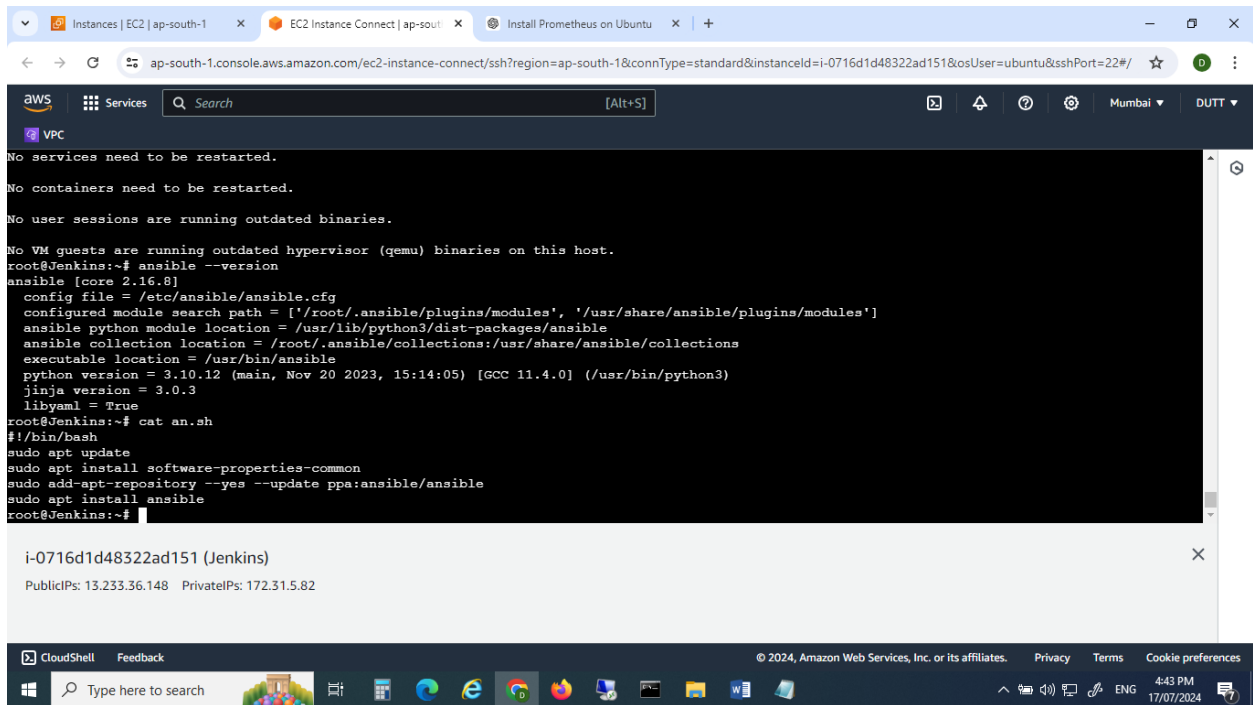
No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@jenkins:~# cat tf.sh
#!/bin/bash
sudo apt-get update && sudo apt-get install -y gnupg software-properties-common
wget -O- https://apt.releases.hashicorp.com/gpg | \
gpg --dearmor | \
sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null
gpg --no-default-keyring \
--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
--fingerprint
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
sudo tee /etc/apt/sources.list.d/hashicorp.list
sudo apt update
sudo apt-get install terraform
root@jenkins:~# terraform --version
Terraform v1.9.2
on linux amd64
root@jenkins:~#
```

Below the terminal output, the instance details are shown:

i-0716d1d48322ad151 (Jenkins)
PublicIPs: 13.233.36.148 PrivateIPs: 172.31.5.82

The bottom of the screenshot shows the AWS CloudShell interface with the search bar, feedback button, and copyright information: © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences. The date and time are 4:33 PM 17/07/2024.

Ansible Installed



The screenshot shows the AWS CloudShell interface with a terminal window. The terminal output indicates that no services, containers, or user sessions need to be restarted. It also shows that no VM guests are running outdated hypervisor (qemu) binaries. The user runs the command `ansible --version`, which displays the configuration file path, module search path, python module location, ansible collection location, executable location, python version (3.10.12), and Jinja version (3.0.3). The user then runs `cat an.sh`, which shows the contents of the `an.sh` script. The script includes commands to update apt, install `software-properties-common`, add the ansible PPA, and install ansible. The terminal output shows the successful installation of ansible.

```
No services need to be restarted.

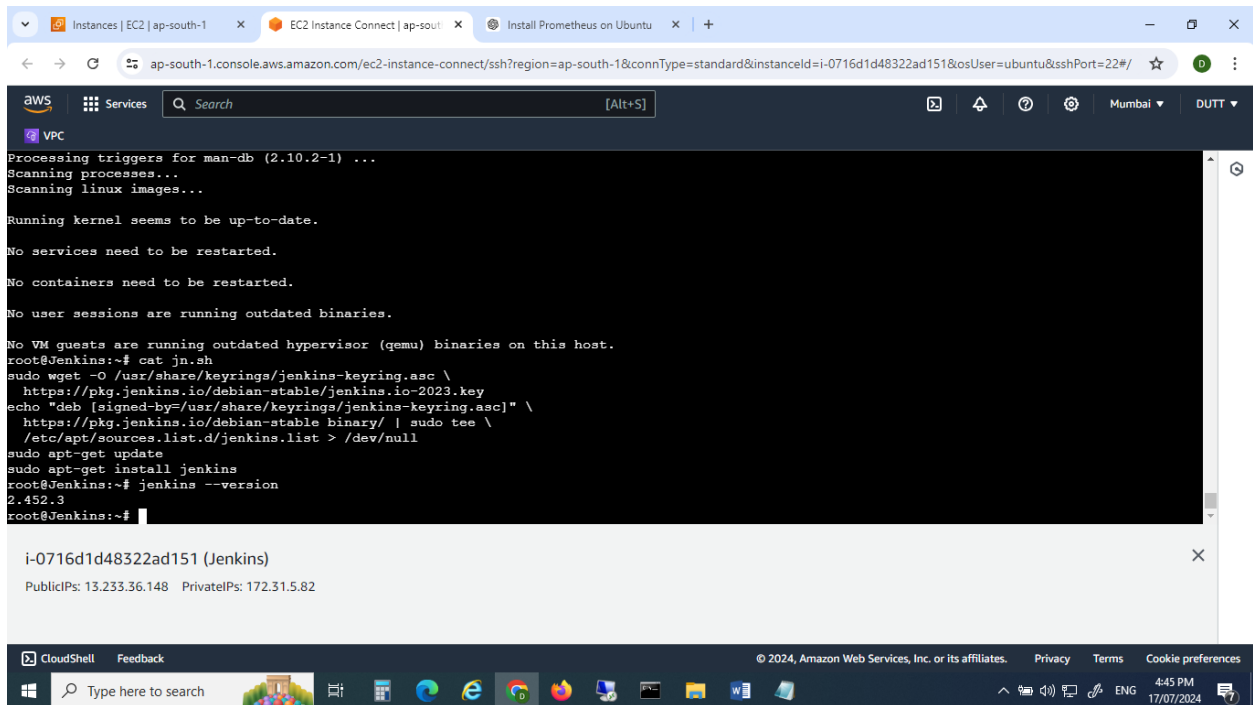
No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@jenkins:~# ansible --version
ansible [core 2.16.0]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /root/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] (/usr/bin/python3)
  jinja version = 3.0.3
  libyaml = True
root@jenkins:~# cat an.sh
#!/bin/bash
sudo apt update
sudo apt install software-properties-common
sudo add-apt-repository --yes --update ppa:ansible/ansible
sudo apt install ansible
root@jenkins:~#
```

i-0716d1d48322ad151 (Jenkins)
PublicIPs: 13.233.36.148 PrivateIPs: 172.31.5.82

Jenkins Installed



The screenshot shows the AWS CloudShell interface with a terminal window. The terminal output indicates that no services, containers, or user sessions need to be restarted. It also shows that no VM guests are running outdated hypervisor (qemu) binaries. The user runs the command `cat jn.sh`, which shows the contents of the `jn.sh` script. The script includes commands to download the Jenkins keyring, add the Jenkins PPA, and install Jenkins. The terminal output shows the successful installation of Jenkins.

```
Processing triggers for man-db (2.10.2-1) ...
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.
root@jenkins:~# cat jn.sh
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/" | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
root@jenkins:~# jenkins --version
2.452.3
root@jenkins:~#
```

i-0716d1d48322ad151 (Jenkins)
PublicIPs: 13.233.36.148 PrivateIPs: 172.31.5.82

Added Jenkins users to sudo access

The screenshot shows an AWS CloudShell terminal window. The terminal is running the nano text editor on the file `/etc/sudoers.tmp`. The configuration includes host and user aliases, and grants sudo access to the `jenkins` user and the `admin` group. The terminal output is as follows:

```
GNU nano 6.2 /etc/sudoers.tmp

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL
jenkins  ALL=(ALL) NOPASSWD:ALL
# Members of the admin group may gain root privileges
%admin   ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "@include" directives:

@include::dir::/etc/sudoers.d

```

Below the terminal window, a metadata box for the instance `i-0716d1d48322ad151 (Jenkins)` is visible, showing Public IPs: 13.233.36.148 and Private IPs: 172.31.5.82.

Add docker to Jenkins group and check ansible working

The screenshot shows an AWS CloudShell terminal window. The terminal output is as follows:

```
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.
root@Jenkins:~# cat jn.sh
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/" | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
root@Jenkins:~# jenkins --version
2.452.3
root@Jenkins:~# visudo
root@Jenkins:~# visudo
root@Jenkins:~# sudo usermod -aG docker jenkins
root@Jenkins:~# ansible -m ping localhost
localhost | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
root@Jenkins:~#
```

Below the terminal window, a metadata box for the instance `i-0716d1d48322ad151 (Jenkins)` is visible, showing Public IPs: 13.233.36.148 and Private IPs: 172.31.5.82.

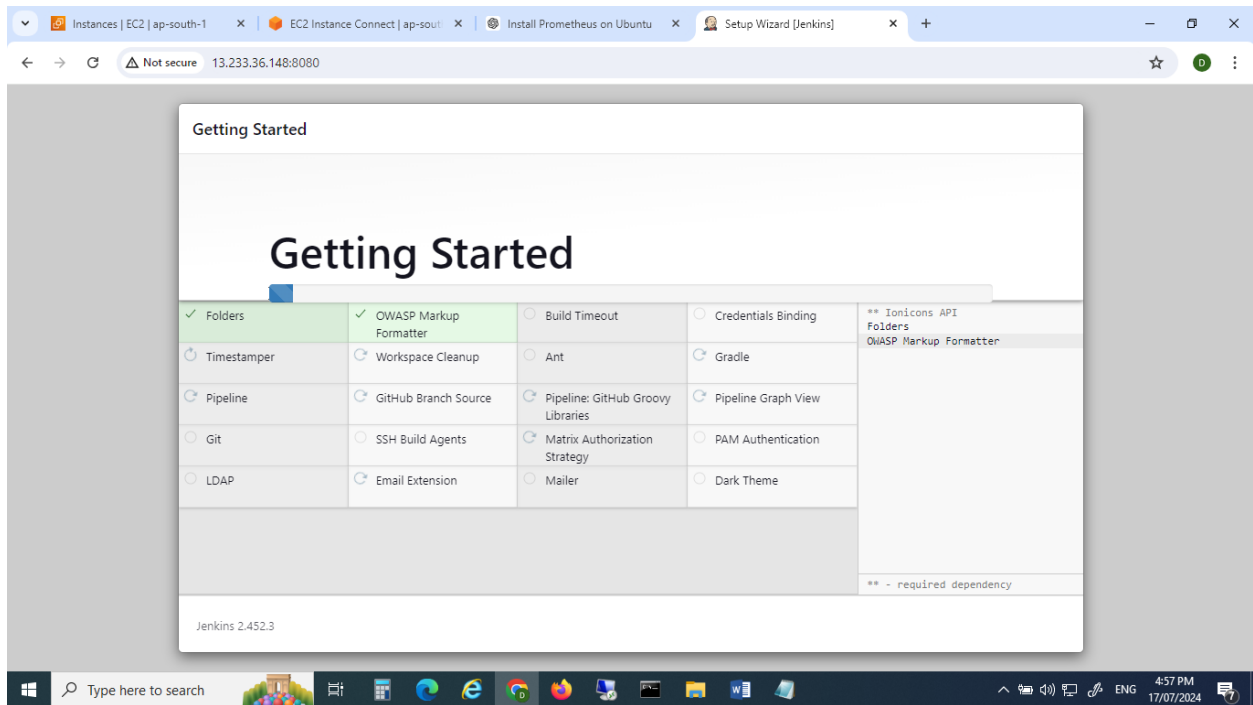
Create roll with ec2 full access

The screenshot shows the AWS IAM console in the 'ap-south-1' region. The left sidebar displays the 'Identity and Access Management (IAM)' menu with options like Dashboard, Access management, Users, Roles, Policies, Identity providers, Account settings, Access reports, Access Analyzer, and External access. The main content area shows the details for the role 'rollfortf'. At the top, it displays the role's ARN, last activity (14 hours ago), and maximum session duration (1 hour). Below this, the 'Permissions' tab is selected, showing 'Permissions policies (1/1)'. A table lists the attached policies, including 'AmazonEC2FullAccess' (AWS managed). The 'Permissions boundary' is noted as 'not set'. The bottom of the screen shows the Windows taskbar with various application icons and the system clock indicating 4:54 PM on 17/07/2024.

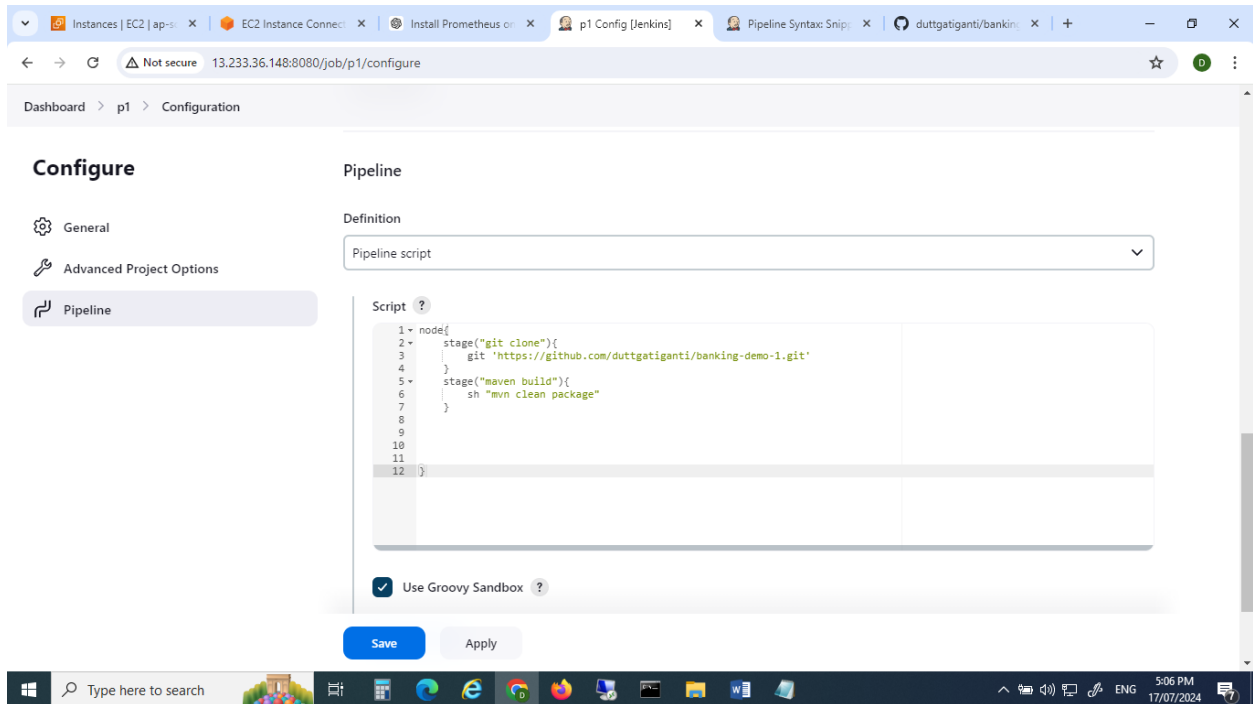
Attach role to ec2

The screenshot shows the AWS EC2 console in the 'ap-south-1' region. The breadcrumb navigation indicates the path: EC2 > Instances > i-0716d1d48322ad151 > Modify IAM role. The main heading is 'Modify IAM role', with a subtext 'Attach an IAM role to your instance.' Below this, a form is displayed with the 'Instance ID' set to 'i-0716d1d48322ad151 (Jenkins)'. Under the 'IAM role' section, a dropdown menu shows 'rollfortf' as the selected role. There are buttons for 'Create new IAM role' and 'Update IAM role'. The bottom of the screen shows the Windows taskbar with various application icons and the system clock indicating 4:55 PM on 17/07/2024.

Access Jenkins



Configure git clone and build



Build success

The screenshot shows the Jenkins Pipeline View for a pipeline named 'p1'. The interface includes a left sidebar with navigation options: Dashboard, Build View, Configure, Delete Pipeline, Full Stage View, Stages, Rename, and Pipeline Syntax. The main area displays the 'Stage View' for the current build (#2). The stages are 'git clone' and 'maven build'. The 'git clone' stage is successful, taking 741ms. The 'maven build' stage is successful, taking 22s. The overall average stage time is 1s, and the average full run time is ~22s. The build history on the left shows two builds: #2 (successful) and #1 (failed). The 'Permalinks' section provides links to various build details.

Stage View

Average stage times: 1s
(Average full run time: ~22s)

Stage	Duration	Status
git clone	741ms	Successful
maven build	22s	Successful

Build History

- #2 (Jul 17, 2024, 11:34 AM) - Successful
- #1 (Jul 17, 2024, 11:32 AM) - Failed

Permalinks

- Last build (#2), 3 min 27 sec ago
- Last stable build (#2), 3 min 27 sec ago
- Last successful build (#2), 3 min 27 sec ago
- Last failed build (#1), 5 min 37 sec ago
- Last unsuccessful build (#1), 5 min 37 sec ago
- Last completed build (#2), 3 min 27 sec ago

Configure Docker build

The screenshot shows the Jenkins Pipeline Configuration page for a pipeline named 'p1'. The 'Configure' section is active, showing the 'Pipeline script' tab. The script is written in Groovy and defines three stages: 'git clone', 'maven build', and 'Docker Image'. The 'Docker Image' stage is currently empty. The 'Use Groovy Sandbox' checkbox is checked. The 'Pipeline Syntax' link is visible at the bottom.

Configure

Pipeline script

```
1 node{
2   stage("git clone"){
3     git 'https://github.com/duttgiganti/banking-demo-1.git'
4   }
5   stage("maven build"){
6     sh "mvn clean package"
7   }
8
9   stage("Docker Image") {
10
11     sh "docker build -t duttl/primg:${BUILD_NUMBER} ."
12     sh "docker tag duttl/primg:${BUILD_NUMBER} duttl/primg:v1.3"
13   }
14
15
16
17
18 }
```

☒ Use Groovy Sandbox

[Pipeline Syntax](#)

[Save](#) [Apply](#)

Docker build success

The screenshot shows the Jenkins 'Stage View' for a pipeline named 'p1'. The interface includes a left sidebar with navigation options: Configure, Delete Pipeline, Full Stage View, Stages, Rename, and Pipeline Syntax. A 'Build History' panel on the left shows three builds: #3 (successful, 11:40 AM), #2 (successful, 11:34 AM), and #1 (failed, 11:32 AM). The main area displays a table of stage execution times for three stages: #3, #2, and #1. Stage #1 is highlighted in red, indicating a failure. The 'Docker Image' stage is shown as successful with a 24s duration.

	git clone	maven build	Docker Image
Average stage times: (Average full run time: ~31s)	1s	12s	24s
#3 Jul 17 17:10 No Changes	696ms	13s	24s
#2 Jul 17 17:04 No Changes	741ms	22s	
#1 Jul 17 17:02 No Changes	2s	3s failed	

Permalinks:

- Last build (#3), 1 min 29 sec ago
- Last stable build (#3), 1 min 29 sec ago

Configure terraform stage and docker login

The screenshot shows the Jenkins 'Configuration' page for the 'p1' pipeline. The 'Pipeline' tab is selected in the left sidebar. The 'Script' section contains a Groovy script for the 'Terraform stage'. The script includes a 'withCredentials' block for Docker login and a 'stage' block for Terraform operations. The 'Use Groovy Sandbox' checkbox is checked.

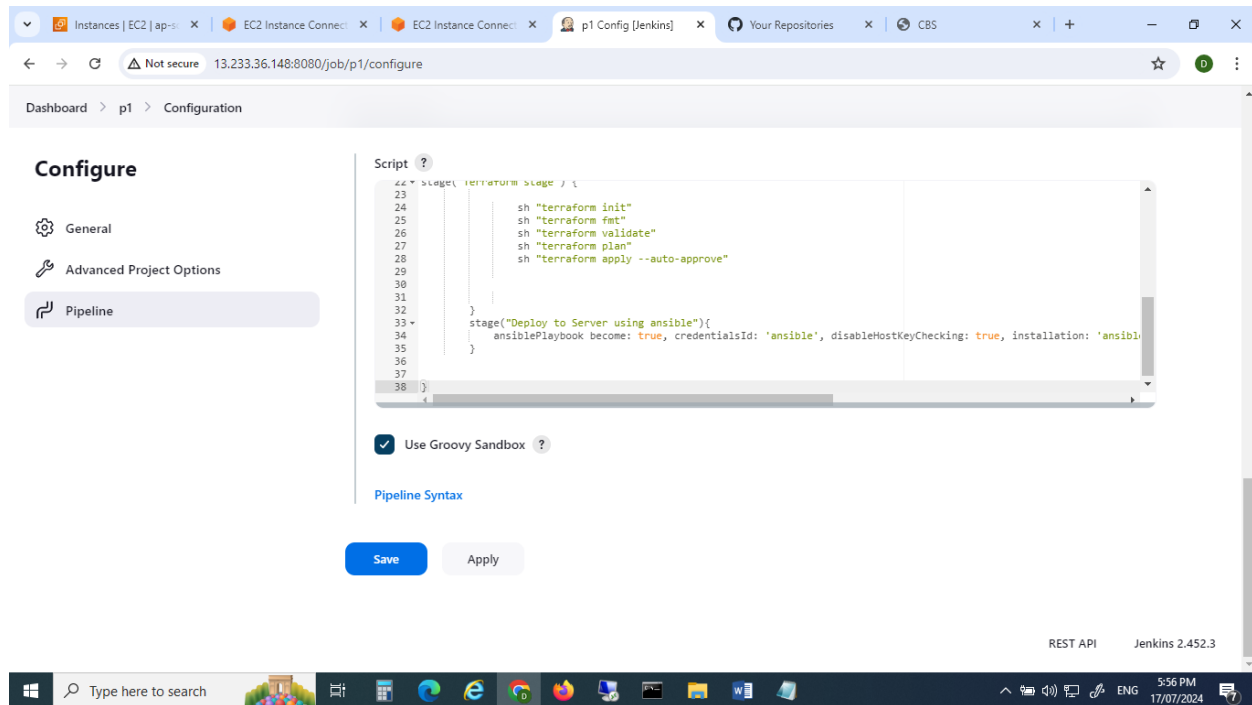
```
17 withCredentials([string(credentialsId: 'dock', variable: 'dock')]) {
18     sh "docker login -u dutt.getigant@gmail.com -p ${dock}"
19     sh "docker push dutt1/primg:v1.3"
20 }
21 stage("Terraform stage") {
22     sh "terraform init"
23     sh "terraform fmt"
24     sh "terraform validate"
25     sh "terraform plan"
26     sh "terraform apply --auto-approve"
27 }
28 }
29 }
30 }
31 }
32 }
33 }
34 }
```

☒ Use Groovy Sandbox

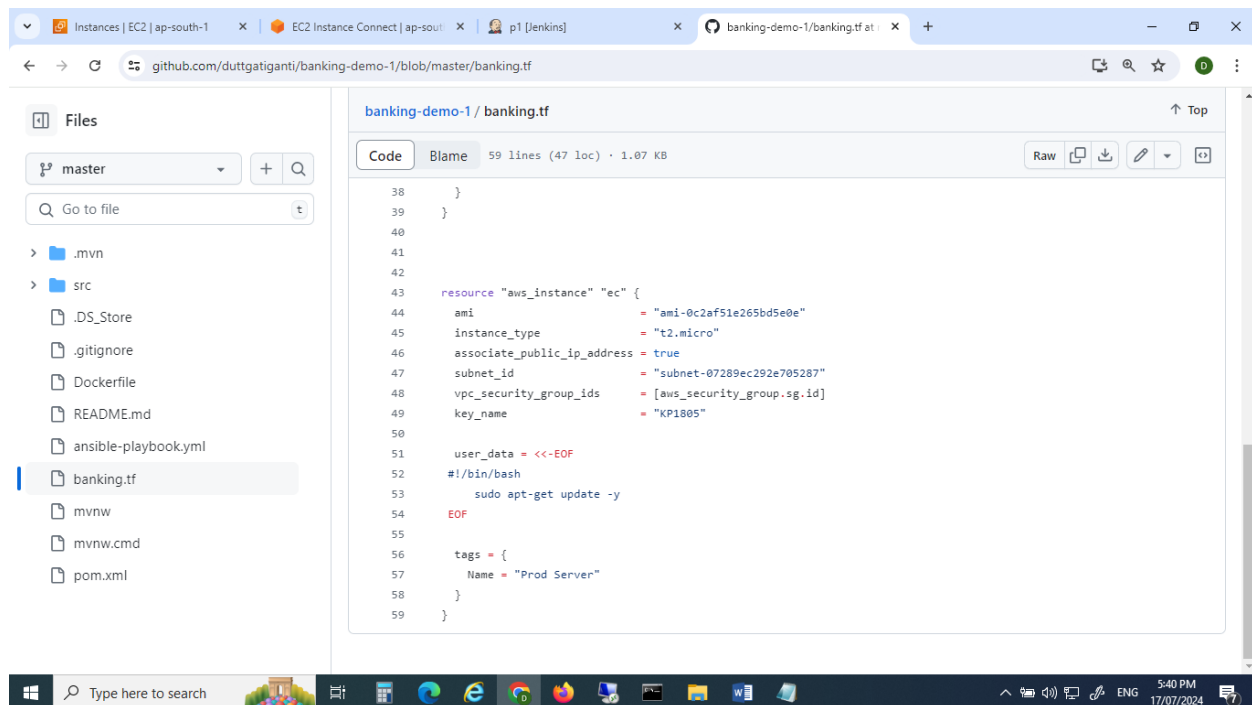
[Pipeline Syntax](#)

[Save](#) [Apply](#)

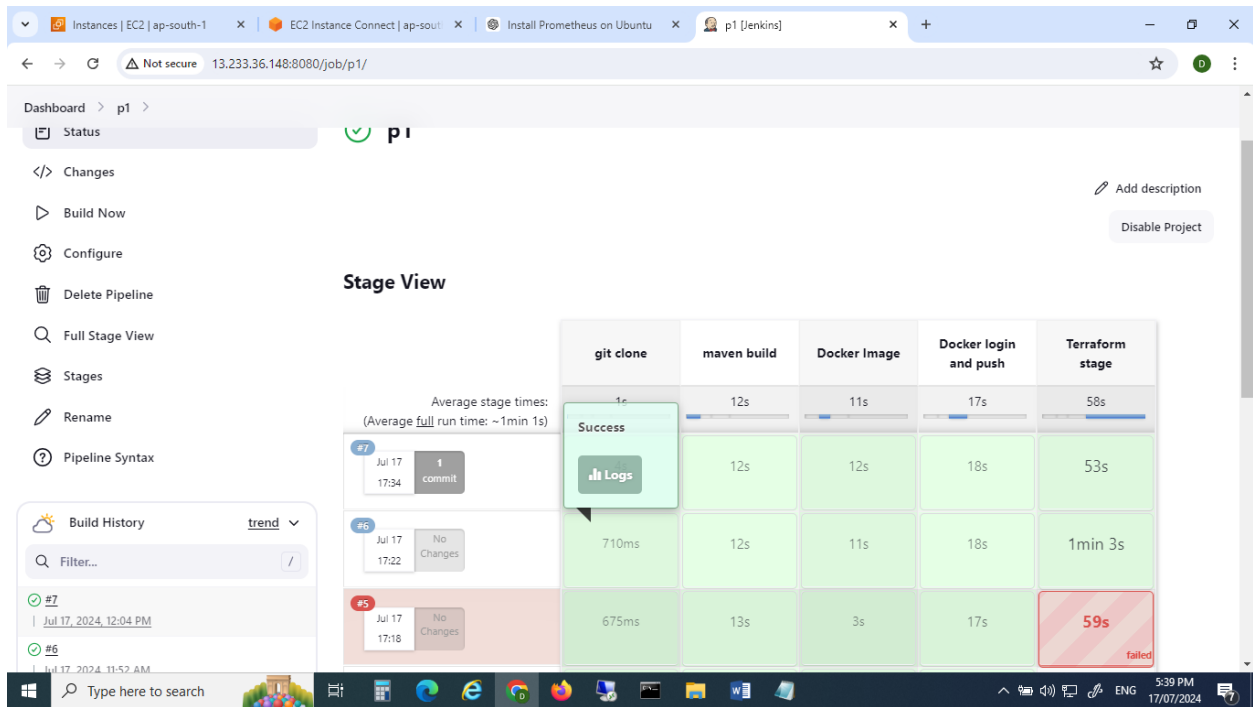
Configure ansible



Script for Prod server



Terraform apply success



Configure ansible credentials,hosts with ssh keys

```
## db-[99:101]-node.example.com

# Ex 3: A collection of database servers in the 'dbservers' group:

## [dbservers]
##
## db01.intranet.mydomain.net
## db02.intranet.mydomain.net
## 10.25.1.56
## 10.25.1.57

# Ex4: Multiple hosts arranged into groups such as 'Debian' and 'openSUSE':

## [Debian]
## alpha.example.org
## beta.example.org

## [openSUSE]
## green.example.com
## blue.example.com
## [n]
## 3.109.1.187
root@Jenkins:/etc/ansible#
```

i-0716d1d48322ad151 (Jenkins)

PublicIPs: 13.233.36.148 PrivateIPs: 172.31.5.82

Ansible stage success

Successfully deployed

Github-webhook configuration

if I configure web-hook at initial Jenkins is triggering for every troubleshooting change.

So I configured web hook at last

Changes pushed

Triggered automatically

Created Monetoring server

Commands executed for installing Prometheus

Prometheus service configured

Prometheus status

Hosts added in monitoring server

Grafana configured

Added Data source t grafana

CPU utilization from Prometheus

Disk Space Utilization

Total Available Memory

View of data in grafana

Added Dash board to grafana