ADHIYAMAAN COLLEGE OF ENGINEERING
(Autonomous), Hosur

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(Accredited by NBA)

622CIT04 - DevOps LABORATORY

(REGULATION - 2022)

STAFF INCHARGE **HOD**

ADHIYAMAAN COLLEGE OF ENGINEERING
(Autonomous), Hosur

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(Accredited by NBA)

Vision of the Institute

To foster ACE as a centre for nurturing and developing world class Engineers and Managers who convert global challenges into opportunities through value-based quality education.

Mission of the Institute

M 1 : To impart value-based quality education through effective teaching-learning processes

M 2 : To nurture creativity, excellence and critical thinking by applying global competencyfactors to contribute and excel in the rapidly growing technological world.

M 3 : To continuously develop and improve holistic and innovative personality for globalMobility.

M 4 : To make ACE a centre for excellence.

Vision of the Department

To empower young minds to become resilient professionals, instilled with ethical principles andequipped with cutting-edge technologies to meet the evolving demands of the world

Mission of the Department

M 1 : To empower individuals with a comprehensive understanding of computer engineering principles and its applications through effective teaching and learning practices.

M 2 : To cultivate excellence and critical thinking, while leveraging global competency, thus enabling significant contributions to societal challenges in the fast-paced technologicallandscape.

M 3 : To facilitate the students to work with modern tools and technologies to foster innovation, azest for higher studies and to build leadership qualities by inculcating the spirit of ethical values.

Program Educational Objectives (PEOs)

PEO1 : The graduates will have sound knowledge in Mathematics, Science and Engineering concepts necessary to formulate, analyse, design and solve Engineering problems and to prepare them for higher learning, research and industry.

PEO2 : The graduates will possess innovative skills to assess and apply the rapid changes intechnology and to engage in research leading to novel solutions for human, social and global competency.

PEO3 : The graduates will acquire knowledge and grab opportunities to work as teams in a multidisciplinary environment, communicate ideas effectively with diverse audiences demonstrate leadership qualities with ethical values and engage in lifelong learning.

ADHIYAMAAN COLLEGE OF ENGINEERING
(Autonomous), Hosur

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(Accredited by NBA)

622CIT04 - DevOps LABORATORY

COURSE OBJECTIVES:

- Understand and implement DevOps principles.
- Learn CI/CD pipelines concepts and related tools.
- Learn infrastructure management using IaC.
- Build expertise configuration management and monitoring systems.
- Illustrate the benefits and drive the adoption of cloud-based Devops tools to solve real world problems

LIST OF EXERCISES:

1. Provision a Virtual Machine (VM) in AWS/Azure using Terraform
2. Provision a Virtual Machine (VM) in AWS with Cloud Formation / Azure with ARM
3. Use Ansible to configure the VM and install Nginx
4. Create a CI pipeline using Jenkins with stages for code checkout, build and test
5. Deploy a sample application inside the VM using Jenkins (CD Pipeline)
6. Run a container using pre-build docker image
7. Build and Run a Custom Docker Image
8. Implement Logging and Monitoring with Prometheus and Grafana

TOTAL:30Hours

COURSE OUTCOMES:

On successful completion of the course the students will be able to

CO1: Understand and implement core DevOps principles and practices.

CO2: Manage version control and CI/CD pipelines.

CO3: Automate infrastructure deployment using Terraform and IaC tools.

CO4: Build, deploy, and orchestrate containerized applications using Docker and Kubernetes.

CO5: Monitor applications and manage configurations with Ansible, Prometheus, and Grafana.

ADHIYAMAAN COLLEGE OF ENGINEERING
(Autonomous), Hosur

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(Accredited by NBA)

CO's-PO's & PSO's MAPPING

322CIP08 DATA STRUCTURES LABORATORY															
CO's / PO's	**PO 1**	**PO2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO7**	**PO8**	**PO9**	**PO 10**	**PO 11**	**PO 12**	**PS0 1**	**PS0 2**	**PSO 3**
CO1	3	1	1	2	1	1	-	-	-	3	3	3	2	2	2
CO2	3	2	1	2	1	1	-	-	-	3	3	3	2	2	2
CO3	3	2	1	3	1	1	-	-	-	3	3	3	2	2	2
CO4	3	2	1	3	1	1	-	-	-	3	3	3	2	2	2
CO5	3	2	1	3	1	1	-	-	-	3	3	3	2	2	2
AVG	3	1.8	1	2.6	1	1	-	-	-	3	3	3	2	2	2

1–Low 2-Medium, 3-High, "-"- No Correlation

INDEX

Exp.No:1 **Provision a Virtual Machine (VM) in AWS using Terraform**

Aim:
To create an EC2 instance with **Ubuntu OS**, install **Nginx**, and configure it using **AWS Management Console** and **Terraform** for deployment automation.

Procedure:
Step 1: EC2 Creation
Step 2: Open Terminal

1. terraform --version
2. aws configure
   - **AWS Access Key ID:** & **AWS Secret Access Key:** (Follow Step 3 to get these credentials.)
   - **Default Region:** EC2 instance region
   - **Default Output Format:** json

Step 3: To Get Credentials

1. **AWS Management Console**
2. **IAM** > Click **User Name**
3. **Create Users**
   - **User Name** > Provide user actions
   - **I want to create an IAM user**
4. **Set Permissions:**
   - Attach policies directly
   - Select **Administrator Access** & **Amazon EC2 Full Access** from the dropdown
5. **Create User** > **Download .csv File**
6. **Select User** > **Security Credentials** > **Access Keys**
7. **Create** > **CLI (I understand)** > **Generate Key**

   Open Terminal:
   - Fill in the details from Step 2.

Step 4: Create Directory:

1. mkdir my-terraform-aws
2. cd my-terraform-aws
   - Create main.tf
   - Write a program
   - Change AMI ID according to the EC2 instance region

Step 5: Initialize Terraform

1. terraform init
2. terraform validate
3. terraform plan
4. terraform apply

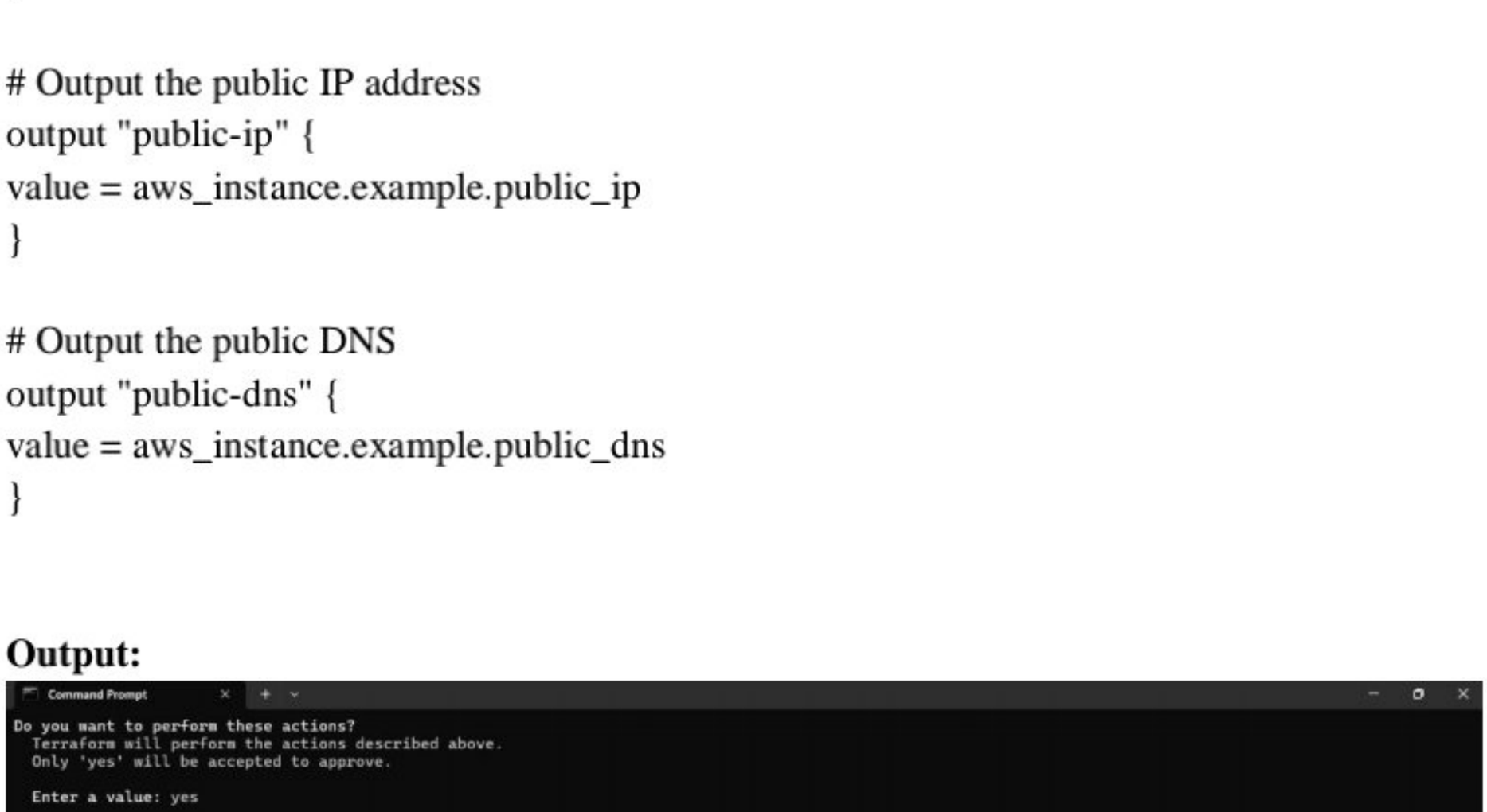
Step 6: Destroy Terraform

1. terraform destroy

Program:

```
# Define the AWS provider

provider "aws" {
region = "us-east-1"
}

# Define the input variable for instance type
variable "instance-type" {
default = "t2.micro"
}

# Specify the EC2 instance details
resource "aws_instance" "example" {
ami           = "ami-04b4f1a9"  # Example AMI ID
instance_type = var.instance-type

tags = {
Name = "Terraform-VM"
}
}

# Output the instance ID
output "instance-id" {
value = aws_instance.example.id
}

# Output the public IP address
output "public-ip" {
value = aws_instance.example.public_ip
}

# Output the public DNS
output "public-dns" {
value = aws_instance.example.public_dns
}
```

Output:



Result:

Thus the above program for Provision a Virtual Machine (VM) in AWS / Azure using Terraform has been executed and verified successfully.

Ex no: 2 Provision a Virtual machine (VM) in Aws with cloud formation

Aim:

To provision a virtual machine (VM) in AWS using an cloud formation template.

Procedure:

Step1: Aws Management console > create and launch instance.

Step 2: Create a cloud formation Template create folder> ec2- template (YAML) > save

EC2.template.yaml:

Aws Template Format Version: '2010-09-09

Resources:

My EC2 Instance:

Type: 'Aws:: EC2:: Instance"

Properties:

Instance Type: t2. micro # Example instance type

Image Id: ami-04bf1 # replace with a valid AMI ID

Key name : cloud #Replace with your key pair name.

Outputs:

Instance Id:

Description: "Instance ID of the created EC2 ins tame

Value: ! Ref My EC2Instance

Public DNS:

Description: public DNS of the EC2 instance"

Value: ! GetAtt My EC2 Instance. Public DNS name

Private IP:

Description: "private IP address of the EC2 instance"

value: ! GetAtt My EC2 Instance. Private Ip

Step3: Stack creation: Aws Management console > cloud formation > create stack> with new Resources (Standard)

prepare template> choose an existing template specify template> upload a template file > others all default > provide stack name > click create shack.

Step 4: Wait for Stack creation: In event tab, program as CREATE COMPLETE

Step 5: Verify the VM: EC2 Dashboard > chock the instance created under Instances.

To show the output in the terminal:

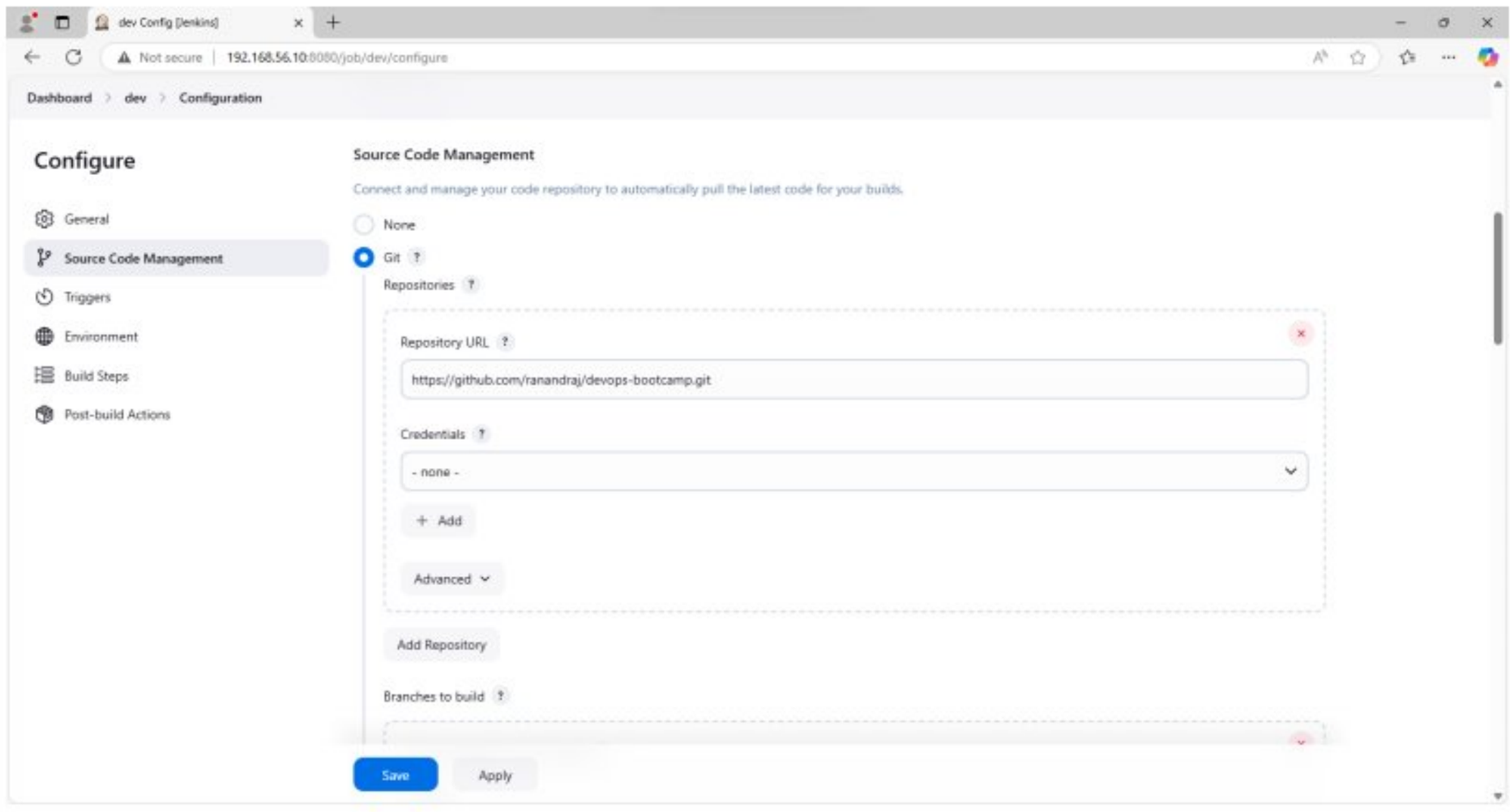
Commands:

AWS configure (chock exp: 1

$ AWS_cloud formation describe-stacks --stack-name

My EC2 Stack region USeast-1

Output:



Result:

Thus to provision a VM in Aws an cloud formation template has been using completed successfully.

Exno:3 Use Ansible To Configure the VM And Install NginX

Aim: Create ansible to configure the VM and install nginx.

Procedure:

Step 1 : In D: Drive open a
New Folder => bootcamp
> > notepad Vagrant file
> > Dir
> > Vagrant validate
> > vagrant up
c:/> users > User> DevOps workshop > Virtual box . box
#copy this file and paste it in bootcamp
> > Dir
> > vagrant box add -- name ubuntu18 virtual box . box
> > vagrant up
> > vagrant ssh vm2

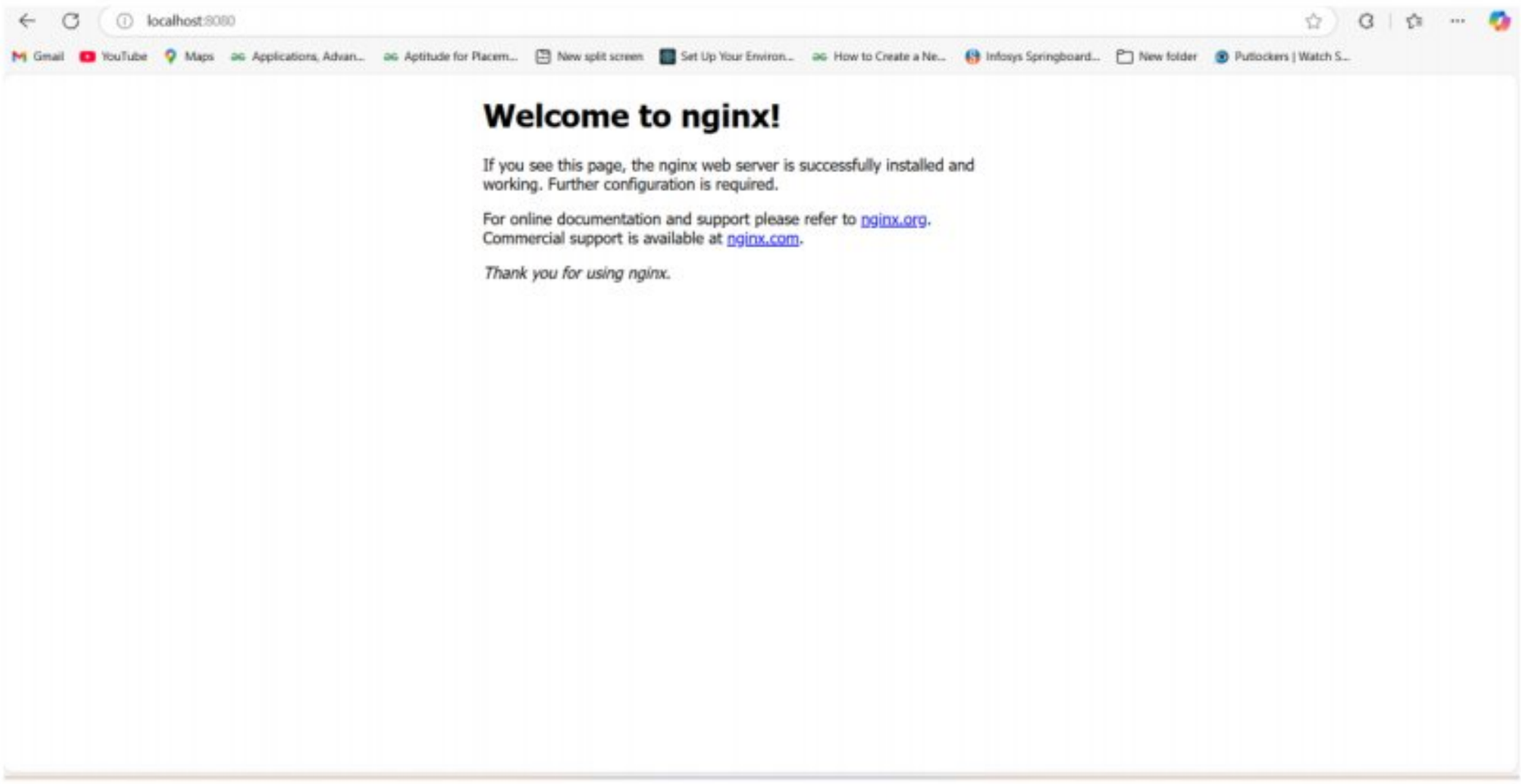
Step 2 : $ sudo apt update - y
$ sudo apt install ansible - y
& ansible -- version
ansible localhost -m ping
$ git clone

Step 3 : $ le
$ cd DevOps - bootcamp
$ le
$ cd Ansible
$ le
$ cat install - nginx. yml
$ cat inventory
$ vi install – nginx . yml
$ cat install – nginx . yml
$ ansible - playbook - i inventory install – nginx . yml

Code:

```
Vagrant.configure("2") do |config|
    config.vm.define "vm2" do |vm2|
        vm2.vm.box = "ubuntu18"
        vm2.vm.network "private_network", ip: "192.168.56.11"
        vm2.vm.provider "virtualbox" do |vb|
            vb.memory = "1024"
            vb.cpus = 2
        end
    end
end
```

Output:



Result:

Thus to create ansible to configure the VM and install nginx is executed successfully.

Ex.no:4 Create a CI pipeline using Jenkins with stages for code checkout, build and test

Aim: to Create a CI pipeline using Jenkins with stages for code checkout, build and test.

procedure:

Step 1 : ensure that the following images present:

- appvm(contains tomcat and mysql installed using ansible)
- controlvm(contains preconfigured jenkins and ansible)

Step 2.check whether if the image file is added in cmd:

- vagrant box list

if not added the use following commands:

- vagrant box add controlvm //address of the image
- vagrant box add appvm //address of the image

Step 3.open command prompt:

- mkdir directory-name
- notepad Vagrant File and save as "Vagrant File" (because while using notepad Vagrant File the file is saved as text file)
  or create a folder and open a notepad and save the file as "Vagrant File"

Step 4.open your oracle virtual box :

- create an network (host only ethernet adapter) if not present and not its ip ex:192.128.56.1

Step 5.write the below code to create 2 virtual machines using the images:

```
Vagrant.configure("2") do |config|
Config.vm.define "vm2" do |vm2|
        vm2.vm.box = "appvm"
        vm2.vm.network = "private network", ip:"ip found(ex:192.128.56.11)"
        vm2.vm.provider "Virtualbox" do |vb|
                vb.memory = 1024
                vb.cpus = 2
                end
        end
Config.vm.define "vm1" do |vm1|
        vm1.vm.box = "controlvm"
        vm1.vm.network = "private network", ip:"ip found(ex:192.128.56.10)"
        vm1.vm.provider "Virtualbox" do |vb|
                vb.memory = 1024
                vb.cpus = 2
                end
        end
    end
```

Step 6.come back to cmd and give the following commands

- vagrant ssh vm1
- ip a //to check the ip of vm1

Step 7.open browser and enter http://"your ip(ex:192.128.56.10)":8080 //to open jenkins

Step 8.use the credentials username: admin password: admin to log in

Step 9.now create a new job (ex:devopjob)

Step 10.give a name to the job and open it as free style project

Step 11.inside the job goto configure>source code management

- select git and paste ur git repo url (ex:https://github.com/anandraj/devopbootcamp)
- change the brach refernce from */master to */main

Step 12. move to build step

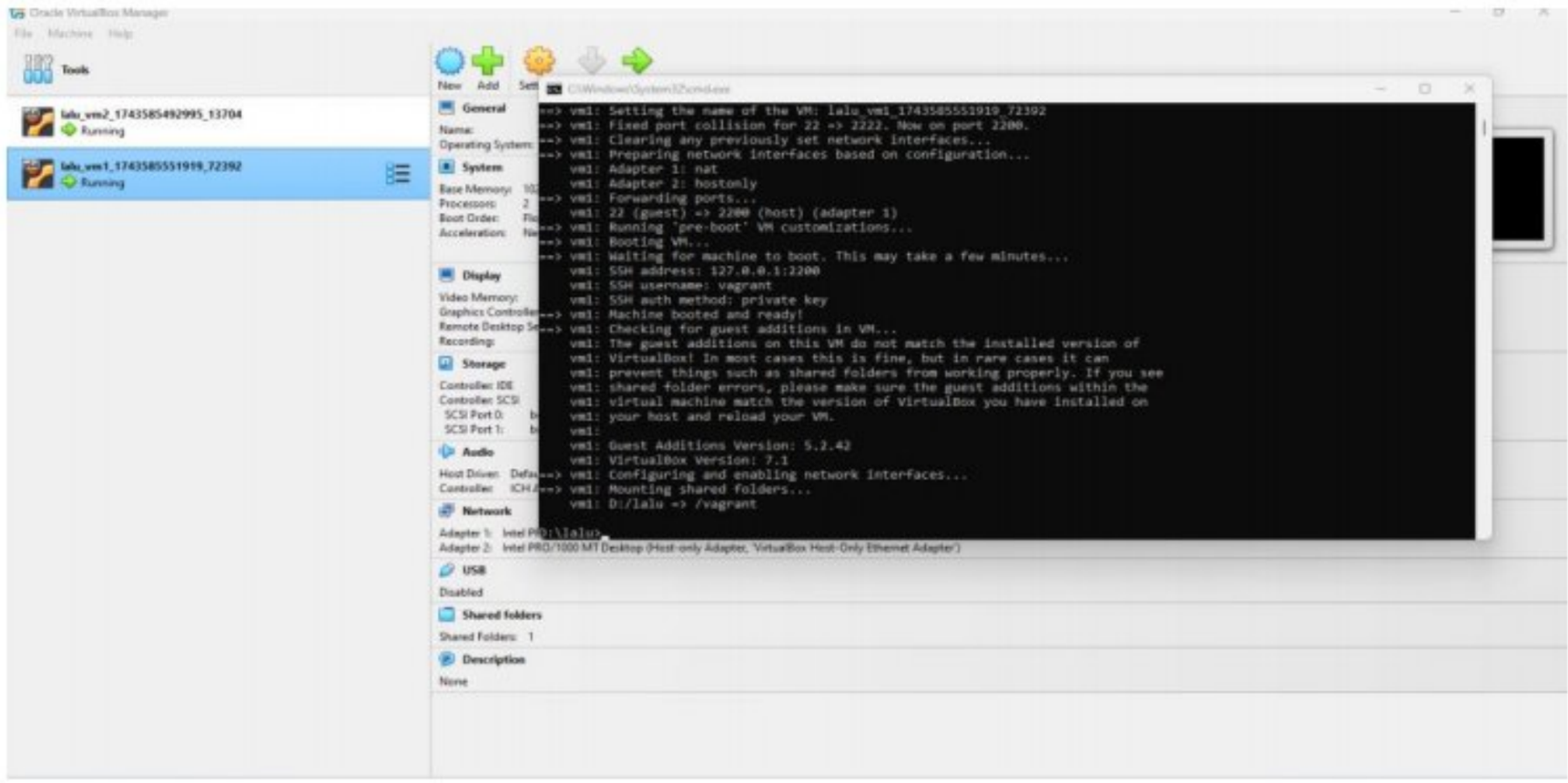
- add build step as invoke top level maven target
- choose a maven (if it doesnt exsist create maven by specifying the version of maven)
- in goals write clean install
- click advanced and enter the pom file locaton in POM (example: onlinebookstore/pom.xml)

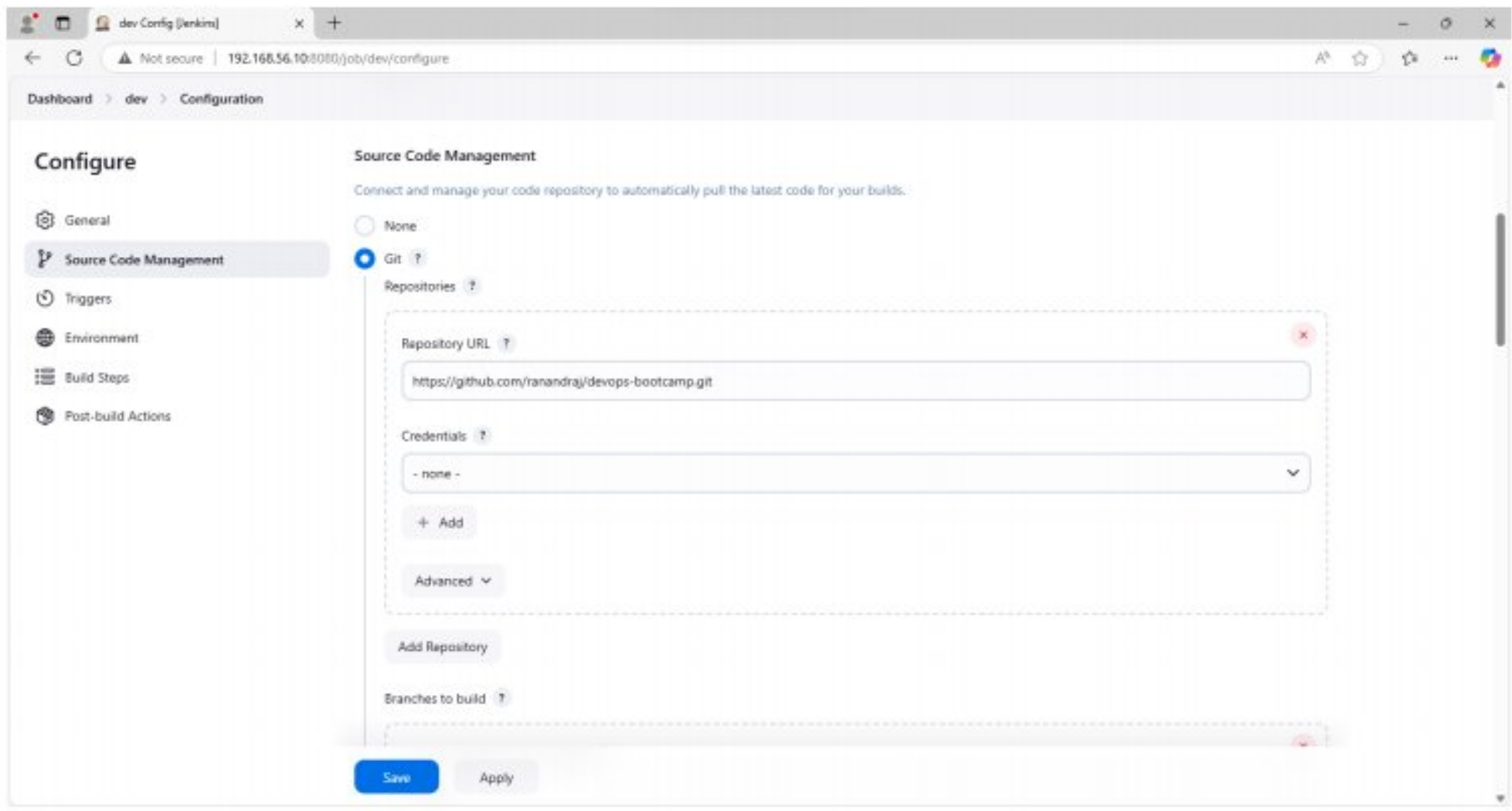
Step 13.save the job

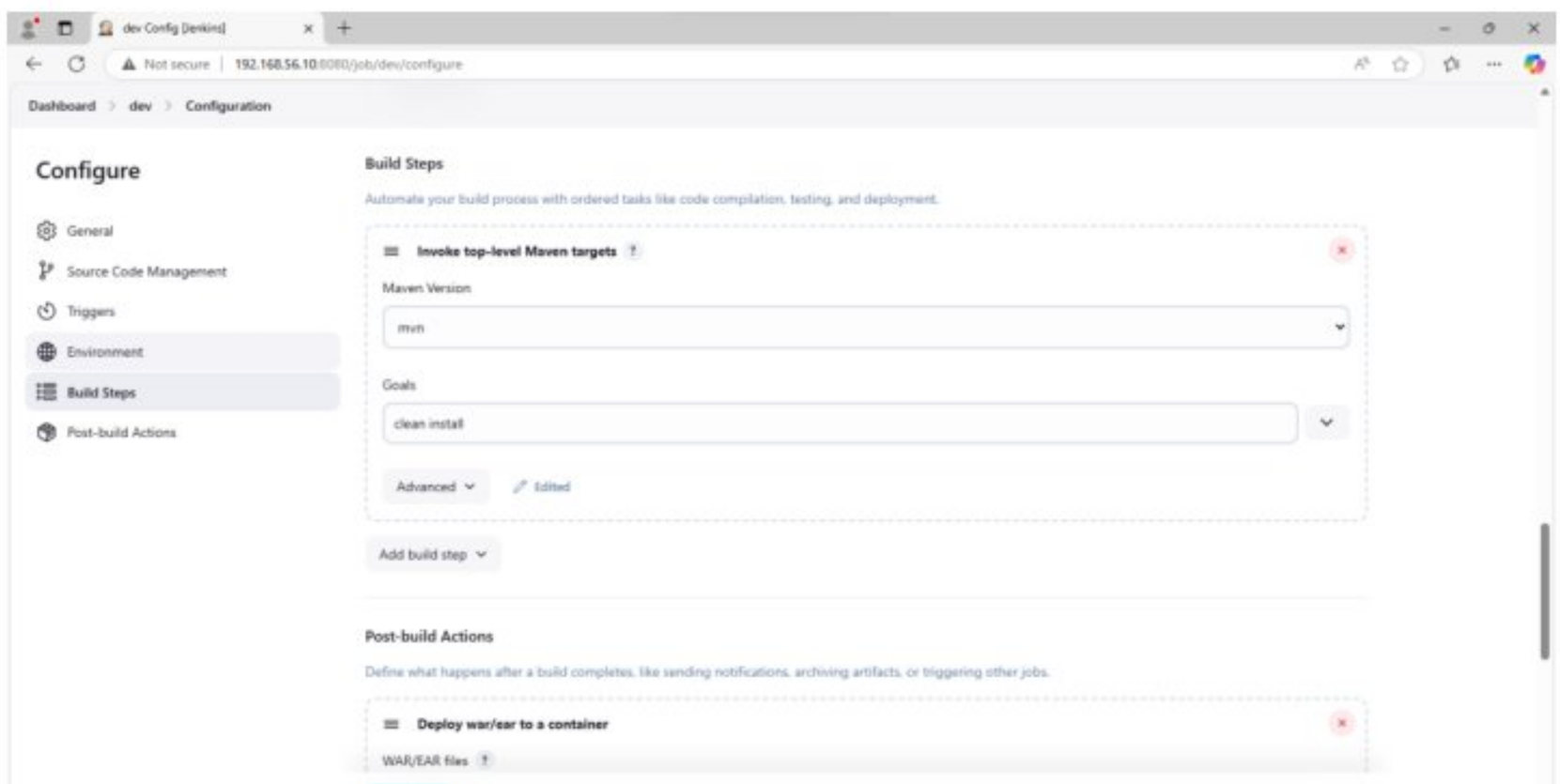
Step 14. click build now and move into the build

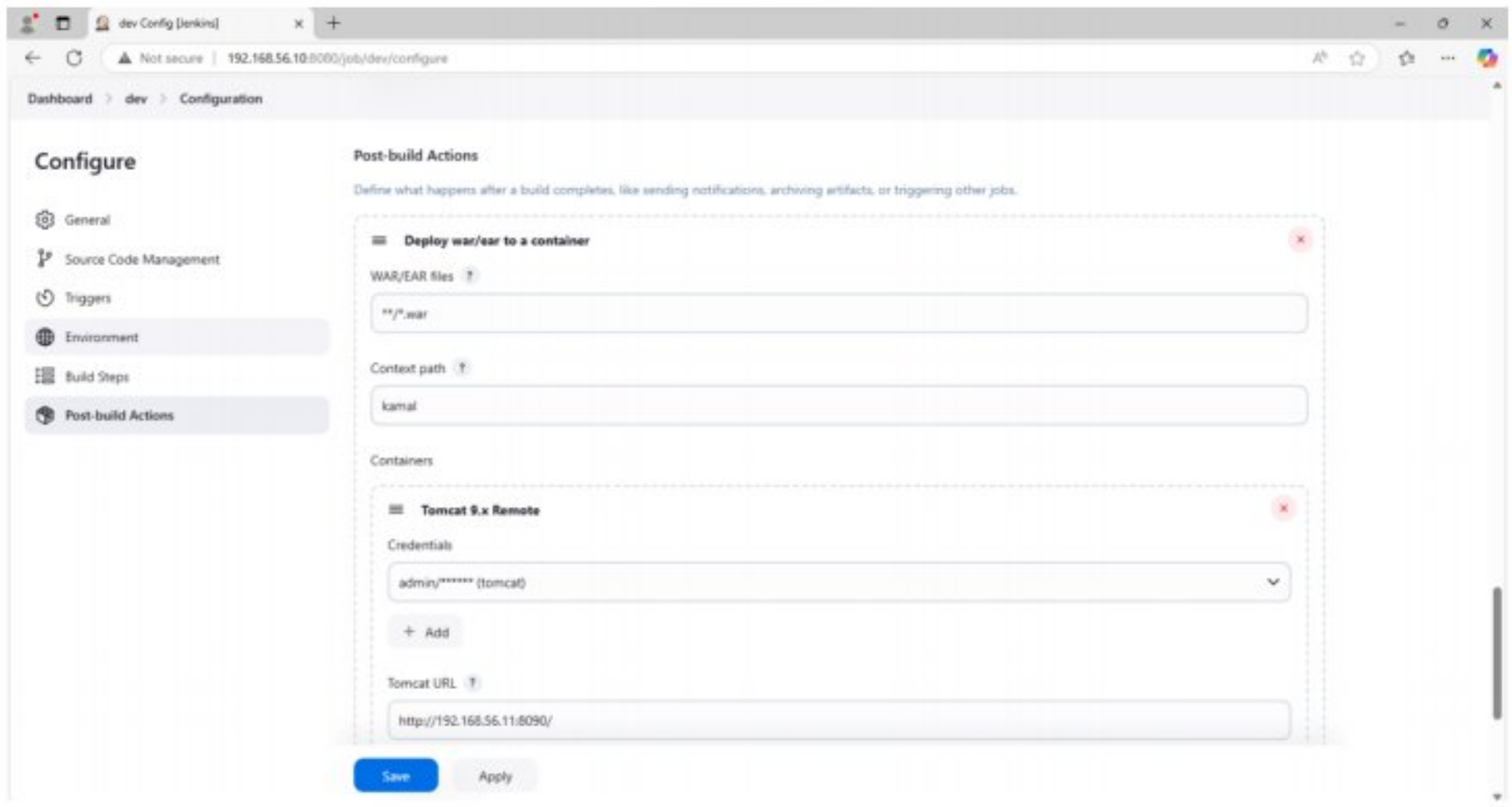
Step 15.click console output and check whether the job is built successfully

Output:

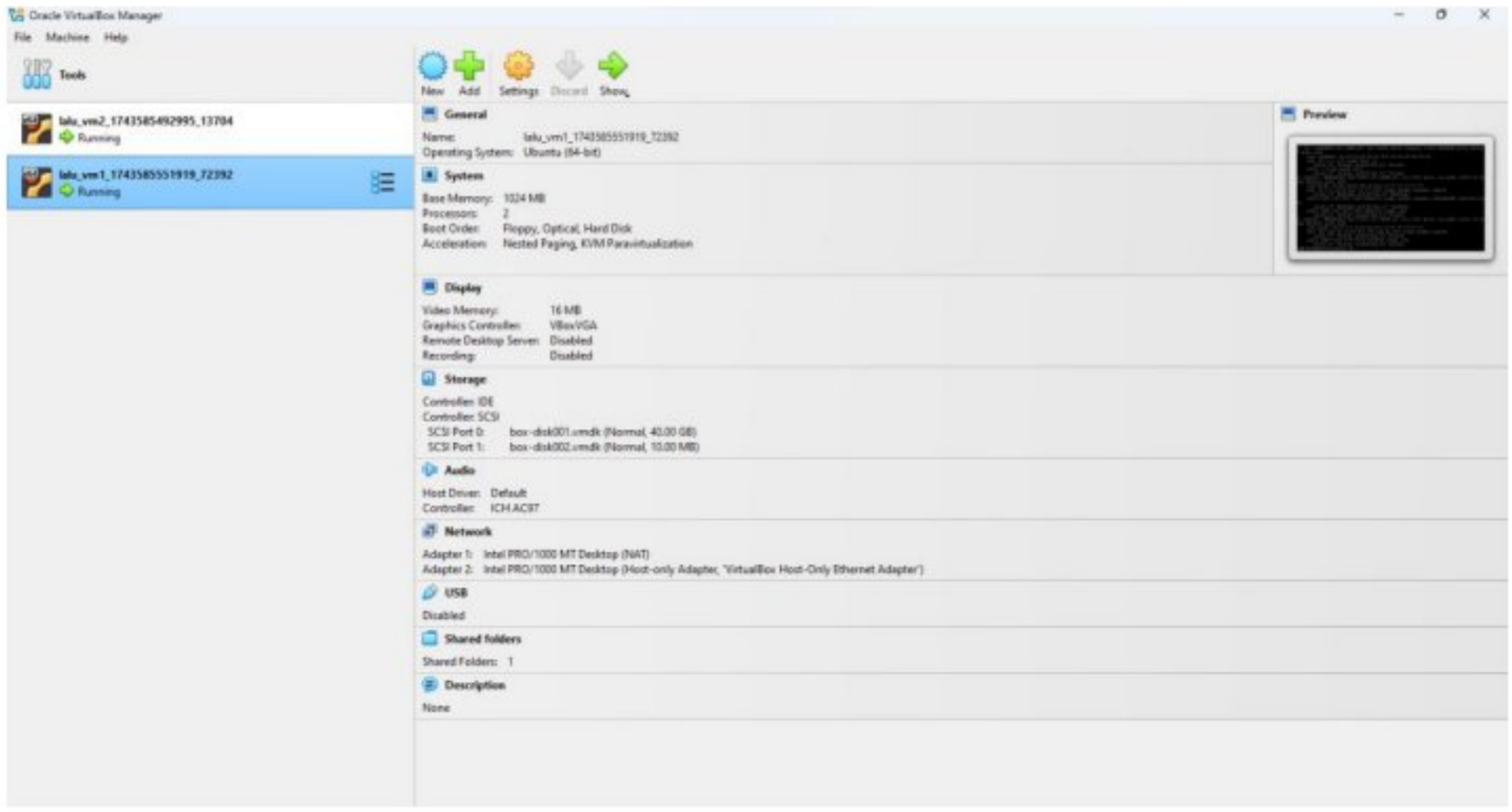


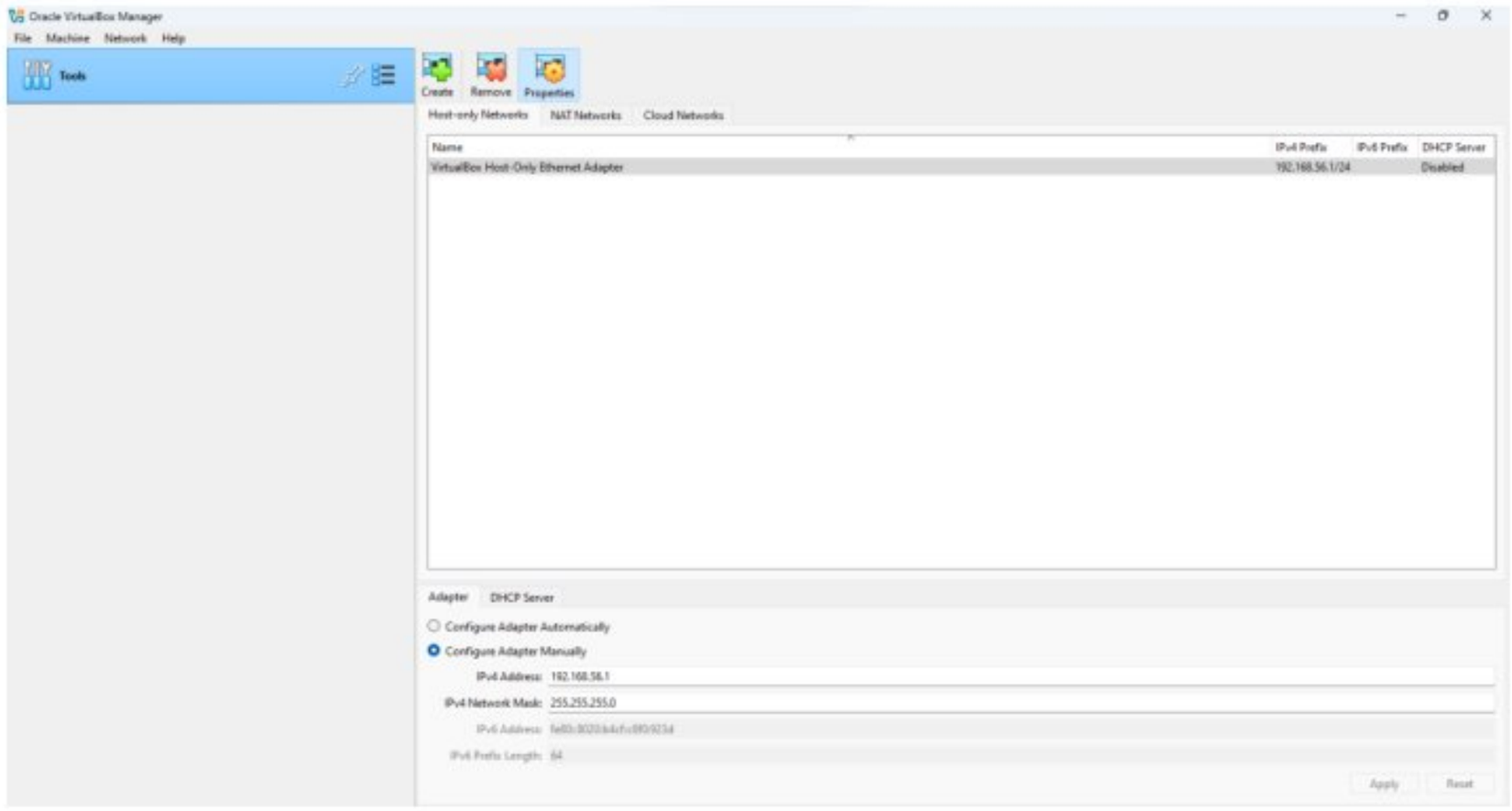





dev Config [Jenkins]
Not secure | 192.168.56.10:8080/job/dev/configure
Dashboard > dev > Configuration
Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
Post-build Actions
Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.
Deploy war/ear to a container
WAR/EAR files
**/*.war
Context path
kamal
Containers
Tomcat 9.x Remote
Credentials
admin/****** (tomcat)
Add
Tomcat URL
http://192.168.56.11:8090/
Save
Apply


vagrantfile - Notepad
File Edit View
Vagrant.configure("2") do |config|
config.vm.define "vm2" do |vm2|
vm2.vm.box = "appvm"
vm2.vm.network "private_network", ip: "192.168.56.11"
vm2.vm.provider "virtualbox" do |vb|
vb.memory = "1024"
vb.cpus = 2
end
end
config.vm.define "vm1" do |vm1|
vm1.vm.box = "controlvm"
vm1.vm.network "private_network", ip: "192.168.56.10"
vm1.vm.provider "virtualbox" do |vb|
vb.memory = "1024"
vb.cpus = 2
end
end
end
Ln 4, Col 57
100%
Windows (CRLF)
UTF-8


Oracle VirtualBox Manager
File Machine Help
Tools
New Add Settings Discard Show
lalu_vm2_1743585492995_13704
Running
lalu_vm1_1743585551919_72392
Running
General
Name: lalu_vm1_1743585551919_72392
Operating System: Ubuntu (64-bit)
System
Base Memory: 1024 MB
Processors: 2
Boot Order: Floppy, Optical, Hard Disk
Acceleration: Nested Paging, KVM Paravirtualization
Preview
Display
Video Memory: 16 MB
Graphics Controller: VBoxVGA
Remote Desktop Server: Disabled
Recording: Disabled
Storage
Controller: IDE
Controller: SCSI
SCSI Port 0: box-disk001.vmdk (Normal, 40.00 GB)
SCSI Port 1: box-disk002.vmdk (Normal, 10.00 MB)
Audio
Host Driver: Default
Controller: ICH AC97
Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT)
Adapter 2: Intel PRO/1000 MT Desktop (Host-only Adapter, 'VirtualBox Host-Only Ethernet Adapter')
USB
Disabled
Shared folders
Shared Folders: 1
Description
None



Result:

Thus the above program for Create a CI pipeline using Jenkins with stages for code checkout, build and test has been executed and verified successfully.

Ex.no:5 Deploy a sample application inside the VM using Jenkins (CD Pipeline)

Aim: to Deploy a sample application inside the VM using Jenkins (CD Pipeline)

procedure:

Step 1. ensure that the following images present:

- appvm(contains tomcat and mysql installed using ansible)
- controlvm(contains preconfigured jenkins and ansible)

Step 2.check whether if the image file is added in cmd:

- vagrant box list
  - if not added the use following commands:
- vagrant box add controlvm //address of the image
- vagrant box add appvm //address of the image

Step 3.open command prompt:

- mkdir directory-name
- notepad VagrantFile and save as "VagrantFile" (becuase while using notepad VagrantFile the file is saved as text file)
  or create a folder and open a notepad and save the file as "VagrantFile"

Step 4.open your oracle virtual box :

- create an network (host only ethernet adapter) if not present and not its ip ex:192.128.56.1

Step 5.write the below code to create 2 virtual machines using the images:

```
Vagrant.configure("2") do |config|
Config.vm.define "vm2" do |vm2|
  vm2.vm.box = "appvm"
  vm2.vm.network = "private network", ip:"ip found(ex:192.128.56.11)"
  vm2.vm.provider "Virtualbox" do |vb|
     vb.memory = 1024
     vb.cpus = 2
     end
 end
 Config.vm.define "vm1" do |vm1|
  vm1.vm.box = "controlvm"
  vm1.vm.network = "private network", ip:"ip found(ex:192.128.56.10)"
  vm1.vm.provider "Virtualbox" do |vb|
     vb.memory = 1024
     vb.cpus = 2
     end
   end
 end
```

Step 6.come back to cmd and give the following commands

- vagrant ssh vm1
- ip a //to check the ip of vm1

Step 7.open browser and enter http://"your ip(ex:192.128.56.10)":8080 //to open jenkins

Step 8.use the credentials username: admin password: admin to log in

Step 9.now create a new job (ex:devopjob)

Step 10.give a name to the job and open it as free style project

Step 11.inside the job goto configure>source code management

- select git and paste ur git repo url (ex:https://github.com/anandraj/devopbootcamp)
- change the brach refernce from */master to */main

Step 12. move to build step

- add build step as invoke top level maven target
- choose a maven (if it doesnt exsist create maven by specifying the version of maven)
- in goals write clean install
- click advanced and enter the pom file locaton in POM (example: onlinebookstore/pom.xml)

Step 13.move to post build actions

- add post build actions as deploy war/ear to a container

in WAR/EAR file fill with **/.war

- give a name to context path(ex: book)
- choose the container tomcat 9x
- add credentials if already present(admin/(tomcat)) if not create credentials username admin and password admin
- enter your tomcat url (ex: http://192.128.56.11:8090)

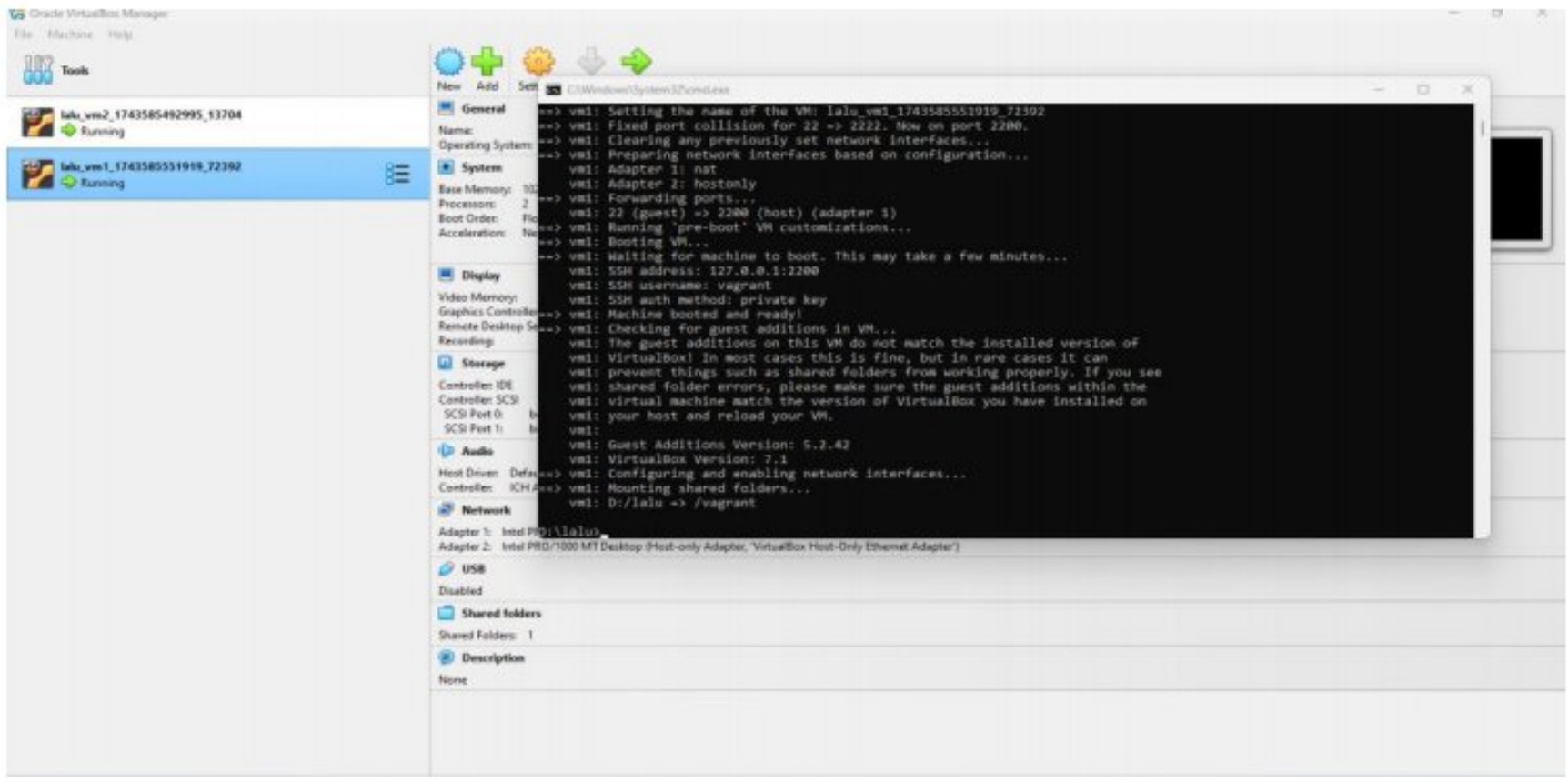
Step 14.save the job

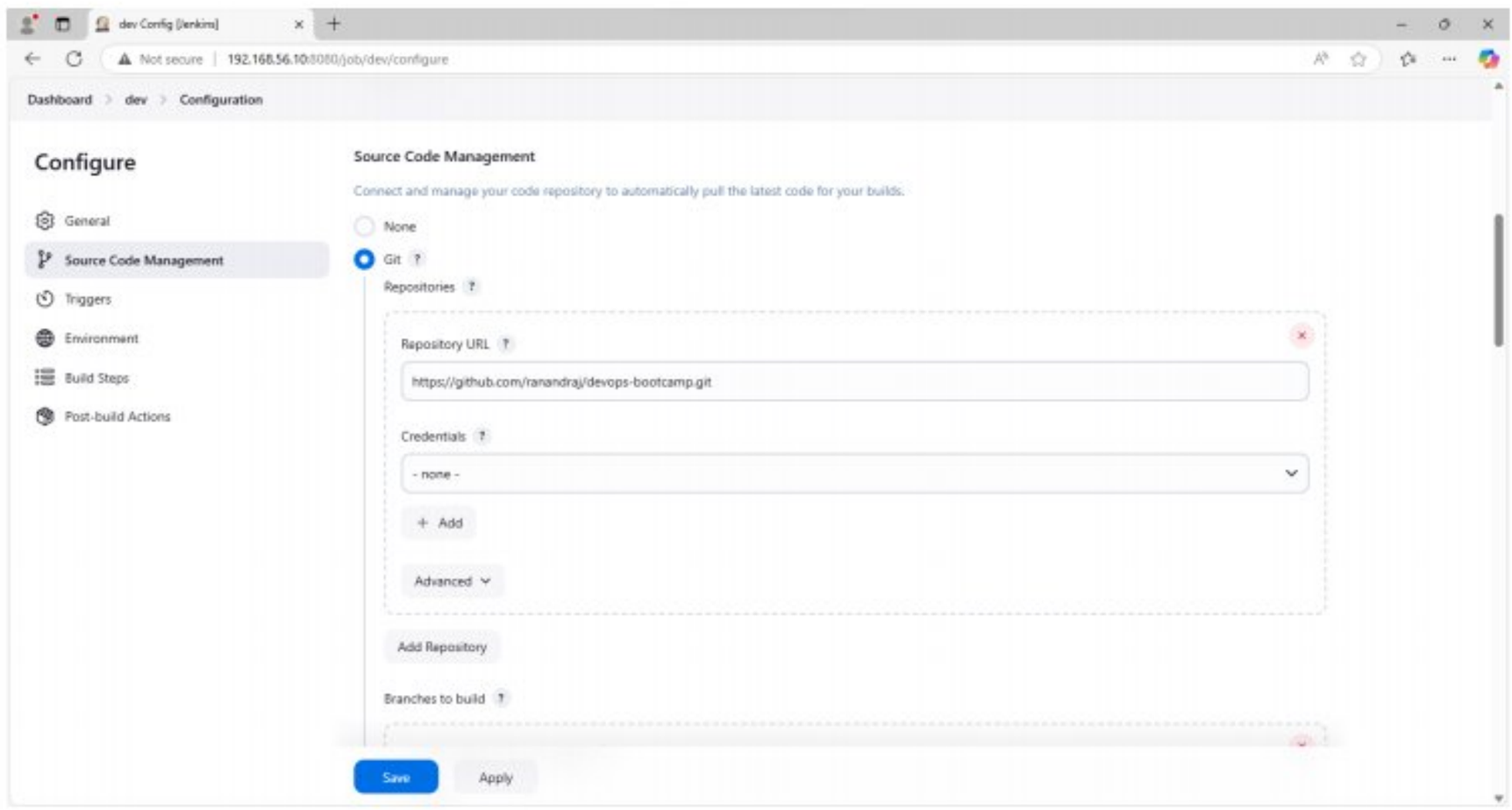
Step 15. click build now and move into the build

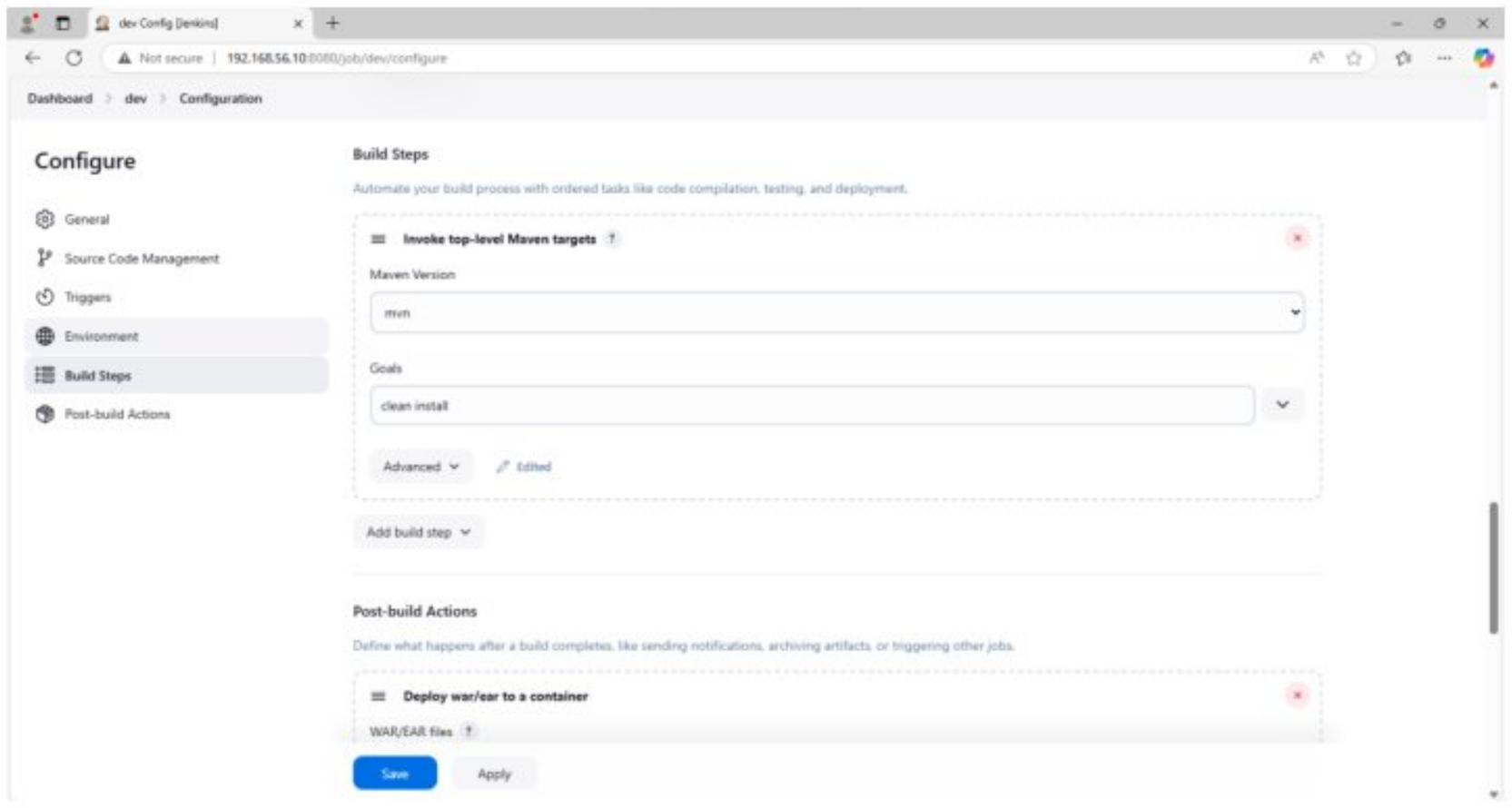
Step 16.click console output and check whether the job is built successfully

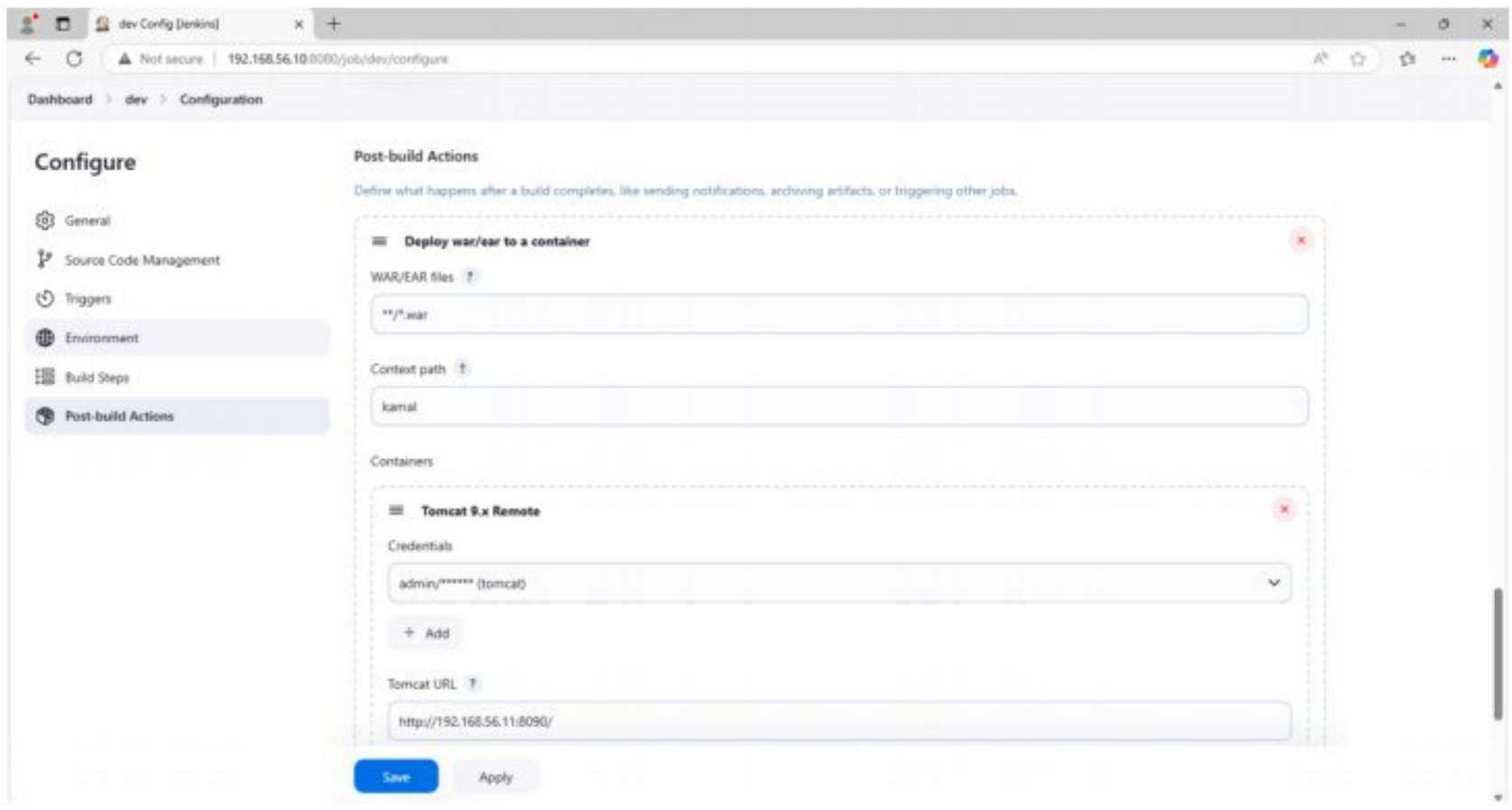
Step 17.now open new tab in broaer and enter the url http://"your tomcat url"/"context path" (ex: http://192.128.56.11:8090/book)

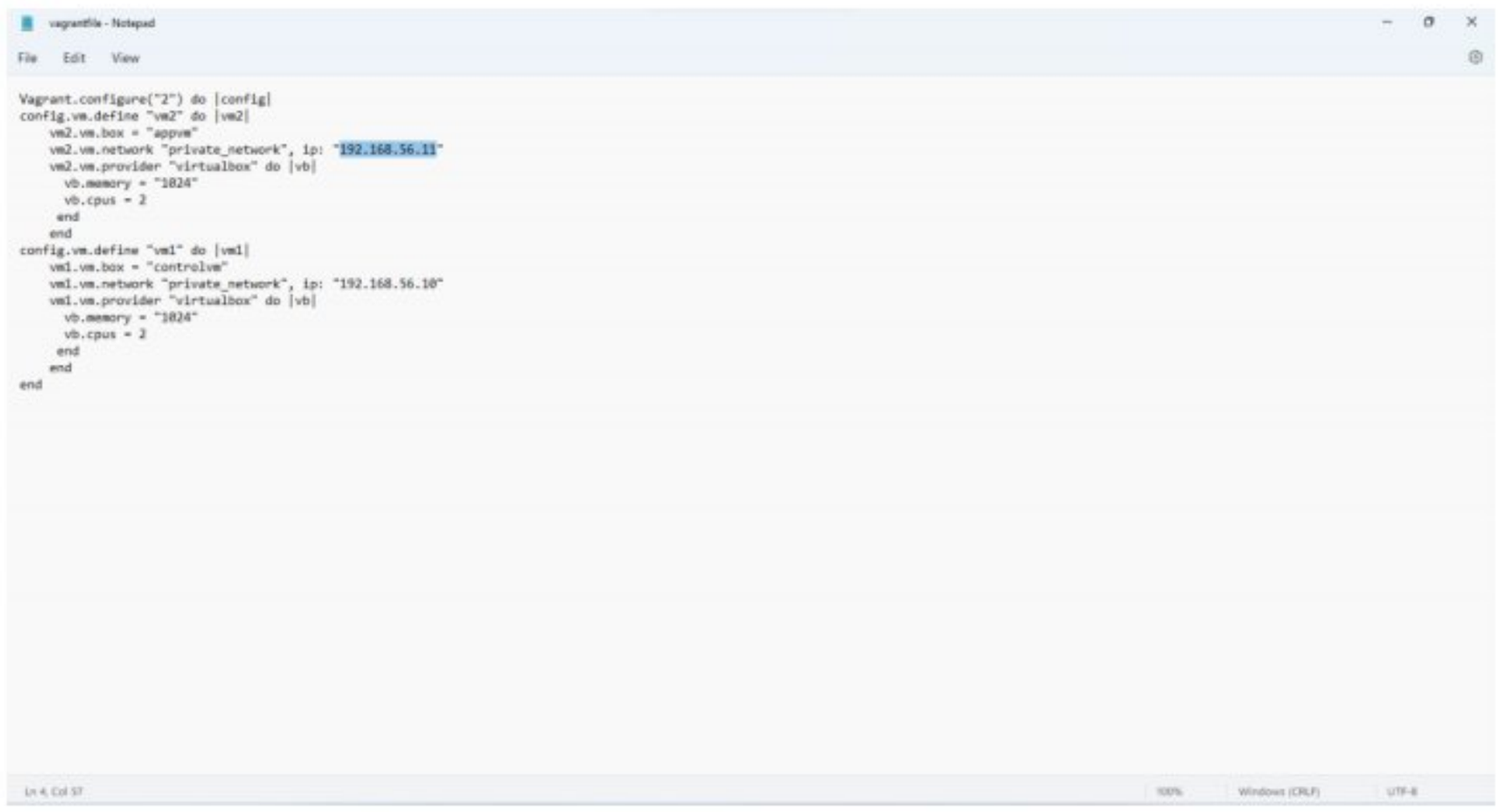
Output:

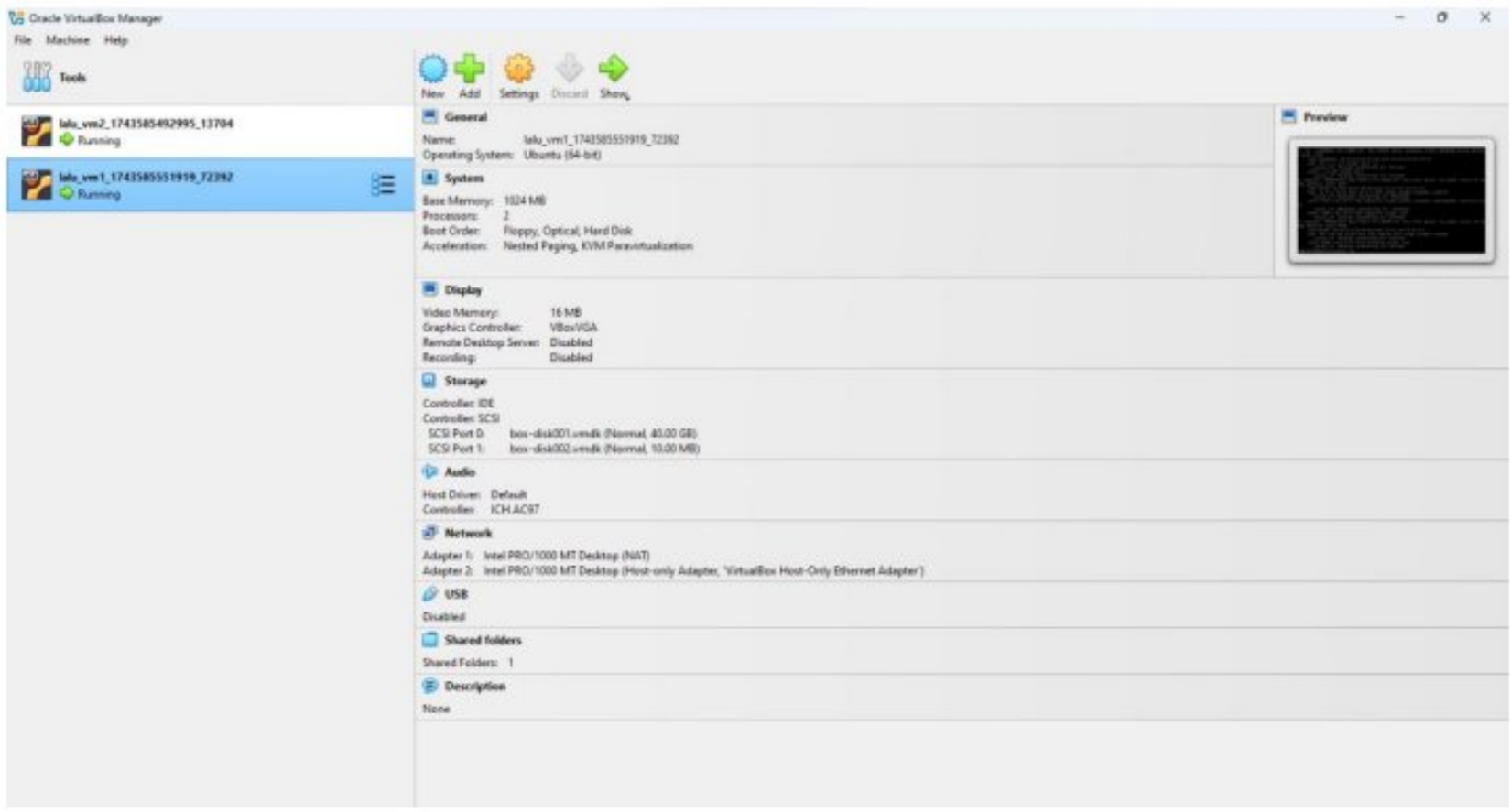


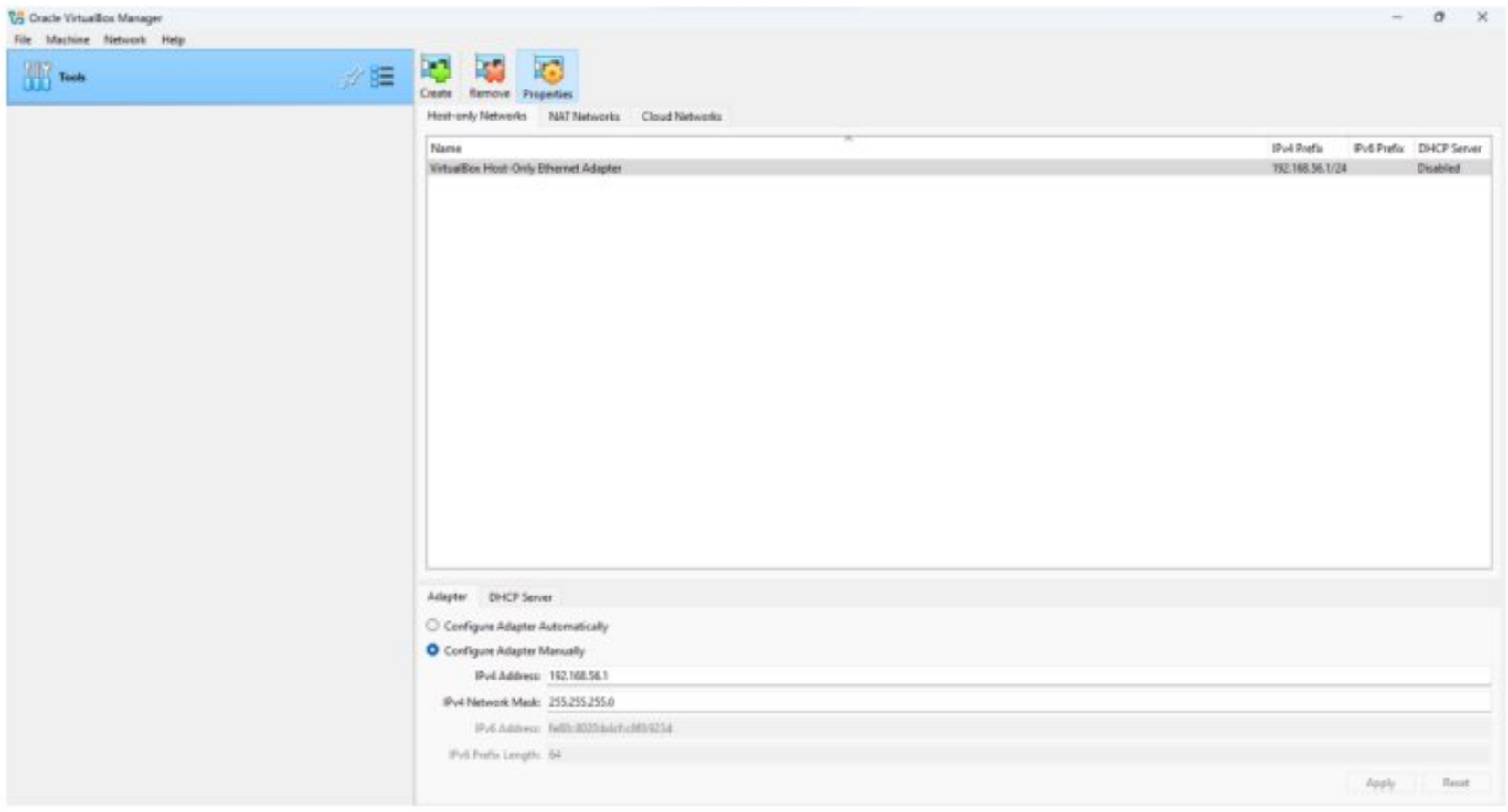



dev Config [Jenkins]
Not secure | 192.168.56.10:8080/job/dev/configure
Dashboard > dev > Configuration
Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
Build Steps
Automate your build process with ordered tasks like code compilation, testing, and deployment.
Invoke top-level Maven targets
Maven Version
mvn
Goals
clean install
Advanced
Edited
Add build step
Post-build Actions
Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.
Deploy war/ear to a container
WAR/EAR files
Save
Apply


dev Config [Jenkins]
Not secure | 192.168.56.10:8080/job/dev/configure
Dashboard > dev > Configuration
Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
Post-build Actions
Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.
Deploy war/ear to a container
WAR/EAR files
**/*.war
Context path
kamal
Containers
Tomcat 9.x Remote
Credentials
admin/****** (tomcat)
Add
Tomcat URL
http://192.168.56.11:8090/
Save
Apply


vagrantfile - Notepad
File Edit View
Vagrant.configure("2") do |config|
config.vm.define "vm2" do |vm2|
vm2.vm.box = "appvm"
vm2.vm.network "private_network", ip: "192.168.56.11"
vm2.vm.provider "virtualbox" do |vb|
vb.memory = "1024"
vb.cpus = 2
end
end
config.vm.define "vm1" do |vm1|
vm1.vm.box = "controlvm"
vm1.vm.network "private_network", ip: "192.168.56.10"
vm1.vm.provider "virtualbox" do |vb|
vb.memory = "1024"
vb.cpus = 2
end
end
end
Ln 4, Col 57
100%
Windows (CRLF)
UTF-8





Result:

Thus the above program for Deploy a sample application inside the VM using Jenkins (CD Pipeline) has been executed and verified successfully.

Ex.No:6 RUN A CONTAINER USING PRE-BULID DOCKER IMAGE

AIM:

To run a container using pre-build docker image .

PROCEDURE:

Step 1: Verify Docker Installation

1. Open a Command Prompt.
2. Run the following command to check if Docker is installed:

docker –version

Step 2: Pull the Pre-Built Docker Image

If the image is not already available locally, pull it from Docker Hub using the command in the command prompt

docker pull nginx:latest

Step 3: Run a Container from the Image

Use the docker run command to start a container in the command prompt

docker run -d --name my_nginx -p 8080:80 nginx:latest

Explanation:

-d → Runs the container in detached mode (background process).

--name my_nginx → Assigns a custom name to the container.

-p 8080:80 → Maps port 8080 on the host to port 80 inside the container.

Step 4: Verify the Running Container

Check running containers:

docker ps

Step 5: Access the Running Container

If the container runs a web service, open a web browser and go to:

http://localhost:8080

Step 6: Stop and Remove the Container

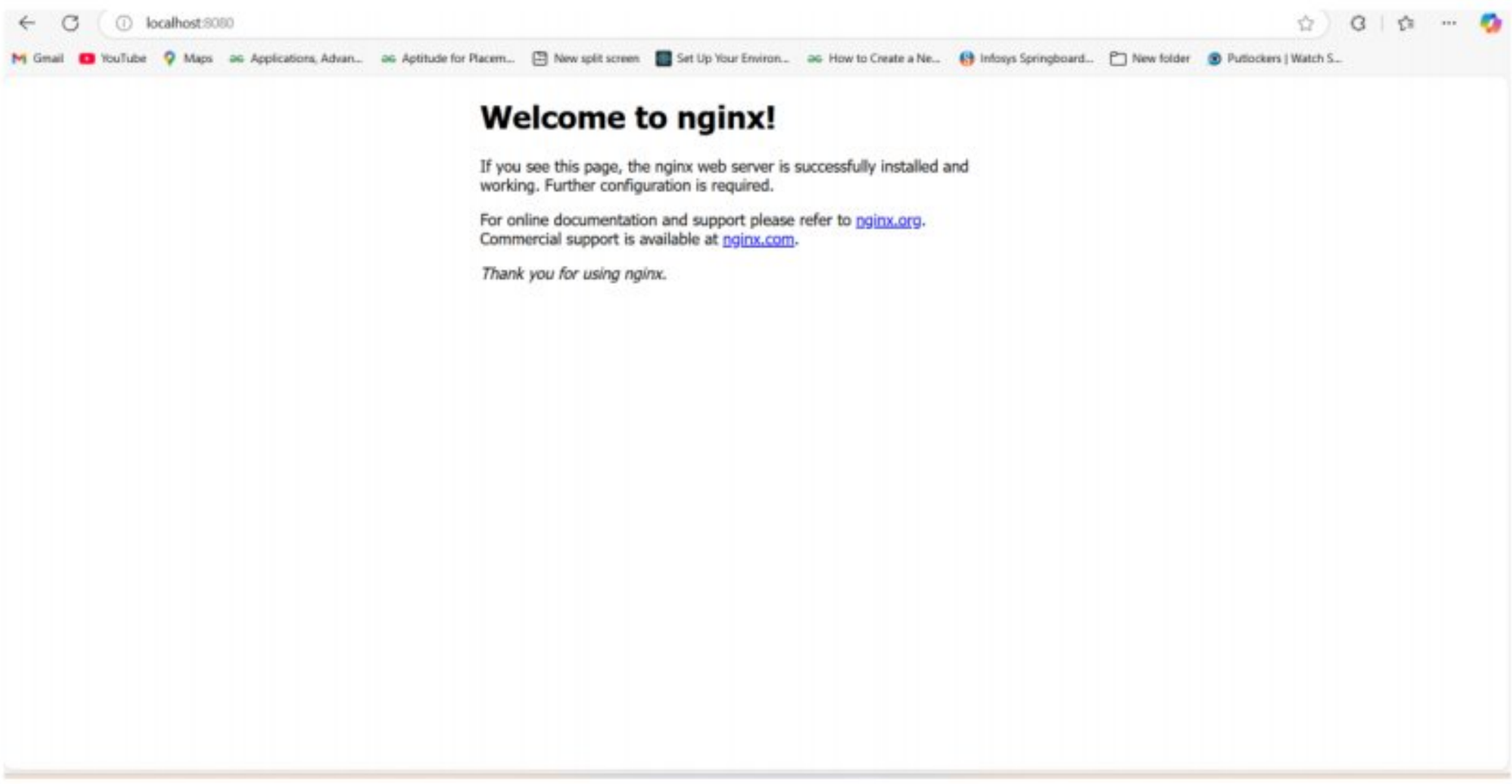
To Stop the running container:

docker stop my_nginx

Remove the container:

docker rm my_nginx

Output:



Result:

Thus the above program for run a container using pre-build docker image has been executed and verified successfully.

Ex.No : 7 BUILD AND RUN A CUSTOM DOCKER IMAGE

AIM:

To build and run a custom docker image.

PROCEDURE:

Step 1: Create a Project Directory

Open a terminal (Command Prompt or PowerShell in Windows).

Create a new directory and navigate into it:

mkdir my-custom-image

cd my-custom-image

Step 2: Create a Dockerfile

Inside the my-custom-image directory, create a new file named Dockerfile.

Open the file in a text editor and add the following content:(Use the command *notepad Dockerfile*)

```
# Use Nginx as the base image
FROM nginx:latest
# Copy the image file to the default Nginx HTML directory
COPY image.jpg /usr/share/nginx/html/image.jpg
# Expose port 80 to allow web access
EXPOSE 80
# Start Nginx when the container runs
CMD ["nginx", "-g", "daemon off;"]
```

Save and close the file.

Step 3:Build the Docker image

docker build -t my-image-server .

Run the container

docker run -d --name image-container -p 9099:80 my-image-server

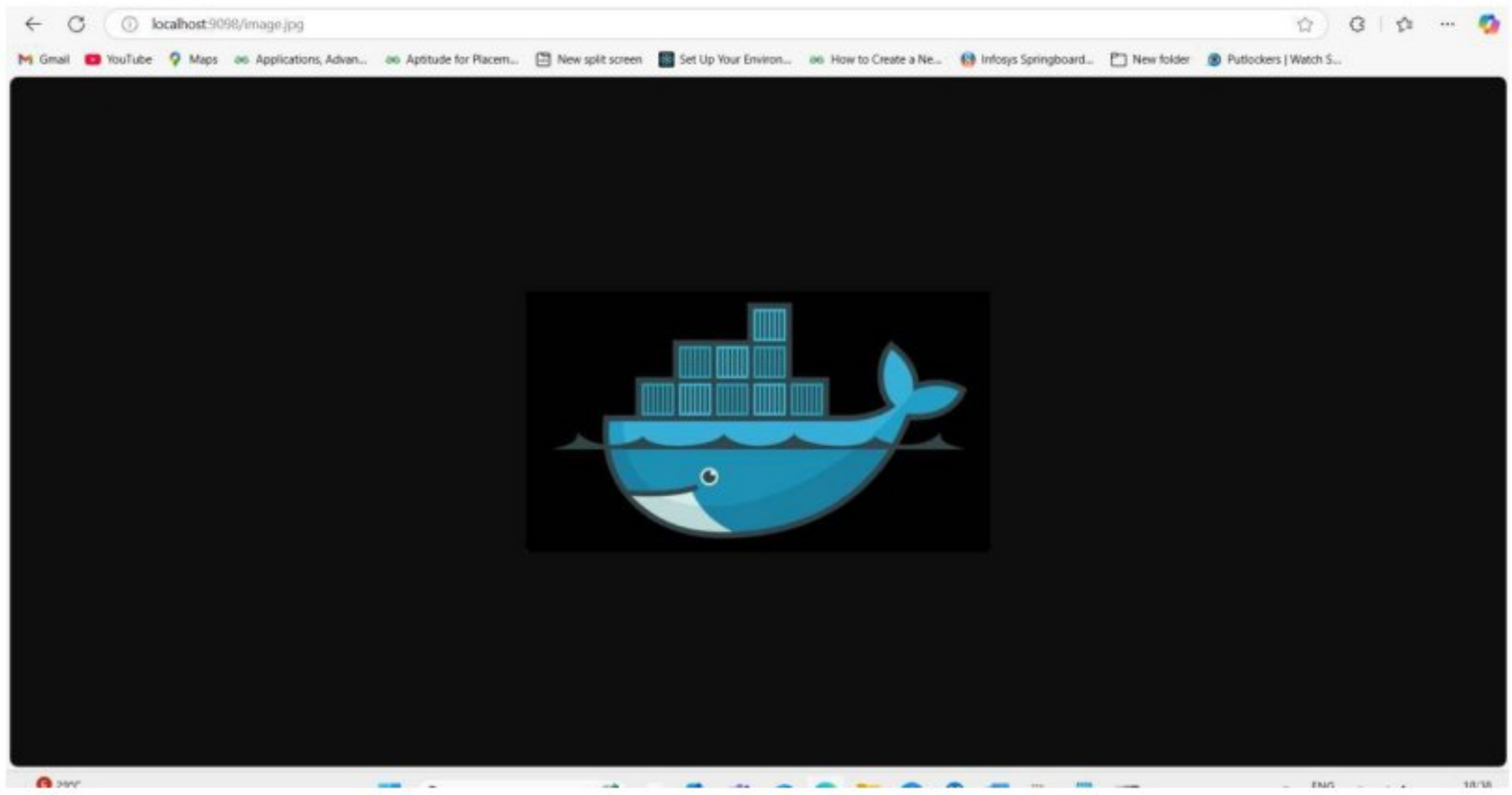
Step 4: Check the running container

docker ps

Step 5:Access the image in your browser

http://localhost:9099/image.jpg

Output:



Result:

Thus the above program for build and run a custom docker image has been executed and verified successfully.

Ex No : 8 IMPLEMENT LOGGING AND MONITORING WITH PROMETHEUS AND GRAFANA

Aim:

To set up a robust logging and monitoring system that enables real-time metric collection, visualization, and alerting by using Prometheus and Grafana on an EC2 instance running Ubuntu.

Procedure

Step 1: Launch an EC2 Instance (Ubuntu)

1. Create an EC2 instance using AWS Management Console or Terraform.
2. Connect to the instance via SSH:

   bash
   ssh -i your-key.pem ubuntu@your-instance-ip
3. Update and upgrade system packages:

   bash
   sudo apt update && sudo apt upgrade -y

Step 2: Install Prometheus

1. Download Prometheus:

   bash
   wget https://github.com/prometheus/prometheus/releases/latest/download/prometheus-linux-amd64.tar.gz
2. Extract the downloaded file:

   bash
   tar xvf prometheus-linux-amd64.tar.gz
3. Move Prometheus binaries:

   bash
   sudo mv prometheus-linux-amd64/prometheus /usr/local/bin/
   sudo mv prometheus-linux-amd64/promtool /usr/local/bin/
4. Create necessary directories:

   bash
   sudo mkdir -p /etc/prometheus /var/lib/prometheus
5. Move the configuration file:

   bash
   sudo mv prometheus-linux-amd64/prometheus.yml /etc/prometheus/
6. Create a Prometheus systemd service:

   bash
   sudo nano /etc/systemd/system/prometheus.service
   Add the following configuration:
   ini
   [Unit]
   Description=Prometheus Monitoring
   Wants=network-online.target
   After=network-online.target

   [Service]
   User=root
   ExecStart=/usr/local/bin/prometheus --config.file=/etc/prometheus/prometheus.yml --storage.tsdb.path=/var/lib/prometheus

   [Install]
   WantedBy=multi-user.target
7. Reload and start Prometheus:

   bash
   sudo systemctl daemon-reload

sudo systemctl start prometheus
sudo systemctl enable prometheus

Step 3: Install Grafana

1. Add the Grafana repository:

   bash
   sudo apt install -y software-properties-common
   sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"
2. Install Grafana:

   bash
   sudo apt update
   sudo apt install grafana -y
3. Start and enable Grafana:

   bash
   sudo systemctl start grafana-server
   sudo systemctl enable grafana-server

Step 4: Configure Prometheus in Grafana

1. Access Grafana's web interface:
2. http://your-ec2-public-ip:3000
3. Log in (default credentials: admin/admin).
4. Navigate to **Configuration → Data Sources**.
5. Add **Prometheus** as a data source, and set the URL to:

   text
   http://localhost:9090
6. Click **Save & Test**.

Step 5: Install Node Exporter

1. Download Node Exporter:

   bash
   wget https://github.com/prometheus/node_exporter/releases/latest/download/node_exporter-linux-amd64.tar.gz
2. Extract and move binaries:

   bash
   tar xvf node_exporter-linux-amd64.tar.gz
   sudo mv node_exporter-linux-amd64/node_exporter /usr/local/bin/
3. Create a Node Exporter systemd service:

   bash
   sudo nano /etc/systemd/system/node_exporter.service
   Add the following configuration:
   ini
   [Unit]
   Description=Node Exporter
   Wants=network-online.target
   After=network-online.target

   [Service]
   User=root
   ExecStart=/usr/local/bin/node_exporter

   [Install]
   WantedBy=multi-user.target
4. Reload and start the Node Exporter service:

   bash

sudo systemctl daemon-reload
sudo systemctl start node_exporter
sudo systemctl enable node_exporter

5. Add Node Exporter to the Prometheus configuration:

bash
sudo nano /etc/prometheus/prometheus.yml
Add the scrape configuration:
yaml
scrape_configs:
- job_name: "node"
static_configs:
- targets: ["localhost:9100"]

6. Restart Prometheus:

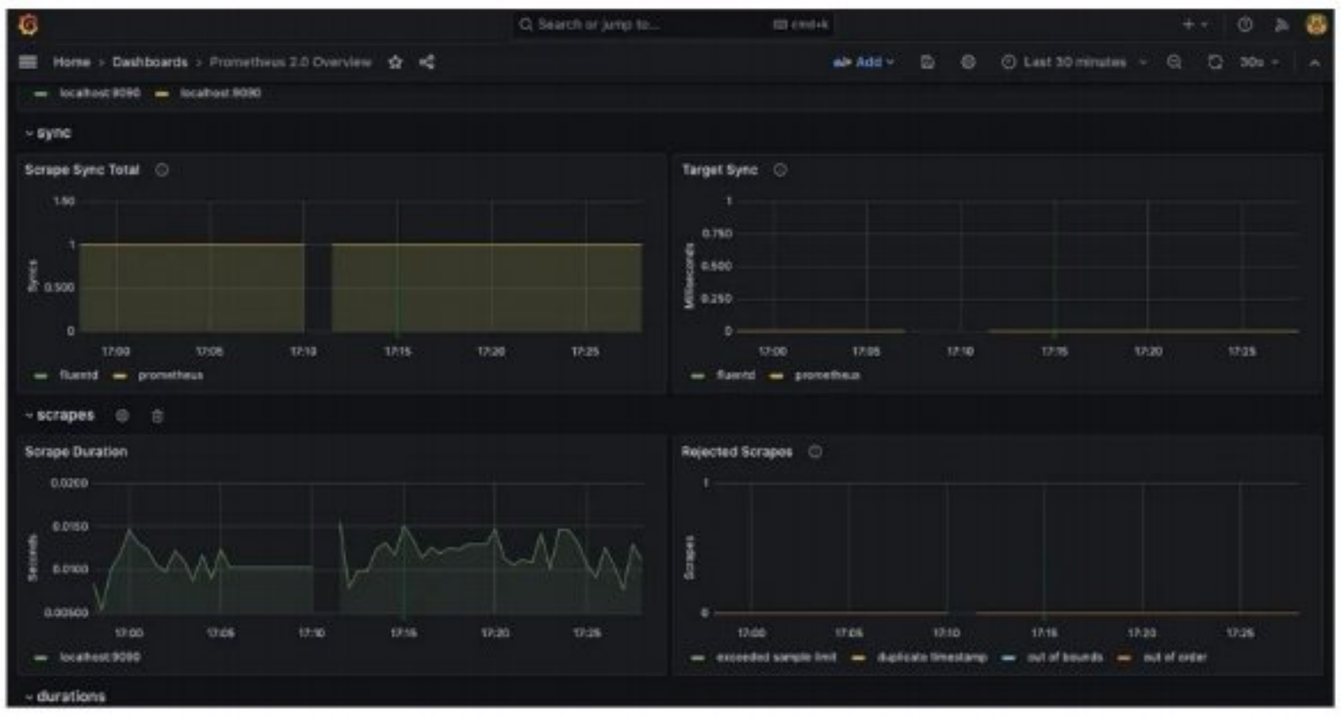
bash
sudo systemctl restart prometheus

Step 6: Configure Grafana Dashboards

1. In Grafana, navigate to **Dashboards → Import**.
2. Use Node Exporter Dashboard ID: **1860**.
3. Set Prometheus as the data source.
4. Click **Import** to visualize system metrics.

Output:

Prometheus Interface



Grafana Dashboard

Result:

Thus To set up a robust logging and monitoring system that enables real-time metric collection, visualization, and alerting by using Prometheus and Grafana on an EC2 instance running Ubuntu is executed successfully.