SRI SHAKTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

COIMBATORE - 641062

POWERING THE YOUTH
EMPOWERING THE NATION

21CY513 – Cloud Computing and DevOps Laboratory

DEPARTMENT OF

COMPUTER SCIENCE ENGINEERING

(CYBER SECURITY)

SRI SHAKTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

21CY513 – Cloud Computing and DevOps Laboratory

NAME:______________________**ROLL NO:** _____________________________

CLASS:______________________**BRANCH:**_____________________________

ACADEMIC YEAR: 2025 - 2026 **BATCH:** 2023 - 2027 **SEMESTER:** V

Certified and bonafide record of work done by ………..…………….……………

Place: Coimbatore

Date:

Staff In-Charge **Head of the Department**

University Register Number: ………………………………………………...

Submitted for the University Practical Examination held on ……..…………

INTERNAL EXAMINER **EXTERNAL EXAMINER**

LIST OF EXPERIMENTS

S.NO	DATE	TITLE OF EXPERIMENT	MARK	SIGN
1		Phases in DevOps		
2		Tools in DevOps		
3		Git		
4		Installation of VM and Windows OS		
5		Create Maven Build Pipeline in Azure		
6		Run Regression Tests Using Maven Build pipeline in Azure		
7		Create an AWS Ec2 instance using Terraform		
8		Resolving GIT Merge Conflicts		
9		Docker Installation and Containerization		
10		Deploy NGINX application using Jenkins		
11		Install Jenkins in cloud		
12		Create CI Pipeline using Jenkins		
13		Ansible Installation on Ubuntu		
14		Ansible Playbooks and Roles		
15 (A)		Installation on GIT in Local Server and Configure		
15 (B)		Install NGINX ON Cloud Instance		
16		Infrastructure as code (IAAC) with Terraform		
17		Launching webapp using Maven and Tomcat		

18		Creating Docker Image With Tomcat and Run		
19		Installing Prometheus on Ubuntu		
20		Install Grafana in Ubuntu		

INTERNAL EXAMINER **EXTERNAL EXAMINER**

Ex.no: 01	**PHASES IN DEVOPS**
DATE:	

AIM:

To explain about eight phases in DevOps.

PHASES IN DEVOPS:

1. Plan: Professionals determine the commercial need and gather end-user opinions throughout this level. In this step, they design a project plan to optimize business impact and produce the intended result.

2. Code – During this point, the code is being developed. To simplify the design process, the developer team employs lifecycle DevOps tools and extensions like Git that assist them in preventing safety problems and bad coding standards.

3. Build – After programmers have completed their tasks, they use tools such as Maven and Gradle to submit the code to the common code source.

4. Test – To assure software integrity, the product is first delivered to the test platform to execute various sorts of screening such as user acceptability testing, safety testing, integration checking, speed testing, and so on, utilizing tools such as JUnit, Selenium, etc.

5. Release – At this point, the build is prepared to be deployed in the operational environment. The DevOps department prepares updates or sends several versions to production when the build satisfies all checks based on the organizational demands.

6. Deploy – At this point, Infrastructure-as-Code assists in creating the operational infrastructure and subsequently publishes the build using various DevOps lifecycle tools.

7. Operate – This version is now convenient for users to utilize. With tools including Chef, the management department take care of server configuration and deployment at this point.

8.Monitor – The DevOps workflow is observed at this level depending on data gathered from consumer behavior, application efficiency, and other sources. The ability to observe the complete surroundings aids teams in identifying bottlenecks affecting the production and operations teams' performance.

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus the phases of **DevO**ps has been explained successful.

Ex.no: 02	TOOLS IN DEVOPS
DATE:	

AIM:

To study and explain about the tools used in DevOps.

TOOLS IN DEVOPS:

1.Docker

Docker is an open-source platform built on Linux. A group of DevOps automation tools called Docker builds containerized environments for apps, improving their portability, security, and test-time conflict reduction. DevOps can quickly and effectively build and execute apps thanks to Docker. Docker apps are independent of platforms and operating systems.

2.Kubernetes

It's a platform for container orchestration and automation that is very well-liked by DevOps teams. It integrates effectively with Docker and manages containers on a wide scale By automating distribution and scheduling across a cluster, Kubernetes manages hundreds of containers and can be used to deploy containerized software to clusters rather than individual workstations.

3.Bamboo

Atlassian's Bamboo is a CI product. Bamboo advertises "integrations that matter" and offers a Small Teams package with a donation to charity component. Bamboo includes prebuilt features similar to Jenkins, thus managing fewer plugins is easier. Compared to open-source alternatives, the interface is more time-efficient and highly intuitive.

4.Raygun

Application performance monitoring (APM) tool Raygun is the best in its field and offers superior monitoring and crash reporting. By pointing to the faulty line of the function,API call, or source code, Raygun aids DevOps in finding performance issues.

5.GitHub

One of the best DevOps automation technologies for developer collaboration since 2000 is GitHub. Developers can quickly iterate on existing code (with notifications provided to team members instantly), and they can also perform speedy rollbacks in the event that an error or unintended consequence occurs.

6.Git

Git is a modern distributed version control technology that is free, cross-platform, and open-source. It has strong support for non-linear development and can quickly and effectively manage everything from small to extremely large projects. Git's key features include local branching, easy staging spaces, and different workflows.

7.SVN(Subversion)

SVN is an advanced VCS that offers efficient windows support and has a perfect blend with GUI (such as TortoiseSVN) and is free and open to use without any charges. By implementing SVN, a team can easily be managed for performing multiple tracking, file locking, MIME support, etc. Besides this, it's a client-server repository model which makes it efficient for a small or large segment team to sync their tasks so that the workflow can be managed in pace.

8.Gitlab

Version control is a means to maintain track of code changes so that, in the event of a problem, we may compare several code versions and roll back to any desired earlier version. It is absolutely necessary that several developers are constantly working on/changing the source code.

9.CVS(Concurrent Version System)

Another widely used version control system is CVS. It is a crucial part of Source Configuration Management (SCM), a tool that developers have relied on since the 1980s. With the aid of CVS, you may simply record the history of sources, files, and documents. It employs delta compression for effective storage use and excludes symbolic links to reduce security risks.

10.Bazaar

It is a version control system that makes it simple to collaborate with other developers, whether they are a single developer, a team working in the same location, or a global community. It scales and modifies to fit your requirements. It is a free, open-source, distributed version control program that is supported by Canonical and offers an excellent user experience. It is very similar to Git and Mercurial.

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

The study on tools of DevOps for specific phases is completed.

Ex.no: 03	**GIT**
DATE:	

AIM:

To study about git and it’s importance.

PROCEDURE:

Git is distributed version control system designed to handle everything from small to large project with Speed and efficiency, it allows multiple developers to work on the same project simultineously without inter Feirying with each other.

Features

1. Version Control
2. Collaboration
3. Branching and merging
4. Staging area

Git commands:

Git init: Initialize new git repository

Git clone [url]: create a copy of a remote repository locally.
Git add [file] : stages change to a file for the next commit.
Git log: display commit history

Git commit: -m “msg” : commit the stage changed with descriptive message.

Git push : uploads local commit to a remote repository

Git pull: Fetches and integrate changes from a remote repository into local repository.

Git branch: list , create, and delete branch.

Git merge [branch]: Merge change from a specific branch it current branch.

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, the study about the git tool is completed.

Ex.no: 04	INSTALLATION OF VIRTUAL BOX AND WINDOWS OS
DATE:	

AIM:

To install VirtualBox and setting up a virtual machine (VM) to install and run Windows OS.

PROCEDURE:

Step 1: Install VirtualBox

1. Download the VirtualBox installer from the official website.
2. Run the installer on your host machine:

    Accept the license agreement.

    Choose installation options and click Next.

    Click Install to complete the installation.

3.Open VirtualBox to verify the installation.

Step 2: Create a New Virtual Machine

1.Open VirtualBox and click on New.

2.Provide a name for your VM (e.g., "Windows10_VM").

3.Select the Type as Microsoft Windows and the Version (e.g., Windows 10/11).

4.Allocate memory (RAM) to the VM (e.g., 4 GB or 4096 MB).

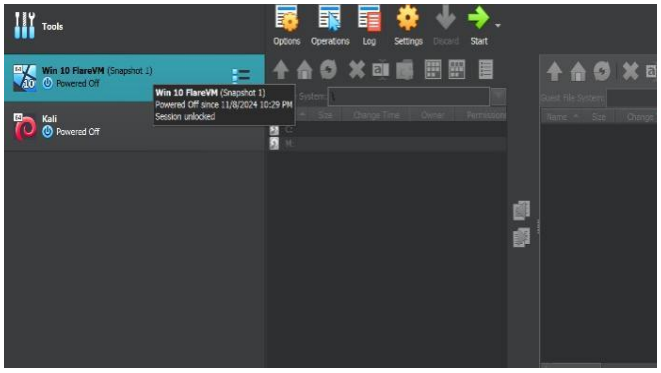
5.Create a virtual hard disk:

    Select Create a virtual hard disk now.

    Choose VDI (VirtualBox Disk Image).

    Select Dynamically allocated and allocate disk space (e.g., 50 GB).

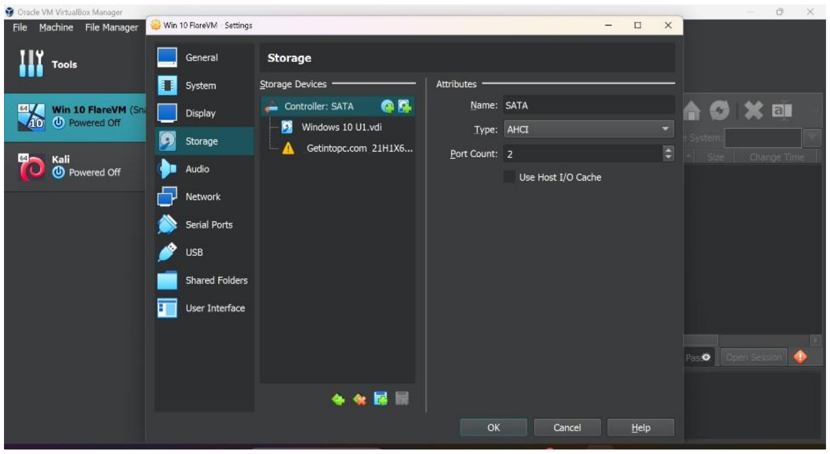
Click Create.



Win 10 FlareVM (Snapshot 1)
Powered Off since 11/8/2024 10:29 PM
Session unlocked

Step 3: Install Windows OS

1. Select the VM and click Settings.
2. Go to Storage and select the empty disk under Controller: IDE.
3. Click on the disk icon and choose the Windows ISO file as the virtual optical disk.
4. Save the settings and start the VM by clicking Start.
5. The VM will boot from the ISO file. Follow these steps in the Windows installer:
    - Select language, time, and keyboard settings.
    - Click Install Now.
    - Enter the product key or skip if not available.
    - Choose the installation type (Custom).
    - Select the virtual disk and proceed with the installation.
6. After installation, the VM will restart, and you will be prompted to set up Windows (e.g., username, password, and initial settings).



Controller: SATA
Windows 10 U1.vdi
Getintopc.com 21H1X6...

Name: SATA
Type: AHCI
Port Count: 2
Use Host I/O Cache

CLASS PERFORMANCE	
RECORD	
VIVA	
TOTAL	

RESULT:

VirtualBox was successfully installed, and a Windows operating system was set up and configured in a virtual machine.

Ex.no: 05	# CREATE MAVEN BUILD PIPELINE IN AZURE
DATE:	

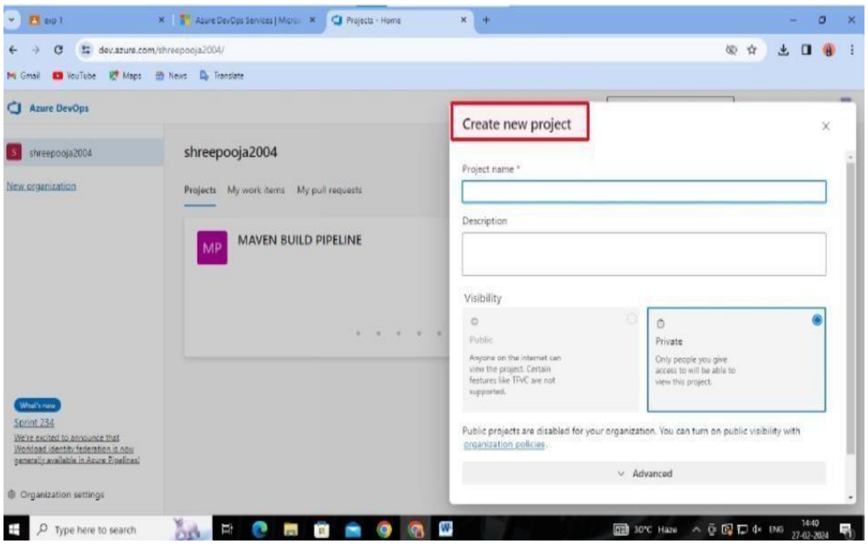
AIM:

To create and configure a Maven build pipeline in Azure DevOps for automating the build and testing process of a Java application.

PROCEDURE:

Step 1: Log in to Azure DevOps Open your web browser and navigate to Azure DevOps. Sign in with your Azure DevOps account.

Step 2: Create a New Project Once logged in, click on "Projects" in the top left corner.Click on the "New Project" button.Fill in the necessary details for your project (name,description,visibility), and Click “create”.



Create new project

Project name *

Description

Visibility

Public: Anyone on the internet can view the project. Certain features like TFVC are not supported.

[✓ Private]: Only people you give access to will be able to view this project.

Public projects are disabled for your organization. You can turn on public visibility with organization policies.

Advanced

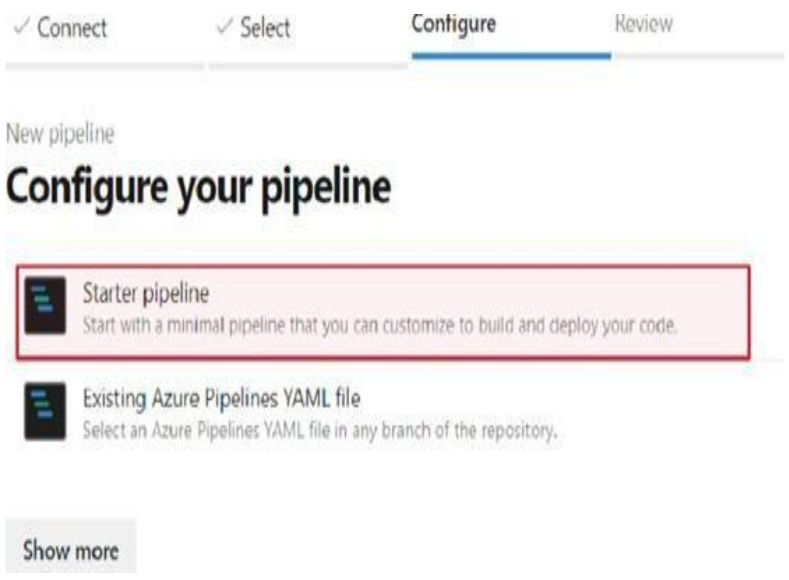
Step3: Set Up Source Control In your new project, navigate to the "Repos" Section. Import or set up your Maven project in the source control system (e.g., Git).

Step4: Create a New Build Pipeline Navigate to the "Pipelines" section in your project. Click on the "New pipeline" button. Choose your source repository (e.g., GitHub, Azure Repos Git).

Step5: Select a Template Azure DevOps will prompt you to choose a template. Search for and select the "Maven" template.

Step6: Configure the Pipeline You'll be presented with a YAML editor for your pipeline. Review and modify the YAML as needed. Make sure the pom.xml path is correctly specified. The default is often $(System.DefaultWorkingDirectory)/pom.xml. Customize Maven goals and options if required.

Step7: Build and Test , The default Maven task runs the clean install command. Adjust this command if needed. Configure any additional Maven goals or options based on your project requirements.


✓ Connect ✓ Select Configure Review

New pipeline

Configure your pipeline

Starter pipeline
Start with a minimal pipeline that you can customize to build and deploy your code.

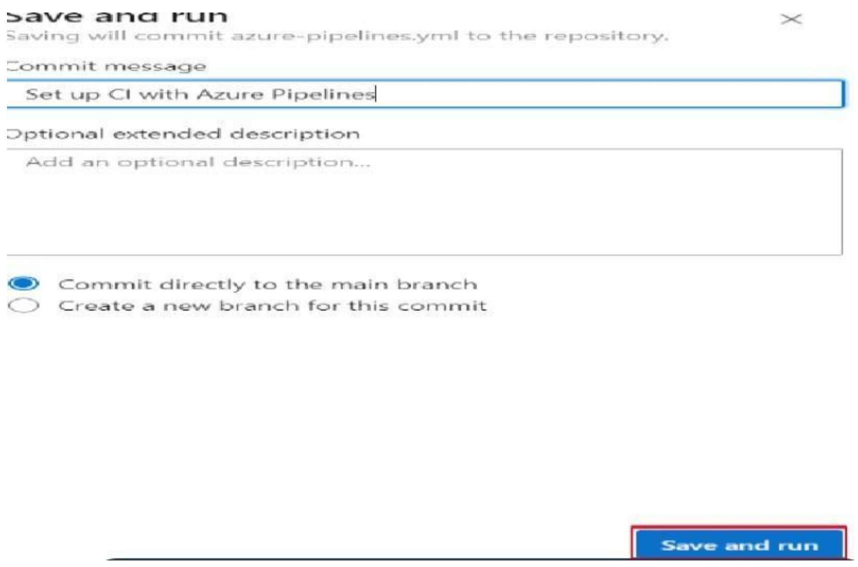
Existing Azure Pipelines YAML file
Select an Azure Pipelines YAML file in any branch of the repository.

Show more

Step 8: Publish Artifacts (Optional) If your Maven build produces artifacts, add a "Publish Build Artifacts" task. Specify the path to the artifacts and choose where to publish them.

Step 9: Save and Queue Save your pipeline configuration by clicking "Save" or "Save & Queue."

Step10: Review Logs Once the build is complete, review the logs to ensure that everything ran successfully. Address any issues that may arise during the build process.


Save and run
Saving will commit azure-pipelines.yml to the repository.

Commit message
Set up CI with Azure Pipelines

Optional extended description
Add an optional description...

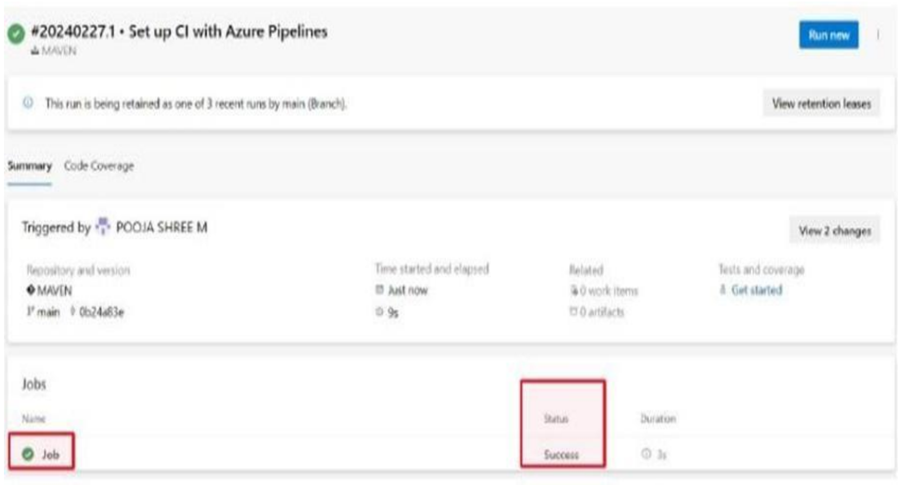
● Commit directly to the main branch
○ Create a new branch for this commit

Save and run

Step 11: Trigger Builds Automatically (Optional) Configure triggers to automatically queue a build whenever changes are pushed to the repository. Navigate to the "Triggers" tab in your pipeline configuration to set up continuous integration.

Step12: Set Up Continuous Integration (Optional) Integrate your build pipeline with other stages for continuous integration and delivery. Define deployment stages and tasks as needed.

Step13: Save and Run . Save your changes and run.save your changes and run the pipeline to ensure that it functions as expected. Monitor the pipeline for any errors or issue.



#20240227.1 • Set up CI with Azure Pipelines
MAVEN

This run is being retained as one of 3 recent runs by main (Branch).

Summary Code Coverage

Triggered by POOJA SHREE M

Repository and version	Time started and elapsed	Related	Tests and coverage
MAVEN	Just now	0 work items	Get started
main 0b24a83e	9s	0 artifacts	

Jobs

Name	Status	Duration
Job	Success	3s

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, the set up of MAVEN PIPELINE in Azure was build and implemented successfully.

Ex.no: 06	RUN REGRESSION TESTS USING MAVEN BUILD PIPELINE IN AZURE
DATE:	

AIM:

To run regression tests using maven build pipeline in azure.

PROCEDURE:

Step 1: Setting up the environment. To start, log in to Azure DevOps and click on User settings and then on Personal access tokens.

- Create a new token by selecting the Agent Pools (read, manage) permission
- Return to the Azure DevOps home page and click on Organization settings at the bottom left.
- Click on Agent Pools and select the Default pool or create a new Self hosted one.
- Click on New Agent and follow the instructions on the page to upload your agent.
- Unzip the downloaded archive on the machine where you want to install the agent and run config.cmd.
- Follow the script to set up your agent, including the token created earlier. When you have the choice between an interactive mode or as a service, choose the interactive mode otherwise you will not be able to launch the ATS tests.
- Once your agent is configured, run run.cmd to start it.

Machine setup

- Once the agent is installed, several elements are necessary for the proper functioning of the
- The .actiontestscript folder: this folder must be present at the root of the user's folder that will run the agent. It is created automatically and updated when Agilitest is launched. If you do not wish to install Agilitest on the machine where the agent is installed, you can copy and paste this folder from another computer where Agilitest is installed. In the future it will be possible to update this folder via npm and to automated it in the pipeline.
- Maven must also be installed, you can download it here: https://maven.apache.org/download.cgi.

Step 2: Creating a new pipeline on Azure DevOps On Azure DevOps click on the Pipelines men.

Step 3: Then click on New pipeline and indicate where your Agilitest project code is located and follow the instructions to connect the Azure Pipeline to it

Step 4: You can choose to use a preconfigured pipeline for Maven.

Step 5: The selection of a maven project will allow AzuredevOps to directly manage the dependencies and external required libraries.

In the pool section, change the "vmImage" type to "name" and enter the name of the pool you created earlier

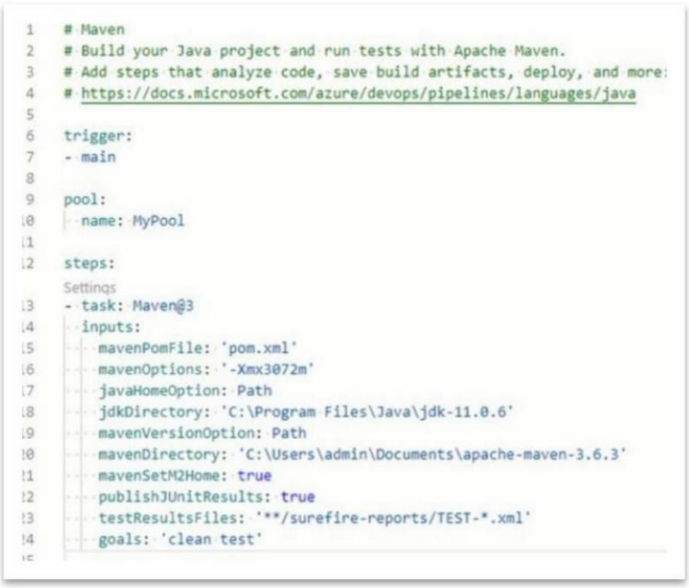
Modify the Maven task:

- Change the value of "goals" to "clean test"
- Change the value of "javaHomeOption" to "Path"
- Add a "jdkDirectory" property and specify the path to the JDK on your machine
- Remove the "jdkVersionOption" and "jdkArchitectureOption" properties

- Add a "mavenVersionOption" property with the value "Path"
- Add a "mavenDirectory" property and specify the path of Maven on your machine
- Add a "mavenSetM2Home" property with the value "true".

Step 6: Your YAML file should look like this

Step 7 :If you click on Save and run, the YAML file will be added to your source code and your tests will be run according to the configuration in your pom.xml file.



```yaml
# Maven
# Build your Java project and run tests with Apache Maven.
# Add steps that analyze code, save build artifacts, deploy, and more:
# https://docs.microsoft.com/azure/devops/pipelines/languages/java

trigger:
- main

pool:
  name: MyPool

steps:
- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'
    mavenOptions: '-Xmx3072m'
    javaHomeOption: Path
    jdkDirectory: 'C:\Program Files\Java\jdk-11.0.6'
    mavenVersionOption: Path
    mavenDirectory: 'C:\Users\admin\Documents\apache-maven-3.6.3'
    mavenSetM2Home: true
    publishJUnitResults: true
    testResultsFiles: '**/surefire-reports/TEST-*.xml'
    goals: 'clean test'
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, To run regression tests using Maven Build pipeline in Azure is executed successfully.

Ex.no: 07	CREATING AN AWS EC2 INSTANCE USING TERRAFORM
DATE:	

AIM:

To create and manage an AWS EC2 instance using Terraform, demonstrating Infrastructure as Code (IaC) capabilities.

PROCEDURE

Prerequisites:
Install Terraform.
Configure AWS CLI with appropriate IAM credentials (aws configure). Ensure you have an AWS account with necessary permissions to manage EC2 instances.
Create Terraform Files & working directory.
* Write the configuration files.
Code:

```
provider "aws" {
region = "us-east-1" # Change region as needed }
resource "aws_instance" "example" {
ami = "ami-0c55b159cbfafe1f0" # Replace with a valid AMI ID instance_type = "t2.micro" tags =
{ Name = "Terraform-Instance"
} }
```

main.tf: Defines the EC2 instance and provider.
variables.tf: Optional, to define variables (not required for basic setup).
Initialize Terraform
Run the following command to download necessary plugins:
Code: terraform init
Validate Configuration
Code: terraform validate
Plan the Deployment
View the resources Terraform will create:
Code: terraform plan Apply
the Configuration
Code: terraform apply Verify
the EC2 Instance
Log in to the AWS Management Console and navigate to the EC2 Dashboard. Verify the created instance matches the specifications in main.tf.

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, the Terraform configuration successfully created an AWS EC2 instance. Demonstrating the application of Infrastructure as Code(IaC) principles.

Ex.no: 08	RESOLVING GIT MERGE CONFLICTS
DATE:	

AIM:

To simulate and resolve merge conflicts in Git during a collaborative workflow, demonstrating conflict resolution techniques.

PROCEDURE:

Prerequisites

Install Git on your system.
Create or clone a Git repository.

Ensure at least two branches (main and feature) exist.
Setup and Simulate a Merge Conflict

Initialize a Git Repository

Code:

```
git init merge-conflict-demo cd
merge-conflict-demo

echo "Initial content" > file.txt git
add file.txt

git commit -m "Initial commit"
```

Create and Modify Branches

Code: echo "Change from main branch" > file.txt git
add file.txt

git commit -m "Update from main branch"

Create a feature branch and make changes:

Code: git checkout -b feature

echo "Change from feature branch" > file.txt git add file.txt git
commit -m "Update from feature branch"

Merge the feature Branch into main

Code: git checkout main
Attempt to merge

Code: git merge feature

Git will detect a conflict and stop the merge. The output will indicate the conflicting file(s)

Resolve the Conflict

Open the conflicted file (file.txt) in a text editor. The content will look like: plaintext

Code:

```
<<<<<<< HEAD

Change from main branch
```

```
=======
```

Change from feature branch

```
>>>>>>> feature
```

Resolve the conflict by editing the file to retain desired changes:

Code:

Change from both branches
Mark the conflict as resolved:

Code:

```
git add file.txt
```

Complete the merge:

Code: git commit -m "Resolved merge conflict"

Verify the Merge

Check the branch history to ensure the merge is complete:

Code:

```
git log --graph –oneline
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, the merge conflict was successfully simulated and resolved in Git, ensuringthe repository reflects the desired state of the file.txt content.

Ex.no: 09	**DOCKER INSTALLATION AND CONTAINERIZATION**
DATE:	

AIM:

To install Docker, create a Docker image for an Nginx web server, serve static content, and push the image to Docker Hub for public or private use.

PROCEDURE:

Install Docker:

For Ubuntu (or other Debian-based systems):

Update the package database: sudo apt update

Install required packages:

```
sudo apt install apt-transport-https ca-
certificates curl        software-properties-
common
```

Add Docker's official GPG key:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Add Docker's APT repository:

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-
                                                              keyring.gp
g] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" |

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Install Docker: sudo apt install docker-ce

Start Docker and enable it to start on boot:

```
sudo systemctl start docker
sudo systemctl enable docker
```

Verify Docker Installation:

```
docker --version
```

Set up Docker Hub Account:

Sign up or log in to your Docker Hub account at https://hub.docker.com.

Prepare Static Website Files:

Create a project folder, and inside it, create an html directory to store your website's static files

Sample index.html:

html

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
        <title>Welcome to Nginx</title>
        </head>
        <body>
            <h1>Hello, Nginx is running in Docker!</h1>
        </body>
        </html>
```

Create a Dockerfile:

In the root directory of your project (my-nginx-app/), create a file named Dockerfile with the following content:

```
FROM nginx:alpine
    COPY ./html /usr/share/nginx/html
    EXPOSE 80
```

This Dockerfile uses the official Nginx image as the base, copies your static files to Nginx's default serving directory (/usr/share/nginx/html), and exposes port 80 for web access.

Build the Docker Image:

```
docker build -t <dockerhub-username>/nginx-website:<tag>
```

Replace <dockerhub-username> with your Docker Hub username, and <tag> with a version tag (e.g., latest).

Test the Docker Image Locally:

```
docker run -d -p 8080:80 <dockerhub-username>/nginx-website:<tag>
```

After running the command, open a browser and go to http://localhost:8080. You should see the Nginx web server serving your static website.

Login to Docker Hub:

Authenticate Docker CLI with your Docker Hub credentials by running:

```
docker login
```

Push the Docker Image to Docker Hub:

```
docker push <dockerhub-username>/nginx-website:<tag>
```

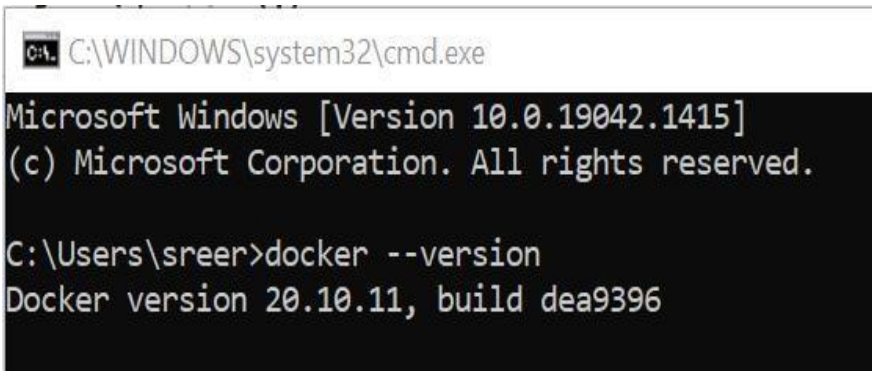
Verify the Image on Docker Hub:

Go to your Docker Hub account at https://hub.docker.com and check that the image has been successfully pushed.

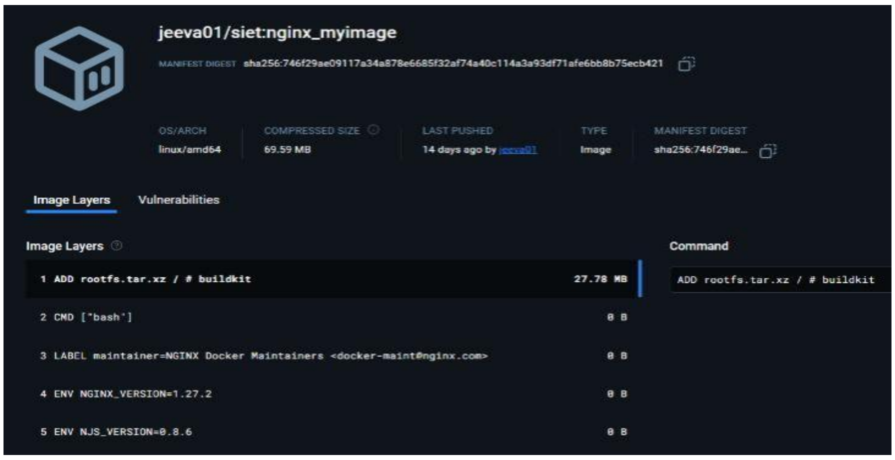
Then, you can run the image with:

```
docker run -d -p 8080:80 <dockerhub-username>/nginx-website:<tag>
```

OUTPUT:



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19042.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sreer>docker --version
Docker version 20.10.11, build dea9396
```



```
jeeva01/siet:nginx_myimage
MANIFEST DIGEST sha256:746f29ae09117a34a878e6685f32af74a40c114a3a93df71afe6bb8b75ecb421

OS/ARCH        COMPRESSED SIZE   LAST PUSHED              TYPE    MANIFEST DIGEST
linux/amd64    69.59 MB          14 days ago by jeeva01   Image   sha256:746f29ae...

Image Layers   Vulnerabilities

Image Layers                                                                     Command
1 ADD rootfs.tar.xz / # buildkit                                       27.78 MB  ADD rootfs.tar.xz / # buildkit
2 CMD ["bash"]                                                         0 B
3 LABEL maintainer=NGINX Docker Maintainers <docker-maint@nginx.com>   0 B
4 ENV NGINX_VERSION=1.27.2                                             0 B
5 ENV NJS_VERSION=0.8.6                                                0 B
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

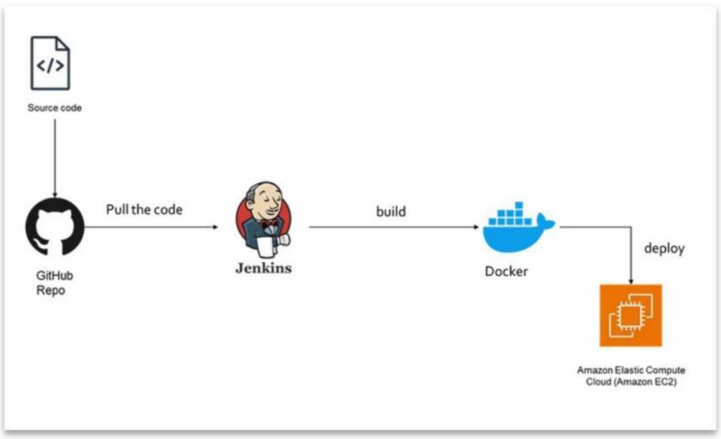
A Docker image for an Nginx web server was created and configured to serve static HTML files.The image was bulit sucessfully and tested locally using Docker.

Ex.no: 10	**DEPLOY NGINX APPLICATION USING JENKINS**
DATE:	

AIM:

To automate the deployment of an Nginx application using Jenkins, ensuring continuous integration and deployment (CI/CD) by leveraging Jenkins for building and automating the deployment process.

ARCHITECTURE:



PROCEDURE:

1.Install Docker

```
$ sudo apt install docker –y

$ systemctl start docker
```

2.Install Jenkins

```
$ sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
 https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
```

Then add a Jenkins apt repository entry:

```
$ 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
```

Update your local package index, then finally install Jenkins:

```
sudo apt-get update

sudo apt-get install fontconfig openjdk-17-jre
sudo apt-get install Jenkins
```

3.Open browser http://vm_ip:8080,it will open the Jenkins portal

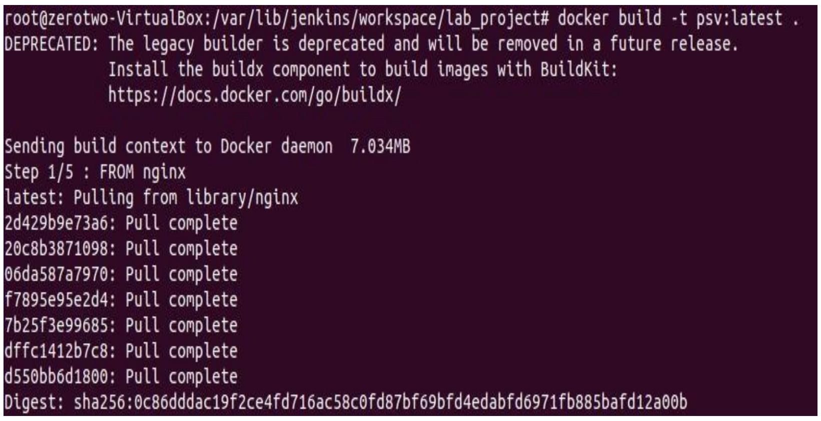
cat /var/lib/jenkins/secrets/initialAdminPassword (to get Jenkins password)

4.Login to Jenkins console (http://ec2-public-ip:8080/login).

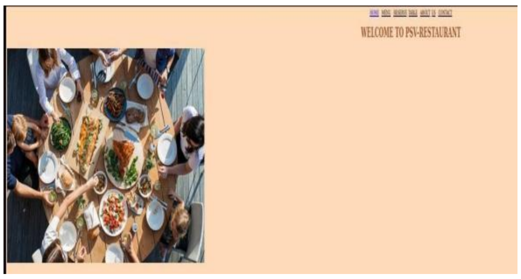
5.Select Install suggested plugin and it will start installing suggested plugin
6.Click on New Item.
7.Enter item name and item type as “Freestyle project”, Click on OK.
8.In Configure -> Source Code Management, Select Git and provide Repository URL (GitHub repo url) and Branch Specifier as “*/main”.
9.In Build Steps, Select Execute Shell and provide below commands to clone GitHub repo, download files from repo, build new docker image and container.

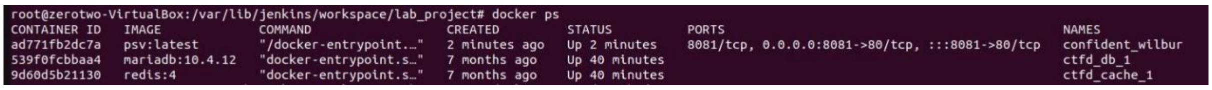
```
git clone https://github.com/Vaishu-psv/psv-restaurant-1.git
cd psv-restaurant-1
docker build -t custom-nginx .
docker run -d — name psv-res -p 8081:80 custom-nginx.
```

OUTPUT:

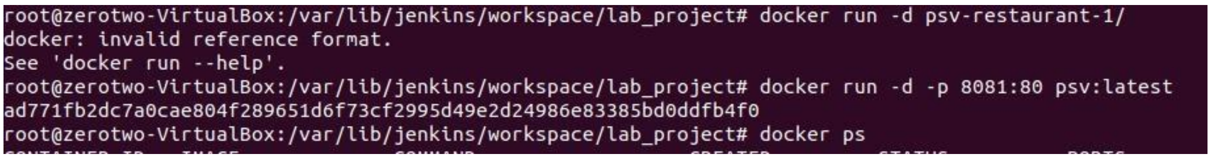


```
root@zerotwo-VirtualBox:/var/lib/jenkins/workspace/lab_project# docker build -t psv:latest .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  7.034MB
Step 1/5 : FROM nginx
latest: Pulling from library/nginx
2d429b9e73a6: Pull complete
20c8b3871098: Pull complete
06da587a7970: Pull complete
f7895e95e2d4: Pull complete
7b25f3e99685: Pull complete
dffc1412b7c8: Pull complete
d550bb6d1800: Pull complete
Digest: sha256:0c86dddac19f2ce4fd716ac58c0fd87bf69bfd4edabfd6971fb885bafd12a00b
```


HOME MENU RESERVE TABLE ABOUT US CONTACT
WELCOME TO PSV-RESTAURANT



```
root@zerotwo-VirtualBox:/var/lib/jenkins/workspace/lab_project# docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED          STATUS          PORTS                                     NAMES
ad771fb2dc7a   psv:latest        "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes    8081/tcp, 0.0.0.0:8081->80/tcp, :::8081->80/tcp   confident_wilbur
539f0fcbbaa4   mariadb:10.4.12   "docker-entrypoint.s…"   7 months ago     Up 40 minutes                                             ctfd_db_1
9d60d5b21130   redis:4           "docker-entrypoint.s…"   7 months ago     Up 40 minutes                                             ctfd_cache_1
```



```
root@zerotwo-VirtualBox:/var/lib/jenkins/workspace/lab_project# docker run -d psv-restaurant-1/
docker: invalid reference format.
See 'docker run --help'.
root@zerotwo-VirtualBox:/var/lib/jenkins/workspace/lab_project# docker run -d -p 8081:80 psv:latest
ad771fb2dc7a0cae804f289651d6f73cf2995d49e2d24986e83385bd0ddfb4f0
root@zerotwo-VirtualBox:/var/lib/jenkins/workspace/lab_project# docker ps
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

The Nginx application was successfully deployed using Jenkins.

Ex.no: 11	INSTALL JENKINS IN CLOUD
DATE:	

AIM:

To install and configure Jenkins, an open-source automation server, on a cloud-based instance to enable continuous integration and delivery (CI/CD) pipeline.

PROCEDURE:

-> Select a Cloud Provider.

Choose a cloud provider like AWS, Google Cloud Platform (GCP), or Microsoft Azure.

-> Set up a virtual machine (VM) instance:

AWS EC2: Launch an EC2 instance using Ubuntu or any preferred Linux distribution. GCP: Create a Compute Engine VM instance.

->Azure: Deploy a Virtual Machine from the Azure portal. & Connect to the Instance Use SSH to connect to your cloud VM:

AWS EC2 Example: $ ssh -i "your-key.pem" ubuntu@your-instance-ip

INSTALL JAVA:

Jenkins requires Java to run. Install Java Development Kit (JDK) on the VM:
Update package manager & Install JDK (for Ubuntu):

```
sudo apt update && sudo apt install openjdk-11-jdk -y
```

Install Jenkins

Add the Jenkins repository:

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key|sudo tee\
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

Append the Jenkins repository entry to the sources list:

```
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
```

Install Jenkins:

$ sudo apt install jenkins –
y Start and Enable Jenkins

```
sudo systemctl start Jenkins
sudo systemctl enable
Jenkins
```

Open Jenkins in a Browser

Jenkins runs on port 8080 by default. To access Jenkins, open a browser and go to: http://your-instance-ip:8080

Unlock Jenkins

Retrieve the initial administrator password:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword Copy the password and paste it into the Jenkins setup page.

Install Suggested Plugins

After unlocking Jenkins, choose Install Suggested Plugins during the initial setup

Create an Admin User

Once plugins are installed, you'll be prompted to create your first admin user. Provide the necessary details to set up the admin account.

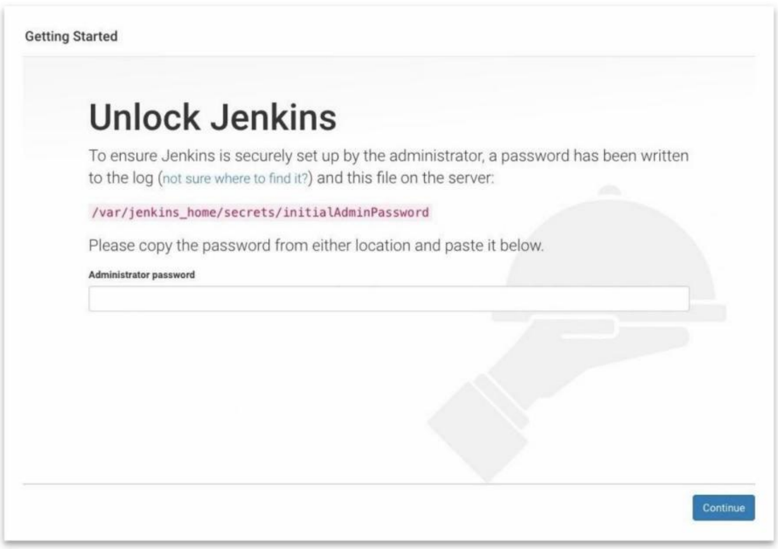
Complete Jenkins Setup

After setting up the admin user, the Jenkins dashboard will load, and you can start creating jobs and pipelines.

OUTPUT:

Access Jenkins in Browser

Accessed Jenkins interface using the URL: http://<instance-public-ip>:8080


Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

/var/jenkins_home/secrets/initialAdminPassword

Please copy the password from either location and paste it below.

Administrator password

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

The installation and configuration of Jenkins on a cloud-based virtual machine instance was successfully completed. Jenkins is now running and accessible through the web interface. The administrator account is set up, and the Jenkins environment is ready for building and automating CI/CD pipelines.

Ex.no: 12	CREATE CI PIPELINE USING JENKINS
DATE:	

AIM:

To create and configure a Continuous Integration (CI) pipeline using Jenkins, enabling automatic build, test, and integration processes for code changes in a version control system like Git.

PROCEDURE:

1.Prerequisites

Jenkins Installed:

Version Control System: A Git

Build Tool: Install any necessary build tools like Maven, Gradle, or npm based on your project.

Install Required Jenkins Plugins
Log into the Jenkins dashboard using your browser and Go to Manage Jenkins > Manage Plugins.

2.Install the following plugins:

Git Plugin (for Git integration) and Pipeline Plugin (to create Jenkins pipelines).
Any build tool-specific plugins like Maven Integration or Gradle Plugin.

Set Up Jenkins Job for CI Pipeline

Create a New Job:

On the Jenkins dashboard, click on New Item.

Enter a name for your pipeline (e.g., My-CI-Pipeline). Configure Git Repository:

In the pipeline configuration page, go to the Pipeline section.
Under Definition, select Pipeline script from SCM.

Choose Git as the source control system.

Enter the URL of your Git repository and provide credentials if required.
Create Jenkinsfile (Pipeline Script):

In your Git repository, create a file named Jenkinsfile. This file defines the CI pipeline stages such as build, test, and deploy.

Example of a basic Jenkinsfile for a Maven project:

```
pipeline { agent any

          stages { stage('Checkout') { steps {

    git 'https://github.com/username/repo-name.git'}}
stage('Build') { steps {

    sh 'mvn clean package'} }
stage('Test') {steps {

    sh 'mvn test'}}
stage('Deploy') { steps {
```

echo 'Deploying application...' } }} }

Configure Build Triggers:

Scroll down to the Build Triggers section.

Check Poll SCM if you want Jenkins to check for changes at regular intervals (e.g., every 5 minutes). Add a schedule like:

Copy code: * * * * * Alternatively, use GitHub hook trigger for GITScm polling to trigger builds automatically when changes are pushed to the repository.

Build the Pipeline

Save the job configuration and return to the dashboard.

Click on the job name and press Build Now to manually trigger the first pipeline run.
Jenkins will start the pipeline and display its status under Build History.

Monitor Pipeline Execution

After starting the pipeline, you can view its progress in real-time by selecting the build number from Build History. The Jenkins console output will show each step of the pipeline:

Checkout: Fetches the code from the repository.
Build: Compiles and packages the code.

Test: Executes unit tests and shows results.

Deploy: Simulates deployment (or performs an actual deployment if configured).

Analyze Pipeline Results

Once the pipeline execution completes, check the build status:

Green checkmark: Build and tests succeeded.

Red cross: There was an error during one of the stages.

If the build fails, Jenkins will provide detailed logs of each stage to help identify and fix issues.

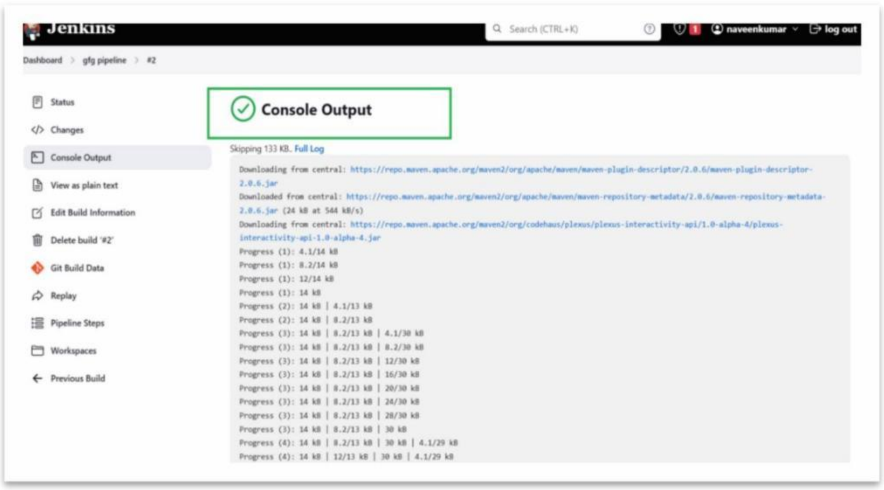
Automate CI with Git Webhooks

To fully automate the pipeline, configure webhooks in your Git repository (GitHub, GitLab, etc.) to notify Jenkins of code changes.

In GitHub, go to Settings > Webhooks, and add the Jenkins webhook URL: http://your-jenkins-server:8080/github-webhook/

This ensures Jenkins automatically triggers the CI pipeline whenever code is pushed to repository.

OUTPUT:



Console Output

Skipping 133 KB. Full Log

```
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-plugin-descriptor/2.0.6/maven-plugin-descriptor-
2.0.6.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-repository-metadata/2.0.6/maven-repository-metadata-
2.0.6.jar (24 kB at 544 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-interactivity-api/1.0-alpha-4/plexus-
interactivity-api-1.0-alpha-4.jar
Progress (1): 4.1/14 kB
Progress (1): 8.2/14 kB
Progress (1): 12/14 kB
Progress (1): 14 kB
Progress (2): 14 kB | 4.1/13 kB
Progress (2): 14 kB | 8.2/13 kB
Progress (3): 14 kB | 8.2/13 kB | 4.1/30 kB
Progress (3): 14 kB | 8.2/13 kB | 8.2/30 kB
Progress (3): 14 kB | 8.2/13 kB | 12/30 kB
Progress (3): 14 kB | 8.2/13 kB | 16/30 kB
Progress (3): 14 kB | 8.2/13 kB | 20/30 kB
Progress (3): 14 kB | 8.2/13 kB | 24/30 kB
Progress (3): 14 kB | 8.2/13 kB | 28/30 kB
Progress (3): 14 kB | 8.2/13 kB | 30 kB
Progress (4): 14 kB | 8.2/13 kB | 30 kB | 4.1/29 kB
Progress (4): 14 kB | 12/13 kB | 30 kB | 4.1/29 kB
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

The CI pipeline was successfully created and configured using Jenkins.

Ex.no: 13	ANSIBLE INSTALLATION ON UBUNTU
DATE:	

AIM:

To install Ansible on ubuntu and setting up a control node and connecting it to one or more ansible hosts.

PROCEDURE:

Step1: Configure Ansible Control Node and Create administrator level user for the control node.

```
sudo adduser ansible
```

Step2: use usermod command to assign privileges to the account

```
sudo usermod -aG sudo ansible
```

Step3: Switch to the newly created user on the control node

```
sudo su ansible
```

Step4: Generate a SSH key pair for the ansible user by executing the command

```
ssh-keygen
```

Step5: Configure an Ansible Host

Create an EC2 instance in aws cloud by launching an instance with ubuntu ami, with LTS 22 version

Step6: copy the ssh key from ansible user. Connect to the ec2 instance with ssh command

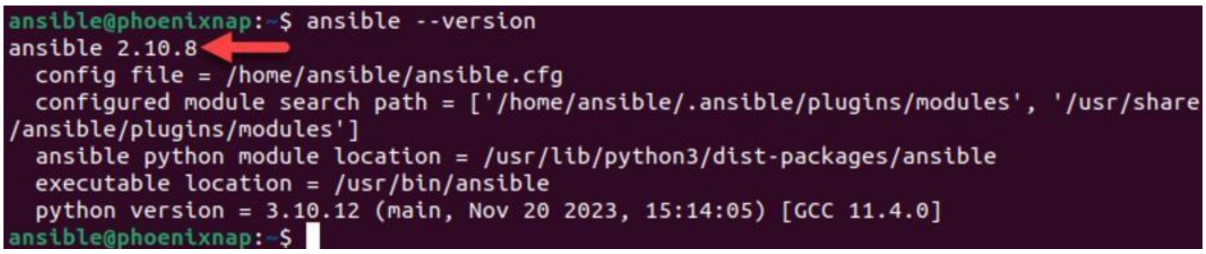
```
ssh -I key.pem ubuntu@<ip_address>
```

Step7: Locate .ssh/authorized_keys, paste the copied ssh-key into authorized keys file using nano
Step8: Install Ansible, Use the APT installer to install the Ansible package on the control node system.

```
sudo apt update

sudo apt install ansible -y
```

Step9: Verify the installation

```
$ ansible –version
```



```
ansible@phoenixnap:~$ ansible --version
ansible 2.10.8
  config file = /home/ansible/ansible.cfg
  configured module search path = ['/home/ansible/.ansible/plugins/modules', '/usr/share
/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
ansible@phoenixnap:~$
```

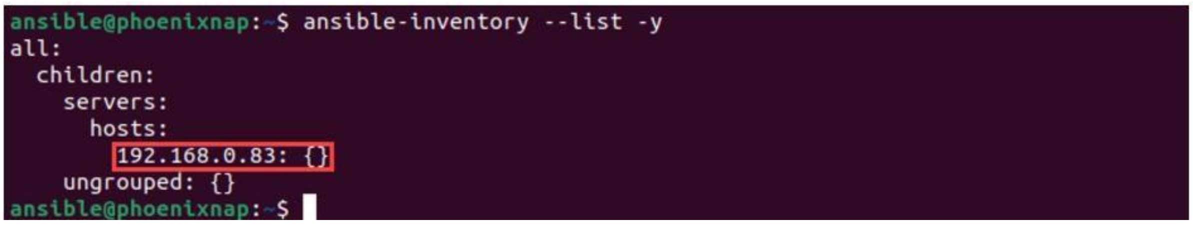
Step10: Set up the inventory file, Once Ansible is installed on the control node, set up an inventory file to allow Ansible to communicate with remote hosts. Use following commands

sudo mkdir -p /etc/ansible
sudo nano /etc/ansible/hosts

Add remote hosts that the control node will manage. Use the following format.
[aws] [server-ip]

Step11: Enter the command below to check the items in the inventory

ansible-inventory –list –y && ansible all -m ping

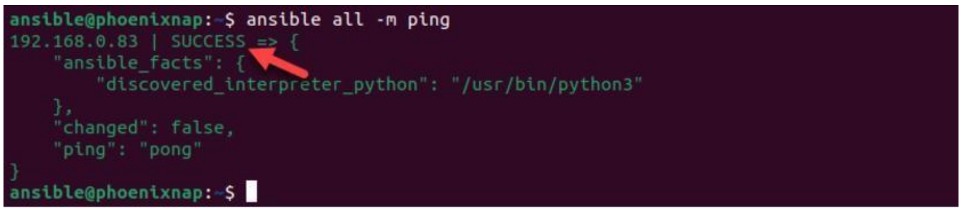


```
ansible@phoenixnap:~$ ansible-inventory --list -y
all:
  children:
    servers:
      hosts:
        192.168.0.83: {}
    ungrouped: {}
ansible@phoenixnap:~$
```

Step12: Test the Connection

ansible all -m ping

OUTPUT:



```
ansible@phoenixnap:~$ ansible all -m ping
192.168.0.83 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
ansible@phoenixnap:~$
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

The Ansible command executed successfully, establishing a connection to the remote host. The ping module verified the connectivity, and the host responded as expected.

Ex.no: 14	ANSIBLE PLAYBOOKS AND ROLES
DATE:	

AIM:

To create a Ansible roles and Playbook to automate the installation and configuration of nginx for web-server development.

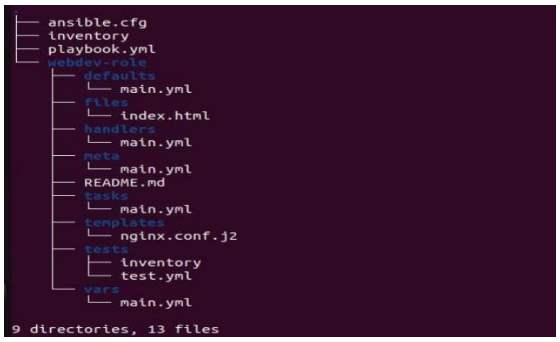
PROCEDURE:

Step1: Create a directory for your project
mkdir ansible_project

cd ansible_project

Step2: create new ansible role

ansible-enerate a directory structure with predefined folders for tasks,handlers, variables, and more.



```
.
├── ansible.cfg
├── inventory
├── playbook.yml
└── webdev-role
    ├── defaults
    │   └── main.yml
    ├── files
    │   └── index.html
    ├── handlers
    │   └── main.yml
    ├── meta
    │   └── main.yml
    ├── README.md
    ├── tasks
    │   └── main.yml
    ├── templates
    │   └── nginx.conf.j2
    ├── tests
    │   ├── inventory
    │   └── test.yml
    └── vars
        └── main.yml

9 directories, 13 files
```

Step3: In defaults/main.yml - Default
variables: nano main.yml

Step4: In files/index.html - Static HTML
file: nano index.html

Step5: In handlers/main.yml - Service
handlers: nano main.yml

Step6: In meta/main.yml - Role
metadata: nano main.yml

Step7: In tasks/main.yml - Main tasks:
nano main.yml

Step8: In create a playbook. yml in ansible_project
dir: nano playbook.yml

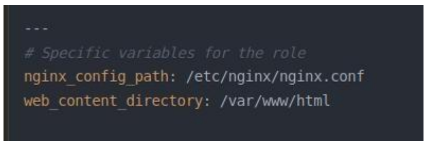
Step9: In Var/main.yml Role-specification variable:
nano main.yml

Step10: In project root folder create a inventory file
nano inventory

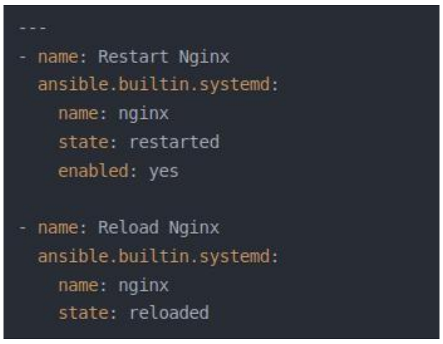
Step11: run this command

ansible-playbook -i inventory playbook.yml -y

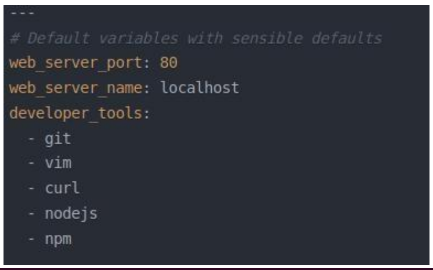
OUTPUT:



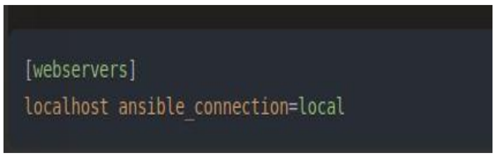
```
---
# Specific variables for the role
nginx_config_path: /etc/nginx/nginx.conf
web_content_directory: /var/www/html
```



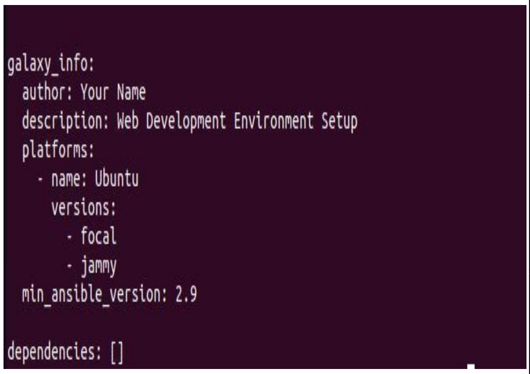
```
---
- name: Restart Nginx
  ansible.builtin.systemd:
    name: nginx
    state: restarted
    enabled: yes

- name: Reload Nginx
  ansible.builtin.systemd:
    name: nginx
    state: reloaded
```



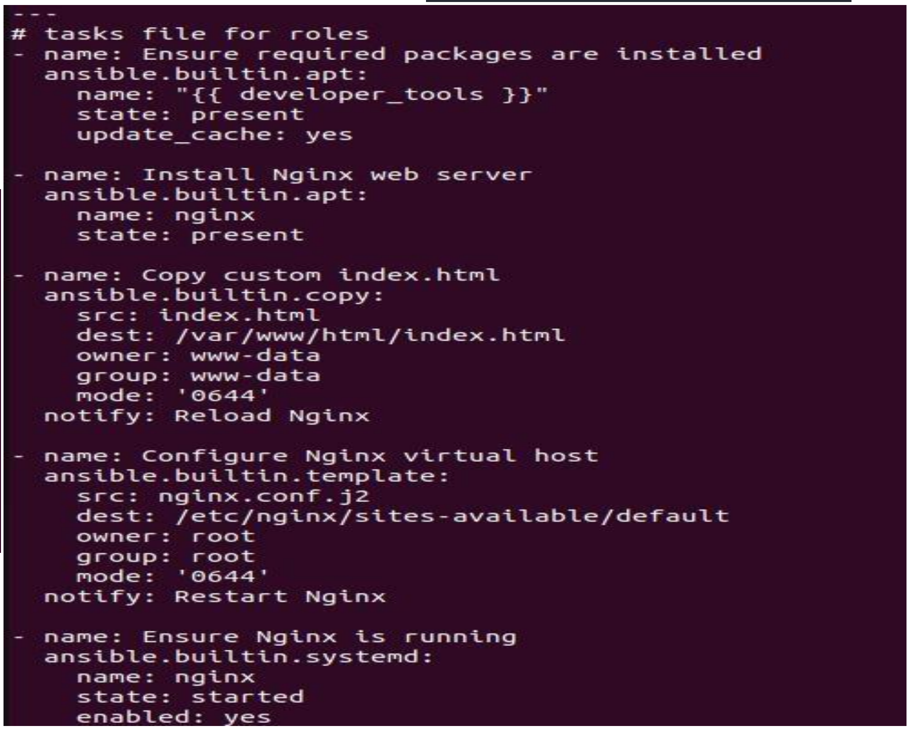
```
---
# Default variables with sensible defaults
web_server_port: 80
web_server_name: localhost
developer_tools:
  - git
  - vim
  - curl
  - nodejs
  - npm
```



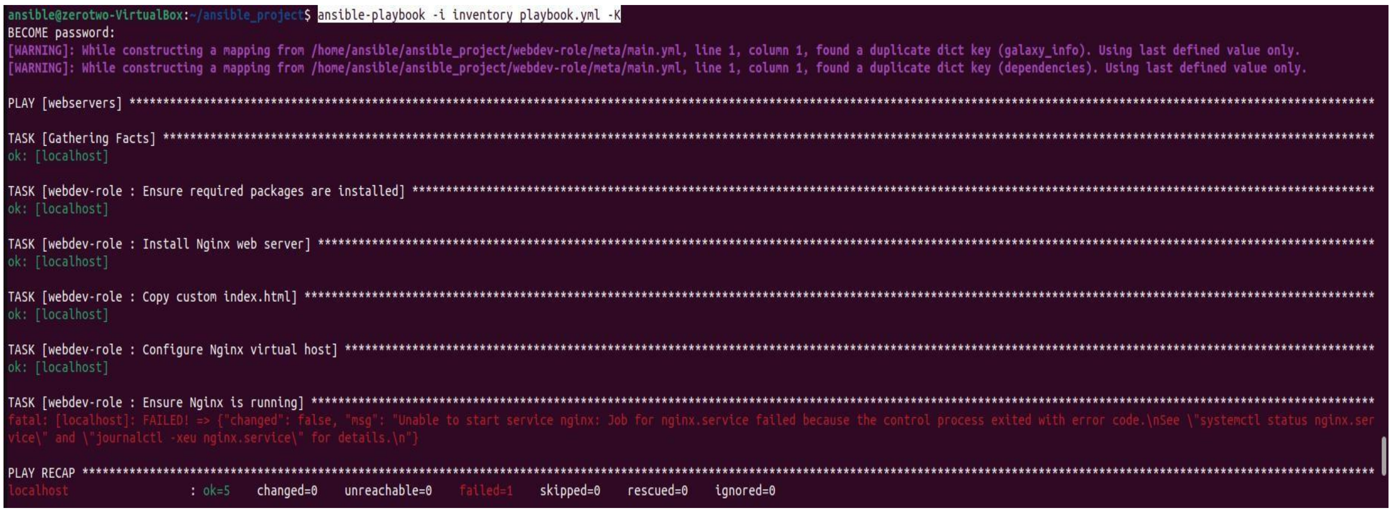
```
[webservers]
localhost ansible_connection=local
```



```
galaxy_info:
  author: Your Name
  description: Web Development Environment Setup
  platforms:
    - name: Ubuntu
      versions:
        - focal
        - jammy
  min_ansible_version: 2.9

dependencies: []
```



```
---
# tasks file for roles
- name: Ensure required packages are installed
  ansible.builtin.apt:
    name: "{{ developer_tools }}"
    state: present
    update_cache: yes

- name: Install Nginx web server
  ansible.builtin.apt:
    name: nginx
    state: present

- name: Copy custom index.html
  ansible.builtin.copy:
    src: index.html
    dest: /var/www/html/index.html
    owner: www-data
    group: www-data
    mode: '0644'
  notify: Reload Nginx

- name: Configure Nginx virtual host
  ansible.builtin.template:
    src: nginx.conf.j2
    dest: /etc/nginx/sites-available/default
    owner: root
    group: root
    mode: '0644'
  notify: Restart Nginx

- name: Ensure Nginx is running
  ansible.builtin.systemd:
    name: nginx
    state: started
    enabled: yes
```



```
ansible@zerotwo-VirtualBox:~/ansible_project$ ansible-playbook -i inventory playbook.yml -K
BECOME password:
[WARNING]: While constructing a mapping from /home/ansible/ansible_project/webdev-role/meta/main.yml, line 1, column 1, found a duplicate dict key (galaxy_info). Using last defined value only.
[WARNING]: While constructing a mapping from /home/ansible/ansible_project/webdev-role/meta/main.yml, line 1, column 1, found a duplicate dict key (dependencies). Using last defined value only.

PLAY [webservers] ***********************************************************************************************************************

TASK [Gathering Facts] ******************************************************************************************************************
ok: [localhost]

TASK [webdev-role : Ensure required packages are installed] *****************************************************************************
ok: [localhost]

TASK [webdev-role : Install Nginx web server] *******************************************************************************************
ok: [localhost]

TASK [webdev-role : Copy custom index.html] *********************************************************************************************
ok: [localhost]

TASK [webdev-role : Configure Nginx virtual host] ***************************************************************************************
ok: [localhost]

TASK [webdev-role : Ensure Nginx is running] ********************************************************************************************
fatal: [localhost]: FAILED! => {"changed": false, "msg": "Unable to start service nginx: Job for nginx.service failed because the control process exited with error code.\nSee \"systemctl status nginx.ser
vice\" and \"journalctl -xeu nginx.service\" for details.\n"}

PLAY RECAP ******************************************************************************************************************************
localhost                  : ok=5    changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

The ansible roles and playbook are successfully executed and webserver deployed with nginx configuration successfully.

Ex.no: 15	INSTALLATION OF GIT IN LOCAL SERVER AND CONFIGURE
DATE:	

AIM:

To Install git in local server and configure with Github.

PROCEDURE:

STEP1: Install GIT (in git downloads).

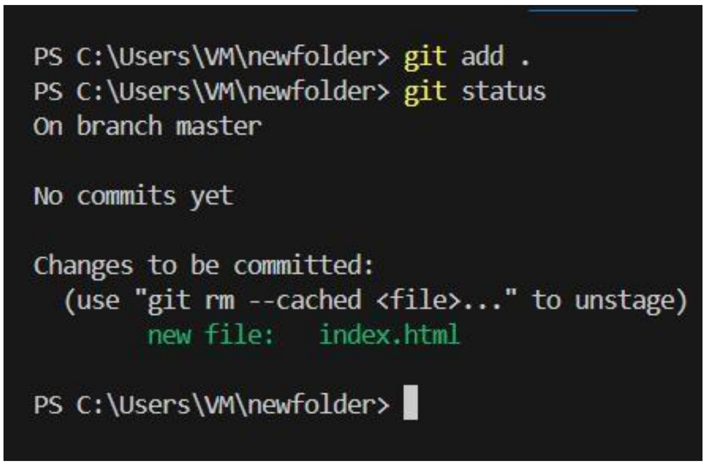
STEP2: Configure with github.

git config --global user.name "username"

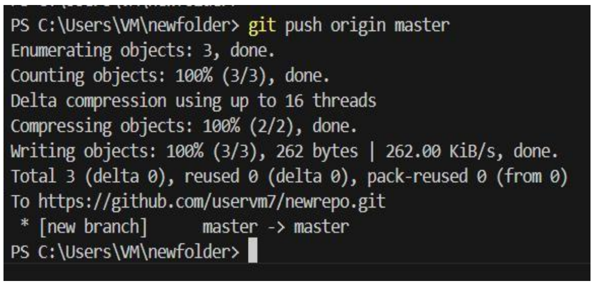
git config --global user.email "usermail@gmail.com"

STEP3: Create new repository in github.

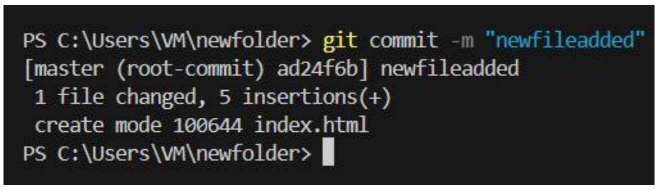
STEP4: Upload your code to Github using GIT-add,commit,push.



```
PS C:\Users\VM\newfolder> git add .
PS C:\Users\VM\newfolder> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html

PS C:\Users\VM\newfolder>
```



```
PS C:\Users\VM\newfolder> git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 262 bytes | 262.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/uservm7/newrepo.git
 * [new branch]      master -> master
PS C:\Users\VM\newfolder>
```



```
PS C:\Users\VM\newfolder> git commit -m "newfileadded"
[master (root-commit) ad24f6b] newfileadded
 1 file changed, 5 insertions(+)
 create mode 100644 index.html
PS C:\Users\VM\newfolder>
```

STEP5: Check if the code is present in Github.

INSTALL JENKINS IN CLOUD INSTANCE:

STEP 1: Create an EC2 instance (Linux) in cloud .
STEP 2: Go to Official website of
Jenkins(Jenkins.io) STEP 3 :In command prompt of
Jenkins "apt update".

STEP 4: Download Jenkins using the links in the official website called Jenkins.io.

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

Update your local package index, then finally install

Jenkins: sudo apt-get update

sudo apt-get install fontconfig openjdk-17-
jre sudo apt-get install Jenkins

STEP 5: Ping the public IP address(instance for Jenkins) in new tab, Jenkins page will open .

STEP 6: Login using password which is present in the path displayed in that login page.

STEP 7: Install suggested plugins in Jenkins.

STEP 8: Create new item in Jenkins.

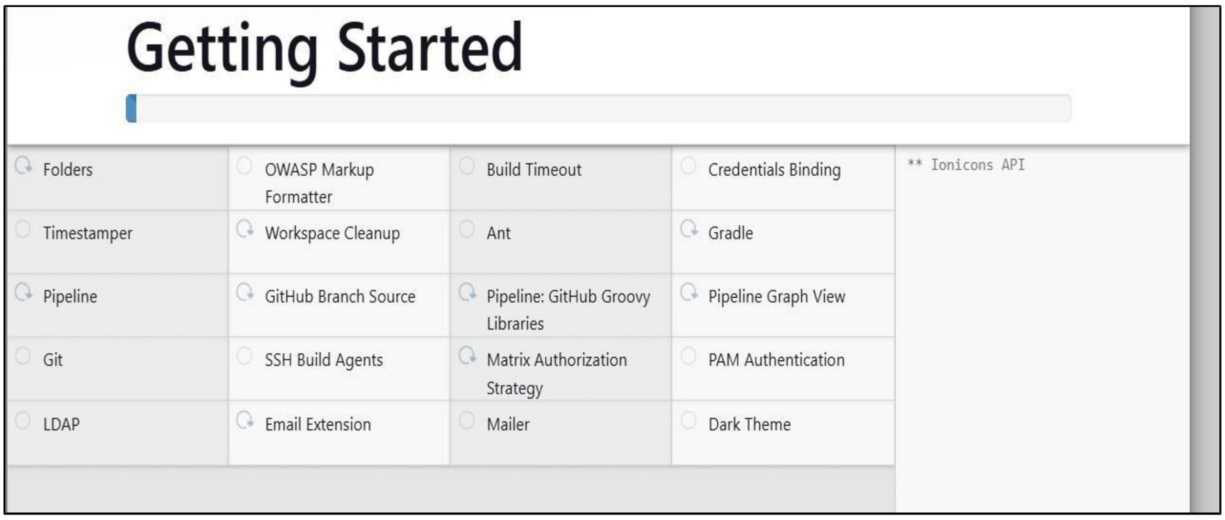
STEP 9: Use the Repository URL in Source Code Management.

STEP10: In Build Triggers Select POLLSCM(* * * * *).

STEP11: Save and apply, new build will create.

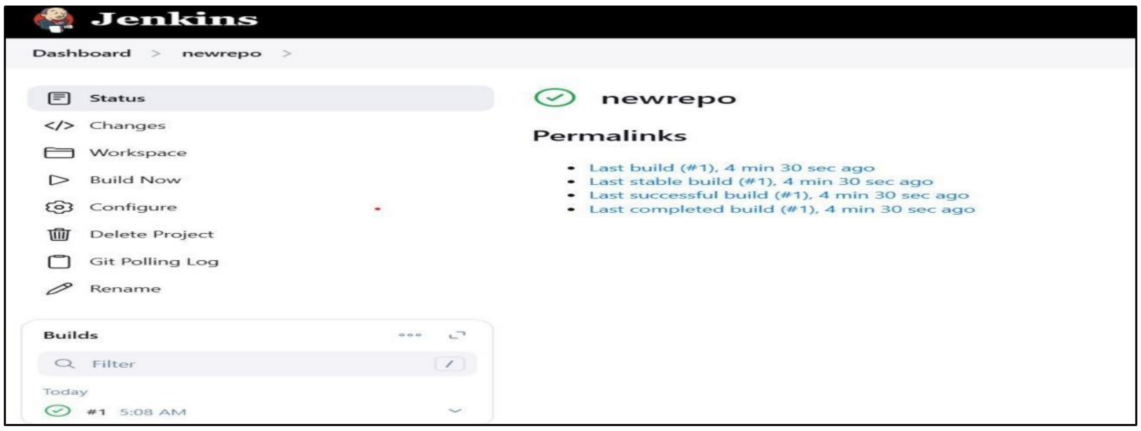
STEP12: Get into new build , check the console output.

OUTPUT:



Getting Started

Folders	OWASP Markup Formatter	Build Timeout	Credentials Binding	** Ionicons API
Timestamper	Workspace Cleanup	Ant	Gradle	
Pipeline	GitHub Branch Source	Pipeline: GitHub Groovy Libraries	Pipeline Graph View	
Git	SSH Build Agents	Matrix Authorization Strategy	PAM Authentication	
LDAP	Email Extension	Mailer	Dark Theme	



Jenkins

Dashboard > newrepo >

Status
Changes
Workspace
Build Now
Configure
Delete Project
Git Polling Log
Rename

Builds
Filter
Today
#1 5:08 AM

newrepo

Permalinks

- Last build (#1), 4 min 30 sec ago
- Last stable build (#1), 4 min 30 sec ago
- Last successful build (#1), 4 min 30 sec ago
- Last completed build (#1), 4 min 30 sec ago

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, the CD pipeline in Jenkins was created successfully and deployed in cloud.

Ex.no: 16	**INSTALL NGINX IN CLOUD INSTANCE**
DATE:	

AIM:

To install nginx in cloud instance and deploy a website in nginx.

PROCEDURE:

STEP 1: Create an EC2 instance (Linux) in cloud.

STEP 2: “apt update” and Install nginx in this instance.

```
Sudo apt update
Sudo apt install nginx -y
```

STEP 3: check the home path of nginx /var/www/html.

STEP 4: Ping the public IP address of nginx in new tab, you will be directed to html page.

STEP 5:Check the code of that html page in the path /var/www/html.

CONNECT JENKINS-NGINX:

STEP 1: In the command prompt of Jenkins, switch into Jenkins user using “su – Jenkins”.

STEP 2: Create SSH key using the command “ssh-keygen”.

STEP 3: After key generation extract the key using cat command with the public key path.

STEP 4: Copy the key and then go to nginx server.

STEP 5: Paste the key in the path /.ssh/authorized_keys, by that transferring of files between Jenkins and nginx got enabled.

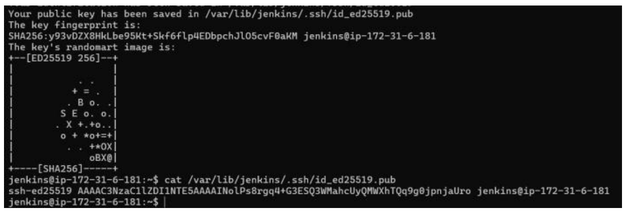
STEP 6: Go to configure ->Build steps->Execute shell

STEP 7: In Execute Shell ->scp /var/lib/Jenkins/workspace/reponame/filename root@nginx public IP address:/var/www/html(which the home path of nginx).

STEP 8: Save and Apply, new build will be created.

STEP 9: Reload the page where you got the output of nginx server by that the output of your html code will be displayed in that page. If you make any changes in your code ,the changes automatically displayed in that page by using Jenkins(automation Devops Tool).

OUTPUT:



```
Your public key has been saved in /var/lib/jenkins/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:y93vDZX8HkLbe95Kt+Skf6flp4EDbpchJlO5cvF0aKM jenkins@ip-172-31-6-181
The key's randomart image is:
+--[ED25519 256]--+
|                 |
|          . .    |
|         + = .   |
|        . B o. . |
|       S E o. o. |
|      . X +.+o.. |
|       o + *o+=+ |
|        . . ++OX |
|            oBX@ |
+----[SHA256]-----+
jenkins@ip-172-31-6-181:~$ cat /var/lib/jenkins/.ssh/id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAINolPs8rgq4+G3ESQ3WMahcUyQMWXhTQq9g0jpnjaUro jenkins@ip-172-31-6-181
jenkins@ip-172-31-6-181:~$ |
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, the CD pipeline in Jenkins was created successfully and deployed in cloud.

Ex.no: 17	INFRASTRUCTURE AS A CODE (IAAC) WITH TERRAFORM
DATE:	

AIM:

To enables users to provision, manage, and scale AWS resources in a declarative and automated manner.

PROCEDURE:

Provisioning cloud infrastructure like VMs, storage buckets, networks, and databases. Managing Kubernetes clusters and deployments.

Automating multi-cloud or hybrid infrastructure.

Ensuring consistent and repeatable infrastructure deploymentsImage.

Installation and Configuration and Install terraform:

Visit the Terraform downloads page.

Download the appropriate binary for your operating system (Windows, macOS, Linux).

```
Terraform --version
```

Resources Needed for AWS:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip" unzip awscliv2.zip

sudo ./aws/instal
l aws –version
```

Aws configure:

```
aws configure
```

To configure the Aws with the Username and Password

To Specify the provider (AWS) and configure the region:

```
provider "aws" { region = "us-east-1" # Defines the AWS region where the resources will be
created.}
resource "aws_security_group" "example_sg" {
  name_prefix = "example-sg-" # A unique name for the security
  group. ingress { from_port  = 22 # Allow SSH traffic.
```

to_port = 22 protocol = "tcp" cidr_blocks = ["0.0.0.0/0"] # Allow traffic from all IPs (use cautiously in production). }

egress { from_port = 0 to_port = 0 protocol = "-1" # All

protocols. cidr_blocks = ["0.0.0.0/0"] }}

To Launch an EC2 instance

resource "aws_instance" "example_instance" {ami = "ami-0c02fb55956c7d316" # Amazon Machine Image (Ubuntu Server in this example).

instance_type = "t2.micro" security_groups = [aws_security_group.example_sg.name] # Attach the security group.

tags = { Name = "ExampleInstance" # Tag to identify the instance in the AWS console. }}

Output the public IP of the instance

output "instance_public_ip" { value = aws_instance.example_instance.public_ip }

Type the following to apply the terraform code

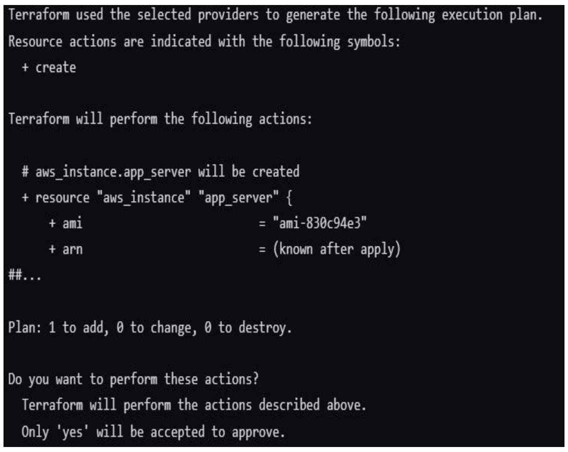
terraform init

terraform validate

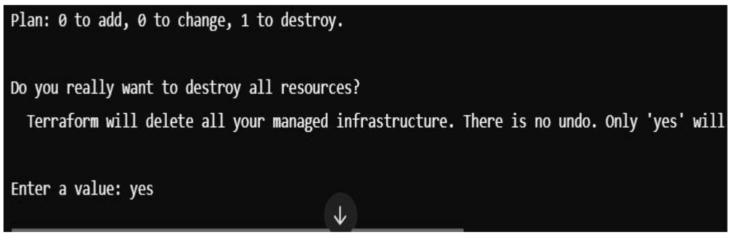
terraform plan

terraform apply

OUTPUT:



```
Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.app_server will be created
  + resource "aws_instance" "app_server" {
      + ami                          = "ami-830c94e3"
      + arn                          = (known after apply)
##...

Plan: 1 to add, 0 to change, 0 to destroy.

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



```
Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
  Terraform will delete all your managed infrastructure. There is no undo. Only 'yes' will

Enter a value: yes
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Terraform was successfully deployed for automated provision management, and scale AWS resources using a declarative approach.

Ex.no: 18	**LAUNCHING A WEBAPP USING MAVEN AND TOMCAT**
DATE:	

AIM:

To Launch and deploy a webapp using Tomcat and Maven in virtual machine.

PROCEDURE:

Step-1: Open terminal on your Linux machine (i.e.,Ubuntu)

Step-2: Update the system by command "apt update" and install JDK in your system by command "apt install default -jdk -y".

Step-3: Next, you need to install maven. Type cd /opt to change the working directory

and type "apt install maven" to install maven.

Step-4: Go to maven website and go to download and copy the binary tar.gz archive. Now go to the terminal and type 'wget <copied link>'

Step-5: Type tar -xvzf <filename that comes after the above process> to extract it

Step-6: Type mv <filename> <newname> to move the file and type ls to list the files in the directory

Step-7: Go to Git-Hub and clone the repository by the code link by git clone <git repository link>

Step-8: Type 'mv <repository name>/ maven' to move the file to maven and Type the

following mvn clean install

Now type cd target/ and type ls. If there is a '.war' file you have successfully installed maven and push your website to it.

Step-9: Now we have to install tomcat .For that go to tomcat website and go to download. In that copy the link of 'tar.gz(pgp,sha512'

Step-10: Type 'wget <copied link> in the terminal and extract the files by 'tar -xvzf <filename> and type ls for viewing the files in it

Step-11: Create a new directory by command 'mkdir <directory name>' and move the files extracted the new directory by 'mv <filenames> /opt/<directory name> and move the extracted file to 'tomcat' by mv <filename> tomcat. Type ls and type cd

Step -12: Type ls /usr/lib/jvm/ to copy a filename. Copy the second file name. Now type vi .bashrc and press i to insert the content into the file. Type "export JAVA_HOME=/usr/lib/jvm/<copied file>" and "export CATERLINE_HOME=/opt/tomcat and save it by clicking esc and type 'wq!'.

Step -13: Type source .bashrc and type cd /opt/tomcat to change the working directory and

type ls to list the files.

Step-14: Type the following command to transfer the '.war' file from maven to

tomcat: cd /opt/maven/ and type ls

cd <directory name> and type

ls cd target/ and type ls

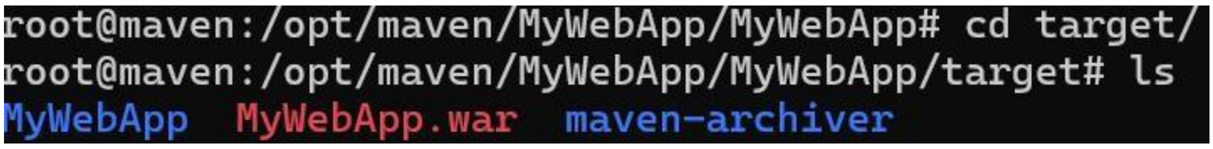
cp <.war file name> /opt/tomcat/webapps/

Step-15: Type cd /opt/tomcat/webapps/ to change the directory and type ls Now you will see the

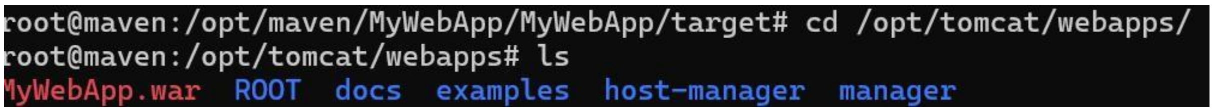
<.war file> in the directory

Step-16: Go to web and type <IP of tomcat>:8080:<directory name> and you will get your website into the webserver.

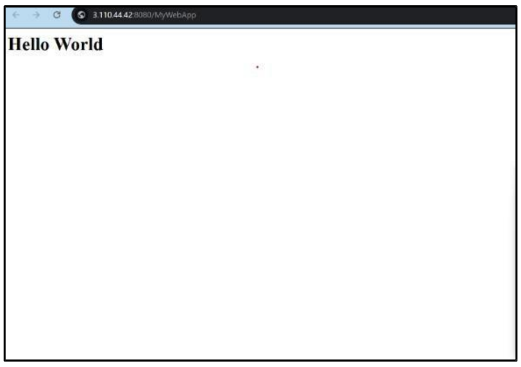
OUTPUT:



```
root@maven:/opt/maven/MyWebApp/MyWebApp# cd target/
root@maven:/opt/maven/MyWebApp/MyWebApp/target# ls
MyWebApp  MyWebApp.war  maven-archiver
```



```
root@maven:/opt/maven/MyWebApp/MyWebApp/target# cd /opt/tomcat/webapps/
root@maven:/opt/tomcat/webapps# ls
MyWebApp.war  ROOT  docs  examples  host-manager  manager
```



3.110.44.42:8080/MyWebApp

Hello World

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

The webApp is successfully Deployed using maven and tomcat in a remote virtual machine.

Ex.no: 19	**CREATING A DOCKER IMAGE WITH TOMCAT AND RUN**
DATE:	

AIM:

To create a Docker image with tomcat and run the image.

PROCEDURE:

Install Docker:

Install the docker on the machine. Download the docker from the official

website. docker –version

create a project repository:

Create a directory for your Tomcat Docker

project Run this command to create a directory

```
mkdir tomcat-docker && cd tomcat-docker
```

write a Docker file:

Create a Dockerfile in the project directory with the following

content. # Use the official Tomcat image from Docker Hub

```
FROM tomcat:latest

# Expose Tomcat default port (8080)

EXPOSE 8080

# Optional: Copy your web application (WAR file) to the Tomcat webapps

directory # COPY myapp.war /usr/local/tomcat/webapps/

# Start Tomcat server

CMD ["catalina.sh",

"run"]
```

Build a Docker image:

In the terminal, navigate to the project directory and build the image.

Command:

```
docker build -t my-tomcat-image .
```

Explanation: -t my-tomcat-image: Tags the image with the name my-tomcat-image. Refers to the current directory containing the Dockerfile.

Run the Docker container:

docker run -d -p 8080:8080 my-tomcat-image

Explanation: -d: Runs the container in detached mode (in the background).

-p 8080:8080: Maps port 8080 on your machine to port 8080 in the container.my-tomcat-image: The name of the Docker image to run.

Access the tomcat Browser:

Open the web browser and go to:

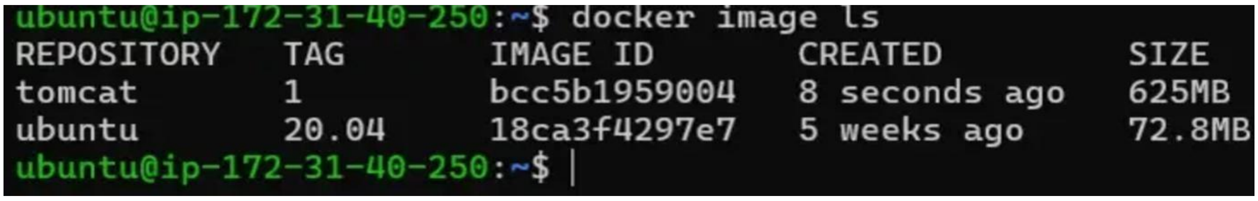
http://localhost:8080 - you can see the welcome tomcat page.

Stop the container:

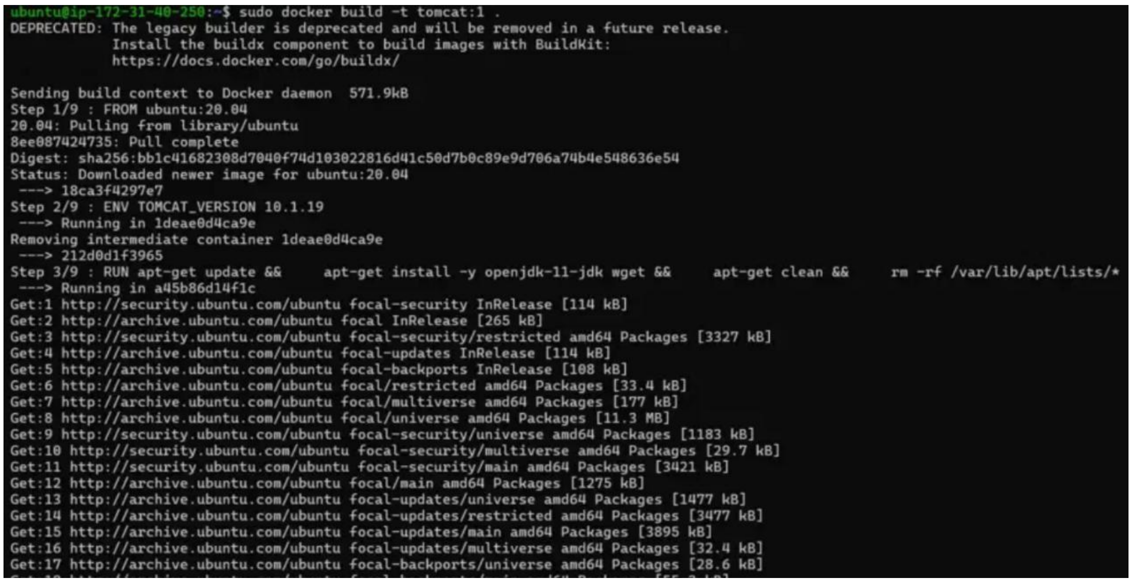
docker ps

docker stop <CONTAINER_ID>

OUTPUT:



```
ubuntu@ip-172-31-40-250:~$ docker image ls
REPOSITORY    TAG        IMAGE ID       CREATED          SIZE
tomcat        1          bcc5b1959004   8 seconds ago    625MB
ubuntu        20.04      18ca3f4297e7   5 weeks ago      72.8MB
ubuntu@ip-172-31-40-250:~$ |
```



```
ubuntu@ip-172-31-40-250:~$ sudo docker build -t tomcat:1 .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  571.9kB
Step 1/9 : FROM ubuntu:20.04
20.04: Pulling from library/ubuntu
8ee087424735: Pull complete
Digest: sha256:bb1c41682308d7040f74d103022816d41c50d7b0c89e9d706a74b4e548636e54
Status: Downloaded newer image for ubuntu:20.04
 ---> 18ca3f4297e7
Step 2/9 : ENV TOMCAT_VERSION 10.1.19
 ---> Running in 1deae0d4ca9e
Removing intermediate container 1deae0d4ca9e
 ---> 212d0d1f3965
Step 3/9 : RUN apt-get update &&     apt-get install -y openjdk-11-jdk wget &&     apt-get clean &&     rm -rf /var/lib/apt/lists/*
 ---> Running in a45b86d14f1c
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [3327 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
Get:8 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [1183 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [29.7 kB]
Get:11 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [3421 kB]
Get:12 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:13 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [1477 kB]
Get:14 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [3477 kB]
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [3895 kB]
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [32.4 kB]
Get:17 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [28.6 kB]
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus the Creating a Docker image with tomcat and run that image has been created and executed successfully.

Ex.no: 20	# INSTALLING PROMETHEUS ON UBUNTU
DATE:	

AIM:

To install Prometheus on ubuntu for log monitoring.

PROCEDURE:

Create a system user for Prometheus

sudo groupadd –system prometheus

sudo useradd -s /sbin/nologin –system -g prometheus prometheus

This will create a system user and group named "prometheus" for Prometheus with limited privileges, reducing the risk of unauthorized access.

Create Directories for Prometheus

Sudo mkdir /etc/Prometheus

Sudo mkdir

/var/lib/Prometheus

Download Prometheus and Extract Files

wget
https://github.com/prometheus/prometheus/releases/download/v2.43.0/prometheus-2.43.0.linux-amd64.tar.gz

After the download has been completed, run the following command to extract the contents of the downloaded file.

tar vxf

prometheus*.tar.gz Navigate to the

Prometheus Directory

After extracting the files, navigate to the newly extracted Prometheus directory using the following command:

cd prometheus*/

Configuring Prometheus on Ubuntu

22.04

Move the Binary Files & Set Owner

First, you need to move some binary files (prometheus and promtool) and change the ownership of the files to the "prometheus" user and group. You can do this with the following commands:

```
sudo mv prometheus

/usr/local/bin sudo mv promtool

/usr/local/bin

sudo chown prometheus:prometheus

/usr/local/bin/prometheus sudo chown

prometheus:prometheus /usr/local/bin/promtool
```

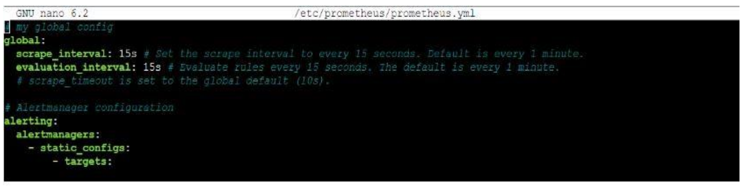
Move the Configuration Files & Set Owner

Next, move the configuration files and set their ownership so that Prometheus can access them. To do this, run the following commands:

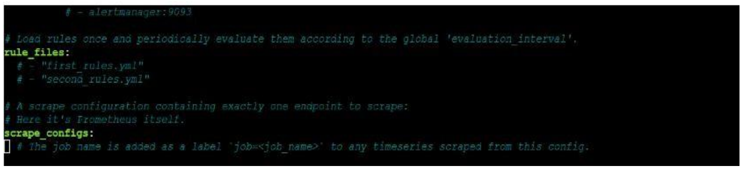
```
sudo mv consoles /etc/prometheus

sudo mv console_libraries

/etc/prometheus sudo mv

prometheus.yml /etc/prometheus

sudo chown prometheus:prometheus /etc/prometheus

sudo chown -R prometheus:prometheus /etc/prometheus/consoles

sudo chown -R prometheus:prometheus

/etc/prometheus/console_libraries sudo chown -R

prometheus:prometheus /var/lib/Prometheus
```

The prometheus.yml file is the main Prometheus configuration file. It includes settings for targets to be monitored, data scraping frequency, data processing, and storage. You can set alerting rules and notification conditions in the file. You don't need to modify this file for this demonstration but feel free to open it in an editor to take a closer look at its contents.

```
sudo nano /etc/prometheus/prometheus.yml
```



```
  GNU nano 6.2                              /etc/prometheus/prometheus.yml
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets:
```



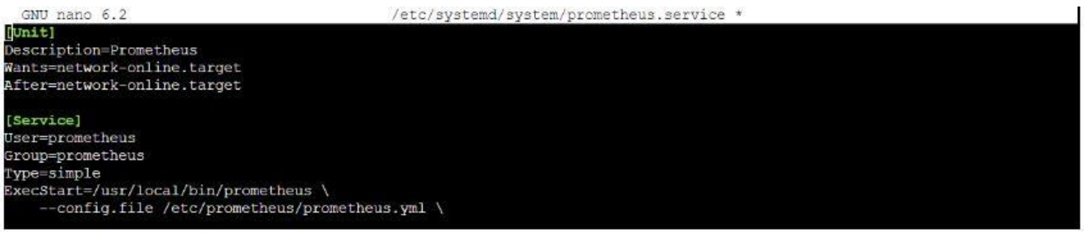
```
          # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
```

Now, you need to create a system service file for Prometheus. Create and open a prometheus.service file with the Nano text editor using:

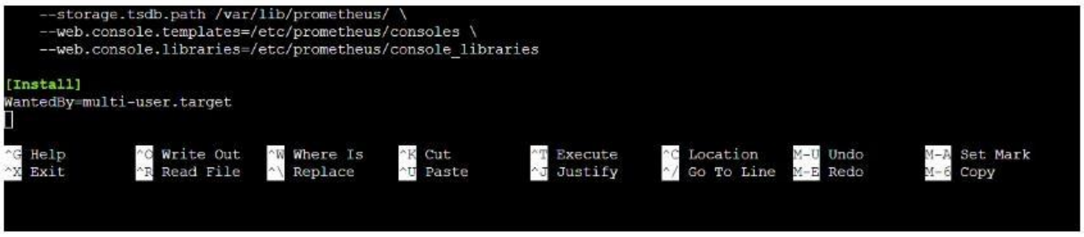
```
sudo nano /etc/systemd/system/prometheus.service
```

Include these settings to the file, save, and exit:

```
[Unit]
Description=Prometheus
Wants=network-
online.target
After=network-
online.target [Service]

User=prometheus
Group=prometheu
s Type=simple
ExecStart=/usr/local/bin/prometheus \
    --config.file /etc/prometheus/prometheus.yml \
    --storage.tsdb.path /var/lib/prometheus/ \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries
[Install]

WantedBy=multi-user.target
```



```
  GNU nano 6.2                    /etc/systemd/system/prometheus.service *
[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
    --config.file /etc/prometheus/prometheus.yml \
```



```
    --storage.tsdb.path /var/lib/prometheus/ \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries

[Install]
WantedBy=multi-user.target

^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
```

Reload Systemd

```
sudo systemctl daemon-reload
```

Start Prometheus Service

Next, you want to enable and start your Prometheus service. Do this using the following commands:

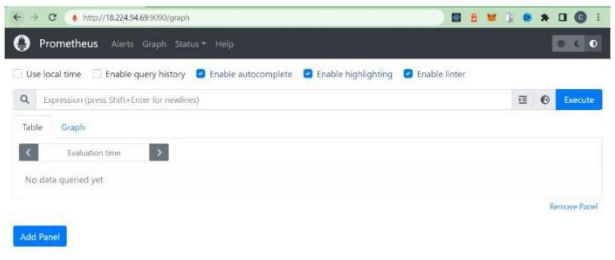
```
sudo systemctl enable prometheus
sudo systemctl start Prometheus
```

Check Prometheus Status

After starting the Prometheus service, you may confirm that it is running or if you have encountered errors using:

```
sudo systemctl status Prometheus
```

OUTPUT:



http://18.224.94.69:9090/graph

☐ Use local time ☐ Enable query history ☑ Enable autocomplete ☑ Enable highlighting ☑ Enable linter

Expression (press Shift+Enter for newlines)

No data queried yet

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

The log monitoring system Prometheus is successfully install and running on ubuntu.

Ex.no: 21	INSTALL GRAFANA IN UBUNTU
DATE:	

AIM:

To install Grafana on Ubuntu is to set up a robust data visualization and monitoring tool.

PROCEDURE:

Install the required packages

Next, run the following command to install the packages needed for the installation: sudo apt install -y apt-transport-https software-properties-common wget

Add the Grafana GPG key

sudo mkdir -p /etc/apt/keyrings/

wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor | sudo tee /etc/apt/keyrings/grafana.gpg > /dev/null Add Grafana APT repository

echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main" | sudo tee -a /etc/apt/sources.list.d/grafana.list

sudo apt update

Install Grafana

sudo apt install grafana

Start the Grafana service

Once the Grafana installation process has been completed, you can verify the version using: sudo grafana-server -v

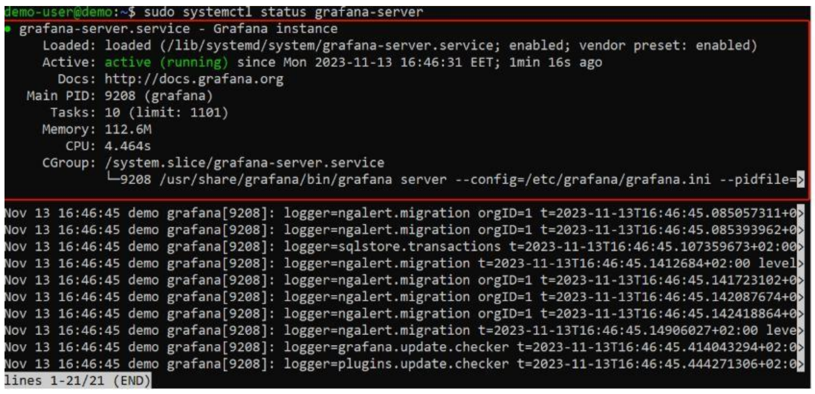
sudo systemctl start grafana-server sudo systemctl enable grafana-server

Verify that the Grafana service is running

Now verify that the Grafana service is active by running the command below:

Sudo systemctl status grafana-server

If the Grafana service was started successfully, you should see a sign that it is active and running



```
demo-user@demo:~$ sudo systemctl status grafana-server
● grafana-server.service - Grafana instance
     Loaded: loaded (/lib/systemd/system/grafana-server.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2023-11-13 16:46:31 EET; 1min 16s ago
       Docs: http://docs.grafana.org
   Main PID: 9208 (grafana)
      Tasks: 10 (limit: 1101)
     Memory: 112.6M
        CPU: 4.464s
     CGroup: /system.slice/grafana-server.service
             └─9208 /usr/share/grafana/bin/grafana server --config=/etc/grafana/grafana.ini --pidfile=>

Nov 13 16:46:45 demo grafana[9208]: logger=ngalert.migration orgID=1 t=2023-11-13T16:46:45.085057311+0>
Nov 13 16:46:45 demo grafana[9208]: logger=ngalert.migration orgID=1 t=2023-11-13T16:46:45.085393962+0>
Nov 13 16:46:45 demo grafana[9208]: logger=sqlstore.transactions t=2023-11-13T16:46:45.107359673+02:00>
Nov 13 16:46:45 demo grafana[9208]: logger=ngalert.migration t=2023-11-13T16:46:45.1412684+02:00 level>
Nov 13 16:46:45 demo grafana[9208]: logger=ngalert.migration orgID=1 t=2023-11-13T16:46:45.141723102+0>
Nov 13 16:46:45 demo grafana[9208]: logger=ngalert.migration orgID=1 t=2023-11-13T16:46:45.142087674+0>
Nov 13 16:46:45 demo grafana[9208]: logger=ngalert.migration orgID=1 t=2023-11-13T16:46:45.142418864+0>
Nov 13 16:46:45 demo grafana[9208]: logger=ngalert.migration t=2023-11-13T16:46:45.14906027+02:00 leve>
Nov 13 16:46:45 demo grafana[9208]: logger=grafana.update.checker t=2023-11-13T16:46:45.414043294+02:0>
Nov 13 16:46:45 demo grafana[9208]: logger=plugins.update.checker t=2023-11-13T16:46:45.444271306+02:0>
lines 1-21/21 (END)
```

Access the Grafana web interface

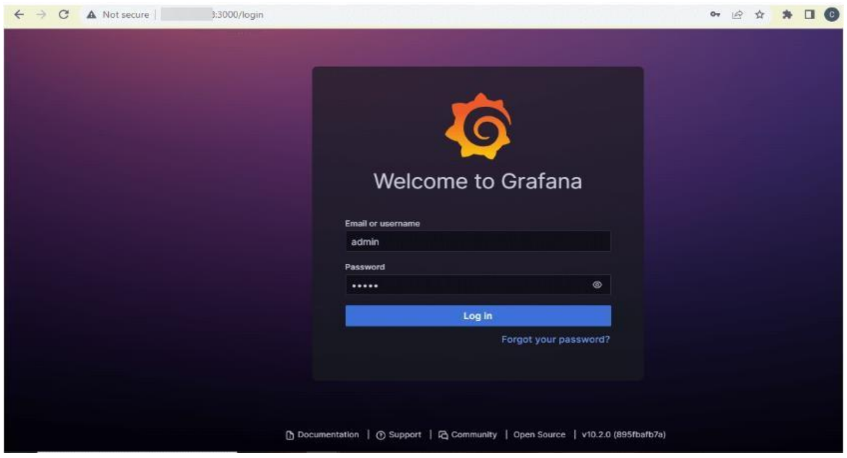
To access the Grafana web interface, open a web browser and enter the IP address of your server (or hostname if applicable), followed by port 3000. The URL format should

be http://your_server_IP:3000. Once loaded, you should see the Grafana login page. The default credentials are:

Username: admin

Password: admin

OUTPUT:



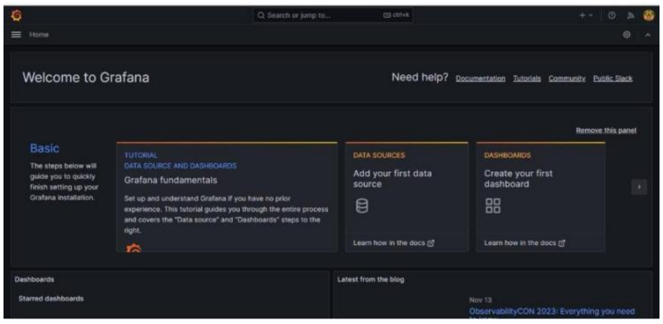
Not secure [redacted]:3000/login

Welcome to Grafana

Email or username: admin

Password: •••••

Log in
Forgot your password?
Documentation | Support | Community | Open Source | v10.2.0 (895fbafb7a)



Welcome to Grafana

Need help? Documentation Tutorials Community Public Slack

Basic

The steps below will guide you to quickly finish setting up your Grafana installation.

TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals

Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.

DATA SOURCES
Add your first data source

DASHBOARDS
Create your first dashboard

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Grafana is successful installed in ubuntu for monitoring the server performance.

Ex.No: 22	Launching an EC2 Instance with AWS CLI
Date:	

AIM:
To launch an Amazon EC2 instance using the AWS Command Line Interface (CLI) and verify connectivity.

PROCEDURE:
1. Set Up AWS CLI: Install and configure the AWS CLI with access credentials.
2. Create a Key Pair: Generate a key pair for SSH access.
3. Launch EC2 Instance: Use the AWS CLI to launch a t2.micro instance.
4. Verify Connectivity: SSH into the instance to confirm it is running.

COMMANDS:

```
# Install AWS CLI (on Ubuntu) [cite: 12]

sudo apt update [cite: 13]

sudo apt install awscli -y [cite: 14]


# Configure AWS CLI [cite: 15]

aws configure [cite: 16]

# Enter AWS Access Key ID, Secret Access Key, region (e.g., us-east-1), and output format (json)
[cite: 17]


# Create a key pair [cite: 18]

aws ec2 create-key-pair --key-name MyKeyPair --query 'KeyMaterial' --output text >
MyKeyPair.pem [cite: 19, 20]

chmod 400 MyKeyPair.pem [cite: 21]


# Launch EC2 instance (Amazon Linux 2 AMI) [cite: 22]

aws ec2 run-instances --image-id ami-0c55b159cbfafe1f0 --count 1 --instance-type t2.micro --key-
name MyKeyPair --security-group-ids <your-security-group-id> --subnet-id <your-subnet-id> --
query 'Instances[0].InstanceId' --output text [cite: 24]


# Get instance public IP (replace <instance-id>) [cite: 25]
```

```
aws ec2 describe-instances --instance-ids <instance-id> --query
'Reservations[0].Instances[0].PublicIpAddress' --output text [cite: 26]

# SSH into the instance [cite: 27]

ssh -i MyKeyPair.pem ec2-user@<public-ip> [cite: 28]
```

OUTPUT:

The run-instances command outputs the instance ID. The describe-instances command returns the public IP. An SSH connection yields the following prompt:

```
[ec2-user@ip-<public-ip> ~]$ [cite: 30]
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, the EC2 instance was successfully launched using the AWS CLI and confirmed to be accessible via SSH.

Ex.No: 23	AUTOMATING EC2 INSTANCE SETUP WITH USER DATA
Date:	

AIM:
To automate the installation of a web server (Apache) on an EC2 instance using a user data script.

PROCEDURE:
Steps:

Create User Data Script: Write a bash script to install Apache and create a simple webpage.

Launch EC2 Instance with User Data: Use the AWS CLI to pass the script during instance launch.

Verify Web Server: Access the instance's public IP in a web browser to confirm the setup.

COMMANDS:

```
# Create user data script (userdata.sh) [cite: 39]

cat <<EOL> userdata.sh [cite: 40, 42]

#!/bin/bash [cite: 41]

yum update -y [cite: 43]

yum install httpd -y [cite: 44]

systemctl start httpd [cite: 45]

systemctl enable httpd [cite: 46]

echo "<h1>Hello from EC2!</h1>" > /var/www/html/index.html [cite: 47]

EOL


# Launch EC2 instance with user data [cite: 48]

aws ec2 run-instances --image-id ami-0c55b159cbfafe1f0 --count 1 --instance-type t2.micro --key-name MyKeyPair --security-group-ids <your-security-group-id> --subnet-id <your-subnet-id> --user-data file://userdata.sh --query 'Instances[0].InstanceId' --output text [cite: 49, 50, 51]


# Get public IP [cite: 52]

aws ec2 describe-instances --instance-ids <instance-id> --query 'Reservations[0].Instances[0].PublicIpAddress' --output text [cite: 54]
```

```
# Access in browser or curl [cite: 55]
curl http://<public-ip> [cite: 56]
```

OUTPUT:

Accessing

http://<public-ip> in a browser or using the curl command displays the following:

```
<h1>Hello from EC2!</h1> [cite: 58]
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, the Apache web server was successfully installed and configured automatically on an EC2 instance using a user data script, confirming the web server is running and accessible.

Ex.No: 24	DEPLOYING A DOCKERIZED APPLICATION ON EC2
Date:	

AIM:
To deploy a Dockerized Node.js application on an EC2 instance using the AWS CLI and Docker.

PROCEDURE:
Steps:

Launch EC2 Instance: Start an instance and install Docker on it.

Create Dockerfile: Write a Dockerfile for a basic Node.js application.

Deploy via SSH: Copy the application files to the instance, then build and run the Docker container.

COMMANDS:

# Launch EC2 instance (Amazon Linux 2) [cite: 67]

aws ec2 run-instances --image-id ami-0c55b159cbfafe1f0 --count 1 --instance-type t2.micro --key-name MyKeyPair --security-group-ids <your-security-group-id> --subnet-id <your-subnet-id> --query 'Instances[0].InstanceId' --output text [cite: 68]

# SSH into instance [cite: 69]

ssh -i MyKeyPair.pem ec2-user@<public-ip> [cite: 70]

# On instance: Install Docker [cite: 71]

sudo yum update -y [cite: 72]

sudo amazon-linux-extras install docker [cite: 73]

sudo systemctl start docker [cite: 74]

sudo usermod -a -g docker ec2-user [cite: 75]

exit [cite: 76]

# Create Node.js app and Dockerfile locally [cite: 77]

mkdir node-app && cd node-app [cite: 78, 79]

echo 'const http = require("http"); http.createServer((req, res) => { res.writeHead(200, {"Content-Type": "text/plain"}); res.end("Hello, Docker on EC2!"); }).listen(8080);' > app.js [cite: 80, 81, 82, 83, 85, 87]

echo '{"name": "node-app", "version": "1.0.0"}' > package.json [cite: 86, 88]

```
echo -e 'FROM node:16\nWORKDIR /app\nCOPY . .\nRUN npm install\nEXPOSE 8080\nCMD ["node", "app.js"]' > Dockerfile [cite: 89, 90, 91, 92, 93, 94]

# Copy to instance [cite: 95]

scp -i MyKeyPair.pem -r node-app ec2-user@<public-ip>:~/ [cite: 96]

# SSH again and deploy [cite: 97]

ssh -i MyKeyPair.pem ec2-user@<public-ip> [cite: 98]

cd node-app [cite: 99]

sudo docker build -t node-app . [cite: 100]

sudo docker run -d -p 8080:8080 node-app [cite: 101]

exit [cite: 102]

# Verify [cite: 103]

curl http://<public-ip>:8080 [cite: 104]
```

OUTPUT:

The curl command to the public IP on port 8080 returns the following message:

```
Hello, Docker on EC2! [cite: 106]
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, a Node.js application was successfully containerized using Docker and deployed on an EC2 instance.

Ex.No: 25	SETTING UP AN AUTO-SCALING GROUP FOR EC2
Date:	

AIM:
To create an EC2 Auto-Scaling Group (ASG) to automatically scale instances based on CPU load.

PROCEDURE:
Steps:

Create Launch Template: Define a launch template that specifies the configuration for new EC2 instances.

Configure ASG: Set up an Auto-Scaling Group with scaling policies based on CPU utilization.

Test Scaling: Simulate a high CPU load on an instance to trigger the scaling policy.

Verify: Check the Auto-Scaling Group in the AWS Console to observe the increase in instance count.

COMMANDS:

```
# Create launch template [cite: 115]

aws ec2 create-launch-template --launch-template-name MyTemplate --version-description v1 --
launch-template-data '{"ImageId":"ami-0c55b159cbfafe1f0", "InstanceType": "t2.micro",
"KeyName":"MyKeyPair", "SecurityGroupIds":["<your-security-group-id>"]}' [cite: 116, 117,
118]


# Create Auto-Scaling Group [cite: 119]

aws autoscaling create-auto-scaling-group --auto-scaling-group-name MyASG --launch-template
LaunchTemplateName=MyTemplate,Version=1 --min-size 1 --max-size 3 --desired-capacity 1 --
vpc-zone-identifier <your-subnet-id> [cite: 121]


# Add scaling policy [cite: 122]

aws autoscaling put-scaling-policy --auto-scaling-group-name MyASG --policy-name cpu-scale --
policy-type TargetTrackingScaling --target-tracking-configuration '{"TargetValue":50.0,
"PredefinedMetricSpecification":{"PredefinedMetricType":"ASGAverageCPUUtilization"}}' [cite
123]


# SSH into instance and simulate load (replace <public-ip>) [cite: 124]

ssh -i MyKeyPair.pem ec2-user@<public-ip>
```

```
stress --cpu 2 --timeout 300 [cite: 124]
```

OUTPUT:

Initially, the AWS Console shows the "MyASG" Auto-Scaling Group with 1 instance. After running the stress command to increase CPU load, the instance count in the ASG increases to 2 or 3. CloudWatch metrics will show that the average CPU utilization has exceeded the 50% target.

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, an EC2 Auto-Scaling Group was successfully created and tested, demonstrating its ability to automatically launch new instances in response to increased CPU load.

Ex.No: 26	CI/CD PIPELINE WITH EC2 AND GITHUB ACTIONS
Date:	

AIM:
To automate the deployment of a Python Flask application to an EC2 instance using GitHub Actions.

PROCEDURE:
Steps:

Create GitHub Repository: Set up a repository containing a simple Flask application.

Configure EC2: Prepare the EC2 instance by installing Python and ensuring SSH access is available.

Set Up GitHub Secrets: Store the EC2 host details and private SSH key as secrets in the GitHub repository.

Create Workflow: Define a GitHub Actions workflow file (.yml) to automate the deployment steps upon a push to the repository.

Verify Deployment: Access the application running on the EC2 instance's public IP to verify the deployment.

COMMANDS:

```
# On EC2 instance: Install Python and dependencies [cite: 137]

ssh -i MyKeyPair.pem ec2-user@<public-ip>

sudo yum update -y [cite: 138]

sudo yum install python3 python3-pip -y [cite: 139]

exit [cite: 139]

# Locally: Create Flask app and GitHub repo [cite: 140]

mkdir flask-app && cd flask-app [cite: 140, 141]

echo 'from flask import Flask\napp = Flask(__name__)\n@app.route("/")\ndef hello():\n    return "Hello from EC2 CI/CD!"\nif __name__ == "__main__":\n    app.run(host="0.0.0.0", port=5000)' > app.py [cite: 142, 143, 144, 145, 146, 148, 149]

echo 'flask' > requirements.txt [cite: 150]

git init [cite: 151]

git add . [cite: 152]
git commit -m "Initial Flask app" [cite: 153]
git remote add origin https://github.com/<your-username>/flask-ec2.git [cite: 154, 155, 157]
git push origin main [cite: 156]

# Create GitHub Actions workflow file: .github/workflows/deploy.yml [cite: 158, 159]
cat <<EOL> .github/workflows/deploy.yml [cite: 160]
name: Deploy to EC2 [cite: 161]
on: [push] [cite: 162]
jobs: [cite: 163]
  deploy: [cite: 164]
    runs-on: ubuntu-latest [cite: 166]
    steps: [cite: 167]
      - uses: actions/checkout@v3 [cite: 168]
      - name: Deploy to EC2 [cite: 169]
        uses: appleboy/ssh-action@v0.1.10 [cite: 170]
        with: [cite: 171]
          host: ${{ secrets.EC2_HOST }} [cite: 172]
          username: ec2-user [cite: 172]
          key: ${{ secrets.EC2_SSH_KEY }} [cite: 173]
          script: | [cite: 173]
            cd ~/flask-app
            git pull origin main [cite: 175]
            pip3 install -r requirements.txt [cite: 176]
            pkill -f "flask" || true [cite: 177]
            nohup python3 app.py & [cite: 178]
EOL

# Commit and push the workflow file [cite: 179]
git add . [cite: 180]
git commit -m "Add GitHub Actions workflow" [cite: 181]
git push origin main [cite: 182]


# Verify [cite: 183]

curl http://<public-ip>:5000 [cite: 184]
```

OUTPUT:

The GitHub Actions workflow run completes successfully with a green checkmark in the repository's "Actions" tab. The

curl command to the public IP on port 5000 returns the following message:

```
Hello from EC2 CI/CD! [cite: 187]
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, a full CI/CD pipeline using GitHub Actions was created to automatically deploy a Flask application to an EC2 instance whenever changes are pushed to the main branch.

Ex.No: 27	SETTING UP A BASIC CI/CD PIPELINE WITH GITHUB ACTIONS
Date:	

AIM:
To create a Continuous Integration/Continuous Deployment

(CI/CD) pipeline using GitHub Actions to automate testing and deployment of a simple Python application.

PROCEDURE:
Steps:

Create a GitHub Repository: Log in to GitHub and create a new repository (e.g., python-ci-example). Initialize it with a README and a

.gitignore for Python.

Add a Python Script: Create a file app.py with a simple function and a test file test_app.py using pytest.

Set Up GitHub Actions Workflow: Create a .github/workflows/ci.yml file to define the CI pipeline and push the code to GitHub.

Verify the Pipeline: Check the "Actions" tab in the GitHub repository to see the workflow run.

COMMANDS:

```
# Clone the repository locally

git clone https://github.com/<your-username>/python-ci-example.git [cite: 203, 204]

cd python-ci-example [cite: 205]


# Create app.py

echo 'def add(a, b): return a + b' > app.py [cite: 207]

# Create test_app.py

echo 'import pytest

from app import add

def test_add(): assert add(2, 3) == 5' > test_app.py [cite: 209, 210, 211, 212]

# Create requirements.txt

echo 'pytest' > requirements.txt [cite: 215]
```

```
# Create GitHub Actions workflow file
mkdir -p .github/workflows [cite: 217]
cat <<EOL> .github/workflows/ci.yml
name: Python CI [cite: 218]
on: [push] [cite: 219]
jobs: [cite: 220]
  build: [cite: 221]
    runs-on: ubuntu-latest [cite: 223]
    steps: [cite: 224]
      - uses: actions/checkout@v3 [cite: 225]
      - name: Set up Python [cite: 226]
        uses: actions/setup-python@v4 [cite: 227]
        with:
          python-version: '3.9' [cite: 228]
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt [cite: 229]
      - name: Run tests [cite: 230]
        run: pytest [cite: 231]
EOL

# Commit and push changes
git add . [cite: 233]
git commit -m "Set up Python CI pipeline" [cite: 234]
git push origin main [cite: 235]
```

OUTPUT:

In the GitHub repository's "Actions" tab, the "Python CI" workflow runs automatically on push. A green checkmark indicates successful execution, with logs showing that one test was collected and passed.

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, a basic CI/CD pipeline was successfully created using GitHub Actions to automate the testing of a Python application upon code push.

Ex.No: 28	AUTOMATING DOCKER CONTAINER BUILDS WITH GITHUB ACTIONS
Date:	

AIM:
To automate the building and pushing of a Docker image to Docker Hub using GitHub Actions for a simple Node.js application.

PROCEDURE:
Steps:

Create a GitHub Repository: Create a new repository (e.g., node-docker-example).

Add a Node.js Application: Create a simple Node.js application with a Dockerfile.

Configure Docker Hub Credentials: Add your Docker Hub username and access token as GitHub Secrets (DOCKER_USERNAME, DOCKER_PASSWORD).

COMMANDS:

```
# Clone the repository

git clone https://github.com/<your-username>/node-docker-example.git [cite: 256, 258]

cd node-docker-example [cite: 259]


# Create a simple Node.js app (app.js)

echo 'const http = require("http");

http.createServer((req, res) => {

  res.writeHead(200, {"Content-Type": "text/plain"});

  res.end("Hello, DevOps!");

}).listen(8080);' > app.js [cite: 261, 262, 263, 264, 265]


# Create package.json

echo '{"name": "node-docker", "version": "1.0.0"}' > package.json [cite: 267]


# Create Dockerfile

echo 'FROM node:16
```

```
WORKDIR /app
COPY . .
RUN npm install
EXPOSE 8080
CMD ["node", "app.js"]' > Dockerfile [cite: 269, 270, 271, 272, 273, 274]

# Create GitHub Actions workflow
mkdir -p .github/workflows [cite: 276]
cat <<EOL> .github/workflows/docker.yml
name: Docker Build and Push [cite: 278]
on: [push] [cite: 279]
jobs: [cite: 280]
  build: [cite: 281]
    runs-on: ubuntu-latest [cite: 282]
    steps: [cite: 283]
      - uses: actions/checkout@v3 [cite: 284]
      - name: Log in to Docker Hub
        uses: docker/login-action@v2
        with:
          username: \${{ secrets.DOCKER_USERNAME }} [cite: 285]
          password: \${{ secrets.DOCKER_PASSWORD }} [cite: 285]
      - name: Build and push Docker image [cite: 286]
        uses: docker/build-push-action@v4 [cite: 286]
        with:
          context: . [cite: 289]
          push: true [cite: 290]
          tags: \${{ secrets.DOCKER_USERNAME }}/node-app:latest [cite: 291]
EOL

# Commit and push
```

git add . [cite: 293]

git commit -m "Add Node.js app and Docker workflow" [cite: 294]

git push origin main [cite: 295]

OUTPUT:

The "Actions" tab shows the "Docker Build and Push" workflow with a green checkmark. The Docker image is available on Docker Hub as

<your-username>/node-app:latest. Running the container locally via

docker run -p 8080:8080 <your-username>/node-app:latest and accessing http://localhost:8080 displays: Hello, DevOps!.

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, the process of building a Docker container for a Node.js application and pushing it to Docker Hub was successfully automated using GitHub Actions.

Ex.No: 29	MANAGING INFRASTRUCTURE AS CODE WITH TERRAFORM AND GITHUB
Date:	

AIM:

To use Terraform to define and deploy a simple AWS S3 bucket, version-controlled in a GitHub repository.

PROCEDURE:

Steps:

Create a GitHub Repository: Create a new repository (e.g., terraform-s3-example).

Set Up Terraform Configuration: Create a main.tf file to define an AWS S3 bucket.

Configure AWS Credentials: Add AWS access key and secret key as GitHub Secrets (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY).

COMMANDS:

```
# Clone the repository

git clone https://github.com/<your-username>/terraform-s3-example.git [cite: 308]

cd terraform-s3-example [cite: 309]


# Create main.tf

cat <<EOL> main.tf

provider "aws" { [cite: 312]

  region = "us-east-1" [cite: 313]

}

resource "aws_s3_bucket" "example" { [cite: 315]

  bucket = "my-unique-bucket-<your-username>" [cite: 316]

}

EOL


# Create GitHub Actions workflow

mkdir -p .github/workflows [cite: 319]

cat <<EOL> .github/workflows/terraform.yml
```

```
name: Terraform Apply [cite: 322]
on: [push] [cite: 323]
jobs: [cite: 324]
  terraform: [cite: 325]
    runs-on: ubuntu-latest [cite: 326]
    steps: [cite: 327]
      - uses: actions/checkout@v3 [cite: 328]
      - name: Set up Terraform [cite: 329]
        uses: hashicorp/setup-terraform@v2 [cite: 330]
        with:
          terraform_version: 1.5.0 [cite: 331]
      - name: Terraform Init [cite: 332]
        run: terraform init [cite: 333]
        env:
          AWS_ACCESS_KEY_ID: \${{ secrets.AWS_ACCESS_KEY_ID }} [cite: 335]
          AWS_SECRET_ACCESS_KEY: \${{ secrets.AWS_SECRET_ACCESS_KEY }}[cite: 335,
336]
      - name: Terraform Apply [cite: 337]
        run: terraform apply -auto-approve [cite: 338]
        env:
          AWS_ACCESS_KEY_ID: \${{ secrets.AWS_ACCESS_KEY_ID }} [cite: 340]
          AWS_SECRET_ACCESS_KEY: \${{ secrets.AWS_SECRET_ACCESS_KEY }} [cite: 340,
341]
EOL

# Commit and push
git add . [cite: 344]
git commit -m "Add Terraform S3 configuration" [cite: 345]
git push origin main [cite: 345]
```

OUTPUT:

The "Actions" tab shows the "Terraform Apply" workflow completing successfully. The AWS Management Console displays a new S3 bucket named

my-unique-bucket-<your-username>. The workflow logs include:

aws_s3_bucket.example: Creation complete.

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, an AWS S3 bucket was successfully provisioned using a Terraform configuration file, which was version-controlled in GitHub and deployed via a GitHub Actions workflow.

Ex.No: 30	**IMPLEMENTING A GITHUB PULL REQUEST WORKFLOW**
Date:	

AIM:
To demonstrate a collaborative development workflow using GitHub pull requests for code review and merging.

PROCEDURE:
Steps:

Create a GitHub Repository: Create a new repository (e.g., collab-example).

Add a Simple File: Create a README.md file with initial content.

Create a Feature Branch: Create a branch, make changes, and push it.

Create a Pull Request: Open a pull request on GitHub and merge it after review.

Verify the Merge: Ensure the changes appear in the main branch.

COMMANDS:

```
# Clone the repository

git clone https://github.com/<your-username>/collab-example.git [cite: 359]

cd collab-example [cite: 359]


# Create initial README.md

echo '# Collaborative Project' > README.md [cite: 361]

git add README.md [cite: 362]

git commit -m "Initial README" [cite: 363]

git push origin main [cite: 364]


# Create a feature branch

git checkout -b feature-update [cite: 366]

echo '## Feature: Added description' >> README.md [cite: 367]

git add README.md [cite: 368]

git commit -m "Add description to README" [cite: 369]
```

git push origin feature-update [cite: 370]

OUTPUT:

On GitHub, a pull request from

feature-update to main is created. After merging, the

README.md in the main branch contains the updated text:

# Collaborative Project

## [cite_start]Feature: Added description [cite: 373, 374]

The "Pull requests" tab shows the merged PR with a status of "Merged".

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, a standard collaborative workflow using branches and pull requests for code review and merging was successfully demonstrated on GitHub.

Ex.No: 31	MONITORING A WEB APPLICATION WITH GITHUB ACTIONS AND PROMETHEUS
Date:	

AIM:
To set up a simple web application with Prometheus monitoring and automate its deployment using GitHub Actions.

PROCEDURE:
Steps:

Create a GitHub Repository: Create a new repository (e.g., web-monitor-example).

Add a Python Flask App: Create a Flask application that exposes Prometheus metrics.

Add Prometheus Configuration: Create a prometheus.yml file.

Set Up GitHub Actions Workflow: Create a workflow to build and test the application.

Push and Verify: Push the code and verify the workflow and metrics endpoint.

COMMANDS:

```
# Clone the repository

git clone https://github.com/<your-username>/web-monitor-example.git [cite: 384, 386]

cd web-monitor-example [cite: 387]


# Create app.py

cat <<EOL> app.py

from flask import Flask [cite: 391]

from prometheus_flask_exporter import PrometheusMetrics [cite: 392]

app = Flask(__name__) [cite: 393]

metrics = PrometheusMetrics(app) [cite: 394]

@app.route('/') [cite: 395]

def hello(): [cite: 396]

    return 'Hello, DevOps!' [cite: 397]

# Metrics are exposed at /metrics by default

if __name__ == '__main__': [cite: 401]

    app.run(host='0.0.0.0', port=5000) [cite: 403]
```

```
EOL

# Create requirements.txt
echo 'flask
prometheus-flask-exporter' > requirements.txt [cite: 405, 406]

# Create prometheus.yml
echo 'scrape_configs:
  - job_name: "flask-app"
    static_configs:
      - targets: ["localhost:5000"]' > prometheus.yml [cite: 408, 409, 410, 411]

# Create GitHub Actions workflow
mkdir -p .github/workflows [cite: 413]
cat <<EOL> .github/workflows/ci.yml
name: Flask CI [cite: 415, 416]
on: [push] [cite: 417]
jobs: [cite: 418]
  build: [cite: 419]
    runs-on: ubuntu-latest [cite: 421]
    steps: [cite: 422]
      - uses: actions/checkout@v3 [cite: 423]
      - name: Set up Python [cite: 424]
        uses: actions/setup-python@v4 [cite: 425]
        with:
          python-version: '3.9' [cite: 426]
      - name: Install dependencies [cite: 427]
        run: pip install -r requirements.txt [cite: 428]
      - name: Run Flask app in background [cite: 429]
        run: python app.py & [cite: 430]
      - name: Test metrics endpoint [cite: 431]
        run: curl http://localhost:5000/metrics [cite: 432]
EOL


# Commit and push
git add . [cite: 434]
git commit -m "Add Flask app with Prometheus monitoring" [cite: 436]
git push origin main [cite: 436]
```

OUTPUT:

The "Actions" tab shows the "Flask CI" workflow completing successfully. The

curl command in the workflow output displays Prometheus metrics. When running locally, accessing

http://localhost:5000 displays Hello, DevOps!.

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Thus, a Python Flask application with a Prometheus metrics endpoint was successfully created, and its testing was automated using a GitHub Actions workflow.

Ex.No: 32	**Basic Ansible Playbook for System Configuration**
Date:	

AIM: To create and execute a basic Ansible playbook to configure a remote server's timezone and install a package.

PROCEDURE:

1. Set Up Ansible Environment: Install Ansible on your control node and ensure SSH access to a target node.
2. Create Inventory File: Define the target host in an inventory file.
3. Create Playbook: Write a YAML playbook to set the timezone and install a package like ntp.
4. Execute Playbook: Run the playbook and verify changes on the target host.

Commands:

```
Bash
# Install Ansible (on control node, assuming Ubuntu)
sudo apt update [cite: 640]
sudo apt install ansible -y [cite: 641]

# Create inventory file
echo '[webservers]
target-host ansible_host=192.168.1.100 ansible_user=ubuntu
ansible_ssh_private_key_file=~/.ssh/id_rsa' > inventory.ini [cite: 643, 644, 645]

# Create playbook YAML
cat <<EOL> configure_timezone.yml
- name: Configure Timezone and Install NTP
  hosts: webservers
  become: yes
  tasks:
  - name: Set timezone to UTC
    timezone:
      name: UTC
  - name: Install NTP package
    apt:
      name: ntp
      state: present
EOL [cite: 647, 648, 649, 650, 652, 653, 654, 655, 656, 657, 658, 659]

# Run the playbook
ansible-playbook -i inventory.ini configure_timezone.yml [cite: 661]
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

The playbook execution shows changed status for setting the timezone and installing the NTP package. On the target host, timedatectl shows UTC timezone, and ntp is installed.

Ex.No: 33	**Ansible Role for Web Server Deployment**
Date:	

AIM: To develop an Ansible role to deploy an Apache web server and a simple HTML page on multiple hosts.

PROCEDURE:

1. Create Role Structure: Use

ansible-galaxy to initialize a role.

2. Define Tasks: Add tasks to install Apache, start the service, and copy an HTML file.
3. Create Playbook: Write a playbook that uses the role.
4. Execute and Verify: Run the playbook and access the web page.

Commands:

Bash

```
# Initialize role

ansible-galaxy init roles/apache_web [cite: 694]


# Add tasks to roles/apache_web/tasks/main.yml

cat <<EOL> roles/apache_web/tasks/main.yml

- name: Install Apache

  apt:

    name: apache2

    state: present

- name: Start Apache service

  service:

    name: apache2

    state: started

    enabled: yes

- name: Copy index.html

  copy:

    content: "<h1>Hello from Ansible!</h1>"
    dest: /var/www/html/index.html
EOL [cite: 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 711]

# Create playbook
cat <<EOL> deploy_web.yml
- name: Deploy Apache Web Server
  hosts: webservers
  become: yes
  roles:
    - apache_web
EOL [cite: 713, 714, 715, 716, 717, 719]

# Run the playbook
ansible-playbook -i inventory.ini deploy_web.yml [cite: 721]
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

Playbook output shows changed tasks for installation and copy. Accessing the web server shows: <h1>Hello from Ansible!</h1>.

Ex.No: 34	Ansible with Variables and Templates
Date:	

AIM: To use Ansible variables and Jinja2 templates to customize configuration files on remote hosts.

PROCEDURE:

Create Template: Make a Jinja2 template for a configuration file.

Define Variables: Add host-specific variables in a vars file.

Create Playbook: Use the template module to deploy the customized file.

Execute and Verify: Run the playbook and check the config file on the host.

Commands:

Bash

```
# Create vars file [cite: 731]

cat <<EOL> vars.yml

app_name: MyApp

log_level: INFO

EOL [cite: 733]


# Create template [cite: 735]

mkdir templates [cite: 736]

cat <<EOL> templates/config.j2

Application: {{ app_name}} [cite: 738]

Log Level: {{ log_level }} [cite: 739]

EOL [cite: 740]


# Create playbook [cite: 741]

cat <<EOL> customize_config.yml

- name: Customize Config with Template [cite: 743]

  hosts: webservers [cite: 743]

  become: yes [cite: 744]

  vars_files: [cite: 745]
```

```
    - vars.yml [cite: 746]
  tasks: [cite: 747]
  - name: Deploy config file [cite: 749]
    template: [cite: 749]
      src: templates/config.j2 [cite: 750]
      dest: /etc/myapp/config.conf [cite: 751]
EOL

# Run the playbook [cite: 752]
ansible-playbook -i inventory.ini customize_config.yml [cite: 753]
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

The playbook shows a "changed" status for the template task. On the target host, the content of /etc/myapp/config.conf is:

```
Application: MyApp [cite: 755]
Log Level: INFO [cite: 756]
```

Ex.No: 35	Ansible Vault for Secure Secrets Management
Date:	

AIM:

To use Ansible Vault to encrypt sensitive data like passwords and deploy them securely.

PROCEDURE:

Create Vault File: Encrypt a file containing secrets.

Create Playbook: Use the encrypted variables in a playbook to create a user with a password.

Execute Playbook: Run with the vault password.

Verify: Check if the user is created on the target host.

Commands:

Bash

```
# Create vault file [cite: 766]

echo 'vault_password: secretpass' > secrets.yml

ansible-vault encrypt secrets.yml [cite: 767]


# Create playbook [cite: 768]

cat <<EOL> secure_user.yml

- name: Create User with Vault Password [cite: 771]

  hosts: webservers [cite: 772]

  become: yes [cite: 773]

  vars_files: [cite: 774]

    - secrets.yml [cite: 775]

  tasks: [cite: 776]

  - name: Create user [cite: 777]

    user: [cite: 778]

      name: secureuser [cite: 779]

      password: "{{ vault_password | password_hash('sha512') }}" [cite: 780]

EOL
```

```
# Run the playbook (provide password when prompted) [cite: 783]

ansible-playbook -i inventory.ini secure_user.yml --vault-id prod@prompt [cite: 784]
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

The playbook executes without exposing the password. On the target machine, the command

Ex.No: 36	Integrating Ansible with GitHub for CI/CD
Date:	

AIM:

To integrate Ansible with GitHub Actions for automated deployment in a CI/CD pipeline.

PROCEDURE:

Create GitHub Repository: Set up a repository with an Ansible playbook.

Add SSH Key to GitHub Secrets: Store the private key for Ansible access.

Create Workflow: Define a GitHub Actions YAML to run Ansible on push.

Push and Verify: Commit, push, and check the Actions tab.

Commands:

YAML

```
# In GitHub repo, create .github/workflows/ansible.yml [cite: 794]
[cite_start]name: Ansible Deployment [cite: 795]
[cite_start]on: [push] [cite: 796]
[cite_start]jobs: [cite: 797]
  [cite_start]deploy: [cite: 798]
    [cite_start]runs-on: ubuntu-latest [cite: 799]
    steps:
      - [cite_start]uses: actions/checkout@v3 [cite: 802]
      - [cite_start]name: Install Ansible [cite: 803]
        [cite_start]run: sudo apt install ansible -y [cite: 804]
      - [cite_start]name: Set up SSH key [cite: 805]
        [cite_start]uses: webfactory/ssh-agent@v0.5.4 [cite: 806]
        [cite_start]with: [cite: 807]
          [cite_start]ssh-private-key: ${{ secrets.SSH_PRIVATE_KEY }} [cite: 808]
      - [cite_start]name: Run Ansible Playbook [cite: 808]
        [cite_start]run: ansible-playbook -i inventory.ini deploy_web.yml [cite: 809]
```

Bash

```
# Commit and push [cite: 810]
```

```
git add . [cite: 812]
git commit -m "Add Ansible CI/CD workflow" [cite: 813]
git push origin main [cite: 814]
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: In GitHub Actions, the workflow runs successfully, deploying the web server. The logs show Ansible output similar to previous experiments.

Ex.No: 37	Creating a Basic Maven Project
Date:	

AIM: To generate a simple Maven project structure and understand the pom.xml file.

PROCEDURE:

Ensure Maven is installed by checking its version.

Navigate to a directory where you want to create the project.

Run the archetype command to generate the project.

Inspect the generated pom.xml and src folders.

Commands:

Bash

```
mvn archetype:generate -DgroupId=com.example -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT:

The console output shows that the project was created from the archetype and concludes with [INFO] BUILD SUCCESS.

Ex.No: 38	Building a Maven Project
Date:	

AIM:

To compile and package a Maven project into a JAR file.

PROCEDURE:

Navigate to the project directory (my-app).

Run the Maven build command.

Check the

target directory for the generated JAR file.

Commands:

Bash

```
cd my-app

mvn clean package
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The command builds a JAR file located at

/path/to/my-app/target/my-app-1.0-SNAPSHOT.jar and the console displays

[INFO] BUILD SUCCESS.

<h1>Hello from Ansible!</h1>.

Ex.No: 39	Adding a Dependency to a Maven Project
Date:	

AIM: To add an external dependency (e.g., JUnit) to a Maven project.

PROCEDURE:

Open the

pom.xml file in the my-app project.

Add the JUnit dependency under the

<dependencies> section.

Save the file and run the dependency resolution command.

Verify that the dependency is downloaded to the local repository.

Commands: Add the following to pom.xml:

XML

```xml
<dependencies>
   <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13.2</version>
      <scope>test</scope>
   </dependency>
</dependencies>
```

Then run:

Bash

```bash
mvn dependency:resolve
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The console output confirms that dependencies were resolved, showing junit:junit:jar:4.13.2:test , and the build is successful.

Ex.No: 40	**Running Tests in a Maven Project**
Date:	

AIM: To execute unit tests in a Maven project.

PROCEDURE:

Ensure the

AppTest.java file exists in src/test/java/com/example.

Run the Maven test command.

Review the test results in the console.

Commands:

Bash

```
mvn test
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The console shows the results of the test execution, such as

[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0 , followed by

[INFO] BUILD SUCCESS

Ex.No: 41	Installing a Maven Artifact Locally
Date:	

AIM: To install a Maven project's artifact into the local repository.

PROCEDURE:

Navigate to the

my-app project directory.

Run the Maven

install command.

Verify the artifact in the local Maven repository (

~/.m2/repository).

Commands:

Bash

```
mvn install
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The output shows the artifact being installed to the local

.m2 repository and concludes with

[INFO] BUILD SUCCESS.

Ex.No: 42	**Initializing a Gradle Project**
Date:	

AIM: To create a basic Gradle project structure.

PROCEDURE:

1. Ensure Gradle is installed.

2. Navigate to a directory for the new project.

3. Run the Gradle

init command.

4. Inspect the generated

build.gradle and settings.gradle files.

Commands:

Bash

```
gradle init --type java-application --dsl groovy --test-framework junit --package com.example --
project-name my-gradle-app
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The console output indicates that the initialization task ran and the result is

BUILD SUCCESSFUL in 5s.

Ex.No: 43	Building a Gradle Project
Date:	

AIM: To compile and package a Gradle project into a JAR file.

PROCEDURE:

1. Navigate to the

my-gradle-app directory.

2. Run the Gradle

build command.

3. Check the

build/libs directory for the generated JAR.

Commands:

Bash

```
cd my-gradle-app
gradle build
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The console shows several tasks being executed and concludes with

BUILD SUCCESSFUL in 2s.

Ex.No: 44	Adding a Dependency to a Gradle Project
Date:	

AIM: To add an external dependency (e.g., JUnit) to a Gradle project.

PROCEDURE:

Open the

build.gradle file in my-gradle-app.

Add the JUnit dependency under the

dependencies block.

Save the file and run the dependency resolution command.

Verify the dependency is downloaded.

Commands: Add to

build.gradle:

Groovy

```
dependencies {
   testImplementation 'junit:junit:4.13.2'
}
```

Then run:

Bash

```
gradle dependencies
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The output shows the dependency tree with

junit:junit:4.13.2 listed and finishes with

BUILD SUCCESSFUL in 1s.

Ex.No: 45	Running Tests in a Gradle Project
Date:	

AIM: To execute unit tests in a Gradle project.

PROCEDURE:

Ensure the

AppTest.java file exists in src/test/java/com/example.

Run the Gradle

test command.

Review the test results in the console.

Commands:

Bash

```
gradle test
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The console shows the test task being executed and concludes with

BUILD SUCCESSFUL in 2s.

Ex.No: 46	Cleaning a Gradle Project
Date:	

AIM: To remove generated build artifacts from a Gradle project.

PROCEDURE:

Navigate to the

my-gradle-app directory.

Run the Gradle

clean command.

Verify that the

build directory is removed.

Commands:

Bash

```
gradle clean
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The output shows the clean task was executed and the result is

BUILD SUCCESSFUL in 1s.

Ex.No: 47	Setting Up a Freestyle Jenkins Job
Date:	

AIM: To create a simple freestyle Jenkins job to run a shell command.

PROCEDURE:

Log in to Jenkins.

Click "New Item," enter a name (e.g., "MyFreestyleJob"), select "Freestyle project," and click OK.

In the "Build" section, add an "Execute shell" step.

Add a simple shell command.

Save and build the job.

Commands: In the "Execute shell" section:

Bash

```
echo "Hello, Jenkins!"
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The Jenkins console output displays

Hello, Jenkins! and concludes with

Finished: SUCCESS

Ex.No: 48	Building a Maven Project in Jenkins
Date:	

AIM: To configure a Jenkins job to build a Maven project.

PROCEDURE:

Create a new freestyle project named "MavenJob".

In the "Source Code Management" section, select "None".

In the "Build" section, add an "Invoke top-level Maven targets" step.

Specify the Maven goal

clean package.

Save and build the job.

Commands: In the Maven build step:

```
clean package
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The console output shows the Maven build process running, ending with

[INFO] BUILD SUCCESS and

Finished: SUCCESS.

Ex.No: 49	Setting Up a Gradle Job in Jenkins
Date:	

AIM: To configure a Jenkins job to build a Gradle project.

PROCEDURE:

Create a new freestyle project named "GradleJob".

In the "Source Code Management" section, select "None".

In the "Build" section, add an "Execute shell" step.

Add the Gradle build command.

Save and build the job.

Commands: In the "Execute shell" section:

Bash

```
gradle build
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The Jenkins console shows the Gradle build tasks running, followed by

BUILD SUCCESSFUL in 2s and

Finished: SUCCESS

Ex.No: 50	Configuring Email Notifications in Jenkins
Date:	

AIM: To set up email notifications for a Jenkins job.

PROCEDURE:

Ensure the Jenkins Email Extension Plugin is installed.

Go to "Manage Jenkins" > "Configure System" and set up SMTP settings.

Create a freestyle project named "EmailJob".

Add a simple shell command.

In the "Post-build Actions" section, add "Editable Email Notification".

Configure the recipient email and save.

Build the job to trigger the email.

Commands: In the "Execute shell" section:

Bash

```
echo "Test email notification"
```

Class Performance (5)	
Record (5)	
Viva (5)	
Total (15)	

RESULT: The console output shows the command being executed and indicates that an email was triggered and sent. The job finishes with a SUCCESS status.