Q1. (15 Marks)
Create a new Git repository locally, add a Java file (HelloWorld.java), commit it, and push it to your GitHub repository.
Expected Output: Code and commit visible on GitHub.

1) Create a new folder and enter it

mkdir hello-world-java

cd hello-world-java

2) Initialize a git repo with main as the branch (If git supports -b):

git init -b main

(Fallback if previous fails)

git init

git checkout -b main

3) (Optional) Configure your Git user if not already set

git config --global user.name "Your Name"

git config --global user.email "you@example.com"

git config –list

4) Create HelloWorld.java

```
cat > HelloWorld.java <<'EOF'
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, world!");
   }
}
EOF
```

5) Verify the file

ls -la

cat HelloWorld.java

6) Stage and commit the file

git add HelloWorld.java

git status

git commit -m "Add HelloWorld.java"

7️⃣ Add GitHub remote repository

Option A: Using HTTPS

git remote add origin https://github.com/<GITHUB_USER>/<REPO>.git

git remote -v

Option B: Using SSH (if you set up SSH keys)

git remote add origin git@github.com:<GITHUB_USER>/<REPO>.git

git remote -v

8️⃣ Push the commit to GitHub

git push -u origin main

- For HTTPS: GitHub may ask for **username and personal access token**
- For SSH: No password needed if SSH key is configured

9️⃣ Verify on GitHub

- **Open your repository in a browser: https://github.com/<GITHUB_USER>/<REPO>**
- **You should see HelloWorld.java and the commit message.**

Q2. (15 Marks)
Create a simple Maven project using the command line and build it using mvn clean package.
Expected Output: **Successful build with .jar file in target folder.**

0) Prerequisite checks (run these first)

 # Check Java and Maven are installed

java -version

javac -version

mvn -version

If any command is missing, install JDK and/or Maven (see optional install notes below).

1) Create a new Maven project (non-interactive)

 # create a new folder (optional) and enter it

mkdir maven-sample

cd maven-sample

 # generate a simple maven project using the quickstart archetype (batch mode)

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

2) Enter the project and inspect files

cd my-app

ls -la

cat pom.xml

cat src/main/java/com/example/App.java

3) Build the project

 # Clean and package (will compile & run tests by default)

mvn clean package

Expected visible result: BUILD SUCCESS and a .jar file created under target/.

4) Confirm the JAR is present and run the app

ls -la target

 # default jar name for this archetype:

ls -la target/my-app-1.0-SNAPSHOT.jar

 # Run the main class from the jar (no special manifest required)

java -cp target/my-app-1.0-SNAPSHOT.jar com.example.App

#You should see the hello output (the archetype App prints a message).

Troubleshooting / shortcuts (if required)

If tests fail and you just want the package anyway:

mvn -DskipTests clean package

Show contents of the jar:

jar tf target/my-app-1.0-SNAPSHOT.jar

Optional: Quick install commands (if Maven not installed)

sudo apt update

sudo apt install maven -y

Single copy-paste block

```
mkdir maven-sample && cd maven-sample

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

cd my-app

mvn clean package

ls -la target

java -cp target/my-app-1.0-SNAPSHOT.jar com.example.App
```

If you need to install tools first: (if required)

```
sudo apt update

sudo apt install git openjdk-21-jdk -y

java -version

git --version
```

Q3. (10 Marks)
Create a Dockerfile for your Maven project and build a Docker image using the command line.
Expected Output: **Docker image visible in docker images.**

Option A — Multi-stage (build inside Docker)

 # 1) go to your project root (where pom.xml is)

cd /path/to/your/maven/project

 # 2) create the Dockerfile (multi-stage) exactly as below

cat > Dockerfile << 'EOF'

 # Stage 1: build with Maven

FROM maven:3.9-jdk-17 AS build

WORKDIR /workspace

COPY pom.xml .

COPY src ./src

RUN mvn -B -DskipTests package

 # Stage 2: runtime

FROM openjdk:17-slim

WORKDIR /app

 # copy the built jar (wildcard in case the jar name includes version)

COPY --from=build /workspace/target/*.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java","-jar","/app/app.jar"]

EOF

 # 3) build the Docker image (tags it as my-maven-app:1.0)

sudo docker build -t my-maven-app:1.0 .

 # 4) list docker images (you should see my-maven-app:1.0 listed)

sudo docker images

 # (optional) 5) run the container and map port 8080 (change port if your app uses another)

sudo docker run --rm -p 8080:8080 my-maven-app:1.0

Option B — Pre-build jar locally, simpler runtime image

 # 1) build jar locally (runs Maven on your host)

cd /path/to/your/maven/project

mvn -B -DskipTests package

 # 2) create a small Dockerfile that copies the built jar

cat > Dockerfile << 'EOF'

FROM openjdk:17-slim

WORKDIR /app

COPY target/*.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java","-jar","/app/app.jar"]

EOF

 # 3) build the image

sudo docker build -t my-maven-app:1.0 .

 # 4) show docker images to confirm

sudo docker images

SET – 2

Q1. (15 Marks)
Create a new Jenkins Freestyle project that pulls code from your GitHub repository and builds it using Maven (clean package).
Expected Output: **Jenkins job builds the project successfully.**

notes

- You have a GitHub repo (public recommended for easiest setup): https://github.com/<GITHUB_USER>/<REPO>.git
- Replace ADMIN_PASSWORD and GITHUB placeholders below with your real values.
- This uses a *shell* build step (mvn clean package) so Jenkins just needs mvn on the machine (we install it).
- I include both CLI and REST commands so you can choose whichever works for you.

Answer

0) Update & install prerequisites (Java, Git, Maven, curl, jq)

sudo apt update

sudo apt install -y maven git curl jq

mvn -version

git –version

Create variables

 # GitHub repo URL

REPO_URL="https://github.com/<GITHUB_USER>/<REPO>.git"

BRANCH="main" **# Or master**

JOB_NAME="my-maven-job" **# Jenkins Freestyle job name**

 # Jenkins admin credentials

ADMIN="admin" **# Your Jenkins username**

ADMIN_PASSWORD="YOUR_PASSWORD" **# Your Jenkins password**

JENKINS_URL=http://localhost:8080

2️⃣ Get Jenkins CSRF crumb

```
CRUMB=$(curl -s --user "$ADMIN:$ADMIN_PASSWORD" "$JENKINS_URL/crumbIssuer/api/xml?xpath=concat(//crumbRequestField,\":\",//crumb)")

echo "Crumb: $CRUMB"
```

3️⃣ Create Jenkins Freestyle job XML

This XML will make Jenkins:

- **Pull from your GitHub repo**
- **Build using mvn clean package**

```
cat > /tmp/${JOB_NAME}-config.xml <<EOF
<?xml version='1.1' encoding='UTF-8'?>
<project>
 <actions/>
 <description>Freestyle job: clone repo and run mvn -B clean package</description>
 <keepDependencies>false</keepDependencies>
 <properties/>
 <scm class="hudson.plugins.git.GitSCM">
  <configVersion>2</configVersion>
  <userRemoteConfigs>
   <hudson.plugins.git.UserRemoteConfig>
    <url>${REPO_URL}</url>
   </hudson.plugins.git.UserRemoteConfig>
  </userRemoteConfigs>
  <branches>
   <hudson.plugins.git.BranchSpec>
    <name>*/${BRANCH}</name>
   </hudson.plugins.git.BranchSpec>
  </branches>
```

```
    <doGenerateSubmoduleConfigurations>false</doGenerateSubmoduleConfigurations>
    <submoduleCfg class="list"/>
    <extensions/>
  </scm>
  <canRoam>true</canRoam>
  <disabled>false</disabled>
  <triggers/>
  <concurrentBuild>false</concurrentBuild>
  <builders>
    <hudson.tasks.Shell>
      <command>mvn -B clean package</command>
    </hudson.tasks.Shell>
  </builders>
  <publishers/>
  <buildWrappers/>
</project>
EOF
```

4️⃣ Create the job via Jenkins REST API

```
curl -s --user "$ADMIN:$ADMIN_PASSWORD" -H "$CRUMB" \
    -X POST "$JENKINS_URL/createItem?name=${JOB_NAME}" \
    --data-binary @/tmp/${JOB_NAME}-config.xml -H "Content-Type: application/xml"
echo "Job '${JOB_NAME}' created."
```

5️⃣ Trigger the build

```
curl -s -X POST --user "$ADMIN:$ADMIN_PASSWORD" -H "$CRUMB" \
    "$JENKINS_URL/job/${JOB_NAME}/build"
echo "Build triggered..."
```

6️⃣ Monitor build status (poll console)

```
while true; do
  sleep 5
  curl -s --user "$ADMIN:$ADMIN_PASSWORD" "$JENKINS_URL/job/${JOB_NAME}/lastBuild/consoleText" | tail -n 20
  STATUS=$(curl -s --user "$ADMIN:$ADMIN_PASSWORD" "$JENKINS_URL/job/${JOB_NAME}/lastBuild/api/json" | jq -r '.result')
  if [ "$STATUS" != "null" ]; then
    echo "Build finished with result: $STATUS"
    break
  fi
done
```

7️⃣ Verify artifact (JAR) in Jenkins workspace

```
sudo ls -la /var/lib/jenkins/workspace/${JOB_NAME}/target/
```

Q2. (15 Marks)
Configure Jenkins to automatically trigger a build whenever new code is pushed to GitHub using a webhook.
Expected Output: **Jenkins job triggers automatically on every commit.**

0️⃣ Install GitHub plugin (if not installed)

sudo jenkins-plugin-cli --plugins github github-branch-source

sudo systemctl restart Jenkins

1️⃣ Create Jenkins API token for your admin user (needed for webhook)

 # log into Jenkins web UI -> Manage Jenkins -> Manage Users -> Click your user -> Configure -> API Token -> Generate

 # Copy the token and save it as GITHUB_JENKINS_TOKEN

GITHUB_JENKINS_TOKEN="YOUR_API_TOKEN_HERE"

ADMIN_USER="admin" **# your Jenkins username**

2️⃣ Enable GitHub hook trigger in the Jenkins job

Option A: Using Jenkins CLI (recommended)

 # Download Jenkins CLI if you haven't already

wget -q http://localhost:8080/jnlpJars/jenkins-cli.jar -O ~/jenkins-cli.jar

 # Update job config to enable GitHub webhook trigger

JOB_NAME="my-maven-job"

curl -s --user "$ADMIN_USER:$GITHUB_JENKINS_TOKEN" "http://localhost:8080/job/${JOB_NAME}/config.xml" -o /tmp/job-config.xml

 # Insert GitHub hook trigger XML before </builders>

sed -i '/<\/builders>/i \

  <triggers>\

    <com.cloudbees.jenkins.GitHubPushTrigger>\

```
    <spec></spec>\
  </com.cloudbees.jenkins.GitHubPushTrigger>\
</triggers>' /tmp/job-config.xml

# Post updated config back to Jenkins
curl -s --user "$ADMIN_USER:$GITHUB_JENKINS_TOKEN" -X POST \
   -H "Content-Type: application/xml" \
   --data-binary @/tmp/job-config.xml \
   http://localhost:8080/job/${JOB_NAME}/config.xml
```

4️⃣ Test the webhook trigger

```
# Make a commit & push to GitHub
cd <your-local-repo>
echo "// test commit" >> test.txt
git add test.txt
git commit -m "Test webhook trigger"
git push origin main
```

Step 4a: Monitor Jenkins

```
# Poll the last build result
ADMIN_USER="admin"
GITHUB_JENKINS_TOKEN="YOUR_API_TOKEN"
JOB_NAME="my-maven-job"
curl -s --user "$ADMIN_USER:$GITHUB_JENKINS_TOKEN" \
   "http://localhost:8080/job/${JOB_NAME}/lastBuild/api/json" | jq '.result'
```

- **Should return "SUCCESS" after Maven build completes automatically.**

- # 5️⃣ Optional: Verify build in Jenkins workspace

sudo ls -la /var/lib/jenkins/workspace/${JOB_NAME}/target/

Your .jar artifact should be updated with the latest commit.

Q3. (10 Marks)
Run a container from a Docker image and verify it using docker ps and container logs.
Expected Output: **Container running successfully with correct output.**

1️⃣ Pull a Docker image

docker pull hello-world

2️⃣ Run a container from the image

docker run --name hello-container hello-world

3️⃣ Verify running containers

docker ps

or

docker ps -a

4️⃣ View container logs

docker logs hello-container

5️⃣ Optional: Clean up

docker rm hello-container **# Remove the container**

docker rmi hello-world **# Remove the image**

SET 3

Q1. (15 Marks)
Clone your GitHub repository, modify one Java file, commit and push the changes.
Expected Output: **Updated code with new commit visible on GitHub.**

1️⃣ Clone your GitHub repository

 # Replace <GITHUB_USER> and <REPO> with your GitHub username and repository name

git clone https://github.com/<GITHUB_USER>/<REPO>.git

cd <REPO>

2️⃣ Verify the files

ls -la

3️⃣ Modify a Java file

nano HelloWorld.java

Add a line or change System.out.println("Hello, world!"); to something else, e.g.:
System.out.println("Hello from updated code!");

Save and exit (Ctrl+O → Enter → Ctrl+X in nano).

4️⃣ Stage the changes

git status

git add HelloWorld.java

5️⃣ Commit the changes

git commit -m "Updated HelloWorld.java with new message"

6️⃣ Push the changes to GitHub

Option A: HTTPS (with username + Personal Access Token if required)

git push origin main

Replace main with your branch name if different.

Option B: SSH (if you configured SSH keys)

git push origin main

No password prompt if SSH key is added to GitHub.

7️⃣ Verify on GitHub

- **Go to your GitHub repository in a browser.**
- **You should see the updated HelloWorld.java and your new commit.**

Q2. (15 Marks)
Create a Jenkins pipeline with stages: *Checkout → Build (→ Test*. Execute the pipeline and show console output.
***Expected Output:* Pipeline stages execute successfully**

1. **Download Jenkins CLI (if not done already):**

wget -q http://localhost:8080/jnlpJars/jenkins-cli.jar -O ~/jenkins-cli.jar

2. **Create a Jenkinsfile locally:**

```
cat > Jenkinsfile <<EOF
pipeline {
  agent any
  stages {
    stage('Checkout') {
      steps {
        git branch: 'main', url: 'https://github.com/<GITHUB_USER>/<REPO>.git'
      }
    }
    stage('Build') {
      steps {
        sh 'mvn clean package'
      }
    }
    stage('Test') {
      steps {
        sh 'mvn test'
      }
    }
  }
  post {
    always {
```

```
        archiveArtifacts artifacts: '**/target/*.jar', allowEmptyArchive: true
    }
  }
}
EOF
```

Use Jenkins REST API or CLI to create a pipeline job pointing to this Jenkinsfile.

- For simplicity, using **Web UI is easier**.

2️⃣ Execute the Pipeline

1. Go to the Pipeline job in Jenkins (e.g., my-pipeline-job)
2. Click **Build Now**
3. Jenkins will start executing stages: **Checkout → Build → Test**

4️⃣ Optional: Verify artifact

sudo ls -la /var/lib/jenkins/workspace/my-pipeline-job/target/

Q3. (10 Marks)
Build a Docker image for your Java project and verify its creation.
Expected Output: **Image listed in Docker images list.**

1. **Navigate to your project folder first:**

cd /path/to/your/java/project

1️⃣ **Create a Dockerfile**

cat > Dockerfile <<EOF

Use OpenJDK base image

FROM openjdk:17-jdk-slim

Set working directory inside container

WORKDIR /app

Copy the project files into the container

COPY . .

Build the project (Maven)

RUN apt-get update && apt-get install -y maven \

&& mvn clean package

Set the entrypoint (run your JAR)

CMD ["java", "-jar", "target/your-app.jar"]

EOF

- **Replace your-app.jar with the actual name of your JAR file generated by Maven.**

2️⃣ Build the Docker image

docker build -t my-java-app .

3️⃣ Verify the Docker image

docker images

4️⃣ Optional: Run the Docker image to test

docker run --name my-java-container my-java-app

SET 1

Q2. (15 Marks) Create a simple Maven project using the command line and build it using mvn clean package. Expected Output: Successful build with .jar file in target folder.

Q2 — Step-by-Step Commands

1️⃣ Create a new folder for your Maven project

mkdir my-maven-app

cd my-maven-app

2️⃣ Generate a Maven project

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

- **This generates a basic Maven Java project with the following structure:**

my-app/

├── pom.xml

└── src/

└── main/java/com/example/App.java

3️⃣ Enter the project folder

cd my-app

4️⃣ Verify project structure

ls -la

tree .

- **You should see pom.xml and src/ folders.**
- **Inside src/main/java/com/example/, there is App.java.**

5️⃣ Modify Java file (optional)

nano src/main/java/com/example/App.java

- **Example modification:**

```
package com.example;


public class App {
   public static void main(String[] args) {
      System.out.println("Hello, Maven!");
   }
}
```

- **Save file (Ctrl+O → Enter → Ctrl+X).**

6️⃣ Build the Maven project

mvn clean package

- **clean → removes old builds**
- **package → compiles code, runs tests, creates .jar file**

✅ **If successful, you'll see BUILD SUCCESS in the terminal.**

7️⃣ Verify the .jar file

ls -la target/

- **You should see something like:**

my-app-1.0-SNAPSHOT.jar

my-app-1.0-SNAPSHOT.jar.original

- **This is the packaged Java application.**

8️⃣ Run the jar file (optional)

java -jar target/my-app-1.0-SNAPSHOT.jar

- **Output should be:**

Hello, Maven!

Q3. (10 Marks)
Create a Dockerfile for your Maven project and build a Docker image using the command line.
Expected Output: **Docker image visible in docker images.**

1️⃣ Navigate to your Maven project root

cd ~/my-maven-app/my-app

- **Make sure you see pom.xml and src/ in this folder:**

ls -la

2️⃣ Create a Dockerfile

cat > Dockerfile <<EOF

Use OpenJDK base image

FROM openjdk:17-jdk-slim

Set working directory inside container

WORKDIR /app

Copy project files into container

COPY . .

Install Maven and build project

RUN apt-get update && apt-get install -y maven \

  && mvn clean package

Set entrypoint to run your JAR

CMD ["java", "-jar", "target/my-app-1.0-SNAPSHOT.jar"]

EOF

- **Important: Replace my-app-1.0-SNAPSHOT.jar with the actual name of your JAR file in target/.**

3️⃣ Verify Dockerfile

cat Dockerfile

- **You should see all lines of the Dockerfile.**

4️⃣ Build the Docker image

docker build -t my-maven-app-image .

- **-t my-maven-app-image → sets the image name**
- **. → uses current folder as Docker build context**
- **Docker will install Maven, build your project, and create the image.**

5️⃣ Verify the Docker image

docker images

- **You should see my-maven-app-image in the list:**

REPOSITORY TAG IMAGE ID CREATED SIZE

my-maven-app-image latest abc123def456 2 minutes ago 500MB

6️⃣ (Optional) Run the Docker image

docker run --name my-maven-container my-maven-app-image

- **You should see your Java program output, e.g., Hello, Maven!**
- **Check container status:**

docker ps -a

- **View logs:**

docker logs my-maven-container

7️⃣ Optional Cleanup

docker rm my-maven-container **# Remove container**

docker rmi my-maven-app-image **# Remove image**

Set 2

Q1. (15 Marks)
Create a new Jenkins Freestyle project that pulls code from your GitHub repository and builds it using Maven (clean package).
Expected Output: **Jenkins job builds the project successfully.**

Step 0️⃣ : Terminal — Install Maven on Ubuntu

sudo apt update

sudo apt install maven -y

mvn --version

- ✅ **Ensures Maven is installed on Jenkins server.**
- **Terminal step.**

Step 1️⃣ : Web UI — Create a new Freestyle project

1. **Open browser: http://localhost:8080**
2. **Log in with your Jenkins admin account**
3. **Click New Item**
4. **Enter Job name: my-maven-job**
5. **Select Freestyle project → Click OK**

- **Web UI step.**

Step 2️⃣ : Web UI — Configure Source Code Management (Git)

1. **Scroll to Source Code Management → Git**
2. **Repository URL:**

https://github.com/<GITHUB_USER>/<REPO>.git

- **Replace <GITHUB_USER> and <REPO> with your GitHub repo**

3. **Add credentials if the repo is private (HTTPS token or SSH key)**
4. **Branch: main (or your branch)**

- **Web UI step.**

Step 3️⃣ : Web UI — Add Build Step (Maven)

1. **Scroll to Build → Add build step → Invoke top-level Maven targets**
2. **In Goals, type:**

clean package

- **This will clean and build the Maven project.**
- **Web UI step.**

Step 4️⃣ : Web UI — Save the Job

- **Click Save at the bottom.**
- **Web UI step.**

Step 5️⃣ : Web UI — Build the Job

1. **Click Build Now on the left sidebar**
2. **Jenkins pulls code from GitHub, runs mvn clean package**

- **Web UI step.**

Step 6️⃣ : Web UI — View Console Output

1. **Click the Build number (e.g., #1)**
2. **Click Console Output**
3. **You should see at the end:**

[INFO] BUILD SUCCESS

Finished: SUCCESS

- **Confirms the build succeeded.**
- **Web UI step.**

Step 7️⃣ : Terminal — Verify Artifacts

sudo ls -la /var/lib/jenkins/workspace/my-maven-job/target/

- **.jar file should be present in the target/ folder.**
- **Terminal step.**

Q2. (15 Marks)
Configure Jenkins to automatically trigger a build whenever new code is pushed to GitHub using a webhook.
Expected Output: **Jenkins job triggers automatically on every commit.**

Step 0️⃣ : Terminal — Install GitHub Plugin (if not installed)

sudo jenkins-plugin-cli --plugins github github-branch-source

sudo systemctl restart jenkins

- **Ensures GitHub plugin is installed in Jenkins.**
- **Terminal step.**

Step 1️⃣ : Web UI — Generate Jenkins API Token

1. **Go to http://localhost:8080 → Manage Jenkins → Manage Users**
2. **Click your user → Configure → API Token → Generate**
3. **Copy the token**
4. **Save it as an environment variable in terminal:**

export ADMIN_USER="your-jenkins-username"

export GITHUB_JENKINS_TOKEN="your-generated-api-token"

- **Terminal step (to store credentials for CLI if needed)**
- **Generating token itself is Web UI step.**

Step 2️⃣ : Web UI — Enable GitHub Hook Trigger in Job

Option A: Web UI (simpler)

1. **Go to your job → Configure**
2. **Scroll to Build Triggers**
3. **Check GitHub hook trigger for GITScm polling**
4. **Click Save**

- **Web UI step**

Option B: Jenkins CLI (terminal)

```
# Download Jenkins CLI if you haven't already
wget -q http://localhost:8080/jnlpJars/jenkins-cli.jar -O ~/jenkins-cli.jar

# Fetch current job config
JOB_NAME="my-maven-job"
curl -s --user "$ADMIN_USER:$GITHUB_JENKINS_TOKEN" \
 "http://localhost:8080/job/${JOB_NAME}/config.xml" -o /tmp/job-config.xml

# Insert GitHub webhook trigger
sed -i '/<\/builders>/i \
 <triggers>\
  <com.cloudbees.jenkins.GitHubPushTrigger>\
   <spec></spec>\
  </com.cloudbees.jenkins.GitHubPushTrigger>\
 </triggers>' /tmp/job-config.xml

# Post updated config back to Jenkins
curl -s --user "$ADMIN_USER:$GITHUB_JENKINS_TOKEN" -X POST \
 -H "Content-Type: application/xml" \
 --data-binary @/tmp/job-config.xml \
 "http://localhost:8080/job/${JOB_NAME}/config.xml"
```

- **Terminal step (alternative to Web UI)**

Step 3️⃣: Web UI — Configure GitHub Webhook

1. **Go to your GitHub repository → Settings → Webhooks → Add webhook**
2. **Payload URL:**

http://YOUR_SERVER_IP:8080/github-webhook/

- **Replace YOUR_SERVER_IP with your Jenkins server IP**

3. **Content type: application/json**
4. **Events: Just the push event**

5. **Click Add webhook**

- **Web UI step**

Step 4️⃣ : Terminal — Test Webhook Trigger

1. **Make a change in your local repo:**

```
cd ~/my-maven-app/my-app
echo "// test commit" >> test.txt
git add test.txt
git commit -m "Test webhook trigger"
git push origin main
```

- **Terminal step**

2. **Jenkins should automatically trigger the build.**

Step 5️⃣ : Terminal — Check Last Build Result

```
curl -s --user "$ADMIN_USER:$GITHUB_JENKINS_TOKEN" \
 "http://localhost:8080/job/${JOB_NAME}/lastBuild/api/json" | jq '.result'
```

- **Should return "SUCCESS" after Maven build completes**
- **Terminal step**

Step 6️⃣ : Terminal — Optional Verify Workspace Artifacts

```
sudo ls -la /var/lib/jenkins/workspace/${JOB_NAME}/target/
```

- **.jar should be updated with latest commit**
- **Terminal step**

Q3. (10 Marks)
Run a container from a Docker image and verify it using docker ps and container logs.
***Expected Output*: Container running successfully with correct output.**

1️⃣ List available Docker images

docker images

- **You should see your image, e.g., my-maven-app-image.**
- **Terminal step**

2️⃣ Run a container from the Docker image

docker run --name my-maven-container -d my-maven-app-image

- **--name → gives the container a name (my-maven-container)**
- **-d → runs the container in detached mode (in background)**
- **Terminal step**

3️⃣ Verify running containers

docker ps

- **You should see the container running:**

CONTAINER ID IMAGE STATUS PORTS NAMES

abc123def456 my-maven-app-image Up 5 seconds my-maven-container

- **Terminal step**

4️⃣ Check container logs

docker logs my-maven-container

- **Output should be your Java program's output, e.g.,**

Hello, Maven!

- **Confirms container ran successfully**
- **Terminal step**

5️⃣ Optional: Enter container (interactive)

docker exec -it my-maven-container /bin/bash

- **Explore container files, e.g., check target/ folder:**

ls -la /app/target/

- **Terminal step**

6️⃣ Optional: Stop and remove container

docker stop my-maven-container

docker rm my-maven-container

- **Terminal step**

7️⃣ Optional: Remove Docker image

docker rmi my-maven-app-image

- **Terminal step**

SET 3

Q1. (15 Marks)
Clone your GitHub repository, modify one Java file, commit and push the changes.
Expected Output: **Updated code with new commit visible on GitHub.**

1️⃣ Open terminal and navigate to workspace

cd ~

mkdir github-projects

cd github-projects

- **Creates a folder to store your cloned repo.**
- **Terminal step**

2️⃣ Clone the GitHub repository

git clone https://github.com/<GITHUB_USER>/<REPO>.git

cd <REPO>

- **Replace <GITHUB_USER> and <REPO> with your GitHub username and repository name**
- **Terminal step**

3️⃣ Verify cloned files

ls -la

- **You should see pom.xml, src/, and existing Java files**
- **Terminal step**

4️⃣ Modify a Java file

nano src/main/java/com/example/App.java

- **Add a simple line, e.g.,**

System.out.println("Updated code for Q1!");

- **Save file (Ctrl+O → Enter → Ctrl+X)**
- **Terminal step**

5️⃣ Check git status

git status

- **Shows the modified files ready to commit**
- **Terminal step**

6️⃣ Stage the changes

git add src/main/java/com/example/App.java

- **Terminal step**

7️⃣ Commit the changes

git commit -m "Update App.java for Q1"

- **Terminal step**

8️⃣ Push changes to GitHub

git push origin main

- **For HTTPS private repo: GitHub may ask for username + personal access token**
- **For SSH: no password needed if SSH key is configured**
- **Terminal step**

9️⃣ Verify changes on GitHub

1. **Open your repository in a browser:**

https://github.com/<GITHUB_USER>/<REPO>

2. **You should see the modified App.java and the new commit**

- **Web UI step**

Q2. (15 Marks)
Create a Jenkins pipeline with stages: *Checkout → Build (→ Test*. Execute the pipeline and show console output.
***Expected Output:* Pipeline stages execute successfully**

Step 0️⃣ : Terminal — Verify Maven

mvn --version

- **Ensures Maven is installed for the pipeline build.**
- **Terminal step**

Step 1️⃣ : Terminal — Create Jenkinsfile

1. **Navigate to your project folder (local Git repo):**

cd ~/github-projects/<REPO>

2. **Create a Jenkinsfile with pipeline stages:**

```
cat > Jenkinsfile <<EOF
pipeline {
  agent any
  stages {
    stage('Checkout') {
      steps {
        git branch: 'main', url: 'https://github.com/<GITHUB_USER>/<REPO>.git'
      }
    }
    stage('Build') {
      steps {
        sh 'mvn clean package'
      }
    }
    stage('Test') {
      steps {
        sh 'mvn test'
```

```
        }
      }
    }
    post {
      always {
        archiveArtifacts artifacts: '**/target/*.jar', allowEmptyArchive: true
      }
    }
}
EOF
```

- **Replace <GITHUB_USER> and <REPO> with your GitHub repo info**
- **Terminal step**

Step 2️⃣: Commit Jenkinsfile to Git

```
git add Jenkinsfile
git commit -m "Add Jenkinsfile for pipeline"
git push origin main
```

- **Terminal step**

Step 3️⃣: Web UI — Create Pipeline Job

1. **Go to http://localhost:8080 → New Item**
2. **Enter job name: my-pipeline-job**
3. **Select Pipeline → Click OK**

- **Web UI step**

Step 4️⃣: Web UI — Configure Pipeline

1. **Scroll to Pipeline section**
2. **Definition: Pipeline script from SCM**
3. **SCM: Git**
4. **Repository URL:**

https://github.com/<GITHUB_USER>/<REPO>.git

5. **Branch: main**
6. **Script Path: Jenkinsfile**
7. **Click Save**

- **Web UI step**

Step 5️⃣: Web UI — Build Pipeline

1. **Click Build Now**
2. **Jenkins will execute the stages:**
    - **Checkout → Build → Test**

- **Web UI step**

Step 6️⃣: Web UI — View Console Output

1. **Click the Build number (e.g., #1)**
2. **Click Console Output**
3. **You should see output of all stages and at the end:**

[INFO] BUILD SUCCESS

Finished: SUCCESS

- **Confirms pipeline executed successfully**
- **Web UI step**

Step 7️⃣: Terminal — Verify Artifacts

sudo ls -la /var/lib/jenkins/workspace/my-pipeline-job/target/

- **.jar file should be present**
- **Terminal step**

Q3. (10 Marks)
Build a Docker image for your Java project and verify its creation.
Expected Output: **Image listed in Docker images list.**

1️⃣ Navigate to your project folder

cd ~/my-maven-app

ls -la

- **Make sure you see pom.xml and src/**
- **Terminal step**

2️⃣ Create a Dockerfile

cat > Dockerfile <<EOF

Use OpenJDK base image

FROM openjdk:17-jdk-slim

Set working directory

WORKDIR /app

Copy project files

COPY . .

Install Maven and build project

RUN apt-get update && apt-get install -y maven \

&& mvn clean package

Run the JAR

CMD ["java", "-jar", "target/my-app-1.0-SNAPSHOT.jar"]

EOF

- **Replace my-app-1.0-SNAPSHOT.jar with your JAR file name in target/**
- **Terminal step**

3️⃣ Verify Dockerfile

cat Dockerfile

- **Ensure all lines are correct**
- **Terminal step**

4️⃣ Build Docker image

docker build -t my-maven-app-image .

- **-t my-maven-app-image → sets image name**
- **. → build context is current folder**
- **Terminal step**

5️⃣ Verify Docker image

docker images

- **You should see:**

REPOSITORY TAG IMAGE ID CREATED SIZE

my-maven-app-image latest abc123def456 1 min ago 500MB

- **Terminal step**

6️⃣ Optional: Run Docker container to verify

docker run --name my-maven-container my-maven-app-image

docker ps -a

docker logs my-maven-container

- **Confirms container runs and outputs your Java program results**
- **Terminal step**

7️⃣ Optional: Cleanup

docker rm my-maven-container

docker rmi my-maven-app-image

- **Terminal step**