

## **PROGRAM NO: 1**

### **1. Build and Run a Java Application with Maven, Migrate the Same Application to Gradle.**

- **Introduction to Maven and Gradle: Overview of Build Automation Tools, Key Differences between Maven and Gradle, Installation and Setup.**

#### **Introduction to Maven and Gradle**

- **Overview of Build Automation Tools**

Build automation tools help developers streamline the process of building, testing, and deploying software projects. They take care of repetitive tasks like compiling code, managing dependencies, and packaging applications, which make development more efficient and error-free.

Two popular tools in the Java ecosystem are **Maven** and **Gradle**. Both are great for managing project builds and dependencies, but they have some key differences.

#### **Maven**

- **What is Maven?** Maven is a build automation tool primarily used for Java projects. It uses an XML configuration file called pom.xml (Project Object Model) to define project settings, dependencies, and build steps.
- **Main Features:**
  - Predefined project structure and lifecycle phases.
  - Automatic dependency management through Maven Central.
  - Wide range of plugins for things like testing and deployment.
  - Supports complex projects with multiple modules.

#### **Gradle**

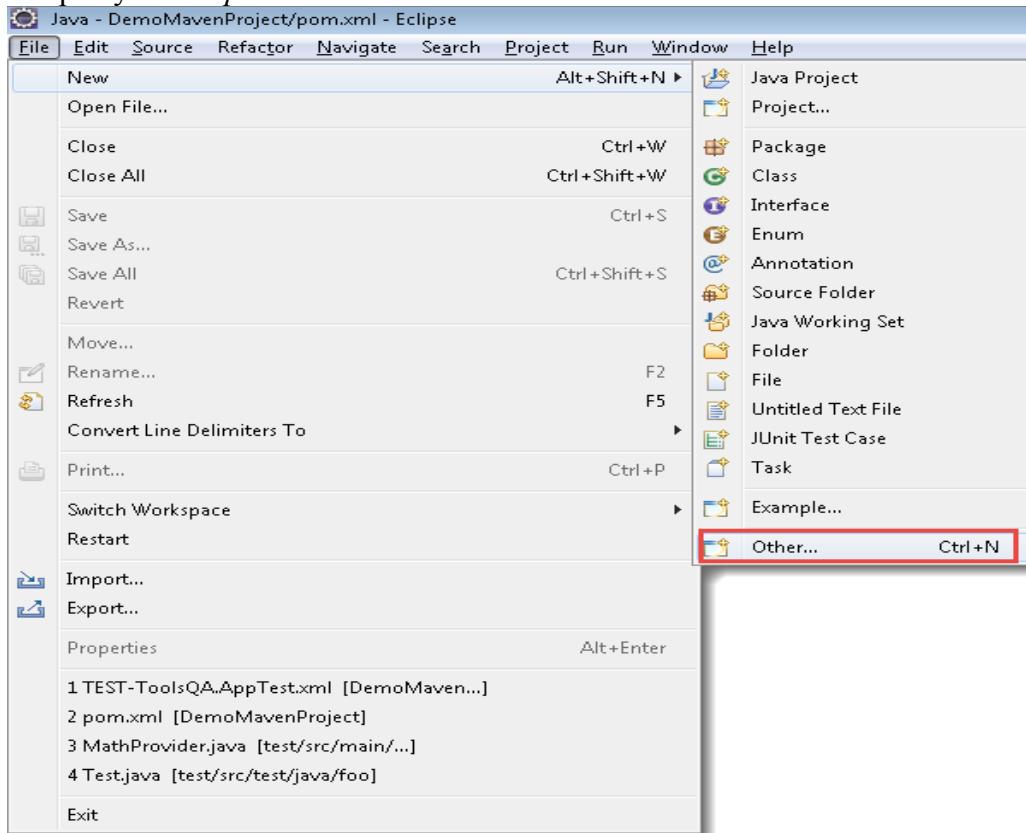
- **What is Gradle?** Gradle is a more modern and versatile build tool that supports multiple programming languages, including Java, Groovy, and Kotlin. It uses a domain-specific language (DSL) for build scripts, written in Groovy or Kotlin.
- **Main Features:**
  - Faster builds thanks to task caching and incremental builds.
  - Flexible and customizable build scripts.
  - Works with Maven repositories for dependency management.
  - Excellent support for multi-module and cross-language projects.
  - Integrates easily with CI/CD pipelines.

### Key Differences Between Maven and Gradle

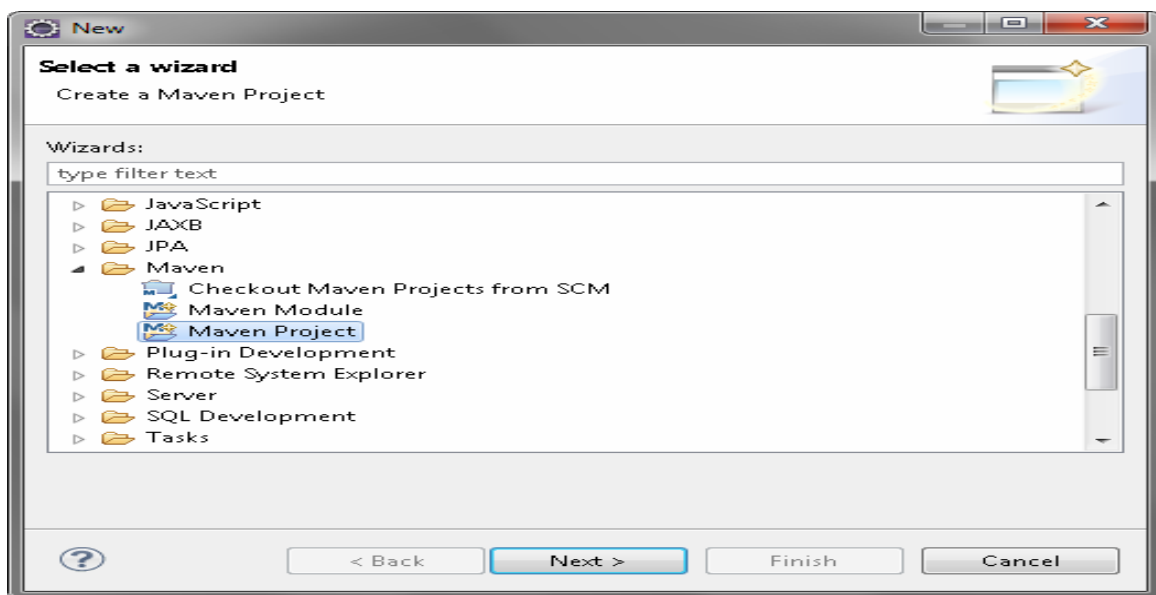
Aspect	Maven	Gradle
Configuration	XML (pom.xml)	Groovy or Kotlin DSL
Performance	Slower	Faster due to caching
Flexibility	Less flexible	Highly customizable
Learning Curve	Easier to pick up	Slightly steeper
Script Size	Verbose	More concise
Dependency Management	Uses Maven Central	Compatible with Maven too
Plugin Support	Large ecosystem	Extensible and versatile

## Create a New Maven Project in Eclipse

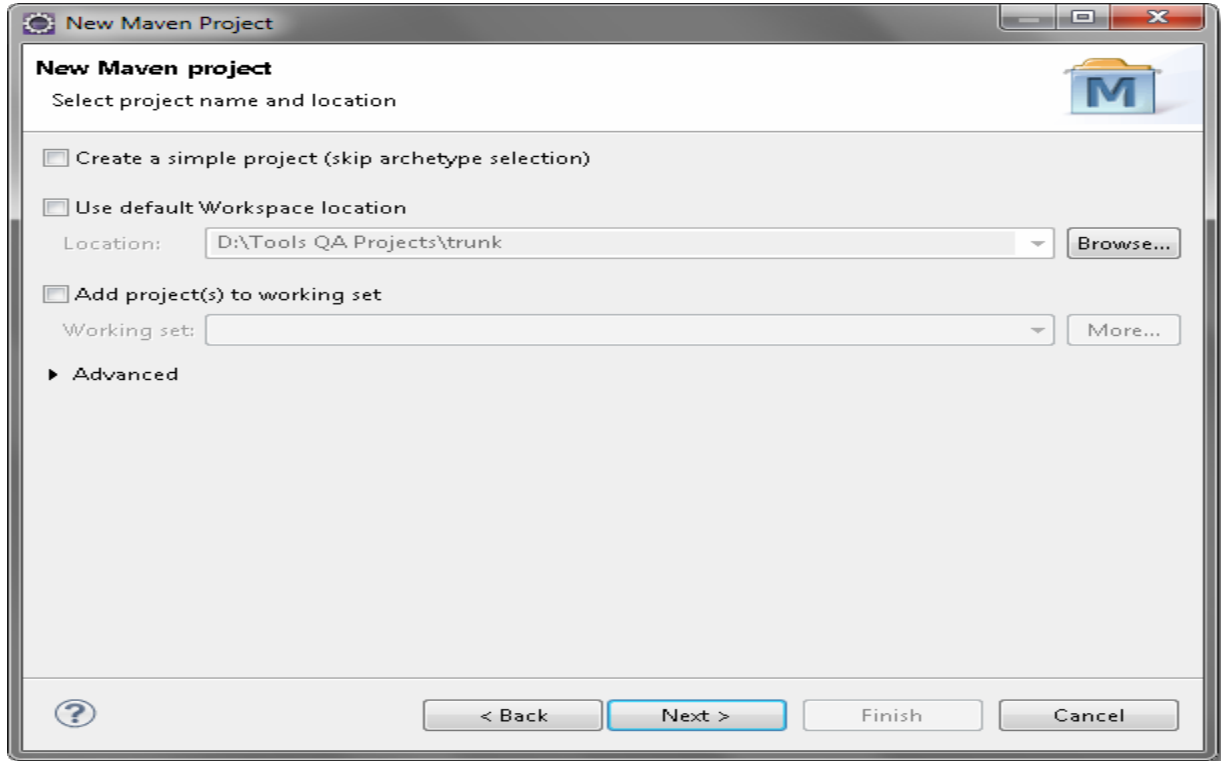
1. Open your *eclipse* and **Go to File > New > Others.**



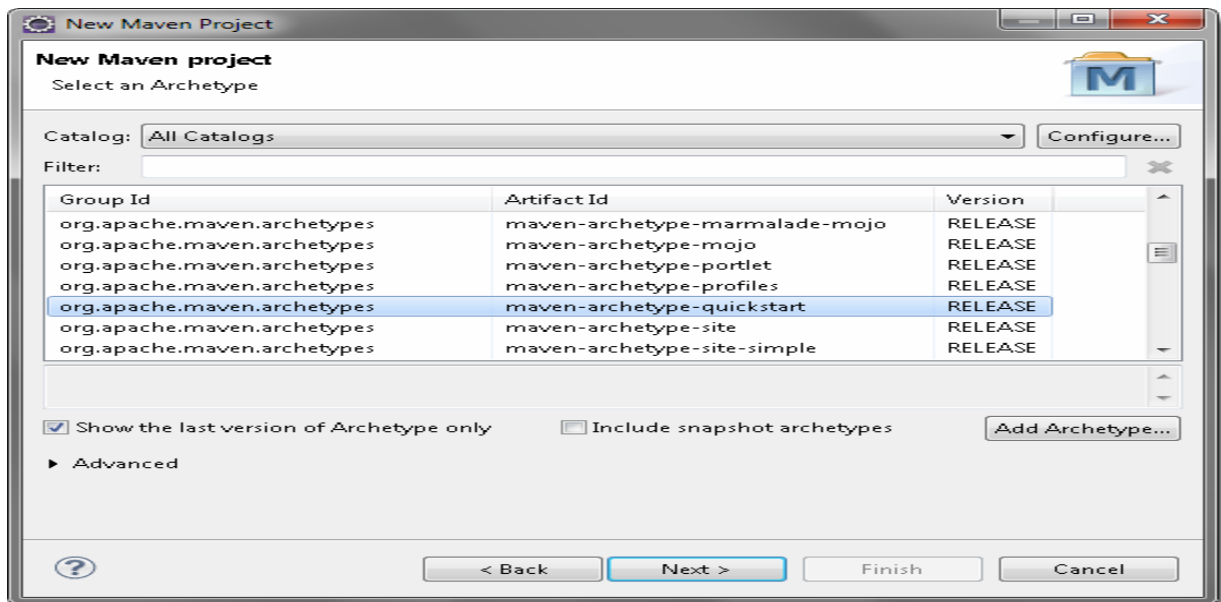
2. Select **Maven Project** and click on **Next**.



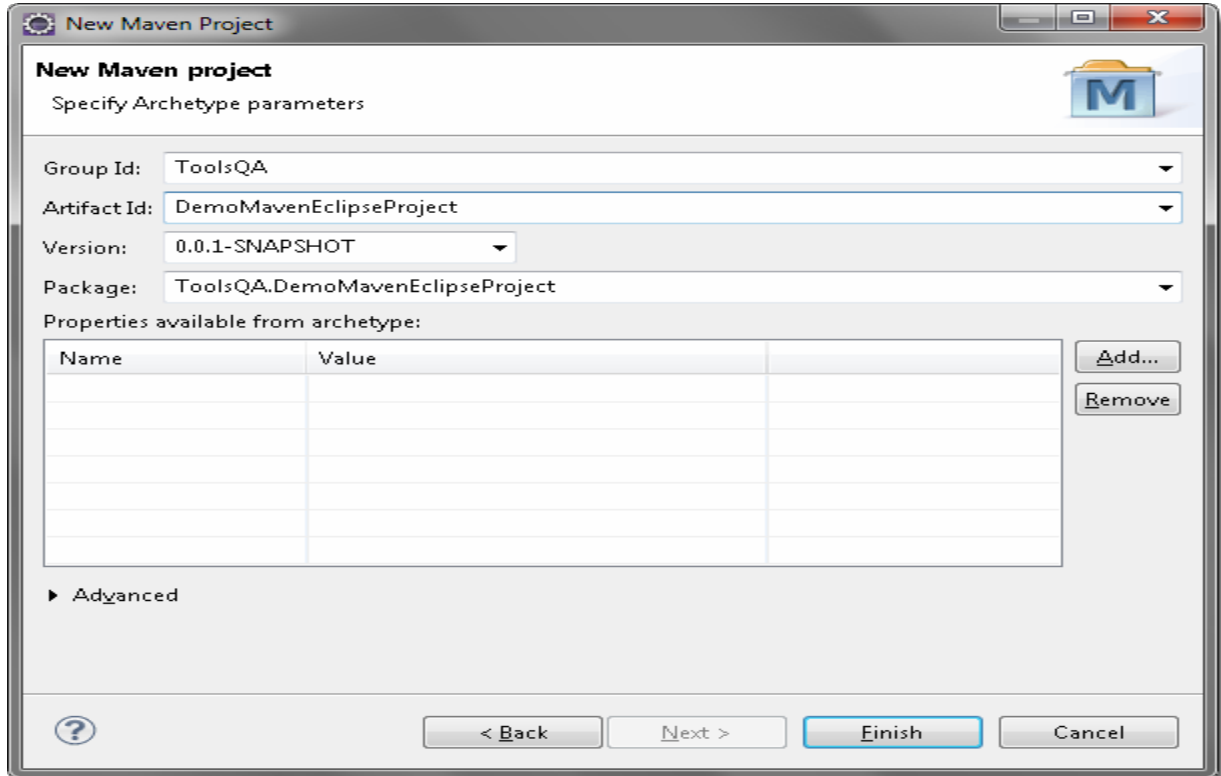
3. Un-check the 'Use default Workspace location' and with the help of the **Browse** button choose your *workspace* where you would like to set up your *Maven project*.



4. Select the *archetype*, for now just select the '*maven-archetype-quickstart*' and click on *Next*.

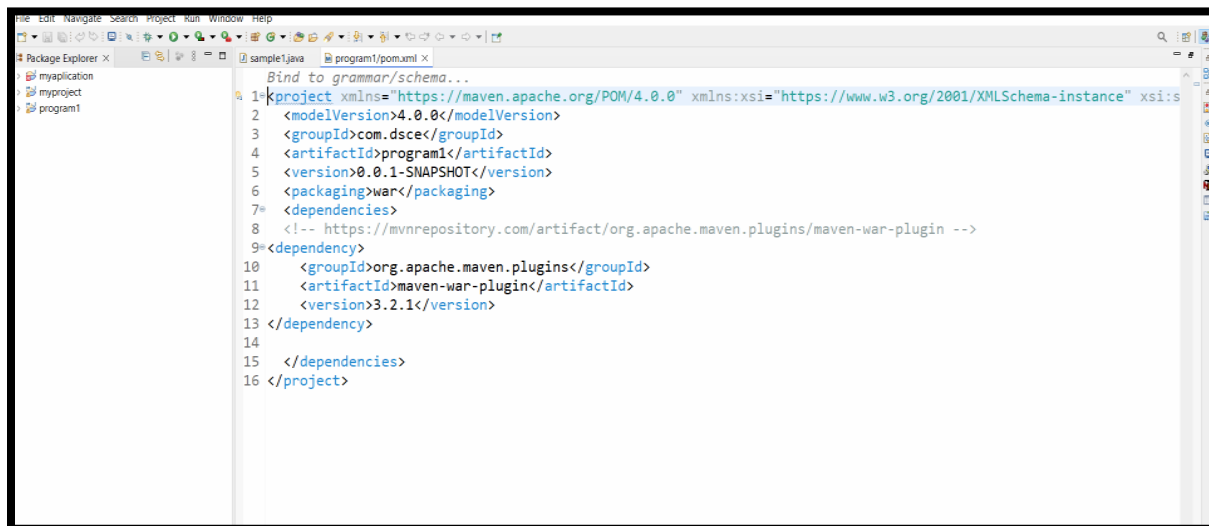


5. Specify the **Group Id** & **Artifact Id** and click on **Finish**.

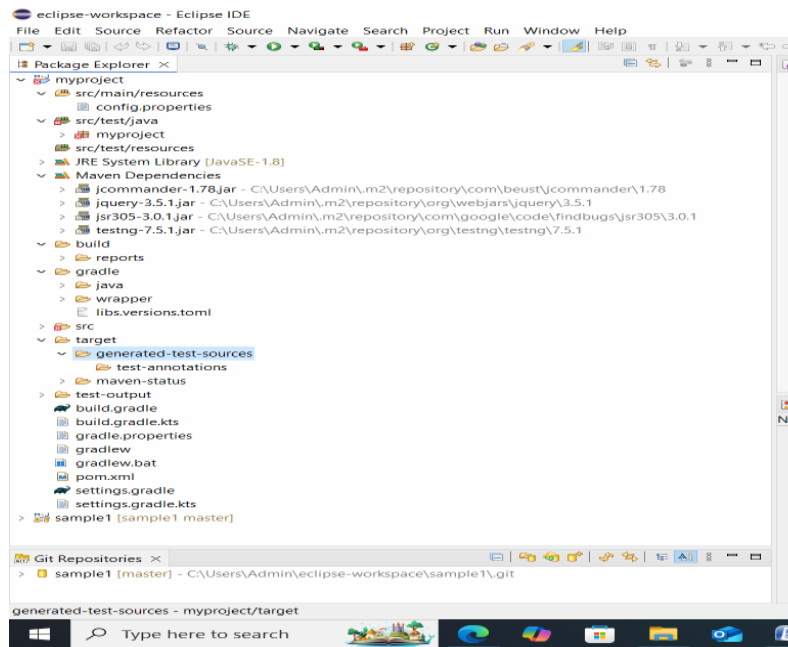


**Note:** Here the 'artifactId' is your project name.

6) Go to the project location to see the newly created maven project. Now open the *pom.xml* file, which resides in the project folder. By default the POM is generated like this:



7) Look at the default folder structure of the Maven project.



## Root Directory (Project Directory):

- This is the top-level directory of your project.
- It contains the `pom.xml` file, which is the Project Object Model, the heart of Maven's configuration.

## Key Subdirectories:

### 1. `src/`:

- This directory contains all the source code for your project.
- **`src/main/java/`:**
  - Contains the Java source code for your application.
  - The package structure of your Java classes mirrors the directory structure within this folder.
- **`src/main/resources/`:**
  - Contains resources used by your application, such as configuration files, property files, and images.
- **`src/main/webapp/`** (for web applications):
  - Contains web related files, such as HTML, CSS, JavaScript, and JSP files.
  - Often contains a `WEB-INF` folder.
- **`src/test/java/`:**
  - Contains the Java source code for your unit tests.
  - The package structure mirrors the `src/main/java/` directory.
- **`src/test/resources/`:**
  - Contains resources used by your unit tests.

### 2. `target/`:

- This directory is created by Maven during the build process.

- It contains the compiled classes, generated JAR/WAR files, and other build artifacts.
  - Contents of this folder are usually deleted when the clean lifecycle phase of maven is executed.
3. **pom.xml:**
- This is the Project Object Model file.
  - It contains the configuration for your Maven project, including dependencies, build settings, and plugins.
- 

**The student's record should include the following contents and snapshots pasted.**

### **MyApp.java**

```
import java.util.ResourceBundle;
public class MyApp
{
    public int userlogin(string inuser,string inpwd)
    {
        ResourceBundle rb= ResourceBundle.getBundle("config");
        String username=rb.getString("username");
        String password=rb.getString("password");
        If(inuser.equals(username)&& inpassword(password))
            return 1;
        else
            return 0;
    }
}
```

### **Config.properties**

```
username=abc
password=abc@1234
```

### **MyAppTest.java code**

```
package myproject;
import org.testng.Assert;
import org.testng.annotations.Test;

import com.myproject.app;

public class apptest {
    @Test
    public void testlogin1()
    {
        app myapp=new app();
    }
}
```

```

        Assert.assertEquals(0,myapp.userlogin("abc","abc1234"));
    }
    @Test
    public void testlogin2()
    {
        app myapp=new app();
        Assert.assertEquals(1,myapp.userlogin("abc","abc@1234"));
    }
}

```

In pom.xml add the following dependency to run Testcases.

```

<dependencies>
    <!-- https://mvnrepository.com/artifact/org.testng/testng
    -->
    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>7.5.1</version>
        <scope>test</scope>
    </dependency>

</dependencies>

```

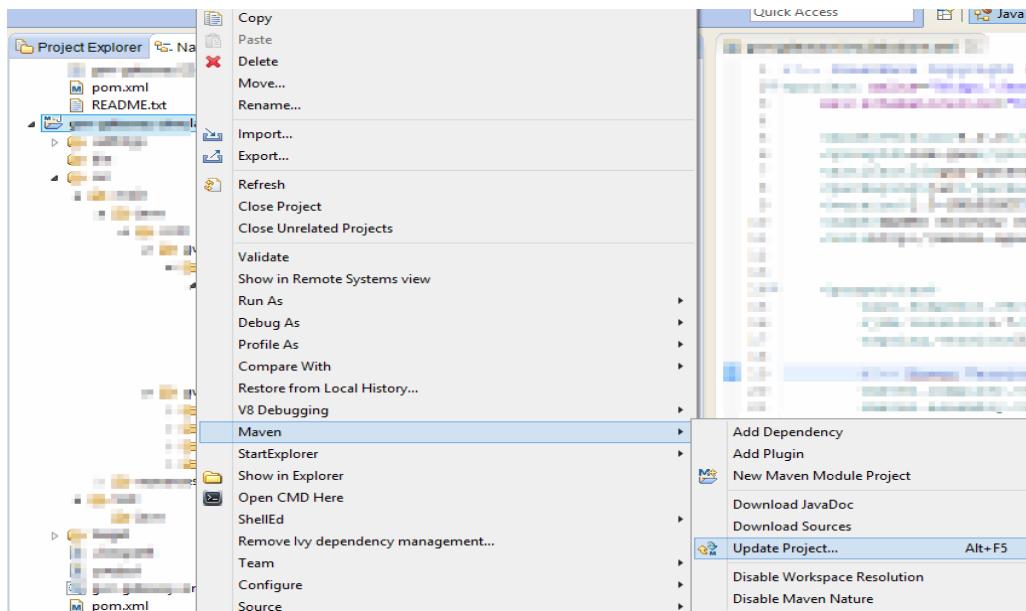
When you build your Maven project (e.g., using `mvn compile` or `mvn test`), Maven reads the `pom.xml` file.

It then downloads the specified TestNG dependency from the Maven repository and stores it in your local Maven repository.

- Your project can then use the TestNG library.

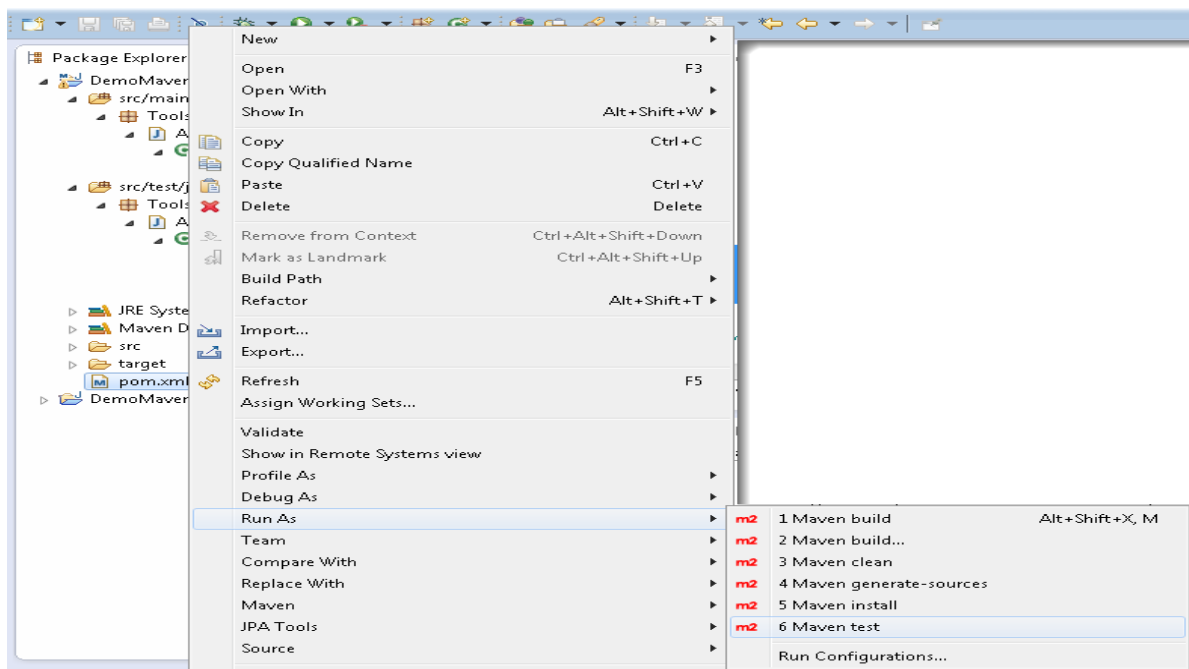
Dependencies can be updated by using "Maven --> Update Project.." after pom.xml file modification.



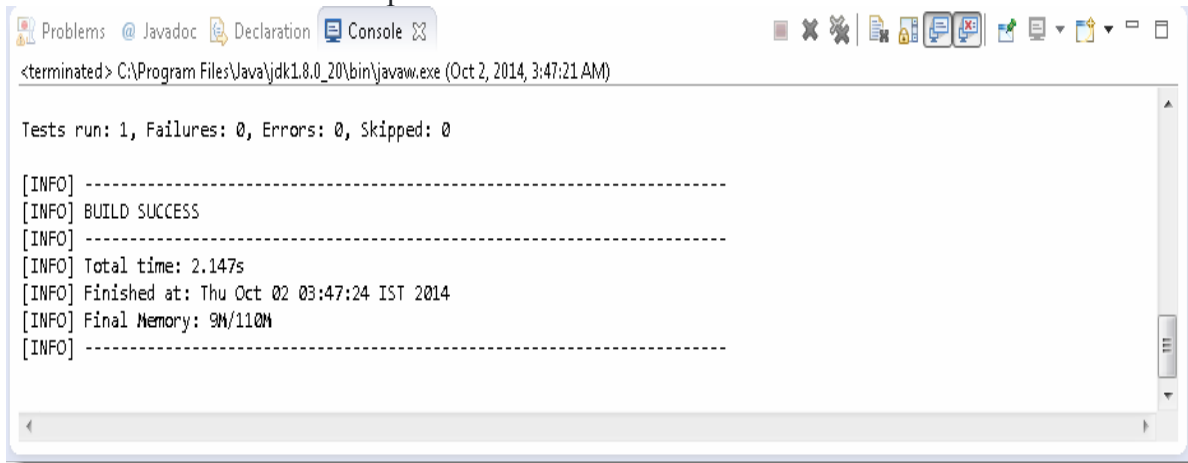


## Run your first Maven Test

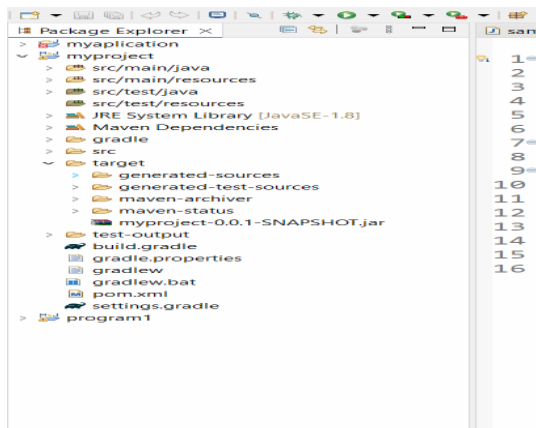
1. Right-click on the *pom.xml* and go to **Run As > Maven test**.



## 2. In the *console window* of Eclipse



## 3. Go to '*target*' folder to view the .jar/.war file



## How to install Gradle

### 1. Download Gradle:

Visit the [Gradle Downloads Page](#) and download the latest binary ZIP file.

### 2. Extract the ZIP File:

- Right-click the downloaded ZIP file and select **Extract All...** or use any extraction tool like WinRAR or 7-Zip.

#### Move the Folder:

- After extraction, move the extracted **Gradle folder** (usually named **gradle-x.x.x**) to a convenient directory like C:\Program Files\.

#### Navigate to the bin Folder:

- Open the **Gradle folder**, then navigate to the **bin** folder inside.
- Copy the path from the File Explorer address bar (e.g., C:\Program Files\gradle-x.x\bin).

#### Set Environment Variables:

- Open the **Start Menu**, search for **Environment Variables**, and select **Edit the system environment variables**.
- Click **Environment Variables**.
- Under **System Variables**:
  - Find the **path**, double click on it and click **New**.
  - Paste the full path to the bin folder of your Gradle directory (e.g., **C:\Program Files\gradle-x.x.x\bin**).

**Save the Changes:**

- Click **OK** to close the windows and save your changes.

**Verify the Installation:**

- Open a terminal or Command Prompt and run: **gradle -v** .if it shows the Gradle version, the setup is complete.
- Run **gradle init** in command prompt and check gradle folder is created in eclipse.

Make sure to run this project we have to install and configure the testNg plugin in eclipse.

**Method 1: Using the Eclipse Marketplace (Recommended)**

1. **Open Eclipse Marketplace:**
  - In Eclipse, go to Help > Eclipse Marketplace....
2. **Search for TestNG:**
  - In the "Find" field, type "TestNG" and press Enter.
3. **Install TestNG:**
  - You should see "TestNG for Eclipse" in the search results.
  - Click the "Install" button next to it.
4. **Review and Confirm:**
  - Eclipse will show you the features that will be installed.
  - Confirm the installation and accept the license agreements.
5. **Restart Eclipse:**
  - Eclipse will prompt you to restart. Click "Restart Now" to complete the installation.

**Method 2: Installing via Install New Software**

1. **Find the TestNG Update Site URL:**
  - The TestNG update site URL is typically: <https://testng.org/testng-eclipse-update-site/>.
2. **Open Install New Software:**
  - In Eclipse, go to Help > Install New Software....
3. **Add the Update Site:**
  - In the "Work with" field, paste the TestNG update site URL and press Enter.
4. **Select TestNG:**
  - Eclipse will display the available TestNG components.
  - Check the box next to "TestNG" (and any other relevant components).
5. **Click Next and Finish:**

- Click "Next" to review the installation details.
  - Click "Next" again, accept the license agreements, and click "Finish."
6. **Restart Eclipse:**
- Eclipse will prompt you to restart. Click "Restart Now" to complete the installation.

### **Verifying the Installation:**

1. **Create a Java Project (if you don't have one):**
  - Go to File > New > Java Project.
2. **Create a New TestNG Class:**
  - Right-click on your project or a source folder.
  - Go to New > Other....
  - In the "New" dialog, type "TestNG" and select "TestNG Class."
  - Click "Next" and follow the wizard to create a TestNG class. If you see the TestNG wizard, that means TestNG is correctly installed.
3. **Run a TestNG Test:**
  - Once the test class is created, you should be able to right click the java file, and see Run As -> TestNG Test.

If you encounter any issues, make sure your Eclipse installation is up to date and that you have a stable internet connection during the installation process.

---

## 2. Continuous Integration with Jenkins:

**Tool: Jenkins, Tomcat**

**Program:**

- **Install and configure Jenkins.**
- **Create a simple pipeline to:**
  - **Clone a Git repository.**
  - **Build a sample project (e.g., a Java Maven or Python project).**

**Run test cases and archive the results.**

- **Install Jenkins and Tomcat on Windows (we have uploaded videos for installation of Jenkins and Tomcat in GCR)**

Step-1: Create maven project with .war format

Step-2: Create one html/jsp file in “src/main/webapp/index.html” and add

HTML code to it.

Step-3: Create “src/main/webapp/WEB-INF” folder. In that

“src/main/webapp/WEB-INF/web.xml” file.

Step-4: Include below plugin in “web.xml”

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-war-plugin</artifactId>
<configuration>
<webXml>src\main\webapp\WEB-INF\index.jsp</webXml> (mention which
page to start)
</configuration>
</plugin>
```

Step-5: Update project – Maven clean install compile test (make sure build

Successful)

Adding Project to git:

Open terminal/command prompt, navigate to project directory and run

“git init”

Step 1: Create a GitHub Account and create new repository

---

Step 2: Install Git on Your PC, Configure your name and email

Step 3: Initialize Git in Eclipse:

Open Eclipse and go to your Maven project in the Project Explorer.

Right-click the project → Team → Share Project.

Select Git, then click Next.

Click Create to create a new local Git repository.

Click Finish.

Step 4: Add & Commit Files

Right-click the project → Team → Add to Index (this stages all files for commit).

Right-click again → Team → Commit.

Enter a commit message like "Initial commit" and click Commit.

Adding Project to git

Step 5: Connect to GitHub and Push

Copy your GitHub repository URL (from the GitHub page where you created the repository).

In Eclipse:

Right-click the project → Team → Remote → Push.

Click Create Remote.

Enter origin as the remote name.

Paste the GitHub repository URL.

Click Next.

Select Branch to Push:

Source ref: master (or main, depending on GitHub).

Destination ref: master (or main).

Click Next, then Finish.

---

Enter your GitHub username and personal access token (create using GitHub ).

### Step 6: Verify on GitHub

Go to your GitHub repository page and refresh. You should see your project files uploaded!

*Note: Next time when you modify your project in eclipse follow step 4 and 5 with commit message as “updated version”.*

---

## Jenkins Configuration

### Step1: Make Sure you have Git and Maven installed

In Jenkins UI, Goto Manage Jenkins -> Global Tool Configuration Section of Jenkins->Add maven configuration->give maven version name and select maven version from the dropdown menu.

The screenshot shows the Jenkins 'Global Tool Configuration' page. The 'Maven' section is highlighted with a red box. It contains an 'Add Maven' button, a 'Name' field with the value 'Maven3.0.5', and a 'MAVEN\_HOME' field with the value '/usr/share/maven'.

### Step2: Install maven integration, Deploy to Container and git & GitHub Plugin.

Manage Jenkins -> Manage Plugins -> Available -> Deploy to Container Plugin, maven integration & git & GitHub Plugin.

The screenshot shows the Jenkins 'Plugin Manager' page. The 'Available' tab is selected, and a search filter 'deploy' is applied. The table below lists the available plugins.

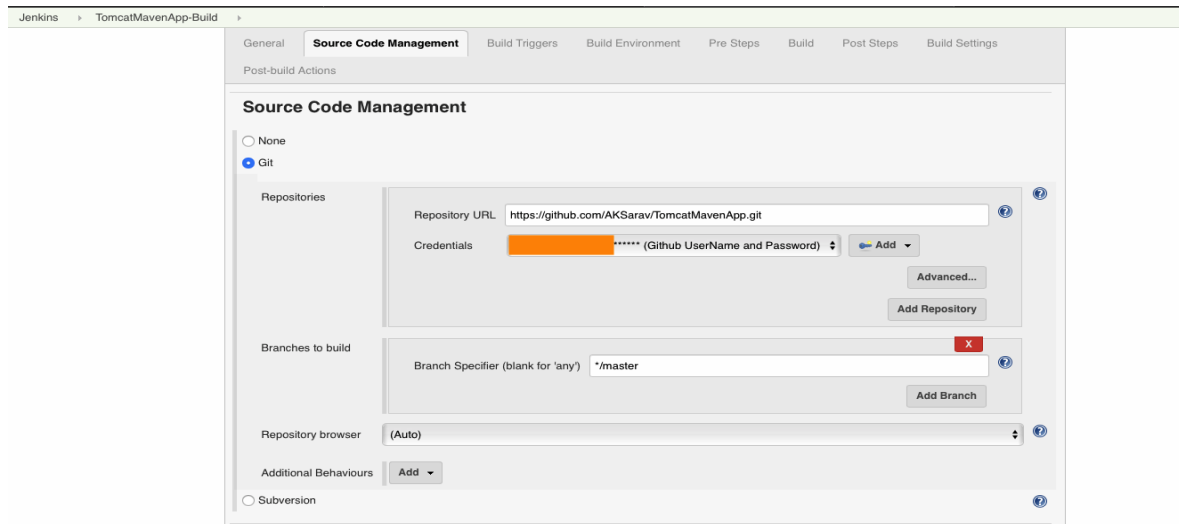
Enabled	Name	Version	Previously installed version	Uninstall
<input type="checkbox"/>	bouncycastle API Plugin	2.17		Uninstall
<input type="checkbox"/>	Command Agent Launcher Plugin	1.3		Uninstall
<input type="checkbox"/>	Credentials Plugin	2.2.0		Uninstall
<input checked="" type="checkbox"/>	Deploy to container Plugin	1.13		Uninstall
<input type="checkbox"/>	JDK Tool Plugin	1.2		Uninstall

---

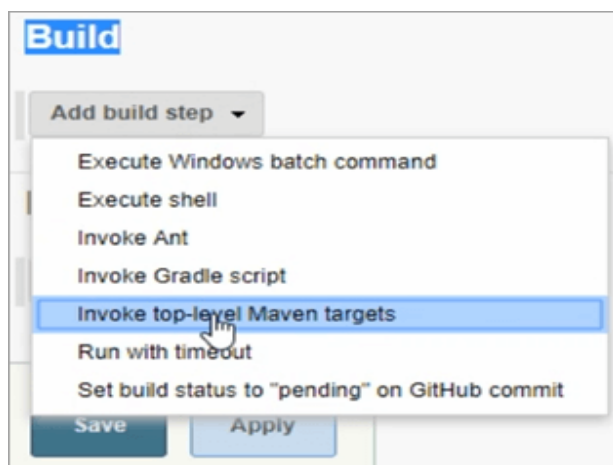
### Step3: Create and Configure a Maven Job with Source Code Management (Github)

New Item -> Select Free Style Project

- In the Configuration Section, Under Source Code Management Fill your Gitlab Repository URL (Create public Repository in GitHub).



Step 4: Next, we need to move to the Build section and select Invoke top-level Maven target options from the dropdown.



- Maven Version will be pre-populated from the Maven version we have defined in the Global Tool Configuration.
- Under Goals, we have to provide the Maven command to trigger the execution. Maven Clean and install

### Step 5 : Configure the Post-build Action and Specify the Tomcat Server Details

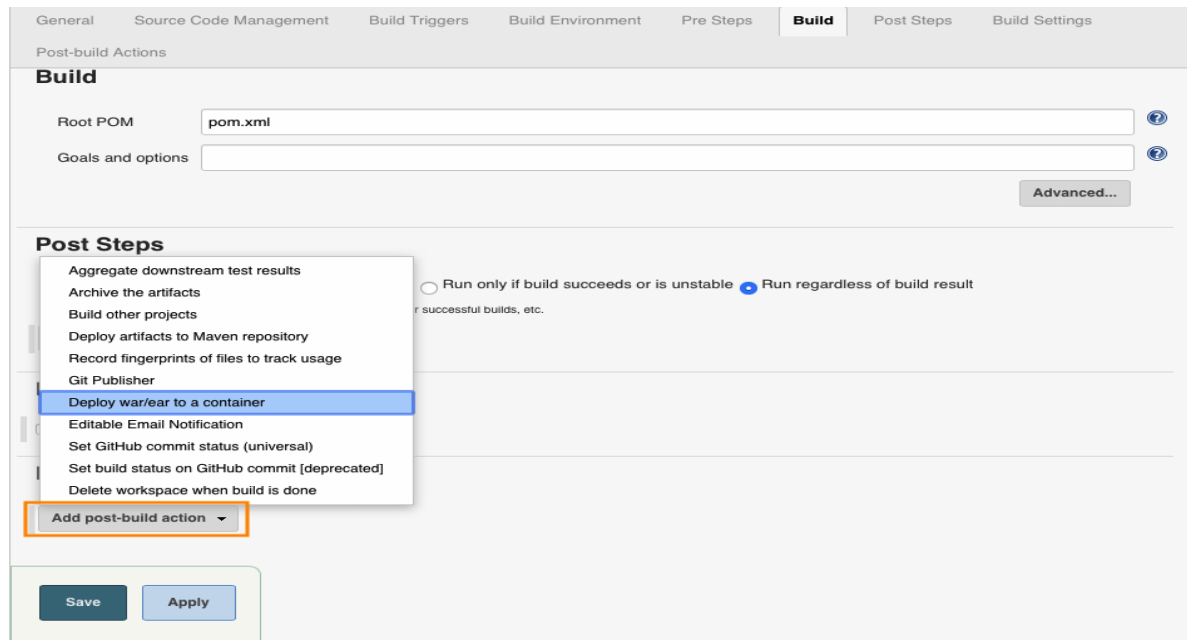


---

Drag to the bottom and Go to the **Post-build Actions** section

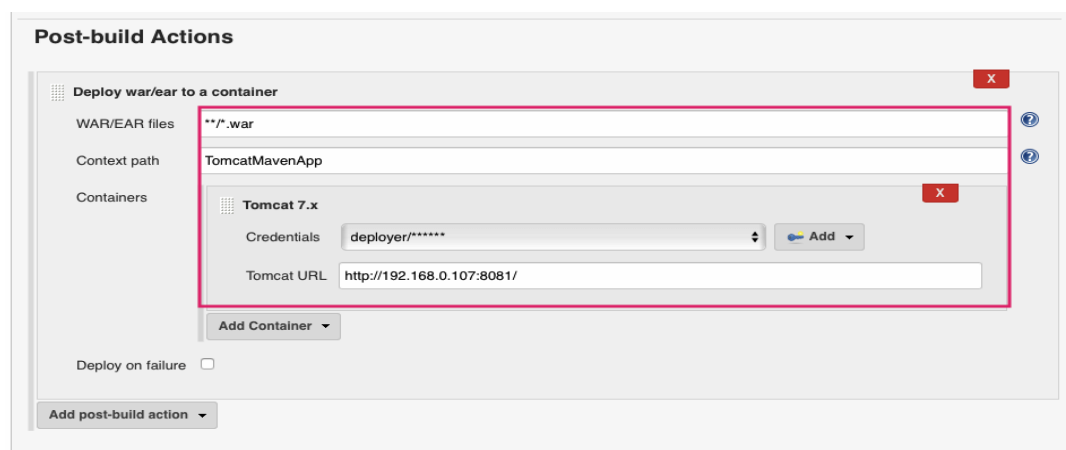
Click on **Add post-build action** button

On the available options click on the **Deploy war/ear to container**



Choose the Context Path in which the application should be installed. It would rename the WAR file before deploying to the server and thereby the application context root would be changed.

Tomcat URL http://[Tomcat Server Host]:[Primary http port]/( http://localhost:portnumber)



## Build Jenkins Job

Execute the Job you have created by clicking on the **Build Now** button

Jenkins

Jenkins > TomcatMavenApp-Build

Back to Dashboard

Status

Changes

Workspace

Build Now

Delete Maven project

Configure

Modules

Rename

Workspace

Recent Changes

Permalinks

- Last build (#5), 20 hr ago
- Last stable build (#5), 20 hr ago
- Last successful build (#5), 20 hr ago
- Last failed build (#4), 21 hr ago
- Last unsuccessful build (#4), 21 hr ago
- Last completed build (#5), 20 hr ago

Build History

trend

find X

#5 Jul 1, 2019 11:37 AM

Console Output after the Successful build.

At the last line you can see that the WAR file has been generated and deployed on the remote server.

```
-----
T E S T S
-----

Results :

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

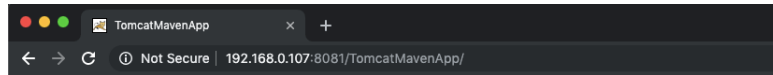
[JENKINS] Recording test results
[INFO]
[INFO] --- maven-war-plugin:2.3:war (default-war) @ TomcatMavenApp ---
[INFO] Packaging webapp
[INFO] Assembling webapp [TomcatMavenApp] in [/var/lib/jenkins/workspace/TomcatMavenApp-Build/target/TomcatMavenApp-2.0]
[INFO] Processing war project
[INFO] Copying webapp resources [/var/lib/jenkins/workspace/TomcatMavenApp-Build/src/main/webapp]
[INFO] Webapp assembled in [33 msecs]
[INFO] Building war: /var/lib/jenkins/workspace/TomcatMavenApp-Build/target/TomcatMavenApp-2.0.war
[INFO]
[INFO] --- maven-install-plugin:2.3.1:install (default-install) @ TomcatMavenApp ---
[INFO] Installing /var/lib/jenkins/workspace/TomcatMavenApp-Build/target/TomcatMavenApp-2.0.war to
/var/lib/jenkins/.m2/repository/com/sarav/TomcatMavenApp/2.0/TomcatMavenApp-2.0.war
[INFO] Installing /var/lib/jenkins/workspace/TomcatMavenApp-Build/pom.xml to
/var/lib/jenkins/.m2/repository/com/sarav/TomcatMavenApp/2.0/TomcatMavenApp-2.0.pom
[INFO] [m-----[m
[INFO] [1:32mBUILD SUCCESS[m
[INFO] [m-----[m
[INFO] Total time: 6.253 s
[INFO] Finished at: 2019-07-01T11:37:44Z
[INFO] [m-----[m
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving /var/lib/jenkins/workspace/TomcatMavenApp-Build/pom.xml to com.sarav/TomcatMavenApp/2.0/TomcatMavenApp-2.0.pom
[JENKINS] Archiving /var/lib/jenkins/workspace/TomcatMavenApp-Build/target/TomcatMavenApp-2.0.war to
com.sarav/TomcatMavenApp/2.0/TomcatMavenApp-2.0.war
channel stopped
Deploying /var/lib/jenkins/workspace/TomcatMavenApp-Build/target/TomcatMavenApp-2.0.war to container Tomcat 7.x Remote with context
TomcatMavenApp
[/var/lib/jenkins/workspace/TomcatMavenApp-Build/target/TomcatMavenApp-2.0.war] is not deployed. Doing a fresh deployment.
Deploying [/var/lib/jenkins/workspace/TomcatMavenApp-Build/target/TomcatMavenApp-2.0.war]
Finished: SUCCESS
```

## Testing the Application

As the deployment is completed and the Jenkins Job ran Successfully without issues.

Let us test our application. Got to your web browser and type the URL should be as follows

http://localhost:8081/projectname



**Welcome to Tomcat Maven Application Home Page!**

---

### 3. Containerization with Docker:

Tool: Docker Desktop, VS code, Eclipse

Program:

- Write a `Dockerfile` for a Python or Node.js application.
- Build and run a Docker image.
- Push the image to Docker Hub.
- Use Docker Compose to manage multi-container applications

#### Docker basic commands:

##### 1. Docker Version

- `docker --version`  
Docker version 24.0.2, build 12345abc

##### 2. List Docker Images

- `docker images`

##### 3. Download Image from Docker Hub

- `docker pull <image_name>:<tag>`

Example: `docker pull nginx:latest`

##### 4. Build Docker Image

- `docker build -t <image_name>:<tag> .`

Example: `docker build -t myapp:1.0 .`

##### 5. List Running Containers

- `docker ps`

##### 6. List All Containers (Including Stopped)

- `docker ps -a`

##### 7. Start a Container

- `docker start <container_name>/<container_id>`

##### 8. Stop a Container

- `docker stop <container_name>/<container_id>`

---

## 9. Restart a Container

```
docker restart <container_name>/ container_id>
```

## 10. Run a Container (Interactive Mode)

- `docker run -it -d <image_name> / <container_id>`

## 11. Run with Port Mapping

- `docker run -p <host_port>:<container_port> <image_name>`

Example : `docker run -p 8080:80 nginx`

## 12. Tag an Image for Pushing to Docker Hub

If you want to push an image to Docker Hub, it must be tagged with your Docker Hub username.

```
docker tag myapp:v1.0 myusername/myapp:v1.0
➤ myusername is your Docker Hub username.
➤ myapp:v1.0 is the image you're tagging.
```

Push the Image to Docker Hub:

```
docker push myusername/myapp:v1.
```

## Write a Dockerfile for an application.

Step-1: Create maven project with .war format

Step-2: Create one html/jsp file in “src/main/webapp/index.html” and add

HTML code to it.

Step-3: Create “src/main/webapp/WEB-INF” folder. In that

“src/main/webapp/WEB-INF/web.xml” file.

Step-4: Include below plugin in “web.xml”

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-war-plugin</artifactId>
<configuration>
<webXml>src\main\webapp\WEB-INF\index.jsp</webXml> (mention which
page to start)
</configuration>
</plugin>
```

Step-5: Update project – Maven clean install compile test (make sure build

---

Successful)

### **Prerequisites:**

Make sure you have the following installed:

1. **Docker Desktop** (running)
2. **VS Code**
3. **Docker extension for VS Code** (optional but helpful)
4. A .war file ready to use (e.g., sample.war)

### **Step 1: Place your .war File on VS code Terminal**

- Right-click the project folder in Eclipse →select properties→ Copy the project path
- Open VS code terminal →type cd\ command --> now you are in local disk c:\> type cd pate the project path
- Go to File > Open Folder
- Select the eclipse project folder you copied
- VS Code will load the project

### **Step 2: Create a new file with name Dockerfile inside ur project folder→include below lines of code**

```
FROM tomcat:9.0
RUN rm -rf /usr/local/tomcat/webapps/*
COPY /target/sample.war /usr/local/tomcat/webapps/ROOT.war
EXPOSE 8080
```

### **Step 3: Build the Docker Image by running the following command**

```
docker build -t sample-app .
```

### **Step 4: Run the Container**

```
docker run -d -p 8095(give unused port number ):8080 sample-app;
```

### **Step 5: Access the App (Goto web browser)**

```
http://localhost:8095
```

- **To Stop and Clean Up**

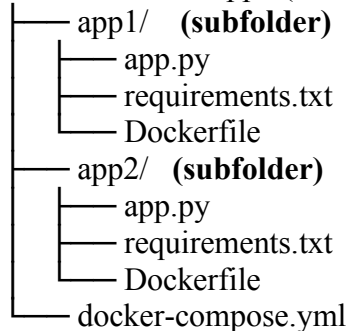
```
docker ps # get container ID
docker stop <container_id>
docker rm <container_id>
```

---

- **Use Docker Compose to manage multi-container applications**

To containerize two simple applications, enable communication between them, and deploy them on a local server using Docker

**Multi-container-app (create one main folder)**



**Step 1: Create App 1 (Service Provider Folder in VS-CODE)**

**app1/app.py(File)**

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello():
    return "Hello from App 1!"
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

**app1/requirements.txt (file)**

```
flask==3.0.0
```

**app1/Dockerfile (file)**

```
FROM python:3.12-slim
WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
EXPOSE 5000
CMD ["python", "app.py"]
```

**Step 2: Create App 2 (Service Consumer Folder in VS-CODE)**

**app2/app.py (file)**

```
import requests
response = requests.get("http://app1:5000/")
print("Response from App 1:", response.text)
```

---

### app2/requirements.txt (file)

```
requests==2.31.0
```

### app2/Dockerfile(file)

```
FROM python:3.12-slim
WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
CMD ["python", "app.py"]
```

## Step 3: Create Docker Compose File

### docker-compose.yml

```
version: '3.9'
services:
  app1:
    build: ./app1
    networks:
      - app-network
    ports:
      - "5000:5000"

  app2:
    build: ./app2
    networks:
      - app-network
    depends_on:
      - app1

networks:
  app-network:
    driver: bridge
```

### Explanation:

- Networks → Both apps are on the same network (app-network) to communicate.
- Depends\_on → Ensures App 1 starts before App 2.
- app1:5000 → App 2 can communicate with App 1 using the service name app1.

## Step 4: Build and Run the Containers

### 1. Build the Docker images:

```
docker-compose build
```



---

2. **Start the containers:**

`docker-compose up`

**Step 5: Access the Applications**

- **Check logs from App 2** to see the response from App 1:

`docker-compose logs app2`

You should see:

Response from App 1: Hello from App 1!

- **Manually test App 1** by opening a browser and visiting:

`http://localhost:5000`

### **Program 3: Container Orchestration with Kubernetes:**

- **Tool:** Kubernetes
- **Program:**
  - Set up a Kubernetes cluster (use Minikube or a cloud provider).
  - Deploy a sample application using a Deployment and Service.
  - Scale the application using `kubectl scale`.

1. Set up a Kubernetes cluster using Minikube:

#### **Prerequisites:**

- Docker Desktop installed and running.
- Windows 10/11 (PowerShell or CMD is fine)

#### **Manual Install**

- Download Minikube for Windows:  
<https://github.com/kubernetes/minikube/releases/latest>
- Download `minikube-windows-amd64.exe`  
Rename it to `minikube.exe`
- Add it to a folder in your system's PATH  
(e.g., `C:\tools\minikube\` and add that to environment variables > PATH)

#### **Start Minikube Using Docker:**

Once installed, start it using Docker as the driver:

Open PowerShell as Administrator. Run the following command in powershell

➤ **`minikube start --driver=docker`**

See it download the base image and initialize the cluster.

#### **Verify its working: Check the status:**

➤ **`minikube status`**

#### **Check cluster:**

➤ **`kubectl get nodes`**

You should see a node named minikube in the Ready state.

#### **What is Kubernetes?**

**Kubernetes** ( "K8s") is an open-source platform that helps you:

- Run, Manage, Scale, Update your containerized applications automatically.

### **Why do people use Kubernetes?**

#### **Python web app in a Docker container.**

Without Kubernetes:

- manually start containers
- monitor them yourself
- If they crash, you restart them manually
- figure out how to load balance traffic
- handle deployments by hand

With Kubernetes:

- It runs multiple copies (pods) of your app
- It restarts them if they crash
- It scales up/down based on traffic
- It load balances requests
- It updates apps with zero downtime (rolling updates)
- It manages configs & secrets securely

### **Key Concepts**

<b>Term</b>	<b>What It Is</b>
Pod	The smallest unit – runs one or more containers
Deployment	Defines how many pods to run and how to manage them
Service	A stable IP or name to access your app (load balancing)
ConfigMap	Stores non-sensitive config (env vars)
Secret	Stores sensitive data (passwords, API keys)
Node	A worker machine (VM or physical) that runs pods
Cluster	A group of nodes controlled by Kubernetes

### **Deploy a sample application using a Deployment and Service.**

In terminal (PowerShell or CMD): type

Verify with:

- **minikube start**
- **minikube status**

## Create a simple Pod YAML

Let's make a pod that runs a basic NGINX container.

Save this as pod.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

## Apply the pod YAML

- Run: **kubectl apply -f pod.yaml**

Check if it's running:

- **kubectl get pods**

You should see:

NAME	READY	STATUS	RESTARTS	AGE
my-nginx	1/1	Running	0	<time>

Access the **pod (for web apps)**. You can access it inside the cluster:

- **kubectl get pods -o wide** (it displays complete information about the each running pods)

```
PS C:\Users\Admin\Desktop\p4\app1> kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE       NOMINATED NODE   RE
ADINESS GATES
hw-deployment-9dcf4b4d6-4zwdb       1/1     Running   0           2m7s  10.244.0.88   minikube   <none>           <n
one>
hw-deployment-9dcf4b4d6-cr14m       1/1     Running   0           36m   10.244.0.86   minikube   <none>           <n
one>
hw-deployment-9dcf4b4d6-tpwkj       1/1     Running   0           36m   10.244.0.87   minikube   <none>           <n
one>
nginx                                1/1     Running   0           59m   10.244.0.81   minikube   <none>           <n
one>
Activate Windows
Go to Settings to activate Windows.
PS C:\Users\Admin\Desktop\p4\app1>
```

➤ **minikube ssh** -> it will login into the minikube cluster

Then use:

➤ **curl <nginx ip-address>** to See the NGINX welcome page inside the cluster.

---

Create a Kubernetes Deployment and Service for a simple Python web application (like Flask) running in Minikube.

### Sample Python App (Flask)

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello from App 1!! Kubernetes, also known as K8s, is an open source system for automating deployment, scaling, and management of containerized applications"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

requirement.txt

```
flask==3.0.0
```

### Dockerfile:

```
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
EXPOSE 5000
CMD ["python", "app.py"]
```

### Build and Push Docker Image

Make sure Docker is running. Then build:

- `docker build -t chethanaravi/app1-k8s:latest .`
- `docker push chethanaravi/app1-k8s:latest`

Now the image is locally available inside Minikube.

### Kubernetes Deployment (deployment.yaml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hw-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hello-world
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
        - name: hw-container
          image: chethanaravi/app1-k8s:latest
          ports:
            - containerPort: 5000
```

### Kubernetes Service (service.yaml)

```

apiVersion: v1
kind: Service
metadata:
  name: hello-world
spec:
  type: NodePort
  selector:
    app: hello-world
  ports:
    - port: 5000
      targetPort: 5000

```

This makes your app accessible via NodePort on port 30005. Apply the Manifests

- **kubectl apply -f deployment.yaml**
- **kubectl apply -f service.yaml**

Verify:

- **kubectl get pods**
- **kubectl get svc**

The screenshot shows a Visual Studio Code interface with a terminal window open. The terminal displays the following commands and output:

```

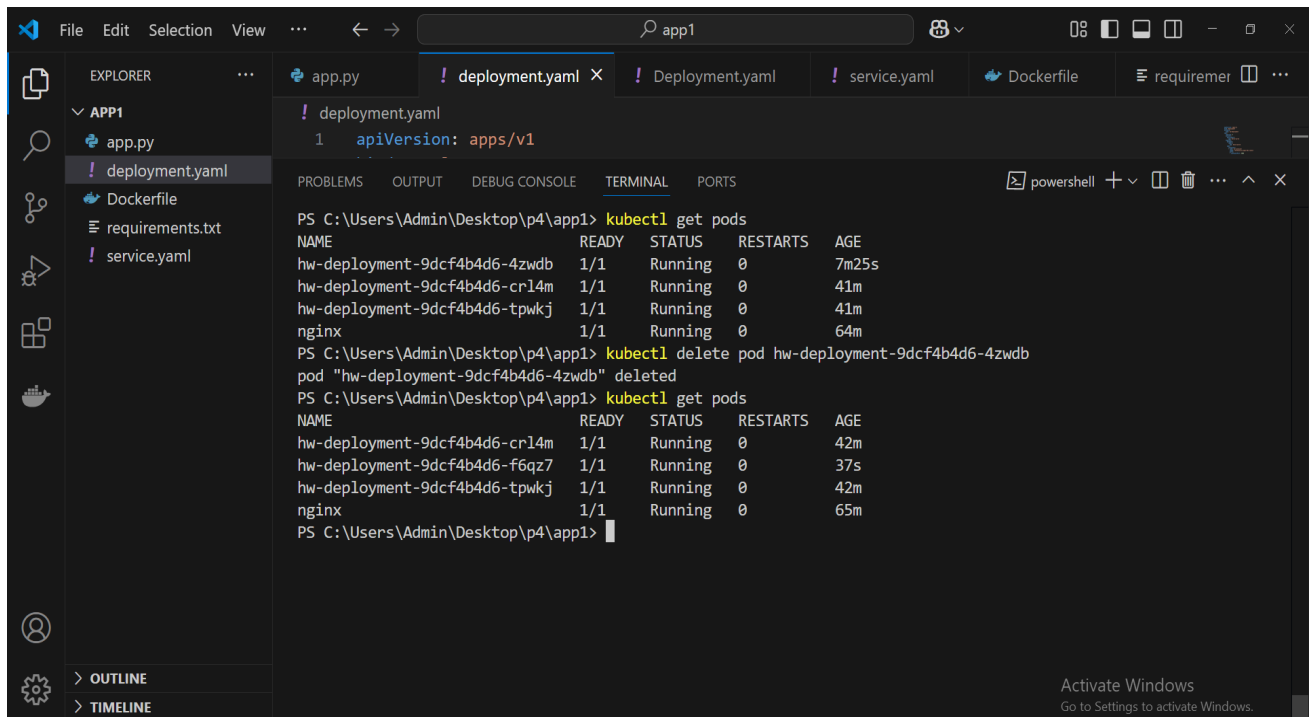
PS C:\Users\Admin\Desktop\p4\app1> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hw-deployment-9dcf4b4d6-cr14m       1/1     Running   0           26m
hw-deployment-9dcf4b4d6-tpwkj       1/1     Running   0           26m
nginx                                1/1     Running   0           49m
PS C:\Users\Admin\Desktop\p4\app1> kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
hw-deployment 2/2      2             2           26m
PS C:\Users\Admin\Desktop\p4\app1>

```

Replicating pods in Kubernetes is easy using **Deployments**. This is to tell Kubernetes how many **replicas** (copies) of your pod you want.

Syntax:

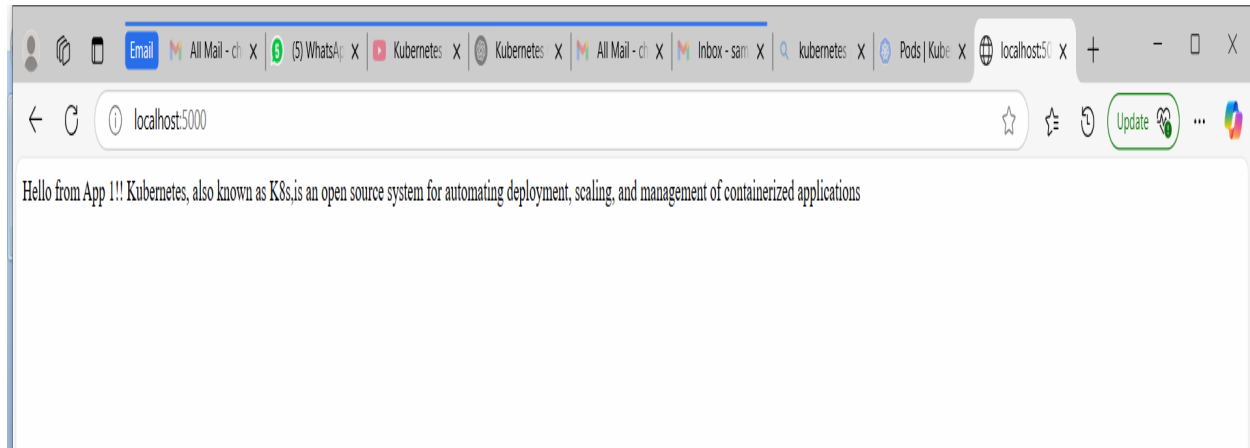
- **kubectl scale deployment <deployment-name> --replicas=<number>**
- **Example: kubectl scale deployment/hw-deployment --replicas=3**





Forwards container port 5000 to host port 5000

- **kubectrl port-forward svc/hello-world 5000:5000**
- **Goto browser and type <http://localhost:5000>**



## Simple Python application in Kubernetes using ConfigMap and Secret.

- Python App (app.py)

```
from flask import Flask
import os

app = Flask(__name__)

@app.route('/')
def index():
    app_env = os.getenv("APP_ENV", "not set")
    db_password = os.getenv("DB_PASSWORD", "not set")
    return f"APP_ENV: {app_env} <br> DB_PASSWORD: {db_password}"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

- Dockerfile

```
FROM python:3.9-slim

WORKDIR /app
COPY app1.py .
RUN pip install flask
CMD ["python", "app1.py"]
```

- Kubernetes Deployment (deployment.yaml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: python-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: python-app
  template:
    metadata:
      labels:
        app: python-app
    spec:
      containers:
        - name: app-container
          image: chethanaravi/python-app:latest
          ports:
```

```
- containerPort: 5000
env:
  - name: APP_ENV
    valueFrom:
      configMapKeyRef:
        name: my-config
        key: APP_ENV
  - name: DB_PASSWORD
    valueFrom:
      secretKeyRef:
        name: my-secret
        key: DB_PASSWORD
```

- Service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: python-service
spec:
  type: NodePort
  selector:
    app: python-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
      nodePort: 30005
```

- ConfigMap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
data:
  APP_ENV: production
```

- Secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
stringData:
  DB_PASSWORD: mypassword123
```

## Build and Push Docker Image

- **docker built -t python-app .**
- **docker built -t chethanaravi/python-app:latest**

## Apply Everything

- **kubectl apply -f configmap.yaml**
- **kubectl apply -f secret.yaml**
- **kubectl apply -f deployment.yaml**
- **kubectl apply -f service.yaml**

## Check the Pod and Service Status

- **kubectl get pods**
- **kubectl get svc**

The screenshot shows a Visual Studio Code window with several files open: app1.py, Deployment.yaml, Service.yaml, ConfigMap.yaml, Secret.yaml, and pod.yaml. The terminal window displays the following commands and output:

```
PS C:\Users\Admin\Desktop\p4\app3> kubectl apply -f deployment.yaml
deployment.apps/python-app created
PS C:\Users\Admin\Desktop\p4\app3> kubectl apply -f service.yaml
service/python-service created
PS C:\Users\Admin\Desktop\p4\app3> kubectl apply -f ConfigMap.yaml
configmap/my-config unchanged
PS C:\Users\Admin\Desktop\p4\app3> kubectl apply -f Service.yaml
service/python-service unchanged
PS C:\Users\Admin\Desktop\p4\app3> kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
hw-deployment-9dcf4b4d6-cn14m	1/1	Running	0	130m
hw-deployment-9dcf4b4d6-f6qz7	1/1	Running	0	89m
hw-deployment-9dcf4b4d6-tpwkj	1/1	Running	0	130m
nginx	1/1	Running	0	153m
python-app-5984c86dc-55f4f	1/1	Running	0	116s

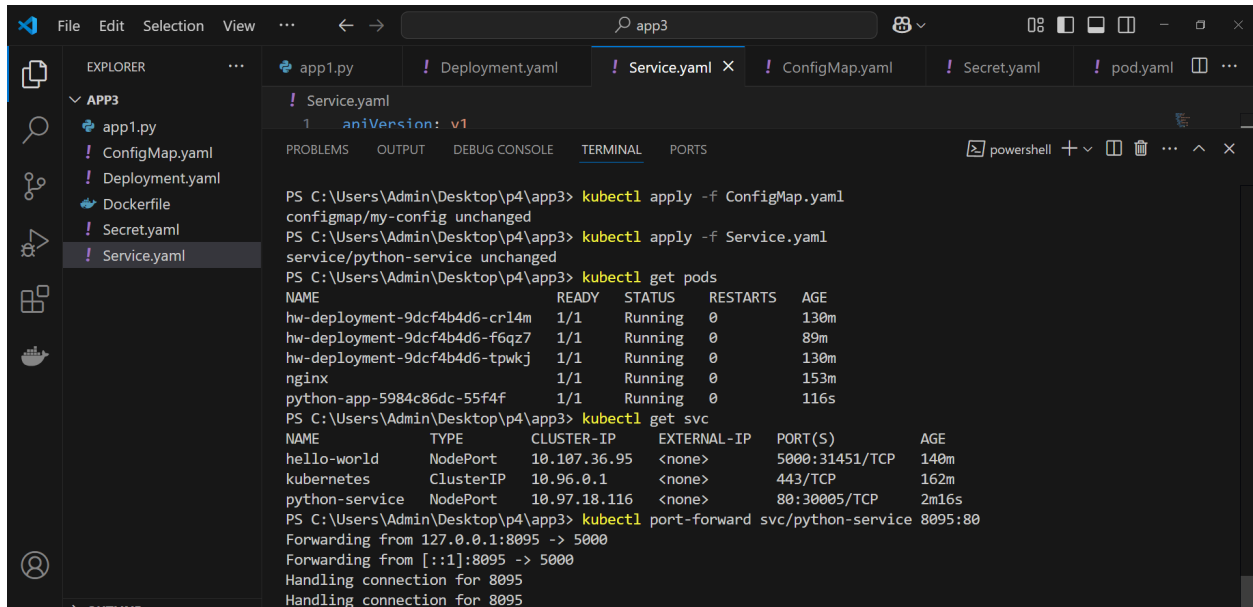
```
PS C:\Users\Admin\Desktop\p4\app3> kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hello-world	NodePort	10.107.36.95	<none>	5000:31451/TCP	140m
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	162m
python-service	NodePort	10.97.18.116	<none>	80:30005/TCP	2m16s

PS C:\Users\Admin\Desktop\p4\app3>

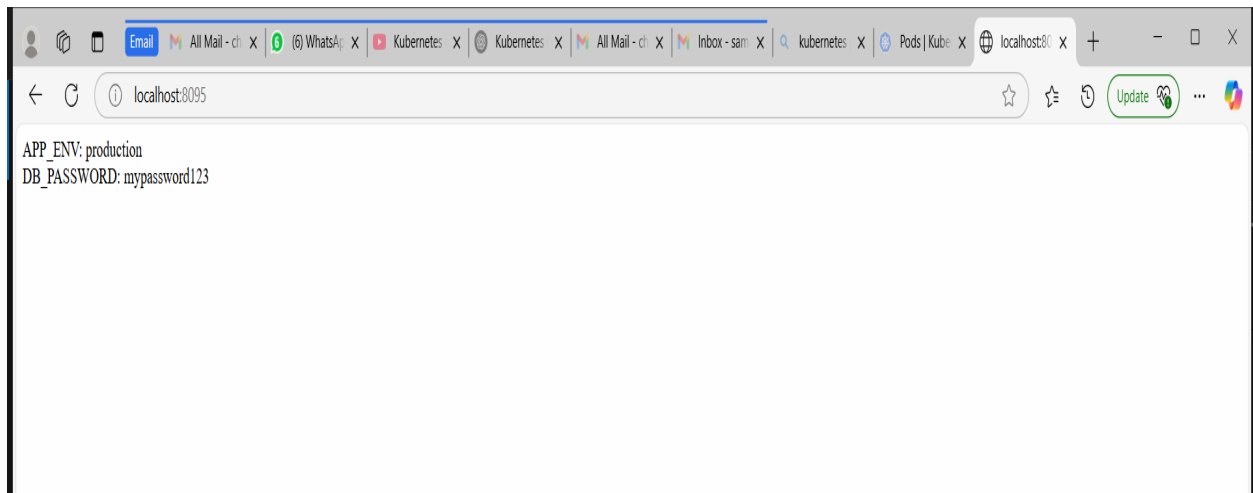
## Port Forwarding

- **kubectl port-forward svc/python-service 8095:80**
- <http://localhost:8095>



```
PS C:\Users\Admin\Desktop\p4\app3> kubectl apply -f ConfigMap.yaml
configmap/my-config unchanged
PS C:\Users\Admin\Desktop\p4\app3> kubectl apply -f Service.yaml
service/python-service unchanged
PS C:\Users\Admin\Desktop\p4\app3> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hw-deployment-9dcf4b4d6-cr14m       1/1     Running   0           130m
hw-deployment-9dcf4b4d6-f6qz7       1/1     Running   0           89m
hw-deployment-9dcf4b4d6-tpwkj       1/1     Running   0           130m
nginx                               1/1     Running   0           153m
python-app-5984c86dc-55f4f          1/1     Running   0           116s
PS C:\Users\Admin\Desktop\p4\app3> kubectl get svc
NAME            TYPE          CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
hello-world     NodePort      10.107.36.95   <none>         5000:31451/TCP   140m
kubernetes       ClusterIP     10.96.0.1      <none>         443/TCP          162m
python-service   NodePort      10.97.18.116   <none>         80:30005/TCP     2m16s
PS C:\Users\Admin\Desktop\p4\app3> kubectl port-forward svc/python-service 8095:80
Forwarding from 127.0.0.1:8095 -> 5000
Forwarding from [::1]:8095 -> 5000
Handling connection for 8095
Handling connection for 8095
```

This bypasses NodePort and goes directly to the service inside the cluster.



**Note:**

- Delete All Pods in the Current Namespace (usually default):

**kubectrl delete pods --all**

- Delete Everything (Pods, Deployments, Services, etc.)

**kubectrl delete all --all**

- Prevent Pods from Coming Back

**kubectrl delete deployment <deployment-name> / kubectrl delete deployments --all**

## **Program No-5**

### **Basic Infrastructure Deployment**

- **Tool:** Terraform
- **Program:**
  - Write a Terraform configuration file to provision a single EC2 instance on AWS.

Use Terraform commands (terraform init, terraform plan, terraform apply, terraform destroy) to manage the infrastructure.

### **Infrastructure as Code (IaC)**

Before the advent of IaC, infrastructure management was typically a manual and time-consuming process. System administrators and operations teams had to:

1. **Manually Configure Servers:** Servers and other infrastructure components were often set up and configured manually, which could lead to inconsistencies and errors.
2. **Lack of Version Control:** Infrastructure configurations were not typically version-controlled, making it difficult to track changes or revert to previous states.
3. **Documentation Heavy:** Organizations relied heavily on documentation to record the steps and configurations required for different infrastructure setups. This documentation could become outdated quickly.
4. **Limited Automation:** Automation was limited to basic scripting, often lacking the robustness and flexibility offered by modern IaC tools.
5. **Slow Provisioning:** Provisioning new resources or environments was a time-consuming process that involved multiple manual steps, leading to delays in project delivery.

IaC addresses these challenges by providing a systematic, automated, and code-driven approach to infrastructure management. Popular IaC tools include Terraform, AWS CloudFormation, Azure Resource Manager templates others.

These tools enable organizations to define, deploy, and manage their infrastructure efficiently and consistently, making it easier to adapt to the dynamic needs of modern applications and services.

### **Why Terraform ?**

There are multiple reasons why Terraform is used over the other IaC tools but below are the main reasons.

1. **Multi-Cloud Support:** Terraform is known for its multi-cloud support. It allows you to define infrastructure in a cloud-agnostic way, meaning you can use the same configuration code to provision resources on various cloud providers (AWS, Azure, Google Cloud, etc.) and even on-premises infrastructure. This flexibility can be beneficial if your organization uses multiple cloud providers or plans to migrate between them.
2. **Large Ecosystem:** Terraform has a vast ecosystem of providers and modules contributed by both HashiCorp (the company behind Terraform) and the community. This means you can find pre-built modules and configurations for a wide range of services and infrastructure components, saving you time and effort in writing custom configurations.
3. **Declarative Syntax:** Terraform uses a declarative syntax, allowing you to specify the desired end-state of your infrastructure. This makes it easier to understand and maintain your code compared to imperative scripting languages.
4. **State Management:** Terraform maintains a state file that tracks the current state of your infrastructure. This state file helps Terraform understand the differences between the desired and actual states of your infrastructure, enabling it to make informed decisions when you apply changes.
5. **Plan and Apply:** Terraform's "plan" and "apply" workflow allows you to preview changes before applying them. This helps prevent unexpected modifications to your infrastructure and provides an opportunity to review and approve changes before they are implemented.
6. **Community Support:** Terraform has a large and active user community, which means you can find answers to common questions, troubleshooting tips, and a wealth of documentation and tutorials online.
7. **Integration with Other Tools:** Terraform can be integrated with other DevOps and automation tools, such as Docker, Kubernetes, Ansible, and Jenkins, allowing you to create comprehensive automation pipelines.
8. **HCL Language:** Terraform uses HashiCorp Configuration Language (HCL), which is designed specifically for defining infrastructure. It's human-readable and expressive, making it easier for both developers and operators to work with.

To get started with Terraform, it's important to understand some key terminology and concepts. Here are some fundamental terms and explanations.

1. **Provider:** A provider is a plugin for Terraform that defines and manages resources for a specific cloud or infrastructure platform. Examples of providers include AWS, Azure, Google Cloud, and many others. You configure providers in your Terraform code to interact with the desired infrastructure platform.
2. **Resource:** A resource is a specific infrastructure component that you want to create and manage using Terraform. Resources can include virtual machines, databases, storage



buckets, network components, and more. Each resource has a type and configuration parameters that you define in your Terraform code.

3. **Module:** A module is a reusable and encapsulated unit of Terraform code. Modules allow you to package infrastructure configurations, making it easier to maintain, share, and reuse them across different parts of your infrastructure. Modules can be your own creations or come from the Terraform Registry, which hosts community-contributed modules.
4. **Configuration File:** Terraform uses configuration files (often with a .tf extension) to define the desired infrastructure state. These files specify providers, resources, variables, and other settings. The primary configuration file is usually named main.tf, but you can use multiple configuration files as well.
5. **Variable:** Variables in Terraform are placeholders for values that can be passed into your configurations. They make your code more flexible and reusable by allowing you to define values outside of your code and pass them in when you apply the Terraform configuration.
6. **Output:** Outputs are values generated by Terraform after the infrastructure has been created or updated. Outputs are typically used to display information or provide values to other parts of your infrastructure stack.
7. **State File:** Terraform maintains a state file (often named terraform.tfstate) that keeps track of the current state of your infrastructure. This file is crucial for Terraform to understand what resources have been created and what changes need to be made during updates.
8. **Plan:** A Terraform plan is a preview of changes that Terraform will make to your infrastructure. When you run terraform plan, Terraform analyzes your configuration and current state, then generates a plan detailing what actions it will take during the apply step.
9. **Apply:** The terraform apply command is used to execute the changes specified in the plan. It creates, updates, or destroys resources based on the Terraform configuration.
10. **Workspace:** Workspaces in Terraform are a way to manage multiple environments (e.g., development, staging, production) with separate configurations and state files. Workspaces help keep infrastructure configurations isolated and organized.
11. **Remote Backend:** A remote backend is a storage location for your Terraform state files that is not stored locally. Popular choices for remote backends include Amazon S3, Azure Blob Storage, or HashiCorp Terraform Cloud. Remote backends enhance collaboration and provide better security and reliability for your state files.

## **Install Terraform:**

### **Windows**

#### **Installing Terraform on Windows**

Here are the steps to install Terraform on Windows:

##### **1. Download Terraform:**

- Open your web browser and go to the official Terraform download page: <https://www.terraform.io/downloads.html>
- Find the appropriate Windows version (usually windows\_amd64 for 64-bit systems).
- Click the download link to download the ZIP archive.

##### **2. Extract the ZIP Archive:**

- Once the download is complete, locate the ZIP file (e.g., terraform\_1.7.5\_windows\_amd64.zip).
- Right-click on the ZIP file and select "Extract All..."
- Choose a destination folder to extract the contents (e.g., C:\Program Files\Terraform).
- Click "Extract".

##### **3. Add Terraform to your PATH:**

- To run Terraform from any command prompt window, you need to add the Terraform executable to your system's PATH environment variable.
- **Open Environment Variables Settings:**
  - Right-click on the Start button and select "System".
  - Click on "Advanced system settings".
  - In the "System Properties" dialog, click on the "Environment Variables..." button.
- **Edit the PATH Variable:**
  - In the "Environment Variables" dialog, under "System variables", find the "Path" variable and click "Edit...".
  - Click "New".
  - Enter the path to the directory where you extracted the Terraform executable (e.g., C:\Program Files\Terraform).
  - Click "OK" to close all the dialog boxes.

##### **4. Verify the Installation:**

- Open a new Command Prompt window (or PowerShell).
- Type terraform -version and press Enter.
- If Terraform is installed correctly, you will see the Terraform version number displayed.

### **Example:**

```
terraform -version
Terraform v1.7.5
on windows_amd64
```

## **awscli Installation on windows:**

### **For Windows:**

1. Download the latest AWS CLI MSI installer from the official AWS documentation:
  - <https://aws.amazon.com/cli/>
2. Execute the installer and follow the installation wizard.
3. To verify installation, open Command Prompt and run:

```
aws --version
```

## **awscli Configuration**

After successful installation, configure the AWS CLI by supplying the necessary credentials:

```
aws configure
```

You will be prompted to enter:

- **AWS Access Key ID**
- **AWS Secret Access Key**
- **Default region name** (e.g., us-east-1)
- **Default output format** (json, yaml, table, etc.)

The credentials are stored in the ~/.aws/credentials and ~/.aws/config files by default.

- **Steps to Create AWS Access and Secret Key**

### **1. Sign in to AWS Console**

- Go to: <https://console.aws.amazon.com/>
- Log in as a user with **IAM privileges** (or as the root user, though this is not recommended for day-to-day use).

### **2. Navigate to IAM (Identity and Access Management)**

- In the AWS Management Console, search for **“IAM”**.
- Click **Users** from the left-hand menu.

### 3. Select or Create a User

#### *Option A: Use an existing user*

- Click the username.
- Go to the “**Security credentials**” tab.

#### *Option B: Create a new user*

1. Click **Add users**.
2. Enter a username (e.g., terraform-user).
3. Select **Programmatic access** (this is what gives you Access and Secret keys).
4. Click **Next** to attach permissions:
  - Attach existing policies like AmazonEC2FullAccess, AmazonS3FullAccess, etc., or use AdministratorAccess (use cautiously).
5. Finish user creation.

### 4. Generate Access Keys

- Under the “**Security credentials**” tab, click **Create access key**.
- Choose the use case (e.g., **CLI**).
- Click **Next**, and then **Create access key**.
- You’ll be shown:
  - **Access Key ID**
  - **Secret Access Key** (only shown once)

**Important:** Save the secret key securely. You won’t be able to view it again. You can download a .csv file at this step.

---

## **1. Simple Terraform Project to Launch an EC2 Instance**

Step 1: Create a Project Folder with name as terraform on your desktop.

### **Step 2: Create the main.tf File**

Create a file named main.tf and paste the following code:

```
provider "aws" {  
    region = "us-west-2"  
}  
resource "aws_instance" "ec2_machine" {  
    ami = "ami-07b0c09aab6e66ee9"  
    instance_type = "t2.micro"  
    tags = {  
        Name = "Terra EC2"  
    }  
}
```

// count=4

### **Step 3: Initialize Terraform: execute following command on VS code terminal :**

```
terraform init
```

This sets up Terraform in the folder and downloads necessary provider plugins.

### **Step 4: Preview the Plan**

```
terraform plan
```

This shows what Terraform will create.

### **Step 5: Apply the Configuration**

```
terraform apply
```

When prompted, type yes to confirm. Terraform will create the EC2 instance.

### **Step 6: Verify**

Go to your **AWS Console** → **EC2** → **Instances**, and you'll see the instance running.

### **Step 7: Destroy When Done (to avoid charges)**

```
terraform destroy
```

Confirm with yes.

## **b. Terraform configuration to create an AWS S3 bucket:**

### **1. Make sure you have a text file ready:**

Create a local file named sample.txt in the current directory

**Sample1.txt**

**Hellow..welcome to terraform.**

### **2. Create a main.tf file:**

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "ec2_machine" {
  ami = "ami-07b0c09aab6e66ee9"
  instance_type = "t2.micro"
  count=4

  tags = {
    Name = "Terra EC2"
  }
}

resource "aws_s3_bucket" "demo_bucket" {
  bucket = "my-unique-s3-bucket-2025-upload-demo" (This name is globally unique, so change
the name while creating s3-bucket)
  tags = {
    Name = "upload-demo"
  }
}

resource "aws_s3_bucket_object" "text_file" {
  bucket = aws_s3_bucket.demo_bucket.bucket
  key   = "sample1.txt"
  source = "./sample1.txt"
}
```

## How to Run

1. Initialize:
  - terraform init
2. Apply:
  - terraform apply
3. Confirm yes when prompted.

## Result

After applying, sample.txt will be uploaded to the specified S3 bucket. You can verify in the AWS Console > S3.

## 3. Deploy a Scalable Web Server on AWS Using Terraform

```
provider "aws" {  
  region = "us-west-2"  
}  
  
resource "aws_security_group" "web_sg" {  
  name      = "web_sg"  
  description = "Allow HTTP inbound traffic"  
  
  ingress {  
    description = "HTTP"  
    from_port   = 80  
    to_port     = 80  
    protocol    = "tcp"  
    cidr_blocks = ["0.0.0.0/0"]  
  }  
  
  egress {  
    from_port = 0  
    to_port   = 0  
    protocol  = "-1"  
  }  
}
```

```

    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "web_server" {
  ami          = "ami-07b0c09aab6e66ee9"
  instance_type = "t2.micro"
  security_groups = [aws_security_group.web_sg.name]

  user_data = <<-EOF
    #!/bin/bash
    yum update -y
    yum install -y httpd
    systemctl start httpd
    systemctl enable httpd
    echo "<h1>Deployed via Terraform</h1>" > /var/www/html/index.html
  EOF

  tags = {
    Name = "TerraformWebServer"
  }
}

```

Save the above code to a file named `main.tf`.

In the same directory, run:

```

terraform init
terraform apply

```

Confirm the prompt to provision resources.

Once complete, go to your AWS EC2 dashboard to find the instance's public IP and test it in a browser (`http://<public-ip>`).