

## TASK 8: Run a Simple Java Maven Build Job in Jenkins

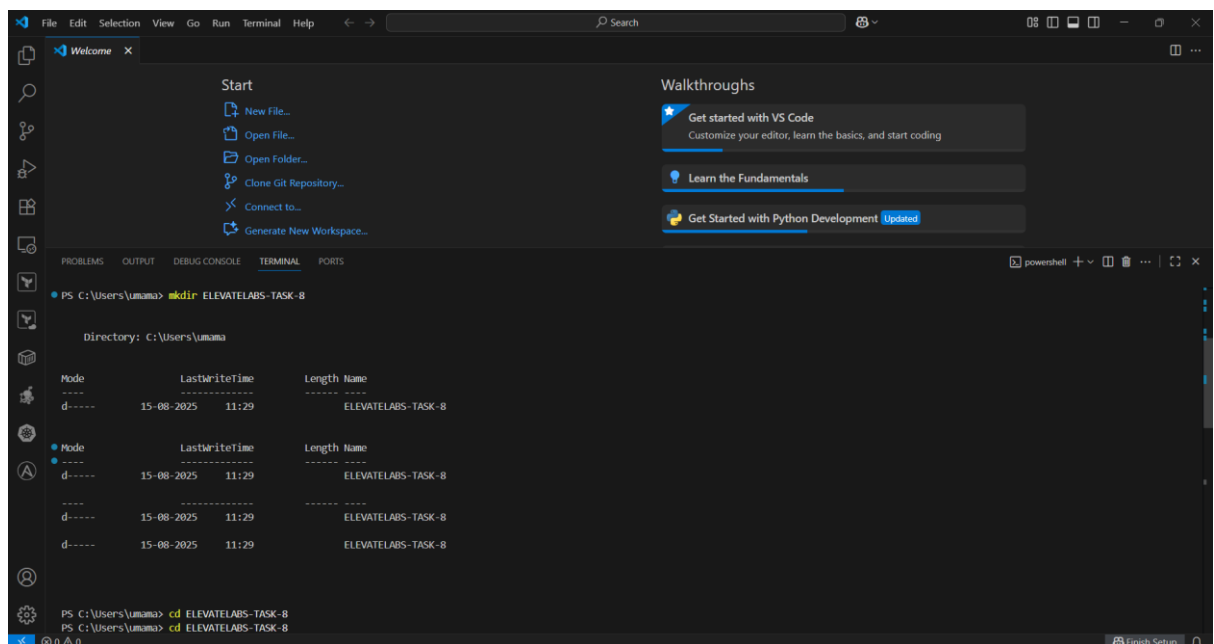
**Objective:** Learn how to use Jenkins to build a simple Java application using Maven a first step into CI/CD.

### Tools Used

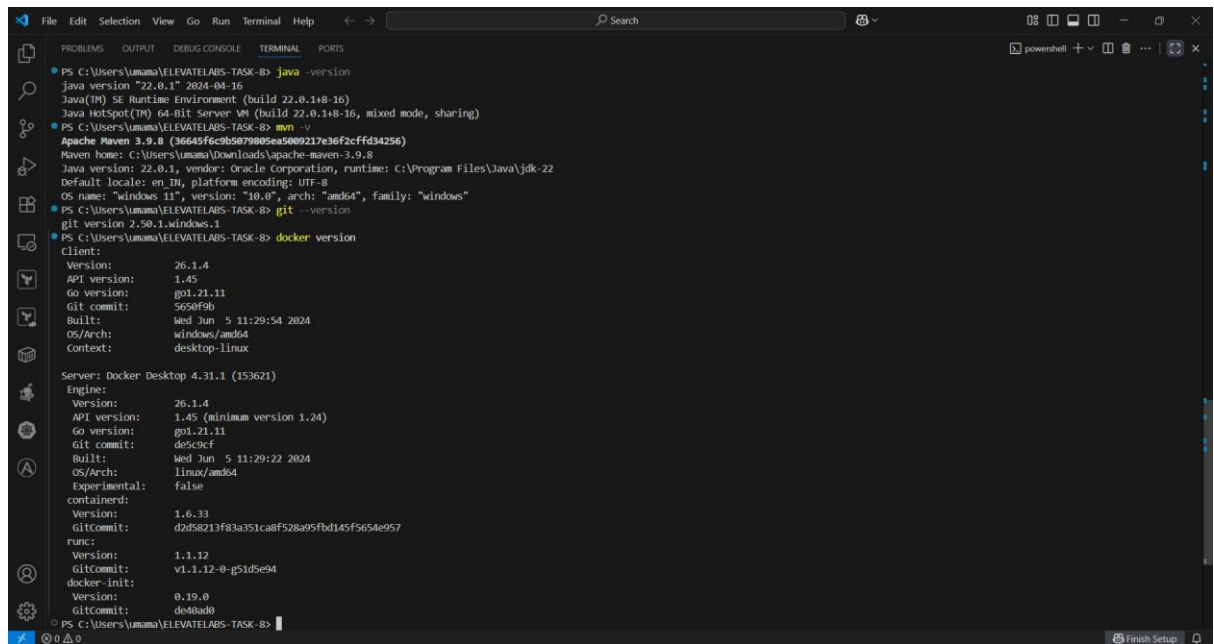
- Jenkins (via Docker)
- Java JDK
- Maven
- Git & GitHub

### Steps Followed:

#### Created the project folder locally



Install needed tools already I have I am checking its versions



```
PS C:\Users\umama\ELEVATELABS-TASK-8> java -version
java version "22.0.1" 2024-04-16
Java(TM) SE Runtime Environment (build 22.0.148-16)
Java HotSpot(TM) 64-Bit Server VM (build 22.0.148-16, mixed mode, sharing)

PS C:\Users\umama\ELEVATELABS-TASK-8> mvn -v
Apache Maven 3.9.8 (36645f6c9b5879805ea5809217e36f2cfd34256)
Maven home: C:\Users\umama\Downloads\apache-maven-3.9.8
Java version: 22.0.1, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-22
Default locale: en_IN, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"

PS C:\Users\umama\ELEVATELABS-TASK-8> git --version
git version 2.50.1.windows.1

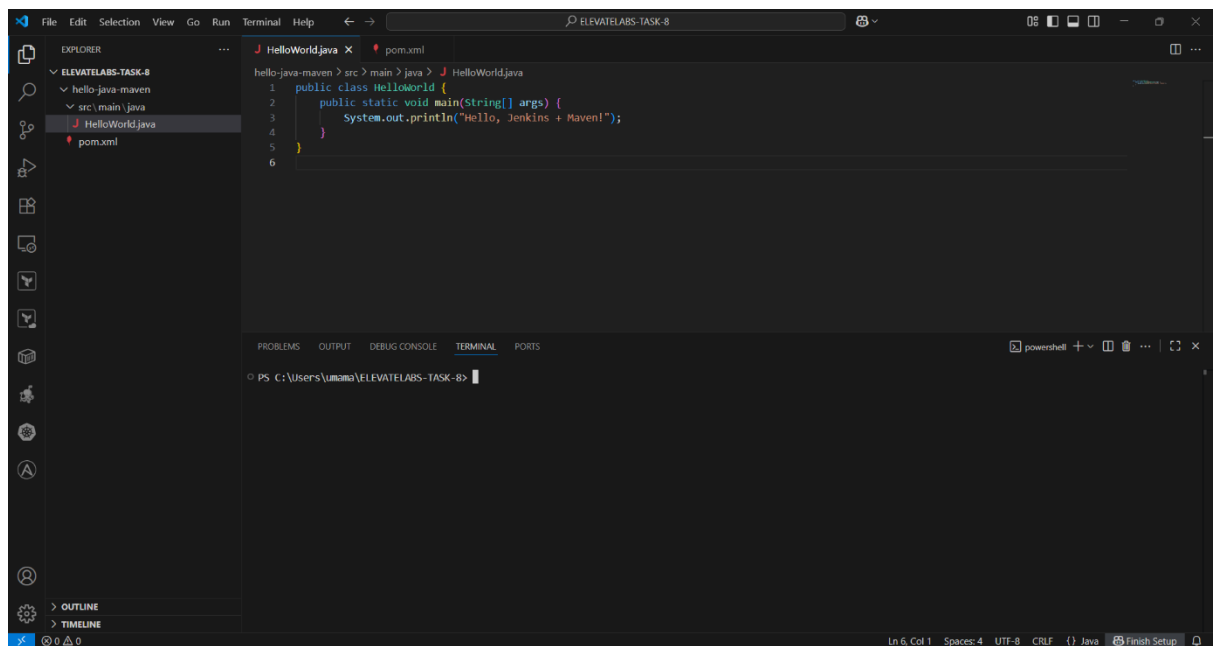
PS C:\Users\umama\ELEVATELABS-TASK-8> docker version
Client:
Version: 26.1.4
API version: 1.45
Go version: go1.21.11
Git commit: 5650f9b
Built: Wed Jun 5 11:29:54 2024
OS/Arch: windows/amd64
Context: desktop-linux

Server: Docker Desktop 4.31.1 (153621)
Engine:
Version: 26.1.4
API version: 1.45 (minimum version 1.24)
Go version: go1.21.11
Git commit: de5c9cf
Built: Wed Jun 5 11:29:22 2024
OS/Arch: linux/amd64
Experimental: false
containerd:
Version: 1.6.33
GitCommit: d2d58213f83a351ca8f528a95fbd145f5654e957
runc:
Version: 1.1.12
GitCommit: v1.1.12-0-g51d5e94
docker-init:
Version: 0.19.0
GitCommit: de40ad0

PS C:\Users\umama\ELEVATELABS-TASK-8>
```

**Create Java App:** File: src/main/java/HelloWorld.java

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



```
hello-java-maven > src > main > java > J HelloWorld.java
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello, Jenkins + Maven!");
4     }
5 }
6
```

**Explanation:**

**package com.example;** – Defines the namespace for the Java class.

**public class HelloWorld** – Declares a public class named HelloWorld.

**public static void main(String[] args)** – Entry point for the program; this method runs when the Java application starts.

**System.out.println(...)** – Prints the message to the console output.

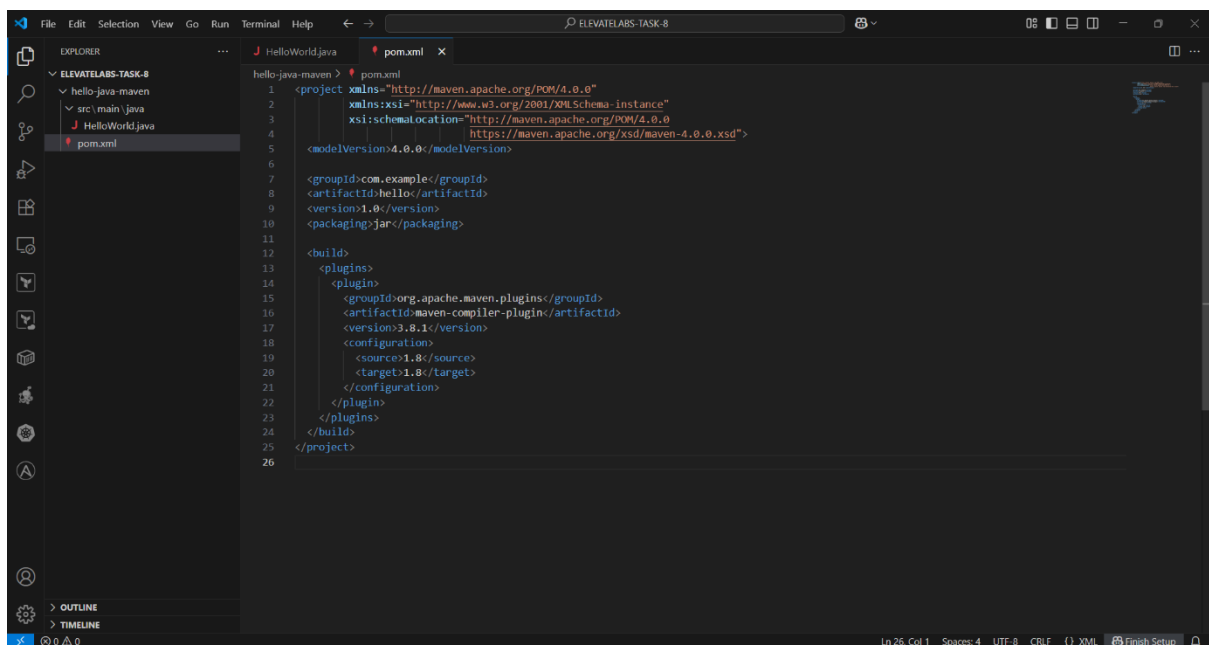
### **Create pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                    https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>hello</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

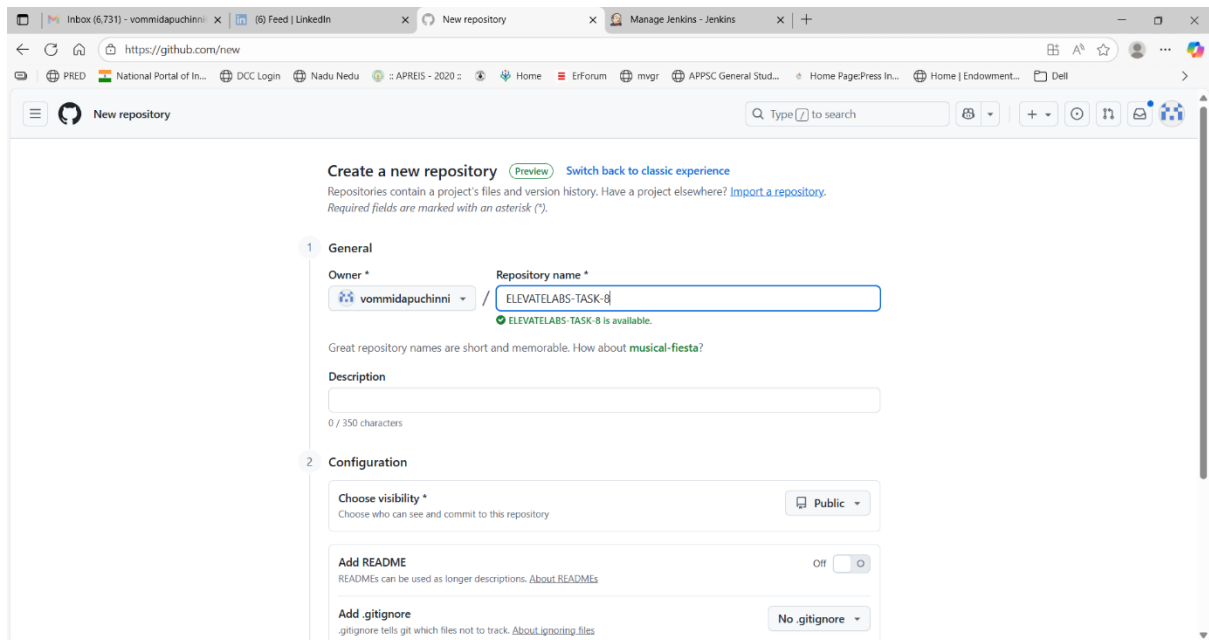
</project>

## POM File Code Explanation:

- <project> – Root element of the Maven Project Object Model (POM) file.
- <modelVersion> – Maven POM model version used (here 4.0.0).
- <groupId> – Unique ID for the project's group or organization (com.example).
- <artifactId> – The project's name or module (hello).
- <version> – Version of the project (1.0).
- <packaging> – Specifies output type (jar).
- <build> – Section containing build-related configurations.
- <plugins> – List of Maven plugins used during build.
- <plugin> – Defines a specific plugin configuration.
  - <groupId> – Group ID for the plugin (org.apache.maven.plugins).
  - <artifactId> – Plugin name (maven-compiler-plugin).
  - <version> – Plugin version (3.8.1).
  - <configuration> – Settings for the plugin.
    - <source> – Java source version (1.8).
    - <target> – Java bytecode version (1.8).



Create git repo

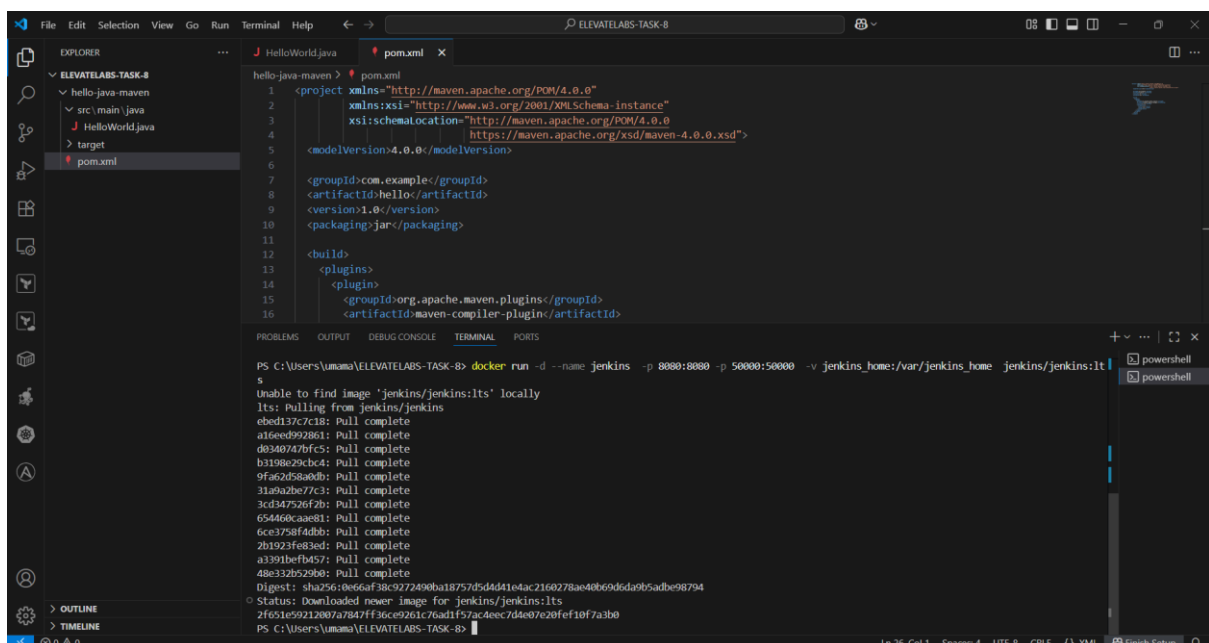


Push the files to repo

## Run Jenkins in Docker:

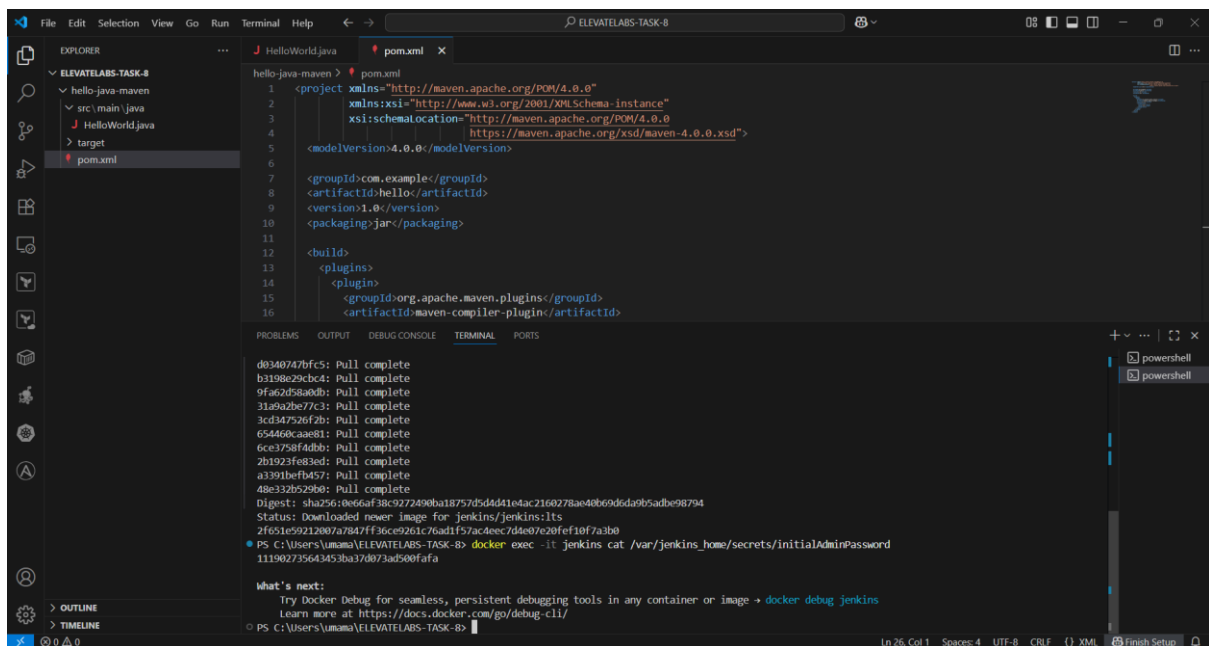
```
docker run --name jenkins -p 8080:8080 -p 50000:50000 -v
jenkins_home:/var/jenkins_home jenkins/jenkins:its
```

- **8080** → Jenkins Web Interface
- **50000** → Jenkins Agent communication



Unlock Jenkins:

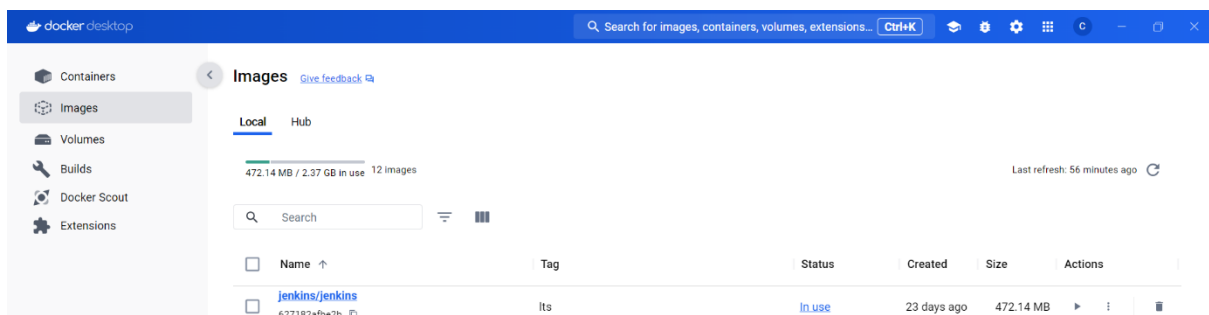
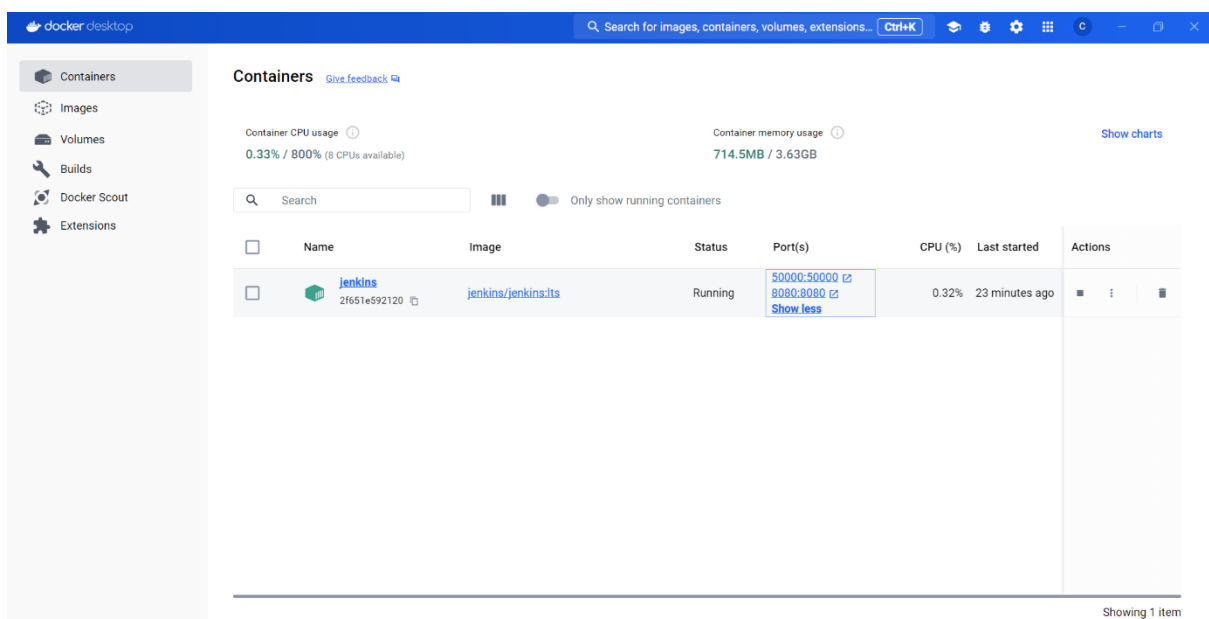
docker exec jenkins cat /var/jenkins\_home/secrets/initialAdminPassword



```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6
7     <modelVersion>4.0.0</modelVersion>
8
9     <groupId>com.example</groupId>
10    <artifactId>hello</artifactId>
11    <version>1.0</version>
12    <packaging>jar</packaging>
13
14    <build>
15        <plugins>
16            <plugin>
17                <groupId>org.apache.maven.plugins</groupId>
18                <artifactId>maven-compiler-plugin</artifactId>
19            </plugin>
20        </plugins>
21    </build>
22
23 </project>
```

```
docker pull d0340747bfc5: Pull complete
docker pull b3198e29cbcd: Pull complete
docker pull 9fa62d5a0dd5: Pull complete
docker pull 31a8a2be77c3: Pull complete
docker pull 3cd347526f2b: Pull complete
docker pull 654460caae81: Pull complete
docker pull 6ce3758f4dbb: Pull complete
docker pull 2b1923fe83ed: Pull complete
docker pull a3391befb457: Pull complete
docker pull 48a32d529d0e: Pull complete
Digest: sha256:0e66af38c9272408ba18757d5d4d41e4ac2160278ae40b69dda9b5adbe98794
Status: Downloaded newer image for jenkins/jenkins:its
2f651e59212807a7847ff36ce9261c76ad1f57ac4ec7d4e07e20ef10f7a3b0
PS C:\Users\umama\ELEVATELABS-TASK-8> docker exec -it jenkins cat /var/jenkins_home/secrets/initialAdminPassword
111902735643453ba37d873ad500fafa
```

In docker desktop we see Jenkins container and image



Access Jenkins dashboard with <http://localhost:8080>

The first screenshot shows the Jenkins 'Getting Started' page with the title 'Unlock Jenkins'. It instructs the user to find the initial administrator password in the log file `/var/jenkins_home/secrets/initialAdminPassword`. A password input field is provided, and a 'Continue' button is at the bottom right.

The second screenshot shows the 'Getting Started' page with the title 'Create First Admin User'. It contains the following form fields:

- Username:
- Password:
- Confirm password:
- Full name:
- E-mail address:

At the bottom, there is a 'Skip and continue as admin' link and a 'Save and Continue' button. The Jenkins version 'Jenkins 2.516.1' is displayed in the bottom left corner.

To integrate we have to install maven plugin for better

The image shows two screenshots of the Jenkins web interface. The top screenshot is the 'Manage Jenkins' page, and the bottom screenshot is the 'Available plugins' page.

**Manage Jenkins Page:**

- System Configuration:**
  - System:** Configure global settings and paths.
  - Tools:** Configure tools, their locations and automatic installers.
  - Plugins:** Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
  - Nodes:** Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
  - Clouds:** Add, remove, and configure cloud instances to provision agents on-demand.
  - Appearance:** Configure the look and feel of Jenkins.
- Security:**
  - Security:** Secure Jenkins; define who is allowed to access/use the system.
  - Credentials:** Configure credentials.
  - Credential Providers:** Configure the credential providers and types.
  - Users:** [localhost8080/manage/pluginManager](#) [fy](#) users that can log in to this Jenkins.

**Available plugins Page:**

Search:

Install	Name	Released	Health
<input checked="" type="checkbox"/>	<a href="#">Maven Integration</a> 3.27 <a href="#">Build Tools</a> This plugin provides a deep integration between Jenkins and Maven. It adds support for automatic triggers between projects depending on SNAPSHOTS as well as the automated configuration of various Jenkins publishers such as Junit.	3 days 10 hr ago	100
<input checked="" type="checkbox"/>	<a href="#">Pipeline Maven Integration</a> 1563.v2e75645a_7d43 <a href="#">pipeline</a> <a href="#">Maven</a> This plugin provides integration with Pipeline, configures maven environment to use within a pipeline job by calling sh mvn or bat mvn. The selected maven installation will be configured and prepended to the path.	3 days 22 hr ago	99
<input checked="" type="checkbox"/>	<a href="#">Pipeline Maven Plugin API</a> 1563.v2e75645a_7d43 <a href="#">pipeline</a> <a href="#">Maven</a> Pipeline Maven Plugin API	3 days 22 hr ago	99
<input checked="" type="checkbox"/>	<a href="#">Pipeline: Stage View</a> 2.38 <a href="#">User interface</a> Pipeline Stage View Plugin.	3 mo 17 days ago	99
<input type="checkbox"/>	<a href="#">Pipeline Aggregator View</a> 119.v4b_ec11953552 <a href="#">User interface</a>		

Configure Java in Jenkins:

Click on tools in manage Jenkins



The image shows two screenshots of the Jenkins web interface. The top screenshot displays the 'Manage Jenkins' page with a warning banner about security on the built-in node. Below the banner, the 'System Configuration' section is visible, containing links for System, Tools, Plugins, Nodes, Clouds, Appearance, and Managed files. The 'Security' section at the bottom includes links for Security, Credentials, and Credential Providers. The bottom screenshot shows the 'Tools' configuration page, specifically the 'JDK installations' section. It features an 'Add JDK' button and a form for adding a new JDK. The form has a 'Name' field with the value 'Temurin-11' and a checked 'Install automatically' checkbox. Below the form is another 'Add JDK' button. The 'Git installations' section is partially visible at the bottom, with 'Save' and 'Apply' buttons.

**Manage Jenkins**

Building on the built-in node can be a security issue. You should set up distributed builds. See the [documentation](#). [Set up agent](#) [Set up cloud](#) [Dismiss](#)

**System Configuration**

- System**  
Configure global settings and paths.
- Tools**  
Configure tools, their locations and automatic installers.
- Plugins**  
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
- Nodes**  
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
- Clouds**  
Add, remove, and configure cloud instances to provision agents on-demand.
- Appearance**  
Configure the look and feel of Jenkins
- Managed files**  
e.g. settings.xml for maven, central managed scripts, custom files, ...

**Security**

- Security**  
Configure who is allowed to access/use the Jenkins instance
- Credentials**  
Configure credentials
- Credential Providers**  
Configure the credential providers and types

**JDK installations**

[Add JDK](#)

**JDK**

Name  
Temurin-11

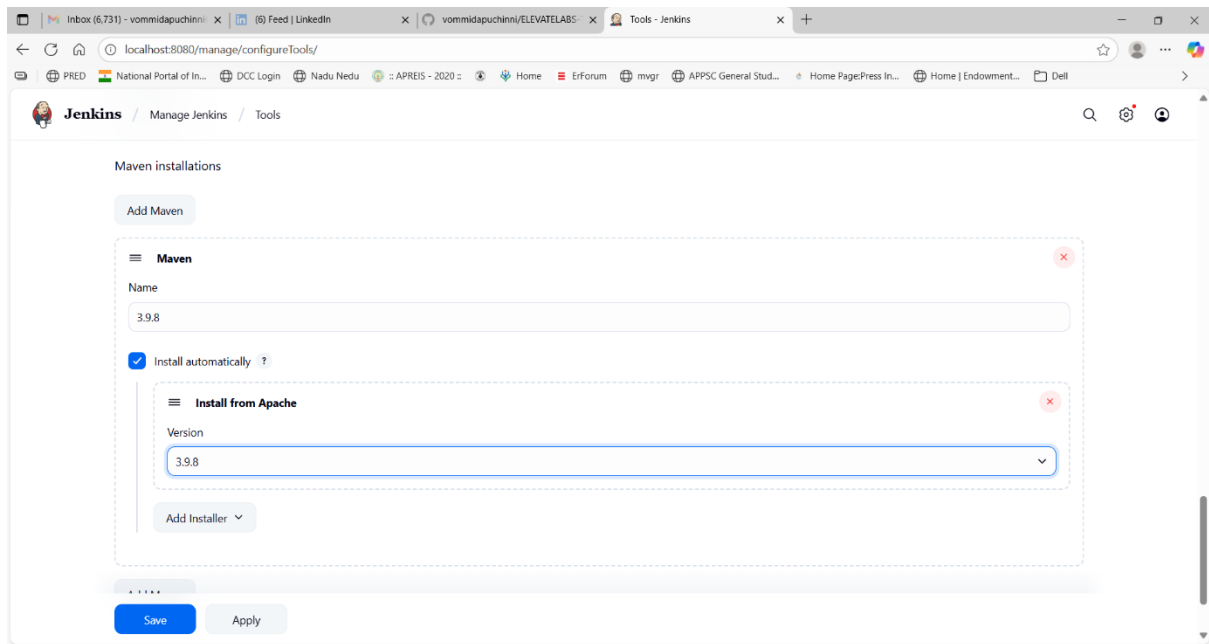
☒ Install automatically ?  
[Add Installer](#)

[Add JDK](#)

**Git installations**

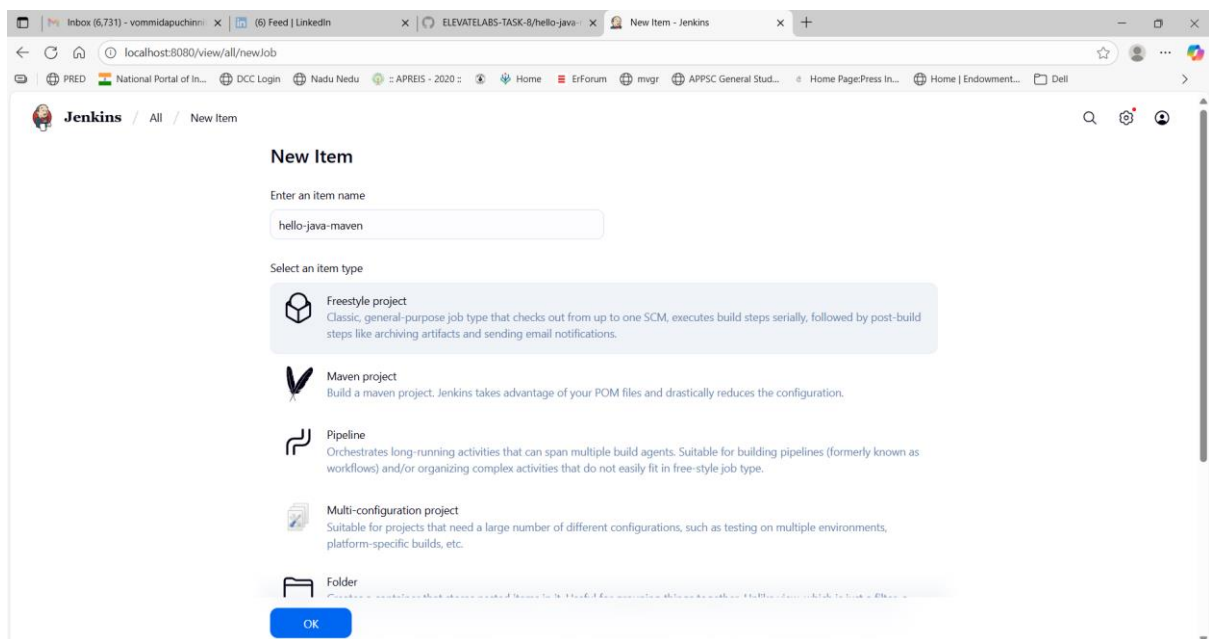
[Save](#) [Apply](#)

- Add Maven → Name: 3.9.8
- Tick Install Automatically



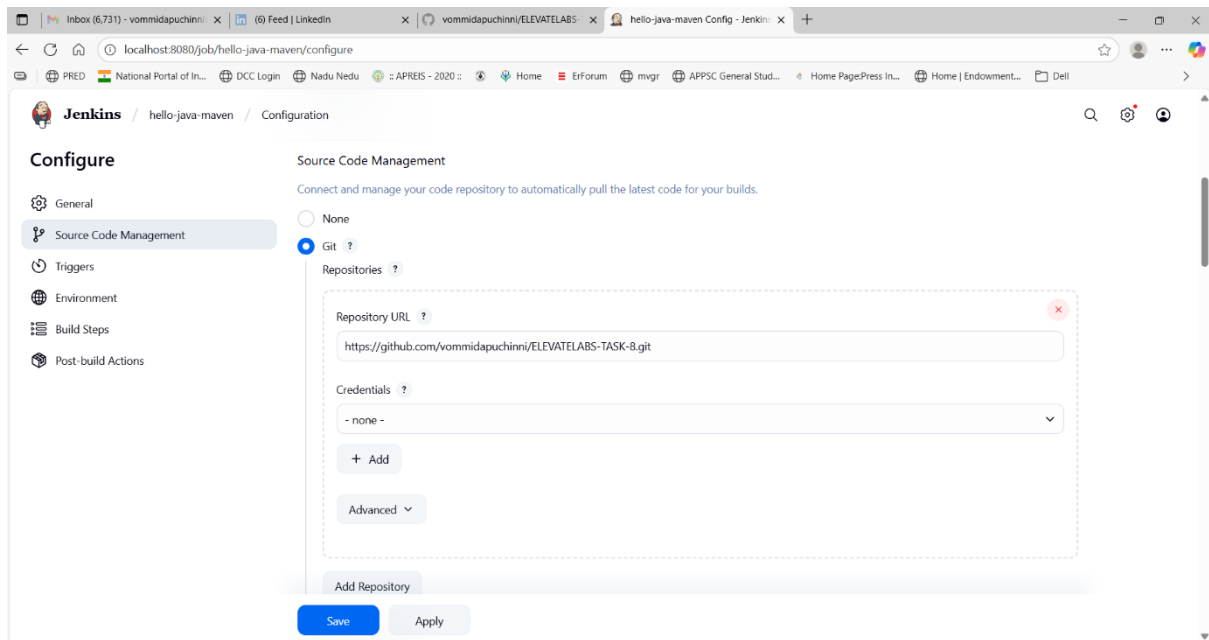
## Create Freestyle Job

- New Item → give name → select free style → click ok



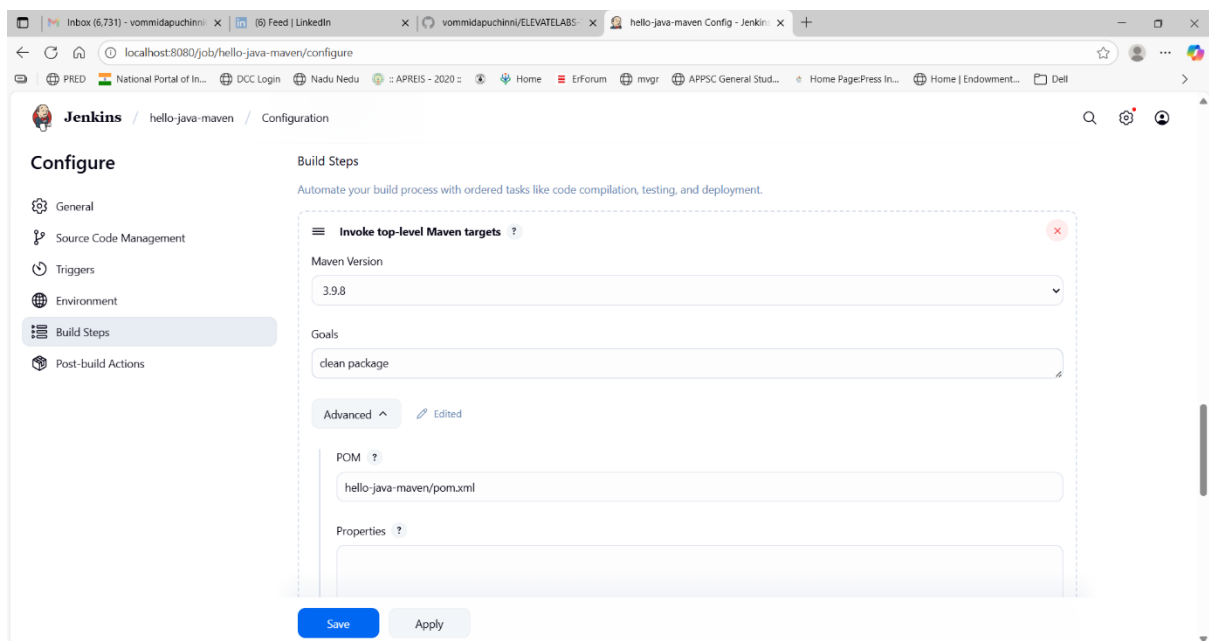
- Source Code Management: GitHub repo URL

Select git → give repo url → its public repo no need of credentials

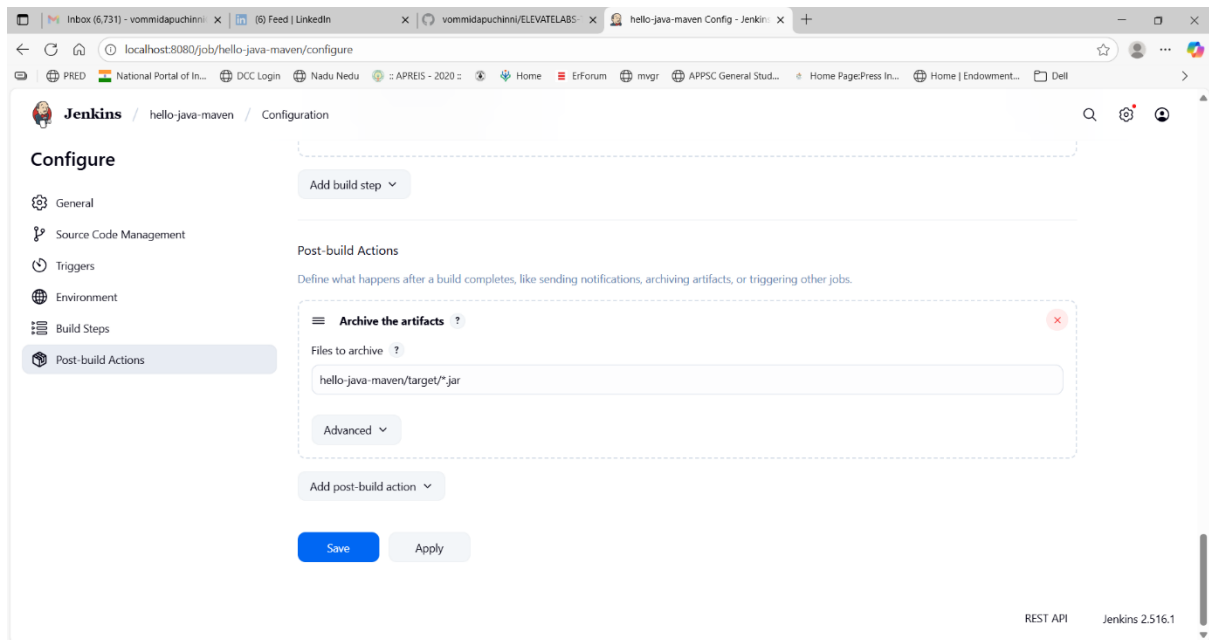


- Build Step: Invoke top-level Maven targets

Keep maven version → goals keep as clean package → mentioning where is our pom file



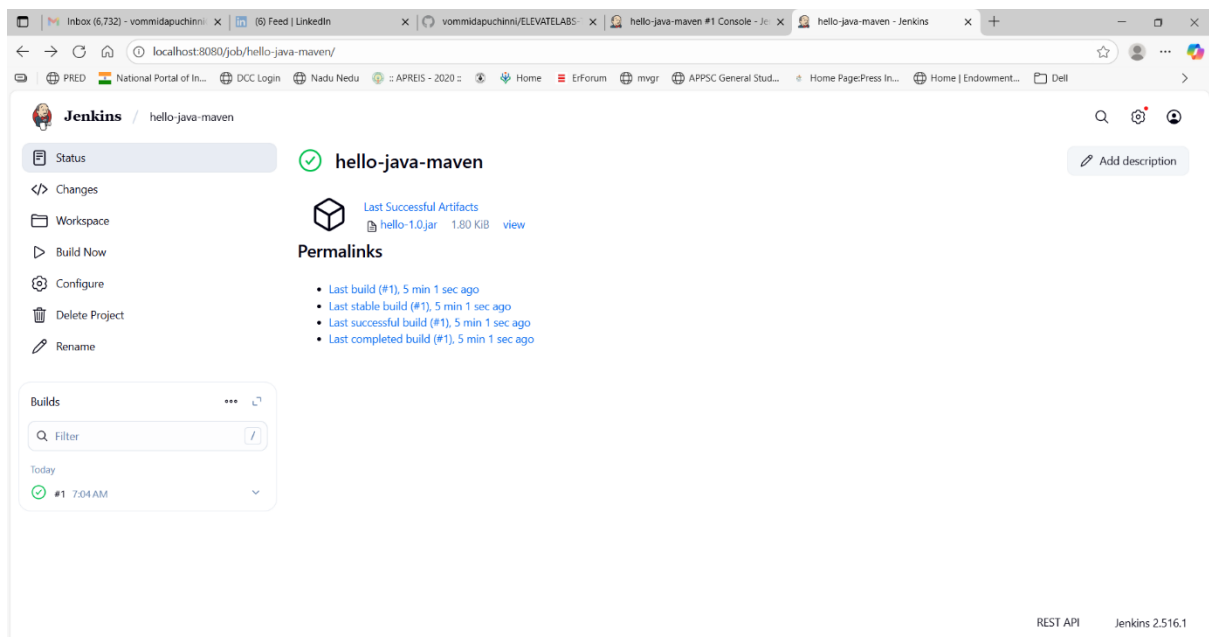
- Goals: clean package
- Added post build steps also artifacts



Click on save

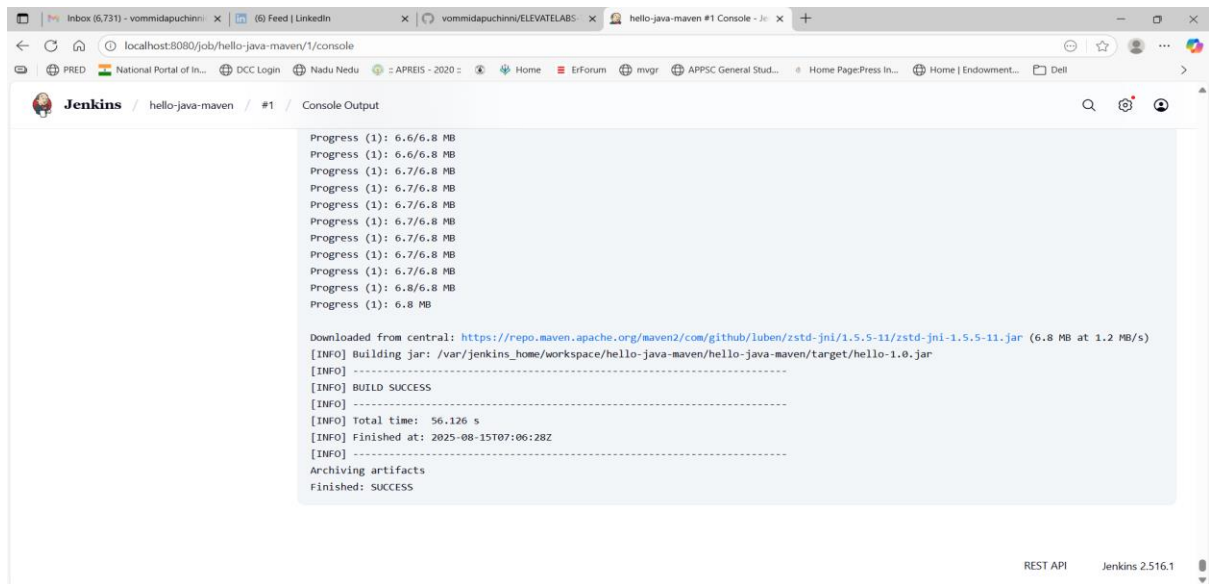
Click on build now

After building we see like this

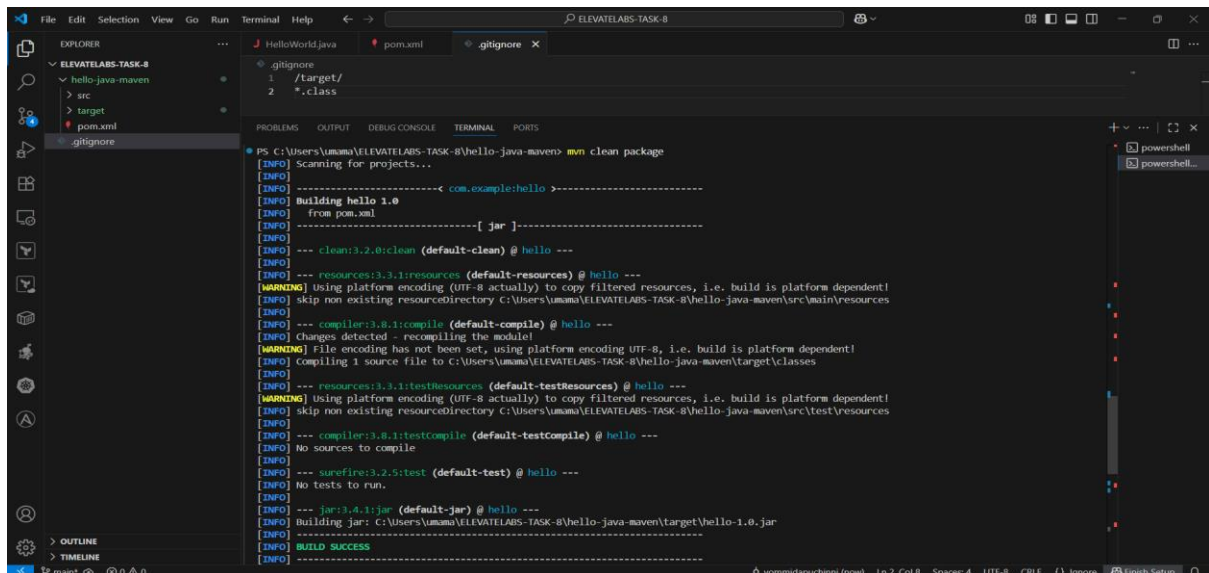


**Verify Build**

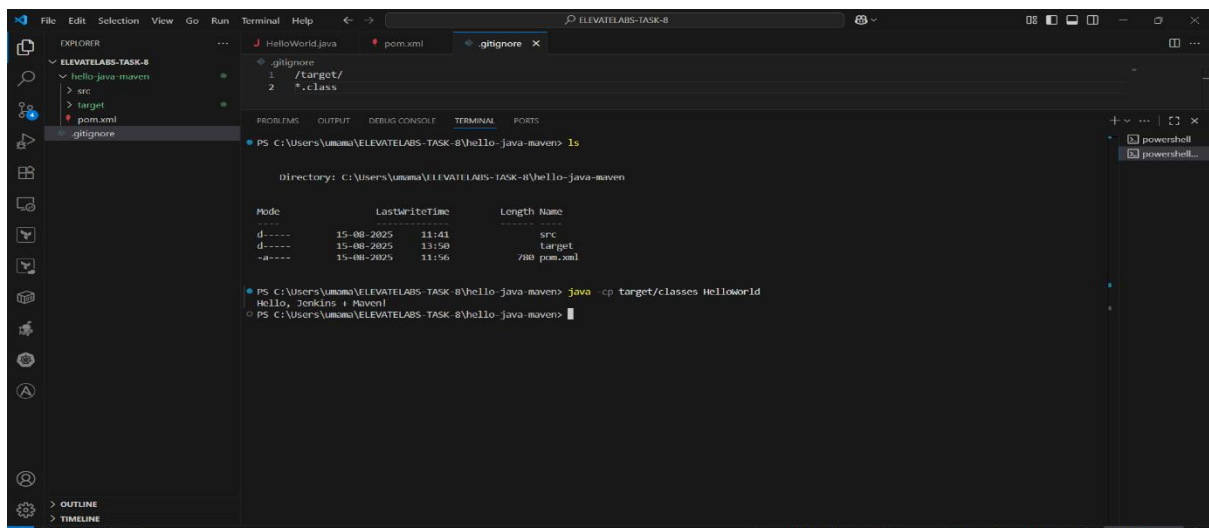
Click on the small icon in build section to see console output



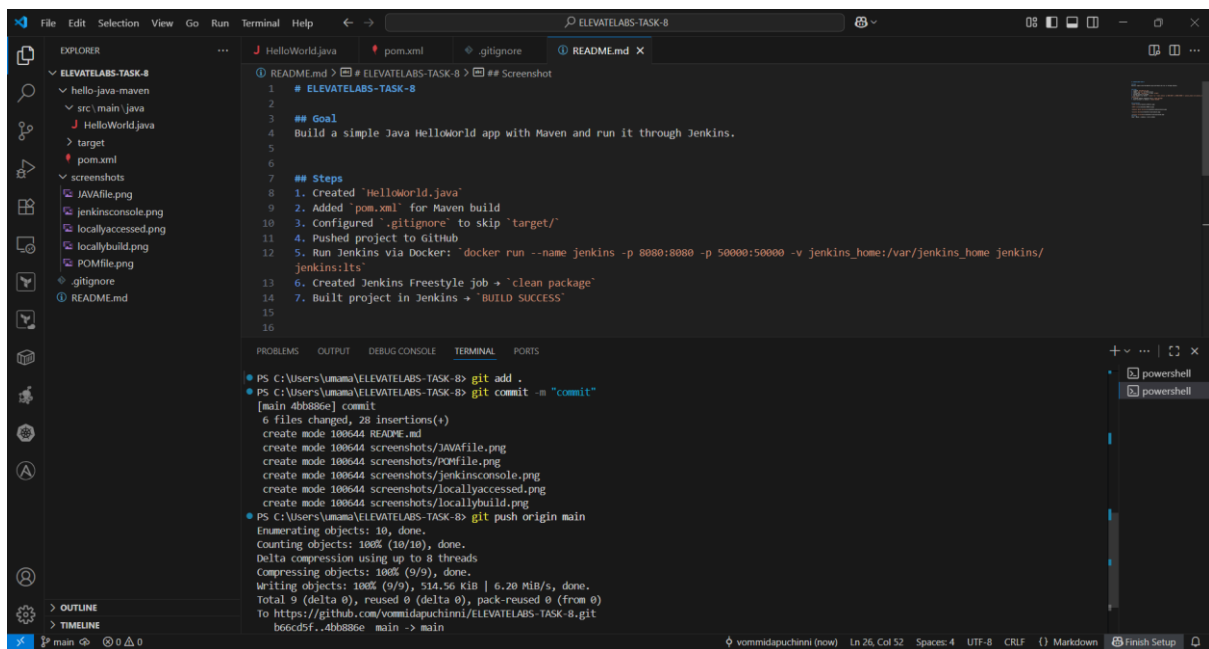
I ran locally



Target file came and we can see the output also



Add readme file and screenshots and keep targets in .gitignore file



## key concepts

- Maven is used to build and manage the Java project.
- pom.xml defines project dependencies, plugins, and build configuration.
- src/main/java contains the source code (HelloWorld.java).
- target/ stores compiled .class files and the generated .jar file.
- .gitignore is used to exclude unnecessary files from Git (like target/).
- Jenkins automates build and deployment using a pipeline.

## Conclusion:

This project demonstrates building a simple Java application using Maven, automating the build with Jenkins, and running the generated .jar file locally. It covers source code organization, dependency management, artifact creation, and version control best practices using .gitignore