

Experiment 06 - Jenkins Pipeline

Roll No.	24
Name	Sreekesh Iyer
Class	D15-A
Subject	DevOps Lab
LO Mapped	<p>LO1: To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meet your business requirements</p> <p>LO3: To understand the importance of Jenkins to Build and deploy Software Applications on server environment</p>

Aim:

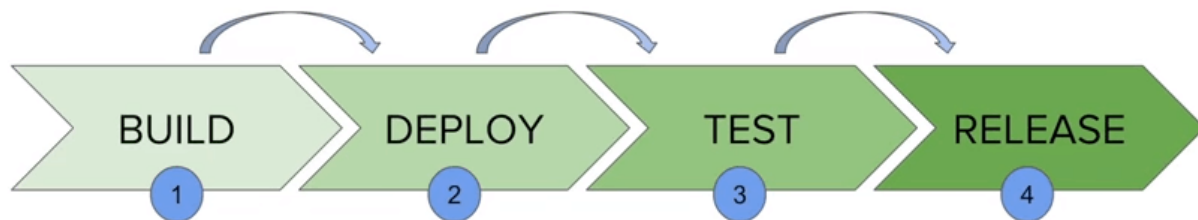
To Build the pipeline of jobs using Maven / Gradle / Ant in Jenkins, create a pipeline script to build and deploy an application over the tomcat server.

Introduction:**Pipeline**

Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a combination of plugins that supports integration and implementation of continuous delivery pipelines. It has an extensible automation server to create simple and complex delivery pipelines as code via pipeline DSL. A Pipeline is a group of events interlinked with each other in a sequence.

Continuous Delivery Pipelines:

In a Jenkins pipeline, every job or event has some sort of dependency on at least one or more events.



The picture above represents a continuous delivery pipeline in Jenkins. It contains a group of states called build, deploy, test and release. These events are interlinked with each other. Every state has its events, which work in a sequence called a continuous delivery pipeline.

A continuous delivery pipeline is an automated expression to display your process for getting software for version control. Thus, every change made in your software goes through a number of complex processes on its way to being released. It also involves developing the software in a reliable and repeatable manner, and progression of the built software through multiple stages of testing and deployment.

Purpose of Pipelining

Jenkins is, fundamentally, an automation engine which supports a number of automation patterns. Pipeline adds a powerful set of automation tools onto Jenkins, supporting use cases that span from simple continuous integration to comprehensive CD pipelines. By modeling a series of related tasks, users can take advantage of the many features of Pipeline:

- Code: Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline.

- Durable: Pipelines can survive both planned and unplanned restarts of the Jenkins controller.
- Pausable: Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.
- Versatile: Pipelines support complex real-world CD requirements, including the ability to fork/join, loop, and perform work in parallel.
- Extensible: The Pipeline plugin supports custom extensions to its DSL and multiple options for integration with other plugins.

While Jenkins has always allowed rudimentary forms of chaining Freestyle Jobs together to perform sequential tasks, Pipeline makes this concept a first-class citizen in Jenkins.

Building on the core Jenkins value of extensibility, Pipeline is also extensible both by users with Pipeline Shared Libraries and by plugin developers.

Advantages Of Jenkins Pipeline

The highlight of Jenkins pipeline is that it offers the feature to define the complete deployment flow-through configuration and code. It states that all the standard Jenkins jobs can be written manually as an entire script and can be managed with a version control system.

It's essentially following the discipline of 'pipeline as code.' Hence instead of creating multiple jobs for each process, it allows us to code the entire workflow and place it in a Jenkins file. Below are some of the reasons that one might consider before using the Jenkins pipeline for Jenkins test automation with Selenium.

- By using Groovy DSL (Domain Specific Language), it models easy to complex pipelines as code.
- The code is stored in the form of a text file called 'Jenkinsfile' that can be scanned into Source Code Management.
- It supports complex pipelines by adding conditional loops, forks, or joining operations and allowing parallel execution tasks.
- It improves user experience by integrating user feedback into the pipeline.
- It's resilient in terms of Jenkins' master unplanned restart.
- It can resume from checkpoints saved.
- It can incorporate multiple additional plugins and add-ins.

Build Tool:

Build tools are programs that automate the creation of executable applications from source code. Building incorporates compiling, linking and packaging the code into a usable or executable form. In small projects, developers will often manually invoke the build process. This is not practical for larger projects, where it is very hard to keep track of what needs to be built, in what sequence and what dependencies there are in the building process. Using an automation tool allows the build process to be more consistent.

A build tool takes care of everything for building a process. It does following:

1. Generates source code (if auto-generated code is used)
2. Generates documentation from source code
3. Compiles source code
4. Packages compiled code into JAR or ZIP file
5. Installs the packaged code in local repository, server repository, or central repository

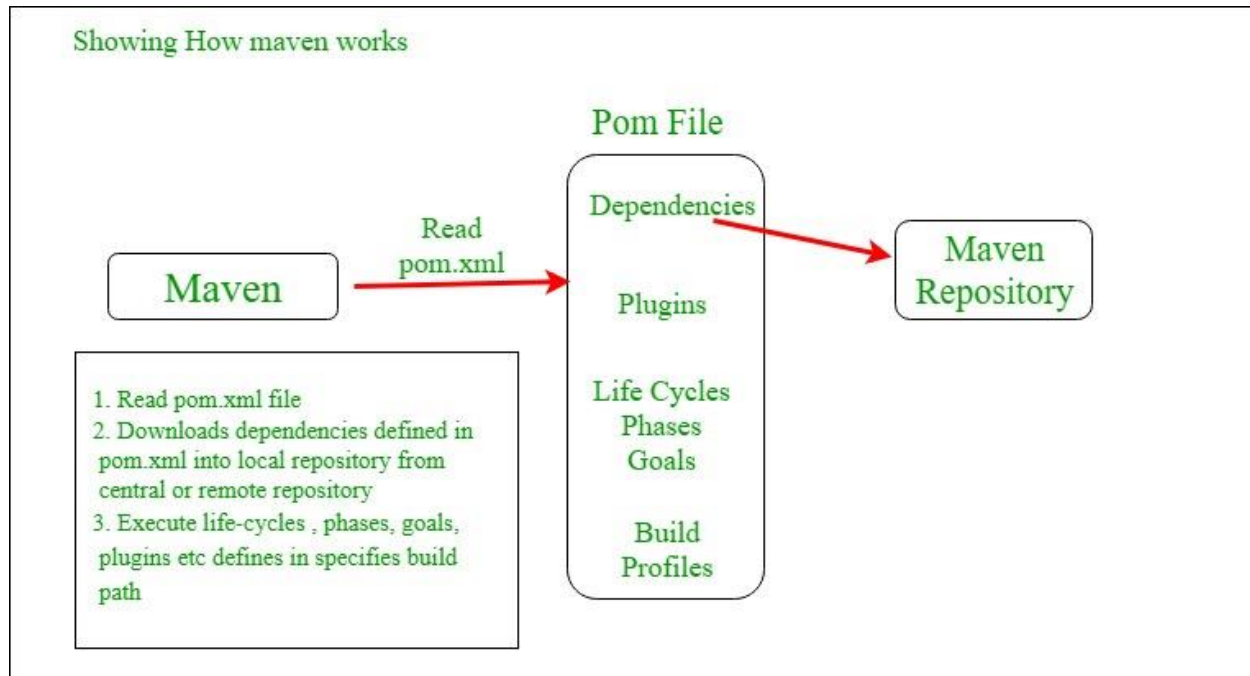
Maven | A build automation tool for Java projects:

Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation. It simplifies the build process like ANT. But it is much more advanced than ANT.

In short terms we can tell maven is a tool that can be used for building and managing any Java-based project. maven makes the day-to-day work of Java developers easier and generally helps with the comprehension of any Java-based project.

Maven does a lot of helpful task like:

1. We can easily build a project using maven.
2. We can add jars and other dependencies of the project easily using the help of maven.
3. Maven provides project information (log document, dependency list, unit test reports etc.)
4. Maven is very helpful for a project while updating the central repository of JARs and other dependencies.
5. With the help of Maven we can build any number of projects into output types like the JAR, WAR etc without doing any scripting.
6. Using maven we can easily integrate our project with source control systems (such as Subversion or Git).



Core Concepts of Maven:

1. POM Files:

Project Object Model(POM) Files are XML files that contain information related to the project and configuration information such as dependencies, source directory, plugin, goals etc. used by Maven to build the project. When you should execute a maven command you give maven a POM file to execute the commands. Maven reads the pom.xml file to accomplish its configuration and operations.

2. Dependencies and Repositories:

Dependencies are external Java libraries required for Project and repositories are directories of packaged JAR files. The local repository is just a directory on your machine hard drive. If the dependencies are not found in the local Maven repository, Maven downloads them from a central Maven repository and puts them in your local repository.

3. Build Life Cycles, Phases and Goals:

A build life cycle consists of a sequence of build phases, and each build phase consists of a sequence of goals. Maven command is the name of a build lifecycle, phase or goal. If a lifecycle is requested to be executed by giving maven command, all build phases in that life cycle are executed also. If a build phase is requested to be executed, all build phases before it in the defined sequence are executed too.

4. Build Profiles:

Build profiles a set of configuration values that allows you to build your project using different configurations. For example, you may need to build your project for your local computer, for development and test. To enable different builds you can add different

build profiles to your POM files using its profile elements and are triggered in a variety of ways.

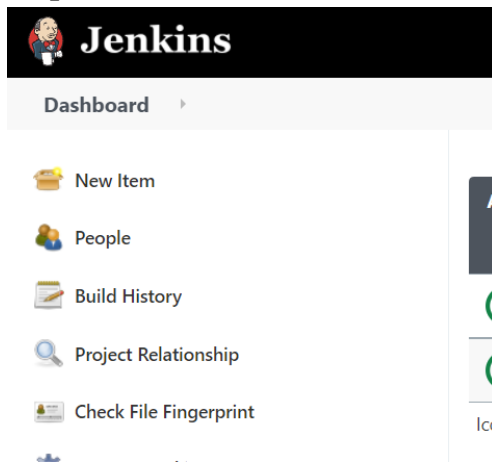
5. Build Plugins:

Build plugins are used to perform specific goals. you can add a plugin to the POM file. Maven has some standard plugins you can use, and you can also implement your own in Java.

Pipeline Jobs:

Creating a Jenkins Pipeline using Maven:

Step 1: In the Jenkins Dashboard, click on New Item from the left menu.



Step 2: Enter a name for your project and choose Pipeline as the project type.

Enter an item name

MyPipeline1
» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

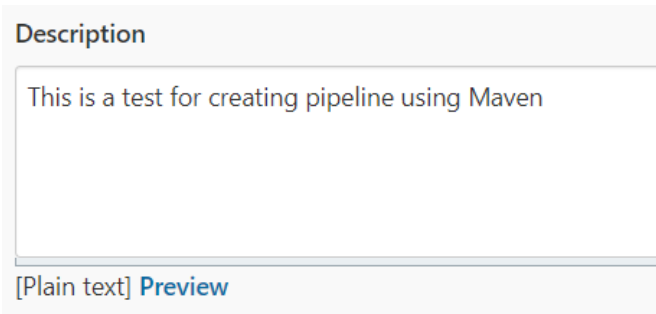
Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

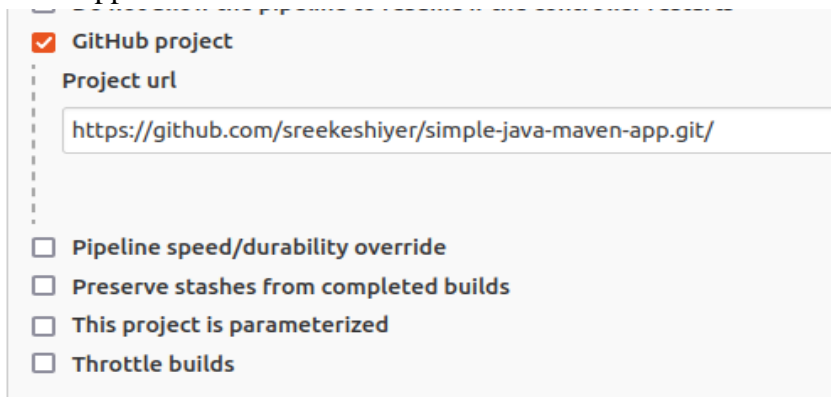
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a namespace, so you can have multiple things of the same name as long as they are in different folders.

OK

Step 3: Enter a description for your project

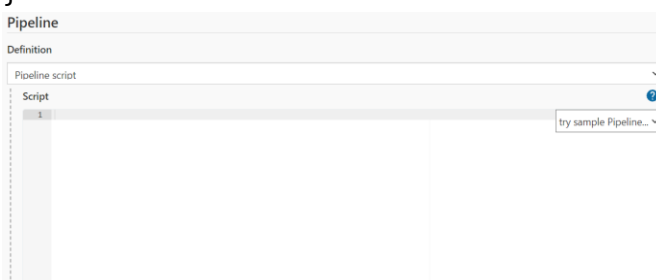
The screenshot shows the 'Description' field in the Jenkins project configuration. The text 'This is a test for creating pipeline using Maven' is entered. Below the text area, there are links for '[Plain text]' and 'Preview'.

Step 4: Check the option of Github Project, and enter the URL of your repository which has a maven app.

The screenshot shows the 'Project url' field in the Jenkins project configuration. The URL 'https://github.com/sreekeshiyer/simple-java-maven-app.git/' is entered. Below the URL field, there are several checkboxes: 'Pipeline speed/durability override', 'Preserve stashes from completed builds', 'This project is parameterized', and 'Throttle builds'.

Step 5: Under the Pipeline section, select Pipeline script option from the drop down and paste the following code in the textbox:

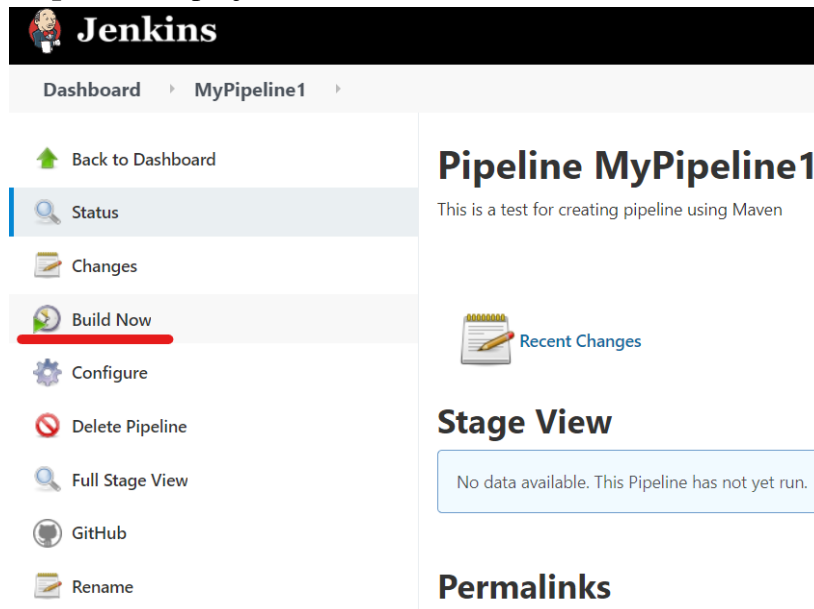
```
pipeline {  
    agent any  
    stages {  
        stage('Hello') {  
            steps {  
                echo 'Hello World'  
            }  
        }  
    }  
}
```



The screenshot shows the Jenkins Pipeline script editor. The 'Pipeline script' option is selected in the dropdown menu. The script content is pasted into the text area. A 'try sample Pipeline...' button is visible in the top right corner of the editor.

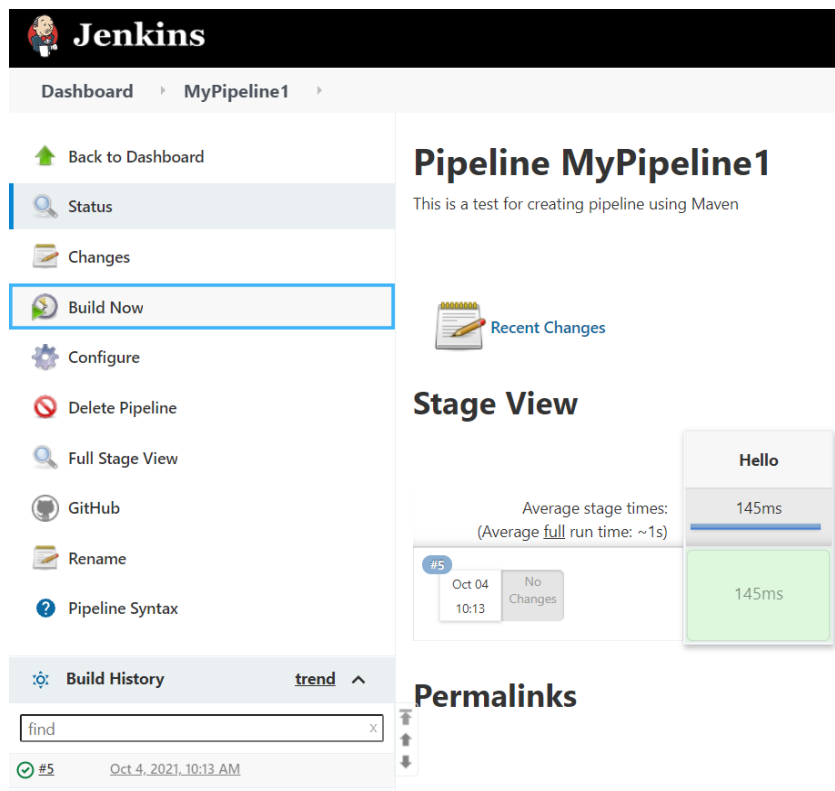
Step 6: Click on Apply and then Save.

Step 7: In the project dashboard, click on Build Now from the left panel.



The screenshot shows the Jenkins interface for a pipeline named 'MyPipeline1'. The left sidebar contains a menu with options: Back to Dashboard, Status, Changes, Build Now (highlighted with a red bar), Configure, Delete Pipeline, Full Stage View, GitHub, and Rename. The main content area has a header 'Pipeline MyPipeline1' with a subtitle 'This is a test for creating pipeline using Maven'. Below this is a 'Recent Changes' section with a notepad icon. The 'Stage View' section displays a message: 'No data available. This Pipeline has not yet run.' The 'Permalinks' section is also visible at the bottom.

Step 8: When the build process is completed, we get the following output in the Stage View section.



The screenshot shows the Jenkins interface for 'MyPipeline1' after a successful build. The left sidebar menu is the same, but 'Build Now' is no longer highlighted. The 'Stage View' section now displays build data. It shows 'Average stage times: (Average full run time: ~1s)' and a table with the following data:

Stage	Time
Hello	145ms

Below the table, a build entry is shown: '#5 Oct 04 10:13 No Changes' with a green status icon. The 'Permalinks' section is also visible at the bottom.

Step 9: Click on the build number and then click on the Console Output option.

Console Output

```
Started by user Sreekesh Iyer
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/MyPipeline1
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Hello)
[Pipeline] echo
Hello World I am Sreekesh and this is my pipeline.
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```
















Installing and setting up Tomcat server

Step 1: Go to <https://tomcat.apache.org> and download Tomcat. Extract all the files to a relevant folder in your system.

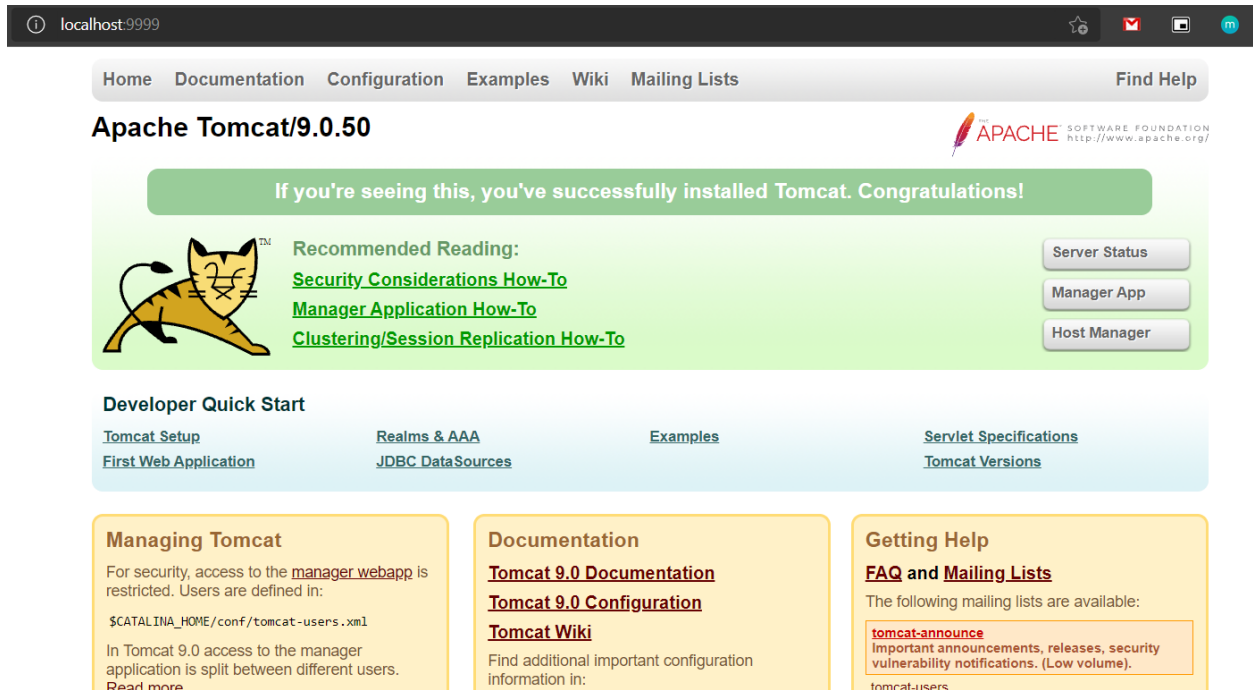
Step 2: Change the port number in conf/server.xml from 8080 to 9999.

```
<Connector port="9999" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" />
```

Step 3: Go to the tomcat directory. Go to the bin folder, and run startup.bat or startup.sh.

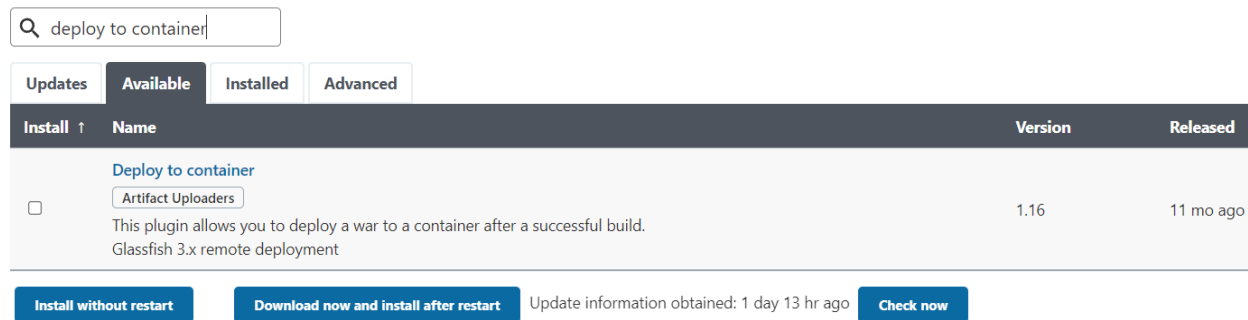
Program Files > Apache Software Foundation > Tomcat 9.0 > bin			
Name	Date modified	Type	Size
 bootstrap	28-06-2021 02:16 PM	Executable Jar File	34 KB
 catalina	28-06-2021 02:16 PM	Windows Batch File	17 KB
 ciphers	28-06-2021 02:16 PM	Windows Batch File	3 KB
 configtest	28-06-2021 02:16 PM	Windows Batch File	2 KB
 digest	28-06-2021 02:16 PM	Windows Batch File	3 KB
 makebase	28-06-2021 02:16 PM	Windows Batch File	4 KB
 service	28-06-2021 02:16 PM	Windows Batch File	9 KB
 setclasspath	28-06-2021 02:16 PM	Windows Batch File	4 KB
 shutdown	28-06-2021 02:16 PM	Windows Batch File	2 KB
 startup	28-06-2021 02:16 PM	Windows Batch File	2 KB
 Tomcat9	28-06-2021 02:16 PM	Application	128 KB
 Tomcat9w	28-06-2021 02:16 PM	Application	118 KB
 tomcat-juli	28-06-2021 02:16 PM	Executable Jar File	46 KB
 tool-wrapper	28-06-2021 02:16 PM	Windows Batch File	5 KB
 version	28-06-2021 02:16 PM	Windows Batch File	2 KB

Step 4: Open your browser and visit <http://localhost:9999>. We can also access the Tomcat home page by using the IP address <http://<IP address>:9999>.



Deploying a Maven from Jenkins to Tomcat

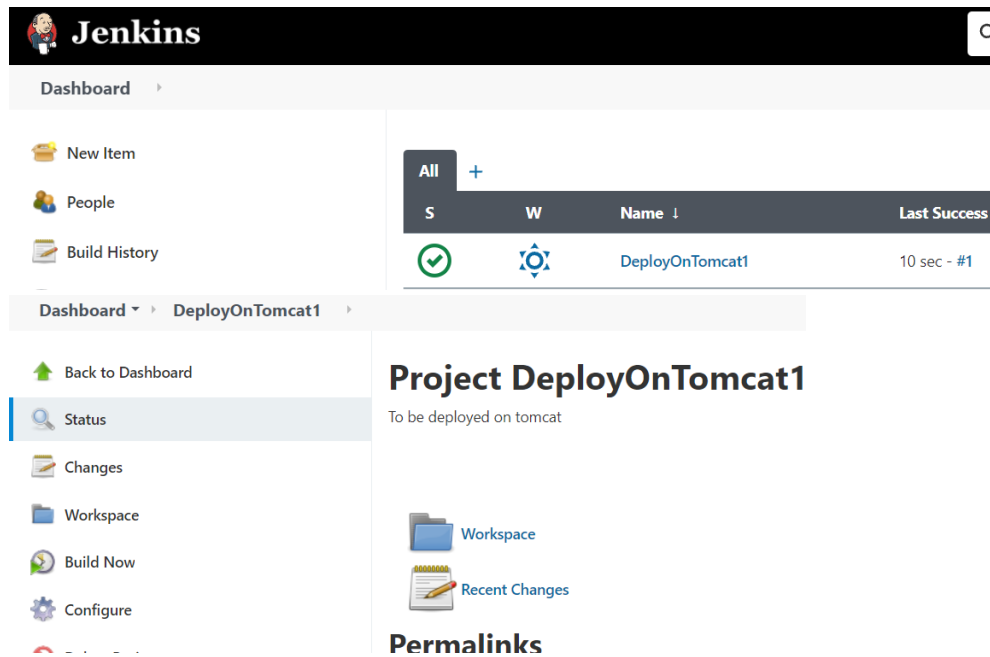
Step 1: On the Jenkins dashboard, go to the Manage Jenkins link and then click on Manage Plugins and install the Deploy to container plugin, by clicking on Install without restart. Wait until the installation of Deploy Plugin is complete.



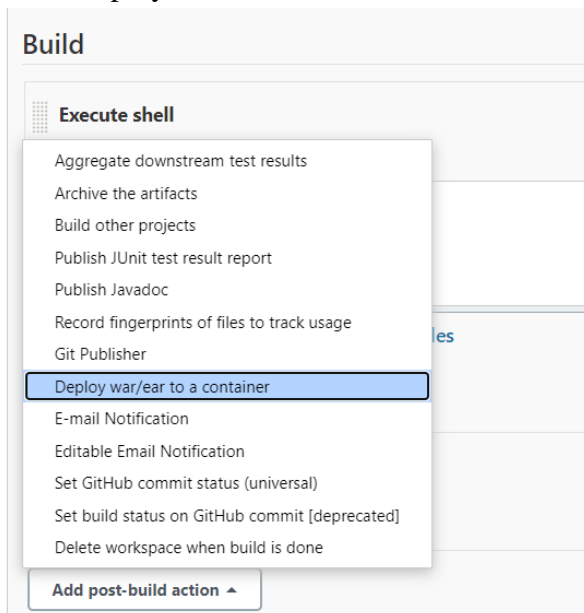
Install	Name	Version	Released
<input type="checkbox"/>	Deploy to container Artifact Uploaders This plugin allows you to deploy a war to a container after a successful build. Glassfish 3.x remote deployment	1.16	11 mo ago

Step 2: Create a Maven build job for a project which generates a war file (instead of jar), as configured in its pom.xml file

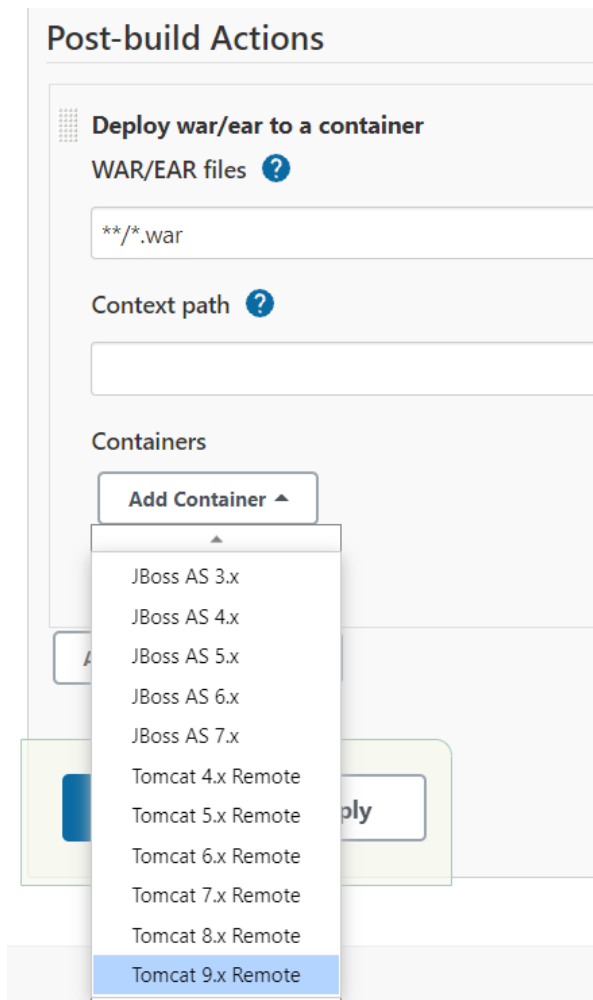
Step 3: Go to the Jenkins dashboard and select any Maven build job. Click on the Configure link of the selected build job.



Step 4: Click on the Add post-build action button on the configuration page of the relevant job and select Deploy war/ear to container, as shown in the following figure.



Step 5: It will add Deploy war/ear to a container in the Post-build Actions section. Provide a war file path that is relative to the workspace, and select Tomcat 9.x as the container from the available list box, as shown in the following figure.



Step 6: Add the following in the uncommented section:

```
<user username="ScriptAdmin" password="pass1" roles="manager-script"/>
<user username="TomcatAdmin" password="pass1" roles="manager-gui"/>
```

Restart Tomcat, visit <http://localhost:9999/manager/html>, and enter a username and password of manager-gui.

Step 7: Use the same username and password in Jenkins for Manager credentials, as given for manager-script role.

Add Credentials

Domain

Global credentials (unrestricted)

Kind

Username with password

Scope

Global (Jenkins, nodes, items, all child items, etc)

Username

ScriptAdmin


☐ Treat username as secret


Password

.....


ID

Post-build Actions

 **Deploy war/ear to a container**


WAR/EAR files 

**/*.war




Context path 


sample1

Containers

 **Tomcat 9.x Remote**

Credentials

ScriptAdmin/*****   Add 

Tomcat URL 

http://localhost:9999

Step 8: Click on Build Now. Click on the build number to view the Console Output.

Console Output

```

Started by user Sreekesh Iyer
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/MyFirstMavenJob
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/MyFirstMavenJob/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/sreekeshiyer/webAppExample.git # timeout=10
Fetching upstream changes from https://github.com/sreekeshiyer/webAppExample.git
> git --version # timeout=10
> git --version # 'git version 2.25.1'
> git fetch --tags --force --progress -- https://github.com/sreekeshiyer/webAppExample.git +re
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision b6b526a1377a021415f6b41de7e427b6e88hd8AA (refs/remotes/origin/master)
[INFO] Webapp assembled in [40 ms]
[INFO] Building war: /var/lib/jenkins/workspace/MyFirstMavenJob/target/webappExample.war
[INFO] WEB-INF/web.xml already added, skipping
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.778 s
[INFO] Finished at: 2021-10-05T01:57:13+05:30
[INFO] -----
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving /var/lib/jenkins/workspace/MyFirstMavenJob/pom.xml to com.javarticles.webapp/webappExample/0.0.
[JENKINS] Archiving /var/lib/jenkins/workspace/MyFirstMavenJob/target/webappExample.war to com.javarticles.webapp/w
channel stopped
[DeployPublisher][INFO] Attempting to deploy 1 war file(s)
[DeployPublisher][INFO] Deploying /var/lib/jenkins/workspace/MyFirstMavenJob/target/webappExample.war to container
[ /var/lib/jenkins/workspace/MyFirstMavenJob/target/webappExample.war] is not deployed. Doing a fresh deployment.
Deploying [ /var/lib/jenkins/workspace/MyFirstMavenJob/target/webappExample.war]
Finished: SUCCESS

```




Step 9: Once the build is complete, verify the console output of the deployment of the application in the Tomcat application server

```

04-Oct-2021 13:15:30.604 INFO [http-nio-9999-exec-8] org.apache.catalina.startup.HostConfig.deployWAR Deploying web
application archive [C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps\sample1.war]
04-Oct-2021 13:15:32.426 INFO [http-nio-9999-exec-8] org.apache.jasper.servlet.TldScanner.scanJars At least one JAR
was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete list of JARs that we
re scanned but no TLDs were found in them. Skipping unneeded JARs during scanning can improve startup time and JSP c
ompilation time.
04-Oct-2021 13:15:32.432 INFO [http-nio-9999-exec-8] org.apache.catalina.startup.HostConfig.deployWAR Deployment of
web application archive [C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps\sample1.war] has finished in
[1,827] ms

```

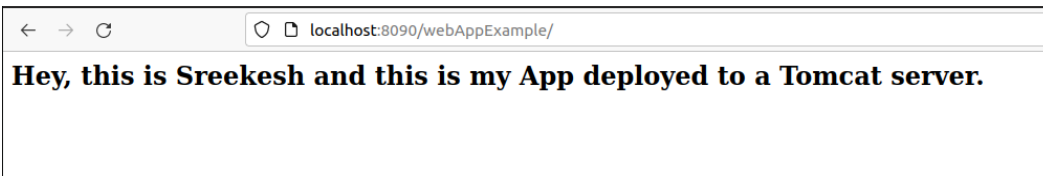
Step 10: Verify the webapps directory in the Tomcat installation directory

 > This PC > OS (C:) > Program Files > Apache Software Foundation > Tomcat 9.0 > webapps >		
Name	Date modified	Type
 sample1.war	04-10-2021 01:15 PM	WAR File
 sample1	04-10-2021 01:15 PM	File folder

Step 11: Verify the Tomcat manager, and check the status of an application in the Tomcat application server.

Applications			
Path	Version	Display Name	Running
/	None specified	Welcome to Tomcat	true
/docs	None specified	Tomcat Documentation	true
/examples	None specified	Servlet and JSP Examples	true
/host-manager	None specified	Tomcat Host Manager Application	true
/manager	None specified	Tomcat Manager Application	true
/webAppExample	None specified	Archetype Created Web Application	true

Deploy



Conclusion

Thus, we have learnt the concept of pipelining and its advantages, along with the build-tools we have used for our projects - Maven. We also learnt the steps to set up Tomcat and deploy our Maven project on it by creating a war file, in our post-build tasks.