Experiment No. 5

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

LO No. & Statement: (LO3): Illustrate the importance of Jenkins to build and deploy Software Applications on a server environment.

Theory:

Jenkins Pipeline:

Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a suite of plugins that supports implementing and integrating continuous delivery pipelines into Jenkins.

A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment.

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the Pipeline domain-specific language (DSL) syntax.

The definition of a Jenkins Pipeline is written into a text file (called a Jenkinsfile) which in turn can be committed to a project's source control repository. This is the foundation of "Pipeline-as-code"; treating the CD pipeline as a part of the application to be versioned and reviewed like any other code.

Creating a Jenkinsfile and committing it to source control provides a number of immediate benefits:

- Automatically creates a Pipeline build process for all branches and pull requests.
- Code review/iteration on the Pipeline (along with the remaining source code).
- Audit trail for the Pipeline.
- Single source of truth for the Pipeline, which can be viewed and edited by multiple members of the project.

While the syntax for defining a Pipeline, either in the web UI or with a Jenkinsfile is the same, it is generally considered best practice to define the Pipeline in a Jenkinsfile and check that into source control.

Declarative versus Scripted Pipeline syntax:

A Jenkinsfile can be written using two types of syntax - Declarative and Scripted.

Declarative and Scripted Pipelines are constructed fundamentally differently. Declarative Pipeline is a more recent feature of Jenkins Pipeline which:

- provides richer syntactical features over Scripted Pipeline syntax, and
- is designed to make writing and reading Pipeline code easier.

Many of the individual syntactical components (or "steps") written into a Jenkinsfile, however, are common to both Declarative and Scripted Pipeline.

Why Pipeline?

Jenkins is, fundamentally, an automation engine that supports a number of automation patterns. Pipeline adds a powerful set of automation tools to 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 [1] 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.

Pipeline concepts

The following concepts are key aspects of Jenkins Pipeline, which tie in closely to Pipeline syntax (see the overview below).

Pipeline

A Pipeline is a user-defined model of a CD pipeline. A Pipeline’s code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.

Also, a pipeline block is a key part of the Declarative Pipeline syntax.

Node

A node is a machine that is part of the Jenkins environment and is capable of executing a Pipeline.

Also, a node block is a key part of Scripted Pipeline syntax.

Stage

A stage block defines a conceptually distinct subset of tasks performed through the entire Pipeline (e.g., "Build", "Test" and "Deploy" stages), which is used by many plugins to visualize or present Jenkins Pipeline status/progress.

Step

A single task. Fundamentally, a step tells Jenkins what to do at a particular point in time (or "step" in the process). For example, to execute the shell command make use of the sh step: sh

'make'. When a plugin extends the Pipeline DSL, that typically means the plugin has implemented a new step.

Pipeline example

Here is an example of a Jenkinsfile using Declarative Pipeline syntax - its Scripted syntax equivalent can be accessed by clicking the Toggle Scripted Pipeline link below:

Jenkinsfile (Declarative Pipeline) pipeline {

agent any options {

skipStagesAfterUnstable()

}

stages {

stage('Build') { steps {

sh 'make'

}

}

stage('Test'){ steps {

sh 'make check'

junit 'reports/**/*.xml'

}

}

stage('Deploy') { steps {

sh 'make publish'

}

}

}

}

The pipeline is Declarative Pipeline-specific syntax that defines a "block" containing all content and instructions for executing the entire Pipeline.

The agent is Declarative Pipeline-specific syntax that instructs Jenkins to allocate an executor (on a node) and workspace for the entire Pipeline.

The stage is a syntax block that describes a stage of this Pipeline. Read more about stage blocks in Declarative Pipeline syntax on the Pipeline syntax page. As mentioned above, stage blocks are optional in Scripted Pipeline syntax.

Steps is a Declarative Pipeline-specific syntax that describes the steps to be run in this stage.

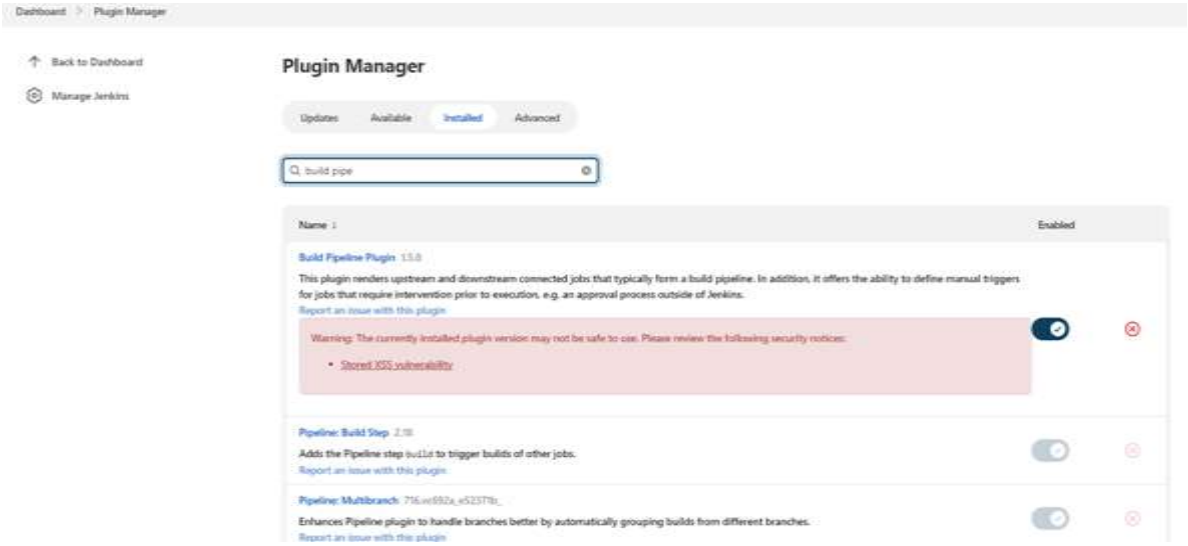
sh is a Pipeline step (provided by the Pipeline: Nodes and Processes plugin) that executes the given shell command.

junit is another Pipeline step (provided by the JUnit plugin) for aggregating test reports.

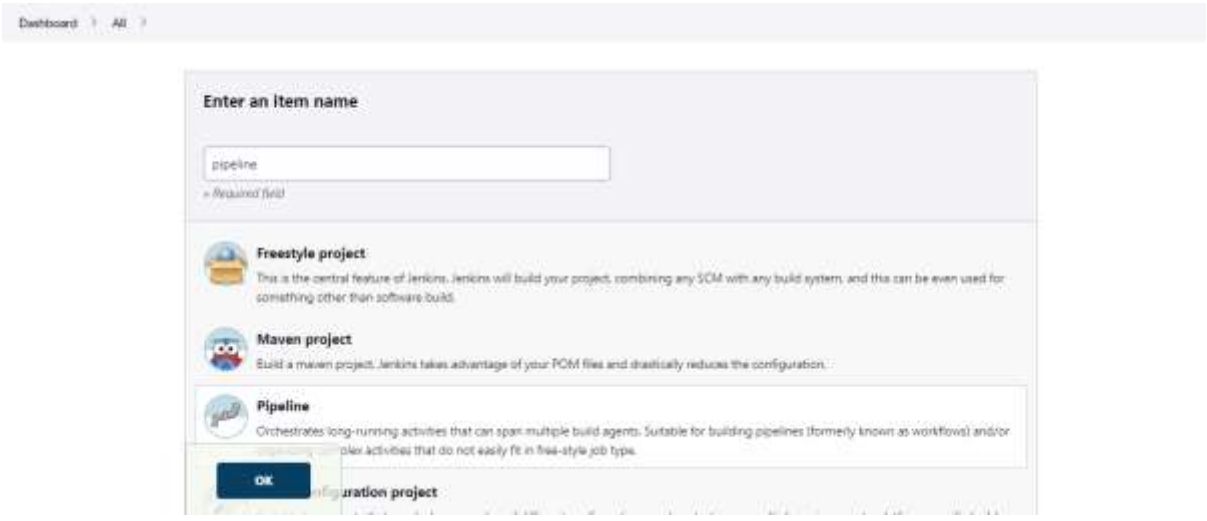
sh is a Pipeline step (provided by the Pipeline: Nodes and Processes plugin) that executes the given shell command.

Steps:

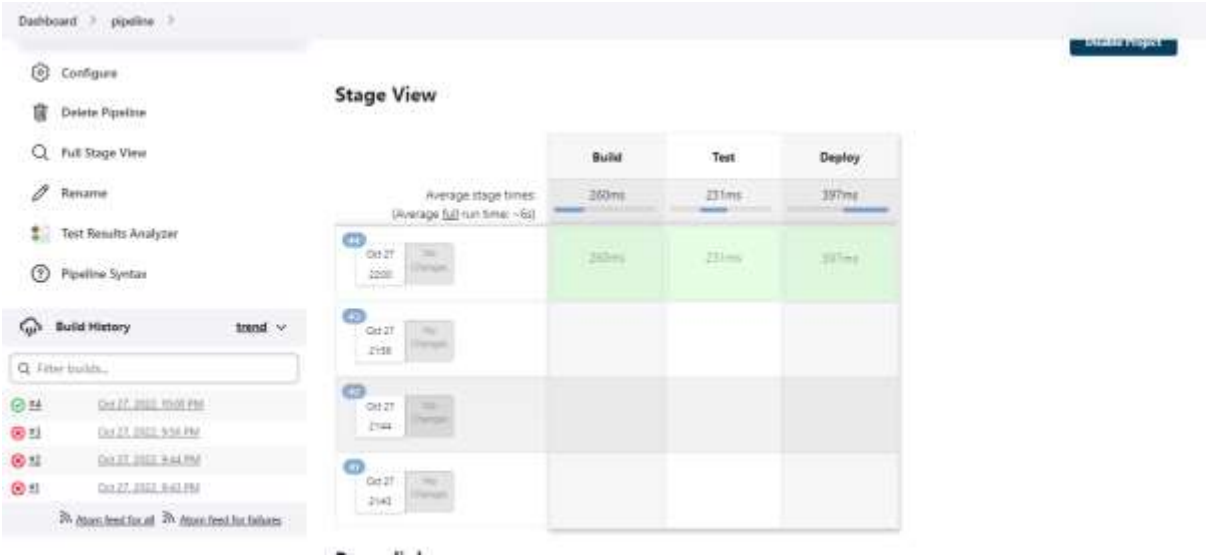
Install the pipeline build plugin:



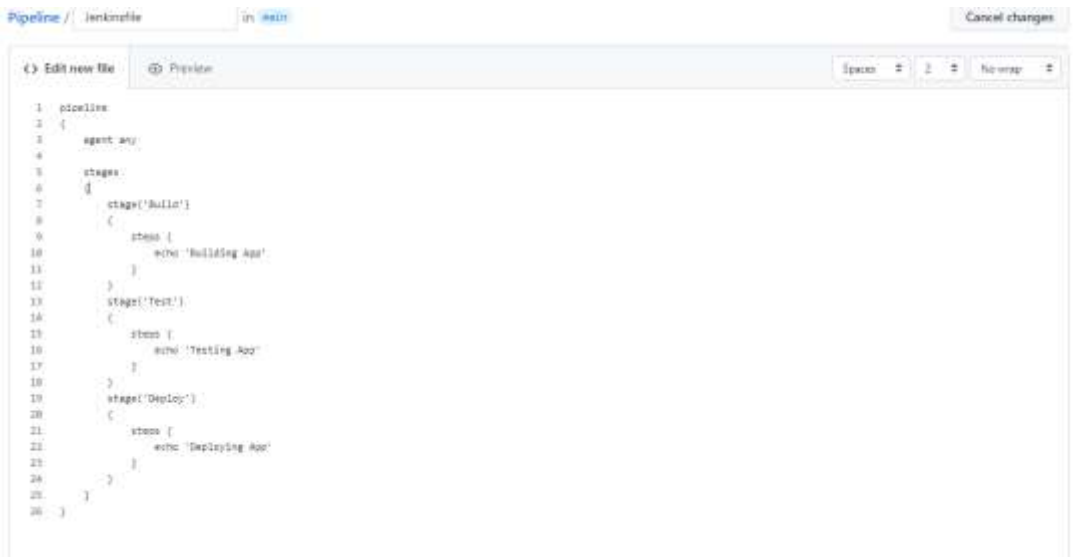
Build a pipeline project

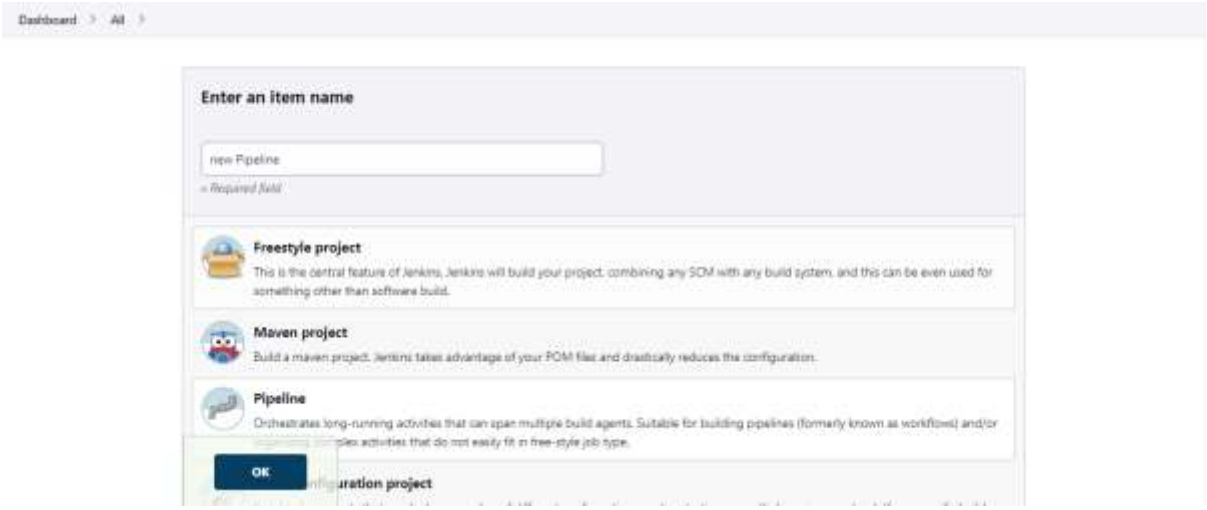


Change the pipeline script and again build the project

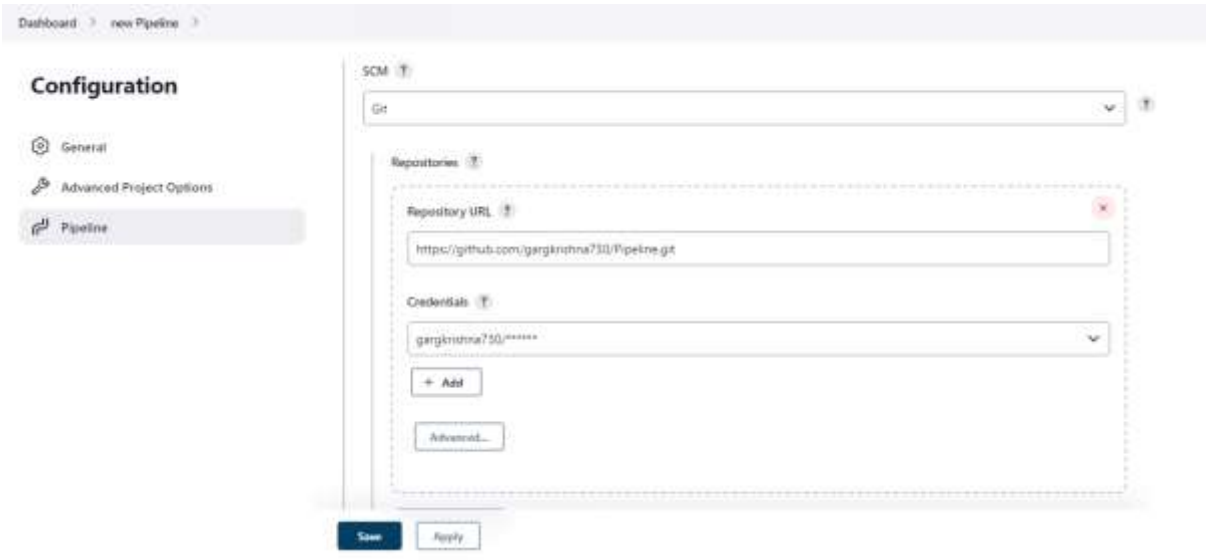


Create Jenkinsfile in GitHub:




Dashboard > All >
Enter an item name
new Pipeline
» 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
OK


Global (Jenkins, nodes, items, all child items, etc)
Username ?
gargkrishna730
Treat username as secret ?
Password ?
ID ?
Description ?


Dashboard > new Pipeline >
Configuration
General
Advanced Project Options
Pipeline
SCM ?
Git
Repositories ?
Repository URL ?
https://github.com/gargkrishna730/Pipeline.git
Credentials ?
+ Add
Advanced...
Save
Apply

Conclusion:

From this experiment, we have learned about the pipeline in Jenkins. We have learned about the simple pipeline script and the SCM script where a simple pipeline script is uploaded elsewhere and linked here in Jenkins. We have implemented two different projects with different scripts.

We have achieved LO3 from this experiment.

We have also achieved Program Outcomes PO1, PO2, PO3, PO4, PO5, PO12.