

The Jenkins logo is a red square with a white border. Inside the square, the word "Jenkins" is written in white, bold, sans-serif font. Below it, the text "CI/CD Pipeline" is written in a smaller, white, sans-serif font.

Jenkins

CI/CD Pipeline

Install Jenkins

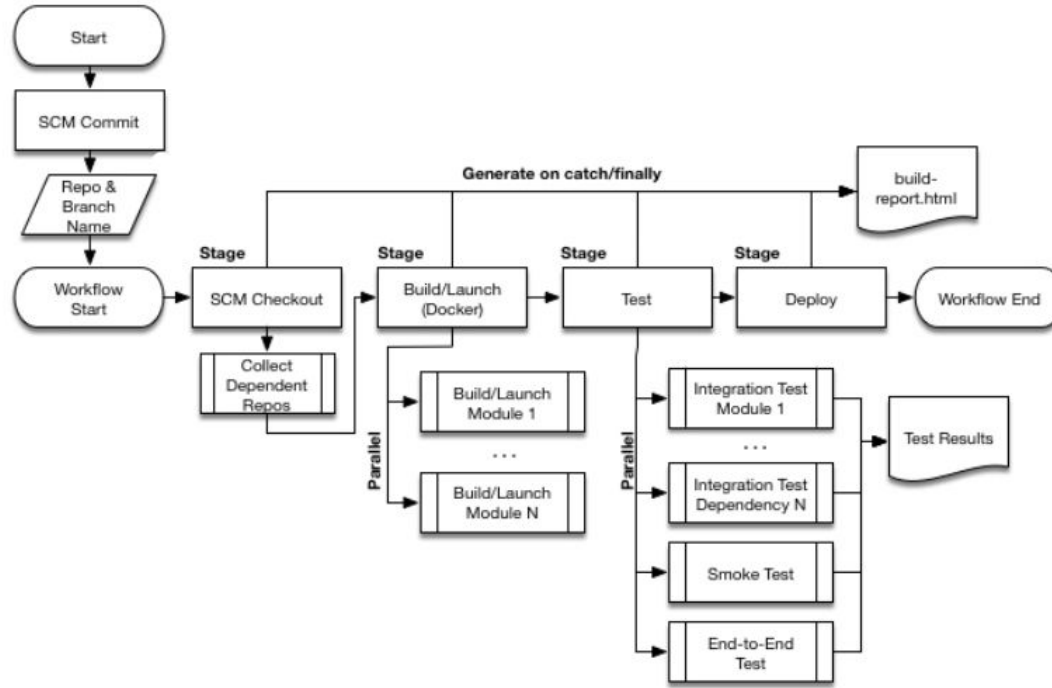
A **LTS (Long-Term Support) release** is chosen every 12 weeks from the stream of regular releases as the stable release for that time period. It can be installed from the **debian-stable apt repository**.

1. `curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \`
`/usr/share/keyrings/jenkins-keyring.asc > /dev/null`
2. `echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \`
`https://pkg.jenkins.io/debian-stable binary/ | sudo tee \`
`/etc/apt/sources.list.d/jenkins.list > /dev/null`
3. `sudo apt-get update`
4. `sudo apt-get install jenkins`

Jenkins Pipeline

- **Jenkins Pipeline (or simply "Pipeline") is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.**
- **A continuous delivery pipeline is an automated expression of your process for getting software from version control right through to your users and customers.**
- **Jenkins Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code". The definition of a Jenkins Pipeline is typically written into a text file (called a Jenkinsfile) which in turn is checked into a project's source control repository.**

Stages of a PipeLine



How Pipelines Can be Built

A Pipeline can be created in one of the following ways:

- **Through Blue Ocean - after setting up a Pipeline project in Blue Ocean, the Blue Ocean UI helps you write your Pipeline's Jenkinsfile and commit it to source control.**
- **Through the classic UI - you can enter a basic Pipeline directly in Jenkins through the classic UI.**
- **In SCM - you can write a Jenkinsfile manually, which you can commit to your project's source control repository. [3]**

The syntax for defining a Pipeline with either approach is the same, but while Jenkins supports entering Pipeline directly into the classic UI, it is generally considered best practice to define the Pipeline in a Jenkinsfile which Jenkins will then load directly from source control.

Running a Docker Pipeline

Jenkinsfile (Declarative Pipeline)

/ Requires the Docker Pipeline plugin */*

```
pipeline {  
    agent { docker { image 'maven:3.9.8-eclipse-temurin-21-alpine' } }  
    stages {  
        stage('build') {  
            steps {  
                sh 'mvn --version'  
            }  
        }  
    }  
}
```

Terraform through Jenkins

Assignment: Create a Git Project that hosts a terraform project to create 2 EC2 instances and shows their publicDNS. Checkout the project through jenkins and run terraform using the jenkins pipeline.

<https://spacelift.io/blog/terraform-jenkins>

Assignment

1. Build a Java app with Maven

a. <https://www.jenkins.io/doc/tutorials/build-a-java-app-with-maven/>

7. You can now click on **Maven tutorial** on the top left, and then on **Stages** at the left. It will list your previous Pipeline runs in reverse chronological order.

The screenshot shows the Jenkins web interface. At the top is a black header with the Jenkins logo, a search bar, and user information (admin, log out). Below the header is a breadcrumb trail: Dashboard > Maven tutorial > Stages. The main content area is titled 'Maven tutorial' and has two buttons: 'Build' (green) and 'Configure' (grey). Below this is a table of pipeline runs. The table has two columns: 'id' and 'pipeline'. There are three rows of pipeline runs, numbered 3, 2, and 1 from top to bottom. Each row shows a sequence of stages: Start, Checkout SCM, Build, Test, Deliver, and End. The 'Build' and 'Test' stages in all runs are marked with green checkmarks, indicating successful completion.

id	pipeline
3	Start → Checkout SCM → Build → Test → Deliver → End
2	Start → Checkout SCM → Build → Test → End
1	Start → Checkout SCM → Build → End