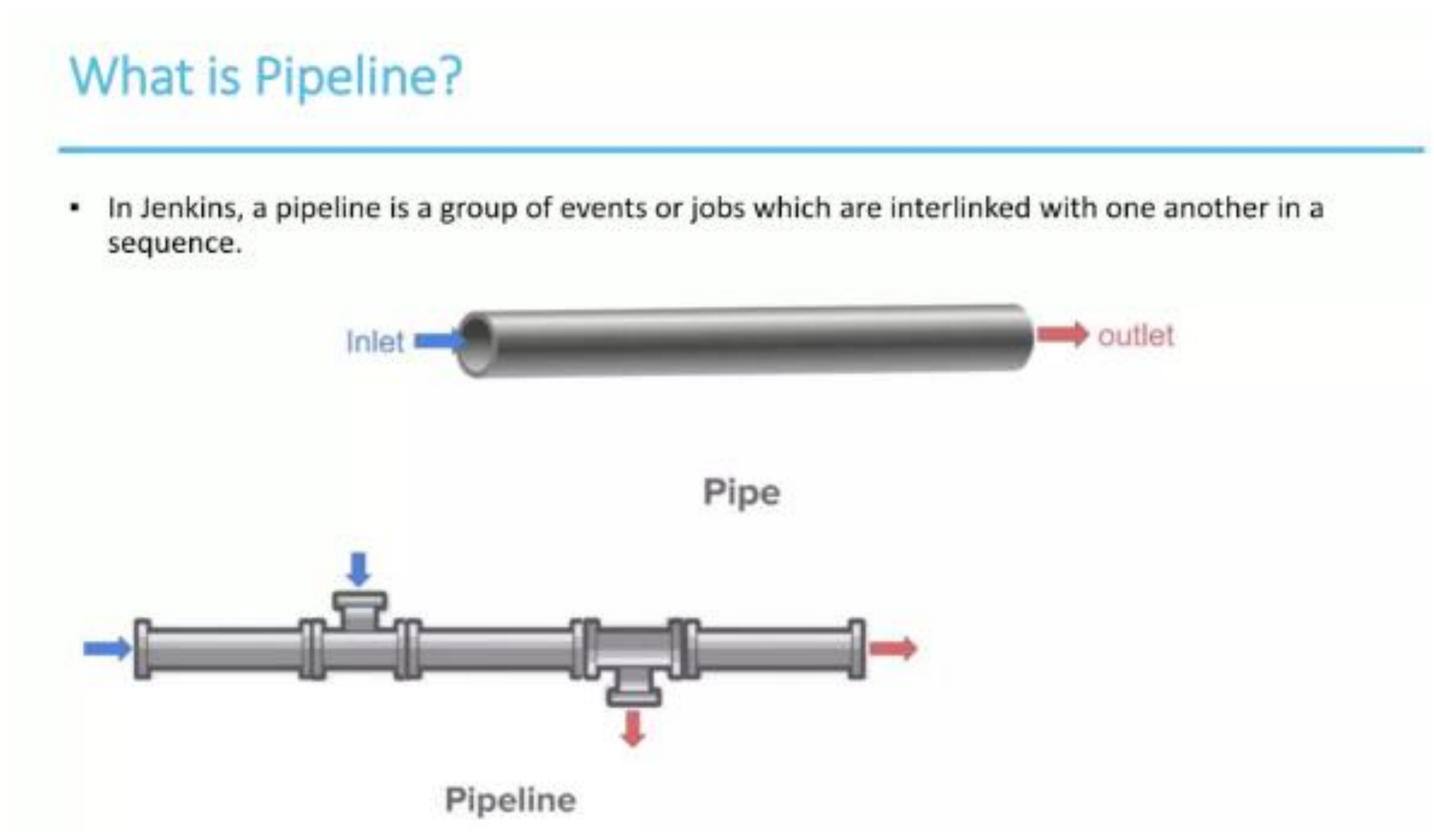
# Jekins pipeline

ow

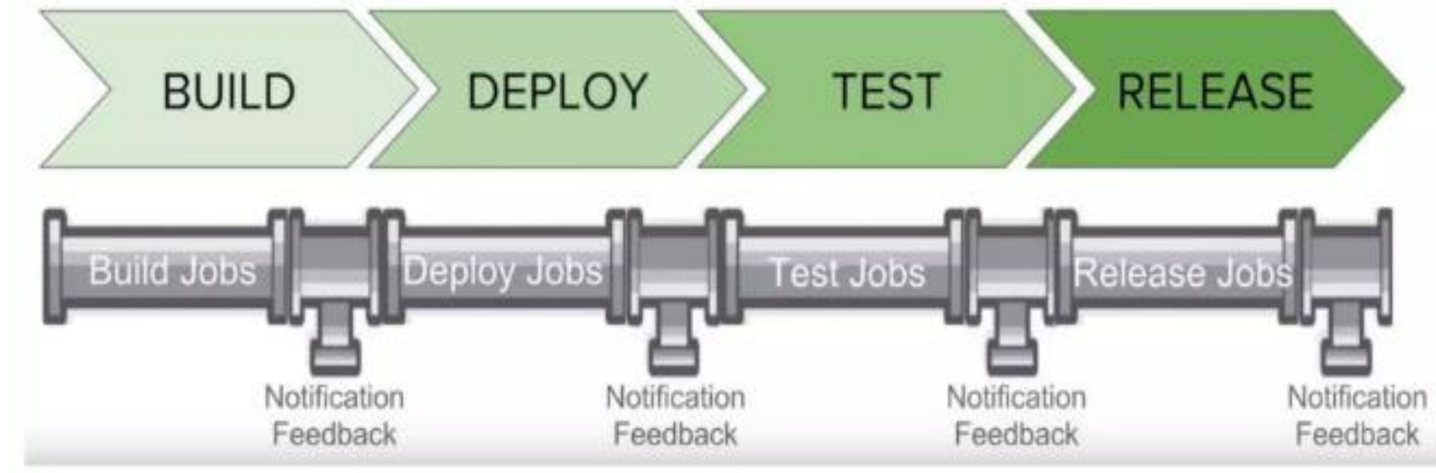
# What is pipeline?



## What is Pipeline?

- In Jenkins, a pipeline is a group of events or jobs which are interlinked with one another in a sequence.

• •


BUILD
DEPLOY
TEST
RELEASE
Build Jobs
Deploy Jobs
Test Jobs
Release Jobs
Notification Feedback
Notification Feedback
Notification Feedback
Notification Feedback

## How to setup BUILD PIPELINE in Jenkins

Step1: Chain required jobs in sequence Add upstream/downstream jobs

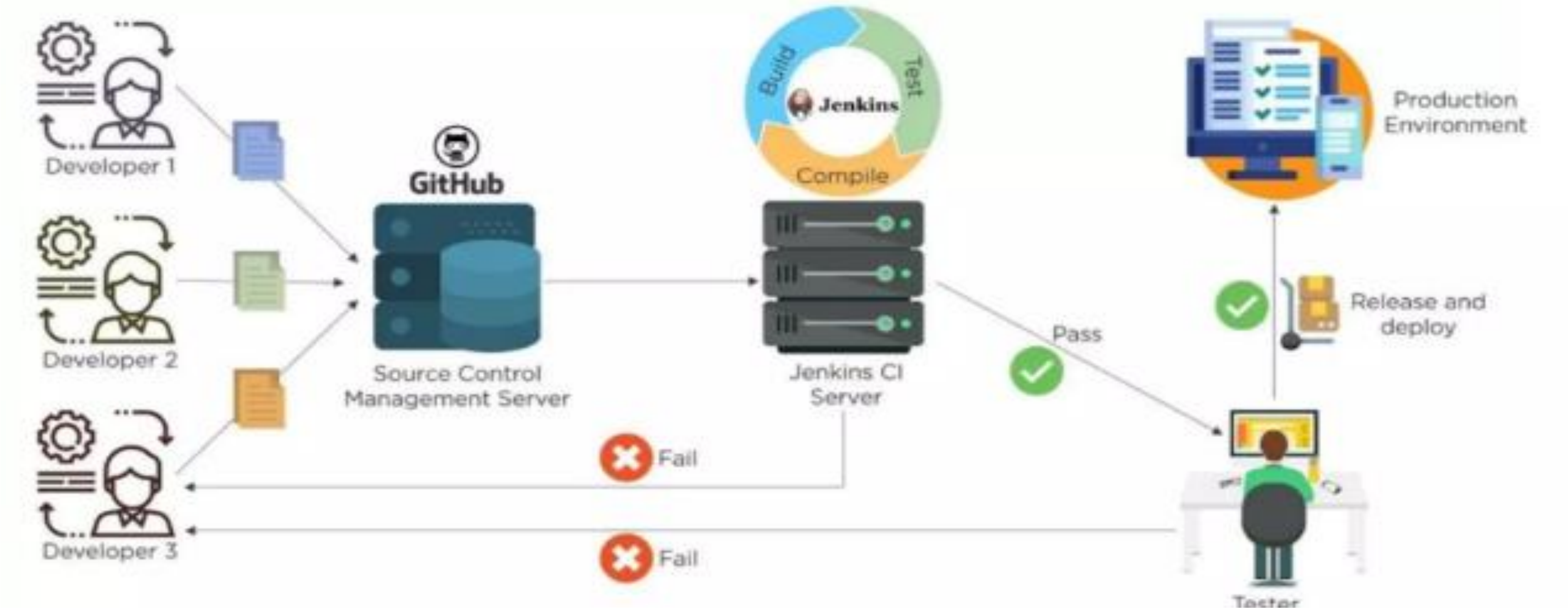
Step2: Install *Build Pipeline Plugin*

Step3: Add

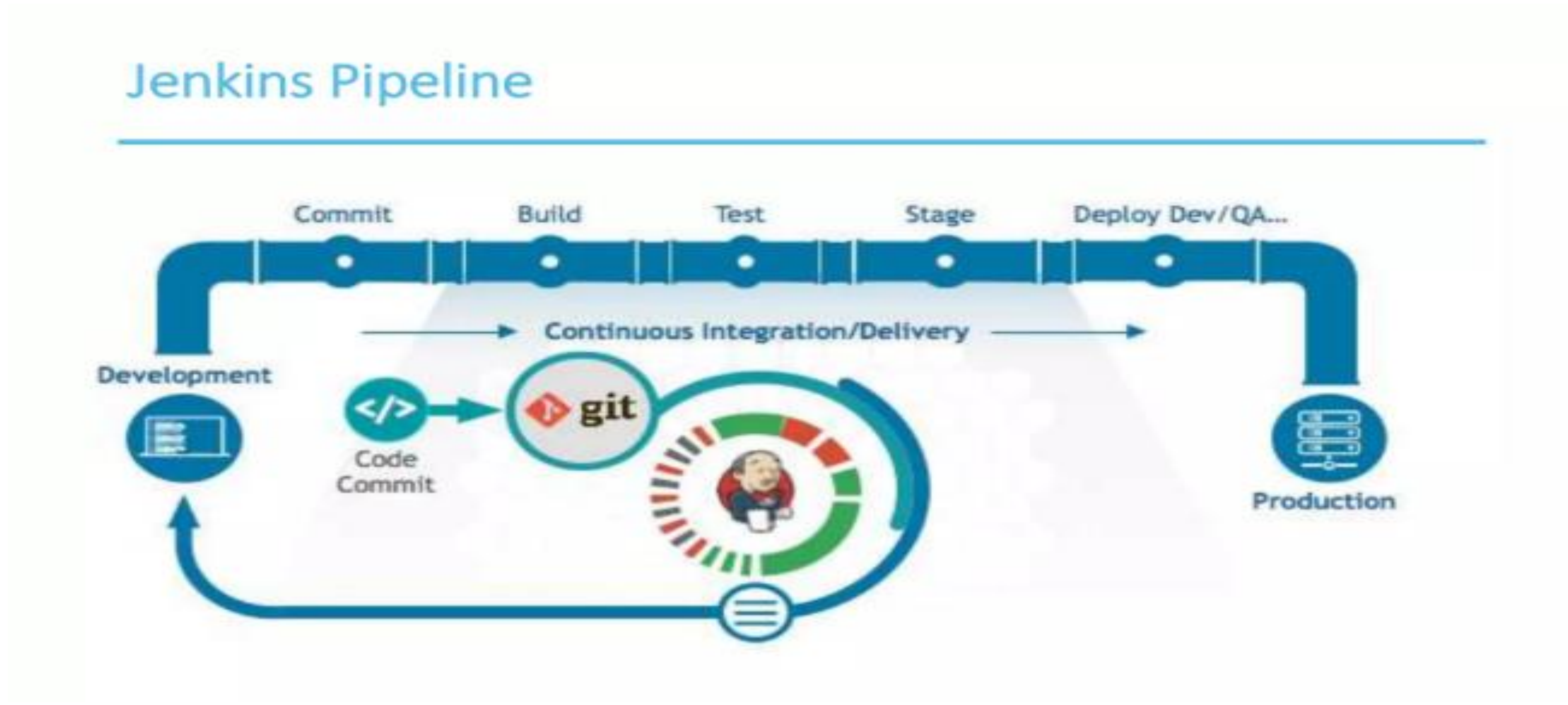
- Build Pipeline View
- Configure the view Step

Step4: Run and Validate

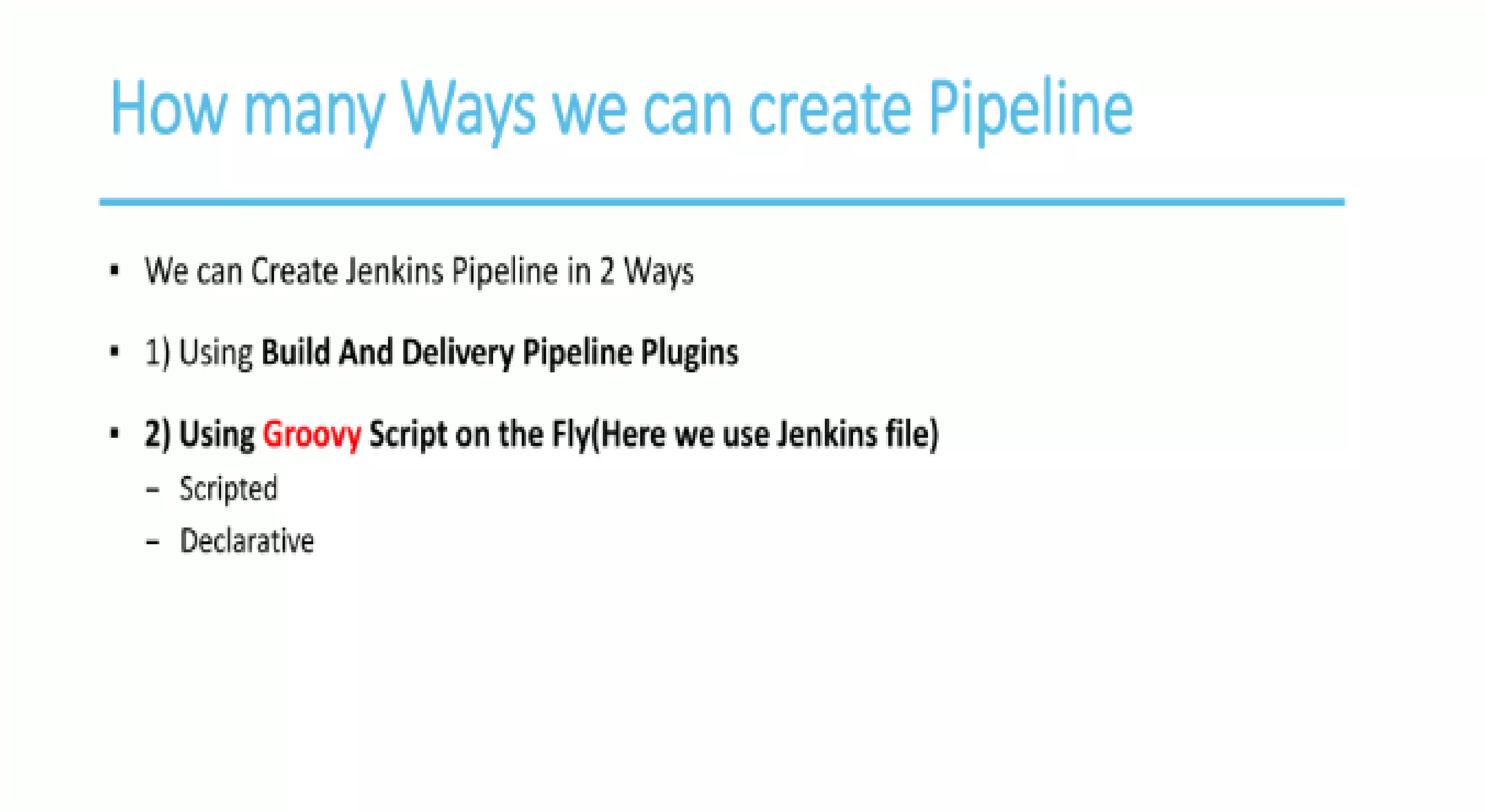
## CI & CD


Developer 1
Developer 2
Developer 3
GitHub
Source Control Management Server
Build
Test
Compile
Jenkins
Jenkins CI Server
Pass
Fail
Fail
Production Environment
Release and deploy
Tester

• •


Jenkins Pipeline
Commit
Build
Test
Stage
Deploy Dev/QA...
Continuous Integration/Delivery
Development
Code Commit
git
Production

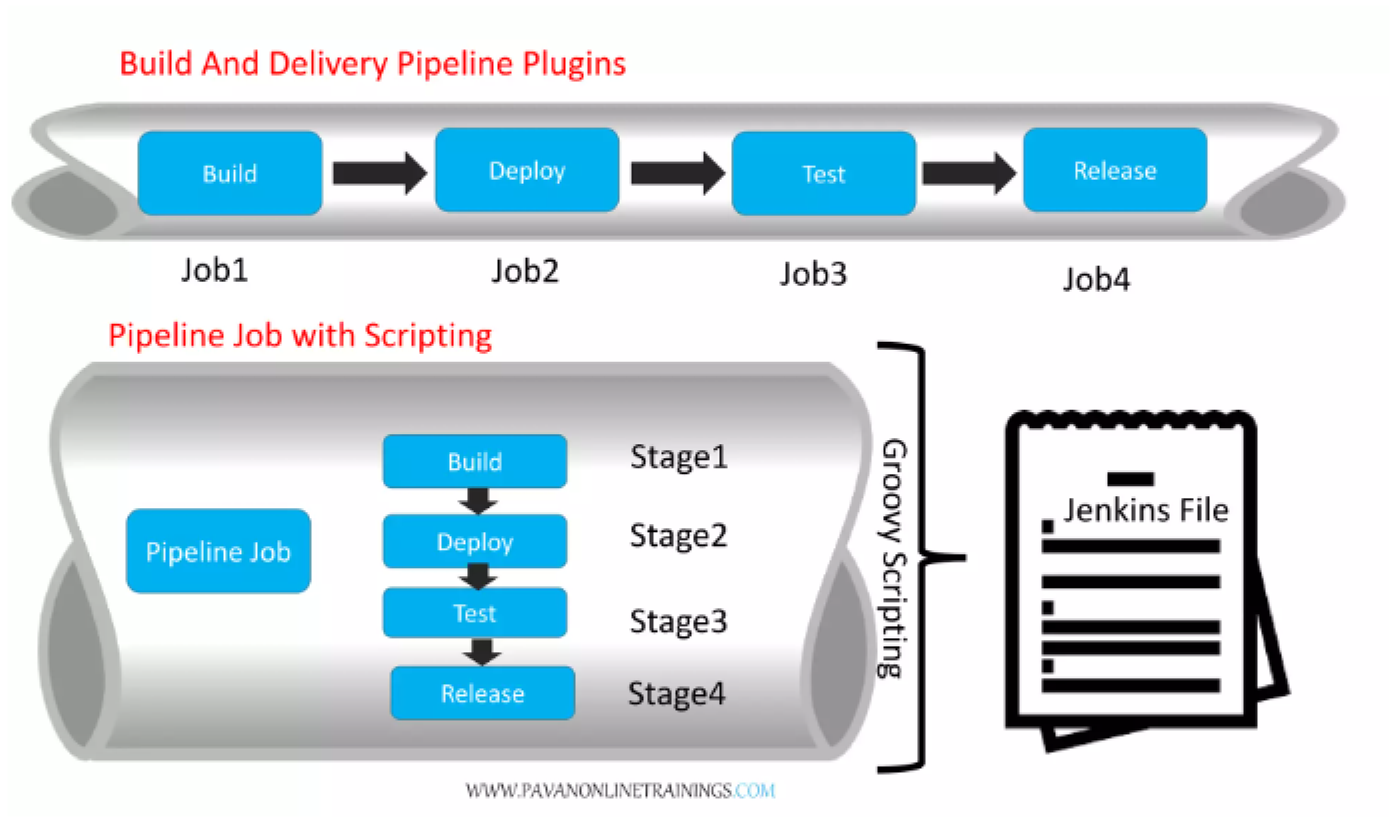
• •

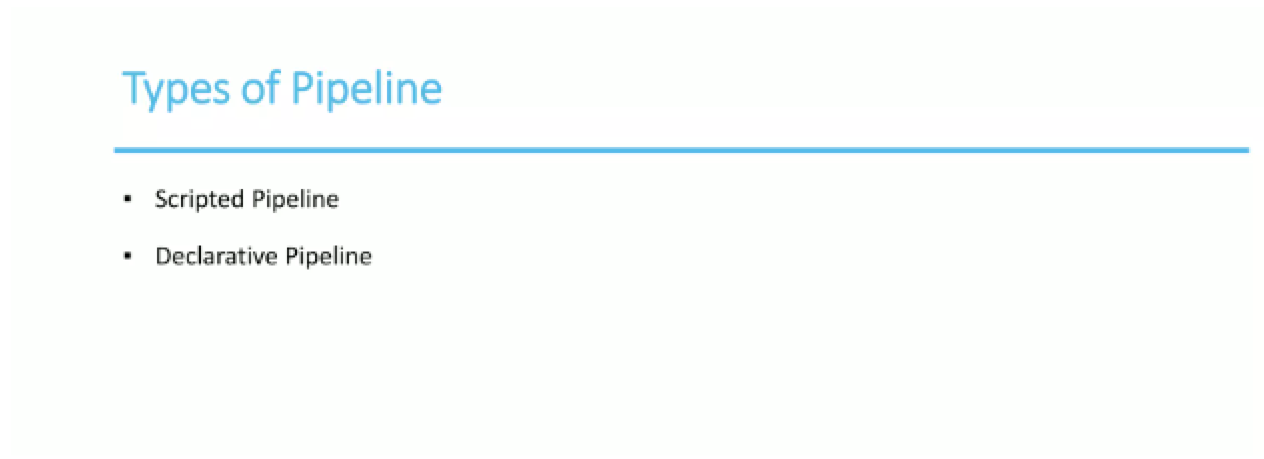


## How many Ways we can create Pipeline

- We can Create Jenkins Pipeline in 2 Ways
- 1) Using **Build And Delivery Pipeline Plugins**
- **2) Using Groovy Script on the Fly(Here we use Jenkins file)**
  - Scripted
  - Declarative

•


Build And Delivery Pipeline Plugins
Build
Deploy
Test
Release
Job1
Job2
Job3
Job4
Pipeline Job with Scripting
Pipeline Job
Build
Deploy
Test
Release
Stage1
Stage2
Stage3
Stage4
Groovy Scripting
Jenkins File
WWW.PAVANONLINETRAININGS.COM



## Types of Pipeline

- Scripted Pipeline
- Declarative Pipeline

## Scripted Pipeline

*Jenkinsfile (Scripted Pipeline)*

```
node {  ❶
    stage('Build') { ❷
        // ❸
    }
    stage('Test') { ❹
        // ❺
    }
    stage('Deploy') { ❻
        // ❼
    }
}
```

❶ Execute this Pipeline or any of its stages, on any available agent.

❷ Defines the "Build" stage. `stage` blocks are optional in Scripted Pipeline syntax. However, implementing `stage` blocks in a Scripted Pipeline provides clearer visualization of each `stage's subset of tasks/steps in the Jenkins UI.

❸ Perform some steps related to the "Build" stage.

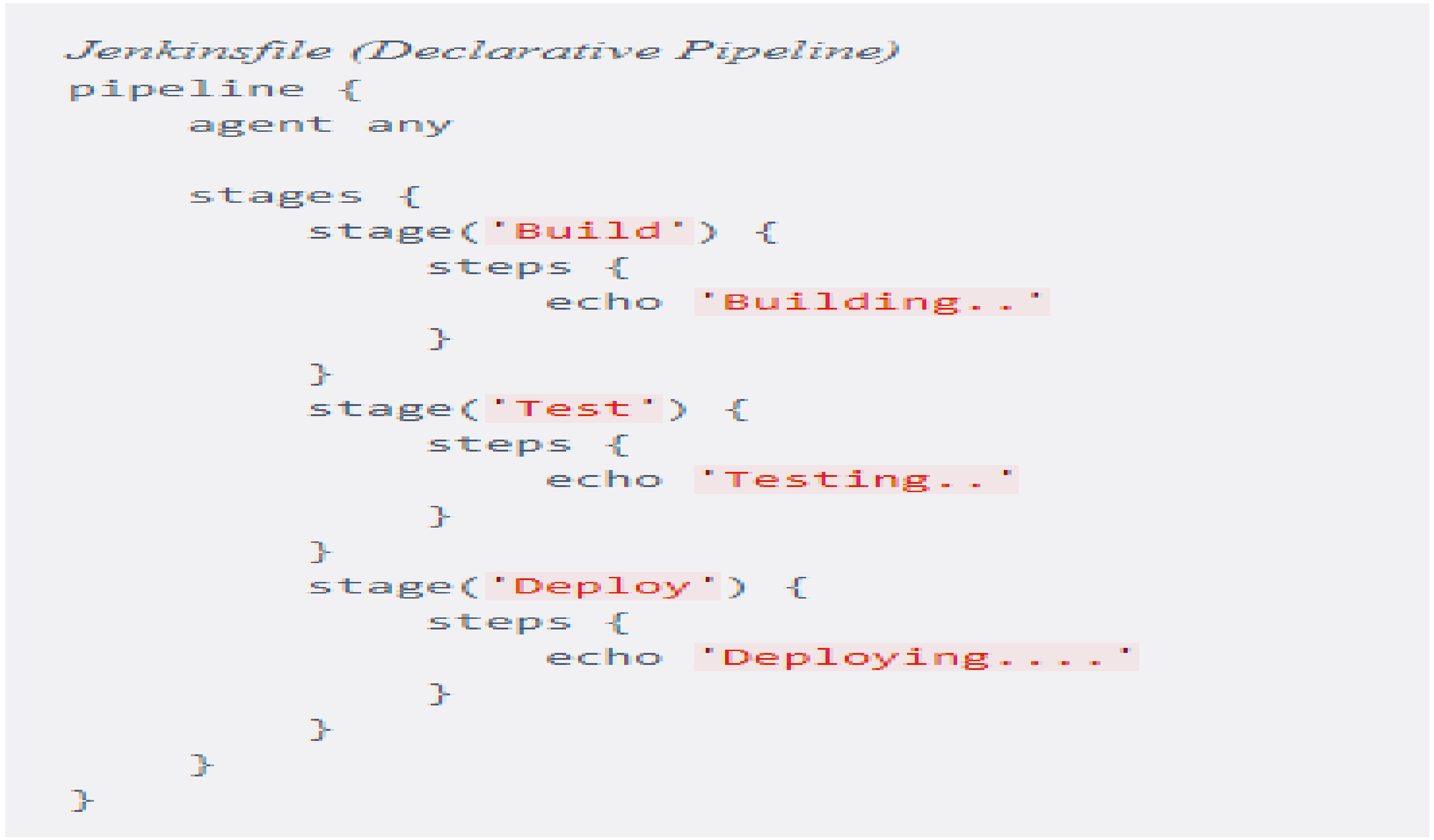
❹ Defines the "Test" stage.

❺ Perform some steps related to the "Test" stage.

❻ Defines the "Deploy" stage.

❼ Perform some steps related to the "Deploy" stage.

## Declarative Pipeline



*Jenkinsfile (Declarative Pipeline)*

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building..'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing..'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying....'
            }
        }
    }
}
```

## Build

- For many projects the beginning of "work" in the Pipeline would be the "build" stage. Typically this stage of the Pipeline will be where source code is assembled, compiled, or packaged. The Jenkinsfile is not a replacement for an existing build tool such as GNU/Make, Maven, Gradle, etc, but rather can be viewed as a glue layer to bind the multiple phases of a project’s development lifecycle (build, test, deploy, etc) together.
- Jenkins has a number of plugins for invoking practically any build tool in general use, but this example will simply invoke make from a shell step (sh)

..

*Jenkinsfile (Declarative Pipeline)*

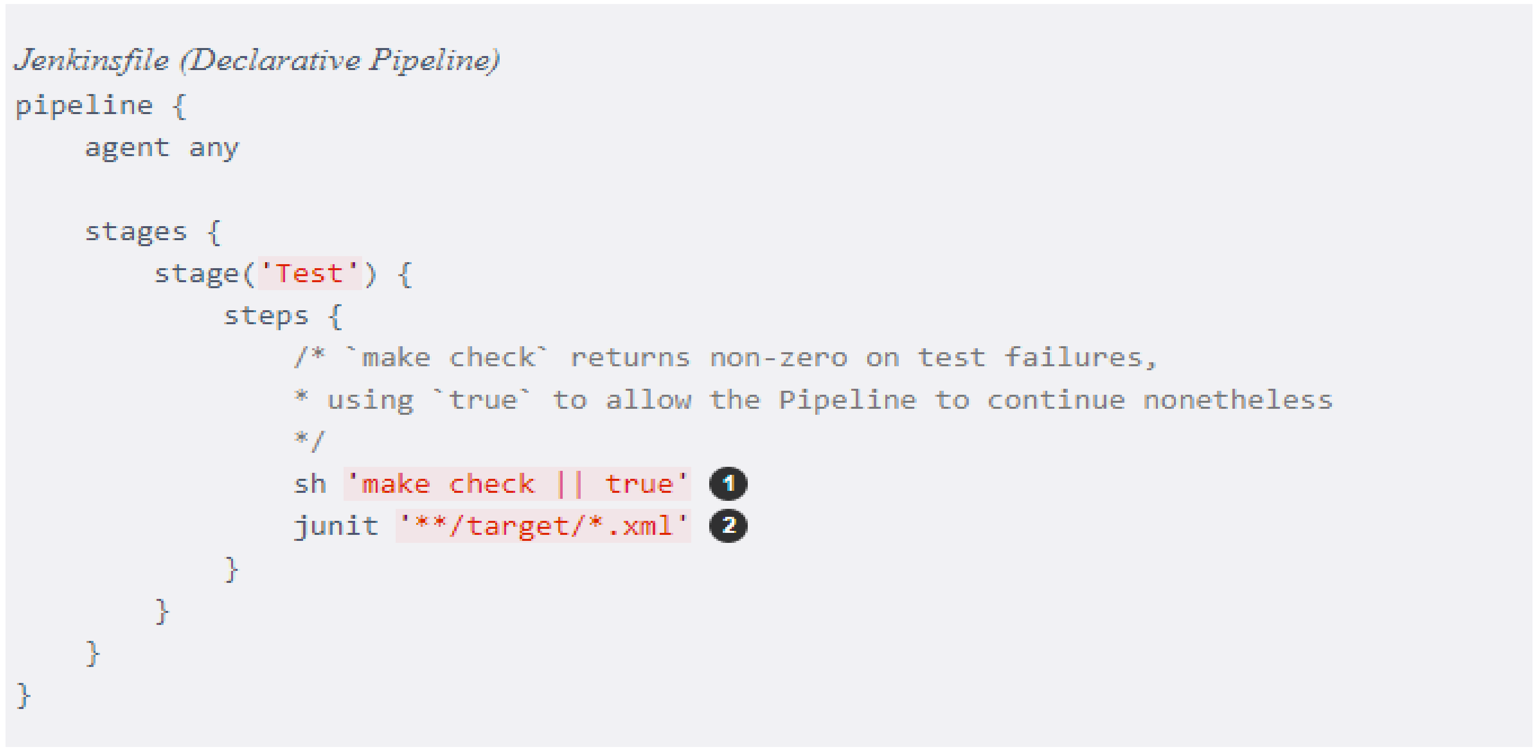
```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                sh 'make' ❶
                archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true ❷
            }
        }
    }
}
```

## Test

- Running automated tests is a crucial component of any successful continuous delivery process. As such, Jenkins has a number of test recording, reporting, and visualization facilities provided by a number of plugins. At a fundamental level, when there are test failures, it is useful to have Jenkins record the failures for reporting and visualization in the web UI.

- The example below uses the junit step, provided by the JUnit plugin.

• •

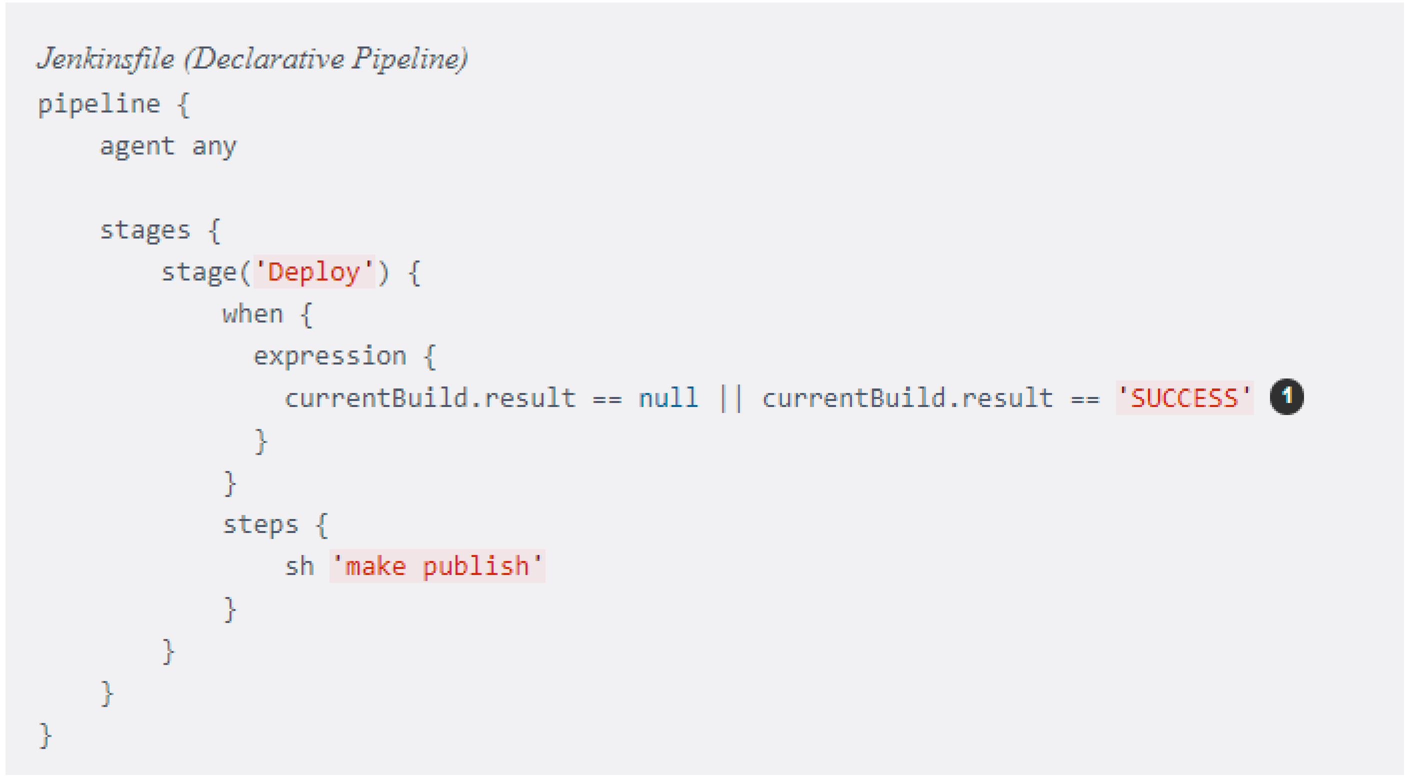


*Jenkinsfile (Declarative Pipeline)*

```
pipeline {
    agent any

    stages {
        stage('Test') {
            steps {
                /* `make check` returns non-zero on test failures,
                * using `true` to allow the Pipeline to continue nonetheless
                */
                sh 'make check || true' ❶
                junit '**/target/*.xml' ❷
            }
        }
    }
}
```

## Deploy

- Deployment can imply a variety of steps, depending on the project or organization requirements, and may be anything from publishing built artifacts to an Artifactory server, to pushing code to a production system.
- At this stage of the example Pipeline, both the "Build" and "Test" stages have successfully executed. In essence, the "Deploy" stage will only execute assuming previous stages completed successful



*Jenkinsfile (Declarative Pipeline)*

```
pipeline {
    agent any

    stages {
        stage('Deploy') {
            when {
              expression {
                currentBuild.result == null || currentBuild.result == 'SUCCESS' ❶
              }
            }
            steps {
                sh 'make publish'
            }
        }
    }
}
```

## Difference

- Syntax: Scripted pipelines are written in Groovy code, while declarative pipelines are written in YAML.
- Flexibility: Scripted pipelines offer more flexibility and control over the pipeline execution, while declarative pipelines have a more opinionated and structured approach.
- Complexity: Scripted pipelines can be more complex and difficult to maintain, while declarative pipelines are designed to be simpler and easier to understand.
- Features: Scripted pipelines offer access to the full Jenkins API and can be used to implement advanced features, while declarative pipelines have a more limited set of features and are more focused on simplicity and ease of use.