Jenkins Pipeline Script

What is Jenkins Pipeline used for?

As we are all aware of that Jenkins used to implementing

Continuous Integration

Continuous Testing

Continuous deployment and other in *DevOps Lifecycle*

During the implementation of Continuous Integration..., Jenkins uses a feature called "Jenkins Pipeline"

Continuous Delivery is the capability to release software at all times. It is a practice which ensures that the software is always in a **production-ready state.**

What does this mean? It means that every time a change is made to the code or the infrastructure, the software team must work in such a way that these changes are built quickly and tested using various **automation tools** after which the build is subjected to production.

By speeding up the delivery process, the development team will get more time to implement any required **feedback**. This process, of getting the software from the build to the production state at a faster rate is carried out by implementing continuous integration and continuous delivery.

Continuous delivery ensures that the software is built, tested, and released more frequently.
It reduces the cost, time, and risk of incremental software releases. To carry out continuous delivery, Jenkins introduced a new feature called ***Jenkins pipeline.***

What is Jenkins Pipeline?

Jenkins Pipeline is a combination of jobs to deliver software continuously using Jenkins.

A Jenkins pipeline consists of several states or stages, and they get executed in a sequence one after the other. Jenkins File is a simple text file that is used to create a pipeline as code in Jenkins. It contains code in Groovy Domain Specific Language (DSL), which is simple to write and human-readable.

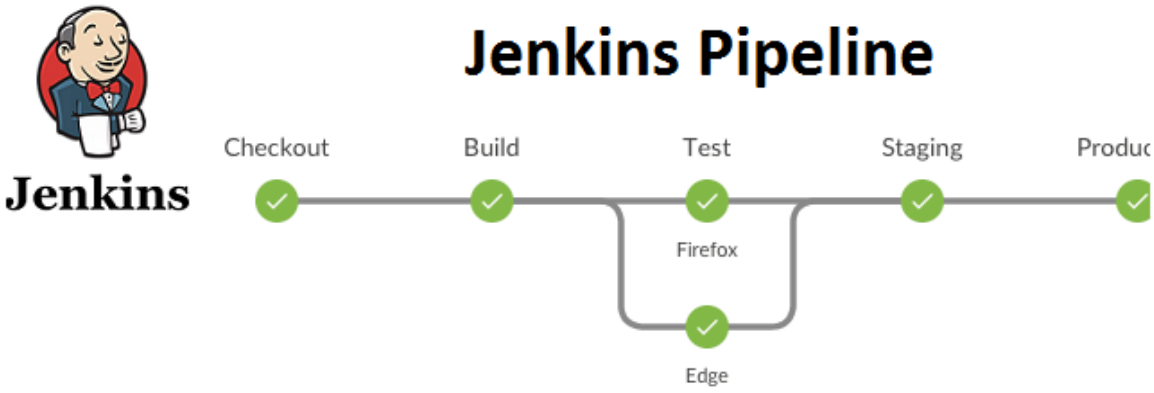
Declarative and Scripted pipelines differ in terms of the programmatic approach. One uses a declarative programming model and the second uses an imperative programming mode. **Declarative pipelines` break down stages into multiple steps, while in scripted pipelines there is no need for this**

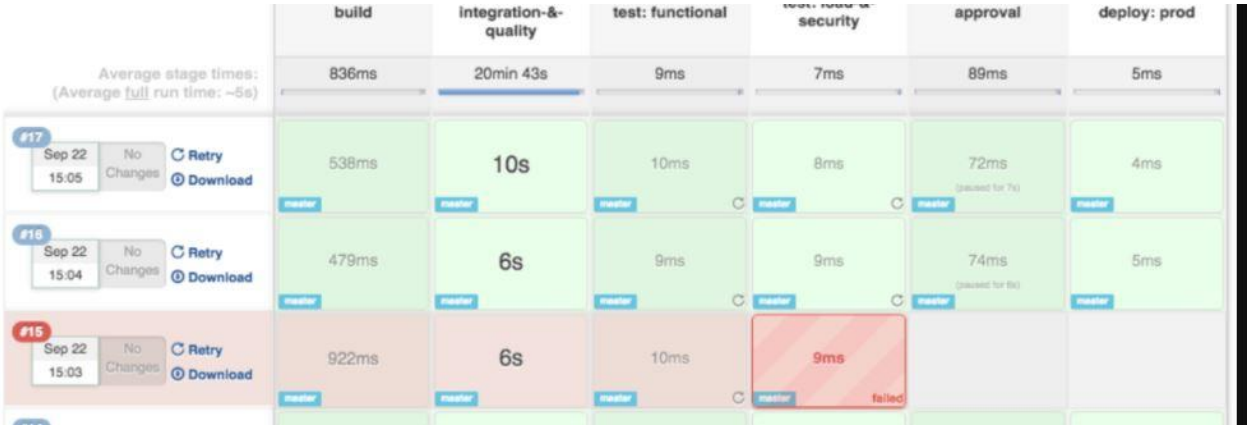
Either you can run JenkinsFile separately, or you can run the pipeline code from Jenkins Web UI also. There are two ways you can create a pipeline using Jenkins.

- **Declarative** – a new way of creating Jenkins Pipeline. Here you write groovy code containing “pipeline” blocks, which is checked into an SCM (Source Code Management)
- **Scripted** – way of writing groovy code where the code is defined inside “node” blocks.

A pipeline is a collection of jobs that brings the software from version control into the hands of the end-users by using automation tools. It is a feature used to **incorporate continuous delivery** in our software development workflow.

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 Jenkins file)





Why Jenkins Pipeline?

Continuous Delivery (CD) is an essential part of DevOps Lifecycle.

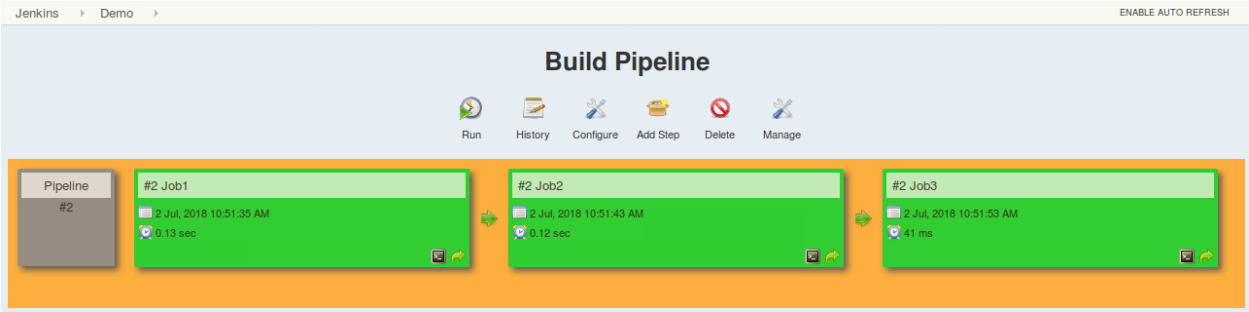
It makes sure that the software/application developers are creating is always production-ready. For this to happen, whenever the code gets updated, it needs to be built, tested, and deployed continuously. This is where Jenkins Pipeline comes into the picture.

In DevOps, Continuous Integration and Continuous Delivery (CI/CD) is achieved through Jenkins Pipeline. Using Jenkins Pipeline for CD helps to deliver the software with faster and frequent releases. This helps to incorporate the feedback in every next release.

What are the key features of plugins?

- They represent multiple Jenkins jobs as one whole workflow in the form of a pipeline.
- What do these pipelines do? These pipelines are a **collection of Jenkins jobs** that trigger each other in a specified sequence.

Let me explain this with an example. Suppose I'm developing a small application on Jenkins and I want to build, test and deploy it. To do this, I will allot 3 jobs to perform each process. So, job1 would be for build, job2 would perform tests and job3 for deployment. I can use the Jenkins build pipeline plugin to perform this task. After creating three jobs and chaining them in a sequence, the build plugin will run these jobs as a pipeline.



This approach is effective for deploying small applications. But what happens when there are complex pipelines with several processes (build, test, unit test, integration test, pre-deploy, deploy, monitor) running 100's of jobs?

The maintenance cost for such a complex pipeline is huge and increases with the number of processes. It also becomes tedious to build and manage such a vast number of jobs. To overcome this issue, a new feature called **Jenkins Pipeline Project** was introduced.

What is pipeline as code in Jenkins

The key feature of this pipeline is to define the entire deployment flow through code. What does this mean? It means that all the standard jobs defined by Jenkins are manually written as one whole script and they can be stored in a version control system. It basically follows the '**pipeline as code**' discipline. Instead of building several jobs for each phase, you can now code the entire workflow and put it in a **Jenkinsfile**. Below is a list of reasons why you should use the Jenkins Pipeline.

What is a JenkinsFile?

Jenkins pipelines can be defined using a text file called JenkinsFile. You can implement pipeline as code using JenkinsFile, and this can be defined by using a domain specific language (DSL). With JenkinsFile, you can write the steps needed for running a Jenkins pipeline.

Jenkins is an open source continuous integration server that provides the ability to continuously perform automated builds and tests. Several tasks can be controlled and monitored by Jenkins, including pulling code from a repository, performing static code analysis, building your project, executing unit tests, automated tests and/or performance tests, and finally, deploying your application. These tasks typically conform a continuous delivery pipeline.

The benefits of using JenkinsFile are:

- You can create pipelines automatically for all branches and execute pull requests with just one JenkinsFile.
- You can review your Jenkins code on the pipeline
- You can audit your Jenkins pipeline
- This is the singular source for your pipeline and can be modified by multiple users.
- Jenkins File can be defined by either Web UI or with a Jenkins File.

Types of Jenkins Pipeline

There are two types of Jenkins pipeline code.

- Declarative Pipeline
- Scripted Pipeline

Declarative Pipeline

Declarative pipeline is a relatively new feature that supports the pipeline as code concept. It makes the pipeline code easier to read and write. This code is written in a Jenkinsfile which can be checked into a source control management system such as Git.

All valid Declarative Pipelines must be enclosed within a pipeline block, for example:

```
pipeline {
    /* insert Declarative Pipeline here */
}
```

The basic statements and expressions which are valid in Declarative Pipeline follow the same rules as Groovy's syntax with the following exceptions:

- The top-level of the Pipeline must be a block, specifically: ***pipeline { }.***
- No semicolons as statement separators. Each statement has to be on its own line.
- Blocks must only consist of Sections, Directives, Steps, or assignment statements.
- A property reference statement is treated as a no-argument method invocation. So, for example, input is treated as input().

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent { docker { image 'maven:3.3.3' } }
    stages {
        stage('build') {
            steps {
                sh 'mvn --version'
            }
        }
    }
}
```

Whereas, the scripted pipeline is a traditional way of writing the code. In this pipeline, the Jenkinsfile is **written on the Jenkins UI instance**. Though both these pipelines are based on the groovy DSL, the scripted pipeline uses stricter groovy based syntaxes because it was the first pipeline to be built on the groovy foundation. Since this Groovy script was not typically desirable to all the users, the declarative pipeline was introduced to offer a simpler and more optioned Groovy syntax.

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any 1
    stages {
        stage('Build') { 2
            steps {
                // 3
            }
        }
        stage('Test') { 4
            steps {
                // 5
            }
        }
        stage('Deploy') { 6
            steps {
                // 7
            }
        }
    }
}
```

1. Execute this Pipeline or any of its stages, on any available agent.
2. Defines the "Build" stage.
3. Perform some steps related to the "Build" stage.
4. Defines the "Test" stage.
5. Perform some steps related to the "Test" stage.
6. Defines the "Deploy" stage.
7. Perform some steps related to the "Deploy" stage.

Understanding Scripted Pipeline Syntax

Jenkinsfile (Scripted Pipeline)

```
node { 1
    stage('Build') { 2
        // 3
    }
    stage('Test') { 4
        // 5
    }
    stage('Deploy') { 6
        // 7
    }
}
```

1. Execute this Pipeline or any of its stages, on any available agent.
2. 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.
3. Perform some steps related to the "Build" stage.
4. Defines the "Test" stage.
5. Perform some steps related to the "Test" stage.
6. Defines the "Deploy" stage.
7. Perform some steps related to the "Deploy" stage.

Example of Jenkins Pipeline Declarative Pipeline Script

Jenkinsfile (Declarative Pipeline)

```
pipeline { (1)
    agent any (2)
    options {
        skipStagesAfterUnstable()
    }
    stages {
        stage('Build') { (3)
            steps { (4)
                sh 'make' (5)
            }
        }
        stage('Test'){
            steps {
                sh 'make check'
                junit 'reports/**/*.xml' (6)
            }
        }
        stage('Deploy') {
            steps {
                sh 'make publish'
            }
        }
    }
}
```

1. `pipeline` is Declarative Pipeline-specific syntax that defines a "block" containing all content and instructions for executing the entire Pipeline.
2. `agent` is Declarative Pipeline-specific syntax that instructs Jenkins to allocate an executor (on a node) and workspace for the entire Pipeline.
3. `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.
4. `steps` is Declarative Pipeline-specific syntax that describes the steps to be run in this `stage`.
5. `sh` is a Pipeline step (provided by the Pipeline: Nodes and Processes plugin) that executes the given shell command.
6. `junit` is another a Pipeline step (provided by the JUnit plugin) for aggregating test reports.
7. `node` is Scripted Pipeline-specific syntax that instructs Jenkins to execute this Pipeline (and any stages contained within it), on any available agent/node. This is effectively equivalent to `agent` in Declarative Pipeline-specific syntax.