**Jenkins Pipeline: Declarative vs. Scripted**

The majority of DevOps Engineers and Developers interact with Jenkins in their day-to-day activities. Jenkins is an automation tool that helps developers and DevOps to build, test and deploy applications. Nowadays, people prefer to use Jenkins pipelines to set up build, test, and deployment workflow over freestyle jobs. Jenkins pipeline is the best and recommended way to build, test and deploy applications.

Jenkins pipeline comes with two types: Declarative and Scripted. One must be aware of the differences between the two before starting the development of pipelines. In this blog, we will go through the differences between Declarative and Scripted pipelines. This blog will help you to select a suitable pipeline type as per your requirement.

In this article, we will go through the below-mentioned points:

1. Jenkins Pipeline
2. Declarative vs. Scripted Pipeline  
   2.1. Introduction  
   2.2. Flexibility & Simplicity  
   2.3. Programming Model  
   2.4. Limitations  
   2.5. Integration with Blue Ocean  
   2.6. Functional
3. Conclusion
4. Reference

**1- Jenkins Pipeline**

A pipeline is a collection of steps or jobs which are interlinked with one another in a sequence.

Jenkins supports different plugins which help to implement continuous integration & continuous delivery pipelines into Jenkins.

A CI/CD pipeline is an automated way of getting source code from version control systems to the users via different stations such as Version Control System, Code Inspection, Test Cases Execution, Build, Artifact Creation, Artifact Repository, and Deployment on the servers.

Pipelines are also referred to as a “Deployment-as-a-Code”.

The Jenkins Pipeline is written into a text file called a Jenkinsfile (You can use a different name) which can be committed to a project’s source control repository.

We can integrate different sets of tools onto Jenkins to create CI and CD pipelines.

When Jenkins Pipeline was first created, Groovy was selected as the foundation. Jenkins has long shipped with an embedded Groovy engine to provide advanced scripting capabilities for Admins, DevOps, and Users alike.

Jenkins is based on DSL, which is Domain Specific Language. A domain-specific language is a language that is developed to solve specific domain problems.

A pipeline can be written using two types of syntax — declarative and scripted.

Both declarative and scripted pipelines are fundamentally the same pipeline sub-system underneath. They are durable implementations of “Pipeline as code”. They both are able to use plugins and shared libraries. Where they differ however is in syntax and flexibility.

**2- Declarative vs. Scripted Pipeline**

**2.1 Introduction:**

**Declarative:** Declarative is a more recent and advanced implementation of a pipeline as a code.

**Scripted:** Scripted was the first and traditional implementation of the pipeline as a code in Jenkins. It was designed as a general-purpose DSL (Domain Specific Language) built with Groovy.

**2.2 Flexibility & Simplicity:**

**Declarative:** The Groovy learning curve isn’t typically desirable for all members of a given team, so a Declarative pipeline was created to offer a simpler, more opinionated & well-defined syntax for authoring Jenkins Pipeline.

**Scripted:** A scripted pipeline is a fully-featured programming environment. Scripted Pipeline offers a tremendous amount of flexibility and extensibility to Jenkins users.

**2.3 Programming Model:**

**Declarative:**Declarative Pipeline encourages a Declarative Programming Model. In declarative programming, you specify what you want. You don’t care how it will get done.

**Scripted:** Scripted Pipelines follow a more Imperative Programming Model. In Imperative programming, you specify what you want and how it should be done.

**2.4 Limitations:**

**Declarative**: Declarative pipeline limits what is available to the user with a more strict and pre-defined structure, making it an ideal choice for simpler CI/CD pipelines.

**Scripted:** Scripted pipeline provides very few limits, such as structure and syntax are the only limits defined by Groovy itself. It is an ideal choice for those users who have more complex requirements.

**2.5 Integration with Blue Ocean:**

**Declarative:**A declarative pipeline can be integrated with Blue Ocean (Blue Ocean is a new user experience for Jenkins based on a personalizable, modern design that allows users to graphically create, visualize and debug CI/CD Pipelines).

**Scripted:**The scripted pipeline does not support Blue Ocean integration.

**2.6 Functional:**

**Code Validation:**

**Declarative:** Declarative pipeline supports code validation. Code validation gets done before executing any stage. So if there is any syntax error or typos in a pipeline, the user will get an error at the start of the execution. This might save us a lot of time.

**Scripted:** Scripted pipeline doesn’t support any kind of code validation before executing the stages. It will throw an error at the time of execution of a specific step, which has syntax error or typo.

**Restart from Stage:**

**Declarative:** Declarative pipeline support to restart from the failed stage. It uses the workspace from the previous build to determine the failed stage and starts the execution from the failed stage. It might save us time and effort.

**Scripted:** The scripted pipeline does not support the “Restart from Stage” option.

**Options Block:**

**Declarative:**Declarative pipeline support Options block. In the declarative pipeline, options are separated from the pipeline logic and can be placed before Stages. Using the options directive, we can configure pipeline-specific options in the pipeline itself.

**Scripted:** The scripted pipeline does not support the Options block but different options such as properties, can be used with the pipeline logic.

**Skip Stage:**

**Declarative:**In the declarative pipeline, stages can be skipped using the “When” block. “When” block skips the whole stage if a condition is not met.

**Scripted:**In the scripted pipeline, stages can not be skipped totally. The logic inside the stage can be skipped using the If statement but execution will go through each stage in the pipeline.

**Environment Block:**

**Declarative:**Declarative pipeline supports the Environment block. Environment block helps to define global variables and load credentials into the pipeline.

**Scripted:** The scripted pipeline does not support Environment block but you can define global variables and use credentials in a different way using Groovy.

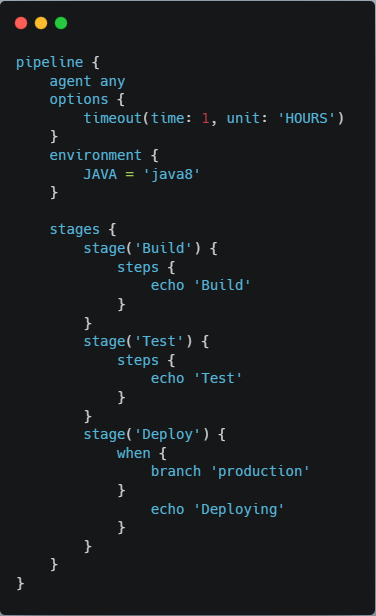
**Cross Functionality Support:**

**Declarative:** We can integrate the scripted pipeline block in a declarative pipeline inside the “script” step.

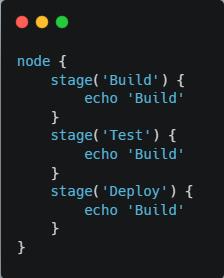
**Scripted:**Declarative pipeline-specific blocks cannot be used in a scripted pipeline.

**Syntax:**

**Declarative:**



**Scripted:**



**3- Conclusion:**

Both declarative and scripted pipelines are a great way of building deployment pipelines but nowadays, the declarative pipeline is a more preferred way to write a pipeline. You can leverage the benefit of scripted pipeline in a declarative pipeline using Script step but not vice versa.