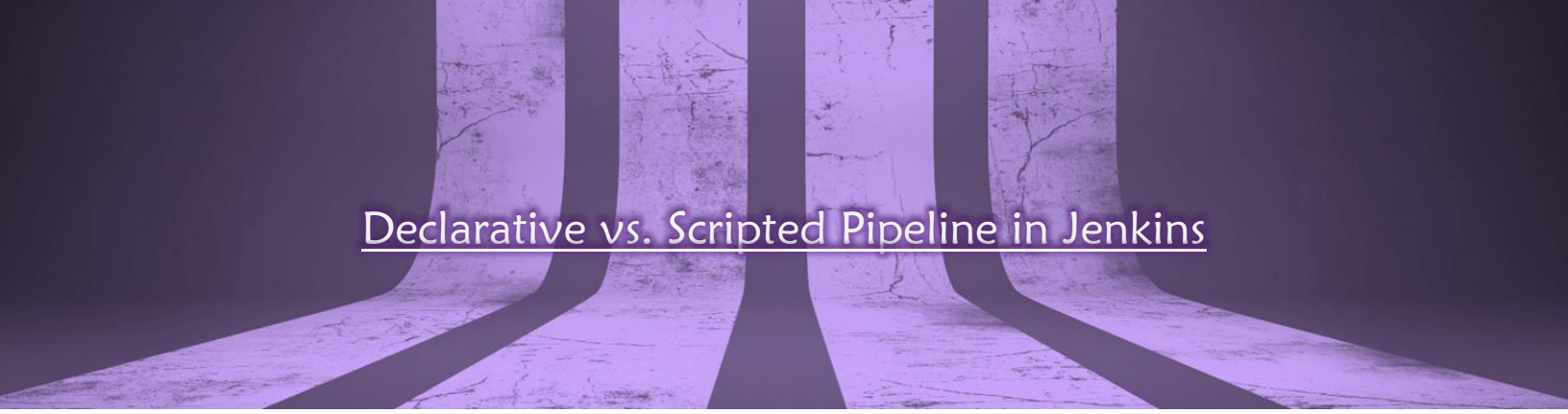

Declarative vs. Scripted Pipeline in Jenkins

The productivity of your development workflow may be greatly increased by automating your build, test, and deployment procedures in the context of Continuous Integration and Continuous Deployment (CI/CD). This is made possible by Jenkins, one of the most widely used automation servers, using Jenkins files. A Jenkins file is a text file checked into source control and contains the definition of a Jenkins pipeline. This short tutorial will assist you in writing your first Jenkins file script, which will be your first step in automating pipelines.

Understanding Jenkins Files:

A domain-specific language (DSL) based on Groovy is used to write Jenkins files. You can script your complete build process since they provide a set of actions and the setting in which they should be performed. Declarative and Scripted Pipelines are the two different syntaxes that Jenkins, the open-source automation server, offers for specifying your CI/CD pipeline inside a Jenkins file. Although their syntax, flexibility, and usability are different, they both accomplish the same basic goal of automating the development, testing, and deployment of software.
Declarative pipeline syntax is recommended for people unfamiliar with Jenkins as it is simpler to understand.

Declarative Pipeline:

Declarative Pipeline syntax was introduced to make pipeline specification more formal and specified, making it easier to write complicated pipelines.

It is distinguished by:

1. Syntax Simplicity: Makes use of a more easily understood and legible syntax, which facilitates learning for novices or individuals who would rather write less code.
2. Pipeline Block: Provides a user-friendly means of defining the phases and steps by encapsulating the complete pipeline operation inside a pipeline block.
3. Included Directives: Provides out-of-the-box support for conditional statements, environment variables, agent selection, post actions, and more.
   Error Checking and Validation: Provides syntactic error checks and validation to lower the likelihood of pipeline failure brought on by scripting mistakes.

```
'''
#!/usr/bin/env groovy

import groovy.json.JsonSlurperClassic
import jenkins.model.Jenkins;
import hudson.model.*

parameters {
    string(name: 'parameters', defaultValue: '', description: 'some parameters')
}
pipeline {
      agent { node { label 'master' } }
      options {
            some pipeline options
            }
            stages {
                  stage ('Build'){
                        steps {
                              script {

                                    }
                              }

                    }
                  stage ('test') {

                        steps {
                              script {

                                    }
                        }
                  }
                  stage ('deploy') {
                        steps {
                              script {

                                    }
                        }
                  }
            }
      }
}
 def somedefenition () {
      return "somedefenition"
}
```

Scripted Pipeline:

Predating Declarative Syntax, Scripted Pipeline offers a more potent and versatile method of pipeline definition. It is distinguished by:

1. Flexibility and Control: Provides complete command over the pipeline logic by utilizing the Groovy language's capability, making it perfect for intricate processes.

2. Groovy-oriented Programming: written in a DSL that uses Groovy and supports standard programming techniques like conditionals and loops.

3. Node Block: In the Jenkins system, an executor and workspace are assigned via a node block.

‘’’

```
node {
    stage('Build') {
      script {
        def mvnHome = tool 'M3'
        def mvnCMD = "${mvnHome}/bin/mvn"
        sh "${mvnCMD} clean package"
      }
    }
    stage('Test') {
      script {
        def mvnHome = tool 'M3'
        def mvnCMD = "${mvnHome}/bin/mvn"
        sh "${mvnCMD} test"
      }
    }
    stage('Deploy') {
      script {
        def mvnHome = tool 'M3'
        def mvnCMD = "${mvnHome}/bin/mvn"
        sh "${mvnCMD} deploy"
      }
    }
}
```

Syntax and Structure: Groovy code is used by Scripted Pipelines to create more complicated and flexible definitions, whilst Declarative Pipelines provide a more straightforward and structured syntax with predefined parts.

Flexibility vs. Ease of Use: Declarative Pipelines are ideal for workflows that are simpler to build and comprehend, or for individuals who are new to Jenkins. Complex, conditional, or highly customized pipelines are better suited for scripted pipelines because of their Groovy-based logic.
Error Handling and Validation: Declarative pipelines lower the possibility of pipeline failures brought on by scripting errors by offering more thorough syntax validation and error handling right out of the box.

In conclusion, the decision you make in Jenkins between Declarative and Scripted Pipelines is based on the complexity of your project, your level of Groovy ability, and the specific requirements of your CI/CD workflow.