Declarative Pipelines vs. Scripted Pipelines

When it comes to writing pipelines as code in Jenkins, there are two main approaches: Declarative Pipelines and Scripted Pipelines. Each approach has its own syntax, structure, and characteristics. In this lesson, we'll explore the differences between Declarative Pipelines and Scripted Pipelines and understand when to use each one.

Declarative Pipelines

Declarative Pipelines, introduced in Jenkins 2.5, provide a more simplified and opinionated way of defining pipelines. They follow a declarative syntax that focuses on "what" the pipeline should do, rather than "how" it should do it.

Think of Declarative Pipelines as a pre-defined template for your CI/CD process. Just like using a recipe book with clear instructions and measurements, Declarative Pipelines provide a structured approach to defining your pipeline.

Key characteristics of Declarative Pipelines:

- Defined using a `pipeline` block
- Consists of sections like `agent`, `stages`, `steps`, and `post`
- Follows a declarative syntax with pre-defined directives
- Suitable for simpler pipelines and readability
- Easier to get started with for beginners

Here's an example of a Declarative Pipeline:

```
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        sh 'npm install'
        sh 'npm run build'
      }
    }
    stage('Test') {
      steps {
        sh 'npm test'
      }
    }
    stage('Deploy') {
      steps {
        sh 'npm run deploy'
      }
    }
  }
```

```
}
```

In this example, the pipeline is defined using the `pipeline` block, and it consists of stages like Build, Test, and Deploy. Each stage contains specific steps, such as running npm commands.

Scripted Pipelines

Scripted Pipelines, on the other hand, provide a more flexible and programmatic approach to defining pipelines. They allow you to write custom Groovy code to define the pipeline logic and flow.

Think of Scripted Pipelines as cooking from scratch. Just like a chef who creates their own recipes and experiments with different ingredients, Scripted Pipelines give you the freedom to define your pipeline using Groovy programming constructs.

Key characteristics of Scripted Pipelines:

- Defined using a `node` block
- Written in Groovy programming language
- Provides flexibility and control over pipeline logic
- Allows custom functions, loops, and conditional statements
- Suitable for complex pipelines and custom requirements
- Requires knowledge of Groovy programming

Here's an example of a Scripted Pipeline:

```
node {
  stage('Build') {
    sh 'npm install'
    sh 'npm run build'
  }
  stage('Test') {
    sh 'npm test'
  }
  stage('Deploy') {
    if (env.BRANCH_NAME == 'main') {
      sh 'npm run deploy'
    } else {
      echo 'Skipping deployment for non-main branch'
    }
  }
}
```

In this example, the pipeline is defined using the `node` block, and the stages are defined using Groovy syntax. The pipeline includes custom logic, such as checking the branch name before deploying.

Choosing Between Declarative and Scripted Pipelines

When deciding between Declarative and Scripted Pipelines, consider the following factors:

1. Simplicity: If you have a straightforward pipeline with a linear flow, Declarative Pipelines offer a simpler and more readable syntax.
2. Flexibility: If you require complex logic, conditional statements, or custom functions, Scripted Pipelines provide more flexibility and control.
3. Readability: Declarative Pipelines are generally easier to read and understand, especially for team members who may not be familiar with Groovy programming.
4. Learning Curve: Declarative Pipelines have a gentler learning curve, making them suitable for beginners. Scripted Pipelines require knowledge of Groovy programming, which may have a steeper learning curve.

In practice, you can mix and match Declarative and Scripted Pipelines within the same `Jenkinsfile`. You can use Declarative Pipelines for the overall structure and use Scripted Pipelines for specific stages or steps that require more advanced logic.

In the upcoming lessons, we'll focus primarily on Declarative Pipelines, as they are the recommended approach for most use cases. We'll explore the syntax and structure of Declarative Pipelines in more detail and learn how to write efficient and maintainable pipelines.

Get ready to master the art of Declarative Pipelines and take your CI/CD game to the next level!