Stages, Steps, and Post Actions in Declarative Pipelines

In the previous lesson, we explored the syntax and structure of Declarative Pipelines in Jenkins. Now, let's dive deeper into the key components of a Declarative Pipeline: stages, steps, and post actions. Understanding these components will allow you to create complex and efficient pipelines.

Stages

Stages are the building blocks of a Declarative Pipeline. Each stage represents a logical grouping of steps that perform a specific task or phase in the CI/CD process. Stages are defined within the `stages` block in the pipeline.

```
pipeline {
  agent any

  stages {
    stage('Build') {
      /* Steps for the Build stage */
    }
    stage('Test') {
      /* Steps for the Test stage */
    }
    stage('Deploy') {
      /* Steps for the Deploy stage */
    }
  }
}
```

In this example, we have three stages: Build, Test, and Deploy. Each stage contains its own set of steps that define the actions to be performed within that stage.

Steps

Steps are the individual actions or tasks that are executed within a stage. They define what actually happens during the pipeline execution. Steps can include various actions, such as running shell commands, invoking Jenkins plugins, or executing custom Groovy code.

```
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, each stage contains specific steps. The Build stage runs `npm install` to install dependencies and `npm run build` to build the application. The Test stage runs `npm test` to execute tests, and the Deploy stage runs `npm run deploy` to deploy the application.

Jenkins provides a wide range of built-in steps that you can use in your pipeline, such as `sh` for running shell commands, `git` for interacting with Git repositories, `junit` for publishing test results, and many more. You can also use custom steps defined by plugins or write your own custom steps using Groovy.

Post Actions

Post actions are steps that are executed at the end of the pipeline, regardless of the outcome of the stages. They are defined within the `post` section of the pipeline. Post actions allow you to perform cleanup tasks, send notifications, or trigger additional actions based on the pipeline's status.

```
pipeline {
  agent any

  stages {
    /* Pipeline stages */
  }

  post {
    always {
      /* Steps to run always, regardless of the pipeline's outcome */
      junit 'test-results/**/*.xml'
    }
    success {
      /* Steps to run when the pipeline succeeds */
      archiveArtifacts 'dist/**'
    }
    failure {
      /* Steps to run when the pipeline fails */
      mail to: 'team@example.com', subject: 'Pipeline Failed', body: 'The pipeline has failed. Please
investigate.'
    }
  }
}
```

In this example, the `post` section defines actions to be executed based on the pipeline's outcome. The `always` block runs regardless of the pipeline's status and publishes JUnit test results. The `success` block runs only if the pipeline succeeds and archives the build artifacts. The `failure` block runs only if the pipeline fails and sends an email notification to the team.

Conditional Stages and Steps

Declarative Pipelines also support conditional execution of stages and steps based on certain criteria. You can use the `when` directive to define conditions for executing a stage or step.

```
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        /* Build steps */
      }
    }
    stage('Test') {
      steps {
        /* Test steps */
      }
    }
    stage('Deploy') {
      when {
        branch 'main'
      }
      steps {
        /* Deploy steps */
      }
    }
  }
}
```

In this example, the Deploy stage is only executed when the branch being built is 'main'. The `when` directive allows you to specify conditions based on branch names, environment variables, parameters, or custom Groovy expressions.

Conclusion

Stages, steps, and post actions are the core components of a Declarative Pipeline in Jenkins. By understanding how to define and use these components effectively, you can create powerful and flexible pipelines that automate your CI/CD processes.

Stages provide a logical grouping of steps, allowing you to organize your pipeline into distinct phases. Steps define the individual actions and tasks to be executed within each stage. Post actions allow you to perform additional actions based on the

pipeline's outcome, such as publishing artifacts, sending notifications, or triggering downstream jobs.

In the next lesson, we'll explore how to write a complete Jenkinsfile using Declarative Pipeline syntax and learn best practices for creating maintainable and efficient pipelines.

Get ready to put your knowledge of stages, steps, and post actions into practice and create robust Declarative Pipelines in Jenkins!