Jenkins pipeline

Jenkins pipeline is a set of modules or plugins which enable the implementation and integration of Continuous Delivery pipelines within Jenkins.

The Jenkins pipeline has an expandable automation system for building basic or complicated 'template' distribution pipelines via the Domain-specific language (DSL) used in the pipeline.

There are four states of Continuous Delivery in Jenkins pipeline-

- Build
- Deploy
- Test
- Release

To implement pipeline as code, a Jenkinsfile needs to be present at the project's root repository.

Jenkinsfile supports two different syntax:

- Declarative
- Scripted

Advantage of the many features of Pipeline:

- Code: Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline.
- Durable: Pipelines can survive both planned and unplanned restarts of the Jenkins controller.
- Pausable: Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.
- Versatile: Pipelines support complex real-world CD requirements, including the ability to fork/join, loop, and perform work in parallel.
- Extensible: The Pipeline plugin supports custom extensions to its DSL footnote:dsl:[] and multiple options for integration with other plugins.

Example for multiple stages, where each stage performs a specific task:

```
pipeline {
    agent any
    stages {
        stage ('Build') {
            ...
        }
        stage ('Deploy') {
            ...
        }
        stage('Monitor'){
            ...
        }
    }
}
```

The Declarative Pipelines is a relatively new feature that supports the concept of code pipeline. It enables the reading and writing of the pipeline code. This code is written within a Jenkinsfile, which can be tested into a tool such as Git for source control.

The Scripted pipeline is a typical method of code writing. The Jenkinsfile is written on the Jenkins user interface instance in this pipeline.

Jenkins Pipeline Concepts

Term	Description
Pipeline	The pipeline is a set of instructions given in the form of code for continuous delivery and consists of instructions needed for the entire build process. With pipeline, you can build, test, and deliver the application.
Node	The machine on which Jenkins runs is called a node. A node block is mainly used in scripted pipeline syntax.
Stage	A stage block contains a series of steps in a pipeline. That is, the build, test, and deploy processes all come together in a stage. Generally, a stage block is used to visualize the Jenkins pipeline process.
Step	A step is nothing but a single task that executes a specific process at a defined time. A pipeline involves a series of steps.

Pipeline

It is a user-defined framework that includes all the processes like create, check, deploy, etc. In a Jenkinsfile, it's a list of all the levels. All of the stages and steps within this block are described. This is the fundamental block to the syntax of a declarative pipeline.

pipeline {

}

Node

A node is a system running a complete workflow. It's an integral part of the syntax of the scripted pipeline.

node {

}

Agent

An agent is described as a directive that can run multiple builds using just one Jenkins instance. This feature helps spread the workload to various agents and execute multiple projects within Jenkins's single instance. It instructs Jenkins to assign the builds to an executor.

A single agent may be defined for a whole Jenkins pipeline, or different agents may be assigned to execute each stage within a pipeline. Some of the most commonly used Agent parameters are:

1. Any

   Runs the stage pipeline on any available agent.

2. None

   This parameter is added to the root of the pipeline. It means that there is no global agent for the entire pipeline, and each stage must define its own agent.

3. Label

   Performs on the labeled agent the pipeline/stage.

4. Docker

This parameter uses a docker container as a pipeline execution environment or as a specific level. For example, the docker can be used to pull an image of Ubuntu. This image can now be used to run multiple commands as an execution environment.

Stages

This section includes all of the work that needs to be completed. The work is defined in the form of stages. Within this Directive, there may be more than one level. Each stage executes a particular task.

```
pipeline {

agent any

   stages {

      stage ('Build') {



      }

      stage ('Test') {



      }

      stage ('QA') {



      }

      stage ('Deploy') {



      }

      stage ('Monitor') {
```

```
    }
  }
}
```

Steps

Within a stage block, the pipeline can be described as a series of steps. Such steps are performed in sequence for the execution of a level. Within a Steps guideline, there must be at least one step.

```
pipeline {
  agent any
    stages {
      stage ('Build') {
        steps {
          echo
          'Running build phase. '
        }
      }
    }
  }
```

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.

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.
- JenkinsFile can be defined by either Web UI or with a Jenkins File.

Parallel stages allow you to run multiple stages of a pipeline simultaneously. This can be useful if you have multiple stages independent of one another and can be run concurrently.

For example, you might have a pipeline that builds and tests a software project and wants to run the build and test stages in parallel to save time.