Day 26 Task: Jenkins Declarative Pipeline

One of the most important parts of your DevOps and CICD journey is a Declarative Pipeline Syntax of Jenkins

What is Pipeline - A pipeline refers to a sequence of connected steps or processes that are executed in a specific order to transform input data into output data.

Pipelines are an effective way to automate and streamline complex workflows, reducing the likelihood of errors and improving overall efficiency. They can also help to ensure that processing steps are consistent and reproducible.

Declarative: Declarative Pipeline Syntax is a simpler and more opinionated syntax for defining pipelines, where the focus is on declaring what the pipeline should do, rather than how it should do it. The Declarative Pipeline Syntax is designed to make it easier to write and understand pipelines, with pre-defined stages, steps, and directives.

Scripted: The Scripted Pipeline Syntax is a more flexible and powerful way to define pipelines, compared to the Declarative Pipeline Syntax. It is a Groovy-based syntax that provides a programmatic way of defining pipelines, with more control over the pipeline flow and more flexibility in defining the pipeline stages and steps.

Why you should have a Pipeline

The definition of a Jenkins Pipeline is written into a text file (called a Jenkinsfile) which in turn can be committed to a project's source control repository.
This is the foundation of "Pipeline-as-code"; treating the CD pipeline as a part of the application to be versioned and reviewed like any other code

There are several reasons why having a Pipeline is important for DevOps and software development:

1. Standardization: A Pipeline and Jenkinsfile provide a standardized way to define and execute the build, test, and deployment process. This ensures that the process is consistent and repeatable, reducing the risk of errors and making it easier to troubleshoot issues.

2. Automation: A Pipeline and Jenkinsfile allow for the automation of the build, test, and deployment process, reducing the need for manual intervention and saving time and effort.

3. Version Control: A Jenkinsfile can be version-controlled along with the source code, making it easier to track changes and maintain the pipeline over time.

4. Portability: A Pipeline and Jenkinsfile are platform-independent, making it possible to execute the same pipeline on different systems and platforms. This portability ensures that the pipeline can be easily moved to different environments, such as test and production environments.

5. Integration: A Pipeline and Jenkinsfile can be integrated with other tools and systems, such as Git, Docker, and Kubernetes, making it possible to streamline the entire software development and deployment process.

6. Collaboration: A Pipeline and Jenkinsfile can be shared among different teams and stakeholders, making it easier to collaborate and communicate during the software development process.

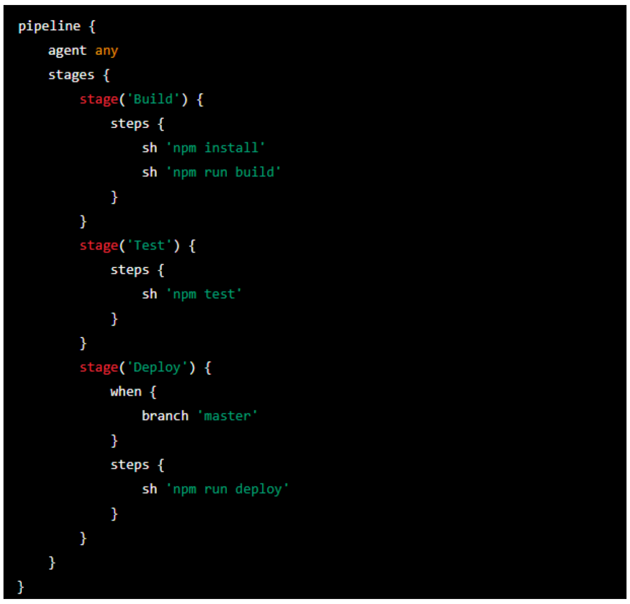
Overall, having a Pipeline and Jenkinsfile is an important part of a modern DevOps and software development process. They provide standardization, automation, version control, portability, integration, and collaboration, all of which are essential for delivering high-quality software efficiently and reliably.

Pipeline syntax

The Pipeline syntax in Jenkins is defined using two different syntaxes: Declarative and Scripted. Here is an overview of the syntax for each:

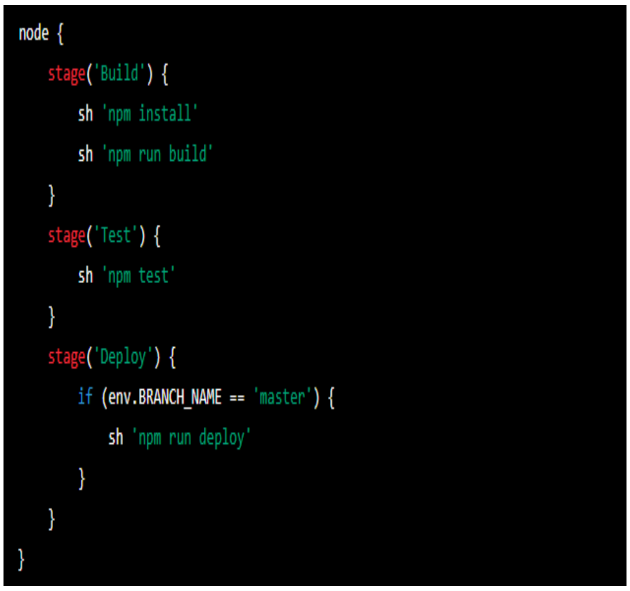
1. Declarative Pipeline Syntax: The Declarative Pipeline Syntax is a more recent addition to Jenkins, and it is designed to provide a simpler, more structured syntax for defining pipelines. It uses a YAML-based syntax, and each pipeline is defined using the pipeline block.

   Here is an example of a simple Declarative Pipeline:



```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                sh 'npm install'
                sh 'npm run build'
            }
        }
        stage('Test') {
            steps {
                sh 'npm test'
            }
        }
        stage('Deploy') {
            when {
                branch 'master'
            }
            steps {
                sh 'npm run deploy'
            }
        }
    }
}
```

2. Scripted Pipeline Syntax: The Scripted Pipeline Syntax is a more flexible syntax for defining pipelines. It is based on the Groovy programming language and provides more control and flexibility over the pipeline flow and stages.
   Here is an example of a simple Scripted Pipeline:

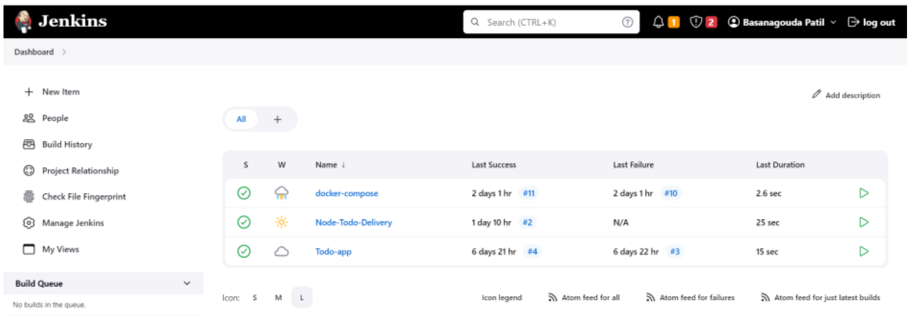


```groovy
node {
    stage('Build') {
        sh 'npm install'
        sh 'npm run build'
    }
    stage('Test') {
        sh 'npm test'
    }
    stage('Deploy') {
        if (env.BRANCH_NAME == 'master') {
            sh 'npm run deploy'
        }
    }
}
```

Task-01

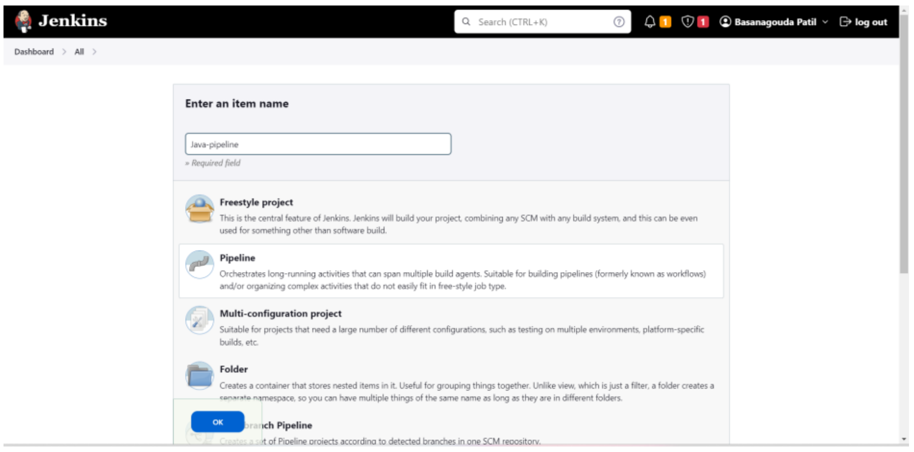
- Create a New Job, this time select Pipeline instead of Freestyle Project.
- Follow the Official Jenkins [Hello world example](#)

1. Start jenkins server and click on “New Item” to create a job.



S	W	Name ↓	Last Success	Last Failure	Last Duration
✓		docker-compose	2 days 1 hr #11	2 days 1 hr #10	2.6 sec
✓		Node-Todo-Delivery	1 day 10 hr #2	N/A	25 sec
✓		Todo-app	6 days 21 hr #4	6 days 22 hr #3	15 sec

2. Enter the desired name and select “Pipeline” then click on Ok.



Enter an item name

Java-pipeline

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

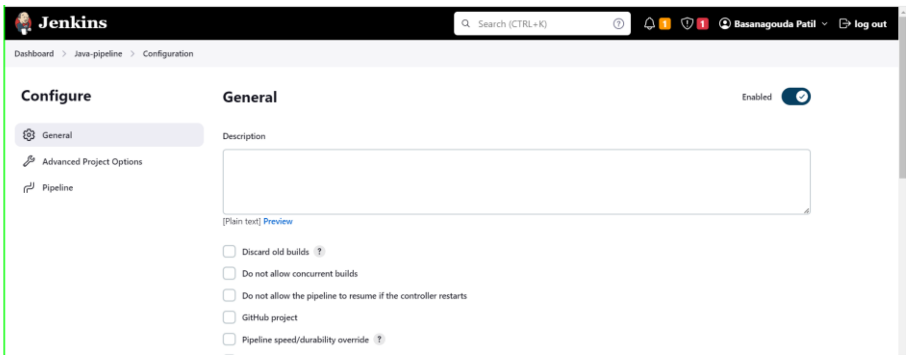
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

ch Pipeline
t of Pipeline projects according to detected branches in one SCM repository.

3. In General, give the description for the job. Scroll down and in the "Pipeline" section



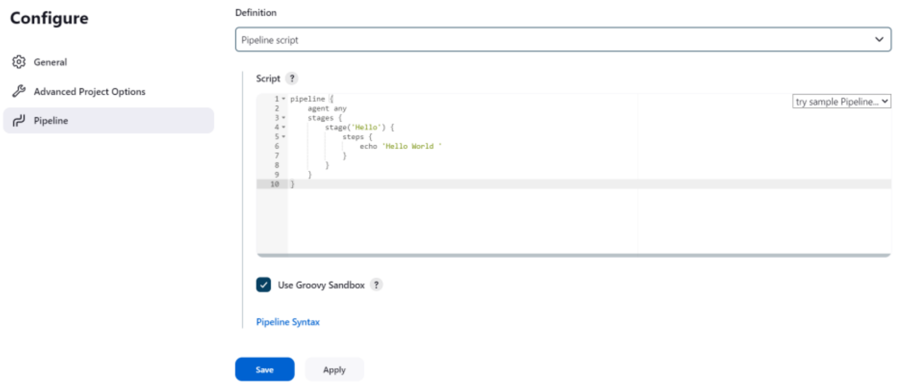
General — Enabled

Description

[Plain text] Preview

- [ ] Discard old builds
- [ ] Do not allow concurrent builds
- [ ] Do not allow the pipeline to resume if the controller restarts
- [ ] GitHub project
- [ ] Pipeline speed/durability override

4. In definition select Pipeline script, start typing the code for building a pipeline and click on Save button.



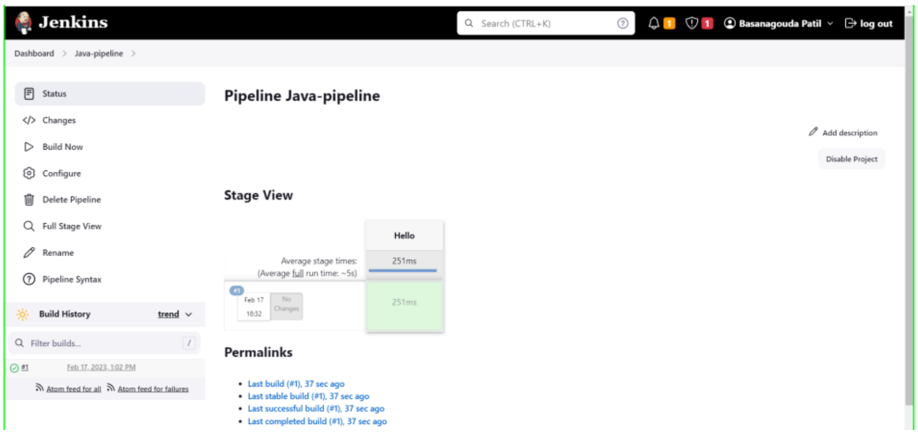
Definition: Pipeline script

Script

```
pipeline {
    agent any
    stages {
        stage('Hello') {
            steps {
                echo 'Hello World '
            }
        }
    }
}
```

- [✓] Use Groovy Sandbox

5. Click on "Build Now" and job will be build successfully.



Pipeline Java-pipeline

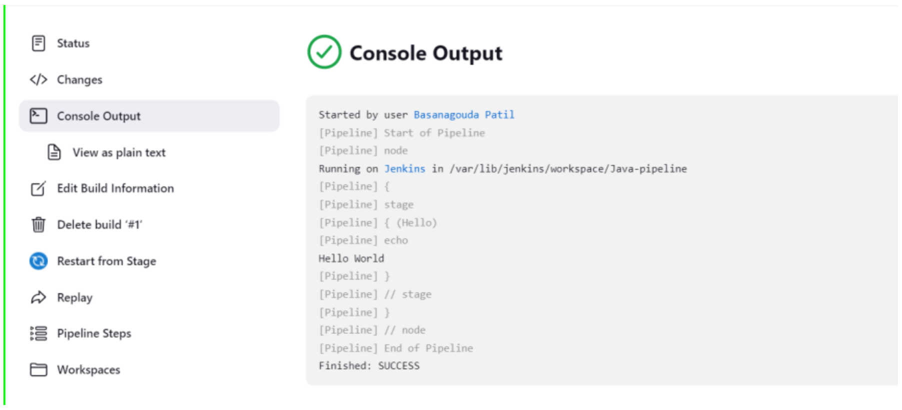
Stage View

	Hello
Average stage times: (Average full run time: ~5s)	251ms
#1 Feb 17 18:32 No Changes	251ms

Permalinks

- Last build (#1), 37 sec ago
- Last stable build (#1), 37 sec ago
- Last successful build (#1), 37 sec ago
- Last completed build (#1), 37 sec ago

6. After a build is completed, you can view the console output in the build page.



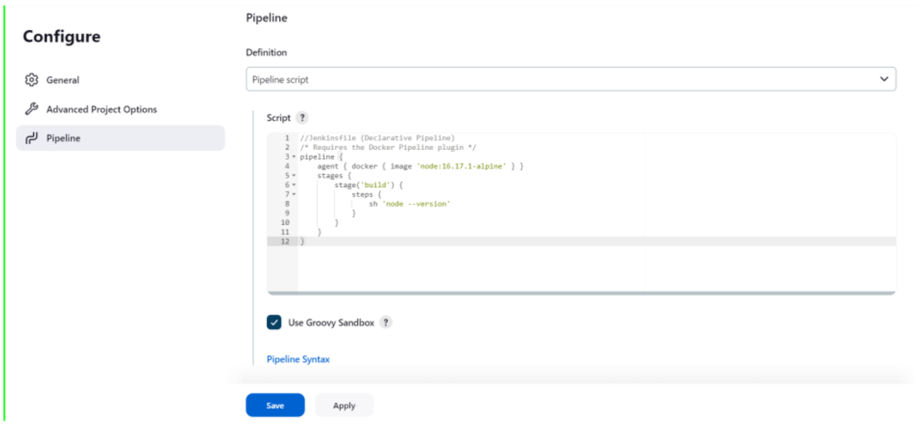
Console Output

```
Started by user Basanagouda Patil
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/Java-pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Hello)
[Pipeline] echo
Hello World
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

- Complete the example using the Declarative pipeline

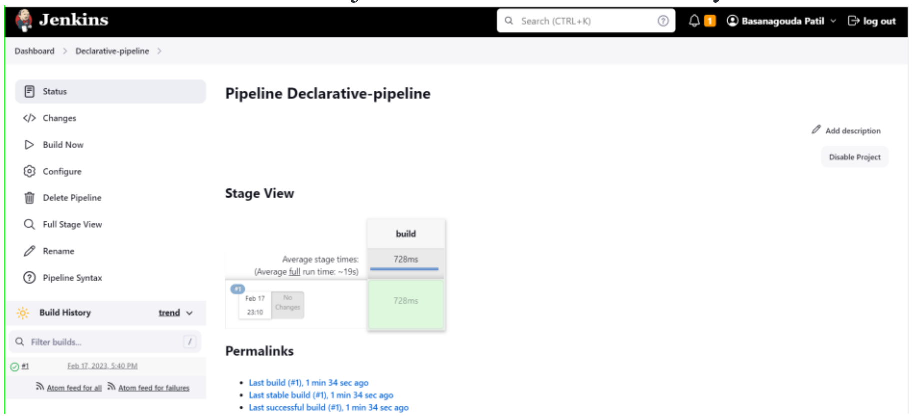
1. Login to your Jenkins and Install the Docker Pipeline plugin through the Manage Jenkins > Manage Plugins page.
2. After installing docker pipeline plugin click on “New Item”.
3. Enter the desired name and select “Pipeline” then click on Ok.
4. In general give the description for the job. Scroll down and in the “Pipeline” section and in definition select Pipeline script, start typing the code for building a pipeline and click on Save button.

```
//Jenkinsfile (Declarative Pipeline)
/* Requires the Docker Pipeline plugin */
pipeline {
    agent { docker { image 'node:16.17.1-alpine' } }
    stages {
        stage('build') {
            steps {
                sh 'node --version'
            }
        }
    }
}
```



```
//Jenkinsfile (Declarative Pipeline)
/* Requires the Docker Pipeline plugin */
pipeline {
    agent { docker { image 'node:16.17.1-alpine' } }
    stages {
        stage('build') {
            steps {
                sh 'node --version'
            }
        }
    }
}
```

5. Click on “Build Now” and job will be build successfully.



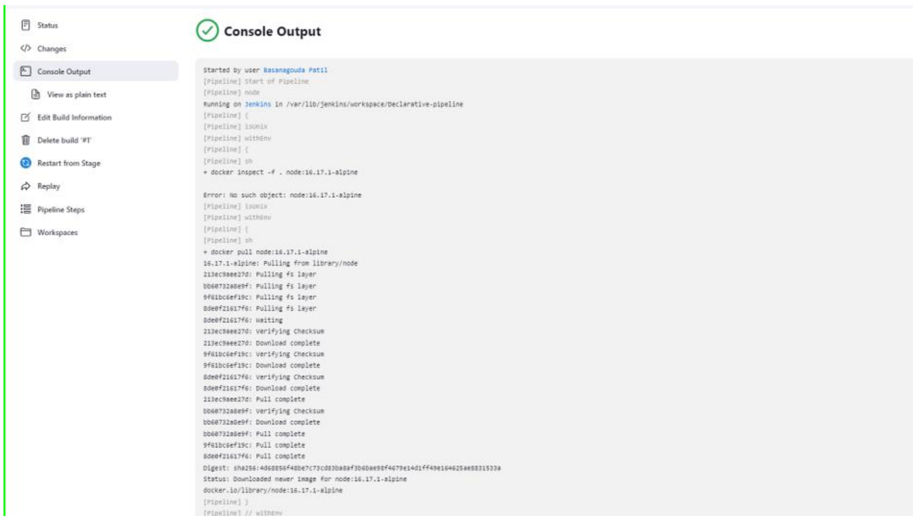
Pipeline Declarative-pipeline

Average stage times:
(Average full run time: ~19s)

Permalinks

- Last build (#1), 1 min 34 sec ago
- Last stable build (#1), 1 min 34 sec ago
- Last successful build (#1), 1 min 34 sec ago

6. After a build is completed, you can view the console output in the build page.



```
Started by user Basanagouda Patil
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/Declarative-pipeline
[Pipeline] {
[Pipeline] isUnix
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
+ docker inspect -f . node:16.17.1-alpine

Error: No such object: node:16.17.1-alpine
[Pipeline] isUnix
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
+ docker pull node:16.17.1-alpine
16.17.1-alpine: Pulling from library/node
213ec9aee27d: Pulling fs layer
bb60732a8e9f: Pulling fs layer
9f61bc6ef19c: Pulling fs layer
8de0f21617f6: Pulling fs layer
8de0f21617f6: Waiting
213ec9aee27d: Verifying Checksum
213ec9aee27d: Download complete
9f61bc6ef19c: Verifying Checksum
9f61bc6ef19c: Download complete
8de0f21617f6: Verifying Checksum
8de0f21617f6: Download complete
213ec9aee27d: Pull complete
bb60732a8e9f: Verifying Checksum
bb60732a8e9f: Download complete
bb60732a8e9f: Pull complete
9f61bc6ef19c: Pull complete
8de0f21617f6: Pull complete
Digest: sha256:4d68856f48be7c73cd83ba8af3b6bae98f4679e14d1ff49e164625ae8831533a
Status: Downloaded newer image for node:16.17.1-alpine
docker.io/library/node:16.17.1-alpine
[Pipeline] }
[Pipeline] // withEnv
```

Happy Learning:)