DevOps Lab

Assignment 5:

Aim: To understand the usage of Jenkins to build a simple pipeline to automate the process of building, testing and deploying.

Theory & Execution:

Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.
A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment.
Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the Pipeline domain-specific language (DSL) syntax.
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 a part of the application to be versioned and reviewed like any other code.

Creating a Jenkinsfile and committing it to source control provides a number of immediate benefits:

i) Automatically creates a Pipeline build process for all branches and pull requests.

ii) Code review/iteration on the Pipeline (along with the remaining source code).

iii) Audit trail for the Pipeline.

iv) Single source of truth for the Pipeline, which can be viewed and edited by multiple members of the project.

While the syntax for defining a Pipeline, either in the web UI or with a Jenkinsfile is the same, it is generally considered best practice to define the Pipeline in a Jenkinsfile and check that in to source control.

Declarative V/S Scripted Pipeline syntax:

A Jenkinsfile can be written using two types of syntax - Declarative and Scripted. Declarative and Scripted Pipelines are constructed fundamentally differently. Declarative Pipeline is a more recent feature of Jenkins Pipeline which:

i) provides richer syntactical features over Scripted Pipeline syntax, and

ii) is designed to make writing and reading Pipeline code easier.

Many of the individual syntactical components (or "steps") written into a Jenkinsfile, however, are common to both Declarative and Scripted Pipeline. Read more about how these two types of syntax differ in Pipeline concepts and Pipeline syntax overview below.

Jenkins is, fundamentally, an automation engine which supports a number of automation patterns. Pipeline adds a powerful set of automation tools onto Jenkins, supporting use cases that span from simple continuous integration to comprehensive CD pipelines. By modeling a series of related tasks, users can take advantage of the many features of Pipeline:

i) **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.

ii) **Durable**: Pipelines can survive both planned and unplanned restarts of the Jenkins master.

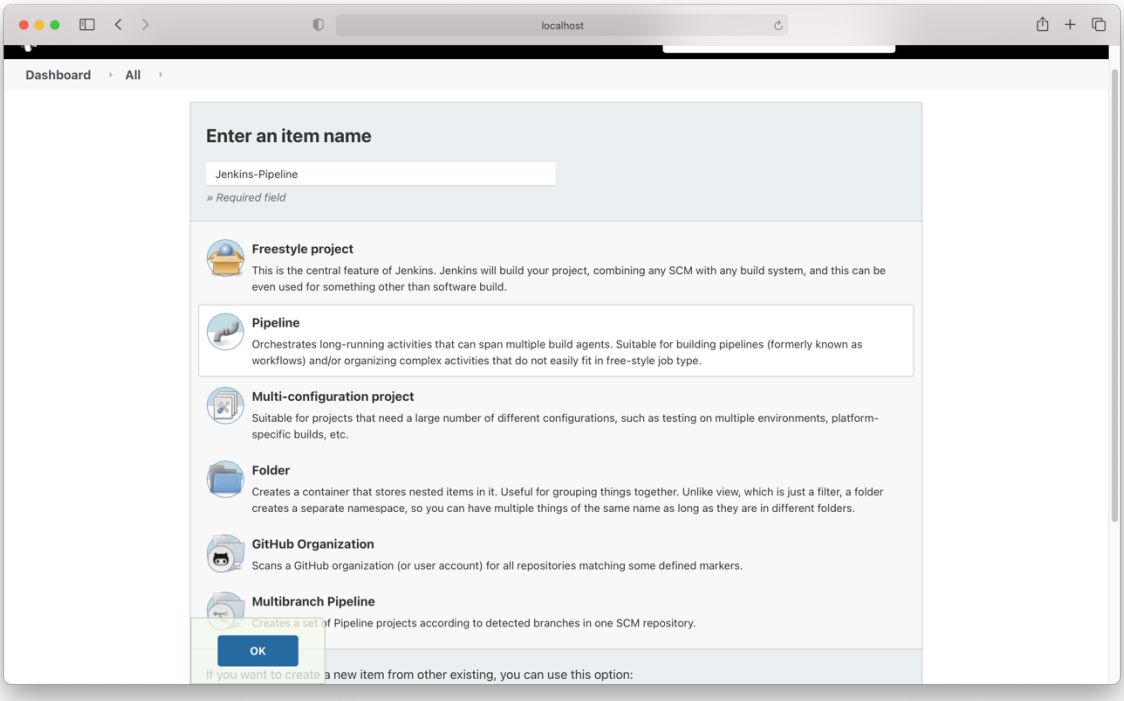
iii) **Pausable**: Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.

iv) **Versatile**: Pipelines support complex real-world CD requirements, including the ability to fork/join, loop, and perform work in parallel.

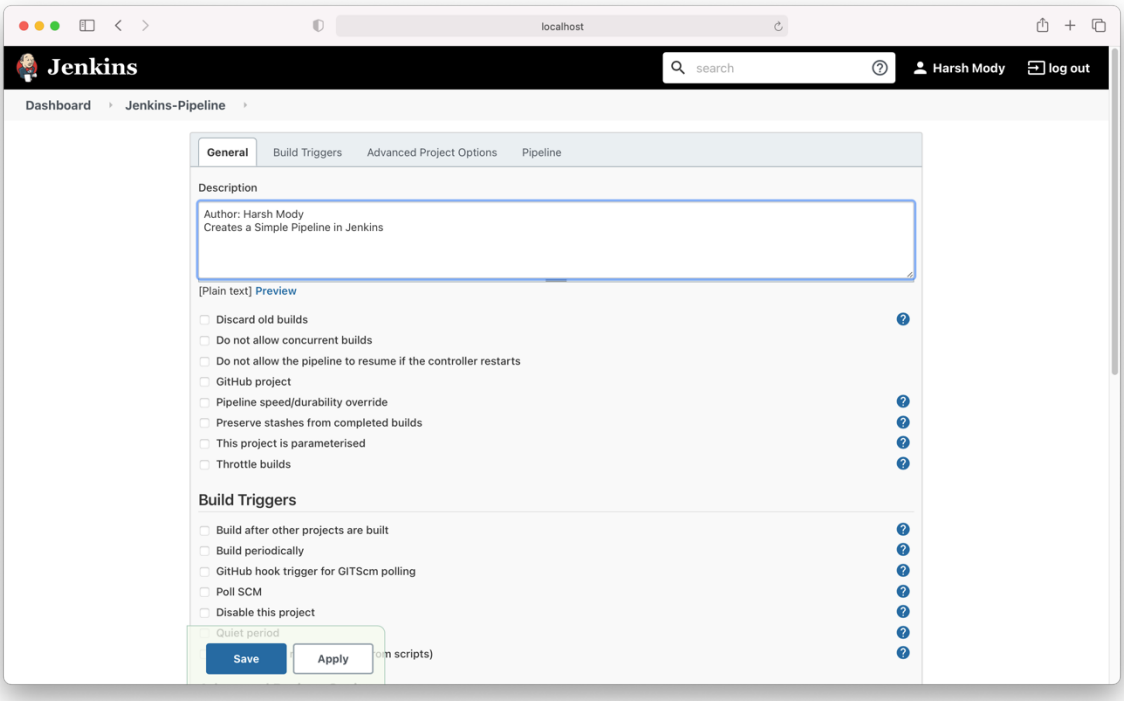
v) **Extensible**: The Pipeline plugin supports custom extensions to its DSL and multiple options for integration with other plugins.

While Jenkins has always allowed rudimentary forms of chaining Freestyle Jobs together to perform sequential tasks, Pipeline makes this concept a first-class citizen in Jenkins. Building on the core Jenkins value of extensibility, Pipeline is also extensible both by users with Pipeline Shared Libraries and by plugin developers.

To build a simple pipeline, we select to create a new pipeline and give the project a unique name.



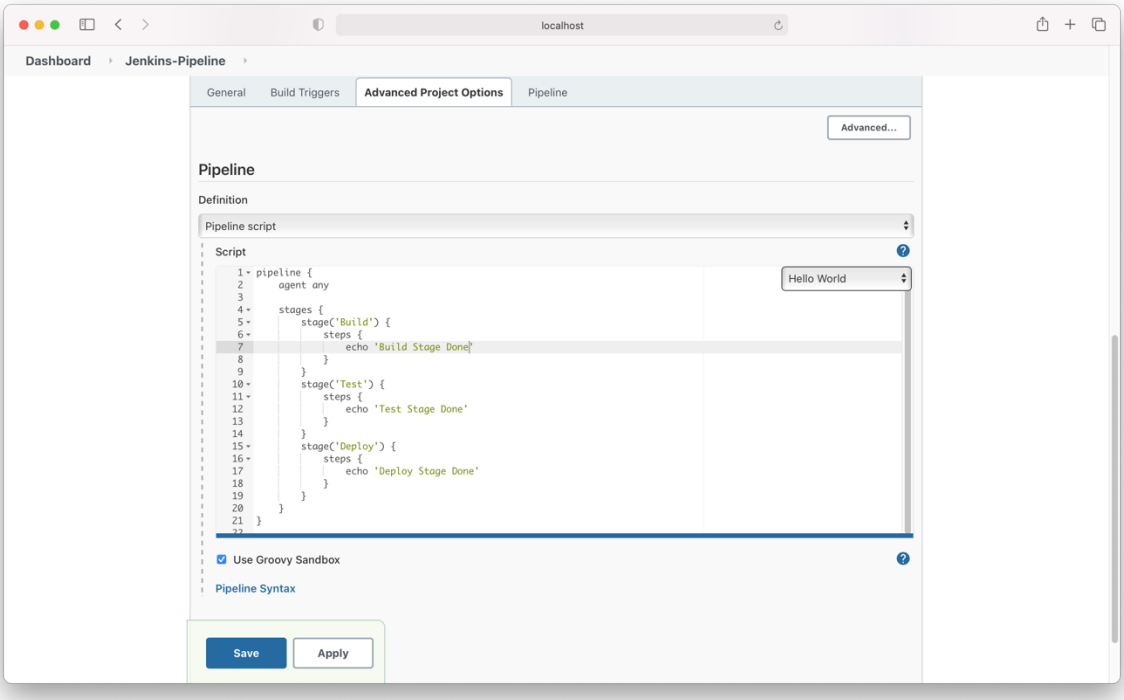
Add the appropriate settings for the project

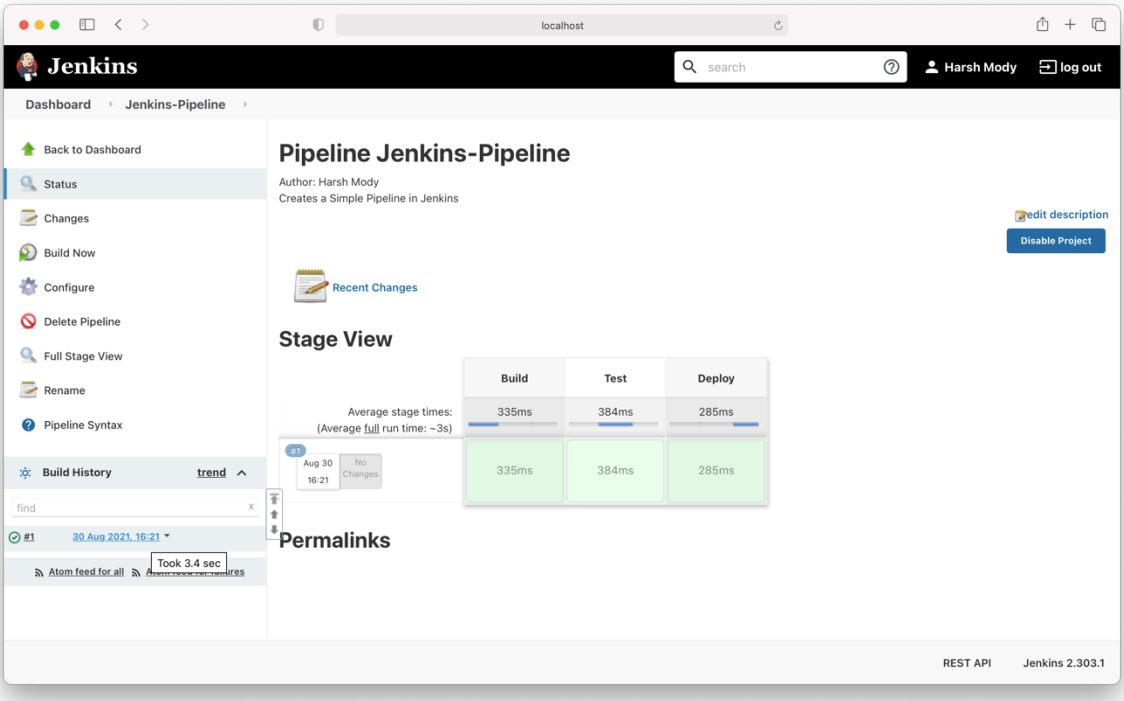


Write a simple pipeline script to create three stages i.e. Build, Test and Deploy and then Run the Build

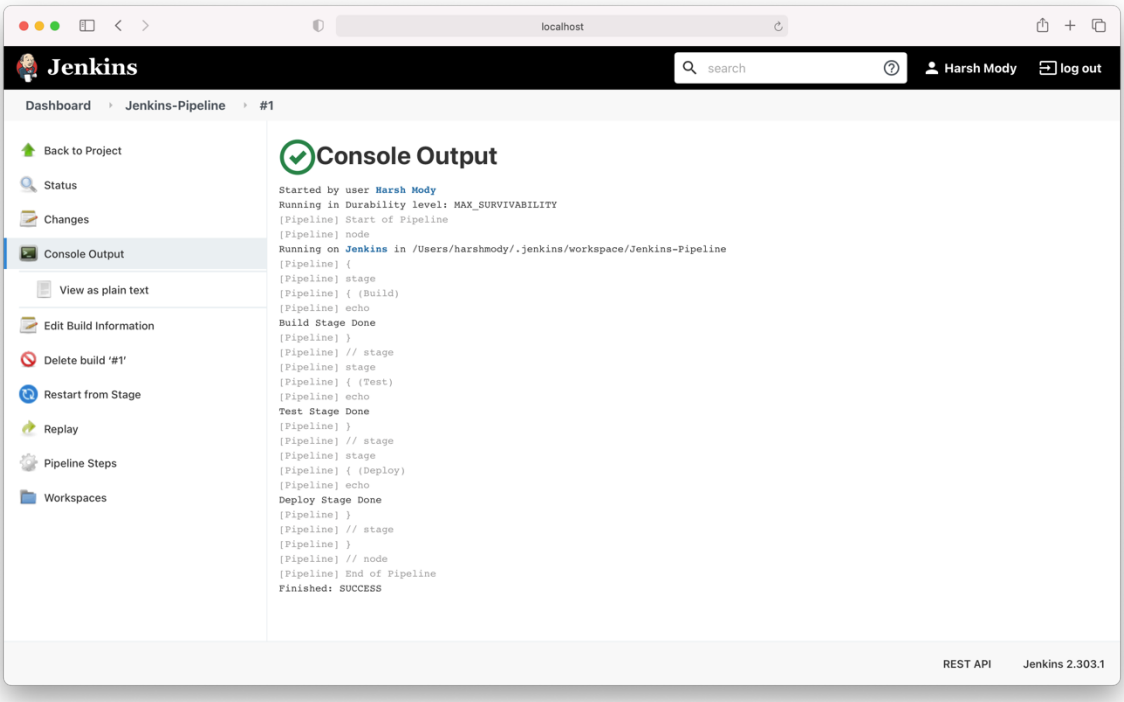
```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Build Stage Done'
            }
        }
        stage('Test') {
            steps {
                echo 'Test Stage Done'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploy Stage Done'
            }
        }
    }
}
```





After we see that all the three stages ran successfully, we can then go to the build history and see the console output.



Conclusion: Thus, successfully understood the importance Jenkins to create a simple pipeline using Jenkins.