# **Experiment-4**

Ashish Patil

D11AD 46

**Aim**:-To understand pipelining scripting using jenkins

**Theory:-**

Pipeline scripting in Jenkins refers to defining and managing continuous integration and continuous delivery (CI/CD) pipelines using code. Jenkins Pipeline provides a powerful way to express your CI/CD workflows as code, enabling teams to version control, review, and automate the entire pipeline process. Here's some theory about pipeline scripting and pipelining in Jenkins:

**1. Declarative vs. Scripted Pipeline :**

Jenkins Pipeline supports two syntaxes for defining pipelines: Declarative Pipeline and Scripted Pipeline.

- Declarative Pipeline : Declarative Pipeline provides a more structured and opinionated syntax for defining pipelines. It offers a simpler and more concise way to express common CI/CD workflows with less boilerplate code.

- Scripted Pipeline : Scripted Pipeline, on the other hand, offers more flexibility and expressive power as it allows you to write your pipeline using Groovy scripting language. It's suitable for complex workflows and scenarios where fine-grained control is required.

**2. Pipeline DSL :**

Pipeline scripting in Jenkins is based on the Groovy scripting language. Jenkins Pipeline provides a Domain-Specific Language (DSL) for defining pipeline stages, steps, conditions, and other constructs. This DSL allows you to express your pipeline logic using a set of predefined keywords, functions, and constructs provided by Jenkins.

**3. Pipeline Steps :**

Jenkins Pipeline offers a wide range of built-in steps (or functions) that you can use to perform various tasks within your pipeline. These steps cover actions such as checking out source code, building applications, running tests, publishing artifacts, deploying to environments, and sending notifications. Additionally, Jenkins supports the use of custom scripted steps to extend the functionality of pipelines.

**4. Pipeline Syntax and Structure :**

Pipelines in Jenkins typically consist of stages, steps, and directives. Stages represent distinct phases of the CI/CD process (e.g., build, test, deploy), while steps define the individual tasks to be executed within each stage. Directives provide additional configuration and control over the pipeline execution, such as parallel execution, error handling, and post-build actions.

**5. Pipeline as Code :**

Pipeline scripting enables teams to treat their CI/CD pipelines as code, allowing them to version control, review, and collaborate on pipeline definitions just like any other software code. This approach brings several benefits, including better visibility, reproducibility, and automation of the pipeline process.

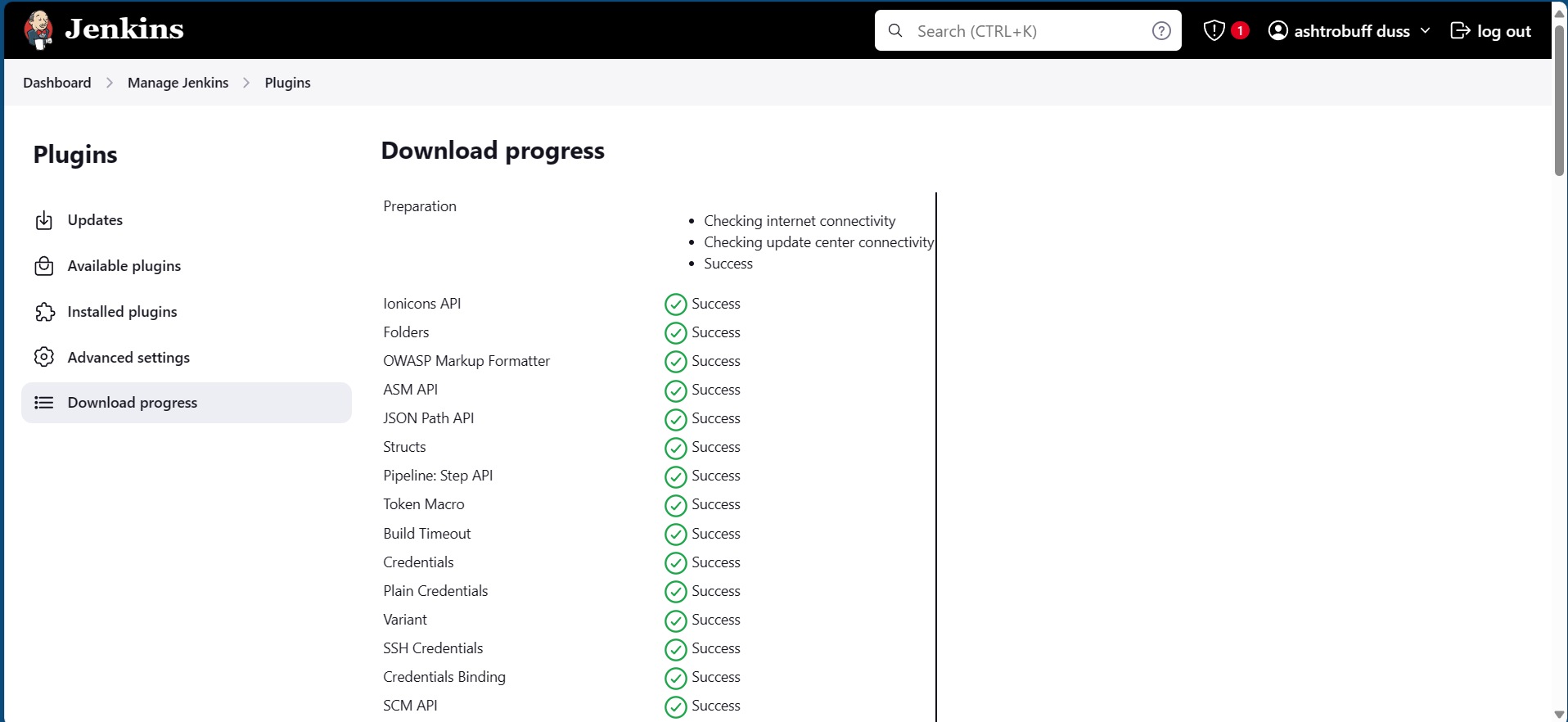
**6. Pipeline Libraries :**

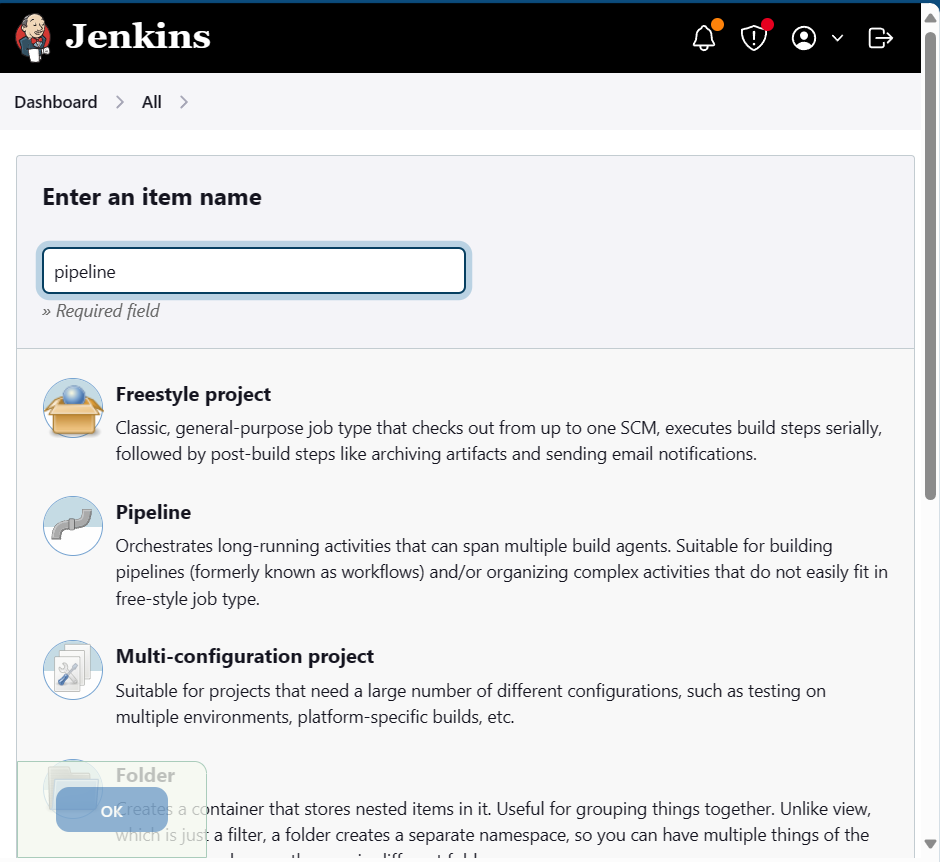
Jenkins Pipeline allows you to define reusable components and utilities as shared libraries, which can be used across multiple pipeline scripts. Pipeline libraries encapsulate common functionality, reducing duplication and promoting code reuse and consistency across pipelines.

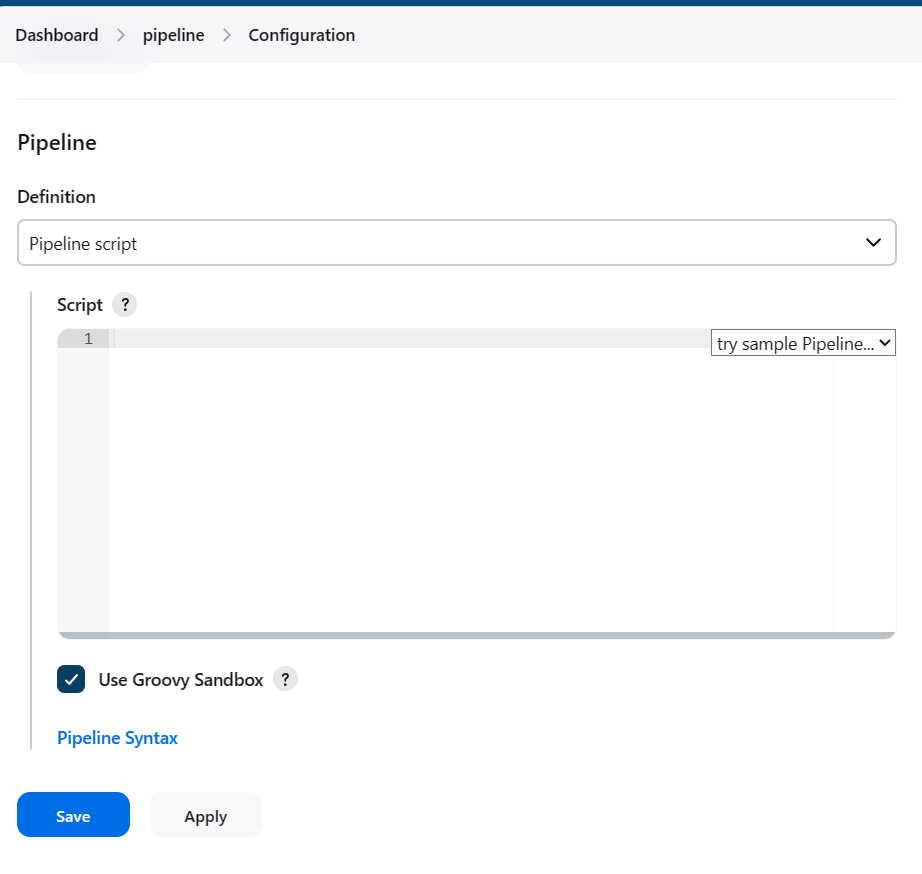
**7. Pipeline Visualization and Monitoring :**

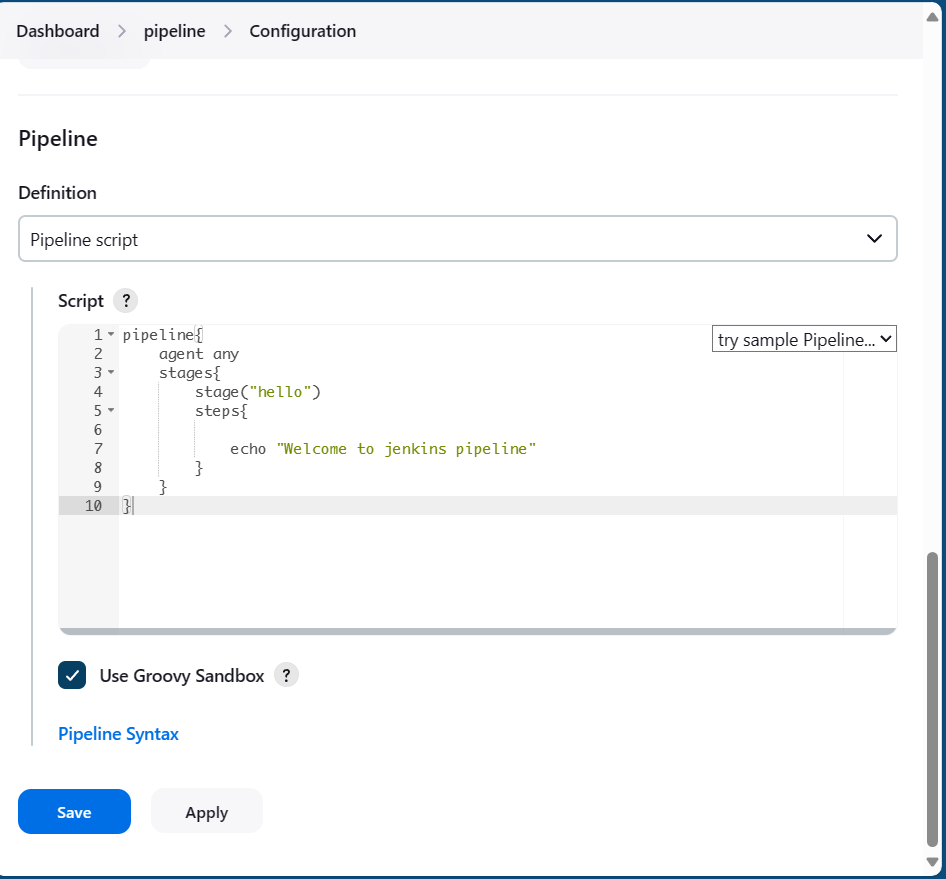
Jenkins provides visualization tools to monitor and visualize pipeline execution. You can view the progress of pipeline runs, visualize the stages and steps within each pipeline, and analyze build logs and test results to identify issues and bottlenecks.

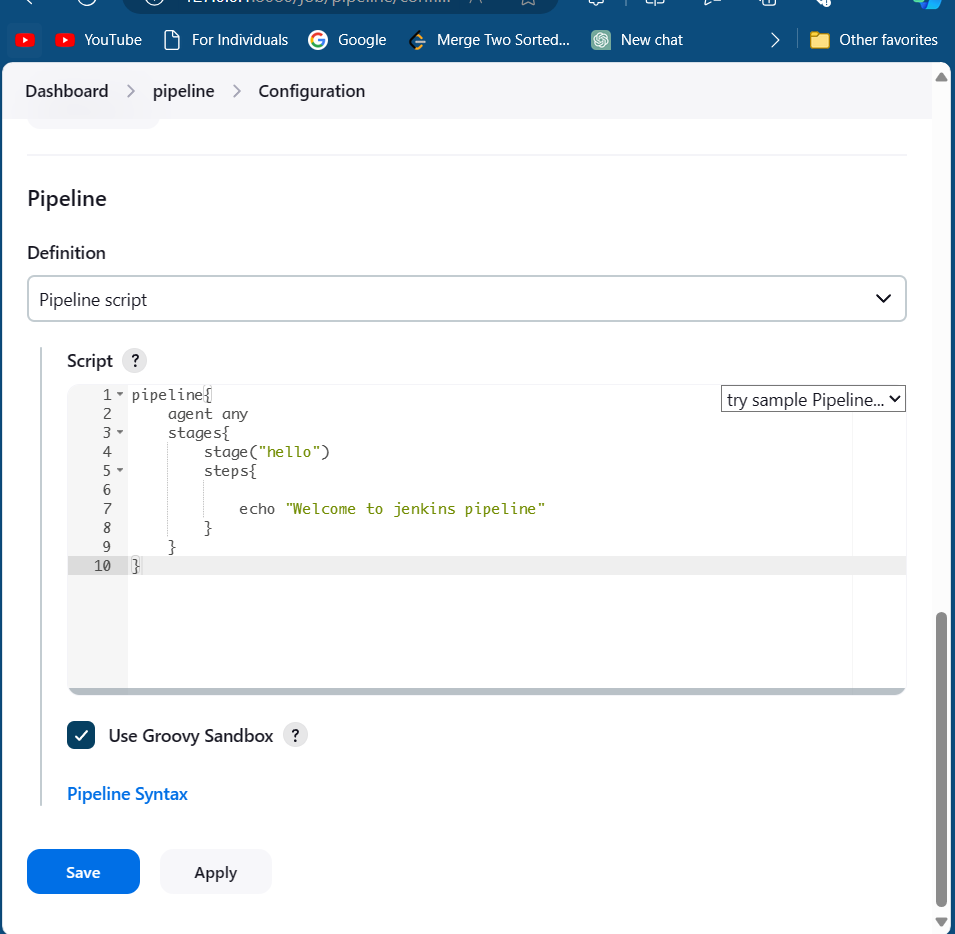
Understanding pipeline scripting and pipelining concepts is essential for designing, implementing, and maintaining efficient CI/CD workflows in Jenkins. It empowers teams to automate and streamline the software delivery process, resulting in faster, more reliable releases with improved quality.

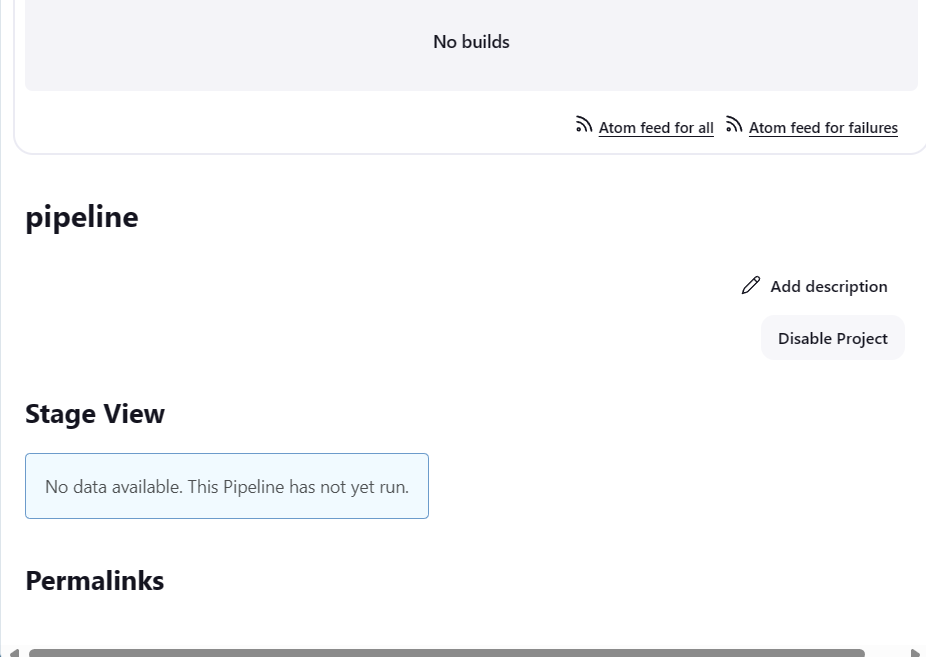


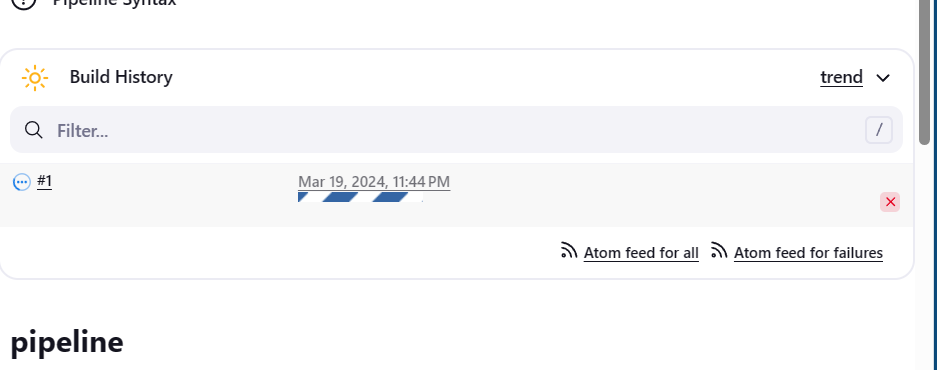


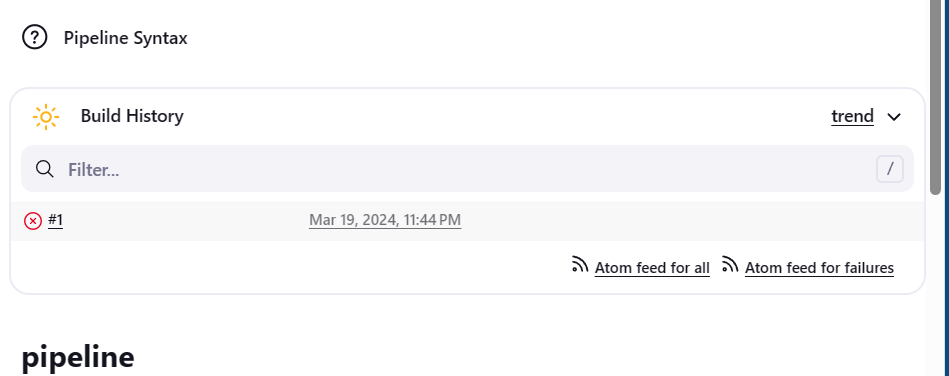


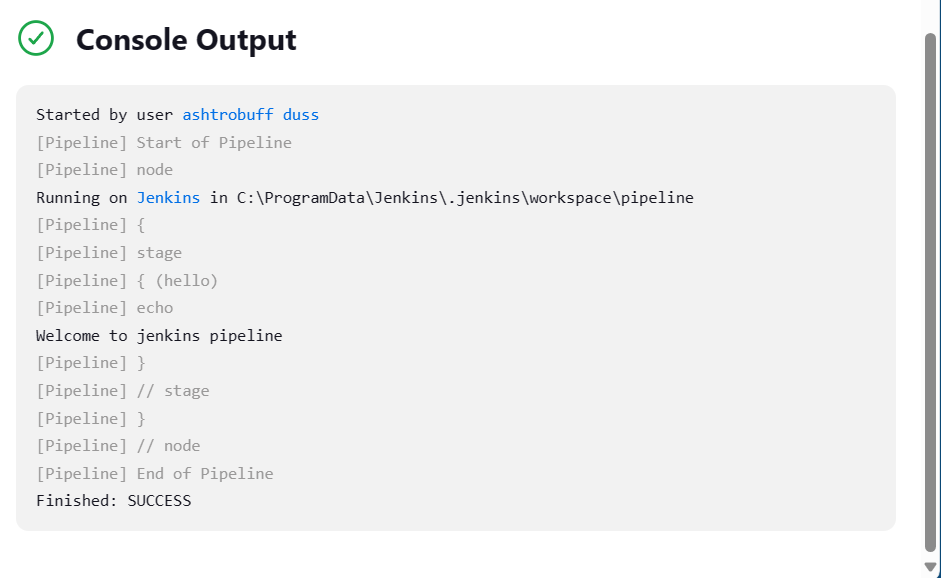












Conclusion: We have successfully implemented a pipeline in jenkins and run it in a build.