**Jenkins - Exploring core concepts**

## Core Jenkins Concepts:

### Objectives:

* Learn Jenkins Pipelines in-depth (Declarative vs Scripted)
* Use GitHub integration for automated builds
* Understand Jenkinsfile structure, syntax, and execution model
* Build a CI job for a real application (Node.js or Python)

### 1. Theoretical Foundations: Jenkins Pipelines

**What is a Jenkins Pipeline?**

A **Jenkins Pipeline** is a suite of plugins that supports integrating and implementing continuous delivery pipelines using code.

* **Pipeline as Code**: Defined in a Jenkinsfile (placed in the repo)
* **Version-controlled**: Pipeline logic changes are trackable via Git
* **Reproducible and portable**: Easy to recreate across environments

**Types of Pipelines:**

| **Type** | **Description** |
| --- | --- |
| Declarative | Simpler, structured, preferred for most use cases |
| Scripted | Based on Groovy, flexible but complex and verbose |

**Freestyle Jobs:**

Freestyle jobs are simple GUI-based configurations where each build step is manually defined. Ideal for initial setup and learning, but they lack version control and scalability.

**Pipeline Jobs:**

Pipeline jobs use Jenkinsfile for pipeline-as-code. This ensures pipelines are version-controlled, reproducible, and can implement advanced logic like conditionals, loops, and parallelism.

### 2. Declarative vs Scripted Pipeline:

| **Feature** | **Declarative** | **Scripted** |
| --- | --- | --- |
| Syntax | Structured, YAML-like blocks | Free-form Groovy code |
| Readability | High | Medium to Low |
| Use Case | CI/CD for most applications | Complex logic, custom loops, etc. |

### 3. What Is a Webhook in CI/CD?

A **webhook** is a way for one system (like GitHub) to notify another system (like Jenkins) when an event occurs — for example, **code is pushed to the repo**.

This allows Jenkins to **automatically trigger a build** instead of you clicking “Build Now” manually.

**Common Trigger Events:**

* push (default for CI)
* pull\_request (for PR validation)
* tag (for release builds)

| **Topic** | **Key Point** |
| --- | --- |

|  |  |
| --- | --- |
| Webhook vs Polling | Webhook is push-based (efficient), polling is pull-based (resource-intensive) |

|  |  |
| --- | --- |
| Payload URL | Must end with /github-webhook/ for GitHub plugin to detect it |

|  |  |
| --- | --- |
| Build Triggers in Jenkins | Located in job config under "Build Triggers" |

|  |  |
| --- | --- |
| Why Webhooks? | Enables real-time CI/CD and removes manual steps |

### 4. Jenkins Pipeline Stages

| **Block** | **Purpose** |
| --- | --- |
| pipeline | Root block |
| agent | Defines where the pipeline runs (can be any, label, docker, etc.) |
| stages | Contains multiple steps logically grouped |
| steps | Actual shell or Jenkins commands |
| post | Always/Success/Failure blocks to define post-build behaviour |

**Practical 02: Node.js Sample Project(sample-node-app)**

* You have a simple GitHub project (Node.js or Python)
* Jenkins pulls the code when there’s a change
* Builds it and prints output
* Prerequisites:

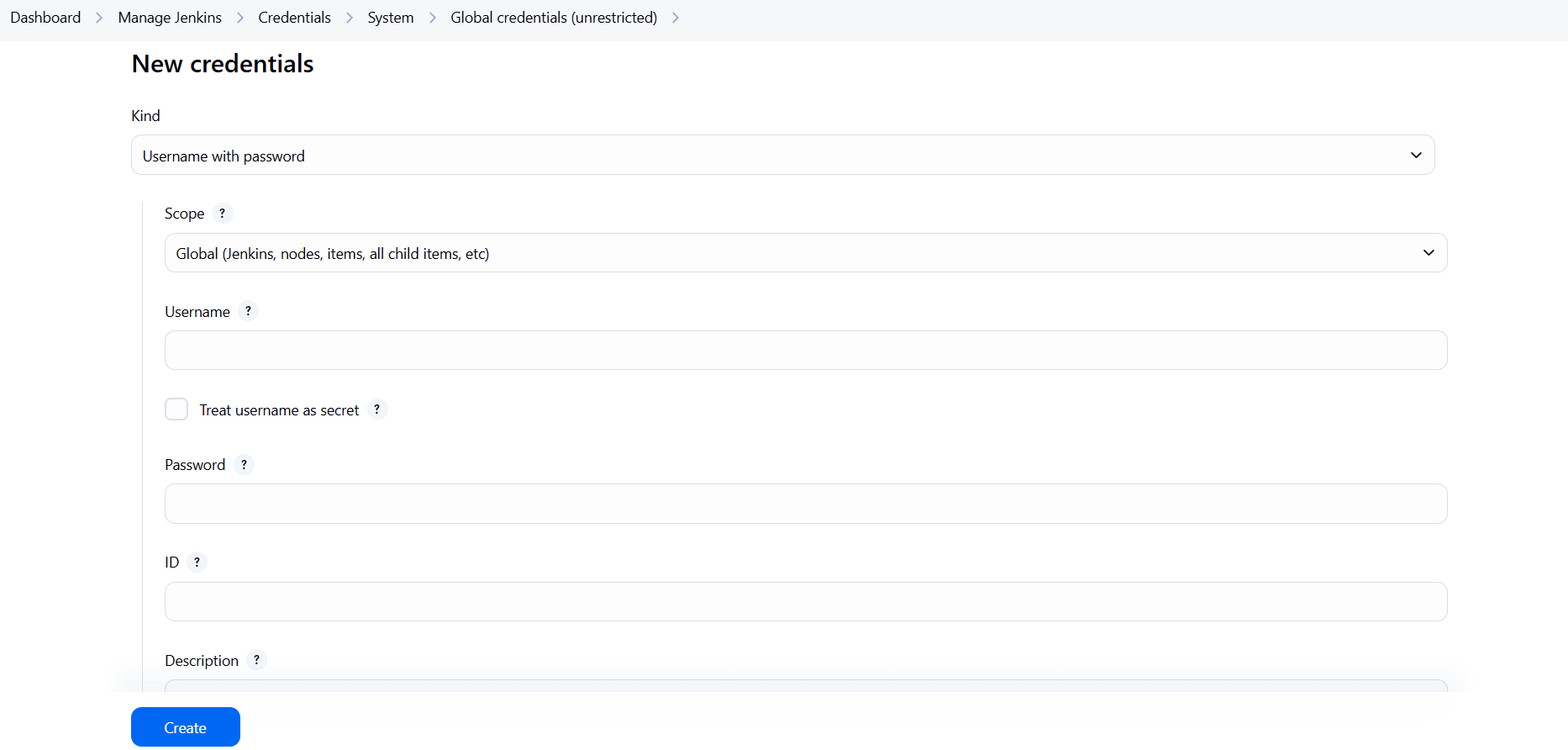
1. Install git, npm and node in the EC2 server.

sudo apt install git npm nodejs -y

1. Create a repository (sample-node-app) in github.
2. Setup git global configurations in the instance.
3. Go to sample-node-app directory (cd sample-node-app) and put all the codes of each file from this repo. (<https://github.com/AjinkyaP-09/simple-node-app.git>)
4. Push the code to the repo (git push origin main/master).
5. Add the github credentials to Jenkins:

Go to Manage Jenkins → Credentials → System → Global credentials.

Add your git username and PAT (Personal Access Token, generate from account securities) and save.



1. **Freestyle Job:**
2. Create a folder structure as below in server.

**Folder structure:**

sample-node-app/

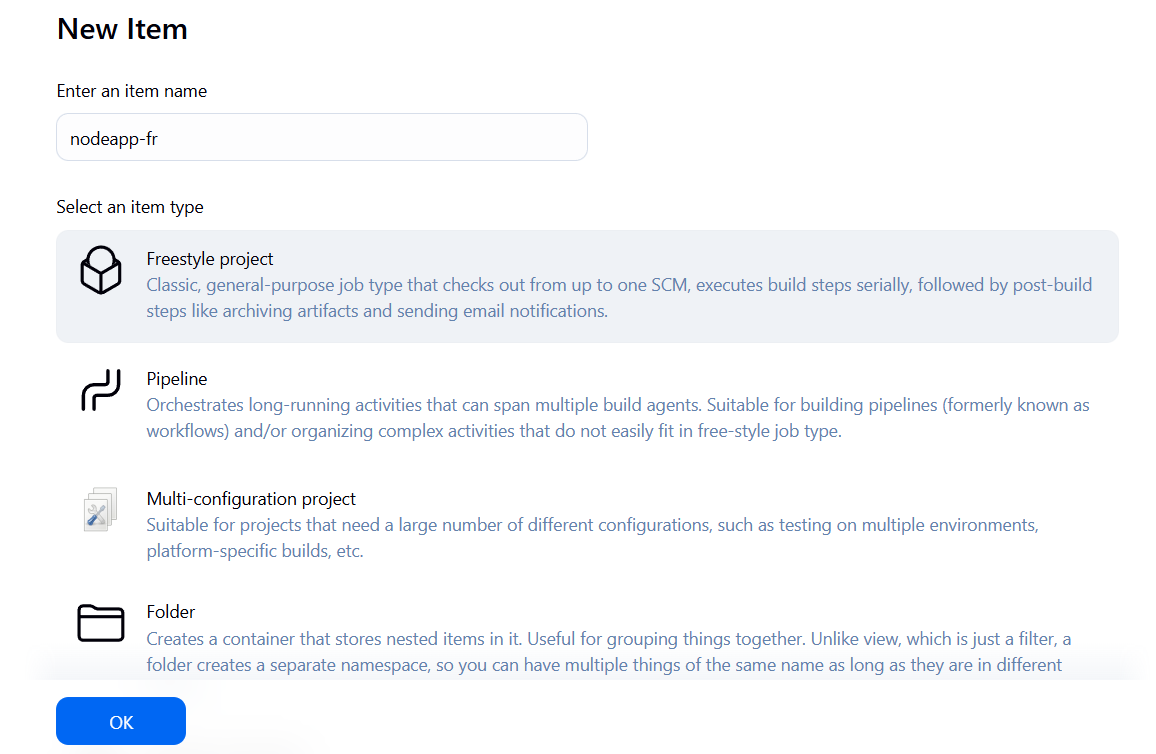
├── app.js

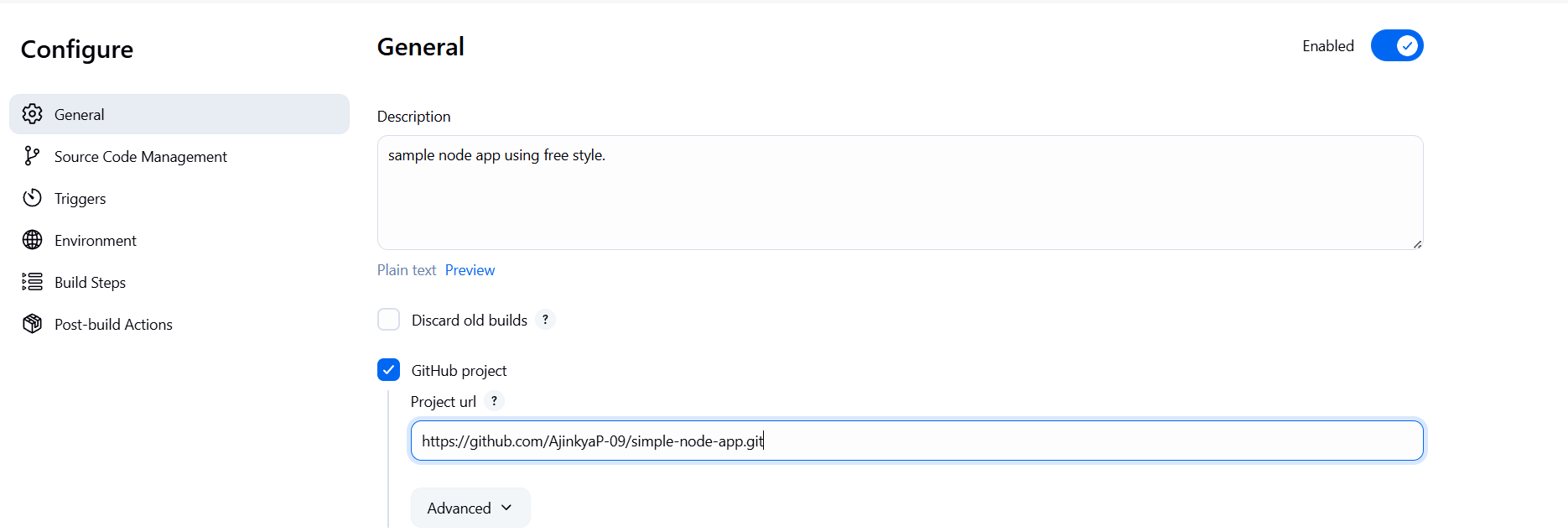
├── package.json

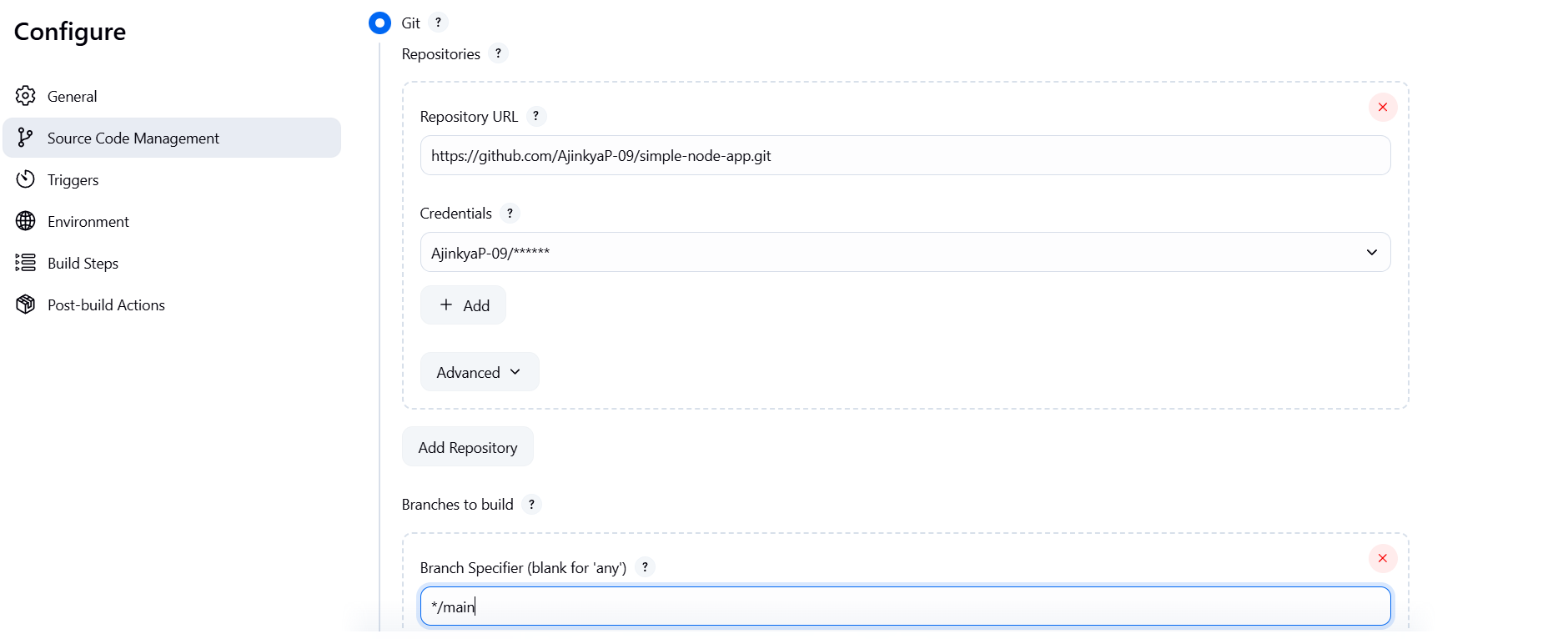
└── test/

└── app.test.js

1. Create new item from Jenkins Dashboard.
2. Enter description and select Github Project & enter repository link.
3. Select SCM as Git.
4. Enter repo link, select credentials and enter branch name.



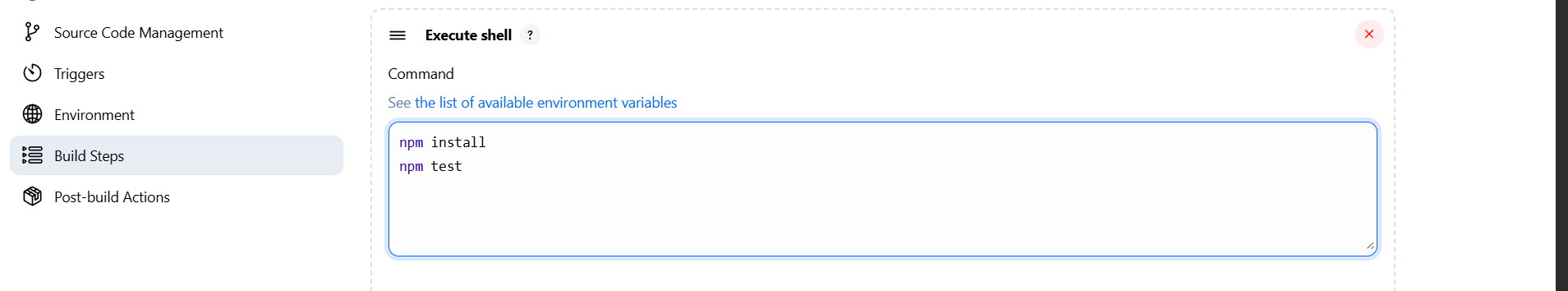




1. Under **Build Steps** → Execute shell:

npm install

npm test



1. Click Save → then Build Now
2. View Console Output like “freestyle-console-op.txt” in repo to verify the test ran successfully.
3. **Pipeline Job:**
4. Create a folder structure as below in server.

**Folder structure:**

sample-node-app/

├── app.js

├── package.json

└── test/

└──Jenkinsfile

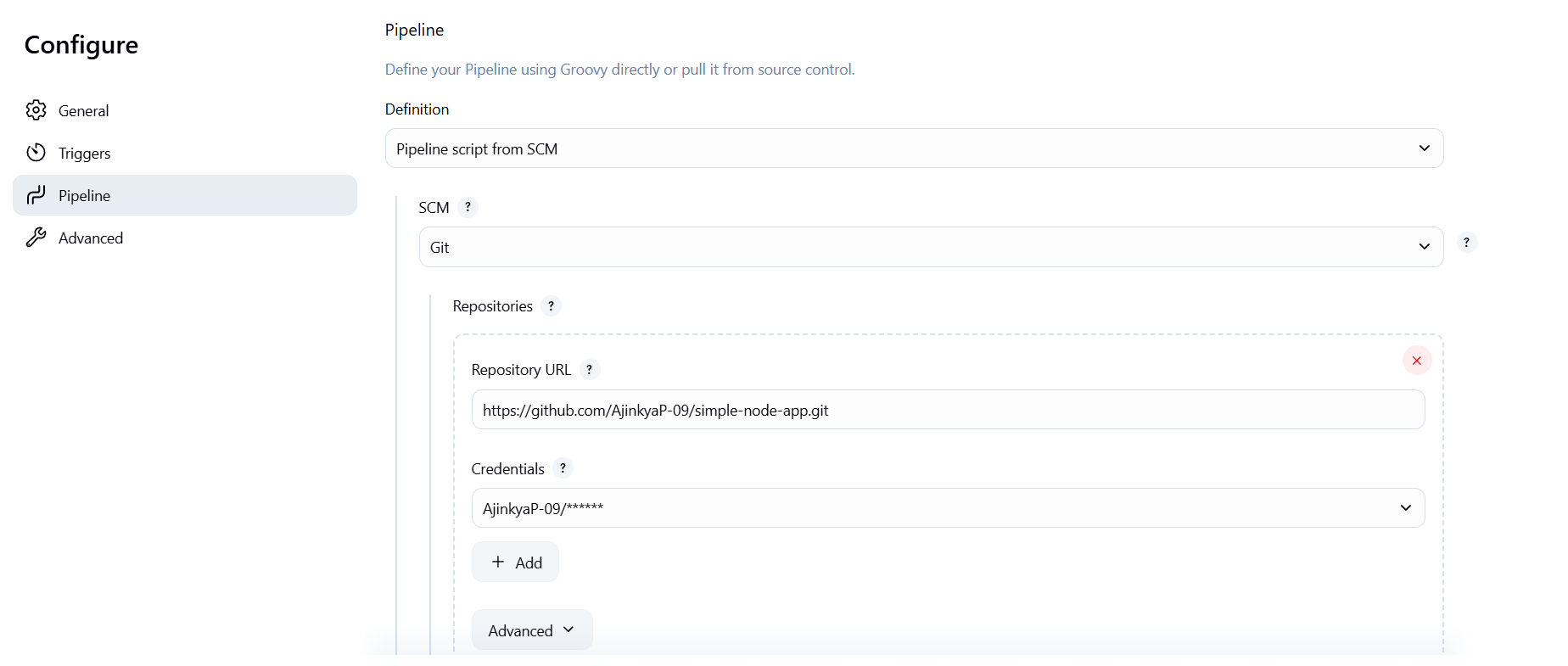
└── app.test.js

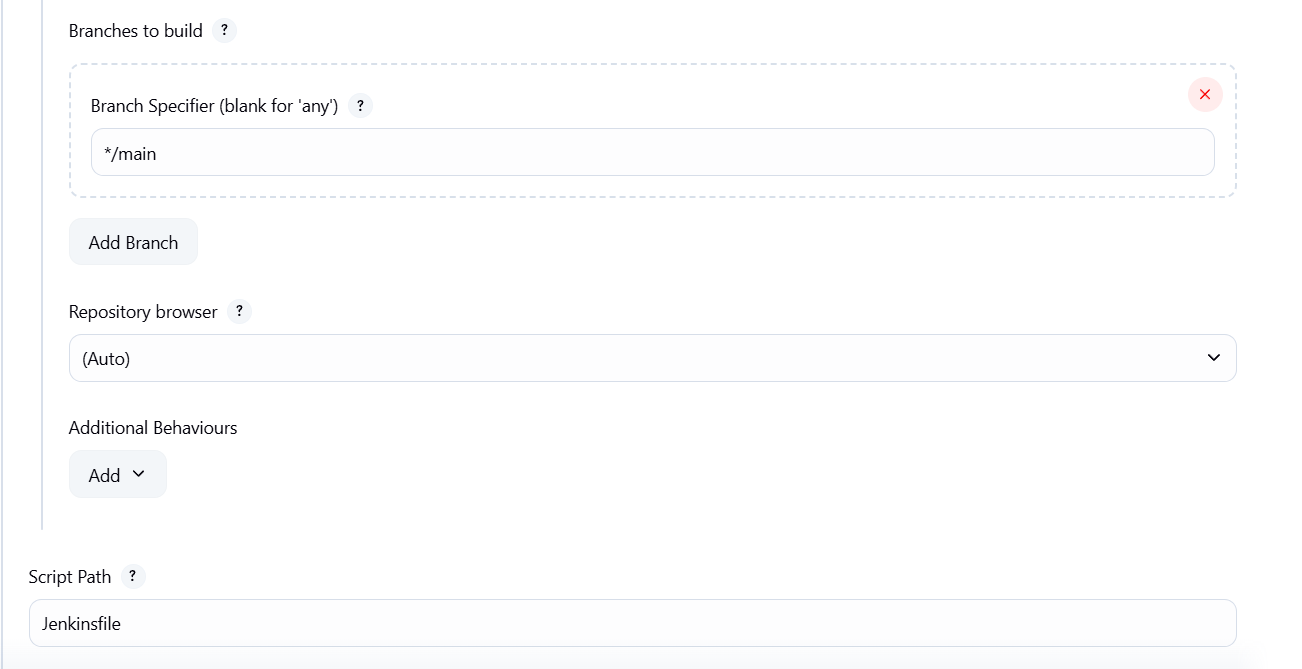
1. Add code of Jenkinsfile as in repository with your repository link.
2. Create new item from Jenkins Dashboard.
3. Enter description and select Github Project & enter repository link.
4. Choose Pipeline
5. Under Pipeline section:

Definition: Pipeline script from SCM

SCM: Git

URL: [https://github.com/<your-username>/sample-node-app.git](https://github.com/%3cyour-username%3e/sample-node-app.git), choose credentials and enter branch name.





1. Script Path: Jenkinsfile
2. Click Save → then Build Now
3. View Console Output like “pipeline-console-op.txt” in repo to verify the test ran successfully.

**What's Next:**

we will:

* Validate GitHub webhook automation by pushing code changes and watching Jenkins trigger builds
* Set up **Docker** on the Jenkins server
* Build a **Docker image** from our Node.js app using a Jenkins pipeline
* Optionally push the image to **Docker Hub**
* Learn how Jenkins manages **artifacts** and integrates build/test/deploy steps

This will take us closer to a real-world CI/CD setup and round off the remaining goals in Phase 2.

♣♣♣