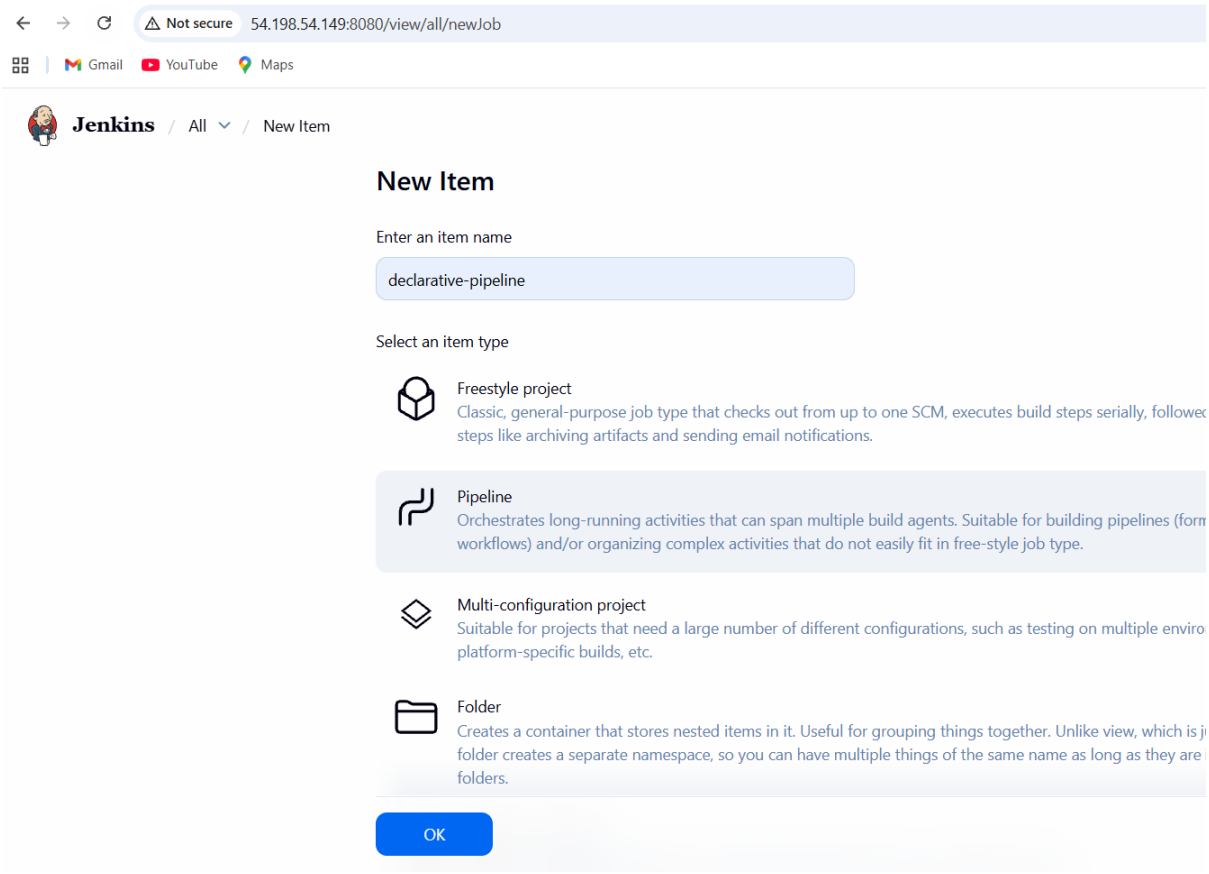
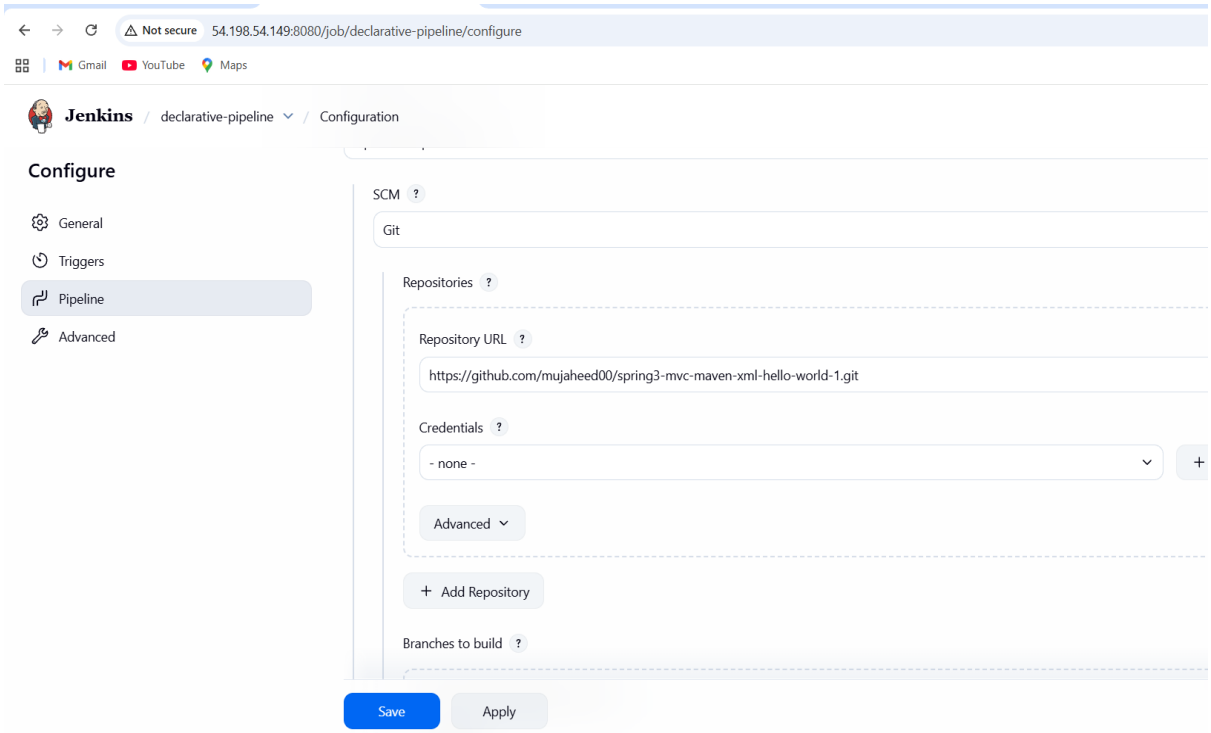
1. Create one Declarative pipeline job

Source Code: https://github.com/betawins/spring3-mvc-maven-xml-hello-world-1.git

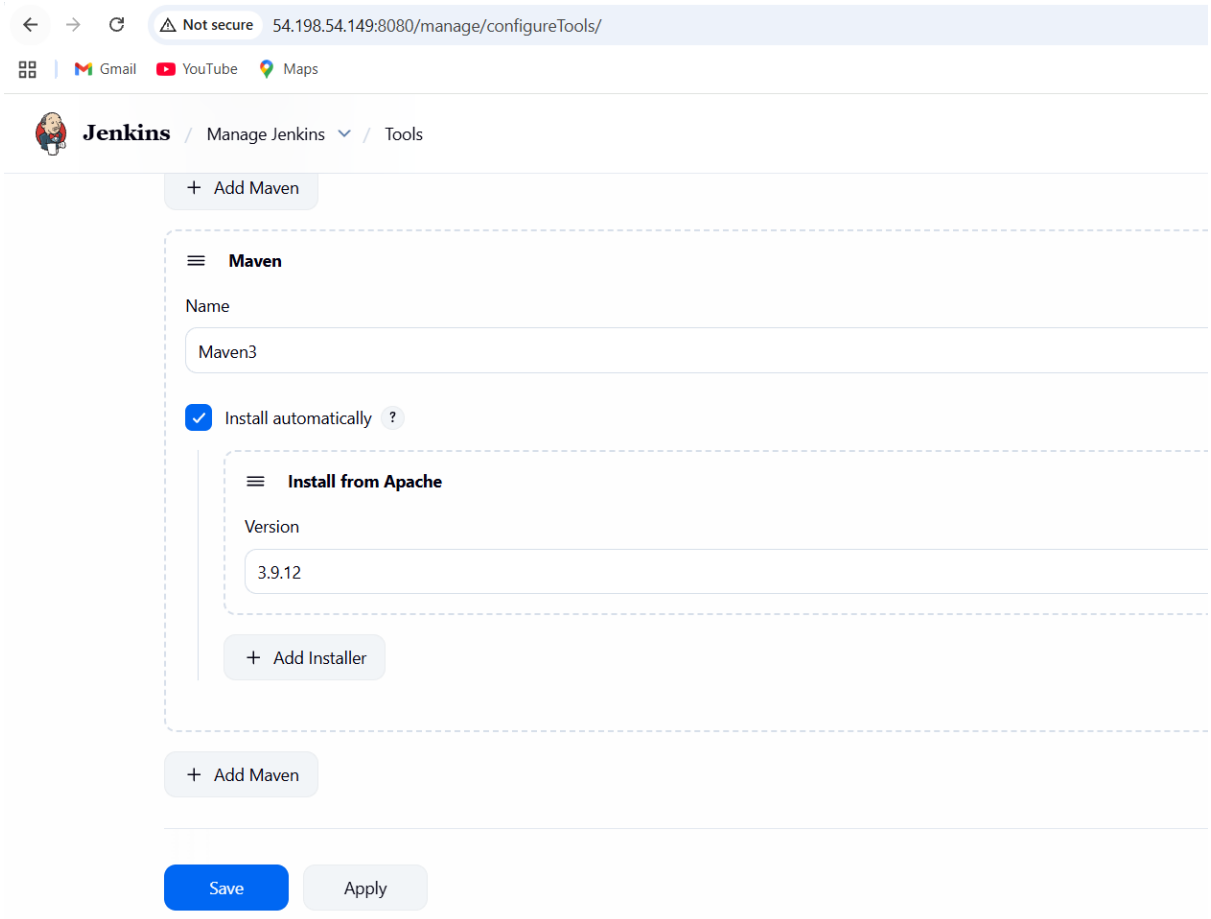
Click on create new job name as declarative-pipeline and type as pipeline.



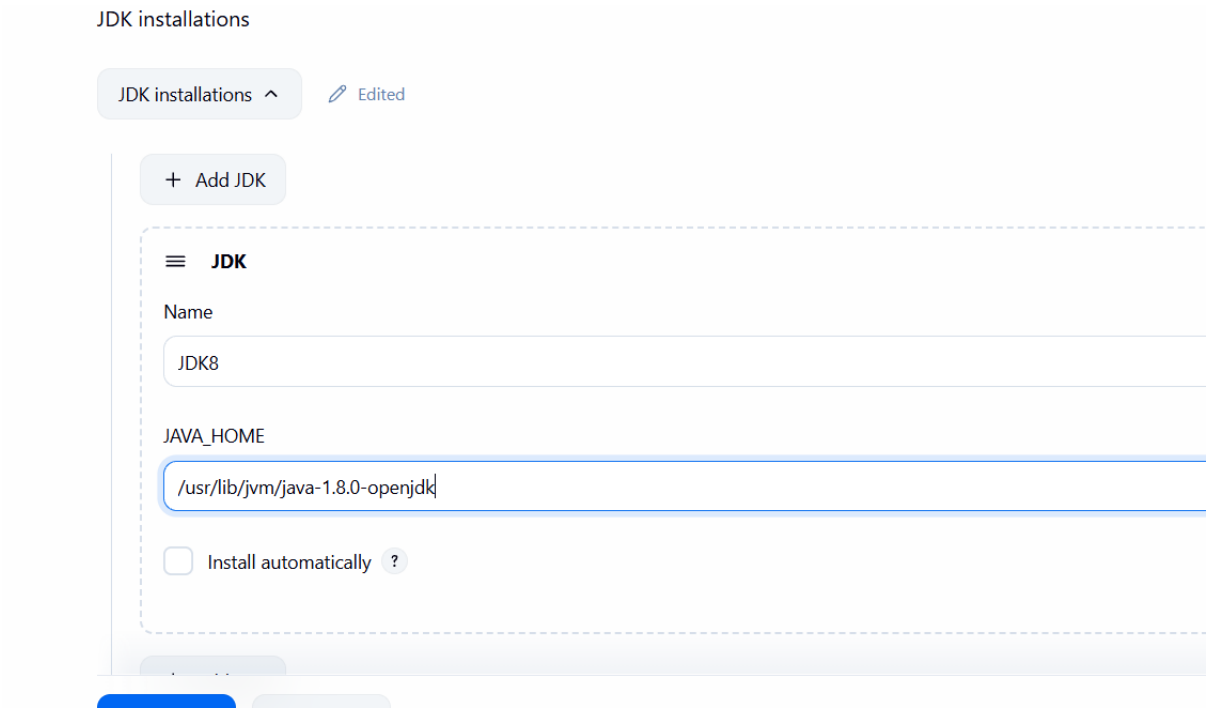
In that job click on gitscm and provide repo url and save it.



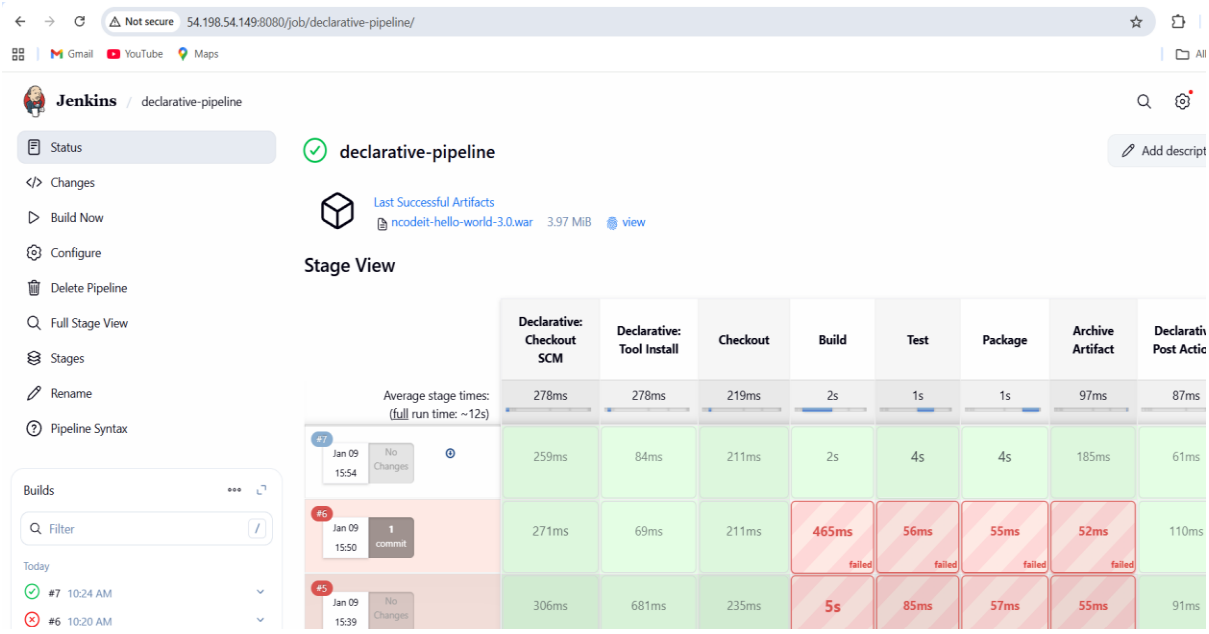
Go to manage Jenkins scroll down click on maven



Add jdk in manage Jenkins,tools



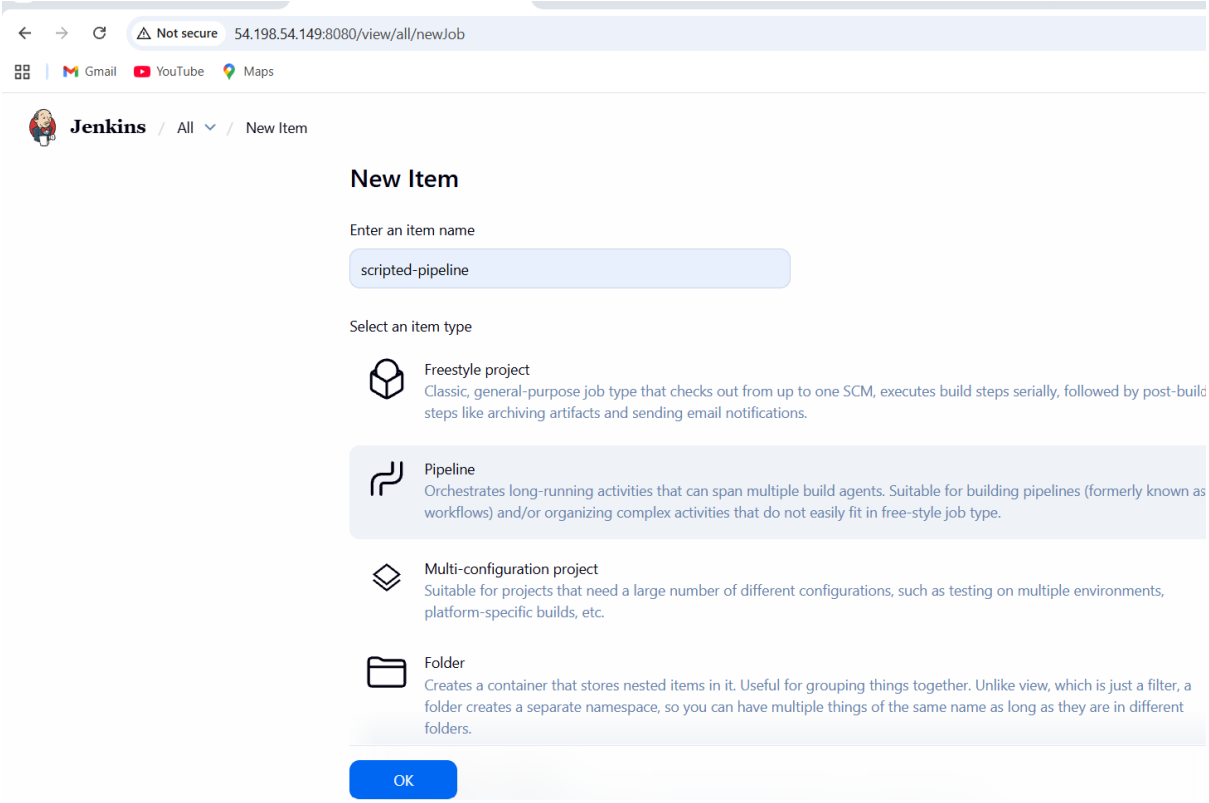
Click on build now



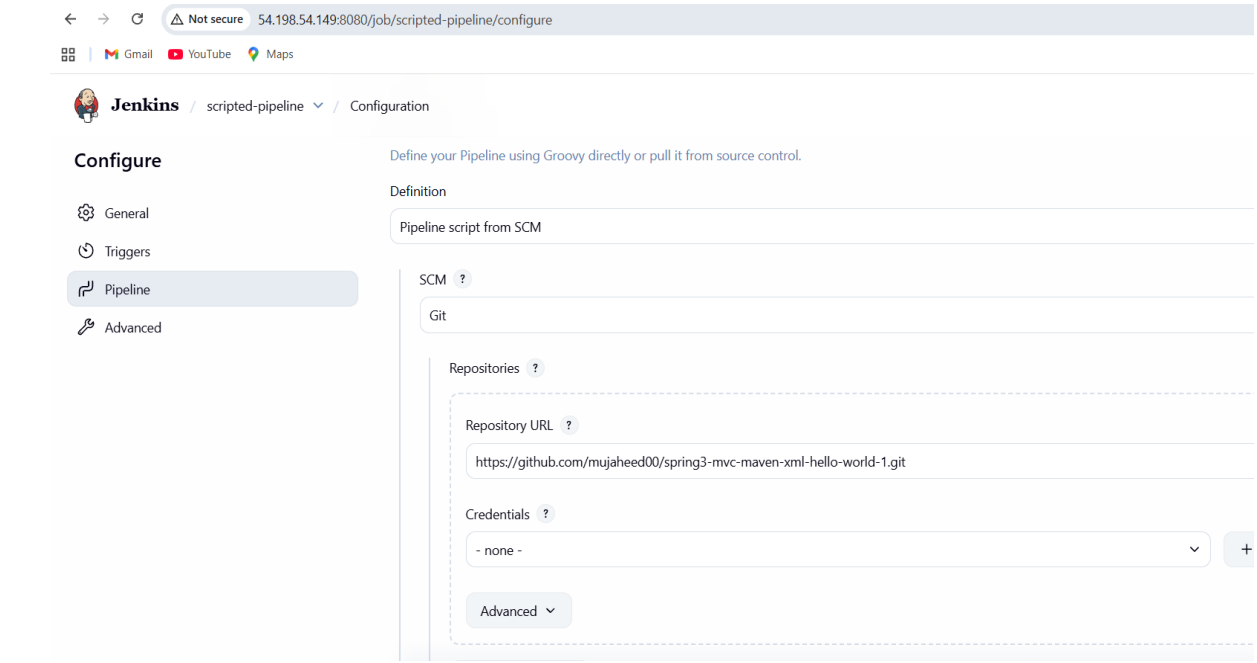
3. Create one Scripted pipeline job

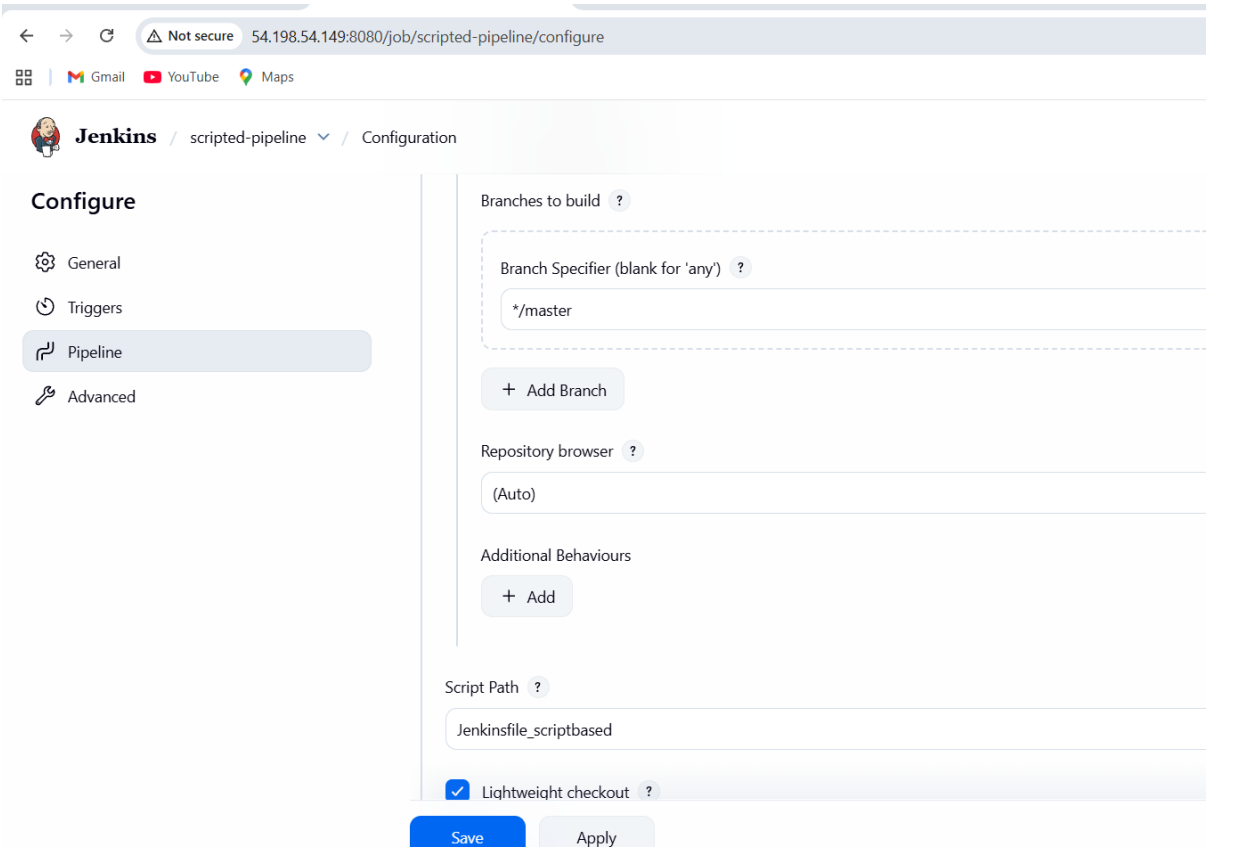
Source Code: https://github.com/betawins/spring3-mvc-maven-xml-hello-world-1.git

Click on build new job and name it as scripted-pipeline and select type as pipeline.

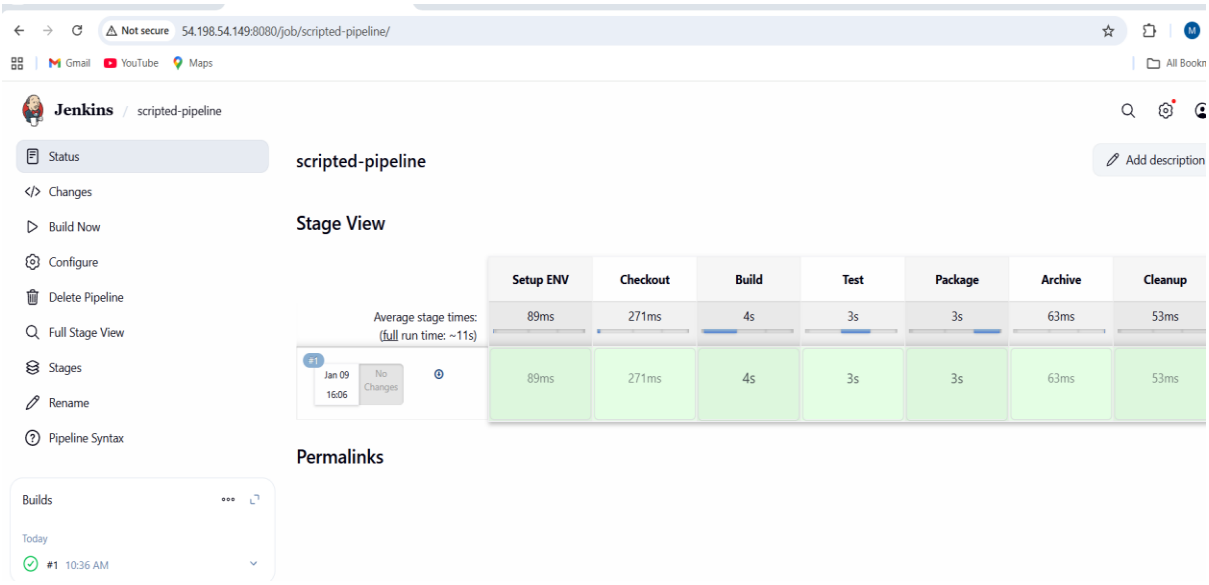


Go to gitscm and give your repository and Jenkins file path.





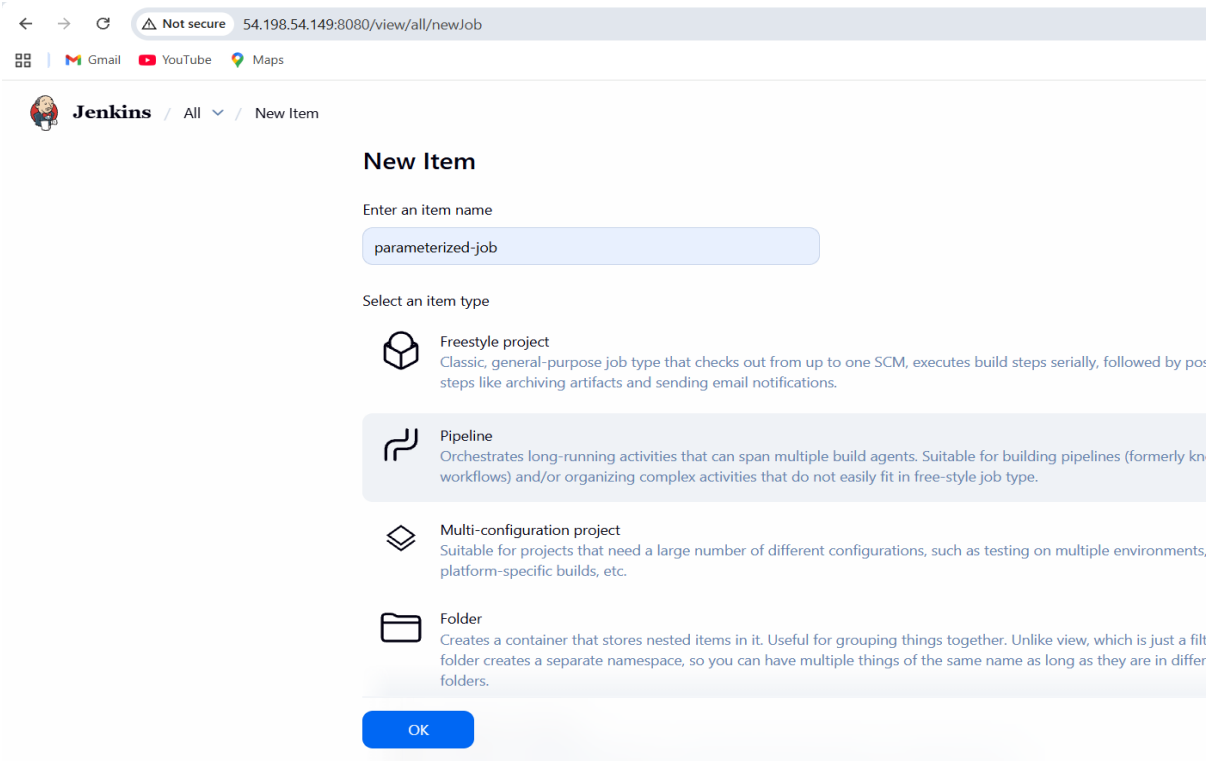
Click on build now.



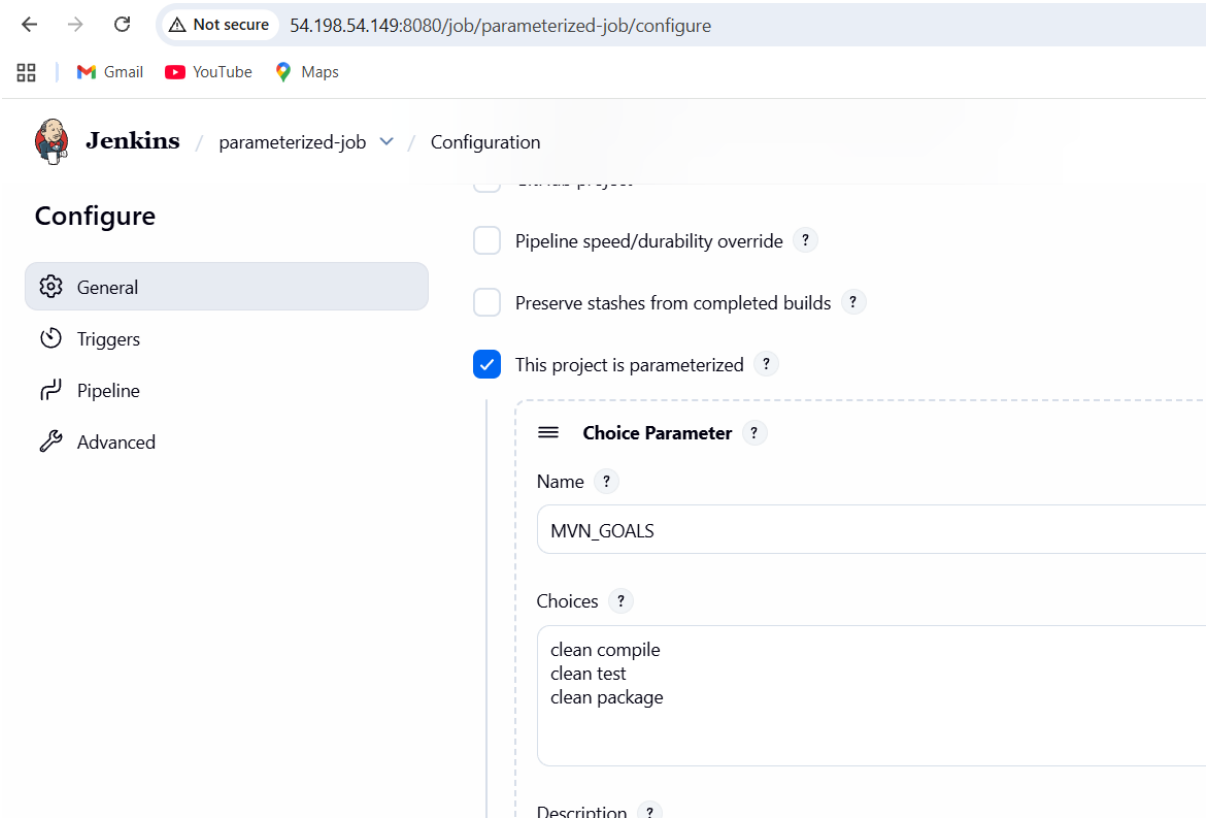
3. Create one Parameterized job

Source Code: https://github.com/betawins/spring3-mvc-maven-xml-hello-world-1.git

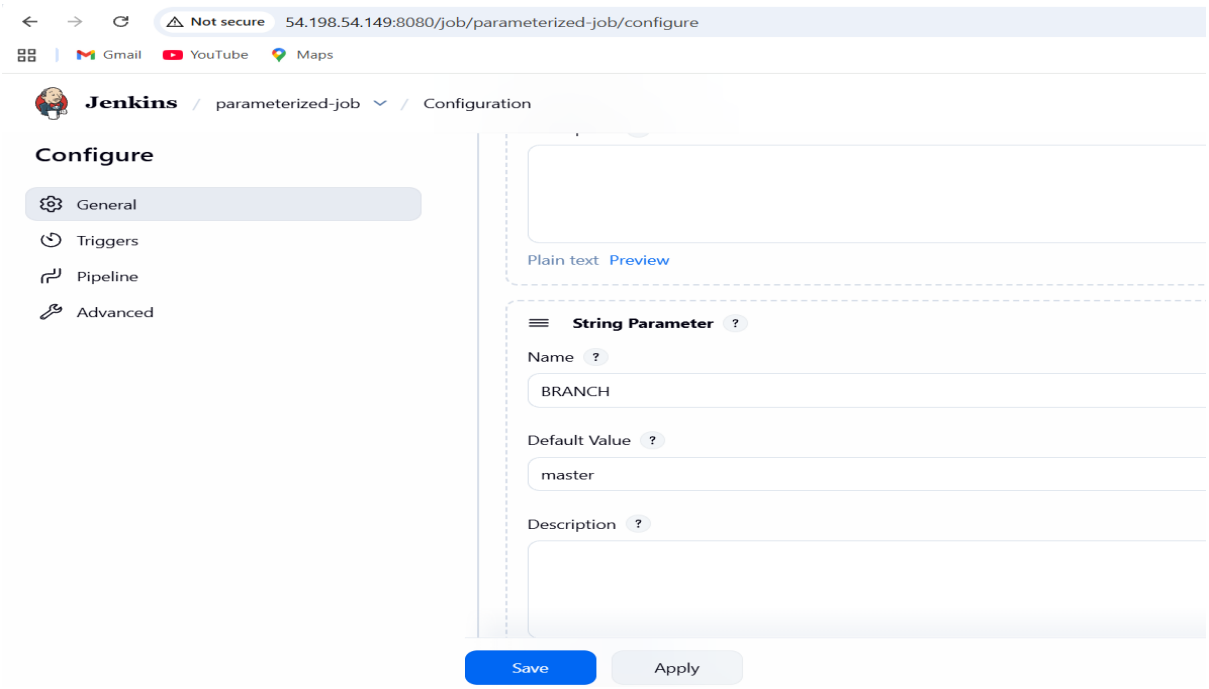
Create a new job for parameterized pipeline named as parameterized-job



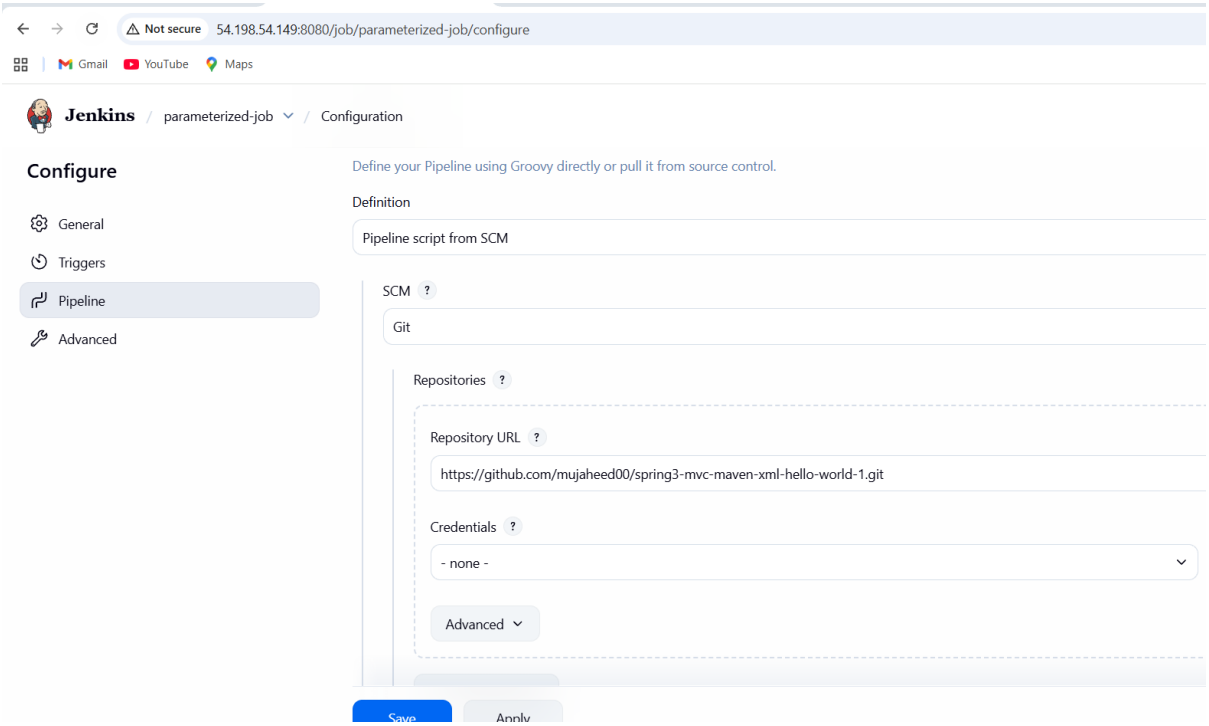
Select this project is parameterized and select choices and give values.

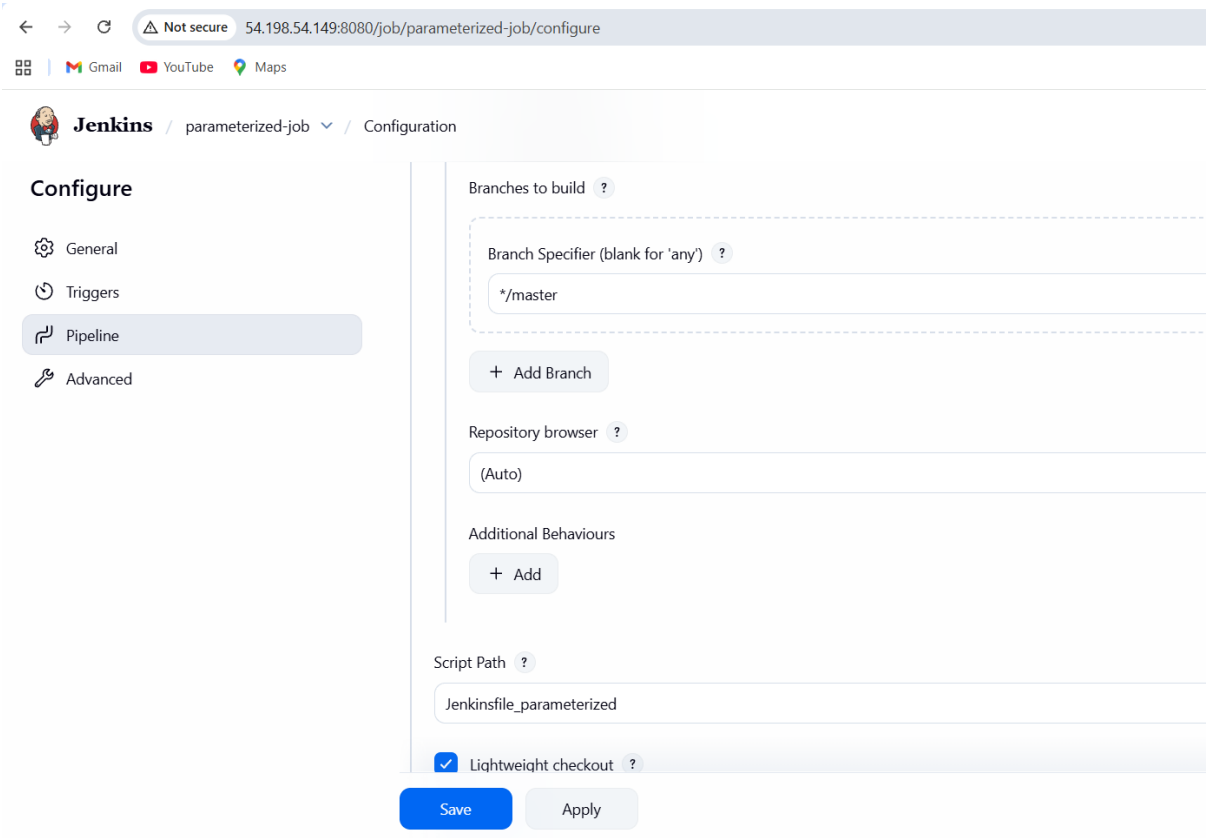


Add another parameter as string give name as branch , value as master.

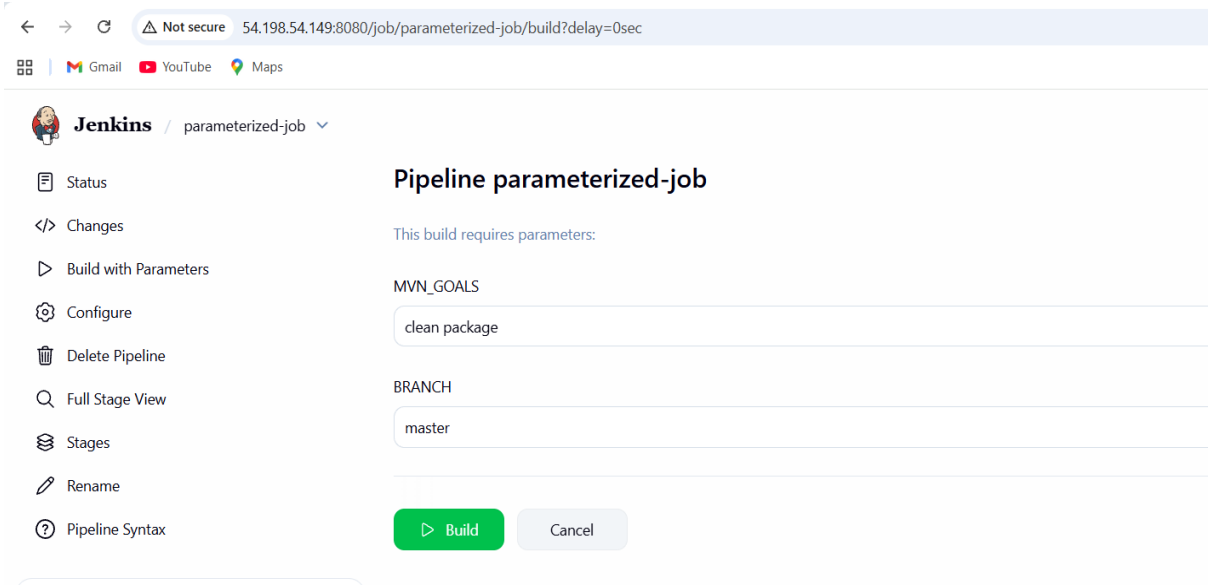


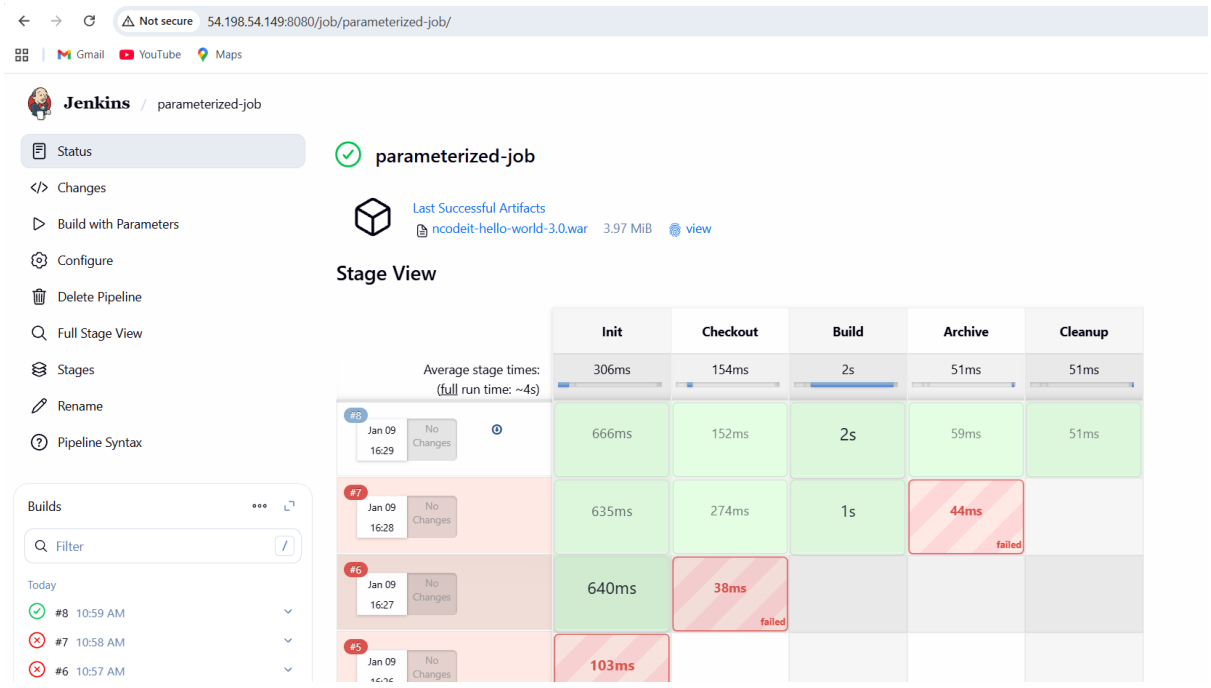
Click on gitscm and give your repo name





Click on build with parameters you can build which environment you need with the same script.





4. Write sample skeleton for Declarative and Scripted pipelines

<u>Declarative pipeline:</u>

```
pipeline {
    agent any
```

```
    tools {
        jdk 'JDK8'          // Optional
        maven 'MAVEN3'      // Optional
    }

    environment {
        APP_NAME = "sample-app"
        BUILD_ENV = "dev"
    }

    parameters {
        string(name: 'BRANCH', defaultValue: 'main',
description: 'Git branch')
    }

    stages {

        stage('Checkout') {
            steps {
                git branch: "${params.BRANCH}",
```

```
            url: 'https://github.com/org/repo.git'
        }
    }

    stage('Build') {
        steps {
            sh 'mvn clean compile'
        }
    }

    stage('Test') {
        steps {
            sh 'mvn test'
        }
    }

    stage('Package') {
        steps {
            sh 'mvn package'
        }
    }
```

```
        stage('Archive Artifacts') {
            steps {
                archiveArtifacts artifacts: 'target/*.jar', fingerprint:
true
            }
        }
    }


    post {
        success {
            echo 'Pipeline executed successfully'
        }
        failure {
            echo 'Pipeline failed'
        }
        always {
            cleanWs()
        }
    }
}
```

Scripted pipeline:

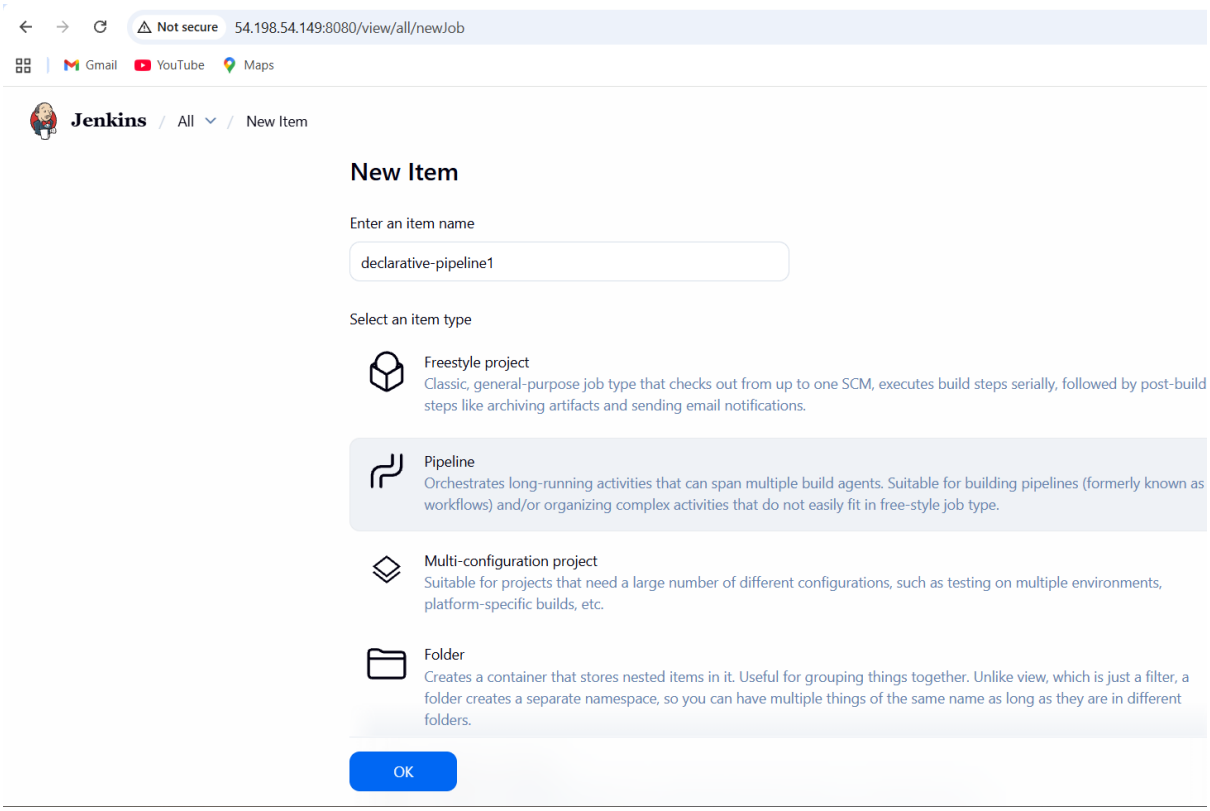
```
node {

    def appName = "sample-app"
    def branchName = "main"

    try {

        stage('Checkout') {
            git branch: branchName,
                url: 'https://github.com/org/repo.git'
        }

        stage('Build') {
            sh 'mvn clean compile'
        }

        stage('Test') {
            sh 'mvn test'
        }
```

```
        stage('Package') {
            sh 'mvn package'
        }

        stage('Archive Artifacts') {
            archiveArtifacts artifacts: 'target/*.jar', fingerprint:
true
        }

        echo "Pipeline completed successfully"

    } catch (err) {
        echo "Pipeline failed"
        throw err
    } finally {
        cleanWs()
    }
}
```

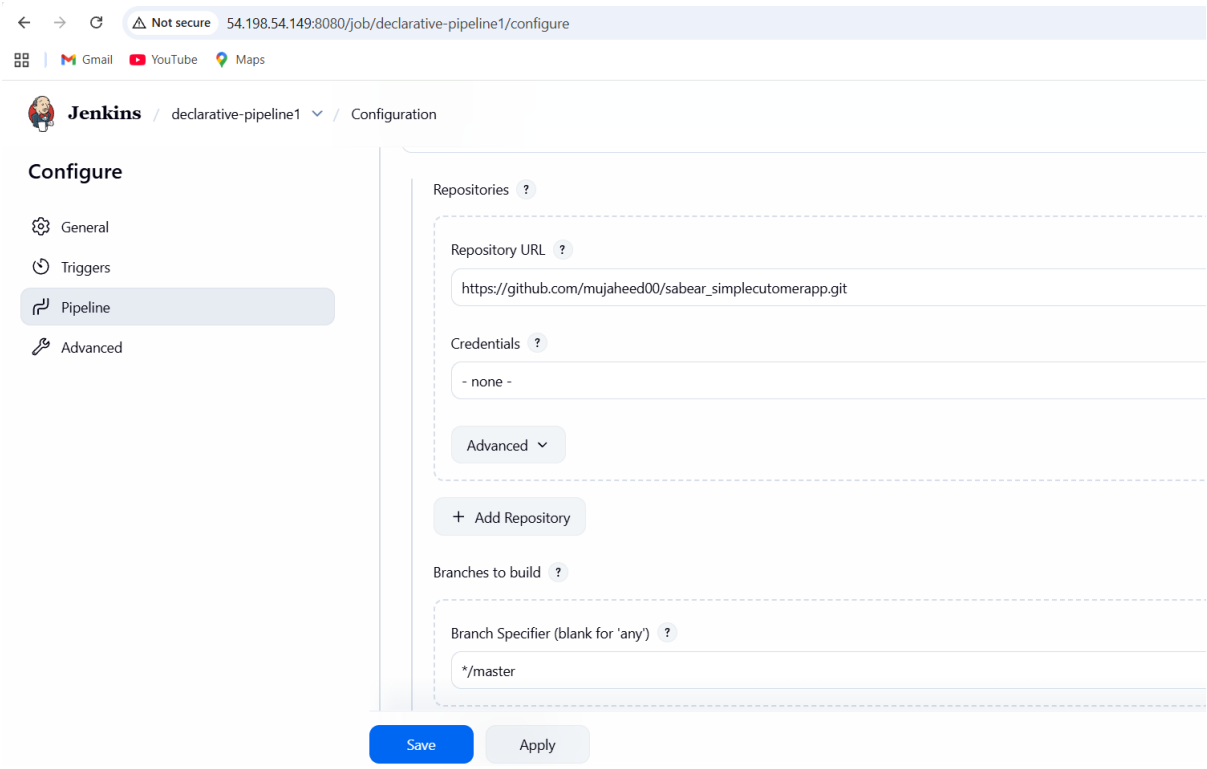
5 .Create one Declarative job

SourceCode: https://github.com/betawins/sabear_simplecustomerapp/tree/feature-1.1

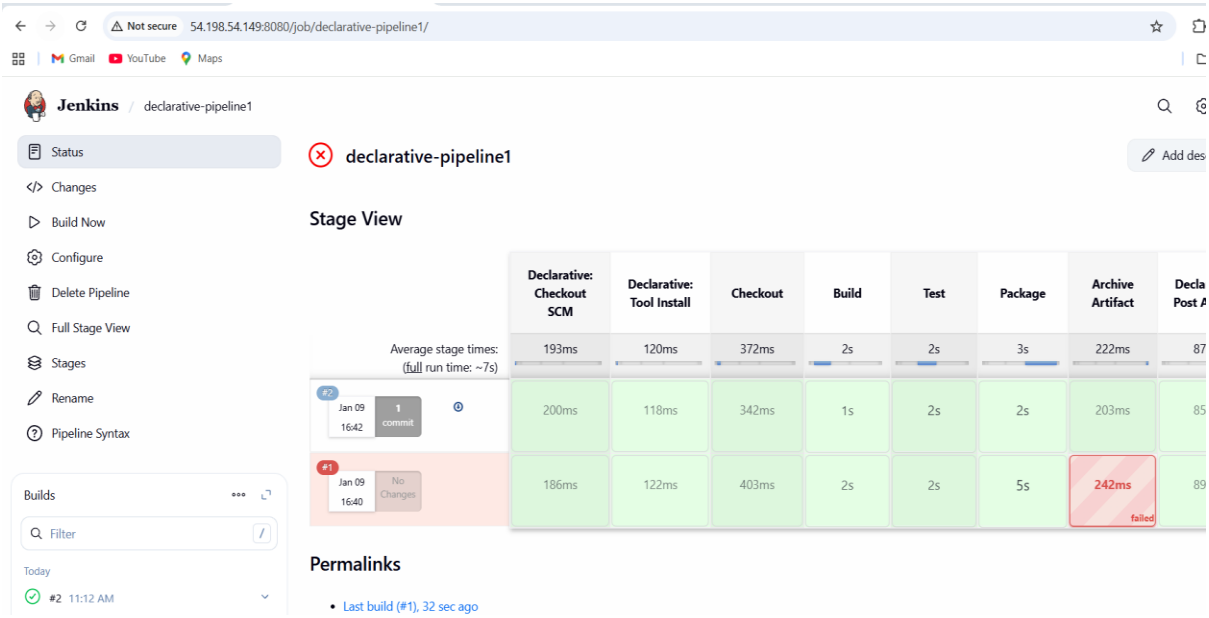
Build a new job.



Select gitscm and give your repo url



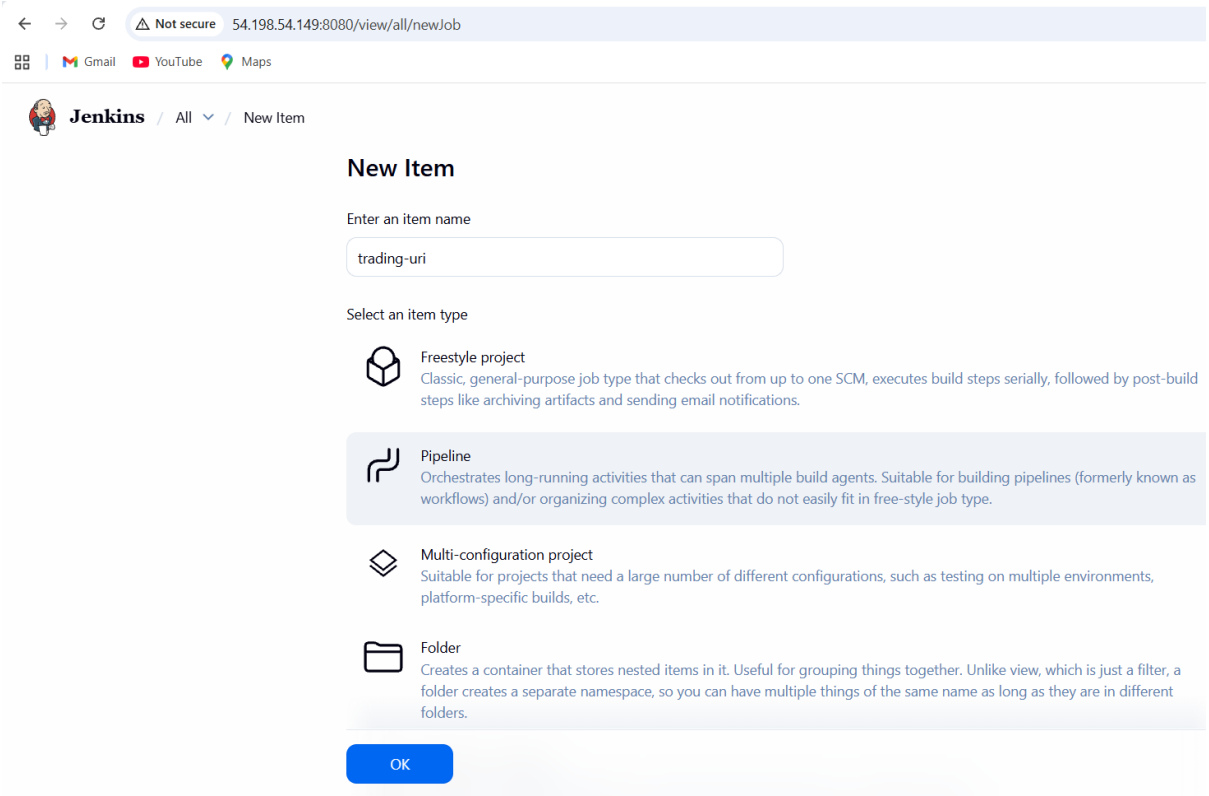
Click on build now.



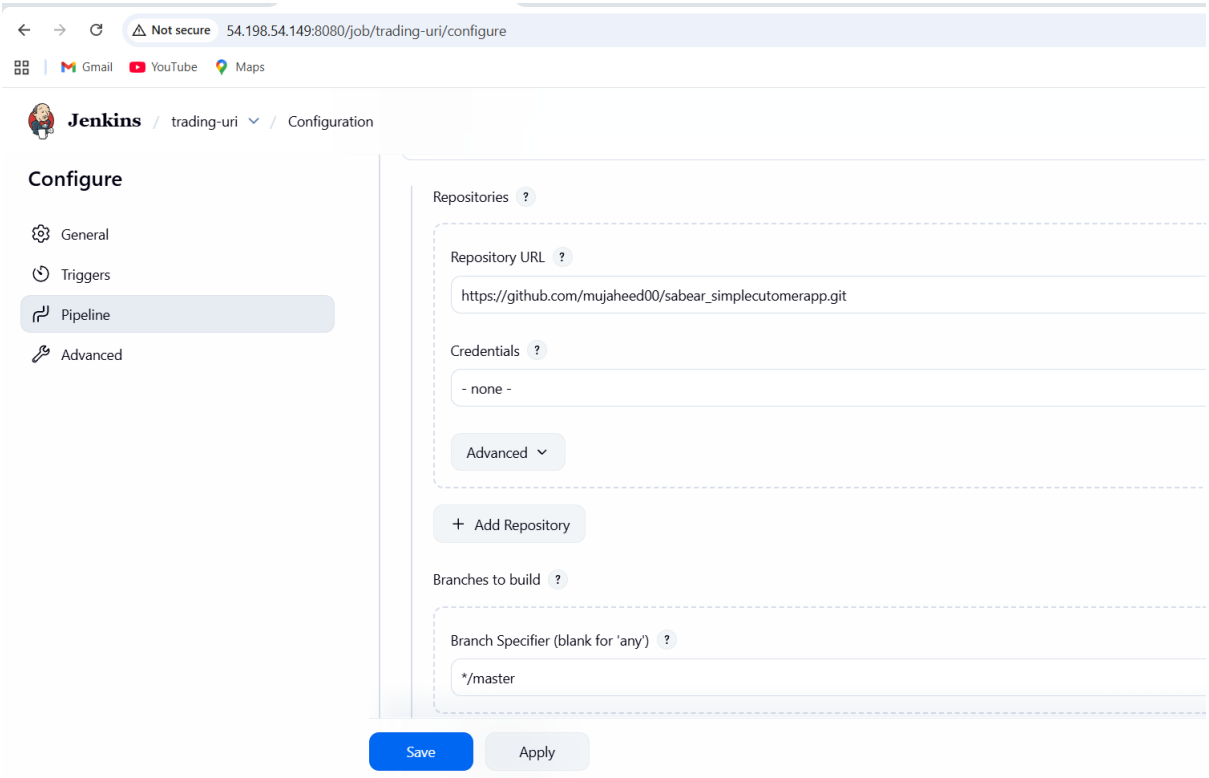
6. Create one Declarative job

Source Code: https://github.com/betawins/Trading-UI.git

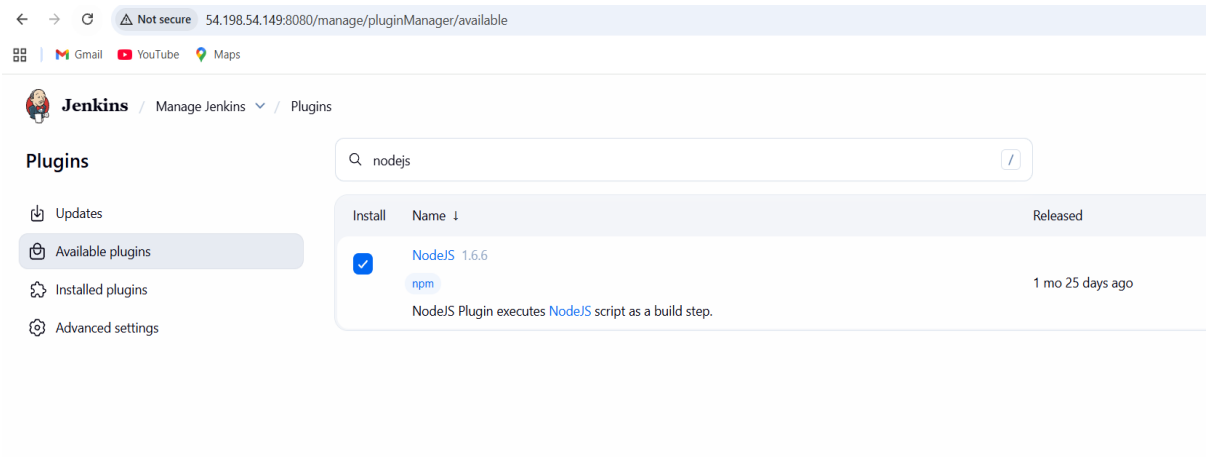
Create new job



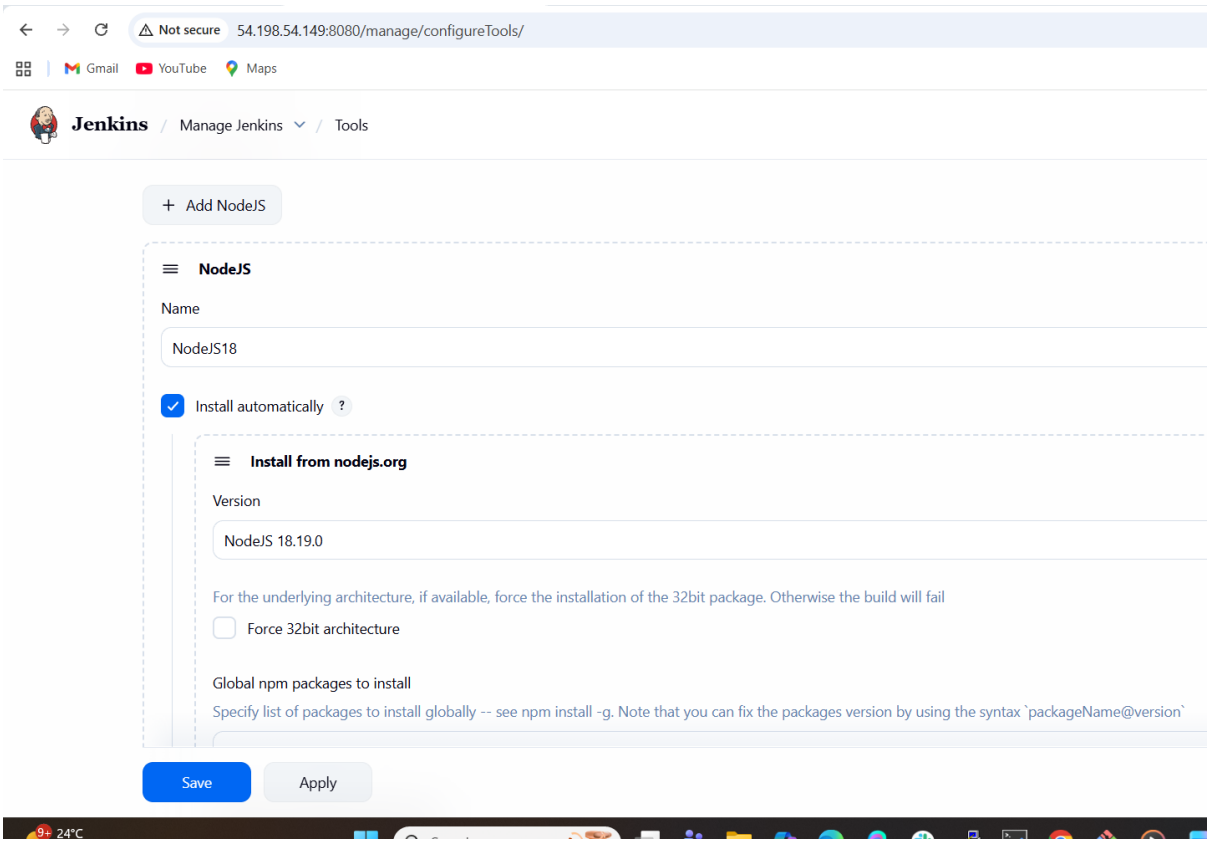
Click on gitscm and give your repository url



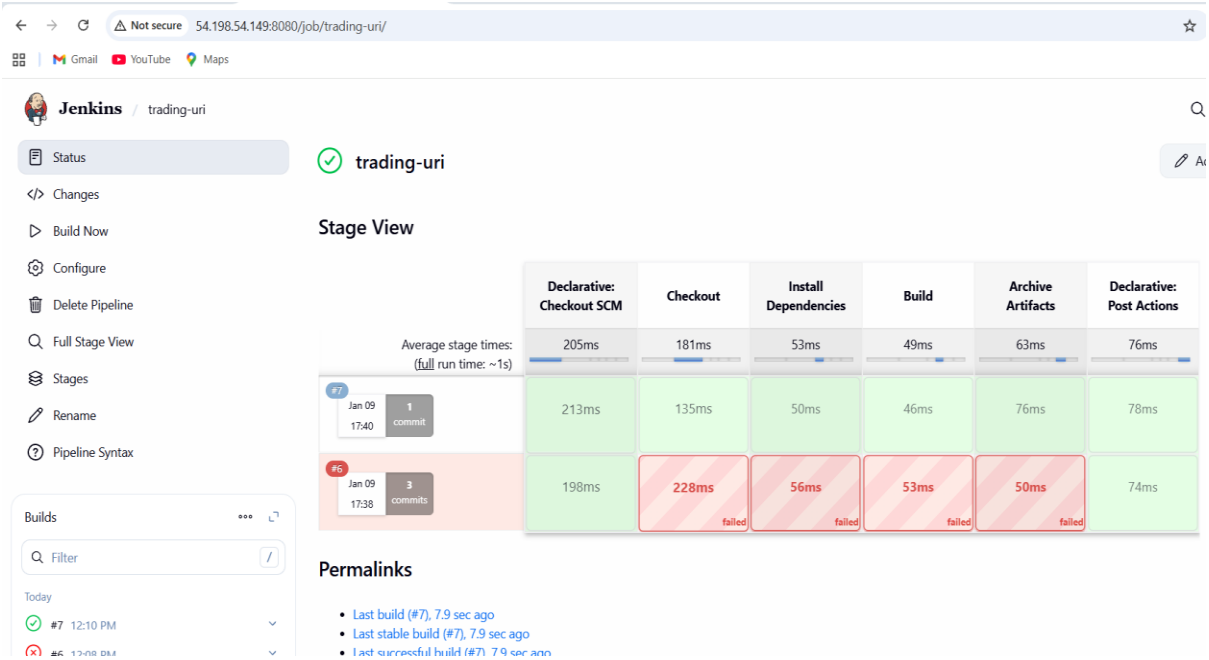
Install plugin nodejs



Go to manage Jenkins, tools and give nodejs version

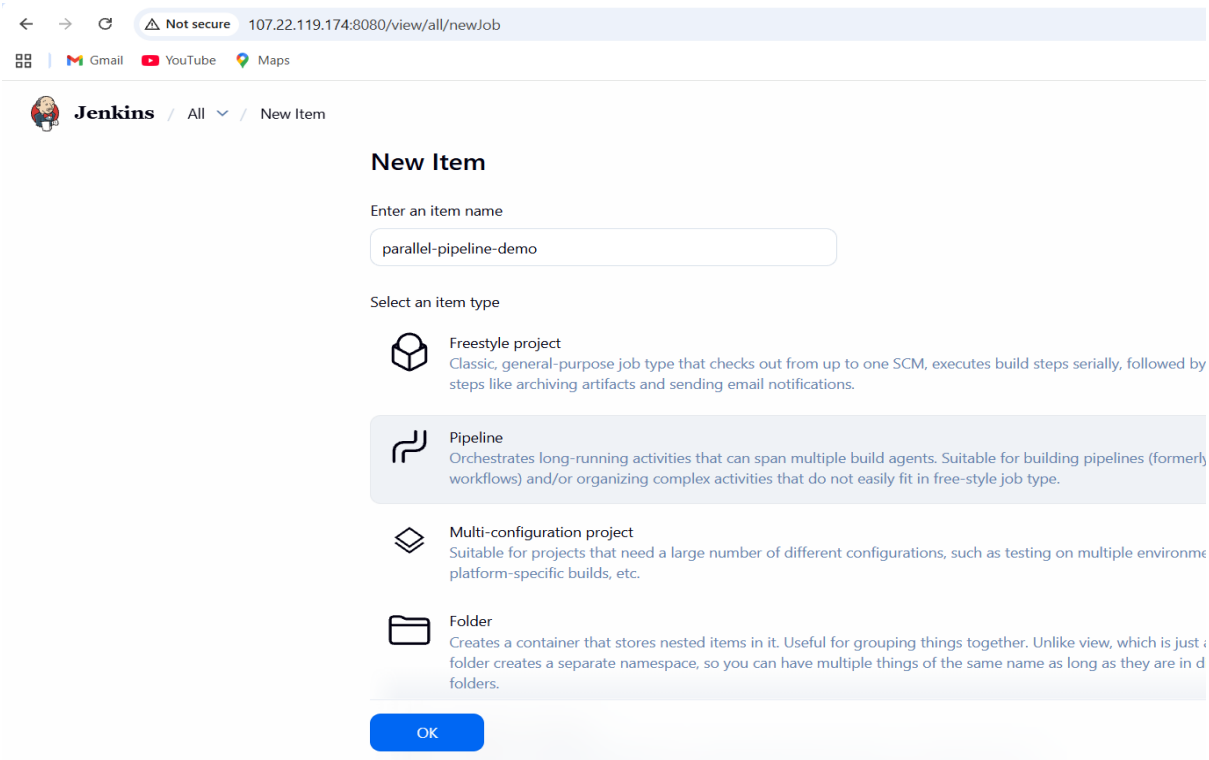


Click on build now.



7. Execute parallel stages using Jenkins pipeline for any sample job

Click on create new job with the name parallel-pipeline-demo and select pipeline as type.

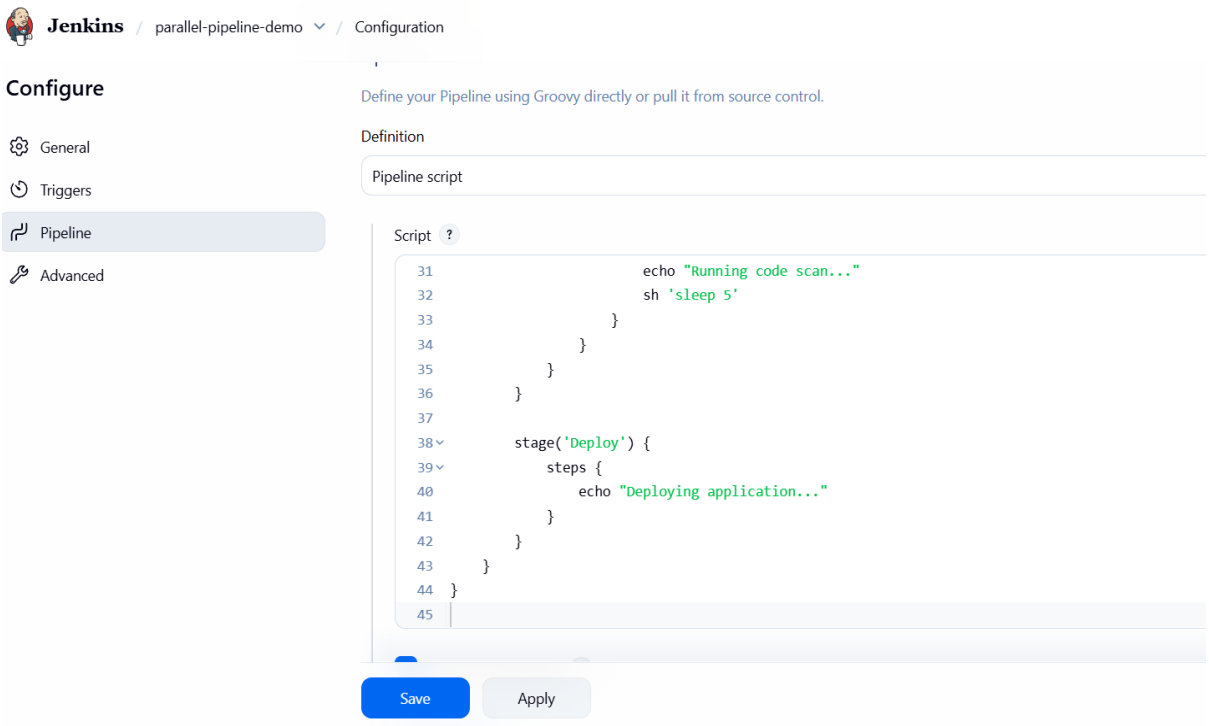


In that job go to pipeline and select pipeline script and give this script.

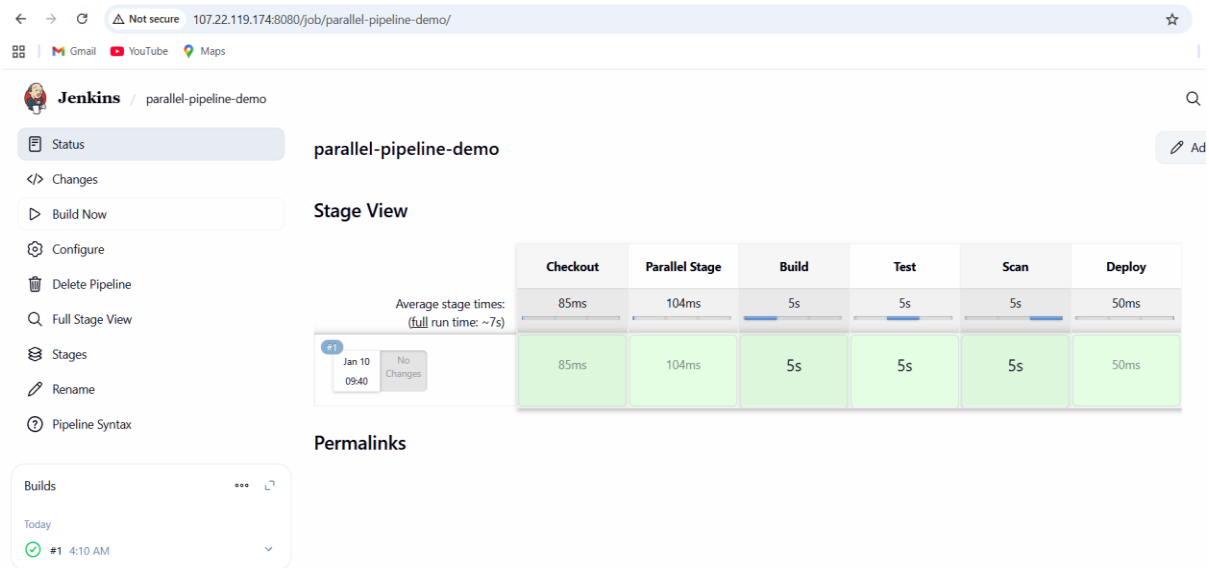
```
pipeline {
    agent any

    stages {

        stage('Checkout') {
            steps {
                echo "Checking out source code..."
            }
        }

        stage('Parallel Stage') {
            parallel {

                stage('Build') {
                    steps {
                        echo "Building application..."
                        sh 'sleep 5'
                    }
```

```
        }

        stage('Test') {
            steps {
                echo "Running tests..."
                sh 'sleep 5'
            }
        }

        stage('Scan') {
            steps {
                echo "Running code scan..."
                sh 'sleep 5'
            }
        }
    }
}

stage('Deploy') {
    steps {
        echo "Deploying application..."
```

```
        }
    }
  }
}
```



Save and click on build now.

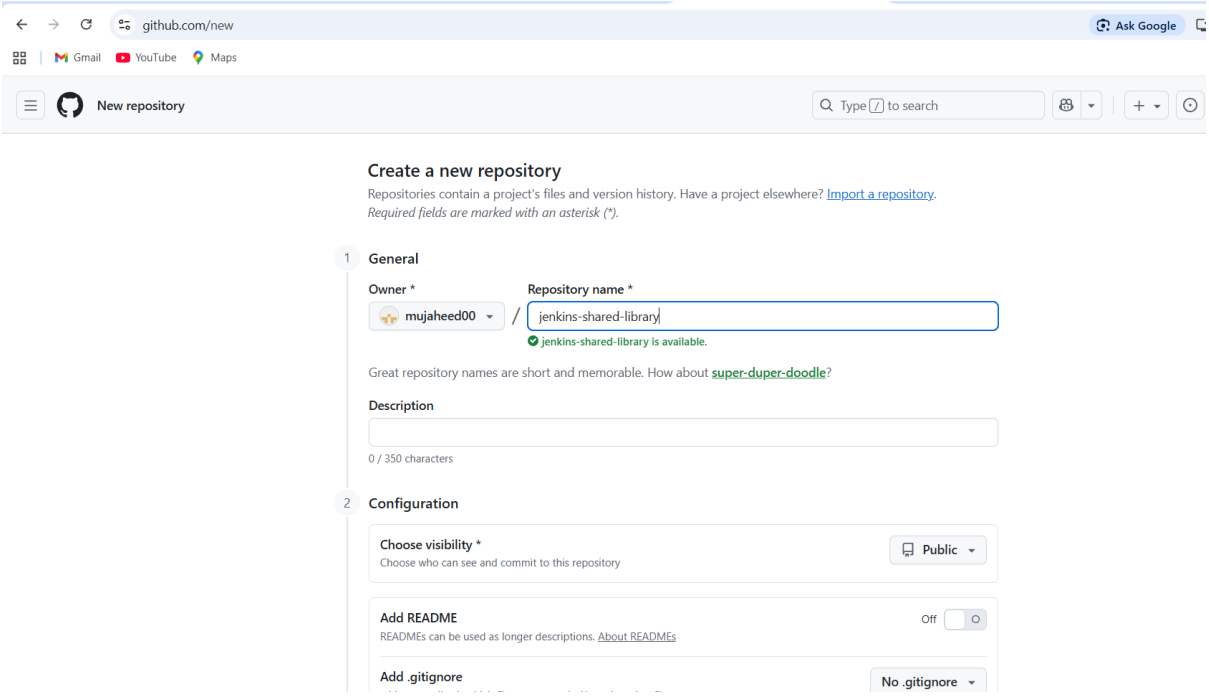


Build,test,scan these 3 steps will executed parallelly with this we can optimize time.

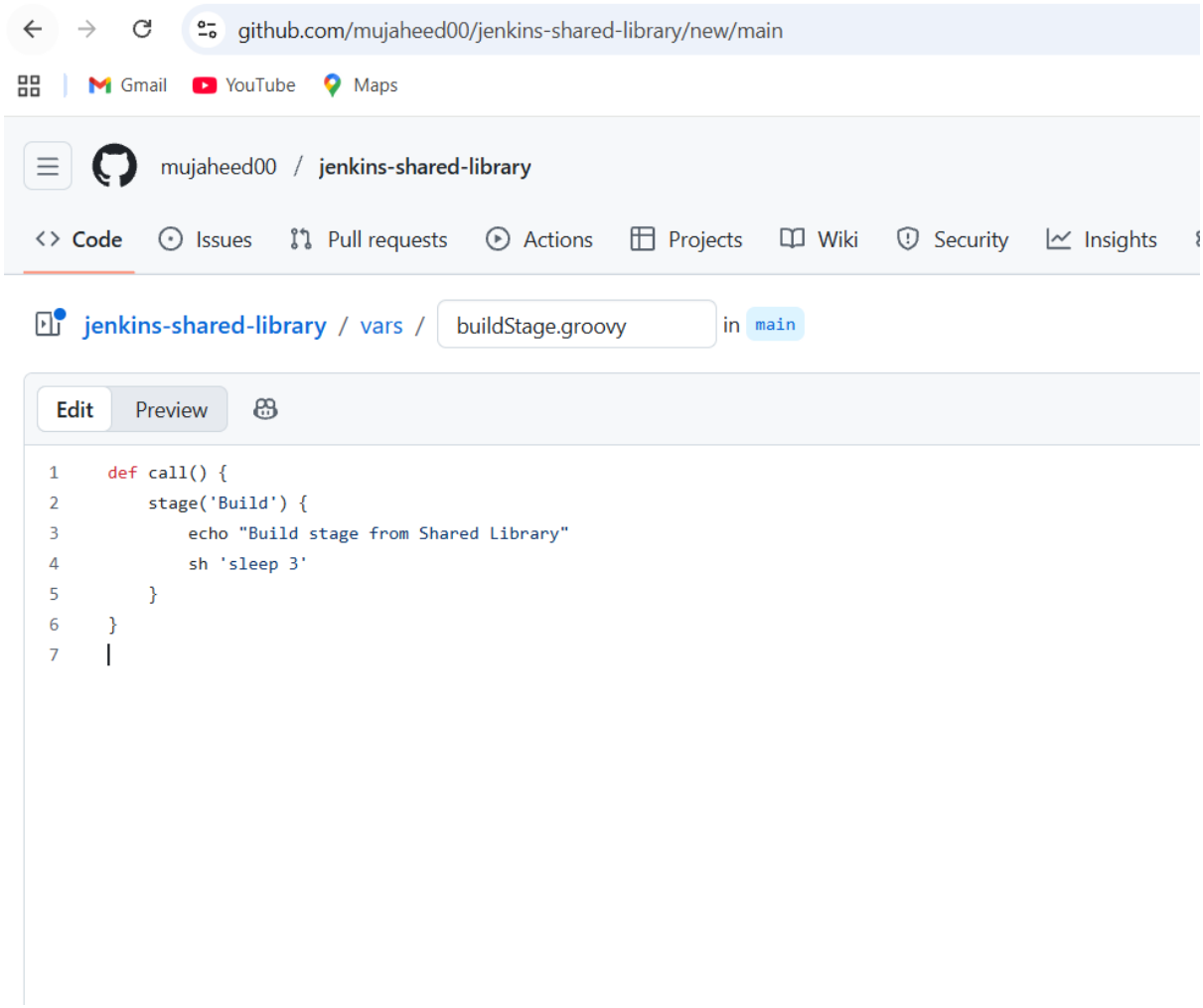
8. Execute Jenkins pipeline stages using shared libraries

Reference: https://phoenixnap.com/kb/jenkins-shared-library

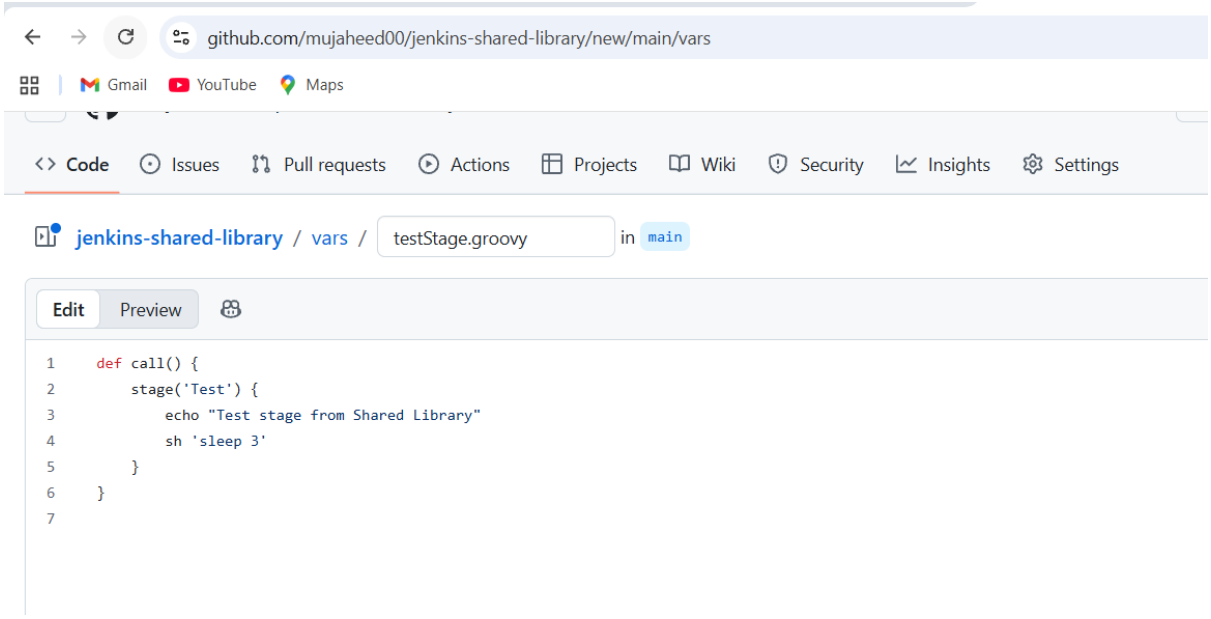
Create a new repository in github with the name Jenkins-shared-library



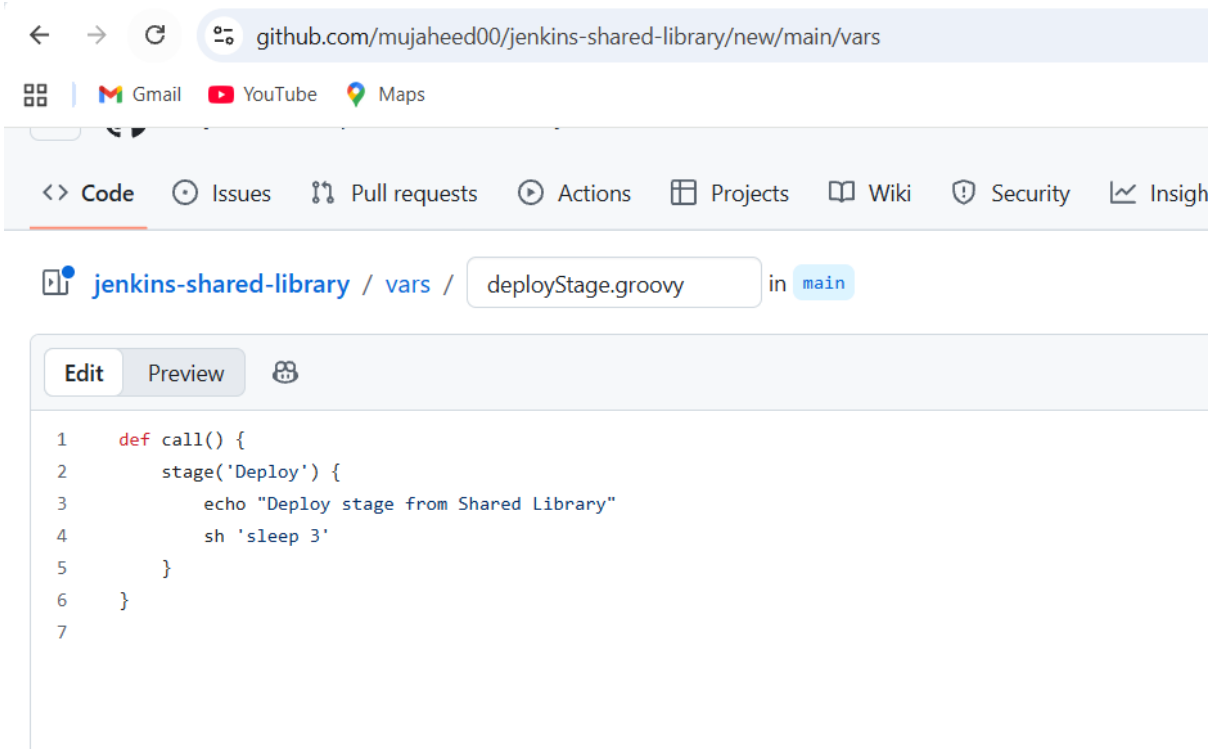
Create a directory in that git hub named as vars so there is now specific type to create directory so, type vars/filename it will create a file inside that directory.



Create an another file with name var/testStage.groovy

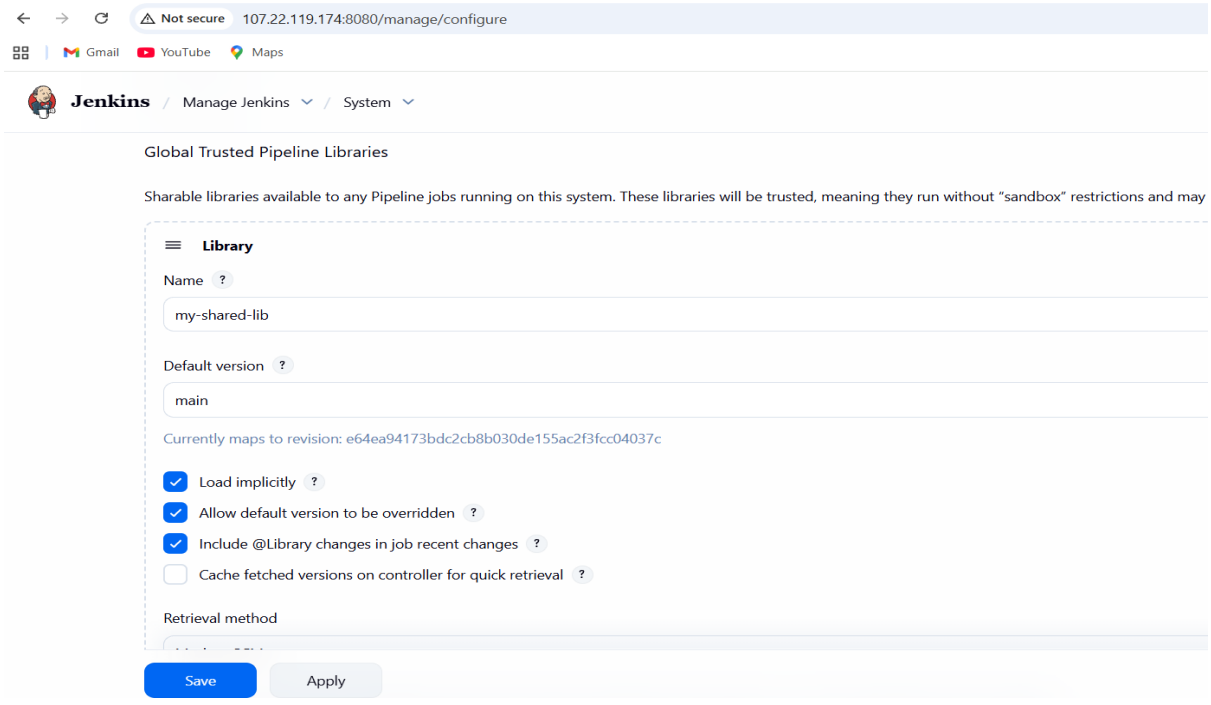


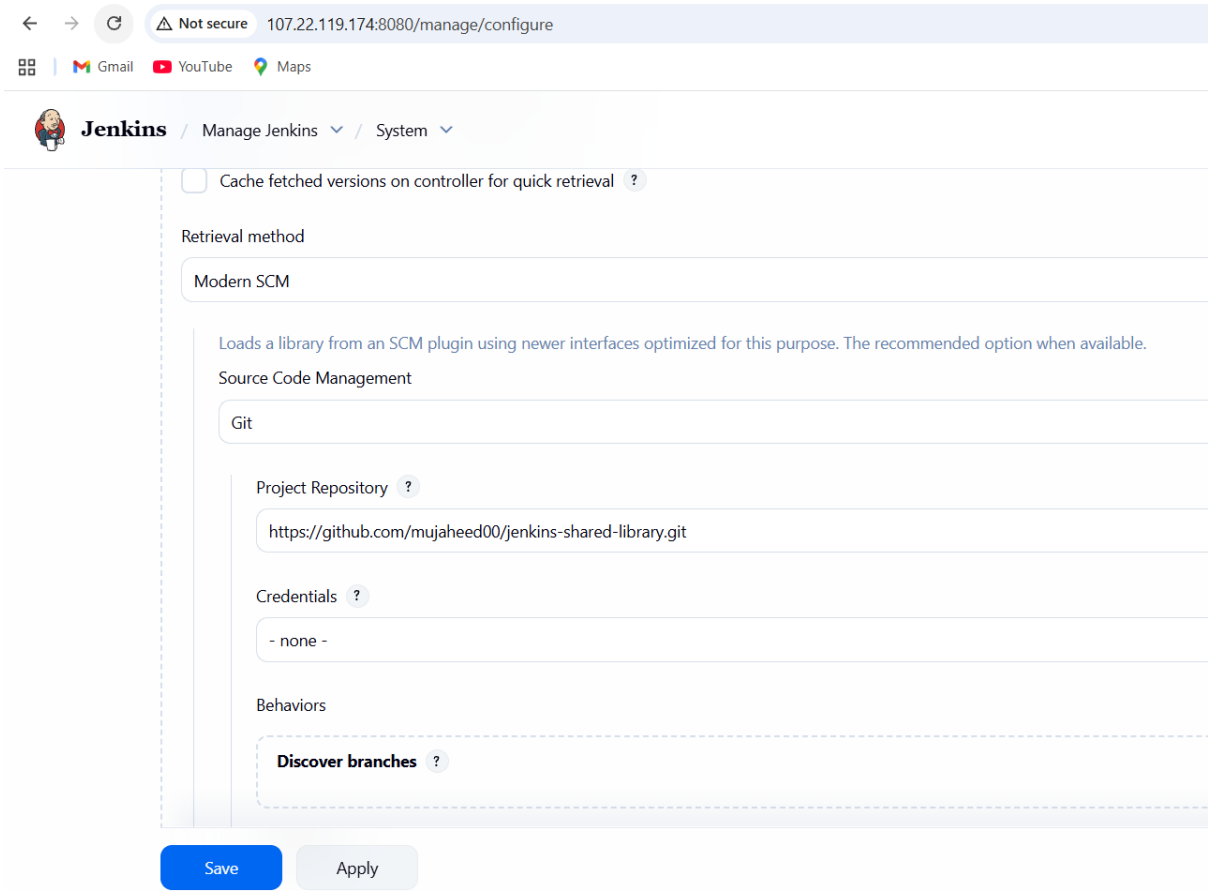
Create another file with the name var/deployStage.groovy



Go to Jenkins dashboard and go to manage Jenkins, and go to system and click on global trusted pipeline libraries click on add and give details

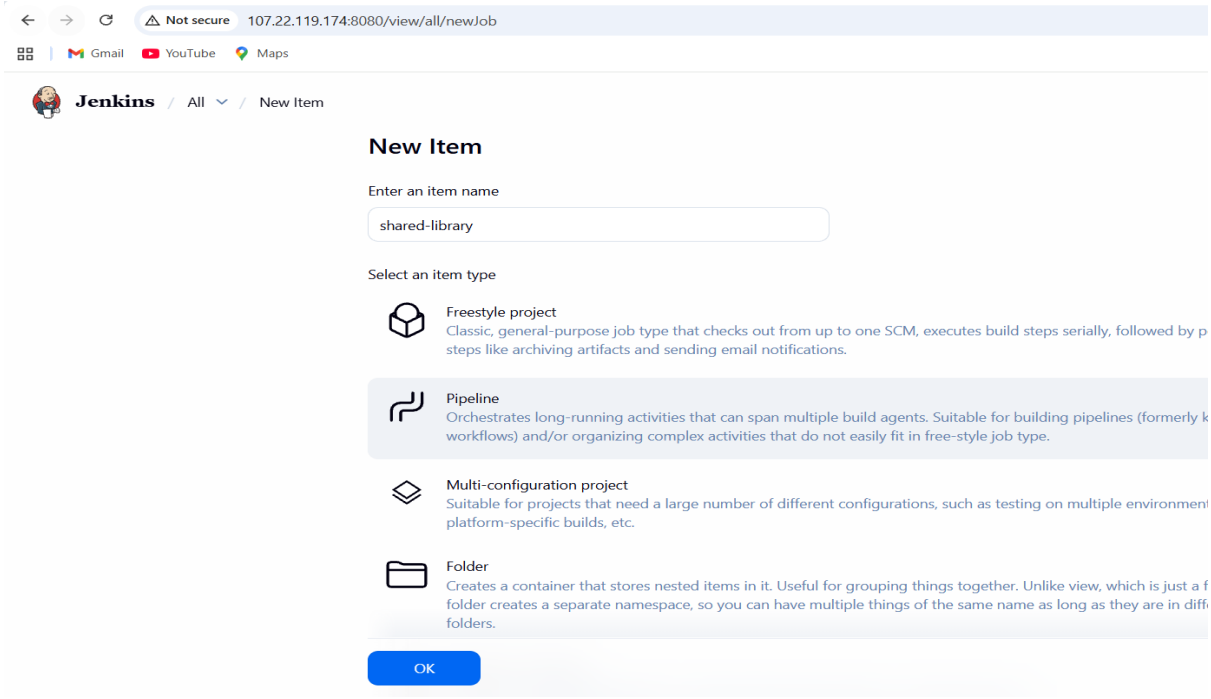
Name, version as branch name, select modern scm and give repo url.





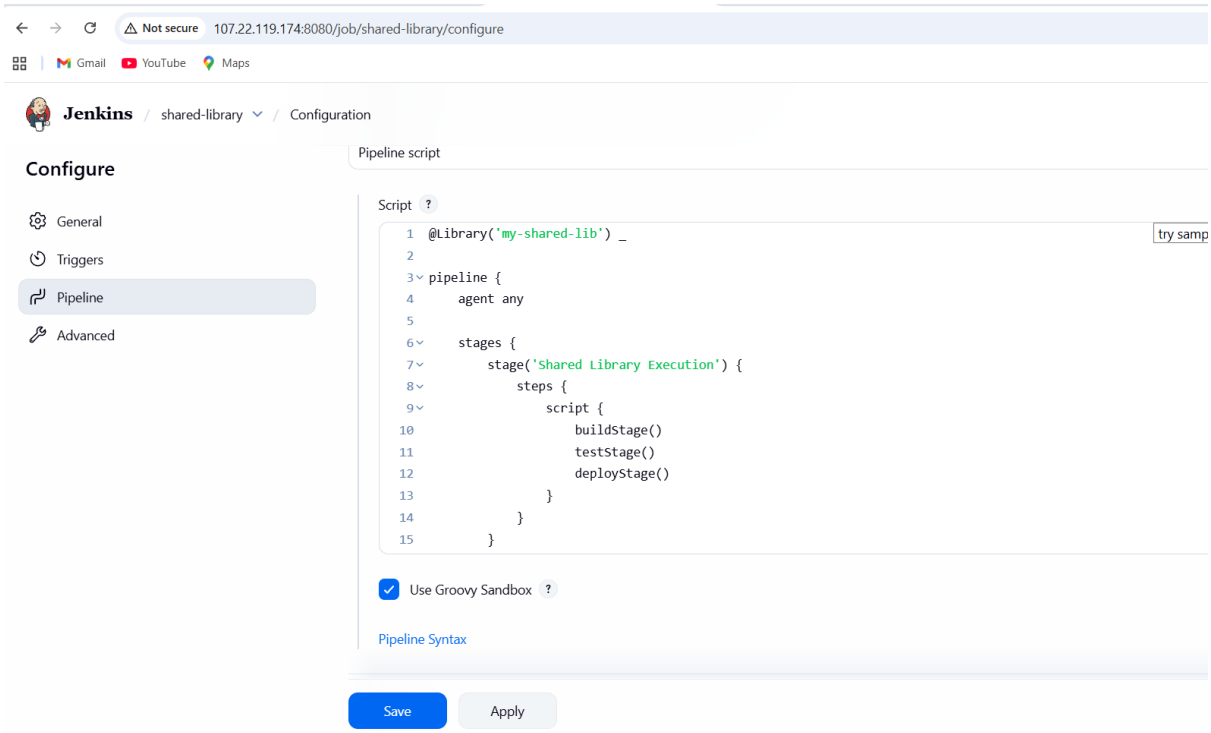
Click on save

Create a new job with the name shared-library and select type as pipeline.



In the pipeline give this pipeline script (about that you configured in global library in that global details you have given details about your github repository that you created your shared library).

```
@Library('my-shared-lib') _


pipeline {
    agent any


    stages {
        stage('Shared Library Execution') {
            steps {
                script {
                    buildStage()
                    testStage()
                    deployStage()
                }
            }
        }
    }
}
```



Click on build now.