Creating a Pipeline as Code in Jenkins involves defining your build and deployment process in a script that Jenkins can interpret and execute. Jenkins supports both scripted and declarative syntax for defining pipelines. In this example, I'll use the declarative syntax.

Let's create a simple pipeline that includes checking out code from a Git repository, building the project, running tests, and deploying the application. Assume you have a basic Maven Java project stored in a Git repository.

**Step 1: Install Necessary Plugins**

Ensure that you have the necessary plugins installed in Jenkins, such as the Git plugin and any other plugins required for your specific project.

**Step 2: Create a New Pipeline Job in Jenkins**

Open Jenkins and click on "New Item."

Enter a name for your pipeline and select "Pipeline" as the job type.

Scroll down to the "Pipeline" section and choose "Pipeline script" from the Definition dropdown.

**Step 3: Write Declarative Pipeline Script**

pipeline {

agent any

environment {

// Define environment variables if needed

// For example: JAVA\_HOME = '/path/to/java'

}

stages {

stage('Checkout') {

steps {

// Checkout code from Git repository

git 'https://github.com/your-username/your-repo.git'

}

}

stage('Build') {

steps {

// Build the Maven project

sh 'mvn clean install'

}

}

stage('Test') {

steps {

// Run tests

sh 'mvn test'

}

}

stage('Deploy') {

steps {

// Add deployment steps

// Example: Deploy to a web server or container

}

}

}

post {

always {

// Clean up steps, notifications, etc.

}

success {

// Actions to take if the pipeline is successful

}

failure {

// Actions to take if the pipeline fails

}

}

}

**Step 4: Save and Run the Pipeline**

Save your pipeline job configuration, and then click "Build Now" to trigger the pipeline. Jenkins will automatically detect changes in your Git repository and initiate the pipeline.

This example is a starting point. You can customize the pipeline script based on your specific project requirements. You may also want to explore Jenkins Shared Libraries for code reuse and maintainability, especially if you have common functionality across multiple pipelines.