

TECHNOLOGY

Computing

Caltech

**Center for Technology &
Management Education**

Jenkins and Docker Java Back End

TECHNOLOGY

Computing

Caltech

**Center for Technology &
Management Education**

Set Up Jenkins

You Already Know

Before we begin, let's recall what we have covered till now:



Agile



Git



SQL



Angular



HTML



CSS



JavaScript



Core Java

You Already Know

Before we begin, let's recall what we have covered till now:



JDBC

JDBC



JSP

JSP



Servlets

Servlets



MongoDB



Maven

You Already Know

Before we begin, let's recall what we have covered till now:

JUnit

JUnit



Spring



Spring Boot



Webservices

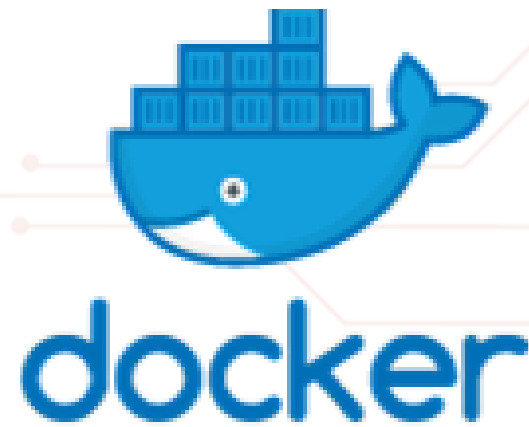


Microservices

Microservices

You Already Know

Before we begin, let's recall what we have covered till now:



Docker



Jenkins



AWS

Developed Angular Front End

- Created various components and services for the Angular eStore admin and end user.

Developed Java Back End

- Developed the Java back end for the admin and end user
- Implemented microservices with Spring Boot

Front End Back End Communication

- Used HTTP Client in Angular to communicate with Java back end
- Implemented communication for both admin and end-user projects



A Day in the Life of a Full Stack Developer

As a full stack web developer, our key role is to develop both client and server software.



Angular and Node can be used to build the front end of the web page.



Spring Boot, Java, and MySQL/MongoDB can be used to build at the back end.



A Day in the Life of a Full Stack Developer

Bob needs to develop a Java back end. He brainstorms a bit and finds a solution.

Let me use Jenkins, Docker, and AWS to build CI/CD Pipelines, containerize the apps, and finally host them to AWS EC2 instance.



In this lesson, we will create Jenkins Pipeline for CI/CD for Java back end. Moving ahead, we will create Dockerfile to build images and run them as containers in Docker and help Bob complete his task effectively and quickly.

Learning Objectives

By the end of this lesson, you will be able to:

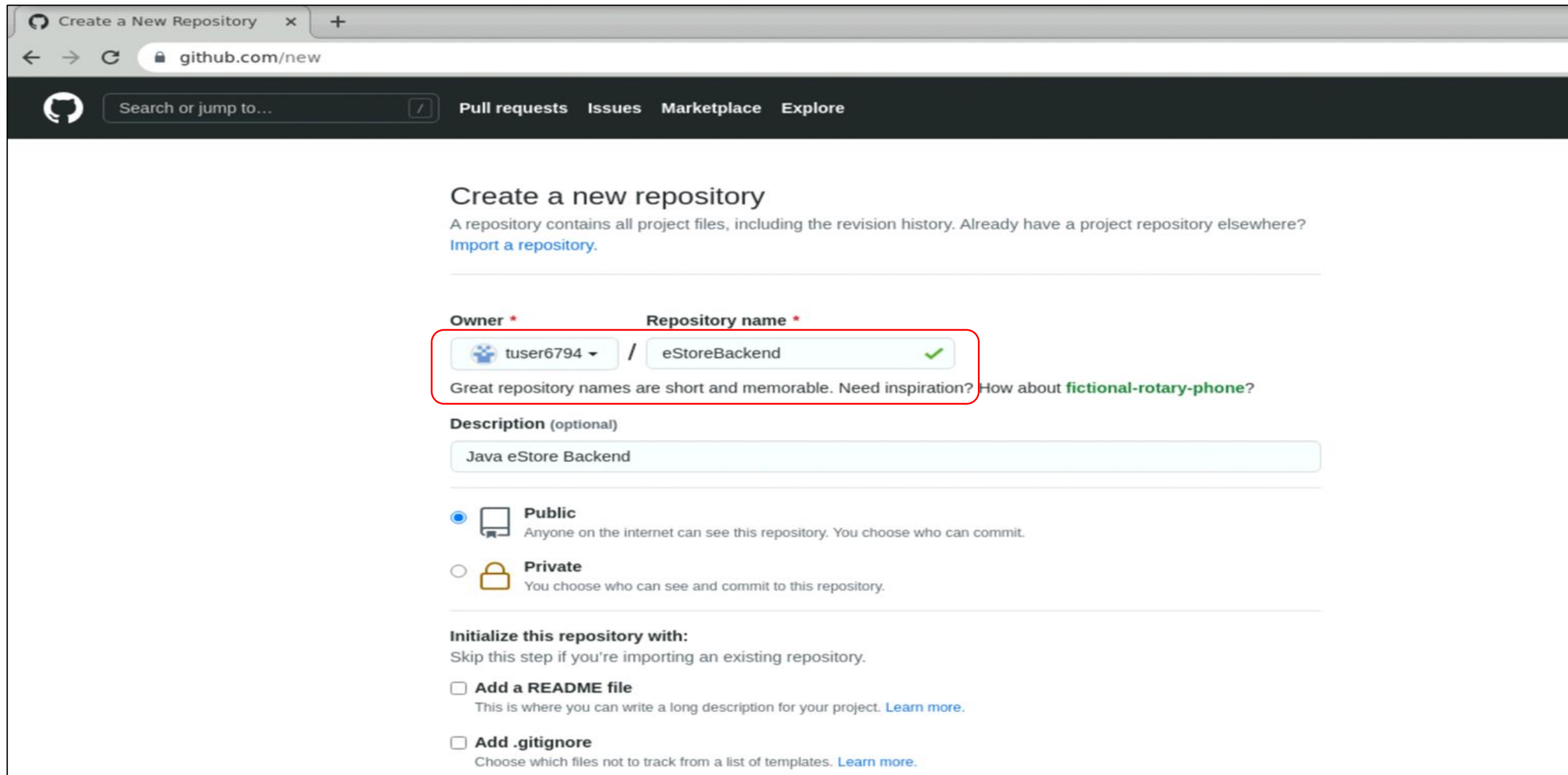
- 👁 Create Jenkins Pipeline for Java back end
- 👁 Implement CI/CD in Jenkins with Git
- 👁 Create Dockerfile for building images
- 👁 Integrate Docker in Jenkins to build and release the images as containers



Set Up Jenkins for the Java Backend Project

Create Git Repository on GitHub for the Java Backend Project

Create a new repository on your GitHub account which will be used by Jenkins to sync the code in SCM.



Create a New Repository x +

github.com/new

Search or jump to... Pull requests Issues Marketplace Explore

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * **Repository name ***

tuser6794 / eStoreBackend ✓

Great repository names are short and memorable. Need inspiration? How about [fictional-rotary-phone?](#)

Description (optional)

Java eStore Backend

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

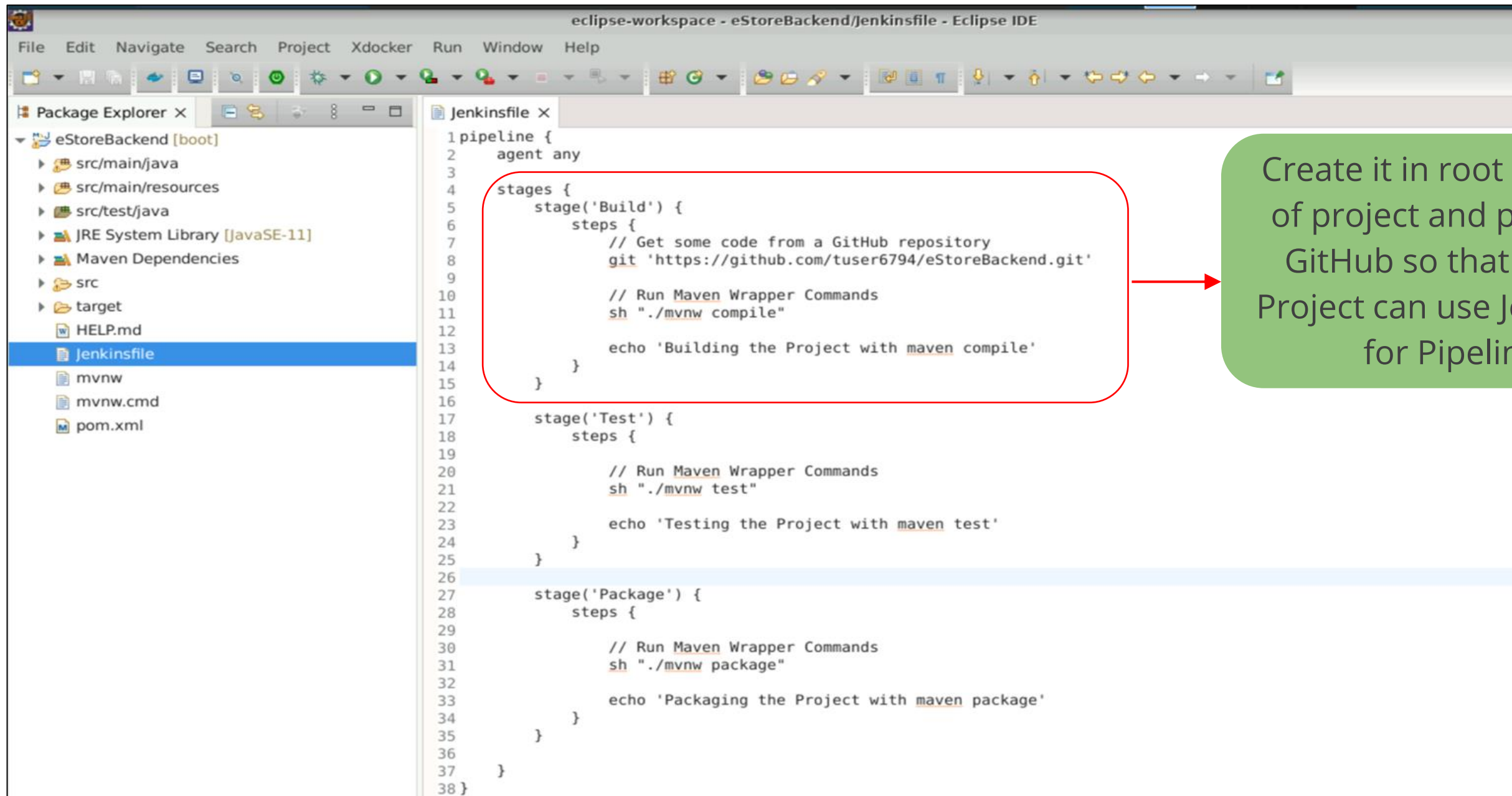
Sync the Project on Github for Java Backend Project

Push the code on GitHub for the Java Backend project.

```
erishantgmail@ip-172-31-84-97: ~/Downloads/eStoreBackend
File Edit View Search Terminal Help
erishantgmail@ip-172-31-84-97:~/Downloads/eStoreBackend$ git push -u origin master
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 4 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 842 bytes | 842.00 KiB/s, done.
Total 7 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/tuser6794/eStoreBackend.git
    ad9634b..8e7b8bc master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
erishantgmail@ip-172-31-84-97:~/Downloads/eStoreBackend$
```

Configure Jenkins Pipeline Stages for the Java Backend Project

Create a Jenkinsfile for Jenkins to build your project as Pipeline in Jenkins.



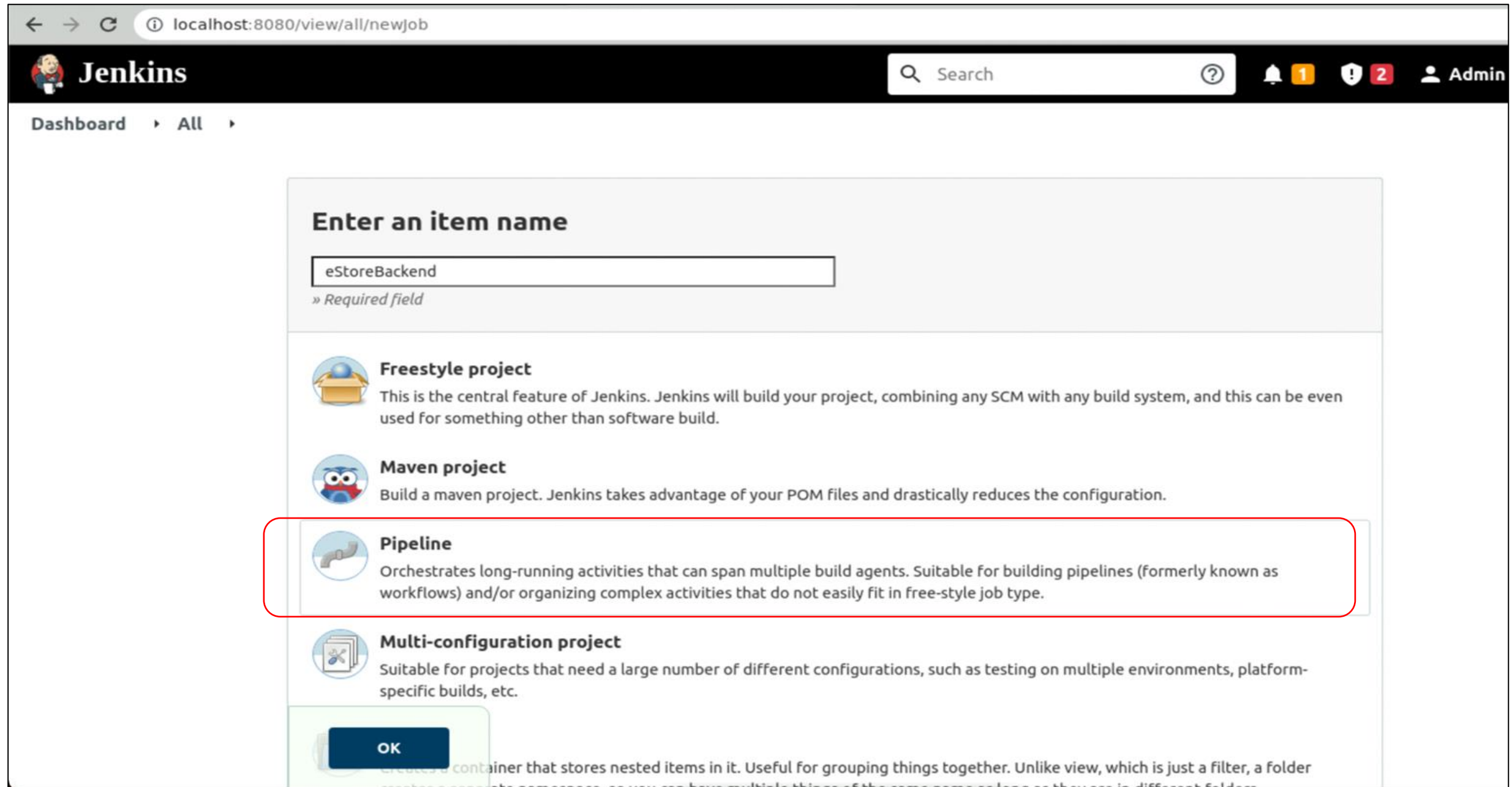
The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays the project structure for 'eStoreBackend [boot]'. The 'Jenkinsfile' is highlighted in the Package Explorer. On the right, the Jenkinsfile content is displayed in the editor. The Jenkinsfile defines a pipeline with three stages: 'Build', 'Test', and 'Package'. The 'Build' stage is highlighted with a red box, and a red arrow points from it to a green callout box on the right.

```
1 pipeline {
2   agent any
3
4   stages {
5     stage('Build') {
6       steps {
7         // Get some code from a GitHub repository
8         git 'https://github.com/tuser6794/eStoreBackend.git'
9
10        // Run Maven Wrapper Commands
11        sh './mvnw compile'
12
13        echo 'Building the Project with maven compile'
14      }
15    }
16
17    stage('Test') {
18      steps {
19
20        // Run Maven Wrapper Commands
21        sh './mvnw test'
22
23        echo 'Testing the Project with maven test'
24      }
25    }
26
27    stage('Package') {
28      steps {
29
30        // Run Maven Wrapper Commands
31        sh './mvnw package'
32
33        echo 'Packaging the Project with maven package'
34      }
35    }
36  }
37 }
38 }
```

Create it in root directory of project and push it to GitHub so that Jenkins Project can use Jenkinsfile for Pipeline.

Create a Jenkins Pipeline Project for the Java Backend Project

Create a new project in Jenkins of type Pipeline.



The screenshot shows the Jenkins web interface at the URL `localhost:8080/view/all/newjob`. The page title is "Jenkins". Below the title bar, there is a search bar and a user profile icon labeled "Admin". The breadcrumb navigation shows "Dashboard" and "All". The main content area is titled "Enter an item name" and contains a text input field with the value "eStoreBackend". Below the input field, it says "» Required field". There is a list of project types with icons and descriptions:

- Freestyle project**: This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
- Maven project**: Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
- Pipeline**: Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type. (This option is highlighted with a red border in the image.)
- Multi-configuration project**: Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

At the bottom of the list, there is an "OK" button. Below the button, there is a partially visible description: "Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders."

Configure Jenkins Pipeline SCM for the Java Backend Project

Configure the Jenkins Pipeline Project by passing GitHub Repository URL.

The screenshot shows the Jenkins web interface for configuring a pipeline. The browser tabs include 'MySpringBootApplication/Jenkinsf...', 'tuser6794/eStoreBackend: x', and 'eStoreBackend Config [Jen] x'. The address bar shows 'localhost:8080/job/eStoreBackend/configure'. The breadcrumb navigation is 'Dashboard > eStoreBackend >'. The configuration tabs are 'General', 'Build Triggers', 'Advanced Project Options', and 'Pipeline'. The 'Pipeline' tab is active, showing a red rounded rectangle highlighting the 'Definition' section. Inside this section, the 'Definition' dropdown is set to 'Pipeline script from SCM'. Below it, the 'SCM' dropdown is set to 'Git'. Under the 'Repositories' section, the 'Repository URL' is set to 'https://github.com/tuser6794/eStoreBackend.git'. The 'Credentials' section shows a dropdown set to '- none -' and an 'Add' button. At the bottom, there are 'Save' and 'Apply' buttons. An 'Advanced...' button is also visible in the top right of the configuration area.

Configure Jenkinsfile in Jenkins for the Java Backend Project

From the branch specifier, select the branch.

localhost:8080/job/eStoreBackend/configure

Dashboard > eStoreBackend >

General Build Triggers Advanced Project Options **Pipeline**

Branches to build ?

Branch Specifier (blank for 'any') ?

*/master

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

Pipeline Syntax

Save Apply

Select the default branch, which is master.

The Script Path contains the Jenkinsfile, which is created in the root directory of project.

Optional: GitHub Hook Trigger for the Java Backend Project

Configure GitHub trigger for GITScm Polling. This option will work when Jenkins is running with a proper URL instead of the localhost.

localhost:8080/job/eStoreBackend/configure

Dashboard > eStoreBackend >

General Build Triggers Advanced Project Options Pipeline

☐ Build after other projects are built ?
☐ Build periodically ?
☒ GitHub hook trigger for GITScm polling ?
☐ Poll SCM ?
☐ Disable this project ?
☐ Quiet period ?
☐ Trigger builds remotely (e.g., from scripts) ?

Configure the same using ngrok.

Advanced Project Options

Advanced...

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

Save Apply 6794/eStoreBackend.git

Run the Jenkins Pipeline Project for the Java Backend Project

Notice stages appearing as mentioned in the Jenkinsfile.
Source > Test > Build > Containerize > Deploy

The screenshot shows the Jenkins web interface for the 'eStoreBackend' project. The left sidebar contains navigation options: Back to Dashboard, Status, Changes, Build Now (highlighted with a red box), Configure, Delete Pipeline, Full Stage View, Rename, and Pipeline Syntax. The main area displays the 'Pipeline eStoreBackend' view. A green callout box with the text 'Finally, click on Build Now and build your project.' has an arrow pointing to the 'Build Now' button. Below the pipeline title, there is a 'Recent Changes' section and a 'Stage View' table. The 'Stage View' table shows the following stages and their durations:

Stage	Duration
Declarative: Checkout SCM	261ms
Build	3s
Test	11s
Package	12s

Below the table, the 'Permalinks' section shows the last build (#1) completed on Feb 27, 2022, at 5:27 AM, with a status of 'No Changes'.

TECHNOLOGY

Computing

Caltech

**Center for Technology &
Management Education**

Dockerise the Pipeline

Containerize the Java Backend Project

Configure Docker Using Dockerfile for Java Backend Project

Write the Dockerfile which will be used to **dockerize** Java for the Backend project.

The screenshot shows the Eclipse IDE workspace for a project named 'eStoreBackend'. The 'Package Explorer' on the left lists the project structure, including 'src/main/java', 'src/main/resources', 'src/test/java', 'JRE System Library [JavaSE-11]', 'Maven Dependencies', 'src', 'target', and a 'Dockerfile'. The 'Dockerfile' is selected and its content is displayed in the editor. The Dockerfile contains the following instructions:

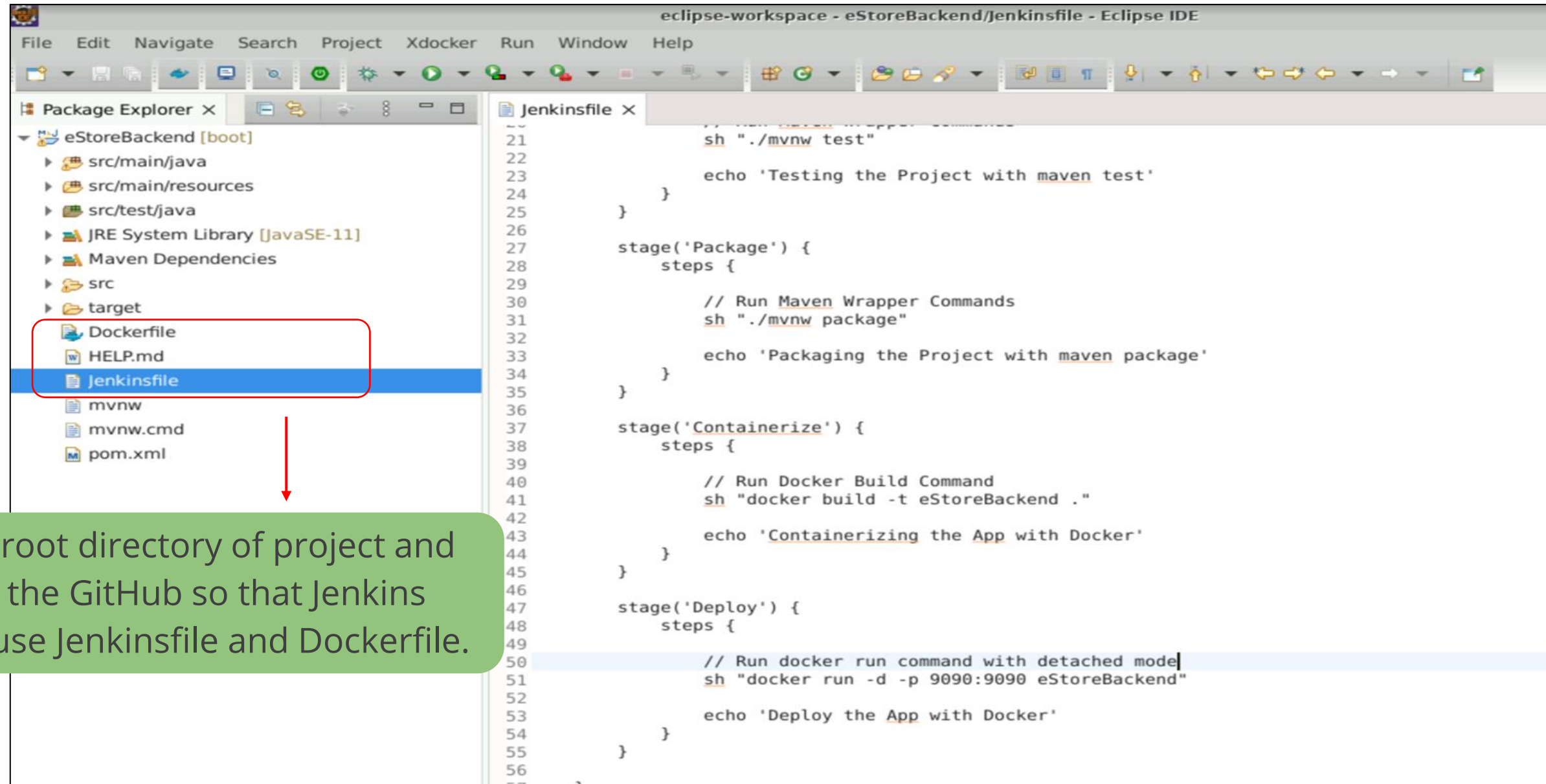
```
1 # Use the base image as JDK 11
2 FROM openjdk:11
3
4 # Create an Argument with default path as target directory
5 ARG JAR_FILE=target/*.jar
6
7 # Copy the Jar file as app.jar
8 COPY ${JAR_FILE} app.jar
9
10 # Execute the jar file which will run the project on port 9090
11 ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Annotations with arrows point from specific lines in the Dockerfile to explanatory text boxes:

- Lines 1-5 are grouped by a red box, with an arrow pointing to a green box containing the text: "Build the jar file of Java Backend project for production."
- Lines 10-11 are grouped by a red box, with an arrow pointing to a green box containing the text: "Serve the application by executing the Jar on port 9090."

Configure Jenkins Pipeline Stages for Java Backend Project

Add the stages for containerizing the Java Backend that is **building and running** the container finally.



The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays the project structure for 'eStoreBackend [boot]'. The 'Jenkinsfile' is highlighted in the Package Explorer, and a red arrow points to it. The main editor shows the content of the 'Jenkinsfile', which defines three stages: 'Test', 'Package', and 'Containerize'. The 'Containerize' stage includes a step to build the Docker image and another to run the container.

```
21 sh "./mvnw test"
22
23 echo 'Testing the Project with maven test'
24 }
25 }
26
27 stage('Package') {
28     steps {
29
30         // Run Maven Wrapper Commands
31         sh "./mvnw package"
32
33         echo 'Packaging the Project with maven package'
34     }
35 }
36
37 stage('Containerize') {
38     steps {
39
40         // Run Docker Build Command
41         sh "docker build -t eStoreBackend ."
42
43         echo 'Containerizing the App with Docker'
44     }
45 }
46
47 stage('Deploy') {
48     steps {
49
50         // Run docker run command with detached mode
51         sh "docker run -d -p 9090:9090 eStoreBackend"
52
53         echo 'Deploy the App with Docker'
54     }
55 }
56 }
```

Create it in root directory of project and push it to the GitHub so that Jenkins project can use Jenkinsfile and Dockerfile.

Run the Jenkins Pipeline Project for Java Backend Project

Notice stages appearing as mentioned in the Jenkinsfile.
Source > Test > Build > Containerize > Deploy

Pipeline eStoreBackend

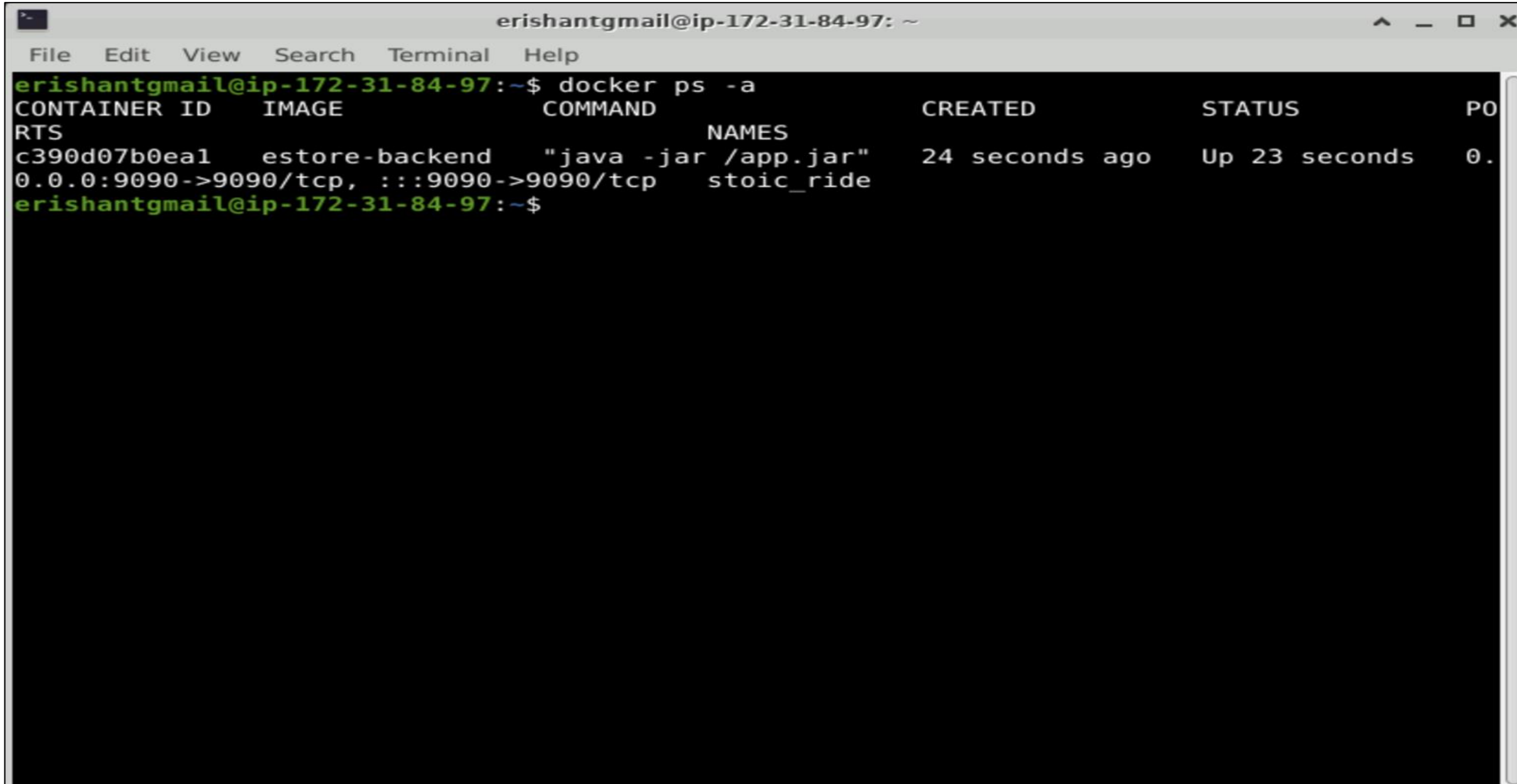
Click on Build Now and build your project.

Stage View

	Declarative: Checkout SCM	Build	Test	Package	Containerize	Deploy
Average stage times: (Average full run time: ~27s)	275ms	2s	11s	11s	1s	881ms
#4 Feb 27 06:29 2 commits	289ms	2s	10s	10s	1s	881ms
#1 Feb 27 05:27 No Changes	261ms	3s	11s	12s		

Check Docker ps -a

Check status of running container by docker ps -a, which is the command to validate Jenkins build.



```
erishantgmail@ip-172-31-84-97: ~  
File Edit View Search Terminal Help  
erishantgmail@ip-172-31-84-97:~$ docker ps -a  
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS  
c390d07b0ea1   estore-backend "java -jar /app.jar"    24 seconds ago Up 23 seconds 0.0.0.0:9090->9090/tcp, :::9090->9090/tcp  
erishantgmail@ip-172-31-84-97:~$
```

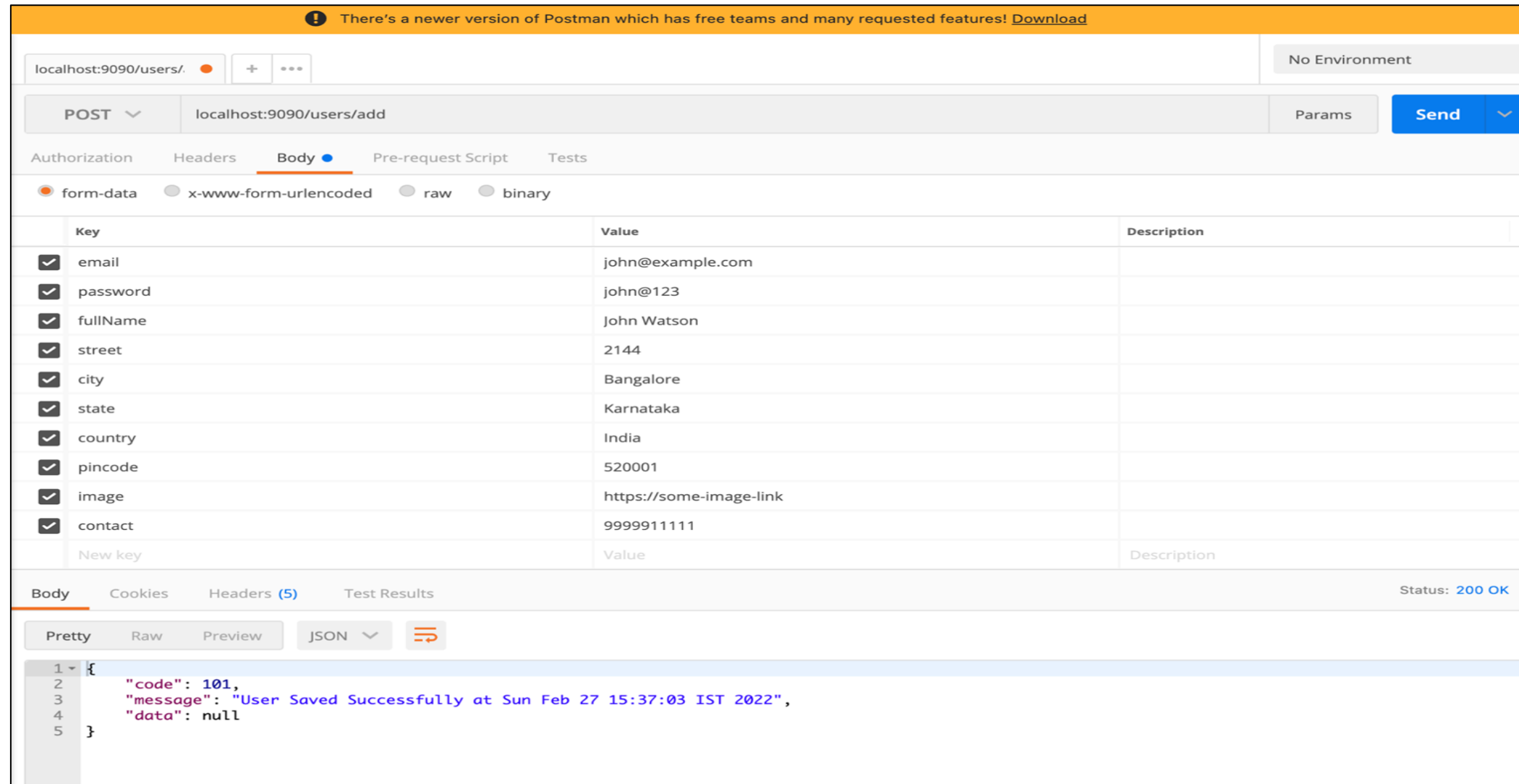
Check the Backend Project on Localhost

The Backend project is up and running in the docker container.



Check Service on Postman

Validate if the project is up with services running.



Key Takeaways

- 🕒 Jenkins Pipeline is built for Java Back end using Jenkinsfile.
- 🕒 Java back end is dockerized using Dockerfile.
- 🕒 Jenkins Pipeline is built to dockerize the Java back end.
- 🕒 Service is validated using Postman.



Before the Next Class

You have successfully completed this session. For the next end-user module discussion, you should:

- Review AWS services
- Explore steps to create EC2 and S3 buckets
- Explore how to connect to EC2 instance
- Review Jenkins
- Review Docker commands



What's Next?

Now we have finished our classes and design pattern for the Backend project with respect to end-user module. In our next live session, we will:

- See how to create EC2 instance on AWS
- See how to work with storage buckets using S3
- Deploy Angular apps on EC2
- Use Jenkins and Docker as DevOps tools on EC2

