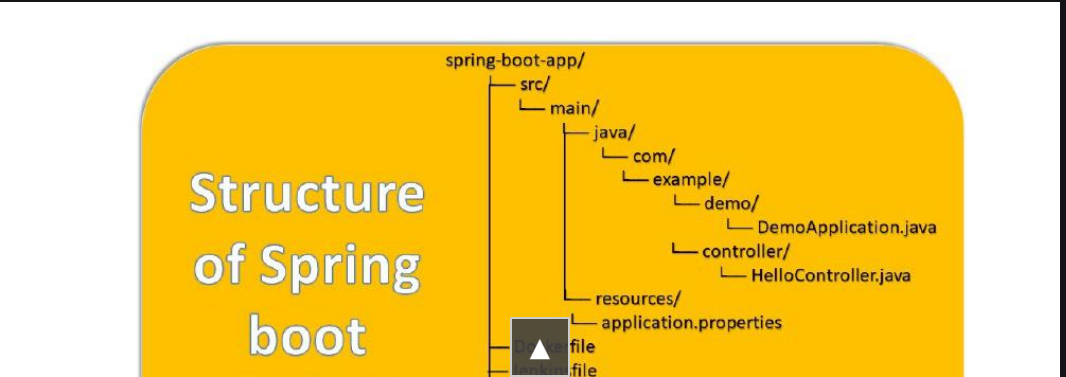
Spring Boot Application Deployment in Kubernetes with Jenkins CI/CD Pipeline

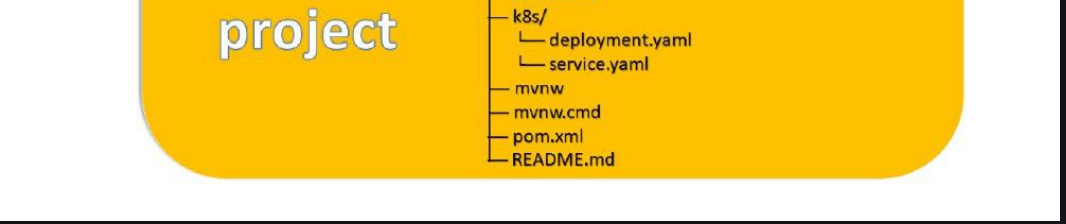
Last Updated : 25 Jul, 2024

As we know in the modern world we use Continuous Integration and Continuous Deployment for fast and and efficient delivery of our application to the market. Here in this Article, we are deep-diving into how we can actually deploy Spring boot applications on Kubernetes with the help of Jenkins CI/CD.

Project WorkFlow

- **.java files:** These files contain the Java source code for the application.
- **application.properties:** This file contains configuration properties for the application.
- **Dockerfile:** This file contains instructions for building a Docker image of the application.
- **Jenkinsfile:** This file contains instructions for building and deploying the application using Jenkins, a continuous integration and continuous delivery (CI/CD) tool.
- **deployment.yaml** and **service.yaml:** These files are YAML files that define how the application will be deployed in a Kubernetes cluster.
- **mvnw** and **mvnw.cmd:** These files are wrappers for the Maven commands that can be used to build, test, and package the application.
- **pom.xml:** This file is the project object model (POM) for the application. It contains information about the project, such as its dependencies and build configuration.
- **README.md:** This file is a markdown file that contains information about the project, such as how to set it up and how to use it.





Primary Terminologies

- **Spring boot:** It is free and open-source java framework which is built on the top of the spring framework. Their main benefits are Faster development, Improve the productivity and suitable for stand-alone application, production ready features and production ready features.
- **Node.js:** This is a free, open-source cross-platform and JavaScript runtime environment that executes JavaScript code outside the browser.
- **Jenkins:** It's also free, Open-source CI/CD tools that automate software development processes like Build, Test, and deployment.
- **Docker:** Docker is a platform that allows you to put all applications and their dependencies in standardized units called containers. These containers are lightweight and portable, allowing for continuity across environments (operating systems and hardware).
- **Kubernetes:** It is open-source containerized orchestration tool that basically uses it for ease of deployment, auto-scaling, auto-healing, and monitoring purposes.

GitHub: It is a free and open-source version control system which is developed by Microsoft. That provides,

- Storage for our codes
- Continuously track our code.
- the capability of sharing with another
- project collaboration support

Step-by-Step Guide For Spring boot application deployment in Kubernetes with Jenkins CI/CD pipeline

Step 1: Update System packages

```
sudo apt-get update
```

The reason for updating is that sometimes some packages of Docker are not working well, so for that reason we update them.

APT stands for Advanced Packaging Tool.

```
romil@LAPTOP-GV251119:~$ sudo apt-get update
[sudo] password for romil:
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:2 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1517 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [259 kB]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [1933 kB]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/restricted Translation-en [329 kB]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [858 kB]
Get:11 http://security.ubuntu.com/ubuntu jammy-security/universe Translation-en [166 kB]
Get:12 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 c-n-f Metadata [16.8 kB]
Get:13 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [37.2 kB]
Get:14 http://security.ubuntu.com/ubuntu jammy-security/multiverse Translation-en [7588 B]
Get:15 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 c-n-f Metadata [260 B]
Get:16 http://archive.ubuntu.com/ubuntu jammy/universe Translation-en [5652 kB]
Get:17 http://archive.ubuntu.com/ubuntu jammy/universe amd64 c-n-f Metadata [286 kB]
Get:18 http://archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [217 kB]
Get:19 http://archive.ubuntu.com/ubuntu jammy/multiverse Translation-en [112 kB]
Get:20 http://archive.ubuntu.com/ubuntu jammy/multiverse amd64 c-n-f Metadata [8372 B]
Get:21 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1731 kB]
Get:22 http://archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [318 kB]
Get:23 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [1990 kB]
Get:24 http://archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [338 kB]
Get:25 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1086 kB]
Get:26 http://archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [251 kB]
Get:27 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 c-n-f Metadata [22.1 kB]
Get:28 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [43.0 kB]
Get:29 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse Translation-en [10.7 kB]
Get:30 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 c-n-f Metadata [472 B]
Get:31 http://archive.ubuntu.com/ubuntu jammy-backports/main amd64 Packages [67.1 kB]
Get:32 http://archive.ubuntu.com/ubuntu jammy-backports/main Translation-en [11.0 kB]
Get:33 http://archive.ubuntu.com/ubuntu jammy-backports/main amd64 c-n-f Metadata [388 B]
Get:34 http://archive.ubuntu.com/ubuntu jammy-backports/restricted amd64 c-n-f Metadata [116 B]
Get:35 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [27.2 kB]
Get:36 http://archive.ubuntu.com/ubuntu jammy-backports/universe Translation-en [16.3 kB]
Get:37 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 c-n-f Metadata [644 B]
Get:38 http://archive.ubuntu.com/ubuntu jammy-backports/multiverse amd64 c-n-f Metadata [116 B]
Fetched 31.8 MB in 13s (2373 kB/s)
Reading package lists... Done
romil@LAPTOP-GV251119:~$
```

Step 2: creating spring-boot-application

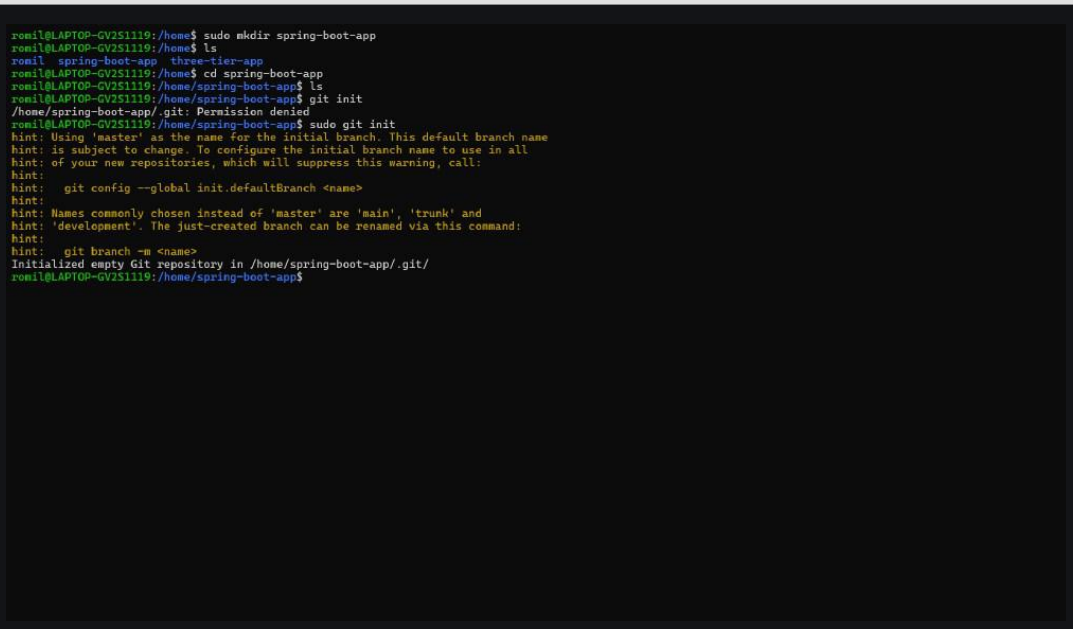
Here First we are creating a directory that names spring-boot-app.

As per the structure of the spring boot application we are creating files and folder.

```
sudo mkdir spring-boot-app
```

after creation of spring-boot-app directory we are moving to inside it using below command.

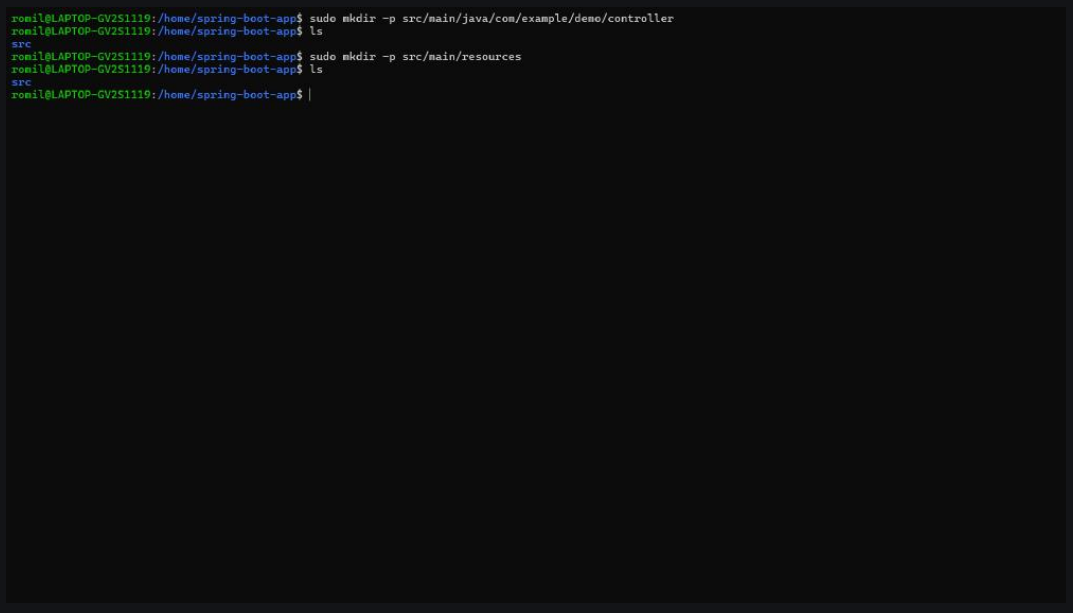
```
cd spring-boot-app
```



Then we are creating another directory as per the structure of the spring boot application diagram.

```
mkdir -p src/main/java/com/example/demo/controller
mkdir -p src/main/resources
```

after completion of the directory we check through **"ls"** command.



Create **DemoApplication.java** file inside the **demo** directory.

```
nano DemoApplication.java
```

we use "**sudo**" for super user permission. and "**nano**" for file creation instead of it you can also use "**vim**".

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

After writing above scripts then Save and Exit nano:

- Press Ctrl + O to write out (save) the file.

- Press Enter to confirm the filename.
- Press Ctrl + X to exit nano.

Create **HelloController.java** file inside the **controller** directory.

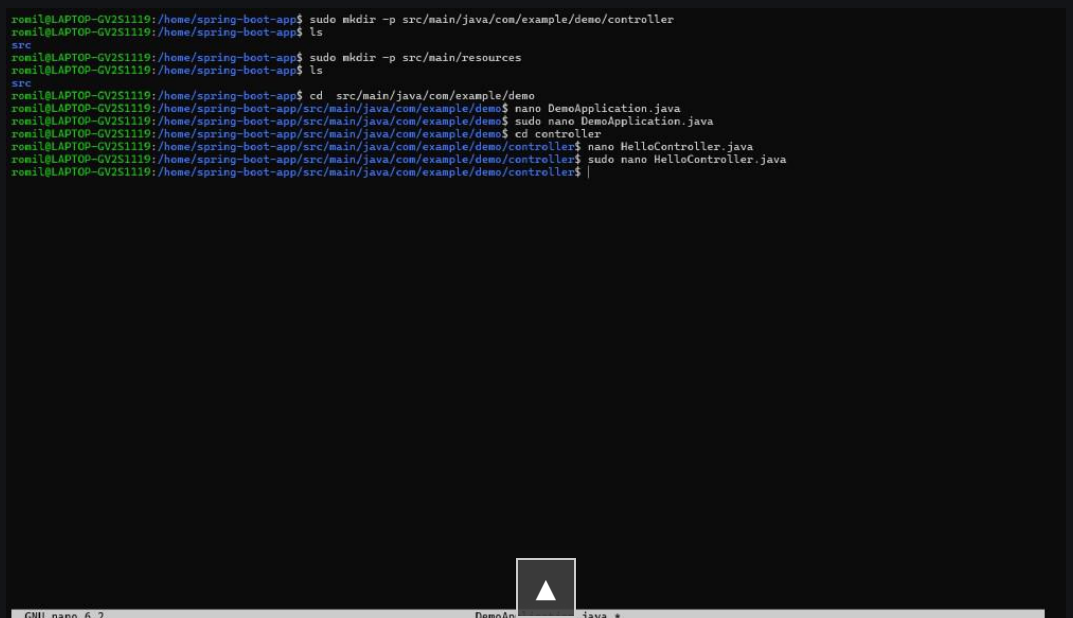
```
nano HelloController.java
```

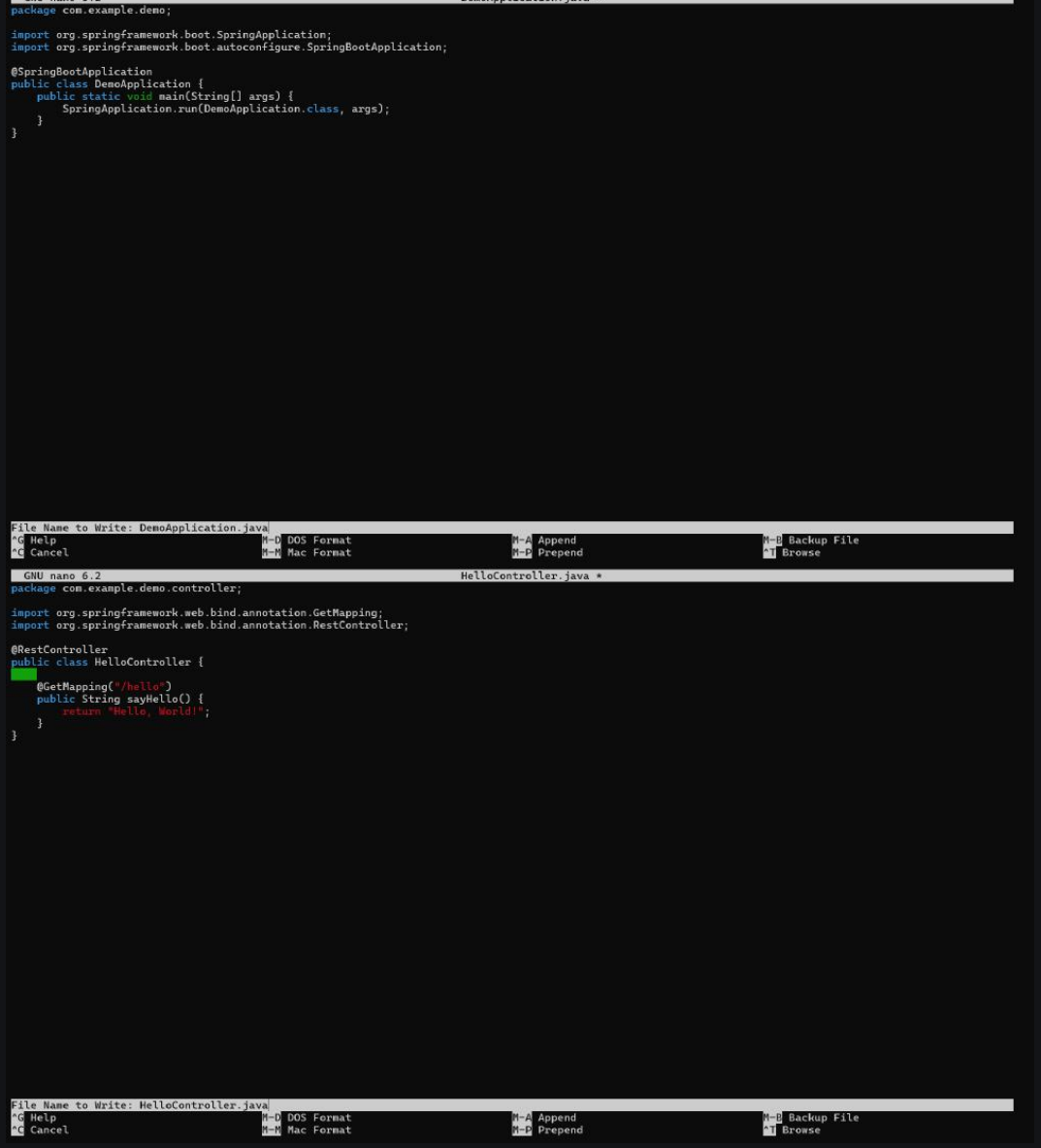
we use **"sudo"** for super user permission. and **"nano"** for file creation instead of it you can also use **"vim".**

```
package com.example.demo.controller;

import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @GetMapping("/hello")
    public String sayHello() {
        return "Hello, World!";
    }
}
```

After writing above scripts then Save and Exit nano:

- Press Ctrl + O to write out (save) the file.
- Press Enter to confirm the filename.
- Press Ctrl + X to exit nano.





Create **application.properties** file inside the **resources** directory.

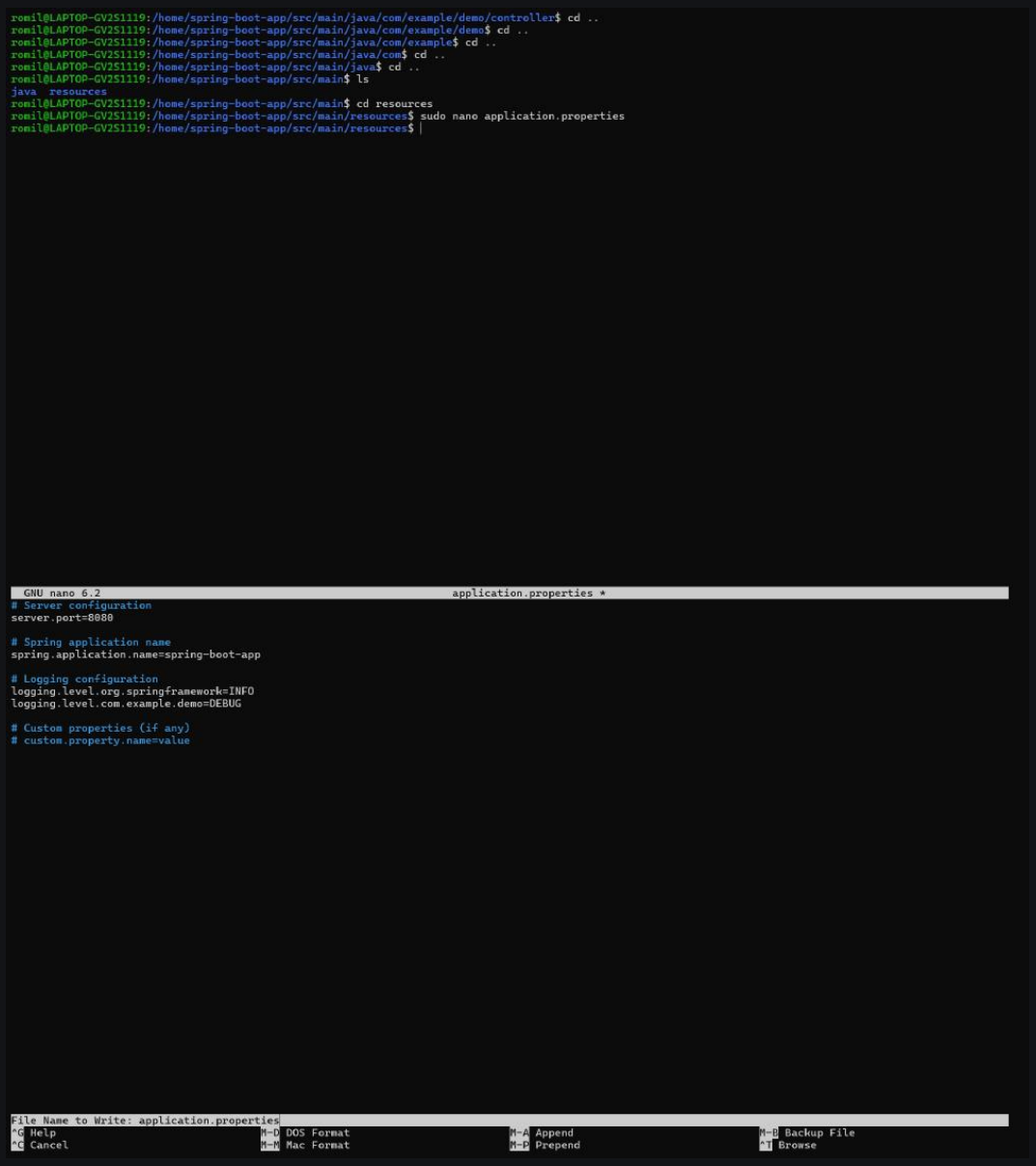
```
nano application.properties
```

we use **"sudo"** for super user permission. and **"nano"** for file creation instead of it you can also use **"vim".**

```
# Server configuration
server.port=8080

# Spring application name
spring.application.name=spring-boot-app

# Logging configuration
logging.level.org.springframework=INFO
logging.level.com.example.demo=DEBUG

# Custom properties (if any)
# custom.property.name=value
```

After writing above scripts then Save and Exit nano:

- Press Ctrl + O to write out (save) the file.
- Press Enter to confirm the filename.
- Press Ctrl + X to exit nano.

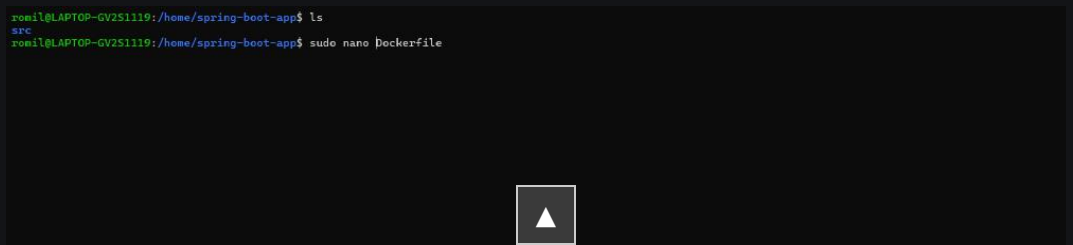


Step 3: Dockerfile Creatition

Create Dockerfile inside the spring-boot-app directory.

```
sudo nano Dockerfile
```

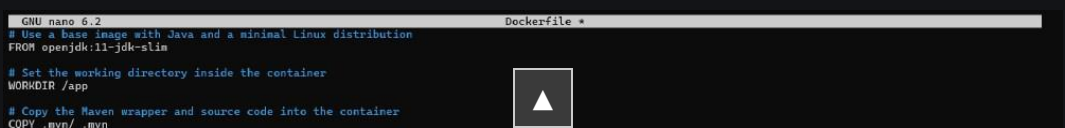
we use **"sudo"** for super user permission. and **"nano"** for file creation instead of it you can also use **"vim".**

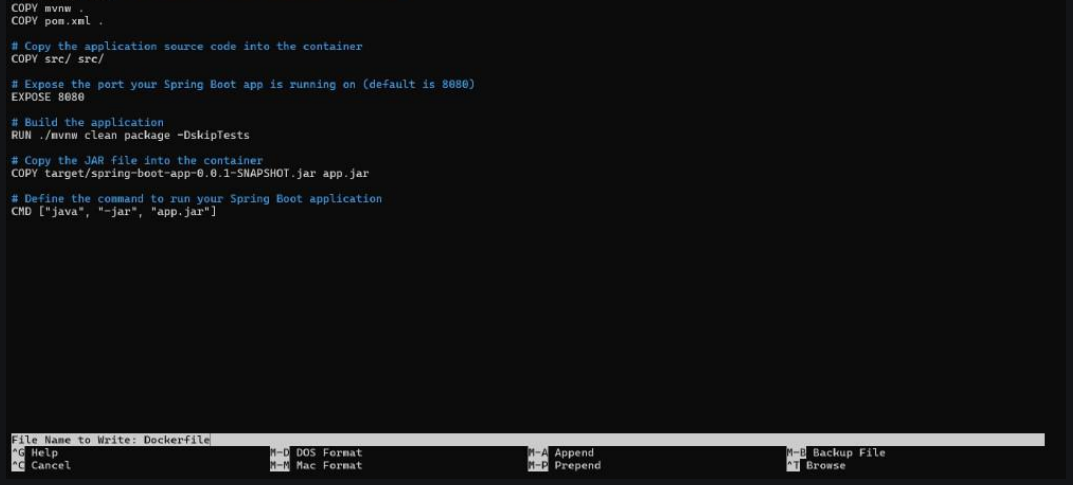


Add configuration scripts in the **Dockerfile**.

```
# Use a base image with Java and a minimal Linux distribution
FROM openjdk:11-jdk-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the Maven wrapper and source code into the container
COPY .mvn/ .mvn
COPY mvnw .
COPY pom.xml .

# Copy the application source code into the container
COPY src/ src/

# Expose the port your Spring Boot app is running on (default
is 8080)
EXPOSE 8080

# Build the application
RUN ./mvnw clean package -DskipTests

# Copy the JAR file into the container
COPY target/spring-boot-app-0.0.1-SNAPSHOT.jar app.jar

# Define the command to run your Spring Boot application
CMD ["java", "-jar", "app.jar"]After writing above scripts
then Save and Exit nano:
```

- Press Ctrl + O to write out (save) the file.
- Press Enter to confirm the filename.
- Press Ctrl + X to exit nano.



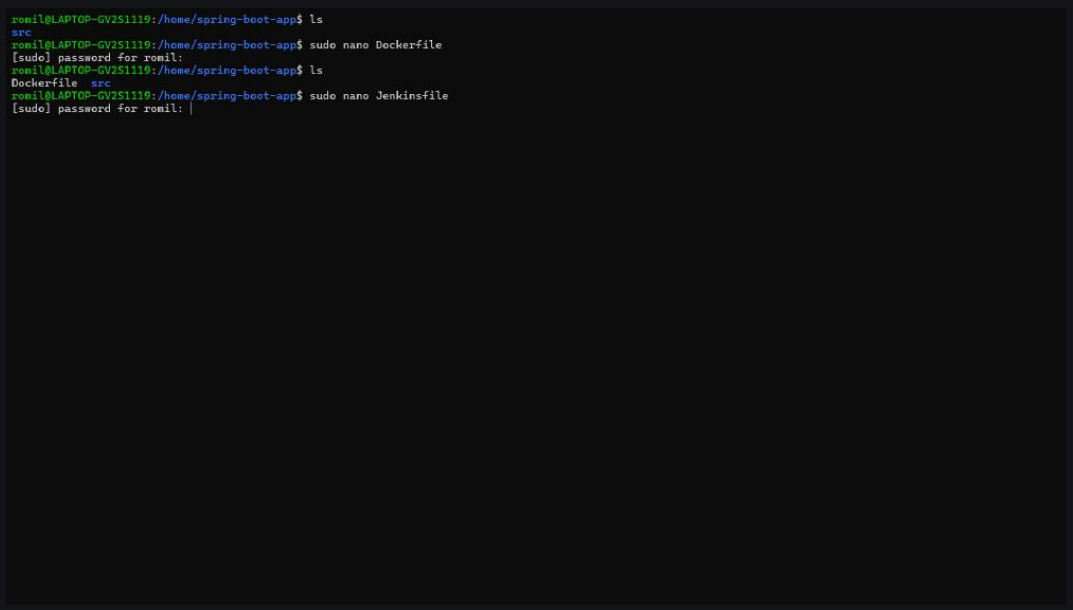


Step 4: Jenkinsfile Creatition

Create Jenkinsfile inside the spring-boot-app directory.

```
sudo nano Jenkinsfile
```

we use **"sudo"** for super user permission. and **"nano"** for file creation instead of it you can also use **"vim".**



Add pipeline configuration scripts in the Jenkinsfile.

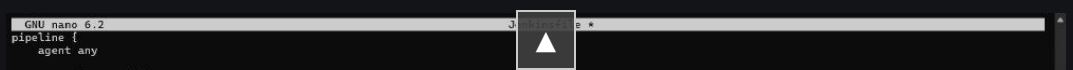
```
pipeline {
    agent any

       environment {
        DOCKER_IMAGE = "romilbhai/spring-boot-app"
        K8S_NAMESPACE = "springboot-demo"
        DOCKER_CREDENTIALS_ID = 'dockerhub-credentials-id' //
Define Docker Hub credentials ID
        KUBECONFIG_CREDENTIALS_ID = 'kubeconfig-springboot'
// Define Kubernetes config credentials ID
```

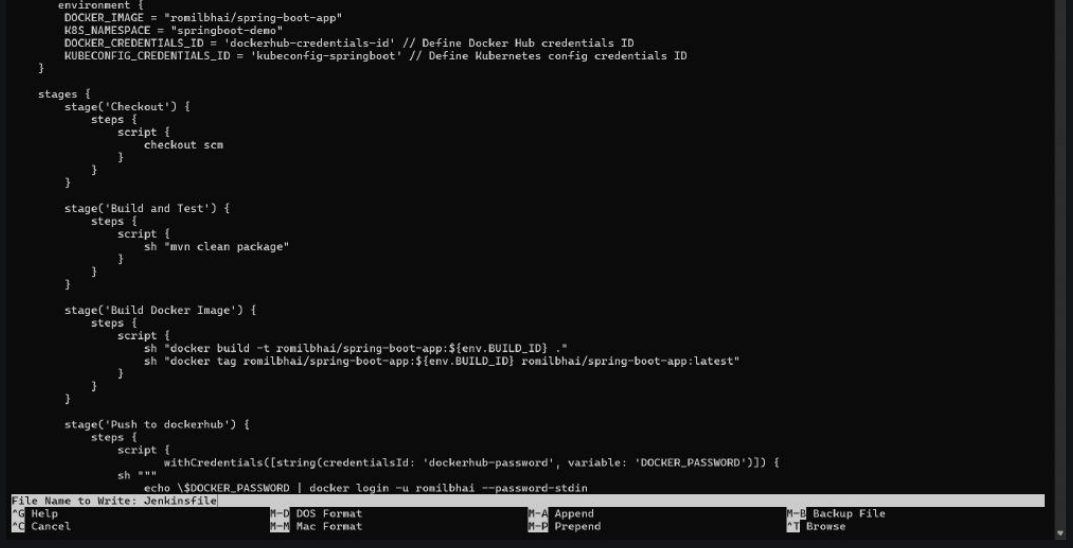
```
    }

    stages {
        stage('Checkout') {
            steps {
                script {
                    checkout scm
                }
            }
        }

        stage('Build and Test') {
            steps {
                script {
                    sh "mvn clean package"
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    sh "docker build -t romilbhai/spring-
boot-app:${env.BUILD_ID} ."
                    sh "docker tag romilbhai/spring-boot-app:
${env.BUILD_ID} romilbhai/spring-boot-app:latest"
                }
            }
        }

        stage('Push to dockerhub') {
            steps {
                script {
                       withCredentials([string(credentialsId:
'dockerhub-password', variable: 'DOCKER_PASSWORD')]) {
                sh """
                    echo \$DOCKER_PASSWORD | docker login -u
romilbhai --password-stdin
                """
                sh "docker push romilbhai/spring-boot-app:
${env.BUILD_ID}"
                sh "docker push romilbhai/spring-boot-
app:latest"
            }
```

```
                    }
                }
            }


        stage('Deploy to Kubernetes') {
            steps {
                script {

                    kubeconfig(credentialsId: 'kubeconfig')
{

                        sh 'kubectl apply -f k8s/
deployment.yaml -n ${env.K8S_NAMESPACE}'
                        sh 'kubectl apply -f k8s/service.yaml
-n ${env.K8S_NAMESPACE}'
                    }


                }
            }
        }
    }

    post {
        always {
            cleanWs()
        }
        success {
            echo 'Build and deployment successful!'
        }
        failure {
            echo 'Build or deployment failed.'
        }
    }
}
```

After writing above scripts then Save and Exit nano:

- Press Ctrl + O to write out (save) the file.
- Press Enter to confirm the filename.
- Press Ctrl + X to exit nano.





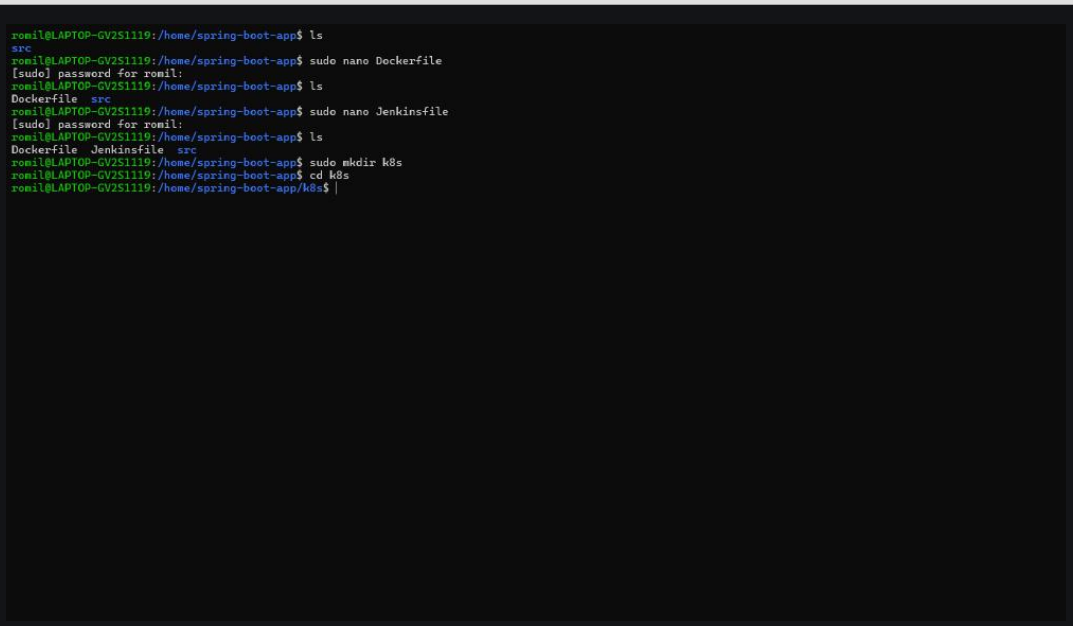
Step 5: Kubernetes files creation

Then we are creating another directory as per the structure of the spring boot application diagram.

```
sudo mkdir k8s
```

after creation of "**k8s**" directory we are moving to inside it using below command.

```
cd k8s
```



Create deployment file inside the k8s directory.

```
sudo nano deployment.yaml
```

we use "**sudo**" for super user permission. and "**nano**" for file creation instead of it you can also use "**vim**".

▲

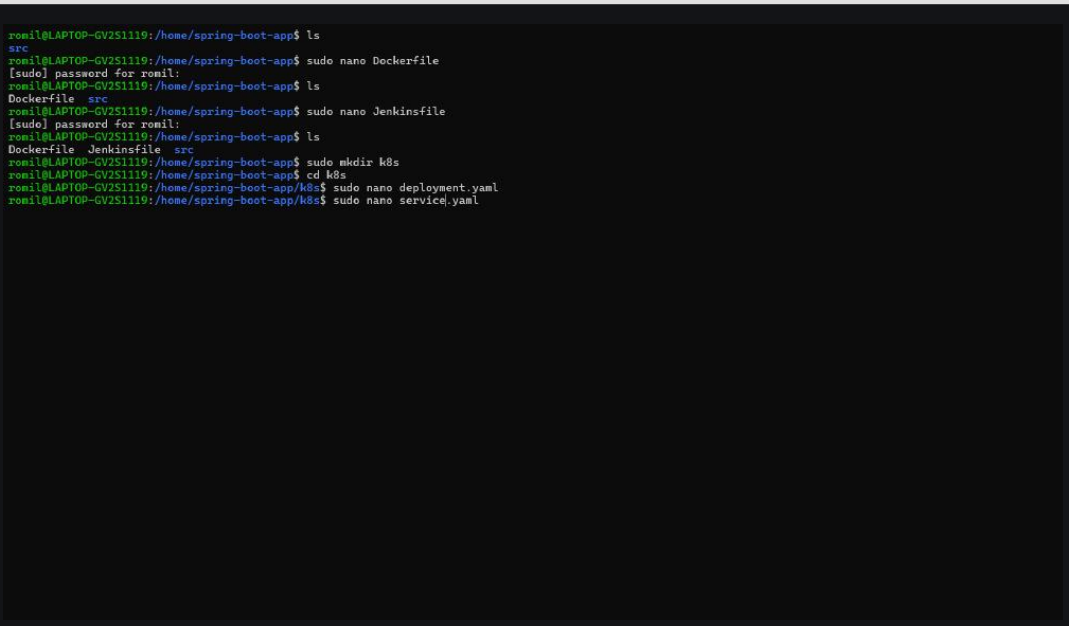
Add deployment configuration scripts in the "**deployment.yaml"** file.

After writing above scripts then Save and Exit nano:

- Press Ctrl + O to write out (save) the file.
- Press Enter to confirm the filename.
- Press Ctrl + X to exit nano.

Create service file inside the k8s directory.

```
sudo nano service.yaml
```



we use "**sudo**" for super user permission. and "**nano**" for file creation instead of it you can also use "**vim**".

Add service configuration scripts in the "**service.yaml**" file.

```
apiVersion: v1
kind: Service
```

```
metadata:
  name: spring-boot-app-service
  namespace: springboot-demo
spec:
  selector:
    app: spring-boot-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```

After writing above scripts then Save and Exit nano:

- Press Ctrl + O to write out (save) the file.
- Press Enter to confirm the filename.
- Press Ctrl + X to exit nano.

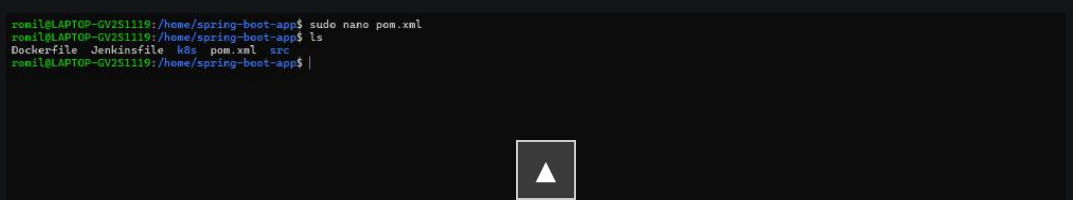
Step 6: POM.xml file creation

Create pom.xml file inside the spring-boot-app directory.

```
sudo nano pom.xml
```

we use "**sudo**" for super user permission. and "**nano**" for file creation instead of it you can also use "**vim**".

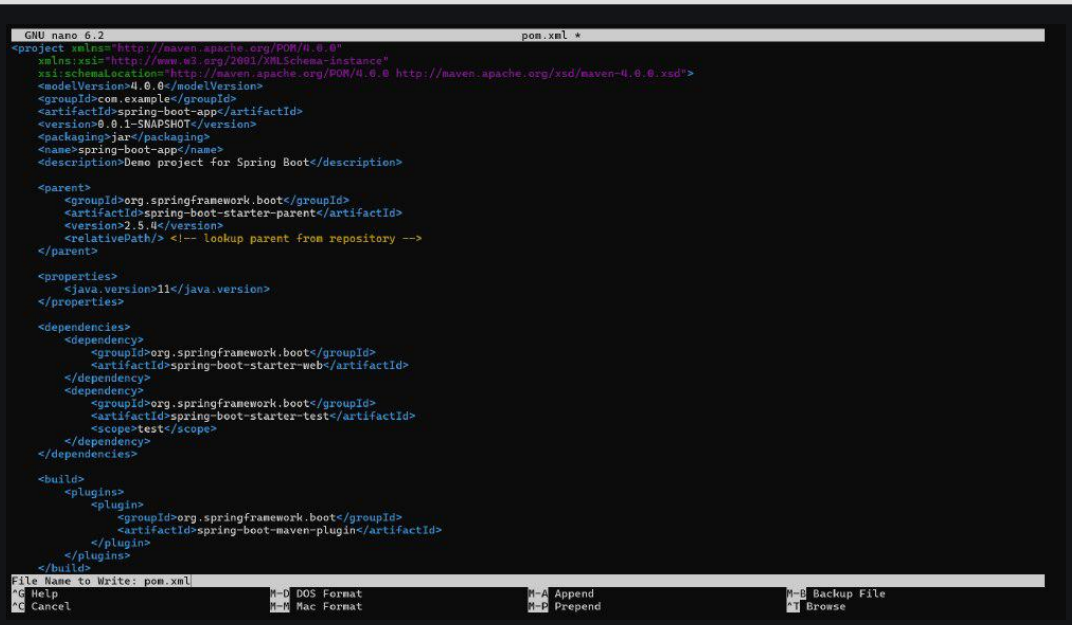
after completion of the directory we check through "**ls**" command.



Add xml code in the POM.xml file.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>spring-boot-app</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>spring-boot-app</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.4</version>
        <relativePath/> <!-- lookup parent from repository --
>
    </parent>

    <properties>
        <java.version>11</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
```

```
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</
artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```



After writing above scripts then Save and Exit nano:

- Press Ctrl + O to write out (save) the file.
- Press Enter to confirm the filename.
- Press Ctrl + X to exit nano.

Step 7: Building and running Docker image.

```
docker build -t spring-boot-app:latest .
docker run -d -it spring-boot-app:latest
```

Here, ('-t') stands for tag, which basically allows you to give a name or tag with a Docker image. In this scenario, spring-boot-app:latest is the tag or name that we want to build. and

('-i') stands for interactive and basically allows us to interact with command-

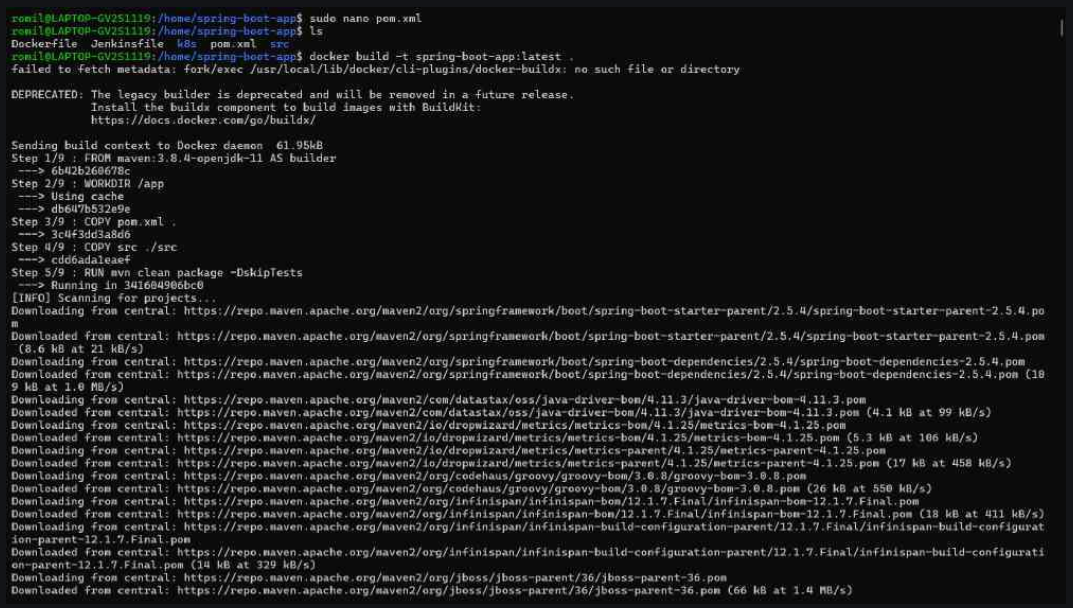
line. It keeps STDIN (standard input) open even if not attached.

If we combine ('-it'), that means that it allows you to run a tagged or named container with an interactive command interface.

Once you write the build command, the image will be created, and through this image, a Docker container will run.

You can check your container working status through 'Docker -ps' command.

('-d') stands for daemon that run process in the background.



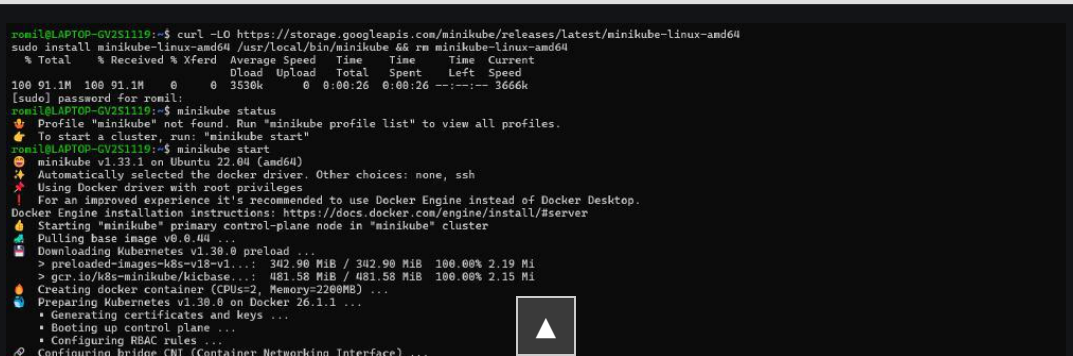
Step 8: Setup Kubernetes

installing latest minikube stable x86-64 Linux variant, Installation as per your requirement.

Follow Minikube for more detailed guidance.

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64
```

```
minikube start
```



```
Verifying Kubernetes components...
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: storage-provisioner, default-storageclass
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
romil@LAPTOP-GV2S1119:~$
```

Step 9: Creating namespace

```
kubectl create namespace springboot-demo
```

Here we are creating a namespace called springboot-demo because otherwise namespace is default.

```
Restarting existing docker container for "minikube" ...
Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
Verifying Kubernetes components...
  • Using image docker.io/kubernetesui/dashboard:v2.7.0
  • Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
Some dashboard features require the metrics-server addon. To enable all features please run:

    minikube addons enable metrics-server

Enabled addons: default-storageclass, storage-provisioner, dashboard
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
romil@LAPTOP-GV2S1119:~$ kubectl status
error: unknown command "status" for "kubectl"
romil@LAPTOP-GV2S1119:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

romil@LAPTOP-GV2S1119:~$ kubectl apply -f deployment.yaml
error: the path "deployment.yaml" does not exist
romil@LAPTOP-GV2S1119:~$ cd ..
romil@LAPTOP-GV2S1119:/home$ cd spring-boot-app
romil@LAPTOP-GV2S1119:/home/spring-boot-app$ ls
Dockerfile  Jenkinsfile  k8s  pom.xml  src
romil@LAPTOP-GV2S1119:/home/spring-boot-app$ cd k8s
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$ ls
deployment.yaml  service.yaml
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$ kubectl apply -f deployment.yaml
Error from server (NotFound): error when creating "deployment.yaml": namespaces "springboot-demo" not found
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$ kubectl get namespace
NAME                   STATUS   AGE
default                Active   10d
kube-node-lease        Active   10d
kube-public            Active   10d
kube-system            Active   10d
kubernetes-dashboard   Active   4d13h
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$ kubectl create namespace
error: exactly one NAME is required, got 0
See 'kubectl create namespace -h' for help and examples
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$ kubectl create namespace springboot-demo
namespace/springboot-demo created
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$
```

Step 9: Checking deployment in local

```
kubectl apply -f deployment.yaml -n springboot-demo
kubectl apply -f service.yaml -n springboot-demo
```

Write above command that are tailored for deployment with specific namespace.

```
kubectl get pods -n springboot-demo
```

once you done deployment after check then you will get output like,

Ready

0/1
0/1
0/1
0/1

Here 0/1 means 0 pods are running out of 1. and it will show 4 times due to replication for disaster recovery.

Then write below command,

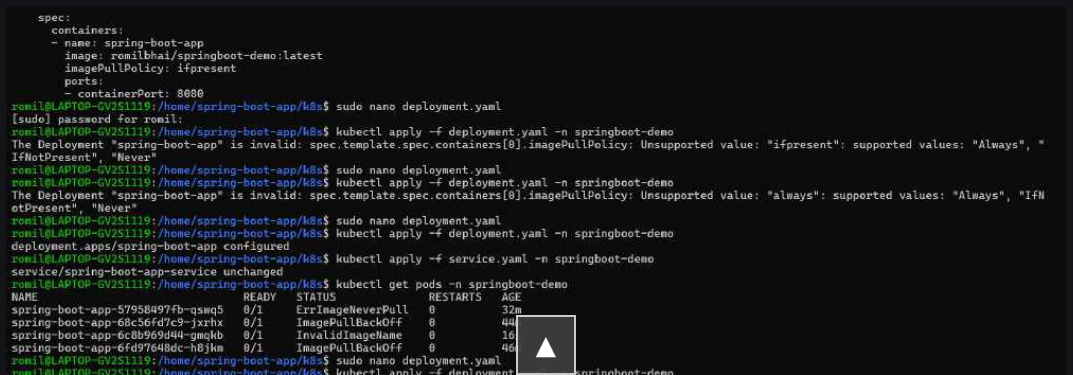
```
eval $(minikube docker-env)
```

And check again through below command

```
kubectl get pods -n springboot-demo
```

now you will get 1/1 ready status like,

Ready
1/1
1/1
1/1
1/1



```
deployment.apps/spring-boot-app configured
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$ kubectl get pods -n springboot-demo
NAME                              READY   STATUS             RESTARTS   AGE
spring-boot-app-57958497fb-qswq5  0/1     ErrImageNeverPull  0          33m
spring-boot-app-68c56fd7c9-jxrhx  0/1     ImagePullBackOff   0          45m
spring-boot-app-6c8b969d44-gmqkb  0/1     InvalidImageName   0          99s
spring-boot-app-6fd9695845-xtfv6  0/1     ContainerCreating  0          3s
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$ eval $(minikube docker-env)
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$ sudo nano deployment.yaml
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$ eval $(minikube docker-env)
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$ kubectl get pods -n springboot-demo
NAME                              READY   STATUS    RESTARTS   AGE
spring-boot-app-6fd9695845-xtfv6  1/1     Running   0          2m1s
spring-boot-app-6fd9695845-xvmd7  1/1     Running   0          72s
spring-boot-app-6fd9695845-zlwpg  1/1     Running   0          60s
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$ sudo nano deployment.yaml
romil@LAPTOP-GV2S1119:/home/spring-boot-app/k8s$
```

then push code in your remote repository or (GitHub repository).

```
* master
romil@LAPTOP-GV2S1119:/home/spring-boot-app$ sudo git -d checkout
unknown option: -d
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]
romil@LAPTOP-GV2S1119:/home/spring-boot-app$ sudo git branch -d checkout
Deleted branch checkout (was 5465f39).
romil@LAPTOP-GV2S1119:/home/spring-boot-app$ sudo git branch
* master
romil@LAPTOP-GV2S1119:/home/spring-boot-app$ git remote add origin https://github.com/RomilMovaliya/spring-boot-app.git
fatal: detected dubious ownership in repository at '/home/spring-boot-app'
To add an exception for this directory, call:

        git config --global --add safe.directory /home/spring-boot-app
romil@LAPTOP-GV2S1119:/home/spring-boot-app$ sudo git remote add origin https://github.com/RomilMovaliya/spring-boot-app.git
romil@LAPTOP-GV2S1119:/home/spring-boot-app$ git branch -M main
fatal: detected dubious ownership in repository at '/home/spring-boot-app'
To add an exception for this directory, call:

        git config --global --add safe.directory /home/spring-boot-app
romil@LAPTOP-GV2S1119:/home/spring-boot-app$ sudo git branch -M main
romil@LAPTOP-GV2S1119:/home/spring-boot-app$ sudo git push -u origin main
Username for 'https://github.com': RomilMovaliya
Password for 'https://RomilMovaliya@github.com':
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 8 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (19/19), 2.80 KiB | 1.40 MiB/s, done.
Total 19 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/RomilMovaliya/spring-boot-app.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
romil@LAPTOP-GV2S1119:/home/spring-boot-app$ sudo git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
romil@LAPTOP-GV2S1119:/home/spring-boot-app$ sudo git remote -v
origin  https://github.com/RomilMovaliya/spring-boot-app.git (fetch)
origin  https://github.com/RomilMovaliya/spring-boot-app.git (push)
romil@LAPTOP-GV2S1119:/home/spring-boot-app$
```

Step 10: Jenkins installation

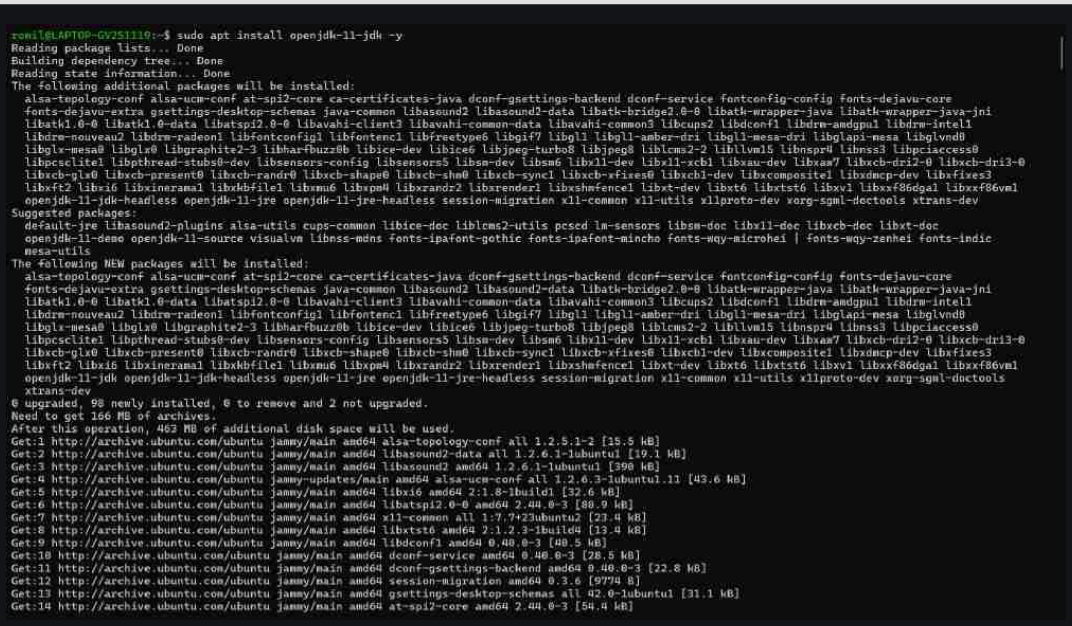
```
sudo apt update
sudo apt upgrade
```

update: It is basically used for refreshing list of available software packages and versions but it not download or install anything new by itself.

upgrade: It is installing newer version of software packages that are available on your system.

```
romil@LAPTOP-GV2S1119:~/nodejs-app$ cd ..
romil@LAPTOP-GV2S1119:~$ sudo apt-get update
[sudo] password for romil:
Hit:1 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:2 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Reading package lists... Done
romil@LAPTOP-GV2S1119:~$ sudo apt upgrade
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following NEW packages will be installed:
  ubuntu-pro-client
The following packages have been kept back:
  python3-update-manager update-manager-core
The following packages will be upgraded:
  apt apt-utils base-files bash bind9-dnsutils bind9-host bind9-libs binutils binutils-common binutils-x86-64-linux-gnu bsdextrautils bsdutils coreutils
  cpio curl distro-info distro-info-data dpkg eject git git-man iptables irqbalance less libapt-pkg6.0 libbinutils libblkid1 libc-bin libc6
  libcryptsetup12 libctf-nobfd0 libctf0 libcurl3-gnutls libcurl4 libexpat1 libglib2.0-0 libglib2.0-bin libglib2.0-data libgnutls30 libip4tc2 libip6tc2
  libldap-2.5-0 libldap-common libmount1 libnghttp2-14 libnss-systemd libpam-modules libpam-modules-bin libpam-runtime libpam-systemd libpam0g libperl5.34
  libpython3.10 libpython3.10-minimal libpython3.10-stdlib libsmartcols1 libsqlite3-0 libssh-4 libssl3 libsystemd0 libudev1 libuuid1 libuv1 libxml2
  libxtables12 locales login motd-news-config mount openssh-client openssl passwd perl perl-base perl-modules-5.34 python-apt-common python3-apt
  python3-cryptography python3-distro-info python3-distupgrade python3-software-properties python3.10 python3.10-minimal snapd software-properties-common
  systemd systemd-hwe-hwdb systemd-sysv systemd-timesyncd tar tcpdump tzdata ubuntu-advantage-tools ubuntu-pro-client-l10n ubuntu-release-upgrader-core
  udev util-linux uuid-runtime vim vim-common vim-runtime vim-tiny xxd
103 upgraded, 1 newly installed, 0 to remove and 2 not upgraded.
58 standard LTS security updates
Need to get 97.9 MB of archives.
After this operation, 1705 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 motd-news-config all 12ubuntu4.6 [4352 B]
Get:2 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libc6 amd64 2.35-0ubuntu3.8 [3235 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 base-files amd64 12ubuntu4.6 [62.5 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 bash amd64 5.1-6ubuntu1.1 [769 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 bsdutils amd64 1:2.37.2-4ubuntu3.4 [80.9 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 coreutils amd64 8.32-4.1ubuntu1.2 [1437 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libnss-systemd amd64 249.11-0ubuntu3.12 [133 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libsystemd0 amd64 249.11-0ubuntu3.12 [319 kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 systemd-timesyncd amd64 249.11-0ubuntu3.12 [31.2 kB]
Get:10 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 systemd-sysv amd64 249.11-0ubuntu3.12 [10.5 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libpam-systemd amd64 249.11-0ubuntu3.12 [203 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 systemd amd64 249.11-0ubuntu3.12 [4581 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 udev amd64 249.11-0ubuntu3.12 [1557 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libudev1 amd64 249.11-0ubuntu3.12 [78.2 kB]
```

```
sudo apt install openjdk-11-jdk -y
```



We need to install openjdk for Jenkins after that run below commands

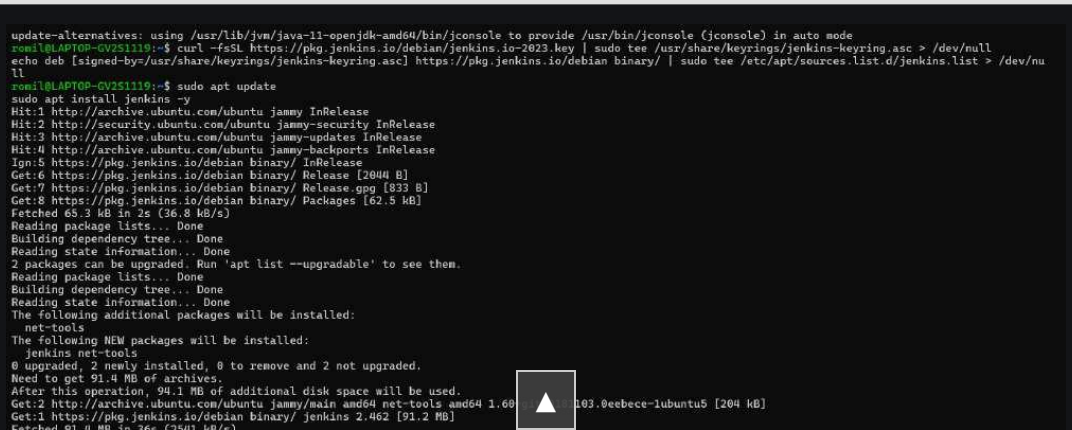
```
curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key
| sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/
null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]
https://pkg.jenkins.io/debian binary/
 | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null
```

finally run below commands for jenkins installation.

```
sudo apt update
sudo apt install jenkins -y
```

Here we use -y means automatically assign yes permission for automation instead of manually assigning.

```
sudo systemctl start jenkins
sudo systemctl enable jenkins
```



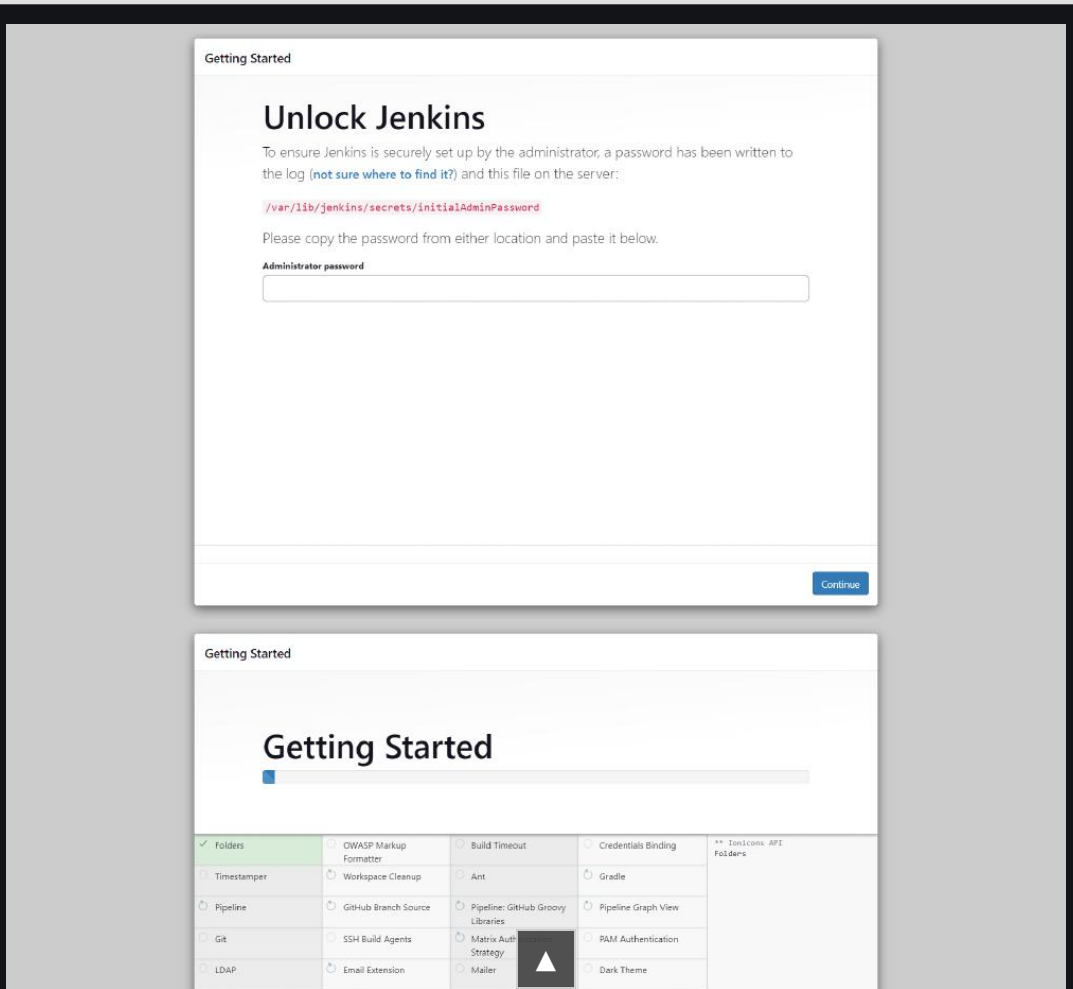
```
Selecting previously unselected package net-tools.
(Reading database ... 27426 files and directories currently installed.)
Preparing to unpack .../net-tools_1.60+git20181103.0eebece-1ubuntu5_amd64.deb ...
Unpacking net-tools (1.60+git20181103.0eebece-1ubuntu5) ...
Selecting previously unselected package jenkins.
Preparing to unpack .../archives/jenkins_2.462_all.deb ...
Unpacking jenkins (2.462) ...
Setting up net-tools (1.60+git20181103.0eebece-1ubuntu5) ...
Setting up jenkins (2.462) ...
Created symlink /etc/systemd/system/multi-user.target.wants/jenkins.service → /lib/systemd/system/jenkins.service.
Processing triggers for man-db (2.10.2-1) ...
romil@LAPTOP-GV2S1119:~$


The following additional packages will be installed:
  net-tools
The following NEW packages will be installed:
  jenkins net-tools
0 upgraded, 2 newly installed, 0 to remove and 2 not upgraded.
Need to get 91.4 MB of archives.
After this operation, 94.1 MB of additional disk space will be used.
Get:2 http://archive.ubuntu.com/ubuntu jammy/main amd64 net-tools amd64 1.60+git20181103.0eebece-1ubuntu5 [204 kB]
Get:1 https://pkg.jenkins.io/debian binary/ jenkins 2.462 [91.2 MB]
Fetched 91.4 MB in 36s (2541 kB/s)
Selecting previously unselected package net-tools.
(Reading database ... 27426 files and directories currently installed.)
Preparing to unpack .../net-tools_1.60+git20181103.0eebece-1ubuntu5_amd64.deb ...
Unpacking net-tools (1.60+git20181103.0eebece-1ubuntu5) ...
Selecting previously unselected package jenkins.
Preparing to unpack .../archives/jenkins_2.462_all.deb ...
Unpacking jenkins (2.462) ...
Setting up net-tools (1.60+git20181103.0eebece-1ubuntu5) ...
Setting up jenkins (2.462) ...
Created symlink /etc/systemd/system/multi-user.target.wants/jenkins.service → /lib/systemd/system/jenkins.service.
Processing triggers for man-db (2.10.2-1) ...
romil@LAPTOP-GV2S1119:~$ sudo systemctl start jenkins
romil@LAPTOP-GV2S1119:~$ sudo systemctl enable jenkins
Synchronizing state of jenkins.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable jenkins
romil@LAPTOP-GV2S1119:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2024-06-15 18:51:43 IST; 3min 21s ago
   Main PID: 9542 (java)
      Tasks: 51 (limit: 4564)
     Memory: 709.1M
     CGroup: /system.slice/jenkins.service
             └─9542 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Jun 15 18:51:37 LAPTOP-GV2S1119 jenkins[9542]: c68bfef68f1e49b39415afde77b15ed5
Jun 15 18:51:37 LAPTOP-GV2S1119 jenkins[9542]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Jun 15 18:51:37 LAPTOP-GV2S1119 jenkins[9542]: ************************************************************
Jun 15 18:51:37 LAPTOP-GV2S1119 jenkins[9542]: ************************************************************
Jun 15 18:51:37 LAPTOP-GV2S1119 jenkins[9542]: ************************************************************
Jun 15 18:51:43 LAPTOP-GV2S1119 jenkins[9542]: 2024-06-15 13:21:43.035+0000 [id=31]        INFO        jenkins.InitReactorRunner$1#onAttained: Completed in
Jun 15 18:51:43 LAPTOP-GV2S1119 jenkins[9542]: 2024-06-15 13:21:43.069+0000 [id=24]        INFO        hudson.lifecycle.Lifecycle#onReady: Jenkins is fully
Jun 15 18:51:43 LAPTOP-GV2S1119 systemd[1]: Started Jenkins Continuous Integration Server.
Jun 15 18:51:43 LAPTOP-GV2S1119 jenkins[9542]: 2024-06-15 13:21:43.970+0000 [id=61]        INFO        h.m.DownloadService$Downloadable#load: Obtained the
Jun 15 18:51:43 LAPTOP-GV2S1119 jenkins[9542]: 2024-06-15 13:21:43.971+0000 [id=61]        INFO        hudson.util.Retrier#start: Performed the action chec
lines 1-19/19 (END)
```

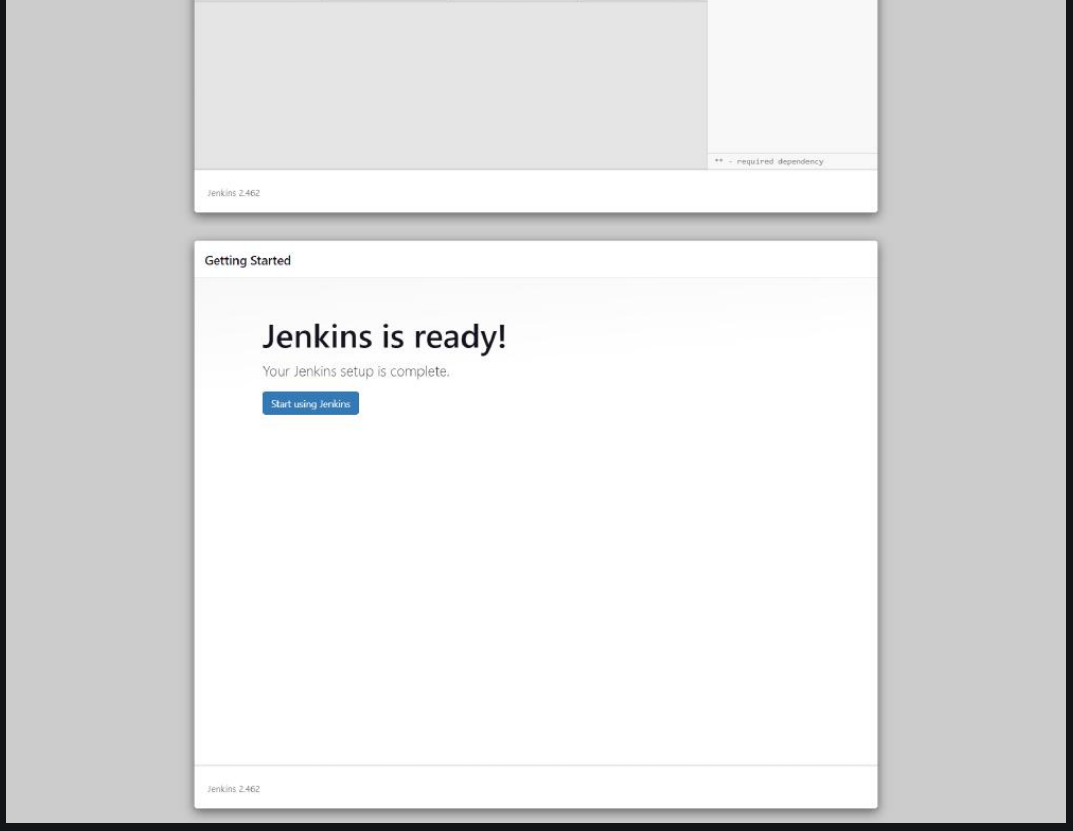
once you write above command then open the browser and write,

https://localhost:8080

Now Jenkins UI(First page) will display but it require the initialAdminPassword so that reason you need to write below command that will show the password.

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

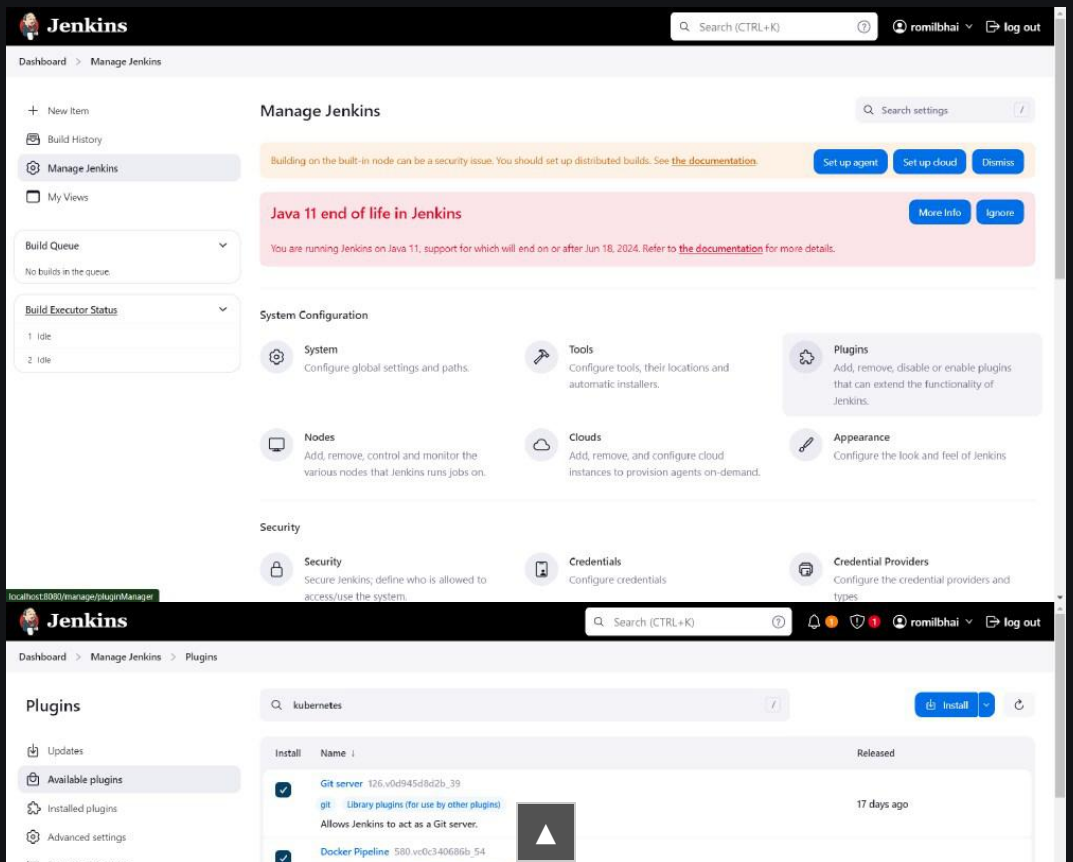


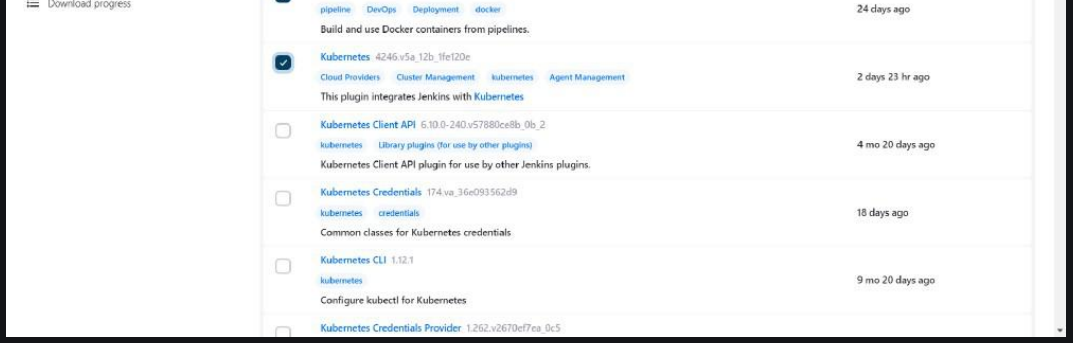


Step 11: Configuring Jenkins

Once you done personal login or as a guest login, you will see the Jenkins Dashboard In that We need to install some necessary plugins: Go to Manage Jenkins > Manage Plugins. And installing following plugins:

- Docker Pipeline
- Kubernetes
- Git





Step 12: Create a New Pipeline Job

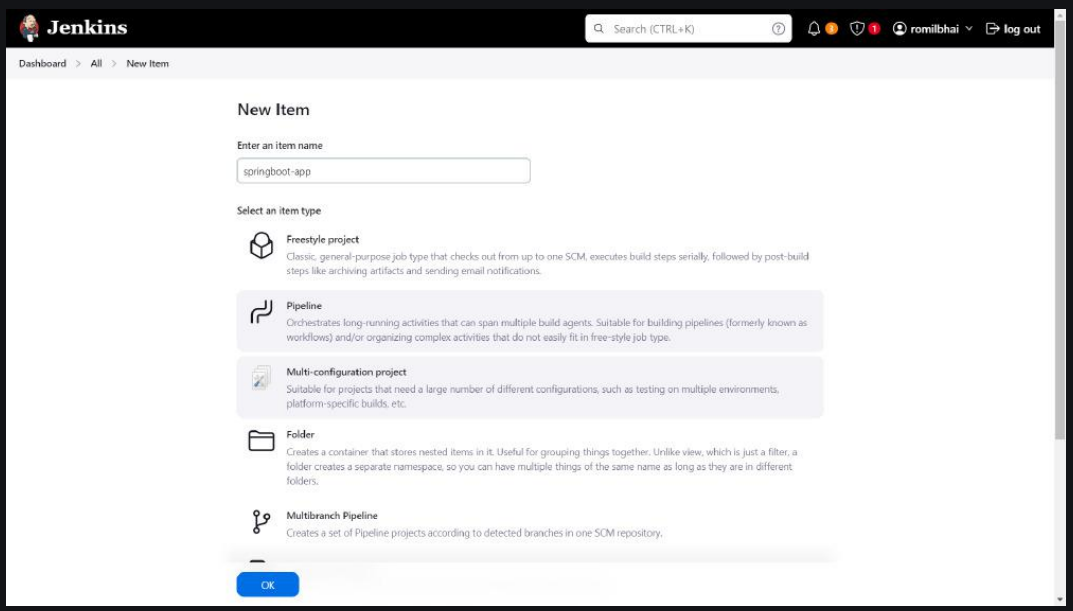
Create New Item:

Create a new jobs or new item from left-hand side menu on the Jenkins Dashboard.

Enter Job Details:

Name: Enter a name for your new job (e.g., springboot-app-Pipeline).

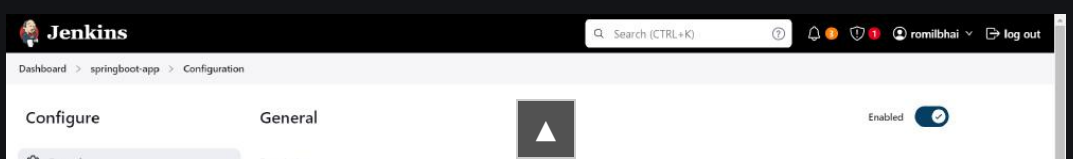
- Type: Select Pipeline.
- Click OK to proceed.

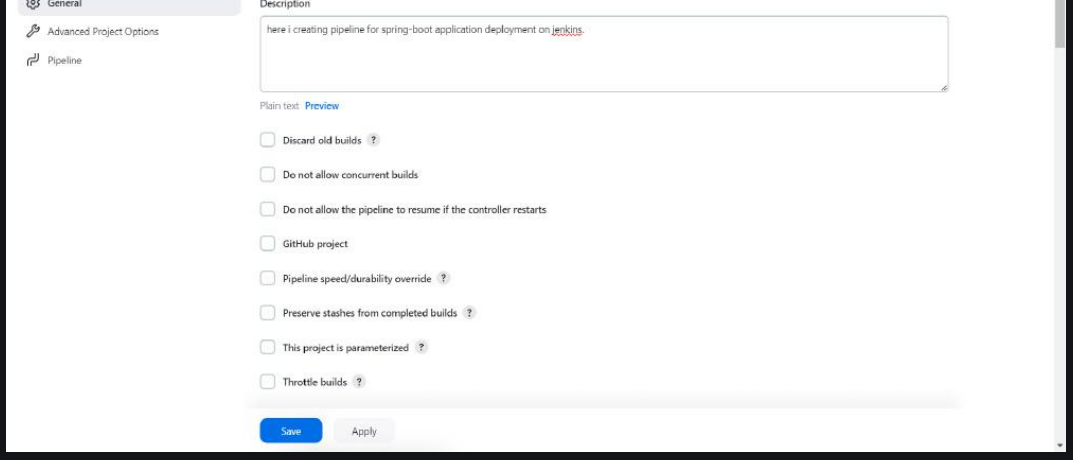


Step 13: Configure the Job

General Configuration:

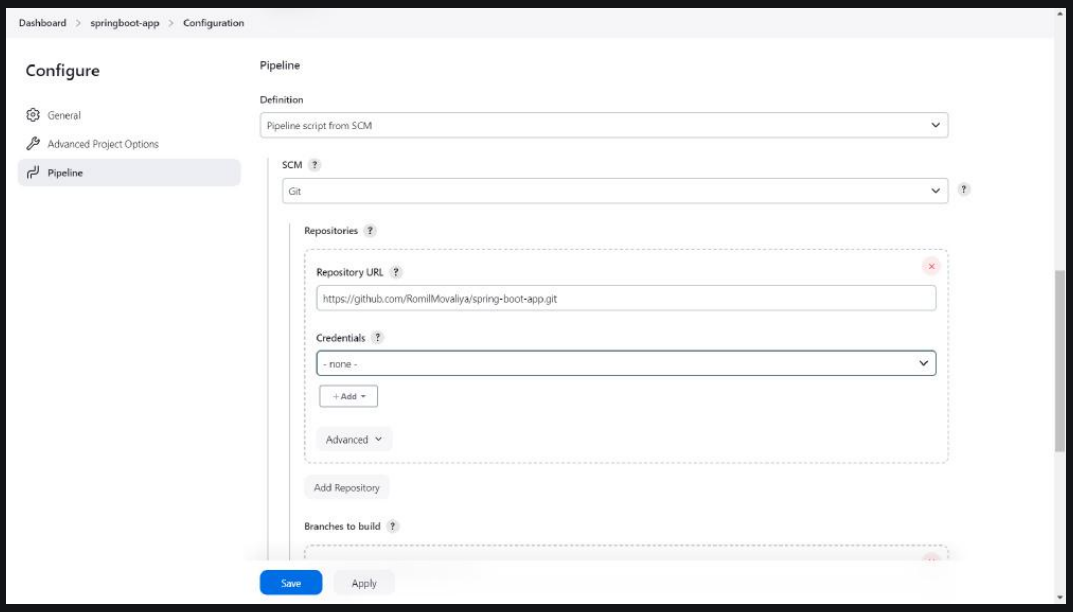
Description: Optionally, provide a description for the job to explain what this pipeline does.





Pipeline Configuration:

- Definition: Choose Pipeline script from SCM.
- SCM: Select Git.
- Repository URL: Enter your Git repository URL (e.g., https://github.com/your-username/your-repo.git).

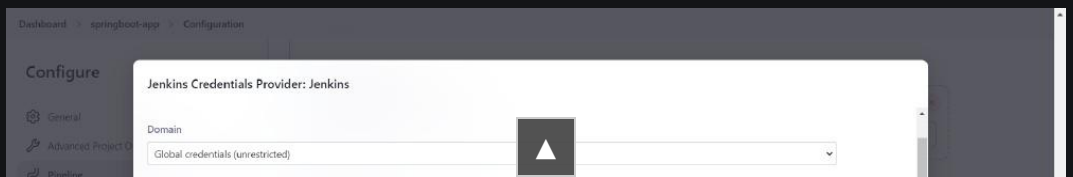


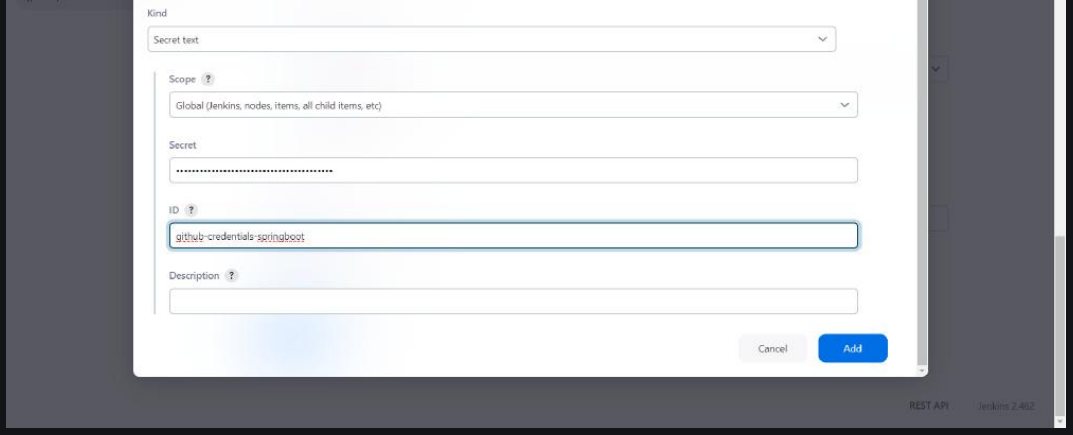
Credentials:

If your repository requires authentication, you need to add credentials.

1. **Credentials**: Click Add to add your credentials (e.g., GitHub username and password, or personal access token).
2. **Kind**: Choose the appropriate kind (e.g., Username with password or SSH Username with private key).
3. **Scope**: Set to Global.
4. **Username**: Enter your GitHub username.
5. **Password**: Enter your GitHub password or personal access token.

Click Add to save.





Branch Specifier:

By default, this is set to ***/master**. If you are using a different branch, update it accordingly (e.g., ***/main**).

Script Path:

Specify the path to your **Jenkinsfile** if it's not in the root directory of your repository. By default, it is set to **Jenkinsfile**.

Adding Pipeline syntax:

click on pipeline syntax and setup the **kubeconfig**.

you need to add credentials for **kubeconfige**,

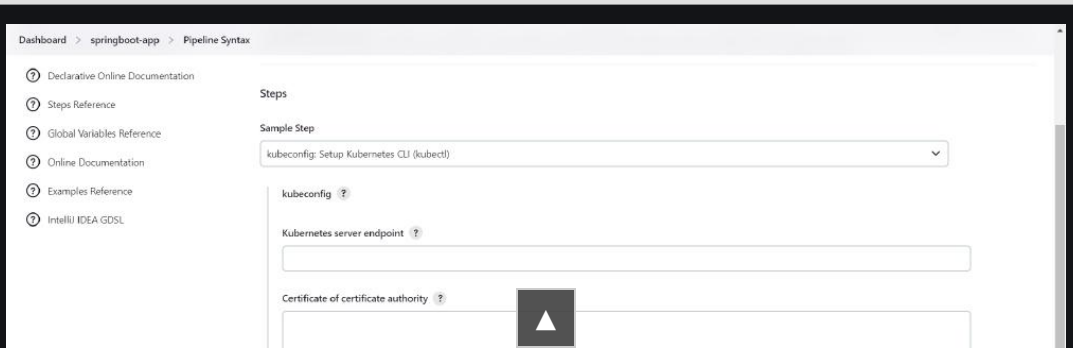
1. Credentials: Click Add to add your credentials.
2. Kind: Choose the appropriate kind (e.g., Secret text).
3. Scope: Set to Global.
4. Secret: Here you need to add the kubeconfig data which is present at .kube/config file in your local system.

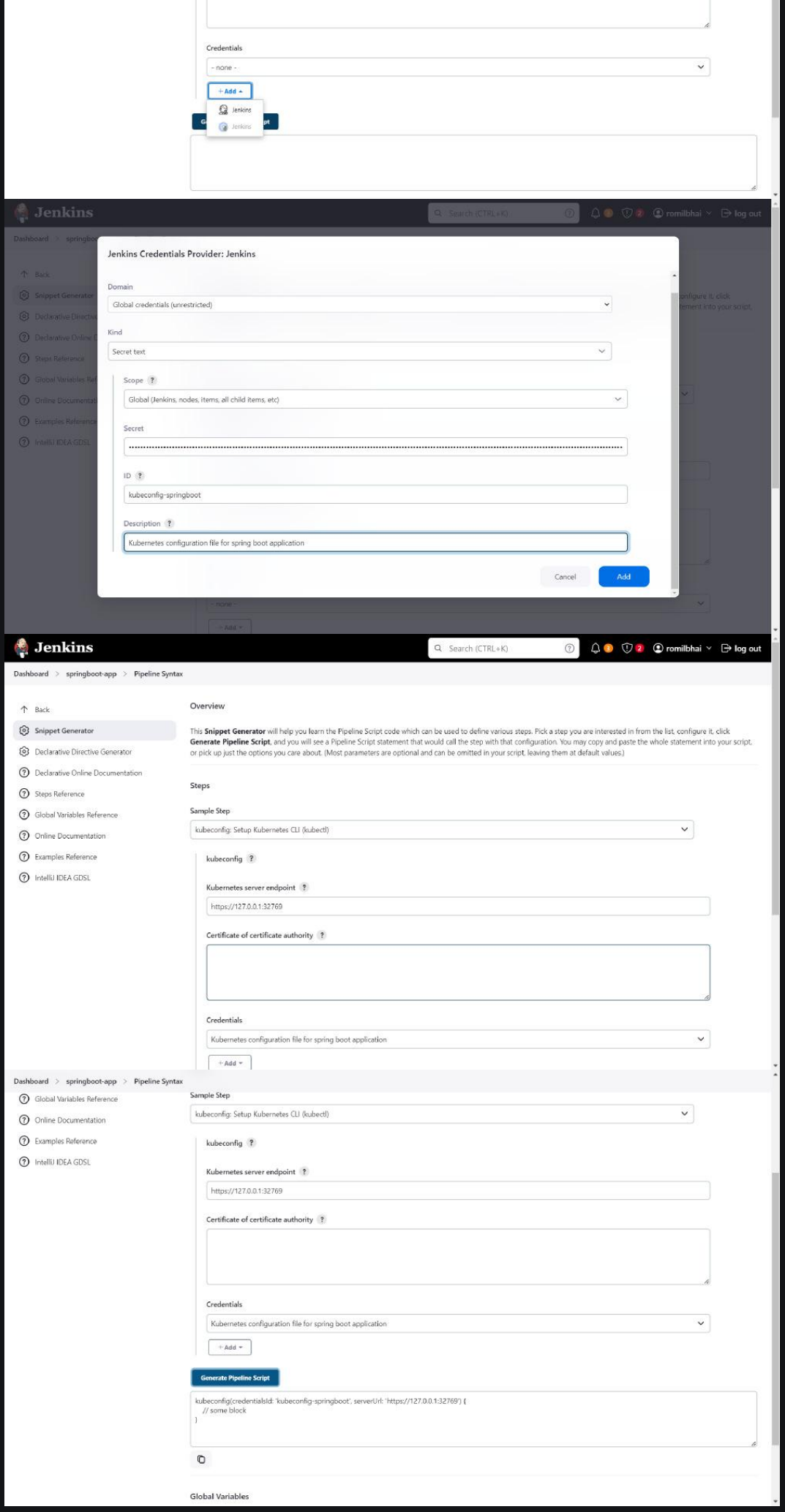
if you want to check your config file's data then write below command,

```
ls -la
```

After that you can see many files from that you move to config file using below command,

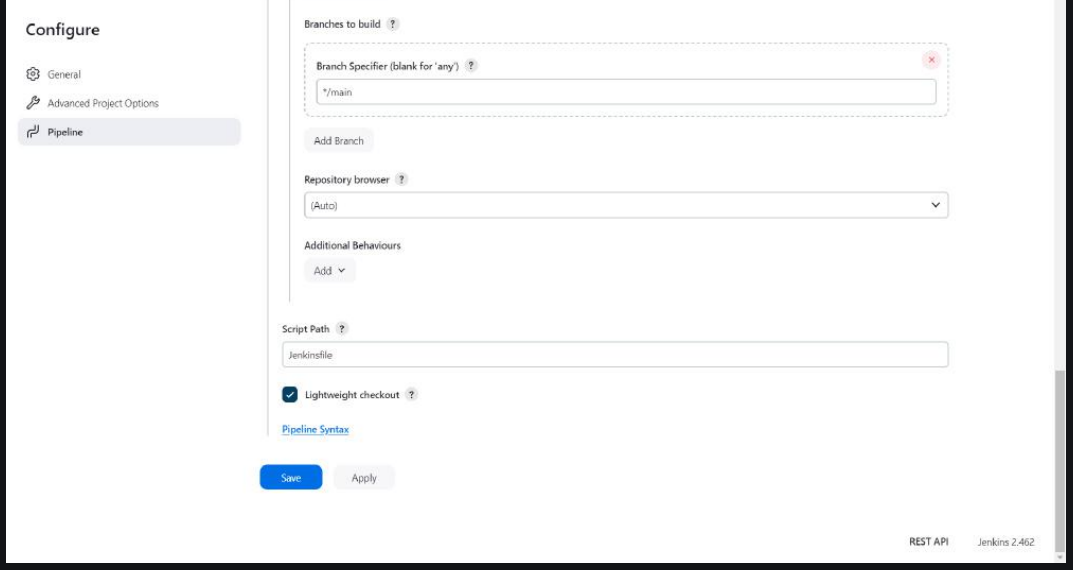
```
cd .kube/config
```



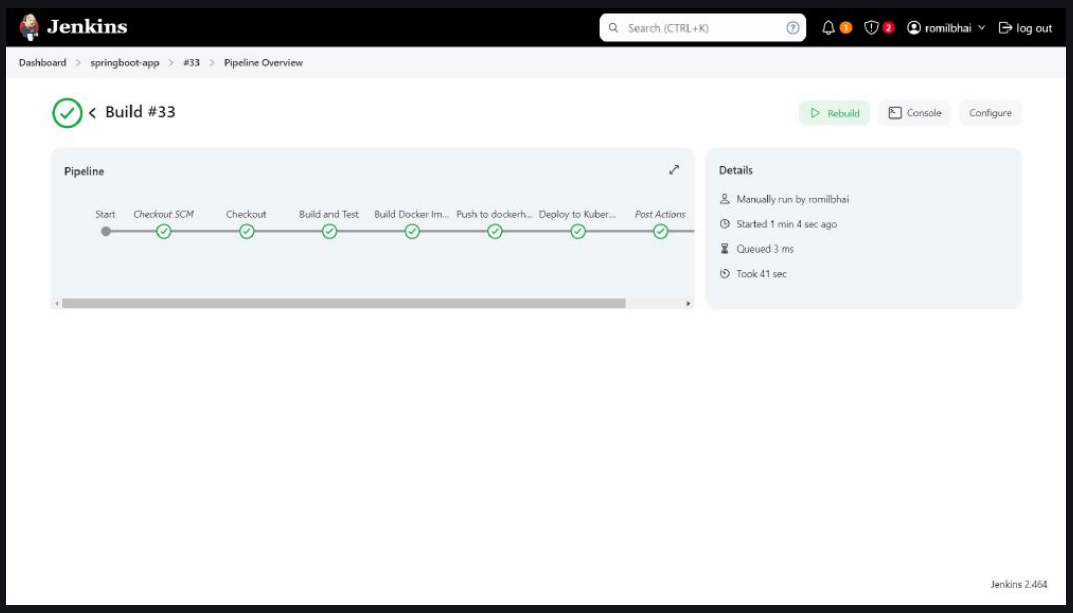


Save Configuration:

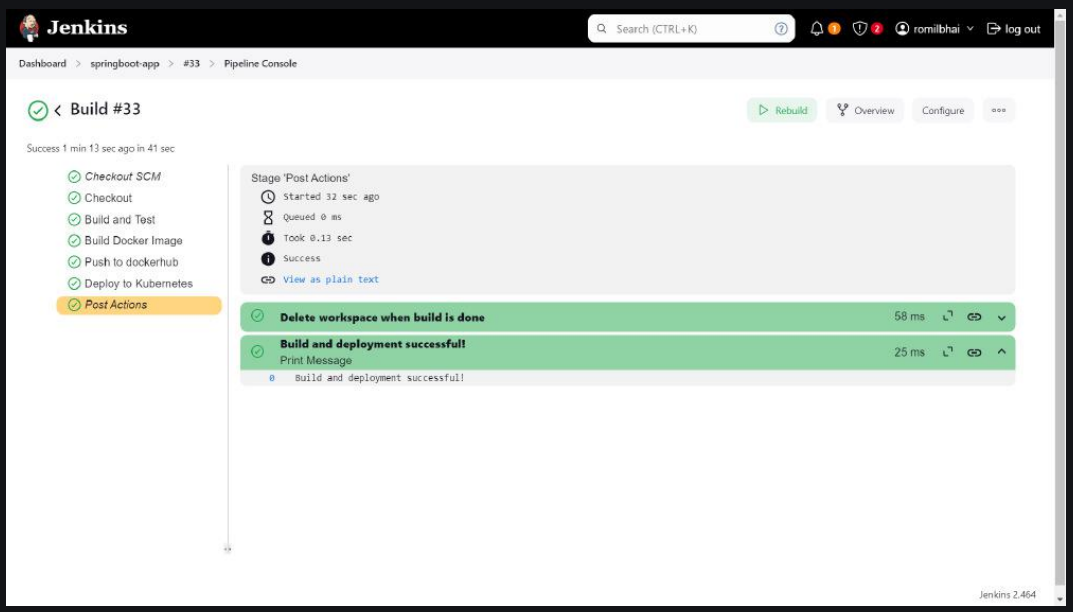
Scroll down and click Save to save the job configuration.



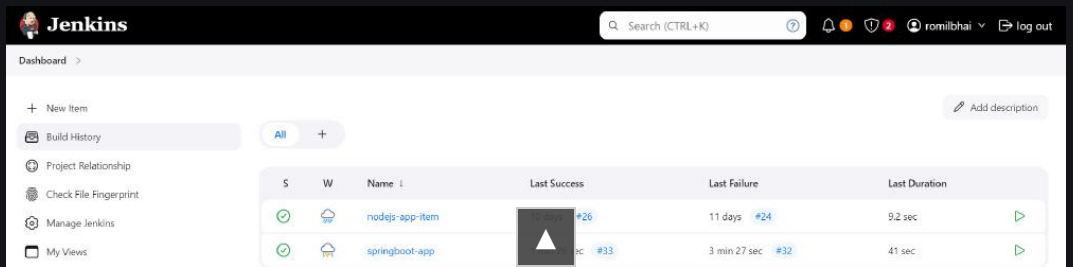
You can check pipeline overview in your Jenkins Dashboard.

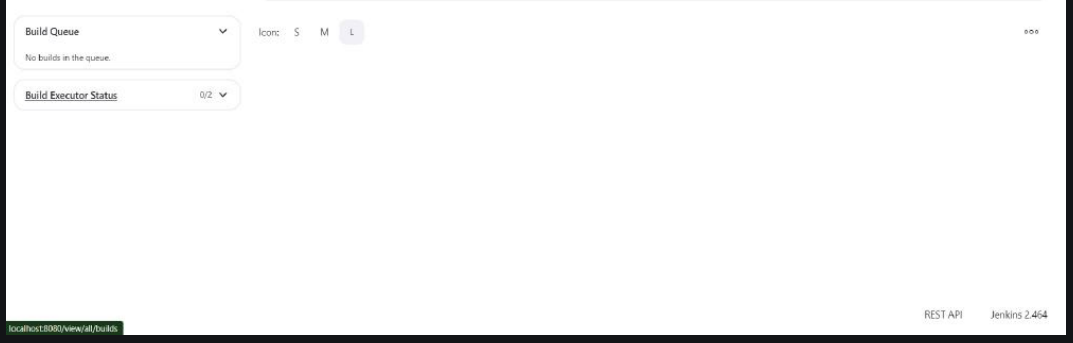


when you click on the console then you will get below result.



Then you can see it in your main dashboard.





Conclusion

Deploying Spring Boot applications on Kubernetes using Jenkins CI/CD ensures fast and efficient deployment. By using Docker for containerization and Kubernetes for orchestration, we achieve easy scalability and robust management of our applications. Jenkins builds, tests, and automates deployments, increasing productivity and reliability. GitHub simplifies version control and collaborative development, ensuring code integrity and traceability. This comprehensive approach accelerates development cycles, ensures implementation stability, and enables teams to consistently deliver high-quality software by effectively meeting dynamic market requirements.

Spring boot application deployment in Kubernetes with Jenkins CI/CD pipeline - FAQs

What are the main difference between Spring boot and other framework?

Here main 2 consideration that are given below

1. language preference: *If you are comfortable or familiar with java then spring boot is right choice.*

2. Project requirements: *Spring boot is vary brilliant for micro service and backend development but If other framework may serve broader web development needs then you select as per your project requirement.*

What is the main role of Autoconfiguration in spring boot?

Autoconfiguration is a powerful features that automates the configuration of your application that based on library included in your project.
Imagine, that you want to connect MYSQL database with your application. Let's see, How it simplifies the process.

1. Add dependencies: *Add the spring-boot-starter-data-jpa dependencies to your project's build configuration (e.g., pom.xml). This dependency provides libraries to integrate with the relational database.*

2. Autoconfiguration tasks: *Spring Boot checks for spring-boot-starter-data-jpa dependencies. It then triggers the appropriate autoconfiguration class, like DataSourceAutoConfiguration.*

3. Automatic Configuration: *If you do not explicitly configure the DataSource bean yourself, the DataSourceAutoConfiguration class will create one based on the default settings or properties you provide in your application.yml or application.properties file*

4. Ready to use: *You can now interact with a MySQL database using Spring Data JPA features without writing any manual configuration code for the data source itself.*

What is pods and nodes in the Kubernetes?

The nodes are the are physical or virtual machine where pods are the smallest unit of Kubernetes.

Each pods has own IP address and only one application is run per pods.

pods are ephemerals means pods can be die easily.

for example, Our DB pod is die due to docker crashes or application crashes or resource at that new pod will take place with new IP address. To overcome this issues Kubernetes has one features called services.

services has fixed IP address that can't be change.

Why .kube file is not sometimes showing?

By default, .kube file resides our home directory and your are finding it in another directory.

```
ls -a ~/.kube
```

*This command is basically used for reveal the hidden files and folders including .**kube** folder.*

- *Here **"ls"** command is used for showing list of content of the directory.*
- ***"-a"** command is used for show the hidden files and folder.*
- ***"~/"** command is used for expand your home directory path.*

How can a basic Spring Boot application be created?

Write the necessary Java classes (e.g. HelloController.java, DemoApplication.java), create directories and files for your application, and put in place the configuration settings of your app.

Are you ready to unleash the power of **DevOps** to streamline your **Software Development and Deployment**? Learn about our DevOps Live Course at GeeksforGeeks, created for all professionals in practice with continuous integration, delivery, and deployment. Learn about leading tools, industry best practices, and techniques for **automation** through an interactive session with hands-on **live projects**. Whether you are new to DevOps or looking to improve your skills, this course equips you with everything needed to streamline workflows and deliver excellent quality software in the least amount of time. Learn to take your skills in DevOps to the next level now, and harness the power of streamlined software development!

R romil... + Follow

Previous Article
How to Deploy Java Application in Kubernetes?

Next Article

Similar Reads

Java Application Deployment In Kubernetes With Jenkins CI/CD Pipeline

In modern software development, deploying applications to Kubernetes clusters has become a common practice due to their scalability and reliability....

5 min read

Python Application Deployment In Kubernetes With Jenkins CI/CD Pipeline

In modern software development, deploying applications to Kubernetes clusters has become a common practice due to their scalability and reliability....

6 min read

Node.Js Application Deployment in Kubernetes with Jenkins CI/CD Pipeline

As we know in the modern world we use Continuous Integration and Continuous Deployment for fast and and efficient delivery of our application to...

9 min read

Deployment Of Spring Boot Application In Jenkins

In this article, we will be exploring how to automate the process of building Spring Boot applications by employing a polling trigger. We will also learn ho...

5 min read

Build, Test and Deploy a Flask REST API Application from GitHub using...

Nowadays even for small web applications or microservices, we need an easier and faster way to deploy applications that is reliable and safe. These...

4 min read

Kubernetes - Creating Deployment and Services using Helm in Kubernetes

Prerequisite: Kubernetes Helm is used for managing your Kubernetes Deployment. With helm, we can tear down and create a deployment with a...

4 min read

How to Deploy Spring Boot Application in Kubernetes ?

The Spring Boot framework provides an assortment of pre-configured templates and tools to make the development of Java-based applications simpler. With...

7 min read

Understanding Jenkins CI/CD Pipeline And Its Stages

Jenkins is an open-source automation server that enables developers to reliably build, test, and deploy applications. It supports continuous integration and...

15 min read

How To Create Methods In Jenkins Declarative Pipeline?

Jenkins is an open-source automation tool used to automate the fetch, build, test, and deploy stages of a software application. Here in this guide, we will fir...

4 min read

Setting Up A CI/CD Pipeline For PHP Applications Using Jenkins

In contemporary software development, Continuous Integration and Continuous Deployment (CI/CD) pipelines have arisen as key devices for smoothing out the...

6 min read

Article Tags : DevOps | Kubernetes | DevOps Projects

Corporate & Communications Address:- A-143, 9th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305) | Registered Address:- K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

Digital Logic
Design
Engineering Maths
Software
Development
Software Testing

Aptitude
Preparation
Puzzles