Arpit Singh

19BCG10069

Kaiburr – Task 5

Technical Task

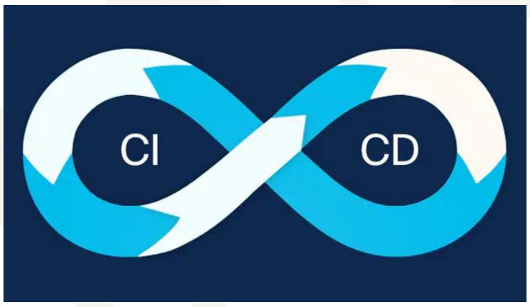
(Placement)

GitHub Link: https://github.com/TSM-ArpitSG/Kaiburr/tree/main/Kaibur_Tasks/Task_5

Task 5:

CICD Pipeline:

Create a CI-CD pipeline for a sample application using any CI-CD tool of your choice like ***Jenkins***, Azure DevOps, Gitlab, Github Actions, AWS CodePipeline or any other tool of your choice. Include a ***code build and a docker build*** step in your pipeline.



- The reason for using the CICD pipeline is ***very significant and useful*** in today's world. It basically provides a way to ***automate the process of integration and development***. ***Continuous integration and development*** that basically means if you make any changes to your code base like let's say you change the title of your website so instead of manually updating your repository and deploying that change to reflect in the actual product you need to again and again integrate and deploy it. Therefore, a better approach is to automate the process by creating a ***CICD pipeline*** which follows certain steps to integrate and deploy your application regularly.

- In our case we will be using ***jenkins*** to create the pipeline allowing continuous integration and deployment of our application.

Steps:

1. **Jenkins**:

- To ***create a CICD pipeline for your application***, you need to install ***jenkins that specify the script configuration***.

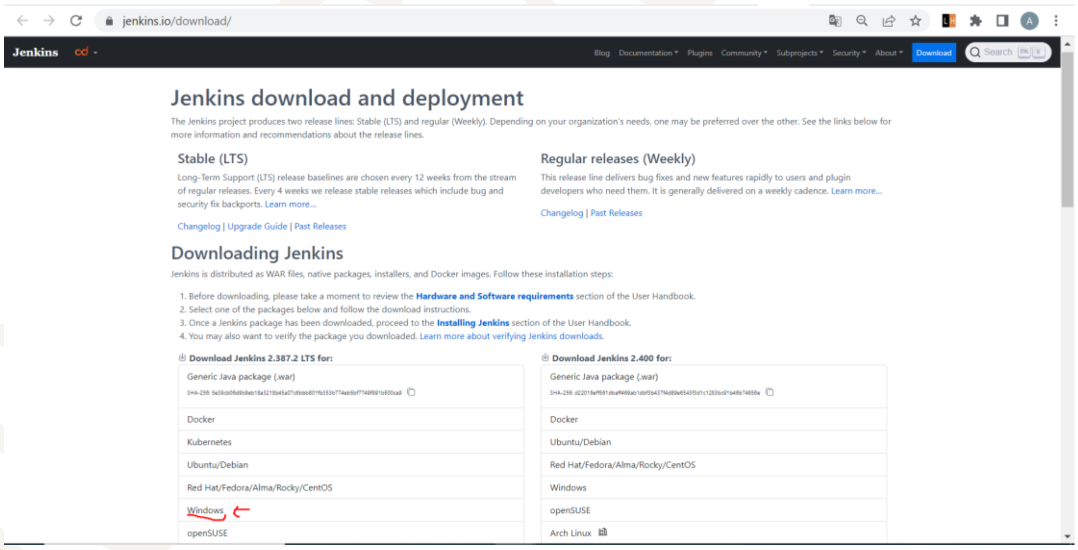
- Download and install Jenkins on your machine or server by following the official installation guide for your operating system.

- Once installed, open Jenkins in your browser and complete the initial setup by following the on-screen instructions.

   a. ***Download/Install jenkins***:

      i. Download the latest version of Jenkins from the official website: ***https://www.jenkins.io/download/***

      ii. Choose the ***Windows installer*** option and download the .exe file.



jenkins.io/download/

Jenkins download and deployment

The Jenkins project produces two release lines: Stable (LTS) and regular (Weekly). Depending on your organization's needs, one may be preferred over the other. See the links below for more information and recommendations about the release lines.

Stable (LTS)

Long-Term Support (LTS) release baselines are chosen every 12 weeks from the stream of regular releases. Every 4 weeks we release stable releases which include bug and security fix backports. Learn more...

Changelog | Upgrade Guide | Past Releases

Regular releases (Weekly)

This release line delivers bug fixes and new features rapidly to users and plugin developers who need them. It is generally delivered on a weekly cadence. Learn more...

Changelog | Past Releases

Downloading Jenkins

Jenkins is distributed as WAR files, native packages, installers, and Docker images. Follow these installation steps:

1. Before downloading, please take a moment to review the **Hardware and Software requirements** section of the User Handbook.
2. Select one of the packages below and follow the download instructions.
3. Once a Jenkins package has been downloaded, proceed to the **Installing Jenkins** section of the User Handbook.
4. You may also want to verify the package you downloaded. Learn more about verifying Jenkins downloads.

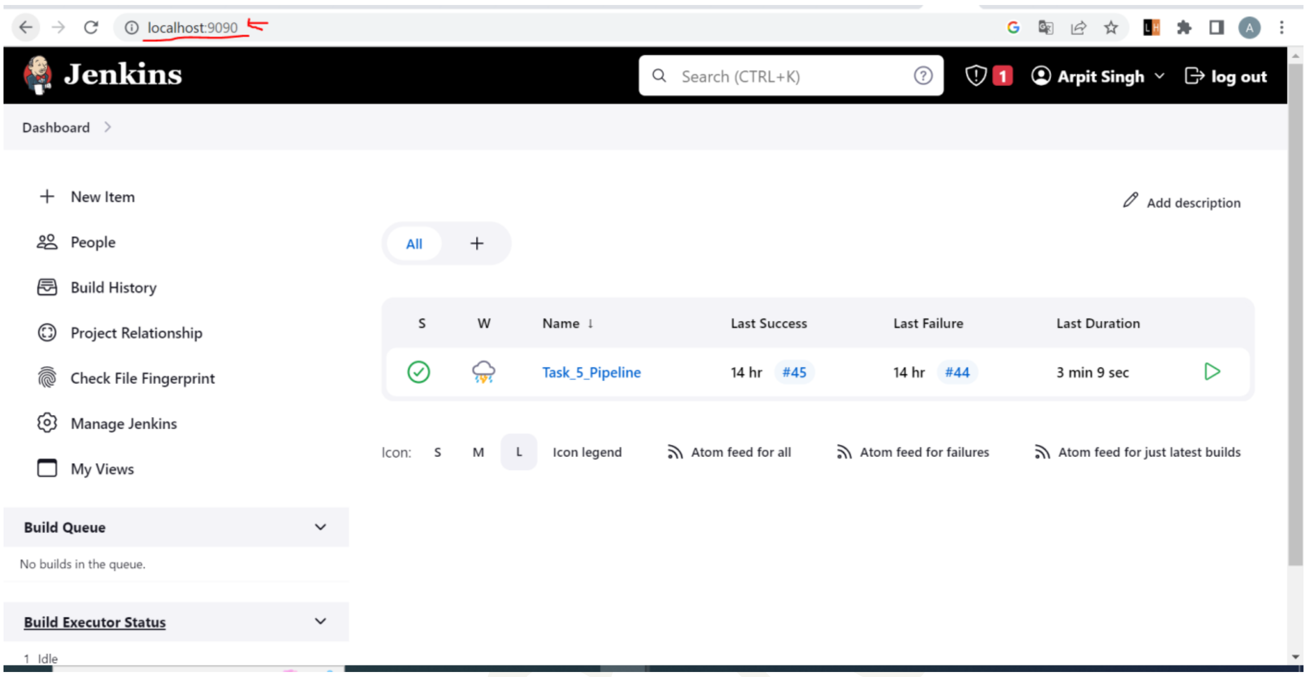
Download Jenkins 2.387.2 LTS for:	**Download Jenkins 2.400 for:**
Generic Java package (.war) SHA-256: [illegible]	Generic Java package (.war) SHA-256: [illegible]
Docker	Docker
Kubernetes	Ubuntu/Debian
Ubuntu/Debian	Red Hat/Fedora/Alma/Rocky/CentOS
Red Hat/Fedora/Alma/Rocky/CentOS	Windows
Windows ←	openSUSE
openSUSE	Arch Linux

      iii. Once the download is complete, double-click the ***.exe file*** to start the installation process.

      iv. Follow the ***on-screen instructions*** to complete the installation.

      v. During the installation, you will be prompted to choose the installation directory for Jenkins. You can accept the ***default directory*** or choose a custom location.(***C:\Program Files\Jenkins***)

vi. After the installation is complete, Jenkins will ***start automatically***. You can access Jenkins by opening your web browser and entering the ***URL: http://localhost:8080/ (In my case I changed the port to 9090, http://localhost:9090/)***



localhost:9090

Jenkins Search (CTRL+K) Arpit Singh log out

Dashboard >

New Item · People · Build History · Project Relationship · Check File Fingerprint · Manage Jenkins · My Views

Add description

All +

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓		Task_5_Pipeline	14 hr #45	14 hr #44	3 min 9 sec

Icon: S M L · Icon legend · Atom feed for all · Atom feed for failures · Atom feed for just latest builds

Build Queue

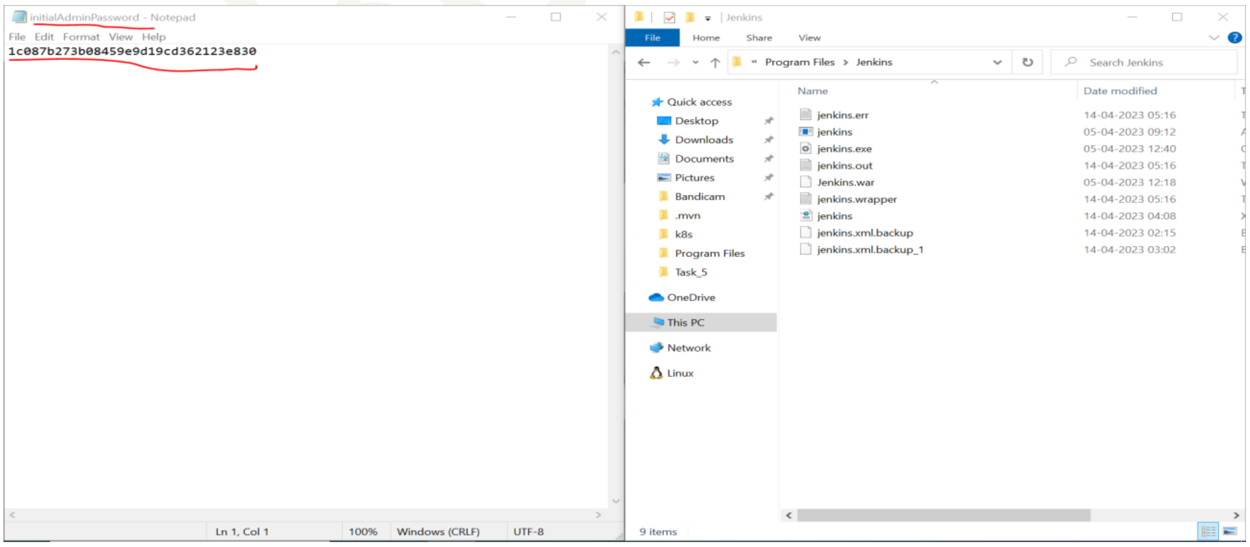
No builds in the queue.

Build Executor Status

1 Idle

vii. When you access Jenkins for the first time, you will be prompted to enter an ***administrator password***. The password can be found in the installation directory, in a file named "***initialAdminPassword***".

viii. Follow the on-screen instructions to ***complete the setup process*** and create your ***Jenkins administrator account***.



initialAdminPassword - Notepad

```
1c087b273b08459e9d19cd362123e830
```

Ln 1, Col 1 · 100% · Windows (CRLF) · UTF-8

Jenkins — Program Files > Jenkins

Name	Date modified
jenkins.err	14-04-2023 05:16
jenkins	05-04-2023 09:12
jenkins.exe	05-04-2023 12:40
jenkins.out	14-04-2023 05:16
Jenkins.war	05-04-2023 12:18
jenkins.wrapper	14-04-2023 05:16
jenkins	14-04-2023 04:08
jenkins.xml.backup	14-04-2023 02:15
jenkins.xml.backup_1	14-04-2023 03:02

9 items

Jenkins

ix. Once the setup is complete, you can start creating your CI/CD pipeline in Jenkins.

b. ***Docker and Docker hub installation has already been done in task 3***:

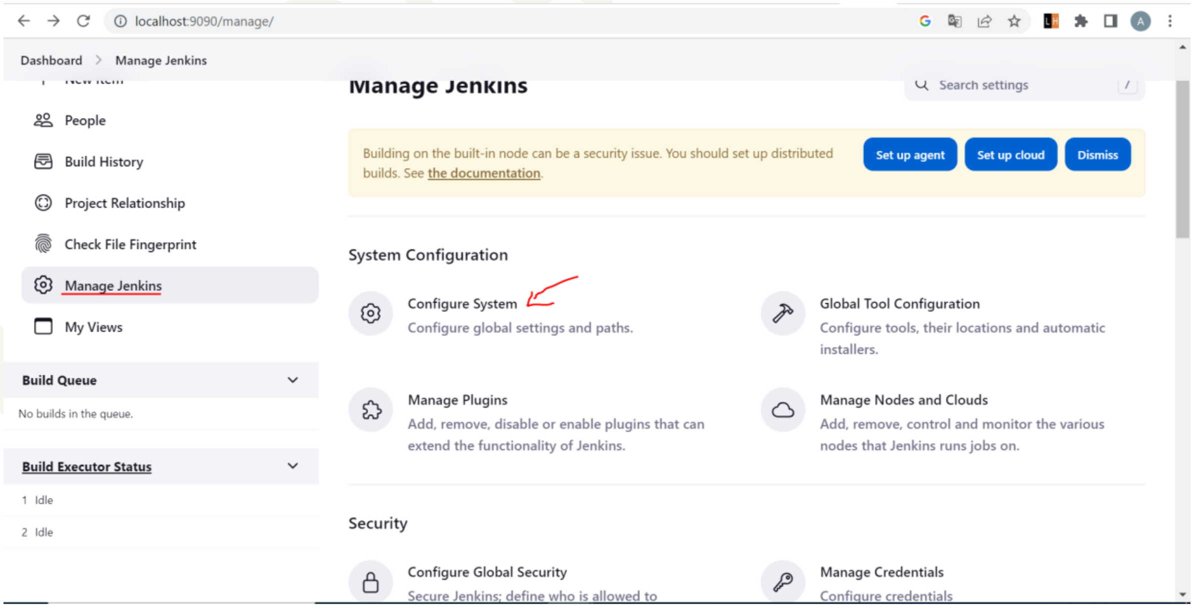
   i. ***https://github.com/TSM-ArpitSG/Kaiburr/tree/main/Kaibur_Tasks/Task_3***

2. **Setting up jenkins to work with our maven sample project**:

For ***Jenkins to work*** with our sample maven project it should have certain ***dependencies and plugins*** like ***JDK, maven, Docker, Pipeline***.

c. ***Setting Environment var 'JDK (JAVA)':***

   i. Open ***Jenkins*** in your web browser and go to the "***dashboard***".
   ii. Click on "***Manage Jenkins***" from the left-hand menu.
   iii. Click on "***Configure System***" from the list of options.



localhost:9090/manage/

Dashboard > Manage Jenkins

Manage Jenkins

Building on the built-in node can be a security issue. You should set up distributed builds. See the documentation.

Set up agent | Set up cloud | Dismiss

System Configuration

Configure System — Configure global settings and paths.

Global Tool Configuration — Configure tools, their locations and automatic installers.

Manage Plugins — Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

Manage Nodes and Clouds — Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Security

Configure Global Security — Secure Jenkins; define who is allowed to

Manage Credentials — Configure credentials

Build Queue

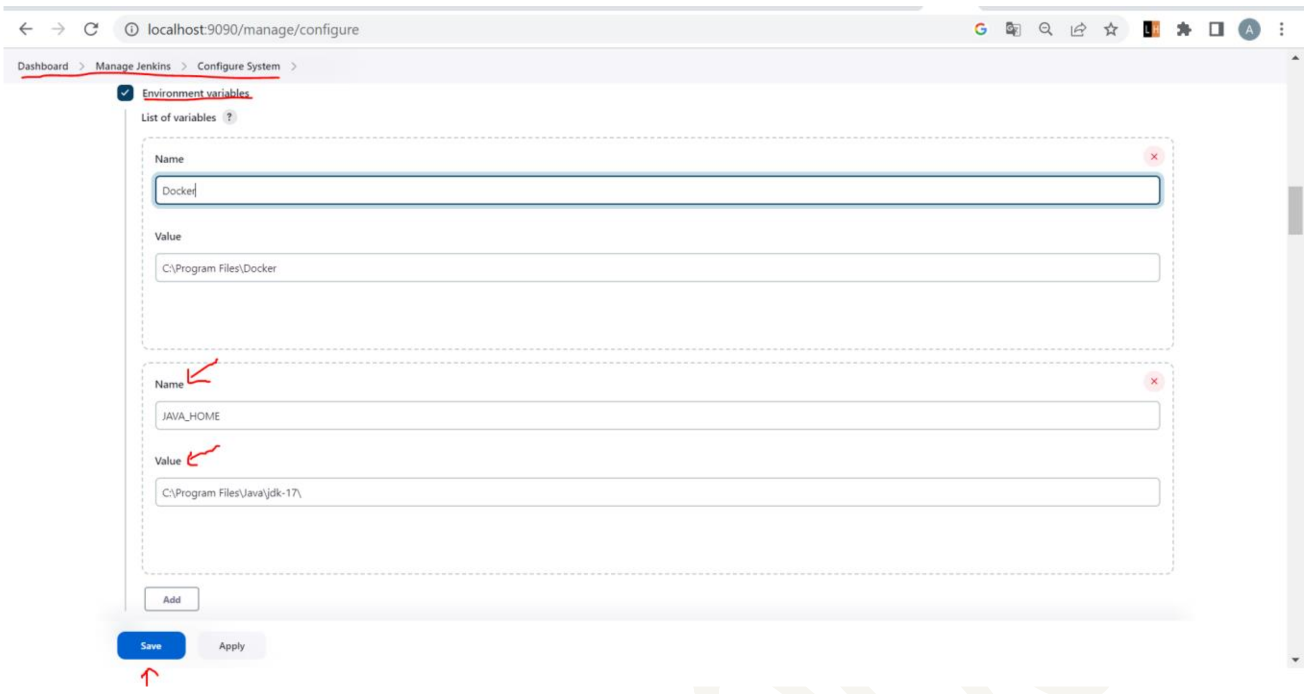
No builds in the queue.

Build Executor Status

1 Idle

2 Idle

   iv. Scroll down to "***Global Properties***" and under that check "***Environments Variables***".
   v. Click on "***Add***".
   vi. Put '***Name***' as "***JAVA_Home***" and '***Value***' as the location to your "***Java directory***". "***C:\Program Files\Java\jdk-17\***" .
   vii. Click "***Save***".


localhost:9090/manage/configure
Dashboard > Manage Jenkins > Configure System
Environment variables
List of variables
Add
Save
Apply

Name: Docker
Value: C:\Program Files\Docker

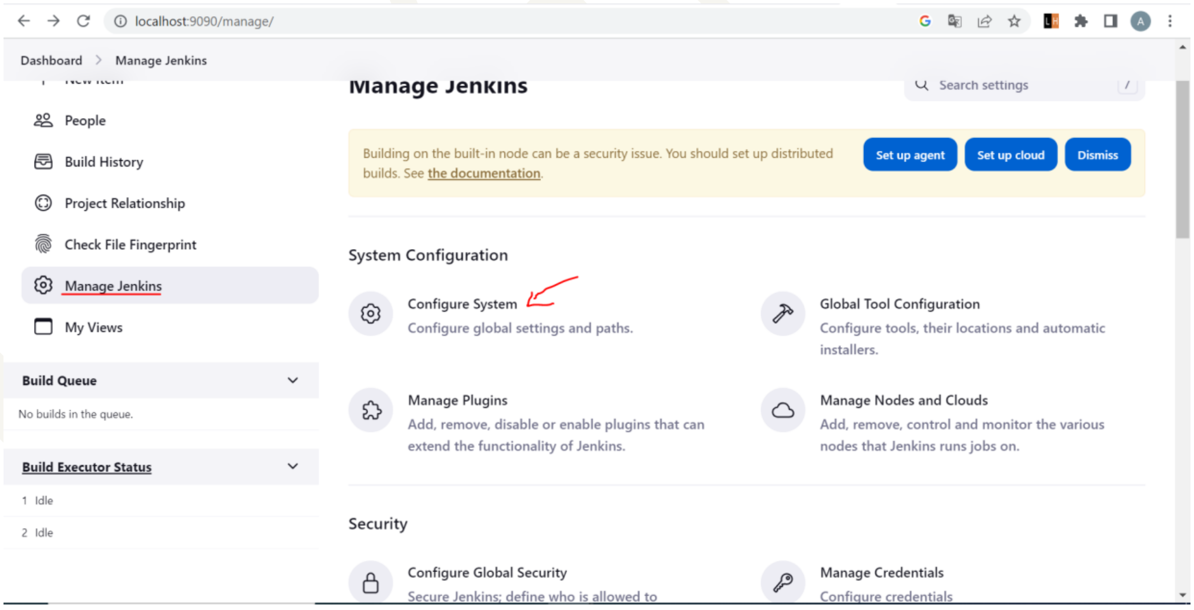
Name: JAVA_HOME
Value: C:\Program Files\Java\jdk-17\

d. ***Setting Environment var 'Docker':***

i. Open ***Jenkins*** in your web browser and go to the "***dashboard***".

ii. Click on "***Manage Jenkins***" from the left-hand menu.

iii. Click on "***Configure System***" from the list of options.


localhost:9090/manage/
Dashboard > Manage Jenkins
Manage Jenkins
Search settings
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
My Views
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
2 Idle
Set up agent
Set up cloud
Dismiss
System Configuration
Configure System
Configure global settings and paths.
Global Tool Configuration
Configure tools, their locations and automatic installers.
Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
Security
Configure Global Security
Secure Jenkins; define who is allowed to
Manage Credentials
Configure credentials

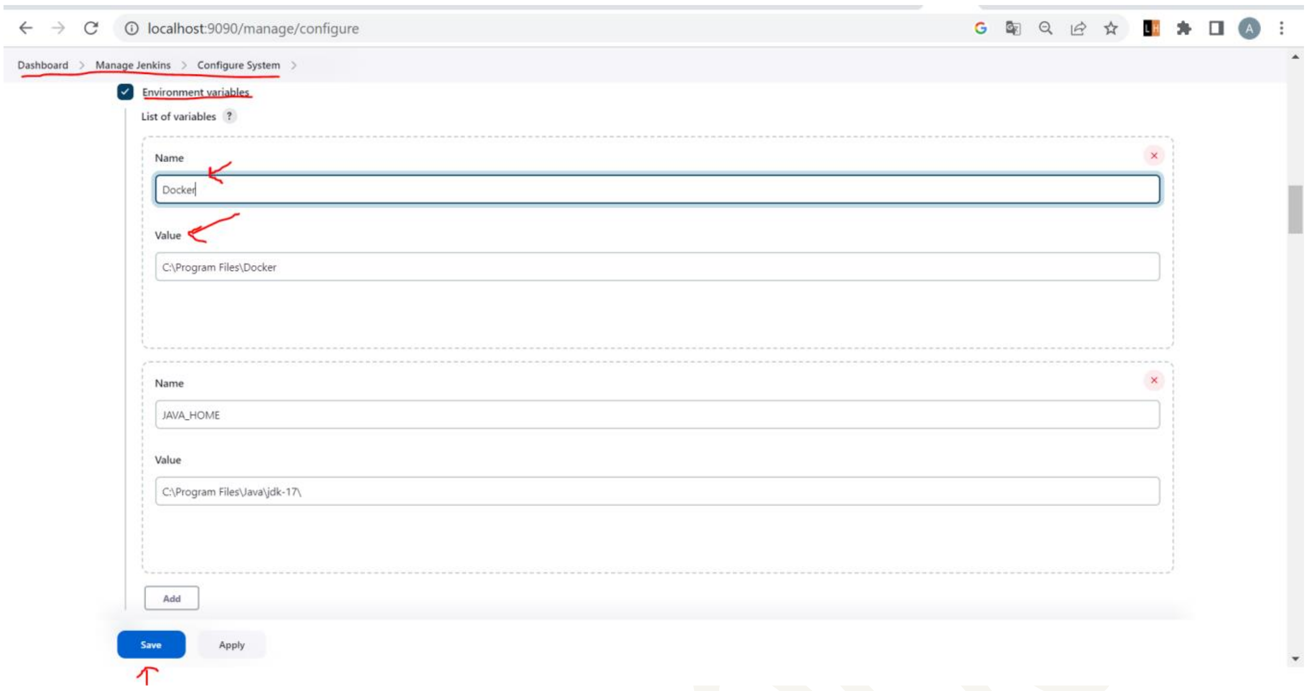
Building on the built-in node can be a security issue. You should set up distributed builds. See the documentation.

iv. Scroll down to "***Global Properties***" and under that check "***Environments Variables***".

v. Click on "***Add***".

vi. Put '***Name***' as "***Docker***" and '***Value***' as the location to your "***Docker directory***". "***C:\Program Files\Docker***" .

vii. Click "***Save***".


localhost:9090/manage/configure

Dashboard > Manage Jenkins > Configure System >

Environment variables

List of variables

Name: Docker
Value: C:\Program Files\Docker

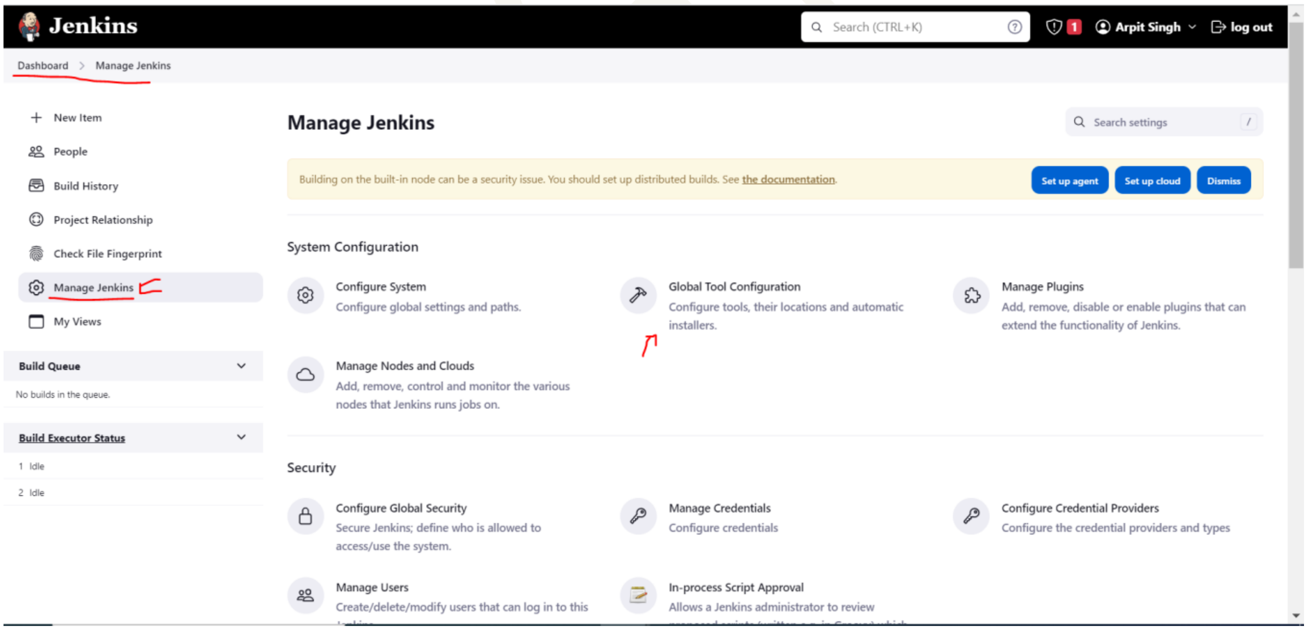
Name: JAVA_HOME
Value: C:\Program Files\Java\jdk-17\

e. ***Setting Global Tool Configuration:***

i. Open ***Jenkins*** in your web browser and go to the "***dashboard***".

ii. Click on "***Manage Jenkins***" from the left-hand menu.

iii. Click on "***Global Configuration***" from the list of options.



No builds in the queue.

1 Idle
2 Idle

Manage Jenkins

Building on the built-in node can be a security issue. You should set up distributed builds. See the documentation.

System Configuration

Configure System — Configure global settings and paths.

Global Tool Configuration — Configure tools, their locations and automatic installers.

Manage Plugins — Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

Manage Nodes and Clouds — Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Security

Configure Global Security — Secure Jenkins; define who is allowed to access/use the system.

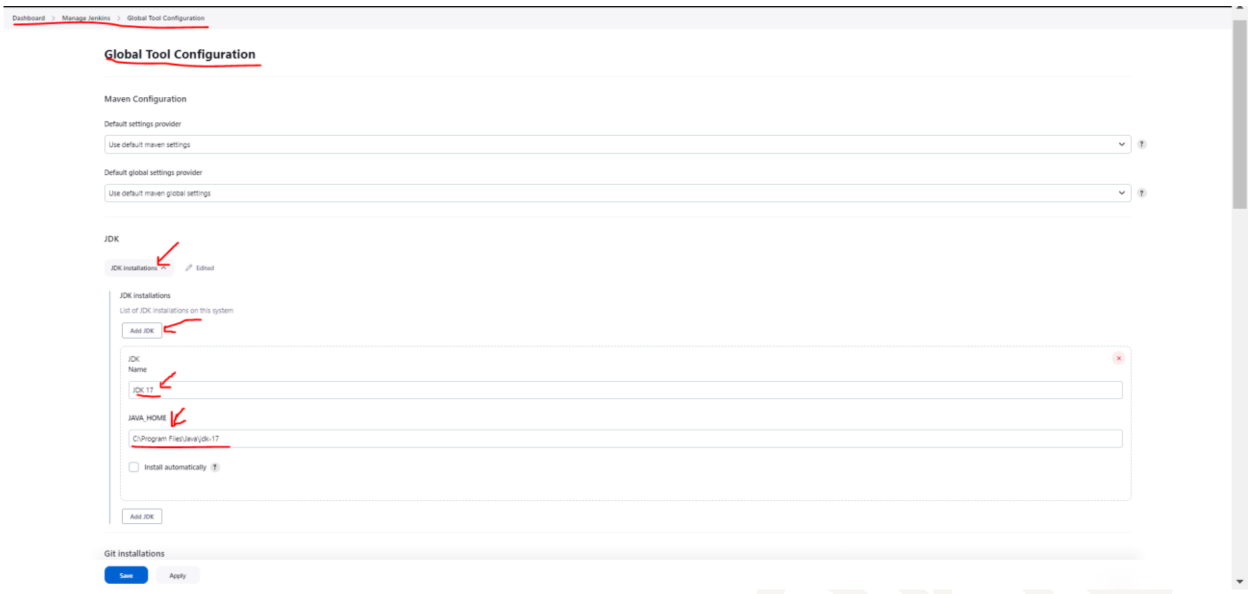
Manage Credentials — Configure credentials

Configure Credential Providers — Configure the credential providers and types

Manage Users — Create/delete/modify users that can log in to this

In-process Script Approval — Allows a Jenkins administrator to review

- ***Java setup:***
  - Click on "***JDK installations***" and "***Add JDK***".
  - Put ***Name*** as "***JDK 17***" and ***JAVA_HOME*** as "***Java directory***". (***C:\Program Files\Java\jdk-17***).



Dashboard > Manage Jenkins > Global Tool Configuration

Global Tool Configuration

Maven Configuration

Default settings provider: Use default maven settings

Default global settings provider: Use default maven global settings

JDK

JDK installations ^ Edited

JDK installations

List of JDK installations on this system

Add JDK

JDK

Name: JDK 17

JAVA_HOME: C:\Program Files\Java\jdk-17

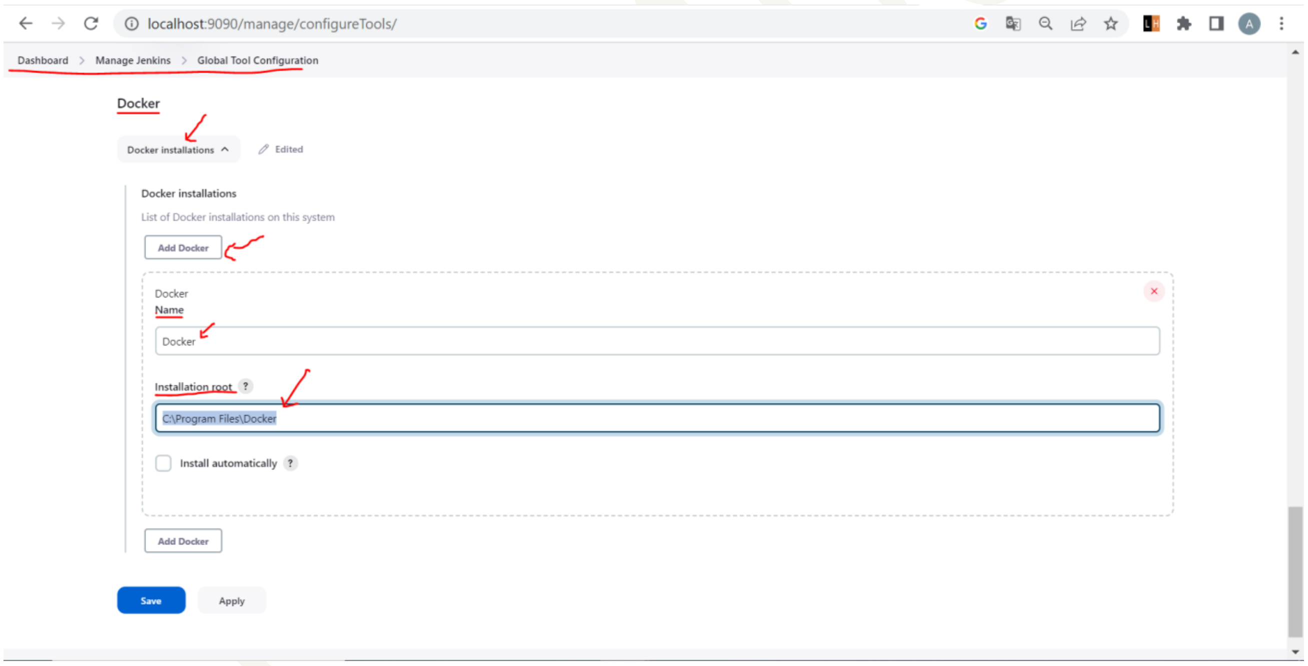
☐ Install automatically

Add JDK

Git installations

Save Apply

- ***Docker setup(similar):***
  - Click on "***Docker installations***" and "***Add Docker***".
  - Put ***Name*** as " ***Docker***" and ***Installation root*** as "***Docker directory***". (***C:\Program Files\Docker***).



localhost:9090/manage/configureTools/

Dashboard > Manage Jenkins > Global Tool Configuration

Docker

Docker installations ^ Edited

Docker installations

List of Docker installations on this system

Add Docker

Docker

Name: Docker

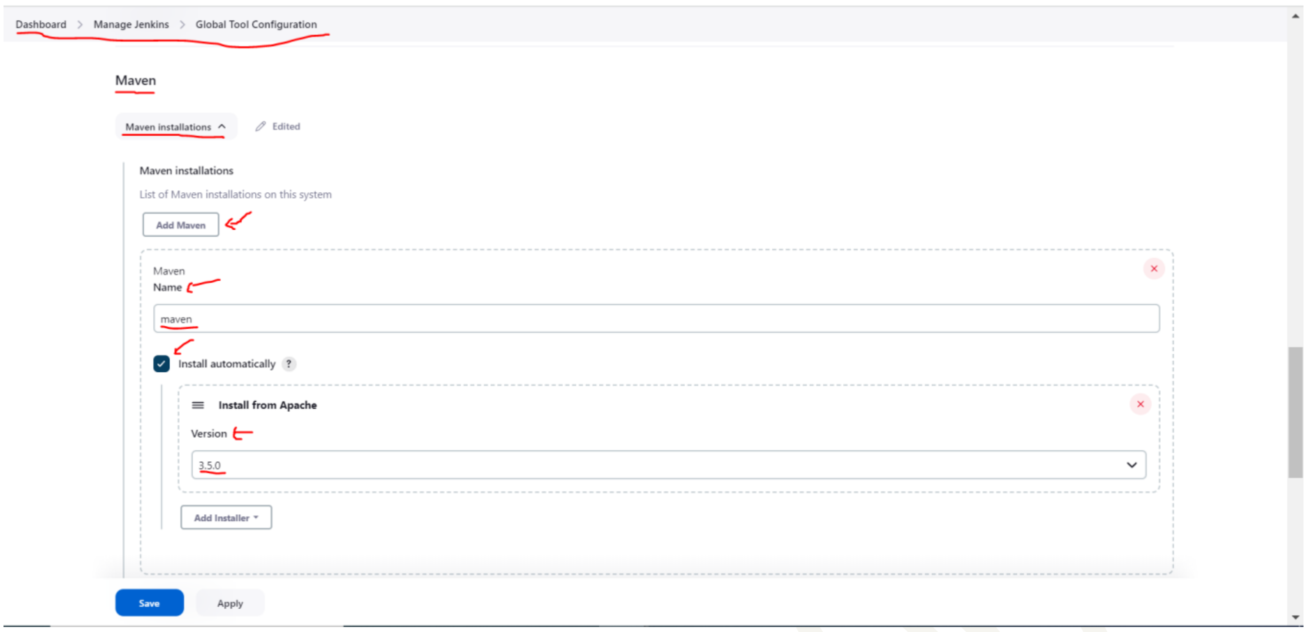
Installation root: C:\Program Files\Docker

☐ Install automatically

Add Docker

Save Apply

- ***Maven setup:***
  - Click on "***Maven installations***" and "***Add Maven***".
  - Put ***Name*** as " ***maven***".
  - Check "***install automatically***" and set the '***Version***' as "***3.5.0***". (This specific version as it is being used to build our sample application).


Dashboard > Manage Jenkins > Global Tool Configuration

Maven

Maven installations

List of Maven installations on this system

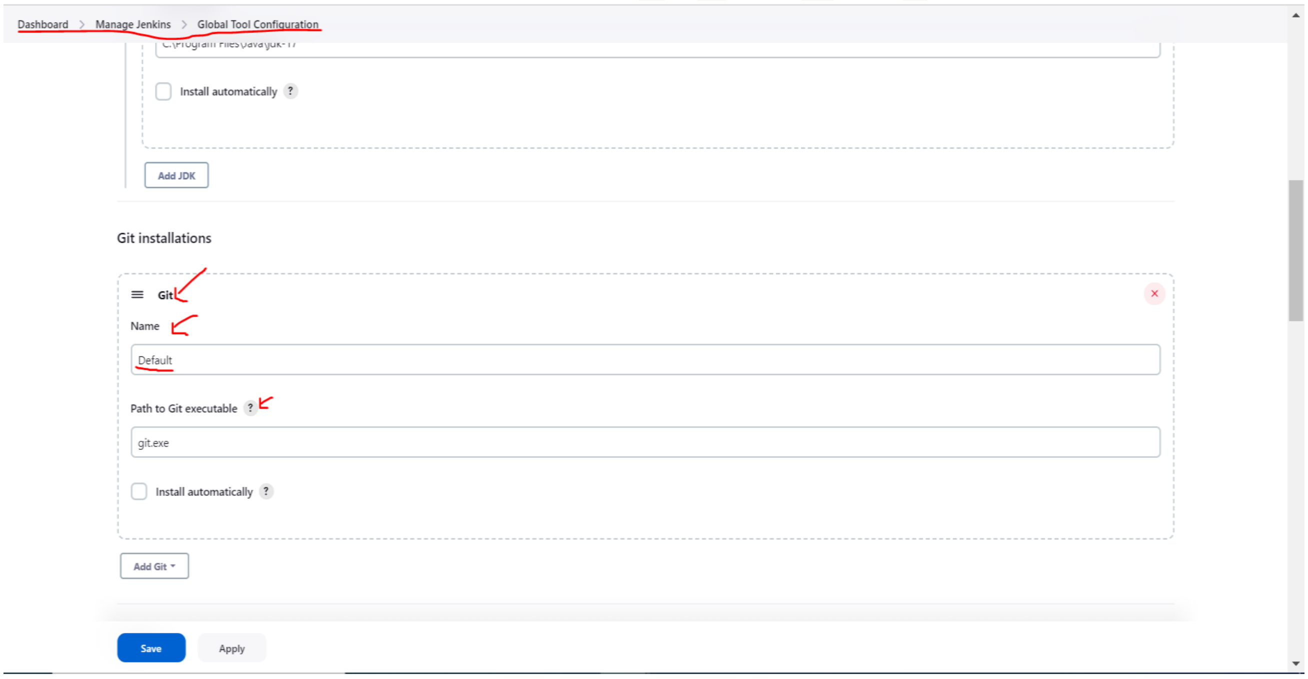
Maven Name: maven

[✓] Install automatically

Install from Apache

Version: 3.5.0

- ***Git setup:***
  - By ***Default Git is already set*** up.


Dashboard > Manage Jenkins > Global Tool Configuration

C:\Program Files\Java\jdk-17

[ ] Install automatically

Git installations

Git

Name: Default

Path to Git executable: git.exe

[ ] Install automatically

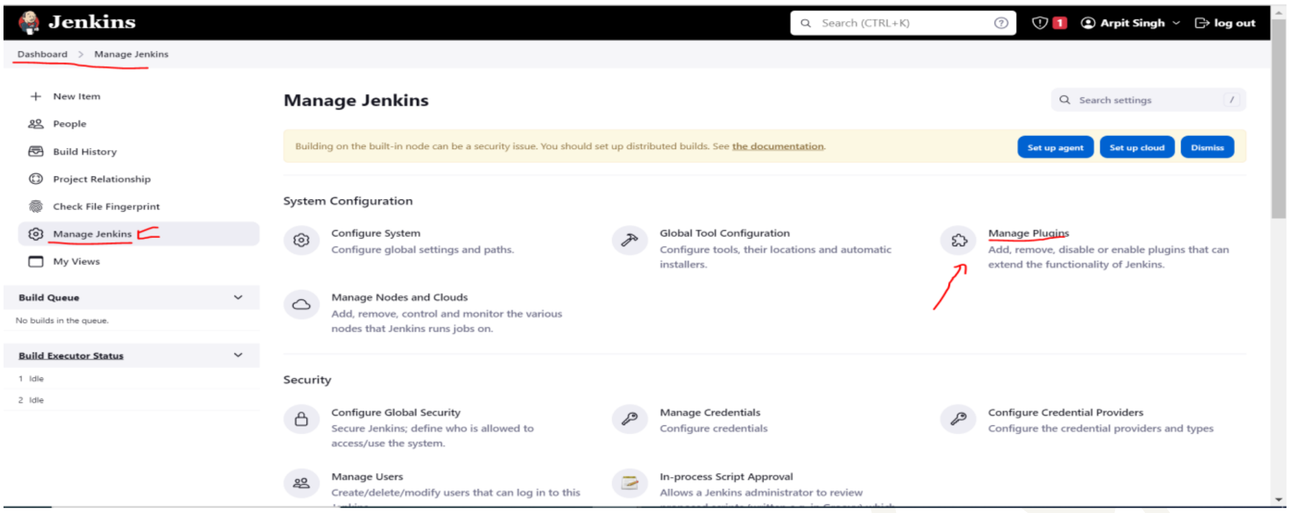
f. Click "***Save***".

g. Add "***Plugins***" - ***Docker and Pipeline***:

   i. Open ***Jenkins*** in your web browser and go to the "***dashboard***".

   ii. Click on "***Manage Jenkins***" from the left-hand menu.

   iii. Click on "***Manage Plugins***" from the list of options.



Building on the built-in node can be a security issue. You should set up distributed builds. See the documentation.

System Configuration

Configure System
Configure global settings and paths.

Global Tool Configuration
Configure tools, their locations and automatic installers.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Security

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

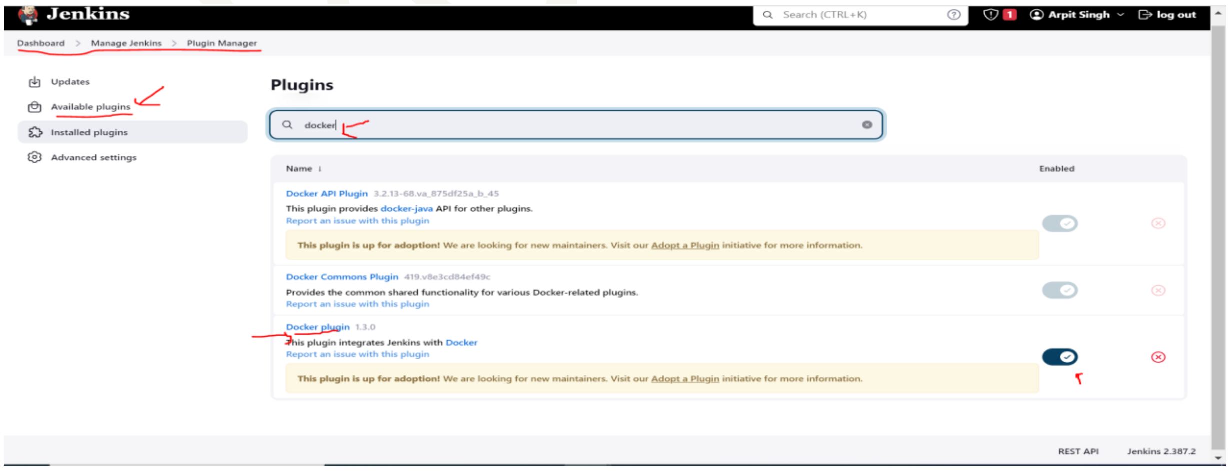
Manage Credentials
Configure credentials

Configure Credential Providers
Configure the credential providers and types

Manage Users
Create/delete/modify users that can log in to this

In-process Script Approval
Allows a Jenkins administrator to review

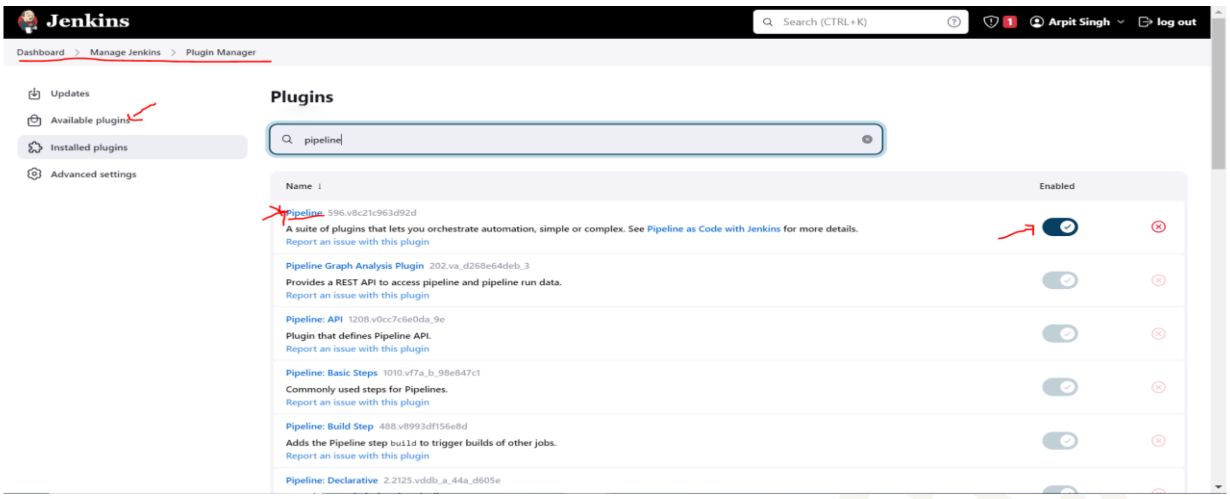
- ***Docker and Pipeline (Plugin):***
    - Click on "***Available Plugins***" and "***Search for Docker and Pipeline***".
    - Click on the "***Available***" tab to see the list of available plugins.
    - Search for "***Docker Plugin***" in the search bar.
    - ***Check the checkbox*** next to "***Docker Plugin***" and then click the "***Download now and install after restart***" button.
    - ***Similarly, search for "Pipeline Plugin"*** and install it by following the same steps.
    - Wait for the installation to complete and then ***restart Jenkins***.



Docker API Plugin 3.2.13-68.va_875df25a_b_45
This plugin provides docker-java API for other plugins.
Report an issue with this plugin
This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.

Docker Commons Plugin 419.v8e3cd84ef49c
Provides the common shared functionality for various Docker-related plugins.
Report an issue with this plugin

Docker plugin 1.3.0
This plugin integrates Jenkins with Docker
Report an issue with this plugin
This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.


Jenkins

Search (CTRL+K) | 1 | Arpit Singh | log out

Dashboard > Manage Jenkins > Plugin Manager

- Updates
- Available plugins
- Installed plugins
- Advanced settings

Plugins

pipeline

Name ↓	Enabled
Pipeline 596.v8c21c963d92d A suite of plugins that lets you orchestrate automation, simple or complex. See Pipeline as Code with Jenkins for more details. Report an issue with this plugin	✓
Pipeline Graph Analysis Plugin 202.va_d268e64deb_3 Provides a REST API to access pipeline and pipeline run data. Report an issue with this plugin	✓
Pipeline: API 1208.v0cc7c6e0da_9e Plugin that defines Pipeline API. Report an issue with this plugin	✓
Pipeline: Basic Steps 1010.vf7a_b_98e847c1 Commonly used steps for Pipelines. Report an issue with this plugin	✓
Pipeline: Build Step 488.v8993df156e8d Adds the Pipeline step `build` to trigger builds of other jobs. Report an issue with this plugin	✓
Pipeline: Declarative 2.2125.vddb_a_44a_d605e	

- With this "***Jenkins***" is all ***set up and ready*** to be used.

Jenkins

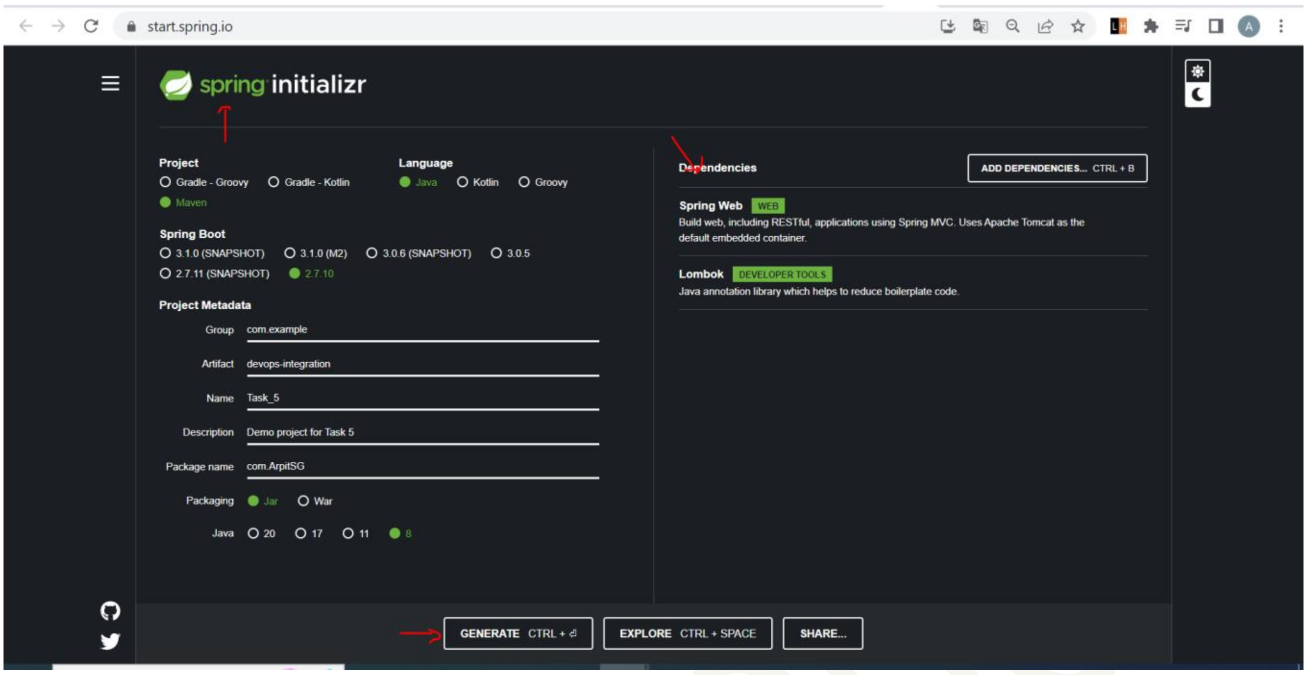
3. **Setting up Sample Application**:

a. ***Creating Sample "Maven Project"(Similar to Task_1)***:

i. Create a ***new Spring Boot***(maven) Project:

1. To create a new Spring Boot project, you can use ***Spring Initializr: https://start.spring.io/.***
2. Choose the following options:
   a. Project: Maven Project.
   b. Language: Java.
   c. Spring Boot: 2.7.0 (as per preference).
   d. Group: com.ArpitSG.
   e. Artifact: devops-integration.
   f. Packaging: Jar.
   g. Java: 8
3. Click on the "***Add dependencies***" button and ***add the following*** dependencies:
   a. Spring Web.
   b. Lombok.

spring boot


start.spring.io

spring initializr

Project
Gradle - Groovy [✓ Maven] Gradle - Kotlin

Language
[✓ Java] Kotlin Groovy

Spring Boot
3.1.0 (SNAPSHOT) 3.1.0 (M2) 3.0.6 (SNAPSHOT) 3.0.5
2.7.11 (SNAPSHOT) [✓ 2.7.10]

Project Metadata

Group	com.example
Artifact	devops-integration
Name	Task_5
Description	Demo project for Task 5
Package name	com.ArpitSG
Packaging	[✓ Jar] War
Java	20 17 11 [✓ 8]

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

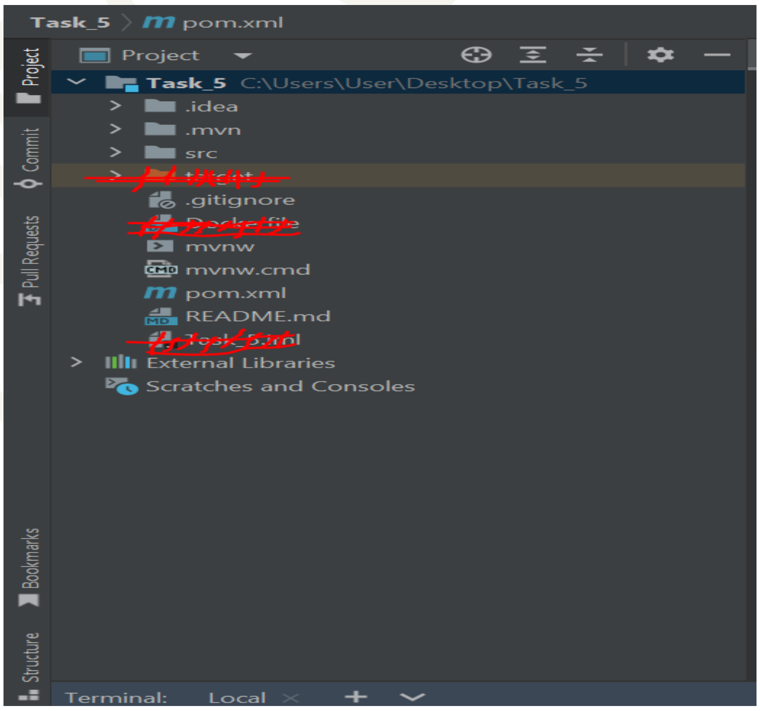
Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

b. **Generate the project and open it in your *preferred IDE* (*IntelliJ IDEA*** in my case).

i. After Clicking generate we get a file named : "***Task_5.zip***"

ii. Unzip it to a specified location on your system. (***C:\Users\User\Desktop\Task_5***).

iii. ***Open this Directory*** in your IDE as a new project. (We get a file structure as show below)


```
Task_5 > pom.xml
Project
v Task_5  C:\Users\User\Desktop\Task_5
  > .idea
  > .mvn
  > src
  > ~~target~~
    .gitignore
    ~~Dockerfile~~
    mvnw
    mvnw.cmd
    pom.xml
    README.md
    ~~Task_5.iml~~
> External Libraries
  Scratches and Consoles
```

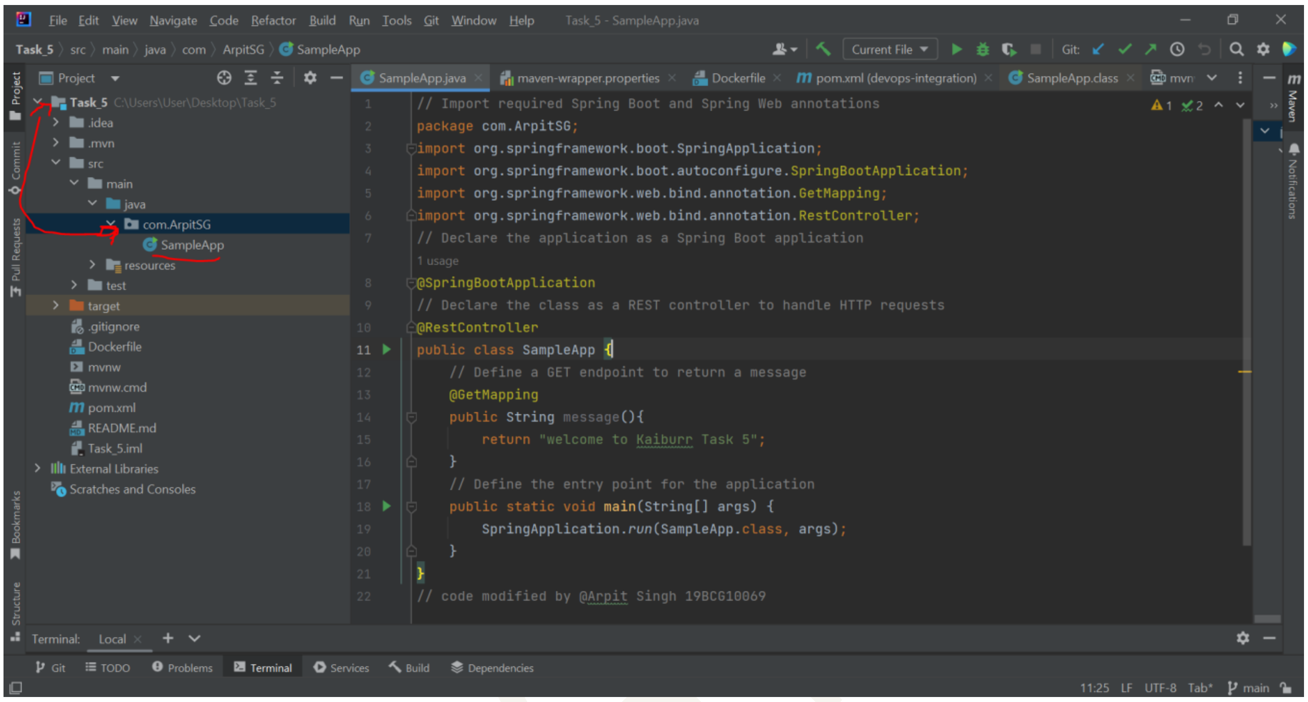
(The crossed files will be added later)

c. **Code Addition to Sample App**:

i. Go to the Main Class Folder. (***C:\Users\User\Desktop\Task_5\src\main\java\com\ArpitSG***).

ii. ***Refactor*** the main class as "***SampleApp***"

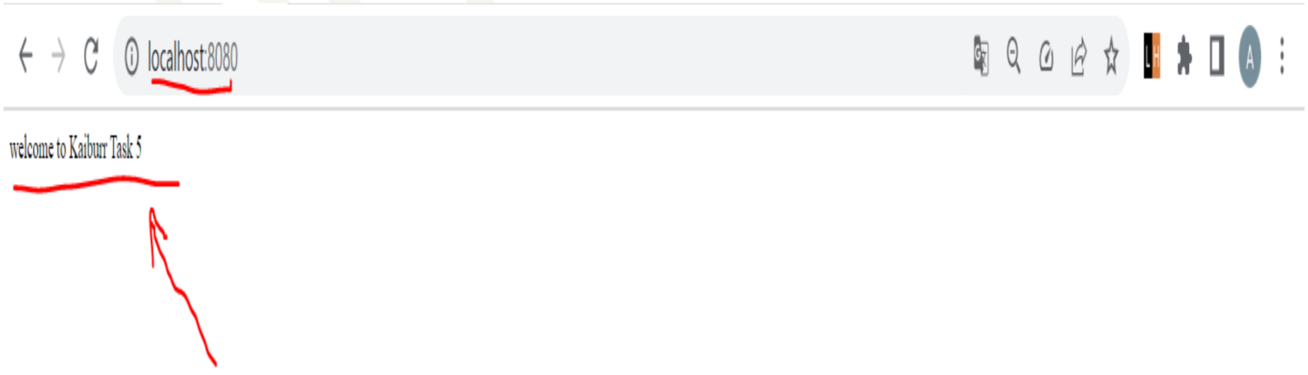
iii. Add the '***Following Code'*** to this class:



```java
// Import required Spring Boot and Spring Web annotations
package com.ArpitSG;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
// Declare the application as a Spring Boot application
@SpringBootApplication
// Declare the class as a REST controller to handle HTTP requests
@RestController
public class SampleApp {
    // Define a GET endpoint to return a message
    @GetMapping
    public String message(){
        return "welcome to Kaiburr Task 5";
    }
    // Define the entry point for the application
    public static void main(String[] args) {
        SpringApplication.run(SampleApp.class, args);
    }
}
// code modified by @Arpit Singh 19BCG10069
```

Explanation code:

- When executed, the application will ***start an embedded Tomcat server*** and listen for HTTP requests on ***port 8080***. When a GET request is made to the root e***ndpoint "/", the message() method*** will be called and return the string "***welcome to Kaiburr Task 5***". This message will be ***displayed in the browser*** or in the response of an API client that makes a GET request to the endpoint.

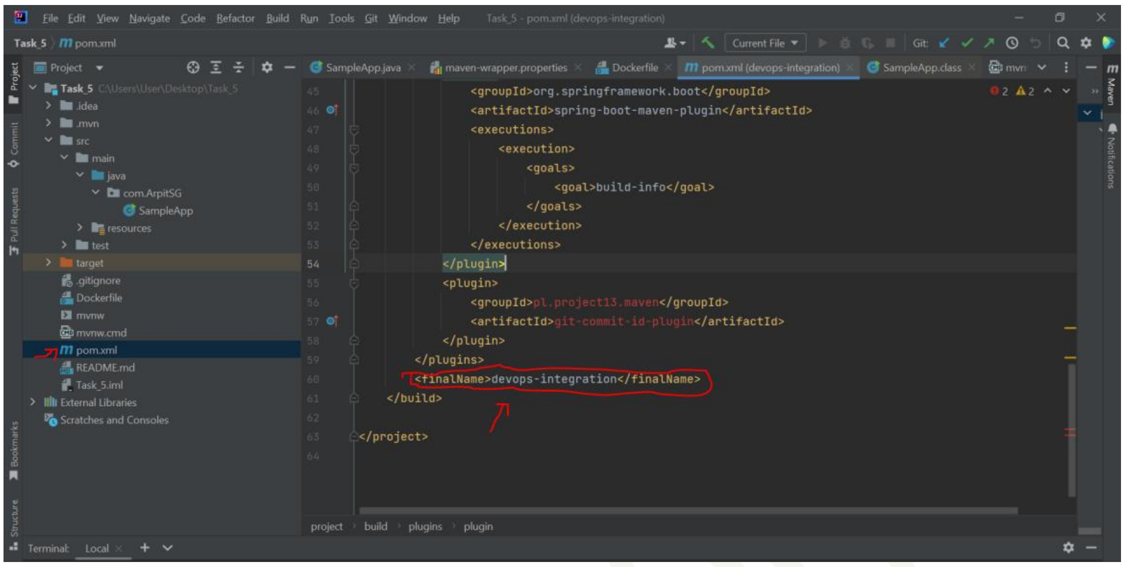


```
localhost:8080

welcome to Kaiburr Task 5
```

d. Add the '***Following Code'*** to "***Pom.xml***":

i. `<finalName>devops-integration</finalName>`

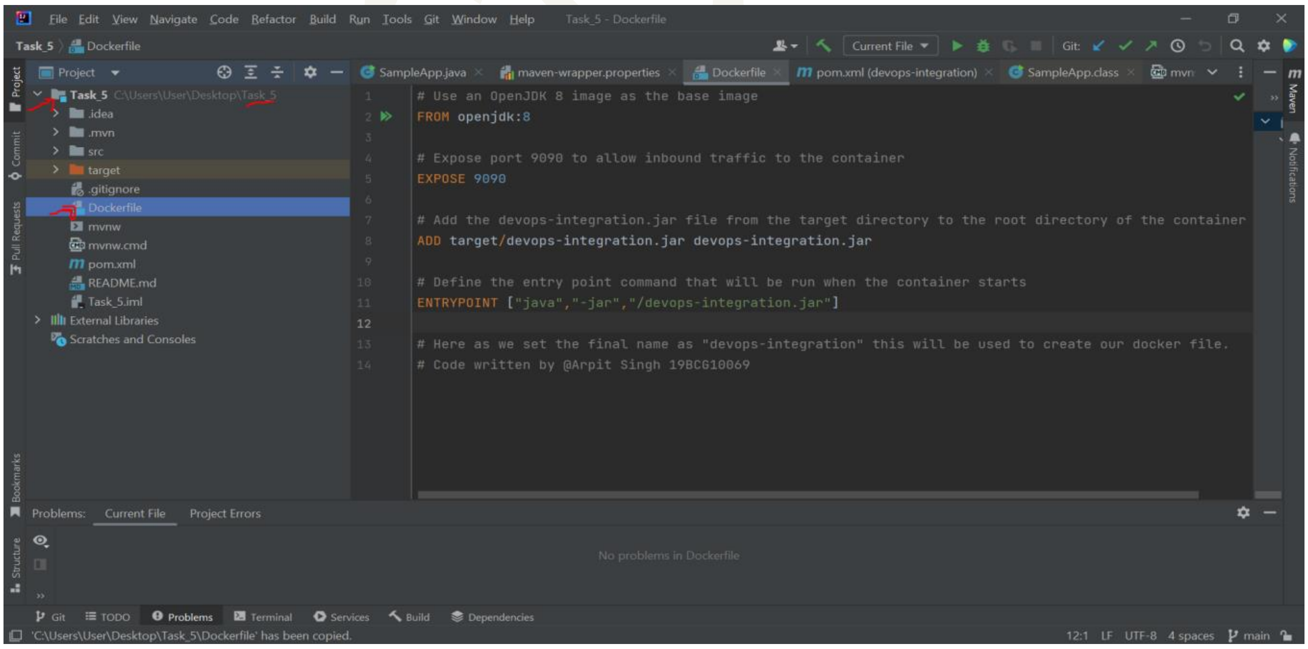
ii. We add this for '***future working***'. That is, in order to create a ***DockerFile*** we need this code line to be added in "***pom.xml***".



```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <executions>
                <execution>
                    <goals>
                        <goal>build-info</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>pl.project13.maven</groupId>
            <artifactId>git-commit-id-plugin</artifactId>
        </plugin>
    </plugins>
    <finalName>devops-integration</finalName>
</build>

</project>
```

e. "***Save***" the Project.

4. **Create DockerFile**:
   a. In the "***Root - Directory***" of the project ***create*** a File named "***DockerFile***".(***C:\Users\User\Desktop\Task_5\Dockerfile***).
   b. Add the "***Following code***" to it:



```
# Use an OpenJDK 8 image as the base image
FROM openjdk:8

# Expose port 9090 to allow inbound traffic to the container
EXPOSE 9090

# Add the devops-integration.jar file from the target directory to the root directory of the container
ADD target/devops-integration.jar devops-integration.jar

# Define the entry point command that will be run when the container starts
ENTRYPOINT ["java","-jar","/devops-integration.jar"]

# Here as we set the final name as "devops-integration" this will be used to create our docker file.
# Code written by @Arpit Singh 19BCG10069
```

Explanation code:

- The first line specifies the base image to use, which is an ***OpenJDK 8 image***.

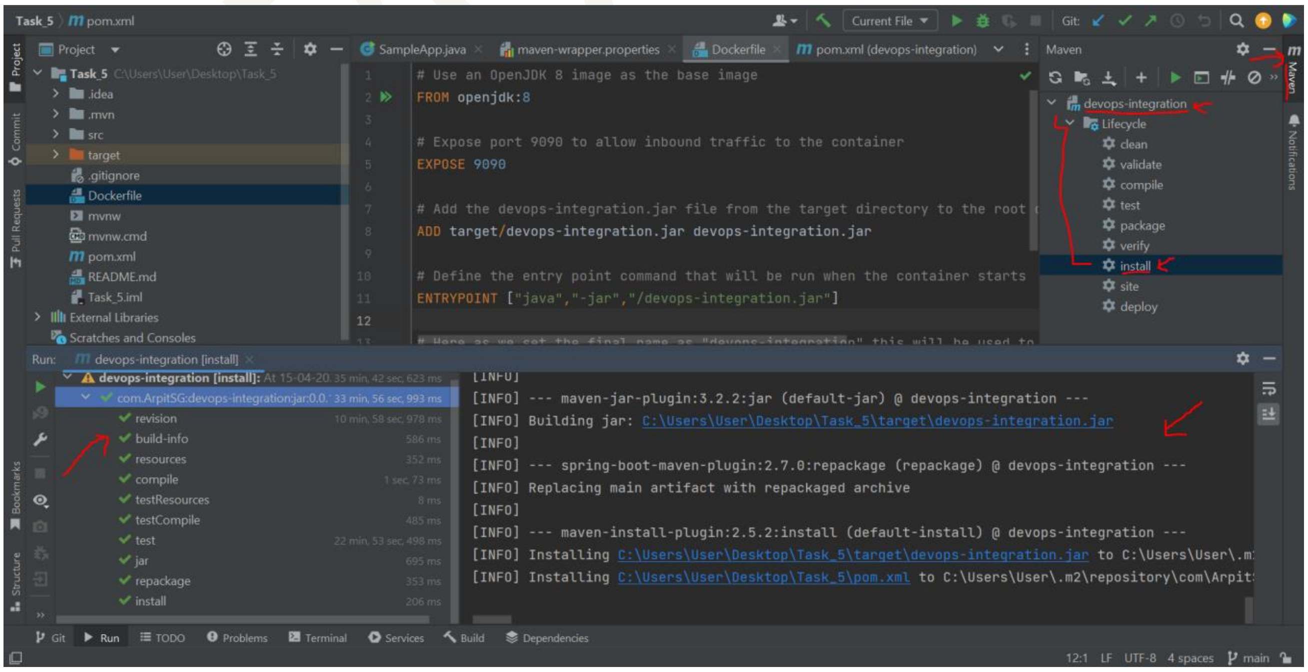
- The second line ***exposes port 9090*** on the container, allowing inbound traffic to reach the application running inside the container.

- The third line adds the ***JAR file named devops-integration.jar from the target directory*** of the host machine to the root directory of the container. This assumes that the JAR file has been built and is available in the target directory.

- Finally, the fourth line specifies the command that will be run when the container starts. In this case, the command is java -jar /devops-integration.jar, which runs the JAR file using the Java runtime environment inside the container. This will ***start the application running on port 9090 inside the container***, which can be accessed from the host machine by mapping port 9090 to a port on the host machine.

c. To add this "***devops-integration build***" to our ***target folder*** from which the "***Jenkins" script*** will fetch the image and push it to our "***DockerHub***" we have to run a "***maven install***". {The full JenkinsFile script Explained Later}.

   i. Click on "***Maven***" available on the right side of the view.
   ii. Open the "***Maven***" tab.
   iii. Expand and run ***"devops-integration" > Lifecycle > install***.



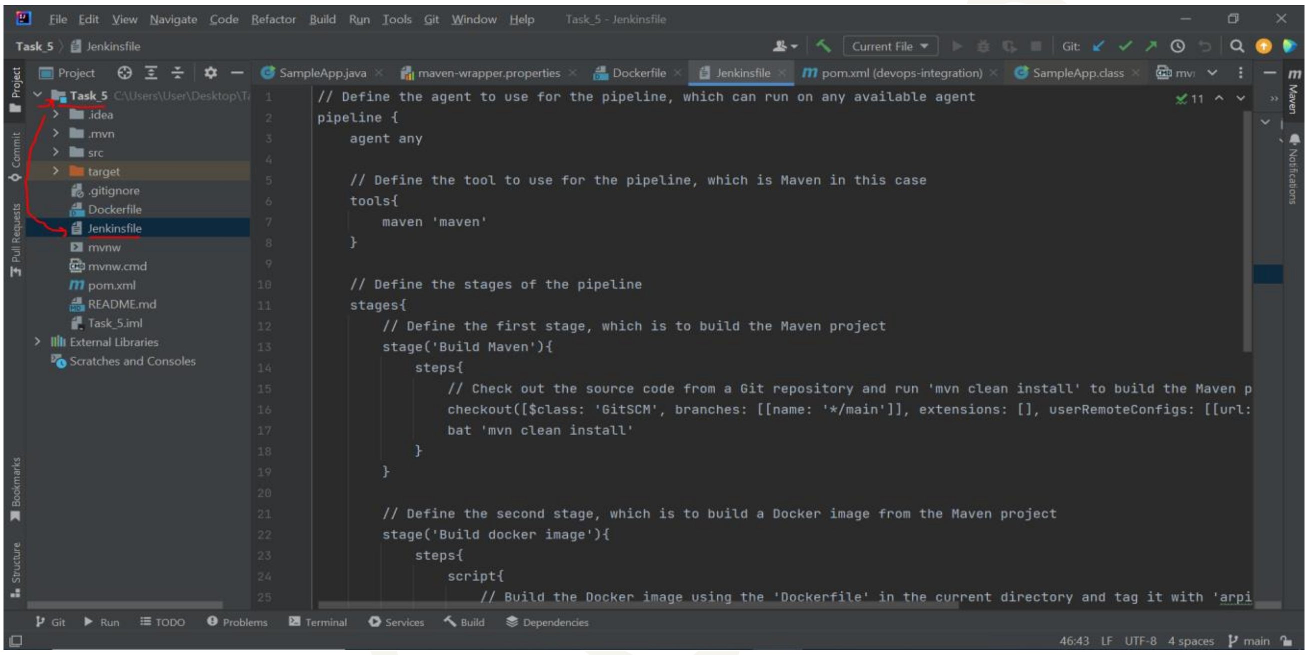
```
# Use an OpenJDK 8 image as the base image
FROM openjdk:8

# Expose port 9090 to allow inbound traffic to the container
EXPOSE 9090

# Add the devops-integration.jar file from the target directory to the root
ADD target/devops-integration.jar devops-integration.jar

# Define the entry point command that will be run when the container starts
ENTRYPOINT ["java","-jar","/devops-integration.jar"]
```

```
[INFO] --- maven-jar-plugin:3.2.2:jar (default-jar) @ devops-integration ---
[INFO] Building jar: C:\Users\User\Desktop\Task_5\target\devops-integration.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.0:repackage (repackage) @ devops-integration ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ devops-integration ---
[INFO] Installing C:\Users\User\Desktop\Task_5\target\devops-integration.jar to C:\Users\User\.m
[INFO] Installing C:\Users\User\Desktop\Task_5\pom.xml to C:\Users\User\.m2\repository\com\Arpit
```

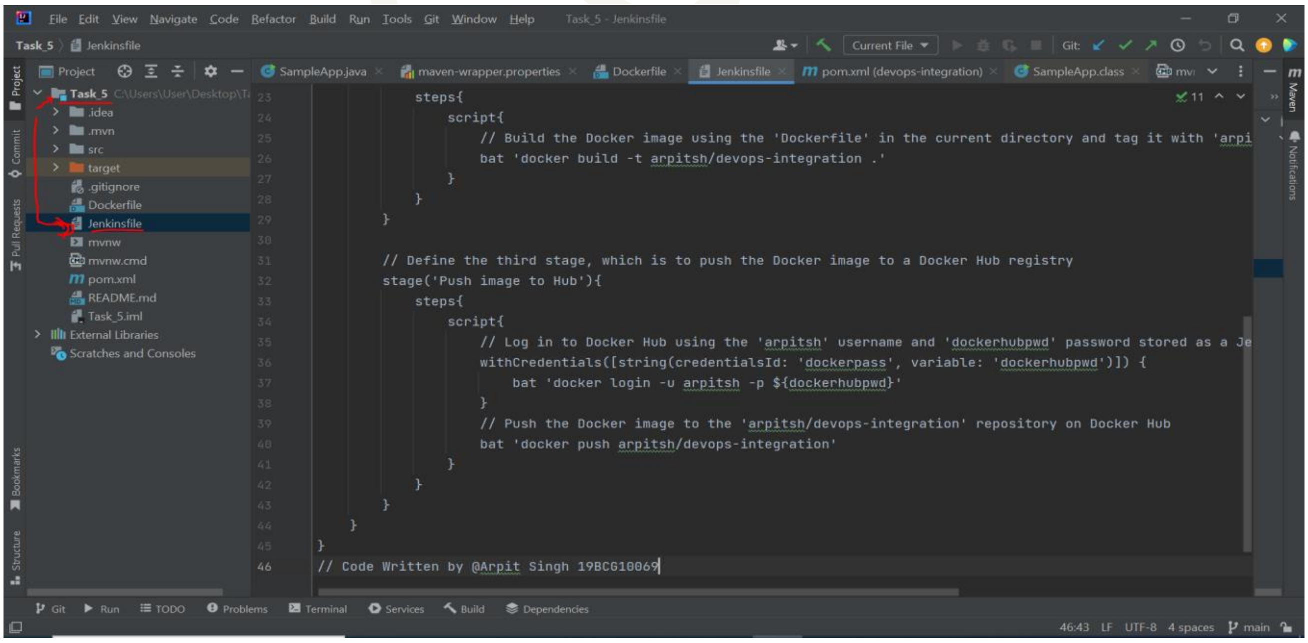
- With this "***DockerFile***" is all ***set up and ready*** to be managed by "***JenkinsFile***".

5. **Create JenkinsFile**:
    a. In the "***Root - Directory***" of the project ***create*** a File named "***JenkinsFile***".(***C:\Users\User\Desktop\Task_5\JenkinsFile***).

    b. Add the "***Following code***" to it:



```
// Define the agent to use for the pipeline, which can run on any available agent
pipeline {
    agent any

    // Define the tool to use for the pipeline, which is Maven in this case
    tools{
        maven 'maven'
    }

    // Define the stages of the pipeline
    stages{
        // Define the first stage, which is to build the Maven project
        stage('Build Maven'){
            steps{
                // Check out the source code from a Git repository and run 'mvn clean install' to build the Maven p
                checkout([$class: 'GitSCM', branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url:
                bat 'mvn clean install'
            }
        }

        // Define the second stage, which is to build a Docker image from the Maven project
        stage('Build docker image'){
            steps{
                script{
                    // Build the Docker image using the 'Dockerfile' in the current directory and tag it with 'arpi
```



```
            steps{
                script{
                    // Build the Docker image using the 'Dockerfile' in the current directory and tag it with 'arpi
                    bat 'docker build -t arpitsh/devops-integration .'
                }
            }
        }

        // Define the third stage, which is to push the Docker image to a Docker Hub registry
        stage('Push image to Hub'){
            steps{
                script{
                    // Log in to Docker Hub using the 'arpitsh' username and 'dockerhubpwd' password stored as a Je
                    withCredentials([string(credentialsId: 'dockerpass', variable: 'dockerhubpwd')]) {
                        bat 'docker login -u arpitsh -p ${dockerhubpwd}'
                    }
                    // Push the Docker image to the 'arpitsh/devops-integration' repository on Docker Hub
                    bat 'docker push arpitsh/devops-integration'
                }
            }
        }
    }
}
// Code Written by @Arpit Singh 19BCG10069
```

Explanation code:

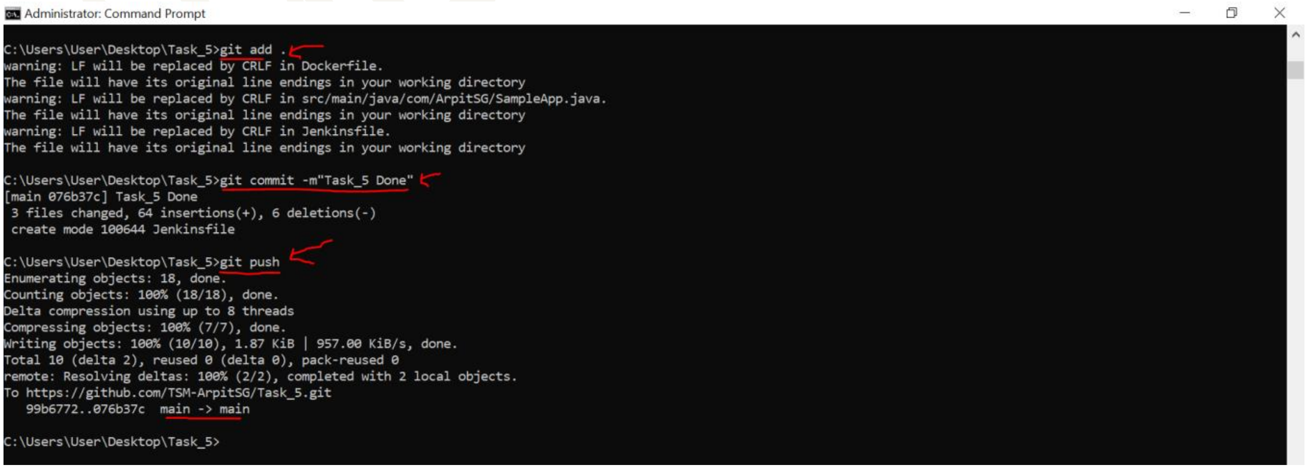
- The provided ***Jenkinsfile*** describes a '***Jenkins pipeline***' that "***automates the build and deployment of a Maven-based Java application***" packaged as a Docker image to a Docker Hub registry.

- When executed, the pipeline will perform the "***following steps***":
    - '***Check out the source code***' for the Java application from a Git repository.
    - ***Build the Maven project*** using the '***mvn clean install***' command.
    - ***Build a Docker image*** from the Maven project using the '***Dockerfile' in the current directory*** and ***tag it with 'arpitsh/devops-integration***'.
    - ***Login to Docker Hub*** using the ***'arpitsh' username*** and ***'dockerhubpwd' password*** stored as a Jenkins credential.
    - ***Push the Docker image*** to the ***'arpitsh/devops-integration'*** repository on ***Docker Hub***.

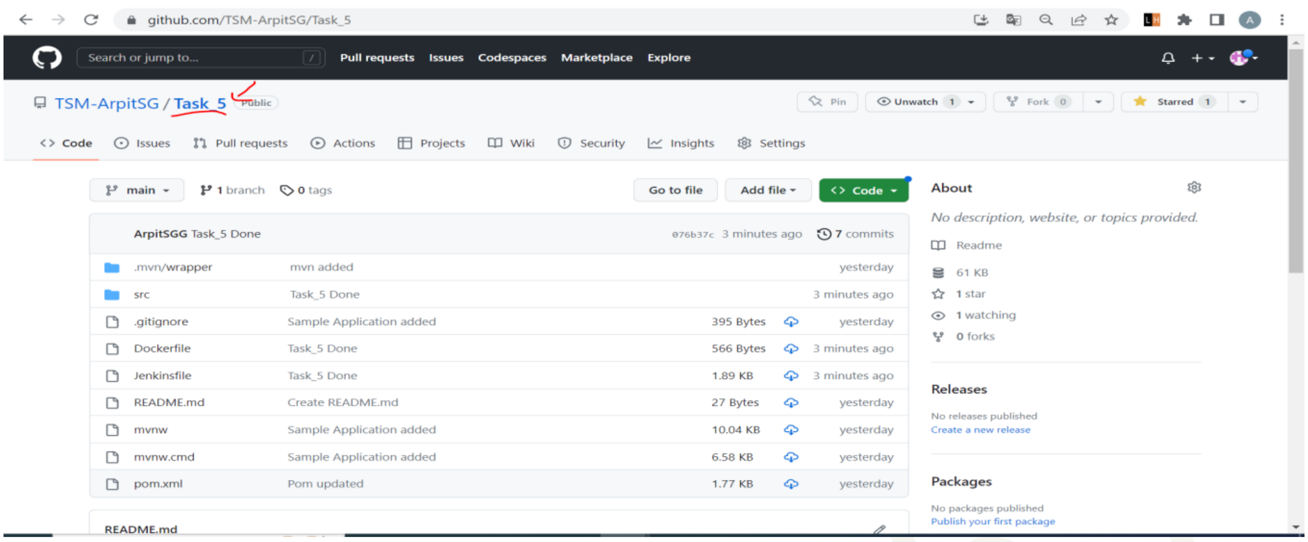
- "***Save***" the Project.

6. **Upload The Project(Task_5)**:
    a. Using "***Git Commands***" I simply upload this '***Task_5***' project to ***GitHub***.
        i. ***cd C:\Users\User\Desktop\Task_5***
        ii. ***git add .***
        iii. ***git commit -m"Task_5 Done"***
        iv. ***git push***

    b. With these commands our code would be uploaded to Github. (https://github.com/TSM-ArpitSG/Task_5).


Administrator: Command Prompt

```
C:\Users\User\Desktop\Task_5>git add .
warning: LF will be replaced by CRLF in Dockerfile.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/main/java/com/ArpitSG/SampleApp.java.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in Jenkinsfile.
The file will have its original line endings in your working directory

C:\Users\User\Desktop\Task_5>git commit -m"Task_5 Done"
[main 076b37c] Task_5 Done
 3 files changed, 64 insertions(+), 6 deletions(-)
 create mode 100644 Jenkinsfile

C:\Users\User\Desktop\Task_5>git push
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (10/10), 1.87 KiB | 957.00 KiB/s, done.
Total 10 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/TSM-ArpitSG/Task_5.git
   99b6772..076b37c  main -> main

C:\Users\User\Desktop\Task_5>
```



github.com/TSM-ArpitSG/Task_5

TSM-ArpitSG / Task_5 Public

main · 1 branch · 0 tags

ArpitSGG Task_5 Done · 076b37c 3 minutes ago · 7 commits

.mvn/wrapper	mvn added		yesterday
src	Task_5 Done		3 minutes ago
.gitignore	Sample Application added	395 Bytes	yesterday
Dockerfile	Task_5 Done	566 Bytes	3 minutes ago
Jenkinsfile	Task_5 Done	1.89 KB	3 minutes ago
README.md	Create README.md	27 Bytes	yesterday
mvnw	Sample Application added	10.04 KB	yesterday
mvnw.cmd	Sample Application added	6.58 KB	yesterday
pom.xml	Pom updated	1.77 KB	yesterday

About: No description, website, or topics provided. Readme · 61 KB · 1 star · 1 watching · 0 forks

Releases: No releases published

Packages: No packages published

- With this "***SampleApp***" is all ***set up and ready*** to be managed by "***JenkinsFile***".

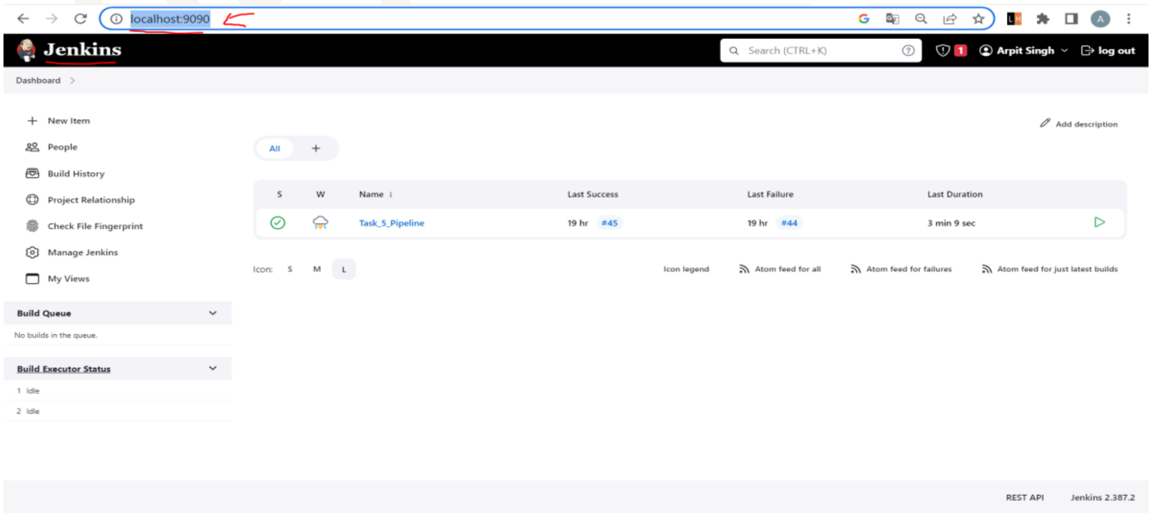
****

NOTE:

- **For avoiding any "unexpected errors" to occur we need to make sure the following are up and running in our system:**
    1. **Docker Hub (logged in).**
    2. **Maven**
    3. **Java 17**
    4. **Jenkins**

****

7. **Create a New Pipeline using Jenkins and Configure it**:
    a. Go to '***Jenkins home page***' by using the port you mentioned during the installation phase. (***http://localhost:9090/***).



localhost:9090

Jenkins · Arpit Singh · log out

Dashboard

New Item · People · Build History · Project Relationship · Check File Fingerprint · Manage Jenkins · My Views

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓		Task_5_Pipeline	19 hr #45	19 hr #44	3 min 9 sec

Build Queue: No builds in the queue.

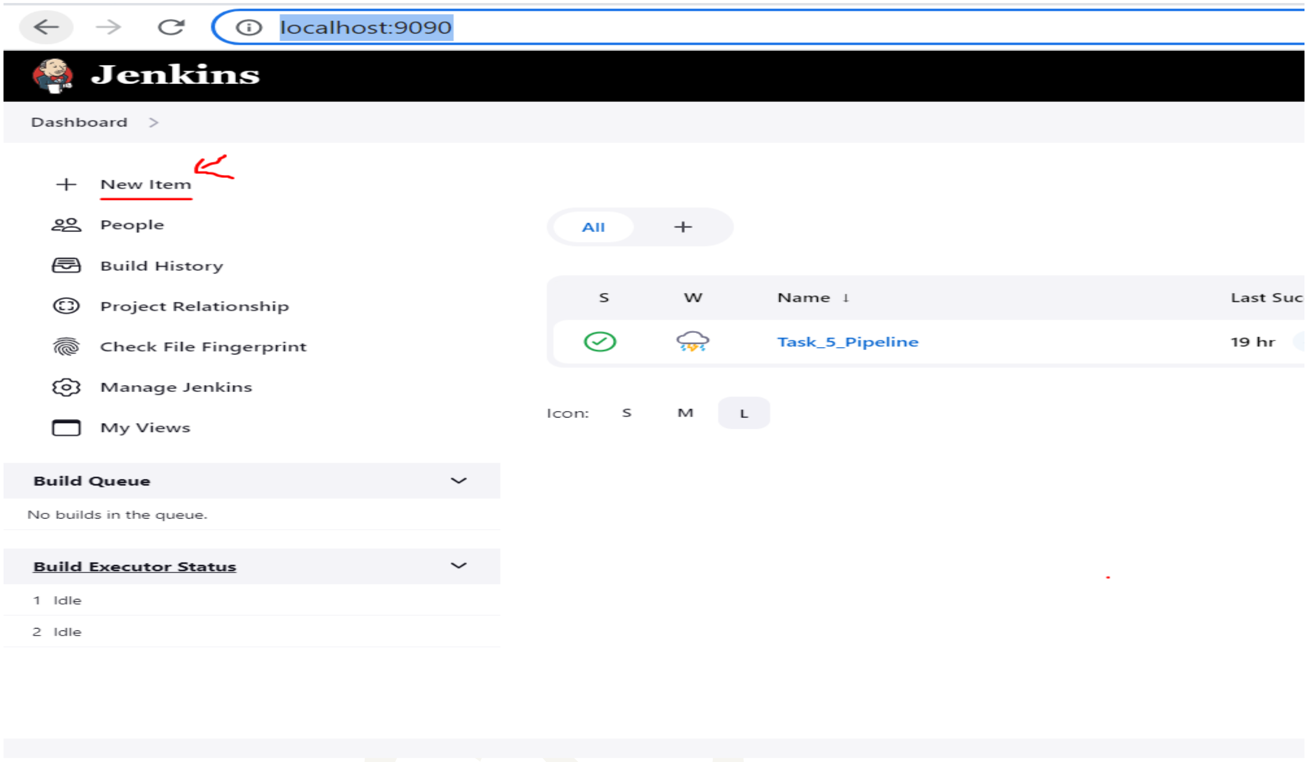
Build Executor Status: 1 Idle · 2 Idle

REST API · Jenkins 2.387.2

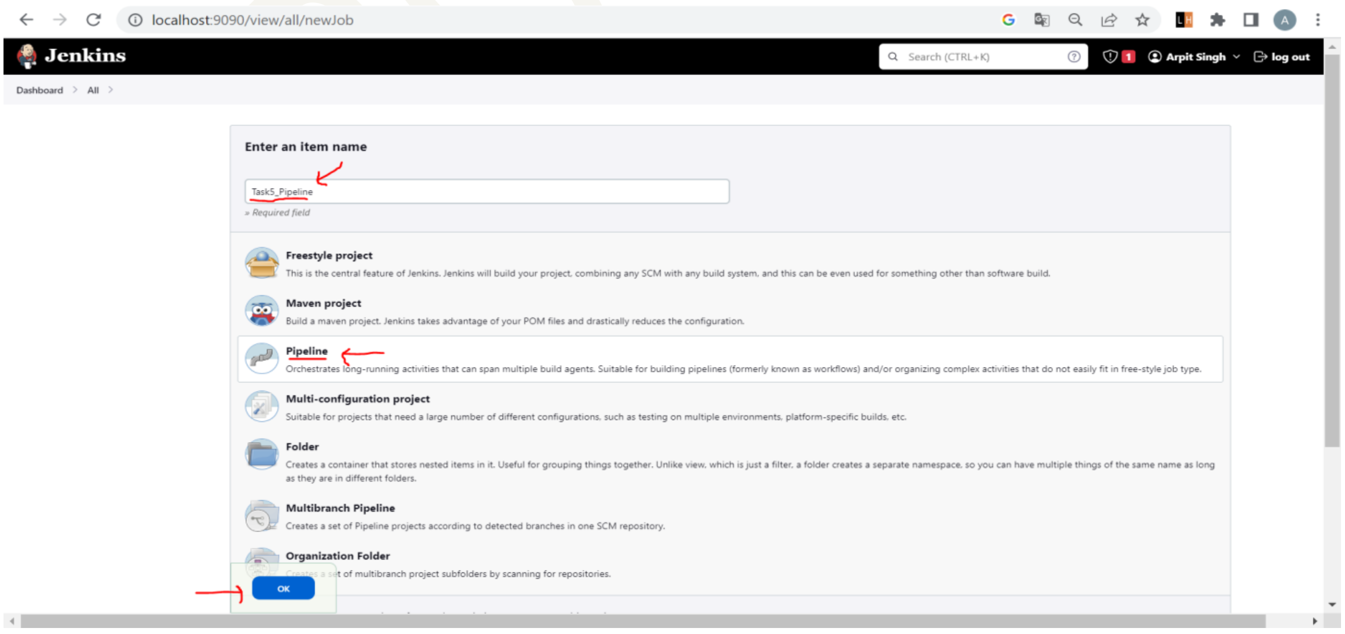
b. ***Create Jenkins*** Pipeline:

   i. Open Jenkins in your ***browser and log in*** with your credentials.

   ii. Click on "***New Item***" on the Jenkins home page to create a new pipeline.



   iii. Enter a ***name*** for your pipeline (***Task5_Pipeline***) , select "***Pipeline***" as the project type, and click on "***OK***".



Enter an item name

Task5_Pipeline

» Required field

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.

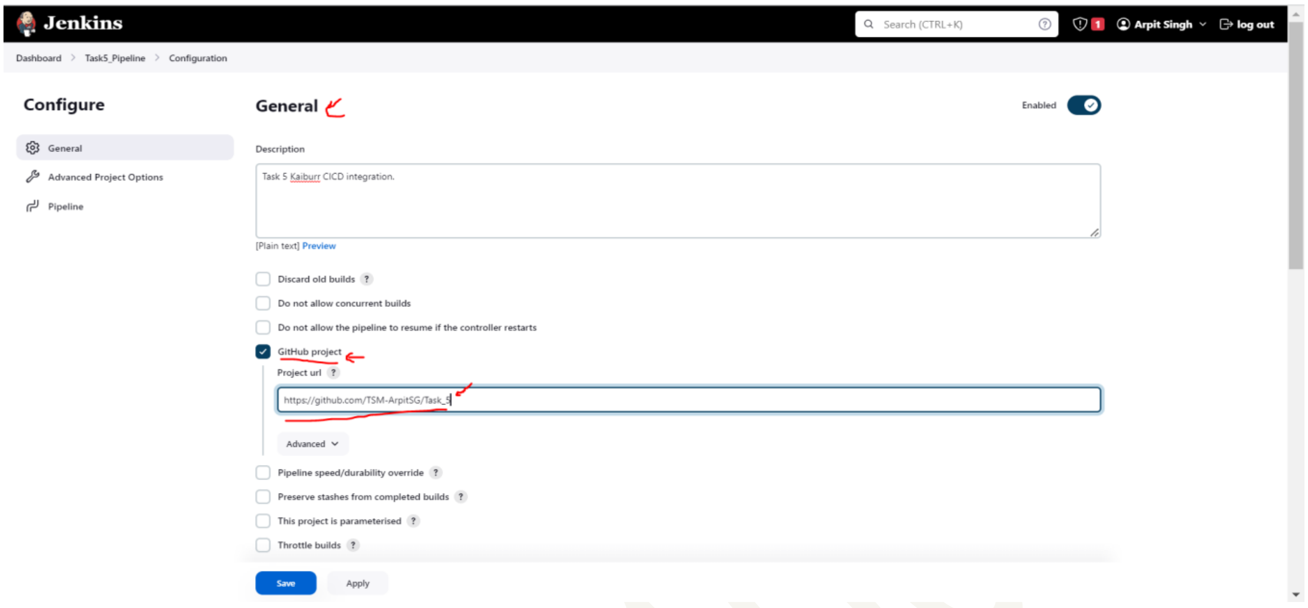
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
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.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

Organization Folder
Creates a set of multibranch project subfolders by scanning for repositories.

iv. Next "***check/mark github project***" under the '***general***' section and put the '***URL of our SampleApp***' - **https://github.com/TSM-ArpitSG/Task_5** .



Jenkins | Search (CTRL+K) | Arpit Singh | log out

Dashboard > Task5_Pipeline > Configuration

Configure
- General
- Advanced Project Options
- Pipeline

General — Enabled

Description

Task 5 Kaiburr CICD integration.

[Plain text] Preview

- [ ] Discard old builds
- [ ] Do not allow concurrent builds
- [ ] Do not allow the pipeline to resume if the controller restarts
- [x] GitHub project

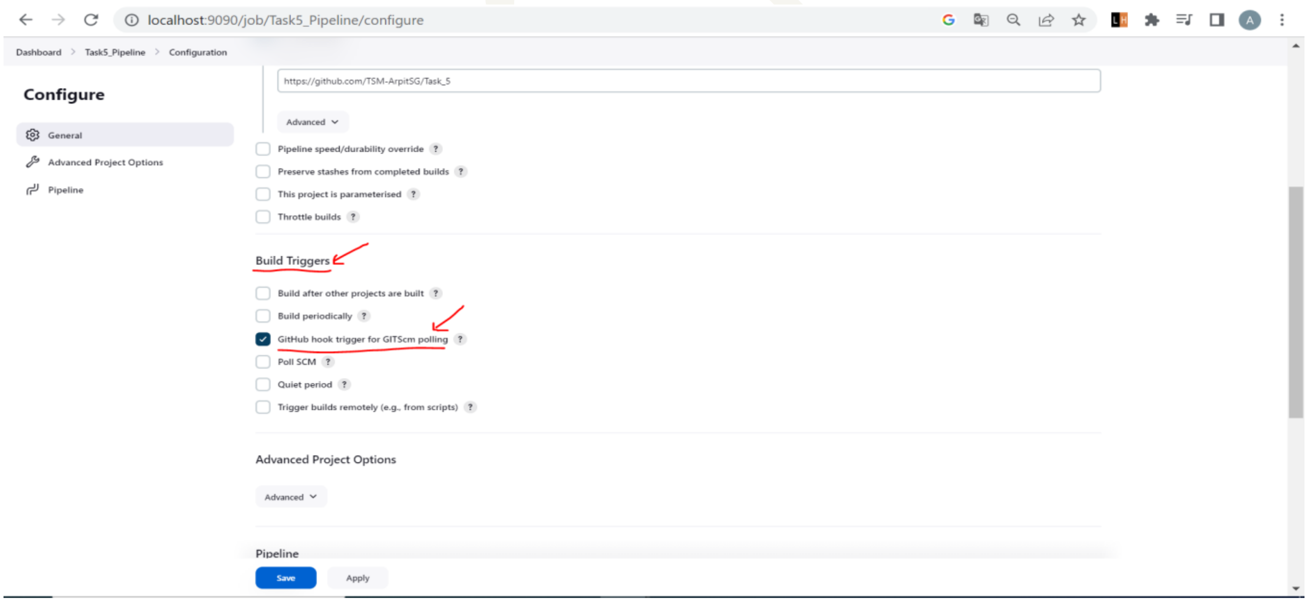
Project url: https://github.com/TSM-ArpitSG/Task_5

Advanced

- [ ] Pipeline speed/durability override
- [ ] Preserve stashes from completed builds
- [ ] This project is parameterised
- [ ] Throttle builds

Save | Apply

v. Under "***Triggers***" Section check/***mark "GitHub hook trigger for GITScm polling"***.



localhost:9090/job/Task5_Pipeline/configure

Dashboard > Task5_Pipeline > Configuration

Configure
- General
- Advanced Project Options
- Pipeline

https://github.com/TSM-ArpitSG/Task_5

Advanced

- [ ] Pipeline speed/durability override
- [ ] Preserve stashes from completed builds
- [ ] This project is parameterised
- [ ] Throttle builds

Build Triggers

- [ ] Build after other projects are built
- [ ] Build periodically
- [x] GitHub hook trigger for GITScm polling
- [ ] Poll SCM
- [ ] Quiet period
- [ ] Trigger builds remotely (e.g., from scripts)

Advanced Project Options

Advanced

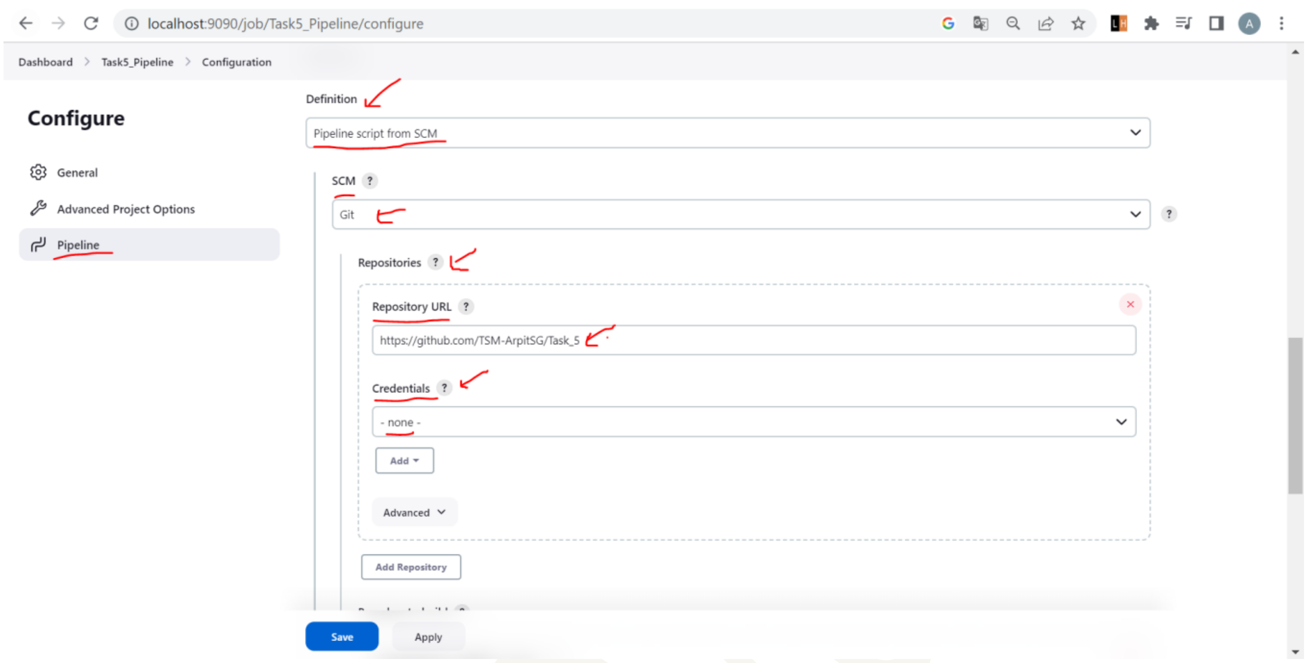
Pipeline

Save | Apply

vi. <u>(IMP) Go to the</u> "**<u>Pipeline</u>**" Section:

1. Within the "***Definition***" Dropdown select " ***Pipeline Script from SCM*** "
2. Within the "***SCM***" Dropdown select " ***GIT*** "

3. Within the "***Repository URL***" section put ***SampleApp repository*** " ***https://github.com/TSM-ArpitSG/Task_5*** "
4. ***Credentials*** are set to "***None***" as our Repository is "***Public***" (default).


localhost:9090/job/Task5_Pipeline/configure

Dashboard > Task5_Pipeline > Configuration

Configure

General
Advanced Project Options
Pipeline

Definition: Pipeline script from SCM

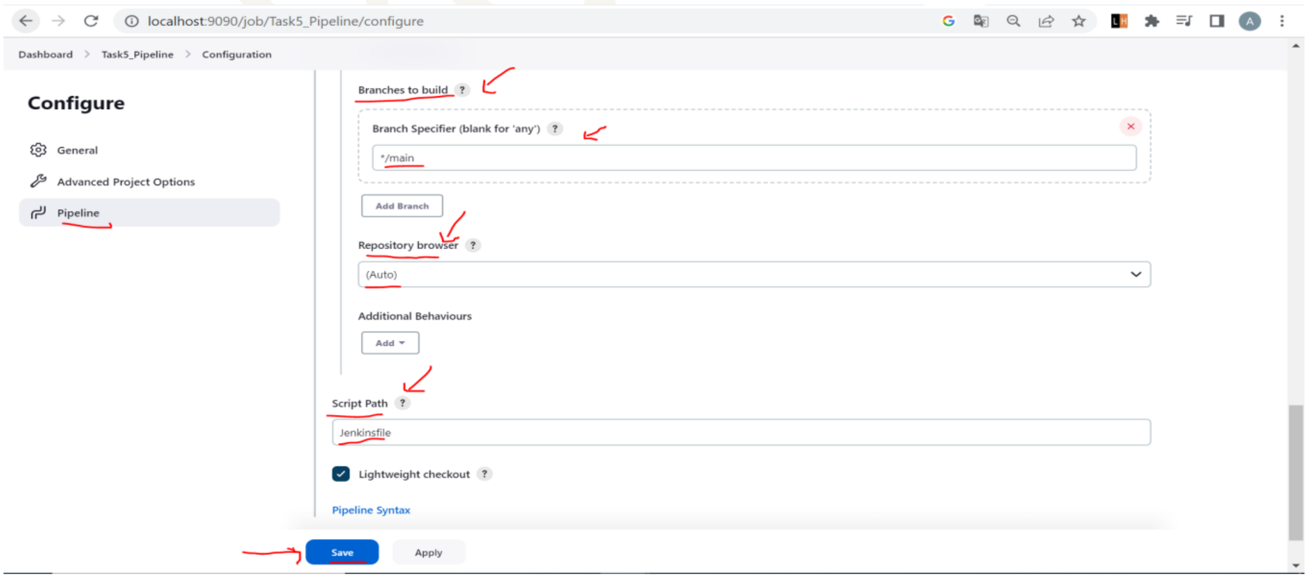
SCM: Git

Repositories

Repository URL: https://github.com/TSM-ArpitSG/Task_5

Credentials: - none -

5. Under "***Branches to Build***" Section put "***/main***" as ***branch specifier*** (not default).
6. ***Repository Browser*** is set to "***Auto***" (default).
7. Lastly, Set "***Script Path***" to the name of the ***Jenkins File*** we created earlier. (In my case "***JenkinsFile***").
8. Press "***Save***".


localhost:9090/job/Task5_Pipeline/configure

Dashboard > Task5_Pipeline > Configuration

Configure

General
Advanced Project Options
Pipeline

Branches to build

Branch Specifier (blank for 'any'): */main

Repository browser: (Auto)

Additional Behaviours

Script Path: Jenkinsfile

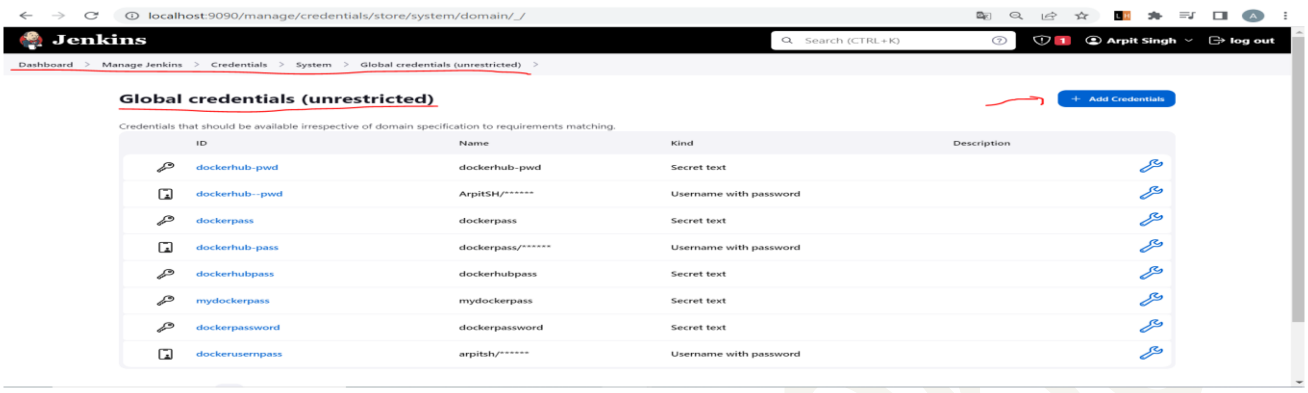
✓ Lightweight checkout

Jenkins

c. ***Configure and Add Docker Hub Password Credentials***:

   i. On the Jenkins ***home page***, click on "***Manage Jenkins***" on the left-hand menu and Scroll to the "***Security***" Section.

   ii. Click on "***Manage credentials***".



localhost:9090/manage/credentials/store/system/domain/_/

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

Global credentials (unrestricted)

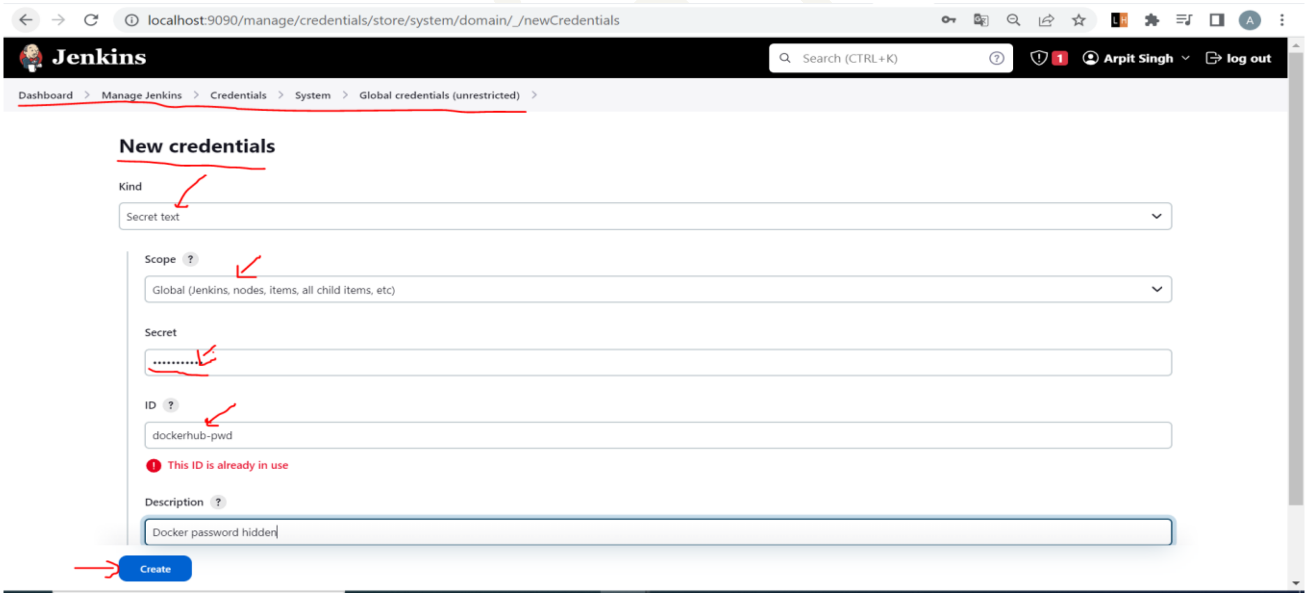
Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
dockerhub-pwd	dockerhub-pwd	Secret text	
dockerhub--pwd	ArpitSH/******	Username with password	
dockerpass	dockerpass	Secret text	
dockerhub-pass	dockerpass/******	Username with password	
dockerhubpass	dockerhubpass	Secret text	
mydockerpass	mydockerpass	Secret text	
dockerpassword	dockerpassword	Secret text	
dockerusernpass	arpitsh/******	Username with password	

   iii. Click on "***Add credentials***".

      1. Select "***Secret Text***" in the "***Kind***" Dropdown.
      2. ***Scope - Default***.
      3. Put your "***DockerHub Password***" in the "***Secret***" Section.
      4. Set "***ID***" as "***dockerhub-pwd***" (According to the ***credentialID*** we specified in our ***JenkinsFile*** script).

   iv. Click "***Create***".



localhost:9090/manage/credentials/store/system/domain/_/newCredentials

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

New credentials

Kind: Secret text

Scope: Global (Jenkins, nodes, items, all child items, etc)

Secret: ....

ID: dockerhub-pwd

This ID is already in use

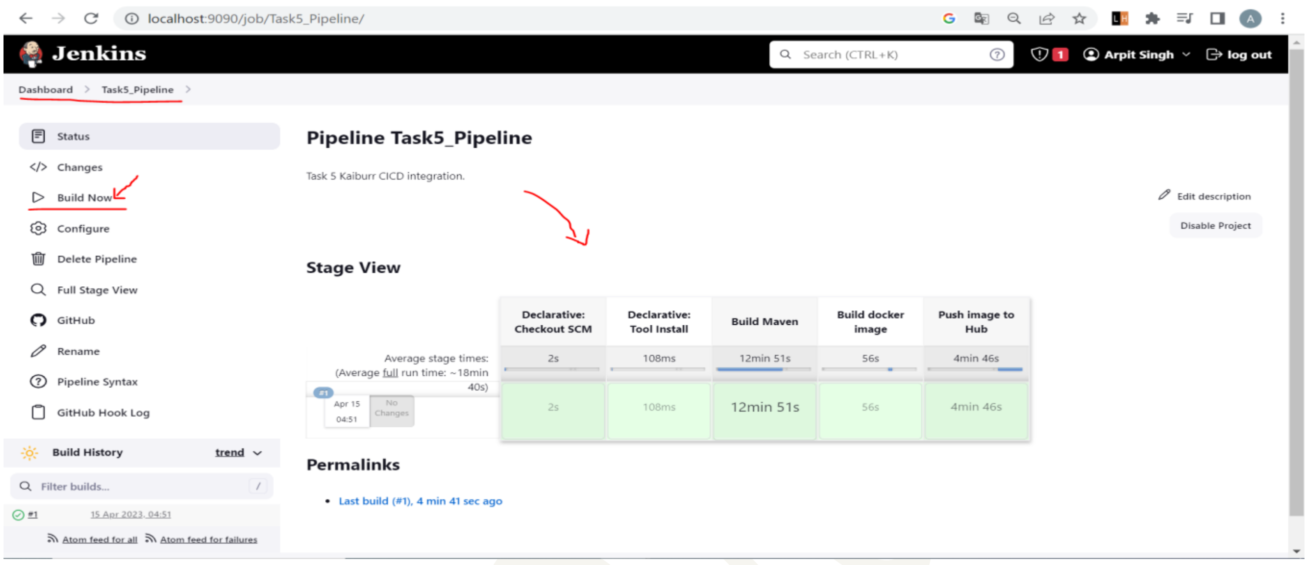
Description: Docker password hidden

Basically with these steps we '***create a secret text'*** that ***stores our docker hub password*** and that is ***not visible directly*** in the script rather it is ***accessed in the script with the help of ID***. I used this to login to my Docker Hub and push a docker image.

```
withCredentials([string(credentialsId: 'dockerhub-pwd', variable: 'dockerhubpwd')]) {
   bat 'docker login -u arpitsh -p ${dockerhubpwd}'
}
```

- With this "***Jenkins CICD***" is all ***set up and ready*** to automate the process of ***integration and deployment(Build now)***.

8. **Build Now**:
   a. Go to your Pipeline. (***Task5_Pipeline***)
   b. From the left menu, Click "***Build now***".



Pipeline Task5_Pipeline

Task 5 Kaiburr CICD integration.

Stage View

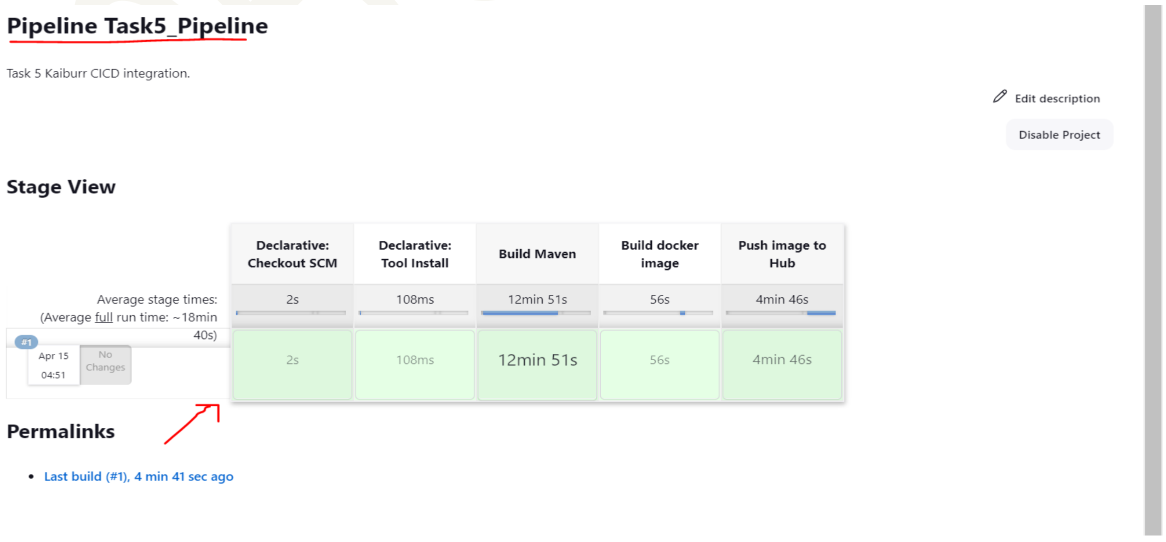
	Declarative: Checkout SCM	Declarative: Tool Install	Build Maven	Build docker image	Push image to Hub
Average stage times: (Average full run time: ~18min 40s)	2s	108ms	12min 51s	56s	4min 46s
#1 Apr 15 04:51 No Changes	2s	108ms	12min 51s	56s	4min 46s

Permalinks

- Last build (#1), 4 min 41 sec ago

❖ That's it! You have successfully Created a CICD Pipeline for a Sample Maven Application. With this you can build the project and create/push docker images using Jenkins.

- **Build/Output**:
  - "***Build Now***" - After Clicking this you should see the following:



Pipeline Task5_Pipeline

Task 5 Kaiburr CICD integration.

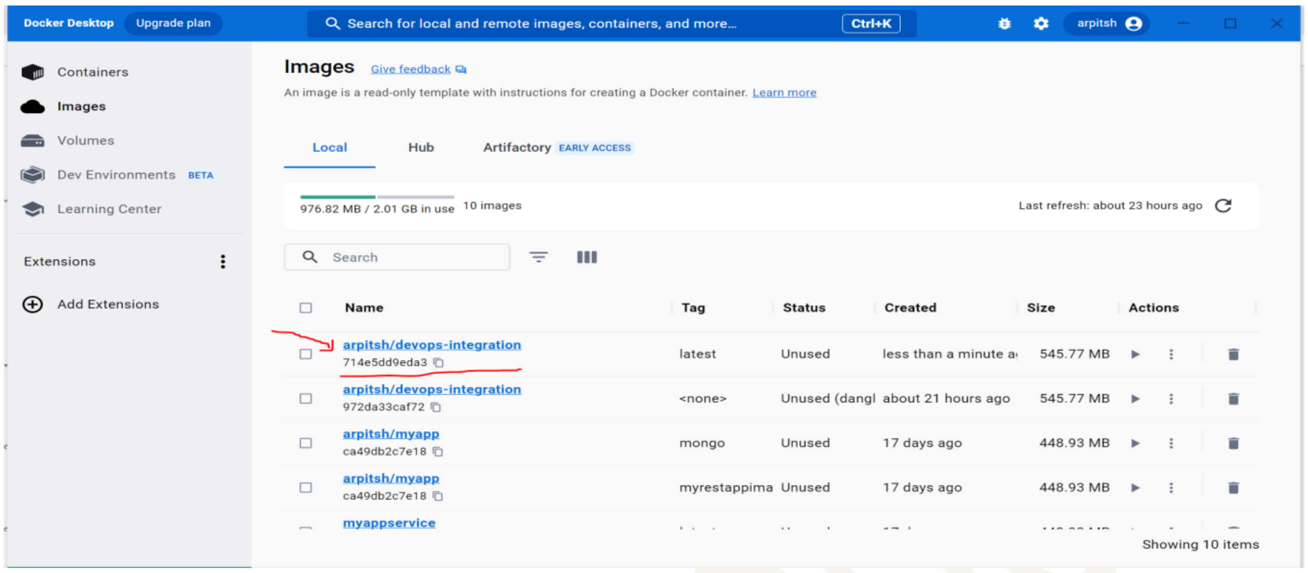
Stage View

	Declarative: Checkout SCM	Declarative: Tool Install	Build Maven	Build docker image	Push image to Hub
Average stage times: (Average full run time: ~18min 40s)	2s	108ms	12min 51s	56s	4min 46s
#1 Apr 15 04:51 No Changes	2s	108ms	12min 51s	56s	4min 46s

Permalinks

- Last build (#1), 4 min 41 sec ago

- ***“Docker Image Creation”***- After Build the Docker image “***arpitsh/devops-integration***” will automatically ***added(locally)***.



Docker Desktop Upgrade plan Search for local and remote images, containers, and more... Ctrl+K arpitsh

Containers
Images
Volumes
Dev Environments BETA
Learning Center
Extensions
Add Extensions

Images Give feedback

An image is a read-only template with instructions for creating a Docker container. Learn more

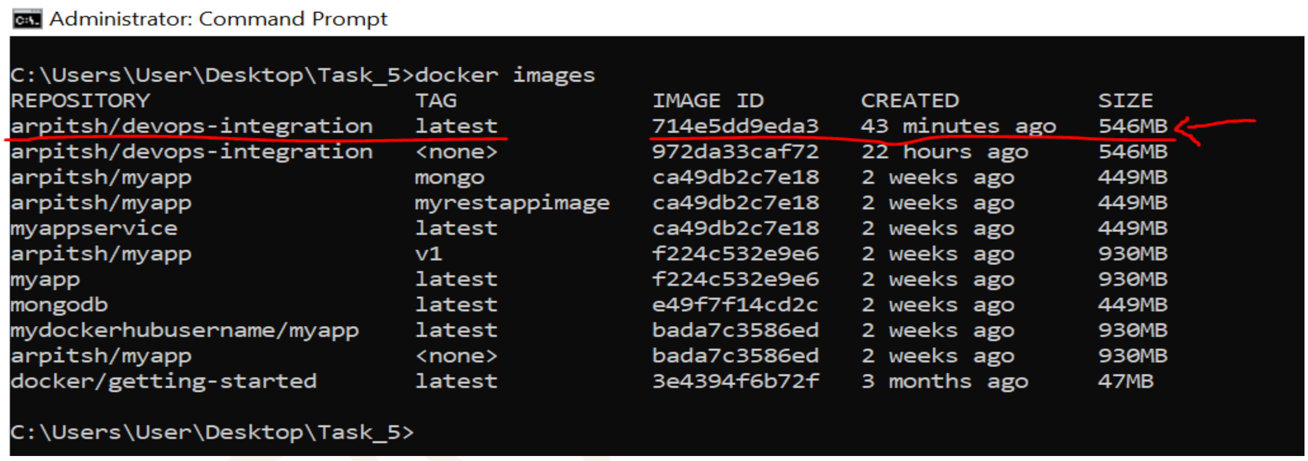
Local Hub Artifactory EARLY ACCESS

976.82 MB / 2.01 GB in use 10 images Last refresh: about 23 hours ago

Search

Name	Tag	Status	Created	Size	Actions
arpitsh/devops-integration 714e5dd9eda3	latest	Unused	less than a minute a	545.77 MB	
arpitsh/devops-integration 972da33caf72	<none>	Unused (dangl	about 21 hours ago	545.77 MB	
arpitsh/myapp ca49db2c7e18	mongo	Unused	17 days ago	448.93 MB	
arpitsh/myapp ca49db2c7e18	myrestappima	Unused	17 days ago	448.93 MB	
myappservice					

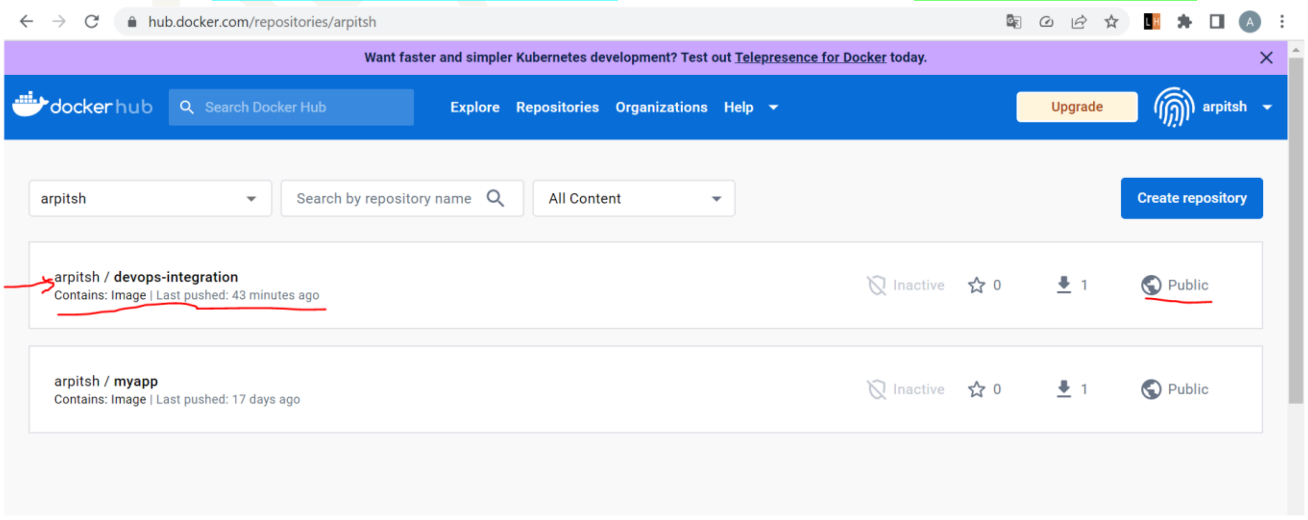
Showing 10 items



Administrator: Command Prompt

```
C:\Users\User\Desktop\Task_5>docker images
REPOSITORY                    TAG              IMAGE ID       CREATED          SIZE
arpitsh/devops-integration    latest           714e5dd9eda3   43 minutes ago   546MB
arpitsh/devops-integration    <none>           972da33caf72   22 hours ago     546MB
arpitsh/myapp                 mongo            ca49db2c7e18   2 weeks ago      449MB
arpitsh/myapp                 myrestappimage   ca49db2c7e18   2 weeks ago      449MB
myappservice                  latest           ca49db2c7e18   2 weeks ago      449MB
arpitsh/myapp                 v1               f224c532e9e6   2 weeks ago      930MB
myapp                         latest           f224c532e9e6   2 weeks ago      930MB
mongodb                       latest           e49f7f14cd2c   2 weeks ago      449MB
mydockerhubusername/myapp     latest           bada7c3586ed   2 weeks ago      930MB
arpitsh/myapp                 <none>           bada7c3586ed   2 weeks ago      930MB
docker/getting-started        latest           3e4394f6b72f   3 months ago     47MB

C:\Users\User\Desktop\Task_5>
```

- ***Docker Image Uploaded to Hub”***After Build the Docker image “***arpitsh/devops-integration***”will automatically ***deployed(globally)***.



hub.docker.com/repositories/arpitsh

Want faster and simpler Kubernetes development? Test out Telepresence for Docker today.

docker hub Search Docker Hub Explore Repositories Organizations Help Upgrade arpitsh

arpitsh Search by repository name All Content Create repository

arpitsh / devops-integration
Contains: Image | Last pushed: 43 minutes ago
Inactive 0 1 Public

arpitsh / myapp
Contains: Image | Last pushed: 17 days ago
Inactive 0 1 Public

Summary of the steps to create a CICD Application using Jenkins (Task 5):

1. Install Jenkins on your machine or server.
2. Install the Docker Plugin and Pipeline Plugin in Jenkins.
3. Create a repository for your sample application on GitHub or any other code hosting service.
4. Push your sample application code to the repository.
5. In Jenkins home page, click on "New Item" to create a new pipeline.
6. Enter a name for your pipeline and select "Pipeline" as the project type.
7. In the "Pipeline" section, select "Pipeline script from SCM" as the definition.
8. Select "Git" as the SCM and provide the repository URL.
9. Under "Script Path", enter the path to your Jenkinsfile (e.g., Jenkinsfile).
10. Create a Jenkinsfile with the necessary stages for your application.
11. Configure Docker credentials by adding your Docker Hub username and password as "Username with password" credentials in Jenkins.
12. Save the credentials.
13. Run the Jenkins pipeline to build, test, build a Docker image, publish it to Docker Hub, and deploy it to your production environment.

Explanation on Tools/Technology(s) Used:

❖ **Docker**:

a. The purpose of using Docker in ***this project*** is to containerize the application and its dependencies, making it easy to deploy and run consistently across different environments. It also allows for easy scaling of the application by running multiple instances of the container on a single machine or across multiple machines. Additionally, using Docker enables separation of concerns between application developers and infrastructure operators, making it easier to manage the application and its infrastructure separately.

❖ **<u>Jenkins</u>**:

a. The purpose of using Jenkins in ***this project*** is to create a Continuous Integration/Continuous Deployment (CI/CD) pipeline. The pipeline is defined using a Jenkinsfile, which includes steps for checking out the code, building and testing the application, building a Docker image, publishing the image to a Docker registry, and deploying the image to a production environment.

b. By using Jenkins, the process of building, testing, and deploying the application is automated and streamlined, which can save time and reduce errors. The pipeline can be triggered automatically when changes are made to the code repository, ensuring that the application is always up-to-date and the latest version is deployed to production.

❖ **<u>Maven</u>**:

a. Maven is a build automation tool used to manage dependencies and build Java projects. ***In this project***, Maven is used to build and package the Java application code. The "mvn clean package" command is used in the Jenkins pipeline to build and package the application code into a JAR file that can be used to run the application.

❖ <u>**IntelliJ**</u>:

a. IntelliJ is an integrated development environment (IDE) that provides a range of features to help developers create, test, and deploy software applications more efficiently. ***In this project***, IntelliJ has been used to write, test, and debug the code for the sample application. IntelliJ can also help manage project dependencies and integrate with build tools like Maven to automate the build process.

Thank You!