

Registration form:

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Event Registration Form</title>
    <style>
      body {
        font-family: Arial, sans-serif;
        margin: 50px;
      }
      form {
        max-width: 400px;
        margin: auto;
        padding: 20px;
        border: 1px solid #ccc;
        border-radius: 10px;
      }
      label, input, select, button {
        display: block;
        width: 100%;
        margin-bottom: 10px;
      }
      input, select {
        padding: 8px;
        border: 1px solid #ccc;
        border-radius: 5px;
      }
    </style>
  </head>
  <body>
    <form>
      <label>Name <input type="text"/></label>
      <label>Email <input type="email"/></label>
      <label>Phone <input type="tel"/></label>
      <label>Address <input type="text"/></label>
      <label>City <input type="text"/></label>
      <label>State / Province <input type="text"/></label>
      <label>Zip / Postcode <input type="text"/></label>
      <label>Country <input type="text"/></label>
      <label>Event Type <input type="text"/></label>
      <label>Event Date <input type="text"/></label>
      <label>Event Time <input type="text"/></label>
      <label>Number of Guests <input type="text"/></label>
      <label>Message <input type="text"/></label>
      <button type="submit">Register</button>
    </form>
  </body>
</html>
```

```
button {  
    padding: 10px;  
    background-color: #28a745;  
    color: #fff;  
    border: none;  
    border-radius: 5px;  
    cursor: pointer;  
}  
  
button:hover {  
    background-color: #218838;  
}  
</style>  
</head>  
<body>  
    <h2>Event Registration Form</h2>  
    <form id="registrationForm" onsubmit="return validateForm()">  
        <label for="name">Full Name</label>  
        <input type="text" id="name" name="name" required>  
  
        <label for="email">Email Address</label>  
        <input type="email" id="email" name="email" required>  
  
        <label for="phone">Phone Number</label>  
        <input type="tel" id="phone" name="phone" required pattern="[0-9]{10}">  
  
        <label for="event">Select Event</label>  
        <select id="event" name="event" required>  
            <option value="">--Select an Event--</option>  
            <option value="workshop">Workshop</option>  
            <option value="seminar">Seminar</option>  
    </form>  
</body>
```

```
<option value="conference">Conference</option>
</select>

<button type="submit">Register</button>
</form>

<script>
    function validateForm() {
        const form = document.getElementById('registrationForm');
        const name = form.name.value.trim();
        const email = form.email.value.trim();
        const phone = form.phone.value.trim();
        const event = form.event.value;

        if (!name || !email || !phone || !event) {
            alert('Please fill out all fields.');
            return false;
        }

        if (!/^\d{10}$/.test(phone)) {
            alert('Please enter a valid 10-digit phone number.');
            return false;
        }

        alert('Registration Successful!');
        return true;
    }
</script>
</body>
</html>
```

Output:

### Event Registration Form

Full Name

Email Address

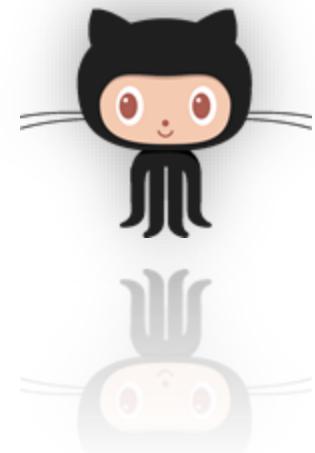
Phone Number

Select Event

-Select an Event-

**Register**

# GIT AND GITHUB



# GIT



- Git is a free and open source *distributed version control system* designed to handle everything from small to very large projects with speed and efficiency.
- Git is used to **track changes in the source code**, enabling multiple developers to work together on non-linear development.
- Linus Torvalds created Git in 2005 for the development of the Linux kernel.

# GIT



- Git is a tool that helps you manage changes to your code over time.
- It keeps track of every little change or update you make, you can always look back at previous versions or undo mistakes if needed.
- GIT is a CLI (Command Line Interface) based tool which is used for taking snapshots of each and every version of your code.

# VERSION CONTROL SYSTEM



- A **version control system (VCS)** is like a smart tool that helps people work together on projects, especially computer programs.
- It keeps track of changes made to files, kind of like saving different versions of a document.

# WHAT DOES GIT DO



- Manage projects with **Repositories**
- **Clone** a project to work on a local copy
- Control and track changes with **Staging** and **Committing**
- **Branch** and **Merge** to allow for work on different parts and versions of a project
- **Pull** the latest version of the project to a local copy
- **Push** local updates to the main project

# Features of Git



## Version Control System:

- Git keeps track of every change you make to your project files.
- You can go back to previous versions if any requirement arises.

## • Repositories:

- A Git repository (or repo) is like a project's central hub where everything related to your work is stored and managed. There are two main types:

## • Local Repository:

- This is a copy of the repository which is stored in your computer.
- You can work on your project and make changes here.

## • Remote Repository:

- This is stored on a server, like GitHub, where you and others can share and work on a project.

# Features of Git



**Commits:** Every time you make changes and save them in Git, these are called commits. The repository keeps track of all the commits you have made.

**Branches:** A repository allows you to create Branches and work on new features and fixes. For example, you can have a “main” branch for stable code and a “feature” branch for new features you’re developing.

# Features of Git

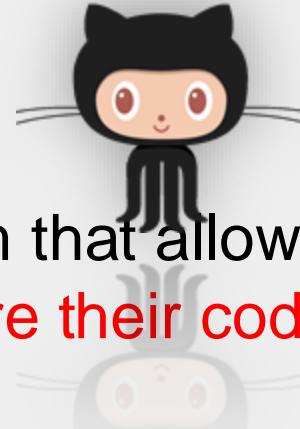


**Merging:** Once you are done with the branches you have made, you can merge those branches into main branch. This adds the changes you have made in the rest of the project.

**Cloning:** Cloning is a process of making a complete copy of Git repository.

- It's like copying the entire project from central location ( like website) to your own computer.

# GITHUB



**GitHub** is a developer platform that allows developers to create, store, manage and share their code.

It uses Git software, providing the distributed version control of Git plus access control, bug tracking, software feature requests, task management, continuous integration

# Local

# Remote

working  
directory

staging  
area

localrepo

remote  
repo

git add

git commit

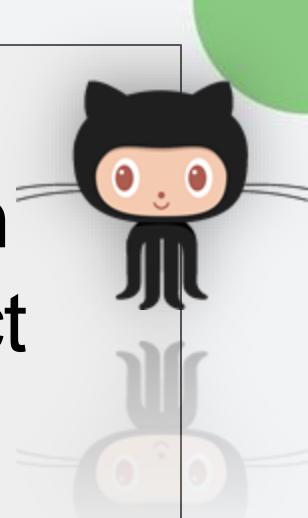
git push

git pull

git checkout

Git **working directory** is the directory on your local filesystem where your project files are located.

It contains all the files and folders in your project, including those that are tracked by Git and those that are not.



Git's **staging area**, also known as **the index or cache**, is an intermediate area that **stores changes to tracked files** before they are committed to the repository.

It serves as a way to organize and review the changes you have made to your project before creating a new commit

# Downloading and Installing Git

For windows version

<https://git-scm.com/download/win>

## Week -2

### Second Half

#### 1) version

command : git --version

#### 2) config

Configuring user information used across all local repositories

git config --global user.name “[firstname lastname]”

git config --global user.email “[valid-email]”

#### 3) init

The command is used to create an empty Git repository.

git init

initialize an existing directory as a Git repository

git init myworkspace

#### 4) add

This command is used to add one or more files to staging (Index) area.

git add Filename

git add\*

#### 5) commit

The commit command makes sure that the changes are saved to the local repository.

git commit -m “commit message”

#### 6)status

The status command is used to display the state of the working directory and the staging area.

It allows you to see which changes have been staged, which haven't, and which files aren't being tracked by Git

```
git status
```

7) log---Checking the commit history:

```
git log
```

8) clone

The git clone command is used to create a local working copy of an existing remote repository.

The command downloads the remote repository to the computer.

```
git clone <remote_URL>
```

9) Creating a new branch:

```
git branch <branch-name>
```

10) Switching to a different branch:

```
git checkout <branch-name>
```

11) Merging two branches:

```
git merge <branch-name>
```

12) push

The command git push is used to transfer the commits or pushing the content from the local repository to the remote repository.

```
git push -u origin <branchname>
```

13) pull

The git pull command is used to fetch and merge changes from the remote repository to the local repository.

```
git pull <branch_name> <remote URL>
```

# Jenkins



**Jenkins** – an *open source automation server* which enables developers around the world to **reliably build, test, and deploy** their software.

**Jenkins** provides **hundreds of plugins** to support building, deploying and automating any project

# Jenkins



Jenkins is an **open source continuous integration (CI) server.**

It **manages and controls** several stages of the software delivery process, including build, documentation, automated testing, packaging, and static code analysis.

Jenkins is a highly popular DevOps tool used by thousands of development teams.

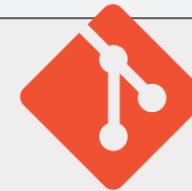
# Jenkins



Jenkins automation is commonly triggered by code changes in repositories like GitHub, Bitbucket, and GitLab, and integrates with build tools like Maven and Gradle.

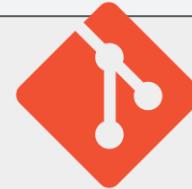
Jenkins supports the use of container technologies like Docker and Kubernetes for testing and packaging of software releases

# Jenkins



- **Continuous Integration and Continuous Delivery**
- **Easy installation**
- **Easy configuration**
- **Plugins**
- **Extensible**
- **Distributed**

# Installing Jenkins



## Prerequisites

Recommended hardware configuration for a small team:

- 4 GB+ of RAM
- 50 GB+ of drive space

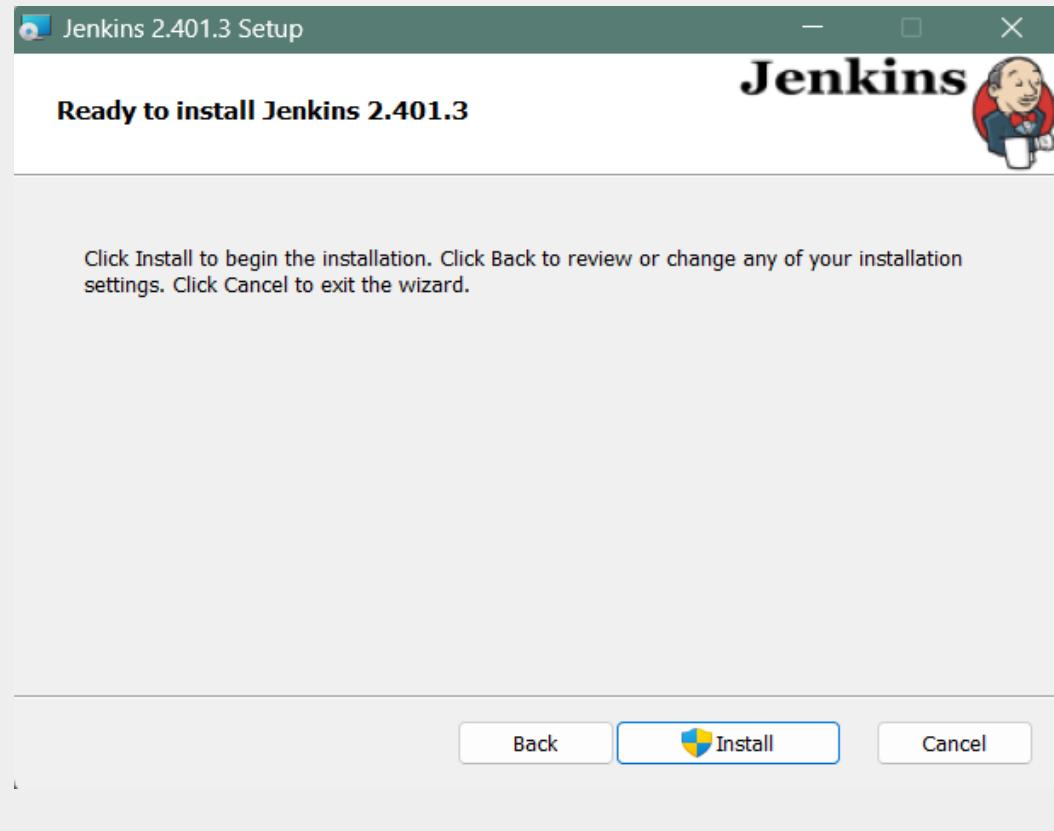
# Installing Jenkins



- Installation steps using Windows MSI installer
- <https://www.jenkins.io/download/#downloading-jenkins>
- After the download completes, open the Windows installer and follow the steps below to install Jenkins.

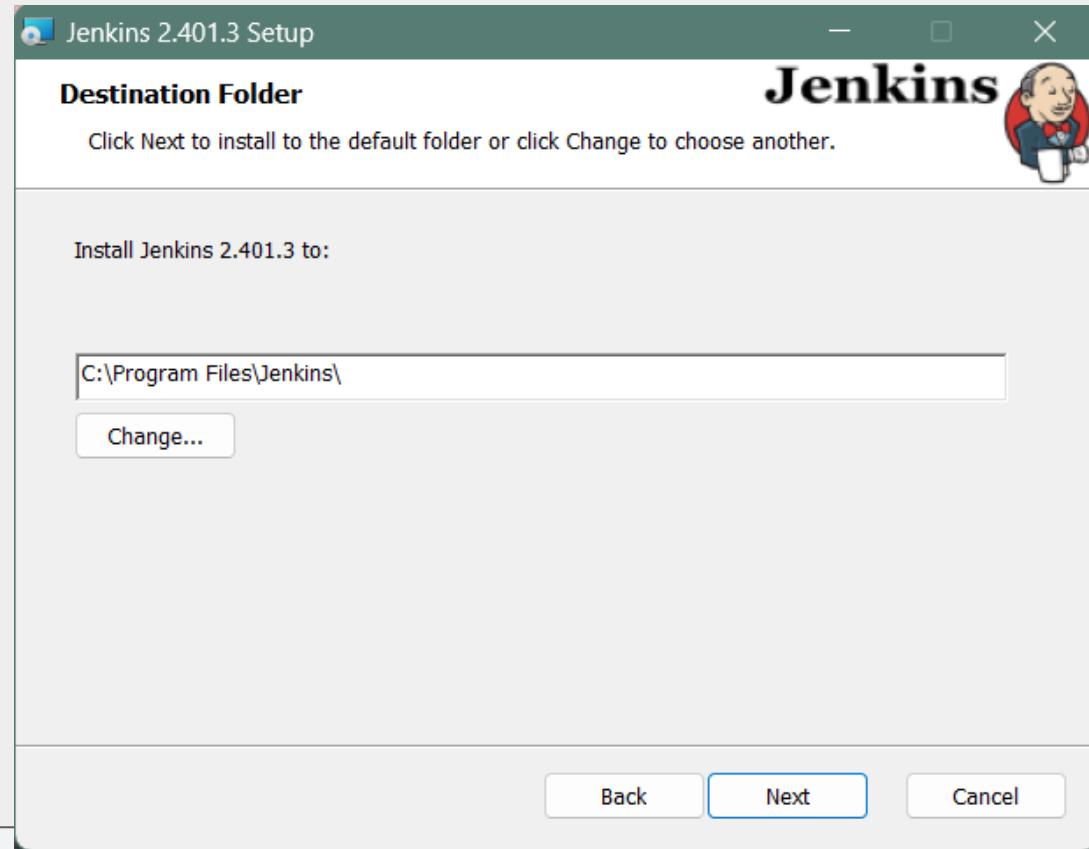
# Installing Jenkins

- Step 1:
- Setup wizard



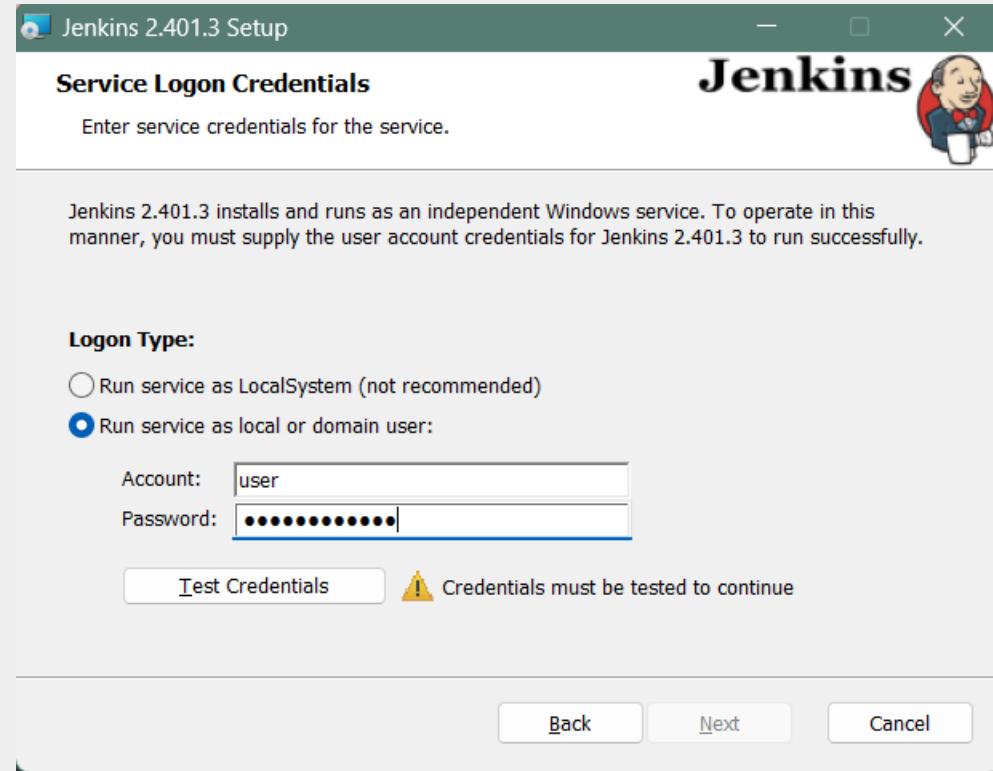
# Installing Jenkins

Select destination folder



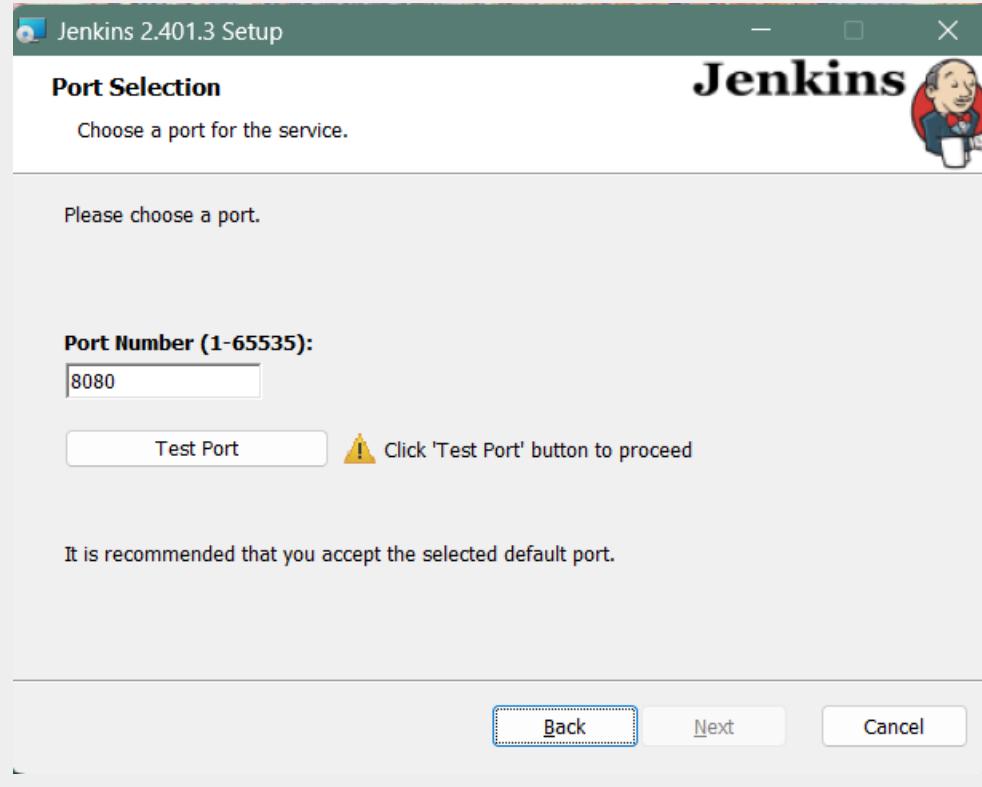
# Installing Jenkins

## Step 3: Service logon credentials



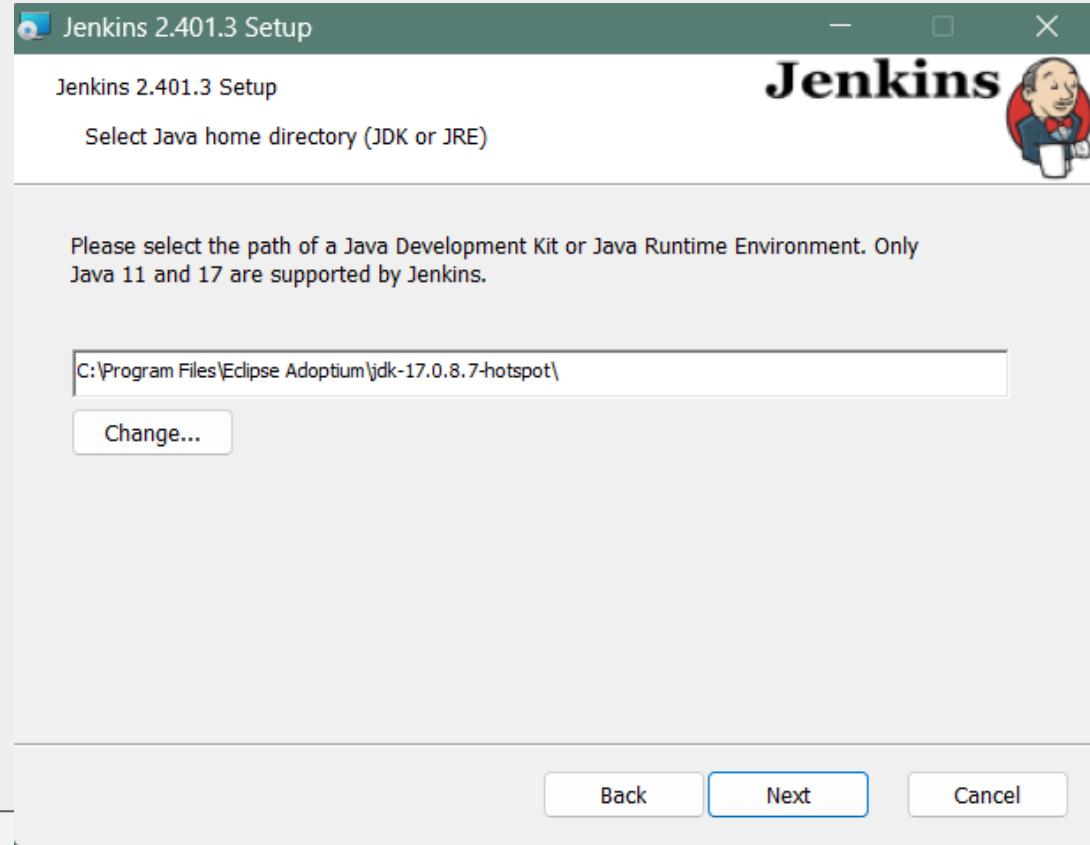
# Installing Jenkins

## Step 4: Port selection



# Installing Jenkins

Step 5:  
Select  
Java home  
directory:



# Step 6: Custom setup

## Installing Jenkins

Jenkins 2.401.3 Setup

### Custom Setup

Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

The screenshot shows the Jenkins setup interface. At the top, it says "Jenkins 2.401.3 Setup". On the right, there's a cartoon Jenkins character holding a coffee cup. Below the title, it says "Custom Setup" and "Select the way you want features to be installed". A sub-instruction "Click the icons in the tree below to change the way features will be installed." is present. A tree view on the left shows "Jenkins" expanded, with "Start Service" and "Firewall Exception" under it. "Firewall Exception" is highlighted with a blue border. To the right of the tree, a detailed description of the "Firewall Exception" feature is provided: "Enables a firewall exception for the Java running Jenkins on port 8080 (not recommended)." It also states that "This feature requires 0KB on your hard drive." At the bottom, there are buttons for "Reset", "Disk Usage", "Back", "Next" (which is highlighted in blue), and "Cancel".

Enables a firewall exception for the Java running Jenkins on port 8080 (not recommended).

This feature requires 0KB on your hard drive.

Browse...

Reset Disk Usage Back **Next** Cancel

# Installing Jenkins

## Step 7: Install Jenkins



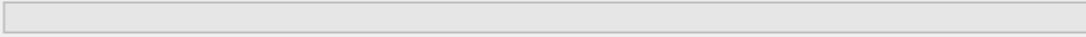
# Jenkins



## Installing Jenkins 2.401.3

Please wait while the Setup Wizard installs Jenkins 2.401.3.

Status: Copying new files File: [1], Directory: [9], Size: [6]



Back

Next

Cancel

# Installing Jenkins

Step 8: Finish Jenkins installation



# Installing Jenkins

## Post Setup wizard

### Unlocking Jenkins

#### Step 1

Browse to `http://localhost:8080` and wait until the  
**Unlock Jenkins page appears.**

# Installing Jenkins

Getting Started

## Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

C:\Program Files (x86)\Jenkins\secrets\initialAdminPassword

Please copy the password from either location and paste it below.

Administrator password

Continue

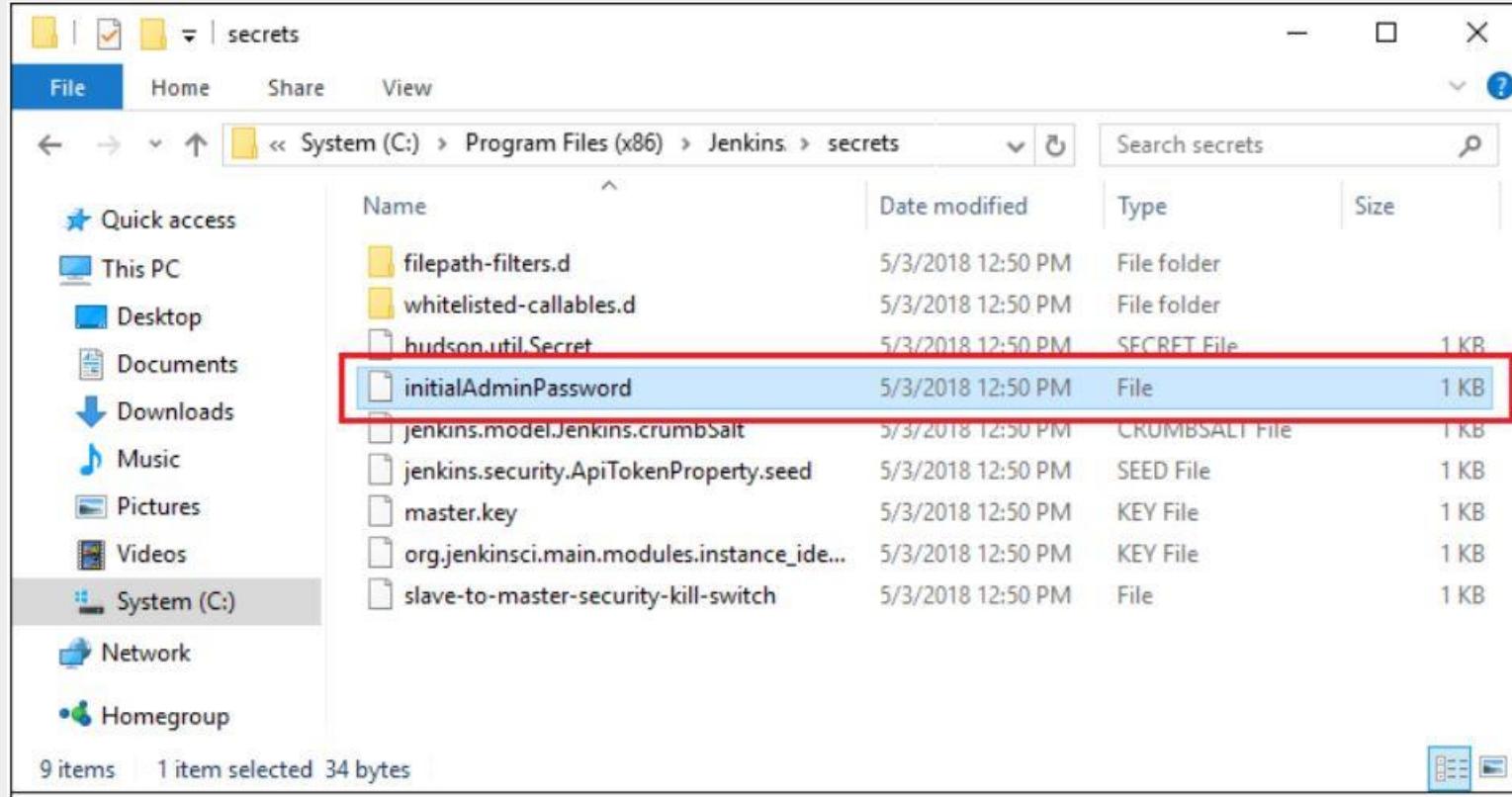
# Installing Jenkins

## Step 2

The initial Administrator password  
should be found under the Jenkins installation path

If a custom path for Jenkins installation was selected,  
then you should check that location for initialAdminPassword file.

# Installing Jenkins



# Installing Jenkins

## Step 3

Open the highlighted file and copy the content of the **initialAdminPassword** file.

# Installing Jenkins

## Step 4

On the **Unlock Jenkins** page,  
paste this password into the **Administrator password** field and  
click **Continue**.

## **Exploring Jenkins Environment**

- 1. Search Bar:** The search bar allows users to quickly find specific items within Jenkins, such as **jobs, nodes, users, or plugins**. It facilitates navigation and helps locate items efficiently, especially in larger Jenkins installations.
- 2. Navigation Bar:** The navigation bar provides access to various sections and functionalities within Jenkins. It typically includes links to items like "New Item" (to create new jobs), "People" (to view user accounts), "Build History" (to track previous builds), "Manage Jenkins" (to configure Jenkins settings), and more.
- 3. New Item:** This option allows users **to create new items or projects within Jenkins**. Users can create different types of items, such as **freestyle projects, pipeline projects, multi-configuration projects**, and more, depending on their requirements.
- 4. Build History:** The Build History section lists the **recent builds of jobs within Jenkins**. Users can view details about **each build, including build status, duration, and any associated changes or comments**
- 5. Manage Jenkins:** It provides access to **Jenkins administration and management functionalities**. Users can configure global Jenkins settings, manage plugins, view system information, and perform other administrative tasks from this section.
- 6. My View:** My View allows users to customize their Jenkins dashboard to display relevant information based on their preferences. Users can **create custom views to filter and organize Jenkins jobs, builds, or other items according to specific criteria**.

**7. Build Queue:** The Build Queue section displays the list of jobs waiting to be executed in Jenkins. It provides information about queued jobs, including their priority, estimated waiting time, and any associated parameters.

**8. Build Execution Status:** This section displays the real-time status of ongoing builds in Jenkins. Users can monitor the progress of builds, view console output, and troubleshoot any issues that may arise during the **build process**.

**9. Create Job:** The Create Job option allows users to create new jobs or projects within Jenkins. *Users can define job configurations, specify source code repositories, configure build triggers, define build steps, and set post-build actions using this feature.*

## **Build Job**

A Jenkins build job contains the **configuration for automating a specific task or step in the application building process**

These tasks include **gathering dependencies, compiling, archiving, or transforming code, and testing and deploying code in different environments.**

Jenkins supports several **types of build jobs, such as freestyle projects, pipelines, multi-configuration projects, folders, multibranch pipelines, and organization folders.**

## **Jenkins Freestyle Project**

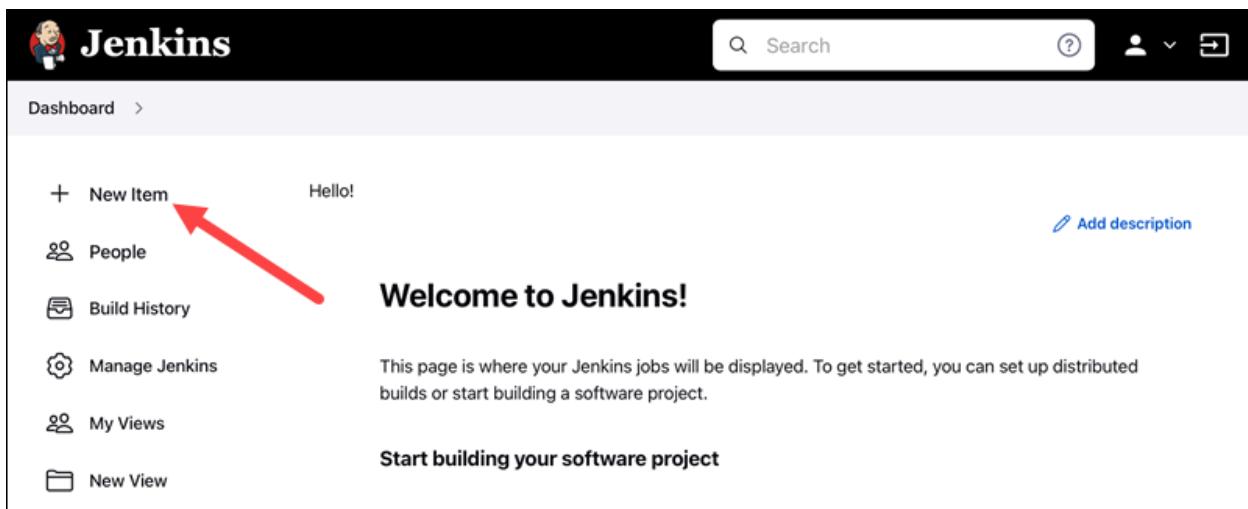
**Jenkins freestyle projects allow users to automate simple jobs, such as running tests, creating and packaging applications, producing reports, or executing commands.**

## **Set up a Build Job in Jenkins**

**Follow the steps outlined below to set up and run a new Jenkins freestyle project.**

### **Step 1: Create a New Freestyle Project**

**1. Click the New Item link on the left-hand side of the Jenkins dashboard.**



**2. Enter the new project's name in the Enter an item name field and select the Freestyle project type. Click OK to continue.**

**Enter an item name**

1  » Required field

2

**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.

**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, 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.

3

3. Under the *General* tab, add a project description in the Description field.

## General

Enabled

Description

This is a simple example of a Jenkins freestyle project, displaying the current version of Java.

[Plain text] [Preview](#)

- Commit agent's Docker container ?
- Define a Docker template
- Discard old builds ?
- GitHub project
- This project is parameterized ?
- Throttle builds ?
- Prepare an environment for the run ?
- Execute concurrent builds if necessary ?

## Step 2: Add a Build Step

1. Scroll down to the *Build* section.
2. Open the Add build step drop-down menu and select Execute Windows batch command.

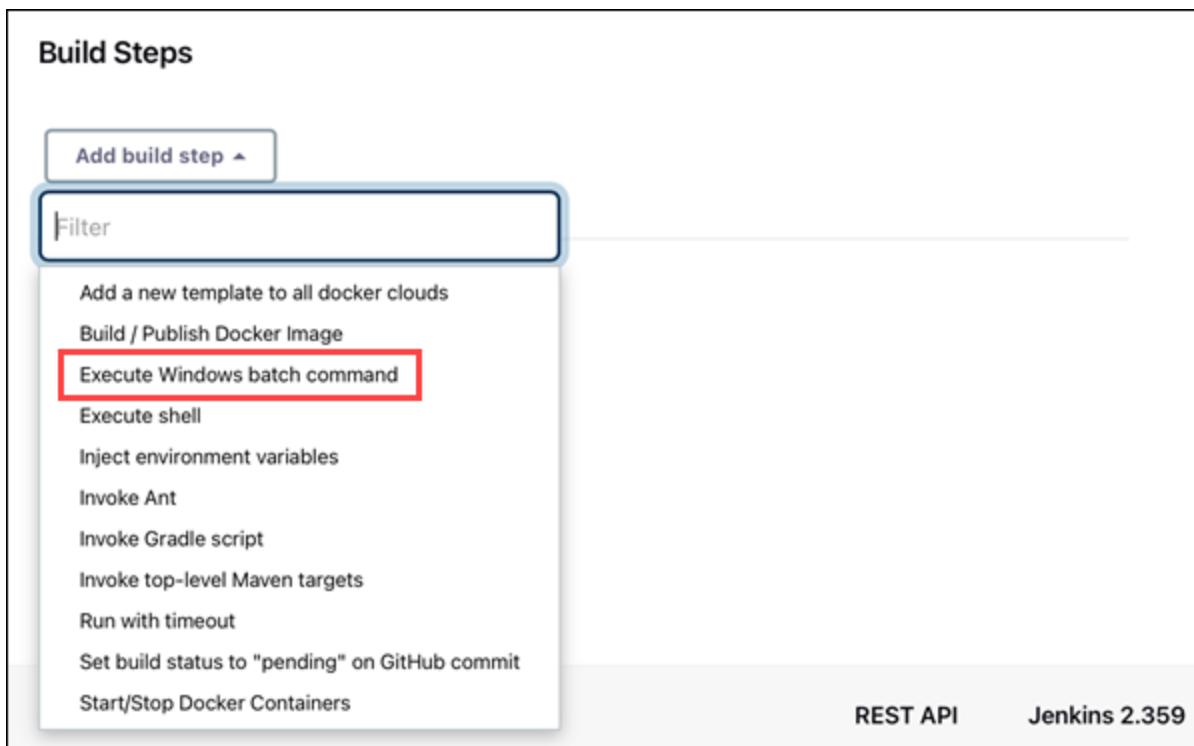
**Build Steps**

Add build step ▾

Filter

- Add a new template to all docker clouds
- Build / Publish Docker Image
- Execute Windows batch command**
- Execute shell
- Inject environment variables
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- Run with timeout
- Set build status to "pending" on GitHub commit
- Start/Stop Docker Containers

REST API Jenkins 2.359



**3. Enter the commands you want to execute in the Command field.**

**java -version**

**dir**

**4. Click the Save button to save changes to the project.**

**Build Steps**

≡ Execute Windows batch command ? ×

Command

See [the list of available environment variables](#)

java -version  
dir|

[Advanced...](#)

[Add build step ▾](#)

Save Apply

## Step 3: Build the Project

1. Click the **Build Now** link on the left-hand side of the new project page.

↑ Back to Dashboard

### Project Freestyle Project Example

This is a simple example of a Jenkins freestyle project, displaying the current version of Java.

[Edit description](#) Disable Project

[Status](#) [Changes](#) [Workspace](#) [Build Now](#) [Configure](#) [Delete Project](#) [Rename](#)

[Workspace](#)

[Recent Changes](#)

**Permalinks**

2. Click the link to the latest project build in the *Build History* section.

A screenshot of the Jenkins Build History page. At the top left is a gear icon followed by the text "Build History". To its right is a dropdown menu labeled "trend" with a downward arrow. Below this is a search bar with the placeholder "Filter builds...". Underneath the search bar is a list item for "Build #1" with a green checkmark icon. A red arrow points to this item. To the right of the list item is a vertical sorting menu with three options: a top arrow, a middle arrow, and a bottom arrow. Below the list item is the date and time "Jul 19, 2022, 2:28 PM". At the bottom of the list item are two links: "Atom feed for all" and "Atom feed for failures".

3. Click the Console Output link on the left-hand side to display the output for the commands you entered.

A screenshot of the Jenkins Build #1 details page. At the top center is the text "Build #1 (Jul 19, 2022, 2:28:32 PM)" with a green checkmark icon. To the right is a button labeled "Keep this build forever". On the far right, it says "Started 34 sec ago" and "Took 0.63 sec". Below the main title are several navigation links: "Back to Project", "Status" (which is highlighted with a grey background), "Changes", "Console Output" (which has a red arrow pointing to it), "Edit Build Information", "Delete build '#1'", and "Environment Variables". To the right of these links, there is a message "No changes." and a note "Started by user Marko Aleksic".

4. The console output indicates that Jenkins is successfully executing the commands, displaying the current version of Java and Jenkins working directory.



## Console Output

```
Started by user unknown or anonymous
Running as SYSTEM
[EnvInject] - Loading node environment variables.
Building in workspace C:\ProgramData\Jenkins\.jenkins\workspace\Freestyle Project Example
[Freestyle Project Example] $ cmd /c call C:\WINDOWS\TEMP\jenkins8981346236355107413.bat

C:\ProgramData\Jenkins\.jenkins\workspace\Freestyle Project Example>java -version
java version "1.8.0_321"  
Java(TM) SE Runtime Environment (build 1.8.0_321-b07)  
Java HotSpot(TM) Client VM (build 25.321-b07, mixed mode, sharing)

C:\ProgramData\Jenkins\.jenkins\workspace\Freestyle Project Example>dir
Volume in drive C has no label.
Volume Serial Number is 3AB4-672E

Directory of C:\ProgramData\Jenkins\.jenkins\workspace\Freestyle Project Example

01/25/2022  10:23 AM    <DIR>          .
01/25/2022  10:23 AM    <DIR>          ..
              0 File(s)           0 bytes
              2 Dir(s)   25,359,540,224 bytes free

C:\ProgramData\Jenkins\.jenkins\workspace\Freestyle Project Example>exit 0
Finished: SUCCESS
```



Week 5 :

### **Create a simple Java application** that you want **to integrate with Jenkins.**

- The application should have some basic functionality, such as printing "Hello World" or performing simple calculations.

**Example; Create a simple java program with file name as Demo.java and save it in a folder on desktop**

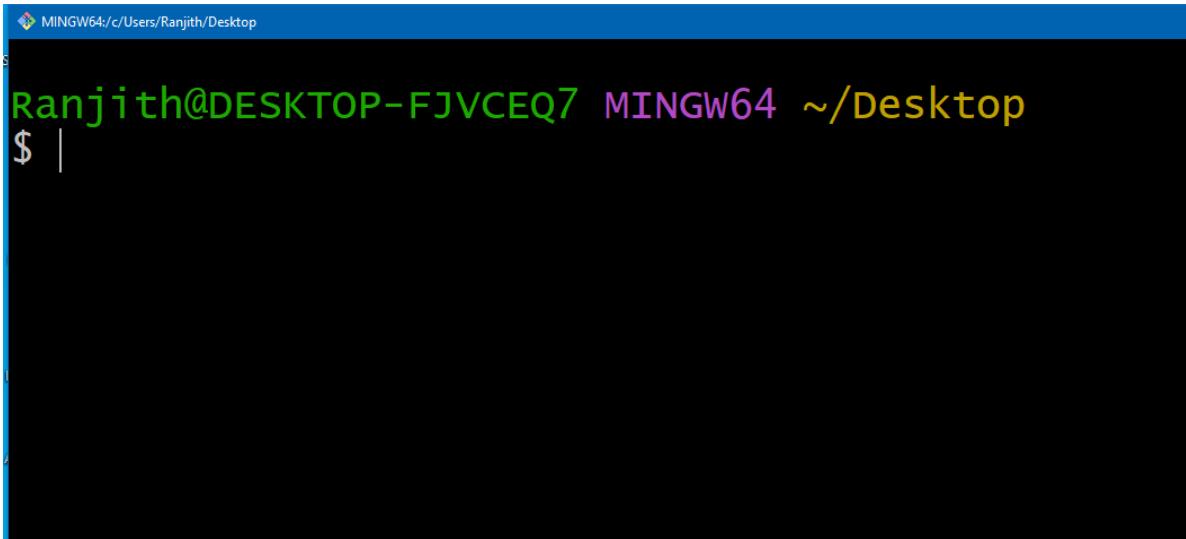
```
public class Demo {  
  
    public static void main (String args[])  
    {  
        System.out.println("hello world");  
    }  
}
```

### **Commit the code to a Git repository:**

- Create a Git repository for the application and commit the code to the repository.

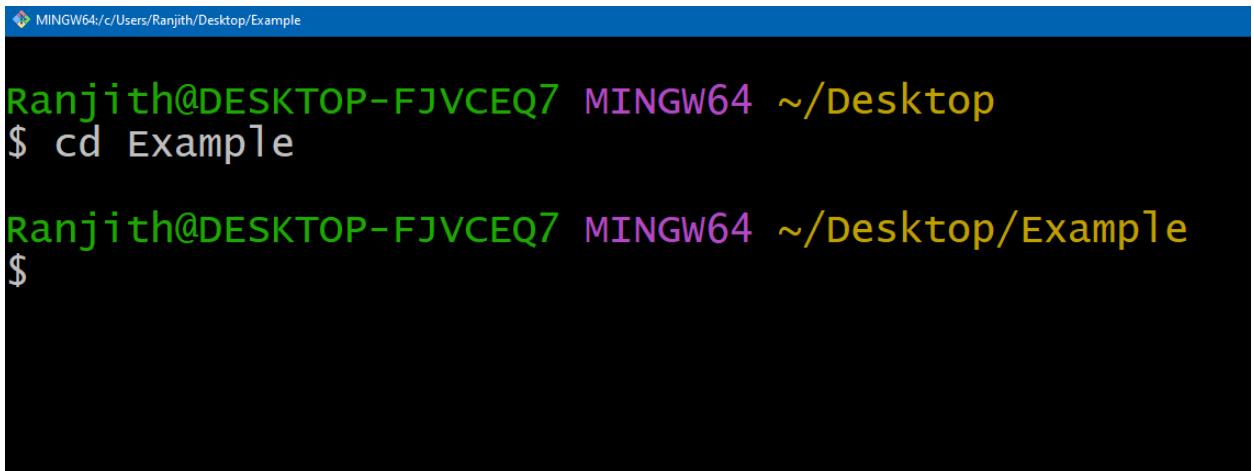
Preform the following steps

1. Login into GitHub account and create a new public repository
2. After login, click on repositories and click on New button to create the repository
3. Right click on the desktop and select open Gitbash here  
Gitbash window opens



Ranjith@DESKTOP-FJVCEQ7 MINGW64 ~/Desktop  
\$ |

Now type the command `cd folder_name` in the above terminal and press enter



Ranjith@DESKTOP-FJVCEQ7 MINGW64 ~/Desktop  
\$ cd Example  
Ranjith@DESKTOP-FJVCEQ7 MINGW64 ~/Desktop/Example  
\$

Execute the following commands to create a local repository and push it on to the GitHub remote repository

*i) git init*

```
Ranjith@DESKTOP-FJVCEQ7 MINGW64 ~/Desktop
$ cd Example

Ranjith@DESKTOP-FJVCEQ7 MINGW64 ~/Desktop/Example
$ git init
Initialized empty Git repository in c:/Users/Ranjith/Desktop/Example/.git/

Ranjith@DESKTOP-FJVCEQ7 MINGW64 ~/Desktop/Example (master)
$ |
```

*ii) git config --global user.name "username of github account"*

*iii) git config --global user.email "email id provided github account"*

*iv) git add .*

*v) git commit -m "adding files updated"*

```
Ranjith@DESKTOP-FJVCEQ7 MINGW64 ~/Desktop/Example (master)
$ git add .

Ranjith@DESKTOP-FJVCEQ7 MINGW64 ~/Desktop/Example (master)
$ git commit -m "adding file updated"
[master (root-commit) 72471d6] adding file updated
 1 file changed, 7 insertions(+)
 create mode 100644 Demo.java
```

vi) **git remote add origin *url address of the git hub repository***

*Ex: git remote add origin https://github.com/username/repository.git*

Copy the url address of the git repository from git hub and paste in the above command

vii) **git push -u origin master**

Refresh the git repository in GitHub and check that files on local machine are pushed on to git remote repository

- Make sure that the Git repository is accessible from the Jenkins server.

**Create a Jenkins job:**

- *Log in to the Jenkins web interface and create a new job.*
- *Configure the job to build the Java application from the Git repository.*
- *Specify the build triggers, such as building after every commit to the repository.*

The screenshot shows the Jenkins dashboard at [localhost:8080](http://localhost:8080). The top navigation bar includes links for Getting Started, TSrevenue & stamps, WriteCodeOnline - PH..., and Other Bookmarks. The main header features the Jenkins logo and a search bar with placeholder text "Search (CTRL+K)". On the left, there's a sidebar with links for New Item, Build History, Manage Jenkins, and My Views. A "Build Queue" section indicates "No builds in the queue." The central content area displays a table of jobs:

S	W	Name	Last Success	Last Failure	Last Duration
✓	☀️	Example1	7 days 0 hr #3	N/A	3.4 sec ➔
⌚	☀️	integrate	8 days 21 hr	N/A	29 sec ➔

Click on New Item to create a new job

The screenshot shows the "New Item" dialog box. At the top, it says "Enter an item name" followed by a text input field containing "Job3". Below that, it says "Select an item type" and shows a card for "Freestyle project":

**Freestyle project**  
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

A blue "OK" button is at the bottom of the dialog.

Give the **Item name**, select the item type as ***Freestyle project*** and click **OK** button

The screenshot shows the Jenkins configuration interface for a job named 'Job3'. The 'General' tab is selected. On the left, there's a sidebar with options: General (selected), Source Code Management, Build Triggers, Build Environment, Build Steps, and Post-build Actions. The main area has a 'Description' section with a large text input field. At the top right, there's an 'Enabled' switch which is turned on. The browser header shows the URL as 'localhost:8080/job/Job3/configure'.

## Configure

### General

Enabled

#### General

- Source Code Management
- Build Triggers
- Build Environment
- Build Steps

#### Description

Plain text [Preview](#)

Give the Description of the project or program

This screenshot shows the 'Source Code Management' tab selected in the Jenkins configuration interface. The sidebar still lists General, Source Code Management (selected), Build Triggers, Build Environment, Build Steps, and Post-build Actions. The main area shows a 'None' option for source code management, which is then changed to 'Git'. A 'Repositories' section appears with a 'Repository URL' input field containing a placeholder 'http://'. The browser header shows the URL as 'localhost:8080/job/Job3/configure'.

## Configure

### Source Code Management

#### General

- Source Code Management (selected)
- Build Triggers
- Build Environment
- Build Steps
- Post-build Actions

#### None

#### Git

##### Repositories

###### Repository URL

***In the source code management, select the option as Git and copy the URL address of the Git repository and paste in Repository URL field***

Dashboard > Job3 > Configuration

Configure

None

Git

General

Source Code Management

Build Triggers

Build Environment

Build Steps

Post-build Actions

Repositories

Repository URL: https://github.com/user/d1.git

Credentials: - none -

This screenshot shows the Jenkins configuration page for Job3. The 'Source Code Management' section is selected. It shows a 'Repository URL' input field containing 'https://github.com/user/d1.git'. Below it, there is a 'Credentials' dropdown set to '- none -'. There is also a red 'X' button in the top right corner of the repository configuration area.

**In branches to build option, mention branch specifier as master**

File Edit View History Bookmarks Tools Help

Job3 Config [Jenkins] ranjith24/d1 localhost:8080/job/Job3/configure 120% Other Bool

Getting Started TSRevenue & stamps WriteCodeOnline - PH...

Dashboard > Job3 > Configuration

Configure

General

Source Code Management

Build Triggers

Build Environment

Build Steps

Post-build Actions

Add Repository

Branches to build

Branch Specifier (blank for 'any') \*/master

Add Branch

This screenshot shows the Jenkins configuration page for Job3. The 'Source Code Management' section is selected. Under 'Branches to build', there is a 'Branch Specifier' input field containing '\*/master'. There is also a red 'X' button in the top right corner of the branch configuration area. A modal window titled 'Add Repository' is partially visible.

## Configure

### Build Triggers

 General

Trigger builds remotely (e.g., from scripts) 

 Source Code Management

Build after other projects are built 

 Build Triggers

Build periodically 

 Build Environment

GitHub hook trigger for GITScm polling 

 Build Steps

Poll SCM 

Schedule 

H/5 \* \* \* \*

**In build triggers option, select POLL SCM and schedule it as H/5 \* \* \* \***

## Configure

### Build Steps

 General

Add build step ^

 Source Code Management

Filter

 Build Triggers

Execute Windows batch command

 Build Environment

Execute shell

 Build Steps

Invoke Ant

 Post-build Actions

Invoke Gradle script

Invoke top-level Maven targets

Run with timeout

Set build status to "pending" on GitHub commit

**In the build steps, select Add build step and select Execute windows batch command and type the following commands to run java program**

The screenshot shows the Jenkins configuration interface for a job named 'Job3'. On the left, there's a sidebar with links: General, Source Code Management, Build Triggers, Build Environment, **Build Steps** (which is selected and highlighted in grey), and Post-build Actions. The main area is titled 'Configure' and shows the 'Build Steps' section. A 'With Ant' link is visible above the build steps. A 'Execute Windows batch command' step is selected, indicated by a dashed border around its configuration panel. The 'Command' field contains the following two lines of code:

```
javac Demo.java  
java Demo
```

**Click on Save to save the job**

The screenshot shows the Jenkins dashboard for the 'Job3' project. The top navigation bar includes the Jenkins logo, a search bar, and a help icon. Below the header, the breadcrumb navigation shows 'Dashboard > Job3 >'. The main content area is titled 'Job3' and displays the following information:

- Status: Shows the status of the job.
- Changes: Shows the last changes made to the job.
- Workspace: Shows the workspace for the job.
- Build Now: Allows triggering a new build.
- Configure: Allows editing the job configuration.
- Delete Project: Allows deleting the project.
- Git Polling Log: Shows the log of git polling events.
- Rename: Allows renaming the project.

Below the workspace section, it says 'execute java program' and lists the following command:

```
javac Demo.java  
java Demo
```

Under the 'Permalinks' section, there are links to various Jenkins pages like 'Status', 'Changes', 'Workspace', etc.

**Now click on Build Now to build the job'**

The screenshot shows a Jenkins job configuration page for 'Job3'. At the top, there's a breadcrumb navigation: 'Dashboard > Job3 >'. Below it, a 'Permalinks' section contains several options: 'Build Now' (with a play icon), 'Configure' (with a gear icon), 'Delete Project' (with a trash bin icon), 'Git Polling Log' (with a clipboard icon), and 'Rename' (with a pencil icon). A large, semi-transparent overlay card is positioned in the center. It has a yellow sun icon and the text 'Build History'. To its right is a dropdown menu set to 'trend'. Below that is a search bar with a magnifying glass icon and a '/' symbol. Underneath the search bar is a green checkmark icon next to '#1' and the date 'Sep 21, 2024, 1:55 PM'. At the bottom of the card are two links: 'Atom feed for all' and 'Atom feed for failures'. To the right of the card is a vertical toolbar with three buttons: a top button with a minus sign, a middle up arrow button, and a bottom down arrow button.

**Click on the Job number from the Build History**

</> Changes

Console Output

Edit Build Information

Delete build '#1'

Timings

Git Build Data

No Tags

 Started by user Ranjith

 This run spent:

- 69 ms waiting;
- 8.8 sec build duration;
- 8.8 sec total from scheduled to completion.

 Revision: 72471d648e224a0bc2ea769bc886c1d9a69e0ac2  
Repository: <https://github.com/ranjitk24/d1.git>

- refs/remotes/origin/master

</> No changes.

## Click on the console output to view the output of Job

Dashboard > Job3 > #1 > Console Output

```

> C:\Program Files\Git\bin\git.exe fetch --tags --force --progress -- https://github.com/ranjitk24/d1.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> C:\Program Files\Git\bin\git.exe config remote.origin.url https://github.com/ranjitk24/d1.git # timeout=10
> C:\Program Files\Git\bin\git.exe config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> C:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision 72471d648e224a0bc2ea769bc886c1d9a69e0ac2 (refs/remotes/origin/master)
> C:\Program Files\Git\bin\git.exe config core.sparsecheckout # timeout=10
> C:\Program Files\Git\bin\git.exe checkout -f 72471d648e224a0bc2ea769bc886c1d9a69e0ac2 # timeout=10
Commit message: "adding file updated"
First time build. Skipping changelog.
[Job3] $ cmd /c call C:\WINDOWS\TEMP\jenkins11303450433661217744.bat

C:\ProgramData\Jenkins\.jenkins\workspace\Job3>javac Demo.java

C:\ProgramData\Jenkins\.jenkins\workspace\Job3>java Demo
Hello world

C:\ProgramData\Jenkins\.jenkins\workspace\Job3>exit 0
Finished: SUCCESS

```

## **Week 6 Explore Docker commands for content management**

**Docker** is the containerization platform that is used to package your application and all its dependencies together in the form of containers to make sure that your application works seamlessly in any environment which can be developed or tested or in production

**Docker** is a tool designed to make it easier **to create, deploy, and run applications by using containers.**

**Containerization** is a **software deployment process** that bundles an application's code with all the files and libraries it needs to run on any infrastructure.

### **Key Components of Docker**

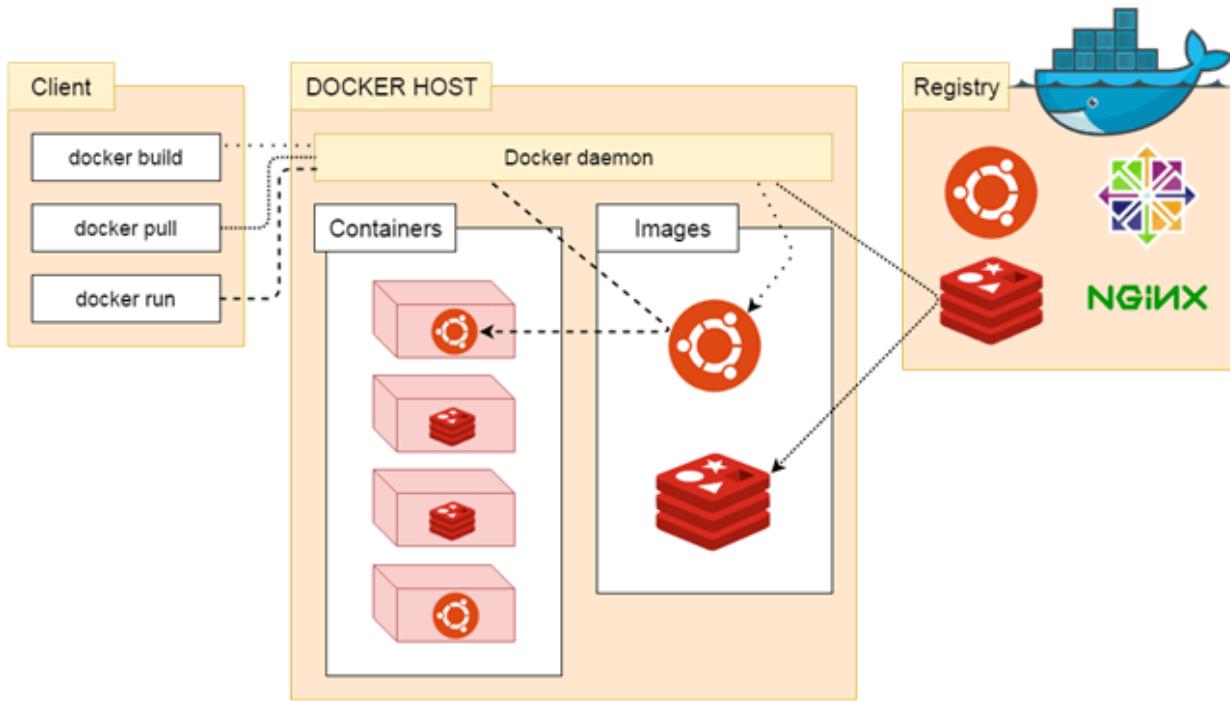
- **Docker Engine:** It is a core part of docker, that handles the creation and management of containers.
- **Container:** A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another
- **Docker Image:** It is a read-only template that is used for creating containers, containing the application code and dependencies. It includes everything needed to run an application: *code, runtime, system tools, system libraries and settings.*
- **Docker Hub:** It is a cloud-based repository that is used for finding and sharing the container images.
- **Docker file:** It is a script that containing instructions to build a docker image.
- **Docker Registry:** It is a storage distribution system for docker images, where you can store the images in both public and private modes

### **Docker daemon**

Docker daemon runs on the host operating system. It is responsible for **running containers to manage docker services.** It offers various Docker objects such as **images, containers, networking, and storage**

### **Docker architecture**

Docker follows Client-Server architecture, which includes the three main components that are **Docker Client, Docker Host, and Docker Registry.**



## 1. Docker Client

Docker client uses commands and REST APIs to communicate with the Docker Daemon (Server).

## 2. Docker Host

Docker Host is used to provide an environment to execute and run applications. It contains the docker daemon, images, containers, networks, and storage.

## 3. Docker Registry

Docker Registry manages and stores the Docker images.

There are two types of registries in the Docker -

**Public Registry** - Public Registry is also called as **Docker hub**.

**Private Registry** - It is used to share images within the enterprise.

## Basic Docker Commands

1 **docker --version**

**docker -v**

**This command is used to get the current version of the docker**

```
PS C:\> docker -v  
Docker version 20.10.17, build 100c701  
PS C:\> docker --version  
Docker version 20.10.17, build 100c701
```

**2 docker pull:** Pull an image or a repository from a registry

Syntax:           **docker pull [OPTIONS] NAME [: TAG|@DIGEST]**

To download an image or set of images (i.e. A Repository)

Ex: **docker pull hello-world**

```
PS C:\> docker pull hello-world  
Using default tag: latest  
latest: Pulling from library/hello-world  
c1ec31eb5944: Pull complete  
Digest: sha256:d211f485f2dd1dee407a80973c8f129f00d54604d2c90732e8e320e5038a0348  
Status: Downloaded newer image for hello-world:latest  
docker.io/library/hello-world:latest
```

**3 docker run:** This command is used to create a container from an image

Syntax : **docker run [OPTIONS] IMAGE [COMMAND] [ARG...]**

Example : **docker run hello-world**

```
PS C:\> docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
c1ec31eb5944: Pull complete  
Digest: sha256:d211f485f2dd1dee407a80973c8f129f00d54604d2c90732e8e320e5038a0348  
Status: Downloaded newer image for hello-world:latest  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.
```

**4. docker ps** This command is used to list all the containers

Syntax : **docker ps [OPTIONS]**

Example : **docker ps (lists currently running containers)**

```
PS C:\> docker ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```

**Command : docker ps -a    -a or -all : Lists all containers that are running & stopped**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e966ebcc530a	hello-world	"/hello"	4 minutes ago	Exited (0) 4 minutes ago		unruful_spence

## 5. docker restart

This command is used to restart one or more containers.

**Syntax:** docker restart [OPTIONS] CONTAINER [CONTAINER...]

**Example:** docker restart my\_container

## 6 docker kill

This command is used to kill one or more containers.

**Syntax:** docker kill [OPTIONS] CONTAINER [CONTAINER...]

**Example:** \$ docker kill my\_container

## 7 Build an Image from a Dockerfile

**docker build -t <image\_name>**

**Ex: docker build -t**

## 8 List local images

**docker images**

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world	latest	d2c94e258dcb	17 months ago	13.3kB
docker/getting-started	latest	3e4394f6b72f	22 months ago	47MB

## 9 docker search To search for an image in Docker Hub.

**docker search <image\_name>**

**Ex: docker search hello-world**

## 10 Delete an Image

**docker rmi <image\_name>**

Ex: **docker rmi hello-world**

## Login into Docker

**docker login -u <username>**

Publish an image to Docker Hub

**docker push <username>/<image\_name>**

## Container commands

Create and run a container from an image, with a custom name:

**docker run --name <container\_name> <image\_name>**

Run a container with and publish a container's port(s) to the host.

**docker run -p <host\_port>:<container\_port> <image\_name>**

Run a container in the background

**docker run -d <image\_name>**

Start or stop an existing container:

**docker start|stop <container\_name> (or <container-id>)**

Remove a stopped container:

**docker rm <container\_name>**

## Week 6 Explore Docker commands for content management

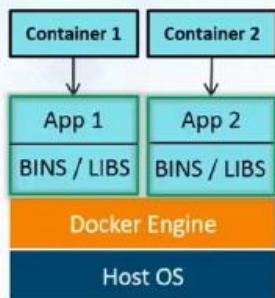
Docker is a *software platform that allows you to build, test, and deploy applications quickly.*

Docker *packages software into standardized units called containers* that have everything the software needs to *run including libraries, system tools, code, and runtime.*

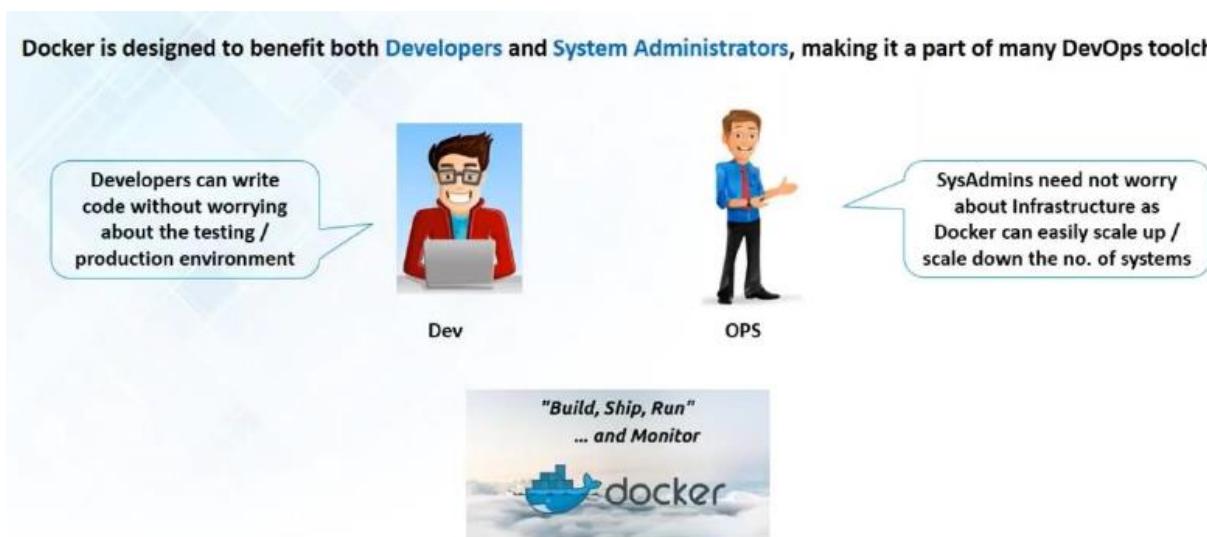
Using Docker, you can *quickly deploy and scale applications into any environment* and know your code will run.

Docker helps *developers build, share, run, and verify applications anywhere* — without tedious environment configuration or management.

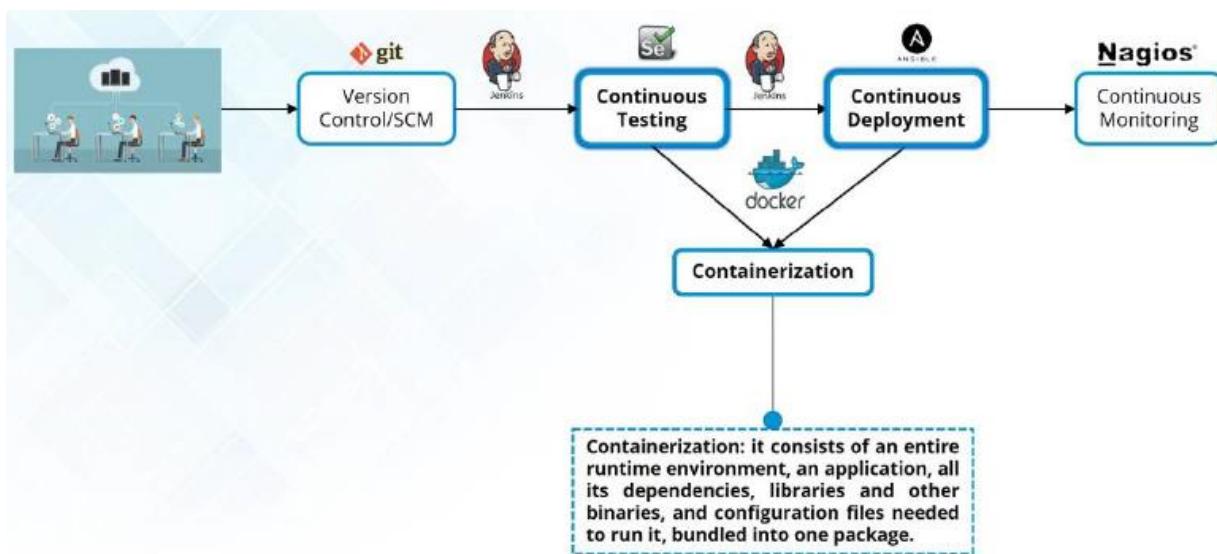
Docker is a Containerization platform which packages your [application](#) and all its dependencies together in the form of [Containers](#) so as to ensure that your application works seamlessly in any environment be it Development or Test or Production.



Docker is designed to benefit both Developers and System Administrators, making it a part of many DevOps toolchains



## How is Docker used in Devops



## Docker Commands

### Docker Run command

This command is used to run a container from an image. The docker run command is a combination of the docker create and docker start commands. It creates a new container from the image specified and starts that container. If the docker image is not present, then the docker run pulls that.

**\$ docker run <image\_name>**

To give name of container

**\$ docker run --name <container\_name> <image\_name>**

### Docker Pull

This command allows you to pull any image which is present in the official registry of docker, Docker hub. By default, it pulls the latest image, but you can also mention the version of the image.

**\$ docker pull <image\_name>**

```
C:\Users\mojha>docker pull redis
Using default tag: latest
latest: Pulling from library/redis
31b3f1ad4ce1: Pull complete
ff29a33e56fb: Pull complete
b230e0fd0bf5: Pull complete
9469c4ab3de7: Pull complete
6bd1cefcc7a5: Pull complete
610e362ffa50: Pull complete
Digest: sha256:b4e56cd71c74e379198b66b3db4dc415f42e8151a18da68d1b61f55fcc7af3e0
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest
```

### Docker PS

This command (by default) shows us a list of all the running containers. Various flags with includes

- -a flag: shows us all the containers, stopped or running.
- -l flag: shows us the latest container.
- -q flag: shows only the Id of the containers.

**\$ docker ps [options..]**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	P
b7e3fefc202d	ubuntu	"bash"	11 hours ago	Up 11 hours	
1b470557a372	ubuntu	"sleep 1000000"	11 hours ago	Up 11 hours	
9088b39c68dd	docker/getting-started	"/docker-entrypoint...."	11 hours ago	Up 11 hours	0

## Docker Stop

This command allows you to stop a container if it has crashed or you want to switch to another one.

**\$ docker stop <container\_ID>**

```
C:\Users\mojha>docker stop b7e3fefc202d  
b7e3fefc202d
```

## Docker Start

To start the stopped container again, with the help of this command.

**\$ docker start <container\_ID>**

## Docker rm

To delete a container. By default when a container is created, it gets an ID as well as an imaginary name such as confident\_boyd, heuristic\_villani, etc. You can either mention the container name or its ID.

**Some important flags:**

- **-f flag: remove the container forcefully.**
- **-v flag: remove the volumes.**
- **-l flag: remove the specific link mentioned.**

**\$ docker rm {options} <container\_name or ID>**

```
C:\Users\mojha>docker rm b7e3fefc202d  
b7e3fefc202d
```



## Docker RMI

To delete the image in docker. You can delete the images which are useless from the docker local storage so you can free up the space

**docker rmi <image ID/ image name>**

## Docker Images

Lists all the pulled images which are present in our system.

**\$ docker images**

```
C:\Users\mojha>docker images  
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE  
redis               latest    9da089657551  4 days ago   117MB  
docker/getting-started  latest   cb90f98fd791  5 months ago  28.8MB  
docker101tutorial    latest    9a419a57dc8  6 months ago  28.8MB  
ubuntu              latest    2b4cba85892a  6 months ago  72.8MB  
alpine/git           latest    c6b70534b534  10 months ago 27.4MB  
hello-world          latest    feb5d9fea6a5  12 months ago 13.3kB
```

## Docker Push

To store the image in the remote registry which is DockerHub for that you need to push your image

**docker push <Image name/Image ID>**

## Docker Build

The docker build command is used to build the docker images with the help of [Dockerfile](#).

**docker build -t image\_name:tag .**

In the place of image\_name use the name of the image you build with and give the tag number and . “dot” represents the current directory

## Docker Stop

`docker stop container_name_or_id`

## Docker Basic Command

Following are the some of the docker basic commands

1. **docker images:** Docker images will list all the images which are pulled or build in that docker host.
2. **docker pull:** Docker pull will the docker images from the dockerhub.
3. **docker run:** Docker run will run the docker image as an container.
4. **docker ps:** Docker run will list all the containers which are running in the docker host.
5. **docker stop:** Docker stop will stop the docker container which are already running.
6. **docker rm:** Docker rm command will remove the containers which are in the stop condition.

## Docker Commands List

Following are the docker commands which listed form build and Docker image to running it an Docker container and attaching the docker volumes to it.

## Docker Image Command

1. **docker build command:** It will build Docker images by using the **Dockerfile**.
2. **docker pull command:** Docker pull command will pull the **Docker image** whcih is available in the **dockerhub**.

3. **docker images command:** It will list all the images which are pulled and build in the docker host.
4. **docker inspect command:** It will help to debug the docker image if any errors occurred while building an image or pulling the image.
5. **docker push command:** Docker command will push the docker image into the Dockerhub.
6. **docker save command:** It will save the docker image in the form of dockerfile.
7. **docker rmi command:** It will remove the docker image.

## Docker Container Command

1. **docker attach command:** Connecting to an Existing Container.
2. **docker ps command:** To list the running containers.
3. **docker container inspect infinite Command:** To Inspect the Docker containers.
4. **docker exec command:** To execute the commands in the running containers.
5. **docker cp command:** To copy the file from docker host to the docker containers,

## **Week 7 Develop a simple containerized application using Docker.**

### **Choose an application:**

- Choose a simple application that you want to containerize. For example, a Python script or any Java program

Here we use simple Java program and containerize it using docker

### **1 Create an application folder**

- 2. Open the Visual Studio Code and select Open Folder from File menu**
- 3. Now select the application folder that is created and open**
- 4. From the explorer bar, click on the application folder**
- 5. Now click on NewFile button to create a file**

**Now ,Create a simple Java program as below**

```
public class Test  
{  
    public static void main(String[] args)  
    {  
        int a=20,b=30,c;  
        c=a*b;  
        System.out.println("Hello world");  
        System.out.println("value of c is "+c);  
    }  
}
```

*Save it as with .java extension*

Now again create another file and name the file as Dockerfile

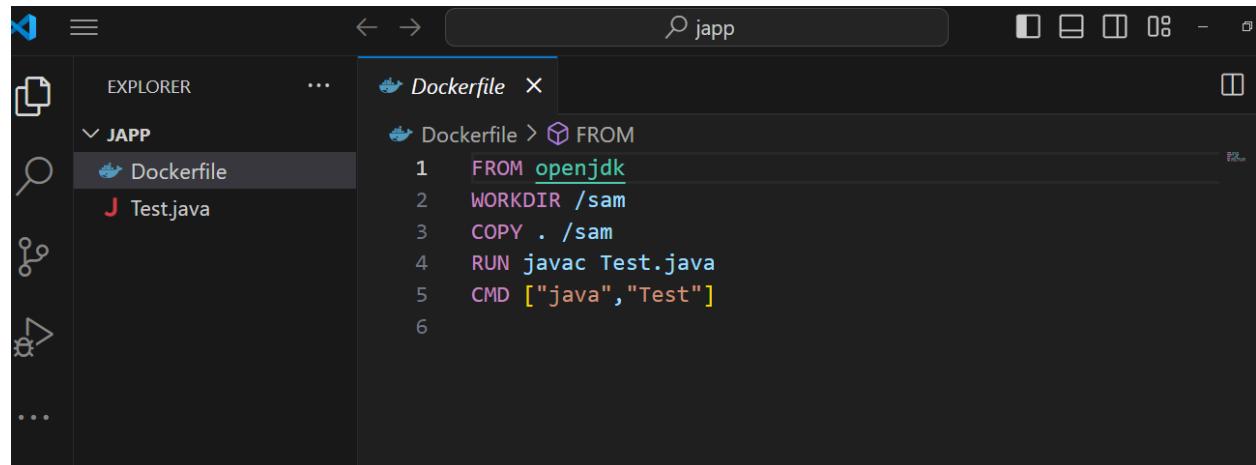
### Create a Dockerfile for the above Java program

In the Dockerfile, specify the base image, copy the application into the container, and specify the command to run the application

#### Dockerfile

```
FROM openjdk    # uses the base image –openjdk with java installed  
WORKDIR /sam      # Set the working directory  
COPY . /sam      # Copies all the files to working directory  
RUN javac Test.java    # Compile the Java program  
CMD ["java", "Test"]    # Set the default command to run the Java program
```

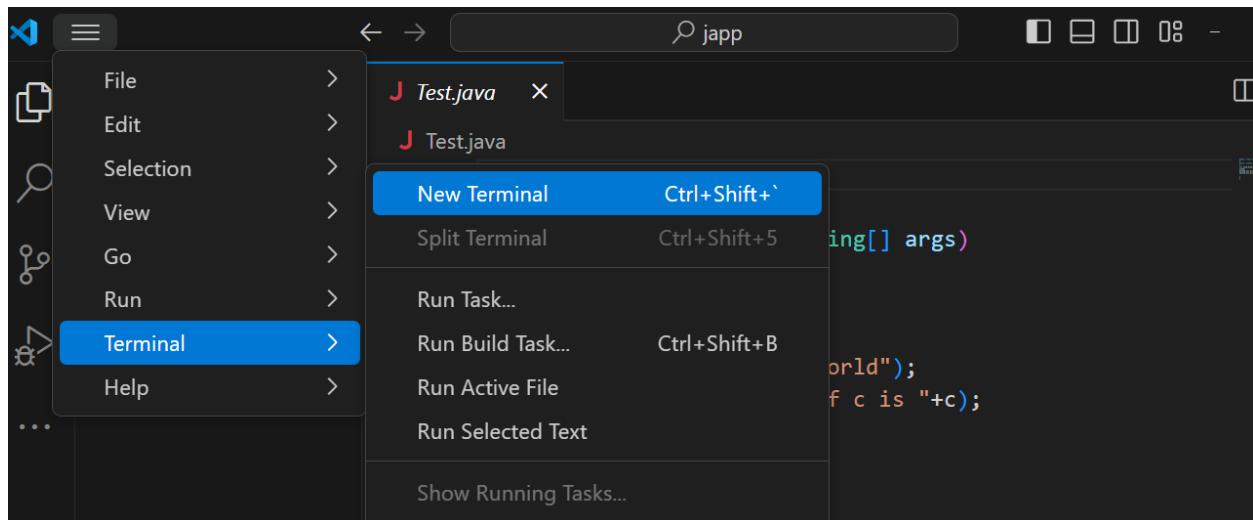
Save the Java file and Docker file



A screenshot of the Visual Studio Code interface. The left sidebar shows a project structure with a folder named 'JAPP' containing a 'Dockerfile' and a file named 'Test.java'. The 'Test.java' file is currently selected and highlighted in blue. The main editor area displays the following Java code:

```
1 public class Test
2 {
3     public static void main(String[] args)
4     {
5         int a=20,b=30,c;
6         c=a*b;
7         System.out.println("Hello world");
8         System.out.println("value of c is "+c);
9     }
10 }
```

Start the New Terminal from Visual studio

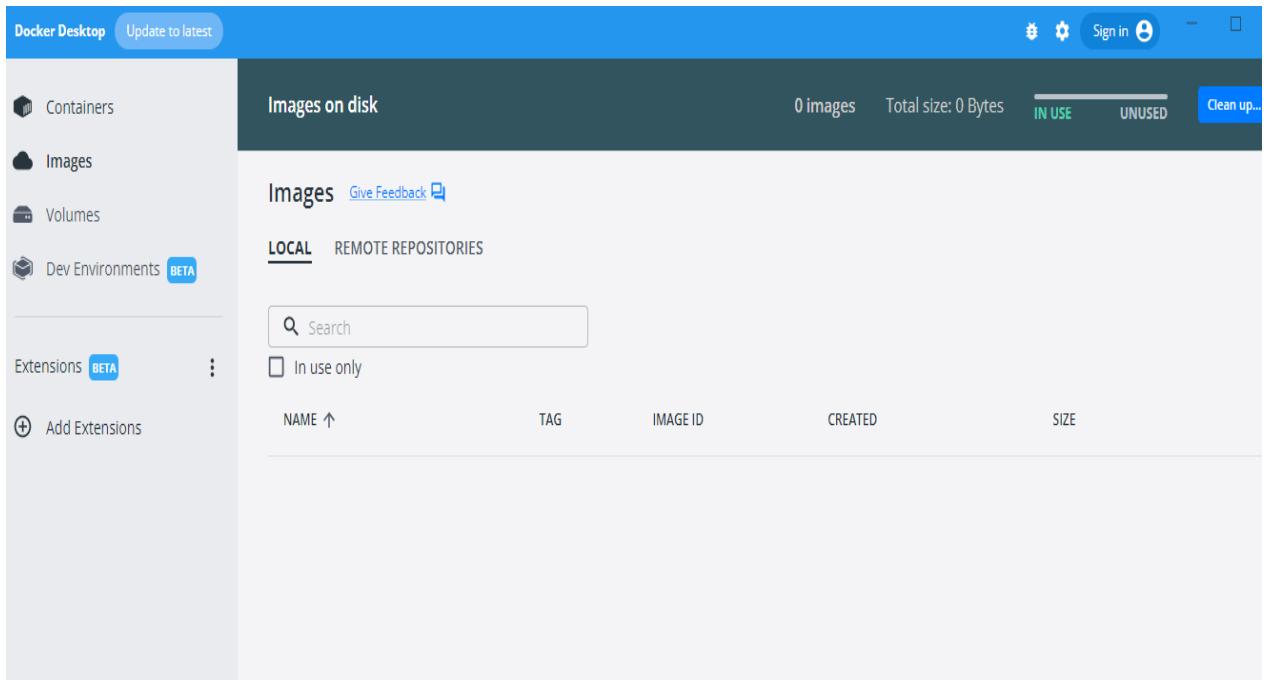


Type the command to list all the images :::: **docker images**

The screenshot shows a terminal window in VS Code with the following content:

```
PS C:\Users\Ranjith\Desktop\japp> docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
PS C:\Users\Ranjith\Desktop\japp>
```

Start the Dockerdesktop



**Build the Docker image** using the **docker build** command. This command builds a new Docker image using the Dockerfile and tags the image with the name

**Command : docker build -t "any name for your docker image" .**

**Example : docker build -t helloworld .**

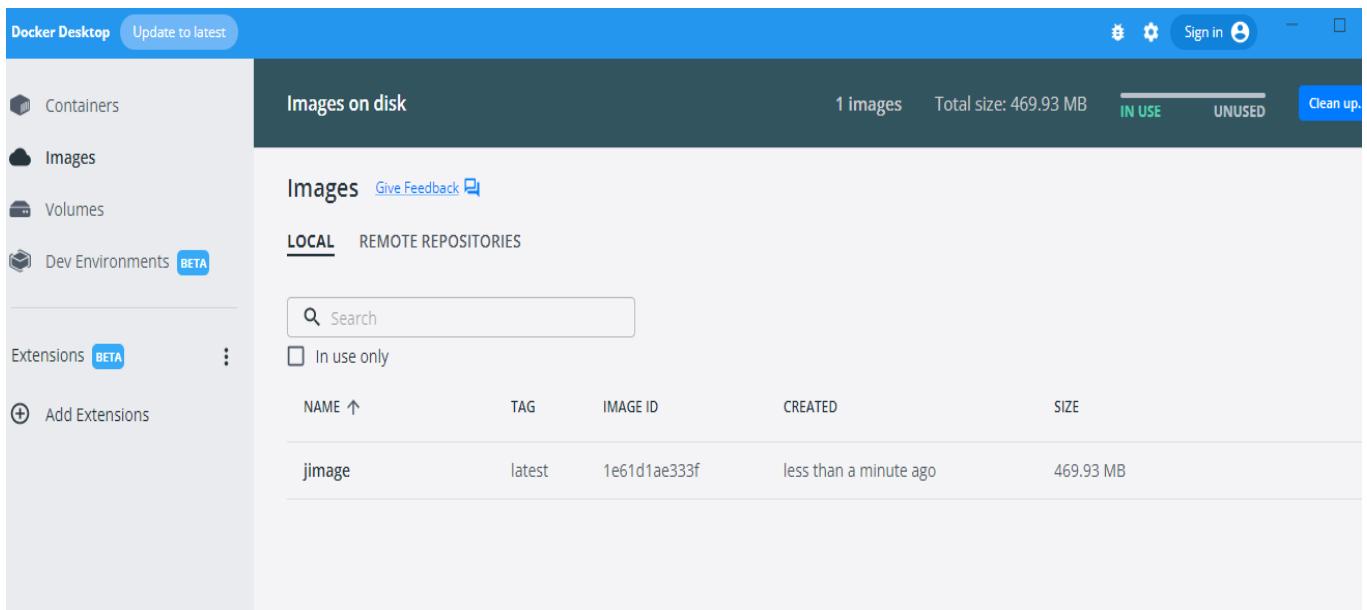
```

Dockerfile
Test.java

PS C:\Users\Ranjith\Desktop\japp> docker build -t jimage .
[+] Building 25.8s (9/9) FINISHED
=> [internal] load build definition from Dockerfile          1.6s
=> => transferring dockerfile: 122B                         0.6s
=> [internal] load .dockerignore                            1.9s
=> => transferring context: 2B                           0.7s
=> [internal] load metadata for docker.io/library/openjdk:lates 5.1s
=> [1/4] FROM docker.io/library/openjdk@sha256:9b448de897d211c9 0.0s
=> [internal] load build context                          0.6s
=> => transferring context: 340B                         0.1s
=> CACHED [2/4] WORKDIR /sam                           0.0s
=> [3/4] COPY . /sam                                    1.4s
=> [4/4] RUN javac Test.java                           13.4s
=> exporting to image                                 2.9s
=> => exporting layers                             2.2s
=> => writing image sha256:1e61d1ae333f65135222bec7259b3ea0ce17 0.2s
=> => naming to docker.io/library/jimage            0.1s

```

View the image created in the docker



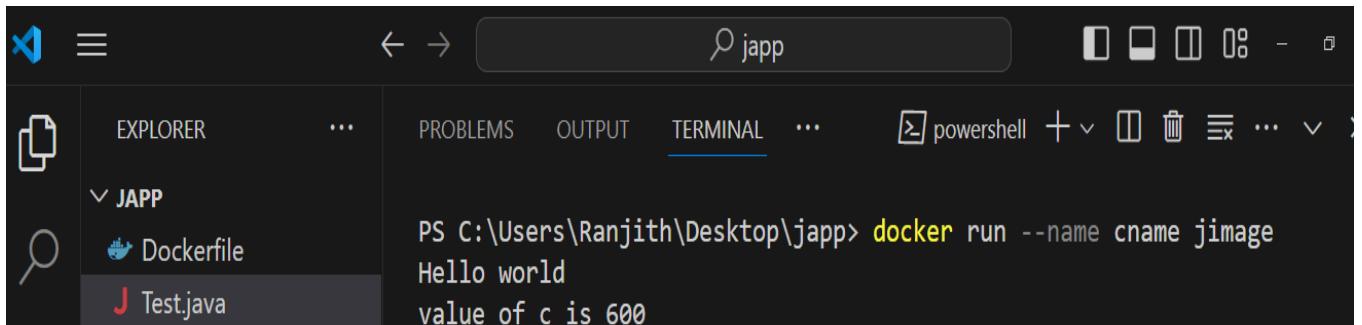
## Run the Docker container:

Run the following command to start a new container based on the image:

```
$ docker run --name mycontainer myimage
```

This command starts a new container named "mycontainer" based on the "myimage" image and runs the Java code inside the container.

## Example



A screenshot of the Visual Studio Code interface. The title bar shows the search term "japp". The left sidebar has sections for EXPLORER, JAPP (with files Dockerfile and Test.java), PROBLEMS, OUTPUT, TERMINAL (which is selected), and other tabs like powershell. The main area shows a terminal window with the following output:

```
PS C:\Users\Ranjith\Desktop\japp> docker run --name cname jimage
Hello world
value of c is 600
```

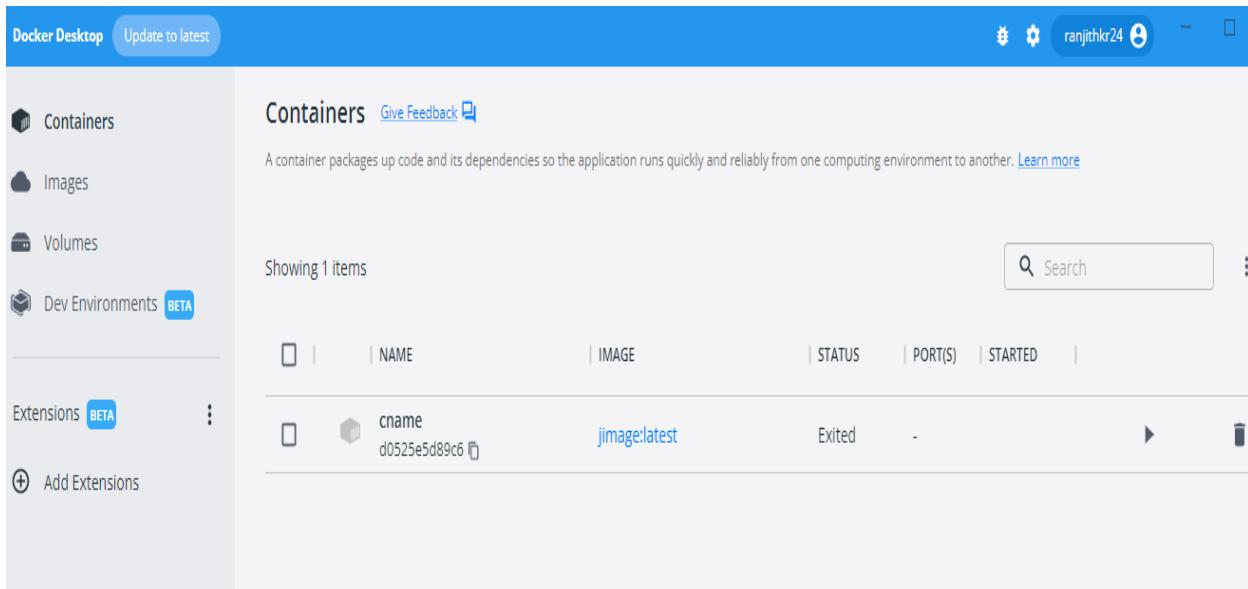
- Verify the output:

Run the following command to verify the output of the container:

```
$ docker logs mycontainer
```

This command displays the logs of the container and should show the programs output

```
PS C:\Users\Ranjith\Desktop\japp> docker logs cname
Hello world
value of c is 600
PS C:\Users\Ranjith\Desktop\japp>
```



## Pushing images into the docker hub

### Create a repository

Login to Dockerhub, Select Create Repository, name it and choose it to make it as public or private

### Tag the Image use the command

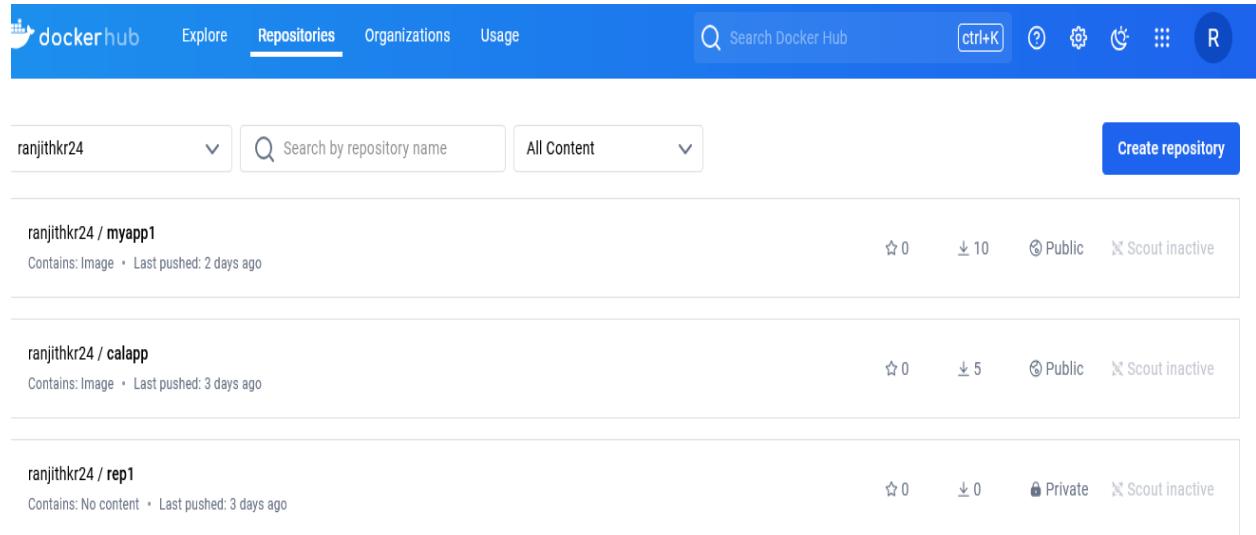
**docker tag <image name> <dockerhub-user name>/<repo-name><version>**

Log into Docker

**docker login give login credentials**

Push the Image use the command **docker push <docker-hub-username>/<image name> <version name>**

## Login into Docker hub



The screenshot shows the Docker Hub interface. At the top, there is a blue header bar with the Docker Hub logo, navigation links for 'Explore', 'Repositories' (which is underlined), 'Organizations', and 'Usage', and a search bar labeled 'Search Docker Hub'. To the right of the search bar are several icons: a magnifying glass for search, a question mark for help, a gear for settings, a refresh symbol, a grid for organization, and a profile icon with the letter 'R'. Below the header, the main content area has a search bar with the placeholder 'Search by repository name' and a dropdown menu set to 'All Content'. On the left, there is a dropdown menu showing 'ranjithkr24'. On the right, there is a blue button labeled 'Create repository'. The main content area displays three repository cards:

- ranjithkr24 / myapp**  
Contains: Image • Last pushed: 2 days ago  
Star count: 0, Pull requests: 10, Public status, Scout inactive
- ranjithkr24 / calapp**  
Contains: Image • Last pushed: 3 days ago  
Star count: 0, Pull requests: 5, Public status, Scout inactive
- ranjithkr24 / rep1**  
Contains: No content • Last pushed: 3 days ago  
Star count: 0, Pull requests: 0, Private status, Scout inactive

## Tagging docker image and login into docker hub

A screenshot of the Visual Studio Code interface. The title bar shows the workspace name 'japp'. The left sidebar has icons for Explorer, Search, Problems, and Terminal. The Explorer view shows a folder named 'JAPP' containing 'Dockerfile' and 'Test.java'. The terminal tab is active, displaying the following command-line session:

```
PS C:\Users\Ranjith\Desktop\japp> docker tag jimage ranjithkr24/myimage:v1.0
PS C:\Users\Ranjith\Desktop\japp> docker login
Authenticating with existing credentials...
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/
PS C:\Users\Ranjith\Desktop\japp>
```

## Issuing docker push command to push the image

```
Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/
PS C:\Users\Ranjith\Desktop\japp> docker push ranjithkr24/myimage:v1.0
The push refers to repository [docker.io/ranjithkr24/myimage]
d9081b57f2d9: Pushed
7f43ae0d9720: Pushed
020c055d83f3: Mounted from ranjithkr24/myapp1
56285d9a7760: Mounted from ranjithkr24/myapp1
077bff59ce57: Mounted from ranjithkr24/myapp1
9cd9df9ffc97: Mounted from ranjithkr24/myapp1
v1.0: digest: sha256:3d3de30bf1a86b342f65536c3122ff6552c015beaeb171e763
7d2d6d2342634f size: 1575
```

Pushed image can be viewed in dockerhub

The screenshot shows the Docker Hub website interface. At the top, there is a blue header bar with the Docker Hub logo, navigation links for Explore, Repositories (which is underlined), Organizations, and Usage, a search bar labeled "Search Docker Hub" with a "ctrl+K" keyboard shortcut, and various icons for help, settings, and account management.

Below the header, there are two search/filter sections: one for "ranjithkr24" and another for "Search by repository name". There is also a dropdown for "All Content" and a "Create repository" button.

The main content area displays two repository cards:

- ranjithkr24 / myimage**  
Contains: Image • Last pushed: 3 minutes ago  
Star count: 0, Download count: 0, Public status, Scout inactive
- ranjithkr24 / myapp1**  
Contains: Image • Last pushed: 2 days ago  
Star count: 0, Download count: 10, Public status, Scout inactive

## **Week 8 Integrate Kubernetes and Docker.**

### **Kubernetes**

*Kubernetes*, also known as K8s, is an **open-source system for automating deployment, scaling, and management of container-based applications** in different kinds of environments like physical, virtual, and cloud-native computing foundations.

### **Some of the benefits of Kubernetes**

- **Container Orchestration:** Kubernetes manages the placement and scheduling of containers across a cluster of machines, ensuring that the desired state of the application is maintained.
- **Scalability:** Kubernetes allows applications to scale horizontally by automatically adding or removing containers based on resource utilization and user-defined rules.
- **Self-Healing:** Kubernetes monitors the health of containers. It automatically restarts failed containers or replaces them with new ones to ensure the desired state of the application is maintained.
- **Rolling Updates and Rollbacks:** Kubernetes supports rolling updates, allowing applications to be updated with minimal downtime. If any issues arise, it also facilitates rollbacks to previous versions.
- **Storage Orchestration:** Kubernetes provides a way to manage persistent storage volumes and attach them to containers as needed.

### **Differences between Docker and Kubernetes**

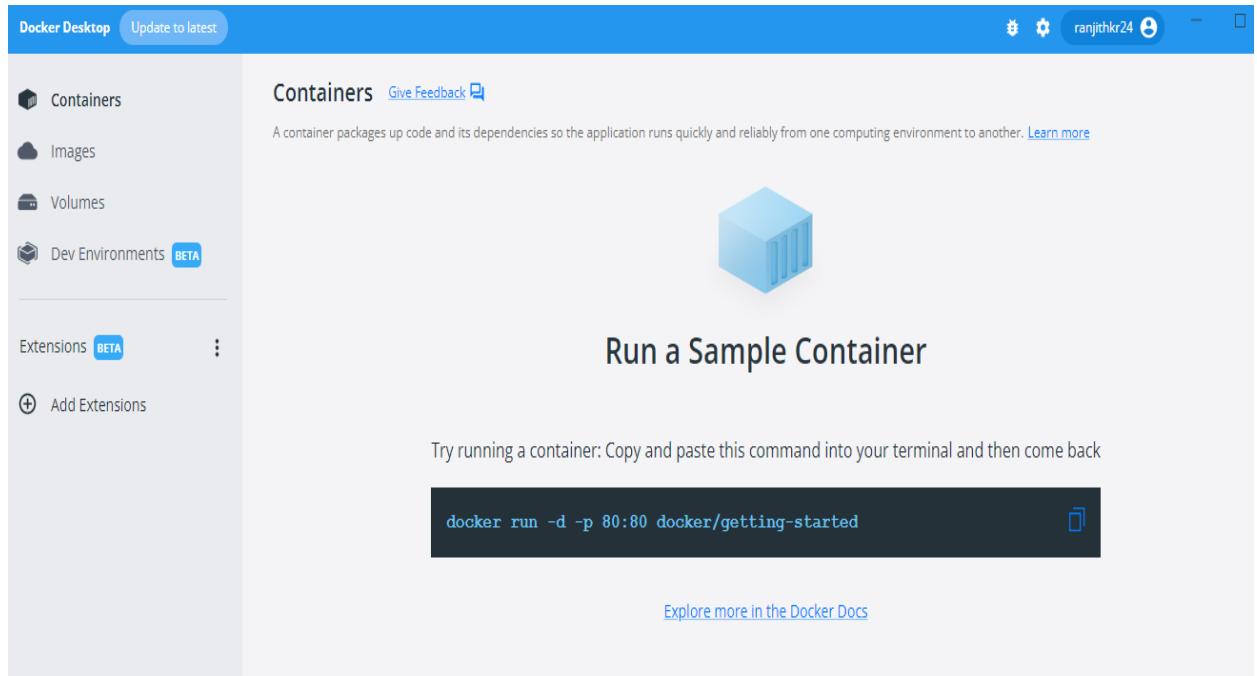
Docker is used for **building, shipping, and running containers**—it provides the means to create isolated environments for applications.

On the other hand, Kubernetes focuses on orchestrating containers, meaning it helps **manage, scale, and ensure the smooth operation of large collections of containers**.

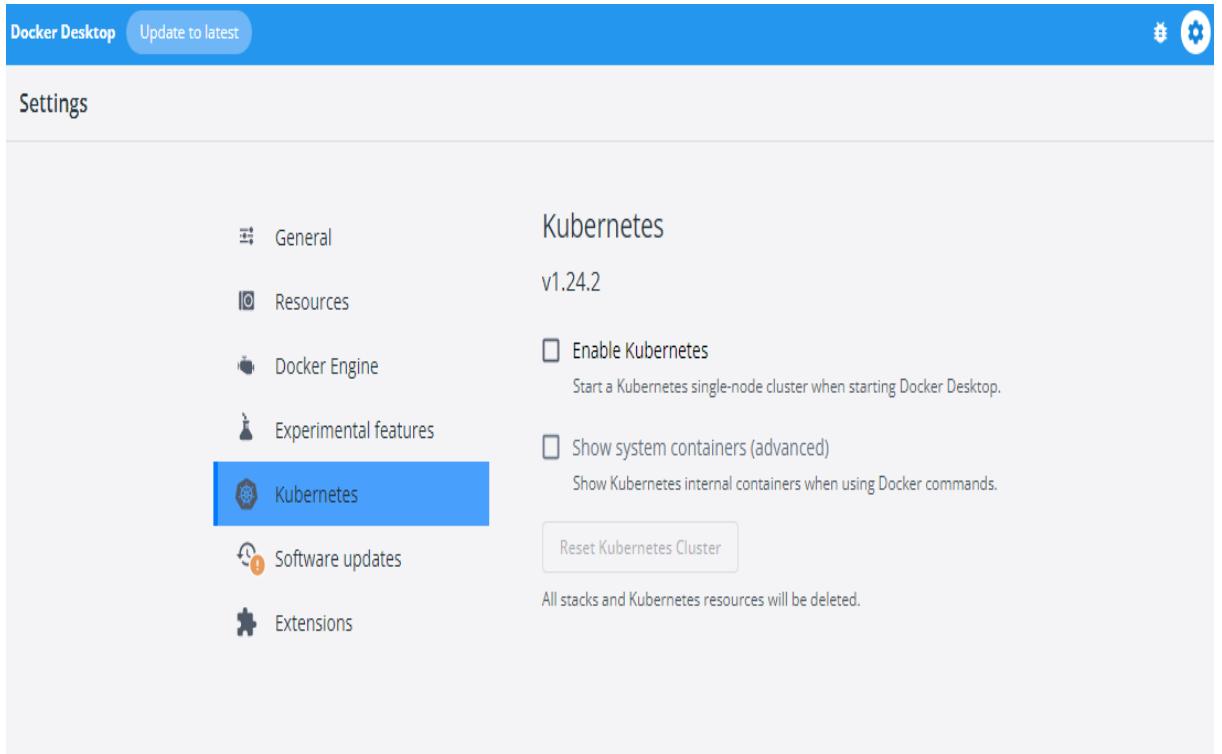
Docker manages individual containers, whereas Kubernetes manages multiple containers across clusters.

## Integrating Kubernetes with Docker

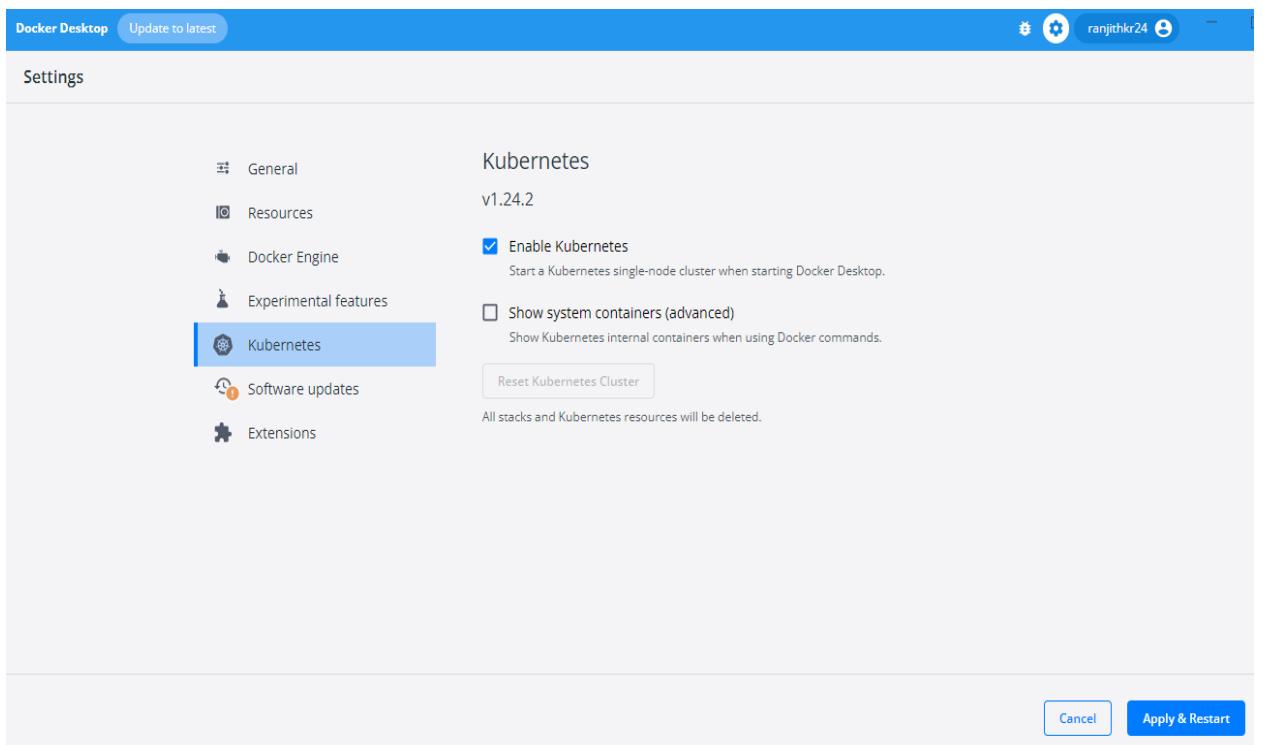
### 1. Start the Docker Desktop



### 2. Select Settings on the top right side from the Docker Desktop Dashboard

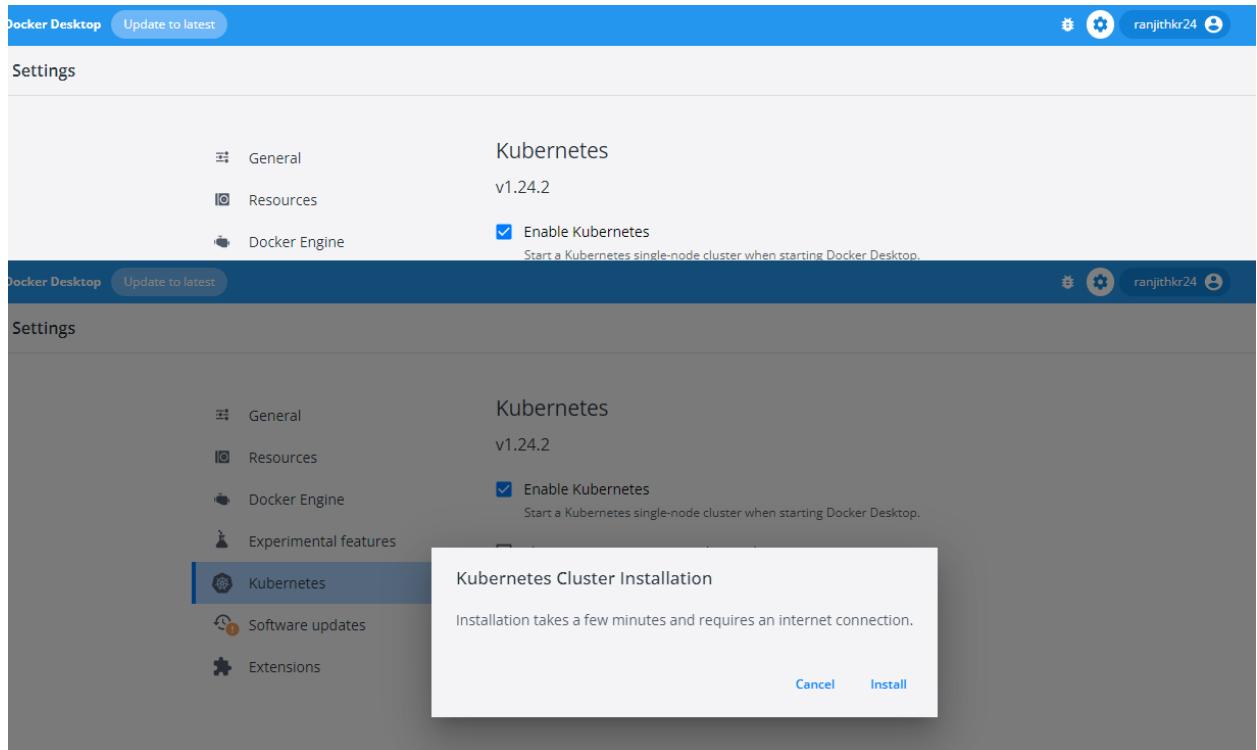


3. Select Kubernetes from the left side bar
4. Check the Enable Kubernetes checkbox



5. Select Apply & restart to save the settings

6. Select Install to confirm



7. Go to Kubernetes website , <https://kubernetes.io/releases/download/#binaries>

Download **kubectlconvert.exe** for Windows OS

8 Save this file **kubectlconvert.exe** in a folder **and copy the folder into**

**C//Windows//System32**

9. Set the path in environment path variable **for the kubectlconvert.exe file**

10. Open command prompt and type the following commands

**kubectl version --client**

**kubectl get pods**

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>kubectl version --client
WARNING: This version information is deprecated and will be replaced with the output from kubectl version --short. Use --output=yaml|json to get the full version.
Client Version: version.Info{Major:"1", Minor:"24", GitVersion:"v1.24.2", GitCommit:"f66044f4361b9f1f96f0053dd46cb7dce5e990a8", GitTreeState:"clean", BuildDate:"2022-06-15T14:22:29Z", GoVersion:"go1.18.3", Compiler:"gc", Platform:"windows/amd64"}
Kustomize Version: v4.5.4

C:\WINDOWS\system32>kubectl get pods
No resources found in default namespace.

C:\WINDOWS\system32>
```

NAME	TAG	IMAGE ID	CREATED	SIZE
aj	latest	1e61d1ae333f	5 days ago	469.93 MB
docker/desktop-storage-...	v2.0	99f89471f470	over 3 years ago	41.85 MB
docker/desktop-vpnkit-c...	v2.0	8c2c38aa676e	over 3 years ago	21.03 MB
hubproxy.docker.interna...	kubernetes-v1.24.2...	5dcc4b79ec39	over 2 years ago	364.07 MB
k8s.gcr.io/coredns/coredns	v1.8.6	a4ca41631cc7	about 3 years ago	46.83 MB
k8s.gcr.io/etcd	3.5.3-0	aebe758cef4c	over 2 years ago	299.5 MB
k8s.gcr.io/kube-apiserver	v1.24.2	d3377ffb7177	over 2 years ago	129.71 MB
k8s.gcr.io/kube-controle...	v1.24.2	34cdf99b1bb3	over 2 years ago	119.35 MB

KUBERNETES RUNNING

RAM 2.74GB CPU 5.47% Connected to Hub v4.11.0

## Week 9. Automate the process of running containerized application

**Kubernetes** is an open-source platform for **managing containerized workloads and services**. It provides a **robust framework for automating deployments, scaling, and managing containerized applications**

### Prerequisites

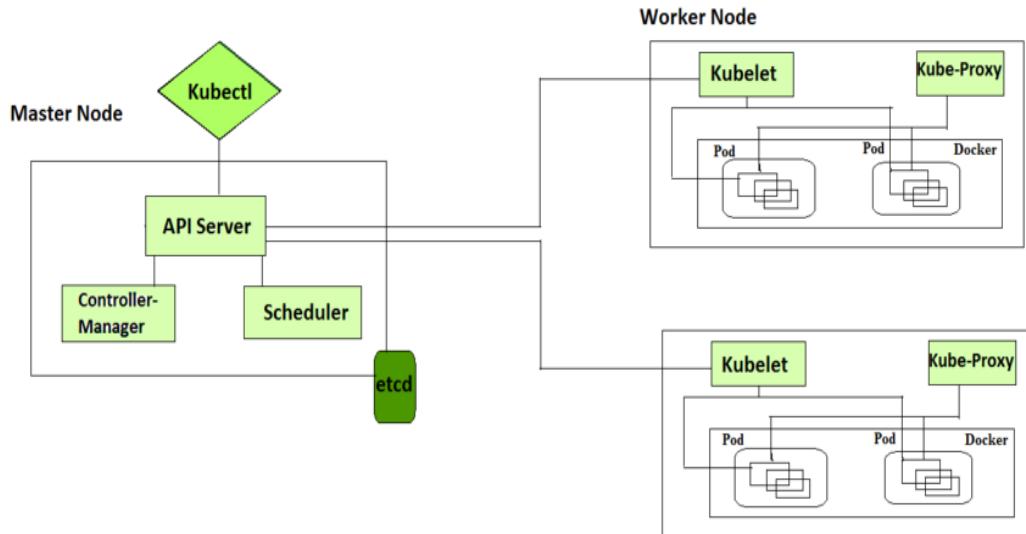
following prerequisites are required:

- **Kubernetes Cluster**: A running Kubernetes cluster. You can set up a local cluster using Minikube or use a managed service like Google Kubernetes Engine (GKE).
- **kubectl**: The command-line tool for interacting with your Kubernetes cluster.
- **Docker**: Installed and running on your machine.
- **Containerized Application**: Your application packaged into a Docker image.

### Kubernetes Cluster Architecture

**Kubernetes comes with a client-server architecture. It consists of master and worker nodes**

The master node, contains the components such as [API](#) Server, controller manager, scheduler, and etcd [database](#) for stage storage. kubelet to communicate with the master, the kube-proxy for networking, and a container runtime such as [Docker](#) to manage containers.



## Kubernetes Components

Kubernetes is composed of a number of components, each of which plays a specific role in the overall system. These components can be divided into two categories:

- **nodes:** Each [Kubernetes cluster](#) requires at least one worker node, which is a collection of worker machines that make up the nodes where our container will be deployed.
- **Control plane:** The worker nodes and any pods contained within them will be under the control plane.

## Control Plane Components

It is basically a collection of various components that help us in managing the overall health of a cluster. For example, if you want to set up new pods, destroy pods, scale pods, etc. Basically, 4 services run on Control Plane:

1. Kube-API server
2. Kube-Scheduler
3. Kube-Controller-Manager
4. etcd

## Node Components

These are the nodes where the actual work happens. Each Node can have multiple pods and pods have containers running inside them.

1. Container runtime
2. kubelet
3. kube-proxy

## Test.java

```
import java.io.IOException;  
  
import java.net.InetSocketAddress;  
  
import com.sun.net.httpserver.HttpServer;
```

```

public class Test {

    public static void main(String[] args) throws IOException
    {
        int port = 8080; // Port on which the server will listen
        System.out.println("Starting server on port " + port);
        HttpServer server = HttpServer.create(new InetSocketAddress(port), 0);
        server.createContext("/", (exchange -> {
            String response = "Hello, Kubernetes!";
            exchange.sendResponseHeaders(200, response.getBytes().length);
            exchange.getResponseBody().write(response.getBytes());
            exchange.close();
        }));
        server.setExecutor(null);
        server.start();
        System.out.println("Server is running at http://localhost:" + port);
    }
}

```

### **Step 1: Create a Docker Image**

First, create a Docker image for your application. This involves writing a Dockerfile that specifies the base image, copies the application code, and sets the command to run the application.

#### **Dockerfile**

```

# Use OpenJDK base image
FROM openjdk:17-alpine
# Set working directory
WORKDIR /app
# Copy all files to the working directory
COPY . /app

```

```

# Compile the Java application
RUN javac Test.java

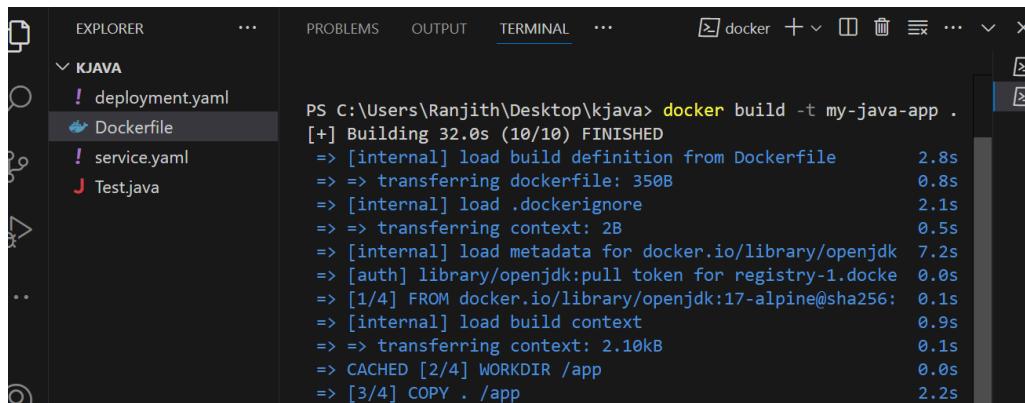
# Expose port 8080 for the application
EXPOSE 8080

# Run the Java application
CMD ["java", "Test"]

```

Build the Docker image using the Dockerfile:

➤ **docker build -t my-java-app .**

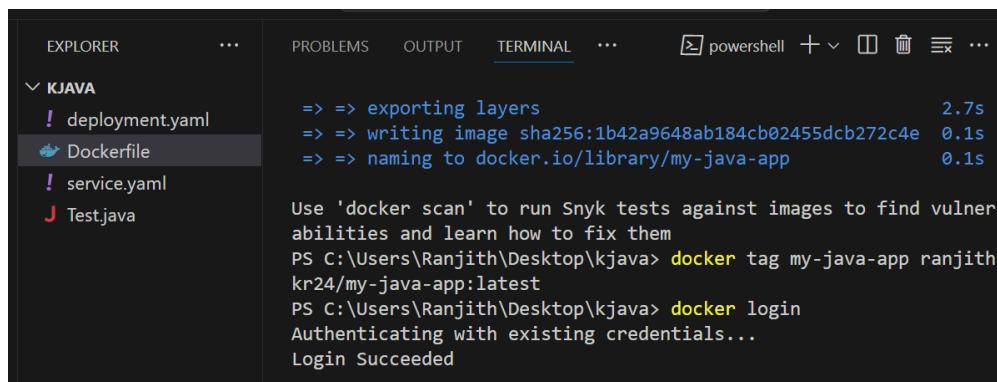


```

PS C:\Users\Ranjith\Desktop\kjava> docker build -t my-java-app .
[+] Building 32.0s (10/10) FINISHED
=> [internal] load build definition from Dockerfile          2.8s
=> => transferring dockerfile: 350B                         0.8s
=> [internal] load .dockerignore                            2.1s
=> => transferring context: 2B                           0.5s
=> [internal] load metadata for docker.io/library/openjdk 7.2s
=> [auth] library/openjdk:pull token for registry-1.docke 0.0s
=> [1/4] FROM docker.io/library/openjdk:17-alpine@sha256: 0.1s
=> [internal] load build context                          0.9s
=> => transferring context: 2.10kB                      0.1s
=> CACHED [2/4] WORKDIR /app                           0.0s
=> [3/4] COPY . /app                                     2.2s

```

Tagging the image and login into dockerhub



```

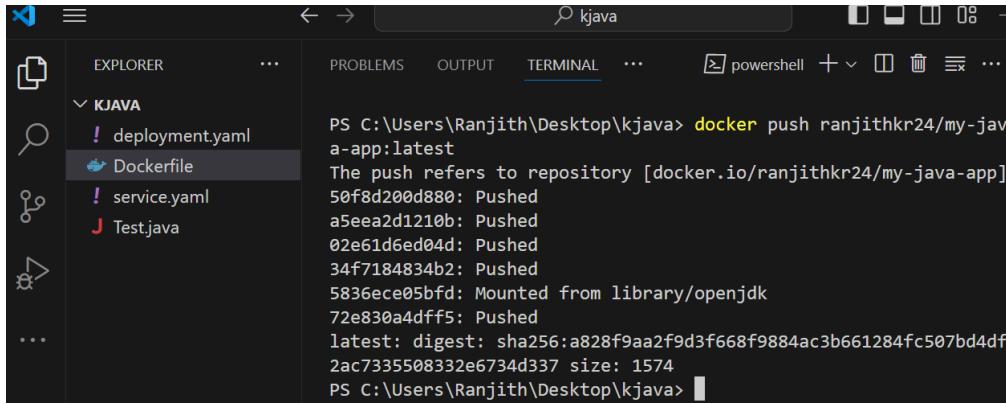
PS C:\Users\Ranjith\Desktop\kjava> docker tag my-java-app ranjith
kr24/my-java-app:latest
PS C:\Users\Ranjith\Desktop\kjava> docker login
Authenticating with existing credentials...
Login Succeeded

```

**Step 2: Push the Docker Image to a Registry**

Push the Docker image to a container registry like Docker Hub

```
docker tag my-java-app:latest <your-docker-hub-username>/my-java-app:latest
docker push <your-docker-hub-username>/my-java-app:latest
```



```
PS C:\Users\Ranjith\Desktop\kjava> docker push ranjithkr24/my-java-app:latest
The push refers to repository [docker.io/ranjithkr24/my-java-app]
50f8d200d880: Pushed
a5eea2d1210b: Pushed
02e61d6ed04d: Pushed
34f7184834b2: Pushed
5836ece05bfd: Mounted from library/openjdk
72e830a4dff5: Pushed
latest: digest: sha256:a828f9aa2f9d3f668f9884ac3b661284fc507bd4df
2ac7335508332e6734d337 size: 1574
PS C:\Users\Ranjith\Desktop\kjava>
```

### Step 3: Create a Kubernetes Deployment

A Kubernetes Deployment is a resource that manages the rollout of new versions of an application. It ensures that a specified number of replicas (i.e., copies) of your application are running at any given time.

Create a YAML file named **deployment.yaml** with the following content:

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: my-java-app
```

```
spec:
```

```
  replicas: 3
```

```
  selector:
```

```
    matchLabels:
```

```
      app: my-java-app
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
app: my-java-app

spec:

containers:

- name: my-java-app

image: ranjithkr24/my-java-app:latest

ports:

- containerPort: 8080

env:

- name: PORT

value: "8080"
```

Apply the deployment configuration to your Kubernetes cluster:

```
kubectl apply -f deployment.yaml
```

#### Step 4: Create a Kubernetes Service

A Kubernetes Service provides a network identity and load balancing for accessing your application. Create a YAML file named service.yaml with the following content:

```
apiVersion: v1

kind: Service

metadata:

name: my-java-app-service

spec:

selector:

app: my-java-app

ports:

- protocol: TCP

port: 80      # Port exposed inside the cluster

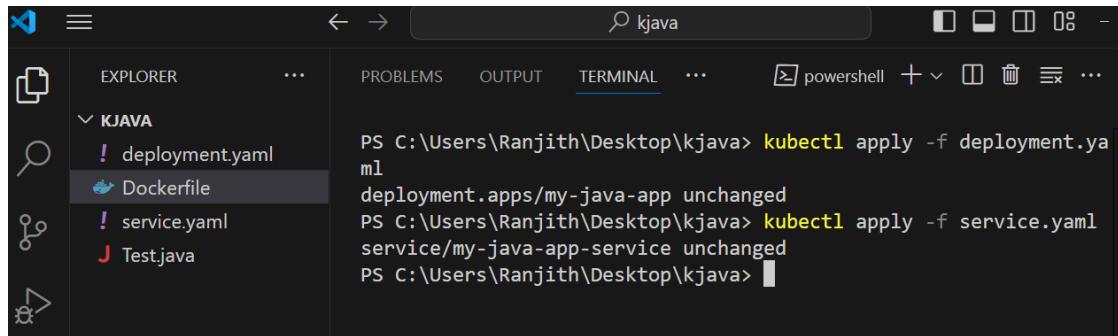
targetPort: 8080 # Port your Java app is running on

nodePort: 30007 # External port
```

**type: NodePort**

Apply the service configuration to your Kubernetes cluster:

**kubectl apply -f service.yaml**

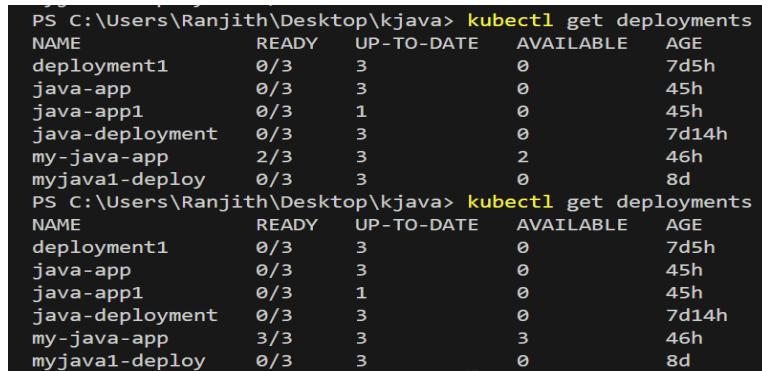


```
PS C:\Users\Ranjith\Desktop\kjava> kubectl apply -f deployment.yaml
deployment.apps/my-java-app unchanged
PS C:\Users\Ranjith\Desktop\kjava> kubectl apply -f service.yaml
service/my-java-app-service unchanged
PS C:\Users\Ranjith\Desktop\kjava>
```

## Step 5: Verify the Deployment

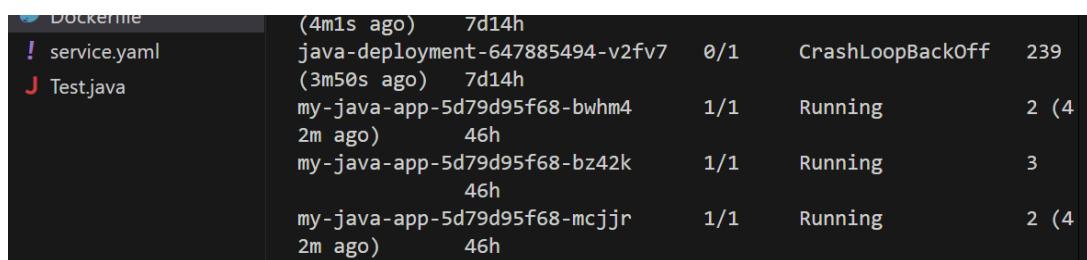
Check the status of your deployment and service:

**kubectl get deployments**



```
PS C:\Users\Ranjith\Desktop\kjava> kubectl get deployments
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
deployment1  0/3     3           0           7d5h
java-app    0/3     3           0           45h
java-app1   0/3     1           0           45h
java-deployment  0/3     3           0           7d14h
my-java-app  2/3     3           2           46h
myjava1-deploy  0/3     3           0           8d
PS C:\Users\Ranjith\Desktop\kjava> kubectl get deployments
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
deployment1  0/3     3           0           7d5h
java-app    0/3     3           0           45h
java-app1   0/3     1           0           45h
java-deployment  0/3     3           0           7d14h
my-java-app  3/3     3           3           46h
myjava1-deploy  0/3     3           0           8d
```

**kubectl get pods**



NAME	READY	UP-TO-DATE	AVAILABLE	AGE
java-deployment-647885494-v2fv7	0/1	CrashLoopBackOff	239	(4m1s ago) 7d14h
my-java-app-5d79d95f68-bwhm4	1/1	Running	2 (4	(3m50s ago) 7d14h
my-java-app-5d79d95f68-bz42k	1/1	Running	3	2m ago) 46h
my-java-app-5d79d95f68-mcjjr	1/1	Running	2 (4	2m ago) 46h

## kubectl get svc

```
✓ KJAVA
! deployment.yaml
Dockerfile
! service.yaml
J Test.java

PS C:\Users\Ranjith\Desktop\kjava> kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP
PORT(S)        AGE
java-app-service   NodePort   10.96.60.200 <none>
                80:30008/TCP 45h
java-app1-service  NodePort   10.107.54.172 <none>
                80:30010/TCP 45h
java-service     LoadBalancer 10.96.207.224 localhost
                80:32029/TCP 7d15h
kubernetes       ClusterIP  10.96.0.1    <none>
                443/TCP    11d
my-java-app-service  NodePort   10.106.150.206 <none>
                80:30007/TCP 46h
myjava1-service   LoadBalancer 10.103.169.241 localhost
                80:31676/TCP 8d
```

## Step 6: Access the Application

To access your application, you need to get the external IP address of the service:

**Open a web browser and navigate to the external IP address to access your application.**

Type: <http://localhost:port> number of the application service

Type in the browser -- <http://localhost:30007> to access the application for the above example



Hello, Kubernetes!

## Install and Explore Selenium for automated testing

Selenium is an open-source, automated testing tool used to test web applications across various browsers. Selenium can be easily deployed on platforms such as Windows, Linux, Solaris and Macintosh.

Selenium supports a variety of programming languages through the use of drivers specific to each language. Languages supported by Selenium include C#, Java, Perl, PHP, Python and Ruby.

Selenium Web driver is most popular with Java and C#.

Selenium test scripts can be coded in any of the supported programming languages and can be run directly in most modern web browsers. Browsers supported by Selenium include Internet Explorer, Mozilla Firefox, Google Chrome and Safari.

Selenium can be integrated with tools like JUnit and TestNG for test management

Selenium has a dedicated suite that facilitates easy testing of web applications.



Installing [Selenium](#) requires going through the following steps:

### **Step 1: Install Java**

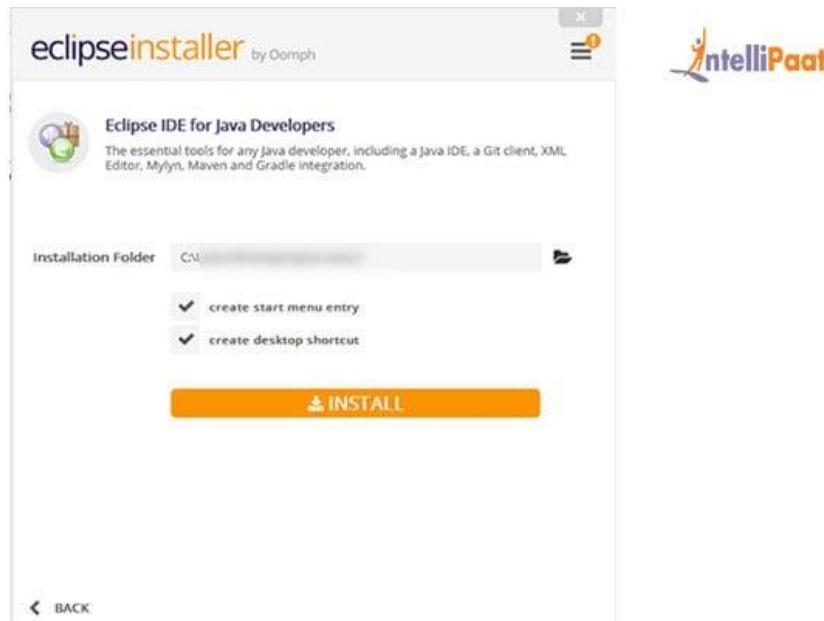
### **Step 2: Download and Configure Eclipse IDE**

- Open URL: <https://www.eclipse.org/downloads/>.
  - Click on the "Download Packages" link-- get "Eclipse IDE for Java Developers"
- Click on the downloaded eclipse set up file**



Select "Eclipse IDE for Java Developers"

"c:\eclipse" and Install.



### Step 3 Download Selenium WebDriver Java Client

- Open URL: <https://docs.seleniumhq.org/download/>  
It will redirect you to the "downloads page" of Selenium official website.
- Scroll down through the web page and locate Selenium Client & WebDriver Language Bindings.
- Click on the "Download" link of Java Client Driver as shown in the image given below.

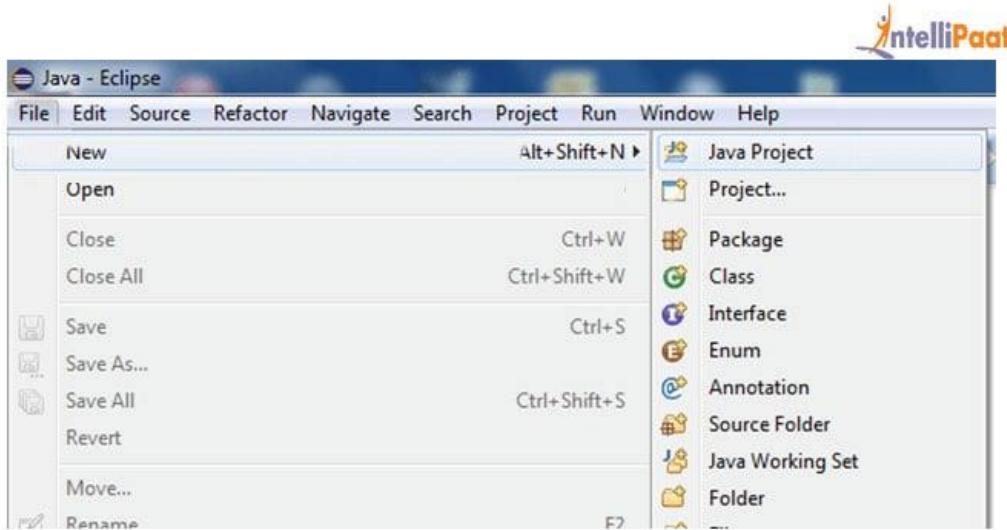


The downloaded file would be in zipped format. Unpack the contents in a convenient directory.

It contains the essential jar files required to configure Selenium WebDriver in Eclipse IDE.

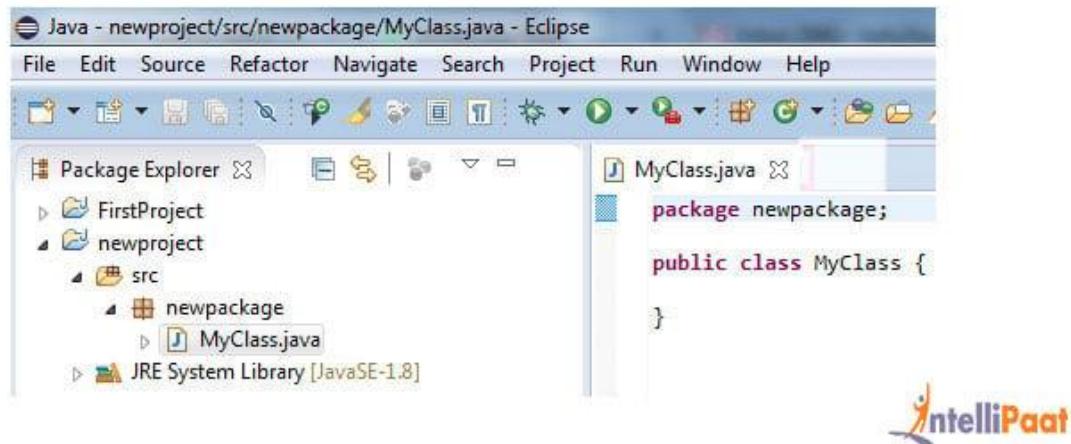
### Step 4 – Configure Eclipse IDE with WebDriver-- Create a new Java Project in Eclipse

Click on the File menu, then go to New and select Java Project. Save the project as 'new project'.



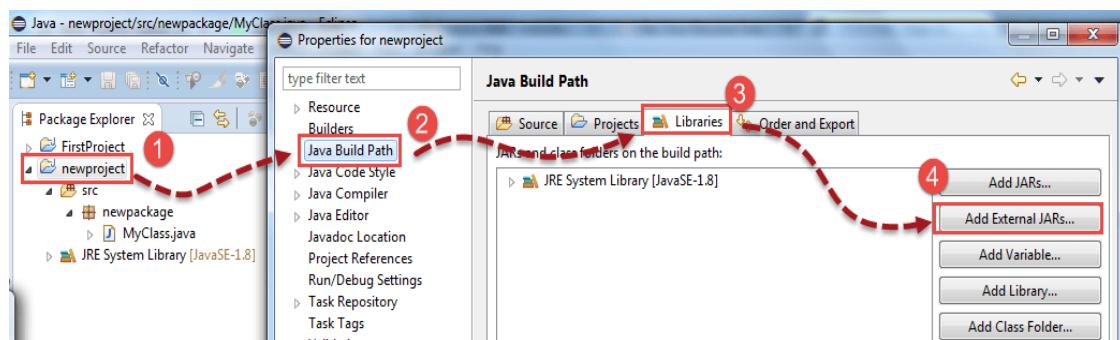
## Create a new Java Class

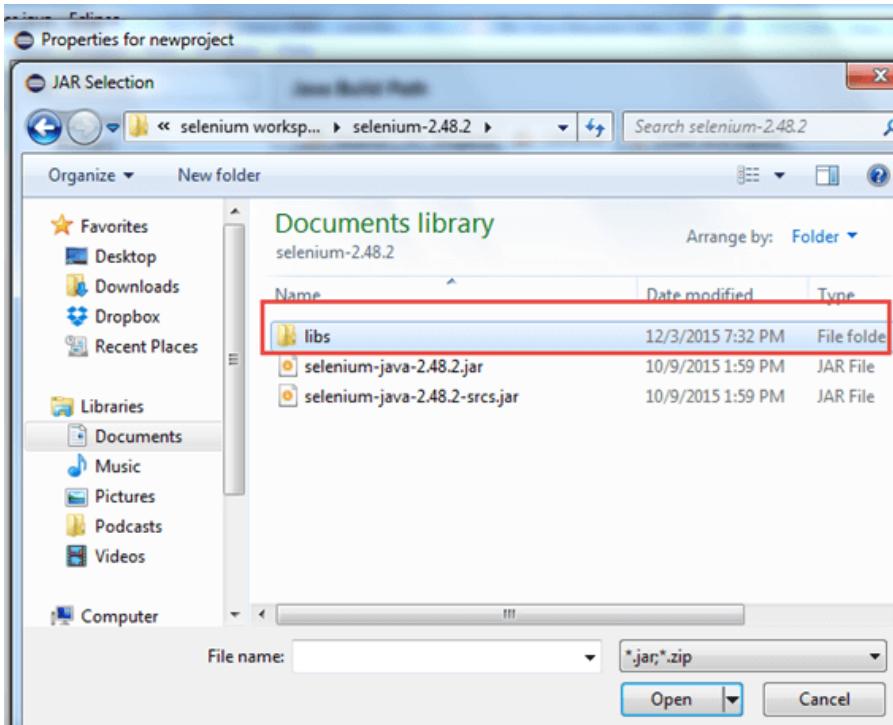
In order to create a new Java Class, click on File Menu and go to New and save it as ‘Myclass’.



set Selenium WebDriver’s libraries into Java Build Path. In this step,

1. Right-click on “newproject” and select **Properties**.
2. On the Properties dialog, click on “Java Build Path”.
3. Click on the **Libraries** tab, and then
4. Click on “Add External JARs.”





Add all the JAR files that are inside and outside the “libs” folder. Now click Ok to finish importing the library files

**Lab Title:****JavaScript Registration Form Testing using Selenium**

---

**Objective:**

To create a simple registration form using HTML and JavaScript, and perform automated functional testing using Selenium.

---

**Step 1: Creating the Registration Form**

1. Open Visual Studio Code.
2. Create a folder named RegistrationForm.
3. Inside the folder, create a file named index.html with the following content:
4. <!DOCTYPE html>
5. <html lang="en">
6. <head>
7.   <meta charset="UTF-8">
8.   <meta name="viewport" content="width=device-width, initial-scale=1.0">
9.   <title>Registration Form</title>
10. <!-- Inline CSS -->
11. <style>
12.   body {
13.     font-family: Arial, sans-serif;
14.     text-align: center;
15.     margin-top: 50px;
16.     background-color: #f5f5f5;
17.   }
18.   form {
19.     display: inline-block;
20.     text-align: left;
21.     background-color: #ffffff;
22.     padding: 20px;
23.     border-radius: 8px;
24.     box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
25.   }
26.   input {
27.     width: 100%;
28.     padding: 10px;
29.     margin: 10px 0;
30.     box-sizing: border-box;
31.     border-radius: 4px;
32.     border: 1px solid #ccc;
33.   }
34.   button {

```
35.     background-color: #4CAF50;
36.     color: white;
37.     padding: 10px 20px;
38.     border: none;
39.     border-radius: 4px;
40.     cursor: pointer;
41. }
42. button:hover {
43.     background-color: #45a049;
44. }
45. #message {
46.     color: green;
47.     font-weight: bold;
48.     margin-top: 20px;
49. }
50. .error {
51.     color: red;
52. }
53. </style>
54. </head>
55. <body>
56.   <h2>Registration Form</h2>
57.   <!-- Registration Form -->
58.   <form id="registrationForm">
59.     <label for="username">Username:</label><br>
60.     <input type="text" id="username" name="username" placeholder="Enter username"
 required><br>
61.
62.     <label for="email">Email:</label><br>
63.     <input type="email" id="email" name="email" placeholder="Enter email" required><br>
64.
65.     <label for="password">Password:</label><br>
66.     <input type="password" id="password" name="password" placeholder="Enter
 password" required><br>
67.
68.     <button type="submit">Register</button>
69.   </form>
70.
71.   <!-- Message Display -->
72.   <p id="message"></p>
73.
74.   <!-- Inline JavaScript -->
75.   <script>
76.     document.getElementById("registrationForm").addEventListener("submit",
 function(event) {
77.       event.preventDefault(); // Prevent form from submitting
78.
79.       // Get input values
```

```

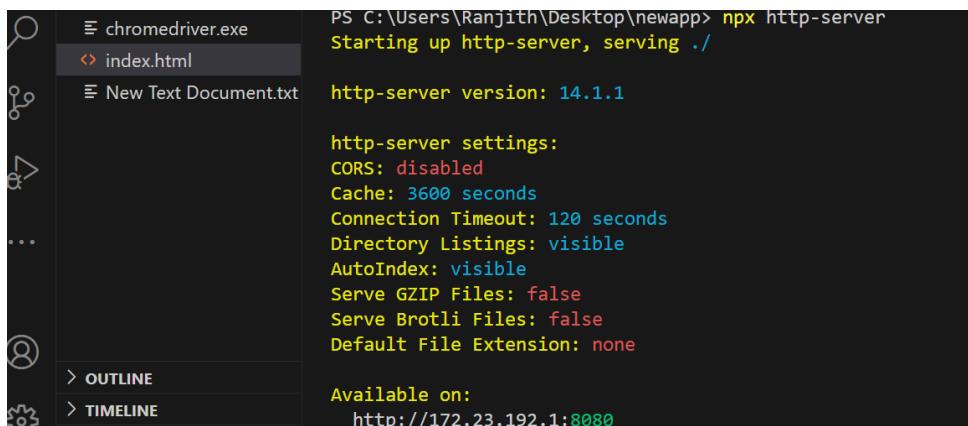
80.     const username = document.getElementById("username").value;
81.     const email = document.getElementById("email").value;
82.     const password = document.getElementById("password").value;
83.     const messageElement = document.getElementById("message");
84.     // Validate inputs
85.     if (username === "" || email === "" || password === "") {
86.         messageElement.textContent = "All fields are required!";
87.         messageElement.classList.add("error");
88.         return;
89.     }
90.     // Check for valid email format
91.     const emailPattern = /^[^@\s]+@[^\s@]+\.[^\s@]+$/;
92.     if (!emailPattern.test(email)) {
93.         messageElement.textContent = "Invalid email format!";
94.         messageElement.classList.add("error");
95.         return;
96.     }
97.     // If all validations pass, show success message
98.     messageElement.textContent = "Registration Successful!";
99.     messageElement.classList.remove("error");
100.    messageElement.style.color = "green";
101.
102.    // Clear the form
103.    document.getElementById("registrationForm").reset();
104.);
105. </script>
106. </body>
107. </html>

```

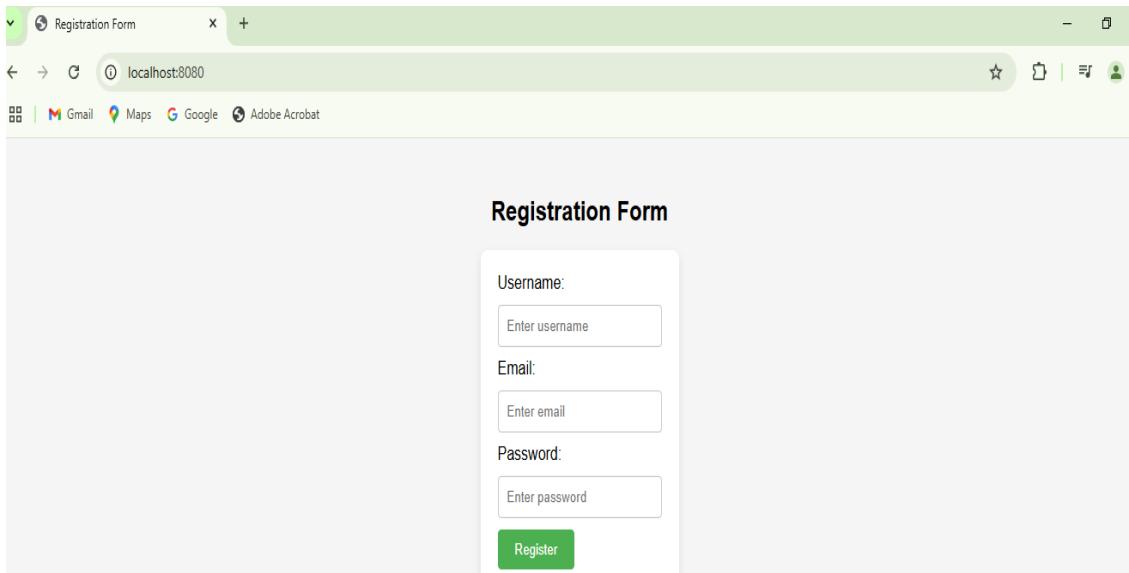
## Step 2: Running the Web Application

1. Open the terminal in Visual Studio Code
2. Use the following command to start a simple HTTP server:

npx http-server



3. Open your browser and go to <http://localhost:8080> to view the form.



## Part 2: Testing the Registration Form using Selenium

### Step 1: Setting up Selenium in Node.js

1. Initialize a Node.js project:

```
npm init -y
```

```
Wrote to C:\Users\Ranjith\Desktop\newapp\package.json:
{
  "name": "newapp",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \\"Error: no test specified\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

A screenshot of a terminal window in Visual Studio Code. The terminal tab is active and shows the command 'npm init -y' being run. The output indicates that a package.json file was created in the directory C:\Users\Ranjith\Desktop\newapp, containing basic project metadata like name, version, main file, and a test script.

2. Install the Selenium WebDriver package:

```
npm install selenium-webdriver
```

```
PS C:\Users\Ranjith\Desktop\newapp> npm install selenium-webdriver
```

A screenshot of a terminal window in PowerShell. The command 'npm install selenium-webdriver' is being typed into the terminal. The prompt shows 'PS C:\Users\Ranjith\Desktop\newapp>'.

3. Download the ChromeDriver that matches your Chrome browser version from ChromeDriver Downloads.
4. Extract the downloaded chromedriver and place it in the project folder.

### Step 2: Writing the Selenium Test Script

Create a file named test.js and add the following code:

javascript

Copy code

```
const { Builder, By, Key, until } = require('selenium-webdriver');

async function testRegistrationForm() {
    // Set up WebDriver
    let driver = await new Builder().forBrowser('chrome').build();
    try {
        // Navigate to the registration form
        await driver.get('http://localhost:8080');

        // Enter username
        await driver.findElement(By.id('username')).sendKeys('student1');

        // Enter email
        await driver.findElement(By.id('email')).sendKeys('student1@example.com');

        // Enter password
        await driver.findElement(By.id('password')).sendKeys('password123');

        // Click the Register button
        await driver.findElement(By.css('button[type="submit"]')).click();

        // Wait for the success message to appear
        await driver.wait(until.elementLocated(By.id('message')), 5000);

        // Get the message text
        let message = await driver.findElement(By.id('message')).getText();
        console.log('Test Result:', message);

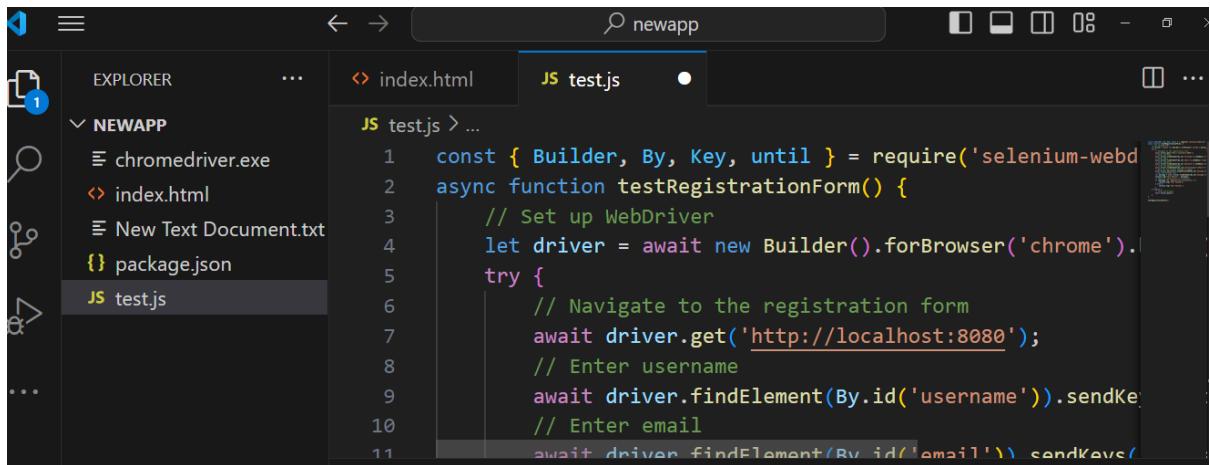
        // Check if registration is successful
        if (message === 'Registration Successful!') {
            console.log('Test Passed');
        } else {
            console.log('Test Failed');
        }
    } finally {
        // Close the browser
        await driver.quit();
    }
}
```

```
}
```

```
}
```

```
testRegistrationForm();
```



```
JS test.js
```

```
1 const { Builder, By, Key, until } = require('selenium-webdriver');
2 async function testRegistrationForm() {
3     // Set up WebDriver
4     let driver = await new Builder().forBrowser('chrome').build();
5     try {
6         // Navigate to the registration form
7         await driver.get('http://localhost:8080');
8         // Enter username
9         await driver.findElement(By.id('username')).sendKeys('testuser');
10        // Enter email
11        await driver.findElement(By.id('email')).sendKeys('testuser@example.com');
```

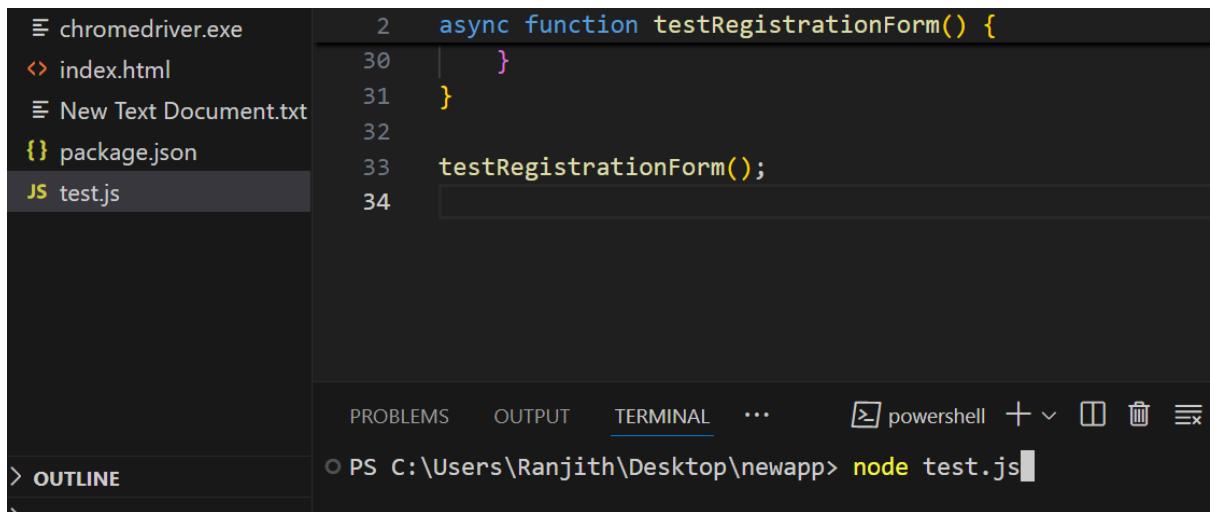
### Step 3: Running the Selenium Test

1. Ensure your HTTP server is running:

```
npx http-server
```

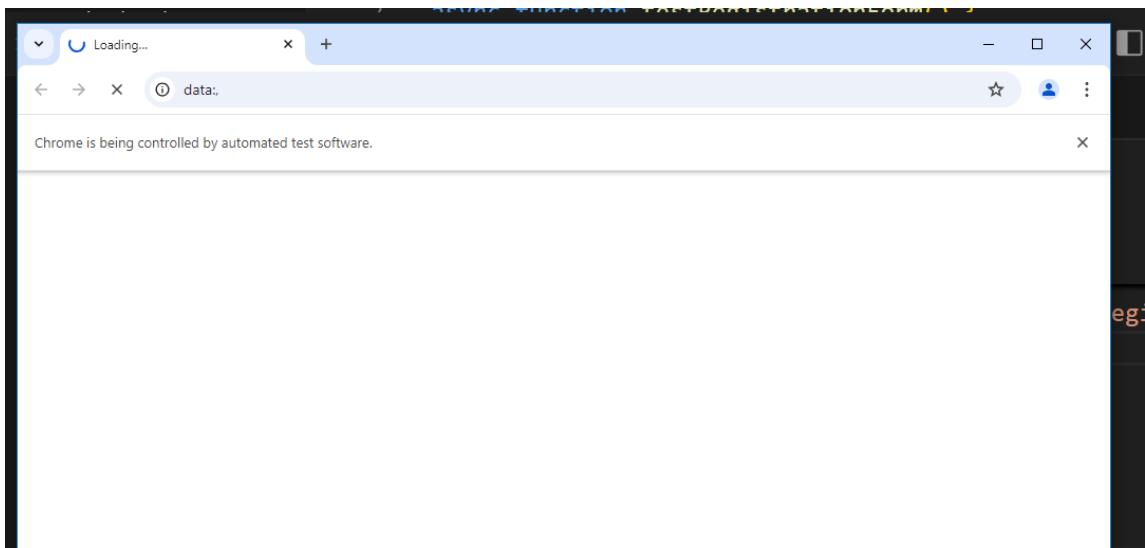
2. Open a new terminal and run the Selenium test script:

```
node test.js
```



```
PROBLEMS OUTPUT TERMINAL ...
```

```
PS C:\Users\Ranjith\Desktop\newapp> node test.js
```



← → ⌛ ⓘ localhost:8080

Chrome is being controlled by automated test software.

### Registration Form

Username:

Email:

Password:

---

#### Expected Output:

The console should display:

Test Result: Registration Successful!

Test Passed

The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the command PS C:\Users\Ranjith\Desktop\newapp> node test.js and its output: DevTools listening on ws://127.0.0.1:49967/devtools/browser/7e29b, Test Result: Registration Successful!, and Test Passed. The file index.html is highlighted in the Explorer sidebar.

#### Step 4: Observations and Conclusion

- The automated test checks if the registration form functions correctly by filling in sample data and verifying the success message.
- If the form works as expected, the test will pass; otherwise, it will fail.

---

#### Viva Questions:

1. What is Selenium used for?
2. What is the difference between functional and non-functional testing?
3. How can you automate browser testing using JavaScript?
4. What are the benefits of automated testing?

---

#### Assignment:

1. Modify the registration form to include additional fields like phone number and address.
2. Write a new Selenium test case to validate the new fields.

**Week 12:**

**Understand the Application:** Identify the functionality and purpose of the containerized app. For example, if it's a web application, determine its main workflows (e.g., login, search, CRUD operations).

**② Set Up Selenium Environment:**

- *Install Selenium in your test environment.*
- *Install the necessary dependencies:*
- `pip install selenium`
- *Use a compatible WebDriver (e.g., ChromeDriver or GeckoDriver) for the browser.*
- *Ensure the containerized app is running and accessible (via localhost or a specific IP).*
- *Ensure the application container is up and running (e.g., docker run -d -p 8080:80 your-app).*

**③ Create Test Scenarios:** Write user stories or cases describing expected behavior.

**④ Implement Test Cases in Selenium:** Use programming languages like Python or Java for scripting.

**⑤ Generate Output Screenshots:** Use Selenium's screenshot functionality to capture results.

*Example Test Case: Containerized Web App with Login Page*

*Scenario: Verify login functionality.*

*Steps:*

*Open the application URL.*

*Enter username and password.*

*Click the login button.*

*Assert the user is redirected to the dashboard page.*

*Selenium Script (Python Example)*

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
```

```
# Set up the WebDriver
driver = webdriver.Chrome() # Ensure chromedriver is installed and in PATH
driver.get("http://localhost:8080") # Replace with your app's URL

# Step 1: Open the application
print("Navigating to the application...")

# Step 2: Locate and interact with the login form
username_field = driver.find_element(By.ID, "username") # Replace with actual ID
password_field = driver.find_element(By.ID, "password") # Replace with actual ID
login_button = driver.find_element(By.ID, "loginButton") # Replace with actual ID

username_field.send_keys("testuser")
password_field.send_keys("password123")
time.sleep(1) # Pause for visibility

# Take a screenshot before login
driver.save_screenshot("screenshots/login_page.png")

# Step 3: Submit the form
login_button.click()

# Step 4: Assert and validate
time.sleep(3) # Wait for the page to load
dashboard_title = driver.title
assert "Dashboard" in dashboard_title, "Login failed or incorrect page loaded"

# Take a screenshot after login
```

```
driver.save_screenshot("screenshots/dashboard.png")  
  
print("Login test passed.")  
  
# Clean up  
driver.quit()
```

### *Outputs*

#### *Screenshots:*

*Before Login:* A screenshot of the login form filled.

*After Login:* A screenshot of the dashboard page.

#### *Logs:*

*Include printed logs such as "Navigating to the application..." or "Login test passed."*

### **Additional Test Cases**

#### *Invalid Login:*

*Enter invalid credentials.*

*Assert error message is displayed.*

#### *UI Element Availability:*

*Verify that all UI elements (buttons, fields, links) load correctly.*

#### *Form Validation:*

*Leave fields empty and attempt login.*

*Verify error messages appear.*

#### *Responsive Design:*

*Test on different screen sizes using Selenium's window resize method.*

### **Basic Test Cases:**

## 1. Verify Home Page Loads

**Steps:**

    Navigate to the application URL (e.g., `http://localhost:8080`).

    Verify the title of the page.

**Code:**

```
from selenium import webdriver

driver = webdriver.Chrome() # Use the driver for your browser

driver.get("http://localhost:8080")

assert "Expected Title" in driver.title, "Home page title did not match"

driver.quit()
```

## 2. Login Functionality

**Steps:**

    Open the login page.

    Enter valid credentials.

    Submit the form.

    Verify redirection or a success message.

**Code:**

```
from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.common.keys import Keys


driver = webdriver.Chrome()

driver.get("http://localhost:8080/login")

driver.find_element(By.ID, "username").send_keys("test_user")

driver.find_element(By.ID, "password").send_keys("secure_password")

driver.find_element(By.ID, "login-button").click()

success_message = driver.find_element(By.ID, "success-message").text

assert "Welcome" in success_message, "Login failed"
```

*driver.quit()*