

CMR TECHNICAL CAMPUS UGC AUTONOMOUS

Kandlakoya(V),Medchal Road,

Hyderabad-501401

Accredited by NBA and NAAC with A Grade



DEVOPS LAB MANUAL

COURSE CODE:22CS504PC

CMRTECHNICALCAMPUS

Kandlakoya(V),MedchalRoad,Hyderabad-501 401

An UGC Autonomous Institute

Accredited by NBA and NAAC with A Grade

Approved by AICTE, New Delhi and Affiliated to JNTU, Hyderabad

LIST OF EXPERIMENTS:

1. Write code for a simple user registration form for an event.
2. Explore Git and GitHub commands.
3. Practice Source code management on GitHub. Experiment with the source code written in exercise 1.
4. Jenkins installation and setup, explore the environment.
5. Demonstrate continuous integration and development using Jenkins.
6. Explore Docker commands for content management.
7. Develop a simple containerized application using Docker.
8. Integrate Kubernetes and Docker
9. Automate the process of running containerized application developed in exercise 7 using Kubernetes.
10. Install and explore Selenium for automated testing.
11. Write a simple program in JavaScript and perform testing using Selenium.
12. Develop test cases for the above containerized application using selenium

Experiment 1: Create a Simple Event Registration Form using HTML & JavaScript

Aim:

To create a basic event registration form using HTML and JavaScript.

Procedure:

1. Open Visual Studio Code.
2. Create a new HTML file and copy the given code into it.
3. Save and run the file in a browser to test the form.

Code:

```
<!DOCTYPE html>
<html>
<head><title>Event Registration</title></head>
<body>
<form id="registrationForm" onsubmit="return validateForm()">
<label for="name">Full Name</label>
<input type="text" id="name" required>
<label for="email">Email</label>
<input type="email" id="email" required>
<label for="phone">Phone Number</label>
<input type="tel" id="phone" required pattern="[0-9]{10}">
<label for="event">Select Event</label>
<select id="event" required>
<option value="">--Select--</option>
<option value="workshop">Workshop</option>
<option value="seminar">Seminar</option>
<option value="conference">Conference</option>
</select>
<button type="submit">Register</button>
</form>
<script>
function validateForm() {
    const phone = document.getElementById('phone').value;
    if (!/\d{10}/.test(phone)) {
        alert('Invalid phone number');
        return false;
    }
    alert('Registration Successful!');
    return true;
}
```

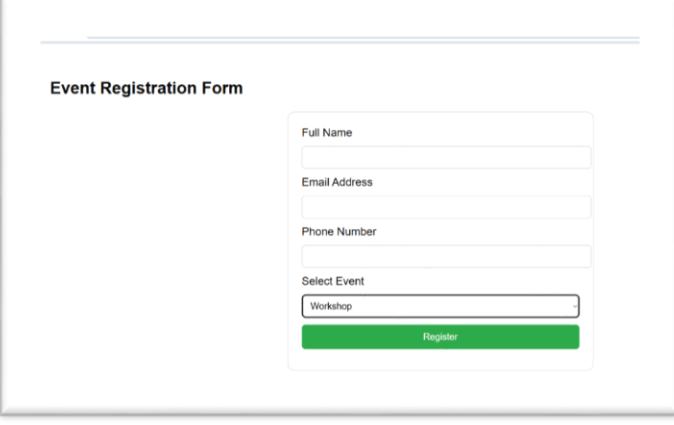
```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

Output



The image shows a clean, modern event registration form titled "Event Registration Form". The form is contained within a light gray rectangular box with rounded corners. It features four input fields: "Full Name", "Email Address", and "Phone Number", each with its own label and a corresponding text input field below it. Below these is a dropdown menu labeled "Select Event" with the option "Workshop" selected. At the bottom right is a prominent green rectangular button with the word "Register" in white capital letters.

Experiment 2: Explore Git and GitHub Commands

Aim:

To explore basic Git and GitHub commands.

Procedure:

Git and GitHub are two of the most popular tools used for version control and collaboration in software development.

Here are some common Git and GitHub commands:

- Initializing a Git repository: `$ git init`
- Checking the status of your repository: `$ git status`
- Adding files to the stage: `$ git add <file-name>`
- Committing changes: `$ git commit -m "commit message"`
- Checking the commit history: `$ git log`
- Undoing changes: `$ git checkout <file-name>`
- Creating a new branch: `$ git branch <branch-name>`
- Switching to a different branch: `$ git checkout <branch-name>`
- Merging two branches: `$ git merge <branch-name>`
- Pushing changes to a remote repository: `$ git push origin <branchname>`
- Cloning a repository from GitHub: `$ git clone <repository-url>`

Creating a pull request on GitHub: Go to the repository on GitHub, select the branch you want to merge and click the "New pull request" button.

These are just a few of the many Git and GitHub commands available. There are many other Git commands and functionalities that you can explore to suit your needs.

Experiment 3: Practice Source Code Management on GitHub

Aim:

To push and manage source code using GitHub.

Procedure:

To practice source code management on GitHub, you can follow these steps:

Create a GitHub account if you don't already have one.

- Create a new repository on GitHub.
- Clone the repository to your local machine: `$ git clone <repositoryurl>`
- Move to the repository directory: `$ cd <repository-name>`
- Create a new file in the repository and add the source code written in exercise 1.
- Stage the changes: `$ git add <file-name>`
- Commit the changes: `$ git commit -m "Added source code for a simple user registration form"`
- Push the changes to the remote repository: `$ git push origin master`

Verify that the changes are reflected in the repository on GitHub.

These steps demonstrate how to use GitHub for source code management. You can use the same steps to manage any source code projects on GitHub. Additionally, you can also explore

GitHub features such as pull requests, code review, and branch management to enhance your source code management workflow.

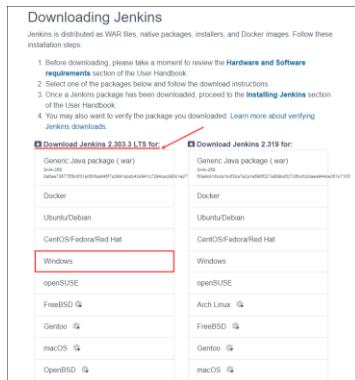
Experiment 4: Install Jenkins on Windows

Aim:

To install and configure Jenkins on Windows OS.

Procedure:

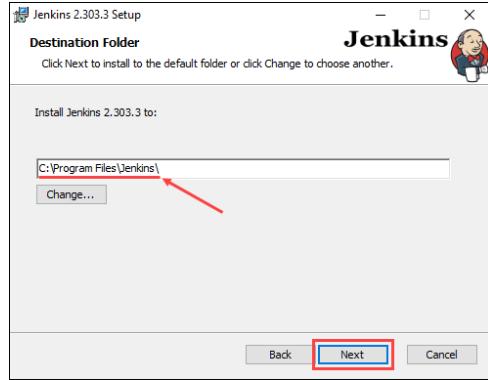
- Download Jenkins from the official site.



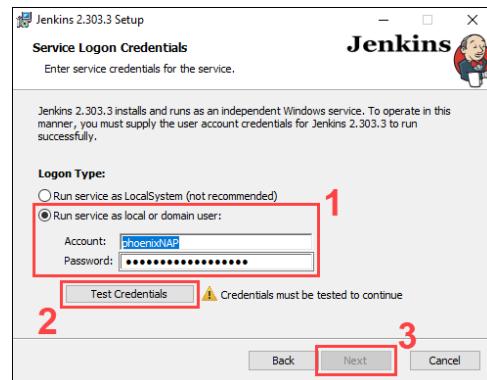
- Run the **jenkins.msi** installation file.
- The setup wizard starts
- . Click **Next** to proceed.



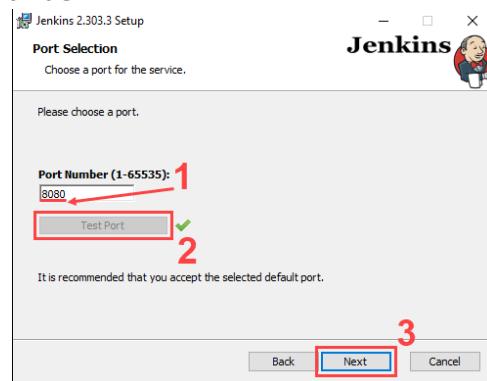
- Select the install destination folder and click **Next** to continue



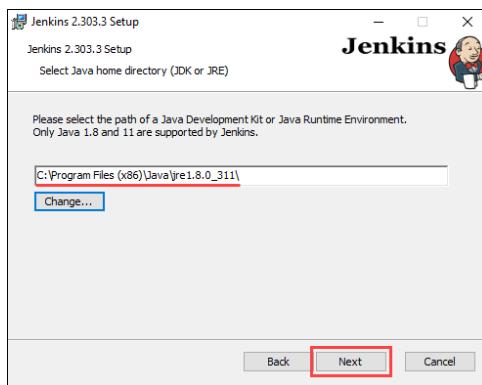
- Under the **Run service as a local or domain user** option,
- Click **Test Credentials** to verify the login data,
- then click **Next** to proceed.



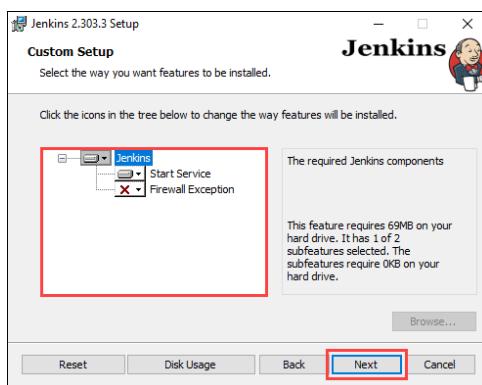
- Enter the port number you want Jenkins to run on.
- Click **Test Port** to check if the selected port is available,
- then click **Next** to continue.



- Select the directory where **Java is installed** on your system and click **Next** to proceed



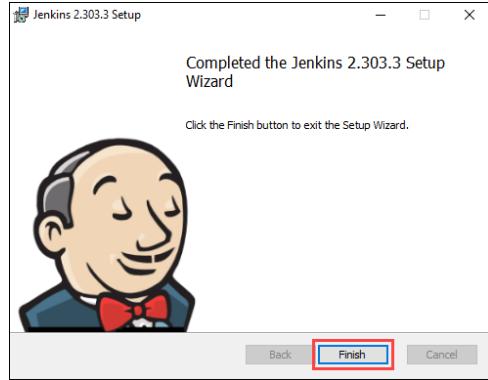
- Select the features you want to install with Jenkins and click **Next** to continue.



- Click **Install** to start the installation process.



- Once the installation is complete,
- click **Finish** to exit the install wizard.



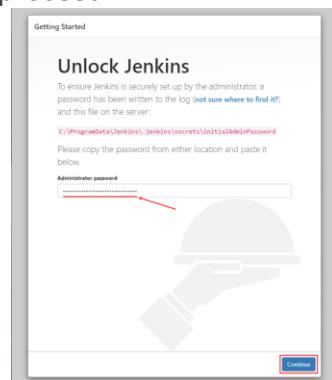
4. Run the installer and follow the steps.
5. Unlock Jenkins and install suggested plugins.



- Open the **initialAdminPassword** file using a text editor such as Notepad.
- Copy the password from the **initialAdminPassword** file.

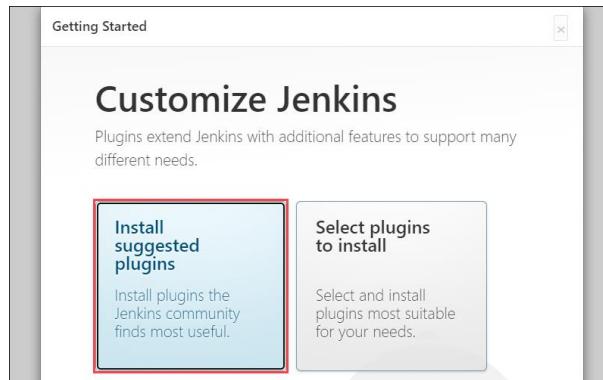


- Paste the password in the **Administrator password** field on the Unblock Jenkins page and click **Continue** to proceed.

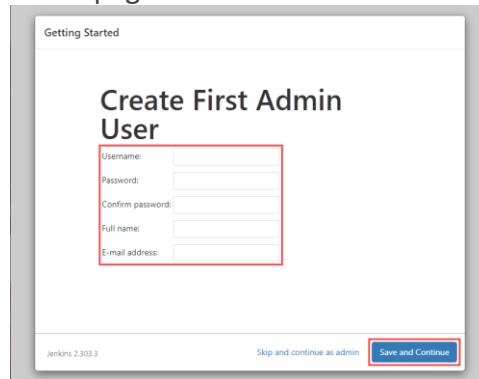


Customize Jenkins

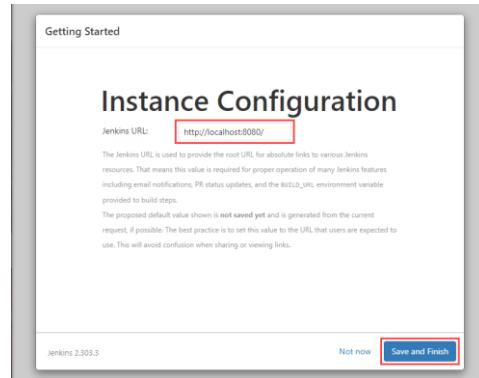
- Once you unlock Jenkins, customize and prepare the Jenkins environment.
- Click the **Install suggested plugins** button to have Jenkins automatically install the most frequently used plugins.



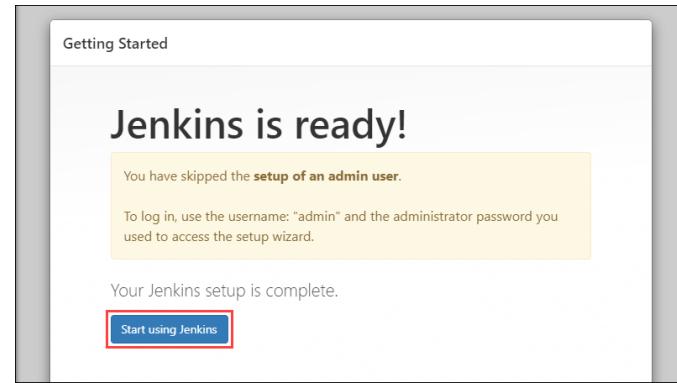
- clicking the **Select plugins to install** button allows you to choose which plugins to install.
- After Jenkins finishes installing the plugins, enter the required information on the **Create First Admin User** page. Click **Save and Continue** to proceed.



- On the **Instance Configuration** page, confirm the port number you want Jenkins to use and click **Save and Finish** to finish the initial customization.



- Click the **Start using Jenkins** button to move to the Jenkins dashboard.



Experiment 5: Integrate Java Project with Jenkins

Aim:

To create a Java program and automate build via Jenkins.

Procedure:

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

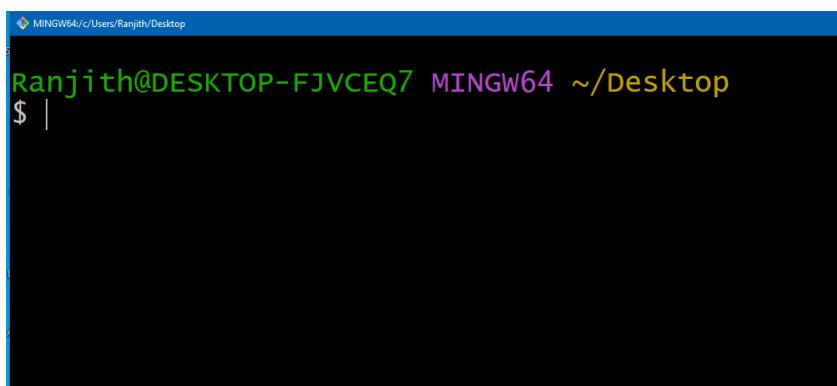
Create a simple java program with file name as Demo.java and save it in a folder on desktop

```
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
 - Login into GitHub account and create a new public repository
 - After login, click on repositories and click on New button to create the repository
 - Right click on the desktop and select open Gitbash here

Gitbash window opens



A screenshot of a terminal window titled 'MINGW64 / c:/Users/Ranjith/Desktop'. The window shows the command line prompt 'Ranjith@DESKTOP-FJVCEQ7 MINGW64 ~/Desktop \$ |'.

```
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)  
$ |
```

- 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`
- 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

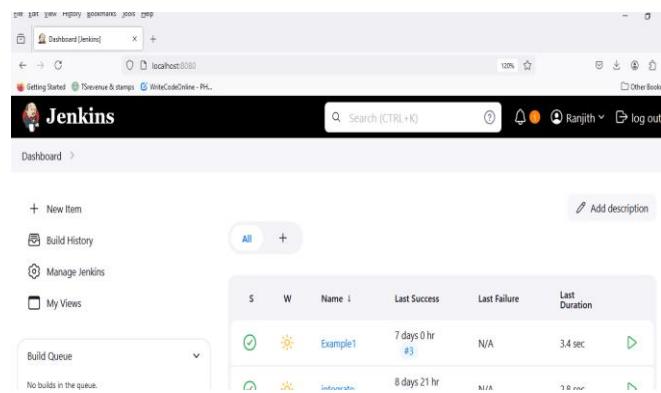
vii) **git push -u origin master**

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 Jenkinsserver.

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 Gitrepository.**
- **Specify the build triggers, such as building after every commit to the repository.**



The screenshot shows the Jenkins dashboard. At the top, there's a navigation bar with links like 'Dashboard', 'Jenkins', 'New Item', 'Build History', 'Manage Jenkins', and 'My Views'. Below the navigation bar, there's a search bar and a user profile icon for 'Ranjith'. The main area is titled 'Dashboard' and features a 'Build Queue' section which says 'No builds in the queue.' To the right, there's a table showing two build jobs: 'Example1' and 'InfoTables'. The 'Example1' job has a green status icon, a success rate of 7 days 0 hr, and a duration of 3.4 sec. The 'InfoTables' job has a yellow status icon, a failure rate of 8 days 21 hr, and a duration of N/A. There are also 'Add description' and '+' buttons.

Click on New Item to create a new job

The screenshot shows the Jenkins interface for creating a new item. At the top, there's a navigation bar with the Jenkins logo, a search bar, and user information (Ranjith). Below it, the breadcrumb path 'Dashboard > All > New Item' is shown. The main title 'New Item' is centered above a form. The first field is 'Enter an item name' with the value 'Job3'. A modal dialog is displayed, asking 'Select an item type' with the option 'Freestyle project' selected. This option is described as a '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.' At the bottom of the modal is a blue 'OK' button.

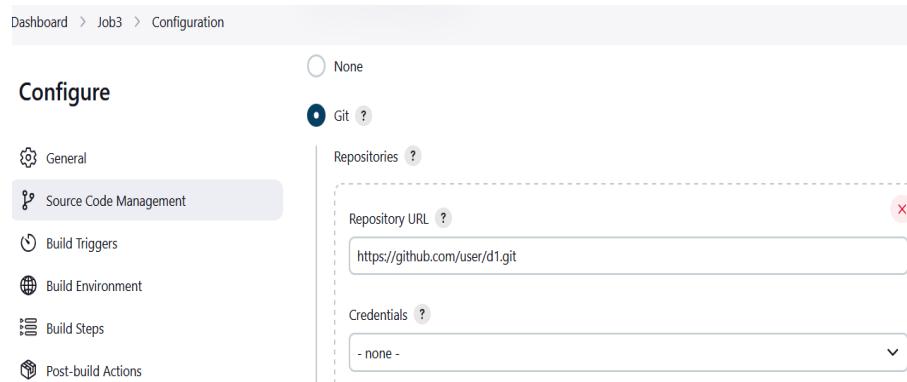
Give the **Item name**, select the item type as**Freestyle project** and click **OK** button

This screenshot shows the Jenkins configuration page for 'Job3'. The browser title is 'Job3 Config [Jenkins]'. The main content area has tabs: 'Configure' (selected), 'General' (disabled), and 'Script'. Under 'General', there's a 'Description' text area which is currently empty. To its right is an 'Enabled' toggle switch, which is turned on. On the left side, there's a sidebar with several options: 'General' (selected), 'Source Code Management' (disabled), 'Build Triggers' (disabled), 'Build Environment' (disabled), 'Build Steps' (disabled), and 'Post-build Actions' (disabled). The URL in the browser is 'localhost:8080/job/job3/configure'.

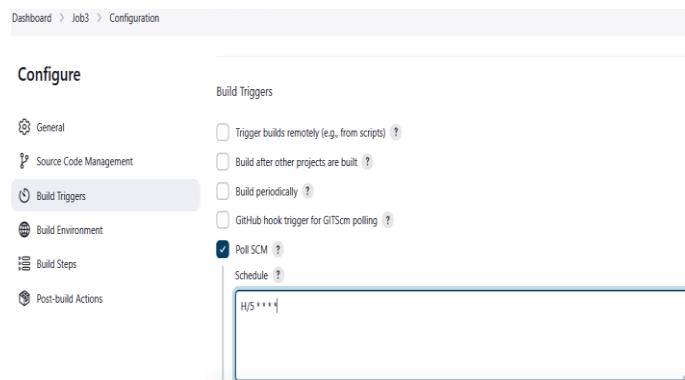
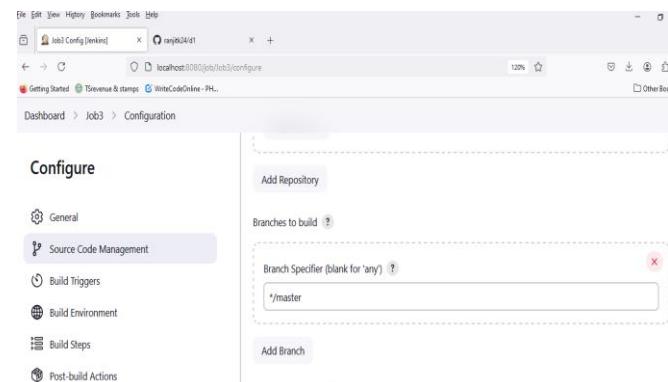
Give the Description of the project or program

This screenshot shows the Jenkins configuration page for 'Job3', specifically the 'Source Code Management' section. The browser title is 'Job3 Config [Jenkins]'. The main content area has tabs: 'Configure' (selected), 'General' (disabled), and 'Script'. Under 'Source Code Management', there are two options: 'None' (radio button) and 'Git' (radio button, selected). Below the 'Git' section is a 'Repositories' button with a tooltip. A modal dialog is open, showing a 'Repository URL' input field with the value 'https://github.com/ranjithreddy/test-repo.git'. There is a red 'X' icon in the top right corner of this dialog. On the left side, there's a sidebar with several options: 'General' (disabled), 'Source Code Management' (selected), 'Build Triggers' (disabled), 'Build Environment' (disabled), 'Build Steps' (disabled), and 'Post-build Actions' (disabled). The URL in the browser is 'localhost:8080/job/job3/configure'.

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



In branches to build option, mention branch specifier as master



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

Configure

General

Source Code Management

Build Triggers

Build Environment

Build Steps

Post-build Actions

Build Steps

Add build step ^

Filter

Execute Windows batch command

Execute shell

Invoke Ant

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

Dashboard > Job3 > Configuration

With Ant ?

Configure

General

Source Code Management

Build Triggers

Build Environment

Build Steps

Post-build Actions

Build Steps

Execute Windows batch command ?

Command

See the list of available environment variables

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

Advanced ▾

Click on Save to save the job

The screenshot shows the Jenkins configuration page for a job named 'Job3'. The 'Build Steps' section is expanded, showing a single step: 'Execute Windows batch command'. The 'Command' field contains the following two lines of code:

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

Below the command field, there is an 'Advanced' dropdown menu.

Now click on Build Now to build the job'

The screenshot shows the Jenkins dashboard for Job3. At the top, there are links for 'Dashboard', 'Job3', and 'Permalinks'. Below these are several actions: 'Build Now' (with a play icon), 'Configure' (with a gear icon), 'Delete Project' (with a trash bin icon), 'Git Polling Log' (with a document icon), and 'Rename' (with a pencil icon). A 'Build History' section is displayed, showing one entry: '#1 Sep 21, 2024, 1:55 PM'. It includes links for 'Atom feed for all' and 'Atom feed for failures', along with up and down navigation arrows.

Click on the Job number from the Build History

This screenshot shows the details for build #1 of Job3. On the left, there are links for 'Changes', 'Console Output', 'Edit Build Information', 'Delete build #1', 'Timings', 'Git Build Data', and 'No Tags'. To the right, it shows that the build was started by user 'Ranjith' and spent 8.8 seconds. It lists three items under 'Timings'. Below that, it shows the 'git' revision and repository information: Revision: 72471d648e224a0bc2ea769bc886c1d9a69e0ac2, Repository: https://github.com/ranjitk24/d1.git, and a note about refs/remotes/origin/master. At the bottom, it says '</> No channels.'

Click on the console output to view the output of Job

This screenshot shows the Jenkins console output for build #1 of Job3. The log starts with a series of git commands to fetch and config the repository. It then indicates a 'First time build. Skipping changelog.' followed by the command 'cmd /c call C:\WINDOWS\TEMP\jenkins11303450433661217744.bat'. The build then runs 'javac Demo.java' and 'java Demo', displaying the output 'Hello world'. Finally, it exits with the status 'Finished: SUCCESS'.

```
> 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: Basic Docker Commands for Content Management

Aim:

To learn how to use simple Docker commands to manage applications inside containers.

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

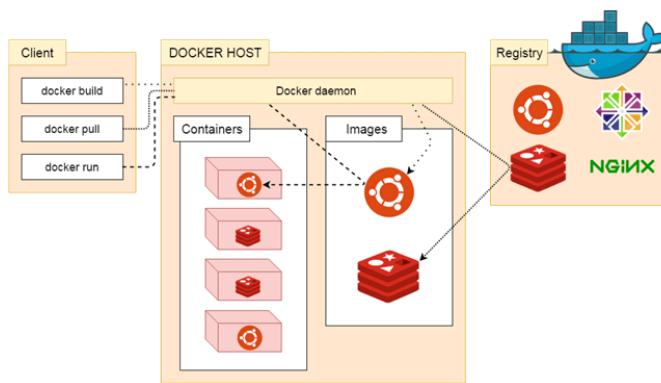
- **Docker Engine:** It is a core part of docker, that handles the creation and management of containers.
- **Container:** A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another
- **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**

```
PS C:\> docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
e966ebcc530a hello-world "/hello" 4 minutes ago Exited (0) 4 minutes ago
fled_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 7: Create a Simple Java 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
```

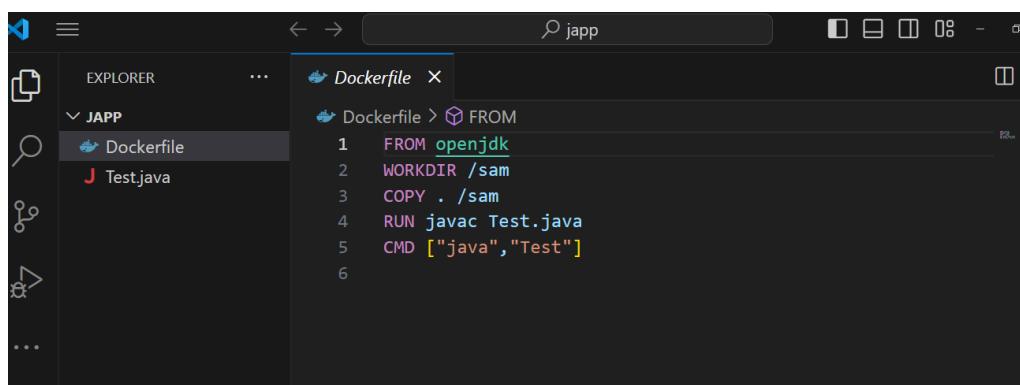
```
WORKDIR /sam# Set the working directory
```

```
COPY. /sam# Copies all the files to working directory
```

```
RUN javac Test.java# Compile the Java program
```

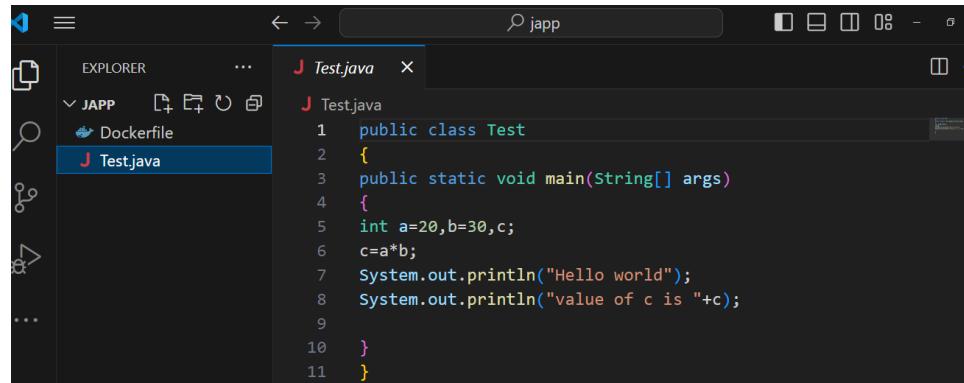
```
CMD ["java", "Test"]# Set the default command to run the Java program
```

Save the Java file and Docker file



The screenshot shows a code editor interface with a dark theme. On the left, there's an 'EXPLORER' sidebar containing a folder named 'JAPP' which contains 'Dockerfile' and 'Test.java'. The main area is a code editor with the following content:

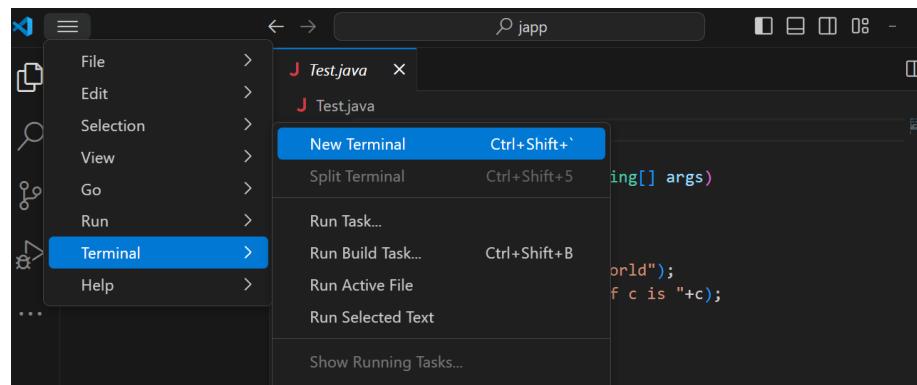
```
FROM openjdk
WORKDIR /sam
COPY . /sam
RUN javac Test.java
CMD ["java", "Test"]
```



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays a project structure with a folder 'JAPP' containing 'Dockerfile' and 'Test.java'. The 'Test.java' file is currently selected. The main workspace on the right shows the code for 'Test.java':

```
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 }
11 }
```

Start the New Terminal from Visual studio



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

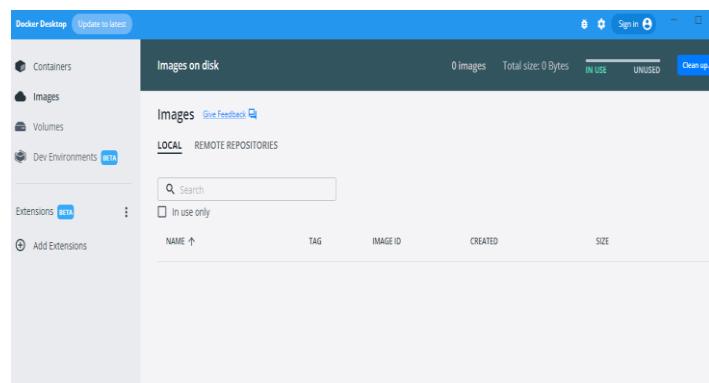
The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a project named 'JAPP' containing a 'Dockerfile' and a 'Test.java' file, which is currently selected. In the center, there's a code editor with the following Java code:

```
1 public class Test
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello, Docker!");
6     }
}
```

Below the code editor is a terminal window showing the command `docker images` being run in a PowerShell session. The output is as follows:

```
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 .

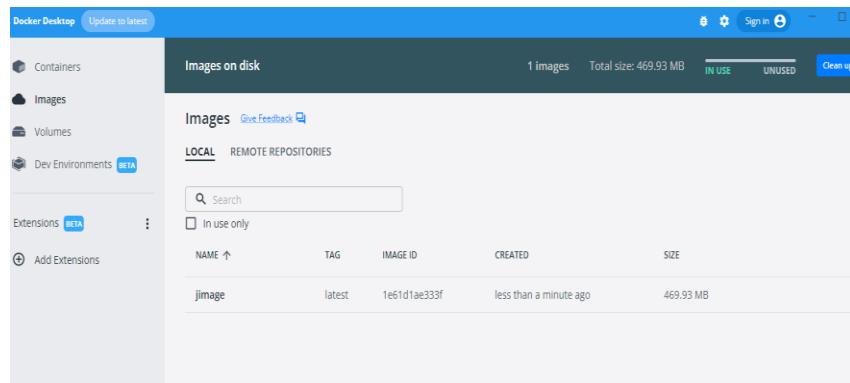
```

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



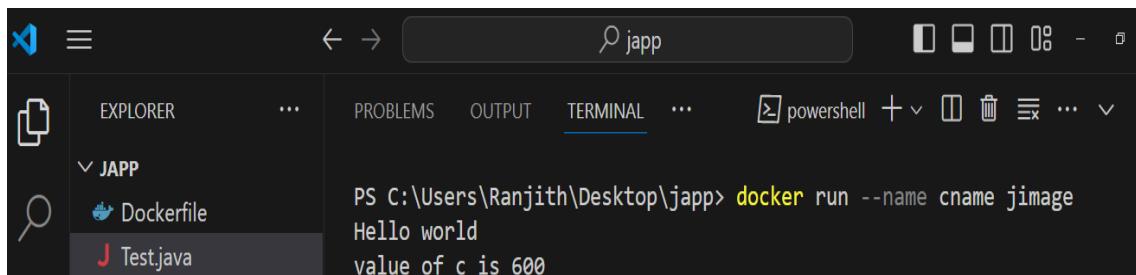
Run the Docker container:

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 says "japp". The left sidebar shows a project named "JAPP" with files "Dockerfile" and "Test.java". The main area is a terminal window with the following text:

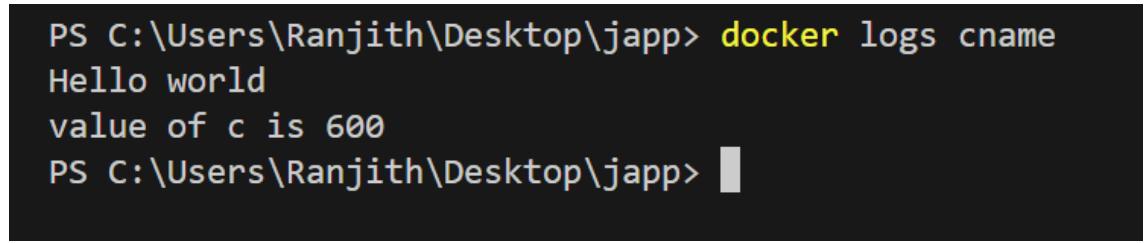
```
PS C:\Users\Ranjith\Desktop\japp> docker run --name cname jimage
Hello world
value of c is 600
```

- Verify the output:

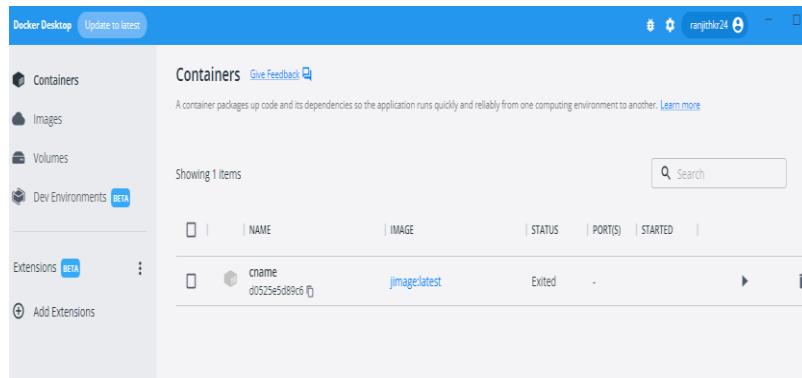
Run the following command to verify the output of the container:

```
$ 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 itas public or private

Tag the Imageuse the command

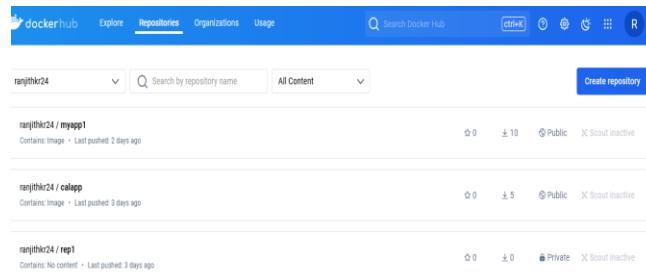
```
docker tag <image name><dockerhub-user name>/<repo-name><version>
```

Log into Docker

```
docker login give login credentials
```

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

Login into Docker hub



Tagging docker image and login into docker hub

A screenshot of a terminal window in Visual Studio Code. The title bar says 'japp'. The left sidebar shows an 'EXPLORER' view with a folder named 'JAPP' containing 'Dockerfile' and 'Test.java'. The terminal tab is active, showing PowerShell output. The commands entered are:

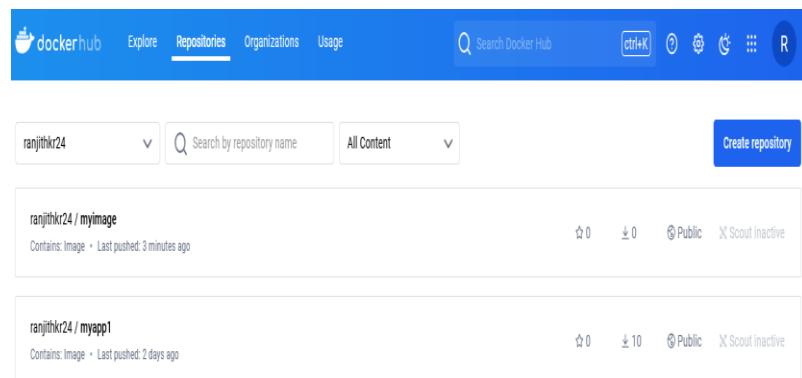
```
PS C:\Users\Ranjith\Desktop\japp> docker tag jimage ranjithkr24/myimage :v1.0
PS C:\Users\Ranjith\Desktop\japp> docker login
Authenticating with existing credentials...
Login Succeeded
```

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



Week 8: Integrate Kubernetes with Docker

Aim:

To enable Kubernetes on Docker Desktop and prepare system to run Kubernetes commands.

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

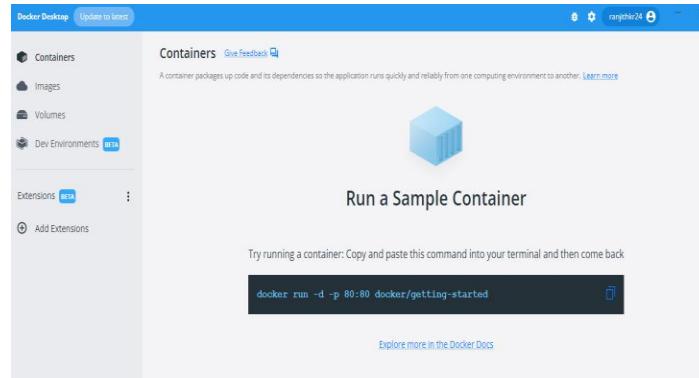
Docker is used for **building, shipping, and running containers**—it provides the means to create isolated environments for applications.

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

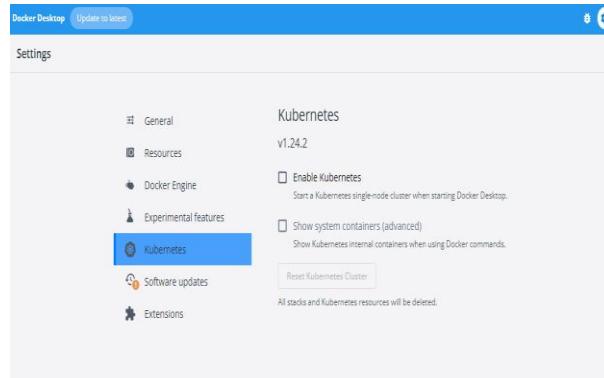
Docker manages individual containers, whereas Kubernetes manages multiple containers across clusters.

Integrating Kubernetes with Docker

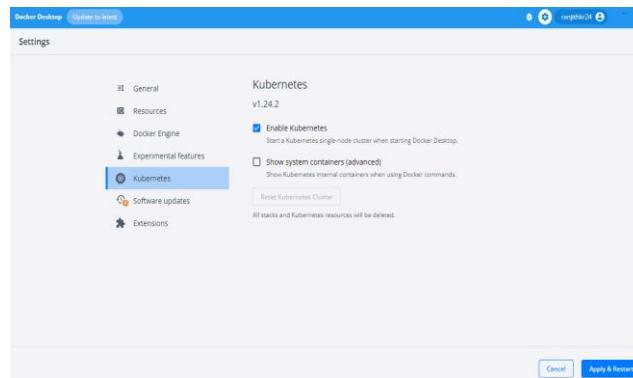
1. Start the Docker Desktop



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

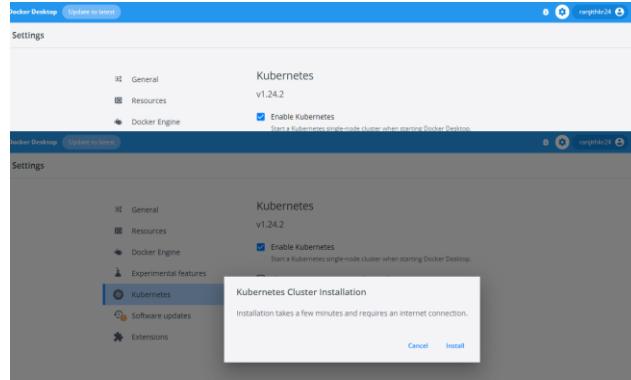


3. Select Kubernetes from the left side bar
4. Check the Enable Kubernetes checkbox



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:"f66044f4361b9f1f56f0053dd46cb7dce5e990a8", 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>
```

The screenshot shows a Docker interface with a list of running containers and system status information.

Containers List:

NAME	TAG	IMAGE ID	CREATED	SIZE
aj	IN USE latest	1e61d1ae333f	5 days ago	469.93 MB
docker/desktop-storage...	IN USE v2.0	99f89471f470	over 3 years ago	41.85 MB
docker/desktop-vpnkit-c...	IN USE 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	IN USE v1.8.6	a4ca41631cc7	about 3 years ago	46.83 MB
k8s.gcr.io/etcd	IN USE 3.5.3-0	aebe758cef4c	over 2 years ago	299.5 MB
k8s.gcr.io/kube-apiserver	IN USE v1.24.2	d3377ff67177	over 2 years ago	129.71 MB
k8s.gcr.io/kube-controll...	IN USE v1.24.2	34cdf99b1bb3	over 2 years ago	119.35 MB

System Status:

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

Week 9: Automate the Running of a 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

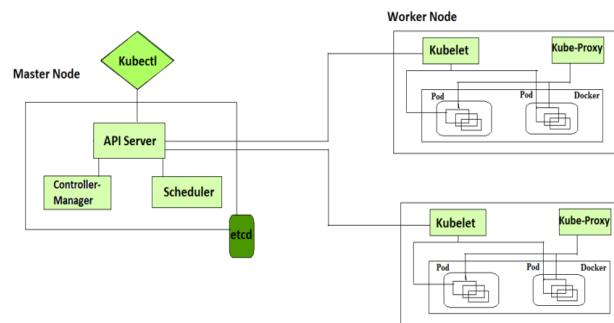
following prerequisites are required:

- **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

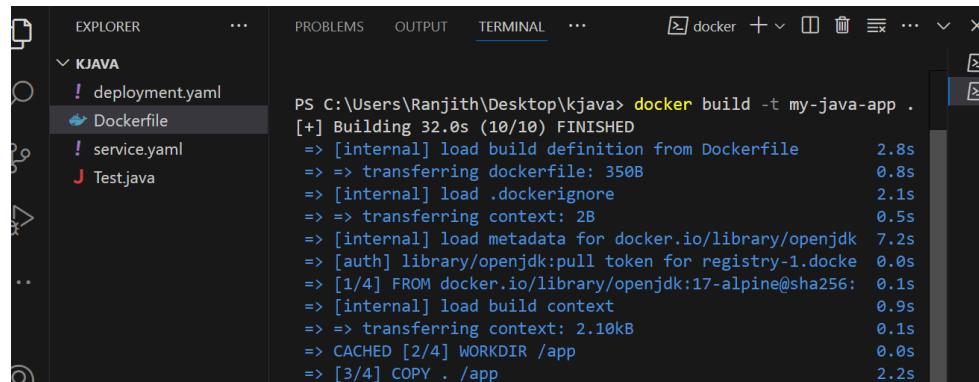
# Run the Java application

CMD ["java", "Test"]

```

Build the Docker image using the Dockerfile:

➤ **docker build -t my-java-app .**

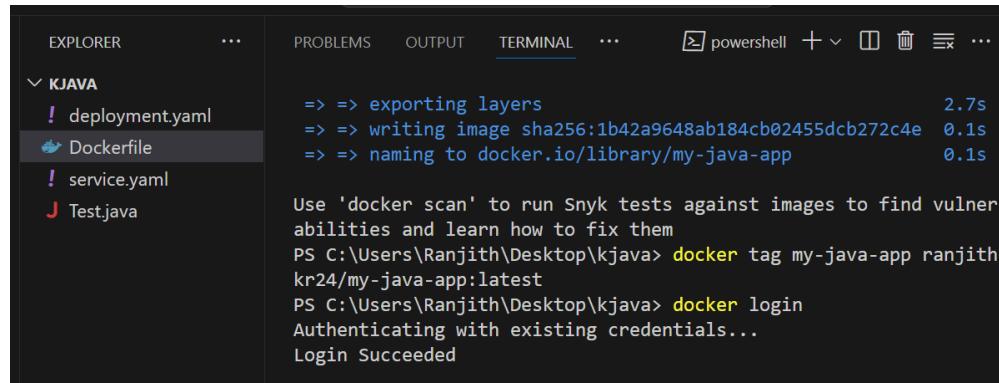


```

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



The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the following command-line session:

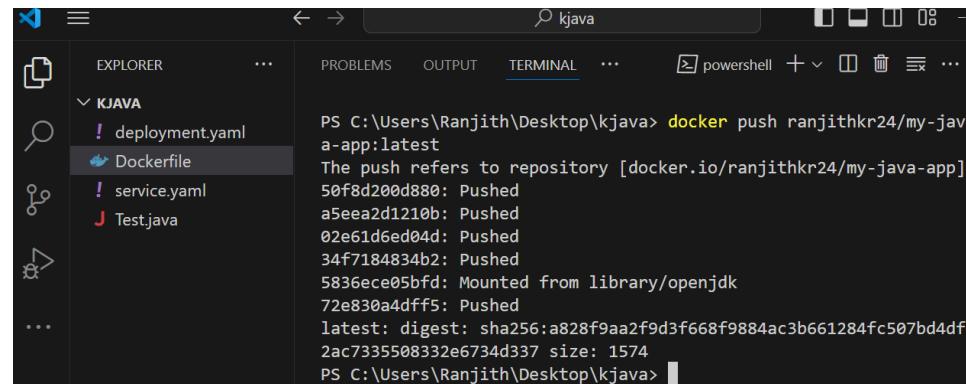
```
=> => exporting layers 2.7s
=> => writing image sha256:1b42a9648ab184cb02455dcb272c4e 0.1s
=> => naming to docker.io/library/my-java-app 0.1s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\Users\Ranjith\Desktop\kjava> docker tag my-java-app ranjithkr24/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
```



The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the following command-line session:

```
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

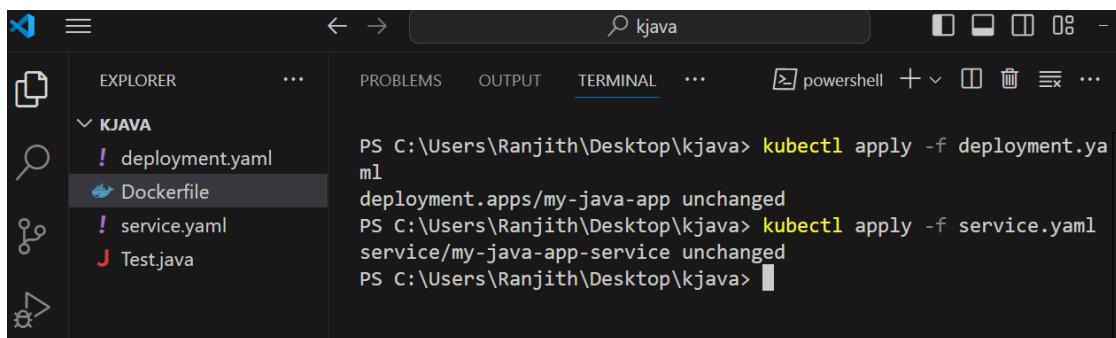
targetPort: 8080 # Port your Java app is running on

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

Dockerfile					
! service.yaml	(4m1s ago)	7d14h			
J Test.java					
	java-deployment-647885494-v2fv7	0/1	CrashLoopBackOff	239	
	(3m50s ago)	7d14h			
	my-java-app-5d79d95f68-bwhm4	1/1	Running	2 (4	
	2m ago)	46h			
	my-java-app-5d79d95f68-bz42k	1/1	Running	3	
	46h				
	my-java-app-5d79d95f68-mcjjr	1/1	Running	2 (4	
	2m ago)	46h			

kubectl get svc

KJAVA		PS C:\Users\Ranjith\Desktop\kjava> kubectl get svc		
		NAME	TYPE	CLUSTER-IP
!	deployment.yaml			
!	Dockerfile	PORT(S)	AGE	
!	service.yaml	java-app-service	NodePort	10.96.60.200
J	Test.java	80:30008/TCP	45h	<none>
		java-app1-service	NodePort	10.107.54.172
		80:30010/TCP	45h	<none>
		java-service	LoadBalancer	10.96.207.224
		80:30029/TCP	7d15h	localhost
		kubernetes	ClusterIP	10.96.0.1
		443/TCP	11d	<none>
		my-java-app-service	NodePort	10.106.150.206
		80:30007/TCP	46h	<none>
		myjava1-service	LoadBalancer	10.103.169.241
		80:31676/TCP	8d	localhost

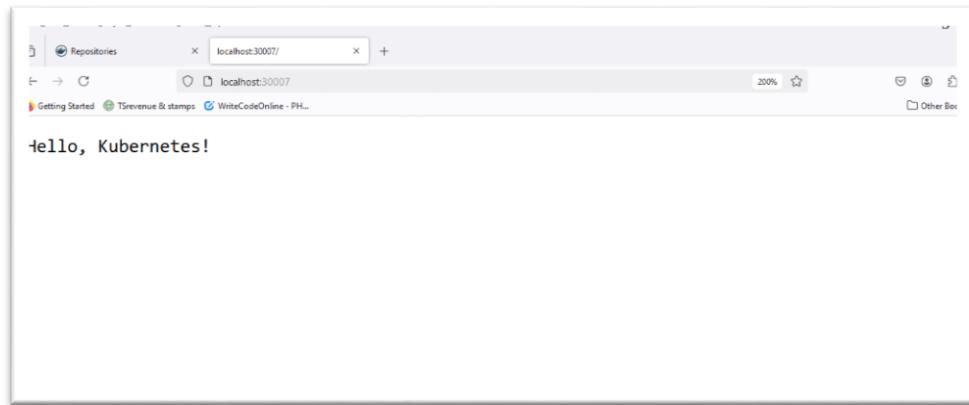
Step 6: Access the Application

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



10 Install and Explore Selenium for automated testin

What is Selenium?

Selenium is an open-source tool used for automating web browsers. It provides a suite of tools for testing web applications across different browsers and platforms. The Selenium WebDriver API allows you to interact with web pages programmatically by simulating user actions like clicking, typing, navigating, etc.

Installation of Selenium WebDriver

Step 1: Install Node.js and npm

```
D:\py54\Java_test\Java_test>npm -v  
10.2.5
```

If you don't have Node.js installed, download it from [Node.js official website](#). Follow the instructions for your operating system.

1. Verify the installation by running the following commands in the terminal:

```
node -v
```

```
npm -v
```

```
D:\py54\Java_test\Java_test>node -v  
v20.10.0
```

```
D:\>mkdir jtest
```

```
D:\>cd jtest
```

Step 2: Create a Project Folder

1. Open Visual Studio Code.
2. Create a new folder named SeleniumLab.

3. Open the terminal in VS Code and initialize a Node.js project:

```
npm init -y
```

```
D:\jtest>npm init -y
Wrote to D:\jtest\package.json:
```

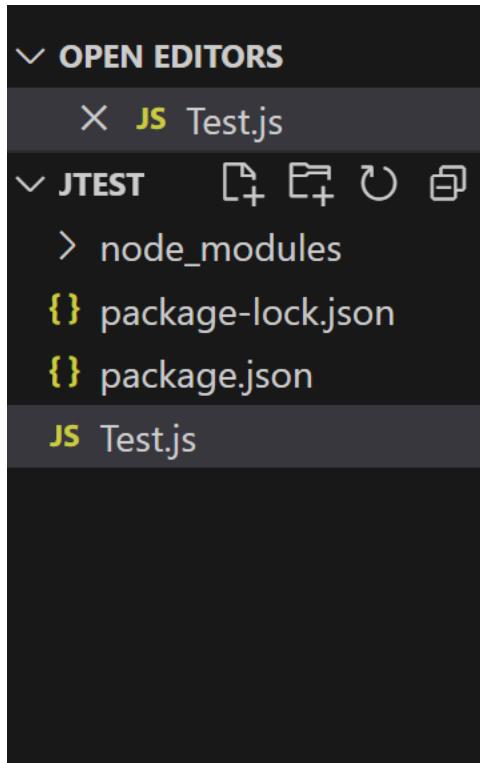
Step 3: Install Selenium WebDriver

1. Install the selenium-webdriver package:

```
npm install selenium-webdriver
```

```
D:\jtest>npm install selenium-webdriver
```

2. Download the ChromeDriver that matches your Chrome browser version from [ChromeDriver Downloads](#).
3. Extract the downloaded ChromeDriver and place it in the project folder.



Part 3: Writing a Simple Selenium Test Script

Step 1: Create a JavaScript Test File

1. Inside the SeleniumLab folder, create a new file named test.js.
2. Copy and paste the following code into test.js:

Test.js

```
D:\jtest>node Test.js
```

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

async function testGoogleSearch() {

    // Step 1: Launch the Chrome browser

    let driver = await new Builder().forBrowser('chrome').build();

    try {

        // Step 2: Open the Google homepage
```

```
await driver.get('https://www.google.com');

// Step 3: Locate the search box and enter a search query
await driver.findElement(By.name('q')).sendKeys('Selenium WebDriver', Key.RETURN);

// Step 4: Wait for the search results to load
await driver.wait(until.titleContains('Selenium WebDriver'), 5000);

// Step 5: Print the page title in the console
let title = await driver.getTitle();
console.log('Page Title:', title);

// Step 6: Check if the search results are displayed
if (title.includes('Selenium WebDriver')) {
    console.log('Test Passed: Search results loaded successfully.');
} else {
    console.log('Test Failed: Search results not loaded.');
}
} catch (error) {
    console.error('Error:', error);
} finally {
    // Step 7: Close the browser
    await driver.quit();
}

}
```

```
testGoogleSearch();
```

Explanation of the Code:

1. Import Selenium WebDriver:

- **Builder:** Used to create a new browser instance.
- **By:** Used to locate elements on the web page.
- **Key:** Simulates keyboard actions.
- **until:** Used for waiting until a condition is met.

2. testGoogleSearch Function:

- **Step 1:** Launches the Chrome browser.
 - **Step 2:** Opens <https://www.google.com>.
 - **Step 3:** Finds the search box using its name attribute and types a query (Selenium WebDriver).
 - **Step 4:** Waits until the page title includes "Selenium WebDriver".
 - **Step 5:** Prints the page title.
 - **Step 6:** Verifies if the search results are displayed correctly.
 - **Step 7:** Closes the browser.
-

Part 4: Running the Selenium Test Script

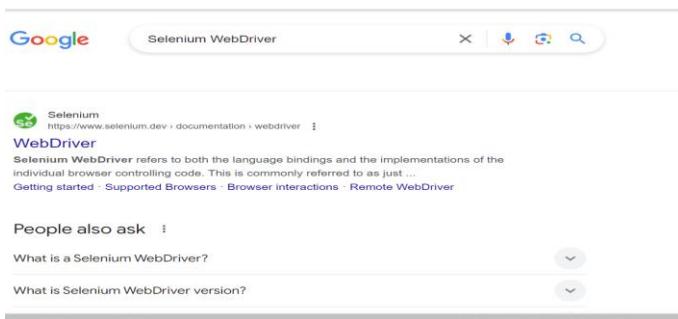
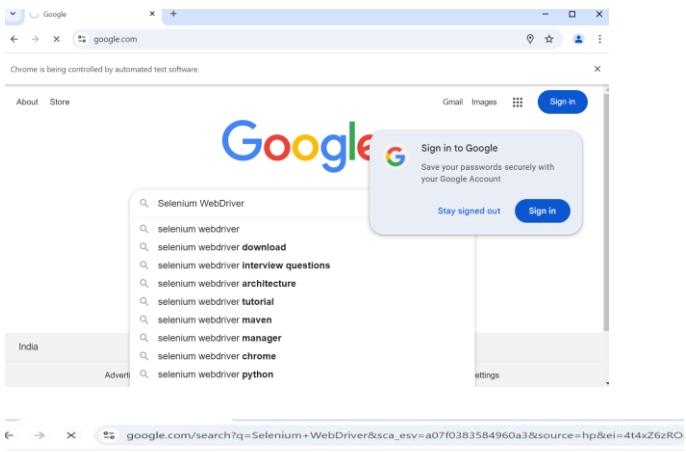
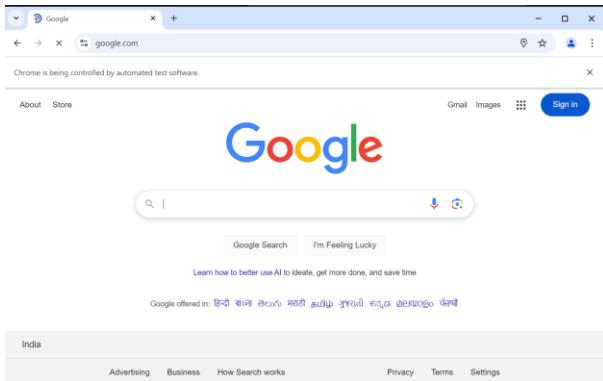
Step 1: Running the Script

1. Make sure that ChromeDriver is in the project folder.
2. Open the terminal in Visual Studio Code.
3. Run the test script:

```
node Test.js
```

Expected Output:

- The Chrome browser should open, navigate to Google, perform a search for "Selenium WebDriver", and display the search results.
- The terminal should display:



Page Title: Selenium WebDriver - Google Search

Test Passed: Search results loaded successfully.

```
D:\jtest>node Test.js

DevTools listening on ws://127.0.0.1:59744/devtools/b
a869-3094c1646ef6
Page Title: Selenium WebDriver - Google Search
Test Passed: Search results loaded successfully.
```

Part 5: Exploring Selenium Commands

Commonly Used Selenium Commands

1. Navigating to a URL:

```
await driver.get('https://example.com');
```

2. Locating Elements:

- By ID:

```
await driver.findElement(By.id('elementId'));
```

- By Class Name:

```
await driver.findElement(By.className('className'));
```

- By Name:

```
await driver.findElement(By.name('elementName'));
```

3. Performing Actions:

- Click an Element:

```
await driver.findElement(By.id('buttonId')).click();
```

- Enter Text:

```
await driver.findElement(By.name('inputName')).sendKeys('Hello World');
```

4. Waiting for Elements:

```
await driver.wait(until.elementLocated(By.id('elementId')), 5000);
```

5. Closing the Browser:

```
await driver.quit();
```

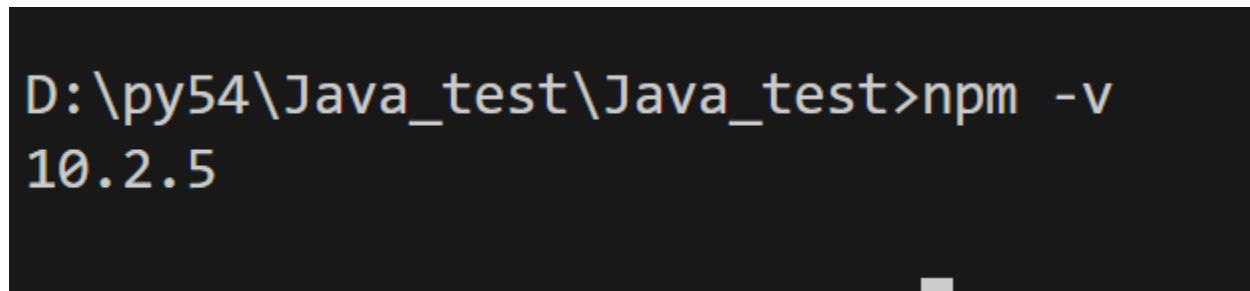
11 JavaScript Registration Form Testing using Selenium

Objective:

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

Firstly we have to download npm and nodejs

To check whether npm is downloaded or not use this command.



```
D:\py54\Java_test\Java_test>npm -v
10.2.5
```

To check whether nodejs is downloaded or not use this command.

```
nodejs -v
```

And then follow below steps

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:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Registration Form</title>
<!-- Inline CSS -->
```

```
<style>

body {
    font-family: Arial, sans-serif;
    text-align: center;
    margin-top: 50px;
    background-color: #f5f5f5;
}

form {
    display: inline-block;
    text-align: left;
    background-color: #ffffff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

input {
    width: 100%;
    padding: 10px;
    margin: 10px 0;
    box-sizing: border-box;
    border-radius: 4px;
    border: 1px solid #ccc;
}

button {
    background-color: #4CAF50;
    color: white;
    padding: 10px 20px;
    border: none;
    border-radius: 4px;
}
```

```
        cursor: pointer;
    }

    button:hover {
        background-color: #45a049;
    }

    #message {
        color: green;
        font-weight: bold;
        margin-top: 20px;
    }

    .error {
        color: red;
    }

```

</style>

</head>

<body>

<h2>Registration Form</h2>

<!-- Registration Form -->

<form id="registrationForm">

<label for="username">Username:</label>

<input type="text" id="username" name="username" placeholder="Enter username" required>

<label for="email">Email:</label>

<input type="email" id="email" name="email" placeholder="Enter email" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password" placeholder="Enter password" required>


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

<!-- Message Display -->
<p id="message"></p>

<!-- Inline JavaScript -->
<script>

    document.getElementById("registrationForm").addEventListener("submit",
function(event) {

event.preventDefault(); // Prevent form from submitting

    // Get input values
const username = document.getElementById("username").value;
const email = document.getElementById("email").value;
const password = document.getElementById("password").value;
const messageElement = document.getElementById("message");

    // Validate inputs
if (username === "" || email === "" || password === "") {
messageElement.textContent = "All fields are required!";
messageElement.classList.add("error");
return;
}

    // Check for valid email format
const emailPattern = /^[^@\s]+@[^\s@]+\.\[^@\s]+$/;
if (!emailPattern.test(email)) {
messageElement.textContent = "Invalid email format!";
messageElement.classList.add("error");
}

```

```
        return;
    }

    // If all validations pass, show success message
    messageElement.textContent = "Registration Successful!";
    messageElement.classList.remove("error");
    messageElement.style.color = "green";

    // Clear the form
    document.getElementById("registrationForm").reset();
}

</script>
</body>
</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

```
D:\>mkdir 11th
D:\>cd 11th
D:\11th>npm init -y
Wrote to D:\11th\package.json:

{
  "name": "11th",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

2. Install the Selenium WebDriver package:

```
npm install selenium-webdriver
```

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
```

```
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
```

```
awaitdriver.findElement(By.id('username')).sendKeys('student1');

// Enter email

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

// Enter password

awaitdriver.findElement(By.id('password')).sendKeys('password123');

// Click the Register button

awaitdriver.findElement(By.css('button[type="submit"]')).click();

// Wait for the success message to appear

awaitdriver.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
```

```
awaitdriver.quit();  
}  
}
```

```
testRegistrationForm();
```

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
```

```
D:\11th>node Test.js  
  
DevTools listening on ws://127.0.0.1:57433/devtools/browser/86a9542d-b92e-4d90-  
a488-57ec6277b090  
Test Result: Registration Successful!  
Test Passed
```

Expected Output:

- The console should display:

```
Test Result: Registration Successful!
```

```
Test Passed
```

```
D:\11th>node Test.js  
  
DevTools listening on ws://127.0.0.1:57433/devtools/browser/86a9542d-b92e-4d90-  
a488-57ec6277b090  
Test Result: Registration Successful!  
Test Passed
```

Registration Form

Username:

Email:

Password:

Registration Form

Username:

Email:

Password:

12 Node.js Registration Form, Docker Containerization, and Selenium Testing

Overview:

This project consists of a simple **Node.js registration form** hosted in a **Docker container**, with automated testing using **Selenium WebDriver**. The main steps involved are:

1. Setting up a Node.js registration form.
2. Dockerizing the application.
3. Writing Selenium tests to automate the form submission and validate the result.

Step 1: Create a Node.js Registration Form

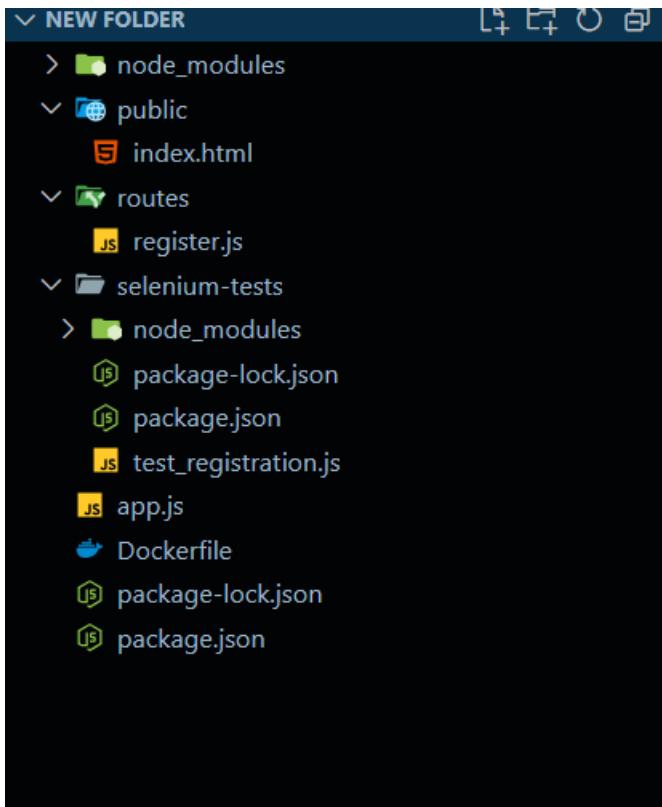
1.1 package.json

The package.json file defines the project metadata and dependencies. It includes necessary packages like express for the server and body-parser for handling form submissions.

express: A lightweight web framework for Node.js that simplifies routing and HTTP handling.

body-parser: Middleware to parse the incoming request bodies, enabling us to handle form submissions easily.

Project Structure



```
{  
  
  "name": "nodejs-registration",  
  
  "version": "1.0.0",  
  
  "description": "Simple Node.js registration form",  
  
  "main": "app.js",  
  
  "scripts": {  
  
    "start": "node app.js"  
  
  },  
  
  "dependencies": {  
  
    "express": "^4.18.2",  
  
    "body-parser": "^1.20.2"  
  
  }  

```

```
}
```

Create a file app.js

1.2 *app.js*

```
const express = require('express');

const bodyParser = require('body-parser');

const path = require('path');

const app = express();

// Middleware to handle form data

app.use(bodyParser.urlencoded({ extended: true }));

app.use(express.static('public'));

// Serve registration form

app.get('/', (req, res) => {

  res.sendFile(path.join(__dirname, 'public', 'index.html'));

});

// Handle form submission

app.post('/register', (req, res) => {

  const{ username, email, password } = req.body;

  console.log(`Username: ${username}, Email: ${email}`);

  res.send('Registration successful!');

});

// Start the server on port 3000

const PORT = 3000;
```

```
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

Express server setup: Sets up an Express server that listens on port 3000.

Static files: Serves static files (like index.html) from the public/ folder.

POST route: Handles registration form submissions and logs the data to the console.

1.3 public/index.html

This is the HTML form where users can input their registration details (username, email, and password). Upon submission, the data is sent to the server.

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Registration Form</title>

</head>

<body>

<h2>Registration Form</h2>

<form method="POST" action="/register">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required><br>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required><br>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required><br>

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

```
</form>  
</body>  
</html>
```

Step 2: Create a Dockerfile

A **Dockerfile** is used to containerize the Node.js application. It includes instructions to install dependencies, copy the application code, and run the app inside a container.

```
# Use Node.js base image  
FROM node:14  
  
# Set the working directory  
WORKDIR /usr/src/app  
  
# Copy package.json and install dependencies  
COPY package.json ./  
RUN npm install  
  
# Copy application files  
COPY . .  
  
# Expose port 3000  
EXPOSE 3000  
  
# Start the application  
CMD ["npm", "start"]
```

Base image: We use the official node:14 Docker image as a base.

Dependencies: It copies the package.json file and installs the dependencies.

Expose port: Port 3000 is exposed, which is the port the application runs on.

CMD: Runs npm start to launch the app in the container

Step 3: Build and Run Docker Container

To build and run the Docker container, execute the following commands:

```
# Build the Docker image  
  
docker build -t nodejs-registration-app .
```

```
PS C:\Users\shiva kumar\OneDrive\Desktop\New folder> docker build -t nodejs-registration-app .
[+] Building 3.6s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> transferring dockerfile: 341B
=> [internal] load metadata for docker.io/library/node:14
=> [auth] library/node:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> > transferring context: 28
=> [internal] load build context
=> > transferring context: 209.40kB
=> [1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
=> CACHED [2/5] WORKDIR /usr/src/app
=> CACHED [3/5] COPY package.json .
=> CACHED [4/5] RUN npm install
=> CACHED [5/5] COPY . .
=> exporting to image
=> > exporting layers
=> > writing image sha256:8c9e1a9f520369ce54eb60ed83ceb3ab077dc1c28349781ca88484acf4cd6aa
=> > naming to docker.io/library/nodejs-registration-app

What's Next?
```

Run the container

```
docker run -d -p 3000:3000 --name nodejs-registration nodejs-registration-app
```

```
What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\shiva kumar\OneDrive\Desktop\New folder> docker run -d -p 3000:3000 --name my-nodejs-app nodejs-registration-app
66c50e1726fe3d7a878564677c3aad33fc6050aa2667e4c20d83a6c35dde6a27
PS C:\Users\shiva kumar\OneDrive\Desktop\New folder> █
```

After running the container, the app is accessible at <http://localhost:3000>.

Step 4: Set Up Selenium for Automated Testing

To automate testing for the registration form, we use **Selenium WebDriver** to simulate a user filling out the form and submitting it.

4.1 Install Selenium WebDriver

Inside the `selenium-tests` directory, initialize a new Node.js project and install `selenium-webdriver` and `chromedriver`:

By following command

```
Npm init -y
```

```
npm install selenium-webdriver chromedriver
```

4.2 Create test_registration.js

This file contains the test logic for Selenium, which fills out the registration form and checks if the registration is successful.

Copy the command

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

const chrome = require('selenium-webdriver/chrome');

async function runTest() {

  let driver = await new Builder().forBrowser('chrome').build();

  try {

    // Open the registration form

    await driver.get('http://localhost:3000');

    // Fill in the registration form

    await driver.findElement(By.id('username')).sendKeys('testuser');

    await driver.findElement(By.id('email')).sendKeys('testuser@example.com');

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

    // Submit the form

    await driver.findElement(By.tagName('button')).click();

    // Wait for the response

    await driver.wait(until.elementLocated(By.tagName('body')), 5000);

    // Verify the result

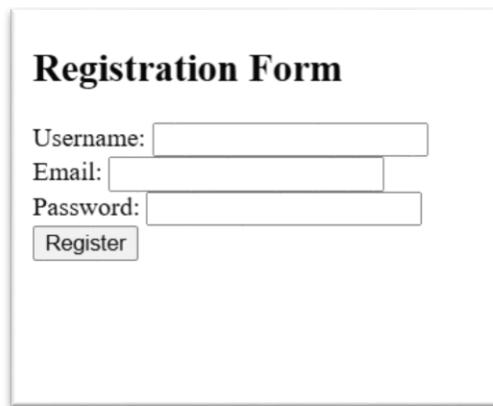
    let bodyText = await driver.findElement(By.tagName('body')).getText();
  }
}
```

```
if (bodyText.includes('Registration successful!')) {  
    console.log('Test Passed: Registration was successful.');//  
} else {  
    console.log('Test Failed: Registration was not successful.');//  
}  
}  
}  
  
} finally {  
    await driver.quit();  
}  
}  
  
runTest();
```

Step 5: Run Selenium Test Cases

To run the automated tests, execute the following command:

First run this app.js



A screenshot of a registration form titled "Registration Form". The form contains three input fields: "Username" (text input), "Email" (text input), and "Password" (text input). Below the password field is a "Register" button.

```
✓ TERMINAL

PS C:\Users\shiva kumar\OneDrive\Desktop\New folder> node app.js
Server is running on http://localhost:3000
Username: admin, Email: shivaks@gmail.com
Username: testuser, Email: testuser@example.com
PS C:\Users\shiva kumar\OneDrive\Desktop\New folder> node app.js
Server is running on http://localhost:3000
Username: testuser, Email: testuser@example.com
|
```

And then node test_registration.js(make sure change the directory to selenium)

```
DevTools listening on ws://127.0.0.1:54346/devtools/browser/e73d1819-604a-4954-a7ba-71755caa56d9
Test Passed: Registration was successful.
PS C:\Users\shiva kumar\OneDrive\Desktop\New folder\selenium-tests> node .\test_registration.js

DevTools listening on ws://127.0.0.1:54346/devtools/browser/e73d1819-604a-4954-a7ba-71755caa56d9
Test Passed: Registration was successful.
PS C:\Users\shiva kumar\OneDrive\Desktop\New folder\selenium-tests> |
```

If the registration is successful, you will see the output:

Test Passed: Registration was successful