DEVOPS LAB

B.Tech. III Year I Sem.

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 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 for 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.

1. WRITE CODE FOR A SIMPLE USER REGISTRATION FORM FOR AN EVENT.

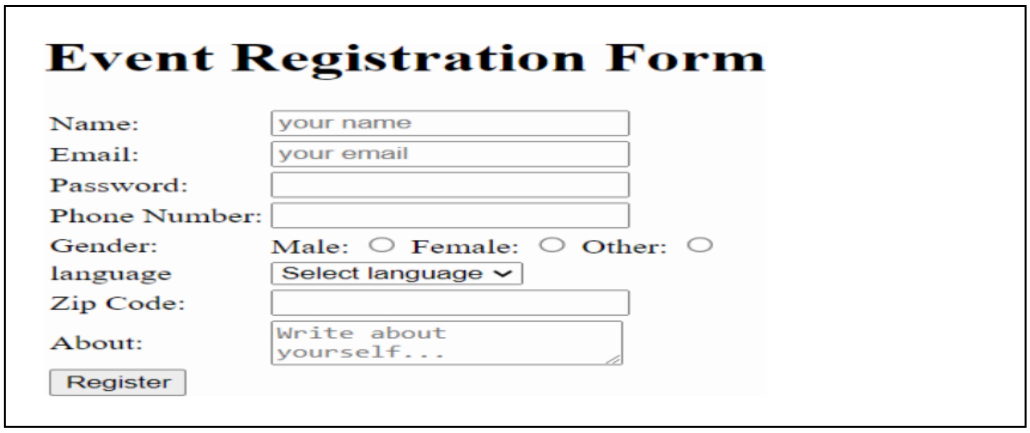
Registration.html

Event Registration Form

Here's a simple example of a registration HTML file:

```
<!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>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f0f0f0;
    }

    .container {
      max-width: 400px;
      margin: 40px auto;
      padding: 20px;
      background-color: #fff;
      border: 1px solid #ddd;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }

    .form-group {
      margin-bottom: 20px;
    }

    label {
      display: block;
      margin-bottom: 10px;
    }

    input[type="text"], input[type="email"], input[type="password"] {
      width: 100%;
      height: 40px;
      padding: 10px;
      border: 1px solid #ccc;
      border-radius: 5px;
    }

    button[type="submit"] {
      width: 100%;
      height: 40px;
      background-color: #4CAF50;
      color: #fff;
      padding: 10px;
```

```
            border: none;
            border-radius: 5px;
            cursor: pointer;
        }

        button[type="submit"]:hover {
            background-color: #3e8e41;
        }
    </style>
</head>
<body>
    <div class="container">
        <h2>Registration Form</h2>
        <form>
            <div class="form-group">
                <label for="firstname">First Name:</label>
                <input type="text" id="firstname" name="firstname" required>
            </div>

            <div class="form-group">
                <label for="lastname">Last Name:</label>
                <input type="text" id="lastname" name="lastname" required>
            </div>

            <div class="form-group">
                <label for="email">Email:</label>
                <input type="email" id="email" name="email" required>
            </div>

            <div class="form-group">
                <label for="password">Password:</label>
                <input type="password" id="password" name="password" required>
            </div>

            <div class="form-group">
                <label for="confirmpassword">Confirm Password:</label>
                <input type="password" id="confirmpassword" name="confirmpassword" required>
            </div>

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



Event Registration Form

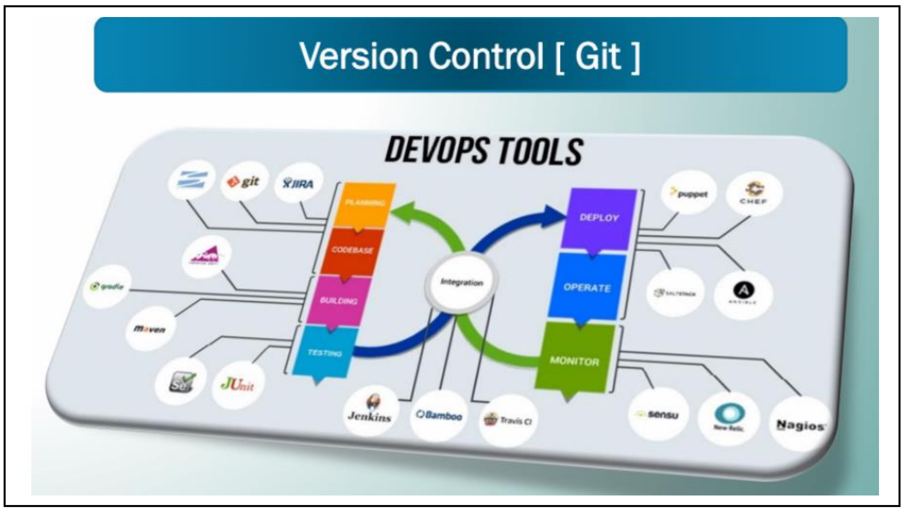
Name:	your name
Email:	your email
Password:	
Phone Number:	
Gender:	Male: ○ Female: ○ Other: ○
language	Select language
Zip Code:	
About:	Write about yourself...

Register

2. EXPLORE GIT AND GITHUB COMMANDS.

Objective:
The objective of this experiment is to familiarise participants with essential Git concepts and commands, enabling them to effectively use Git for version control and collaboration.

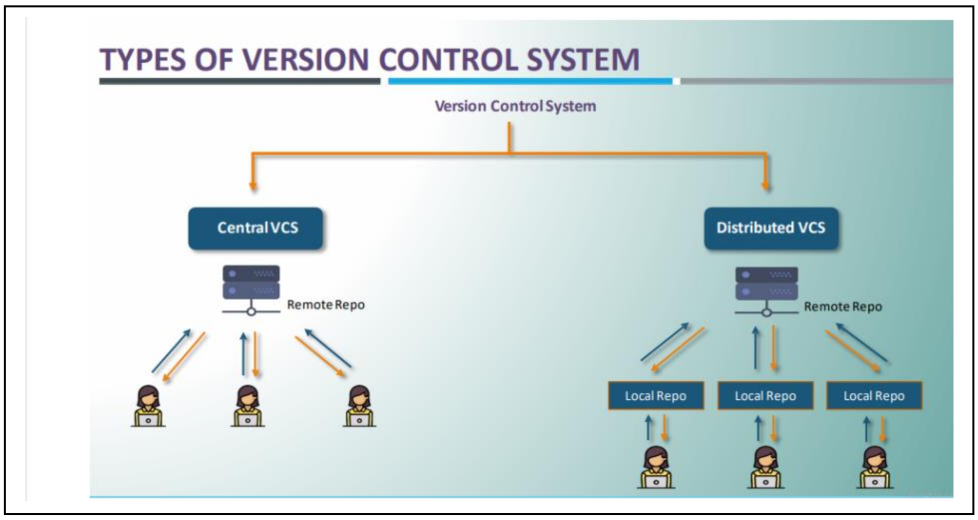
Git is a distributed version control system (VCS) that helps developers track changes in their codebase, collaborate with others, and manage different versions of their projects efficiently. It was created by Linus Torvalds in 2005 to address the shortcomings of existing version control systems.



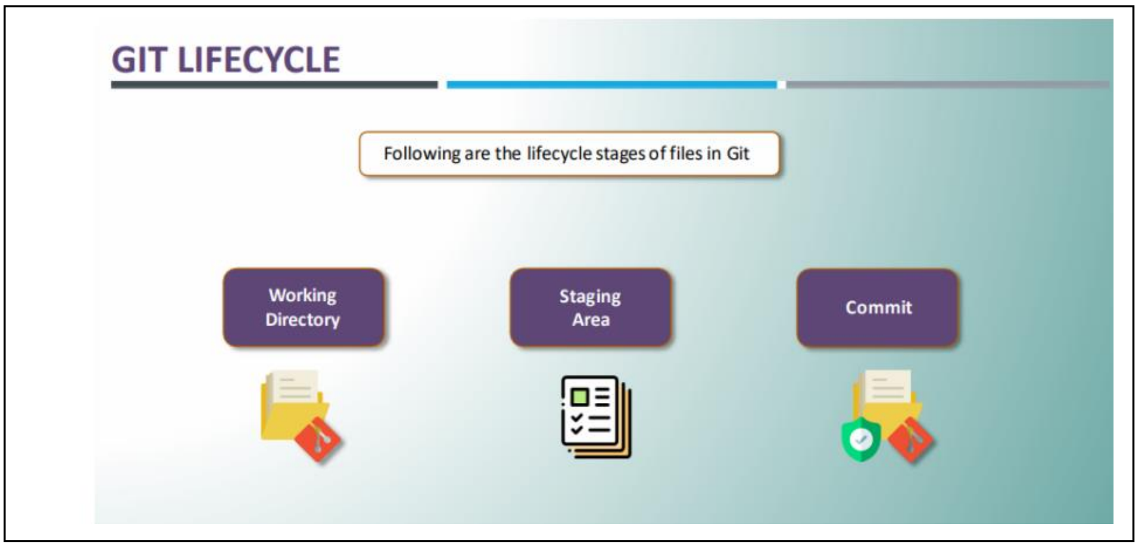
Unlike traditional centralised VCS, where all changes are stored on a central server, Git follows a distributed model. Each developer has a complete copy of the repository on their local machine, including the entire history of the project. This decentralisation offers numerous advantages, such as offline work, faster operations, and enhanced collaboration.

Git is a widely used version control system that allows developers to collaborate on projects, track changes, and manage codebase history efficiently. This experiment aims to provide a hands-on introduction to Git and explore various fundamental Git commands. Participants will learn how to set up a Git repository, commit changes, manage branches, and collaborate with others using Git.

Advantages of Version Control:
- ✓ Versioning is Automatic.
- ✓ Team Collaboration is simple.
- ✓ Easy Access to previous Versions.
- ✓ Only modified code is stored across different versions, hence saves storage.



Git is the most popular tool among all the DVCS tools. It is a version-control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source-code management in software development, but it can be used to keep track of changes in any set of files.



Working Directory:
The place where your project resides in your local disk. This project may or may not be tracked by git In either case, the directory is called the working directory

The project can be tracked by git, by using the command git init. By doing git init, it automatically creates a hidden .git folder

Staging Area:
Once we are in the working directory, we have to specify which files are to be tracked by git. We do not specify all files to be tracked in git, because some files could be temporary data which is being generated while execution. To add files in the staging area, we use the command git add.

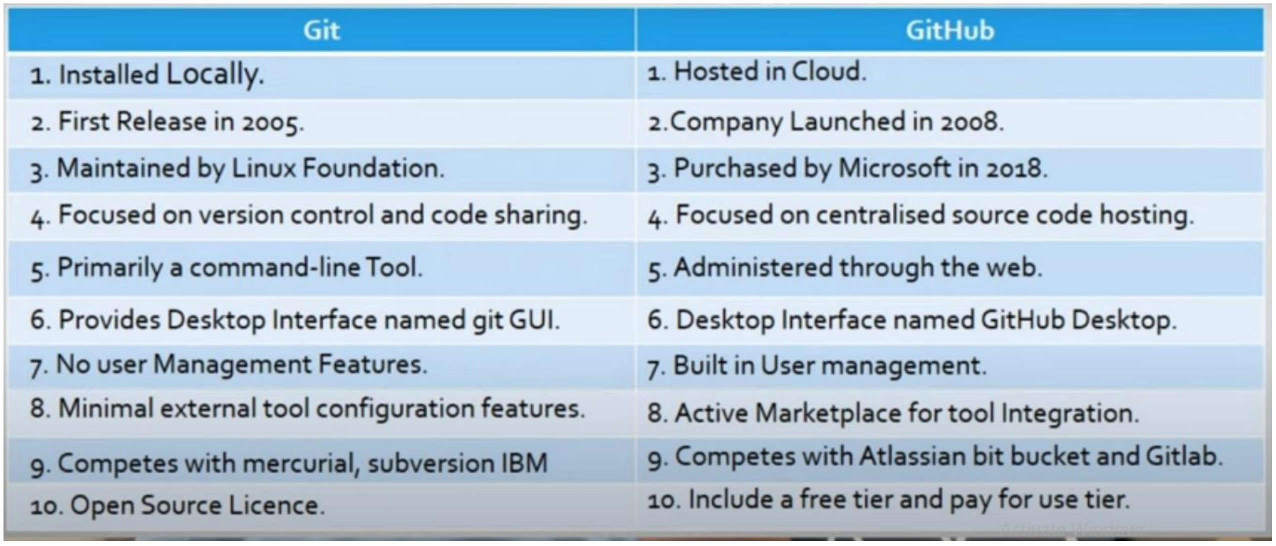
Commit:
Once the files are selected and are ready in the staging area, they can now be saved in repository. Saving a file in the repository of git is known as doing a commit When we commit a repository in git, the commit is identified by a commit id.

The command for initializing this process is git commit –m "message"

Key Concepts:

- Repository: A Git repository is a collection of files, folders, and their historical versions. It contains all the information about the project's history, branches, and commits.
- Commit: A commit is a snapshot of the changes made to the files in the repository at a specific point in time. It includes a unique identifier (SHA-1 hash), a message describing the changes, and a reference to its parent commit(s).
- Branch: A branch is a separate line of development within a repository. It allows developers to work on new features or bug fixes without affecting the main codebase. Branches can be merged back into the main branch when the changes are ready.
- Merge: Merging is the process of combining changes from one branch into another. It integrates the changes made in a feature branch into the main branch or any other target branch.

- Pull Request: In Git hosting platforms like GitHub, a pull request is a feature that allows developers to propose changes from one branch to another. It provides a platform for code review and collaboration before merging
- Remote Repository: A remote repository is a copy of the Git repository stored on a server, enabling collaboration among multiple developers. It can be hosted on platforms like GitHub, GitLab, or Bitbucket



Git	GitHub
1. Installed Locally.	1. Hosted in Cloud.
2. First Release in 2005.	2.Company Launched in 2008.
3. Maintained by Linux Foundation.	3. Purchased by Microsoft in 2018.
4. Focused on version control and code sharing.	4. Focused on centralised source code hosting.
5. Primarily a command-line Tool.	5. Administered through the web.
6. Provides Desktop Interface named git GUI.	6. Desktop Interface named GitHub Desktop.
7. No user Management Features.	7. Built in User management.
8. Minimal external tool configuration features.	8. Active Marketplace for tool Integration.
9. Competes with mercurial, subversion IBM	9. Competes with Atlassian bit bucket and Gitlab.
10. Open Source Licence.	10. Include a free tier and pay for use tier.

Basic Git Commands:

- git init: Initialises a new Git repository in the current directory.
- git clone: Creates a copy of a remote repository on your local machine.
- git add: Stages changes for commit, preparing them to be included in the next commit.
- git commit: Creates a new commit with the staged changes and a descriptive message.
- git status: Shows the current status of the working directory, including tracked and untracked files.
- git log: Displays a chronological list of commits in the repository, showing their commit messages, authors, and timestamps.
- git branch: Lists, creates, or deletes branches within the repository.
- git checkout: Switches between branches, commits, or tags. It's used to navigate through the repository's history.
- git merge: Combines changes from different branches, integrating them into the current branch.
- git pull: Fetches changes from a remote repository and merges them into the current branch.
- git push: Sends local commits to a remote repository, updating it with the latest changes.

Materials:

- Computer with Git installed (https://git-scm.com/downloads)
- Command-line interface (Terminal, Command Prompt, or Git Bash)

3. Practice Source code management on GitHub. Experiment with the source code in exercise 1.

Step 1: Download and install required git version.

Step 2: check the know git version using command :

```
$git  --version
```

Step 3: set up your information in bash type below commands

```
$ git config  --global user name "Your Name"
 $ git config  --global user. email Your email@example.com"
```

To push a repository

Open text editor

Create a file named Registration.html

Registration.html

Event Registration Form

Here's a simple example of a registration HTML file:

```
<!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>
   <style>
      body {
         font-family: Arial, sans-serif;
         background-color: #f0f0f0;
      }

      .container {
         max-width: 400px;
         margin: 40px auto;
         padding: 20px;
         background-color: #fff;
         border: 1px solid #ddd;
         border-radius: 10px;
         box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      }

      .form-group {
         margin-bottom: 20px;
      }
```

```
        label {
            display: block;
            margin-bottom: 10px;
        }

        input[type="text"], input[type="email"], input[type="password"] {
            width: 100%;
            height: 40px;
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 5px;
        }

        button[type="submit"] {
            width: 100%;
            height: 40px;
            background-color: #4CAF50;
            color: #fff;
            padding: 10px;
            border: none;
            border-radius: 5px;
            cursor: pointer;
        }
            button[type="submit"]:hover {
            background-color: #3e8e41;
        }
    </style>
</head>
<body>
    <div class="container">
        <h2>Registration Form</h2>
        <form>
            <div class="form-group">
                <label for="firstname">First Name:</label>
                <input type="text" id="firstname" name="firstname" required>
            </div>

            <div class="form-group">
                <label for="lastname">Last Name:</label>
                <input type="text" id="lastname" name="lastname" required>
            </div>

            <div class="form-group">
                <label for="email">Email:</label>
                <input type="email" id="email" name="email" required>
            </div>

            <div class="form-group">
                <label for="password">Password:</label>
                <input type="password" id="password" name="password" required>
            </div>
            <div class="form-group">
                <label for="confirmpassword">Confirm Password:</label>
                <input type="password" id="confirmpassword" name="confirmpassword" required>
            </div>

            <button type="submit">Register</button>
        </form>
    </div>
</body>
</html>
```

Step 4: Save the file

Go to file location

Right click on the file

Click on open with

Select git bash

Now use the following commands to push file into git repository

Initialize git

```
$ git init
```

Step 5: Add files to your Repo

```
$ git add Registration.html
```

Step 6: commit your changes

bash:

```
$ git commit -m "Initial commit"
```

Step 7: Create a Remote Repository in github

Link it to local Repo

bash:

```
$ git remote add origin https://github.com/youruserName/yourRepository.git

$ git  branch -m master main

$ git push-u origin master main

$ git status

$ git remote -v
```

4. Jenkins installation and setup, explore the environment.

Installing Jenkins

Jenkins is typically run as a standalone application in its own process. The Jenkins WAR file bundles Winstone, a Jetty servlet container wrapper, and can be started on any operating system or platform with a version of Java supported by Jenkins.

Windows

The simplest way to install Jenkins on Windows is to use the Jenkins Windows installer. That program will install Jenkins as a service using a 64 bit JVM chosen by the user. Keep in mind that to run Jenkins as a service, the account that runs Jenkins must have permission to login as a service.

Prerequisites

Minimum hardware requirements:

- 256 MB of RAM
- 1 GB of drive space (although 10 GB is a recommended minimum if running Jenkins as a Docker container)

Recommended hardware configuration for a small team:

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

Software requirements:

- Java:
- Web browser:
- Windows operating system
-

Java Support Policy

Running Jenkins system
The following Java versions are required to run Jenkins:

Supported Java versions	Long term support (LTS) release	Weekly release
Java 17 or Java 21	2.479.1 (October 2024)	2.463 (June 2024)

Download and deploy

The Jenkins project produces two release lines: Stable (LTS) and weekly. Depending on your organization's needs, one may be preferred over the other.

Stable (LTS)

Long-Term Support (LTS) release baselines are chosen every 12 weeks from the stream of regular releases.

Downloading Jenkins

Jenkins is distributed as WAR files, native packages, installers, and Docker images.

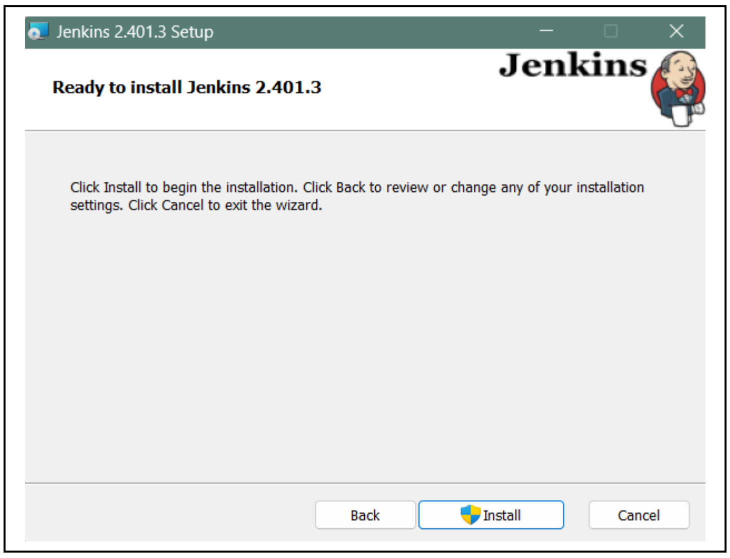
Download Jenkins 2.479.1 LTS for:

Generic Java package (.war)SHA-256:
dbf987b3aaab16ce20e9413b3082fa323e3724cbb64562ddb64c1e4d4f58b470

After the download completes, open the Windows installer and follow the steps below to install Jenkins.

Step 1: Setup wizard

On opening the Windows Installer, an **Installation Setup Wizard** appears, Click **Next** on the Setup Wizard to start your installation.


Jenkins 2.401.3 Setup

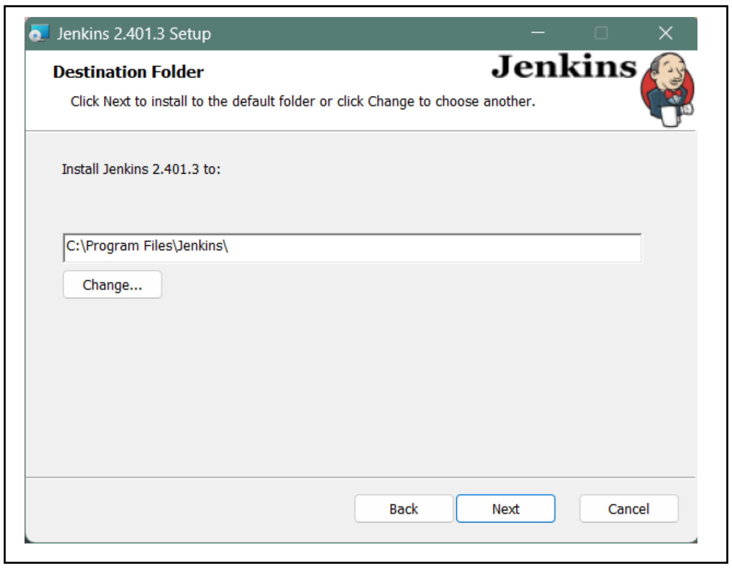
Jenkins

Ready to install Jenkins 2.401.3

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Step 2: Select destination folder

Select the destination folder to store your Jenkins Installation and click **Next** to continue.


Jenkins 2.401.3 Setup

Jenkins

Destination Folder

Click Next to install to the default folder or click Change to choose another.

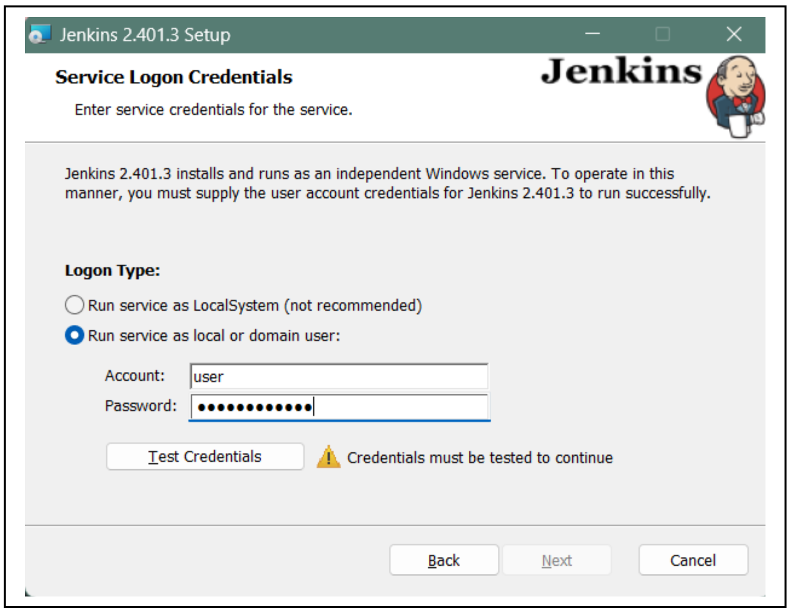
Install Jenkins 2.401.3 to:

C:\Program Files\Jenkins\

Step 3: Service logon credentials

When Installing Jenkins, it is recommended to install and run Jenkins as an independent windows service using a **local or domain user** as it is much safer than running Jenkins using **Local System(Windows equivalent of root)** which will grant Jenkins full access to your machine and services.

To run Jenkins service using a **local or domain user**, specify the domain user name and password with which you want to run Jenkins, click on **Test Credentials** to test your domain credentials and click on **Next**.


Jenkins 2.401.3 Setup

Service Logon Credentials

Enter service credentials for the service.

Jenkins 2.401.3 installs and runs as an independent Windows service. To operate in this manner, you must supply the user account credentials for Jenkins 2.401.3 to run successfully.

Logon Type:

Run service as LocalSystem (not recommended) [✓ Run service as local or domain user:]

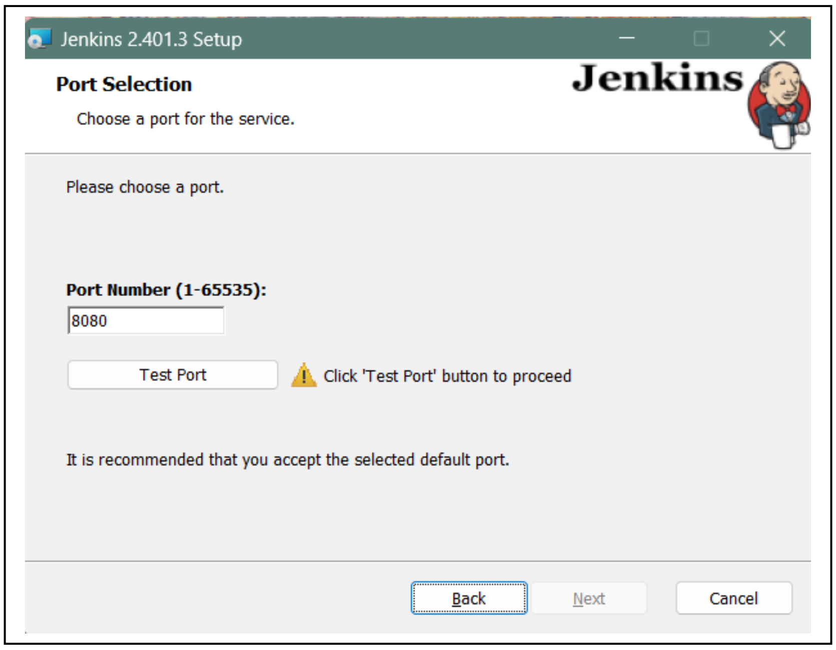
Account: user

Password: ●●●●●●●●●●●●

Test Credentials — Credentials must be tested to continue

Step 4: Port selection

Specify the port on which Jenkins will be running, **Test Port** button to validate whether the specified port if free on your machine or not. Consequently, if the port is free, it will show a green tick mark as shown below, then click on **Next**.


Jenkins 2.401.3 Setup

Port Selection

Choose a port for the service.

Please choose a port.

Port Number (1-65535): 8080

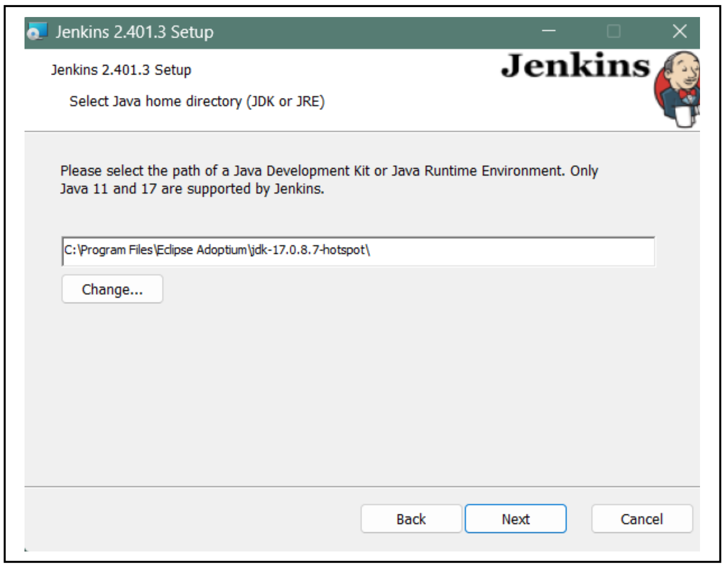
Test Port — Click 'Test Port' button to proceed

It is recommended that you accept the selected default port.

Step 5: Select Java home directory

The installation process checks for Java on your machine and prefills the dialog with the Java home directory. If the needed Java version is not installed on your machine, you will be prompted to install it.

Once your Java home directory has been selected, click on **Next** to continue.



Jenkins 2.401.3 Setup

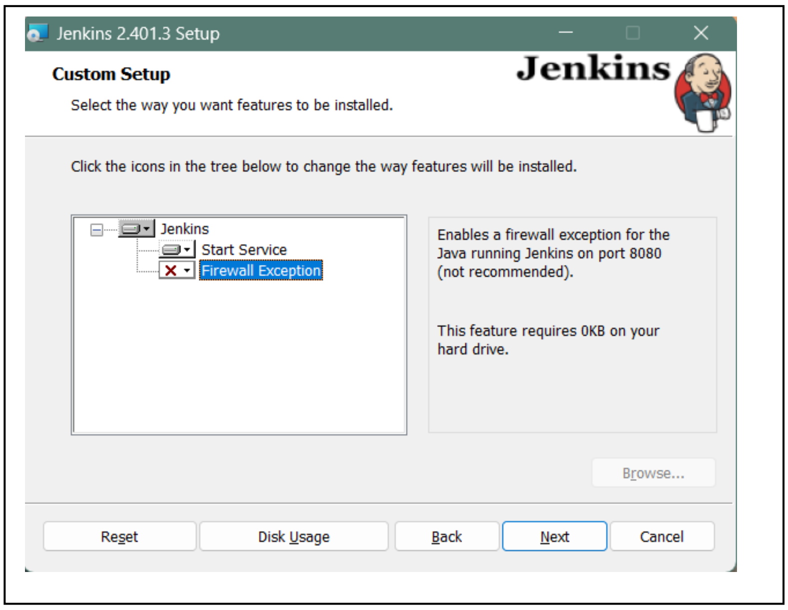
Select Java home directory (JDK or JRE)

Please select the path of a Java Development Kit or Java Runtime Environment. Only Java 11 and 17 are supported by Jenkins.

C:\Program Files\Eclipse Adoptium\jdk-17.0.8.7-hotspot\

Step 6: Custom setup

Select other services that need to be installed with Jenkins and click on **Next**.



Custom Setup

Select the way you want features to be installed.

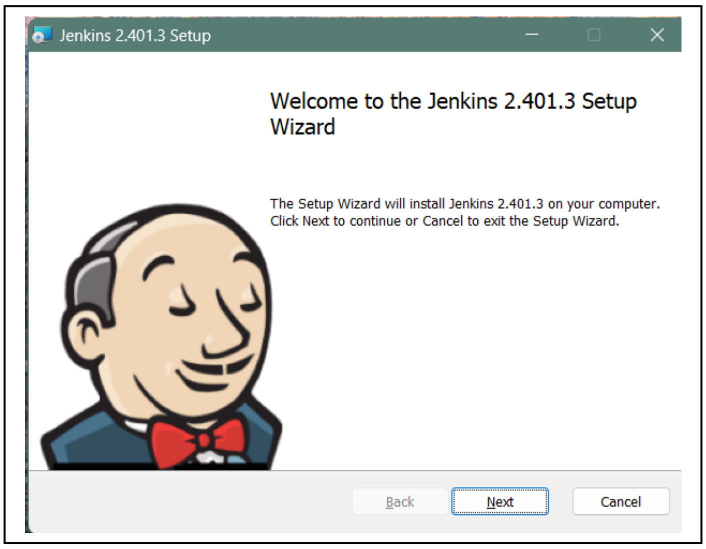
Click the icons in the tree below to change the way features will be installed.

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

This feature requires 0KB on your hard drive.

Step 7: Install Jenkins

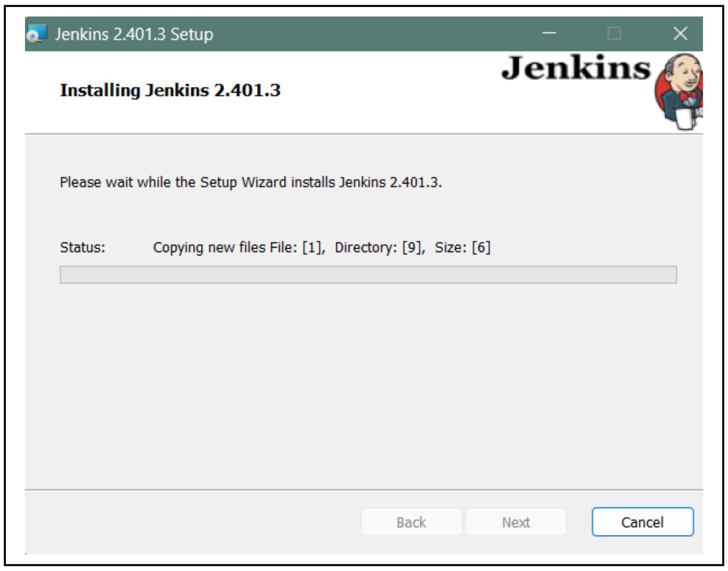
Click on the **Install** button to start the installation of Jenkins.


Jenkins 2.401.3 Setup

Welcome to the Jenkins 2.401.3 Setup Wizard

The Setup Wizard will install Jenkins 2.401.3 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Additionally, clicking on the **Install** button will show the progress bar of installation, as shown below:


Jenkins 2.401.3 Setup

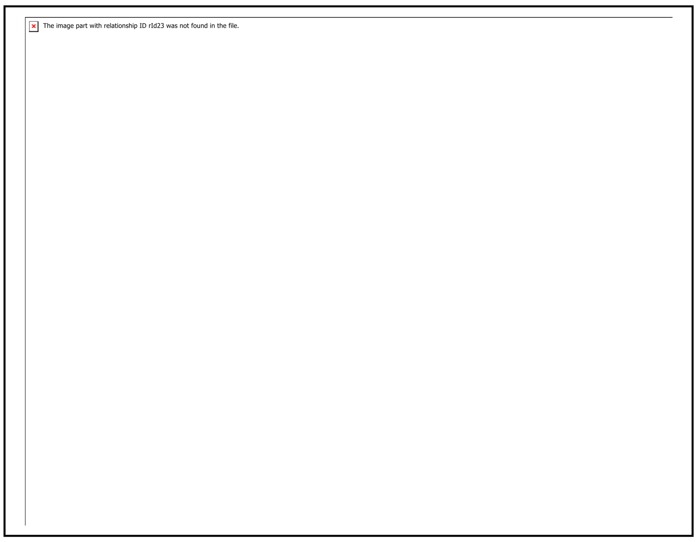
Installing Jenkins 2.401.3

Please wait while the Setup Wizard installs Jenkins 2.401.3.

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

Step 8: Finish Jenkins installation

Once the installation completes, click on **Finish** to complete the installation.


The image part with relationship ID rId23 was not found in the file.

Jenkins will be installed as a **Windows Service**. You can validate this by browsing the **services** section, as shown below:

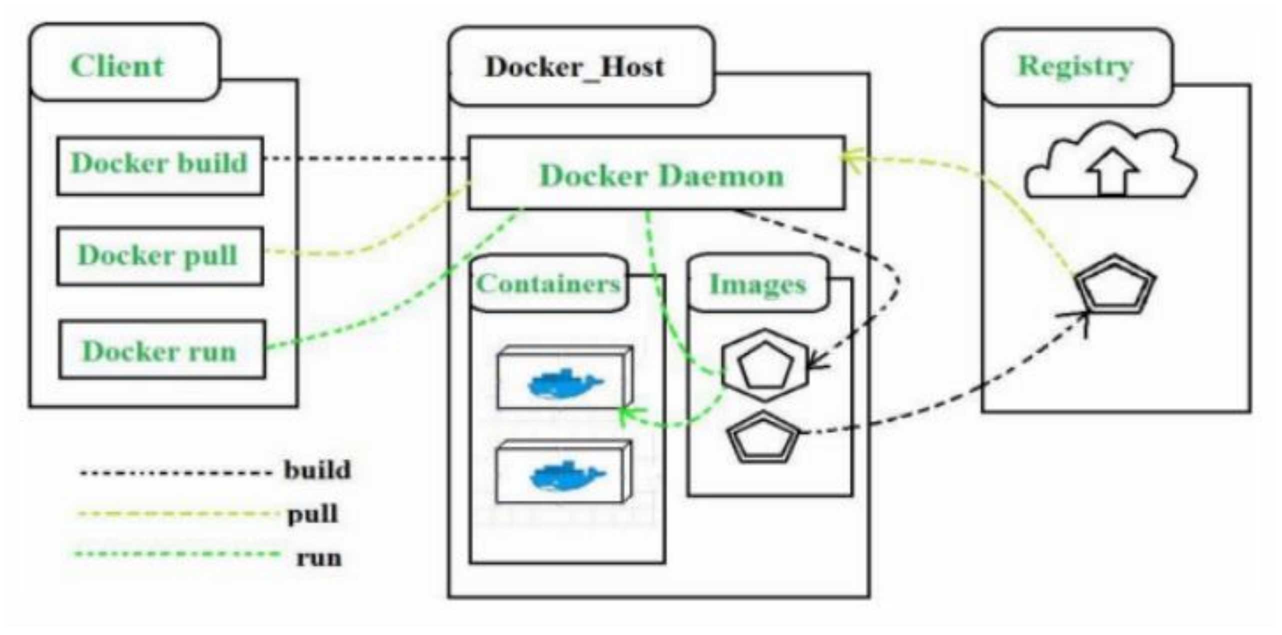
The image part with relationship ID rId23 was not found in the file.

EXPERIMENT NO.: 6. Explore Docker commands for content management.

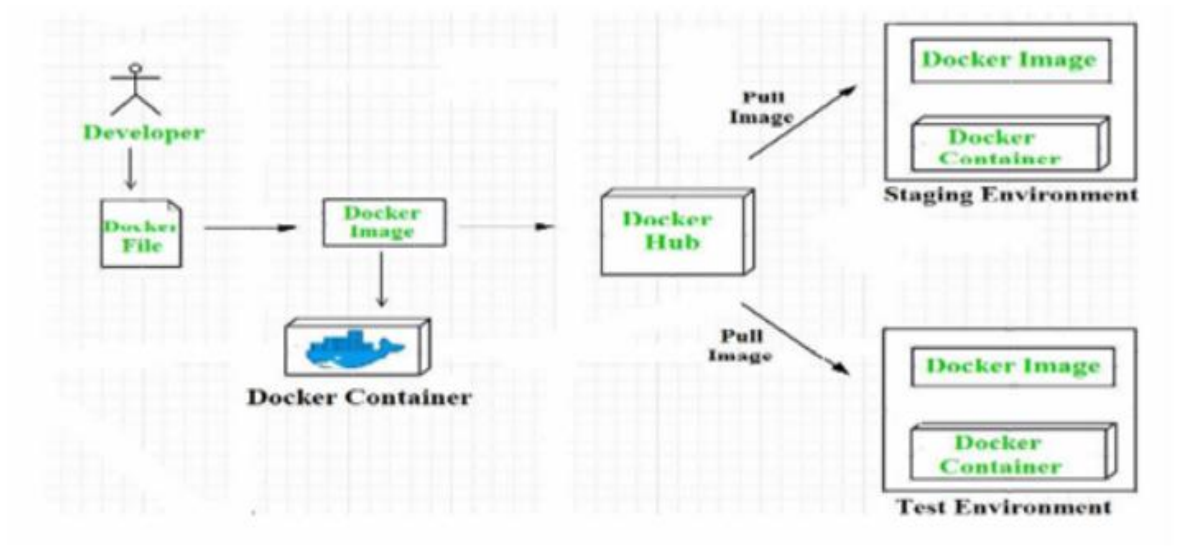
AIM: Explore Docker commands for content management.

DESCRIPTION: Docker is a containerization technology that is widely used for managing application containers.

Docker Architecture:
Docker architecture consists of Docker client, Docker Daemon running on Docker Host and DockerHub repository.



Components of Docker
The main components of Docker include–Docker clients and servers, Docker images, Docker file, Docker Registries, and Docker containers.



Here are some commonly used Docker commands for content management:

Docker run:
Run a command in a new container.

For example: $ sudo docker run --name mycontainer -it ubuntu:16.04 /bin/bash

This command runs a new container based on the Ubuntu 16.04 image and starts a shell session in the container.

Docker start:
Start one or more stopped containers.

For example: $ sudo docker start mycontainer

This command starts the container named "mycontainer".

Docker stop:
Stop one or more running containers.

For example: $ sudo docker stop mycontainer

This command stops the container named "mycontainer".

Docker rm:
Remove one or more containers.

For example: $ sudo docker rm mycontainer

This command removes the container named "mycontainer".

Docker ps:
List containers.

For example: $ docker ps

This command lists all running containers.

Docker images:
List images.

For example: $ docker images

This command lists all images stored locally on the host.

Docker pull:
Pull an image or a repository from a registry.

For example: $ docker pull ubuntu:16.04

This command pulls the Ubuntu 16.04 image from the Docker Hub registry.

Docker push:
Push an image or a repository to a registry.

For example: $ docker push myimage

This command pushes the image named "myimage" to the Docker Hub registry.

These are some of the basic Docker commands for managing containers and images. There are many other Docker commands and options that you can use for more advanced use cases, such as managing networks, volumes, and configuration.

7. Develop a simple containerized application using Docker.

A simple user registration form using Flask and Docker in DevOps.

Solution :

Create a sub folder with expt 7, in your working directory & store all the below files in subfolder.

STEP 1:

Required additional softwares for this prg:

In the terminal of VSCODE install the below commands:

```
pip install flask

python.exe -m pip install --upgrade pip
```

STEP 2:
Create a Docker file with the following content to create a Docker image for your Flask application:

Dockerfile

```
FROM python:3.13
WORKDIR /app
COPY . .
RUN pip install --no-cache-dir -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

STEP 3:

Create a requirements.txt file with the following content to list the dependencies of your Flask application:

requirements.txt :

```
Flask==2.3.2
Jinja2>=3.1
itsdangerous>=2.1
Werkzeug>=2.3
```

STEP 4:
Create app.py file with the following code for a simple user registration form in Flask:

app.py :
```
from flask import Flask, request, render_template
app = Flask(__name__)
@app.route('/register',methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        password = request.form['password']
# Store the user data in a database or file
        return render_template('success.html',name=name)
    return render_template('register.html')
if __name__ == '__main__' :
    app.run(host='0.0.0.0')
```

STEP 5:
Create a **templates folder** and add the following two files: register.html and success.html.

register.html :

```
<form method="post" style="align-items: center;background-color: azure;color: chocolate;">
    name:<input type="text" name="name" placeholder="Name" required><br>
    email:<input type="email" name="email" placeholder="Email" required><br>
    password:<input type="password" name="password" placeholder="Password" required><br>
    <input type="submit" value="Submit">
  </form>
```

success.html :

```
<h2>Registration Successful </h2>
```

STEP 6:
Build the Docker image for your Flask application using the following command:

```
docker build -t simpleflaskapp .
```

STEP 7:
Run a Docker container from the image using the following command in any browser:

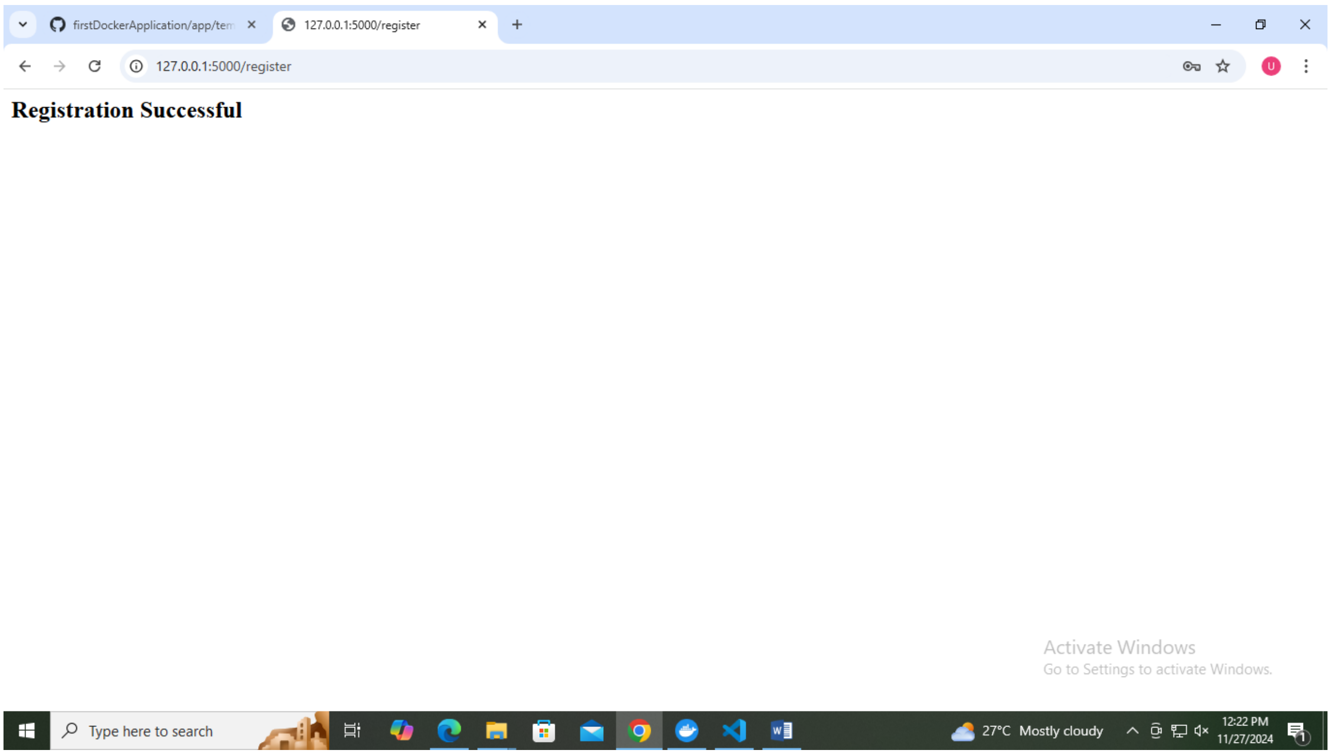
```
docker run -p 5000:5000 simpleflaskapp
```

STEP 8:
Open a web browser and access the registration form at

```
http://localhost:5000/register
```

This example demonstrates how to build a simple user registration form in Flask and run it in a Docker container in DevOps.

Output :



Registration Successful

OTHER WAY FOR 7 PRG (OPTIONAL)

STEPS :

7 . TO CREATE A SIMPLE CONTAINERIZED APPLICATION USING DOCKER

Source Code :

STEP 1 : Install Docker

Make sure Docker is installed on your machine. Follow the instructions for your operating system at Docker's installation page.

STEP 2 : Create a Simple Application

For this example, we'll use a basic Node.js application. Create a folder for your project, then create the following files inside it.

app.js

This will be our main application file.

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

const hostname = '0.0.0.0';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello, Docker World!');
});

server.listen(port, hostname, () => {
  console.log(Server running at http://${hostname}:${port}/);
});
```

package.json

This file is needed to manage dependencies for Node.js applications.

```
{
  "name": "docker-node-app",
  "version": "1.0.0",
  "description": "A simple Node.js app to demonstrate Docker containerization",
  "main": "app.js",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {}
}
```

STEP 3 : Create a Dockerfile

In the same directory, create a Dockerfile. This file defines the environment for the app and how Docker should build it.

```
# Step 1: Use the official Node.js image from the Docker Hub
FROM node:14

# Step 2: Create a directory for the app inside the container
WORKDIR /usr/src/app

# Step 3: Copy package.json and package-lock.json files
COPY package*.json ./

# Step 4: Install dependencies
RUN npm install

# Step 5: Copy the application code into the container
COPY . .

# Step 6: Expose the port the app runs on
EXPOSE 3000

# Step 7: Command to run the application
CMD ["npm", "start"]
```

STEP 4 : Build the Docker Image

In your terminal, navigate to the project directory (where the Dockerfile is located), and run:

```
docker build -t docker-node-app .
```

This command builds the Docker image and tags it as docker-node-app.

STEP 5 : Run the Docker Container

Once the image is built, you can run the container:

```
docker run -p 3000:3000 docker-node-app
```

This command maps port 3000 in the container to port 3000 on your local machine. Now, if you go to http://localhost:3000, you should see "Hello, Docker World!" displayed.

STEP 6 : Stop the Container

To stop the container, you can use Ctrl + C in the terminal where it's running or use Docker commands:

```
docker ps               # to find the container ID
docker stop <container_id>   # replace <container_id> with the actual ID
```

Optional: Push the Image to Docker Hub

If you want to make your Docker image available for others, you can push it to Docker Hub (or any other container registry):

1. Log in to Docker Hub:

```
docker login
```

2. Tag your image with your Docker Hub username:

```
docker tag docker-node-app YOUR_DOCKERHUB_USERNAME/docker-node-app
```

3. Push the image:

```
docker push YOUR_DOCKERHUB_USERNAME/docker-node-app
```

Your containerized app is now set up!

8. Integrate Kubernetes and Docker

AIM: Integrate Kubernetes and Docker

DESCRIPTION:
Kubernetes and Docker are both popular technologies for managing containers, but they are used for different purposes. Kubernetes is an orchestration platform that provides higher-level abstractions for managing containers, while Docker is a containerization technology that provides a lower-level runtime for containers.

To integrate Kubernetes and Docker, you need to use Docker to build and package your application as a container image, and then use Kubernetes to manage and orchestrate the containers. Here's a high-level overview of the steps to integrate Kubernetes and Docker:

• Build a Docker image:

Use Docker to build a Docker image of your application. You can use a Dockerfile to specify the base image, copy the application into the container, and specify the command to run the application.

• Push the Docker image to a registry:

Push the Docker image to a container registry, such as Docker Hub or Google Container Registry, so that it can be easily accessed by Kubernetes.
Deploy the Docker image to a Kubernetes cluster: Use Kubernetes to deploy the Docker image to a cluster.

This involves creating a deployment that specifies the number of replicas and the image to be used, and creating a service that exposes the deployment to the network.

• Monitor and manage the containers: Use Kubernetes to monitor and manage the containers. This includes scaling the number of replicas, updating the image, and rolling out updates to the containers.

• Continuously integrate and deploy changes:
Use a continuous integration and deployment (CI/CD) pipeline to automatically build, push, and deploy changes to the Docker image and the Kubernetes cluster. This makes it easier to make updates to the application and ensures that the latest version is always running in the cluster.

By integrating Kubernetes and Docker, you can leverage the strengths of both technologies to manage containers in a scalable, reliable, and efficient manner

9. Automate the process of running containerized application developed in exercise 7 using Kubernetes

AIM: Automate the process of running containerized application developed in exercise 7 using Kubernetes

DESCRIPTION To automate the process of running the containerized application developed in exercise 7 using Kubernetes, follow the below mentioned steps:

Create a Kubernetes cluster: Create a Kubernetes cluster using a local installation of Minikube.

Push the Docker image to a registry:
Push the Docker image of your application to a container registry, such as Docker Hub or Google Container Registry.

Create a deployment:
Create a deployment in Kubernetes that specifies the number of replicas and the Docker image to use. Here's an example of a deployment YAML file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: myapp
spec:
 replicas: 3
 selector:
 match Labels:
 app: myapp
 template:
 metadata:
 labels:
 app: myapp
 spec:
 containers:
 - name: myapp
 image: myimage
 ports:
 - containerPort: 80
```

• Create a service:
Create a service in Kubernetes that exposes the deployment to the network. Here's an example of a service YAML file:

```
 apiVersion: v1
 kind: Service
 metadata:
name: myapp-service
 spec:
 selector:
 app: myapp
ports:
- name: http
```

```
port: 80
targetPort: 80
 type: ClusterIP
```

• Apply the deployment and service to the cluster:
Apply the deployment and service to the cluster using the kubectl command line tool.
For example: $ kubectl apply -f deployment.yaml

```
 $ kubectl apply -f service.yaml
```

• Verify the deployment: Verify the deployment by checking the status of the pods and the service.

• For example: $ kubectl get pods
$ kubectl get services

This is a basic example of how to automate the process of running a containerized application using Kubernetes. In a real-world scenario, you would likely have more complex requirements, such as managing persistent data, scaling, and rolling updates, but this example should give you a good starting point for using Kubernetes to manage your containers

10. Install and Explore Selenium for automated testing.

AIM: Install and Explore Selenium for automated testing

DESCRIPTION: To install and explore Selenium for automated testing, you can follow these steps:

- Install Java Development Kit (JDK): Selenium is written in Java, so you'll need to install JDK in order to run it. You can download and install JDK from the official Oracle website.
- Install the Selenium WebDriver: You can download the latest version of the Selenium WebDriver from the Selenium website.
- You'll also need to download the appropriate driver for your web browser of choice (e.g. Chrome Driver for Google Chrome).
- Install an Integrated Development Environment (IDE): To write and run Selenium tests,
- you'll need an IDE. Some popular choices include Eclipse, IntelliJ IDEA, and Visual Studio Code
- Write a simple test:
- Once you have your IDE set up, you can write a simple test using the Selenium WebDriver.

Here's an example in Java:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
public class Main
{
 public static void main(String[] args)
{
 System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
 WebDriver driver = new ChromeDriver();
 driver.get("https://www.google.com");
 System.out.println(driver.getTitle());
 driver.quit();
}
}
```

Run the test: Run the test using your IDE or from the command line using the following command:

```
$ javac Main.java
$ java Main
```

This is a basic example of how to get started with Selenium for automated testing. In a real-world scenario, you would likely write more complex tests and organize your code into test suites and test cases, but this example should give you a good starting point for exploring Selenium

EXPERIMENT NO: 11. Write a simple program in JavaScript and perform testing using Selenium

AIM: Write a simple program in JavaScript and perform testing using Selenium

Simple JavaScript program that you can test using Selenium

PROGRAM:

```
<!DOCTYPE html>
<html>
<head>
 <title>Simple JavaScript Program</title>
</head>
<body>
 <p id="output">0</p>
 <button id="increment-button">Increment</button>
 <script>
const output = document.getElementById("output");
const incrementButton=document.getElementById("increment-button");
let count = 0;
incrementButton.addEventListener("click", function() {
count += 1;
output.innerHTML = count;
});
 </script>
</body>
</html>
```

• Write a test case for this program using Selenium

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
public class Main {
private WebDriver driver;
@Before
public void setUp()
{
System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");
driver = new ChromeDriver();
}
@Test
public void testIncrementButton()
{ driver.get("file:///path/to/program.html");
driver.findElement(By.id("increment-button")).click();
String result = driver.findElement(By.id("output")).getText();
assert result.equals("1");
}
@After
public void tearDown()
 {
driver.quit();
}
}
```

You can run the test case using the following command:

```
$ javac Main.java
$ java Main
```

The output of the test case should be:

```
.Time: 0.189
OK (1 test)
```

This output indicates that the test case passed, and the increment button was successfully clicked, causing the output to be incremented by 1.