MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

Department of Computer Science and Engineering(AIML)

DEVOPS LAB

III B.TECH -I SEMESTER
MLRS-R24 REGULATION

A.Y: 2026-27

INDEX

OPEN ENDED EXPERIMENTS		
1.	Implement Kanban board	
2.	Implement list of Docker operations	
3.	Write a Program to implement the containerized methods.	

Department of Computer Science & Engineering (AI&ML)

CERTIFICATE

This is to certify that this manual is a Bonafide record of practical work carried out **in DEVOPS LAB** for the **B. Tech Computer Science Engineering (AI&ML) III Semester** Programme during the academic year **2026–2027.**

This manual has been prepared by **Dr. Hussain Sharif (Associate Professor),** Department of Computer Science Engineering (AI&ML), with my own efforts and to the best of our knowledge.

Signature of Lab Faculty **Signature of HOD**

PREFACE

This Lab Manual entitled "DevOps Lab" is intended for the use of III B. Tech I Semester Computer Science and Engineering(AIML) students of Marri Laxman Reddy Institute of Technology and Management, Dundigal, Hyderabad. The main objective of a DevOps lab is to provide a hands-on, practical environment for implementing **Continuous Integration and Continuous Deployment (CI/CD) pipelines**, which accelerate software delivery speed while enhancing quality, reliability, and security through automation. It serves as a "paved road" or "engine room" for digital transformation, enabling teams to move from manual, error-prone processes to automated, consistent workflows.

By,

Dr. Hussain Sharif

ACKNOWLEDGEMENT

It was really a good experience, working at DEVOPS Lab. First, I would like to thank Dr. Hussain Sharif, Associate Professor, Department of Computer Science & Engineering (AI&ML), Marri Laxman Reddy Institute of technology & Management for giving the technical support in preparing the document.

I express my sincere thanks to Dr. B Ravi Prasad, Head of the Department of Computer Science & Engineering (AI&ML), Marri Laxman Reddy Institute of technology & Management, for his concern towards me and gave me opportunity to prepare D E V O P S laboratory manual.

I am deeply indebted and gratefully acknowledge the constant support and valuable patronage of Dr. B Ravi Prasad, Dean Academics, Marri Laxman Reddy Institute of technology & Management. I am unboundedly grateful to him for timely corrections and scholarly guidance.

I express my heartful thanks to Dr. P. Sridhar, Director, and Dr. R. Murali Prasad, Principal, Marri Laxman Reddy Institute of technology & Management, for giving me this wonderful opportunity for preparing the D E V O P S laboratory manual.

At last, but not the least I would like to thank the entire Computer Science & Engineering Department faculties those who had inspired and helped me to achieve my goal.

By,

Dr.Hussain Sharif

Department of Computer Science & Engineering (AI&ML)

GENERAL INSTRUCTIONS

1. Students are instructed to come to Soft Computing laboratory on time. Late comers are not entertained in the lab.
2. Students should be punctual to the lab. If not, the conducted experiments will not be repeated.
3. Students are expected to come prepared at home with the experiments which are going to be performed.
4. Students are instructed to display their identity cards before entering into the lab.
5. Students are instructed not to bring mobile phones to the lab.
6. Any damage/loss of system parts like keyboard, mouse during the lab session, it is student's responsibility and penalty or fine will be collected from the student.
7. Students should update the records and lab observation books session wise. Before leaving the lab the student should get his/her lab observation book signed by the faculty.
8. Students should submit the lab records by the next lab to the concerned faculty members in the staff room for their correction and return.
9. Students should not move around the lab during the lab session.
10. If any emergency arises, the student should take the permission from faculty member concerned in written format.
11. The faculty members may suspend any student from the lab session on disciplinary grounds.
12. Never copy the output from other students. Write down your own outputs.

Department of Computer Science & Engineering (AI&ML)

SAFETY MEASURES

To ensure the safe and efficient use of the Computer Science and Engineering(AI&ML) laboratory, all students must strictly adhere to the following safety guidelines:

1. General Conduct

- Maintain silence and discipline during lab sessions.
- Do not bring food, drinks, or chewing gum into the lab.
- Use lab resources responsibly and follow all instructions provided by the instructor or lab assistant.

2. Electrical Safety

- Do not touch electrical switches, sockets, or plugs with wet hands.
- Avoid overloading power sockets with unauthorized devices.
- Immediately report any loose connections, sparks, or unusual noises from equipment.

3. Computer and Equipment Handling

- Handle all computer systems, keyboards, mice, and peripherals with care.
- Do not attempt to open or tamper with any hardware components.
- Use only the assigned computer system; do not switch systems without permission.

4. Software and Data Safety

- Use only authorized software installed by the lab administrator.
- Do not attempt to install, uninstall, or modify any software without approval.
- Save your work frequently and ensure backups of important files.

5. Cybersecurity and Network Usage

- Keep your login credentials confidential.
- Do not attempt to access restricted websites or server
- Avoid activities such as hacking, gaming, or the use of pirated content.

6. Emergency Preparedness

- Be familiar with the location of emergency exits, fire extinguishers, and first aid kits.
- In the event of a fire, electrical hazard, or anyemergency, remain calm and inform the lab instructor immediately.
- Follow the evacuation procedure as instructed.

7. Post-Lab Procedures

- Log out of your session and shut down the system properly after use.
- Leave your workstation clean and organized.
- Return any borrowed materials or equipment to their proper place.

8. Hygiene and Cleanliness

- Wash or sanitize your hands before and after using shared devices.
- Do not write or place unnecessary items on the workstation.
- Report any spills or cleanliness issues to the lab staff.

Department of Computer Science & Engineering (AI&ML)

VISION & MISSION OF THE INSTITUTE

Vision of the Institute:

To be a globally recognized institution that fosters innovation, excellence, and leadership in education, research, and technology development, empowering students to create sustainable solutions for the advancement of society.

Mission of the Institute:

- To foster a transformative learning environment that empowers students to excel in engineering, innovation, and leadership.
- To produce skilled, ethical, and socially responsible engineers who contribute to sustainable technological advancements and address global challenges.
- To shape future leaders through cutting-edge research, industry collaboration, and community engagement.

VISION & MISSION OF THE DEPARTMENT

Department Vision:

To nurture globally competent professionals in Artificial Intelligence and Machine Learning through excellence in education, research, and innovation, committed to developing sustainable and impactful solutions for the betterment of society.

Department Mission:

- To provide a transformative learning environment that equips students with in-depth knowledge and practical skills in Artificial Intelligence and Machine Learning, fostering innovation, leadership, and lifelong learning.
- To advance AI and ML through cutting-edge research, strong industry collaboration, and community engagement, preparing students to address real-world challenges on a global scale.
- To produce competent and ethical AI professionals who contribute to technological progress while addressing societal and environmental challenges with sustainable solutions.
- To foster a research-driven culture by partnering with industry and academia, encouraging entrepreneurship, and engaging in community-centered technology development.

Program Educational Objectives (PEOs)

PEO:

Professional Competence:

Graduates will possess strong theoretical and practical knowledge in Artificial Intelligence and Machine Learning, enabling them to solve complex real-world problems, pursue higher education, or excel in professional careers.

Innovation and Research Orientation:

Graduates will engage in innovative practices, cutting-edge research, and contribute to the advancement of AI and ML technologies through collaboration with industry and academia.

Leadership and Lifelong Learning:

Graduates will exhibit leadership qualities, effective communication, and teamwork skills, and will continuously upgrade their knowledge to adapt to evolving technological landscapes.

Entrepreneurial and Community Engagement:

Graduates will leverage entrepreneurial skills and a sense of civic responsibility to create AI-driven solutions that benefit local and global communities.

Course Outcomes (COs): After Completion of the course, Students should be able to

1. Apply version control tools (Git and GitHub) to manage, track, and collaborate on source code effectively.
2. Configure Continuous Integration/Continuous Deployment (CI/CD) pipelines using Jenkins, ensuring automated builds and deployments.
3. Develop, containerize, and deploy applications using Docker and Kubernetes, achieving scalability and automation in application management.
4. Perform automated testing of applications using Selenium, integrating it with CI/CD workflows for quality assurance.
5. Design and execute an end-to-end DevOps workflow, from code development and version control to containerization, deployment, and testing of applications.

Program Outcomes (POs)

PO:

PO1.Engineering Knowledge:
Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO 2. Problem Analysis:
Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO 3.Design/Development of Solutions:
Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO 4. Conduct Investigations of Complex Problems:
Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO 5. Modern Tool Usage:
Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO 6. The Engineer and Society:
Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO 7. Environment and Sustainability:

Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO 8.Ethics:

Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9. Individual and Team Work:

Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO 10. Communication:

Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO 11. Project Management and Finance:

Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO 12. Life-long Learning:

Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: Able to identify, analyze and solve the problems related to Artificial Intelligence and Machine Learning by applying the fundamental knowledge of Computer Science and Engineering. 1. Problem Identification and Formulation 2. Application of Computer Science Fundamentals 3. AI/ML Techniques and Tools Proficiency 4. Analytical and Critical Thinking
PSO2: Build innovative tools and techniques to develop project models in the areas related to Deep Learning, Machine learning, Artificial Intelligence. 1. Innovation and Tool Development 2. Implement end-to-end project models using real-world datasets in domains like image processing, NLP, or predictive analytics. 3. Advanced Technical Proficiency 4. Evaluation and Optimization.
PSO 3: Make use of the Artificial Intelligence and Machine Learning knowledge to assess societal, environmental, health, safety issues, Sustainable development goals with professional ethics and can also pursue higher studies, involve in research activities, be employable or entrepreneur. 1. Application of AI/ML for Societal and Environmental Impact 2. Ethical and Responsible AI Practice 3. Lifelong Learning and Research Orientation 4. Employability and Entrepreneurship

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING(AIML)

B.TECH V SEMESTER
CSM TIME TABLE

ACADEMIC YEAR: 2026-2027

2450585: DEVOPS LABORATORY

L T P C
0 0 2 1

Course Objectives:

- To introduce students to fundamental DevOps practices such as version control, continuous integration, continuous deployment, and automation.
- To enable learners to explore and apply modern tools like Git, GitHub, Jenkins, Docker, Kubernetes, and Selenium in real-world scenarios.
- To develop the ability to design and manage containerized applications, ensuring scalability, efficiency, and reliability.
- To provide hands-on experience in automating software development workflows, from source code management to deployment and testing.
- To cultivate problem-solving and analytical skills by integrating DevOps tools for building, deploying, and testing applications in a collaborative environment.

Course Outcomes: After completion of the course, student should be able to

- Apply version control tools (Git and GitHub) to manage, track, and collaborate on source code effectively.
- Configure Continuous Integration/Continuous Deployment (CI/CD) pipelines using Jenkins, ensuring automated builds and deployments.
- Develop, containerize, and deploy applications using Docker and Kubernetes**,** achieving scalability and automation in application management.
- Perform automated testing of applications using Selenium**,** integrating it with CI/CD workflows for quality assurance.
- Design and execute an end-to-end DevOps workflow, from code development and version control to containerization, deployment, and testing of applications.

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.
13. Develop Package and deploy the multi-service registration app to Kubernetes using Helm charts and expose it via Ingress.
14. Convert the single registration app into a small multi-service system and orchestrate locally with Docker Compose.

TEXT BOOKS:

1. Joakim Verona., Practical DevOps, Packt Publishing, 2016.

REFERENCE BOOKS:

1. Deepak Gaikwad, Viral Thakkar. DevOps Tools from Practitioner's Viewpoint. Wiley publications.
2. Len Bass, Ingo Weber, Liming Zhu. DevOps: A Software Architect's Perspective. Addison Wesley.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AIML)

DEVOPS LABORATORY

Virtual lab details

Name of the Virtual Lab: DevOps Lab Virtual

Lab Host Institute: IIT KHARAGPUR

URL/Link to Lab: https://scte-iitkgp.vlabs.ac.in/

AcademicYear:2026-2027

Semester: V

List of Experiments Available in Virtual Lab

1. Introduction to Fundamental of Git and GitHub Operations
2. Docker implementation
3. Kubernetes installation

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI & ML)

DEVOPS LABORATORY

LAB PLANNER CSM-A BATCH-1

S.No	Experiment	CO	Virtual Lab Availabilty	Date planned	Date Conducted
1	Write a code for a simple user registration form for an event	CO1	NA		
2	Explore Git and GitHub Commands	CO1	NA		
3	Practice Source code management on GitHub .experiment with the source code in exercise 1	CO2	NA		
4	Jenkins installation and setup, explore the environment	CO2	NA		
5	Demonstrate continuous integration and development using Jenkins.	CO3	NA		
MID-I					
6	Explore Docker commands for content management .	CO4	NA		
7	Develop a simple containerized application using Docker	CO4	NA		
8	Integrate Kubernetes and Docker	CO5	NA		
9	Automate the process of running containerized application for exercise 7 using kubernetes	CO5	NA		
10	Install and explore selenium for automated testing	CO5	NA		
MID-II					

CSM-A BATCH-2

S.No	Experiment	CO	Virtual Lab Availabilty	Date planned	Date Conducted
1	Write a code for a simple user registration form for an event	CO1	NA		
2	Explore Git and GitHub Commands	CO1	NA		
3	Practice Source code management on GitHub .experiment with the source code in exercise 1	CO2	NA		
4	Jenkins installation and setup, explore the environment	CO2	NA		
5	Demonstrate continuous integration and development using Jenkins.	CO3	NA		
MID-I					
6	Explore Docker commands for content management .	CO3	NA		
7	Develop a simple containerized application using Docker	CO4	NA		
8	Integrate Kubernetes and Docker	CO4	NA		
9	Automate the process of running containerized application for exercise 7 using kubernetes	CO5	NA		
10	Install and explore selenium for automated testing	CO5	NA		
MID-II					

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI & ML)

DEVOPS LABORATORY

LAB PLANNER

Date planed																																
Date conducted																																
Roll Number	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*		Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	

Note:VL*-Virtual Lab Availabilty

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING (AI & ML)

DEVOPS LABORATORY

RUBRICS USED TO ASSESS LEARNINGS IN LABORATORIES

1. RUBRICS FOR DAY TO DAY EVALUATION

Parameter	Max Marks	Level-1 (Very Poor)	Level-2 (Poor)	Level-3 (Average)	Level-4 (Good)	Level-5 (Excellent)
Observation Book	05	No observations or irrelevant data. (0–1)	Incomplete or incorrect data. (2)	Basic values with some errors. (3)	Mostly correct with good format. (4)	Fully correct, clear, and well-formatted. (5)
Record Writing	05	Not submitted. (0-1)	Submitted but mostly incomplete. (2)	Submitted with some missing/wrong parts. (3)	Submitted with minor issues. (4)	Fully complete, correct algorithm & flowchart. (5)
Result	05	No result or major errors. (0–1)	Result partially obtained. (2)	Acceptable result with limited error. (3)	Near-correct result and reasonable error. (4)	Accurate result. (5)
Viva-Voce	05	Did not answer any questions. (1)	Answered very few questions. (2)	Answered some questions with help. (3)	Answered most questions correctly. (4)	Answered all questions accurately. (5)

2. RUBRICS FOR INTERNAL EVALUATION

Criterion	Max Marks	Level-1 *(Very Poor)*	Level-2 *(Poor)*	Level-3 *(Average)*	Level-4 *(Good)*	Level-5 *(Excellent)*
Design/Tool/Apparatus Selection	2 Marks	Incorrect tool/design and no reasoning. **(0)**	Tool/design selection attempted with unclear logic. **(0.5)**	Satisfactory selection with partial justification. **(1)**	Correct selection and proper analysis with few errors. **(1.5)**	Smart selection with accurate, relevant analysis. **(2)**
Execution (Code/Debug/Run) /Analysis/Method Used	4 Marks	Did not attempt or completely failed to execute. **(0)**	Attempted but unable to proceed or with major errors. **(1)**	Partial execution with some logic/syntax errors. **(2)**	Mostly correct execution with minimal help. **(3)**	Fully correct and independently executed program. **(4)**
Results& Documentation	2 Marks	Incomplete or poorly presented. **(0)**	Basic structure but lacks clarity or formatting. **(0.5)**	Complete but genericor with formatting issues. **(1)**	Well-structured and mostly clear. **(1.5)**	Well-organized, professional, and engaging documentation. **(2)**
Viva-Voce (Understanding of Concepts)	2 Marks	No understanding; could not answer questions. **(0)**	Answered a few with difficulty. **(0.5)**	Answered half the questions with basic clarity. **(1)**	Good understanding with confident answers. **(1.5)**	Answered all questions with clarity and depth. **(2)**

3. RUBRICS FOR SEMESTER END EXAMINATIONS

Criterion	Max Marks	Level-1 (Very Poor *(0–2 marks)*	Level-2 (Poor)*(3–4 marks)*	Level-3 (Average)*(5–6 marks)*	Level-4 (Good)*(7–9 marks)*	Level-5 (Excellent)*(10–12 marks)*
Preparedness for the Experiment	12 marks	No clarity or objective or procedure. Unable to explain basics.	Limited idea of the objective/procedure. Needed prompting.	Has basic understanding; minor gaps in concept or preparation.	Well-prepared, with clear understanding of steps and background.	Fully prepared with strong conceptual clarity and confident explanation.
Performance in the Laboratory	12 marks	Unable to perform experiment. Relied entirely on examiner's help.	Performed with multiple errors and constant support.	Performed with some errors; required occasional help.	Performed mostly independently with minimal support.	Performed independently, efficiently, and with precision.
Calculations & Graphs	12 marks	No or incorrect calculations. Graphs missing or irrelevant.	Multiple calculation errors. Graphs/plots inaccurate or poorly labeled.	Calculations partially correct. Graphs present but with some flaws.	Correct calculations and graphs with minor errors.	Accurate calculations and well-labeled graphs with proper interpretation.
Results & Error Analysis	12 marks	No result or invalid result. No error analysis attempted.	Incorrect result with vague or no error discussion.	Acceptable result. Error analysis attempted but limited.	Correct result with sound error discussion.	Accurate result with detailed and relevant error analysis.
Viva-Voce (Subject Knowledge)	12 marks	Unable to answer any questions. No conceptual understanding.	Answered few questions with poor logic.	Answered half of the questions with average understanding.	Answered most questions with clarity and confidence.	Answered all questions with depth, clarity,and reasoning.

EXPERIMENT-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%;
```

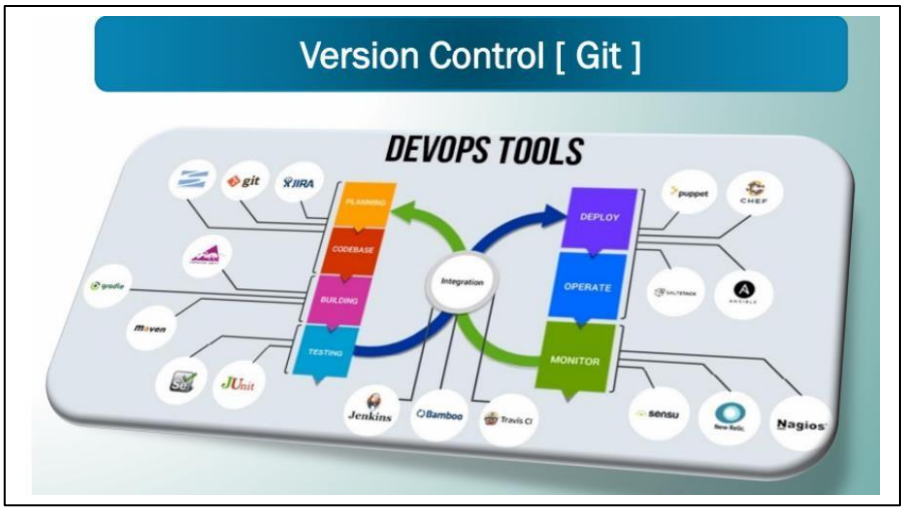
```
    height:
    40px;
    padding:
    10px;
    border: 1px solid
    #ccc; border-
    radius: 5px;
}

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

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

      button[type="submit"]:ho
        ver { 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>
```

EXPERIMENT- 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.

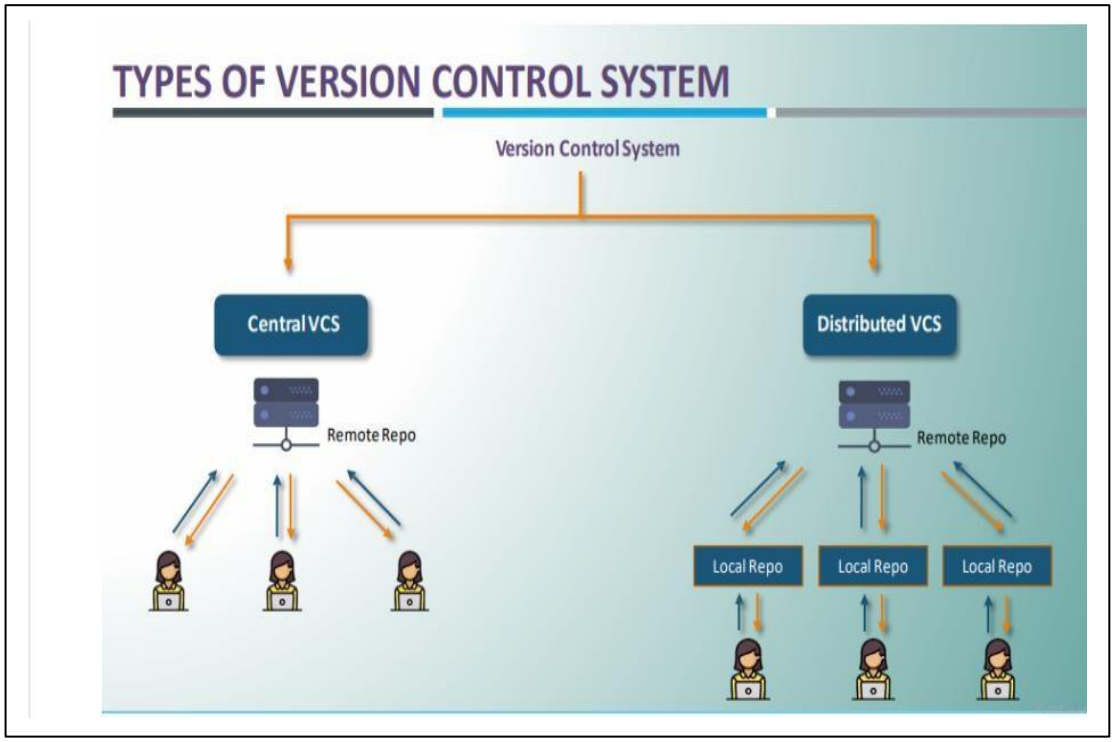
Git is a distributed version control system (VCS) that helps developers track changes in their codebase, collaborate with others, and manage different versions of their projects efficiently. It was created by Linus Torvalds in 2005 to address the shortcomings of existing version control systems.



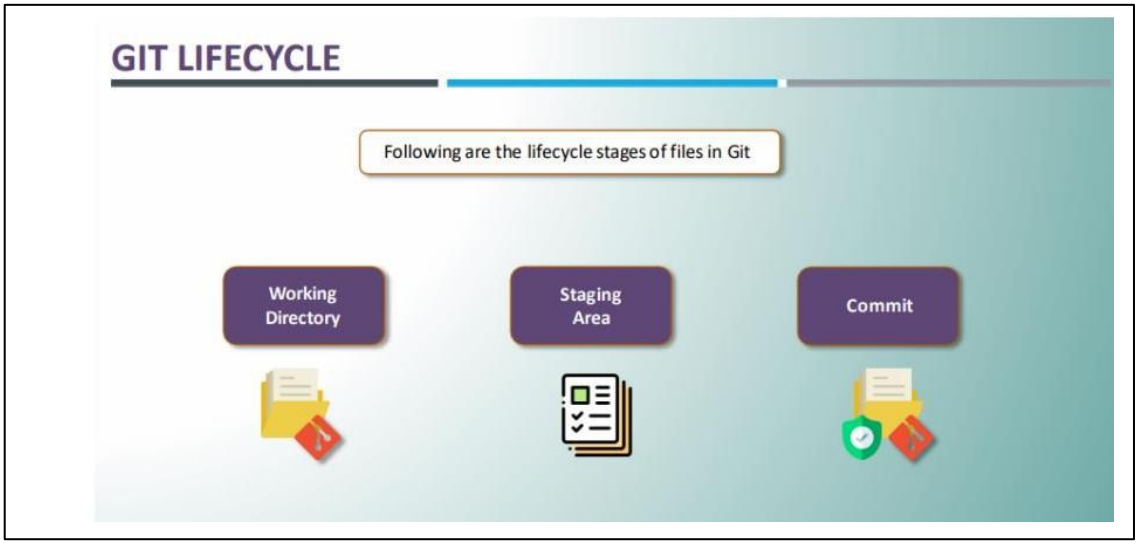
Unlike traditional centralised VCS, where all changes are stored on a central server, Git follows a distributed model. Each developer has a complete copy of the repository on their local machine, including the entire history of the project. This decentralisation offers numerous advantages, such as offline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


TYPES OF VERSION CONTROL SYSTEM
Version Control System
Central VCS
Distributed VCS
Remote Repo
Remote Repo
Local Repo
Local Repo
Local Repo

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)

EXPERIMENT-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"]:ho
      ver { 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

EXPERIMENT-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

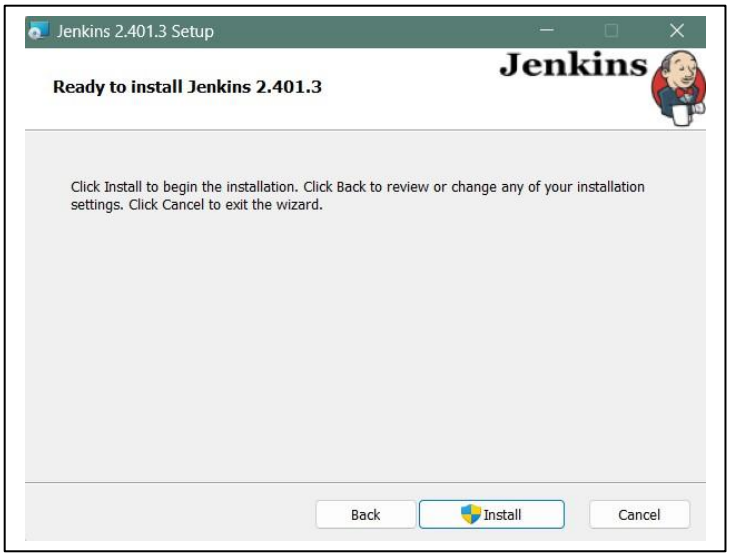
Docker images. Download Jenkins 2.479.1 LTS for:

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

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

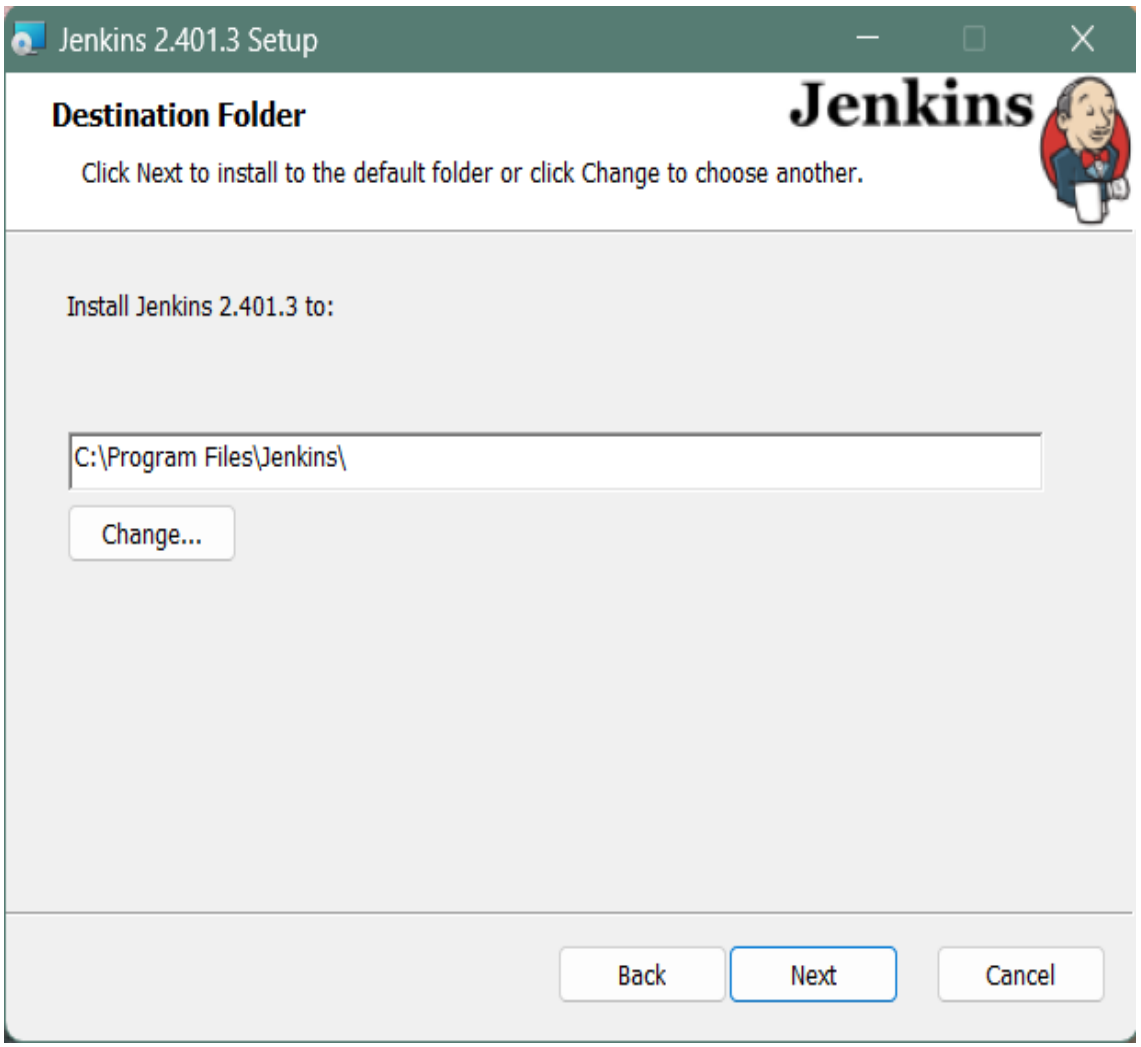
Step 1: Setup wizard

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



Step 2: Select destination folder

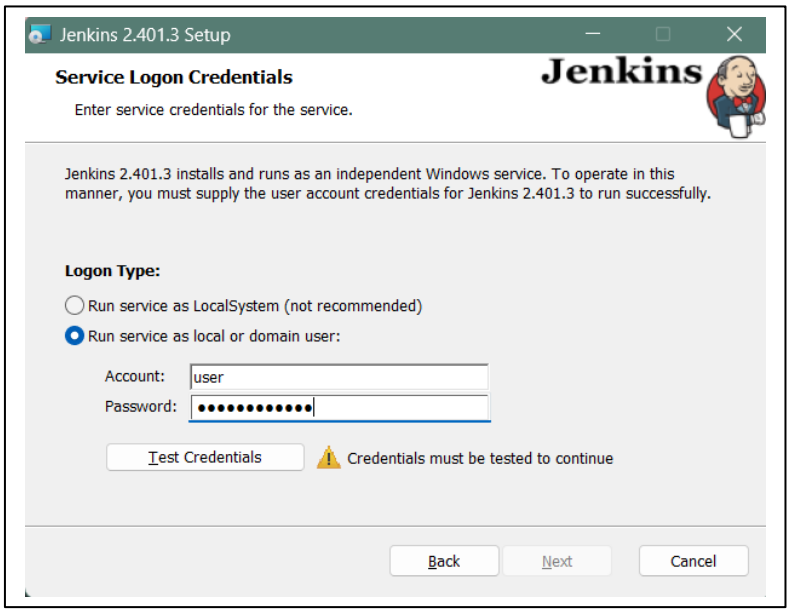
Select the destination folder to store your Jenkins Installation and click **Next** to continue.


Jenkins 2.401.3 Setup
Destination Folder
Click Next to install to the default folder or click Change to choose another.
Jenkins
Install Jenkins 2.401.3 to:
C:\Program Files\Jenkins\
Change...
Back
Next
Cancel

Step 3: Service logon credentials

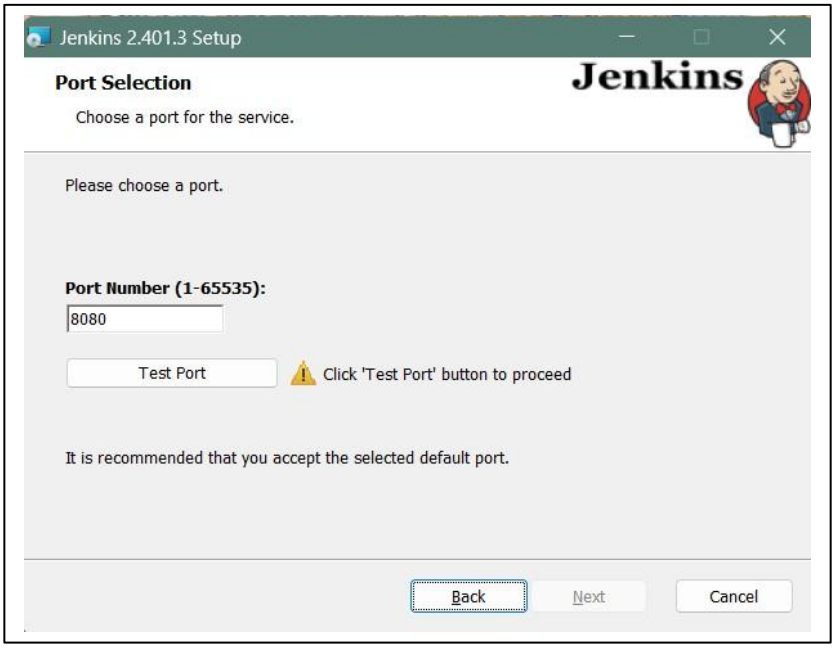
When Installing Jenkins, it is recommended to install and run Jenkins as an independent windows service using a **local or domain user** as it is much safer than running Jenkins using **Local System(Windows equivalent of root)** which will grant Jenkins full access to your machine and services.

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**.



Step 4: Port selection

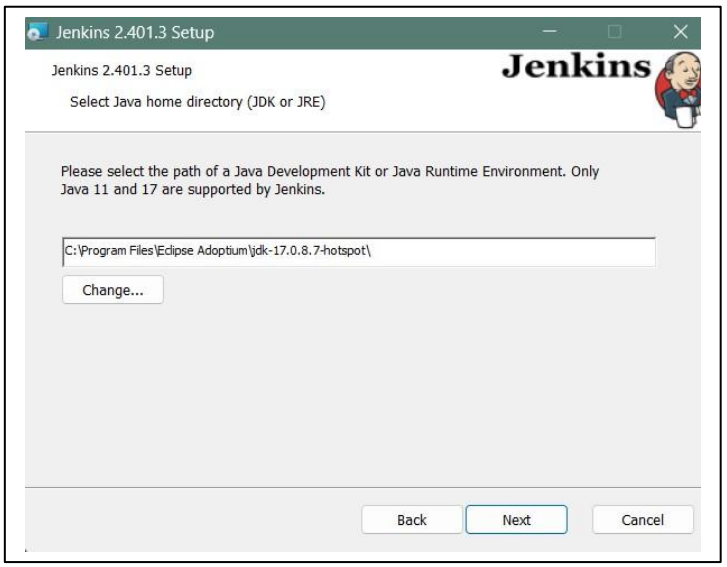
Specify the port on which Jenkins will be running, **Test Port** button to validate whether the specified port if free on your machine or not. Consequently, if the port is free, it will show a green tick mark as shown below, then click on **Next**.



Step 5: Select Java home directory

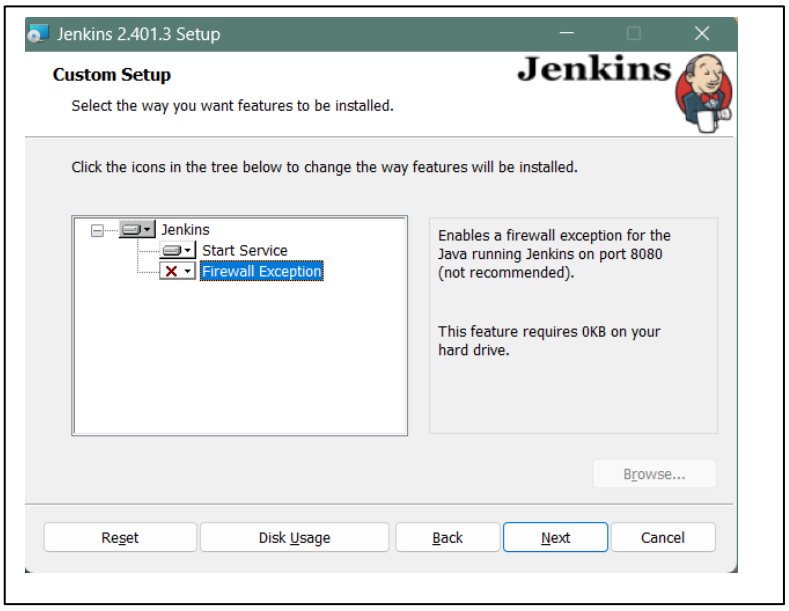
The installation process checks for Java on your machine and prefills the dialog with the Java home directory. If the needed Java version is not installed on your machine, you will be prompted to install it.

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



Step 6: Custom setup

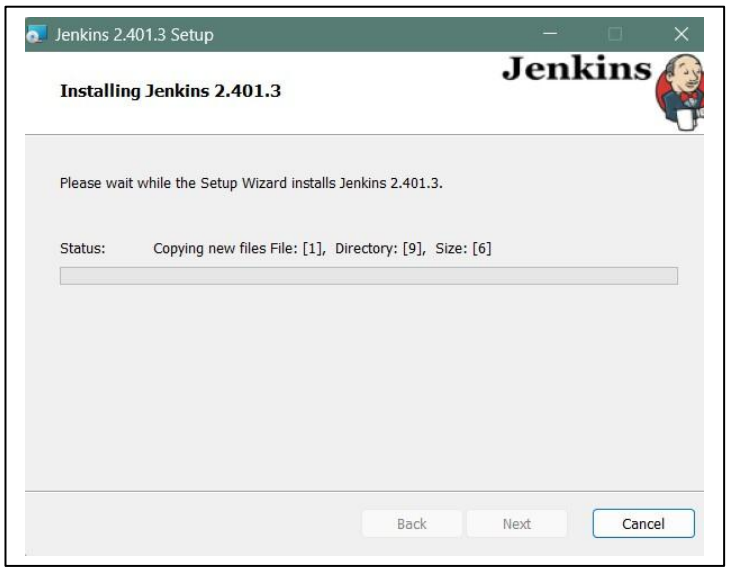
Select other services that need to be installed with Jenkins and click on **Next**.



Step 7: Install Jenkins

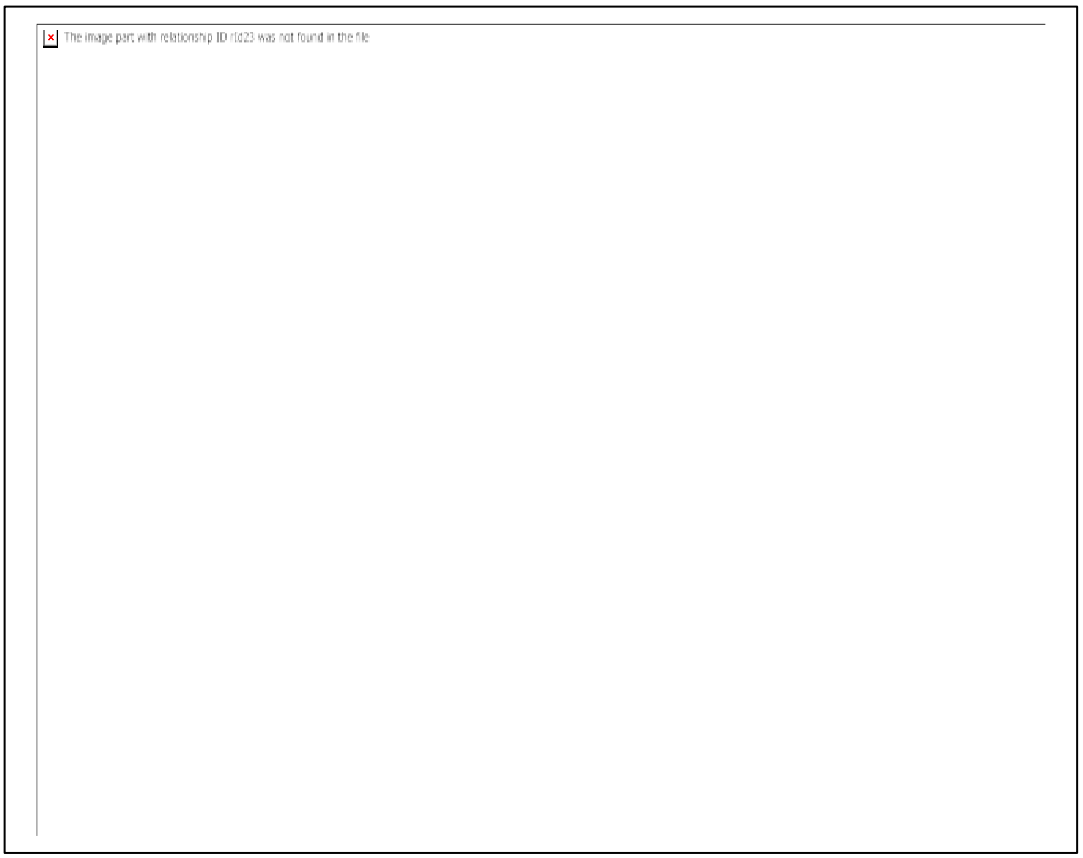
Click on the **Install** button to start the installation of Jenkins.

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

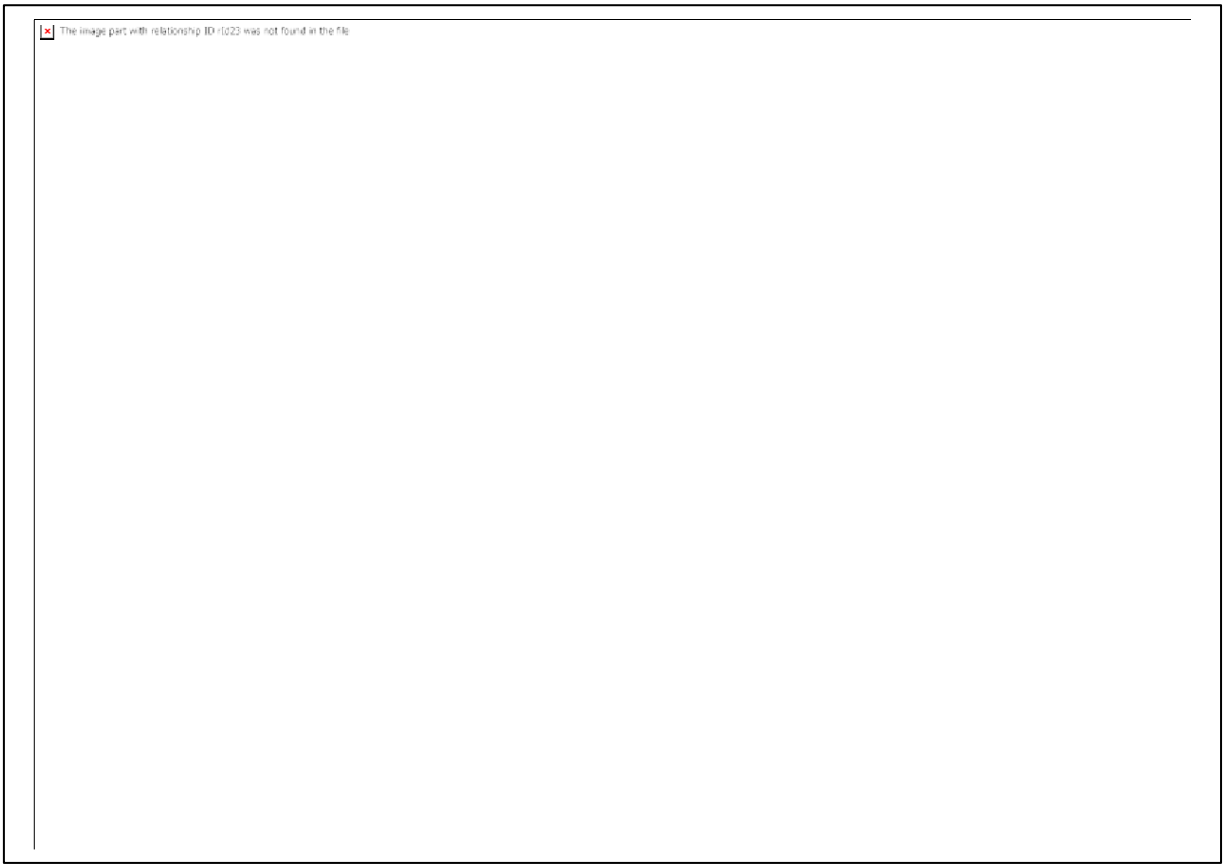


Step 8: Finish Jenkins installation

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:



EXPERIMENT-5

Demonstrate continuous integration and development using Jenkins.

Here's a clear **demonstration of Continuous Integration (CI) and Continuous Development using Jenkins**, step by step, as you might present in a lab or practical exam.

- **Continuous Integration (CI):** Developers frequently push code → automatically built & tested.
- **Continuous Delivery/Deployment (CD):** Code is automatically prepared or deployed to production.

Jenkins is a popular automation server used to implement CI/CD pipelines.

- Java installed (Jenkins requirement)
- Jenkins installed
- Git installed
- A sample project (e.g., Java/Maven or Node.js)
- GitHub repository

```
# Ubuntu example
sudo apt update
sudo apt install openjdk-11-jdk -y
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
sudo apt install jenkins -y
sudo systemctl start jenkins
```

Access Jenkins:

```
http://localhost:8080
```

```
mvn archetype:generate \
  -DgroupId=com.example \
  -DartifactId=demo-app \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DinteractiveMode=false
```

Push code to GitHub.

Steps:

1. Open Jenkins Dashboard
2. Click **New Item**
3. Enter name → Select **Freestyle Project**
4. Configure:

Source Code Management

- Select **Git**
- Enter repository URL

```
mvn clean install
```

Instead of freestyle, use **Pipeline**:

Create a `Jenkinsfile` in your repo:

```
pipeline {
    agent any

    stages {
        stage('Clone') {
            steps {
                git 'https://github.com/user/demo-app.git'
            }
        }

        stage('Build') {
            steps {
                sh 'mvn clean compile'
            }
        }

        stage('Test') {
            steps {
                sh 'mvn test'
            }
        }
```

EXPERIMENT-6

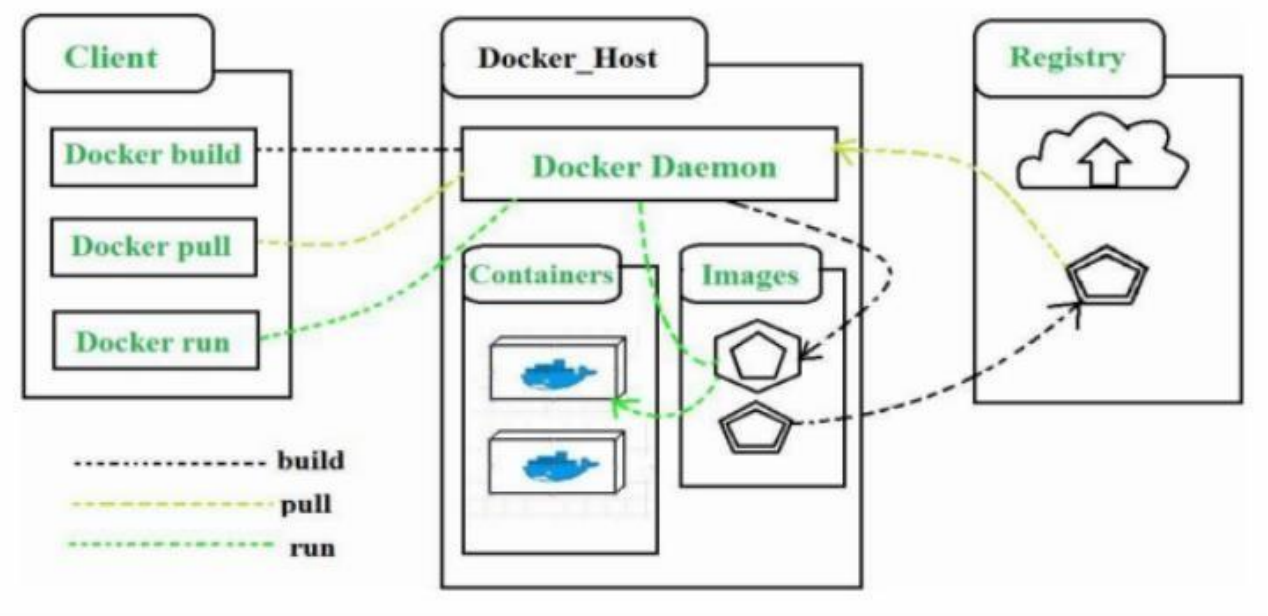
Explore Docker commands for content management.

AIM: Explore Docker commands for content management.

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

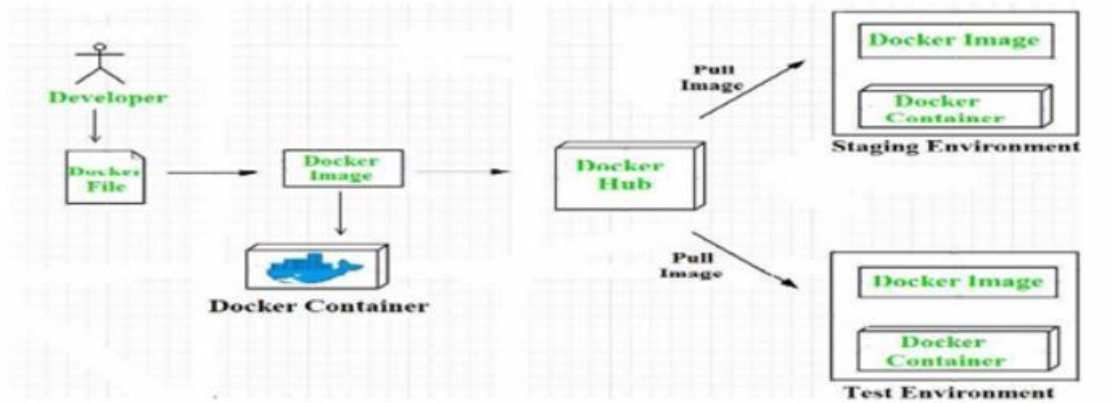
Docker Architecture:

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.

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.

EXPERIMENT-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:

```
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:

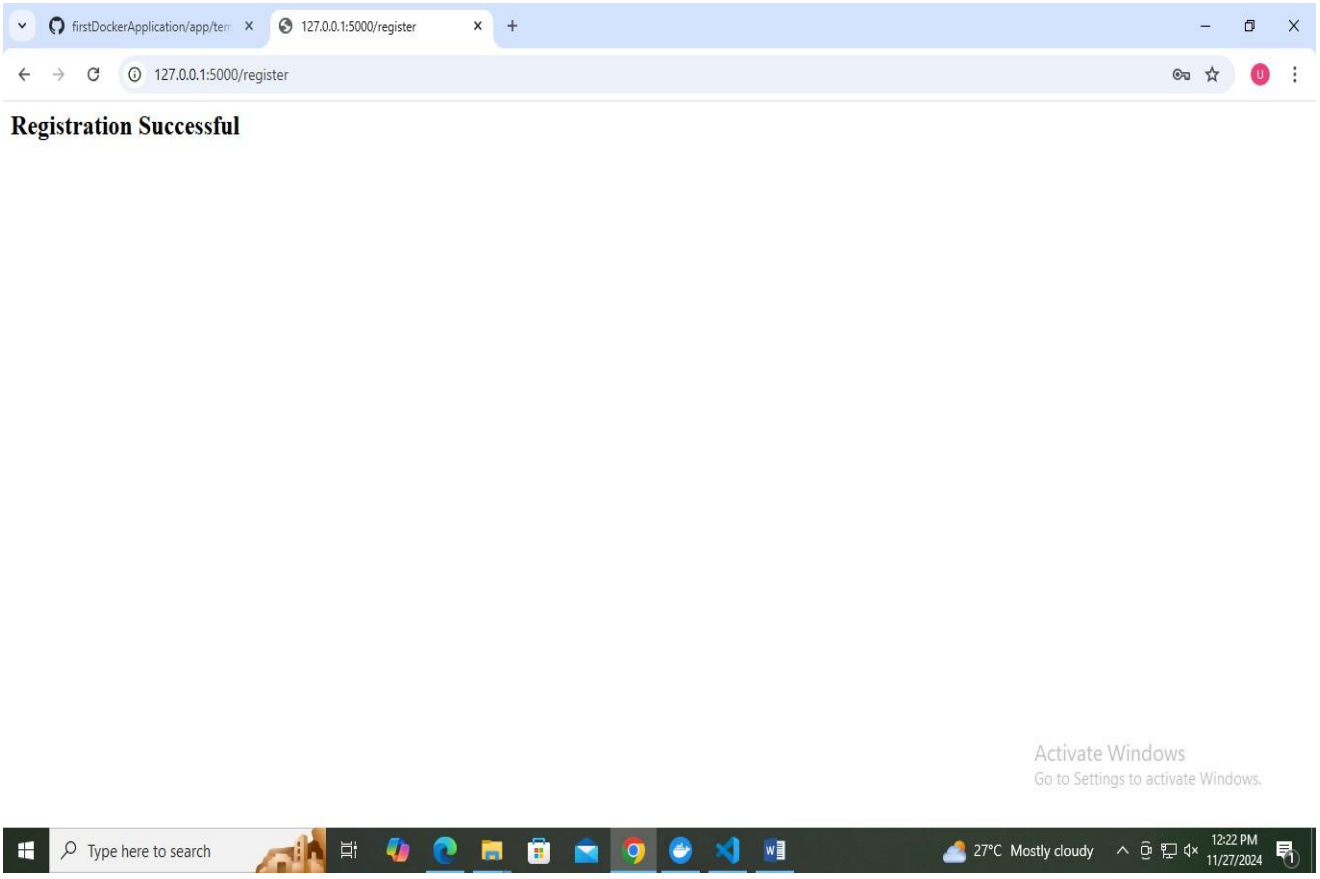
```
docker run -p 5000:5000 simpleflaskapp
```

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 :


firstDockerApplication/app/ter
127.0.0.1:5000/register
127.0.0.1:5000/register
Registration Successful
Activate Windows
Go to Settings to activate Windows.
Type here to search
27°C Mostly cloudy
12:22 PM
11/27/2024

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!

EXPERIMENT-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

EXPERIMENT-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

EXPERIMENT-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-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.

1. ndle multiple fuzzy output sets?

EXPERIMENT-12

Aim: Develop test cases for the above containerized application using selenium

Functional Test Cases

1. Verify application loads successfully
2. Verify login functionality (valid credentials)
3. Verify login failure (invalid credentials)
4. Verify navigation between pages
5. Verify form submission
6. Verify logout functionality

Test case 1:

```
@Test
public void testAppLaunch() {
    WebDriver driver = new ChromeDriver();
    driver.get("http://localhost:8080");

    String title = driver.getTitle();
    Assert.assertTrue(title.contains("My App"));

    driver.quit();
}
```

Test case 2:

```
@Test
public void testValidLogin() {
    WebDriver driver = new ChromeDriver();
    driver.get("http://localhost:8080/login");

    driver.findElement(By.id("username")).sendKeys("admin");
    driver.findElement(By.id("password")).sendKeys("password");
    driver.findElement(By.id("loginBtn")).click();
```

```
    String dashboard = driver.findElement(By.id("dashboard")).getText();
    Assert.assertTrue(dashboard.contains("Welcome"));

    driver.quit();
}
```

Test case 3:

```
@Test
public void testInvalidLogin() {
    WebDriver driver = new ChromeDriver();
    driver.get("http://localhost:8080/login");

    driver.findElement(By.id("username")).sendKeys("wrong");
    driver.findElement(By.id("password")).sendKeys("wrong");
    driver.findElement(By.id("loginBtn")).click();

    String error = driver.findElement(By.id("errorMsg")).getText();
    Assert.assertTrue(error.contains("Invalid"));

    driver.quit();
}
```

Test case 4:

```
@Test
public void testNavigation() {
    WebDriver driver = new ChromeDriver();
    driver.get("http://localhost:8080");

    driver.findElement(By.linkText("Profile")).click();
    Assert.assertTrue(driver.getCurrentUrl().contains("profile"));

    driver.quit();
}
```

Test case 5:

```
@Test
public void testFormSubmission() {
```

```
WebDriver driver = new ChromeDriver();
   driver.get("http://localhost:8080/form");

   driver.findElement(By.id("name")).sendKeys("Sharif");
   driver.findElement(By.id("submit")).click();

   String successMsg = driver.findElement(By.id("success")).getText();
   Assert.assertTrue(successMsg.contains("Submitted"));

   driver.quit();
}
```

EXPERIMENT-13

Aim: Develop Package and deploy the multi-service registration app to Kubernetes using Helm charts and expose it via Ingress.

Below is a **complete, practical guide** to **package and deploy a multi-service registration app to Kubernetes using Helm** and **expose it via Ingress**.

Assumed Application Architecture

Your app has multiple services:

- **frontend-service** (UI)
- **user-service** (registration logic)
- **auth-service** (authentication)
- **database** (MySQL/PostgreSQL)

Project Structure (Helm Chart)

```
registration-app/
│
├── charts/
├── templates/
│   ├── frontend-deployment.yaml
│   ├── frontend-service.yaml
│   ├── user-deployment.yaml
│   ├── user-service.yaml
│   ├── auth-deployment.yaml
│   ├── auth-service.yaml
│   ├── db-deployment.yaml
│   ├── db-service.yaml
│   ├── ingress.yaml
│
├── values.yaml
├── Chart.yaml
```

Create Helm Chart

```
helm create registration-app
cd registration-app
```

Edit files accordingly.

Chart.yaml

```
apiVersion: v2
name: registration-app
description: Multi-service registration app
version: 1.0.0
appVersion: "1.0"
```

values.yaml (Central Configuration)

```
frontend:
  image: frontend:latest
  port: 80

user:
  image: user-service:latest
  port: 8081

auth:
  image: auth-service:latest
  port: 8082

db:
  image: mysql:5.7
  port: 3306
  rootPassword: root

ingress:
  enabled: true
  host: registration.local
```

Sample Deployment

templates/frontend-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: {{ .Values.frontend.image }}
        ports:
        - containerPort: {{ .Values.frontend.port }}
```

templates/frontend-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  type: ClusterIP
  selector:
    app: frontend
  ports:
    - port: 80
      targetPort: {{ .Values.frontend.port }}
```

Database Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db
spec:
  replicas: 1
  selector:
    matchLabels:
      app: db
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
      - name: mysql
        image: {{ .Values.db.image }}
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: {{ .Values.db.rootPassword }}
        ports:
        - containerPort: 3306
```

Ingress Configuration

templates/ingress.yaml

```
{{- if .Values.ingress.enabled }}
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: registration-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  -
```

```
host: {{ .Values.ingress.host }}
http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend-service
            port:
              number: 80
{{- end }}
```

EXPERIMENT-14

Convert the single registration app into a small multi-service system and orchestrate locally with Docker Compose.

User Service (Node.js Example)

```
// user-service/app.js
const express = require('express');
const app = express();
app.use(express.json());

app.post('/register', (req, res) => {
    const { username, password } = req.body;
    res.json({ message: "User registered successfully" });
});

app.listen(8081, () => console.log("User Service running"));
```

Auth Service

```
// auth-service/app.js
const express = require('express');
const app = express();
app.use(express.json());

app.post('/login', (req, res) => {
    res.json({ message: "Login successful" });
});

app.listen(8082, () => console.log("Auth Service running"));
```

Frontend (Simple HTML)

```
<!-- frontend/index.html -->
<!DOCTYPE html>
<html>
<body>
<h2>Registration</h2>

<form id="form">
  <input id="username" placeholder="Username">
  <input id="password" placeholder="Password">
  <button type="submit">Register</button>
</form>
```

```
<script>
document.getElementById('form').onsubmit = async (e) => {
  e.preventDefault();
  await fetch('http://localhost:8081/register', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({
      username: username.value,
      password: password.value
    })
  });
  alert("Registered!");
};
</script>

</body>
</html>
```

User Service Dockerfile

```
FROM node:18
WORKDIR /app
COPY . .
RUN npm install express
CMD ["node", "app.js"]
```

Auth Service Dockerfile

```
FROM node:18
WORKDIR /app
COPY . .
RUN npm install express
CMD ["node", "app.js"]
```

Frontend Dockerfile

```
FROM nginx:alpine
COPY . /usr/share/nginx/html


services:

  frontend:
    build: ./frontend
    ports:
      - "3000:80"
    depends_on:
      - user-service
      - auth-service

  user-service:
    build: ./user-service
    ports:
      - "8081:8081"
```

```
environment:
      DB_HOST: db

  auth-service:
    build: ./auth-service
    ports:
      - "8082:8082"
    depends_on:
      - db
    environment:
      DB_HOST: db

  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: root
    ports:
      - "3306:3306"
```