KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

DEVOPS LABORATORY MANUAL

Subject Code : KG23ACS315
Regulation : KGR23
Academic Year : 2025-2026

III B .Tech II Sem
COMPUTER SCIENCE AND ENGINEERING

KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

Affiliated o JNTUH, Chilkur,(V), Moinabad(M) R. R Dist, TS-501504

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

VISION AND MISSION OF THE INSTITUTION

VISION:

To become an institution which is internationally recognized for its holistic approach to engineering, innovative teaching and learning culture, research and entrepreneurial ecosystem, and sustainable social impact in the community.

MISSION:

• To offer undergraduate and post-graduate programs which are supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed in their professional careers.
• To provide faculty and students with an ecosystem that fosters innovation, research, entrepreneurship, and international exposure through strategic partnerships with governmentorganizations and collaboration with industries.
• To provide holistic learning environment to students which will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
• To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic development.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION:

To be recognized as a department of excellence by stimulating a learning environment in which students and faculty will thrive and grow to achieve their professional, institutional and societal goals.

MISSION:

- To provide high quality technical education to students that will enable life-long learning and build expertise in advanced technologies in Computer Science and Engineering.
- To promote research and development by providing opportunities to solve complex engineering problems in collaboration with industry and government agencies.
- To encourage professional development of students that will inculcate ethical values and leadership skills through entrepreneurship while working with the community to address societalissues.

PROGRAM EDUCATIONAL OBJECTIVES

PEO 1: Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles.

PEO 2: Graduates have successful careers in computer science and engineering fields or will be able to successfully pursue advanced degrees.

PEO 3: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism, moral and ethical responsibility.

PEO 4: Graduates will develop the ability to understand and analyze engineering issues in a broader perspective with ethical responsibility towards sustainable development.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

PO I: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO II: 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 III: 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 IV: 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 V: 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 VI: 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 VII: 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 VIII: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO IX: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO X: 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 XI: 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 XII: 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.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM SPECIFIC OUTCOMES

PSO1: The Computer Science and Engineering graduates are able to analyze, design, develop, test and apply management principles, mathematical foundations in the development of intelligent systems with computational solutions, make them to expert in designing the secure application and hardware prototype
PSO2: The graduating student will be analyze the contemporary research issues in different areas of computer science & engineering and explore research gaps, analyze and carry out research in the specialized/emerging areas.
PSO3: Develop their skills to solve problems in the broad area of programming concepts and appraise environmental and social issues with ethics and manage different projects in multi- disciplinary field to conducive in cultivating skills for successful career, entrepreneurship and higher studies.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEVOPS LABORATORY

	L	T	P	C
Course Code: KG23ACS315				
B.Tech. III Year II – Semester	**0**	**0**	**2**	**1**

Prerequisites:

1. Software Engineering

Course Objectives: The objectives of this course for the students are to:

1. Enable students to understand the fundamentals and significance of DevOps practices and tools.
2. Equip students with practical skills in source code management using Git and GitHub.
3. Provide hands-on experience in Continuous Integration/Continuous Deployment (CI/CD) using Jenkins.
4. Train students in containerization using Docker and orchestration using Kubernetes.
5. Develop skills in automated software testing using Selenium for high-quality software deliver

Course Outcomes: After completion of this course, the students will be able to

CO1: Understand the necessity, principles, and advantages of using DevOps tools.

CO2: Demonstrate proficiency in source code management and version control using Git and GitHub.

CO3: Apply Jenkins to implement CI/CD pipelines for software delivery.

CO4: Use Docker to develop containerized applications and automate them using Kubernetes.

CO5: Perform automated software testing using Selenium to ensure quality assurance in the DevOps pipeline.

LIST OF EXPERIMENTS:

S.NO	Name of the experiment
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 exerc
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 Kubernet
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.

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.

EXPERIMENT-1:Write code for a simple user registration form for an event.

CODE:

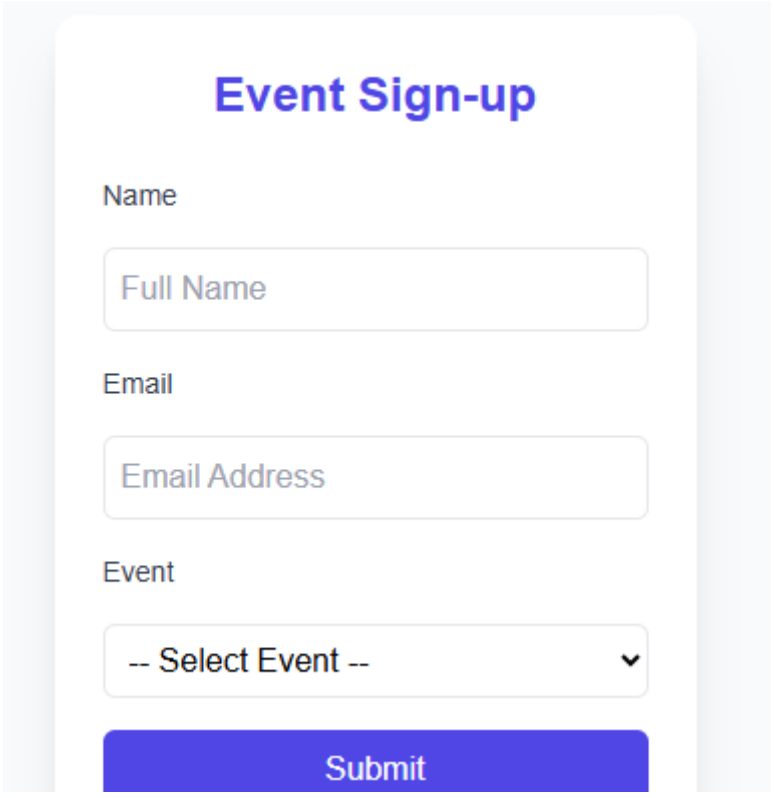
html lang="en">

```
<head>
<meta charset="UTF-8"><meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Event Registration</title>
<script src="https://cdn.tailwindcss.com"></script>
<script>
tailwind.config = { theme: { extend: { fontFamily: { sans: ['Inter', 'sans-serif'] } } } };
</script>
</head>
<body class="flex items-center justify-center min-h-screen p-4 bg-gray-50 font-sans">
<div class="w-full max-w-xs bg-white p-6 rounded-xl shadow-xl">
<h1 class="text-2xl font-semibold mb-6 text-center text-indigo-600">Event Sign-up</h1>
<div id="error-message" class="hidden mb-4 p-2 text-sm bg-red-100 border border-red-400 text-red-700 rounded-md">
Fill all fields.
</div>
<form id="registration-form" class="space-y-4">
<label class="block text-sm font-medium text-gray-700">Name</label>
<input type="text" id="fullName" required placeholder="Full Name" class="w-full p-2 border rounded-md focus:ring-indigo-500 focus:border-indigo-500">
<label class="block text-sm font-medium text-gray-700">Email</label>
<input type="email" id="email" required placeholder="Email Address" class="w-full p-2 border rounded-md focus:ring-indigo-500 focus:border-indigo-500">
<label class="block text-sm font-medium text-gray-700">Event</label>
```

```
<select id="eventChoice" required class="w-full p-2 border rounded-md bg-white focus:ring-indigo-500 focus:border-indigo-500">
<option value="">-- Select Event --</option>
<option value="a">Day 1 Keynote</option>
<option value="b">Workshop</option>
</select>
<button type="submit" class="w-full py-2 mt-4 bg-indigo-600 text-white font-medium rounded-md hover:bg-indigo-700">Submit</button>
</form>
<div id="success-message" class="hidden mt-6 p-4 bg-green-100 border-l-4 border-green-500 text-green-700 rounded-lg">
<p class="font-bold">Registered!</p><p id="confirmation-details" class="text-sm"></p>
<button onclick="resetF()" class="mt-2 px-3 py-1 bg-green-600 text-white text-xs font-medium rounded hover:bg-green-700">New</button>
</div>
</div>
<script>
const form = document.getElementById('registration-form'), msgBox = document.getElementById('success-message');
const detailsEl = document.getElementById('confirmation-details'), errorEl = document.getElementById('error-message');
const resetF = () => { form.reset(); msgBox.classList.add('hidden'); form.classList.remove('hidden'); errorEl.classList.add('hidden'); };
form.addEventListener('submit', function(e) {
e.preventDefault(); errorEl.classList.add('hidden');
const n = document.getElementById('fullName').value.trim(), em = document.getElementById('email').value.trim();
const el = document.getElementById('eventChoice'); const et = el.options[el.selectedIndex].text;
```

```
if (!n || !em || !el.value) { errorEl.classList.remove('hidden'); return; }
detailsEl.innerHTML = `Registered for: ${et}. Confirmed via ${em}.`;
form.classList.add('hidden'); msgBox.classList.remove('hidden');
});
</script>
</body>
</html>
```

OUTPUT:



EXPERIMENT-2: Explore Git and GitHub commands.

CODE:

Essential Git & GitHub Workflow: The 6 Core Commands (Simple List)

This lab program shows you the six commands needed to go from an empty folder to a synchronized repository on GitHub.

1. Local Tracking (Git)

These commands save your work locally on your machine.

Command 1: Start Tracking

- Command: git init
- Action: Initializes a new local repository in your project folder.

Command 2: Prepare Files

- Command: git add .
- Action: Stages all modified files, preparing them for the commit.

Command 3: Save Changes

- Command: git commit -m "First commit"
- Action: Commits the staged changes, creating a permanent snapshot (your "save point").

2. Remote Synchronization (GitHub)

These commands connect your local work to a remote server like GitHub.

Command 4: Connect Remote

- Command: git remote add origin <URL>
- Action: Links your local repo to the remote GitHub repository URL

  (replace <URL> with your GitHub link).Command 5: Set Main Branch
- Command: git branch -M main
- Action: Renames your local branch to main (standard practice).

Command 6: Upload Work

- Command: git push -u origin main
- Action: Uploads your local commits to the remote repository on GitHub.

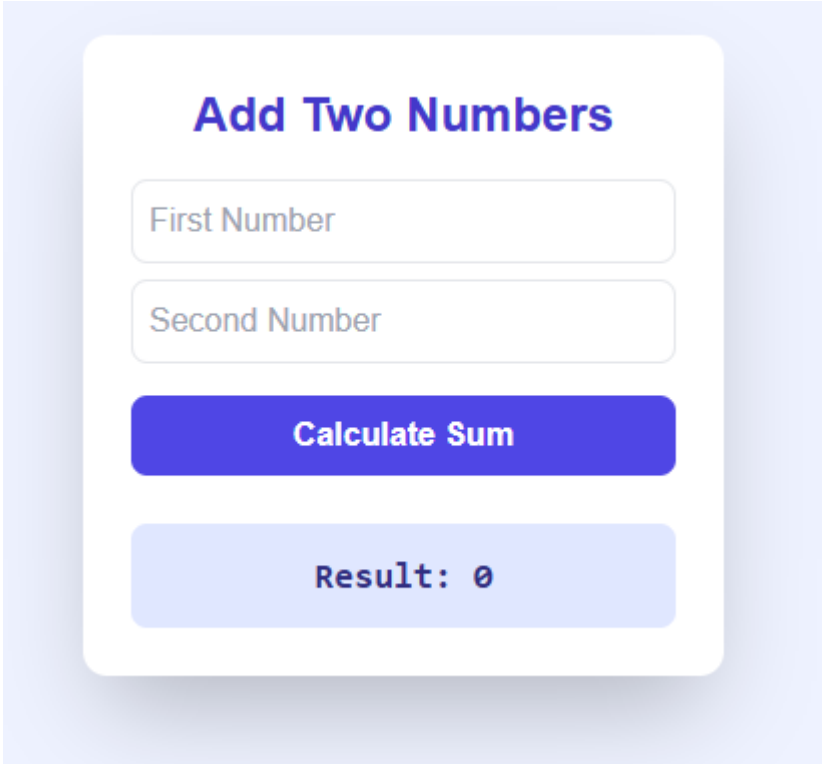
EXPERIMENT-3: Practice Source code management on GitHub. Experiment with the source code in exercise1

CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Simple Adder</title>
<script src="https://cdn.tailwindcss.com"></script>
<script>
tailwind.config = { theme: { extend: { fontFamily: { sans: ['Inter', 'sans-serif'] } } } };
</script>
</head>
<body class="flex items-center justify-center min-h-screen bg-indigo-50 font-sans p-4">
<div class="bg-white p-6 rounded-xl shadow-2xl w-full max-w-xs text-center">
<h1 class="text-2xl font-bold mb-4 text-indigo-700">Add Two Numbers</h1>
<input type="number" id="num1" placeholder="First Number" class="w-full p-2 border rounded-lg mb-2 focus:ring-indigo-500">
<input type="number" id="num2" placeholder="Second Number" class="w-full p-2 border rounded-lg mb-4 focus:ring-indigo-500">
<button onclick="calculate()" class="w-full py-2 bg-indigo-600 text-white font-semibold rounded-lg hover:bg-indigo-700 transition">
Calculate Sum
</button>
<div id="result" class="mt-6 p-3 bg-indigo-100 text-indigo-900 rounded-lg text-lg font-mono font-bold">
```

```
Result: 0
</div>
</div>
<script>
function calculate() {
const n1 = parseFloat(document.getElementById('num1').value) || 0;
const n2 = parseFloat(document.getElementById('num2').value) || 0;
const sum = n1 + n2;
document.getElementById('result').textContent = `Result: ${sum}`;
}
</script>
</body>
</html>
```

OUTPUT:



EXPERIMENT-4: Jenkins installation and setup, explore the environment.

CODE:

Jenkins is an open-source automation tool used for building, testing, and deploying code. It supports Continuous Integration and Continuous Delivery (CI/CD) by automating repetitive development tasks.

1)To install Jenkins on Ubuntu or any Debian-based system, first update your system and install Java:

- sudo apt update
- sudo apt install openjdk-17-jdk -y

2)Next, add the Jenkins repository and install Jenkins:

- wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
- echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null
- sudo apt update
- sudo apt install jenkins -y

3)Start Jenkins and enable it to run automatically:

- sudo systemctl start jenkins
- sudo systemctl enable jenkins

4)Now open your web browser and go to:
http://localhost:8080

5)To unlock Jenkins, get the admin password:

- sudo cat /var/lib/jenkins/secrets/initialAdminPassword

6)Copy the password, paste it into the Jenkins setup page, install suggested plugins, and create your admin user.

7)After setup, the Jenkins dashboard will appear. From here, you can create new jobs, manage system settings, install plugins, and check build history.

8)To explore the environment, go to Manage Jenkins to view system configuration, credentials, and tool installations. Jenkins files are stored in /var/lib/jenkins.

9)Create a sample job by clicking New Item, selecting Freestyle Project, and adding a build step:

- echo "Hello Jenkins!"

10)Save and click Build Now to see the results in Console Output.

You can view environment variables inside a build by running:

- printenv | sort

Common variables include BUILD_NUMBER, JOB_NAME, WORKSPACE, and GIT_COMMIT.

Jenkins is now installed, configured, and ready to automate your projects.

EXPERIMENT-5: Demonstrate continuous integration and development using Jenkins.

CODE:

Continuous Integration (CI): Automatically builds and tests your code whenever you make changes (like pushing to GitHub).

Continuous Deployment (CD): Automatically deploys your code after successful testing.

Jenkins helps do both — it builds, tests, and deploys your project automatically.

Step 1: Prerequisites

Jenkins installed and running (http://localhost:8080)

A GitHub repository with your project

Jenkins Git plugin installed

Step 2: Create a Pipeline Job

Go to Jenkins Dashboard → New Item

Enter a name → select Pipeline → click OK

Scroll to Pipeline section → choose Pipeline script or Pipeline script from SCM (Git)

Step 3: Add a Jenkinsfile

In your project folder, create a file named Jenkinsfile with this content:

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Building the project...'
            }
        }
        stage('Test') {
            steps {
                echo 'Running tests...'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying the application...'
```

```
        }
      }
    }
}
```

This tells Jenkins to build, test, and deploy your code step by step.

Step 4: Connect Jenkins with GitHub

In your Jenkins job, go to Pipeline → Definition → Pipeline script from SCM

Select Git, enter your repository URL, and add credentials if needed.

Save the job.

Now Jenkins will pull the code from GitHub.

Step 5: Run the Pipeline

Click Build Now

You'll see stages like Build, Test, and Deploy run automatically in the console output.

If all stages pass, your pipeline is successful

Step 6: Continuous Integration in Action

Whenever you push code to GitHub, Jenkins automatically detects the changes, pulls the latest code, builds it, runs tests, and deploys it — no manual steps needed.

EXPERIMENT-6: Explore Docker commands for content management.

CODE:

Docker is a tool that lets you package applications and their dependencies into containers, which can run anywhere. Content management in Docker usually means managing images, containers, and data.

Step 1: Check Docker Version docker --version

This shows if Docker is installed.

Step 2: List Docker Commands docker help

It lists all Docker commands like run, ps, images, pull, rm, etc.

Step 3: Work with Images

List images:docker images

Pull an image from Docker Hub: docker pull ubuntu

Remove an image:docker rmi ubuntu

Step 4: Work with Containers

Run a container:docker run -it ubuntu

List running containers:docker ps

List all containers: docker ps -a

Stop a container:docker stop <container_id>

Remove a container:docker rm <container_id>

Step 5: Manage Content Inside Containers

Copy files from host to container:docker cp localfile.txt <container_id>:/path/in/container/

Copy files from container to host:docker cp <container_id>:/path/in/container/file.txt ./hostfolder/

Open a shell in a running container:docker exec -it <container_id> /bin/bash

Step 6: Volumes (Persistent Storage)

Create a volume:docker volume create mydata

Run container with volume:docker run -v mydata:/data -it ubuntu

This keeps data even if the container is removed.

EXPERIMENT-7: Develop a simple containerized application using Docker.

CODE:

Docker is a tool that lets you package applications and their dependencies into containers, which can run anywhere. Content management in Docker usually means managing images, containers, and data.

Step 1: Check Docker Version

docker --version

This shows if Docker is installed.

Step 2: List Docker Commands

docker help

It lists all Docker commands like run, ps, images, pull, rm, etc.

Step 3: Work with Images

List images:docker images

Pull an image from Docker Hub:docker pull ubuntu

Remove an image:docker rmi ubuntu

Step 4: Work with Containers

Run a container:docker run -it ubuntu

List running containers:docker ps

List all containers:docker ps -a

Stop a container:docker stop <container_id>

Remove a container:docker rm <container_id>

Step 5: Manage Content Inside Containers

Copy files from host to container:docker cp localfile.txt <container_id>:/path/in/container/

Copy files from container to host:docker cp <container_id>:/path/in/container/file.txt ./hostfolder/

Open a shell in a running container:docker exec -it <container_id> /bin/bash

Step 6: Volumes (Persistent Storage)

Create a volume:docker volume create mydata

Run container with volume:docker run -v mydata:/data -it ubuntu

This keeps data even if the container is removed.

EXPERIMENT- 8: Integrate Kubernetes and Docker

CODE:

Step 1: Understand the Basics

Docker: Packages your application into containers.

Kubernetes (K8s): Manages and runs containers across multiple machines automatically.
Think of Docker as creating the containers and Kubernetes as orchestrating them.

Step 2: Install Kubernetes and Docker

Install Docker on your system.

Install Minikube (a simple Kubernetes cluster for local use):

- curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
- sudo install minikube-linux-amd64 /usr/local/bin/minikube
- minikube start

Check Kubernetes status:

- kubectl get nodes

Step 3: Build Docker Image

Build your Docker image as usual:

- docker build -t myapp:1.0 .

If using Minikube, make Docker use Minikube's environment:

- eval $(minikube docker-env)
- docker build -t myapp:1.0 .

Step 4: Create Kubernetes Deployment

A Deployment tells Kubernetes to run containers. Create deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
```

```
    labels:
      app: myapp
  spec:
    containers:
    - name: myapp
      image: myapp:1.0
      ports:
      - containerPort: 80
```

Step 5: Apply Deployment

- kubectl apply -f deployment.yaml
- kubectl get pods

This creates and runs your containers in Kubernetes.

Step 6: Expose Application

Expose your deployment so it's accessible:

- kubectl expose deployment myapp-deployment --type=NodePort --port=80
- minikube service myapp-deployment

Your app will open in the browser.

EXPERIMENT-9: Automate the process of running containerized application for exercise 7 using Kubernetes

CODE:

Step 1: Make Sure Prerequisites Are Ready

Docker image of your app is built (from Exercise 7):

- docker build -t myapp:1.0 .

Minikube (or any Kubernetes cluster) is installed and running:

- minikube start

kubectl is installed and configured.

Step 2: Use Minikube's Docker Environment

To make Kubernetes use your local Docker image:

- eval $(minikube docker-env)
- docker build -t myapp:1.0 .

Step 3: Create a Deployment YAML

Create a file called myapp-deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myapp:1.0
        ports:
```

- containerPort: 80

replicas: 2 means Kubernetes will run 2 copies of your app automatically.

Step 4: Apply the Deployment

Run this command to tell Kubernetes to create the deployment:

- kubectl apply -f myapp-deployment.yaml
- kubectl get pods

You will see the pods (containers) running automatically.

Step 5: Expose Your App

Make your app accessible:

- kubectl expose deployment myapp-deployment --type=NodePort --port=80
- minikube service myapp-deployment

Kubernetes automatically creates a service and opens your app in the browser.

Step 6: Automate Updates

If you update your Docker image, rebuild it:

- docker build -t myapp:1.0 .

Then tell Kubernetes to update the deployment:

- kubectl rollout restart deployment myapp-deployment
- kubectl get pods

EXPERIMENT-10: Install and Explore Selenium for automated testing.

CODE:

Selenium is a tool used for automated testing of web applications. It can control browsers like Chrome, Firefox, etc., and perform tasks automatically.

Step 1: Install Python (if not installed)

Check Python version:

- python --version

Install Python if needed from python.org.

Step 2: Install Selenium Library

Use pip to install Selenium:

- pip install selenium

Check installation:

- python -c "import selenium; print(selenium.__version__)"

Step 3: Download WebDriver

Selenium needs a WebDriver to control the browser:

- Chrome → ChromeDriver
- Firefox → GeckoDriver

Make sure the WebDriver version matches your browser version.

Step 4: Create a Simple Test Script

Example Python script (test.py) for Chrome:

```
from selenium import webdriver
from selenium.webdriver.common.by import By

# Open Chrome browser
driver = webdriver.Chrome(executable_path='/path/to/chromedriver')

# Open a website
driver.get("https://www.google.com")

# Find search box and type something
search_box = driver.find_element(By.NAME, "q")
```

```
search_box.send_keys("Selenium Automation")

# Submit the search
search_box.submit()

# Close the browser
driver.quit()
```

Step 5: Run the Script

- python test.py

You will see Chrome open, perform the search automatically, and close.

Step 6: Explore Selenium Features

- Find elements: find_element(By.ID, "id"), find_element(By.CLASS_NAME, "class")
- Click buttons: element.click()
- Send keys: element.send_keys("text")
- Waits: driver.implicitly_wait(10)
- Screenshots: driver.save_screenshot("screen.png")

EXPERIMENT-11: Write a simple program in JavaScript and perform testing using Selenium.

CODE:

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

# Open Chrome browser (set path to chromedriver)
driver = webdriver.Chrome(executable_path='/path/to/chromedriver')

# Open the local HTML file (use absolute path)
driver.get("file:///full/path/to/index.html")

# Click the button
driver.find_element(By.ID, "changeText").click()

# Wait a moment for the text to change
time.sleep(1)

# Verify the heading text
heading = driver.find_element(By.ID, "greeting")
assert heading.text == "Hello, Selenium!"

print("Test Passed!")

# Close the browser
driver.quit()
```

OUTPUT:

```
python test_selenium.py
Test Passed
```

EXPERIMENT-12: Develop test cases for the above containerized application using selenium.

CODE:

Application: A web page with a heading and a button:

- Initial heading: "Hello, World!"
- Button changes the heading to "Hello, Selenium!"

Test Cases

Test Case 1: Verify Page Loads

Objective: Ensure the web page opens successfully in the browser.

Steps:

- Open the web page.
- Check the heading element exists.

Expected Result: Page loads and heading "Hello, World!" is visible.

- heading = driver.find_element(By.ID, "greeting")
- assert heading.text == "Hello, World!"

Test Case 2: Verify Button Click Changes Text

Objective: Ensure clicking the button changes the heading.

Steps:

- Find the button element.
- Click the button.
- Check the heading text.

Expected Result: Heading changes to "Hello, Selenium!".

```
button = driver.find_element(By.ID, "changeText")
button.click()
heading = driver.find_element(By.ID, "greeting")
assert heading.text == "Hello, Selenium!"
```

Test Case 3: Verify Button Exists

Objective: Ensure the button is present on the page.

Steps:

Find the button element by ID.

Expected Result: Button exists.

```
button = driver.find_element(By.ID, "changeText")
assert button is not None
```

Test Case 4: Verify Heading Doesn’t Change Without Click

Objective: Ensure heading text remains the same if the button is not clicked.

Steps:

- Open the page.
- Wait for 1 second.
- Check the heading text.

Expected Result: Heading remains "Hello, World!".

```
import time
time.sleep(1)
heading = driver.find_element(By.ID, "greeting")
assert heading.text == "Hello, World!"
```