

Bachelor of Technology

CSE (AI&ML)

REGULATION: R18



LAB MANUAL

for

DEVOPS LAB

VAAGESWARI COLLEGE OF ENGINEERING

BESIDE LMD POLICE STATION, RAMAKRISHNA COLONY

KARIMNAGAR - 505481

INDEX

S.NO	CONTENTS	PAGE NO
I	V/M /PEOS/POS/PSOS	3
II	Syllabus	6
III	Course Objectives, Course outcomes	6
IV	List of Experiments	6
V	Experiments	6
Exp No	Experiment Name	
1	Write code for a simple user registration form for an event.	7
2	Explore Git and GitHub commands	8-9
3	Practice Source code management on GitHub. Experiment with the source code written in exercise 1.	10
4	Jenkins installation and setup, explore the environment.	11-12
5	Demonstrate continuous integration and development using Jenkins.	13
6	Explore Docker commands for content management.	14-15
7	Develop a simple containerized application using Docker.	16-17
8	Integrate Kubernetes and Docker	18-19
9	Automate the process of running containerized application developed in exercise 7 using Kubernetes.	
10	Install and Explore Selenium for automated testing.	
11	Write a simple program in JavaScript and perform testing using Selenium.	
12	Develop test cases for the above containerized application using selenium.	

Department Of Computer Science & Engineering(AI&ML)

VISION:

To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in them.

MISSION:

DM1: To prepare Computer Science and Engineering students to be an avid learners with competence in basic science & engineering with proficiency in multidisciplinary areas, so that they can succeed in industry as an individual or as a team member or as an entrepreneur.

DM2: To enable the graduates to use modern tools, design and create novelty based products required for the society and communicate effectively with professional ethics.

DM3: To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students into competent and confident engineers

Sl.No.	PEO#	Program Educational Objectives
1	PEO - 1	To promote industrial growth and social transformation students must practice their profession with ethics and produce responsible graduates within them
2	PEO – 2	Apply fundamental knowledge for problem analysis and conduct computer science and engineering research for continual progress
3	PEO – 3	To adapt to a fast changing environment through acquisition and applications of skills and technologies illustrated by students using their abilities

PROGRAM EDUCATIONAL OBJECTIVES:

Computer Science & Engineering

PO1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	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.
PO3	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.
PO4	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.
PO5	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
PO6	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 pra
PO7	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.
PO8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	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
P011	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.
P012	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

PSO1	The students must design and develop the solutions using the latest software for the objective of task execution in a particular domain.
PSO2	Utilize computer science and Information Technology Techniques for industrial application in independent systems like cloud computing , Artificial intelligence, natural language processing, computer vision and emerging areas

DEVOPS LAB

B.Tech. III Year II Sem.

Course Objectives:

1. Describe the agile relationship between development and IT operations.
2. Understand the skill sets and high-functioning teams involved in
3. DevOps and related methods to reach a continuous delivery capability
4. Implement automated system update and DevOps lifecycle

Course Outcomes:

1. Identify components of Devops environment
2. Apply different project management, integration, testing and code deployment tool
3. Investigate different DevOps Software development, models
4. Demonstrate continuous integration and development using Jenkins.

List of Experiments:

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

10. Install and Explore Selenium for automated testing.
11. Write a simple program in JavaScript and perform testing using Selenium.
12. Develop test cases for the above containerized application using selenium.

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

```
<html>

<body>

<form>

  <label for="name">Name:</label>

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

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

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

  <label for="phone">Phone:</label>

  <input type="tel" id="phone" name="phone"><br><br>

  <label for="event">Event:</label>

  <select id="event" name="event">

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

    <option value="seminar">Seminar</option>

    <option value="workshop">Workshop</option>

  </select><br><br>

  <label for="comments">Comments:</label><br>

  <textarea id="comments" name="comments"></textarea><br><br>
```



```
<input type="submit" value="Register">

</form>

</body>

</html>
```

Experiment 2:

Explore Git and GitHub commands.

Version control systems are tools that help a software team manage changes to source code over time. For almost all software projects, the source code is like the crown jewels—a precious asset whose value must be protected. VCS tools are sometimes known as SCM (Source Code Management) tool.

The most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source tool originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel.

Two types Version Controlling

- 1) Centralized Version controlling
- 2) Distributed Version controlling

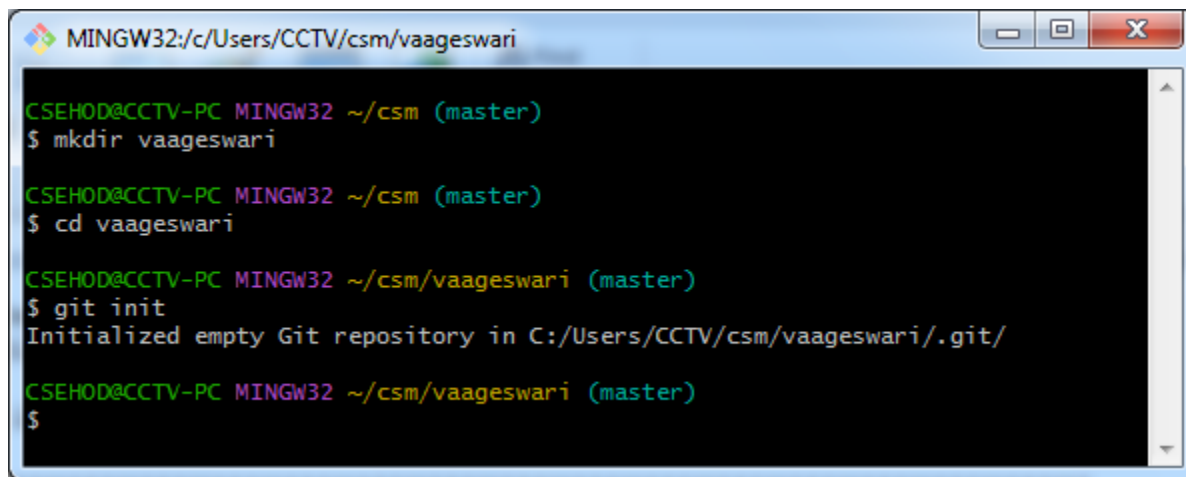
Git is Distributed Version controlling.

Git is a version control system that is widely used in software development to manage changes to source code. GitHub is a popular web-based hosting service for Git repositories. Here are some common Git and GitHub commands:

Git commands:

1.git init: Initializes a new Git repository in the current directory.

Example:



```
MINGW32:/c/Users/CCTV/csm/vaageswari

CSEHOD@CCTV-PC MINGW32 ~/csm (master)
$ mkdir vaageswari

CSEHOD@CCTV-PC MINGW32 ~/csm (master)
$ cd vaageswari

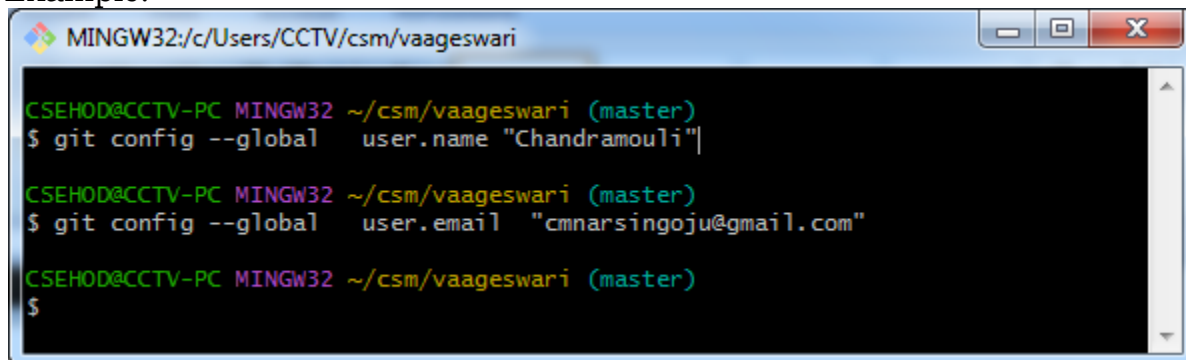
CSEHOD@CCTV-PC MINGW32 ~/csm/vaageswari (master)
$ git init
Initialized empty Git repository in C:/Users/CCTV/csm/vaageswari/.git/

CSEHOD@CCTV-PC MINGW32 ~/csm/vaageswari (master)
$
```

2. To configure username and email for git

```
$ git config --global user.name "Chandramouli"
$ git config --global user.email "cmnarsingoju@gmail.com"
```

Example:



```
MINGW32:/c/Users/CCTV/csm/vaageswari

CSEHOD@CCTV-PC MINGW32 ~/csm/vaageswari (master)
$ git config --global user.name "Chandramouli"

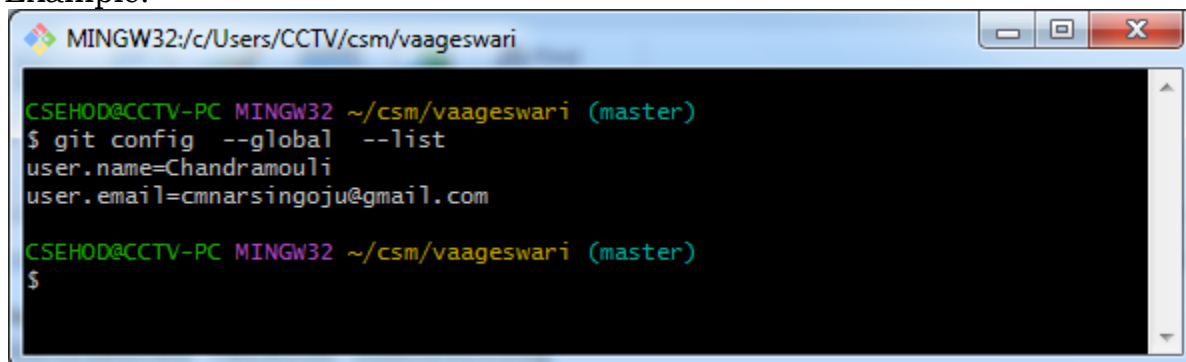
CSEHOD@CCTV-PC MINGW32 ~/csm/vaageswari (master)
$ git config --global user.email "cmnarsingoju@gmail.com"

CSEHOD@CCTV-PC MINGW32 ~/csm/vaageswari (master)
$
```

3. TO check the configurations

```
$ git config --global --list
```

Example:



```
MINGW32:/c/Users/CCTV/csm/vaageswari

CSEHOD@CCTV-PC MINGW32 ~/csm/vaageswari (master)
$ git config --global --list
user.name=Chandramouli
user.email=cmnarsingoju@gmail.com

CSEHOD@CCTV-PC MINGW32 ~/csm/vaageswari (master)
$
```

4. git add: Adds changes to the staging area for the next commit.

Example:

```
$ git add file1.txt
```

```
$ git add .
```

5. Bring file back to untracked section

```
$ git rm --cached f1
```

or

```
$ git reset f2
```

5. git commit: Commits changes to the repository.

To move the files from staging area to LR

Example:

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

To see the list of commit

```
$ git log --oneline
```

6. git status: Shows the status of the repository, including any changes or untracked files.

Example:

```
$ git status
```

7. git clone: Clones an existing Git repository into a new directory.

Example:

```
$ git clone https://github.com/username/repo.git
```

8. git push: Pushes changes to a remote repository.

Example:

```
$ git push origin master
```

9. git pull: Fetches changes from a remote repository and merges them with the local repository.

Example:

```
$ git pull origin master
```

10. git branch: Lists all branches in the repository.

Example:

```
$ git branch * master feature-branch
```

11. git checkout: Switches to a different branch or commit.

Example:

```
$ git checkout feature-branch
```

12.git merge: Merges changes from one branch into another.

Example:

```
$ git merge feature-branch
```

13.git reset: Resets the repository to a previous commit.

Example:

```
$ git reset --hard a73f2358a1b469c689df6dfeac6a5b8a28f6c5b7
```

GitHub commands:

1.git clone: Clones a remote GitHub repository to the local machine.

Example:

```
$ git clone https://github.com/username/repo.git
```

2.git fork: Creates a copy of a GitHub repository on your account.

Example:

1. Go to the repository page on GitHub, click on the "Fork" button in the top-right corner, and select your account.
2. **git pull-request:** Submits a pull request to merge changes from your branch to the main branch.

Example:

1. Push the changes to your branch.

```
$ git push origin feature-branch
```

2. On GitHub, go to the repository page and click on "New pull request".
3. Select the base branch and the compare branch.

EXPERIMENT 3:

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

Here's an example workflow for practicing source code management on GitHub with the user registration form code from exercise 1:

1. Create a new GitHub repository by logging in to GitHub and clicking on the "New" button on the main dashboard.
2. Give the repository a name (e.g. "event-registration-form") and a short description. Choose whether the repository should be public or private, and whether to initialize the repository with a README file.
3. Once the repository is created, click on the "Code" button to get the repository's URL. You can use this URL to clone the repository to your local machine using a Git client such as Git Bash.
4. After cloning the repository, modify the code in exercise 1 to include any necessary changes for the specific event registration form. Save the changes and commit them to your local Git repository.
5. Use the Git Bash command line to push the local changes to the remote GitHub repository using the **following commands**:

git add .

git commit -m "Add changes to event registration form"

git push

6. The changes should now be visible in the remote repository on GitHub. You can use the GitHub website to view the changes, create new branches, merge branches, and perform other source code management tasks.
7. Repeat steps 4-6 as needed to continue modifying and updating the code for the event registration form.

This is just a basic example of how to use GitHub for source code management, but there are many other features and workflows available depending on the specific project requirements and team preferences.

EXPERIMENT 4:

Jenkins installation and setup, explore the environment

a brief overview of how to install and set up Jenkins, as well as some information about the environment:

1. Download the latest version of Jenkins from the official website:
<https://www.jenkins.io/download/>
2. Install Jenkins on your machine using the appropriate installation method for your operating system.
3. Once Jenkins is installed, open a web browser and navigate to <http://localhost:8080/> to access the Jenkins dashboard.
4. Follow the prompts to set up the initial admin user account and plugins for Jenkins.
5. Once the initial setup is complete, you can start creating and configuring jobs in Jenkins to automate your software build, test, and deployment processes.

Jenkins is a popular open-source automation server that can be used for continuous integration and continuous delivery (CI/CD) of software projects. It provides a wide range of plugins and integrations with other tools and technologies to streamline the software development and delivery process.

Jenkins runs on a Java-based environment and is compatible with a variety of operating systems, including Windows, Linux, and macOS. It also supports many programming languages and frameworks, making it a versatile tool for software development teams.

In addition to job configuration and management, Jenkins also provides extensive reporting and analysis features, allowing users to monitor and improve the performance of their software delivery pipeline over time.

EXPERIMENT 5:

Demonstrate continuous integration and development using Jenkins.

Here's an example of how to set up a basic continuous integration (CI) and continuous development (CD) pipeline using Jenkins:

1. Create a new Jenkins job by clicking on "New Item" on the Jenkins dashboard. Give the job a name (e.g. "my-app-build"), choose "Freestyle project", and click "OK".
2. In the job configuration page, under "Source Code Management", select the version control system you're using for your project (e.g. Git).
3. Under "Build Triggers", select "Poll SCM" and set the schedule for Jenkins to check for changes in the repository (e.g. "* * * * *", which means every minute).
4. Under "Build", add a build step to compile and package your application (e.g. "mvn clean package" for a Maven project).
5. Under "Post-build Actions", add a step to deploy the application to a test environment (e.g. "ssh user@server 'docker-compose up -d'").
6. Save the job configuration and run the job manually to test the pipeline.
7. As changes are made to the source code, Jenkins will automatically detect and trigger the job to rebuild and redeploy the application to the test environment, allowing developers to continuously test and improve their code.

Note that this is just a basic example of how to set up a continuous integration and development pipeline using Jenkins, and there are many other configuration options and plugins available to customize the pipeline for specific project requirements. Additionally, it's important to ensure that the pipeline is secure and reliable, with appropriate testing and validation procedures in place to prevent errors and vulnerabilities from being introduced into the codebase.

EXPERIMENT 6:

Explore Docker commands for content management

Docker is a powerful containerization platform that enables developers to package, deploy, and manage their applications as containers. Here are some Docker commands for content management:

1. **docker build:** This command is used to build a Docker image from a Dockerfile. A Dockerfile is a script that contains instructions for building a Docker image.
2. **docker push:** This command is used to push a Docker image to a Docker registry, such as Docker Hub or a private registry.

3. **docker pull:** This command is used to pull a Docker image from a Docker registry.
4. **docker run:** This command is used to run a Docker container from a Docker image. You can specify various options, such as port mapping, volume mapping, and environment variables.
5. **docker ps:** This command is used to list all running Docker containers.
6. **docker stop:** This command is used to stop a running Docker container.
7. **docker rm:** This command is used to remove a stopped Docker container.
8. **docker images:** This command is used to list all Docker images on your system.
9. **docker rmi:** This command is used to remove a Docker image from your system.
10. **docker-compose:** This command is used to define and run multi-container Docker applications. It allows you to define the containers, their dependencies, and the network configuration in a single YAML file.

These are just a few examples of Docker commands for content management. Docker provides many more commands and options to manage containers, images, and networks, as well as to monitor and troubleshoot your Docker environment.

EXPERIMENT 7:

Develop a simple containerized application using Docker.

here's an example of how to develop a simple containerized application using Docker:

1. **Create a new directory for your application and navigate to it.**
2. **Create a new file called "app.py" and add the following Python code:**

```
from flask import Flask

app = Flask(__name__)

@app.route("/")

def hello():

    return "Hello, world!"
```



```
if __name__ == "__main__":
```

```
    app.run(host='0.0.0.0', port=80)
```

This code defines a simple Flask application that listens on port 80 and returns the string "Hello, world!" when accessed at the root URL.

3. Create a new file called "Dockerfile" and add the following code:

```
sql
```

```
FROM python:3.9
```

```
WORKDIR /app
```

```
COPY requirements.txt .
```

```
RUN pip install --no-cache-dir -r requirements.txt
```

```
COPY . .
```

```
CMD [ "python", "app.py" ]
```

This code defines a Dockerfile that uses the official Python 3.9 image as the base image, sets the working directory to "/app", copies the "requirements.txt" file into the container and installs the dependencies, copies the rest of the application files into the container, and specifies the command to run when the container is started.

4. Create a new file called "requirements.txt" and add the following code

```
Flask==2.0.1
```

This file lists the dependencies for the application, which in this case is just Flask version 2.0.1.

5. Build the Docker image by running the following command:

```
docker build -t my-app .
```

This command builds the Docker image using the Dockerfile in the current directory, and tags it with the name "my-app".

6. Run the Docker container by running the following command:

```
docker run -p 8080:80 my-app
```

This command runs the Docker container using the "my-app" image, maps port 80 in the container to port 8080 on the host, and starts the container.

7. Access the application by opening a web browser and navigating to <http://localhost:8080>. You should see the message "Hello, world!" displayed in the browser.

That's it! You've just developed a simple containerized application using Docker. You can modify the application code, rebuild the Docker image, and run the container again to test your changes.

EXPERIMENT 8:

Integrate Kubernetes and Docker

Kubernetes is a container orchestration platform that automates the deployment, scaling, and management of containerized applications. Docker is a containerization platform that enables developers to package, deploy, and manage their applications as containers. Together, Kubernetes and Docker provide a powerful solution for developing and deploying containerized applications at scale.

Here's how to integrate Kubernetes and Docker:

1. Install Docker on your local machine or server, if it's not already installed.
2. Install Kubernetes on your local machine or server, if it's not already installed. You can use a cloud provider like AWS or Google Cloud, or you can install Kubernetes on your own servers.
3. Build your Docker image using a Dockerfile, as described in the previous exercise.
4. Push the Docker image to a container registry, such as Docker Hub or a private registry.
5. Create a Kubernetes deployment by creating a YAML file that describes the deployment. The deployment YAML file should specify the Docker image, the number of replicas, and any other configuration options.
6. Use the Kubernetes command-line tool, `kubectl`, to create the deployment:

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

This command creates a Kubernetes deployment using the configuration defined in the YAML file.

7. Use `kubectl` to check the status of the deployment:

```
kubectl get deployments
```

This command lists all Kubernetes deployments in the current namespace, along with their current status.

8. Use `kubectl` to create a Kubernetes service that exposes the deployment:

```
kubectl expose deployment my-app --type=LoadBalancer --port=80 --target-port=80
```

This command creates a Kubernetes service that exposes the deployment on port 80. The service type is `LoadBalancer`, which means that Kubernetes will automatically provision a load balancer for the service.

9. Use `kubectl` to check the status of the service:

```
kubectl get services
```

This command lists all Kubernetes services in the current namespace, along with their current status.

10. Access the application by opening a web browser and navigating to the IP address of the load balancer. You should see the application running in the browser.

That's it! You've just integrated Kubernetes and Docker to deploy and manage a containerized application at scale. You can modify the application code, rebuild the Docker image, and update the Kubernetes deployment to test your changes.

Regenerate response

EXPERIMENT 9:

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

To automate the process of running the containerized application developed in exercise 7 using Kubernetes, we can use Kubernetes YAML files to define the deployment, service, and other resources needed to run the application.

Here are the steps to automate the process:

1. Create a deployment YAML file that defines the deployment for your containerized application. The YAML file should include the following details:
 - The Docker image to be used
 - The number of replicas to be created

- The port on which the container will listen

Here is an example of a deployment YAML file:

```
apiVersion: apps/v1

kind: Deployment

metadata:
  name: my-app-deployment

spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app-container
          image: my-app-image:latest
          ports:
            - containerPort: 8080
```

2. Create a service YAML file that defines the service for your containerized application. The YAML file should include the following details:
 - The port on which the service will listen
 - The type of the service (ClusterIP, NodePort, or LoadBalancer)

- The target port on which the container is listening

Here is an example of a service YAML file:

```
apiVersion: v1

kind: Service

metadata:

  name: my-app-service

spec:

  selector:

    app: my-app

  ports:

    - name: http

      port: 80

      targetPort: 8080

  type: LoadBalancer
```

3. Apply the deployment YAML file using the kubectl apply command:

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

4. Apply the service YAML file using the kubectl apply command:

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

5. Verify that the deployment and service are running using the kubectl get command:

```
kubectl get deployments
```

```
kubectl get services
```

6. Access the application using the IP address of the load balancer.

By automating the process of running the containerized application using Kubernetes, we can easily scale the application up or down, roll out updates, and manage the application with ease.

EXPERIMENT 10:

10. Install and Explore Selenium for automated testing.

Selenium is an open-source tool used for automating web browsers. It provides a simple and effective way to automate web applications for testing purposes. Here are the steps to install and explore Selenium for automated testing:

1. Install Python on your local machine if it's not already installed. You can download and install the latest version of Python from the official website.
2. Install Selenium using pip, the Python package manager:

```
pip install selenium
```

3. Download the web driver for the browser you want to automate. Selenium WebDriver is a collection of language-specific bindings that allow you to control a browser from your code. You can download the web driver for various browsers like Chrome, Firefox, Safari, etc. from the official Selenium website.
4. Set up the Selenium environment by creating a new Python file and importing the necessary modules:

```
from selenium import webdriver
```

5. Initialize a new instance of the WebDriver:

```
driver = webdriver.Chrome()
```

This will open a new instance of the Chrome browser.

6. Navigate to a webpage using the get method:

```
driver.get("https://www.google.com")
```

This will load the Google homepage in the Chrome browser.

7. Interact with elements on the webpage using the various methods provided by the WebDriver API. For example, to search for an element on the page and click on it, you can use the following code:

```
search_box = driver.find_element_by_name("q")
```

```
search_box.send_keys("selenium")
```

```
search_box.submit()
```

This will search for the term "selenium" on Google and display the search results.

8. Close the WebDriver instance using the quit method:

```
driver.quit()
```

This will close the Chrome browser.

By using Selenium for automated testing, you can write scripts to simulate user actions on a webpage, such as filling out forms, clicking on buttons, and navigating through pages. This can help you automate repetitive and time-consuming testing tasks and improve the quality and reliability of your web applications.

EXPERIMENT 11:

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

here's an example of a simple program in JavaScript that calculates the sum of two numbers:

```
function calculateSum(a, b) {  
  return a + b;  
}
```

To perform testing using Selenium, we can write a test script that opens a webpage, enters two numbers, and verifies that the sum is calculated correctly. Here are the steps to write and run a test script using Selenium and JavaScript:

1. Set up the Selenium environment as described in the previous answer.
2. Create a new JavaScript file and import the necessary modules:

```
const webdriver = require('selenium-webdriver');  
const assert = require('assert');
```

3. Initialize a new instance of the WebDriver:

```
const driver = new webdriver.Builder().forBrowser('chrome').build();
```

4. Navigate to a webpage where you want to test your JavaScript code:

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

5. Interact with elements on the webpage using the various methods provided by the WebDriver API. For example, to enter two numbers and calculate their sum, you can use the following code:

```
const num1 = driver.findElement(webdriver.By.name('num1'));
```

```
num1.sendKeys('2');

const num2 = driver.findElement(webdriver.By.name('num2'));
num2.sendKeys('3');

const addButton = driver.findElement(webdriver.By.id('add'));
addButton.click();

const result = driver.findElement(webdriver.By.id('result'));
result.getText().then(text => assert.equal(text, '5'));
```

This will enter the numbers 2 and 3 in the respective input fields, click on the "add" button, and verify that the result is equal to 5.

6. Close the WebDriver instance using the **quit** method:
driver.quit();

This will close the Chrome browser.

By writing and running test scripts using Selenium and JavaScript, you can automate the testing of your web applications and ensure that they work as expected.

EXPERIMENT 12:

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

here are some test cases that can be developed for the containerized application developed in exercise 7 using Selenium:

1. Test that the application is accessible: Open the browser and navigate to the URL of the containerized application. Verify that the application loads without any errors.
2. Test that the login functionality works correctly: Enter valid login credentials and click on the login button. Verify that the user is redirected to the dashboard page.
3. Test that the logout functionality works correctly: Click on the logout button and verify that the user is redirected to the login page.
4. Test that new records can be added: Enter valid data in the input fields and click on the add button. Verify that the new record is added to the table.

5. Test that records can be updated: Click on the edit button for a record and update its values in the input fields. Click on the update button and verify that the record is updated in the table.
6. Test that records can be deleted: Click on the delete button for a record and verify that the record is removed from the table.
7. Test that search functionality works correctly: Enter a search term in the search input field and click on the search button. Verify that only the records matching the search term are displayed in the table.
8. Test that pagination functionality works correctly: Verify that the table displays a maximum of 10 records per page. Click on the next and previous buttons and verify that the table displays the correct records for each page.

By developing and running these test cases using Selenium, you can ensure that the containerized application works as expected and meets the requirements.