DEVOPS

DevOps, short for Development Operations, is a software development approach that combines software development (Dev) with IT operations (Ops) to improve the delivery of business value. It is a set of practices, principles, and tools used to increase an organization's ability to deliver software and services at a faster pace, with higher quality, and more reliability.

The main goal of DevOps is to break down the traditional barriers between development and operations teams by fostering collaboration, communication, and integration between them. This allows for continuous integration, continuous delivery, and continuous monitoring of software development processes, resulting in faster development cycles and more frequent releases.

DevOps has gained popularity in recent years due to its ability to address the challenges faced by organizations in the fast-paced, constantly changing world of software development. By adopting DevOps practices, organizations can increase efficiency, reduce errors, and improve the overall quality of their software products.

One of the key components of DevOps is automation. Automation allows organizations to streamline repetitive tasks, such as testing, deployment, and monitoring, which helps speed up the development process and reduce the chances of human error. By automating these processes, organizations can increase the frequency of releases and improve the reliability of their software products.

Another important aspect of DevOps is collaboration. By breaking down the silos between development and operations teams, DevOps encourages collaboration and communication between different stakeholders in the software development process. This allows for faster decision-making, better problem-solving, and improved overall efficiency.

DevOps also emphasizes the importance of continuous feedback and improvement. By continuously monitoring and analyzing the performance of software products, organizations can identify areas for improvement and make necessary adjustments in real-time. This allows for faster iteration and continuous improvement of software products.

**APPLICATIONS**

There are many applications of DevOps across various industries and sectors. Some common applications of DevOps include:

1. Continuous integration and delivery (CI/CD): DevOps allows organizations to automate the process of integrating code changes, testing them, and deploying them to production environments. This enables organizations to release software updates more frequently and with greater confidence.

2. Infrastructure as Code (IAC): DevOps promotes the use of IAC, which allows organizations to manage and provision their infrastructure through code. This makes it easier to deploy and scale applications in a more efficient and reliable manner.

3. Monitoring and logging: DevOps encourages organizations to implement robust monitoring and logging systems that provide real-time insights into the performance of their software products. This allows organizations to detect and respond to issues quickly, improving overall reliability and user experience.

In conclusion, DevOps is a software development approach that combines development and operations practices to improve the efficiency, quality, and reliability of software products. By fostering collaboration, automation, and continuous improvement, DevOps enables organizations to deliver software faster and more effectively. With its many applications and benefits, DevOps has become an essential practice for organizations looking to stay competitive in the fast-paced world of software development.

**EXPERIMENT NO :**1.Write code for a simple user registration form for an event.

**AIM**:write code for a simple user registration form for an event.

**DESCRIPTION**:

**HTML CODE:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Course Registration</title>

    <link rel="stylesheet" href="styles.css">

</head>

<body>

    <div class="container">

        <h2>Course Registration</h2>

        <form id="registrationForm" onsubmit="submitForm(event)">

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

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

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

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

            <label for="course">Select Course:</label>

            <select id="course" name="course" required>

                <option value="">Select</option>

                <option value="course1">Course 1</option>

                <option value="course2">Course 2</option>

                <option value="course3">Course 3</option>

            </select>

            <button type="submit">Submit</button>

        </form>

        <div id="thankYouMessage" class="hidden">Thank you for registering!</div>

    </div>

    <script src="script.js"></script>

</body>

</html>

**CSS CODE:**

.container {

    max-width: 400px;

    margin: 50px auto;

    padding: 20px;

    border: 1px solid #ccc;

    border-radius: 5px;

}

form {

    display: flex;

    flex-direction: column;

}

label {

    margin-bottom: 5px;

}

input, select, button {

    margin-bottom: 10px;

    padding: 8px;

    border: 1px solid #ccc;

    border-radius: 5px;

}

button {

    cursor: pointer;

}

.hidden {

    display: none;

}

**JAVA SCRIPT CODE:**

function submitForm(event) {

    event.preventDefault();

    // Get form data

    const formData = new FormData(event.target);

    const name = formData.get('name');

    const email = formData.get('email');

    const course = formData.get('course');

    // Display thank you message

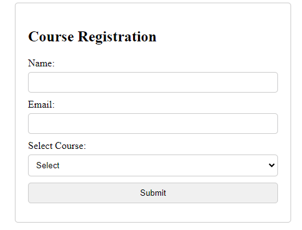
    document.getElementById('registrationForm').reset();

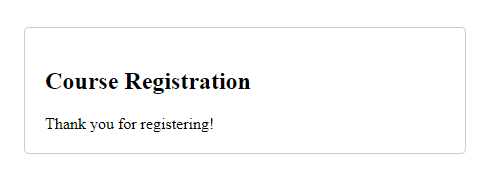
    document.getElementById('registrationForm').classList.add('hidden');

    document.getElementById('thankYouMessage').classList.remove('hidden');

}

**OUTPUT:**

****

****

**EXPERIMENT NO :** 2.Explore Git and GitHub commands.

**AIM**: Explore Git and GitHub commands.

**DESCRIPTION**:

Git and GitHub are widely used tools in software development and version control. Git is a distributed version control system that allows multiple developers to collaborate on a project simultaneously by tracking changes to files and facilitating merging of different versions. GitHub, on the other hand, is a web-based platform that serves as a central repository for Git projects, as well as providing additional collaboration features such as issue tracking and project management tools.

The main reasons for using Git and GitHub in software development are:

1. Version control: Git allows developers to keep track of changes made to code over time, making it easy to revert to previous versions if needed. This ensures that team members are always working on the latest version of the code and can easily collaborate without fear of losing their work.

2. Collaboration: GitHub provides a centralized platform for developers to share their code with team members, as well as the wider open-source community. Developers can create branches to work on different features or fixes independently, before merging them back into the main codebase. GitHub also allows for code reviews, issue tracking, and discussions, making it easier for teams to collaborate and communicate effectively.

3. Backup and redundancy: By hosting code on GitHub, developers have a backup of their code in the cloud, reducing the risk of data loss due to hardware failures or accidental deletions. GitHub also provides redundancy and disaster recovery mechanisms to ensure that code is always accessible and secure.

4. Open-source contributions: GitHub has become a hub for open-source software development, allowing developers from around the world to collaborate on projects and contribute code. By hosting projects on GitHub, developers can attract a wider community of contributors and users, leading to faster development cycles and higher-quality software.

Common applications of Git and GitHub in software development include:

1. Software development: Git and GitHub are commonly used in software development projects to manage code changes, collaborate with team members, and track project progress. Developers can create branches for new features, fix bugs, and merge changes back into the main codebase seamlessly.

2. Continuous integration and deployment: Git and GitHub can be integrated with automated build and deployment tools to streamline the software development lifecycle. By automating tasks such as testing, building, and deployment, developers can deliver high-quality software faster and more reliably.

3. Documentation and knowledge sharing: GitHub can be used to host documentation, wikis, and tutorials related to a project, making it easier for developers to learn about the codebase and contribute

effectively. Developers can also use GitHub Pages to host websites and showcase their projects to the world.

**To install Git and GitHub on your computer, follow these steps:**

1. Download Git from the official Git website (https://git-scm.com/). Choose the appropriate installer for your operating system and download it.

2. Run the installer and follow the on-screen instructions to complete the installation process.

3. Once Git is installed, open up a terminal or command prompt and enter the following commands to configure your Git username and email address:

**git config --global user.name "Your Name"**

**git config --global user.email "youremail@example.com"**

4. Next, you'll need to create a GitHub account if you don't already have one. Go to the GitHub website (https://github.com/) and sign up for a free account.

5. After creating your GitHub account, you can log in to your GitHub account from the terminal by running the following command and entering your GitHub username and password when prompted:

**git config --global user.username "Your GitHub Username"**

**git config --global user.password "Your GitHub Password"**

6. Once you're logged in, you can start using Git and GitHub to manage your projects. To create a new repository on GitHub, you can use the following command in the terminal:

**git init**

**git add .**

**git commit -m "Initial commit"**

**git remote add origin https://github.com/your-username/your-repository.git**

**git push -u origin master**

This will create a new repository on GitHub and push your files to it. You can then manage your repository through the GitHub website or continue working on your project locally with Git.

**EXPERIMENT NO :** 3.Practice Source code management on GitHub programs like fork and branch

**AIM**: Understanding small commands.

**DESCRIPTION**:

1. Fork a repository:

- Go to the repository you want to fork on GitHub

- Click on the "Fork" button in the top right corner of the page

- This will create a copy of the repository in your own GitHub account

2. Clone the forked repository to your local machine:

- Go to your forked repository on GitHub

- Click on the "Code" button and copy the HTTPS or SSH URL

- Open a terminal window on your local machine and run the following command:

**Command: git clone <repository\_url>**

3. Create a new branch:

- To create a new branch, run the following command in the terminal:

**Command: git checkout -b <branch\_name>**

4. Make changes to the source code:

- Make any necessary changes to the source code files in your local repository

5. Add and commit the changes:

- Stage the changes by running the following command:

**Command: git add .**

- Commit the changes with a descriptive message:

**Command: git commit -m "Your commit message here"**

6. Push the changes to your forked repository:

- Push the changes to your forked repository on GitHub by running the following command:

**Command: git push origin <branch\_name>**

7. Create a pull request:

- Go to your forked repository on GitHub

- Click on the "Pull request" button

- Select the branch you made changes to and the branch you want to merge into

- Add a title and description for your pull request

- Click on the "Create pull request" button

8. Merge the pull request:

- If there are no conflicts, the owner of the original repository can review and merge your pull request

By following these steps, you can practice source code management on GitHub by forking a repository, making changes, creating a new branch, and submitting a pull request.

EXPERIMENT NO: 4. Jenkins installation and setup, explore the

environment

Aim: Jenkins installation and setup, explore the environment

**EXPERIMENT NO : 4.** Jenkins installation and setup, explore the environment.

**AIM**: Jenkins installation and setup, explore the environment

**DESCRIPTION**:

Jenkins is a widely-used open-source automation server that facilitates the building, testing, and deployment of software applications. It is a continuous integration and continuous delivery (CI/CD) tool that has become crucial in modern software development practices due to its ability to automate repetitive tasks and streamline the software development process.

Jenkins allows developers to automate the process of integrating code changes from different team members into a shared repository, running tests to ensure the code is functioning correctly, and deploying the application to various environments. This automation helps reduce manual errors, accelerates the development process, and enables teams to deliver high-quality software faster.

The key features of Jenkins include:

1. Automation: Jenkins automates the entire software delivery process, from code integration to deployment, reducing manual intervention and ensuring consistency in the software build process.

2. Extensibility: Jenkins offers a wide range of plugins that enable users to customize and extend its functionality to suit their specific requirements.

3. Scalability: Jenkins can easily scale up to accommodate projects of any size, making it suitable for teams working on small projects as well as large enterprises.

4. Integration: Jenkins integrates seamlessly with other tools and technologies commonly used in the software development lifecycle, such as version control systems, build tools, and testing frameworks.

With its flexibility and ease of use, Jenkins has become a staple tool in the DevOps ecosystem, enabling organizations to achieve faster time-to-market, improved collaboration between development and operations teams, and higher quality software releases.

Some popular applications of Jenkins include:

1. Continuous Integration: Jenkins automates the process of integrating code changes made by multiple team members into a shared repository, allowing developers to quickly identify and resolve integration issues.

2. Continuous Deployment: Jenkins automates the deployment of applications to various environments, such as development, staging, and production, ensuring a consistent and reliable deployment process.

3. Automated Testing: Jenkins can be used to automate the execution of tests, including unit tests, integration tests, and end-to-end tests, helping teams identify bugs and issues early in the development cycle.

4. Infrastructure as Code: Jenkins can be integrated with tools like Terraform and Ansible to automate the provisioning and configuration of infrastructure, enabling teams to manage their infrastructure as code.

**INSTALLATION:**

Jenkins is a popular open-source tool for Continuous Integration and Continuous Deployment (CI/CD) in software development. Here are the steps to install and set up Jenkins:

Download and install Jenkins:

* Download the Jenkins package for your operating system from theJenkins website.
* Follow the installation instructions for your operating system to install Jenkins.

Start the Jenkins service:

* On Windows, use the Windows Services Manager to start the Jenkins service.
* On Linux, use the following command to start the Jenkins service: $ sudo service jenkins start

Access the Jenkins web interface:

* Open a web browser and navigate to http://localhost:8080 to access the Jenkins web interface.
* If the Jenkins service is running, you will see the Jenkins login page.

Initialize the Jenkins environment:

* Follow the instructions on the Jenkins setup wizard to initialize the Jenkins environment.
* This process involves installing recommended plugins, setting up security, and creating the first admin user.

Explore the Jenkins environment:

* Once the Jenkins environment is set up, you can explore the various features and functionalities available in the web interface.
* Jenkins has a rich user interface that provides access to features such as build history, build statistics, and system information.

These are the basic steps to install and set up Jenkins. Depending on your use case, you may need to customize your Jenkins environment further. For example, you may need to configure build agents, set up build pipelines, or integrate with other tools. However, these steps should give you a good starting point for using Jenkins for CI/CD in your software development projects.

**EXPERIMENT NO:** 5. Demonstrate continuous integration and development using Jenkins.

**Aim:** Demonstrate continuous integration and development using Jenkins.

**DESCRIPTION:**

Continuous Integration (CI) and Continuous Development (CD) are important practices in software development that can be achieved using Jenkins. Here's an example of how you can demonstrate CI/CD using Jenkins: Create a simple Java application:

Create a simple Java application that you want to integrate with Jenkins.

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

**Commit the code to a Git repository:**

Create a Git repository for the application and commit the code to the repository.

Make sure that the Git repository is accessible from the Jenkins server.

**Create a Jenkins job:**

Log in to the Jenkins web interface and create a new job.

• Configure the job to build the Java application from the Git repository.

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

**Build the application**:

* Trigger a build of the application using the Jenkins job.
* The build should compile the code, run any tests, and produce an executable jar file.

**Monitor the build:**

Monitor the build progress in the Jenkins web interface.

The build should show the build log, test results, and the status of the build.

Deploy the application:

* If the build is successful, configure the Jenkins job to deploy the application to a production environment.
* The deployment could be as simple as copying the jar file to a production server or using a more sophisticated deployment process, such as using a containerization technology like Docker.

**Repeat the process**:

* Repeat the process for subsequent changes to the application.
* Jenkins should automatically build and deploy the changes to the production environment.

This is a basic example of how you can use Jenkins to demonstrate CI/CD in software development. In a real-world scenario, you would likely have more complex requirements, such as multiple environments, different types of tests, and a more sophisticated deployment process. However, this example should give you a good starting point for using Jenkins for CI/CD in your software development projects.

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

**AIM:** Explore Docker commands for content management.

**DESCRIPTION**

Docker is a containerization technology that is widely used for managing application containers. Here are some commonly used Docker commands for content management:

* **Docker run: Run a command in a new container.**

For example: $ 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: $ docker start mycontainer This command starts the container named "mycontainer"

* **Docker stop: Stop one or more running containers.**

For example: $ docker stop mycontainer This command stops the container named "mycontainer".

* **Docker rm: Remove one or more containers.**

For example: $ docker rm mycontainer This command removes the container named "mycontainer".

* **Docker ps: List containers.**

For example: $ docker ps This command lists all running containers.

* **Docker images: List images.**

For example: $ docker images This command lists all images stored locally on the host.

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

For example: $ docker pull ubuntu:16.04 This command pulls the Ubuntu 16.04 image from the Docker Hub registry.

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

For example: $ docker push myimage This command pushes the image named "myimage" to the Docker Hub registry

**EXPERIMENT NO:** 7. Develop a simple containerized application using Docker

**AIM**: Develop a simple containerized application using Docker

**DESCRIPTION:**

Here's an example of how you can develop a simple containerized application using Docker:

**Choose an application:**

* Choose a simple application that you want to containerize. For example, a Python script that prints "Hello World".

**Write a Dockerfile:**

* Create a file named "Dockerfile" in the same directory as the application.

In the Dockerfile, specify the base image, copy the application into the container, and specify the command to run the application. Here's an example Dockerfile for a Python script:

# Use the official Python image as the base image

FROM python:3.9

# Copy the Python script into the container

COPY hello.py /app/

# Set the working directory to /app/

WORKDIR /app/

# Run the Python script when the container starts

CMD ["python", "hello.py"]

* **Build the Docker image:**

Run the following command to build the Docker image:

$ docker build -t myimage .

This command builds a new Docker image using the Dockerfile and tags the image with the name "myimage"

* **Run the Docker container:**

Run the following command to start a new container based on the image:

$ docker run --name mycontainer myimage

This command starts a new container named "mycontainer" based on the "myimage" image and runs the Python script inside the container.

* **Verify the output:**

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

$ docker logs mycontainer

This command displays the logs of the container and should show the "Hello World" output.

This is a simple example of how you can use Docker to containerize an application. In a real-world scenario, you would likely have more complex requirements, such as running multiple containers, managing network connections, and persisting data. However, this example should give you a good starting point for using Docker to containerize your applications.

**EXPERIMENT NO: 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 a 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 NO** : 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, you can follow these steps:

* Create a Kubernetes cluster:

Create a Kubernetes cluster using a cloud provider, such as Google Cloud or Amazon Web Services, or 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-deployment**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**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 commandline 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 NO: 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("Title: " + driver.getTitle());

driver.quit();

}

}

* **Run the test:**

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

$ javac Main.java

$ java Main

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

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

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

**PROCEDURE:**

**Step 1: Install Node.js**

**You can download and install Node.js from its official website: https://nodejs.org/en/download/**

**Step 2: Open Command Prompt and install Selenium WebDriver**

**Open Command Prompt and run the following command to install Selenium WebDriver using npm (Node Package Manager):**

**COMMAND: npm install selenium-webdriver**

**Step 3: Create a directory**

**Open Command Prompt and navigate to the location where you want to create the directory. Then, use the following command to create a directory:**

**COMMAND: mkdir <directory\_name>**

**Replace <directory\_name> with the desired name of your directory.**

**For example, if you want to create a directory named "selenium\_project", you would type:**

**COMMAND: mkdir selenium\_project**

**Step 4: Create a file named test.js**

**Once you've created the directory, navigate into it using the cd command. Then, use the following command to create a file named test.js:**

**COMMAND: type nul > test.js**

**This command creates an empty file named test.js in the current directory.**

**Now you can open the test.js file using a text editor and write your Selenium automation script in JavaScript, as described in the previous message.**

**Step 4: Open the test.js file in a text editor and write your JavaScript code**

**Open the test.js file using a text editor (such as Notepad, Sublime Text, or Visual Studio Code), and write your Selenium automation script in JavaScript.**

**CODE:**

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

**async function exampleTest() {**

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

**try {**

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

**let title = await driver.getTitle();**

**} finally {**

**}**

**}**

**exampleTest();**

**Step 5: Run the script in Command Prompt**

**Navigate to the directory where your test.js file is located using Command Prompt, then run the script by typing:**

**COMMAND: node test.js**

**EXPERIMENT NO:** 12.Develop test cases for the above containerized application using selenium

**AIM:** Develop test cases for the above containerized application using selenium

**PROGRAM:**

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

**async function runTests() {**

**try {**

**// Test Case 1: Open Google**

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

**console.log('Test Case 1: Google homepage opened successfully');**

**// Test Case 2: Search for "Selenium WebDriver"**

**let searchBox = await driver.findElement(By.name('q'));**

**await searchBox.sendKeys('Selenium WebDriver', Key.RETURN);**

**console.log('Test Case 2: Search performed for "Selenium WebDriver"');**

**// Test Case 3: Click on Images**

**let imagesLink = await driver.findElement(By.linkText('Images'));**

**await imagesLink.click();**

**console.log('Test Case 3: Clicked on Images');**

**// Test Case 4: Click on News**

**let newsLink = await driver.findElement(By.linkText('News'));**

**await newsLink.click();**

**console.log('Test Case 4: Clicked on News');**

**// Test Case 5: Navigate back to the previous page (Google)**

**await driver.navigate().back();**

**console.log('Test Case 5: Navigated back to Google');**

**} finally {**

**// Close the WebDriver session after all tests have been executed**

**await driver.quit();**

**}**

**}**

**async function runTests() {**

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

**try {**

**// Test Case 1: Open Google**

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

**console.log('Test Case 1: Google homepage opened successfully');**

**// Test Case 2: Search for "Selenium WebDriver"**

**let searchBox = await driver.findElement(By.name('q'));**

**await searchBox.sendKeys('Selenium WebDriver', Key.RETURN);**

**console.log('Test Case 2: Search performed for "Selenium WebDriver"');**

**// Test Case 3: Click on Images**

**let imagesLink = await driver.findElement(By.linkText('Images'));**

**await imagesLink.click();**

**console.log('Test Case 3: Clicked on Images');**

**// Test Case 4: Click on News**

**let newsLink = await driver.findElement(By.linkText('News'));**

**await newsLink.click();**

**console.log('Test Case 4: Clicked on News');**

**// Test Case 5: Navigate back to the previous page (Google)**

**await driver.navigate().back();**

**console.log('Test Case 5: Navigated back to Google');**

**} finally {**

**// Close the WebDriver session after all tests have been executed**

**await driver.quit();**

**}**

**}**

**runTests():**