DEVOPS LAB

Class: CSM 3-2

Experiment 1:

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

Execution steps:

- Write all the three codes (pgm1.html, pgm1.js and pgm1.css) in three different notepads and save the files with same name.
- All the files should be kept in same directory.
- For output; go to folder and open the pgm1.html file.
- It opens in a browser.

Pgm1.html

```
<!DOCTYPE html>

<html>

  <head>

    <title>Sign up to {this}</title>

    <link rel="stylesheet" type="text/css" href="pgm1.css" />

    <script type="text/javascript" src="pgm1.js"></script>

  </head>

  <body>

    <form method="post" target="#" onsubmit="return validate()">

    <h1>Sign up to {this}</h1>

    <p>To be able to use all features of {this} you will require an account. The form below will allow
you to create a new account if you don't have one yet.</p>

        Name<br>

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

    Email<br>
```

```
        <input type="text" name="email"><br>

    Retype email<br>

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

    Create a password<br>

        <input type="password" name="password"><br>

    Retype password<br>

        <input type="password" name="retype_password"><br>

        <label>

     <input type="checkbox" name="terms">By signing up to {this} you agree to your

      <a href="#service">Terms of Service</a> and <a href="#policy">Privacy Policy</a>

    </label>

   <div class="button">

     <input type="submit" class="button" value="Set Up Account">

    </div>

        </form>

</body>

</html>
```

Pgm1.js

```
function validate() {
            var result = "";
            result += validateName();
            result += validateEmail();
            result += validatePassword();
            result += validateTerms();
            if(result != "")
            {
            alert("Validation Result:\n\n" + result);
            }
            else
            {
            alert("Congrats!!! Your account Created.");
            }
    }

function validateName() {
            var name = document.getElementsByName("name")[0].value;
            if (name.length <= 2)
                        return "Name should be at least three characters.\n";
            return "";
}

function validatePassword() {
            var password = valueOf("password");
            var retype = valueOf("retype_password");
```

```
        if (password.length <= 5)

                return "Password should be at least 6 characters.\n";

        if (password != retype)

                return "Passwords do not match.\n";

        return "";

}


function validateEmail() {

        var email = valueOf("email");

        var retype = valueOf("retype_email");

        if (email.indexOf('@') == -1)

                return "Email should be a valid address.\n";

        if (email != retype)

                return "Email addresses do not match.\n";

        return "";

}


function validateTerms() {

        var terms = document.getElementsByName("terms")[0];

        if (!terms.checked)

                return "Please accept the Terms of Service and Privacy Policy";

        return "";

}

function valueOf(name) {

        return document.getElementsByName(name)[0].value;

}
```

<u>Pgm1.css</u>

```
body {
  font-family: Verdana, sans-serif;
}

input {
        border-radius: 5px;
        padding: 5px;
}

div.button {
        text-align: right;
}

a {
        text-decoration: none;
        color: #e51;
}

form {
        width: 480px;
        margin: auto;
        padding: 5px;
        border: solid black 1px;
        border-radius: 5px;
        box-shadow: 5px 5px 2px #888;
}
```

DEVOPS LAB

Class: CSM 3-2

Experiment 2 : Explore Git and GitHub commands.

Git: Git is a version-control system for tracking changes in source code during software development.

WHat is Bash? : Bash(also know as "Bourne Again SHell) is an implmentation of Shell and allows you to efficiently perform many tasks.

Features:

- easily navigate your computer to manage files and folders
- run programs that provide more functionality at the command line (e.g. git).
- launch programs from specific directories on your computer
- use repeatable commands for these tasks across many different operating systems (WIndows, Mac, Linux).

There are many Version Control Sysytem Tools amongst all Git is popular.

other tools like:SVN, CVS, mercurial.

Install Git using below link

https://git-scm.com/downloads

Choose Windows

Download and Install

Goto Start button and select Git Bash

Console window will be opened

Try the below Commands:

```
$ Git --version

$ mkdir <directory name>

$ CD <directory name>

$ dir     etc
```

GitHub

SIgnup with Github using below link

https://github.com/

EXPERIMENT NO: 3. Practice Source code management on GitHub. Experiment with the source code written in exercise 1

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

Description:

To practice source code management on GitHub, you can follow these steps:

- Create a GitHub account if you don't already have one.
- Create a new repository on GitHub.
- Clone the repository to your local machine: $ git clone <repository-url>
- Move to the repository directory: $ cd <repository-name>
- Create a new file in the repository and add the source code written in exercise 1.
- Stage the changes: $ git add <file-name>
- Commit the changes: $ git commit -m "Added source code for a simple user registration form"
- Push the changes to the remote repository: $ git push origin master
- Verify that the changes are reflected in the repository on GitHub.

These steps demonstrate how to use GitHub for source code management. You can use the same steps to manage any source code projects on GitHub. Additionally, you can also explore GitHub features such as pull requests, code review, and branch management to enhance your source code management workflow.

STEPS TO BE FOLLOWED TO INSTALL 'JENKINS' ON WINDOWS

STEP-1:Install Java Development Kit (JDK):

Make sure that there is a 'java jdk 17' software version has already installed in the system,
if not click the link to install
"https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe"

STEP-2: Set the Path for the Environmental Variable for JDK:

STEP-3: Install jenkins using the link
"https://www.jenkins.io/download/thank-you-downloading-windows-installer-stable"

STEP 4: In the Jenkin Setup screen, click Next.

STEP-5: Choose the location where you want to have the Jenkins instance installed (default location is C:\Program Files (x86)\Jenkins),
then click on Next button.

STEP-6: For Service logon credentials, choose Logon type as "Run Service as LocalSystem".

STEP-7: Choose the port available on the system and test it (Recommended is 8080).

STEP-8: Choose the JDK path and click Next.

STEP-9: Click on the Install button.

STEP-10: Once install is complete, click Finish.

STEP-11: After completing the Jenkins installation process ,Open the web Browser(Chrome) and type 'http://localhost:8080'

STEP-12: To Unlock Jenkins ,administration password has to be entered and you can get this from the link present just above in the form like
'C:\Program Files(x86)\Jenkins\secrets\initialAdminPassword'(Not recommended to copy this).

NOTE:Recommended to copy the link that is present on the screen.

STEP-13: Open the highlighted file using Notepad and copy the content of the initialAdminPassword file.

STEP-14: Paste the password it into browser’s pop-up tab
(http://localhost:8080/login?form=%2F) and click on Continue button.

STEP-15: Click on the “Install suggested plugins button” so Jenkins will retrieve and install the essential plugins

STEP-16: After all suggested plugins were installed, the “Create First Admin User” panel will show up.
Fill all the fields with desired account details and hit the “Save and Finish” button.

NOTE:Remember the Username and Password to login further.

We have successfully installed a new Jenkins Server. Hit the “Start using Jenkins” button.

Login with the username and password to get started with jenkins.

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 images: List images.

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

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

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

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

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

These are some of the basic Docker commands for managing containers and images. There are many other Docker commands and options that you can use for more advanced use cases, such as managing networks, volumes, and configuration. However, these commands should give you a good starting point for using Docker for content management.

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
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 command-line tool. For example:

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

- Verify the deployment:

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

```
$ kubectl get pods
$ kubectl get services
```

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

EXPERIMENT 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(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

PROGRAM:

- Simple JavaScript program that you can test using Selenium

```
<!DOCTYPE html>
<html>
<head>
 <title>Simple JavaScript Program</title>
</head>
<body>
 <p id="output">0</p>
 <button id="increment-button">Increment</button>
 <script>
  const output = document.getElementById("output");
  const incrementButton = document.getElementById("increment-button");

  let count = 0;
  incrementButton.addEventListener("click", function() {
   count += 1;
   output.innerHTML = count;
  });
 </script>
</body>
</html>
```

- Write a test case for this program using Selenium

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class Main {
 private WebDriver driver;
 @Before
 public void setUp() {
  System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
  driver = new ChromeDriver();
 }

 @Test
 public void testIncrementButton() {
  driver.get("file:///path/to/program.html");
  driver.findElement(By.id("increment-button")).click();
  String result = driver.findElement(By.id("output")).getText();
  assert result.equals("1");
 }
```

```
 @After
 public void tearDown() {
  driver.quit();
 }
}
```

You can run the test case using the following command:

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

The output of the test case should be:

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

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

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:

Here is an example of how you could write test cases for the containerized application using Selenium

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class Main {
 private WebDriver driver;

 @Before
 public void setUp() {
  System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
  driver = new ChromeDriver();

 }

 @Test
 public void testHomePageLoads() {
  driver.get("http://localhost:8080");
  String title = driver.getTitle();
  assert title.equals("My Containerized Application");
 }

 @Test
 public void testSubmitForm() {
  driver.get("http://localhost:8080");
  driver.findElement(By.name("name")).sendKeys("John Doe");
    driver.findElement(By.name("email")).sendKeys("john.doe@example.co
m");
  driver.findElement(By.name("submit")).click();
  String result = driver.findElement(By.id("result")).getText();
  assert result.equals("Form submitted successfully!");
 }

 @After
 public void tearDown() {
  driver.quit();
 }
}
```

You can run the test cases using the following command:

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

The output of the test cases should be:

```
..

Time: 1.135

OK (2 tests)
```

This output indicates that both test cases passed, and the containerized application is functioning as expected.