DEVOPS LAB MANUAL (III B. TECH II SEM)

Experiment No:1. Write a code for a simple user registration form for an event.

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

DESCRIPTION: Here's an example of a simple user registration form using Flask and Docker in DevOps:

- Create a Docker file with the following content to create a Docker image for your Flask application:

-
  ```
  FROM Python: 3.8
   WORKDIR/app
   COPY..
   RUN pip install--no-cache-dir-rrequirement.txt EXPOSE
   5000
   CMD ["Python", "app.py"]
  ```

- Create a requirements.txt with the following content to list the dependencies of your Flask application: Flask==1.1.2

- Create a app.py file with the following code for a simple user registration form in Flask:

```
 from flask import Flask, request, render_template
app=Flask(__name__)
@app.route('/register',methods=['GET','POST'])
def register():
if request.method== 'POST':
   name = request.form['name']
   email=request.form['email']
   password=request.form['password']
   #store the user data in database or file
   return render_template('success.html')
   return render_template('register.html')
if name_____=='main':
   app.run(host='0.0.0.0')
```

- Create a templates folder and add the following two files: register.html and success.html.

register.html

```
<form method =" post">
   <input type="text" name="name" placeholder= "n=Name">
   <input type="email" name="email" placeholder="Email">
   <input type="password" name="password" placeholder="Password">
   <input type="submit" value="Submit">
</form>
```

Success.html

```
<h2>Registration Successful</h2>
```

- Build a docker image for your flask application using the following command:
  Docker build –t simple-flask-app.
- Run a docker container from the image using the following command:
  Docker run –p 5000:5000 simple-flask-app.
- Open the web browser access registration format **http://localhost:5000/register**

This example demonstrates how to build a simple user registration form in Flask and run it in a Docker container in DevOps. A note that this code is only meant to demonstrate the basic structure of user registration form and does not include any security measure or proper error handling. It is highly recommended to add security measures or proper error handling. It is highly recommended to add security measure such as password hashing and validation before using it in a production environment.

VIVA QUESTIONS

1. **Define Flask in DevOps**

Experiment no: 2.Explore Git and GitHub commands

Aim: Explore Git and Git Hub commands

Description: Git and Git Hub are two of the most popular tools used for version control and collaboration in software development.

Here are some common **Git and GitHub commands:**

- Initializing a Git repository: **$git init**
- Checking the status of the repository: **$git status**
- Adding file to the stage: **$git add <file-name>**
- Committing changes: **$git commit –m "commit message"**
- Checking the commit history: **$git log**
- Undoing changes: **$git checkout <file-name>**
- Creating a new branch: **$git branch <branch-name>**
- Switching to different branch: **$git checkout <branch-name>**
- Merging two branches: **$git merge <branch-name>**
- Pushing changes to a remote repository: **$git push origin <branch-name>**
- Cloning a repository from GitHub: **$git clone <repository-url>**
- Creating a pull request on GitHub: Go to the repository on GitHub, select the branch you want to Merge and click the "New pull request" button.

These are just a few of the many Git and GitHub commands available. There are many other Git commands and functionalities that you can explore to suit your needs.

VIVA QUESTIONS

1. **What is GitHub**
2. **Difference between Git and GitHub**

Experiment no3. 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 this step:

- 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 thesourcecodewritteninexercise1.
- 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 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.

VIVA QUESTIONS

1. **What is GitHub management**

s

Experiment no: 4. Jenkins installation and setup, explore the environment.

Aim: Jenkins installation and setup, explore the environment.

Description: 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 setup Jenkins:

Download and Install Jenkins:

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

Start the Jenkins service:

- On windows, use the Window Services Manager to start a Jenkins service.
- On Linux, use the following commands 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 Jenkins login page.

Initialize the Jenkins environment:

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

Explore the Jenkins environment:

- Once the Jenkins environment setup, you can explore the various feature and functionalities available in the web interface.
- Jenkins as 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 setup 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.

VIVA QUESTIONS

1. **Define 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 practice in software development that can be achieved using Jenkins. Here's the example of how you can demonstrate CI/CD using Jenkins:

- 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 an application and commit the code to repository.
- Make sure that the Git repository is accessible from the Jenkins server.

Create a Jenkins jobs:

- Login 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 repository.

Build the application:

- Trigger the build application using Jenkins job.
- The build should compile the code, run any test, 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 result, 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.
- Jenkin should automatically build and deploy the changes to the production environment.

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

VIVA QUESTIONS

1. **Define CD & CI**

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 ubuntu16.04 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 a container named "mycontainer".

- Docker stop: stop one or more running containers.
  For example: **$ docker stop mycontainer**.
  This command stops a container named "mycontainer"

- Docker rm: Remove one or more containers.
  For example: **$ docker rm mycontainer**
  This command removes a 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 ubuntu16.04 image from the Docker Hub registry.

- Docker push: push an image or a repository from 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 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 a good starting point for using Docker for content management.

VIVA QUESTIONS

1. **Give briefly about Docker commands**

Experiment no: 7. Develop a simple containerized application using

Docker Aim: Develop a simple containerized application using Docker

Description: Here 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 Docker file: Create a file named "Docker file" in the same directory as the application.

In the docker file, specify the base image, copy the application in to the container, and specify the command to run the application. Here's an example 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.**

  The command builds a new Docker image using the Docker file and tag 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 by named "mycontainer" based on the "myimage" image and run 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 the simple example of how you can use Docker to containerize an application. In the 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.

VIVA QUESTIONS: 1. Name the application using Docker

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 abstraction 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 the 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 Docker file 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 expose the deployment to the network.

- **Monitor and manage the containers:** Use Kubernetes to monitor and manage the containers. This includes the scaling the number of replicas, update 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 Kubernetes cluster. This makes it easier to make updates to the application and ensure that the latest version is always running in the cluster. By integrating Kubernetes and docker, you can leverage the strength of both technologies to manage container in a scalable, reliable and efficient manner.

VIVAQUESTIONS

1. **What is docker?**

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 exercise7 using Kubernetes, you can follow these steps:

- Create a Kubernetes cluster: Create a Kubernetes cluster that cloud provided such as google cloud or amazon web services, or using local installation of Minikube.
- Push the Docker image to 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:
container:
-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: Applying the deployment and service to the cluster using the kubect1 command- line tool. For example:

```
$kubect1apply-fdeployment.yaml
$kubect1apply-fservice.yaml
```

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

```
$kubect1getpods
$kubect1getservices
```

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

VIVAQUESTIONS

1. **Define Kubernetes**

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 the steps:

Install java development kit (JDK):

- Selenium is written in Java, so you will need to install JDK in order to run it. You can download and install JDK form the official Oracle website
- Install the Selenium WebDriver:
- You can download the latest version of Selenium Web Driver from the selenium website. You will 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 will need an IDE. Some popular choices include Eclipse, IntelliJ IDEA, and Visual Studio Code.
- Write a simple test:
- Once you have an IDE setup, you can write a simple test using the Selenium Web Driver. 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 [] arg) {
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
$javacMain
```

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.

VIVA QUESTIONS

1. **What is automation testing and list the automation testing**

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 Java Script program that you can test using Selenium

```
<!DOCTYPE html>
<html>
<head>
<title> Simple Java Script Program </title>
</head>
<body>
<pid=”output”>0</p>
<button id=”increment-button”>Increment</button>
<script>
const output=document.getElementById(“output”);
const incrementButton=document.getElemetById(“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 tear Down()
{ driver.quit();      }
}
```

You can run the test case using the following commands:

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

The output of the test case should be:

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

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

VIVAQUESTIONS

1. **How java can be useful by using selenium**

Experiment: 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 testHomePageLoad()
{
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("JohnDeo");
driver.findElement(By.name("email")).sendkeys("john.deo@example.com");
driver.findElement(By.name("submit")).click();
String result=driver.findElement(By.id("result")).getText();
assert result.equals("Formsubmitted successfully!");
}

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

You can run the test cases using the following command:

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

The output of test cases should be:

```
..

Time: 1.135
OK (2test)
```

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

VIVA QUESTIONS

1. **Define selenium**
2. **Name the test case used in selenium**