

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 seleniu

EXPERIMENT-1

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

SOFTWARE REQUIREMENT: Html Compiler

Html>

<head>

<title>

Registration Page

</title>

</head>

<body bgcolor="Lightskyblue">

<form>

<label> Firstname </label>

<input type="text" name="firstname" size="15"/>

<label> Middlename: </label>

<input type="text" name="middlename" size="15"/>

<label> Lastname: </label>

<input type="text" name="lastname" size="15"/>

<label>

Course :

</label>

<select>

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

<option value="BCA">BCA</option>

<option value="BBA">BBA</option>

<option value="B.Tech">B.Tech</option>

```
<option value="MBA">MBA</option>
<option value="MCA">MCA</option>
<option value="M.Tech">M.Tech</option>
</select>

<br>

<br>

<label>
```

Gender :

```
</label><br>
<input type="radio" name="male"/> Male <br>
<input type="radio" name="female"/> Female <br>
<input type="radio" name="other"/> Other
```

```
<br>
```

```
<br>
```

```
<label>
```

Phone :

```
</label>
```

```
<input type="text" name="country code" value="+91" size="2"/>
```

```
<input type="text" name="phone" size="10"/> <br> <br>
```

Address

```
<br>
```

```
<textarea cols="80" rows="5" value="address">
```

```
</textarea>
```

```
<br> <br>
```

Email:

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

```
<br> <br>
```

Password:

```
<input type="Password" id="pass" name="pass"> <br>
```

```
<br> <br>
```

Re-type password:

```
<input type="Password" id="repass" name="repass"> <br> <br>
```

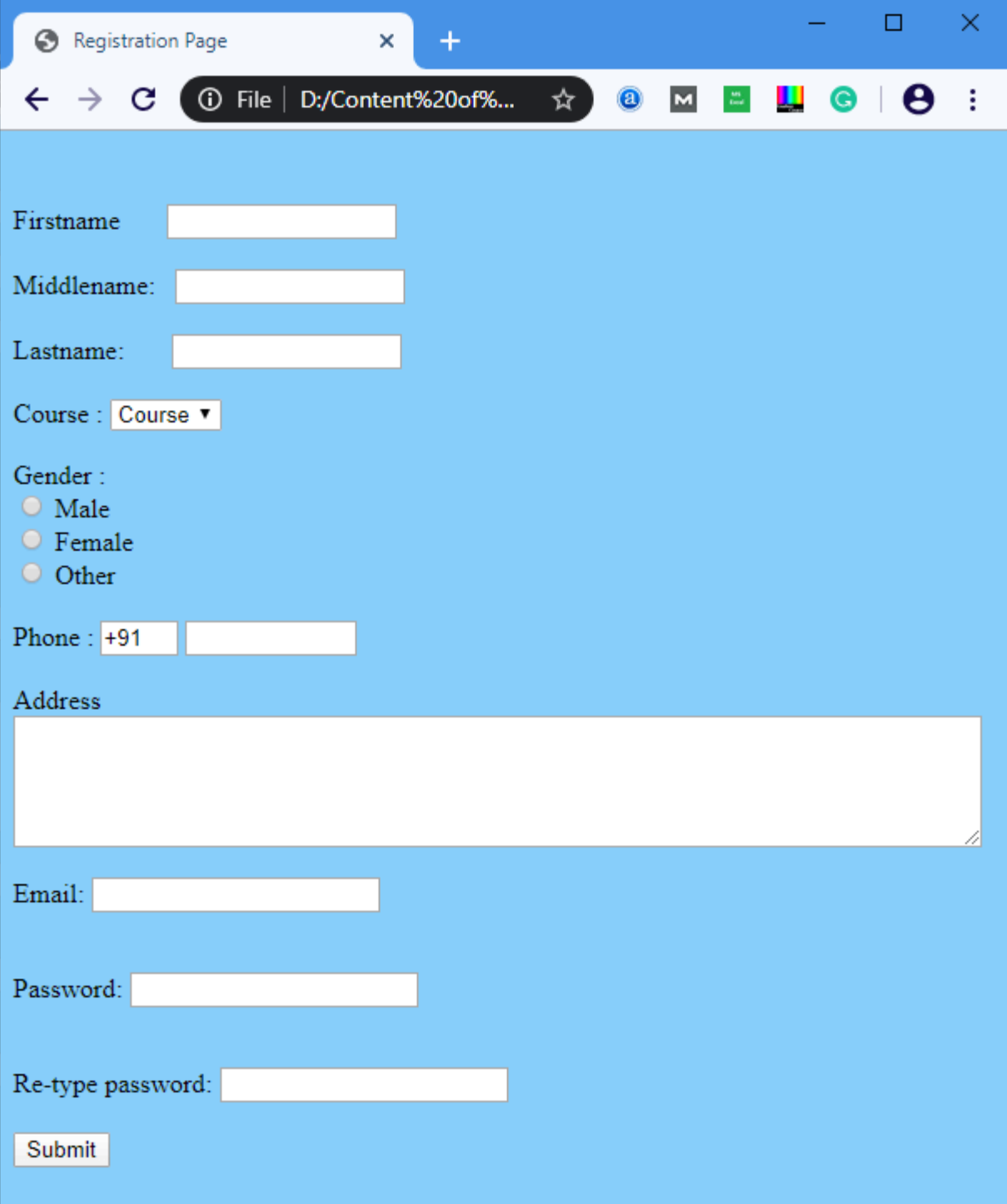
```
<input type="button" value="Submit"/>
```

```
</form>
```

```
</body>
```

```
</html>
```



OUTPUT

The screenshot shows a web browser window with a single tab titled "Registration Page". The address bar displays "File | D:/Content%20of%..." and includes navigation icons (back, forward, refresh) and a star icon for bookmarks. The page content is a registration form on a light blue background. The form includes the following fields and controls:

- Firstname:** A text input field.
- Middlename:** A text input field.
- Lastname:** A text input field.
- Course:** A dropdown menu currently showing "Course ▾".
- Gender:** Three radio button options: "Male", "Female", and "Other".
- Phone:** A text input field with "+91" pre-filled, followed by another empty text input field.
- Address:** A large, empty text area.
- Email:** A text input field.
- Password:** A text input field.
- Re-type password:** A text input field.
- Submit:** A button at the bottom left of the form.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta name="viewport" content="width=device-width, initial-scale=1">
5. <style>
6. body{
7.   font-family: Calibri, Helvetica, sans-serif;
8.   background-color: pink;
9. }
10. .container {
11.   padding: 50px;
12.   background-color: lightblue;
13. }
14.
15. input[type=text], input[type=password], textarea {
16.   width: 100%;
17.   padding: 15px;
18.   margin: 5px 0 22px 0;
19.   display: inline-block;
20.   border: none;
21.   background: #f1f1f1;
22. }
23. input[type=text]:focus, input[type=password]:focus {
24.   background-color: orange;
25.   outline: none;
26. }
27. div {
28.   padding: 10px 0;
29. }
30. hr {
31.   border: 1px solid #f1f1f1;
32.   margin-bottom: 25px;
```

```
33. }
34. .registerbtn {
35.   background-color: #4CAF50;
36.   color: white;
37.   padding: 16px 20px;
38.   margin: 8px 0;
39.   border: none;
40.   cursor: pointer;
41.   width: 100%;
42.   opacity: 0.9;
43. }
44. .registerbtn:hover {
45.   opacity: 1;
46. }
47. </style>
48. </head>
49. <body>
50. <form>
51.   <div class="container">
52.     <center> <h1> Student Registration Form</h1> </center>
53.     <hr>
54.     <label> Firstname </label>
55.     <input type="text" name="firstname" placeholder= "Firstname" size="15" required />
56.     <label> Middlename: </label>
57.     <input type="text" name="middlename" placeholder="Middlename" size="15" required />
58.     <label> Lastname: </label>
59.     <input type="text" name="lastname" placeholder="Lastname" size="15"required />
60.   <div>
61.     <label>
62.     Course :
63.   </label>
64.
```



```

65. <select>
66. <option value="Course">Course</option>
67. <option value="BCA">BCA</option>
68. <option value="BBA">BBA</option>
69. <option value="B.Tech">B.Tech</option>
70. <option value="MBA">MBA</option>
71. <option value="MCA">MCA</option>
72. <option value="M.Tech">M.Tech</option>
73. </select>
74. </div>
75. <div>
76. <label>
77. Gender :
78. </label><br>
79. <input type="radio" value="Male" name="gender" checked > Male
80. <input type="radio" value="Female" name="gender"> Female
81. <input type="radio" value="Other" name="gender"> Other
82.
83. </div>
84. <label>
85. Phone :
86. </label>
87. <input type="text" name="country code" placeholder="Country Code" value="+91" size="2"
    "/>
88. <input type="text" name="phone" placeholder="phone no." size="10"/ required>
89. Current Address :
90. <textarea cols="80" rows="5" placeholder="Current Address" value="address" required>
91. </textarea>
92. <label for="email"><b>Email</b></label>
    <input type="text" placeholder="Enter Email" name="email" required>
93.
94. <label for="psw"><b>Password</b></label>

```



```
<input type="password" placeholder="Enter Password" name="psw" required>
```

95.

96.

```
<label for="psw-repeat"><b>Re-type Password</b></label>
```

97.

```
<input type="password" placeholder="Retype Password" name="psw-repeat" required>
```

98.

```
<button type="submit" class="registerbtn">Register</button>
```

99.

```
</form>
```

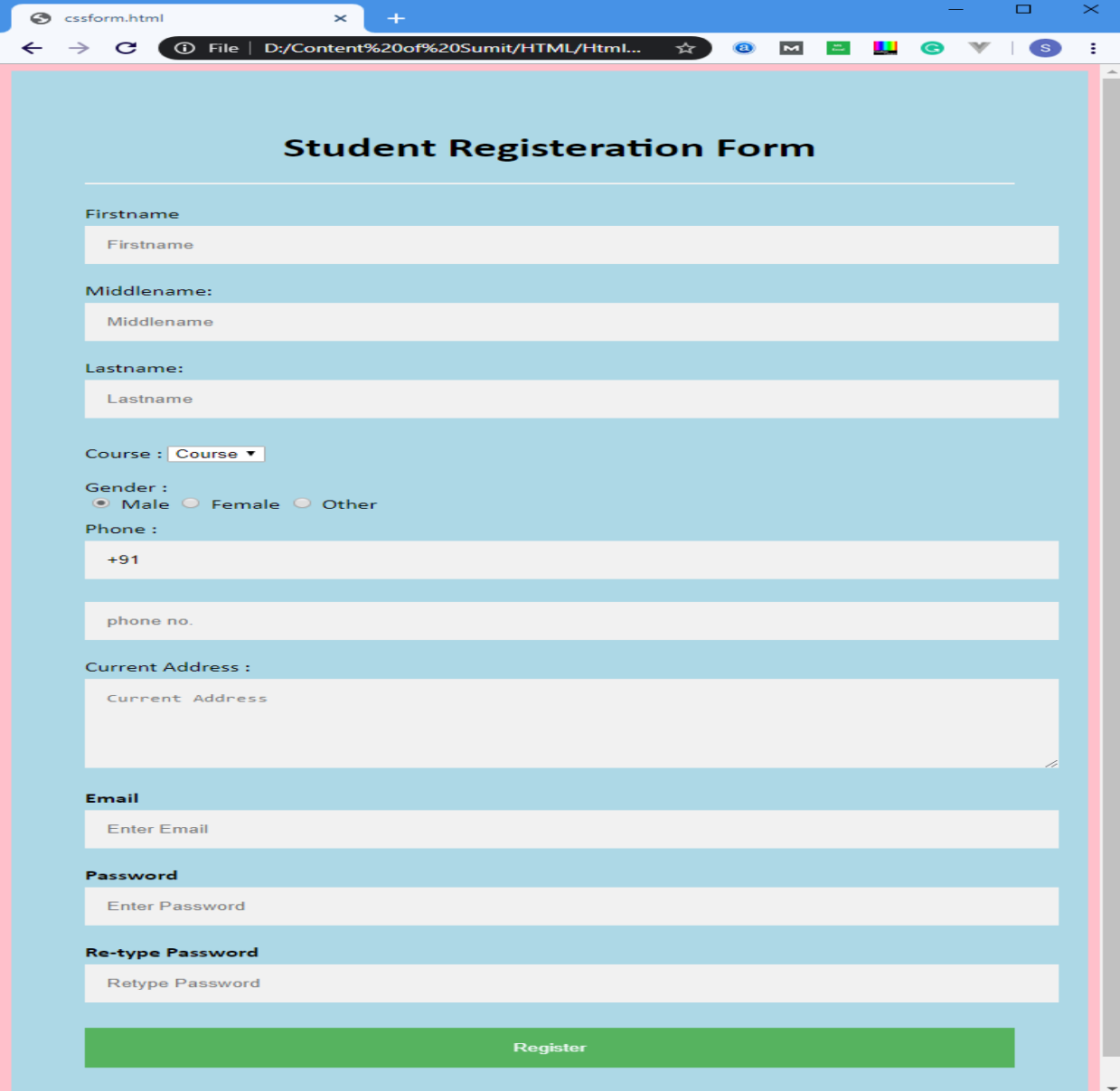
100.

```
</body>
```

101.

```
</html>
```

OUTPUT



The screenshot shows a web browser window with the title 'cssform.html'. The address bar shows the file path 'D:/Content%20of%20Sumit/HTML/Html...'. The form is titled 'Student Registration Form' and contains the following fields and elements:

- Firstname:** A text input field with the placeholder 'Firstname'.
- Middlename:** A text input field with the placeholder 'Middlename'.
- Lastname:** A text input field with the placeholder 'Lastname'.
- Course :** A dropdown menu with the selected option 'Course'.
- Gender :** Three radio buttons labeled 'Male', 'Female', and 'Other'.
- Phone :** A text input field with the placeholder '+91'.
- phone no.:** A text input field with the placeholder 'phone no.'.
- Current Address :** A text input field with the placeholder 'Current Address'.
- Email:** A text input field with the placeholder 'Enter Email'.
- Password:** A text input field with the placeholder 'Enter Password'.
- Re-type Password:** A text input field with the placeholder 'Retype Password'.
- Register:** A green button with the text 'Register'.

EXPERIMENT-2

AIM : Explore Git and GitHub commands.

SOFTWARE REQUIREMENT:

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 are sometimes known as SCM (Source Code Management) tool.

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.

What is gitbash ?

How to configure username and email for git

```
$ git init
```

```
$ git config --global user.name "sunilkumark11"
```

```
$ git config --global user.email "sunilkumark11@gmail.com"
```

TO check the configurations

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

working directory -----> staging area -----> LR

untracked files

staged files

committed files

```
+++++
```

How to make working directory as git repository

```
$ git init
```

-Observation -

```
+++++
```

```
$ git status
```

```
+++++
```

To move file to staging area

```

$ git add f1
$ git add f2 f3
$ git add .
+++++
Bring file back to untracked section
$ git rm --cached f1
or
$ git reset f2
+++++
To move the files from staging are to LR
$ git commit -m "first commit"
$ git status
+++++
To see the list of commit
$ git log
+++++
remaining files I want to move to LR
$ git add .
$ git commit -m "second commit"
$ git log
(or)
git log --oneline
$ git status
+++++

```

Notes

1) Setting username and email

2) As per git , there are three comopnents
 working directory -- untracked files
 staging area -- staged files
 LR -- committed files

3) commands

To initilize working directory as git repo
 To move files to staging are
 To bring back files from staging to untracked section
 To move files from staging to LR
 To check the status of untracked files and staged files
 To see the commit history

Branching

This feature is provided in git, so that developers can create code related to different functionalities on separate branches. This helps the development team in creating the code in an uncluttered way. Later this code can be merged with master branch. Default branch of git is "Master"

Developer

Home Page -- f1 f2 f3

Services -- f4 f5

Contact US -- f6 f7 f8

branch

git init -- master

+++++

git branch branchname

git branch test

+++++

Git rebase

=====

This is called as fastforward merge.

The commits from the child branch are added to the top of the master branch.

This is helpful when we want code from a branch to be reflected as the latest working version on master.

Rearrange the commit order

git rebase -i HEAD~4

Merge the commit's using "squash"

git rebase -i HEAD~4

Remove the pick word and replace it with squash

How to selectively pickup the commit's from child branch (Cherry-pick)

\$ git cherry-pick af7f4dc 6b8166d (commit id's)

+++++

Git Amend

\$ git commit --amend -m "a"

How to go back to previous version of code

git reset --hard c9187df

1. git init - Initializes a new Git repository in the current directory.
2. git clone - Clones a remote repository to your local machine.
3. git add - Adds changes to the staging area.
4. git commit - Commits changes to the local repository.

5. git push - Pushes changes to the remote repository.
6. git pull - Pulls changes from the remote repository.
7. git branch - Lists all branches in the local repository.
8. git checkout - Switches to a different branch.
9. git merge - Merges changes from one branch to another.
- 10.git status - Shows the status of the local repository.
- 11.git log - Shows the commit history.
- 12.git diff - Shows the difference between two versions of a file.

For GitHub specifically, here are some additional commands you may find useful:

1. git remote - Lists all remote repositories.
2. git fetch - Fetches changes from the remote repository.
3. git pull-request - Creates a pull request to merge changes from one branch to another.
4. git fork - Forks a remote repository to your personal account.
5. git clone --recursive - Clones a repository and all its submodules.

These are just a few examples of the many commands available in Git and GitHub. I hope this helps you get started!



EXPERIMENT-3

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

Here are some steps you can follow:

1. Create a GitHub account if you don't already have one.
2. Create a new repository on GitHub. You can either create an empty repository or import an existing repository.
3. Clone the repository to your local machine using the command `git clone <repository URL>`.
4. Make changes to the code on your local machine.
5. Use the command `git add <filename>` to add the changes to the staging area.
6. Use the command `git commit -m "<commit message>"` to commit the changes to the local repository.
7. Use the command `git push` to push the changes to the remote repository on GitHub.
8. Repeat steps 4-7 as necessary.
9. Use the command `git pull` to pull changes from the remote repository to your local machine.
10. Use the command `git merge <branch name>` to merge changes from one branch to another.
11. Use the command `git branch <branch name>` to create a new branch.
12. Use the command `git checkout <branch name>` to switch to a different branch.

EXPERIMENT-4

AIM: Jenkins installation and setup, explore the environment

REQUIREMENT : CLOUD ACCOUNT

AWS OR AZURE OR GCP

DESCRIPTION

- What is Jenkins?
- What is CI/CD (Continuous Integration Continuous Delivery)
- Stages in CI/CD
- Creating CI/CD Infrastructure in AWS
- Installing Jenkins in EC2 Machine
- How to run a sample job in Jenkins?

Jenkins is a self contained, open source automation server which can be used to automate all tasks related to building , testing and delivery activities.

Jenkins can be installed even on standalone be any machine with a java runtime envirowment (JRE) Installed.

Jenkins is a tool for Implementing CI-CD (Continuous Integration - Continuous Delivery)

Stages in CI-CD

Stage 1 : Continuous Download

Stage 2: Continuous Build

Stage 3: Continuous Deployment

Stage 4: Continuous Testing

Stage 5: Continuous Delivery

1-4 ----- Continuous Integration

5 ---- Continuous Delivery

+++++

Create Instance in AWS

1) Create the AWS Account

2) Login with your aws account

3) Click on Services

4) Click on EC2

5) Click on Instance

- 6) Click on launch instance
- 7) Select Ubuntu Server 18 (Free For Eligible)
- 8) Select t2.micro (Free For Eligible)
- 9) Click on Next: Configure Instance Details
- 10) Enter 3 in Number of Instance
- 11) Click on Add storage
- 12) Click on Next : Add Tags
- 13) Click on Next : Configure Security Group
- 14) Click on Add Rule
- 15) Select Type as All Traffic
- 16) Select Source as Anywhere
- 17) Click on Review and launch
- 18) If you are doing first time then you need to select Create a new key pair
- 19) Enter any name in key pair name
- 20) Click on download key pair

This key pair helps us to connect us to our data center.

- 21) Click on launch instance
- 22) Give all 3 instance proper name (Dev Server, QA Server, Prod Server)

How to Connect with the AWS Instance

- 1) Select that Instance
- 2) Press Connect
- 3) You will get the SSH Command
- 4) Copy the SSH Command
- 5) Go to the folder where you have place your key then Open GITBASH in local machine

Git Bash you will get automatically when you have install GIT in your local machine.

6) Check your current location using pwd command

7) Now Paste the SSH Command in GITBASH

8) Just Type Yes

9) Now you are connect to the AWS Instance

Now we run any command that command run in the AWS Intance

Important Point : If you are not doing practice in AWS Stop all the instance.

Install Jenkins in AWS Instance

To install Jenkins the first thing we need java file so first we need to install java like we have done in the local instance.

We need to download Java 1.8 or more.

1) Update the apt repository

`sudo apt update`

2) `sudo apt install openjdk-8-jdk -y`

3) Check the Java Version

`java -version`

4) Install Maven & Git

`sudo apt-get install -y git maven`

5) Check the Verion of Git & Maven

For Git : `git --version`

For Maven : `mvn --version`

6) Download & install Jenkins

Open Jenkins website (<https://jenkins.io/download/>)

Go to Long Term Support

Select Generic Java Package (.war)

We are selecting generic java package file because jenkins will install on those machine where java is already install. If we have java install in windows machine jenkins will work. Only pre requirement is java needs to be install.

For Windows we just need to click on the file and it will download automatically.

For Linux machine enter command `wget` and paste the url to download the file.

To get the URL right click on generic java package and click on copy link address.

(wget http://mirrors.jenkins.io/war-stable/latest/jenkins.war)

wget https://get.jenkins.io/war-stable/2.277.2/jenkins.war

11) Start the Jenkins.war file

java -jar jenkins.war

Every day if we want to run the jenkins we need to run this command.

12) Access Jenkins Home Page

Select DEV Instance & Press Connect.

Copy the Domain Name On 4th point.

Paste the Domain name in the browser and in the end enter :8080 with the default port number.

We can access the jenkins with dev server Public IP.

Copy the public ip of the dev server and paste the ip address in the browser and in the end enter :8080 with the default port number.

Public

13) Unlock Jenkins

When we are installing jenkins it will automatically give you the password in the github terminal.

Copy the password and paste the browser.

You will get the password on the step 11

14) Press Install Suggested Plugins

15) Create First admin user

The first user which we create here is the admin user of the jenkins.

Click on save and continue.

Click on save and finish

+++++

Create Sample Job

Build tab

Click on execute Shell

In Command Box Enter echo " Hello Jenkins"

Click on Console Output

+++++

1. Install Java: Jenkins is built with Java, so you'll need to install Java on your system. You can download Java from the Oracle website.
2. Download Jenkins: You can download Jenkins from the official website.
3. Install Jenkins: Once you've downloaded the Jenkins package, you can install it on your system. The installation process will vary depending on your operating system. On Linux, you can use the command `sudo apt-get install jenkins` to install Jenkins.
4. Start Jenkins: After installation, you can start Jenkins using the command `sudo systemctl start jenkins` on Linux. Jenkins will run on port 8080 by default.
5. Access Jenkins: Once Jenkins is running, you can access the Jenkins dashboard by opening a web browser and navigating to `http://localhost:8080`. You'll need to complete the initial setup process by entering the administrator password that was generated during installation.
6. Install plugins: Jenkins has a large collection of plugins that extend its functionality. You can install plugins by navigating to the "Manage Jenkins" page and selecting "Manage Plugins". From here, you can search for and install plugins.
7. Create a job: To set up a Jenkins job, you'll need to create a new project and configure its settings. From the Jenkins dashboard, select "New Item" and enter a name for your project. From here, you can configure settings such as the source code repository, build triggers, and post-build actions.
8. Run a job: Once you've set up a Jenkins job, you can run it by clicking the "Build Now" button on the project's dashboard. Jenkins will execute the build process and display the results on the dashboard.
9. Explore Jenkins environment: Jenkins provides a variety of features and tools for managing build processes, such as build history, build status, and build logs. You can explore these features by navigating through the Jenkins dashboard and project pages.

EXPERIMENT-5

AIM: Demonstrate continuous integration and development using Jenkins

REQUIREMENT : CLOUD ACCOUNT

AWS OR AZURE OR GCP

DESCRIPTION

- Install Tomcat on AWS Environment
- Performing continuous download
- Performing continuous build
- How to install Plug-in
- Performing continuous Deployment
- Importance of Context Path

Install TOMCAT In QA & Production Server

- 1) Select QA Server and press connect
- 2) Copy the SSH Command
- 3) Open GIT Bash & paste the SSH Command

Press Yes

- 4) Update the apt repository
`sudo apt-get update`

- 5) Install tomcat8
`sudo apt-get install -y tomcat8`

After this we need to install one more package
`sudo apt-get install -y tomcat8-admin`

- 6) Check the tomcat is intall or not

Copy the public IP of the QA Server then paste in the browser and in the end enter :8080

`qa_server_public_ip:8080`

Setting the path of tomcat in jenkins

- 7) enter linux command in QA Server - `cd /etc/tomcat8/`

- 8) enter linux command in QA Server - `ls`

9) You will find the file tomcat-users.xml

10) Open the file -- sudo vim tomcat-users.xml

11) In the end we need to add one statement

```
<user username="training" password="sunilsunil" roles="manager-script,manager-status,manager-gui"/>
```

save and quit

press esc

type :wq

press enter

12) When ever we do any changes done in any service we need to restart the service
sudo service tomcat8 restart

13) After this the same above 12 steps we need to do in the prod server also.

+++++++=

Prod Instance

```
<user username="learning" password="sunilsunil" roles="manager-script,manager-status,manager-gui"/>
```

+++++++

First Start All the AWS Machines.

Connect Dev Server

Start the Jenkins

```
java -jar jenkins.war
```

Stage 1 : Continuous Download START CI-CD

1) Create New item as free style project

2) Click on source code managment

3) Select GIT

4) Enter the URL of github reposiditory

<https://github.com/sunildevops77/maven.git>

5) Click on apply and save

6) Run the Job

7) Check the console output.

8) Connect to the dev server

9) Go to the location where code is downloaded

sudo su -

cd path of the folder

ls

Stage 2 : Continuous Build

Convert the java files in to artifact (.war file)

10) Click on configure of the same job

11) Go to Build Section

12) Click on add build step

13) Click on Invoke top level maven targets

14) Enter the goal as package

15) click on apply and save

16) Run the Job

17) Click on number & click on console output

18) Copy the path of the war file and check the file in the linux machine

sudo su -

cd path

ls

Stage 3 :Continuous Deployment

Now we need to deploy the war file into the QA Server.

19) For this we need to install "deploy to container" plugin.

Go to Dashboard

Click on manage jenkins

Click on manage plugins

Click on available section

Search for plugin (deploy to container)

Select that plugin and click on install without restart.

20) Click on post build actions of the development job

21) Click on add post build actions

22) Click on deploy war/ear to container

23) Enter the path of the war file (or)
we can give `**/*.war` in war/ear files.

24) Context path: `qaenv`

25) Containers : select tomcat 8

Credentials : Click on add

select jenkins

enter tomcat user name and password

Click on add

Select credentials.

give the private ip of the QA server.

`http://private_ip:8080`
`http://172.31.32.183:8080`

26) Click on apply and save

27) Run the job

28) To access the home page

`public_ip_Qa_server:8080/qaenv`

- Performing continuous download
- Performing continuous build
- How to install Plug-in
- Performing continuous Deployment

- Importance of Context Path
- How to call one job from another job
- Copy Artifact Plugin
- Performing all the five Stages in CI-CD

First Start All the AWS Machines.

Connect Dev Server

Start the Jenkins

```
java -jar jenkins.war
```

Stage 1 : Continuous Download START CI-CD

1) Create New item as free style project

2) Click on source code management

3) Select GIT

4) Enter the URL of github repository

```
https://github.com/sunildevops77/maven.git
```

5) Click on apply and save

6) Run the Job

7) Check the console output.

8) Connect to the dev server

9) Go to the location where code is downloaded

```
sudo su -
```

```
cd path of the folder
```

```
ls
```

Stage 2 : Continuous Build

Convert the java files in to artifact (.war file)

10) Click on configure of the same job

11) Go to Build Section

- 12) Click on add build step
- 13) Click on Invoke top level maven targets
- 14) Enter the goal as package
- 15) click on apply and save
- 16) Run the Job
- 17) Click on number & click on console output
- 18) Copy the path of the war file and check the file in the linux machine
sudo su -

cd path

ls

Stage 3 :Continuous Deployment

Now we need to deploy the war file into the QA Server.

- 19) For this we need to install "deploy to container" plugin.

Go to Dashboard

Click on manage jenkins

Click on manage plugins

Click on available section

Search for plugin (deploy to container)

Select that plugin and click on install without restart.

- 20) Click on post build actions of the development job

- 21) Click on add post build actions

- 22) Click on deploy war/ear to container

- 23) Enter the path of the war file (or)
we can give **/*.war in war/ear files.

- 24) Context path: qaenv

- 25) Containers : select tomcat 8

Credentials : Click on add

select jenkins

enter tomcat user name and password

Click on add

Select credentials.

give the private ip of the QA server.

`http://private_ip:8080`

`http://172.31.47.36:8080`

26) Click on apply and save

27) Run the job

28) To access the home page

`public_ip_Qa_server:8080/qaenv`

`13.127.177.32:8080/qaenv`

+++++

`https://github.com/sunildevops77/TestingNew.git`

Step 1: Connect to Devserver from git bash

Step 2: Start Jenkins (`java -jar jenkins.war`)

Step 3: Create new item (Name - testing)

Source code management tab, Git

Repository URL - `https://github.com/sunildevops77/TestingNew.git`

Apply -- Save

Step 4: Run the job.

Step 5: Check the path of the files which are downloaded.

`/home/ubuntu/.jenkins/workspace/testing`

Step 6: Configure the same job (testing)

Build -- Add build Step -- Execute shell

(Command: `java -jar testing.jar`)

Command: `echo " Testing passed"`

Now both are independent job.

To call testing job after development job is completed

Go to first job (demo) -- configure
Post build actions -- add post build action -- build other project -
Projects to build - testing (name of the job)

+++++

Copying artifacts from development job to testing job

The artifacts (war) created by the development job should be passed to the testing job so that the testing job can deploy that into tomcat in the prod environment.

Install Plugins

- 1) Go to Jenkins dashboard
- 2) Go to manage jenkins
- 3) Click on Manage plugins
- 4) Search for "Copy Artifact" plugin
- 5) Install the plugin

Stage 5 : Continous Delivery

- 1) Go to Development job
- 2) Go to Configure
- 3) Go to Post build actions tab
- 4) Click on add post build action
- 5) Click on Archive the artifacts
- 6) Enter **/*.war
- 7) Click on apply and save
- 8) Go to testing Job
- 9) Click on configure
- 10) Go to Build section
- 11) Click on add build steps
- 12) Click on copy artifacts from another project

- 13) Enter Development as project name
- 14) For Deployment Go to Post build actions section
- 15) Click on add post build action
- 16) Click on deploy war/ear to a container
- 17) Enter **/*.war in war/ear files
- 18) Context path : prodenv
- 19) Click on add container
- 20) Select tomcat 8
- 21) Select your Credentials
- 22) Enter private ip:8080 of the prod server

`http://172.31.39.130:8080`

- 23) Click on Apply and save

+++++

- 7) enter linux command in Prod Server - `cd /etc/tomcat8/`

- 8) enter linux command in prod Server - `ls`

- 9) You will find the file tomcat-users.xml

- 10) Open the file -- `sudo vim tomcat-users.xml`

- 11) In the end we need to add one statement

```
<user username="learning" password="sunilsunil" roles="manager-script,manager-status,manager-gui"/>
```

- 12) we need to restart the service

```
sudo service tomcat8 restart
```

EXPERIMENT-6

AIM . Explore Docker commands for content management.

Docker is a popular containerization platform that allows developers to build, ship, and run applications in a consistent and portable way. Docker provides a range of commands that can be used for content management, including:

1. **docker build:** This command is used to build a Docker image from a Dockerfile. The Dockerfile contains instructions for how to create the image, including what base image to use, what packages to install, and what commands to run.
2. **docker push:** Once you have built a Docker image, you can push it to a Docker registry such as Docker Hub. This command uploads the image to the registry, making it available for others to use.
3. **docker pull:** This command is used to download a Docker image from a registry. It pulls the image from the registry and saves it on your local machine.
4. **docker run:** This command is used to start a Docker container from an image. You can specify options such as port mapping, volume mounting, and environment variables when running the container.
5. **docker exec:** Once a container is running, you can use the docker exec command to run a command inside the container. This can be useful for debugging or troubleshooting purposes.
6. **docker cp:** This command is used to copy files between your local machine and a Docker container. You can use this command to copy files into the container or to retrieve files from the container.
7. **docker commit:** If you make changes to a running container, you can use the docker commit command to create a new image based on the changes. This can be useful for creating custom images that include specific configurations or software.
8. **docker save:** This command is used to save a Docker image as a tar archive. You can use this command to backup your images or to transfer them to another machine.
9. **docker load:** Once you have saved a Docker image as a tar archive, you can use the docker load command to load the image into Docker. This makes the image available for use on your local machine.

EXPERIMENT-7

AIM: Develop a simple containerized application using Docker.

REQUIREMENT : CLOUD ACCOUNT

AWS OR AZURE OR GCP

1. Create a new directory for your application, and create a file named `app.py` with the following contents:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello, World!'
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

This is a very simple Flask web application that just displays a message when you access the root URL.

2. Next, create a file named `Dockerfile` in the same directory, with the following contents:

```
FROM python:3.9-alpine
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "app.py"]
```

This Dockerfile defines a new image based on the `python:3.9-alpine` base image. It sets the working directory to `/app`, copies the `requirements.txt` file into the container and installs the dependencies using pip. Then, it copies the entire

application code into the container, exposes port 5000, and sets the command to run the `app.py` file using the `python` interpreter.

3. Create a file named `requirements.txt` with the following contents:

```
Flask==2.0.2
```

This file lists the dependencies that our application needs to run.

4. Finally, build the Docker image using the following command:

```
docker build -t myapp .
```

This command will build a Docker image based on the `Dockerfile` in the current directory and tag it with the name `myapp`.

5. Run the container using the following command:

```
docker run -p 5000:5000 myapp
```

This will start the container and map port 5000 on the container to port 5000 on the host machine. You can now access the application by visiting `http://localhost:5000` in your web browser.

That's it! You now have a simple containerized web application running using Docker.

EXPERIMENT-8

AIM: . Integrate Kubernetes and Docker

REQUIREMENT : CLOUD ACCOUNT

AWS OR AZURE OR GCP

Kubernetes is a powerful container orchestration tool that can manage, scale, and deploy Docker containers across a cluster of machines. Here's a simple example of how to integrate Kubernetes and Docker:

1. Create a Docker image for your application following the steps in the previous question.
2. Create a Kubernetes deployment configuration file named `deployment.yaml` with the following contents:

```
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: myapp:latest
```


ports:

- containerPort: 5000

This deployment configuration specifies a desired state for your application to run in Kubernetes. It defines a deployment named `myapp` that should run 3 replicas of your application. It also specifies a selector that matches the labels on the pods that should be part of this deployment, and a template for the pod specification. The pod template specifies that it should run a container named `myapp` using the `myapp:latest` image we created earlier, and expose port 5000.

3. Create a Kubernetes service configuration file named `service.yaml` with the following contents:

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  selector:
    app: myapp
  ports:
    - name: http
      port: 80
      targetPort: 5000
  type: LoadBalancer
```

This service configuration specifies a Kubernetes service that exposes your application to the outside world. It uses a selector to route traffic to the pods that match the labels for our `myapp` deployment. It also specifies that the service should listen on port 80 and forward traffic to port 5000 on the pods, and it sets the service type to `LoadBalancer`, which will allow us to access the service from outside the cluster.

4. Deploy your application to Kubernetes using the following command:

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

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

This will create the Kubernetes deployment and service using the configuration files we just created.

5. Finally, check the status of your deployment and service using the following command:

```
kubectl get deployment myapp
```

```
kubectl get service myapp-service
```

This will show you the current status of your deployment and service.

That's it! You now have a Docker container running your application managed by Kubernetes, and accessible from outside the cluster through a Kubernetes service.

EXPERIMENT-9

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

REQUIREMENT : CLOUD ACCOUNT

AWS OR AZURE OR GCP

Automating the process of running a containerized application using Kubernetes involves using a combination of Kubernetes deployment configurations and CI/CD tools to create a continuous delivery pipeline. Here's an example of how you can automate the process of deploying a containerized application using Kubernetes:

1. Create a Kubernetes deployment configuration file for your application as described in the previous question.
2. Create a Kubernetes service configuration file for your application as described in the previous question.
3. Create a Dockerfile for your application as described in the first question.
4. Create a Kubernetes deployment pipeline using a CI/CD tool like Jenkins or GitLab CI. Here's an example of how you can create a deployment pipeline using Jenkins:
 - a. Install the Kubernetes plugin for Jenkins.
 - b. Create a new Jenkins job and configure it to use the Kubernetes plugin.
 - c. In the Jenkins job, add a build step that builds the Docker image using the Dockerfile.
 - d. Add another build step that tags the Docker image with a version number or unique identifier.
 - e. Add a deploy step that uses the Kubernetes plugin to deploy the new Docker image to your Kubernetes cluster.
 - f. Finally, add a step that cleans up old images or pods to prevent accumulation of unused resources.
5. Set up a webhook in your version control system to trigger the Jenkins job whenever changes are pushed to the application's repository. This will automatically trigger a new build and deployment whenever changes are made to the application's code.

With this setup, any changes made to your application's code will automatically trigger a new build and deployment to your Kubernetes cluster. This can significantly simplify the process of managing and deploying your application, and help ensure that your deployment is always up-to-date with the latest version of your code.

EXPERIMENT-10

AIM: . Install and Explore Selenium for automated testing.

REQUIREMENT Selenium WebDriver

Selenium is a popular open-source tool used for automated testing of web applications. Here are the steps to install and explore Selenium:

1. Install the Selenium WebDriver

You can install the Selenium WebDriver using the language-specific bindings. Here, we will use Python as an example:

```
pip install selenium
```

2. Download the appropriate web driver executable

You need to download a web driver executable that corresponds to the browser you want to automate. Selenium supports a range of browsers, including Chrome, Firefox, Safari, and more. You can download the driver from the official website of the corresponding browser.

3. Create a Selenium test script

Create a new Python script to run your Selenium test. Here's an example of a simple script that opens a webpage in Chrome and closes the browser:

```
from selenium import webdriver

# Path to the chromedriver executable
driver_path = 'path/to/chromedriver'

# Initialize a new Chrome browser instance
driver = webdriver.Chrome(driver_path)

# Open a webpage
driver.get('https://www.google.com')

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

4. Run the Selenium test script

To run the test script, simply execute it in the Python environment:

```
python my_selenium_test.py
```

This will launch the Chrome browser, navigate to the Google homepage, and then close the browser.

5. Explore Selenium's capabilities

Selenium provides a wide range of capabilities for automating web application testing. You can interact with elements on a webpage, simulate user input, and perform various types of tests, such as functional tests, regression tests, and load tests.

Here are some examples of what you can do with Selenium:

- Find an element on a webpage and interact with it (e.g., clicking a button, filling out a form field)
- Navigate through multiple pages and verify that certain elements are present
- Test the responsiveness of a webpage by resizing the browser window
- Test the performance of a webpage by measuring page load times
- Simulate user input by typing text, clicking buttons, and scrolling
- Test different scenarios by setting up test data and verifying that the expected results are achieved

Overall, Selenium is a powerful tool for automating web application testing, and it can help save time and effort in manual testing while increasing the quality of your software.

EXPERIMENT-11

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

REQUIREMENT Selenium WebDriver

Here is a simple program in JavaScript that performs the addition of two numbers:

JAVA SCRIPT

```
function addNumbers(num1, num2) {  
    return num1 + num2;  
}  
  
console.log(addNumbers(2, 3));
```

To perform testing of this program using Selenium, we can write a test script that:

1. Launches a web browser
2. Navigates to a web page that contains the JavaScript code
3. Executes the JavaScript code and retrieves the output
4. Verifies that the output is correct

Here's an example of a Selenium test script written in Python that performs the above steps:

PYTHON

```
from selenium import webdriver  
  
# Path to the chromedriver executable  
driver_path = 'path/to/chromedriver'  
  
# Initialize a new Chrome browser instance  
driver = webdriver.Chrome(driver_path)  
  
# Navigate to a web page that contains the JavaScript code  
driver.get('https://example.com/test.js')  
  
# Execute the JavaScript code and retrieve the output
```

```
output = driver.execute_script('return addNumbers(2, 3)')  
  
# Verify that the output is correct  
  
assert output == 5, f"Expected 5, but got {output}"  
  
# Close the browser  
  
driver.quit()
```

Note: Replace `path/to/chromedriver` with the actual path to the chromedriver executable on your system, and `https://example.com/test.js` with the URL of the web page that contains the JavaScript code.

This script launches a Chrome browser, navigates to the web page containing the JavaScript code, executes the `addNumbers` function, retrieves the output, and then verifies that the output is correct. If the output is not equal to 5, an assertion error will be raised. Finally, the browser is closed.

You can modify this script to test your own JavaScript code by replacing the URL of the web page containing the code, and modifying the function call and expected output to match your code.



EXPERIMENT-12

AIM: Develop test cases for the above containerized application using seleniu

REQUIREMENT Selenium WebDriver

here are some test cases for the containerized application using Selenium:

- 1. Test case: Verify that the login page is displayed correctly**
 - Launch the web browser
 - Navigate to the login page
 - Verify that the login page is displayed correctly (check for the presence of login form elements, logo, etc.)
 - Close the browser
- 2. Test case: Verify that an error message is displayed when incorrect credentials are entered**
 - Launch the web browser
 - Navigate to the login page
 - Enter incorrect credentials (e.g. wrong username or password)
 - Click the login button
 - Verify that an error message is displayed (check for the presence of an error message element)
 - Close the browser
- 3. Test case: Verify that a user can log in successfully**
 - Launch the web browser
 - Navigate to the login page
 - Enter valid credentials
 - Click the login button
 - Verify that the user is redirected to the correct page (e.g. dashboard)
 - Close the browser
- 4. Test case: Verify that a user can log out successfully**
 - Launch the web browser
 - Navigate to the login page
 - Enter valid credentials
 - Click the login button
 - Verify that the user is redirected to the correct page (e.g. dashboard)
 - Click the logout button
 - Verify that the user is logged out and redirected to the login page
 - Close the browser
- 5. Test case: Verify that a user can create a new record**
 - Launch the web browser
 - Navigate to the login page
 - Enter valid credentials
 - Click the login button
 - Navigate to the create record page

- Enter valid data into the form fields
- Click the save button
- Verify that the record is saved and displayed correctly on the records page
- Close the browser

These are just a few examples of test cases that can be developed for the containerized application using Selenium. The specific test cases may vary depending on the functionality and requirements of the application.

