

Introduction to DevOps.

The term 'DevOps' was coined in the year 2009 by Patrick Debois. As its name implies it is a conjunction of two different words, 'Development' and 'Operations'. The reason behind this name is that it combines these two areas of an organization. Unlike the conventional organizations where these two areas were considered separate, DevOps believes in bridging this gap by incorporating their operations with one another at each step. By means of combining, DevOps helps achieve agility, quality, and consistency at the same time.



In the early year of software development when it was just coming up, waterfall model was used. Waterfall model is a very standard model used in many different fields, not just software development. It was very useful when the requirements were concrete and the development cycles were long. New Methodologies like Agile, Spiral Model, Extreme Programming, Feature Driven Development, Lean Methodology have come into existence.

Definition of DevOps:

DevOps is a collaborative way of developing and deploying software. DevOps is a software development technique that combines development (Dev) and operations (Ops) to optimize software production and deployment efficiency, speed, and security. It aims to shorten the systems development life cycle and provide continuous delivery with high software quality. It was designed to create a more agile software delivery process that facilitates faster and more reliable application value in the market.

There are several practices in DevOps, which include automation, collaborative systems, fast feedback, and continuous improvement. This concept borrows from Agile methodologies, where applications were built and iteratively shipped at much quicker rates.

Features:

- DevOps is an approach based on agile and lean principles in which business owners, development, operations, and quality assurance team collaborate to deliver software in a continuous stable manner
- DevOps is an environment that promotes cross practicality, shared business tasks and belief
- DevOps is a movement that improves IT service delivery agility
- DevOps is a culture that promotes better working relationship within the company
- DevOps is a set of practices that provides rapid, reliable software delivery

DevOps Phases:

- Continuous Development
- Continuous Testing
- Continuous Integration
- Continuous Deployment
- Continuous Monitoring

EXPERIMENT NO. 1

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

Program:

registration.html

```
<html>
<head>
  <link href="register.css" rel="stylesheet" />
</head>

<body>
<h1> DevOps Lab</h1>

<h2> Student Registration Form</h2>
<form>
<table border="5" align="center" cellspacing="10" cellpadding="10" >

<tr>
<td>Name</td><td><input type="text"></td>
</tr>

<tr>
<td>Contact Number</td>
<td><input type="text"></td>
</tr>

<tr>
<td>Gender</td>
<td><input type="radio" name="g">Male
      <input type="radio" name="g">Female</td>
</tr>

<tr>
<td>Address</td>
<td><textarea rows="5" cols="15"></textarea></td>
</tr>

<tr>
<td>Hobbies</td>
<td><input type="checkbox">Singing
      <input type="checkbox">Travelling
      <input type="checkbox">Reading novels
    </td>
</tr>

<tr>
<td>Skillset</td>
<td><input type="checkbox">C
      <input type="checkbox">Python
      <input type="checkbox">Java
    </td>
</tr>

<tr>
<td>Highest Qualification</td>
```

```

        <td><select>
            <option><--SELECT--></option>
            <option>Ph.D</option>
            <option>M.E/M.Tech</option>
            <option>B.E/B.Tech</option>
            <option>Diploma</option>
            <option>Inter</option>
            <option>SSC</option>
        </select>
    </td>
</tr>
<td>District</td>
    <td><select>
        <option>--SELECT--></option>
        <option>Adilabad</option>
        <option>Zaheerabad</option>
    </select>
    </td>
</tr>

<tr>
<td><input type="submit" value="Register"></td>
<td><input type="reset" value="Clear"></td>
</tr>

</table>
</body>
</html>

```

register.css

```

h1{
    color:green;
    text-align:center;
}

h2{
    color:blue;
    text-align:center;
}

p{
    color:red;
}

table{
    background-color: cyan;
}

td{
    color:red;
    font-size:24px;
}

input
{
    color:blue;
    font-size:24px;
    text-align : center;
}

select

```

```

{
    color:blue;
    font-size:24px;
    text-align : center;
}

```

Output:

DevOps Lab

Student Registration Form

Name	<input type="text"/>
Contact Number	<input type="text"/>
Gender	<input type="radio"/> Male <input type="radio"/> Female
Address	<input type="text"/>
Hobbies	<input type="checkbox"/> Singing <input type="checkbox"/> Travelling <input type="checkbox"/> Reading novels
Skillset	<input type="checkbox"/> C <input type="checkbox"/> Python <input type="checkbox"/> Java
Highest Qualification	<--SELECT--> v
District	--SELECT--> v
Register	Clear

EXPERIMENT NO. 2

AIM: Explore Git and GitHub commands.

Overview of Version Control Systems:

These days when software is developed, It is not developed with the mind-set that there will only be one piece of code that will be deployed and that's it. Smaller snippets of code are deployed in regular successions with regular feedbacks. This leads to many different versions of the code. And that creates a need to organise the code and all of its different version of it. This is where need for Version Control comes in. It is a practice of managing and storing different version of a source code especially with larger companies that have multiple projects and multiple teams working within it.

Version control systems are tools used in software development to manage changes to source code, documents, computer programs etc over time. They track modifications to files, allowing teams to collaborate efficiently and maintain a history of project changes.

Importance of Version Control in Software Development:

Version control is crucial in software development as it enables developers to:

- Coordinate collaboration among team members.
- Track changes and revert to previous versions if needed.
- Maintain a consistent and organized codebase.
- Facilitate code review and integration.
- Ensure traceability and accountability in the development process.

Types Of Version Control System

- **Centralized Version Control System**
 - Centralized Version Control System has one single copy of code in the central server
 - Developers will have to “commit” their changes in the code to this central server
 - “Committing” a change simply means recording the change in the central system
 - Examples: SubVersion , HelixCore.
- **Distributed Version Control System**
 - In Distributed VCS, one does not necessarily rely on a central server to store all the versions of a project's file
 - Every developer “clones” a copy of the main repository on their localsystem
 - This also copies, all the past versions of the code on the local systemtoo
 - Therefore,the developer need not be connected to the internet to work on the code
 - Examples: Perforce, git, mercurial.

Introduction to Git

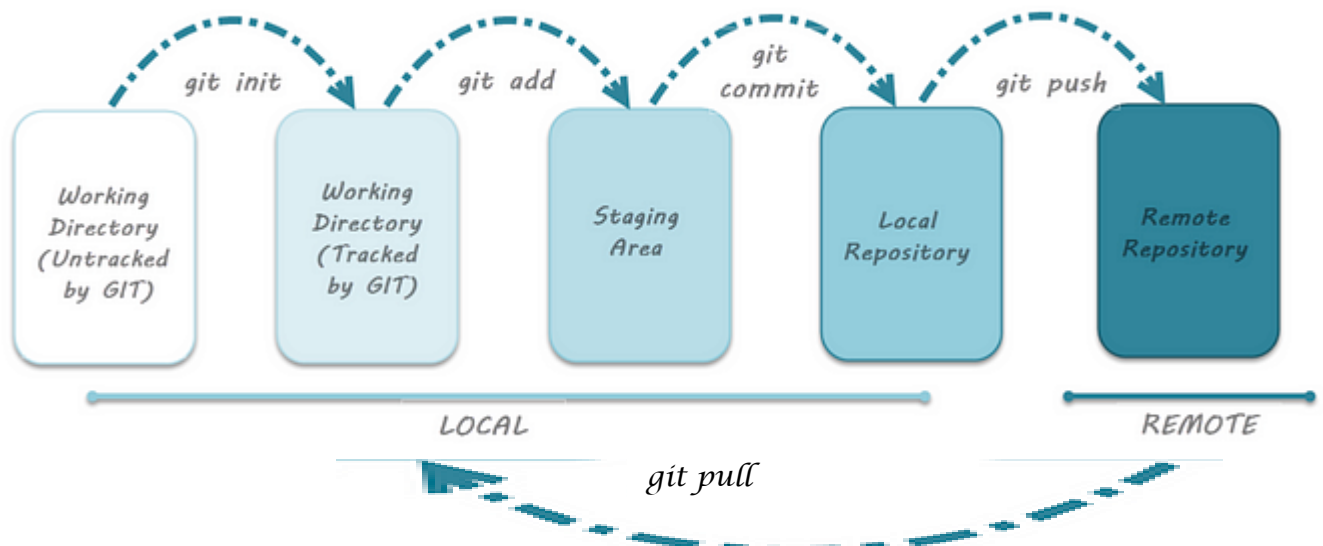
Git is a version-control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source-code management in software development, but it can be used to keep track of changes in any set of files. Git is the most popular tool among all the DVCS tools.

Git Lifecycle

The lifecycle stages of files in Git are

- **Working Directory**
 - The place where your project resides in your local disk
 - This project may or may not be tracked by git
 - In either case, the directory is called the working directory
 - The project can be tracked by git, by using the command `git init`
 - By doing `git init`, it automatically creates a hidden `.git` folder
- **Staging Area**
 - Once we are in the working directory, we have to specify which files are to be tracked by git
 - We do not specify all files to be tracked in git, because some files could be temporary data which is being generated while execution
 - To add files in the staging area, we use the command `git add`
- **Commit**
 - Once the files are selected and are ready in the staging area, they can now be saved in repository
 - Saving a file in the repository of git is known as doing a commit
 - When we commit a repository in git, the commit is identified by a commit id
 - The command for initializing this process is `git commit -m "message"`

Once the files are committed, they can be pushed to a remote repository such as GitHub



Program: GIT COMMANDS

Git Installation :

```
$sudo apt-get update
$sudo apt-get install git
$git --version
```

Git Configuration:

Configure user information for all local repositories

```
$ git config --global user.name "username"
    set a name that is identifiable for credit when review version history
$ git config --global user.email "user@mail.com"
    set an valid email address that will be associated with each history marker
$git config --global color.ui auto
    set automatic command line coloring for Git for easy reviewing
$ git config --list
```

Setup & Init :

When starting out with a new repository, you only need to do it once; either locally, then push to GitHub, or by cloning an existing repository.

```
$git init
    initialize an existing directory as a Git repository
$git clone [url]
    retrieve an entire repository from a hosted location via URL
```

Stage & Snapshot

```
$git status
    show modified files in working directory, staged for your next commit
$git add [file]
    add a file as it looks now to your next commit (stage)
$git reset [file]
    unstage a file while retaining the changes in working directory
$git diff
    diff of what is changed but not staged
$git diff --staged
    diff of what is staged but not yet committed
$git commit -m "[descriptive message]"
    commit your staged content as a new commit snapshot
```


Branch & Merge

Isolating work in branches, changing context, and integrating changes

\$git branch

list your branches. a * will appear next to the currently active branch

\$git branch [branch-name]

create a new branch at the current commit

\$git checkout

switch to another branch and check it out into your working directory

\$git merge [branch]

merge the specified branch's history into the current one

\$git log

show all commits in the current branch's history

Share & Update

Retrieving updates from another repository and updating local repos

\$git remote add [alias] [url]

add a git URL as an alias

\$git fetch [alias]

fetch down all the branches from that Git remote

\$git merge [alias]/[branch]

merge a remote branch into your current branch to bring it up to date

\$git push [alias] [branch]

Transmit local branch commits to the remote repository branch

\$git pull

fetch and merge any commits from the tracking remote branch

Introduction to GitHub:

GitHub is a web-based platform that offers version control and collaboration services for software development projects. It provides a way for developers to work together, manage code, track changes, and collaborate on projects efficiently. GitHub is built on top of the Git version control system, which allows for distributed and decentralised development.

Key Features of GitHub:

- **Version Control:** GitHub uses Git, a distributed version control system, to track changes to source code over time. This allows developers to collaborate on projects while maintaining a history of changes and versions.
- **Repositories:** A repository (or repo) is a collection of files, folders, and the entire history of a project. Repositories on GitHub serve as the central place where code and project-related assets are stored.
- **Collaboration:** GitHub provides tools for team collaboration. Developers can work together on the

same project, propose changes, review code, and discuss issues within the context of the project.

- **Pull Requests:** Pull requests (PRs) are proposals for changes to a repository. They allow developers to submit their changes for review, discuss the changes, and collaboratively improve the code before merging it into the main codebase.

- **Issues and Projects:** GitHub allows users to track and manage project-related issues, enhancements, and bugs. Projects and boards help organize tasks, track progress, and manage workflows.

- **Forks and Clones:** Developers can create copies (forks) of repositories to work on their own versions of a project. Cloning a repository allows developers to create a local copy of the project on their machine.

- **Branching and Merging:** GitHub supports branching, where developers can create separate lines of development for features or bug fixes. Changes made in branches can be merged back into the main codebase.

- **Actions and Workflows:** GitHub Actions enable developers to automate workflows, such as building, testing, and deploying applications, based on triggers like code pushes or pull requests.

- **GitHub Pages:** This feature allows users to publish web content directly from a GitHub repository, making it easy to create websites and documentation for projects.

Experiment Steps for using GitHub:

Step 1: Create a Github Account

- Go to **<https://github.com>** and click on **Sign up for GitHub**
- Provide with your details – username, email address, password & click on **Create Account**

Step 2: Cloning a Repository

- Sign in to your GitHub account.
- Find a repository to clone (you can use a repository of your own or any public repository).
- Click the "Code" button and copy the repository URL.
- Open your terminal or command prompt.
- Navigate to the directory where you want to clone the repository.
- Run the following command:

```
$ git clone <repository_url>
```
- Replace <repository_url> with the URL you copied from GitHub.
- This will clone the repository to your local machine.

Step 3: Making Changes and Creating a Branch

- Navigate into the cloned repository:

```
$ cd <repository_name>
```
- Create a new text file named "example.txt" using a text editor.
- Add some content to the "example.txt" file.
- Save the file and return to the command line.
- Check the status of the repository:

```
$ git status
```

- Stage the changes for commit:
\$ git add example.txt
- Commit the changes with a descriptive message
\$ git commit -m "Add content to example.txt"
- Create a new branch named "feature"
\$ git branch feature
- Switch to the "feature" branch:
\$ git checkout feature

Step 4: Pushing Changes to GitHub

- Add Repository URL in a variable
\$ git remote add origin <repository_url>
- Replace <repository_url> with the URL you copied from GitHub.
- Push the "feature" branch to GitHub
\$ git push origin feature
- Check your GitHub repository to confirm that the new branch "feature" is available.

Step 5: Collaborating through Pull Requests

- Create a pull request on GitHub
- Go to the repository on GitHub.
- Click on "Pull Requests" and then "New Pull Request."
- Choose the base branch (usually "main" or "master") and the compare branch ("feature").
- Review the changes and click "Create Pull Request."
- Review and merge the pull request
- Add a title and description for the pull request.
- Assign reviewers if needed.
- Once the pull request is approved, merge it into the base branch.

Step 6: Syncing Changes

- After the pull request is merged, update your local repository:
\$ git checkout main
\$ git pull origin main

Program: GitHub Commands for Pushing the data from local repository to remote github repository

Step 1: Sign in to your GitHub account.

Step 2: Create a Repository

- Click on **Create a new repository**
- Create a Public Repository by name **samplerepo**
- **Copy the git url**

Step 3: Check version & Configure the settings

- `$ git --version`
- `$ git config --global user.name "sowjanya"`
- `$ git config --global user.email "sowjanya@gmail.com"`
- `$ git remote add origin https://github.com/sowjanya/samplerepo.git`
- `$ git config --list`

Step 3: Create directory & initialize it

- `$ mkdir demo`
- `$ cd demo`
- `$ git init`

Step 4: Create new file, see status, put in staging area & commit into local repo

- `$ touch myfile.txt`
- `$ vi myfile.txt` (put some content)
- `$ git status`
- `$ git add .`
- `$ git commit -m "1st commit file added"`
- `$ git log`
- `$ git show <commit-id>`
- `$ git status`
- `$ git branch -M main`
- `$ git push -u origin main`

Step 5: Go to remote repository and check whether project is uploaded on github.

EXPERIMENT NO. 3

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

Program:

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

Remote repository to Local Repository:

1. Create a GitHub account if you don't already have one.
2. Create a new repository on GitHub with name **Registrationrepo**
3. Copy the URL to your GitHub repository.
Eg: `https://github.com/sowjanya-it/registrationrepo.git`
4. Clone the repository to your local machine:
`$ git clone <repository- url>`
5. Move to the repository directory:
`$ cd <repository-name>`
6. Create a new file in the repository and add the source code written in exercise 1.
7. Stage the changes:
`$ git add <file-name>`
8. Commit the changes:
`$ git commit -m "Added source code for a simple user registration form"`
9. Push the changes to the remote repository:
`$ git push origin master`
10. 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.

Local repository to Remote Repository:

Step 1: Create directory

```
$ mkdir studregistration
$ cd studregistration
$ git init
```

Step 2: Create new repository in your GitHub account and copy URL

Step 3: Configure your user details and Set URL

```
$ git remote add origin <centralgit repo url>
$ git config --global user.name "username"
$ git config --global user.email "user@mail.com"
```

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

Step 4: Create new files and add the source code written in exercise 1.

```
$ vi registration.html  
$ vi register.css
```

Step 5: Check status and add files to git and commit

```
$ git status  
$ git add .  
$ git commit -m "Added 2 files"  
$ git status
```

Step 6: Branch to main and push the code

```
$ git branch -M main  
$ git push -u origin main  
$ git status  
$ git log
```

Step 7: Create a new Branch to main and make changes to file and merge the file.

```
$ git checkout -b branch1           //creates new branch  
$ git branch                       // check the branch  
$ vi registration.html              //make changes to the content of the file  
$ git add .                        // add file to git  
$ git commit -m "made changes in text file on branch1"  
$ git status  
$ git push -u origin branch1  
$ git status  
$ git diff main                    // compares main branch and branch1  
$ git merge main                   // merge main branch and branch1  
$ git pull origin main  
$ git log
```


EXPERIMENT NO. 4

AIM: Jenkins installation and setup, explore the environment.

Introduction to Jenkins:

- Jenkins is a popular open-source tool for Continuous Integration and Continuous Deployment (CI/CD) in software development.
- Jenkins is an open-source automation tool written in Java programming language.
- Jenkins is a server-based application and requires a webserver like Apache Tomcat.
- Jenkins builds and tests our software projects continuously. The the main reason for Jenkins to became so popular is, continuously monitoring of repeated tasks which arise during the development of a project. Example, if your team is developing a project, Jenkins will continuously test your project builds and show you the errors in early stages of your development.

Benefits of using Jenkins CI

- **Reduced Development Cycle** – Since every commit is built and tested, it allows releasing new features to the user faster and with fewer errors.
- **Shorter Time to Integrate Code**—Before the use of Jenkins CI, integration of code was done manually, thus taking a fewdays. In some cases, it might happen that the code is not running, and it is hard to debug as it might have gone through various commits in the repository. Integrating code after every commit ensures that the functionality is not broken after a commit.
- **Faster Feedback Loops** – Developers get feedback and improve the code whenever a test breaks during a commit. Otherwise, debugging the issue can be very difficult, given teams would not be sure which commit resulted in the bug.
- **Automated Workflow** – Teams should not worry about running a manual test for each commit. The Jenkins CI pipeline checks the latest code and builds the code along with the tests. The test can deploy the project in a specific environment if it is green. It can notify the developer by breaking the build.

Steps to install and set up Jenkins:

Step 1: Download and install Jenkins:

- Download the Jenkins package for 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 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.

Step 2: 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.

Step 3: 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.

Program : Jenkins Installation Steps:

```
$ sudo apt-get update
$ sudo apt-get remove --purge jenkins
```

```
$ sudo apt-get install jenkins
$ sudo apt-get enable jenkins
$ sudo apt-get start jenkins
$ sudo apt-get status jenkins
```

```
$ sudo systemctl enable jenkins
$ sudo systemctl start jenkins
$ sudo systemctl status jenkins
```

In browser open <https://localhost:8080>

Copy the path of passcode and open the terminal, and in terminal type

```
$ sudo cat <path of passcode>
```

Copy the password and paste in browser to continue installation. Skip user creation, and after login, in user plug-ins set & update the admin password to “admin”

EXPERIMENT NO. 5

AIM: Demonstrate continuous integration and development using Jenkins.

Continuous Integration (CI) and Continuous Development (CD) are important practices in software development that can be achieved using Jenkins.

CI/CD Principles:

- **Continuous Integration (CI):** CI focuses on automating the process of integrating code changes into a shared repository frequently. It involves building and testing the application each time code is pushed to the repository to identify and address issues early in the development cycle.
- **Continuous Deployment (CD):** CD is the practice of automatically deploying code changes to production or staging environments after successful testing. CD aims to minimize manual intervention and reduce the time between code development and its availability to end-users.

Continuous integration (CI):

CI is the practice of frequently and automatically integrating code changes from multiple contributors into a shared repository. The core idea is that developers regularly merge their code into a central repository, triggering automated builds and tests. Continuous integration refers to the build and unit testing stages of the software release process. Every revision that is committed triggers an automated build and test. With continuous delivery, code changes are automatically built, tested, and prepared for a release to production.

Key aspects of CI include:

- **Automation:** CI tools, like Jenkins, Travis CI, or CircleCI, automate the building and testing of code whenever changes are pushed to the repository.
- **Frequent Integration:** Developers commit and integrate their code changes multiple times a day, reducing integration conflicts and catching bugs early.
- **Testing:** Automated tests, including unit tests and integration tests, are run to ensure that new code changes do not introduce regressions.
- **Quick Feedback:** CI provides rapid feedback to developers about the quality and correctness of their code changes.

Continuous Deployment (CD):

CD is the natural extension of CI. It is the practice of automatically and continuously deploying code changes to production or staging environments after successful integration and testing.

Key aspects of CD include:

- **Automation:** CD pipelines automate the deployment process, reducing the risk of human error and ensuring consistent deployments.
- **Deployment to Staging:** Code changes are deployed first to a staging environment where

further testing and validation occur.

- **Deployment to Production:** After passing all tests in the staging environment, code changes are automatically deployed to the production environment, often with zero downtime.
- **Rollbacks:** In case of issues, CD pipelines provide the ability to rollback to a previous version quickly.

Steps for demonstrating CI/CD using Jenkins:

Step 1: 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.

Step 2: 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.

Step 3: 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.

Step 4: 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.

Step 5: 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.

Step 6: 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.

Step 7: 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.

Program #1 : Create a Jenkins Job for executing shell commands

Steps:

- In browser open <https://localhost:8080> and login in using your account.
- Create a new Jenkins job using the "Freestyle project" type.
 - In the dash board, Click on New item to create job.
 - Assign a meaningful name for the job and select free style project option and click on OK button.
- Provide a description about application and under Buildsteps option choose execute shell.
- Type few linux commands in shell window.
 - **whoami**
 - **pwd**
 - **mkdir demo**
 - **ls**
 - **cd demo**
 - **echo "Hello Jenkins" > sample.txt**
 - **ls**
 - **cat sample.txt**
- Click on Save button.
- Click on build now option available in dash board.
- To watch the output click on Console output

Program #2 : Create a Jenkins Job for executing Java program – Reverse of a number.

Steps:

- Create a file **ReverseNumber.java** and type the code for reverse of a number and save file

```
class ReverseNumber
{
    public static void main(String args[])
    {
        int n=Integer.parseInt(args[0]);
        int rev=0;
        int r;
        while(n>0)
        {
            r=n%10;
            rev=(rev*10)+r;
            n=n/10;
        }
        System.out.println("Reverse number:"+rev);
    }
}
```

- In browser open <https://localhost:8080> and login in using your account.
- Create a new Jenkins job using the "Freestyle project" type.
 - In the dash board, Click on New item to create job.

- Assign a meaningful name for the job and select free style project option and click on OK button.
- Provide a description about application and under Buildsteps option choose execute shell.
- **Copy the program into the Jenkins environment workspace.**
- Type commands for execution of your java program.
 - **javac ReverseNumber.java**
 - **java ReverseNumber 1234**
- Click on Save button.
- Click on build now option available in dash board.
- To watch the output click on Console output

Program #3 : Create a Jenkins Job for executing parameterized Java program – Reverse of a number.

Steps:

- Create a file **PatternDemo.java** and type the code for creating pyramid pattern and save file

```
class PatternDemo
{
    public static void main(String args[])
    {
        int n=Integer.parseInt(args[0]);
        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=i;j++)
            {
                System.out.print(i+"\t");
            }
            System.out.println();
        }
    }
}
```

- In browser open <https://localhost:8080> and login in using your account.
- Create a new Jenkins job using the "Freestyle project" type.
 - In the dash board, Click on New item to create job.
 - Assign a meaningful name for the job and select free style project option and click on OK button.
- In the job configuration, provide description about job and choose "This project is parameterized."
- Add a "String Parameter" named "n1" and set its default value to 5.
- Under Buildsteps option choose execute shell.
- **Copy the program into the Jenkins environment workspace.**
- Type commands for execution of your java program.
 - **javac PatternDemo.java**
 - **java PatternDemo \$n1**

- Click on Save button.
- Click on build now option available in dash board and it will prompt you to enter value for n1.
- Enter value for parameter and click ok.
- To watch the output click on Console output

Program #4 : Create a Jenkins Job for executing parameterized Java program from the Git repository – Biggest of three Numbers.

Steps:

- Create a file **Biggest.java** and type the code for Biggest of three numbers and save file

```
class Biggest
{
    public static void main(String args[])
    {
        int n1=Integer.parseInt(args[0]);
        int n2=Integer.parseInt(args[1]);
        int n3=Integer.parseInt(args[2]);

        if((n1>n2)&&(n1>n3))
        {
            System.out.println(n1+" is biggest");
        }
        else if((n2>n1)&&(n2>n3))
        {
            System.out.println(n2+" is biggest");
        }
        else
        {System.out.println(n3+" is biggest");
        }
    }
}
```

- Login into your GitHub Account and create a new repository with name “Biggestnum”
- Upload the file Biggest.java into your GitHub repository and commit the changes.
- Copy the URL of your Git Repository
- In browser open <https://localhost:8080> and login in using your account.
- Create a new Jenkins job using the "Freestyle project" type.
 - In the dash board, Click on New item to create job.
 - Assign a meaningful name for the job and select free style project option and click on OK button.
- In the job configuration, provide description about job and choose "This project is parameterized."
- Add 3 "String Parameter" named “n1” , “n2” nad “n3” and set its default values.
- Under source management choose **Git** and provide the **Git URL** there.
- Set Branches to build -> Branch Specifier to the working Git branch (ex */main)

- Under Buildsteps option, choose execute shell.
- Type commands for execution of your java program.
 - **javac Biggest.java**
 - **java Biggest \$n1 \$n2 \$n3**
- Click on Save button.
- Click on build parameterized now option available in dash board and it will prompt you to enter value for n1, n2 ad n3.
- Enter values for parameter and click ok.
- To watch the output click on Console output

Program #5 : Create a Jenkins pipeline Job using pipeline script.

Steps:

- In browser open <https://localhost:8080> and login in using your account.
- Create a new Jenkins job using the "pipeline" project type.
 - In the dash board, Click on New item to create job.
 - Assign a meaningful name for the job and select pipeline project option and click on OK button.
- In the job configuration, provide description about job.
- Type the following script under the space provided for writing the script.

```

pipeline {
    agent any
    stages{
        stage('compile'){
            steps{
                echo "Compiled Successfully";
            }
        }
        stage('JUnit'){
            steps{
                echo "JUnit test passed Successfully";
            }
        }
        stage('Qualitycheck'){
            steps{
                echo " Quality Check  passed Successfully";
            }
        }
        stage('Deploy'){
            steps{
                echo "Deployed Successfully";
            }
        }
    }

    post{
        always{

```

```
        echo "This will always run"
    }
    success{
        echo "This will run on success"
    }
    failure{
        echo "This will run on failure"
    }
    unstable{
        echo "This will run only if the run is marked as Unstable"
    }
    changed{
        echo "Pipe line changed"
    }
}
}
```

- Click on Save button.
- Click on build now option available in dash board.
- To watch the output click on Console output

EXPERIMENT NO. 6

AIM: Explore Docker commands for content management.

Introduction

Containerization is a technology that has revolutionised the way applications are developed, deployed, and managed in the modern IT landscape. It provides a standardised and efficient way to package, distribute, and run software applications and their dependencies in isolated environments called containers.

Containerization technology has gained immense popularity, with Docker being one of the most well-known containerization platforms. This introduction explores the fundamental concepts of containerization, its benefits, and how it differs from traditional approaches to application deployment.

Key Concepts of Containerization:

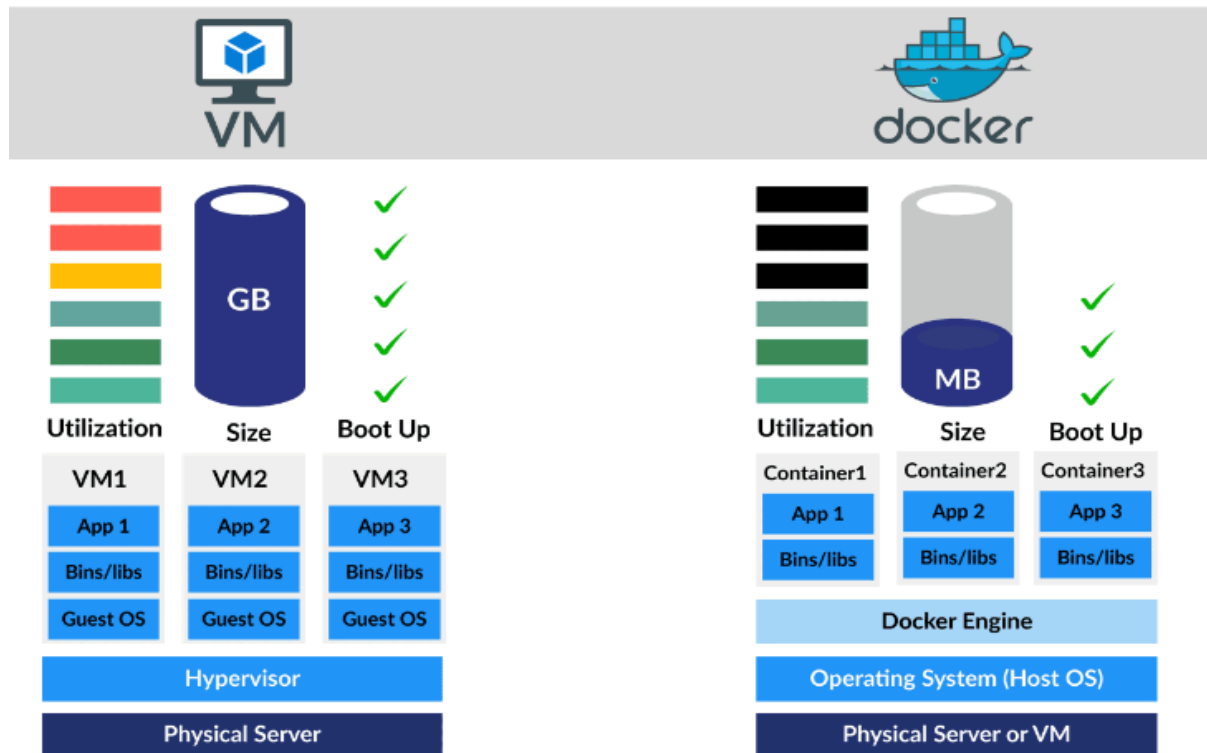
- **Containers:** Containers are lightweight, stand-alone executable packages that include everything needed to run a piece of software, including the code, runtime, system tools, libraries, and settings. Containers ensure that an application runs consistently and reliably across different environments, from a developer's laptop to a production server.
- **Images:** Container images are the templates for creating containers. They are read-only and contain all the necessary files and configurations to run an application. Images are typically built from a set of instructions defined in a Dockerfile.
- **Docker:** Docker is a popular containerization platform that simplifies the creation, distribution, and management of containers. It provides tools and services for building, running, and orchestrating containers at scale.
- **Isolation:** Containers provide process and filesystem isolation, ensuring that applications and their dependencies do not interfere with each other. This isolation enhances security and allows multiple containers to run on the same host without conflicts.

Benefits of Containerization:

- **Consistency:** Containers ensure that applications run consistently across different environments, reducing the "it works on my machine" problem.
- **Portability:** Containers are portable and can be easily moved between different host machines and cloud providers.
- **Resource Efficiency:** Containers share the host operating system's kernel, which makes them lightweight and efficient in terms of resource utilization.
- **Scalability:** Containers can be quickly scaled up or down to meet changing application demands, making them ideal for microservices architectures.
- **Version Control:** Container images are versioned, enabling easy rollback to previous application states if issues arise.
- **DevOps and CI/CD:** Containerization is a fundamental technology in DevOps and CI/CD pipelines, allowing for automated testing, integration, and deployment.

Containerization vs. Virtualization:

- Containerization differs from traditional virtualization, where a hypervisor virtualizes an entire operating system (VM) to run multiple applications. In contrast:
- Containers share the host OS kernel, making them more lightweight and efficient.
- Containers start faster and use fewer resources than VMs.
- VMs encapsulate an entire OS, while containers package only the application and its dependencies.



Docker

Docker is a tool that helps in developing, building, deploying and executing software in isolation. It does so by creating containers that completely wrap software. Docker is a containerization technology that is widely used for managing application containers. Docker is Simple, Fast, has Easy Collaboration, Built for Developers by Developers and has large Docker Community.

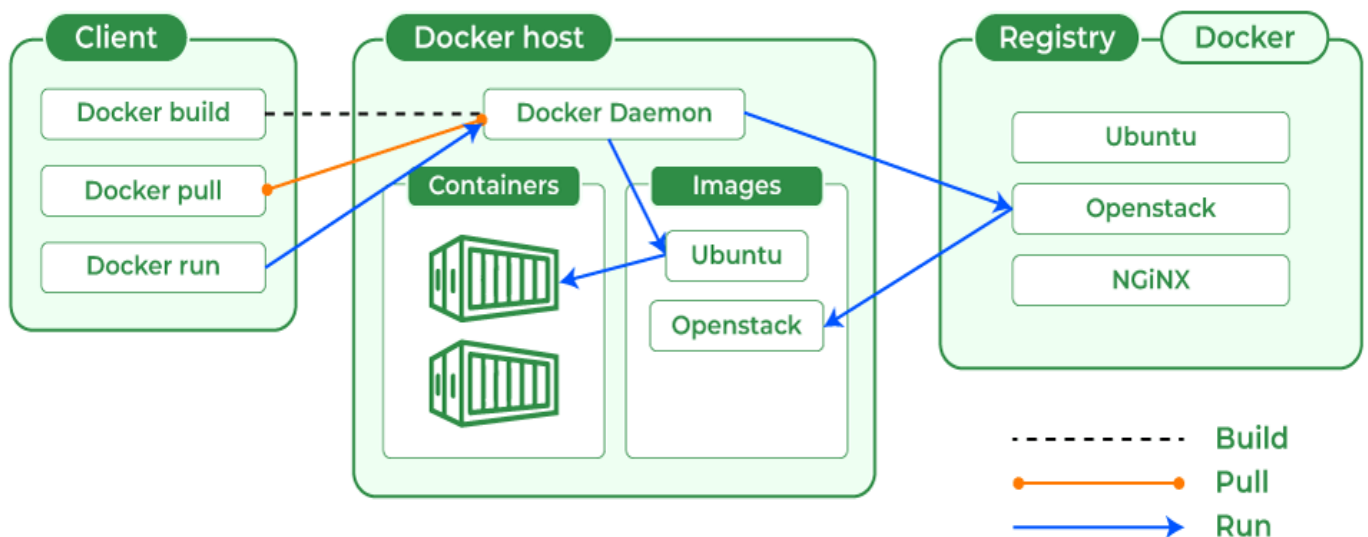
Docker Environment

- **Docker Engine**
 - Docker engine is as the name suggests, it's a technology that allows for the creation and management of all the Docker Processes. It has three major parts to it.
 - Docker CLI
 - Docker API
 - Docker Daemon
- **Docker Objects**
 - When you use Docker, you are creating and using docker objects like

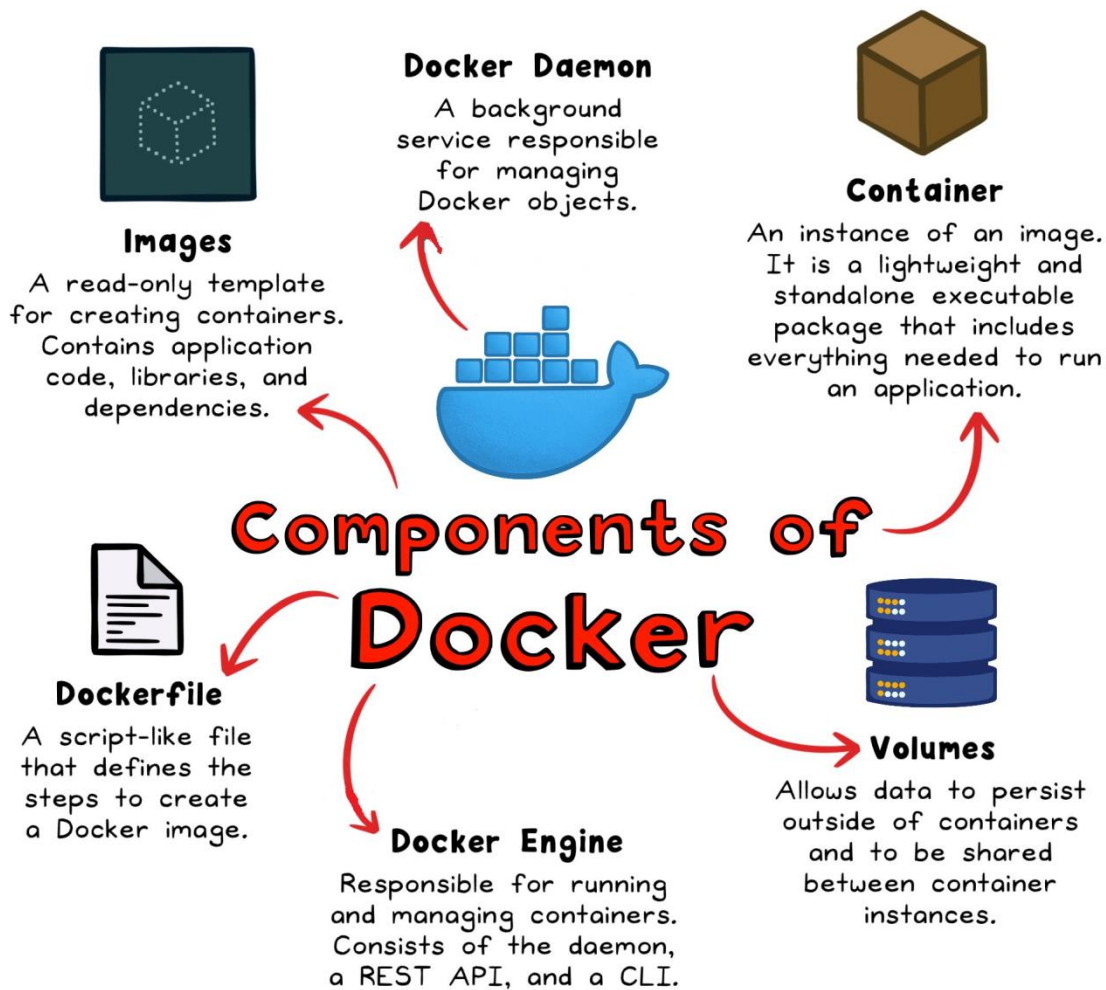
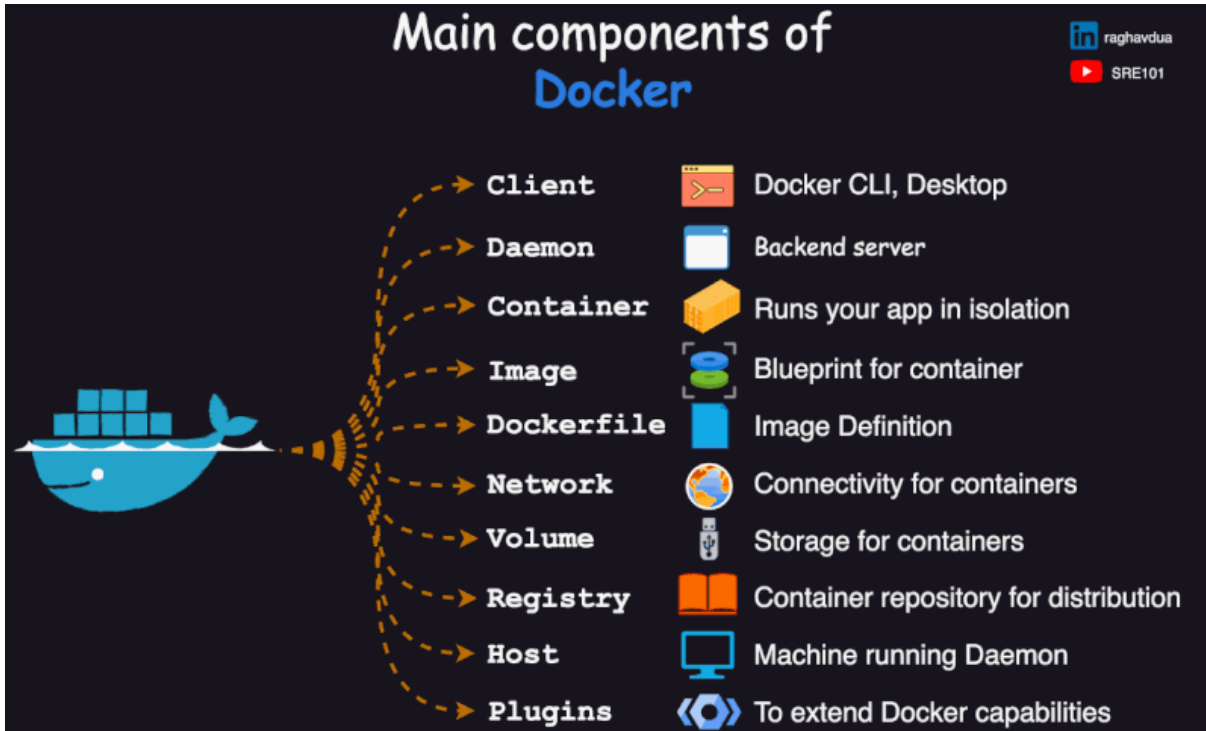
- **Docker Images:** Docker images are sets of instructions that are used to create containers and execute code inside it.
- **Docker Containers :** Wraps code and all its dependencies, libraries, OS into a single deployable unit
- **Docker Volumes:** Storage structure- three types are
 - Volume – storage managed by docker
 - Bind mount – external storage not managed by Docker
 - Tmpfs mount – store data for temporary use and throw.
- **Docker Networks:** provides connection between one or more containers
 - Bridge Network (between 2 containers)
 - Host Network (container and external host)
 - Overlay Network (many to many)
 - None Network (no network – single container)
- **Docker Swarm Nodes & Services**
- **Docker Registry** (Docker Hub)
- **Docker Compose** – Service that lets us launch multiple containers at the same time.
- **Docker Swarm** - an orchestration management tool that runs on Docker applications. It helps end-users in creating and deploying a cluster of Docker nodes. Each node of a Docker Swarm is a Docker daemon, and all Docker daemons interact using the Docker API.

Docker Architecture

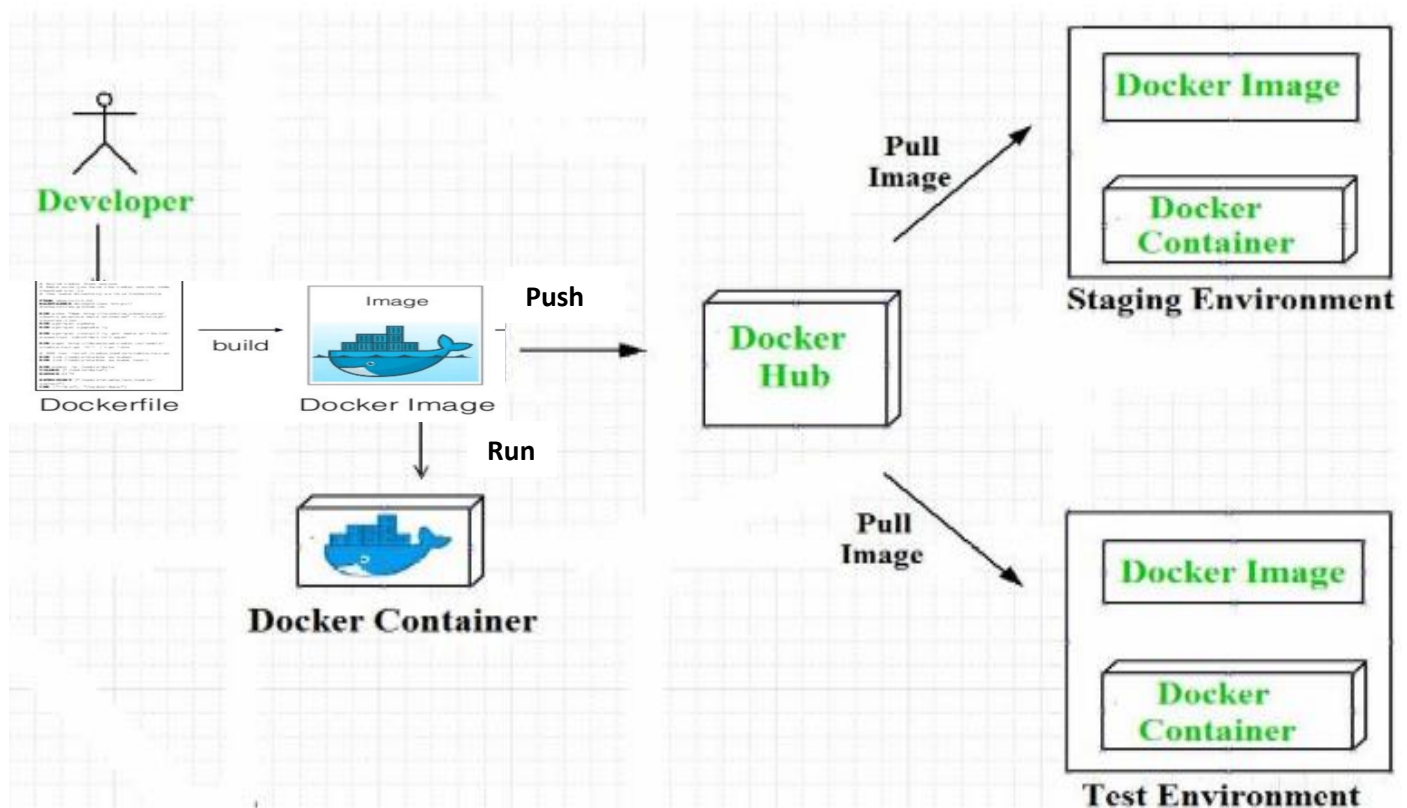
Docker architecture consists of Docker client, Docker Daemon running on Docker Host and DockerHub repository.



Main Components of Docker



Dockerization of a Application



Docker commands for content management:

INSTALLATION

Docker installation steps

```
$ sudo apt-get update
```

```
$ sudo apt-get install docker.io
```

```
$ docker --version
```

Docker Commands

Show the version of the local Docker installation.

```
$ docker version
```

Log in to a Docker registry.

```
$ docker login
```

Log out of a Docker registry.

```
$ docker logout
```

Show low-level information about an object.

```
$ docker inspect [object]
```

Display information about the system.

```
$ docker info
```


Remove unused images, containers, and networks

```
$ docker system prune
```

Container Management Commands

List the running containers.

```
$ docker ps
```

List all the containers, both running and stopped.

```
$ docker ps -a
```

Lists all images stored locally on the host

```
$ docker images
```

Run a command in a container based on an image.

```
$ docker run [image] [command]
```

Create, start, and name a container.

```
$ docker run --name [container-name] [image]
```

Map a host port to a container port.

```
$ docker run -p [host]: [container-port] [image]
```

Run a container and remove it after it stops.

```
$ docker run --rm [image]
```

Run a detached (background) container.

```
$ docker run -d [image]
```

Run an interactive process, e.g., a shell, in a container.

```
$ docker run -it [image]
```

Run a shell inside a running container.

```
$ docker exec -it [container] [shell]
```

Create a container without starting it.

```
$ docker create [image]
```

Create an interactive container with pseudo-TTY.

```
$ docker create -it [image]
```

Docker start: Start one or more stopped containers.

```
$ docker start [mycontainer]
```

Docker stop: Stop one or more running containers.

```
$ docker stop [mycontainer]
```

Stop a container and start it again.

`$ docker restart [container]`

Pause processes in a running container.

`$ docker pause [container]`

Unpause processes in a running container.

`$ docker unpause [container]`

Block input until the container stops.

`$ docker wait [container]`

Send a SIGKILL signal to stop a container.

`$ docker kill [container]`

Attach local standard input, output and error.

`$ docker attach [container]`

Rename a container.

`$ docker rename [container] [new-name]`

Remove a stopped container.

`$ docker rm [container]`

Force remove a container, even if it is running.

`$ docker rm -f [container]`

View logs for a running container.

`$ docker logs [container]`

Copy a local file to a directory in a container

`$ docker cp [file-path] CONTAINER:[path]`

Update the configuration of a container.

`$ docker update [container]`

Show port mapping for a container.

`$ docker port [container]`

Show running processes in a container.

`$ docker top [container]`

Show live resource usage statistics for a container.

`$ docker stats [container]`

Show changes to files or directories on the filesystem.

`$ docker diff [container]`

Image Management Commands

Create an image from a Dockerfile.

```
$ docker build [dockerfile-path]
```

Build an image using the files from the current path.

```
$ docker build .
```

Create an image from a Dockerfile and tag it.

```
$ docker build -t [name]:[tag] [location]
```

Specify a file to build from.

```
$ docker build -f [file]
```

Pull an image from a registry.

```
$ docker pull [image]
```

Push an image to a registry.

```
$ docker push [image]
```

Create an image from a tarball.

```
$ docker import [url/file]
```

Create an image from a container.

```
$ docker commit [container] [new-image]
```

Tag an image.

```
$ docker tag [image] [image]:[tag]
```

Show all locally stored top level images.

```
$ docker images
```

Show history for an image.

```
$ docker history [image]
```

Remove an image.

```
$ docker rmi [image]
```

Load an image from a tar archive file.

```
$ docker load --image [tar-file]
```

Network Management Commands

Save an image to a tar archive file.

```
$ docker save [image] > [tar-file]
```

Search Docker Hub for images.

```
$ docker search [query]
```

Remove unused images

```
$ docker image prune
```

View available networks.

```
$ docker network ls
```

Remove a network.

```
$ docker network rm [network]
```

Show information about a network.

```
$ docker network inspect [network]
```

Connect a container to a network.

```
$ docker network connect [network] [container]
```

Disconnect a container from a network.

```
$ docker network disconnect [network] [container]
```

Programs:

Program #1: Run an Hello-World image in Docker

```
$ docker version //checks docker version
$ docker pull hello-world //pulls hello-world image from docker hub
$ docker run hello-world //runs hello-world image
$ docker ps -a // lists all containers
$ docker images // lists all images
```

Program #2: Run an Ubuntu image in Docker container named “MyContainerA” and execute some shell commands

```
$ docker version //checks docker version
$ docker pull ubuntu:latest //pulls ubuntu image from docker hub
```

```
// runs ubuntu image in container named MyContainerA and opens shell script
```

```
$ docker run -it --name MyContainerA ubuntu -- /bin/bash
```

```
root @[containerid]... $ whoami
```

```
root @... $ ls
```

```
root @... $ mkdir mydir
```

```
root @... $ cd mydir
```

```
root @... $ echo “Hello from MVSR” > sample.txt
```

```
root @... $ ls
```

```
root @... $ pwd
```

```
root @... $ ping www.google.com
```

```
root @... $ exit
```

```
$ docker ps -a // lists all containers
```

```
$ docker images // lists all images
```

```
$ docker logs MyContainerA // log of container
```

Program #3: Run an Mongo image in Docker container named “mongoCon” and execute some mongodb commands

```
$ docker version //checks docker version
$ docker pull mongo:latest //pulls mongo image from docker hub

// runs mongo image in container named mongoCon and opens mongo command prompt

$ docker run -d -p 27017:27014 --name mongoCon mongo:latest
$ docker ps -a // lists all containers
$ docker images // lists all images
$ docker exec -it mongoCon mongosh .
root @[mongoConid]... # show databases;
# db.student.insert( { name : “Sunny”, age: 20, dept : “IT” } );
# db.student.insert( { name : “Sony”, age: 22, gender : “Female”, dept : “IT” } );
# db.student.find();
# db.student.delete( { name : “Sunny” } );
# db.student.find();
# show collections;
# exit
$ docker logs mongoCon // log of container
```


EXPERIMENT NO. 7

AIM: Develop a simple containerized application using Docker.

Program #1: HTML docker image

Steps to create Web-HTML docker image

- Login to super user using sudo su command
 - \$ sudo su
- Create a directory HtmlDemo and create a HTML file and Docker file inside it.
 - \$ mkdir HtmlDemo
 - \$ cd HtmlDemo
 - vi index.html

```
<html>
  <head>
    <title> Hello Page</title>
  </head>
  <body>
    <<h1> Welcome to HTML Docker </h1>
  </body>
</html>
```

- vi Dockerfile

```
FROM nginx:latest
WORKDIR /usr/share/nginx/html
COPY ./index.html .
EXPOSE 80
```

- Build and run the Docker image
 - \$ docker build -t htmlimage .
 - \$ docker run -it htmlimage
- Login to Docker hub and push the image to your account
 - \$ docker login -u [Dockerhubusername]
Eg: \$ docker login -u sowjanyaajindam
 - \$ docker tag imagename Dockerhubusername/imagename
Eg: \$ docker tag htmlimage sowjanyaajindam/htmlimage
 - \$ docker push Dockerhubusername/imagename
Eg: \$ docker push sowjanyaajindam/htmlimage
- Pull the code from Docker hub and execute
 - \$ docker pull sowjanyaajindam/htmlimage
 - \$ docker run -it sowjanyaajindam/htmlimage

Program # 2: JAVA docker image

Steps to create JAVA docker image

- Login to super user using sudo su command
 - \$ sudo su
- Create a directory JavaDemo and create a JAVA file and Docker file inside it.
 - \$ mkdir JavaDemo
 - \$ cd JavaDemo
 - vi Hello.java

```
class Hello
{
    public static void main(String args[])
    {
        System.out.println("Hello Docker from java");
    }
}
```

- vi Dockerfile

```
FROM openjdk:11
WORKDIR /app
COPY ./Hello.java .
RUN javac Hello.java
CMD ["java", "Hello"]
```

- Build and run the Docker image
 - \$ docker build -t javaimage .
 - \$ docker run -it javaimage
- Login to Docker hub and push the image to your account
 - \$ docker login -u [Dockerhubusername]
Eg: \$ docker login -u sowjanyaajindam
 - \$ docker tag imagename Dockerhubusername/imagename
Eg: \$ docker tag javaimage sowjanyaajindam/javaimage
 - \$ docker push Dockerhubusername/imagename
Eg: \$ docker push sowjanyaajindam/javaimage
- Pull the code from Docker hub and execute
 - \$ docker pull sowjanyaajindam/ javaimage
 - \$ docker run -it sowjanyaajindam/ javaimage

Program # 3: Python docker image

Steps to create Python docker image

- Login to super user using sudo su command
 - \$ sudo su
- Create a directory pythonDemo and create a PYTHON file and Docker file inside it.
 - \$ mkdir pythonDemo
 - \$ cd pythonDemo
 - vi Hello.py

```
print("Hello from Python. \n Welcome to Devops Lab")
```

- vi Dockerfile

```
FROM python: latest
WORKDIR /pythonapp
copy ./Hello.py .
CMD ["python", "Hello.py"]
```

- Build and run the Docker image
 - \$ docker build -t pythonimage .
 - \$ docker run -it pythonimage
- Login to Docker hub and push the image to your account
 - \$ docker login -u [Dockerhubusername]
Eg: \$ docker login -u sowjanyaajindam
 - \$ docker tag imagename Dockerhubusername/imagename
Eg: \$ docker tag javaimage sowjanyaajindam/pythonimage
 - \$ docker push Dockerhubusername/imagename
Eg: \$ docker push sowjanyaajindam/pythonimage
- Pull the code from Docker hub and execute
 - \$ docker pull sowjanyaajindam/ pythonimage
 - \$ docker run -it sowjanyaajindam/ pythonimage

Program # 4: Node docker image

Steps to create Node docker image

- Login to super user using sudo su command
 - \$ sudo su
- Create a directory nodeDemo and create a Node file and Docker file inside it.
 - \$ mkdir nodeDemo
 - \$ cd nodeDemo
 - vi hello.js

```
console.log("Hello Docker from node program");
```

- vi Dockerfile

```
FROM node:alpine
WORKDIR /app
COPY . /app
CMD node hello.js
```

- Build and run the Docker image
 - \$ docker build -t nodeimage .
 - \$ docker run -it nodeimage
- Login to Docker hub and push the image to your account
 - \$ docker login -u [Dockerhubusername]
Eg: \$ docker login -u sowjanyaajindam
 - \$ docker tag imagename Dockerhubusername/imagename
Eg: \$ docker tag nodeimage sowjanyaajindam/ nodeimage
 - \$ docker push Dockerhubusername/imagename
Eg: \$ docker push sowjanyaajindam/nodeimage
- Pull the code from Docker hub and execute
 - \$ docker pull sowjanyaajindam/nodeimage
 - \$ docker run -it sowjanyaajindam/nodeimage

Program # 5: Python FLASK docker image

Steps to create Python FLASK docker image

- Login to super user using sudo su command
 - \$ sudo su
- Create a directory flaskDemo and create a python file and Docker file inside it.
 - \$ mkdir flaskDemo
 - \$ cd flaskDemo
 - vi app.py

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')  
def run():
```

```
    return "Hello, World!"
```

```
if __name__ == '__main__':
```

```
    app.run(debug=True, host="0.0.0.0", port=8000)
```

- vi requirements.txt

```
flask
```

- vi Dockerfile

```
FROM python:latest
```

```
WORKDIR /python-flask-docker
```

```
copy . /python-flask-docker
```

```
RUN pip install -r requirements.txt
```

```
EXPOSE 8000
```

```
CMD python ./app.py
```

- Build and run the Docker image
 - \$ docker build -t flaskimage .
 - \$ docker run -it flaskimage
- Login to Docker hub and push the image to your account
 - \$ docker login -u [Dockerhubusername]
Eg: \$ docker login -u sowjanyaajindam
 - \$ docker tag imagename Dockerhubusername/imagename
Eg: \$ docker tag nodeimage sowjanyaajindam/flaskimage
 - \$ docker push Dockerhubusername/imagename
Eg: \$ docker push sowjanyaajindam/flaskimage

Program # 6: Creating multiple containers in single step Using Docker Compose

1. Busy Box : Create two instances of busybox images in two containers named bbConA and bbConB and ping bbConB from bbConA.

Note: BusyBox combines tiny versions of many common Unix utilities into a single small executable. It provides most of the Unix utilities (Eg: ping)

- Login to super user using sudo su command
 - \$ sudo su
- Install docker compose and check version
 - \$ apt-get update
 - \$ apt-get install docker-compose
 - \$ docker-compose version
- Create a directory dockercomposeDemo and create a docker compose yaml file inside it.
 - \$ mkdir dockercomposeDemo
 - \$ cd dockercomposeDemo
 - vi **bb-docker-compose.yaml**

```
version: '3'
services:

  busybox1:
    image: busybox:latest
    container_name: bbConA
    command: sh -c 'ping busybox2'
    links:
      - busybox2

  busybox2:
    image: busybox:latest
    container_name: bbConB
    command: sleep infinity
```

- Build and run the Docker images
 - \$ docker-compose -f bb-docker-compose.yaml up -d

Program # 7: Create MYSQL image Using Docker Compose yaml file and execute some SQL commands inside image

- Login to super user using sudo su command
 - \$ sudo su
- Install docker compose and check version
 - \$ apt-get update
 - \$ apt-get install docker-compose
 - \$ docker-compose version
- Create a directory mysqlDemo and create a docker compose yaml file inside it.
 - \$ mkdir mysqlDemo
 - \$ cd mysqlDemo
 - vi **mysql-docker-compose.yaml**

```
version: "3.7"
services:
  mysql:
    image: mysql:5.7
    container_name: mysql-5.7
    restart: always                                # always restart
    environment:
      MYSQL_DATABASE: 'test'                        # name of database
      MYSQL_USER: 'sample'                          # sample is the name of user
      MYSQL_PASSWORD: 'password'                    # password for sample user
      MYSQL_ROOT_PASSWORD: 'password'               # password for root user
    ports:
      - '3306:3306'    # host port 3306 is mapper to docker port 3306
    expose:
      - '3306'
    volumes:
      - mysql-db:/var/lib/mysql
volumes:
  mysql-db:
```
- Build and run the Docker image
 - \$ docker-compose -f mysql-docker-compose.yaml up -d
 - Open another terminal and type

```
$ mysql -h 127.0.0.1 -P 3306 -u root -p
```

or

```
$ sudo apt-get install mysql-server
```

```
$ mysql -u root -p password
```

- **Execution of some SQL commands inside Docker image**

- Mysql> show databases;
- Mysql> use test;
- Mysql> show tables;
- Mysql> create table student(ID int, NAME varchar(20), BRANCH varchar(20));
- Mysql> insert into student values(1,"RAM", "IT");
- Mysql> insert into student values(1,"SHYAM", "IT");
- Mysql> insert into student values(1,"JOHN", "CSE");
- Mysql> select * from student;
- Mysql> exit

EXPERIMENT NO. 8

AIM: Dockerize the Student Registration Application from the Program 1 and push the code to GITHUB and DOCKERHUB

Program:

Dockerization Steps:

- Login to super user using sudo su command
 - \$ sudo su
- Check whether docker and docker compose are installed.
 - \$ docker version
 - \$ docker-compose version
- Create a directory StudReg and code file, docker and docker compose yaml file inside it.
 - \$ mkdir StudReg
 - \$ cd StudReg
 - Add streg.jpg to the directory
 - **vi stud.html**

```
<html>
<head>
  <link href="studreg.css" rel="stylesheet" />
</head>

<body>

<h1> DevOps Lab</h1>

<h2> Student Registration Form</h2>

<form action = "http://127.0.0.1:8081/process_get" method = "GET">

<table border="5" align="center" cellpadding="10" cellspacing="10" >

<tr>
<td>Name</td><td><input type="text" name="sname"></td>
</tr>

<tr>
<td>Contact Number</td>
<td><input type="text" name="scon"></td>
</tr>

<tr>
<td>Gender</td>
<td><input type="radio" name="g">Male
      <input type="radio" name="g">Female</td>
</tr>

<tr>
```

```

<td>Address</td>
<td><textarea rows="5" cols="15" name="sadd"></textarea></td>
</tr>

<tr>
<td>Hobbies</td>
<td><input type="checkbox" name="shob">Singing
      <input type="checkbox" name="shob">Travelling
      <input type="checkbox" name="shob">Reading novels
    </td>
</tr>

<tr>
<td>Skillset</td>
<td><input type="checkbox" name="sss">C
      <input type="checkbox" name="sss">Python
      <input type="checkbox" name="sss">Java
    </td>
</tr>

<tr>
<td>Highest Qualification</td>
<td><select name="shq">
      <option><--SELECT--></option>
      <option>Ph.D</option>
      <option>M.E/M.Tech</option>
      <option>B.E/B.Tech</option>
      <option>Diploma</option>
      <option>Inter</option>
      <option>SSC</option>
    </select>
    </td>
</tr>
<td>District</td>
<td><select name="sdis">
      <option><--SELECT--></option>
      <option>Adilabad</option>
      <option>Zaheerabad</option>
    </select>
    </td>
</tr>

<tr>
<td><input type="submit" name="submit" value="Register"></td>
<td><input type="reset" value="Clear"></td>
</tr>

</table>
</form>
</body>
</html>

```

vi studreg.css

```

h1{
  color:green;
  text-align:center;
}

```

```

    }

h2{
    color:blue;
    text-align:center;
}

p{
    color:red;
}

table{
    background-color: cyan;
}
td{
    color:red;
    font-size:24px;
}
input
{
    color:blue;
    font-size:24px;
    text-align : center;
}
select
{
    color:blue;
    font-size:24px;
    text-align : center;
}
.center {
    display: block;
    margin-left: auto;
    margin-right: auto;
    width: 50%;
}

```

vi studregnode.js

```

var express = require('express');
var app = express();

app.use(express.static('public'));
app.get('/studreg.html', function (req, res) {
    res.sendFile( __dirname + "/" + "studreg.html" );
})

app.get('/process_get', function (req, res) {
    // Prepare output in JSON format
    response = {
        stud_name:req.query.sname,
        stud_contact:req.query.scon,
        stud_gender:req.query.g,
        stud_address:req.query.sadd,
        stud_hobbies:req.query.shob,
        stus_skillset:req.query.sss,
        stud_highest_qualification:req.query.shq,
        stud_district:req.query.sdis
    };
    console.log(response);
    res.end(JSON.stringify(response));
}

```

```

}))

var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)
})

```

vi Dockerfile

```

FROM node:14
WORKDIR /app
COPY package*.json .
RUN npm install
RUN npm install express
COPY . /app
EXPOSE 8081
CMD ["node", "studregnode.js"]

```

vi studreg-docker-compose.yaml

```

version: "3.0"
services:
  myweb:
    image: node:14
    build: .

    container_name: nodecons
    restart: always
    ports:
      - "8081:8081"
    expose:
      - "8081"

```

- Execute the program in local environment
 - \$ apt-get update
 - \$ apt install nodejs
 - \$ apt install npm
 - \$ npm install -g express
 - \$ node -v
 - \$ npm -v
 - \$ npm init
 - \$ node studregnode.js
- Build and run the Docker image

- \$ docker build -t studregimage .
- \$ docker run --rm -it -p 8081:8081 --name nodecon studregimage
- **Go to `https://localhost:8081/stud.html` to see the output**
- Build and run the Docker images using Docker Compose
 - \$ docker-compose -f studreg-docker-compose.yaml up -d
- Login to GITHUB and create a new GIT repository name **StudRegRepo** and copy the URL
- Push the code to GITHUB
 - \$ git --version
 - \$ git config --global user.name "sowjanya"
 - \$ git config --global user.email sowjanya@gmail.com
 - \$ git remote add origin <https://github.com/sowjanya/StudRegRepo.git>
 - \$ git config --list
 - \$ git init
 - \$ git status
 - \$ git add .
 - \$ git commit -m "1st commit files added"
 - \$ git branch -M main
 - \$ git status
 - \$ git push -u origin main

Go to remote repository and check whether project is uploaded on github.

- Login to Docker hub and push the image to your account
 - \$ docker login -u [Dockerhubusername]
 - Eg: \$ docker login -u sowjanyaajindam
 - \$ docker tag imagename Dockerhubusername/imagename
 - Eg: \$ docker tag nodeimage sowjanyaajindam/studregimage
 - \$ docker push Dockerhubusername/imagename
 - Eg: \$ docker push sowjanyaajindam/studregimage
- Pull the code from Docker hub and execute
 - \$ docker pull sowjanyaajindam/studregimage
 - \$ docker run -it sowjanyaajindam/studregimage

Go to `https://localhost:8081/stud.html` to see the output

EXPERIMENT NO. 9

AIM: Integrate Kubernetes and Docker

Introduction:

Container orchestration is a critical component in modern application deployment, allowing you to manage, scale, and maintain containerized applications efficiently.

Kubernetes is a popular container orchestration platform that automates many tasks associated with deploying, scaling, and managing containerized applications.

Kubernetes, often abbreviated as K8s, is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. Developed by Google and later donated to the Cloud Native Computing Foundation (CNCF), Kubernetes has become the de facto standard for container orchestration in modern cloud-native application development.

Key Concepts in Kubernetes:

- **Containerization:** Kubernetes relies on containers as the fundamental unit for packaging and running applications. Containers encapsulate an application and its dependencies, ensuring consistency across various environments.
- **Cluster:** A Kubernetes cluster is a set of machines, known as nodes, that collectively run containerized applications. A cluster typically consists of a master node (for control and management) and multiple worker nodes (for running containers).
- **Nodes:** Nodes are individual machines (virtual or physical) that form part of a Kubernetes cluster. Nodes run containerized workloads and communicate with the master node to manage and orchestrate containers.
- **Pod:** A pod is the smallest deployable unit in Kubernetes. It can contain one or more tightly coupled containers that share the same network and storage namespace. Containers within a pod are typically used to run closely related processes.
- **Deployment:** A Deployment is a Kubernetes resource that defines how to create, update, and scale instances of an application. It ensures that a specified number of replicas are running at all times.
- **Service:** A Service is an abstraction that exposes a set of pods as a network service. It provides a stable IP address and DNS name for accessing the pods, enabling load balancing and discovery.
- **Namespace:** Kubernetes supports multiple virtual clusters within the same physical cluster, called namespaces. Namespaces help isolate resources and provide a scope for organizing and managing workloads.

Key Features of Kubernetes:

- **Automated Scaling:** Kubernetes can automatically scale the number of replicas of an application based on resource usage or defined metrics. This ensures applications can handle varying workloads efficiently.
- **Load Balancing:** Services in Kubernetes can distribute traffic among pods, providing high availability and distributing workloads evenly.
- **Self-healing:** Kubernetes monitors the health of pods and can automatically restart or replace failed

instances to maintain desired application availability.

- **Rolling Updates and Rollbacks:** Kubernetes allows for controlled, rolling updates of applications, ensuring zero-downtime deployments. If issues arise, rollbacks can be performed with ease.
- **Storage Orchestration:** Kubernetes provides mechanisms for attaching storage volumes to containers, enabling data persistence and sharing.
- **Configuration Management:** Kubernetes supports configuration management through ConfigMaps and Secrets, making it easy to manage application configurations.
- **Extensibility:** Kubernetes is highly extensible, with a vast ecosystem of plugins and extensions, including Helm charts for packaging applications and custom resources for defining custom objects.

Kubernetes has become a cornerstone of cloud-native application development, enabling organisations to build, deploy, and scale containerized applications effectively. Its ability to abstract away infrastructure complexities, ensure application reliability, and provide consistent scaling makes it a powerful tool for modern software development and operations.

Important K8S resources:

Namespaces ,Pods, deployments, daemonsets, statefulsets, replicaset, ConfigMaps, Secrets, Service, Ingress, endpoints

Minikube

Minikube is a tool that sets up a Kubernetes environment on a local machine. By default, it creates a one-node cluster, but you can create a multi-node cluster with a Minikube environment if desired.

Minikube supports a flexible deployment mode based on the concept of “drivers.” Depending on the driver you choose for deployment, your Kubernetes node, or nodes, can run in one of several ways:

- Inside a Docker container.
- Inside a virtual machine. Several VM platforms are supported, such as VirtualBox, KVM, Hyper-V, and Parallels, depending on which host operating system you use.
- Directly on bare metal, if your host OS supports it.

For configuring minikube drivers we need Container or virtual machine manager, such as: Docker, QEMU, Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox, or VMware Fusion/Workstation

Program:

Installation of kubernetes

```
sudo su
docker version ( install docker if not available)
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64
minikube version
minikube start --driver=docker
minikube kubectl -- get po -A
snap install kubectl --classic
kubectl version
  minikube start --force
  minikube addons enable ingress
minikube start --nodes 2 -p multinode-demo
minikube dashboard
```

Kubernetes Commands:

Cluster Management

- Display the Kubernetes version running on the client and server
`kubectl version`
- Display endpoint information about the master and services in the cluster
`kubectl cluster-info`
- Get the configuration of the cluster
`kubectl config view`
- List the API resources that are available
`kubectl api-resources`
- List the API versions that are available
`kubectl api-versions`
- List everything
`kubectl get all --all-namespaces`

Namespaces

- Create namespace <name>
`kubectl create namespace <namespace_name>`
- List one or more namespaces
`kubectl get namespace <namespace_name>`
- Display the detailed state of one or more namespace
`kubectl describe namespace <namespace_name>`

- Delete a namespace
`kubectl delete namespace <namespace_name>`
- Edit and update the definition of a namespace
`kubectl edit namespace <namespace_name>`
- Display Resource (CPU/Memory/Storage) usage for a namespace
`kubectl top namespace <namespace_name>`

Nodes

- List one or more nodes
`kubectl get node`
- Delete a node or multiple nodes
`kubectl delete node <node_name>`
- Display Resource usage (CPU/Memory/Storage) for nodes
`kubectl top node`
- Resource allocation per node
`kubectl describe nodes | grep Allocated -A 5`
- Pods running on a node
`kubectl get pods -o wide | grep <node_name>`
- Add or update the labels of one or more nodes
`kubectl label node`

Pods

- List one or more pods
`kubectl get pod`
`kubectl get pods -o wide`
`kubectl get pods -o=yaml`
`kubectl get pods -o=json`
`kubectl get pods -n=[namespace_name]`
- Delete a pod
`kubectl delete pod <pod_name>`
- Display the detailed state of a pods
`kubectl describe pod <pod_name>`
- Create a pod
`kubectl create pod <pod_name>`
- Execute a command against a container in a pod
`kubectl exec <pod_name> -c <container_name> <command>`
- Get interactive shell on a a single-container pod
`kubectl exec -it <pod_name> /bin/sh`
- Display Resource usage (CPU/Memory/Storage) for pods

```
kubectl top pod
```

- Add or update the annotations of a pod
`kubectl annotate pod <pod_name> <annotation>`
- Add or update the label of a pod
`kubectl label pod <pod_name>`

Deployments

- List one or more deployments
`kubectl get deployment`
- Display the detailed state of one or more deployments
`kubectl describe deployment <deployment_name>`
- Edit and update the definition of one or more deployment on the server
`kubectl edit deployment <deployment_name>`
- Create one a new deployment
`kubectl create deployment <deployment_name>`
- Delete deployments
`kubectl delete deployment <deployment_name>`
- See the rollout status of a deployment
`kubectl rollout status deployment <deployment_name>`

ReplicaSets

- List ReplicaSets
`kubectl get replicaset`
- Display the detailed state of one or more ReplicaSets
`kubectl describe replicaset <replicaset_name>`
- Scale a ReplicaSet
`kubectl scale --replicas=[x]`

Logs

- Print the logs for a pod
`kubectl logs <pod_name>`
- Print the logs for the last hour for a pod
`kubectl logs --since=1h <pod_name>`
- Get the most recent 20 lines of logs
`kubectl logs --tail=20 <pod_name>`
- Get logs from a service and optionally select which container
`kubectl logs -f <service_name> [-c <$container>]`

- Print the logs for a pod and follow new logs
`kubectl logs -f <pod_name>`
- Print the logs for a container in a pod
`kubectl logs -c <container_name> <pod_name>`
- Output the logs for a pod into a file named 'pod.log'
`kubectl logs <pod_name> pod.log`
- View the logs for a previously failed pod
`kubectl logs --previous <pod_name>`

Manifest Files

- Apply a configuration to an object by filename or stdin. Overrides the existing configuration.
`kubectl apply -f manifest_file.yaml`
- Create objects
`kubectl create -f manifest_file.yaml`
- Create objects in all manifest files in a directory
`kubectl create -f ./dir`
- Create objects from a URL
`kubectl create -f 'url'`
- Creating a pod using data in a file named newpod.json.
`kubectl create -f ./newpod.json`
- Delete an object
`kubectl delete -f manifest_file.yaml`

Secrets

- Create a secret
`kubectl create secret`
- List secrets
`kubectl get secrets`
- List details about secrets
`kubectl describe secrets`
- Delete a secret
`kubectl delete secret <secret_name>`

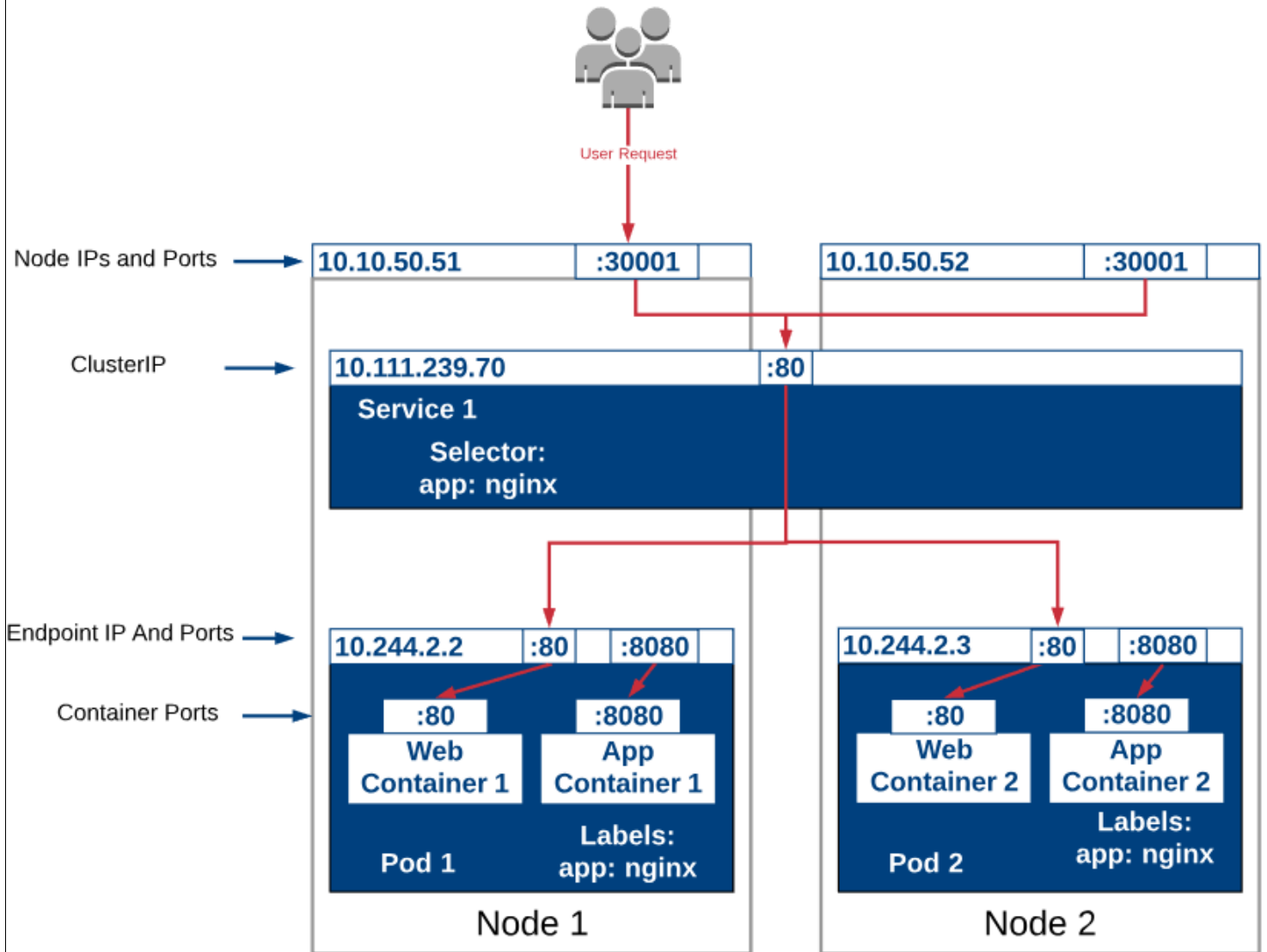
Services

- List one or more services
`kubectl get services`
- Display the detailed state of a service
`kubectl describe services`

- Expose a replication controller, service, deployment or pod as a new Kubernetes service
`kubectl expose deployment [deployment_name]`
- Edit and update the definition of one or more services
`kubectl edit services`

There are four types of services that Kubernetes supports: ClusterIP, NodePort, LoadBalancer, Ingress

Workflow in Kubernetes



EXPERIMENT NO. 10

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

Program # 1: Create an nginx and mongo deployment and run some kubectl commands

Step 1: create nginx deployment and check status

- `$ kubectl create deployment nginx-depl --image=nginx`
- `$ kubectl get deployments`
- `$ kubectl get nodes`
- `$ kubectl get pods`
- `$ kubectl get replicaset`

Step 2: Execute the deployment and run some commands in nginx

- `$ kubectl get pods` (copy pod id and use it in next command to run the pod)
- `$ kubectl exec -it nginx-depl-85zrbt9 -- /bin/bash`
- `root@nginxid..$ ls`
- `root@nginxid..$ mkdir mydir`
- `root@nginxid..$ echo "Hello nginx" > sample.txt`
- `root@nginxid..$ ls`
- `root@nginxid..$ exit`

Step 3: Edit deployment, change replicaset and check log

- `$ kubectl edit deployment nginx-depl`
(in opened file make some changes to version, replicas, namespace...)
- `$ kubectl get replicaset`
- `$ kubectl scale deployment/nginx-depl --replicas=5`
- `$ kubectl get replicaset`
- `$ kubectl get pods`
- `$ kubectl logs [nginx-depl pod ID]`

Step 4: create mongo deployment and check status

- `$ kubectl create deployment mongo-depl --image=mongo`
- `$ kubectl get deployments`
- `$ kubectl get pods`
- `$ kubectl get replicaset`

Step 5: Execute the mongo deployment and run some commands in mongo

- `$ kubectl get pods` (copy pod id and use it in next command to run the pod)
- `$ kubectl exec -it mongo-depl-85shtudf -- /bin/bash`
- `root@mongo-depl-id..# show databases;`
`# db.student.insert({ name : "Sunny", age: 20, dept : "IT" });`

```
# db.student.insert( { name : "Sony", age: 22, gender : "Female", dept : "IT" } );
# db.student.find();
# db.student.delete( { name : "Sunny" } );
# db.student.find();
# show collections;
# exit
```

Step 6: Delete deployments

- \$ kubectl delete deployment nginx-depl
- \$ kubectl get pods
- \$ kubectl delete deployment mongo-depl
- \$ kubectl get pods

Program # 2: Create an nginx deployment using manifest (YAML) files

Step 1: create a yaml file for nginx deployment and apply it

- \$ touch nginx-depl.yaml
- \$ vi nginx-depl.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

- \$ kubectl apply -f nginx-depl.yaml
- \$ kubectl get deployments
- \$ kubectl get pods
- \$ kubectl get replicaset
- \$ kubectl describe deployment nginx-deployment

Or

- \$ kubectl create namespace myspace
- \$ kubectl apply -f nginx-depl.yaml -n=myspace
- \$ kubectl get deployments -n=myspace

Program # 3: Run a containerized application (Docker) using Kubernetes.

Step 1: Create a Dockerized Web Application

- Create a simple web application (e.g., a static HTML page) or use an existing one.
 - Login to super user using sudo su command
 - \$ sudo su
 - Create a directory HtmlDemo and create a HTML file and Docker file inside it.
 - \$ mkdir kubernestesDemo
 - \$ cd kubernestesDemo
 - vi index.html

```
<html>
  <head>
    <title> Hello Page</title>
  </head>
  <body>
    <<h1> Welcome to HTML Docker run using Kubernetes</h1>
  </body>
</html>
```

- Create a Dockerfile to package the web application into a Docker container.
 - vi Dockerfile

```
FROM nginx:latest
WORKDIR /usr/share/nginx/html
COPY ./index.html .
EXPOSE 80
```

- Build and run the Docker image
 - \$ docker build -t my-web-app .
 - \$ docker run -it my-web-app

Step 2: Deploy the Web Application with Kubernetes

- Create a Kubernetes Deployment YAML file (web-app-deployment.yaml) to deploy the web application:

web-app-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-web-app-deployment
spec:
  replicas: 3 # Number of pods to create
  selector:
    matchLabels:
      app: my-web-app # Label to match pods
  template:
    metadata:
      labels:
        app: my-web-app # Label assigned to pods
    spec:
      containers:
        - name: my-web-app-container
          image: my-web-app:latest # Docker image to use
          ports:
            - containerPort: 80 # Port to expose
```

Step 3: Deploy the Application

- Apply the deployment configuration to your Kubernetes cluster:

```
$ kubectl apply -f web-app-deployment.yaml
```

Step 4: Verify the Deployment

- Check the status of your pods:
\$ kubectl get deployments
\$ kubectl get pods
\$ kubectl describe deployment my-web-app-deployment

EXPERIMENT NO. 11

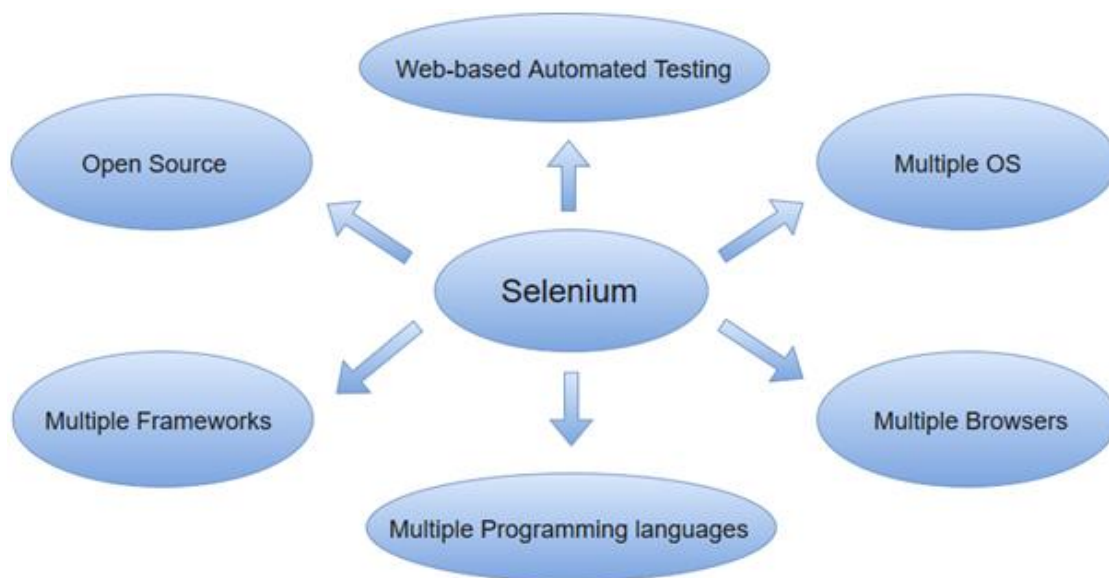
AIM: Install and Explore Selenium for automated testing.

Introduction to Selenium Automation Testing:

Selenium is one of the most widely used open source Web UI (User Interface) automation testing suite. It was originally developed by Jason Huggins in 2004 as an internal tool at Thought Works. Selenium supports automation across different browsers, platforms and programming languages.

Selenium can be easily deployed on platforms such as Windows, Linux, Solaris and Macintosh. Moreover, it supports OS (Operating System) for mobile applications like iOS, windows mobile and android.

Selenium supports a variety of programming languages through the use of drivers specific to each language. Languages supported by Selenium include C#, Java, Perl, PHP, Python and Ruby. Currently, Selenium Web driver is most popular with Java and C#. Selenium test scripts can be coded in any of the supported programming languages and can be run directly in most modern web browsers. Browsers supported by Selenium include Internet Explorer, Mozilla Firefox, Google Chrome and Safari.



Selenium can be used to automate functional tests and can be integrated with automation test tools such as **Maven, Jenkins, & Docker** to achieve continuous testing. It can also be integrated with tools such as **TestNG, & JUnit** for managing test cases and generating reports.

One disadvantage of Selenium automation testing is that it works only for web applications, which leaves desktop and mobile apps out in the cold.

Selenium consists of a set of tools that facilitate the testing process.



Fig: Selenium suite

1. Selenium IDE

Shinya Kasatani developed the Selenium Integrated Development Environment (IDE) in 2006. Conventionally, it is an easy-to-use interface that records the user interactions to build automated test scripts. It is a Firefox or Chrome plugin, generally used as a prototyping tool. It was mainly developed to speed up the creation of automation scripts.

IDE ceased to exist in August 2017 when Firefox upgraded to the new Firefox 55 version, which no longer supported Selenium IDE. Applitools rewrote the old Selenium IDE and released a new version recently. The latest version came with several advancements, such as:

- Reusability of test scripts
- Debugging test scripts
- Selenium side runner
- Provision for control flow statements
- Improved locator functionality

Installing IDE:

- Step 1- Open the Firefox browser
- Step 2- Click on the menu in the top right corner
- Step 3- Click on Add-ons in the drop-down box.
- Step 4- Click on Find more add-ons and type “Selenium IDE”
- Step 5- Click on Add to Firefox

2. Selenium Remote Control (RC)

Paul Hammant developed Selenium Remote Control. Selenium RC is a server written in Java that makes provision for writing application tests in various programming languages like Java, C#, Perl, PHP, Python, etc. The RC server accepts commands from the user program and passes them to the browser as Selenium-Core JavaScript commands.

3. Selenium WebDriver

Developed by Simon Stewart in 2006, Selenium WebDriver was the first cross-platform testing framework that could configure and control the browsers on the OS level. It served as a programming interface to create and run test cases.

Unlike Selenium RC, WebDriver does not require a core engine like RC and interacts natively with the browser applications. WebDriver also supports various programming languages like Python, Ruby, PHP, and Perl. It can also be integrated with frameworks like TestNG and JUnit for Selenium automation testing management.

Steps for Selenium WebDriver installation :

1. Download and Install Java 8 or higher version - Install the latest version of the Java development kit.
2. Download and configure Eclipse or any Java IDE of your choice
3. Download Selenium WebDriver Java Client
 1. Navigate to the official Selenium page.
 2. Scroll down through the web page and locate Selenium Client and WebDriver Language Bindings.
 3. Click on the "Download" link of Java Client Driver . Unzip the file in a directory. It consists of the Jar files required to configure Selenium WebDriver in the IDE.
 4. Download the Browser driver - The automation scripts must be compatible with any browser. Every browser supported by Selenium comes with its driver files.
 5. Configure Selenium WebDriver - The final step is to configure the Selenium WebDriver with the Eclipse IDE. In simple terms, we create a new Java project to build our test script.

4. Selenium Grid

Selenium Grid was developed by Patrick Lightbody to minimize the execution time of Selenium automation testing. Selenium Grid allows the parallel execution of tests on different browsers and different operating systems, facilitating parallel execution. Grid is exceptionally flexible and is integrated with other suite components for simultaneous performance.

The Grid consists of a hub connected to several nodes. It receives the test to be executed along with information about the operating system and browser to be run on. The Grid then picks a node that conforms to the requirements (browser and platform) and passes the test to that node. The node now runs the browser and executes the Selenium commands within it.

EXPERIMENT NO. 12

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

Program # 1: Write a JavaScript program to open Google website in the browser

Step 1: Create directory and add file.

- \$ mkdir googleDemo
- \$ cd googleDemo
- vi app.js

```
const {Builder, By, Key} = require("selenium-webdriver");

async function example(){
    let driver = await new Builder().forBrowser("chrome").build();
    await driver.get("https://www.google.com/");
    console.log("browser opened");
    await driver.quit();
}
example()
```

Step 2: Initialize the project and execute it

Execution Steps for Selenium:

- node -v
// check whether node is installed. If not, install using below commands.
// sudo apt-get update
//sudo apt install nodejs
- npm -v
// check whether npm is installed. If not, install using below commands.
//sudo apt install npm
- npm init // Initilaze the node package
- npm install selenium-webdriver // add selenium web driver as dependency
- npm init //check out for addition of selenium dependency
- node app.js //execute the program

Note: Selenium web drivers run only in Higher versions of node, so see to that node version 16 and above are installed.

For installation of higher version of node

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash
source ~/.bashrc
nvm list-remote
nvm install v16.14.0
```


Program # 2: Write a JavaScript program for login validation application and test it using Selenium.

Step 1: Create directory and add file.

- \$ mkdir myloginDemo
- \$ cd myloginDemo
- **vi login.html**

```
<html>
<head>
<title> Login Page</title>
<script language="javascript">

function validate()
{
var u=document.f1.u.value;
var p=document.f1.p.value;

if(u=="MVSREC" && p=="ITD")
{
window.open("loginsuccess.html");
}
else
{
window.open("loginfail.html");
}
}
</script>
</head>
<body>
<form name="f1">
<h1 align="center" style="color:blue">Login Page</h1>
<table align="center" bgcolor="pink">
<tr>
<td>UserId</td>
<td><input type="text" name="u" id="un"></td>
</tr>
<tr>
<td>Password</td>
<td><input type="password" name="p" id="pw"></td>
</tr>
<tr>
<td><input type="button" value="Signin" id="s"
onclick="validate()"></td>
<td><input type="reset" value="Reset id="r"></td>
</tr>
</table>
</form>
</body>
</html>
```

- **vi loginsucess.html**

```
<html>
<head>
<title> Success </title>
```

```

</head>
<body>
<h1 align="center" style="color:red"> Login Succuess</h1>
</body>
</html>

```

- **vi loginfail.html**

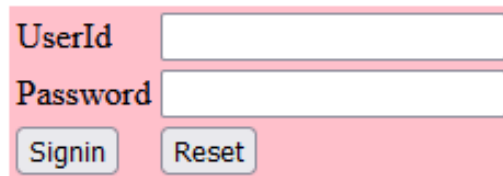
```

<html>
<head>
<title> Fail </title>
</head>
<body>

<h1 align="center" style="color:red"> Login Failed</h1>
</body>
</html>

```

Login Page



- **vi mylogin.js**

```

const { Builder, By, until } = require("selenium-webdriver");
const assert = require("assert");

async function loginTest() {

    // launch the browser

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

    try {

        await driver.get("file:///home/mvsr/myloginDemo/login.html");

        await driver.findElement(By.id("un")).sendKeys("MVSREC");

        await driver.findElement(By.id("pw")).sendKeys("ITD");

        await driver.findElement(By.id("s")).click();

        const title = await driver.getTitle();

        assert.strictEqual(title, "Login Page");
        console.log("success");

    } finally {
        await driver.quit();
    }
}

loginTest();

```

Step 2: Initialize the project and execute it

Execution Steps for Selenium:

- `node -v`
// check whether node is installed. If not, install using below commands.
// `sudo apt-get update`
// `sudo apt install nodejs`
- `npm -v`
// check whether npm is installed. If not, install using below commands.
// `sudo apt install npm`
- `npm init` // Initilaze the node package
- `npm install selenium-webdriver` // add selenium web driver as dependency
- `npm init` //check out for addition of selenium dependency
- `node mylogin.js` //execute the program

Program # 3: Write a JavaScript program for checking exam results on college website (<https://results.mvsrec.edu.in>) and test it using Selenium.

Step 1: Create directory and add file.

- `$ mkdir collegelloginDemo`
- `$ cd collegelloginDemo`
- **`vi collegellogin.js`**

```
const { Builder, By, until } = require("selenium-webdriver");
const assert = require("assert");

async function loginTest() {

    // launch the browser
    let driver = await new Builder().forBrowser("chrome").build();

    try {

        await driver.get("http://results.mvsrec.edu.in/SBLogin.aspx");

        await driver.findElement(By.id("txtUserName")).sendKeys("245121737129");

        await driver.findElement(By.id("txtPassword")).sendKeys("245121737129");

        await driver.findElement(By.id("btnSubmit")).click();

        const user = await driver.findElement(By.id("lblHTNo")).getText();

        assert.strictEqual(user, "245121737129");
        console.log("success");
    }
}
```

```

        await driver.findElement(By.id("Stud_cpModules_imgbtnExams")).click();

        await driver.findElement(By.id("cpBody_lnkSem")).click();

        const ur = await driver.getCurrentUrl();

        assert.strictEqual(ur,
            "http://results.mvsrec.edu.in/STUDENTLOGIN/Frm_SemwiseStudMarks.aspx");

        console.log("Display marks success");
    }
    finally {
        await driver.quit();
    }
}
loginTest();

```

Step 2: Initialize the project and execute it

Execution Steps for Selenium:

- node -v
// check whether node is installed. If not, install using below commands.
// sudo apt-get update
//sudo apt install nodejs
- npm -v
// check whether npm is installed. If not, install using below commands.
//sudo apt install npm
- npm init // Initilaze the node package
- npm install selenium-webdriver // add selenium web driver as dependency
- npm init //check out for addition of selenium dependency
- node collegelogin.js //execute the program

EXPERIMENT NO. 13

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

Program # 1: Write a Java program in eclipse IDE to open Google website in the browser

Step 1: Download selenium-server-standalone-3.141.59.jar and chromedriver suitable for you chrome browser

Step 2: Open Eclipse IDE and create a new Project

- Provide a project name and select the JRE that you wish to use. It is advisable to use the default JRE. Select it and click on finish.
- On the project name, right click and add new package (loginval)
- Now, right click on package name and add new java class.(Loginval)
- To do that, Right-click on Src folder>>New>>Class
- Write the following code.

```
package loginval;

import org.openqa.selenium.*;
import org.openqa.selenium.chrome.*;

public class Loginval {

    public static void main(String[] args) throws Exception{

        System.setProperty("webdriver.chrome.driver","//home/mvsr/seleniumlib/chromedriver");

        WebDriver driver = new ChromeDriver();

        driver.get("http://www.google.com/");

        Thread.sleep(5000); // Let the user actually see something!

        WebElement searchBox = driver.findElement(By.name("q"));

        searchBox.sendKeys("ChromeDriver");

        searchBox.submit();

        Thread.sleep(5000); // Let the user actually see something!

        driver.quit();

    }

}
```

Step 3: Add JAR files

- The next and most crucial step is to add the downloaded JAR file [Step 1].

- To add JAR file, right-click on the Project>>Build Path>>Configure Build Path.
- Select libraries and then Add External JARs.
- Open the folders in which you've saved your JAR files and select the executable JAR file. Click on open to add it.
- Click on the libs folder>> Select the files>> Open
- Once you're done adding the library files, click on Apply and Close.

Step 4: Run the Java Program

- Click on Run>>Run As>>Java Application >> Loginval

Step 5: Output

- Chrome browser should open automatically and navigate to www.google.com and search for the word chrome driver and display results.

EXPERIMENT NO. 14

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

Case Study #1: Develop a Web-Application for Calculator, Write test cases for different operations and test it using Selenium. Dockerize the program and push the containerized application code to GitHub and DockerHub.

Program: Steps

- Login to super user using sudo su command
 - \$ sudo su
- Check whether Git, node and docker are installed.
 - \$ git version
 - \$ docker version
 - \$ node -v
 - \$ npm -v
- Create a directory StudReg and code file, docker and docker compose yaml file inside it.
 - \$ mkdir Calculator
 - \$ cd Calculator
 - **vi index.html**

```
<html>
<script language="javascript">
var prev,oper;
function num(s)
{
var n=parseInt(document.f1.t1.value);
var sum=(n*10)+s;
document.f1.t1.value=sum;
}

function op(k)
{
oper=k;
prev=parseInt(document.f1.t1.value);
document.f1.t1.value=0;
}
function mod()
{
a=parseInt(document.f1.t1.value);
b=Math.floor(a/10);
document.f1.t1.value=b;
}

function eq()
{
curr=parseInt(document.f1.t1.value);
switch(oper)
```

```

{
case 1: var result=prev+curr;break;

case 2: var result=prev-curr;break;

case 3: var result=prev*curr;break;

case 4: var result=prev/curr;break;

case 5: var result=prev%curr;break;
}

document.fl.tl.value=result;
}
</script>

<body >
<form name="f1">
<h1 align="center">Basic Calculator</h1>

<table align="center" bgcolor="cyan" cellspacing="10" cellpadding="20">

<tr><td colspan="4"><input id = "res" type="text" name = "tl" value="0"
size="50"></td>

<tr><td><input id = "1" type="button" value=" 1 " onclick="num(1)">
<td><input id = "2" type="button" value=" 2 " onclick="num(2)">
<td><input id = "3" type="button" value=" 3 " onclick="num(3)">
<td><input id = "add" type="button" value=" + " onclick="op(1)">
</tr>

<tr>
<td><input id = "4" type="button" value=" 4 " onclick="num(4)">
<td><input id = "5" type="button" value=" 5 " onclick="num(5)">
<td><input id = "6" type="button" value=" 6 " onclick="num(6)">
<td><input id = "sub" type="button" value=" - " onclick="op(2)">
</tr>
<tr>
<td><input id = "7" type="button" value=" 7 " onclick="num(7)">
<td><input id = "8" type="button" value=" 8 " onclick="num(8)" >
<td><input id = "9" type="button" value=" 9 " onclick="num(9)">
<td><input id = "mul" type="button" value=" * " onclick="op(3)">
</tr>
<tr>
<td><input id = "0" type="button" value=" 0 " onclick="num(0)">
<td><input id = "div" type="button" value=" / " onclick="op(4)">
<td><input id = "mod" type="button" value=" % " onclick="op(5)">
<td><input id = "equ" type="button" value=" = " onclick="eq()">
</tr>
<tr>
<td colspan="2"><input id = "c" type="reset" value=" c " ><td>
<input id = "bs" type="button" value=" <--- " onclick="mod()">
</tr>

</table>
</form>
</body>
</html>

```

- vi calculatortest.js

```

const { Builder, By , util } = require("selenium-webdriver");

const assert = require("assert");

async function calciTest() {

    // launch the browser

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

    try {

        await driver.get("file:///home/mvsr/Calculator/index.html");
        await driver.sleep(2000);

        await driver.findElement(By.id("5")).click();
        await driver.findElement(By.id("add")).click();
        await driver.findElement(By.id("8")).click();
        await driver.findElement(By.id("equ")).click();
        await driver.sleep(2000);

        const value = await driver.findElement(By.name("t1")).getAttribute("value");

        console.log(" answer is " + value);

        await driver.sleep(2000);

        assert.equal(value, '13');

        console.log(" Addition success");

        await driver.findElement(By.id("c")).click();
        await driver.findElement(By.id("5")).click();
        await driver.findElement(By.id("sub")).click();
        await driver.findElement(By.id("2")).click();
        await driver.findElement(By.id("equ")).click();
        await driver.sleep(2000);

        const v1 = await driver.findElement(By.name("t1")).getAttribute("value");

        console.log(" answer is " + v1);
        await driver.sleep(2000);
        assert.equal(v1, '3');

        console.log(" Subtraction success");

        await driver.findElement(By.id("c")).click();
        await driver.findElement(By.id("5")).click();
        await driver.findElement(By.id("mul")).click();
        await driver.findElement(By.id("2")).click();
        await driver.findElement(By.id("equ")).click();
        await driver.sleep(2000);

        const v2 = await driver.findElement(By.name("t1")).getAttribute("value");

        console.log(" answer is " + v2);

        await driver.sleep(2000);

        assert.equal(v2, '10');
    }
}

```

```

    console.log(" Multiplication success");

    await driver.findElement(By.id("c")).click();
    await driver.findElement(By.id("6")).click();
    await driver.findElement(By.id("div")).click();
    await driver.findElement(By.id("2")).click();
    await driver.findElement(By.id("equ")).click();
    await driver.sleep(2000);

    const v3 = await driver.findElement(By.name("t1")).getAttribute("value");

    console.log(" answer is " + v3);

    await driver.sleep(2000);

    assert.equal(v3, '3');

    console.log(" Division success");

    } finally {
        await driver.quit();
    }
}
calciTest();

```

- **vi Dockerfile**

```

FROM node:16.14.0
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . /app
CMD ["node" , "calculatortest.js"]

```

- **vi calci-docker-compose.yaml**

```

version: "3.0"
services:
  myweb:
    image: node:16.14.0
    build: .
    container_name: calcicons
    restart: always
    ports:
      - "8081:8081"
    expose:
      - "8081"

```

- **Execute the program in local environment**

- \$ apt-get update
- \$ apt install nodejs
- \$ apt install npm
- \$ npm install -g express
- \$ node -v

- \$ npm -v
- \$ npm init
- \$ npm install selenium-webdriver // add selenium web driver as dependency
- \$ npm init
- \$ node calculatortest.js
- Build and run the Docker image
 - \$ docker build -t calciimage .
 - \$ docker run --rm -it -p 8081:8081 --name calcicon calciimage
- **Go to `https://localhost:8081/index.html` to see the output**
- Build and run the Docker images using Docker Compose
 - \$ docker-compose -f calci-docker-compose.yaml up -d
- Create a Kubernetes Deployment YAML file (calci-app-deployment.yaml) to deploy the web application:

calci-app-deployment.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: calci-app-deployment
spec:
  replicas: 1 # Number of pods to create
  selector:
    matchLabels:
      app: calci-webapp # Label to match pods
  template:
    metadata:
      labels:
        app: calci-webapp # Label assigned to pods
    spec:
      containers:
        - name: calci-webapp-container
          image: calciimage:latest # Docker image to use
          ports:
            - containerPort: 80 # Port to expose

```

- Apply the deployment configuration to your Kubernetes cluster:

```
$ kubectl apply -f calci-app-deployment.yaml
```

- Check the status of your pods:


```
$ kubectl get deployments
$ kubectl get pods
```

\$ kubectl describe deployment my-web-app-deployment

- Login to GITHUB and create a new GIT repository name **CalculatorRepo** and copy the URL
- Push the code to GITHUB
 - \$ git --version
 - \$ git config --global user.name "sowjanya"
 - \$ git config --global user.email sowjanya@gmail.com
 - \$ git remote add origin <https://github.com/sowjanya/CalculatorRepo.git>
 - \$ git config --list
 - \$ git init
 - \$ git status
 - \$ git add .
 - \$ git commit -m "1st commit files added"
 - \$ git branch -M main
 - \$ git status
 - \$ git push -u origin main

Go to remote repository and check whether project is uploaded on github.

- Login to Docker hub and push the image to your account
 - \$ docker login -u [Dockerhubusername]
Eg: \$ docker login -u sowjanyaajindam
 - \$ docker tag imagename Dockerhubusername/imagename
Eg: \$ docker tag calciimage sowjanyaajindam/calciimage
 - \$ docker push Dockerhubusername/imagename
Eg: \$ docker push sowjanyaajindam/calciimage
- Pull the code from Docker hub and execute
 - \$ docker pull sowjanyaajindam/calciimage
 - \$ docker run -it sowjanyaajindam/calciimage

EXPERIMENT NO. 15

AIM: Containerization of an application.

Case Study #2: Develop a Application (MINI-Project) Dockerize the program and push the containerized application code to GitHub and DockerHub.

- 1. Abstract of you Mini Project.**
- 2. Code.**
- 3. Execution steps.**
- 4. GITHUB URL.**
- 5. DOCKERHUB URL.**

