Assignment 1

Task: Demonstrate the use of Git as a Version Control System on a local machine for multiple users.

Steps To Achieve:

- Set up multiple users in Git by configuring separate user profiles.
- Create a repository where each user can commit code, create branches (forks), and merge changes.
- Assign a “team leader” role who can review and merge the “coder” branches.
- Use basic Git commands to demonstrate creating branches, comparing changes (diff), and merging code.
- Finally, we'll compare the main differences between Git and SVN.

Commands:

1. **Creating project and adding files to it.**

```
mkdir project

cd project

touch main.txt

git init

git add .

git commit -m "First commit"

touch main2.txt

git add .
```

```
git commit -m "Second commit"
```

2.Creating coder 1 And Branching the project And adding files.

```
git checkout -b coder1

git config user.name "c1"

 git config user.email "c1@gmail.com"

touch c1.txt

git add .

git commit -m "Coder 1 Feature"
```

3.Creating coder 2 And Branching the project And adding files.

```
git checkout master

git checkout -b c2

git config user.name "c2"

git config user.email "c2@gmail.com"

touch t2.txt

git add .

git commit -m "Second Coder Commit"
```

4.After Branching making changes to main branch.

```
git checkout master

 touch After_branch.txt

git add .

git commit -m "Bug Fixed"
```

5.Merging The branches.

```
git diff c1

git merge coder1

git merge c2
```

6.To see the Graph.

```
git log --all --graph
```

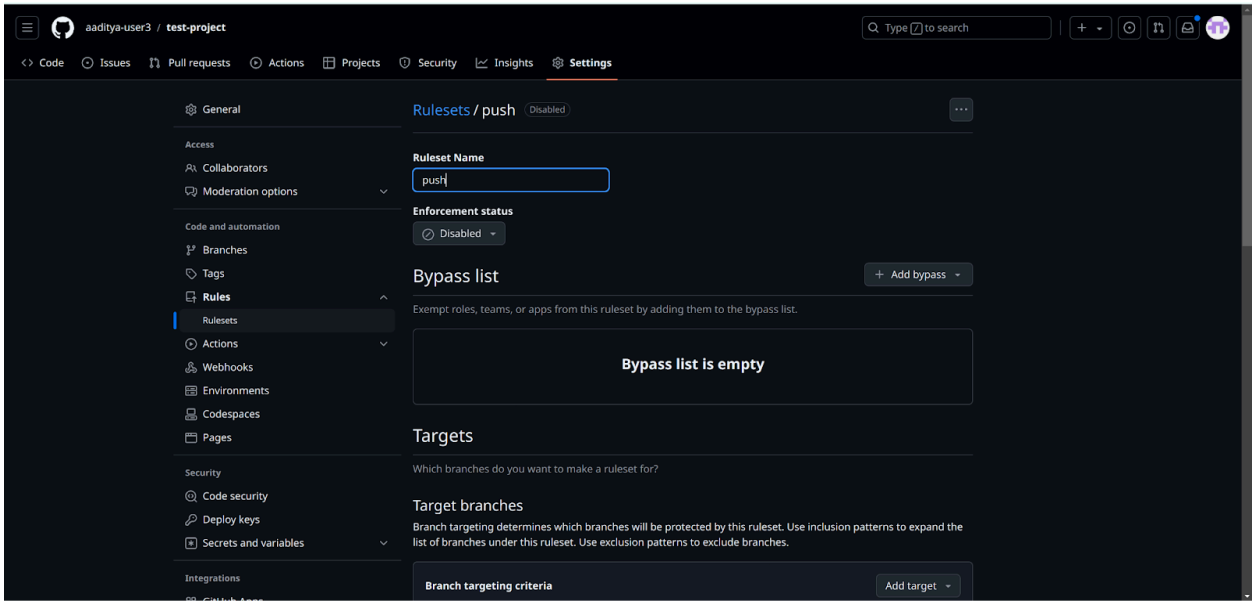
Feature	SVN (Subversion)	Git
Repository Type	Centralized (single repository server)	Distributed (each user has a complete repository)
Offline Capability	Limited; requires server access for most actions	Full offline access; can commit, branch, and merge
Branching and Merging	Branches are directory copies, often slower	Lightweight, faster branching and merging
Version Tracking	Tracks changes per file	Tracks snapshots of the entire project
Commit History	Linear and centralized	Non-linear; each user can have their own commit history
Storage	Stores changes on a per-file basis	Uses snapshots of the project tree

Assignment 2

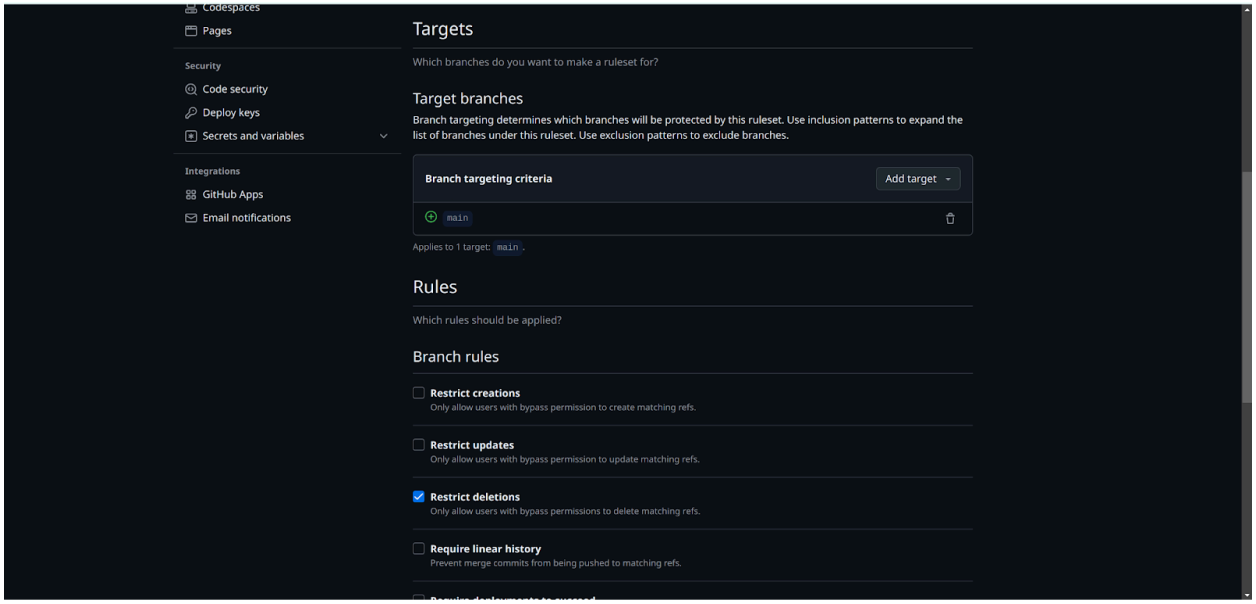
2. Demonstrate to use of Version Control System (Git offline and connect to github account).
Create multiple users and usage with team leader role and coder roles.
Demonstrate code pull and push, fork (branching) amongst the users
Compare it with hg. (on answer sheet)

Procedure:

Create 3 users - 1 leader, 2 coders (user1 and user2)
The leader creates GitHub repository and adds 2 collaborators (coders).
Rules are created for coders by leader e.g. coder cannot directly push to main branch without Pull Request.
To do this, log onto leader account.
Go to Settings -> Rules -> Ruleset
Create new rule

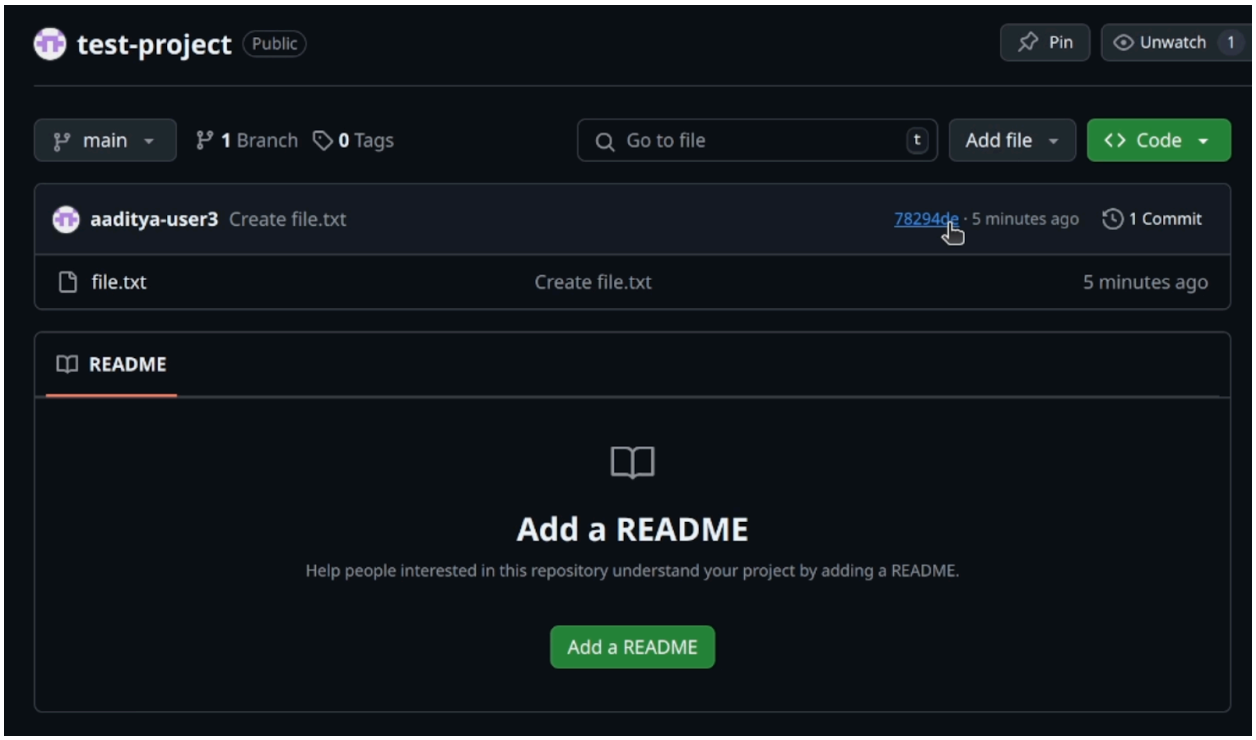


Set target branch -> by pattern .e.g main.

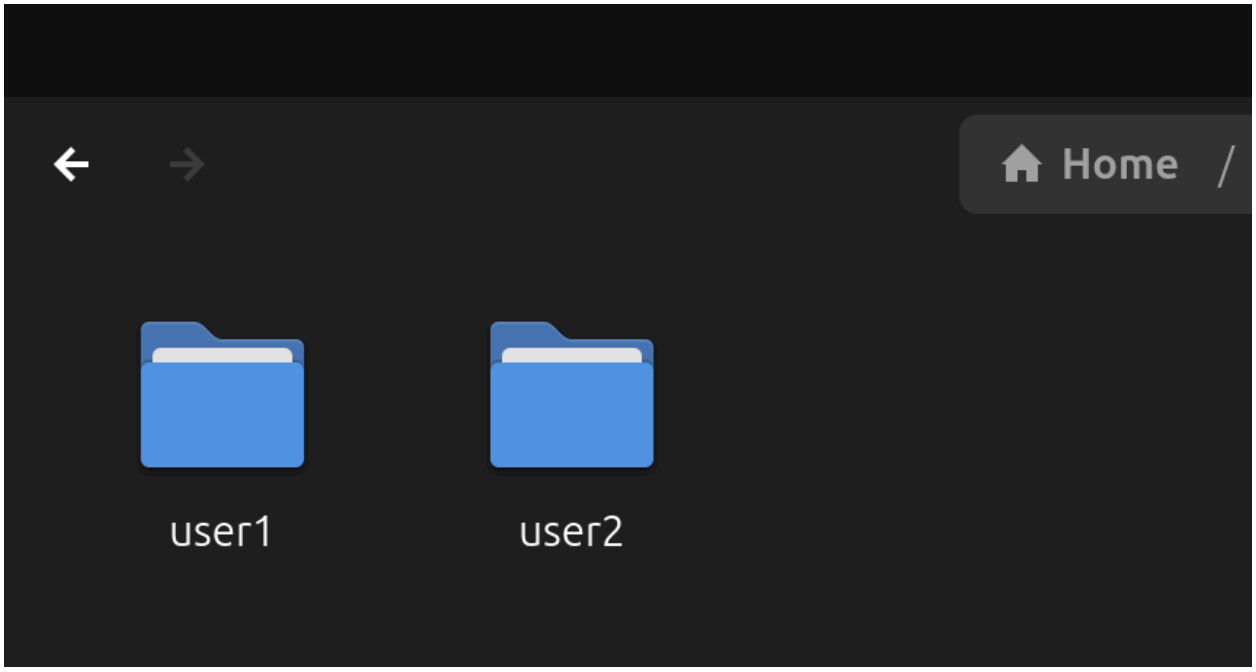


Add rules e.g. Require a pull request before merging
Save changes.

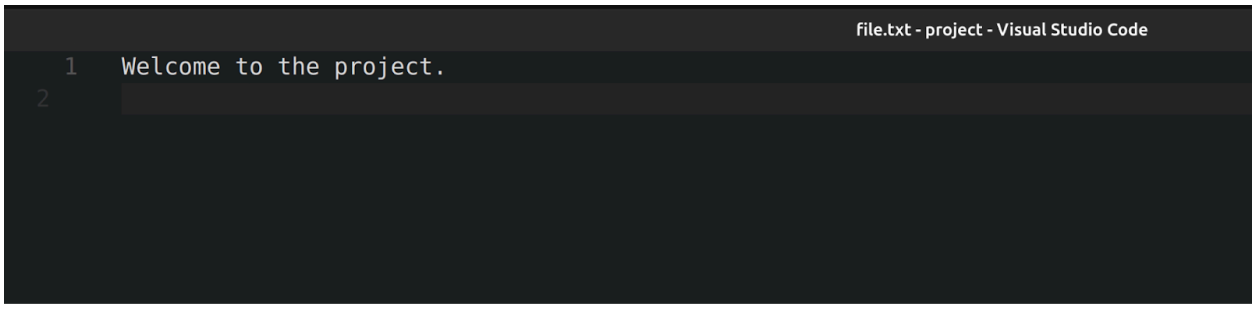
Workflow:
The leader creates a main branch and adds some files. E.g. file.txt



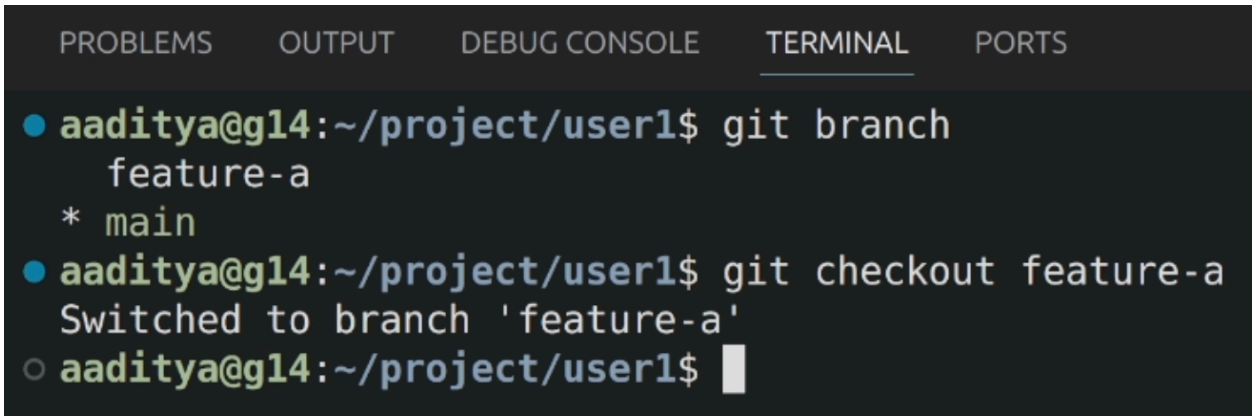
Now coders - user1 and user2 can clone this repository locally using:
git clone <repo url>
For demonstration, consider user1 clones it to folder ‘user1’ and user2 clones it to folder ‘user2’.



Consider this file.txt from main branch.



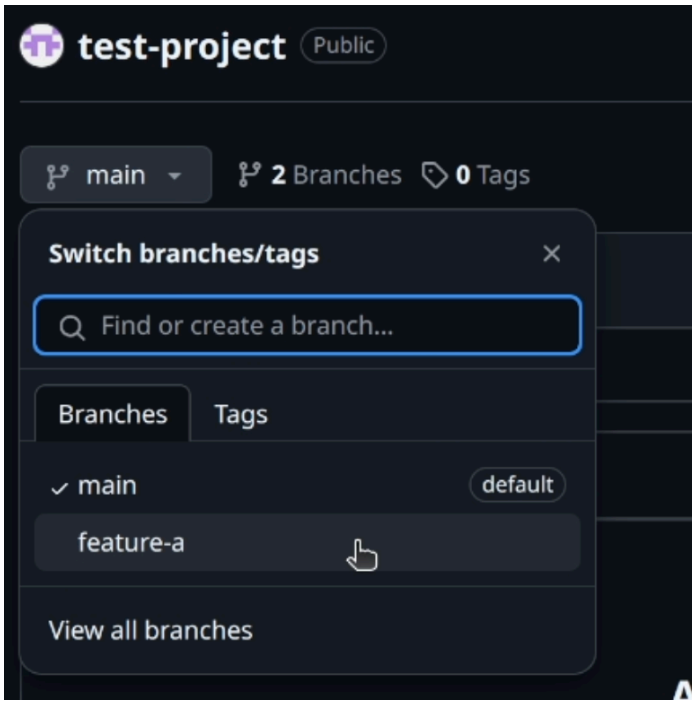
To start working, user1 will create a feature branch e.g. feature-a, and add some code.



```
3   Welcome to the project.
2
1   ===
4   Feature A code written by user1
```

Now user1 will push his/her changes onto the remote feature-branch as shown:

```
aaditya@g14:~/project/user1$ git push origin feature-a
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 316 bytes | 316.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature-a' on GitHub by visiting:
remote:      https://github.com/aaditya-user3/test-project/pull/new/feature-a
remote:
To https://github.com/aaditya-user3/test-project.git
 * [new branch]      feature-a -> feature-a
aaditya@g14:~/project/user1$
```

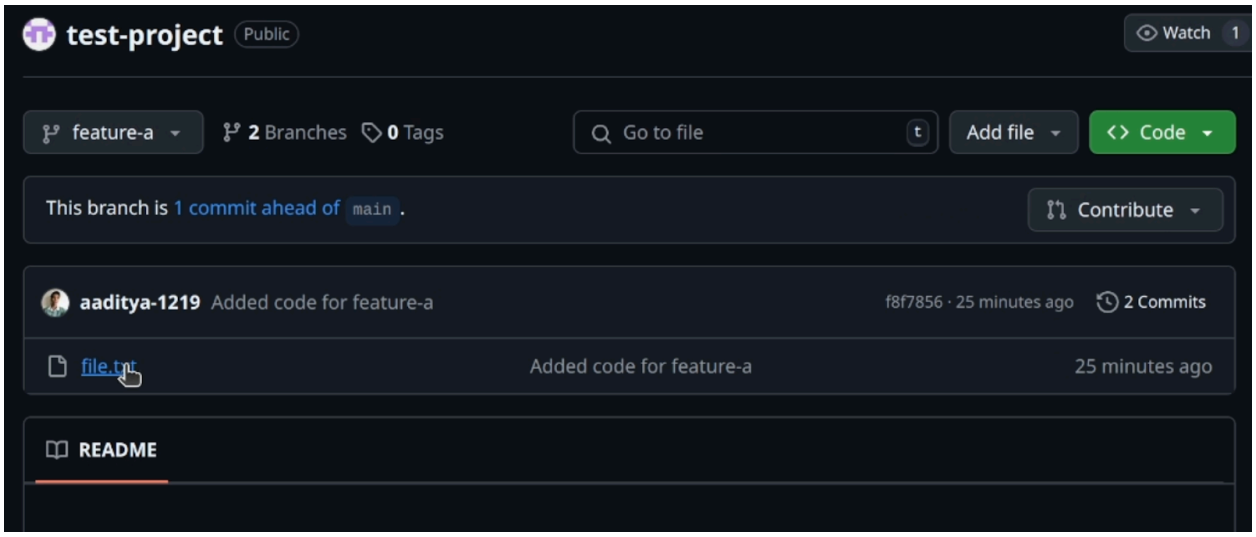


Note: If you get error like this…

```
aaditya@g14:~/project/user1$ git push origin feature-a
remote: Permission to aaditya-user3/test-project.git denied to aaditya-1219.
fatal: unable to access 'https://github.com/aaditya-user3/test-project.git/': The requested URL returned error: 403
aaditya@g14:~/project/user1$
-- NORMAL --
```

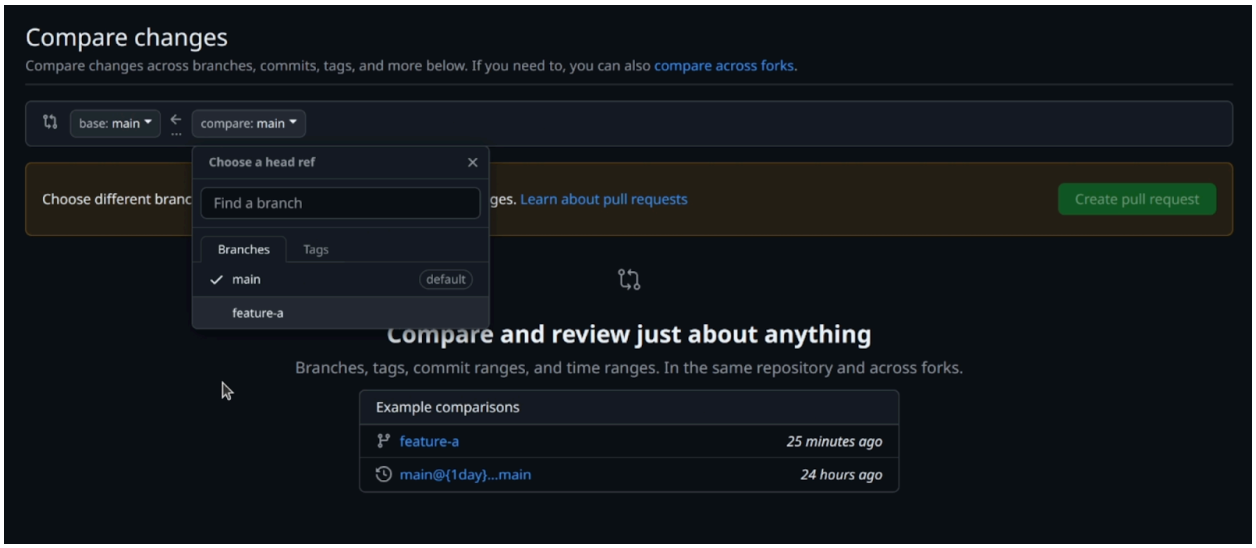
Refer this youtube video to fix it.
[Resolved Git push fatal unable to access the requested url returned error 403 - Github]

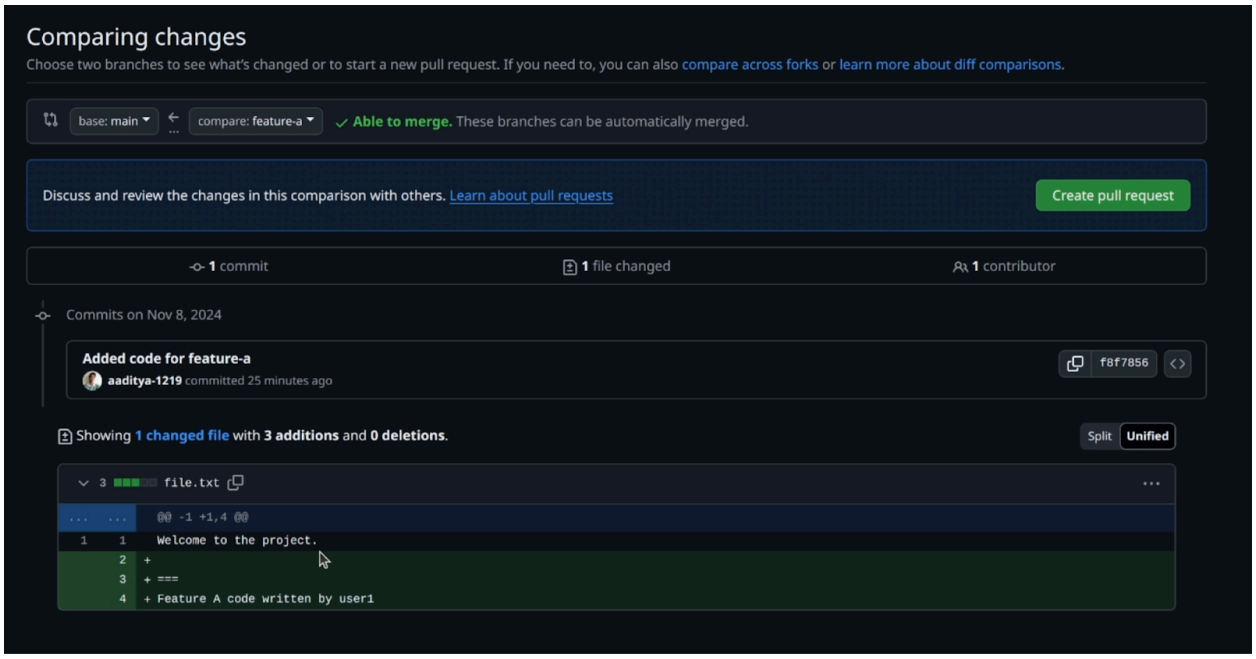
Moving on…



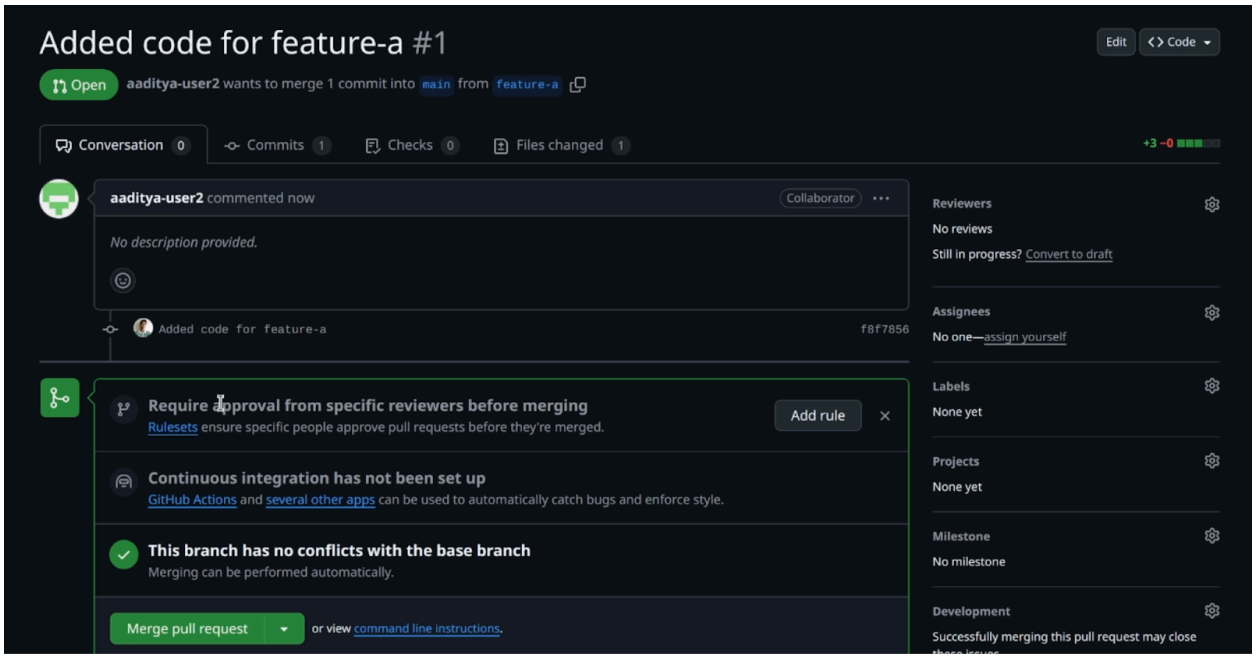
To merge these changes, user1 will create a Pull Request. The leader will approve this later.

To create a Pull Request, go to Pull Requests -> set compare: feature-a





Now the leader will approve this Pull Request:



The ‘feature-a’ branch will now be merged with the main branch.

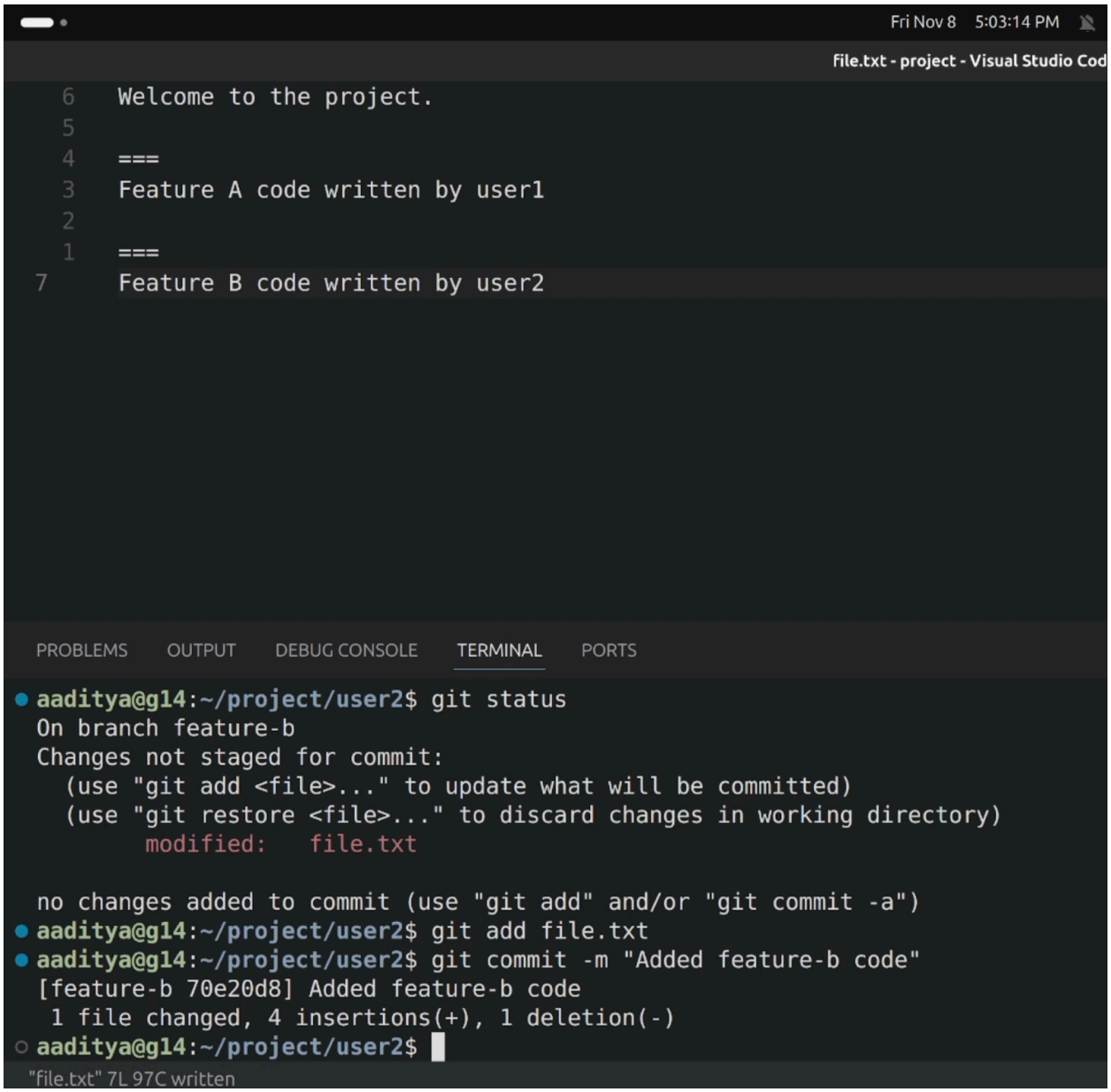
At this point, user2 does not have the changes made to the main branch.
So user2 will use **git pull** command to get the latest changes from main.
Command: **git pull**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
aaditya@g14:~/project/user2$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 7 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (7/7), 2.00 KiB | 511.00 KiB/s, done.
From https://github.com/aaditya-user3/test-project
 * [new branch]      main       -> origin/main
 * [new branch]      feature-a  -> origin/feature-a
aaditya@g14:~/project/user2$
```

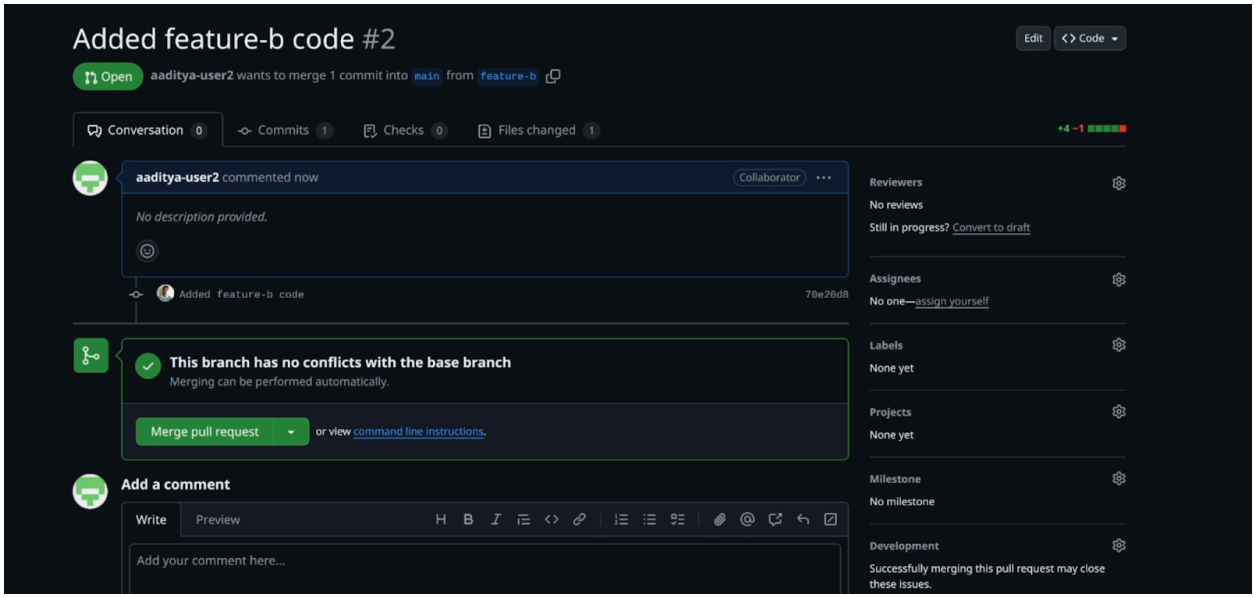
Now user2 has changes from main.

```
Fri Nov 8
file.txt - project - Vi
  3  Welcome to the project.
  2
  1  ===
  4  Feature A code written by user1
```

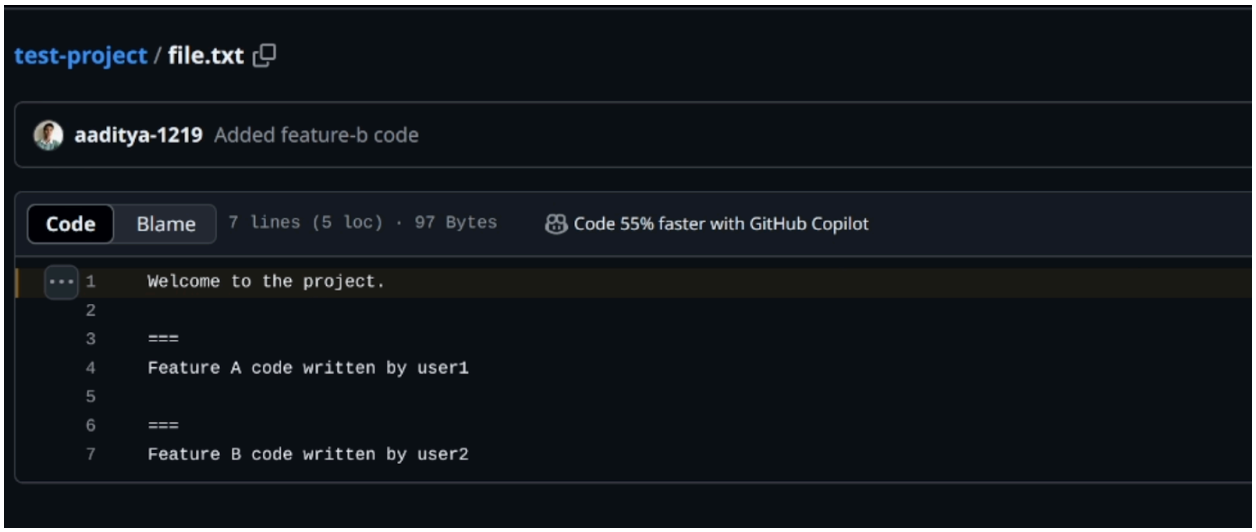
Similarly, user2 will create a feature branch ‘feature-b’ as shown earlier.
Now user2 will add some code, commit the changes then push.



As shown earlier, user2 will also create a pull request, which will be approved by the leader.



Final result:



Git vs Mercurial (Hg)

Feature	Git	Mercurial (Hg)
Performance	Generally faster, especially with large repos	Slower with large repositories
Branching	Lightweight, encourages branching workflows	Heavier branching, limited branching flexibility
Storage Model	Uses SHA-1 for integrity; stored as snapshots	Uses SHA-1; stored as changesets (patch-based)
Ease of Use	Powerful but has a steeper learning curve	More user-friendly, simpler commands
Popularity & Community	Very popular, widely adopted (e.g., GitHub)	Less popular, smaller community
Tooling	Strong tooling (GitHub, GitLab, etc.)	Limited compared to Git's ecosystem
File Renames	Detects renames implicitly	Explicitly tracks renames, potentially more accurate

Assignment 3 :-

3.Demonstrate to use of Version Control System (Git offline and connect to github account). Create multiple users and usage with team leader role and coder roles. Demonstrate code pull and push, fork (branching) amongst the users, code merge, code diff amongst the users. Compare git with gitea. (on answer sheet)

Same question as question 2 done by Aditya S Patil

For git diff it finds the changes made from

Last commit to current stagged changes just type git diff in terminal

Assignment 4

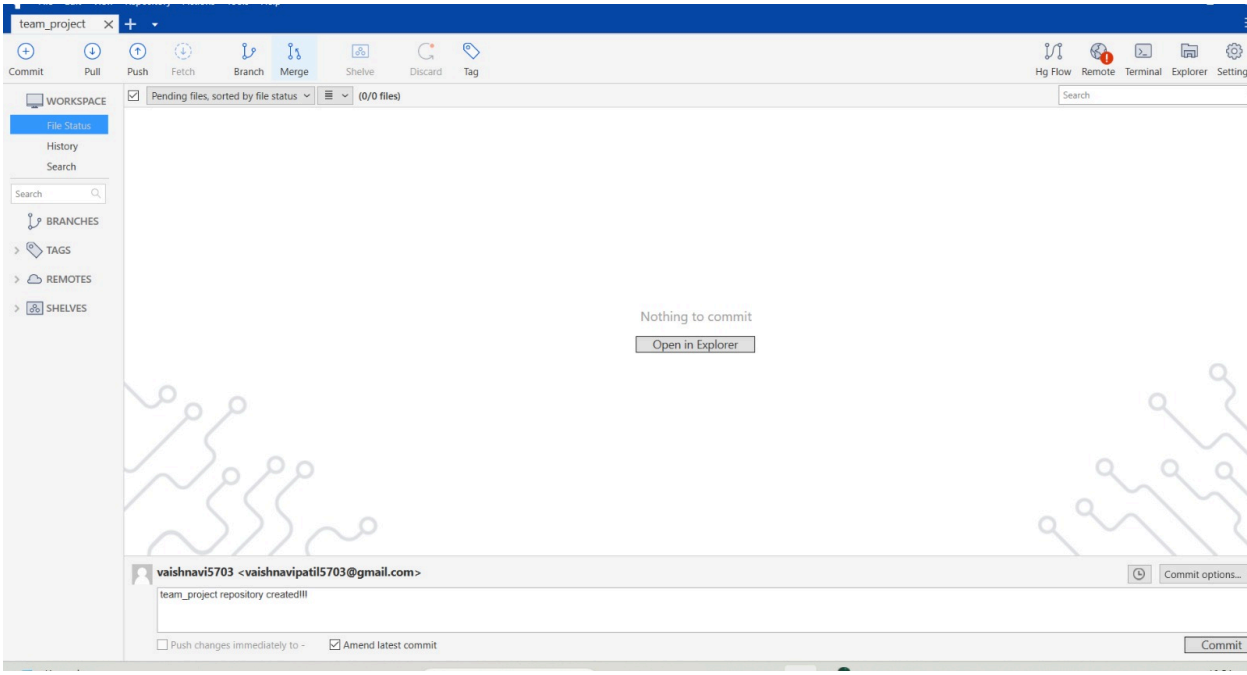
Title: Demonstrate to use of Version Control System (hg mercurial offline: on local machine with multiple user). Create multiple users and usage with team leader role and coder roles. Demonstrate code merge, fork (branching) and code diff amongst the users Compare it with svn.

How to Achieve This:

1. **Prerequisites: Installing Mercurial**

   Windows Installation (Using SourceTree)

   - Download SourceTree and install it on Windows.
   - SourceTree supports Mercurial, so you'll be able to manage your repositories with a graphical interface.
   - Verify Mercurial Installation:
     Open the SourceTree's terminal (present on right corner) and run:
     #hg --version



Ubuntu Installation

#sudo apt update
#sudo apt install mercurial
#hg --version

2. **Setup Project Repository and Users**

- Create a Project Directory
  #mkdir C:\Users\Vaishnavi\Downloads\team_project
  #cd C:\Users\Vaishnavi\Downloads\team_project
- Initialize the Mercurial Repository
  #hg init
- Create a File to Track
  #echo "Welcome to the team project!" > README.txt
- Add the File to the Repository
  #hg add README.txt
- Set the Username for the First Commit (Team Leader)
  - create an `.hg/hgrc` file with their name
  - Open a new text file named `hgrc` (without any extension) in a text editor and add:

    [ui]
    username = TeamLeader <teamleader@example.com>
- Commit the Initial Changes:
  #hg commit -m "Initial commit: Added README file"
  #hg log

```
C:\Users\Vaishnavi\Downloads\team_project>echo "Welcome to the team project
!" > README.txt

C:\Users\Vaishnavi\Downloads\team_project>hg add README.txt

C:\Users\Vaishnavi\Downloads\team_project>cd .hg

C:\Users\Vaishnavi\Downloads\team_project\.hg>cd ..

C:\Users\Vaishnavi\Downloads\team_project>hg commit -m "Initial commit: Add
ed README file"

C:\Users\Vaishnavi\Downloads\team_project>hg log
changeset:   0:ba5ea77a1e39
tag:         tip
user:        TeamLeader <teamleader@example.com>
date:        Fri Nov 08 11:00:06 2024 +0530
summary:     Initial commit: Added README file
```

3. **Simulating Work for Multiple Users**
   - Simulate Coder1's Work
     - Open the `.hg/hgrc` file again, and change the username line to:

       [ui]

       username = Coder1 <coder1@example.com>

     - Create a Branch for Coder1: #hg branch feature1_branch
     - Add a New File and Commit as Coder1:
       - #echo "Coder1's feature 1 code" > feature1.txt
       - #hg add feature1.txt
       - #hg commit -m "Coder1: Added feature1"
     - Make an Improvement and Commit Again:
       - #echo "Coder1's improved feature 1 code" >> feature1.txt
     - #hg commit -m "Coder1: Improved feature1"

```
C:\Users\Vaishnavi\Downloads\team_project>hg branch feature1_branch
marked working directory as branch feature1_branch
(branches are permanent and global, did you want a bookmark?)

C:\Users\Vaishnavi\Downloads\team_project>echo "Coder1's feature 1 code" > feature1.txt

C:\Users\Vaishnavi\Downloads\team_project>hg add feature1.txt

C:\Users\Vaishnavi\Downloads\team_project>hg commit -m "Coder1: Added feature1"

C:\Users\Vaishnavi\Downloads\team_project>echo "Coder1's improved feature 1 code" >> feature1.txt

C:\Users\Vaishnavi\Downloads\team_project>hg commit -m "Coder1: Improved feature1"

C:\Users\Vaishnavi\Downloads\team_project>hg log
changeset:   2:6e0243f37306
branch:      feature1_branch
tag:         tip
user:        Coder1 <coder1@example.com>
date:        Fri Nov 08 11:01:16 2024 +0530
summary:     Coder1: Improved feature1

changeset:   1:72c8399c8365
branch:      feature1_branch
user:        Coder1 <coder1@example.com>
date:        Fri Nov 08 11:01:01 2024 +0530
summary:     Coder1: Added feature1

changeset:   0:ba5ea77a1e39
user:        TeamLeader <teamleader@example.com>
date:        Fri Nov 08 11:00:06 2024 +0530
summary:     Initial commit: Added README file
```

Simulate Coder2's Work (use same commands…just change names)

- Set the Username to Coder2:
  - Reopen `.hg/hgrc` and update the username line to:

    `[ui]`

    `username = Coder2 <coder2@example.com>`
- Create a Branch for Coder2:

- Add a New File and Commit as Coder2:
- Make an Improvement and Commit Again:

```
C:\Users\Vaishnavi\Downloads\team_project>hg branch feature2_branch
marked working directory as branch feature2_branch

C:\Users\Vaishnavi\Downloads\team_project>echo "Coder2's feature 2 code" > feature2.txt

C:\Users\Vaishnavi\Downloads\team_project>hg add feature2.txt

C:\Users\Vaishnavi\Downloads\team_project>hg commit -m "Coder2: Added feature2"

C:\Users\Vaishnavi\Downloads\team_project>echo "Coder2's improved feature 2 code" >> feature2.txt

C:\Users\Vaishnavi\Downloads\team_project>hg commit -m "Coder2: Improved feature2"

C:\Users\Vaishnavi\Downloads\team_project>hg log
changeset:   4:0d052392876f
branch:      feature2_branch
tag:         tip
user:        Coder2 <coder2@example.com>
date:        Fri Nov 08 11:04:40 2024 +0530
summary:     Coder2: Improved feature2

changeset:   3:2d3139ff8cef
branch:      feature2_branch
user:        Coder2 <coder2@example.com>
date:        Fri Nov 08 11:04:34 2024 +0530
summary:     Coder2: Added feature2

changeset:   2:6e0243f37306
branch:      feature1_branch
user:        Coder1 <coder1@example.com>
date:        Fri Nov 08 11:01:16 2024 +0530
summary:     Coder1: Improved feature1

changeset:   1:72c8399c8365
branch:      feature1_branch
user:        Coder1 <coder1@example.com>
date:        Fri Nov 08 11:01:01 2024 +0530
summary:     Coder1: Added feature1
```

4. **Merging Branches (Team Leader's Role)**

- Set the Username to Team Leader: Update the `.hg/hgrc` file again:

  [ui] username = TeamLeader <teamleader@example.com>

- Update to the Default Branch:
  - #hg update default
- **Merge `feature1_branch` into Default**:
  - #hg merge feature1_branch
  - #hg commit -m "Merged feature1_branch into default by Team Leader"
- **Merge `feature2_branch` into Default**:
  - #hg merge feature2_branch
  - #hg commit -m "Merged feature2_branch into default by Team Leader"
- hg log

```
C:\Users\Vaishnavi\Downloads\team_project>hg update default
0 files updated, 0 files merged, 2 files removed, 0 files unresolved

C:\Users\Vaishnavi\Downloads\team_project>hg merge feature1_branch
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
(branch merge, don't forget to commit)

C:\Users\Vaishnavi\Downloads\team_project>hg commit -m "Merged feature1_branch into default by Team Leader"

C:\Users\Vaishnavi\Downloads\team_project>hg merge feature2_branch
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
(branch merge, don't forget to commit)

C:\Users\Vaishnavi\Downloads\team_project>hg commit -m "Merged feature2_branch into default by Team Leader"

C:\Users\Vaishnavi\Downloads\team_project>hg diff -r feature1_branch -r feature2_branch
diff -r 6e0243f37306 -r 0d052392876f feature2.txt
--- /dev/null  Thu Jan 01 00:00:00 1970 +0000
+++ b/feature2.txt     Fri Nov 08 11:04:40 2024 +0530
@@ -0,0 +1,2 @@
+"Coder2's feature 2 code"
+"Coder2's improved feature 2 code"


C:\Users\Vaishnavi\Downloads\team_project>hg log
changeset:   6:ec19c96800a1
tag:         tip
parent:      5:18063f440752
parent:      4:0d052392876f
user:        TeamLeader <teamleader@example.com>
date:        Fri Nov 08 11:06:05 2024 +0530
summary:     Merged feature2_branch into default by Team Leader

changeset:   5:18063f440752
parent:      0:ba5ea77a1e39
parent:      2:6e0243f37306
user:        TeamLeader <teamleader@example.com>
date:        Fri Nov 08 11:05:53 2024 +0530
summary:     Merged feature1_branch into default by Team Leader

changeset:   4:0d052392876f
branch:      feature2_branch
user:        Coder2 <coder2@example.com>
date:        Fri Nov 08 11:04:40 2024 +0530
summary:     Coder2: Improved feature2
```

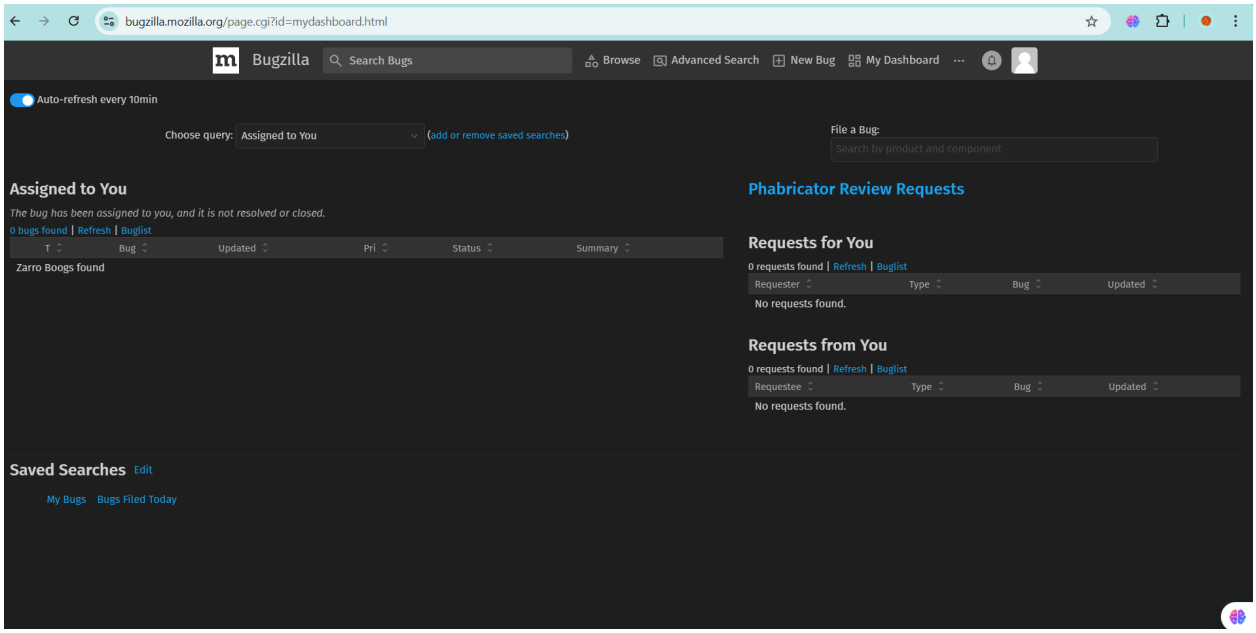
Feature	Mercurial (Hg)	Subversion (SVN)
Distributed VCS	Fully distributed; each user has a complete repo.	Centralized; requires a server.
Branching	Lightweight branches are easy and fast.	Branching is more complex and server-dependent.
Offline Access	Full offline access to repo history.	Limited offline access; server needed for most operations.
Performance	Faster for larger projects and teams.	Can slow down due to server reliance.

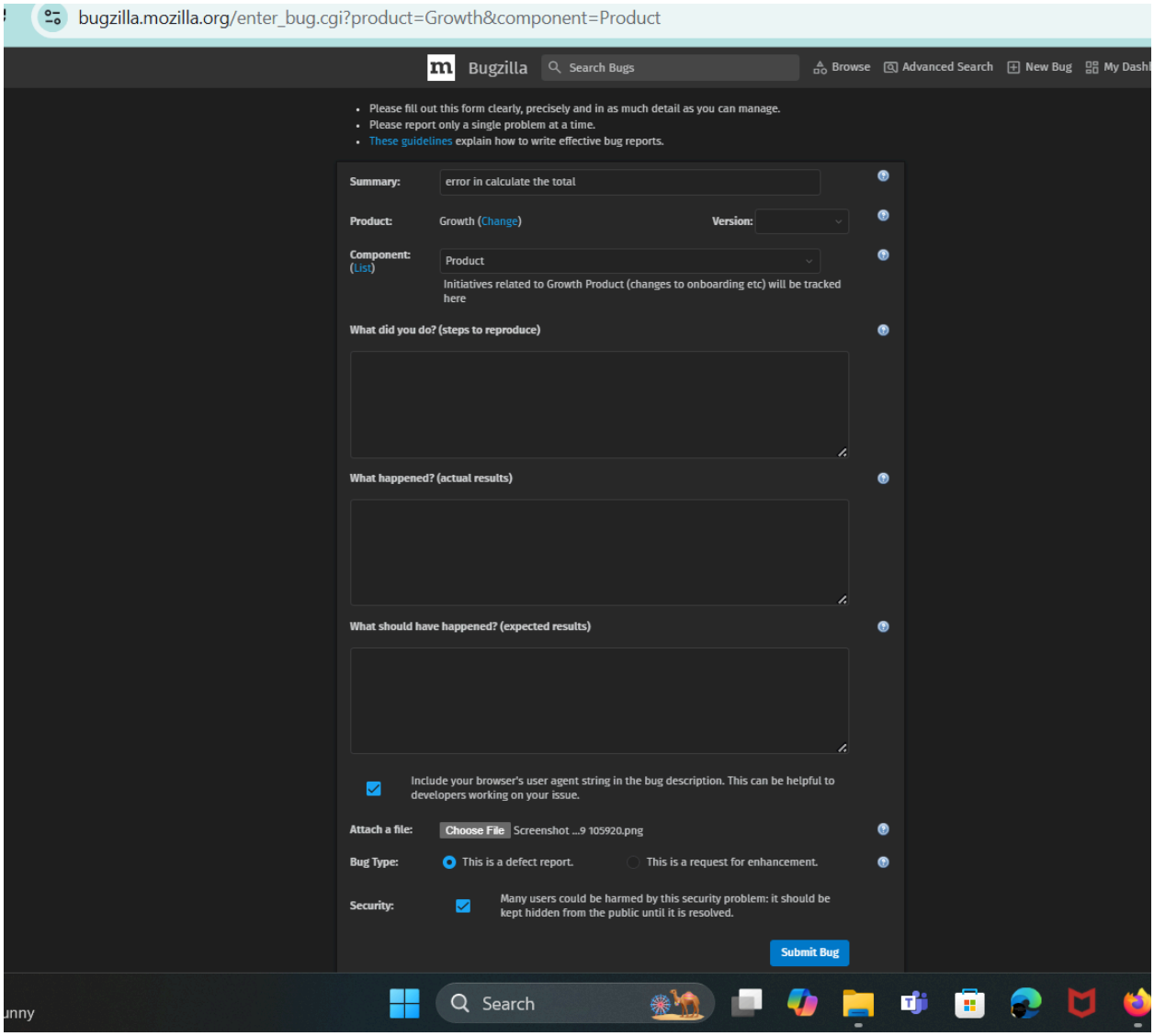
Assignment 8

Part 1: Setting Up Bugzilla

1. Create a Bugzilla Account:
    - Visit the Bugzilla website or your organization's Bugzilla setup.
    - Sign up for a new account if you don't have one.
2. Familiarize Yourself with the Interface:
    - Once logged in, explore the dashboard, especially focusing on creating bugs, searching for existing issues, and reporting/monitoring bugs.
3. Choose an Open-Source Project:
    - Select an open-source project from a site like GitHub, GitLab, or SourceForge. Some well-documented and actively developed projects are easier for identifying bugs. Make sure the project has an open-source license and allows you to modify and test the code.

Bugzilla account created





Step 2 : Report Each Bug in Bugzilla

1. **Open Bugzilla and Go to “File a Bug”**:
    - In Bugzilla, go to the **File a Bug** section and select the project where you’ll report the bugs (if you don’t see the project, you might need to create a new product in Bugzilla).
2. **Fill in Bug Details for Each Issue**:
    - **Component**: Enter the specific part of the application affected by the bug (e.g., UI, Calculation Module).

    - **Severity**: Rate how severe the issue is (e.g., **Critical** for a major crash, **Normal** for minor bugs).
    - **Summary**: Write a brief but descriptive summary of the bug (e.g., “Calculator Division by Zero Error”).
    - **Description**:
        - **Steps to Reproduce**: List the steps you followed to encounter the bug.
        - **Expected Result**: Describe what should have happened.
        - **Actual Result**: Describe what actually happened.
    - **Attachment**: Attach screenshots or code snippets relevant to the issue.
3. **Submit the Bug Report**:
    - Once everything is filled out, submit the bug report.
    - Repeat this process for each of the three issues (Bug/Error, Improvement, and New Feature).

Step 3: Track and Update Bug Status

1. **Monitor Bug Status**:
    - Keep an eye on your reported bugs in Bugzilla. If you’re actively working on fixes, update the **Status** field (e.g., In Progress, Resolved).
    - **Add Comments** if you need to clarify information or receive feedback from other contributors.
2. **Test and Resolve**:
    - For bugs you’re fixing, implement the necessary code changes locally, then re-run the application to ensure the bug is resolved.
    - Once resolved, change the status in Bugzilla to **Resolved** or **Fixed** and add a final comment summarizing the fix.

Assignment No.9

9 .Development of contribution to existing Open Source Software (on line upload of its git/svn repository with your valid login)(Language: java/pyth/perl/c/cpp/etc)

Pull to your github the **Open Source Software and** contribute in terms removing error/bug or adding feature to it. Upload the contributed code your repository as well original FOSS.

Steps for contributing to an open-source project by forking and submitting a pull request:

(It is recommended to use your own GitHub repository. After pushing your changes, send a pull request to the project maintainers for merging, so that the changes remain visible until the maintainers merge them.)

1. **Fork the Repository on GitHub(for own repo, this step not need)**
   - Go to the GitHub page of the project you want to contribute to.
   - Click the **Fork** button to create a copy of the repository under your GitHub account.
2. **Clone the Repository to Your Local Machine**
   - Clone your forked repository to your local machine using the git clone command.
3. **Navigate to the Project Directory**
4. **Set Up the Upstream Remote**
   - Add the original repository (upstream) as a remote to keep your local repository up to date.
5. **Verify the Remote Setup**
   - Ensure that both origin (your fork) and upstream (the original repository) remotes are correctly set up.
6. **Fetch the Latest Changes from Upstream**
   - Fetch the latest changes from the original repository (upstream) to sync your fork.
7. **Merge Changes from Upstream into Your Local Repository**
   - Merge changes from the upstream master (or main) branch into your local repository to stay up-to-date.
8. **Switch to the Master Branch**
   - Ensure you're on the master (or main) branch before proceeding.
9. **Pull the Latest Changes to Your Master Branch**
   - Pull the latest changes from upstream into your master branch.
10. **Create a New Branch for Your Changes**

- Create a new branch for making your changes (e.g., my-contribution).

11. **Make Your Changes**
    - Edit files, fix bugs, or add new features as needed.
12. **Stage and Commit Your Changes**
    - Stage the modified files and commit your changes with a descriptive message.
13. **Push Your Changes to Your Fork**
    - Push the new branch containing your changes to your fork on GitHub.
14. **Create a Pull Request**
    - Go to your GitHub repository and create a pull request for the branch you just pushed.
    - Provide a description of your changes and submit the pull request to the original repository.
15. **Wait for Review and Feedback**
    - The project maintainers (in case you own)will review your pull request.
    - They may request changes or approve your changes for merging into the original repository.

Commands:

1. **Clone the Repository to Your Local Machine**

   git clone https://github.com/YOUR_USERNAME/REPOSITORY_NAME.git
2. **Navigate to the Project Directory**

   cd REPOSITORY_NAME
3. **Set Up the Upstream Remote**

   git remote add upstream
   https://github.com/ORIGINAL_OWNER/REPOSITORY_NAME.git
4. **Verify the Remote Setup**

Check if both the origin (your fork) and upstream (original repository) remotes are set up correctly.

git remote -v

Output:

You should see both remotes listed as follows:

https://github.com/YOUR_USERNAME/REPOSITORY_NAME.git (fetch)

origin https://github.com/YOUR_USERNAME/REPOSITORY_NAME.git (push)

upstream https://github.com/ORIGINAL_OWNER/REPOSITORY_NAME.git (fetch)

upstream https://github.com/ORIGINAL_OWNER/REPOSITORY_NAME.git (push)

5. **Fetch the Latest Changes from Upstream**

   git fetch upstream

6. **Merge Changes from Upstream into Your Local Repository**

   git merge upstream/master

7. **Switch to the Master Branch**

   Make sure you're on the master branch (or main, if that's the default branch name).

   git checkout master

8. **Pull the Latest Changes to Your Master Branch**

   git pull upstream master

9. **Create a New Branch for Your Changes**

   git checkout -b my-contribution

10. **Now, make the necessary changes to the code. You can edit files, fix bugs, add new features, etc.**

11. **Stage and Commit Your Changes**

    git add .

    git commit -m "Describe your changes here"

12. **Push Your Changes to repo**

    git push origin my-contribution

13. **Create a Pull Request**

- After pushing your changes, go to your GitHub repository.
- You'll see an option to create a pull request for your newly pushed branch.
- Click Compare & pull request and provide a description of the changes you made.
- Submit the pull request to the original repository to propose your changes.

15. **Wait for Review and Feedback**

- The project maintainers(if your repo, you see the pulled request and merge option)will review your pull request.
- Once they approve it, your changes will be merged into the original repository!

Assignment 11

Step 1: Set Up Your Project

1. **Create Project Directory**:
    - Create a folder on your system, e.g., `java-calculator-rpm`, where you'll place all the necessary files.
2. **Create Java Source File**:

In your project directory, create a file called `Calculator.java` with the following code:
java
Copy code

```
import java.util.Scanner;

public class Calculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter first number: ");
        double num1 = scanner.hasNextDouble() ?
scanner.nextDouble() : 0.0;  // Default value if no input

        System.out.print("Enter second number: ");
        double num2 = scanner.hasNextDouble() ?
scanner.nextDouble() : 0.0;  // Default value if no input

        System.out.println("Choose operation: +, -, *, /");
        String operation = scanner.hasNext() ? scanner.next() :
"+";  // Default operation if no input

        double result = 0;
        switch (operation) {
            case "+":
                result = num1 + num2;
                break;
```

```
            case "-":
                result = num1 - num2;
                break;
            case "*":
                result = num1 * num2;
                break;
            case "/":
                if (num2 != 0) {
                    result = num1 / num2;
                } else {
                    System.out.println("Cannot divide by zero");
                }
                break;
            default:
                System.out.println("Invalid operation");
        }
        System.out.println("Result: " + result);
        scanner.close();
    }
}
```

-

Step 2: Write the Dockerfile

1. **Create Dockerfile**:

In the same directory, create a file named `Dockerfile` with the following content:
dockerfile
Copy code

```
# Use CentOS 7 as the base image
FROM centos:7

# Use CentOS Vault mirrors to avoid repository issues
RUN sed -i 's|^mirrorlist=|#mirrorlist=|g'
/etc/yum.repos.d/CentOS-* && \
```

```
    sed -i
's|^#baseurl=http://mirror.centos.org|baseurl=http://vault.cento
s.org|g' /etc/yum.repos.d/CentOS-*

# Install necessary tools
RUN yum -y install java-1.8.0-openjdk-devel rpm-build

# Set up RPM build environment
RUN mkdir -p /root/rpmbuild/{BUILD,RPMS,SOURCES,SPECS,SRPMS}

# Copy the Java source code to the container
COPY Calculator.java /root/rpmbuild/SOURCES/

# Compile the Java code
RUN javac /root/rpmbuild/SOURCES/Calculator.java

# Create a script to run the Java program
RUN echo -e '#!/bin/bash\njava -cp /usr/local/bin Calculator' >
/root/rpmbuild/SOURCES/run_calculator.sh && \
    chmod +x /root/rpmbuild/SOURCES/run_calculator.sh

# Create RPM spec file
RUN echo "Name: calculator" >
/root/rpmbuild/SPECS/calculator.spec && \
    echo "Version: 1.0" >> /root/rpmbuild/SPECS/calculator.spec
&& \
    echo "Release: 1" >> /root/rpmbuild/SPECS/calculator.spec &&
\
    echo "Summary: A simple Java calculator" >>
/root/rpmbuild/SPECS/calculator.spec && \
    echo "License: GPL" >> /root/rpmbuild/SPECS/calculator.spec
&& \
    echo "Source: Calculator.java" >>
/root/rpmbuild/SPECS/calculator.spec && \
```

```
    echo "%description" >> /root/rpmbuild/SPECS/calculator.spec
&& \
    echo "This is a simple calculator application written in
Java." >> /root/rpmbuild/SPECS/calculator.spec && \
    echo "%prep" >> /root/rpmbuild/SPECS/calculator.spec && \
    echo "# No need to unpack Calculator.java, just copy it
directly" >> /root/rpmbuild/SPECS/calculator.spec && \
    echo "%build" >> /root/rpmbuild/SPECS/calculator.spec && \
    echo "%install" >> /root/rpmbuild/SPECS/calculator.spec && \
    echo "mkdir -p %{buildroot}/usr/local/bin" >>
/root/rpmbuild/SPECS/calculator.spec && \
    echo "cp /root/rpmbuild/SOURCES/Calculator.class
%{buildroot}/usr/local/bin/" >>
/root/rpmbuild/SPECS/calculator.spec && \
    echo "cp /root/rpmbuild/SOURCES/run_calculator.sh
%{buildroot}/usr/local/bin/" >>
/root/rpmbuild/SPECS/calculator.spec && \
    echo "%files" >> /root/rpmbuild/SPECS/calculator.spec && \
    echo "/usr/local/bin/Calculator.class" >>
/root/rpmbuild/SPECS/calculator.spec && \
    echo "/usr/local/bin/run_calculator.sh" >>
/root/rpmbuild/SPECS/calculator.spec && \
    echo "%changelog" >> /root/rpmbuild/SPECS/calculator.spec

# Build the RPM
RUN rpmbuild -ba /root/rpmbuild/SPECS/calculator.spec

# Install the RPM package to test it
RUN yum -y localinstall
/root/rpmbuild/RPMS/x86_64/calculator-1.0-1.x86_64.rpm

# Run the calculator to verify installation
CMD ["/usr/local/bin/run_calculator.sh"]
```

-

Step 3: Build the Docker Image

Run the following command from the directory where your `Dockerfile` and `Calculator.java` files are located:

bash

Copy code

```
docker build -t calculator-rpm .
```

-

Step 4: Run the Docker Container

To test the calculator program interactively, run the following command:

bash

Copy code

```
docker run -it --rm calculator-rpm
```

-

This will start the container in interactive mode, allowing you to input values and test the calculator functionality.

Assignment 12

Create RPM packages. (multiple modules/code packaging of c/cpp). Pull or take any FOSS project and one feature or remove bug and then create the package on suitable OS Compare RPM packaging with Debian packaging (on answer sheet)

1. Objective

The goal of this assignment is to modify the `htop` program by adding a custom message, package it as an RPM, and install it on Fedora. We will be using Docker to create a controlled environment for the build.

2. Execution Plan

We will follow these steps:
1. Install Docker and set up a container environment.
2. Prepare the system by installing necessary dependencies.
3. Clone the `htop` repository, modify its code, and package it as an RPM.
4. Verify the RPM installation.

3. Detailed Steps and Commands

Step 1 : Install Docker via link or commands
Link : https://docs.docker.com/engine/install/ubuntu/
Commands:

1. for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done
2. # Add Docker's official GPG key:
   sudo apt-get update
   sudo apt-get install ca-certificates curl
   sudo install -m 0755 -d /etc/apt/keyrings
   sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
   sudo chmod a+r /etc/apt/keyrings/docker.asc

   # Add the repository to Apt sources:
   echo \
   "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
   $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
   sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
   sudo apt-get update

3. sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

Step 2 : Set Up the Build Environment

1. Start Docker and run a Fedora container:

```
sudo docker run -it fedora:latest /bin/bash
```

Now **either** copy this script inside container by following commands

1.dnf install nano
2.nano script.sh
3.paste the script content
4.save and exit by ctrl+o and ctrl+x
5.bash script.sh

script.sh :

```
#!/bin/bash

echo "Updating system and installing build dependencies..."
dnf update -y
dnf install -y rpm-build gcc make git autoconf automake ncurses-devel gdb


echo "Setting up RPM build directories..."
mkdir -p /root/rpmbuild/{BUILD,RPMS,SOURCES,SPECS,SRPMS}
echo '%_topdir /root/rpmbuild' > ~/.rpmmacros


echo "Cloning the htop repository from GitHub..."
cd /root/rpmbuild/SOURCES
git clone https://github.com/htop-dev/htop.git
cd htop


echo "Modifying source code to include a custom message..."
sed -i '1s|^|#include <stdio.h>\n|' htop.c
sed -i '/int main(/a \      printf("Running custom command\\n");' htop.c


echo "Creating a tarball of the modified htop source..."
cd ..
tar -czvf htop-custom.tar.gz htop
```

```
echo "Creating the RPM spec file..."
cat << 'EOF' > /root/rpmbuild/SPECS/htop.spec
Name:           htop
Version:        3.2.2
Release:        1%{?dist}
Summary:        Interactive process viewer with custom message

License:        GPL
URL:            https://github.com/htop-dev/htop
Source0:        %{name}-custom.tar.gz

BuildRequires:  ncurses-devel, gcc, make, autoconf, automake
Requires:       ncurses

%description
A modified version of htop with a custom message.

%prep
%setup -q -n htop

%build
./autogen.sh
./configure
make

%install
mkdir -p %{buildroot}/usr/local/bin
install -m 0755 htop %{buildroot}/usr/local/bin/htop

%files
/usr/local/bin/htop

%changelog
* Wed Nov 8 2024 User <user@example.com> - 3.2.2-1
- Added custom message in htop
EOF

# Step 7: Build the RPM package
echo "Building the RPM package..."
cd /root/rpmbuild
rpmbuild -ba SPECS/htop.spec
```

```
# Step 8: Check the generated RPM package
echo "Listing the generated RPM package..."
ls /root/rpmbuild/RPMS/x86_64/

echo "Installing the RPM package after building...."
dnf install -y /root/rpmbuild/RPMS/x86_64/htop-3.2.2-1.fc*.rpm

echo "Verifying ......"
/usr/local/bin/htop
```

or follow below steps :

1. Install necessary packages in the container:

```
dnf update -y
dnf install -y rpm-build gcc make git autoconf automake ncurses-devel gdb
```

2. Create the RPM build directories:

```
mkdir -p /root/rpmbuild/{BUILD,RPMS,SOURCES,SPECS,SRPMS}
echo '%_topdir /root/rpmbuild' > ~/.rpmmacros
```

3. Modify the `htop` Source Code ,Clone the `htop` GitHub repository and make the required code modifications:

```
cd /root/rpmbuild/SOURCES
git clone https://github.com/htop-dev/htop.git
cd htop
```

4. Add a custom message to the source code:

```
sed -i '1s|^|#include <stdio.h>\n|' htop.c
sed -i '/int main(/a \  printf("Running custom command\\n");' htop.c
```

5. Create a compressed tarball of the modified source code:

```
cd ..
tar -czvf htop-custom.tar.gz htop
```

6. Create the RPM Spec File

```
cat << 'EOF' > /root/rpmbuild/SPECS/htop.spec
Name:       htop
Version:    3.2.2
Release:    1%{?dist}
Summary:    Interactive process viewer with custom message

License:    GPL
URL:        https://github.com/htop-dev/htop
Source0:    %{name}-custom.tar.gz

BuildRequires:  ncurses-devel, gcc, make, autoconf, automake
Requires:   ncurses

%description
A modified version of htop with a custom message.

%prep
%setup -q -n htop

%build
./autogen.sh
./configure
make

%install
mkdir -p %{buildroot}/usr/local/bin
install -m 0755 htop %{buildroot}/usr/local/bin/htop

%files
/usr/local/bin/htop

%changelog
* Wed Nov 8 2024 User <user@example.com> - 3.2.2-1
- Added custom message in htop
EOF
```

7. Build the RPM Package

```
cd /root/rpmbuild
rpmbuild -ba SPECS/htop.spec


ls /root/rpmbuild/RPMS/x86_64/
```

8. Install and Verify the Custom RPM

dnf install -y /root/rpmbuild/RPMS/x86_64/htop-3.2.2-1.fc*.rpm

/usr/local/bin/htop

PRESS CTRL+C TO EXIT HTOP COMMAND AND YOU CAN SEE A CUSTOM MESSAGE PRINTED ON SCREEN

13. Create Debian packages.

(multiple modules/code packaging of java/c/cpp).
Pull or take any FOSS project and one feature and then create the package on suitable OS
Compare RPM packaging with Debian packaging (on answer sheet)

Step 1) Choose any simple github repository of cpp code i.e. having one code file (so that it'll be easy to implement and understand) and clone it on desktop by running command on in terminal
Cmd: git clone https://github.com/Yashashwi0708/Balanced_Hashmap

Step 2) CD into that folder
Cmd: cd Balanced_Hashmap

Step 3) Create a folder named Debian for keeping control file in it and make a directory structure as usr/local/bin for keeping binary/executable file of code in it

Cmd: mkdir Debian
    mkdir -p usr/local/bin

Step 4) Go inside Debian and make a file named control

Cmd: cd Debian
    nano control

Write following things in it

Package: balanced-hashmap
Version: 0.2
Maintainer: User_Name
Architecture: all
Description: A balanced hashmap in cpp

Make changes according to you
Step 5) Compile the CPP code in main folder and give a suitable name for the executable file (because it will be the name of your package/command)
Cmd: cd ..
g++ Balanced_HMap.cpp -o BalancedMap

Step 6) Move the compiled binary to usr/local/bin
Cmd: mv BalancedMap usr/local/bin

Step 7) Build the package and install it
Cmd: dpkg-deb --build Balanced_Hashmap
sudo dpkg -i Balanced_Hashmap.deb

Step 8) Check if the installed package is working
Cmd: BalancedMap

If the steps are correctly followed then you should see following on the output screen:

Value for banana: 20
Bucket 0:
Bucket 1: date: 40 | elderberry: 50 | grape: 70 |
Bucket 2: fig: 60 | lemon: 100 | mango: 50 |
Bucket 3: banana: 20 | dragonfruit: 80 |
Bucket 4: kiwi: 90 |
Bucket 5: cherry: 30 | apricot: 60 |
Bucket 6: jackfruit: 70 |
Bucket 7:
Bucket 8: apple: 10 | honeydew: 80 |
Bucket 9:

End!

14. Create Debian packages.

(multiple modules/code packaging of java/c/cpp).
Pull or take any FOSS project and one feature and then create the package on suitable OS
Compare RPM packaging with Debian packaging (on answer sheet)

sudo apt update
sudo apt install libcurl4-openssl-dev g++

To complete this task, you can package a feature or module of an open-source project
written in Java, C, or C++ into a Debian package. For simplicity, let's use a popular
open-source C++ project—CURL (a command-line tool for transferring data with URLs).
We'll package a single feature of CURL, such as curl_easy_perform, which performs a
file transfer in a simplified manner.

Steps for Creating the Debian Package for CURL

Step 1: Set Up the Directory Structure
Create a directory structure for the Debian package.

mkdir curl-feature-package
cd curl-feature-package
mkdir -p DEBIAN usr/local/bin

Step 2: Download and Prepare the Source Code
Download the CURL source code (assuming CURL is not installed) or use apt source

curl to obtain the source if you're on a Debian-based system.

wget https://curl.se/download/curl-7.82.0.tar.gz
tar -xzf curl-7.82.0.tar.gz
cd curl-7.82.0

1. Navigate to the specific feature code you want to use. For this example, we'll
compile a minimal code using the curl_easy_perform function.

Step 3: Create a Simple C++ File (Feature Extraction)
Create a new C++ file, curl_feature.cpp, and add the
curl_easy_perform
Functionality.

```
#include <iostream>
#include <curl/curl.h>
int main() {
CURL *curl = curl_easy_init();

if(curl) {
curl_easy_setopt(curl, CURLOPT_URL, "https://example.com");
curl_easy_perform(curl);
curl_easy_cleanup(curl);
} else {
std::cerr << "Failed to initialize CURL" << std::endl;
}
return 0;
}
```

1.
Step 4: Compile the Code
Compile the code to create an executable, assuming dependencies like libcurl are

installed:
g++ curl_feature.cpp -o curl_feature -lcurl

First navigate to root directory
Move the executable to the Debian package directory.

mv curl-feature-package/curl_feature
curl-feature-package/usr/local/bin/

1.
Step 5: Create the Control File
Go to the DEBIAN folder in curl-feature-package and create a control file with package
Metadata.

Cd DEBIAN
gedit control
1.
Add the following content to the control file:
Package: curl-feature-package
Version: 1.0
Section: utils
Priority: optional
Architecture: amd64
Depends: libcurl4
Maintainer: Your Name <your.email@example.com>
Description: A feature package for CURL using curl_easy_perform

2.
Step 6: Build the Debian Package
Navigate to the root directory of curl-feature-package.
cd ..

1.
Build the package using dpkg-deb:

dpkg-deb --build curl-feature-package

2. This will create curl-feature-package.deb in the current directory.
Step 7: Install and Test the Package
Install the package using dpkg:
sudo dpkg -i curl-feature-package.deb

1.
Run the installed program to verify it works:
Curl_feature

2.
If set up correctly, this should display the response from https://example.com.
Like following not exact but something
<!doctype html>
<html>
<head>
<title>Example Domain</title>
...
</head>
<body>
** **
<h1>Example Domain</h1>
** This domain is for use in illustrative examples in documents. You may use this**

**domain in literature without prior coordination or asking for permission. **
** More information... **
** **
</body>
</html>

18.Demonstrate the use/features of online Project Management tool & communication tools “slack with ASANA” for managing projects.

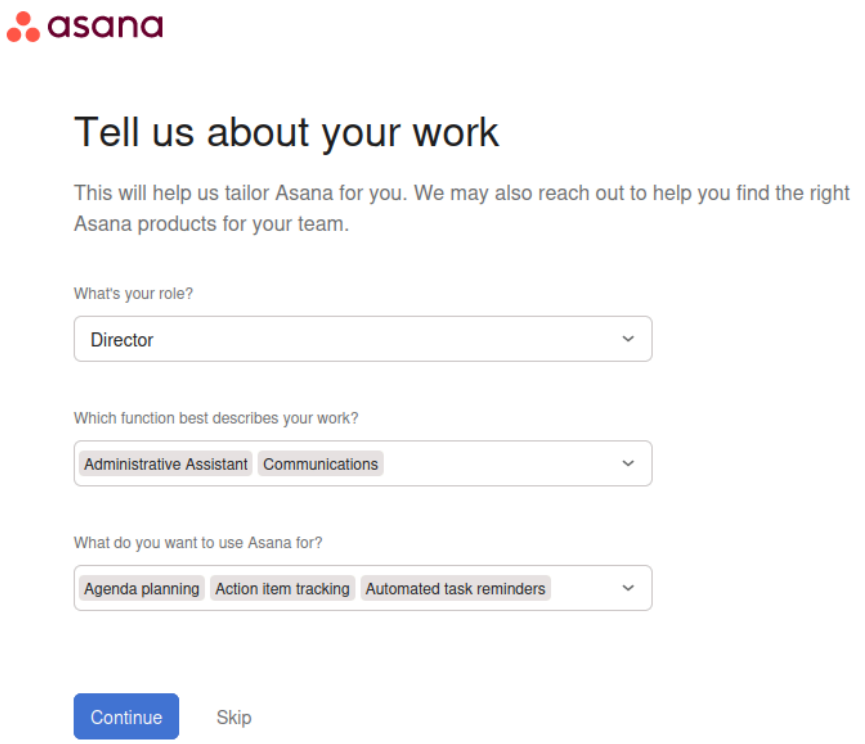
Demonstrate the one foss project for Project planning and scheduling/ Product roadmap and release planning/ Task management and
team collaboration/ Agile and Scrum/Time tracking, cost reporting and budgeting/ Bug tracking on any suitable open source (code) from internet.

Compare asana with phabricator (on answer sheet)

Part 1: Setting Up Slack and Asana Integration

Step 1: Create a Slack Account

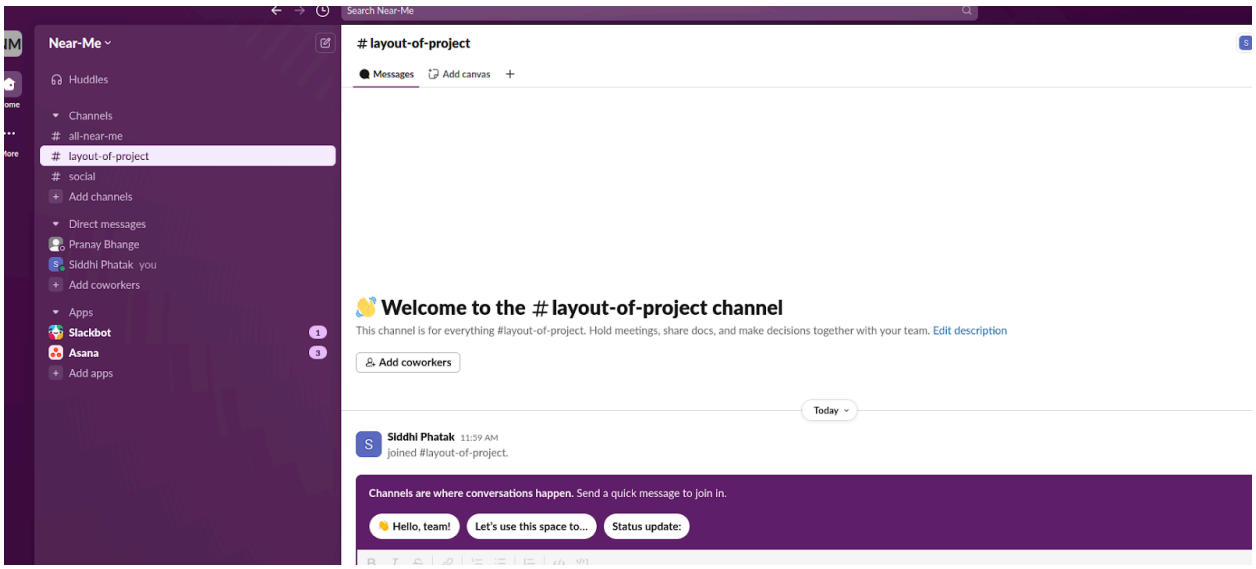
1. Visit Slack’s website.
2. Sign up using your email, or log in if you already have an account.
3. After signing up, create a workspace for your project. You can name it after your team or project.



Follow the initial steps for creating account on asana

Step 2: Set Up Channels in Slack

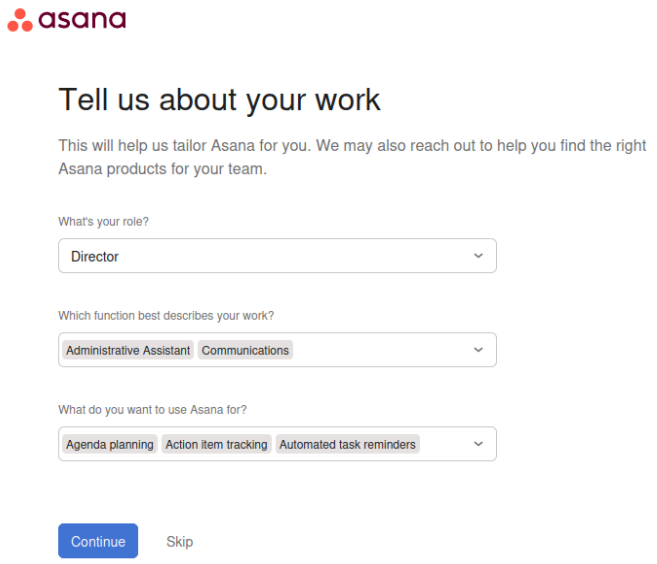
1. In your Slack workspace, click on **Add channels** (e.g., #general, #project-updates, #bugs).
2. Channels allow you to organize discussions based on project parts or team functions.

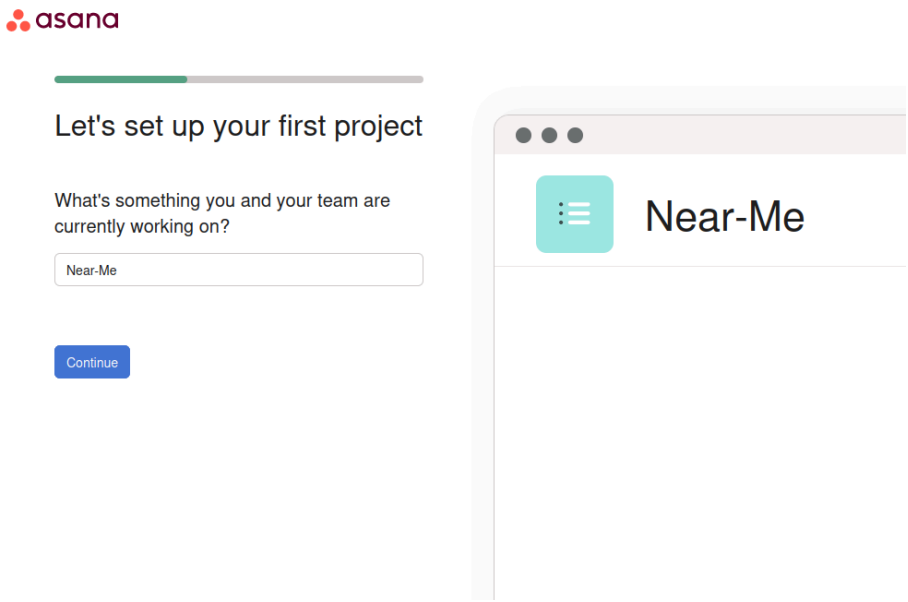


Step 3: Create an Asana Account

1. Visit Asana's website.
2. Sign up for a free account using your email, or log in if you already have one.
3. Create a new project in Asana (e.g., "Project Alpha").

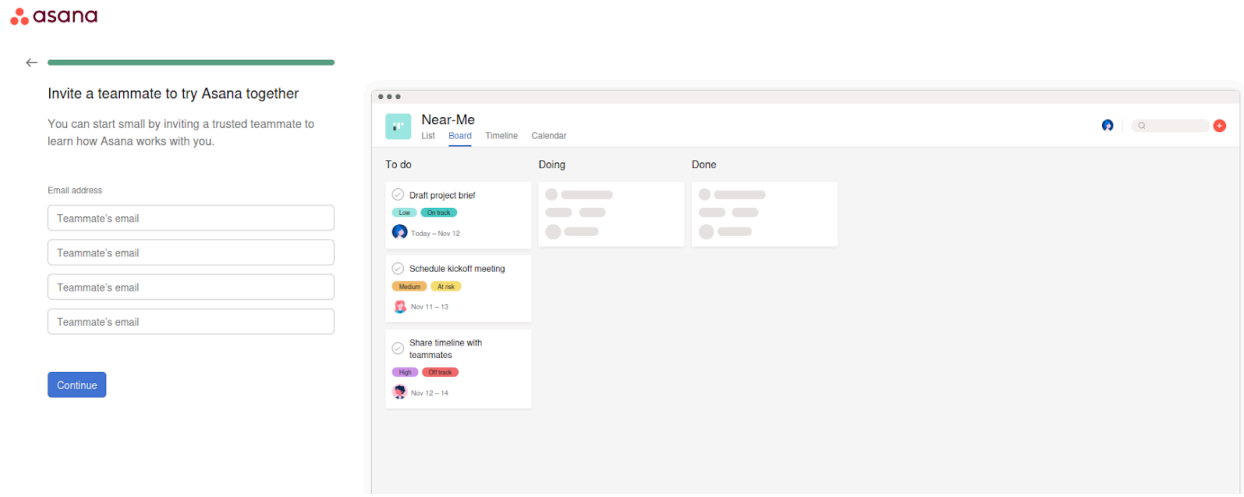
Follow the initial steps for creating account on asana





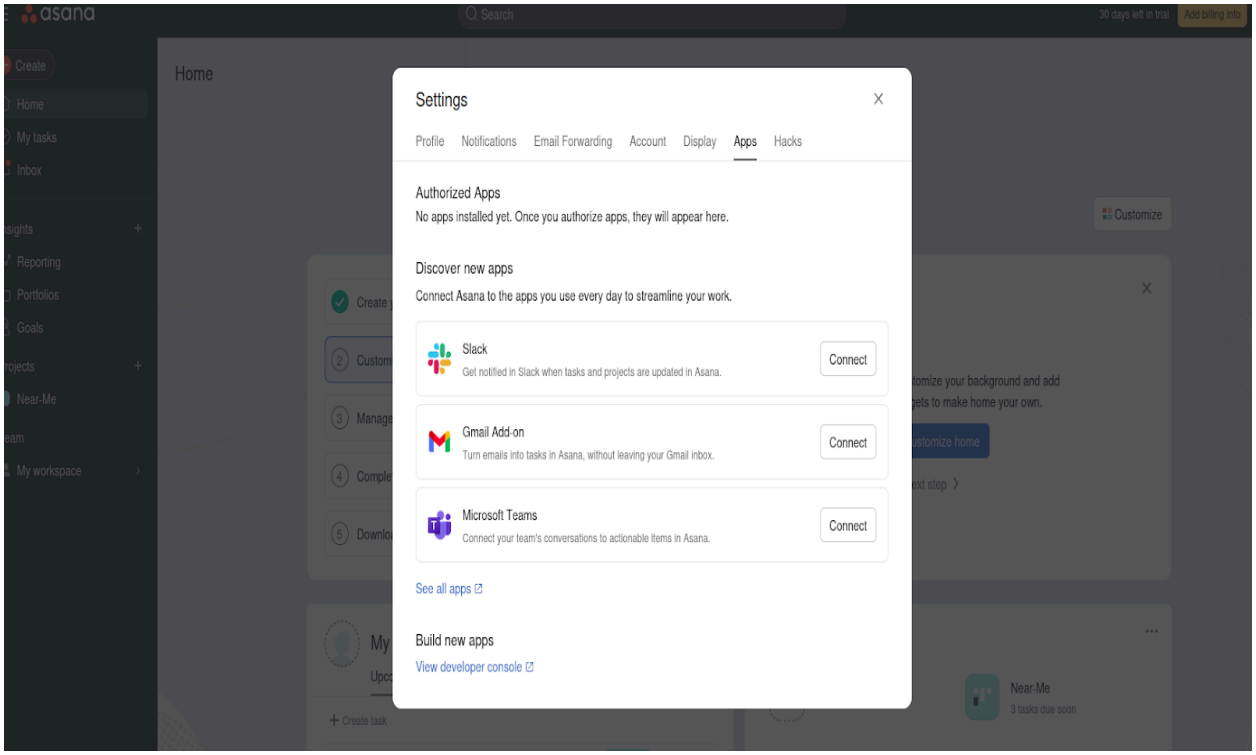
Step 4: Invite Team Members to Asana

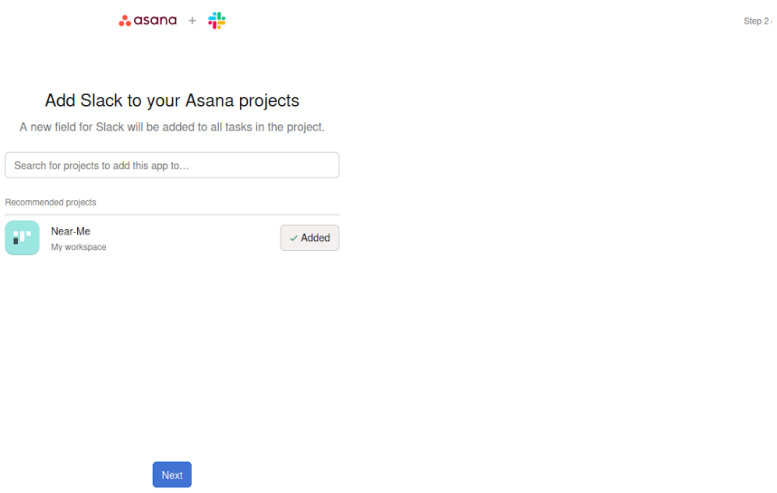
1. In your Asana project, go to the project dashboard.
2. Click on **Invite** and add the email addresses of your team members.



Step 5: Integrate Asana with Slack

1. In Slack, go to **Apps** on the left sidebar, search for “Asana,” and click on it.
2. Authorize Slack to connect with Asana by following the on-screen instructions.
3. Once connected, use the command `/asana` in Slack to create tasks or link to your Asana workspace directly from Slack.

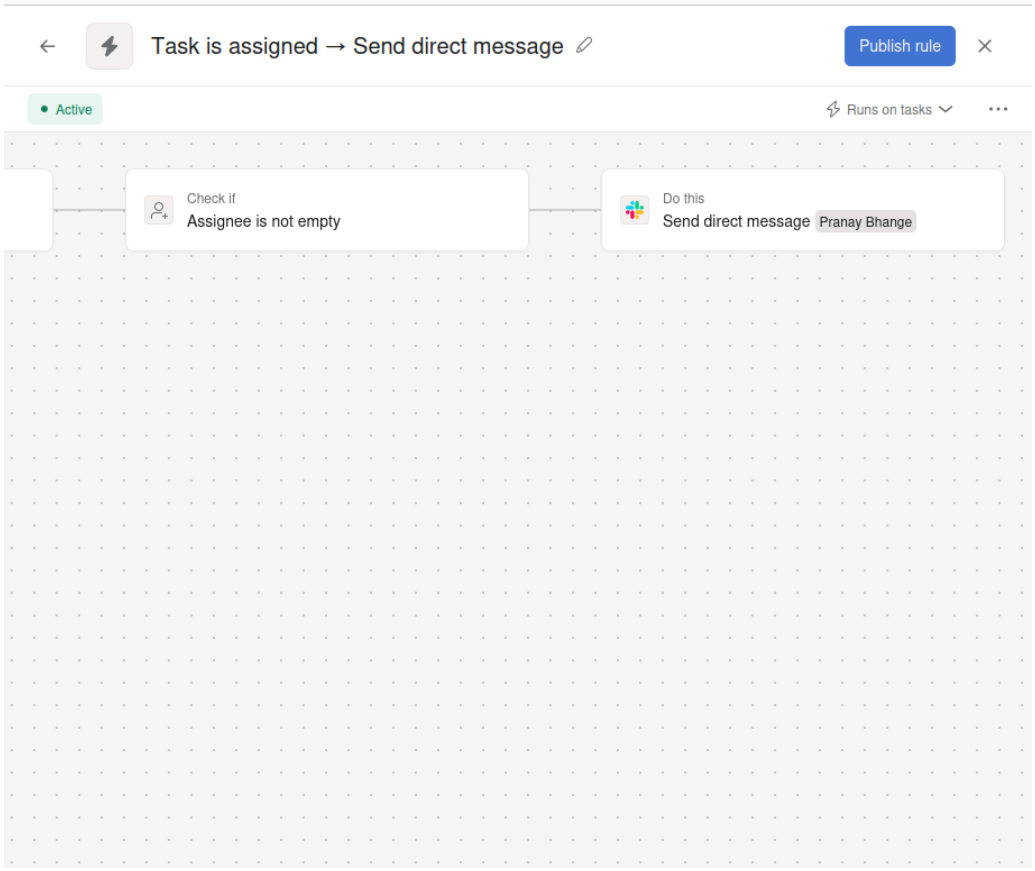




Part 2: Using Asana and Slack for Project Management

Step 6: Create and Assign Tasks in Asana

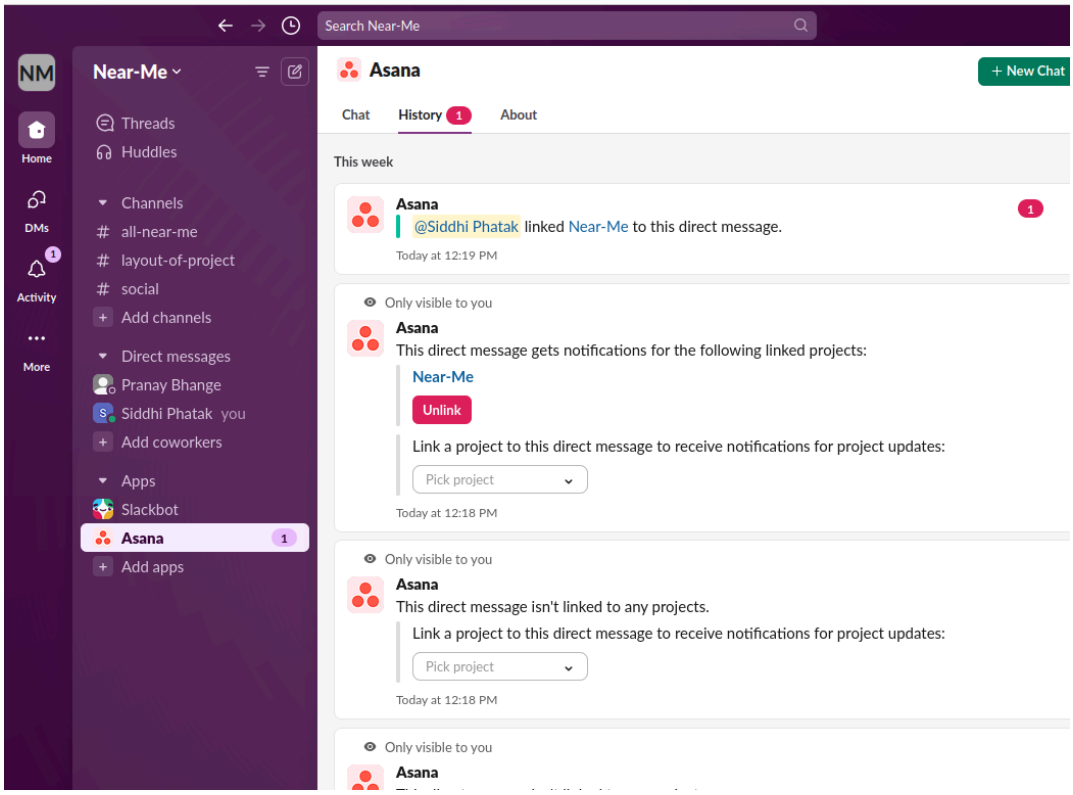
1. Go to your project in Asana, and click on **Add Task**.
2. Enter the task's details, set due dates, assign team members, and add any notes or files.
3. Tasks created in Asana will send notifications to Slack if you've set up the integration.



Create a rule to notify on slack

Step 7: Use Slack for Communication

1. You can discuss Asana tasks, make quick decisions, and share updates within Slack channels.
2. Notify team members by tagging them (e.g., `@username`) in the relevant Slack channel.



Part 3: OpenProject - An Open-Source Project Management Tool

Step 1: Set Up OpenProject

1. Visit any open source project or use your own project.
2. You can use OpenProject in the cloud (hosted by OpenProject) or install it on your system.
3. Create a new project after logging in.

Step 2: Create Tasks and Set Up a Timeline

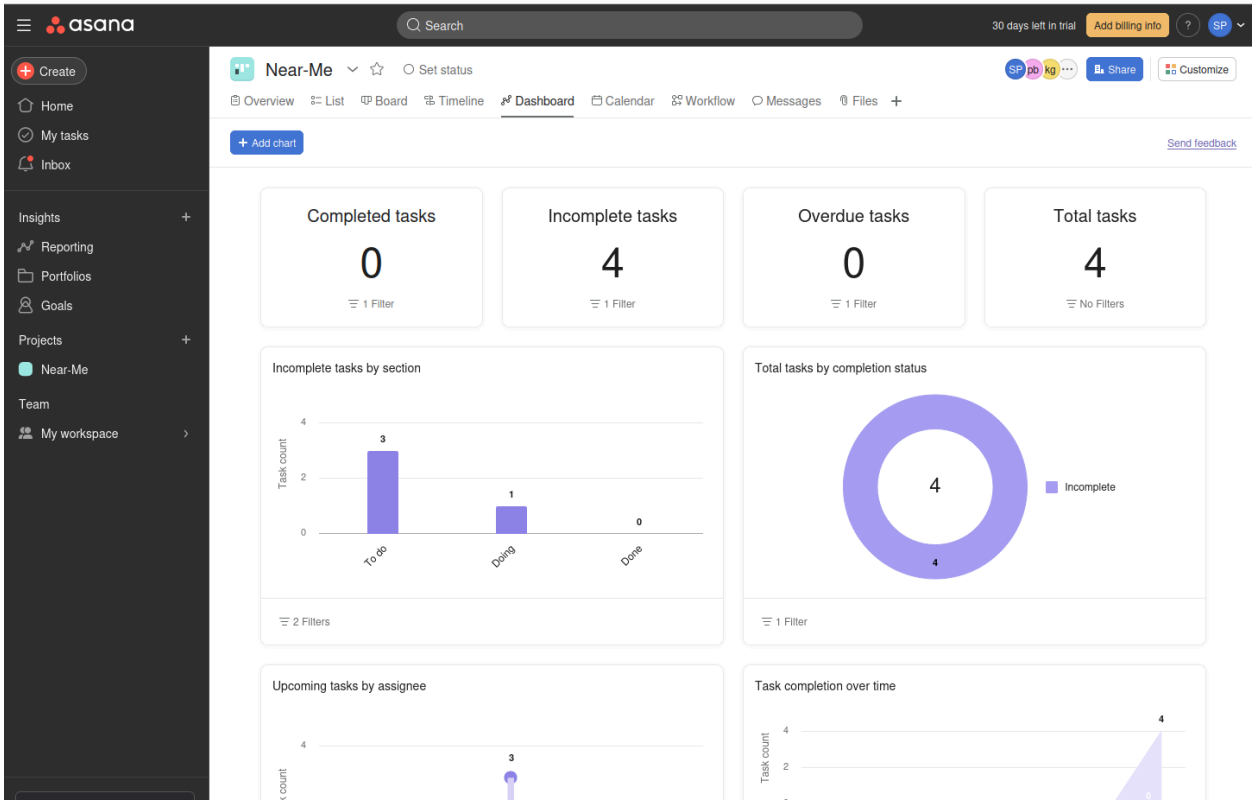
1. In OpenProject, click on **Work packages** to create tasks and assign them to team members.

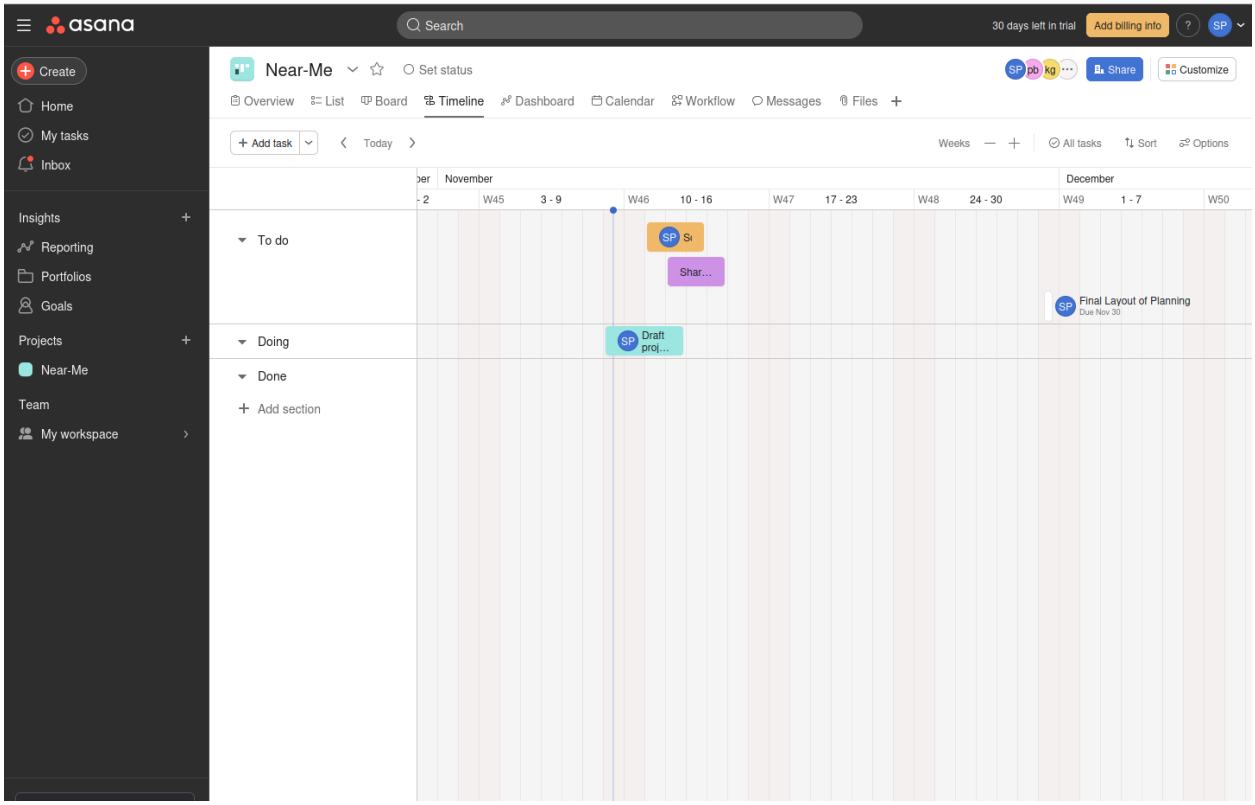
Step 3: Enable Agile and Scrum Boards

1. Go to **Boards** in OpenProject to set up a Kanban or Scrum board.
2. Use the board to manage sprints and move tasks between stages (e.g., Backlog, In Progress, Done).

Step 4: Track Time and Costs

1. For time tracking, go to the **Time & Costs** section, where you and your team can log hours.


asana
Search
30 days left in trial
Add billing info
Create
Home
My tasks
Inbox
Insights
Reporting
Portfolios
Goals
Projects
Near-Me
Team
My workspace
Near-Me
Set status
Share
Customize
Overview
List
Board
Timeline
Dashboard
Calendar
Workflow
Messages
Files
Add chart
Send feedback
Completed tasks
0
1 Filter
Incomplete tasks
4
1 Filter
Overdue tasks
0
1 Filter
Total tasks
4
No Filters
Incomplete tasks by section
Task count
To do
Doing
Done
2 Filters
Total tasks by completion status
Incomplete
1 Filter
Upcoming tasks by assignee
Task completion over time


asana
Search
30 days left in trial
Add billing info
Create
Home
My tasks
Inbox
Insights
Reporting
Portfolios
Goals
Projects
Near-Me
Team
My workspace
Near-Me
Set status
Share
Customize
Overview
List
Board
Timeline
Dashboard
Calendar
Workflow
Messages
Files
Add task
Today
Weeks
All tasks
Sort
Options
November
December
W45
3 - 9
W46
10 - 16
W47
17 - 23
W48
24 - 30
W49
1 - 7
W50
To do
Doing
Done
Add section
Final Layout of Planning
Due Nov 30
Draft proj...

Create or upload any open source project on asana. Add team members ,timelines,tasks,add comments ,create flows on the user interface.

Use all the features available.

Features on asana and slack may vary as we install applications on the system.

Part 4: Compare Asana and Phabricator

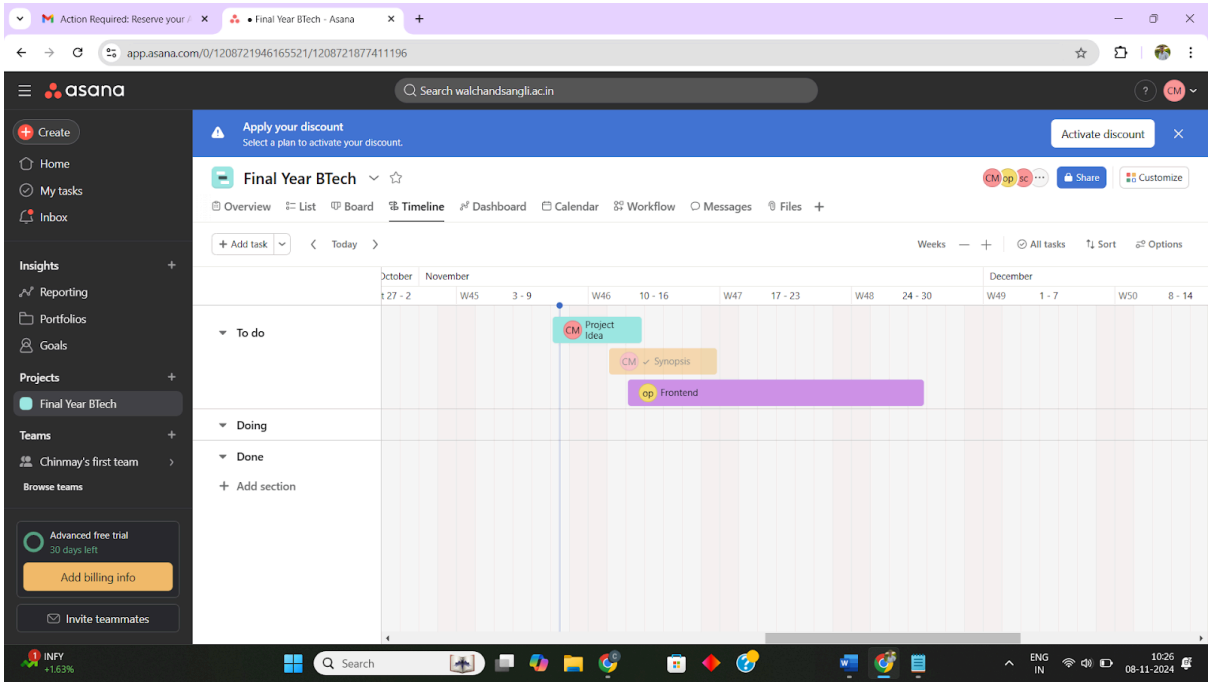
Comparison: Asana vs. Phabricator

Feature	Asana	Phabricator
Project Planning and Scheduling	Timeline and Gantt chart views for planning	Limited Gantt chart functionality, more suitable for task tracking
Product Roadmap and Release	Supports roadmaps, task dependencies, and milestones	Primarily focused on task and issue tracking, with limited roadmap tools
Task Management	Excellent for managing tasks, with dependencies and custom fields	Strong task management, ideal for developers and code-related tasks
Agile and Scrum	Supports Agile project workflows, including Kanban	Supports Agile boards, but not as intuitive for non-technical users
Time Tracking	Time-tracking via integrations like Harvest	No built-in time tracking, can be achieved via extensions
Cost Reporting and Budgeting	Limited budgeting features; relies on integrations	Basic reporting, with limited cost-tracking capabilities
Bug Tracking	Basic bug-tracking support	Advanced bug-tracking features, specifically tailored for developers
Integrations	Integrates with Slack, Google, GitHub, and more	Supports GitHub, GitLab, Slack, and other developer tools

Best for	General project management across various industries	Technical and development-focused teams

19. <u>Demonstrate the use/features</u> of online Project Management tool & communication tools "asana with github" for managing projects.

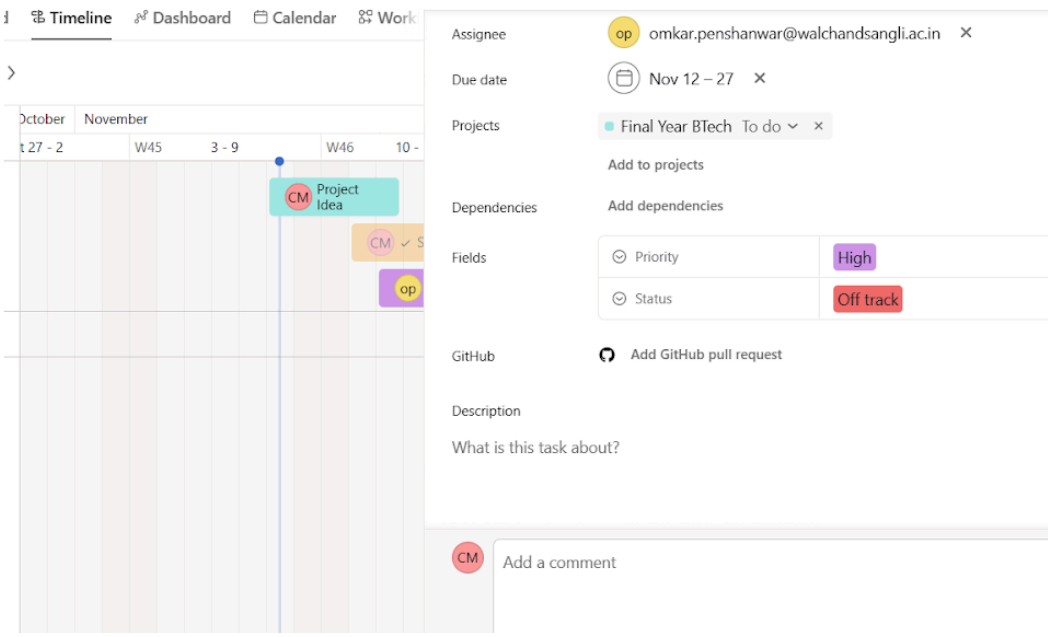
1. Create an account on asana.com
2. Create an one project like your mini project
3. In that project add tasks related to your project.
4. Add your teammates email address



5. You can add assignee to that task ,prioritise,update the status of the tasks by clicking on that task.
6. You can view this dashboard in different formats like calendar, list , timeline etc
7. Also you can chat there with team members and you can add files for your projects.

For Integrating with github

1. Go to the project and click on the customise options. In that you need to select apps and in that github.
2. Sign in to github account and authorize the github account.
3. After that you have given pull request url then automatically status will update.
4. In specific task option you can see the github option by clicking on that you will be asked for a github url which will be pulled according to that status update.



If anyone wants opensource project url: https://github.com/johnuberbacher/invoice-generator

Feature	Asana	Phabricator
Audience	General teams, cross-functional	Engineering, technical teams
Task Management	Comprehensive task and subtask features	Strong task and bug tracking
Views	List, Kanban, Timeline, Calendar	Workboard (Kanban)
Code Review	Limited	Advanced (Differential tool)
Reporting	Dashboards, workload management	Limited, but extendable via API
Collaboration	Real-time collaboration, file sharing, notifications	Code-related discussions, inline commenting, API integration
Integrations	Extensive third-party integrations (Slack, Zoom, Microsoft Teams, Google Workspace, etc.)	Strong API, limited native integrations, VCS support (Git, SVN)

Assignment 22: CMS Software: Drupal

Demonstrate the use/features of CMS software: "Drupal".
Create users and show how Drupal manages contents of web sites for a client. Also implement the working of core features of Drupal.
Compare it with other CMS like schoology/(on answer sheet)

A **CMS**, or **Content Management System**, is a software application or platform that allows users to create, manage, and modify content on a website without needing specialized technical knowledge. In simple terms, it's a tool that helps people easily update and maintain a website by providing a user-friendly interface. You can add text, images, videos, and other content to a site without needing to know how to code.Some popular CMS examples include **WordPress**, **Joomla**, and **Drupal**. Drupal installation - ubuntu:

Updated Step-by-Step Guide to Install Drupal on Ubuntu (Home Directory)

Step 1: Update Your System

```
sudo apt update

sudo apt upgrade -y
```

Step 2: Install Apache Web Server

```
sudo apt install apache2 -y

sudo systemctl enable apache2

sudo systemctl start apache2
```

Step 3: Install PHP

```
sudo apt update
sudo apt install software-properties-common -y
sudo add-apt-repository ppa:ondrej/php
sudo apt update

sudo apt install php8.3 php8.3-cli php8.3-fpm php8.3-mysql php8.3-gd
php8.3-xml php8.3-mbstring php8.3-zip php8.3-curl php8.3-xmlrpc
php8.3-imagick libapache2-mod-php8.3 -y

php -v
```

Step 4: Install MySQL Database

```
sudo apt install mysql-server -y
```

```
sudo mysql_secure_installation
```

```
sudo mysql -u root -p
```

Create the database and user:

sql

```
CREATE DATABASE drupaldb;
```

```
CREATE USER 'drupaluser'@'localhost' IDENTIFIED BY 'your_password';
```

```
GRANT ALL PRIVILEGES ON drupaldb.* TO 'drupaluser'@'localhost';
```

```
FLUSH PRIVILEGES;
```

```
EXIT;
```

Step 5: Install Drupal in Your Home Directory

```
cd ~
```

```
wget https://www.drupal.org/download-latest/tar.gz
```

```
tar -xvzf tar.gz
```

```
mv drupal-* ~/drupal
```

```
sudo chown -R www-data:www-data ~/drupal
```

```
sudo chmod -R 755 ~/drupal
```

Step 6: Configure Apache for Your Drupal Installation in Home Directory

Edit Apache configuration to point to your home directory:

```
sudo nano /etc/apache2/sites-available/000-default.conf
```

Modify the `DocumentRoot` and `<Directory>` sections:

```
DocumentRoot /home/your_username/drupal


<Directory /home/your_username/drupal>
    AllowOverride All
    Require all granted
</Directory>
```

Enable mod_rewrite:

```
sudo a2enmod rewrite
```

Restart Apache:

```
sudo systemctl restart apache2
```

Modify permissions:

sudo chown -R www-data:www-data /home/your_username/drupal

sudo chmod -R 755 /home/your_username/drupal

Step 7: Complete Drupal Installation in the Browser

Open your web browser and go to:

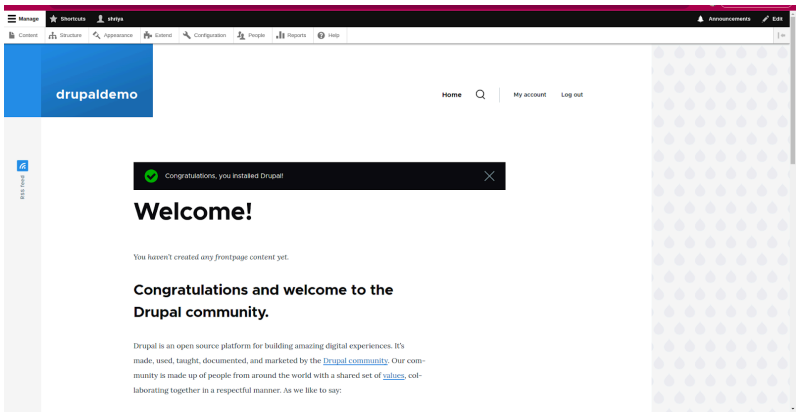
`http://localhost`

Proceed with the Drupal setup:

1. Choose the "Standard" installation profile.
2. Enter the database details you configured earlier:
    - **Database type**: MySQL
    - **Database name**: `drupaldb`

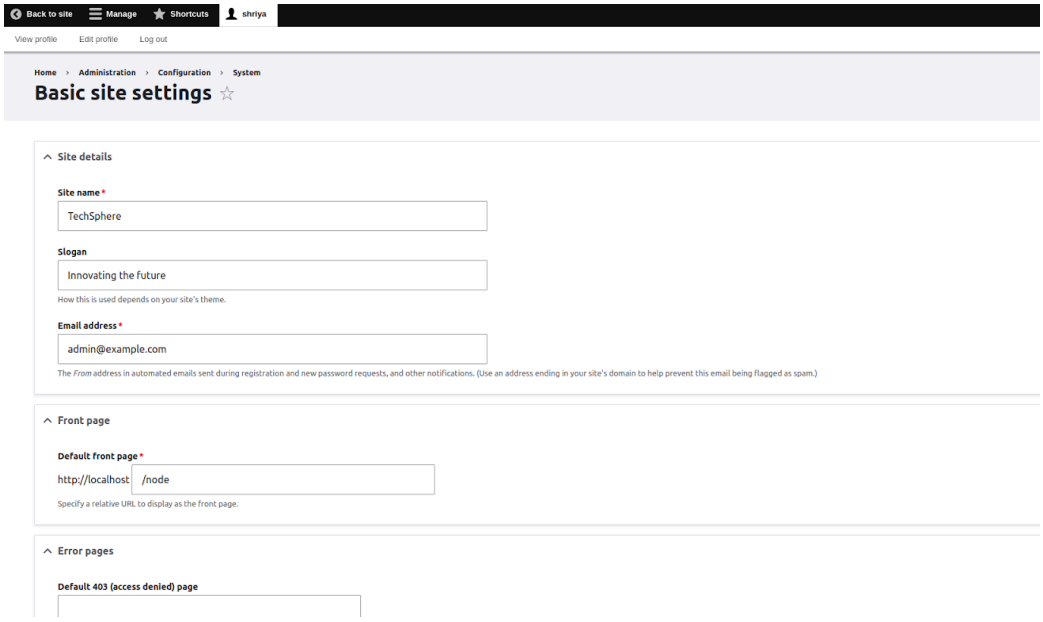
- **Database username**: `drupaluser`
- **Database password**: `your_password`

3. Configure your site details (site name, admin username, etc.)



1: Set Up the Website Name and Basic Settings

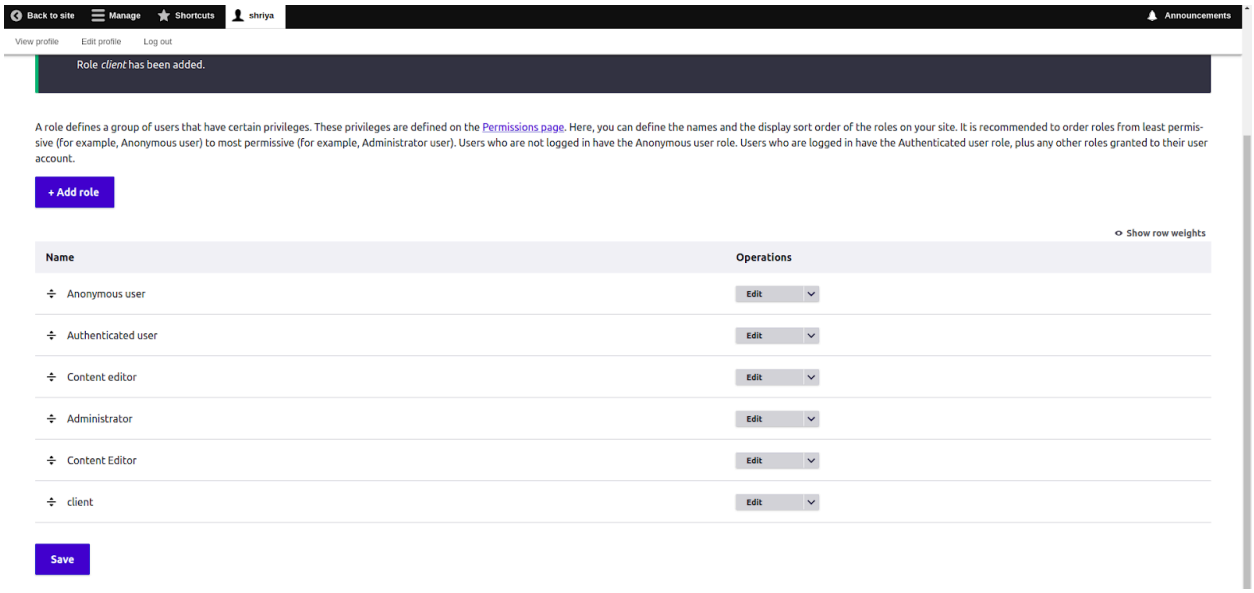
1. **Log in to Drupal**:
   - Go to `http://localhost/user/login`
   - Go to **Configuration** → **System** → **Basic site settings**.
   - Give any names you want. I have choosed technical website where they post about projects/blogs



2: Create User Roles and Permissions

a. Create New User Roles

1. Go to **People** → **Roles**.
2. Click **Add role** and create the following roles: (add any roles acc to your website)
    - **Content Editor**: Can create and edit content.
    - **Client**: Can view certain content not available to the public.

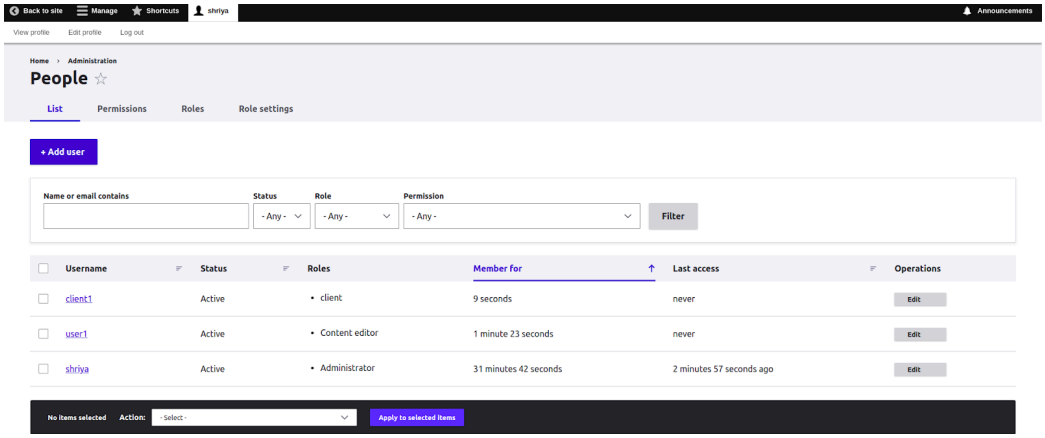


b. Assign Permissions to Each Role

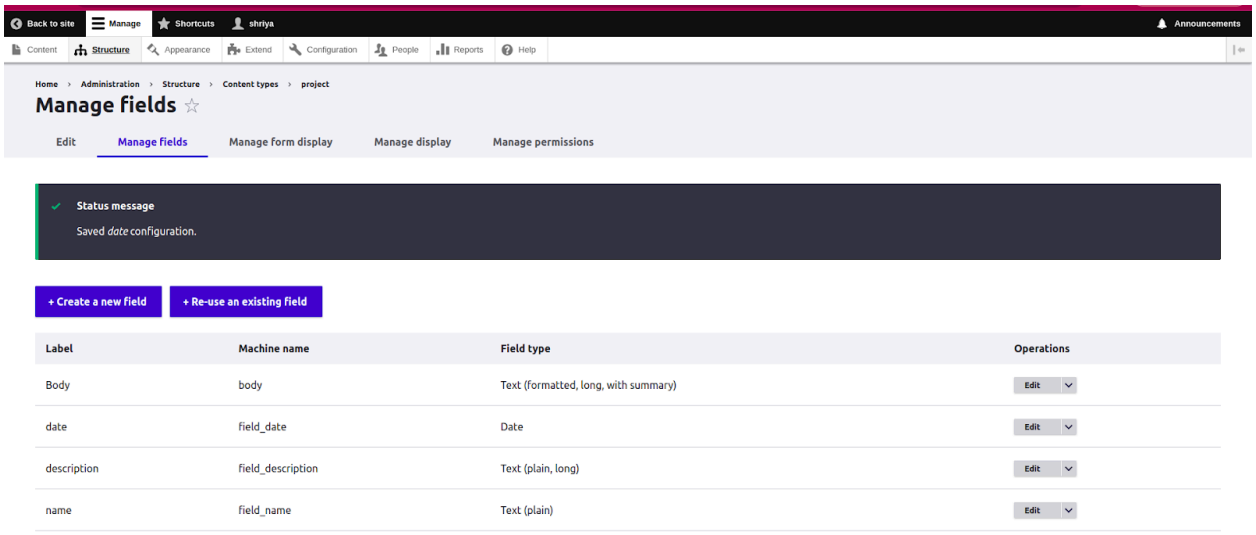
1. Go to **People** → **Permissions**.
2. Assign permissions as follows:

c. Create Users and Assign Roles

1. Go to **People** → **Add user**.
2. Create sample users:
    - **Editor abc**(Role: Content Editor)
    - **Client xyz** (Role: Client)



Add fields.



3: Create Content Types

a. Create a Custom Content Type: "Project"

1. Go to **Structure → Content types → Add content type**.
2. **Name**: Project
    - **Description**: Used to showcase TechSphere projects.
3. Click **Save and manage fields**.

b. Add Fields to the "Project" Content Type

c. Configure Content Type Settings

- Go to the **Manage display** tab and adjust the field order as desired.

4: Add Content

a. Create Articles

1. Go to **Content** → **Add content** → create what you want

5: you can go on exploring other options like themes, etc

6: Demonstrate User Login and Content Management

a. Test Different User Roles

1. Log out as admin and log in as **Editor** .
    - Demonstrate adding and editing content.
2. Log in as **Client** .
    - Show access to specific client-only content.

b. Use Drupal's Core Features

- Demonstrate creating a **Content View**:
    1. Go to **Structure** → **Views** → **Add view**.
    2. Create a view for displaying a list of projects.
    3. Enable **Page** and set path to `/projects`.

7: Final Touches and Demonstration

1. **Showcase the Homepage**:
    - Highlight menus, blocks, and recent content.
2. **Demonstrate Content Search**:
    - Use the search bar to find articles or projects.
3. **Show User Access Control**:
    - Log in/out to demonstrate different role permissions.
4. **Discuss the Extensibility**:
    - Mention how to extend Drupal using additional modules (e.g., SEO, Social Media).

Comparison: schoology vs drupal →

Feature	Drupal	Schoology
Purpose	Open-source **Content Management System (CMS)** for building websites and managing content	**Learning Management System (LMS)** focused on education and online learning
Use Case	Best for **organizations** needing complex, customizable websites (e.g., universities, enterprises)	Ideal for **schools and educators** needing a platform for classroom management and student engagement
Flexibility & Customization	Highly flexible, supports extensive customization through **modules and themes**	Limited customization, focused on ease of use with pre-built **educational tools**
Content Management	Supports complex content types, taxonomies, and **multilingual sites**	Focuses on **course content**, assignments, quizzes, and student assessments
User Roles & Permissions	Granular control over user roles and permissions for **multiple user types**	Predefined user roles like **administrators, teachers, students, and parents**
Ease of Use	Steeper learning curve, requires some technical knowledge for advanced features	User-friendly interface designed for **teachers and students** with minimal setup
Integration & Extensibility	Highly extensible with **APIs and third-party integrations** (e.g., CRM, ERP)	Built-in integrations with **educational tools** like Google Workspace, Microsoft Teams, and Zoom
Scalability & Performance	Designed to scale, ideal for **high-traffic websites** with caching and optimization options	Cloud-based infrastructure, optimized for **educational institutions** of all sizes

Assignment 23

1. FTP Server:

1. Install FTP Server:

Run the following commands to install it:

sudo apt update
sudo apt install vsftpd

This will install the FTP server.

2. Create a New User for FTP Access

   You can create a new user that will be able to access the FTP server using:

   sudo adduser newuser

   Replace newuser with the desired username.

   Set the password for the new user:

   sudo passwd newuser

3. Configure vsftpd for FTP Access

   Now, you need to configure vsftpd to allow FTP access.
   Open the vsftpd configuration file:

   sudo nano /etc/vsftpd.conf

   Modify the configuration to allow the user to connect:
   Make sure the following lines are present and set as shown:

```
listen=YES
listen_ipv6=NO
anonymous_enable=NO
local_enable=YES
write_enable=YES
chroot_local_user=YES
allow_writeable_chroot=YES
```

Explanation of settings:

listen=YES: Allows vsftpd to listen on IPv4.

listen_ipv6=NO: Disable IPv6 listening.

anonymous_enable=NO: Disables anonymous FTP login (ensure only registered users can log in).

local_enable=YES: Allows local users to log in.

write_enable=YES: Allows write access (uploading files).

chroot_local_user=YES: Chroots (locks) users to their home directory for security.

allow_writeable_chroot=YES: Allows writing in the chrooted environment.

Save the file and exit.

4. Restart vsftpd Service using:

   ```
   sudo systemctl restart vsftpd
   ```

5. Allow FTP Through the Firewall (if applicable)

   If you're using a firewall like UFW (Uncomplicated Firewall), you will need to allow FTP traffic (ports 20 and 21) through.

   ```
   sudo ufw allow 20/tcp
   sudo ufw allow 21/tcp
   sudo ufw reload
   ```

6. Verify FTP Service is Running

To check if vsftpd is running, use the following command:

sudo systemctl status vsftpd

You should see something like:

● vsftpd.service - VSFTPD FTP server
  Loaded: loaded (/lib/systemd/system/vsftpd.service; enabled; vendor preset: enabled)
  Active: active (running) since ...

7. Log in Using FTP

   Once the FTP server is running, you can test logging in with your new user.

   Connect using FTP:

   ftp localhost

   Enter the username: Enter the new username you created (newuser).

   Name (localhost:your-username): newuser

   Enter the password: Type the password for newuser.

8. Upload/Download Files

   Once logged in, you can use FTP commands to upload or download files:

   To list files in the current directory:

   ls

   To upload a file:

put filename

To download a file:
get filename

To change directories:
cd directory_name

To exit the FTP session:
bye

9. Secure FTP (Optional)

By default, FTP is insecure as it sends data (including passwords) in plaintext. If security is a concern, you should consider setting up FTPS (FTP Secure) or SFTP (SSH File Transfer Protocol), which encrypts the communication.

To set up FTPS or SFTP, you would need additional configuration (e.g., SSL certificates for FTPS or enabling SSH for SFTP).
Summary of Commands

Install vsftpd:

sudo apt install vsftpd

Create a new user:

sudo adduser newuser
sudo passwd newuser

Configure vsftpd (Edit /etc/vsftpd.conf):

sudo nano /etc/vsftpd.conf

Restart vsftpd:

sudo systemctl restart vsftpd

Allow FTP through UFW firewall:

```
sudo ufw allow 20/tcp
sudo ufw allow 21/tcp
sudo ufw reload
```

Test FTP login:

```
ftp localhost
```

2. Telnet Server:

1. Create a New User

Add a new user:

```
sudo adduser newuser
```

Replace newuser with the username you want to create.

Set a password for the new user:

```
sudo passwd newuser
```

2. Check or Modify the Telnet Configuration

Open the configuration file:

```
sudo nano /etc/xinetd.d/telnet
```

Configure the Telnet service to allow the user:
Ensure that the file has the following basic settings:

```
service telnet
{
   disable = no
   socket_type = stream
   wait = no
```

```
    user = root
    server = /usr/sbin/in.telnetd
    log_on_failure += USERID
}
```

Restart xinetd:
After making changes, restart xinetd to apply the new settings:

```
sudo systemctl restart xinetd
```

3. Allow User Through Firewall

If you're using a firewall like ufw, you might need to allow the Telnet port (port 23) through the firewall:

```
sudo ufw allow 23/tcp
sudo ufw reload
```

4. Check User Permissions

If the Telnet service is configured to allow logins and you've created the new user, ensure that the user has the necessary permissions to log in. For this, check /etc/passwd and /etc/group files to ensure the user isn't restricted from logging in.

Check the /etc/passwd file:

```
cat /etc/passwd | grep newuser
```

This should show an entry for newuser.

5. Log in Using Telnet

You should be able to log in using Telnet with the new user credentials:

```
telnet localhost
```

Assignment 26

> Configure and demonstrate the use of NIS and NFS. (on centos/rpm based OS VM/container) Show the imp steps and file name of configurations. (on answer sheet) Create 5 users and make two groups, demonstrate the NIS and NFS concepts by example on LAN connected linux os.

Use CentOs / Fedora

1. What is NIS?

NIS stands for 'Network Information Service' and it is a centrally administrated system featuring the distribution of user account and group information to different hosts on the same network.

For instance, a region with numerous computers (which can be termed clients) is locally networked to a central computer termed as the server. Instead of creating accounts on each individual client, they are administratively created on a centralized server which clients will access any time a user logs in.

2. Setting Up NIS Server and Clients:

On the Server: We have to create a domain (name associated with NIS) and configure all the files necessary for that domain to ensure user and group data can be shared. We also create additional users and groups on the server.

On the Clients: We have individual clients that need to be administered this domain and such clients are also set up to seek information on accounts from the NIS server. The end result is that whenever a user attempts to log into a client, the client makes an inquiry to the NIS server to verify the user account information.

3. What is NFS?

NFS stands for Network File System and it is a file storage system with capability of enabling file and folder sharing across a network such that a specific piece of information can be accessed by more than one computer user.

Consider it as forming a folder on the server that is accessible by all client machines. When NFS is configured, users on any client may access this shared folder as if it were a local folder, reading or writing files as required.

4. Configuring the NFS is done with the NFS Server and exposing a server's directory.

On the NFS Server: A shared directory is created and NFS is configured to provide access to the shared directory for other computers (clients). Management of the shared folder is also included and we define how clients may even interact with some aspects of the shared folder.

On the Clients: We need the API to define a location (which will later be referred to as a "mount point") where the shared folder appears and then map this location to the shared one on the NFS server.

5. The Final Coalescence:

After both NIS and NFS setup, users can access their files in a shared folder on the NFS server from any client machine they logged into. This is essential for setups such as poke or lab where multiple clients are required to get access to shared folders on different computers.

Part 1: Set Up the NIS Server on CentOS/RHEL/Fedora

1. **Install NIS and Related Packages**:

Install the required packages on the NIS server.
bash
Copy code

```
sudo yum install -y ypserv rpcbind
```

2. **Set the NIS Domain Name**:

Set a domain name for NIS (for example, `exampledomain`), and make it persistent.
bash
Copy code
```
sudo vim /etc/sysconfig/network
```

Add this line to the file:
bash
Copy code
```
NISDOMAIN=exampledomain
```

Apply the domain name immediately:
bash
Copy code
```
sudo domainname exampledomain
```

3. **Configure the NIS Server**:

Initialize the NIS database by running `ypinit -m`:
bash
Copy code
```
sudo ypinit -m
```

   - Follow the prompts to configure the NIS master server. This will create NIS maps based on the existing user and group information on the server.
4. **Start and Enable NIS Services**:

Enable and start the NIS and RPC bind services:
bash
Copy code
```
sudo systemctl enable --now ypserv
sudo systemctl enable --now rpcbind
```

5. **Add Users and Groups**:

Create user accounts and groups on the NIS server. These accounts will be available across the network.
bash

Copy code
```
sudo useradd user1
sudo useradd user2
sudo groupadd group1
```

6. **Rebuild NIS Maps After Adding Users or Groups**:

Each time you add or modify users or groups, rebuild the NIS maps:
bash
Copy code
```
sudo make -C /var/yp
```

Part 2: Configure NIS Clients

1. **Install NIS Client Packages**:

On each client, install the necessary packages to connect to the NIS server.
bash
Copy code
```
sudo yum install -y ypbind
```

2. **Set the NIS Domain on Each Client**:

Configure the NIS domain name in `/etc/sysconfig/network`.
bash
Copy code
```
sudo vim /etc/sysconfig/network
```

Add this line to match the NIS domain on the server:
bash
Copy code
```
NISDOMAIN=exampledomain
```

3. **Configure the NIS Client to Use the Server**:

Edit `/etc/yp.conf` on each client to specify the NIS server's IP address and domain name:
bash
Copy code
```
sudo vim /etc/yp.conf
```

- Add a line like this:
bash
Copy code
```
domain exampledomain server <NIS_server_IP>
```

4. **Update `/etc/nsswitch.conf`**:

Tell the system to use NIS for user and group lookups by editing `/etc/nsswitch.conf`:
bash
Copy code
```
sudo vim /etc/nsswitch.conf
```

- Update the following lines to include `nis`:
bash
Copy code
```
passwd:     files nis
shadow:     files nis
group:      files nis
```

5. **Start the NIS Client Service**:

Start and enable the `ypbind` service on each client:
bash
Copy code
```
sudo systemctl enable --now ypbind
```

Part 3: Set Up the NFS Server on CentOS/RHEL/Fedora

1. **Install NFS Server Package**:

On the NFS server, install the `nfs-utils` package:
bash
Copy code
```
sudo yum install -y nfs-utils
```

2. **Create a Shared Directory**:

Set up a directory to share with NFS clients.
bash
Copy code
```
sudo mkdir /shared
sudo chown nobody:nogroup /shared
sudo chmod 755 /shared
```

3. **Configure `/etc/exports`**:

Define which clients can access the shared folder and with what permissions in `/etc/exports`.
bash
Copy code
```
sudo vim /etc/exports
```

Add a line like this to share `/shared` with all clients in a specific network range:
bash
Copy code
```
/shared <client_IP_or_network>(rw,sync,no_root_squash)
```

- Replace `<client_IP_or_network>` with the IP range of your LAN, e.g., `192.168.1.0/24`.

4. **Export the Directory and Start NFS**:

Apply the export settings and start the NFS server:
bash
Copy code
```
sudo exportfs -a
sudo systemctl enable --now nfs-server
```

Part 4: Configure NFS Clients on CentOS/RHEL/Fedora

1. **Create a Mount Point on Each Client**:

Create a directory on each client where the NFS share will appear:
bash
Copy code
```
sudo mkdir /mnt/shared
```

   ○
2. **Mount the NFS Share**:

Mount the NFS shared folder from the server:
bash
Copy code
```
sudo mount <NFS_server_IP>:/shared /mnt/shared
```

   ○
3. **Auto-Mount the NFS Share on Boot**:

To automatically mount the NFS share on boot, add it to `/etc/fstab` on each client:
bash
Copy code
```
sudo vim /etc/fstab
```

   ○

Add this line to mount the shared folder on boot:
bash
Copy code
```
<NFS_server_IP>:/shared /mnt/shared nfs defaults 0 0
```

   ○

Testing NIS and NFS

- **NIS Testing**: Try logging in as one of the NIS users on a client (e.g., `su - user1`). This confirms that the client is correctly retrieving user data from the NIS server.
- **NFS Testing**: On a client, navigate to `/mnt/shared` and try creating or accessing a file to confirm that the NFS share is working.

Key Configuration Files on RPM-based OS

- **NIS Server**:
  - `/etc/sysconfig/network` – Set NIS domain.
  - `/var/yp/Makefile` – Define shared files for NIS.
- **NIS Client**:
  - `/etc/yp.conf` – NIS server details.
  - `/etc/nsswitch.conf` – Specify NIS for user/group lookups.
- **NFS Server**:
  - `/etc/exports` – Define shared directories and permissions.

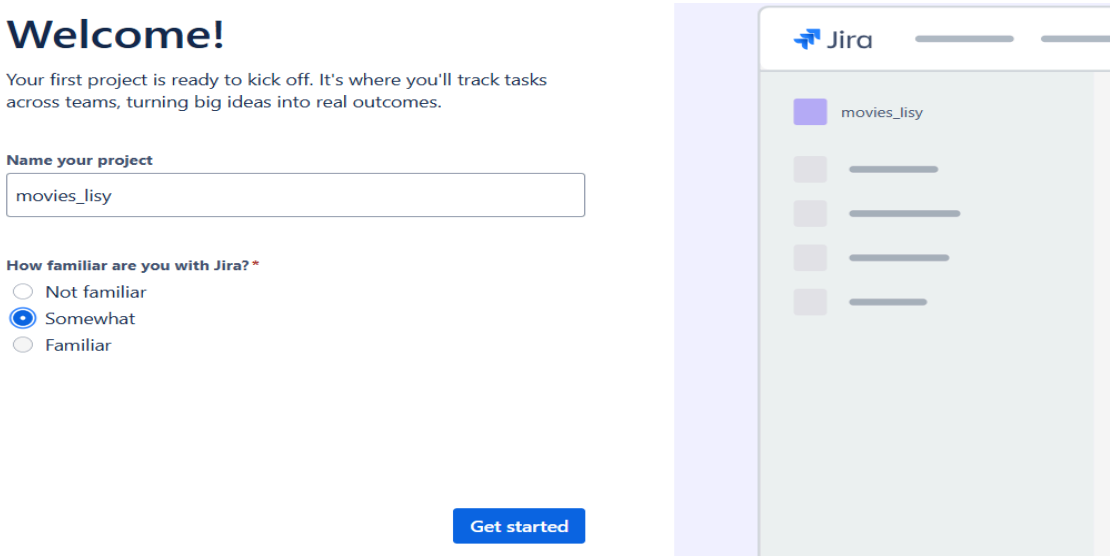
Reference:
NIS:https://www.server-world.info/en/note?os=CentOS_7&p=nis&f=1
NFS:https://medium.com/@shubnimkar/how-to-setup-nfs-server-on-centos-7-rhel-7-db59f32170f7

Assignment 28

Create a Jira Account:

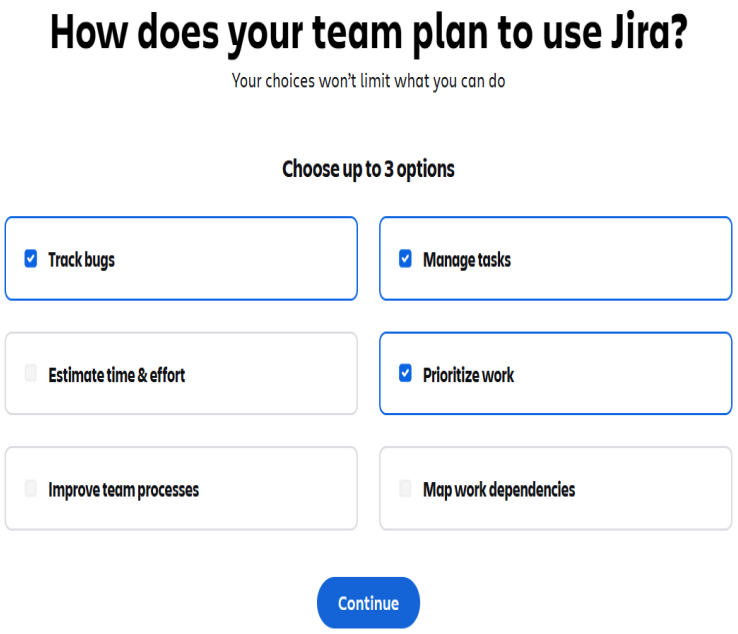
- Go to https://id.atlassian.com/ to create your Jira account. If the site does not automatically take you to the next steps, use this alternative link.
- Once your account is created, you'll be directed to a site setup page where you can provide a site name (random names work fine).

Than choose software development as it give relevant features



Choose Your Project Type:

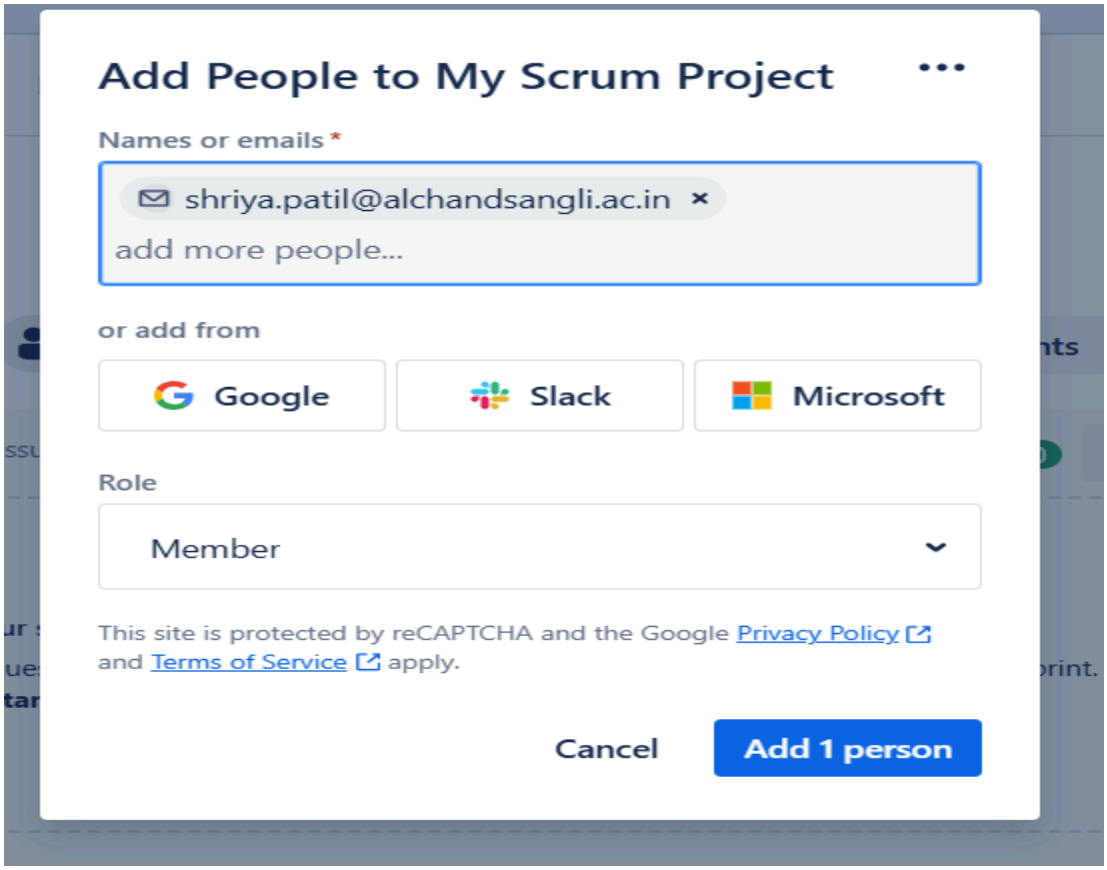
- Select **Software Development** to unlock relevant features designed for managing development workflows. And in feature select the below features



Set Up Your Project:

- Enter a suitable project name, then proceed to configure it.

Add team members by selecting the **+** icon at the top of the screen and sending invitations to team members.

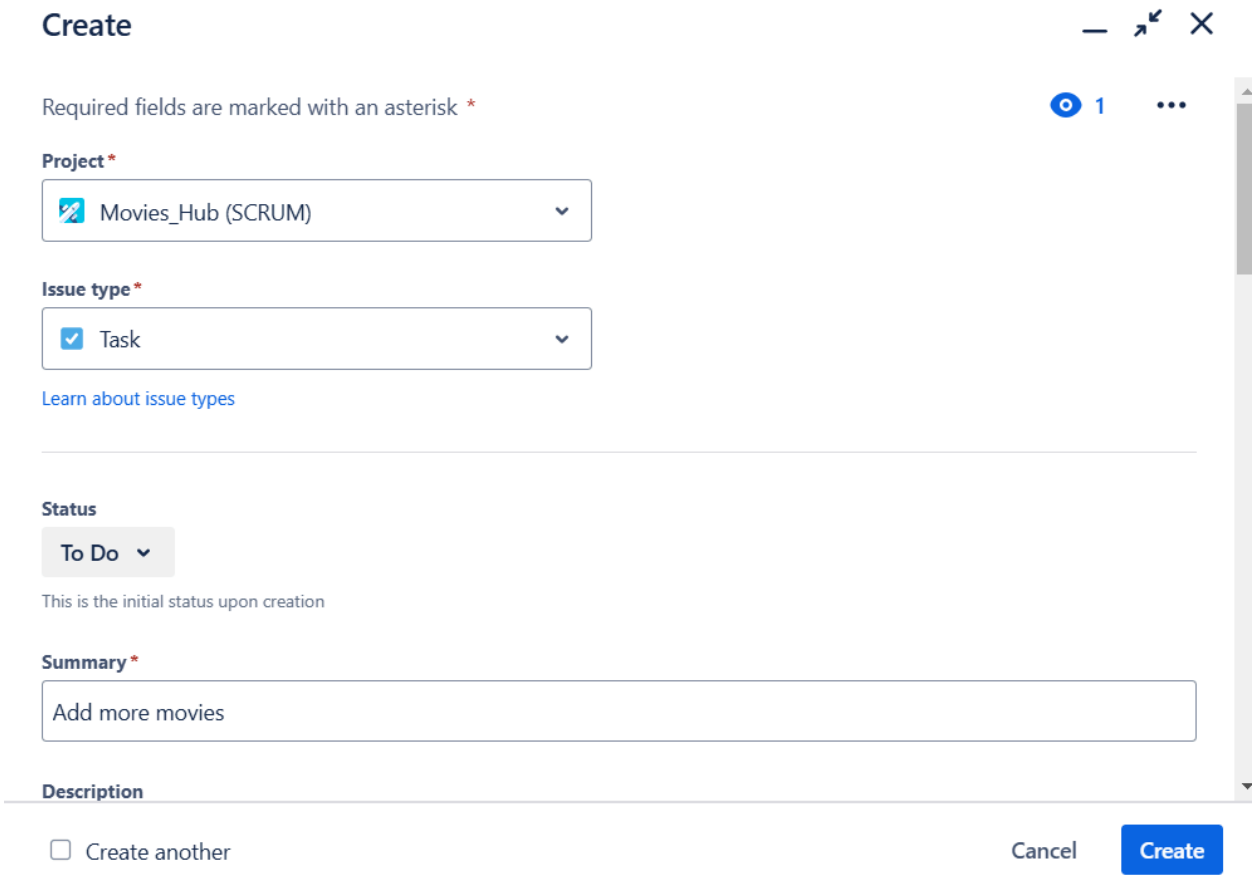


Create Issues:

- Use the **Create** option to add issues. Start by creating one general issue and three **Bugs**.
- For each bug, set its status (e.g., "To Do," "In Progress," "Done") using the dropdown.

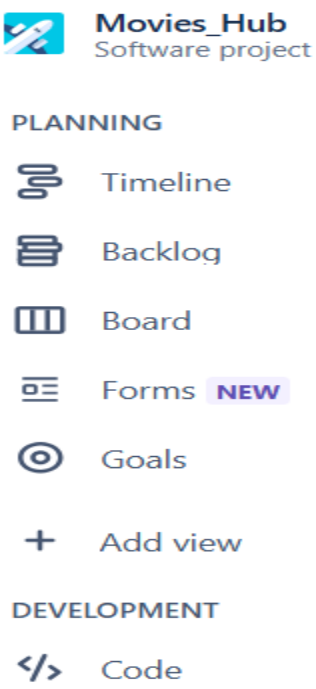
Assign Tasks to Team Members:

- Assign tasks to team members you've invited to the project.

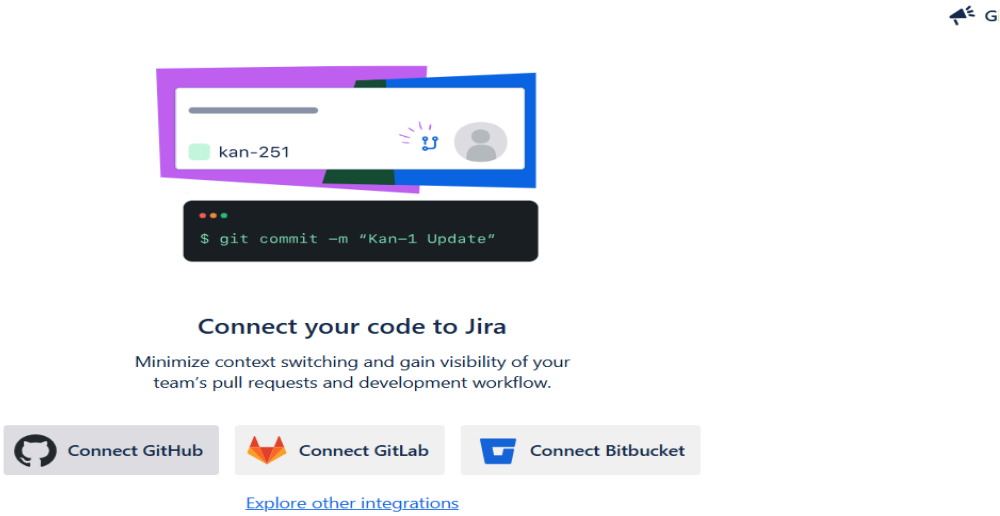


Connect Jira to GitHub:

- Go to the **Code** option on the left side panel and select **Connect GitHub**.
- Follow the prompts to authenticate your GitHub account and connect it to Jira for seamless code updates.

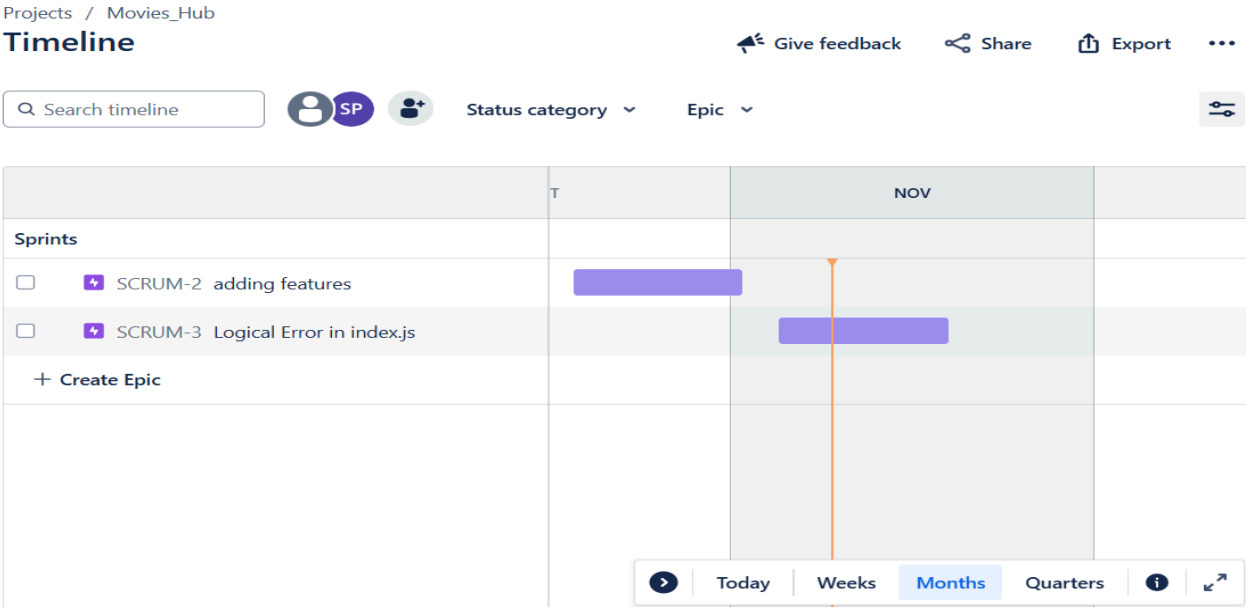


Authenticate with github



Create a Timeline:

- Navigate to the **Timeline** view.
- Manually add 3-4 tasks and assign a duration to each by selecting the timeline bar and setting the start and end dates.



Assignment 32

32. Demonstrate go Applications: (any one)
Some notable open source applications written in Go include:
• Caddy, an open source HTTP/2 web server with automatic HTTPS capability.
• CockroachDB, an open source, survivable, strongly consistent, scale-out SQL database.
• Docker, a set of tools for deploying Linux containers
• Ethereum, The go-ethereum implementation of the Ethereum Virtual Machine blockchain for the Ether cryptocurrency
• Hugo, a static site generator
• InfluxDB, an open source database specifically to handle time series data with high availability and high performance requirements.
• InterPlanetary File System, a content-addressable, peer-to-peer hypermedia protocol.
• Juju, a service orchestration tool by Canonical, packagers of Ubuntu Linux
• Kubernetes container management system
• Lightning Network, a bitcoin network that allows for fast Bitcoin transactions and scalability.
• Mattermost, a teamchat system
• OpenShift, a cloud computing platform as a service by Red Hat
• Snappy, a package manager for Ubuntu Touch developed by Canonical.
• Syncthing, an open-source file synchronization client/server application

• Terraform, an open-source, multiple cloud infrastructure provisioning tool from HashiCorp.

Note: I will underline exact command to be put in the terminal, just copy and paste

There are many specified tools written in Go language, but what we are most familiar with is docker.

1. Docker
    a. First We install docker
    b. We create a simple project to demonstrate docker

a. Installation of docker

Uninstall old versions

Before you can install Docker Engine, you need to uninstall any conflicting packages.

Distro maintainers provide unofficial distributions of Docker packages in APT. You must uninstall these packages before you can install the official version of Docker Engine.

The unofficial packages to uninstall are:

- `docker.io`
- `docker-compose`
- `docker-compose-v2`
- `docker-doc`
- `podman-docker`

Run the following command in terminal:

for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done

---

Install using the `apt` repository

Set up Docker's `apt` repository.

Add Docker's official GPG key:

sudo apt-get update

sudo apt-get install ca-certificates curl

sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

Add the repository to Apt sources:

echo \

  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \

  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \

  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

Then Install the Docker packages. To install the latest version, run:

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

Verify that the Docker Engine installation is successful by running the `hello-world` image.

sudo docker run hello-world

Note: from now onwards if you want to use docker command, use it with sudo

---

b. Create a simple project.
   We will create a simple index.html file, we will install apache or ngnix webserver docker image and run our project in that docker image container.

Create a directory for project, and go to that directory.

Create a index.html file and write below contents and save it

---

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Hello World</title>
</head>
<body>
   <h1>Hello, World!</h1>
</body>
</html>
```

Or only below is enough

```
<h1>Hello, World!</h1>
```

Now create a file named Dockerfile and paste the below contents

---

```
# Use the official Nginx image from Docker Hub
FROM nginx:alpine

# Copy the index.html to the default Nginx HTML location
COPY index.html /usr/share/nginx/html/index.html

# Expose port 80 to the outside world
EXPOSE 80

# Start Nginx server
CMD ["nginx", "-g", "daemon off;"]
```

—----------

Now we have to run commands
To build the docker image

sudo docker build -t hello-world-app .

To run the docker container

sudo docker run -p 80:80 hello-world-app

Now go to your browser and type localhost:8000
Your docker container should be running and hello world should be displayed.

Assignment 33

Download the suitable version of kernel/linux code from distro/linux.org and add print statement and compile it and demonstrate the change kernel/print statement added.

1. Download the Kernel Source Code

First, go to https://www.kernel.org/ to download the latest stable version of the Linux kernel, or you can use Git to clone it directly.

Clone the kernel repository-
git clone
https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
cd linux-stable

2. Checkout a Stable Version
List tags to see available versions - **git tag**

Checkout a specific version (for example, 5.10)-
git checkout v5.10

3. Add Print Statement
Open init/main.c in a text editor-
nano init/main.c

Add code in above file-

```
#include <linux/kernel.h>

void my_custom_function(void) {
        printk(KERN_INFO "Hello from your modified kernel!\n");
}
```

4. Configure the Kernel
Load default configuration for system -
make defconfig

Optionally, customize the kernel -
make menuconfig

5. Compile the Kernel

Compile the kernel -
make -j$(nproc)

Compile modules, if any -
make modules

Install the modules -
sudo make modules_install

6. Install the Kernel

Install the kernel-
sudo make install

Update GRUB bootloader -
sudo update-grub

7. Reboot and Verify

Reboot your system and select new kernel from the boot menu. To verify that the change worked, check the boot logs:

Reboot the system -

sudo reboot

After reboot, check the boot log for print statement

dmesg | grep "Custom Kernel"

Output message is **“Custom Kernel: Hello from your modified kernel!”**

Assignment 35

Write a Docker File to pull the Ubuntu with open jdk and write any java application.

Step 1: Create a Project Directory

Create a directory for your project. This will hold your Dockerfile and Java application files.

mkdir java-docker-project
cd java-docker-project

Step 2: Create a Simple Java Application

Inside the `java-docker-project` directory, create a simple Java program. Create a file called `HelloWorld.java`:

```
// HelloWorld.java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, Docker World!");
    }
}
```

Step 3: Write the Dockerfile

In the same directory, create a Dockerfile to build an image with Ubuntu, OpenJDK, and your Java application.

 # Dockerfile

 # Step 1: Use an official Ubuntu image as the base FROM ubuntu:latest

 # Step 2: Install OpenJDK RUN apt-get update && \ apt-get install -y openjdk-11-jdk && \ apt-get clean;

 # Step 3: Set the working directory WORKDIR /app

 # Step 4: Copy the Java application file into the container COPY HelloWorld.java .

 # Step 5: Compile the Java application RUN javac HelloWorld.java
 # Step 6: Set the command to run the Java application CMD ["java", "HelloWorld"]

Step 4: Build the Docker Image

In your terminal, navigate to the `java-docker-project` directory (if not already there) and run the following command to build the Docker image:

```
docker build -t java-docker-app .
```

This command will create a Docker image named `java-docker-app` by using the instructions in the Dockerfile.

Step 5: Run the Docker Container

Run a container from the image to see the output of the Java application:

docker run java-docker-app

Steps In Short:

Create a Project Directory: Set up a folder to hold your files.

Write a Java Application: Create a simple `HelloWorld.java` file.

Create a Dockerfile: Write Docker instructions to pull Ubuntu, install OpenJDK, and run the Java application.

Build the Image: Use `docker build` to create the Docker image.

Run the Container: Use `docker run` to start the container and see the output.

38.Create two applications/socket/IPC in two different docker containers. Push those applications and run to show the communications between two dockers.
Ex Message passing between two container
Setting up docker if not installed

(1. Install Docker on Linux:

If Docker is not installed on your machine, follow these steps:

- First, update the package database:

Command:

sudo apt-get update

- **Then, install Docker:**

Command:

sudo apt-get install docker.io

- After installation, **start the Docker service:**

Command:

sudo systemctl start docker

sudo systemctl enable docker

- **Verify that Docker is installed and running:**

Command:

docker --version

Step2: Check if docker-compose present or not: **docker-compose --version**

If not present follow the below steps to install it:

sudo curl -L "https://github.com/docker/compose/releases/download/v2.30.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

Then configure it:

sudo chmod +x /usr/local/bin/docker-compose

Verify if properly installed **: docker-compose --version)**

Step 1: Set Up Your Project Directory

1.Create a project folder on your computer to keep all files organized.

```
mkdir docker-socket-communication
cd docker-socket-communication
```

2.**Inside this folder**, create two subdirectories for the server and client applications.

```
mkdir server client
```

Step 2: Write the Server and Client Applications

You will write Python scripts that will communicate via sockets.

1. Create the Server Application

Inside the `server` folder, create a file named `server.py`:
cd server
`gedit server.py`
Edit the file and paste the following code:

```
import socket

SERVER_HOST = "0.0.0.0"  # Accept connections on any network interface
SERVER_PORT = 5000

# Create and set up server socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((SERVER_HOST, SERVER_PORT))
server_socket.listen(1)
print(f"Server listening on {SERVER_HOST}:{SERVER_PORT}")

# Accept client connection
client_socket, client_address = server_socket.accept()
print(f"Connection from {client_address}")

# Receive message from client
message = client_socket.recv(1024).decode()
print(f"Received from client: {message}")

# Send a response to the client
client_socket.send("Hello from server!".encode())

# Close connections
client_socket.close()
server_socket.close()
```

2. Create the Client Application

Inside the `client` folder, create a file named `client.py`:
cd client
`gedit client.py`
Edit the file and paste the following code:

```
import socket
import time
```

```
SERVER_HOST = "server"  # Using the server container name to resolve
the address
SERVER_PORT = 5000

# Wait a few seconds to ensure the server is ready
time.sleep(2)

# Create client socket and connect to server
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((SERVER_HOST, SERVER_PORT))

# Send message to server
client_socket.send("Hello from client!".encode())

# Receive and print response from server
message = client_socket.recv(1024).decode()
print(f"Received from server: {message}")

# Close the socket
client_socket.close()
```

Step 3: Create Dockerfiles for Each Application

Each application will have its own Dockerfile to define its environment.

1. Dockerfile for the Server Application

Inside the `server` folder, create a Dockerfile named `Dockerfile.server`:

```
gedit Dockerfile.server
```

Edit the file and paste the following instructions:

```
# Use a minimal Python image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy the server script into the image
```

```
COPY server.py /app

# Run the server application
CMD ["python", "server.py"]
```

2. Dockerfile for the Client Application

Inside the `client` folder, create a Dockerfile named `Dockerfile.client`:
`gedit Dockerfile.client`
Edit the file and paste the following instructions:

```
# Use a minimal Python image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy the client script into the image
COPY client.py /app

# Run the client application
CMD ["python", "client.py"]
```

Step 4: Create a Docker Network

To allow the two containers to communicate, you need to create a Docker network. Open a terminal in your project folder and run

Come to folder docker-socket-communication

```
sudo docker network create app-network
```

This creates a network named `app-network` that we'll use to connect the server and client containers.

Step 5: Build Docker Images

Next, build Docker images for both the server and client applications.

Build the server image:

```
sudo docker build -t server-app -f server/Dockerfile.server server
```

Build the client image:

```
sudo docker build -t client-app -f client/Dockerfile.client client
```

This will create two images, `server-app` and `client-app`, each containing one of the applications.

Step 6: Run the Docker Containers

Now, run each container on the same network.

Run the Server Container:

sudo `docker run -d --name server --network app-network server-app`

1. This command runs the server application in the background, connecting it to the `app-network` network and naming the container `server`.

Run the Client Container:

sudo `docker run --name client --network app-network client-app`

2. The client container will connect to the server container by resolving the `server` hostname on the `app-network` network.

Step 7: Verify Communication

After both containers are running, you can check their logs to verify that they have communicated successfully.

Check the server logs:

sudo `docker logs server`

You should see output similar to:

```
Server listening on 0.0.0.0:5000
Connection from ('client-IP-address', random-port)
Received from client: Hello from client!
```

Check the client logs:
sudo `docker logs client`
You should see output similar to:
`Received from server: Hello from server!`

Common commands you may need:

sudo docker ps -a

sudo docker ps

sudo docker rm <container_name_or_id>

sudo docker network rm <network_name>

sudo rm -rf <folder_name>

Assignment 37:.Create two applications/socket/IPC in two different containers. Push and run them to show communications between two dockers . Ex. Shared memory between two container

Step 1: Setting Up Docker Environment

1. Install Docker on Linux:
If Docker is not installed on your machine, follow these steps:
- First, update the package database:

Command:
sudo apt-get update

- **Then, install Docker:**
Command:

sudo apt-get install docker.io

- After installation, **start the Docker service:**
Command:
sudo systemctl start docker
sudo systemctl enable docker

- **Verify that Docker is installed and running:**
Command:
docker --version

Step2: Check if docker-compose present or not: **docker-compose --version**

If not present follow the below steps to install it:
sudo curl -L "https://github.com/docker/compose/releases/download/v2.30.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

Then configure it:
sudo chmod +x /usr/local/bin/docker-compose

Verify if properly installed **: docker-compose --version**

Stopping the Containers if required at execution

To stop the containers, press `Ctrl+C` or run:

```
docker-compose down
```

- To clean up unused Docker images and containers, you can run:

```
docker system prune -a
```

Step3:start with the below commands

```
mkdir dockerIpcsocket
cd dockerIpcsocket
mkdir shared_memory
cd shared_memory
mkdir app1
mkdir app2
cd app1
gedit app.py
gedit Dockerfile
cd ..
cd app2
gedit app.py
gedit Dockerfile
cd ..
gedit docker-compose.yml
```

Folder Structure:

```
  dockerIpcsocket
   --- shared_memory
       --app1
          --app.py
          --Dockerfile
       --app2
          --app.py
          --Dockerfile
       –docker-compose.yml
```

Commands to be executed at end inside dockerIpcsocket directory only:

1. sudo docker-compose build
2.sudo docker-compose up -d
3.sudo docker logs app1
4.sudo docker logs app2

Note:screenshot of final output is attached at the end

#app1./app.py

```
import mmap
import os
# Initialize shared memory
print("app1: Initializing shared memory...")
shm_file = "/dev/shm/shared_mem"
```

```
if not os.path.exists(shm_file):
with open(shm_file, "wb") as f:
f.write(b'\x00' * 1024) # Allocate space in shared memory
print("app1: Shared memory initialized.")
# Write to shared memory
print("app1: Opening shared memory for writing...")
with open(shm_file, "r+b") as f:
shm = mmap.mmap(f.fileno(), 1024)
data = b"Hello from app1!"
shm.write(data) # Write data to shared memory
print(f"app1: Data written to shared memory: {data.decode()}")
```

app2/app.py

```
import mmap
import os
import time
# Wait until the shared memory file exists
while not os.path.exists("/dev/shm/shared_mem"):
print("app2: Waiting for shared memory file...")
time.sleep(1)
# Open shared memory and read the data
with open("/dev/shm/shared_mem", "r+b") as f:
shm = mmap.mmap(f.fileno(), 1024)
data = shm.read(1024).strip(b'\x00')
if data:
print("app2: Read from shared memory:", data.decode())
else:
print("app2: No data read from shared memory.")
shm.close()
```

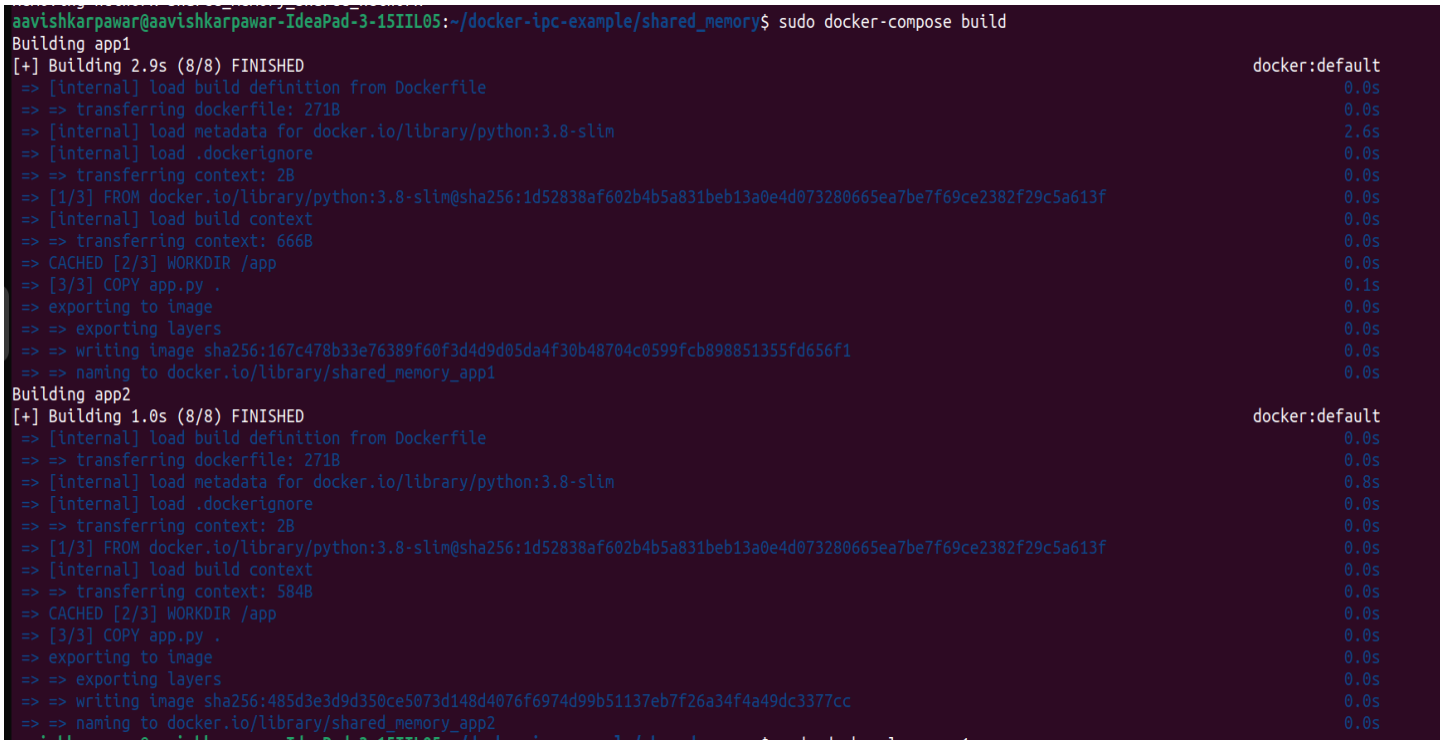
Dockerfile (same for both app1 and app2 but put it in both folders):

```
# Use the Python 3.8 slim image
FROM python:3.8-slim
# Set the working directory in the container
WORKDIR /app
# Copy the Python app code into the container
COPY app.py .
# Command to run the Python app
CMD ["python", "app.py"]
```

docker-compose.yml:

```
version: '3'
services:
app1:
build:
context: ./app1
```

container_name: app1
volumes:
- ./app1:/app
- shared-memory:/dev/shm # Mount shared memory volume
networks:
- shared_network
app2:
build:
context: ./app2
container_name: app2
volumes:
- ./app2:/app
- shared-memory:/dev/shm # Mount shared memory volume
networks:
- shared_network
networks:
shared_network:
driver: bridge
volumes:
shared-memory: # Define a named volume for shared memory
driver: local



```
aavishkarpawar@aavishkarpawar-IdeaPad-3-15IIL05:~/docker-ipc-example/shared_memory$ sudo docker-compose up -d
Creating network "shared_memory_shared_network" with driver "bridge"
Creating app1 ... done
Creating app2 ... done
aavishkarpawar@aavishkarpawar-IdeaPad-3-15IIL05:~/docker-ipc-example/shared_memory$ sudo docker logs app1
app1: Initializing shared memory...
app1: Shared memory initialized.
app1: Opening shared memory for writing...
app1: Data written to shared memory: Hello from app1!
aavishkarpawar@aavishkarpawar-IdeaPad-3-15IIL05:~/docker-ipc-example/shared_memory$ sudo docker logs app2
app2: Read from shared memory: Hello from app1!
```

39.Create two applications/socket/IPC in two different Docker containers. Push those applications and run to show the communications between two dockers. Ex. TCP/UDP Socket communication between two container

→

IF U NEED THE FILES DIRECTLY THEN CLONE THE GITHUB REPO GIVEN BELOW:
https://github.com/Akashbhilwande/tcp-udp-comm-using-docker.git

Step 1: Setting Up the Project Structure

Let's organize the project directory as follows:

```
docker-socket-communication/
├── server/
│   ├── Dockerfile
│   ├── server.py
├── client/
│   ├── Dockerfile
│   ├── client.py
├── docker-compose.yml
```

Step 2: Writing the Server Code (`server/server.py`)

We'll create a simple TCP server that listens for connections from the client.

server/server.py :

```
# server/server.py
import socket

HOST = '0.0.0.0'  # Listen on all network interfaces
```

```
PORT = 12345       # Port to listen on

def start_server():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        print(f"Server listening on {HOST}:{PORT}...")

        conn, addr = s.accept()
        with conn:
            print(f"Connected by {addr}")
            while True:
                data = conn.recv(1024)
                if not data:
                    break
                print(f"Received from client: {data.decode()}")
                conn.sendall(b"Hello from Server!")

if __name__ == "__main__":
    start_server()
```

Step 3: Writing the Client Code (`client/client.py`)

We'll create a simple TCP client that connects to the server and sends a message.client/client.py :

```
# client/client.py
import socket

SERVER_HOST = 'server'  # Hostname of the server as defined in
docker-compose.yml
SERVER_PORT = 12345     # Port to connect to

def start_client():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((SERVER_HOST, SERVER_PORT))
        print(f"Connected to server at {SERVER_HOST}:{SERVER_PORT}")
        s.sendall(b"Hello from Client!")
        data = s.recv(1024)
        print(f"Received from server: {data.decode()}")
```

```
if __name__ == "__main__":
    start_client()
```

Step 4: Creating Dockerfiles for Both Applications

Server Dockerfile (`server/Dockerfile`)

```
# server/Dockerfile
FROM python:3.9-slim
WORKDIR /app
COPY server.py .
CMD ["python", "server.py"]
```

Client Dockerfile (`client/Dockerfile`)

```
# client/Dockerfile
FROM python:3.9-slim
WORKDIR /app
COPY client.py .
CMD ["python", "client.py"]
```

Step 5: Creating `docker-compose.yml`

We'll use Docker Compose to set up the network and link the two containers.

```
# docker-compose.yml
version: '3.8'

services:
 server:
   build: ./server
   container_name: tcp_server
   networks:
     - socket-network
   ports:
     - "12345:12345"

 client:
   build: ./client
   container_name: tcp_client
```

```
    networks:
      - socket-network
    depends_on:
      - server

networks:
  socket-network:
    driver: bridge
```

Step 6: Building and Running the Containers

Open a terminal in the `docker-socket-communication` directory and run the following commands:

1. Build the Docker images:

```
docker-compose build
```

2. Run the containers:

```
docker-compose up
```

Step 7: Verifying Communication

You should see output similar to the following:

```
Creating network "docker-socket-communication_socket-network" with driver "bridge"
Creating tcp_server ... done
Creating tcp_client ... done
Attaching to tcp_server, tcp_client
tcp_server | Server listening on 0.0.0.0:12345...
tcp_server | Connected by ('172.18.0.3', 33734)
tcp_server | Received from client: Hello from Client!
tcp_client | Connected to server at server:12345
tcp_client | Received from server: Hello from Server!
tcp_client exited with code 0
tcp_server exited with code 0
```

Explanation

- Server listens on port `12345` for incoming connections.
- Client connects to the server and sends a message.

- Docker Compose sets up a bridge network called `socket-network` to allow the two containers toCommunicate.

Step 8: Stopping the Containers

To stop the containers, press `Ctrl+C` or run:

```
docker-compose down
```

- To clean up unused Docker images and containers, you can run:

```
docker system prune -a
```

Assignment 41

Title: Create a web application with simple web page containing login details and create a docker image of the application.(Use Apache Web server)
Run the Docker container from recently created image and run the container at port number 80 in host system. Push that image to repository. Make use of database.
Try to access it from other instance of docker.

In this assignment, I will build a simple web application that features a login page. This web application will be containerized using Docker, which will use Apache as the web server and connect to a MySQL database. The goal is to build and test the application in a containerized environment and push the Docker image to Docker Hub for remote access.

How to Achieve This

1. **Create the Web Application**:
    - HTML page for login.
    - PHP code to process the login and connect to MySQL.
2. **Set up Docker**:
    - Use Docker to create a container with Apache, PHP, and MySQL.
    - Configure a `Dockerfile` to install Apache and PHP.
    - Use `docker-compose` to run the app and MySQL in separate containers.
3. **Push Docker Image to Docker Hub**:
    - Log in to Docker Hub.
    - Tag the image and push it to the Docker Hub repository.
4. **Test Access from Another Container**:
    - Use Docker's networking features to connect the app container and a test container.

Step-by-Step Process and Commands

1.Set Up the Web Application

Create a Project Directory:

```
mkdir simple-web-app
cd simple-web-app
mkdir public
mkdir db
touch public/index.html
touch public/login.php
touch Dockerfile
touch docker-compose.yml
touch db/init.sql
```

Write the HTML for Login (index.html): In `public/index.html`

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <form action="login.php" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required><br>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password"
required><br>
        <button type="submit">Login</button>
    </form>
</body>
</html>
```

Write the PHP Code for Login (login.php): In public/login.php

```
<?php
$servername = "mysql";
$username = "root";
$password = "rootpassword";
$dbname = "userdb";
```

```
$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $user = $_POST['username'];
    $pass = $_POST['password'];

    $sql = "SELECT * FROM users WHERE username='$user' AND
password='$pass'";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        echo "Login successful!";
    } else {
        echo "Invalid credentials.";
    }
}

$conn->close();
?>
```

MySQL Database Setup (init.sql): In db/init.sql

```
CREATE DATABASE IF NOT EXISTS userdb;
USE userdb;

CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    password VARCHAR(50) NOT NULL
);

INSERT INTO users (username, password) VALUES ('testuser',
'password123');
```

Set Up Docker with Apache and MySQL

1. **Write the Dockerfile**: In Dockerfile

```
# Use the official PHP Apache image

FROM php:7.4-apache


# Install mysqli extension and any dependencies

RUN apt-get update && \

    apt-get install -y libpng-dev libjpeg-dev libfreetype6-dev &&
\

    docker-php-ext-install mysqli


# Copy application files to the container

COPY public/ /var/www/html/


# Expose port 80

EXPOSE 80
```

2. Write docker-compose.yml: In docker-compose.yml

```
version: '3'


services:

  web:

    build: .

    ports:
```

```
      - "80:80"
    depends_on:
      - mysql
    networks:
      - my-network

  mysql:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: userdb
    volumes:
      - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
    networks:
      - my-network

networks:
  my-network:
    driver: bridge
```

Build and Run Docker Containers

1. `sudo apt install docker-compose`

Build the Docker Image

2.sudo docker-compose build

Start the Containersdocl

3.sudo docker-compose up -d

Push Docker Image to Docker Hub

docker tag <image-id> your-docker-id/simple-web-app:latest

docker login

docker push your-docker-id/simple-web-app:latest

Test Access from Another Docker Instance

docker run --network=my-network --rm appropriate/curl http://web/login.php

Assignment 42

42.With the help of Docker-compose deploy the 'Wordpress' and 'Mysql' container and access the front end of 'Wordpress.

Step 1- Install Docker CE

$ sudo apt-get update

$ sudo apt-get install docker-ce

Step 2- Create a Directory for WordPress

$ mkdir wordpress-docker

$ cd wordpress-docker

Step 3- Create a Docker Compose File

In the **wordpress-docker** directory, create a **docker-compose.yml** file

```
version: '3.8'
```

```
services:
  wordpress:
    image: wordpress:latest
    ports:
    - "5000:80"      # Expose port 80 of the container
on port 5000 of the host
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb
    depends_on:
    - db

  db:
    image: mysql:5.7
    environment:
      MYSQL_DATABASE: exampledb
      MYSQL_USER: exampleuser
      MYSQL_PASSWORD: examplepass
```

```
      MYSQL_ROOT_PASSWORD: rootpass
    volumes:
    - db_data:/var/lib/mysql

volumes:
  db_data:
```

Step 4- Pulling hello-world image from docker-hub

$ sudo docker run hello-world

Step 5- Pulling wordpress image from docker-hub

$ sudo docker pull wordpress

Step 6- Pulling my-sql image

$ sudo docker pull mysql

Step 7- Deploy the Containers

Start the services with Docker Compose:

$ docker-compose up -d

Step 8- Accessing wordpress on local host after binding port 5000 onn host machine to container port (port no. 80/tcp)

$ sudo docker run -p 5000:80 wordpress

Step 8 – Paste below URL in website

http://localhost:5000

After this wordpress page is shown by localhost, it conclude that we have accessed the front end of wordpress.
After that select language and then put the username and password that was already put in **docker-compose.yml**

Assignment 43

43 A. Create a Simple Hello-World Python Flask Application

Setup the Flask App: Create a new folder, e.g., `flask_app`, and inside this folder, create a Python file named `app.py` with the following code:

Contents of app.py

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return "Hello, World from Flask App!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Create a `requirements.txt` file: List the dependencies required for the Flask app:

Contents of requirements.txt

```
Flask==2.3.2
```

Create the Docker Image for the Flask Application

Create a `Dockerfile`: In the same `flask_app` folder, create a file named `Dockerfile` with the following content:

Contents of Dockerfile

```
# Use the official Python image
FROM python:3.10-slim

# Set the working directory
WORKDIR /app

# Copy the current directory contents into the container
COPY . .

# Install the dependencies
RUN pip install -r requirements.txt

# Expose port 5000 for the Flask app
EXPOSE 5000

# Command to run the Flask app
CMD ["python", "app.py"]
```

Build the Docker Image: Open your terminal in the `flask_app` folder and run the following command to build the Docker image:

Commands

```
docker build -t flask-hello-world .
(Consider the “.” also for the command)
```

B. Run the Docker Container on Port 5000

Run the Docker Container: Use the following command to run the container:

Commands

```
docker run -d -p 5000:5000 flask-hello-world
```

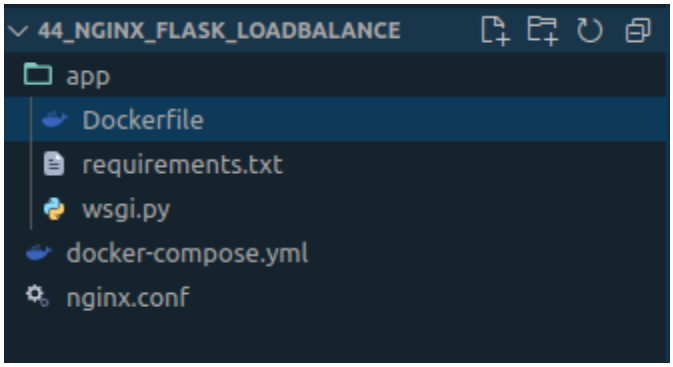
Access the Flask App: Open a browser and navigate to:

```
http://localhost:5000
```

You should see the message:

```
Hello, World from Flask App!
```

44. Create the ‘nginx’ container from ‘nginx’ image. And create the load balancing so that if we go to tha address of ‘nginx ‘ it can redirect it to the above created applications (Flask and Wordpress).



Dockerfile -

```
FROM python:3.9-slim

WORKDIR /app

COPY . .

RUN pip install -r requirements.txt

CMD gunicorn --bind 0.0.0.0:5000 wsgi:app
```

requirements.txt

```
flask
Gunicorn
```

wsgi.py

```
from flask import Flask
import socket

app = Flask(__name__)

# Route for the default page
@app.route("/")
def home():
    # Display message
    return f"<center><h3>Welcome to OSS. Running - {socket.gethostname()}
</h3></center>"

if __name__ == '__main__':
    app.run('0.0.0.0')
```

docker-compose.yml

```
version: '3'

services:
  app:
    build:
      context: app
    ports:
      # - "5000:5000"
      - "5000"

  nginx:
    image: nginx:latest
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    depends_on:
      - app
    ports:
      - "80:80"
```

Nginx.conf -

```
events {
    worker_connections 1000;
}

http {
    upstream app_servers {
        server app:5000;
    }

    server {
        listen 80;
```

```
        location / {
            proxy_pass http://app_servers;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

Commands -

```
docker-compose up -d --build --scale app=3
// This will create 3 instance of the applications
```

```
 Docker ps
// to see the running container

// to stop all instances
 docker-compose down
```

Open your browser and navigate to `http://localhost:80`, where the Nginx server is running. When you send a request to the Nginx server, it will forward the request to one of the running instances of your Flask application. Each time you refresh the page, the request will be directed to a different Flask instance, demonstrating that load balancing is working as expected.

Assignment 45 : Create a web application with simple web page containing login details and create a docker image of the application.(Use Apache Web server)

1. Assignment Specification:

The task is to create a simple web application containing a login page, hosted on an Apache Web Server. We will package the application into a Docker container, which can be run on any machine without the need for manual installation of dependencies.

2. How Will We Achieve It?

To achieve this, we will:

- ➢ Create HTML files for the login page and the post-login page.
- ➢ Style the pages using CSS to make them visually appealing.
- ➢ Implement JavaScript to handle basic form interactions.

- ➢ Configure Apache Web Server to serve the pages.
- ➢ Use Docker to containerize the Apache Web Server and the application files.
- ➢ Build and run the Docker container to make the application accessible via a web browser.

3. Step-by-Step Guide:

Step 1: Setting Up Docker Environment

1. Install Docker on Linux:

If Docker is not installed on your machine, follow these steps:

- First, update the package database:

Command:

```
sudo apt-get update
```

- **Then, install Docker:**

Command:

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

- After installation, **start the Docker service:**

Command:

```
sudo systemctl start docker
sudo systemctl enable docker
```

- **Verify that Docker is installed and running:**

Command:

```
docker --version
```

2. Create a Project Folder:

- Create a new directory for the project and navigate into it:

Command:

```
mkdir apache-login-app
cd apache-login-app
```

3. Create Dockerfile:

- Inside the project folder, create a file named `Dockerfile`:

Command:

```
touch Dockerfile
```

- Add the following content to the `Dockerfile`:

```
# Use Apache base image
FROM httpd:latest

# Copy the login page to the Apache web server's document root
COPY login.html /usr/local/apache2/htdocs/

COPY index.html /usr/local/apache2/htdocs/

COPY ./login.js /usr/local/apache2/htdocs/
COPY ./style.css /usr/local/apache2/htdocs/
# Expose the HTTP port
EXPOSE 80
# Set permissions for logs
RUN chmod -R 777 /usr/local/apache2/logs
```

Step 2: Create Application Files

1. HTML Files:

- **Create the following directory structure:**

Command:

```
touch index.html login.html style.css login.js apache.conf
```

Add these file along with Dockerfile in your folder

2. **Create `index.html`**:

Add the following code to `index.html`:

```
<!DOCTYPE html>
<html lang="en">
```

```
  <head>
   <meta charset="UTF-8">
  <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Welcome</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="welcome-container">
        <h1>Welcome to Your Dashboard</h1>
        <p>You are successfully logged in!</p>
        <button onclick="logout()">Logout</button>
    </div>

    <script>
        function logout() {
            window.location.href = "login.html"; // Redirect to login page
        }
    </script>
</body>
</html>
```

3. **Create `login.html`**:
   Add the following code to `login.html`:

```
 <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="login-container">
        <h2>Login</h2>
```

```
        <form id="login-form">
            <input type="text" id="username" placeholder="Username" required><br>
                    <input  type="password"  id="password"  placeholder="Password"
required><br>
            <button type="submit">Login</button>
        </form>
        <p id="error-message" class="error-message"></p>
                    <a   href="forgot-password.html"   class="forgot-password">Forgot
Password?</a>
    </div>

    <script src="login.js"></script>
</body>
</html>
```

4. **Create style.css**:
   Add the following code to `style.css` for styling the pages:/* General styling */

```
/* General styling */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: Arial, sans-serif;
    background: linear-gradient(135deg, #ff7e5f, #feb47b);
    height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
    color: #333;
}

a {
    text-decoration: none;
```

```
    color: #ff7e5f;
}

h2 {
    text-align: center;
    color: #fff;
}

button {
    padding: 10px 20px;
    border-radius: 5px;
    background-color: #ff7e5f;
    color: #fff;
    border: none;
    cursor: pointer;
    transition: all 0.3s ease;
}

button:hover {
    background-color: #feb47b;
}

.error-message {
    color: red;
    text-align: center;
    margin-top: 10px;
}

.login-container,
.welcome-container {
    background-color: white;
    border-radius: 10px;
    padding: 30px;
    box-shadow: 0 5px 20px rgba(0, 0, 0, 0.1);
    width: 300px;
}

.login-container input,
.welcome-container button {
    width: 100%;
```

```
    padding: 15px;
    margin: 10px 0;
    border-radius: 5px;
    border: 1px solid #ddd;
    font-size: 16px;
    outline: none;
}

input:focus {
    border-color: #ff7e5f;
}

/* Welcome container */
.welcome-container {
    text-align: center;
}

.welcome-container button {
    margin-top: 20px;
}
```

5. Create a login.js file

```
document.getElementById("login-form").addEventListener("submit",
function(event) {
    event.preventDefault(); // Prevent the default form submission behavior

    let username = document.getElementById("username").value;
    let password = document.getElementById("password").value;
    let errorMessage = document.getElementById("error-message");

    // Simple validation
    if(username === "" || password === "") {
        errorMessage.textContent = "Please fill in both fields.";
        return;
    }

     // Example check (In real applications, this should be replaced with backend
validation)
    if(username === "user" && password === "password123") {
        window.location.href = "index.html"; // Redirect to the index page
```

```
    } else {
        errorMessage.textContent = "Invalid credentials, please try again.";
    }
});
```

6. Create a Apache configuration file:
Add following in the apache.conf file:

```
ServerName localhost
# Load necessary modules
LoadModule mpm_event_module modules/mod_mpm_event.so
LoadModule dir_module modules/mod_dir.so
LoadModule alias_module modules/mod_alias.so
LoadModule authz_core_module modules/mod_authz_core.so
<VirtualHost *:80>
    DocumentRoot "/usr/local/apache2/htdocs"
    DirectoryIndex login.html
    <Directory "/usr/local/apache2/htdocs">
        AllowOverride None
        Require all granted
    </Directory>
</VirtualHost>
```

Step 3: Build the Docker Image

1. Build Docker Image:
- Navigate to your project directory where the `Dockerfile` is located and build the Docker image: **Command**:

```
docker build -t apache-login-app .
```

Don't miss the dot at the end of the command above

2. Run the Docker Container:
- Run the Docker container, mapping port `8080` to port `80` inside the container:
Command:

```
docker run -d -p 8080:80 apache-login-app
```

Step 4: Test the Application

1. Open the Application in Browser:

- Open your browser and navigate to:

  http://localhost:8080

2. **Login**:

- Enter a username and password (note: this form is static, so the data won't be validated in this simple example), and the page will redirect to `index.html`.

User and password id mentioned in the login.js page

User = user and password = password123

4. Common Errors and Troubleshooting

1. Apache Configuration Issues:

- If you get an error like `Invalid command 'DirectoryIndex'`, ensure that the necessary Apache modules are loaded:
  - **Solution: Edit `apache.conf`:**
    `

```
LoadModule dir_module modules/mod_dir.so
LoadModule mpm_event_module modules/mod_mpm_event.so
```

2. Docker Port Binding Issues:

- If the container doesn't run correctly:
  - Check if port `8080` is being used by another process:
    Command:

```
sudo lsof -i :8080
```

  - Stop any processes occupying the port:

Mention the process id given by command above in place of PID

Command:

```
sudo kill <PID>
```

5. Conclusion

In this assignment, we successfully created a simple login web application using HTML, CSS, and JavaScript. The application was hosted on Apache Web Server, and we

containerized it using Docker. We learned how to build a Docker image, run a Docker container, and test the application in a browser.

Links to Articles and References:
- [Docker Documentation](https://docs.docker.com/get-docker/)
- [Apache HTTP Server Configuration](https://httpd.apache.org/docs/)

Assignment 46

Run the Docker container from recently created image and run the container at port number 80 in host system

1. Assignment Specification:

The task is to create a simple web application containing a login page, hosted on an Apache Web Server. We will package the application into a Docker container, which can be run on any machine without the need for manual installation of dependencies, Running the application on port 80.

2. How Will We Achieve It?

To achieve this, we will:
- ➢ Create HTML files for the login page and the post-login page.
- ➢ Style the pages using CSS to make them visually appealing.
- ➢ Implement JavaScript to handle basic form interactions.
- ➢ Configure Apache Web Server to serve the pages.
- ➢ Use Docker to containerize the Apache Web Server and the application files.
- ➢ Build and run the Docker container to make the application accessible via a web browser.

3. Step-by-Step Guide:

Step 1: Setting Up Docker Environment

1. Install Docker on Linux:

If Docker is not installed on your machine, follow these steps:

- First, update the package database:

Command:

```
sudo apt-get update
```

- **Then, install Docker:**

Command:

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

- After installation, **start the Docker service:**

Command:

```
sudo systemctl start docker
sudo systemctl enable docker
```

- **Verify that Docker is installed and running:**

Command:

```
docker --version
```

2. Create a Project Folder:

- Create a new directory for the project and navigate into it:

Command:

```
mkdir apache-login-app
cd apache-login-app
```

3. Create Dockerfile:

- Inside the project folder, create a file named `Dockerfile`:

Command:

```
touch Dockerfile
```

- Add the following content to the `Dockerfile`:

```
 # Use Apache base image
FROM httpd:latest

# Copy the login page to the Apache web server's document root
COPY login.html /usr/local/apache2/htdocs/

COPY index.html /usr/local/apache2/htdocs/

COPY ./login.js /usr/local/apache2/htdocs/
COPY ./style.css /usr/local/apache2/htdocs/
# Expose the HTTP port
EXPOSE 80
# Set permissions for logs
RUN chmod -R 777 /usr/local/apache2/logs
```

Step 2: Create Application Files

1. HTML Files:
- **Create the following directory structure:**
Command:

```
touch index.html login.html style.css login.js apache.conf
```

Add these file along with Dockerfile in your folder

2. **Create `index.html`**:
Add the following code to `index.html`:

```
  <!DOCTYPE html>
  <html lang="en">
  <head>
    <meta charset="UTF-8">
  <!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Welcome</title>
   <link rel="stylesheet" href="style.css">
```

```
</head>
<body>
    <div class="welcome-container">
        <h1>Welcome to Your Dashboard</h1>
        <p>You are successfully logged in!</p>
        <button onclick="logout()">Logout</button>
    </div>

    <script>
        function logout() {
            window.location.href = "login.html"; // Redirect to login page
        }
    </script>
</body>
</html>
```

3. **Create `login.html`**:
   Add the following code to `login.html`:

```
 <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="login-container">
        <h2>Login</h2>
        <form id="login-form">
            <input type="text" id="username" placeholder="Username" required><br>
                    <input  type="password"  id="password"  placeholder="Password"
required><br>
            <button type="submit">Login</button>
        </form>
        <p id="error-message" class="error-message"></p>
                    <a   href="forgot-password.html"   class="forgot-password">Forgot
Password?</a>
```

```
    </div>

    <script src="login.js"></script>
</body>
</html>
```

4. **Create style.css**:
   Add the following code to `style.css` for styling the pages:/* General styling */

```
/* General styling */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: Arial, sans-serif;
    background: linear-gradient(135deg, #ff7e5f, #feb47b);
    height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
    color: #333;
}

a {
    text-decoration: none;
    color: #ff7e5f;
}

h2 {
    text-align: center;
    color: #fff;
}

button {
```

```
    padding: 10px 20px;
    border-radius: 5px;
    background-color: #ff7e5f;
    color: #fff;
    border: none;
    cursor: pointer;
    transition: all 0.3s ease;
}

button:hover {
    background-color: #feb47b;
}

.error-message {
    color: red;
    text-align: center;
    margin-top: 10px;
}

.login-container,
.welcome-container {
    background-color: white;
    border-radius: 10px;
    padding: 30px;
    box-shadow: 0 5px 20px rgba(0, 0, 0, 0.1);
    width: 300px;
}

.login-container input,
.welcome-container button {
    width: 100%;
    padding: 15px;
    margin: 10px 0;
    border-radius: 5px;
    border: 1px solid #ddd;
    font-size: 16px;
    outline: none;
}

input:focus {
```

```
    border-color: #ff7e5f;
}

/* Welcome container */
.welcome-container {
    text-align: center;
}

.welcome-container button {
    margin-top: 20px;
}
```

5. Create a login.js file

```
document.getElementById("login-form").addEventListener("submit",
function(event) {
    event.preventDefault(); // Prevent the default form submission behavior

    let username = document.getElementById("username").value;
    let password = document.getElementById("password").value;
    let errorMessage = document.getElementById("error-message");

    // Simple validation
    if(username === "" || password === "") {
        errorMessage.textContent = "Please fill in both fields.";
        return;
    }

     // Example check (In real applications, this should be replaced with backend
validation)
    if(username === "user" && password === "password123") {
        window.location.href = "index.html"; // Redirect to the index page
    } else {
        errorMessage.textContent = "Invalid credentials, please try again.";
    }
});
```

6. Create a Apache configuration file:

Add following in the apache.conf file:

```
ServerName localhost
# Load necessary modules
LoadModule mpm_event_module modules/mod_mpm_event.so
LoadModule dir_module modules/mod_dir.so
LoadModule alias_module modules/mod_alias.so
LoadModule authz_core_module modules/mod_authz_core.so
<VirtualHost *:80>
    DocumentRoot "/usr/local/apache2/htdocs"
    DirectoryIndex login.html
    <Directory "/usr/local/apache2/htdocs">
        AllowOverride None
        Require all granted
    </Directory>
</VirtualHost>
```

Step 3: Build the Docker Image

1. Build Docker Image:

- Navigate to your project directory where the `Dockerfile` is located and build the Docker image: **Command**:

```
docker build -t apache-login-app .
```

Don't miss the dot at the end of the command above

2. Run the Docker Container:

- Run the Docker container, mapping port `8080` to port `80` inside the container:

Command:

```
docker run -d -p 8080:80 apache-login-app
```

Step 4: Test the Application

1. Open the Application in Browser:

- Open your browser and navigate to:

```
http://localhost:8080
```

2. **Login**:

- Enter a username and password (note: this form is static, so the data won't be validated in this simple example), and the page will redirect to `index.html`.

User and password id mentioned in the login.js page

User = user and password = password123

4. Common Errors and Troubleshooting

1. Apache Configuration Issues:

- If you get an error like `Invalid command 'DirectoryIndex'`, ensure that the necessary Apache modules are loaded:
  - **Solution: Edit `apache.conf`:**
    `

```
LoadModule dir_module modules/mod_dir.so
LoadModule mpm_event_module modules/mod_mpm_event.so
```

2. Docker Port Binding Issues:

- If the container doesn't run correctly:
  - Check if port `8080` is being used by another process:
    Command:

```
sudo lsof -i :8080
```

  - Stop any processes occupying the port:

Mention the process id given by command above in place of PID

Command:

```
sudo kill <PID>
```

5. Conclusion

In this assignment, we successfully created a simple login web application using HTML, CSS, and JavaScript. The application was hosted on Apache Web Server, and we containerized it using Docker. We learned how to build a Docker image, run a Docker container, and test the application in a browser.

Assignment - 47

Write a python program to perform arithmetic operations and create Docker image accordingly.

// make a folder
mkdir my_folder
cd my_folder

// make two files inside this folder - Dockerfile & calculator code file

Nano Dockerfile

```
# Use the official Python image as a base
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the Python script into the container
COPY arithmetic_operations.py .

# Command to run the Python script
CMD ["python", "arithmetic_operations.py"]
```

Nano arithmetic_operations.py

```
# arithmetic_operations.py

def add(x, y):
   return x + y

def subtract(x, y):
```

```
    return x - y

def multiply(x, y):
    return x * y

def divide(x, y):
    if y == 0:
        return "Error! Division by zero."
    return x / y

def main():
    print("Select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")

    choice = input("Enter choice (1/2/3/4): ")

    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))

    if choice == '1':
        print(f"{num1} + {num2} = {add(num1, num2)}")
    elif choice == '2':
        print(f"{num1} - {num2} = {subtract(num1, num2)}")
    elif choice == '3':
        print(f"{num1} * {num2} = {multiply(num1, num2)}")
    elif choice == '4':
        print(f"{num1} / {num2} = {divide(num1, num2)}")
    else:
        print("Invalid input")

if __name__ == "__main__":
    main()
```

Run this command and you are good to go .

Docker-compose up -d

Assignment - 48

Create a simple web application using LAMP Stack on docker container.

{In This i also added mysql database if not needed then simply do till step 4 }

Also give this to chat gpt for small adjustments

NOTE install `mattrayner/lamp:latest-1804` this version only

Step 1: Pull and Run the LAMP Docker Container

```
sudo docker run -d -p 80:80 -p 3306:3306 --name my-lamp-container mattrayner/lamp:latest-1804
```

Step 2: Access the Docker Container

```
sudo docker exec -it my-lamp-container /bin/bash
```

Navigate to the web root directory:

```
cd /var/www/html/
```

Step 3: Install nano (Optional Editor)

```
apt install nano
```

Step 4: Create `index.php`

```
cat <<EOL > index.php
```

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <title>Notes App</title>
</head>
<body>
    <h1>Notes</h1>
    <form action="db.php" method="POST">
        <label for="name">Name:</label>
        <input type="text" name="name" required><br>
        <label for="message">Message:</label>
        <textarea name="message" required></textarea><br>
        <input type="submit" value="Submit">
    </form>
    <h2>Entries</h2>
    <?php
    \$conn = new mysqli('localhost', 'root', '', 'notes');
    if (\$conn->connect_error) {
        die("Connection failed: " . \$conn->connect_error);
    }
    \$result = \$conn->query("SELECT * FROM entries ORDER BY
created_at DESC");
    while (\$row = \$result->fetch_assoc()) {
```

```
        echo "<p><strong>" . htmlspecialchars(\$row['name']) .
"</strong> (" . \$row['created_at'] . "):<br>" .
htmlspecialchars(\$row['message']) . "</p>";

    }

    \$conn->close();

    ?>

</body>

</html>

EOL
```

Step 5: Create db.php

```
cat <<EOL > db.php

<?php

if (\$_SERVER['REQUEST_METHOD'] === 'POST') {

    \$conn = new mysqli('localhost', 'root', '', 'notes');

    if (\$conn->connect_error) {

        die("Connection failed: " . \$conn->connect_error);

    }

    \$name = \$conn->real_escape_string(\$_POST['name']);

    \$message = \$conn->real_escape_string(\$_POST['message']);

    \$sql = "INSERT INTO entries (name, message) VALUES ('\$name',
'\$message')";

    if (\$conn->query(\$sql) === TRUE) {

        echo "New record created successfully";
```

```
    } else {
        echo "Error: " . \$sql . "<br>" . \$conn->error;
    }
    \$conn->close();
    header("Location: index.php");
    exit();
}
?>
EOL
```

Step 6: Set Up the MySQL Database

Access MySQL:

```
mysql -u root -p
```

Create the database and table for storing notes:

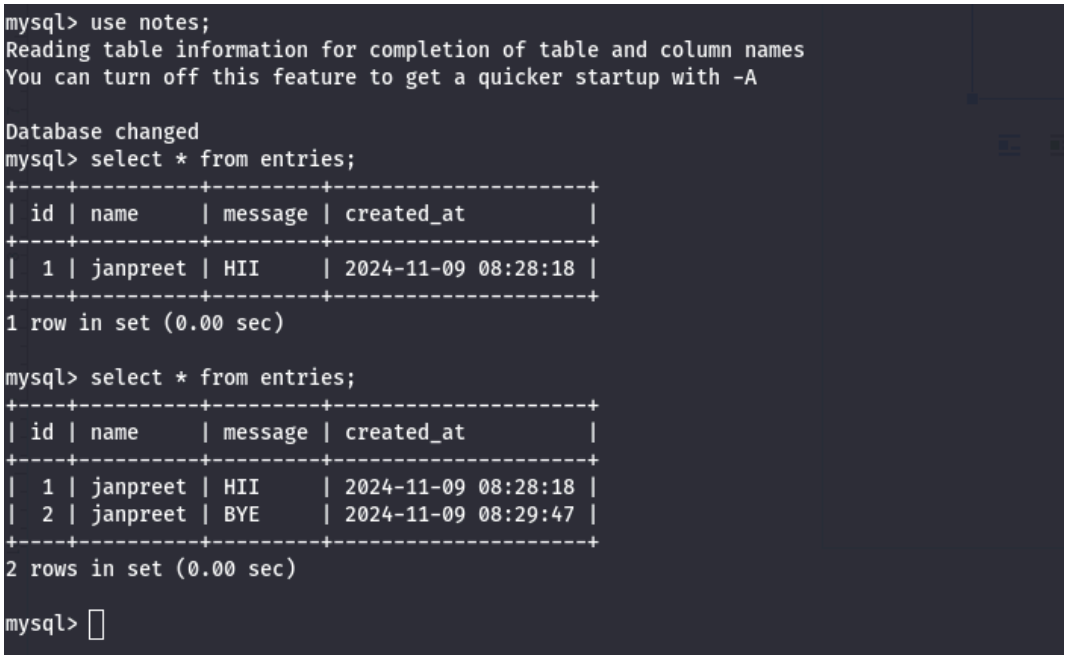
```
CREATE DATABASE notes;
USE notes;

CREATE TABLE entries (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    message TEXT NOT NULL,
```

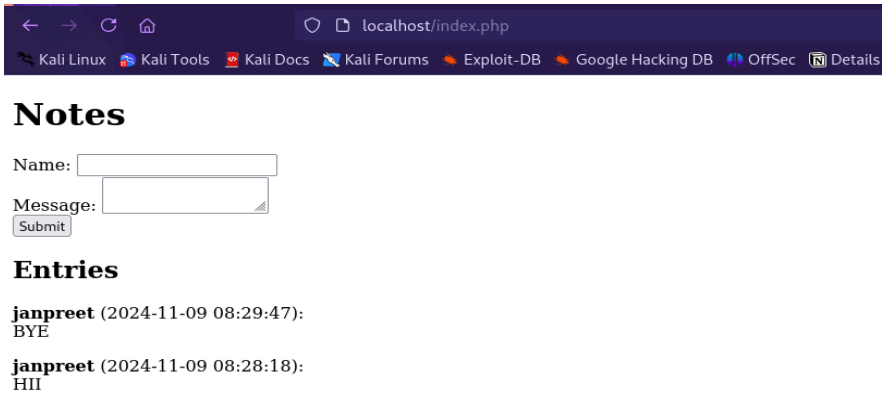
```
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Exit MySQL:

```
EXIT;
```

Now you can check data in mysql





Assignment 49

Title: Create a web application with simple web page containing login details and create a docker image of the application.(Use Ngnix Web server)

Procedure:

Get Docker on system

1.To install Docker on your Ubuntu system, follow these step-by-step instructions

```
sudo apt update
```

2.Install Prerequisites:

Install the required packages for allowing apt to use a repository over HTTPS:

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

3.Add Docker’s GPG Key

Add Docker's official GPG key so that apt can verify the authenticity of the Docker packages:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

4. Set Up the Docker Repository

Add the Docker APT repository to your system:

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

5.Install Docker Engine

Update the package index again and install Docker:

```
sudo apt update
```

sudo apt install docker-ce docker-ce-cli containerd.io

6.Verify: sudo docker --version
(No need to login as we are working with public image and not modifying or creating a new private image.)

Main assignment steps:

1. Open your terminal and create a project folder:

mkdir my_web_app

cd my_web_app

2.Inside my_web_app, create subdirectories for your HTML files and Nginx configuration:

mkdir html

mkdir nginx

3.Create an index.html file in the html directory:

touch html/index.html

4.Add the following code to html/index.html to create a simple login page:

CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login Page</title>
```

```
</head>
<body>
    <h2>Login Page</h2>
    <form action="login" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required><br><br>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required><br><br>
        <button type="submit">Login</button>
    </form>
</body>
</html>
```

5.Create a default.conf file in the nginx directory:

```
touch nginx/default.conf
```

6.Add the following Nginx configuration to nginx/default.conf:

CODE:

```
server {
    listen 80;
    server_name localhost;
    location / {
        root /usr/share/nginx/html;
```

```
        index index.html;
    }}
```

This configuration tells Nginx to serve the index.html file from the /usr/share/nginx/html directory.

7.In the my_web_app directory, create a Dockerfile:

```
touch Dockerfile
```

8.Add the following content to the Dockerfile:

```
# Use the official Nginx image as the base image
FROM nginx:latest

# Copy the custom Nginx configuration file
COPY nginx/default.conf /etc/nginx/conf.d/default.conf

# Copy the HTML file to the Nginx HTML directory
COPY html/index.html /usr/share/nginx/html/index.html

# Expose port 80 to make the web app accessible
EXPOSE 80
```

9.Build the Docker image using the docker build command:

```
docker build -t my_web_app .
```

This command creates a Docker image named my_web_app based on the instructions in the Dockerfile

10.Run a container from the image and map it to a port on your machine:

```
docker run -d -p 8080:80 my_web_app
```

This maps port 8080 on your local machine to port 80 in the container. The -d flag runs the container in detached mode.

11.Open a web browser and navigate to: http://localhost:8080

You should see the login page displayed. Hurraaayyyhh!

—----------

Ohh no…. - - “

Steps to Fix the Permission Denied Error:

Add Your User to the Docker Group:

sudo usermod -aG docker $USER

1. This command adds your user to the docker group, giving you the necessary permissions.
2. Log Out and Log Back In: For the changes to take effect, log out of your current session and log back in. Alternatively, you can restart your computer.

Verify Membership in the Docker Group: Run the following command to check if you are in the docker group:

groups $USER

3. You should see docker listed in the output.

Try Running Docker Again: Run a Docker command (e.g., docker ps) to ensure that the error is resolved:

docker ps

4. If you don't see the permission error, the issue has been fixed.

Alternative Workaround (Temporary)

If you need a temporary solution before logging out or restarting, you can use sudo with your Docker commands:

sudo docker build -t my_web_app .

sudo docker run -d -p 8080:80 my_web_app

This will work for now, but adding your user to the docker group is the recommended solution for convenience and better usability.

If you still see the "permission denied" error after adding your user to the docker group, follow these steps to troubleshoot:

1. Confirm Your User is in the docker Group

Run the following command to check if your user is in the docker group:

groups $USER

2.Ensure that docker is listed in the output. If it is not, re-run:

sudo usermod -aG docker $USER

Then log out and log back in to apply the changes.

ELSE Restart Docker Service:

Restart the Docker service to ensure the changes take effect:

sudo systemctl restart docker

3. Check Docker Daemon Status

Ensure the Docker daemon is running:

sudo systemctl status docker

You should see the status as "active (running)". If it’s not running, start it with:

sudo systemctl start docker

4. Log Out and Log Back In

If you haven’t already, log out of your current user session and log back in. This ensures that group membership changes take effect.

5. Use newgrp Command (Optional) optional but worked ummmmm!!

Chill….If logging out is not feasible, you can use the newgrp command to apply the group change immediately:

newgrp docker

This command will create a new shell session with the updated group.

6. Use sudo as a Temporary Solution

If the issue persists, you can run Docker commands with sudo:

sudo docker ps

This is not ideal for long-term use but will work while you troubleshoot.

Relaxxxxx…Last Resort: Reboot the System

If none of the above steps work, reboot your computer. This will ensure all changes are applied.

You did it even with hurdles..Grrrrrreeattt!!

Assignment 56

To create a Docker image of a simple Django login form that runs on port 6000, follow these steps:

1. **Set Up Django Project**: Create a basic Django project and app for the login form.
2. **Create a Dockerfile**: Define the Docker image configuration.
3. **Create a `docker-compose.yml` file** (optional, for easier management).

Here's how to set everything up:

Step 1: Set Up Django Project

Initialize a Django Project:

```
django-admin startproject login_project
cd login_project
python manage.py startapp login
```

1.

Define a Login Form in `login/forms.py`:

```
from django import forms

class LoginForm(forms.Form):
    username = forms.CharField(label='Username', max_length=100)
    password = forms.CharField(widget=forms.PasswordInput())
```

2.

Create the Login View in `login/views.py`:
python
Copy code

```
from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login
from .forms import LoginForm

def login_view(request):
```

```
    if request.method == 'POST':
        form = LoginForm(request.POST)
        if form.is_valid():
            username = form.cleaned_data['username']
            password = form.cleaned_data['password']
                 user = authenticate(request, username=username,
password=password)
            if user is not None:
                login(request, user)
                return redirect('home')
            else:
                        form.add_error(None, "Invalid username or
password")
    else:
        form = LoginForm()
    return render(request, 'login/login.html', {'form': form})
```

3.

Set Up URL Routing in `login/urls.py`:

```
from django.urls import path
from .views import login_view

urlpatterns = [
    path('', login_view, name='login'),
]
```

4.

Include Login URLs in the Main Project in `login_project/urls.py`:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
```

```
    path('login/', include('login.urls')),
]
```

5.

Create the Login Template in `login/templates/login/login.html`:

```
<h2>Login</h2>
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Login</button>
</form>
```

6.

Migrate Database:

```
python manage.py migrate
```

7.

Create a Superuser (for testing purposes):

```
python manage.py createsuperuser
```

8.

Step 2: Create a Dockerfile

In the `login_project` directory, create a file named `Dockerfile`:

```
# Use the official Python image
FROM python:3.10

# Set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1
```

```
# Set work directory
WORKDIR /app

# Install dependencies
COPY requirements.txt /app/
RUN pip install -r requirements.txt

# Copy project files
COPY . /app/

# Expose port 6000
EXPOSE 6000

# Run Django server on port 6000
CMD ["python", "manage.py", "runserver", "0.0.0.0:6000"]
```

Step 3: Create a `requirements.txt` File

In the `login_project` directory, create a `requirements.txt` file to list the dependencies:

text
Copy code
```
Django==4.2.3
```

Step 4: Build and Run the Docker Image

Build the Docker Image:

```
docker build -t django-login-app .
```

1.

Run the Docker Container:

```
docker run -p 6000:6000 django-login-app
```

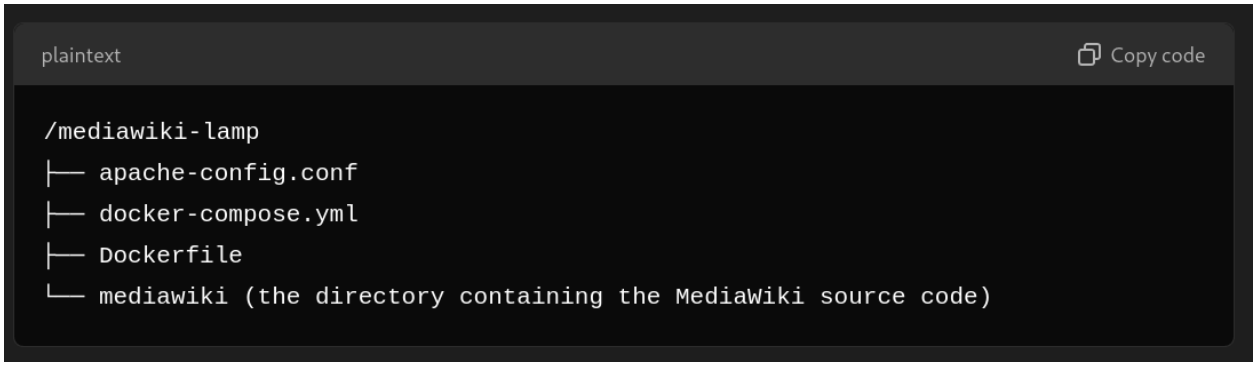
2.

Step 5: Access the Application

Go to `http://localhost:6000/login` in your web browser to see the login form.

Assignment No. 61

Title: Run a LAMP Stack Container at port 8080 and host media wiki site on native machine.



Dockerfile:

```
# Use the PHP 7.4 Apache image
FROM php:7.4-apache

# Install required libraries and PHP extensions
RUN apt-get update && \
    apt-get install -y libicu-dev libpng-dev libjpeg-dev libfreetype6-dev && \
    docker-php-ext-configure intl && \
    docker-php-ext-install intl && \
    docker-php-ext-configure gd --with-freetype --with-jpeg && \
    docker-php-ext-install gd && \
    a2enmod rewrite
```

docker-compose.yml:

```
version: "3"

services:
  apache:
    build:
      context: .
      dockerfile: Dockerfile # We'll use a custom Dockerfile to enable Apache modules.
```

```
    container_name: apache-server
    ports:
      - "8080:80" # Map port 8080 on the host to port 80 in the container
    volumes:
      - ./mediawiki:/var/www/html # Map the MediaWiki files to the Apache
document root
      - ./apache-config.conf:/etc/apache2/sites-available/000-default.conf # Use a
custom Apache config file
    depends_on:
      - mysql
    environment:
      - MYSQL_ROOT_PASSWORD=rootpassword
      - MYSQL_DATABASE=mediawiki
      - MYSQL_USER=mediawiki
      - MYSQL_PASSWORD=mediawikipassword

  mysql:
    image: mysql:5.7
    container_name: mysql-server
    environment:
      - MYSQL_ROOT_PASSWORD=rootpassword
      - MYSQL_DATABASE=mediawiki
      - MYSQL_USER=mediawiki
      - MYSQL_PASSWORD=mediawikipassword
    volumes:
      - mysql-data:/var/lib/mysql # Use a Docker volume for MySQL data
persistence

volumes:
  mysql-data:
```

apache-config.conf:

```
<VirtualHost *:80>
    DocumentRoot /var/www/html
    <Directory /var/www/html>
        AllowOverride All
        Require all granted
```

```
    </Directory>
</VirtualHost>
```

Prepare the MediaWiki Source Files:

```
mkdir mediawiki
cd mediawiki
wget https://releases.wikimedia.org/mediawiki/1.39/mediawiki-1.39.0.tar.gz
tar -xzvf mediawiki-1.39.0.tar.gz --strip-components=1
cd ..
sudo chown -R apache:apache mediawiki
sudo chmod -R 755 mediawiki
```

Build and run Commands:

```
docker-compose up --build -d
docker cp ./mediawiki apache-server:/var/www/html
docker ps
```

Access MediaWiki in Your Browser:

Once the containers are running:
Open your web browser and go to http://localhost:8080.
You will see the MediaWiki installation page.

Complete the Installation Process:

Follow the on-screen instructions to complete the installation.
When you reach the step to download the LocalSettings.php file, download it and run:

```
    docker cp ./mediawiki/LocalSettings.php
apache-server:/var/www/html/LocalSettings.php
```

Ensure the file is owned by the www-data user (inside the container):

```
    docker exec -it apache-server bash
    chown www-data:www-data /var/www/html/LocalSettings.php
    chmod 644 /var/www/html/LocalSettings.php
    exit
```

Restart Apache:

```
docker-compose restart apache
```

Assignment No. 62

Title: Write a C program to create a singly linked list and containerize it.
Prerequisite: https://docs.docker.com/engine/install/ubuntu/

singly_linked_list.c:

```
#include <stdio.h>
#include <stdlib.h>

// Define the structure of a node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the start of the list
void insertAtStart(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

// Function to insert a node at the end of the list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
```

```
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

// Function to insert a node at a specific index
void insertAtIndex(struct Node** head, int data, int index) {
    struct Node* newNode = createNode(data);
    if (index == 0) {
        insertAtStart(head, data);
        return;
    }
    struct Node* temp = *head;
    for (int i = 0; temp != NULL && i < index - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Index out of bounds\n");
        free(newNode);
    } else {
        newNode->next = temp->next;
        temp->next = newNode;
    }
}

// Function to search for an element in the list
void search(struct Node* head, int key) {
    struct Node* temp = head;
    int index = 0;
    while (temp != NULL) {
        if (temp->data == key) {
            printf("Element %d found at index %d\n", key, index);
            return;
```

```
        }
        temp = temp->next;
        index++;
    }
    printf("Element %d not found in the list\n", key);
}

// Function to display the linked list
void displayList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Function to calculate and print the size of the list
void printSize(struct Node* head) {
    int count = 0;
    struct Node* temp = head;
    while (temp != NULL) {
        count++;
        temp = temp->next;
    }
    printf("Size of the list: %d\n", count);
}

// Main menu to interact with the list
int main() {
    struct Node* head = NULL;
    int choice, data, index;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Print list\n");
        printf("2. Print size of the list\n");
```

```
printf("3. Add node at the start\n");
printf("4. Add node at the end\n");
printf("5. Add node at specific index\n");
printf("6. Search for an element\n");
printf("7. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Current List: ");
        displayList(head);
        break;
    case 2:
        printSize(head);
        break;
    case 3:
        printf("Enter the data for the new node at start: ");
        scanf("%d", &data);
        insertAtStart(&head, data);
        break;
    case 4:
        printf("Enter the data for the new node at end: ");
        scanf("%d", &data);
        insertAtEnd(&head, data);
        break;
    case 5:
        printf("Enter the data for the new node: ");
        scanf("%d", &data);
        printf("Enter the index to insert at: ");
        scanf("%d", &index);
        insertAtIndex(&head, data, index);
        break;
    case 6:
        printf("Enter the element to search: ");
        scanf("%d", &data);
        search(head, data);
```

```
            break;
        case 7:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }

   return 0;
}
```

Dockerfile:

```
# Use an official GCC image to compile and run C code
FROM gcc:latest
# Set the working directory inside ahe container
WORKDIR /app
# Copy the C source file to the container
COPY singly_linked_list.c .
# Compile the C program
RUN gcc -o singly_linked_list singly_linked_list.c
# Command to run the executable
CMD ["./singly_linked_list"]
```

To build an image follow the instructions:

1. docker build -t linked_list_app .
2. docker run --rm -it linked_list_app

To push on DockerHub follow the instructions:

1. docker login
2. docker tag linked_list_app your_dockerhub_username/linked_list_app:latest
3. docker push your_dockerhub_username/linked_list_app:latest

Assignment 63

Create a LAMP Stack container and host a web application of your own.

Assignment Objective:

The goal is to create a **LAMP Stack container** and host a basic web application on it. LAMP stands for **Linux, Apache, MySQL, and PHP**—a commonly used stack for web development.

Step-by-Step Guide

Step 1: Set up Docker

- If Docker is not installed, you'll need to install it first. On Linux, open a terminal and run the following commands:

  sudo apt update

  sudo apt install docker.io -y

- Once Docker is installed, start and enable it:

  sudo systemctl start docker

  sudo systemctl enable docker

Step 2: Pull the LAMP Stack Image

- We'll use a pre-built LAMP Docker image for simplicity. Run:

  docker pull fauria/lamp

- If the "permission denied" error then Add Your User to the Docker Group using

  sudo usermod -aG docker $USER

After adding yourself to the Docker group, you need to log out and log back in for the changes to take effect.

Step 3: Create a Directory for the HTML File

- Create a directory on your system to store your web application files:

  mkdir ~/my_web_app

cd ~/my_web_app

Step 4: Create an HTML File for the Application

- Inside the `my_web_app` directory, create an `index.html` file:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Web Application</title>
</head>
<body>
    <h1>Welcome to My Web Application!</h1>
    <p>This is a simple HTML page hosted on a LAMP stack container.</p>
</body>
</html>
```

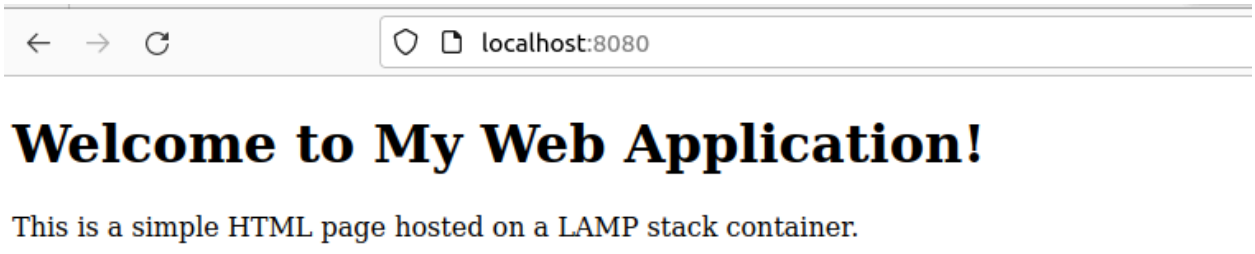
Step 5: Run the LAMP Container

- Start a new Docker container with your HTML file mounted to the web server directory:

```
docker run -d -p 8080:80 -v ~/my_web_app:/var/www/html --name my_lamp_container fauria/lamp
```

Step 6: Access the Web Application

- Open your web browser and visit `http://localhost:8080`. You should see the content of your HTML file displayed.



Step 7: Stopping and Removing the Container (if needed)

- Stop and remove the container with these commands:

  docker stop my_lamp_container

  docker rm my_lamp_container

65. Create & Demonstrate the containers of a particular distro and show all Docker compose file fields. With example

We'll use Docker to create containers, set up a simple service in them (like a web server), and use Docker Compose to manage and run multiple containers.

1. Installing Docker and Docker Compose

Update package database and install prerequisites:

sudo apt update

sudo apt install apt-transport-https ca-certificates curl software-properties-common

Add Docker's GPG key and repository:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

Install Docker:

sudo apt update

sudo apt install docker-ce

Install Docker Compose:

- sudo curl -L "https://github.com/docker/compose/releases/download/$(curl -s https://api.github.com/repos/docker/compose/releases/latest | grep -oP '"tag_name": "\K(.*)(?=")')/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
- sudo chmod +x /usr/local/bin/docker-compose

Verify installation:

docker --version

docker-compose --version

2. Set Up Docker Project Directory

Create a new project directory, navigate into it, and create a new docker-compose.yml file.

- mkdir docker_lab
- cd docker_lab
- touch docker-compose.yml

3. Understanding Docker Compose File Fields

Example of a basic docker-compose.yml file with a web service:

Code :

```
version: "3.8"

services:

  web:

    image: nginx:latest

    ports:

      - "8080:80"

    environment:

      - NGINX_HOST=localhost

      - NGINX_PORT=80

    networks:

      - webnet

  db:

    image: mysql:5.7

    environment:

      MYSQL_ROOT_PASSWORD: example

    ports:

      - "3306:3306"

    volumes:

      - db_data:/var/lib/mysql
```

```
  networks:
    - webnet
volumes:
  db_data:
networks:
  Webnet:
```

4. Running the Docker Compose File

To start the containers, use:

```
docker-compose up -d
```

Verify containers are running:

```
docker ps
```

Access the nginx web server by going to http://localhost:8080 in your browser.

5. Stopping and Cleaning Up Containers

To stop containers:

```
docker-compose down
```

To remove all containers and volumes:

```
docker-compose down --volumes
```

Example Commands and Screenshots

You can take screenshots at various steps to show:

- Docker and Docker Compose version outputs.
- Containers running with docker ps.
- Accessing the web service at http://localhost:8080.

Assignment 57

Create a container with ngnix web server and create one more container with mysql.

Step 1: Create a Network (Optional)

To allow the two containers to communicate with each other, you may want to create a custom Docker network:

```
docker network create my_network
```

Step 2: Run the Nginx Container

Run an Nginx container and connect it to the network created in the previous step:

```
docker run -d --name my_nginx --network my_network -p 8080:80 nginx
```

Step 3: Run the MySQL Container

Next, run a MySQL container and connect it to the same network:

```
docker run -d --name my_mysql --network my_network -e
MYSQL_ROOT_PASSWORD=root_password mysql
```

Step 4: Verify the Containers

To check if both containers are running, use:

```
docker ps
```

Step 5: Access Nginx and MySQL

- **Nginx: Visit `http://localhost:8080` in your browser. You should see the default Nginx welcome page.**
- **MySQL: To connect to the MySQL container, you can use `docker exec` to access it from within the Docker environment:**

```
docker exec -it my_mysql mysql -uroot -p
```

Step 6:

Check on localhost:8080 wether nginix is running or not.

```
rajan@rajan-lature:~/flask_app/templates$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                   NAMES
db83f88bed6b   mysql     "docker-entrypoint.s…"   42 minutes ago   Up 42 minutes   3306/tcp, 33060/tcp                     my_mysql
dc3838da3dd6   nginx     "/docker-entrypoint.…"   46 minutes ago   Up 46 minutes   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   my_nginx
rajan@rajan-lature:~/flask_app/templates$
```

Assignment 58

Steps to Create a Web Form for Inserting Records into MySQL Database

Step 1: Install Apache, MySQL, and PHP

1. Open your terminal and update the package list:
bash
sudo apt update

2. Install Apache:
bash
sudo apt install apache2 -y

3. Install MySQL:
bash
sudo apt install mysql-server -y

4. Secure the MySQL Installation:
sudo mysql_secure_installation

5. Install PHP and Required Modules:
bash
sudo apt install php libapache2-mod-php php-mysql -y

6. Restart Apache:
bash
sudo systemctl restart apache2

Step 2: Create a MySQL Database and Table

1. Log into MySQL as the root user:
bash
sudo mysql -u root -p

2. Create a database and a table:
sql

```
CREATE DATABASE webform_db;
USE webform_db;
CREATE TABLE records (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(50),
        email VARCHAR(100),
        message TEXT
);
```

3. (Optional) Create a dedicated MySQL user:

```
sql
CREATE USER 'php_user'@'localhost' IDENTIFIED BY 'MyS3cur3P@ssword!';
GRANT ALL PRIVILEGES ON webform_db.* TO 'php_user'@'localhost';
FLUSH PRIVILEGES;
```

Step 3: Create the HTML Form and PHP Script

1. Navigate to the Apache Web Directory:

```
bash
cd /var/www/html
```

2. Create an HTML Form File (`form.html`):

```
html
<!DOCTYPE html>
<html lang='en'>
<head>
<meta charset='UTF-8'>
<title>Insert Record</title>
</head>
<body>
<h2>Insert Record Form</h2>
<form action='insert.php' method='post'>
<label for='name'>Name:</label>
<input type='text' id='name' name='name' required><br><br>
<label for='email'>Email:</label>
<input type='email' id='email' name='email' required><br><br>
<label for='message'>Message:</label>
<textarea id='message' name='message' required></textarea><br><br>
<input type='submit' value='Submit'>
</form>
</body>
</html>
```

3. Create the PHP Script (`insert.php`):
php
<?php
$servername = 'localhost';
$username = 'php_user';
$password = 'MyS3cur3P@ssword!';
$dbname = 'webform_db';
$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
    die('Connection failed: ' . $conn->connect_error);
}
$name = $_POST['name'];
$email = $_POST['email'];
$message = $_POST['message'];
$sql = "INSERT INTO records (name, email, message) VALUES ('$name', '$email',
'$message')";
if ($conn->query($sql) === TRUE) {
    echo 'Record inserted successfully!';
} else {
    echo 'Error: ' . $sql . '<br>' . $conn->error;
}
$conn->close();
?>

Step 4: Set Permissions

Set the correct permissions for Apache to read these files:
bash
sudo chown -R www-data:www-data /var/www/html
sudo chmod -R 755 /var/www/html

Step 5: Test the Setup

1. Open a web browser and go to http://localhost/form.html.
2. Fill in the form and submit to test if data is inserted successfully.
3. Verify data in MySQL by checking the records:
sql
USE webform_db;

SELECT * FROM records;

Step 6: Troubleshooting

If you encounter an HTTP 500 error, enable error reporting in `insert.php`:
php
<?php
error_reporting(E_ALL);
ini_set('display_errors', 1);
// Rest of your code...
?>

Check Apache logs:
bash
sudo tail -f /var/log/apache2/error.log

Assignment No - 60

Write a Docker File to pull the Ubuntu with open jdk and write any java application

1.Overview of the Approach

To complete this assignment, I will:

1. Write a basic Java program for a calculator.
2. Create a Dockerfile that installs Java in a lightweight Ubuntu environment and compiles the Java program.
3. Use Docker commands to build a Docker image and run a container that executes the calculator program.
4. Explain each step and command clearly, ensuring

2.Step-by-Step Instructions

Step 1: Install Docker (if not already installed)

```
docker --version
```

Step 2:**Create a new project folder**:

- Open your terminal and create a directory named `JavaDockerApp` to organize your files.

  mkdir JavaDockerApp

  cd JavaDockerApp

Step 3:**Create the Calculator Java Program**:

- Inside `JavaDockerApp`, create a file named `Calculator.java` with the following content:

  // Calculator.java

  import java.util.Scanner;

  public class Calculator {

  public static void main(String[] args) {

  Scanner scanner = new Scanner(System.in);

  System.out.println("Simple Calculator");

  System.out.print("Enter first number: ");

  double num1 = scanner.nextDouble();

```
        System.out.print("Enter second number: ");
        double num2 = scanner.nextDouble();

        System.out.print("Choose an operation (+, -, *, /): ");
        char operation = scanner.next().charAt(0);

        double result;
        switch (operation) {
            case '+':
            result = num1 + num2;
            System.out.println("Result: " + result);
            break;
            case '-':
            result = num1 - num2;
            System.out.println("Result: " + result);
            break;
```

```
            case '*':
            result = num1 * num2;
            System.out.println("Result: " + result);
            break;
            case '/':
            if (num2 != 0) {
                  result = num1 / num2;
                  System.out.println("Result: " + result);
            } else {
                  System.out.println("Error: Cannot
divide by zero.");
            }
            break;
            default:
            System.out.println("Invalid operation.");
            break;
      }
```

```
        scanner.close();
        }
    }
```

Step 4: Create a Dockerfile

1. **Create a Dockerfile** in the `JavaDockerApp` folder with the following content:

```
# Use the official Ubuntu base image
FROM ubuntu:latest

# Install OpenJDK
RUN apt-get update && \
    apt-get install -y openjdk-11-jdk && \
    rm -rf /var/lib/apt/lists/*

# Set JAVA_HOME environment variable
ENV JAVA_HOME /usr/lib/jvm/java-11-openjdk-amd64
```

```
ENV PATH $JAVA_HOME/bin:$PATH


# Set the working directory in the container
WORKDIR /app


# Copy the Java file to the container
COPY Calculator.java /app


# Compile the Java file
RUN javac Calculator.java


# Define the default command to run the application
CMD ["java", "Calculator"]
```

Step 5:**Build the Docker Image**:

- In your terminal, make sure you are in the `JavaDockerApp` directory.

- Run the following command to build the Docker image:

  docker build -t java-calculator .

Explanation:

- `docker build`: Builds a Docker image from the Dockerfile.
- `-t java-calculator`: Tags the image with the name `java-calculator`.
- `.`: Specifies the current directory as the build context.

Check the Image:

- Verify that the image was created successfully by listing Docker images:

  docker images

Step 6:**Run the Container in Interactive Mode**:

- To run the calculator program interactively, use the following command:

  docker run --rm -it java-calculator

Explanation:

- `docker run`: Runs a container from the Docker image.
- `--rm`: Automatically removes the container when it stops.
- `-it`: Runs the container in interactive mode, allowing user input for the calculator program.
- `java-calculator`: Specifies the image name.

Assignment 64 : Create & Demonstrate the container of a particular distro and show all dockerfile fields

Objective: Create a Docker container for a specific Linux distribution (such as Ubuntu), demonstrate the container by building it from a Dockerfile, and explain all the fields in the Dockerfile.

To achieve this assignment, follow these steps:

1. **Choose a base Linux distribution** (e.g., Ubuntu, Kali Linux, Debian).
2. **Write a Dockerfile** to create an image from that base distro.
3. **Build the Docker image** from the Dockerfile
4. **Run the container** created from the image.
5. **Test the container** by installing a package or running commands inside it.
6. **Explain the Dockerfile fields** (like FROM,RUN,CMD etc.) and their purpose.

Steps:

1. Docker installation:

installation link

(You can directly copy 3 commands from above link under install using apt repository)

1. **sudo apt-get update**

sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

**echo **

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \ $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \ sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

2. **sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin**

3. **sudo docker run hello-world**

2. Create folder-named docker-demo: (You will have Dockerfile and Hello.txt in this folder)

3. Create a text file with some content, and Dockerfile with following contents

Cmd to pull image : **docker pull leplusorg/kali**

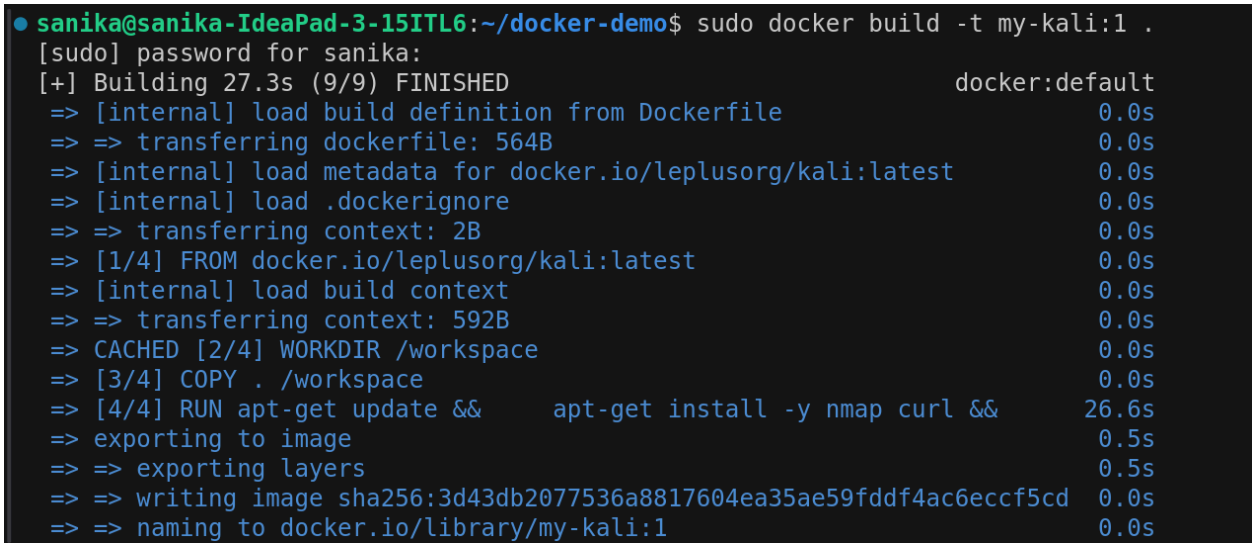
Dockerfile: Copy the following contents in Dockerfile

```
# Use Kali Linux as the base image
FROM leplusorg/kali

# Set the working directory to /workspace
WORKDIR /workspace

# Copy all files from the host's current directory to the /workspace
directory in the container
COPY . /workspace

# Update package lists and install nmap and curl
RUN apt-get update && \
   apt-get install -y nmap curl && \
   apt-get clean && \
   echo "Hello from the /workspace directory in Kali!" > welcome.txt
```

```
# Set the default command to list files in the working directory
CMD ["ls", "-la"]
```

4. sanika@sanika-IdeaPad-3-15ITL6:~/docker-demo$ **sudo docker build -t my-kali:1 .**

Output should be as follows:



5. sanika@sanika-IdeaPad-3-15ITL6:~/docker-demo$ **sudo docker run -it -p 8080:80 my-kali:1 /bin/bash**

Output:

```
sanika@sanika-IdeaPad-3-15ITL6:~/docker-demo$ sudo docker run -it -p 8080:80 my-kali:1 /bin/bash
┌──(root㉿3bf0fb4c827a)-[/workspace]
└─# ls
Dockerfile  Hello.txt  welcome.txt

┌──(root㉿3bf0fb4c827a)-[/workspace]
└─# apt-get update
Hit:1 http://http.kali.org/kali kali-rolling InRelease
Reading package lists... Done

┌──(root㉿3bf0fb4c827a)-[/workspace]
└─# cat /etc/os-release
PRETTY_NAME="Kali GNU/Linux Rolling"
NAME="Kali GNU/Linux"
VERSION_ID="2024.1"
VERSION="2024.1"
VERSION_CODENAME=kali-rolling
ID=kali
ID_LIKE=debian
HOME_URL="https://www.kali.org/"
```

6 Test whether curl has installed using **curl --version**

Output:

```
┌──(root㉿3bf0fb4c827a)-[/workspace]
└─# curl --version
curl 8.10.1 (x86_64-pc-linux-gnu) libcurl/8.10.1 GnuTLS/3.8.6 zlib/1.3 brotli/1.1.0 zstd/1.5.5 libidn2/2.3.
7 libpsl/0.21.2 libssh2/1.11.1 nghttp2/1.59.0 ngtcp2/1.6.0 nghttp3/1.4.0 librtmp/2.3 OpenLDAP/2.5.13
Release-Date: 2024-09-18, security patched: 8.10.1-2
Protocols: dict file ftp ftps gopher gophers http https imap imaps ipfs ipns ldap ldaps mqtt pop3 pop3s rtm
p rtsp scp sftp smb smbs smtp smtps telnet tftp ws wss
Features: alt-svc AsynchDNS brotli GSS-API HSTS HTTP2 HTTP3 HTTPS-proxy IDN IPv6 Kerberos Largefile libz NT
LM PSL SPNEGO SSL threadsafe TLS-SRP UnixSockets zstd
```

Explanation of Dockerfile fields:

1. FROM leplusorg/kali
    - Starts with a Kali Linux image as the base.
2. WORKDIR /workspace
    - Sets /workspace as the directory where all commands will run inside the container.
3. COPY . /workspace
    - Copies all files from your current folder on your computer into /workspace inside the container.
4. RUN apt-get update && apt-get install -y nmap curl
    - Updates the package list and installs nmap (for network scanning) and curl (for data transfer).
5. RUN apt-get clean

    - Cleans up unnecessary files to keep the image small.
6. RUN echo "Hello from the /workspace directory in Kali!" > welcome.txt
    - Creates a file welcome.txt in /workspace with the text "Hello from the /workspace directory in Kali!"
7. CMD ["ls", "-la"]
    - When you run the container, it lists all files in /workspace to show the contents.

Alternative References:

Github_link

Docker_installation

Original_Kali_image

Medium_article

Experiment no-55.

Configure and demonstrate the use of FTP and. (on Ubuntu container) Show the imp steps and file name of configurations. (on answer sheet) Put the Pub folder available for access to all.

 # Step 1

sudo apt update

sudo apt install vsftpd

 # Step 2 configuration file

sudo nano /etc/vsftpd.conf

 # Add the content in that file

listen=YES

listen_ipv6=NO

anonymous_enable=YES

write_enable=YES

anon_upload_enable=YES

anon_mkdir_write_enable=YES

anon_other_write_enable=YES

 # creation of Pub Directory

sudo mkdir -p /srv/ftp/Pub

sudo chmod 777 /srv/ftp/Pub

sudo chown -R ftp:ftp /srv/ftp/Pub

 # Firewall config

sudo ufw status

sudo ufw allow 21/tcp

sudo ufw allow 1024:1048/tcp

sudo ufw reload

sudo ufw allow 10000:10100/tcp

 # Restart

sudo systemctl restart vsftpd

 # Ensure the status is active

sudo systemctl status vsftpd

Telnet configure

sudo apt update

sudo apt install xinetd telnetd -y

sudo nano /etc/xinetd.d/telnet

service telnet

{

disable = no

flags = REUSE

socket_type = stream

wait = no

user = root

server = /usr/sbin/in.telnetd

```
    log_on_failure += USERID
    only_from = 0.0.0.0   # Allows access from any IP
}
sudo service xinetd restart
sudo systemctl enable xinetd
```