[OSS All Experiments](https://docs.google.com/document/d/1z6jpvF3MaAcmzJs90wcFd3s4j6h7HckxiYxhguKg2nA/edit?pli=1)

[**1,4,40,56**](https://raosh.notion.site/raosh/OSS-5b6ed4cd3b624304811533b2fe306082)

**Orange = theory  
Yellow = Available  
red= not available**

| **1** | **7** | **13** | **19** | **25** | **31** | **37** | **43** | **28** | **34** | **40** | **46** | **52** | **58** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2** | **8** | **14** | **20** | **26** | **32** | **38** | **23** | **29** | **35** | **41** | **47** | **53** | **59** |
| **3** | **9** | **15** | **21** | **27** | **33** | **39** | **24** | **30** | **36** | **42** | **48** | **54** | **60** |
| **4** | **10** | **16** | **22** | **28** | **34** | **40** | **25** | **31** | **37** | **43** | **49** | **55** |  |
| **5** | **11** | **17** | **23** | **29** | **35** | **41** | **26** | **32** | **38** | **44** | **50** | **56** |  |
| **6** | **12** | **18** | **24** | **30** | **36** | **42** | **27** | **33** | **39** | **45** | **51** | **57** |  |

1. **VCS: Demonstrate to use of Version Control System (Git offline: on local machine with multiple user and connect to online vcs/github/bitbucket). Multiuser usage with team leader role and coder role/Merge/fork (branching)/diff/versions/commit/pull/push on repository. Compare it with svn. (on answer sheet)**
2. **\*VCS: Demonstrate to use of Version Control System using git containers (Git offline: on local machine with multiple user and connect to online vcs/github/bitbucket). Multiuser usage with team leader role and coder role/Merge/fork (branching)/diff/versions/commit/pull/push on repository. Compare it with svn. (on answer sheet)**

1. Step1: Create new folder

**12.Create packages: Create of Debian packages.**

**Take utility source code (Multiple modules/code packaging of python,C/C++/Java etc) of any open source you like and contribute in terms removing error/bug or adding feature to it.**

**Demonstrate the package on suitable OS and upload at its repository.**

**Compare it with RPM package manager (on answer sheet)**

Step 1: Set Up the Directory Structure

calc

└── calculator

├── DEBIAN

│ └── control

├── usr

│ └── bin

│ └── executable\_file i.e cpp code

Open a terminal and navigate to the location where you want to create the Debian package. Inside this sample folder create another folder , name it DEBIAN (Inside calc dir)

# Create a directory named "calculator"

--mkdir calculator

# Navigate to the " calculator " directory

--cd calculator

# Create a directory named "DEBIAN" inside " calculator "

--mkdir DEBIAN

Step 2: Create the Control File

Now, create the control file inside the DEBIAN directory. You can use a text editor like nano or vim. Here, I'm using nano:

--nano DEBIAN/control

Inside the text editor, add the following control file information:

Package: calculator

Version: 1.0

Section: custom

Priority: optional

Architecture: all

Essential: no

Installed-Size: 1024

Maintainer: SupriyaPawar

Description: Display String.

Step 3: Create Additional Directories

Now, create the additional directories required for the Debian package:

# Inside "sample" directory, create "usr" directory

mkdir -p usr/bin

Step 4: Write a Simple cpp Program

Inside the usr/bin directory, you can create a simple cpp program. Let's create a file named calc:

nano usr/bin/calc.cpp

Write a simple cpp program in the editor:

| #include <iostream>  using namespace std;  int main() {  char operation;  float num1, num2;  cout << "Enter operator (+, -, \*, /): ";  cin >> operation;  cout << "Enter two numbers: ";  cin >> num1 >> num2;  switch (operation) {  case '+':  cout << "Result: " << num1 << " + " << num2 << " = " << num1 + num2;  break;  case '-':  cout << "Result: " << num1 << " - " << num2 << " = " << num1 - num2;  break;  case '\*':  cout << "Result: " << num1 << " \* " << num2 << " = " << num1 \* num2;  break;  case '/':  if (num2 != 0)  cout << "Result: " << num1 << " / " << num2 << " = " << num1 / num2;  else  cout << "Error! Division by zero is not allowed.";  break;  default:  cout << "Error! Invalid operator.";  break;  }  return 0;  } |
| --- |

Save and exit the text editor.

Step 5: Compile and run the cpp Program

Compile the cpp program using the command:

**g++ calc.cpp -o calc**

**./calc**

Step 6: Create the Debian Package

Now, it's time to create the Debian package using the dpkg-deb command. Ensure that the build-essential package is installed on your system:

**sudo apt-get install build-essential**

Then, build the Debian package:

**dpkg-deb --build calculator**

Step 7: Install the Debian Package

Once the package is built, you can install it on your system:

**sudo dpkg -i calculator.deb**

Step 8: Run the Program

After installation, you can run your program: You can run the program from anywhere.

Calc

**13. Create packages: Create of Debian packages.**

**(multiple modules/code packaging of java/c/cpp).**

**Take utility source code of any open source in c/cpp/ajava/python/shell script you like.**

**Demonstrate the package on suitable OS and upload at its repository.**

**Compare it with RPM package manager (on answer sheet)**

Note: 12 & 13 are the same ..just repeated by mistake ig

**14. Project Management tool: Demonstrate the use of Project Management tool: “SONAR” for managing projects.**

[**Project planning and scheduling**](https://www.openproject.org/collaboration-software-features/#project-planning)**/** [**Product roadmap and release planning**](https://www.openproject.org/collaboration-software-features/#product-management)**/** [**Task management and team collaboration**](https://www.openproject.org/collaboration-software-features/#task-management)**/** [**Agile and Scrum**](https://www.openproject.org/collaboration-software-features/#agile-scrum)**/**[**Time tracking, cost reporting and budgeting**](https://www.openproject.org/collaboration-software-features/#time-tracking)**/** [**Bug tracking**](https://www.openproject.org/collaboration-software-features/#bug-tracking) **on any suitable open source (code) from internet.**

**Compare it with other Project Management tool (on answer sheet)**

**15. \*Project Management tool: Demonstrate the features of Project Management tool: “SONAR” for managing projects.**

**Demonstrate the code quality features of sonar for open source code (yours project code) c/cpp/java/python codes.**

**Compare it with other Project Management tool (on answer sheet)**

**16. Project Management tool: Demonstrate the features of Project Management tool: “Jira” for managing projects.**

**Demonstrate the code quality features of jira for open source code (yours project code) c/cpp/java/python codes.**

**Compare it with other Project Management tool (on answer sheet)**

| Jira is a widely used project management tool developed by Atlassian. It offers a variety of features for managing projects, tracking issues, and facilitating collaboration among team members. Let's focus on the key features of Jira and how it handles code quality for open source projects written in C, C++, Java, and Python.  Jira Features:  Issue Tracking:  Jira provides a robust issue tracking system. You can create, prioritize, assign, and track issues throughout the development process.  Issues can be customized to fit various project needs, including bugs, new features, tasks, and improvements.  Agile Boards:  Jira supports Agile methodologies through Scrum and Kanban boards. It allows teams to plan, track, and release software iteratively.  Custom Workflows:  Tailor workflows to match your team's processes. For example, you can set up stages like "To Do," "In Progress," "Code Review," and "Done."  Code Integration:  Jira seamlessly integrates with version control systems like Git, SVN, Mercurial, allowing you to associate code changes with specific issues.  Code Reviews:  For code quality, Jira integrates with Bitbucket, Atlassian's Git repository management solution.  You can create and review pull requests directly within Jira, ensuring code quality before merging changes.  Build and Deployment Integration:  Integrates with continuous integration and deployment tools, such as Jenkins, Bamboo, and others.  Automatically triggers builds and deployments based on code changes.  Reporting and Dashboards:  Jira provides a range of pre-built and customizable reports and dashboards to monitor project progress, team performance, and other metrics.  Code Quality in Jira:  Code Reviews:  Facilitates peer code reviews, ensuring collaboration and maintaining code quality standards.  Code Metrics:  Integrates with plugins and extensions that offer code quality metrics, such as code complexity, test coverage, and adherence to coding standards.  Automated Testing:  Supports integration with testing tools to automate the testing process, helping catch bugs early in the development cycle.  Integration with Static Analysis Tools:  Jira can integrate with static code analysis tools to identify potential issues and enforce coding standards.  Comparison with Other Project Management Tools (e.g., Trello, Asana):  Trello:  Trello is more lightweight and visually oriented, suitable for simpler projects.  Lacks advanced project management features compared to Jira.  Limited code quality and version control integrations.  Asana:  Asana is user-friendly and flexible, suitable for various project management styles.  It lacks the deep integration with code repositories and code review tools that Jira offers.  In summary, Jira excels in managing complex software development projects and ensures code quality through seamless integration with version control systems, code review tools, and continuous integration tools. Its flexibility and extensibility make it a preferred choice for many development teams. |
| --- |

**17. Project Management tool: Demonstrate the features of Project Management tool: “Jira” for managing projects.**

**Demonstrate the code quality features of jira for open source code (yours project code) c/cpp/java/python codes.**

**Compare it with other Project Management tool (on answer sheet)**

**18. Bug Tracking: Demonstrate the use/features of Bug Tracking/management: "YouTrack".**

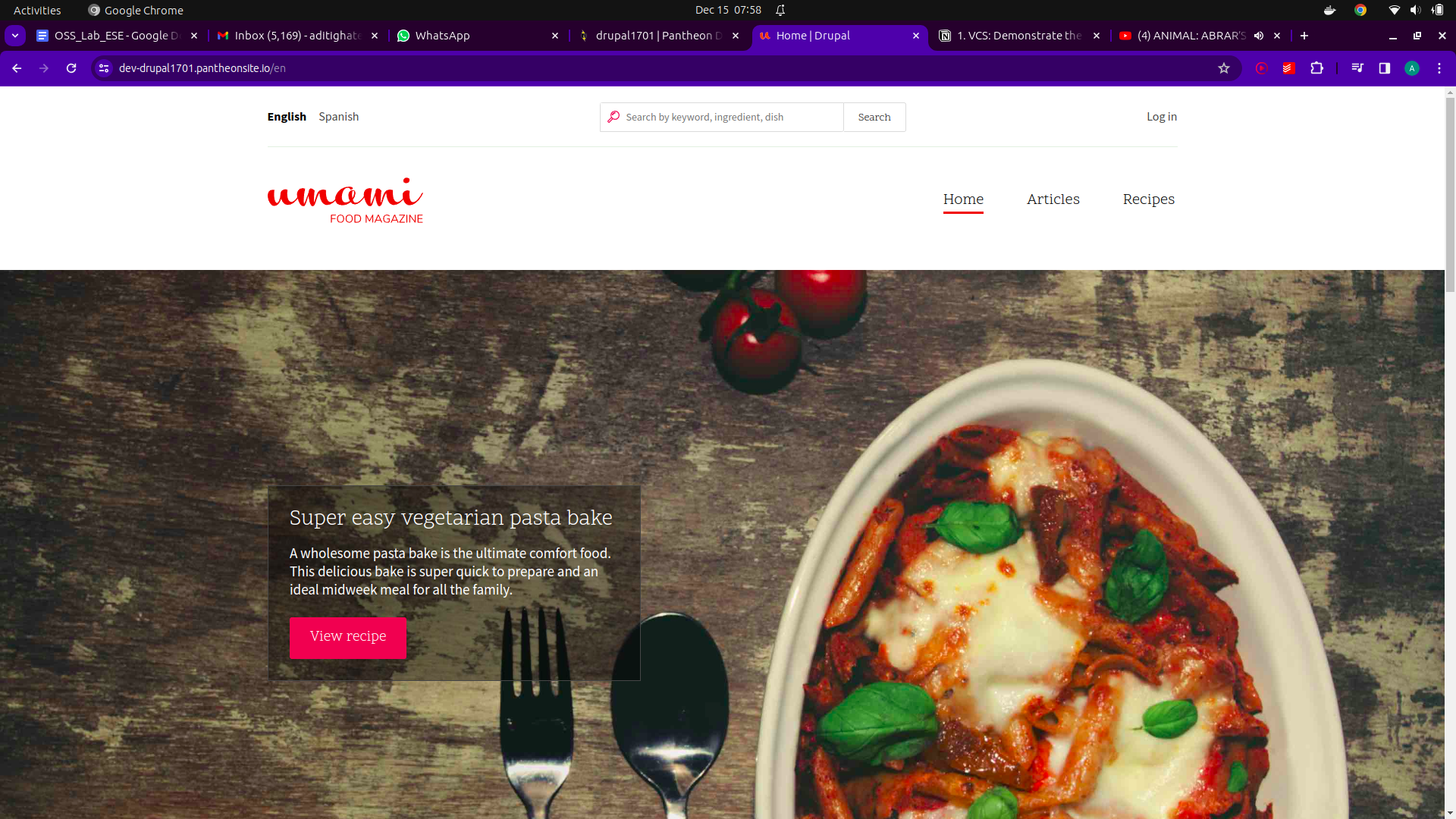
**Take source code of any open source you like and find minimum three bugs (mention their type on answer sheet ) in terms removing error/bug or adding feature to it.**

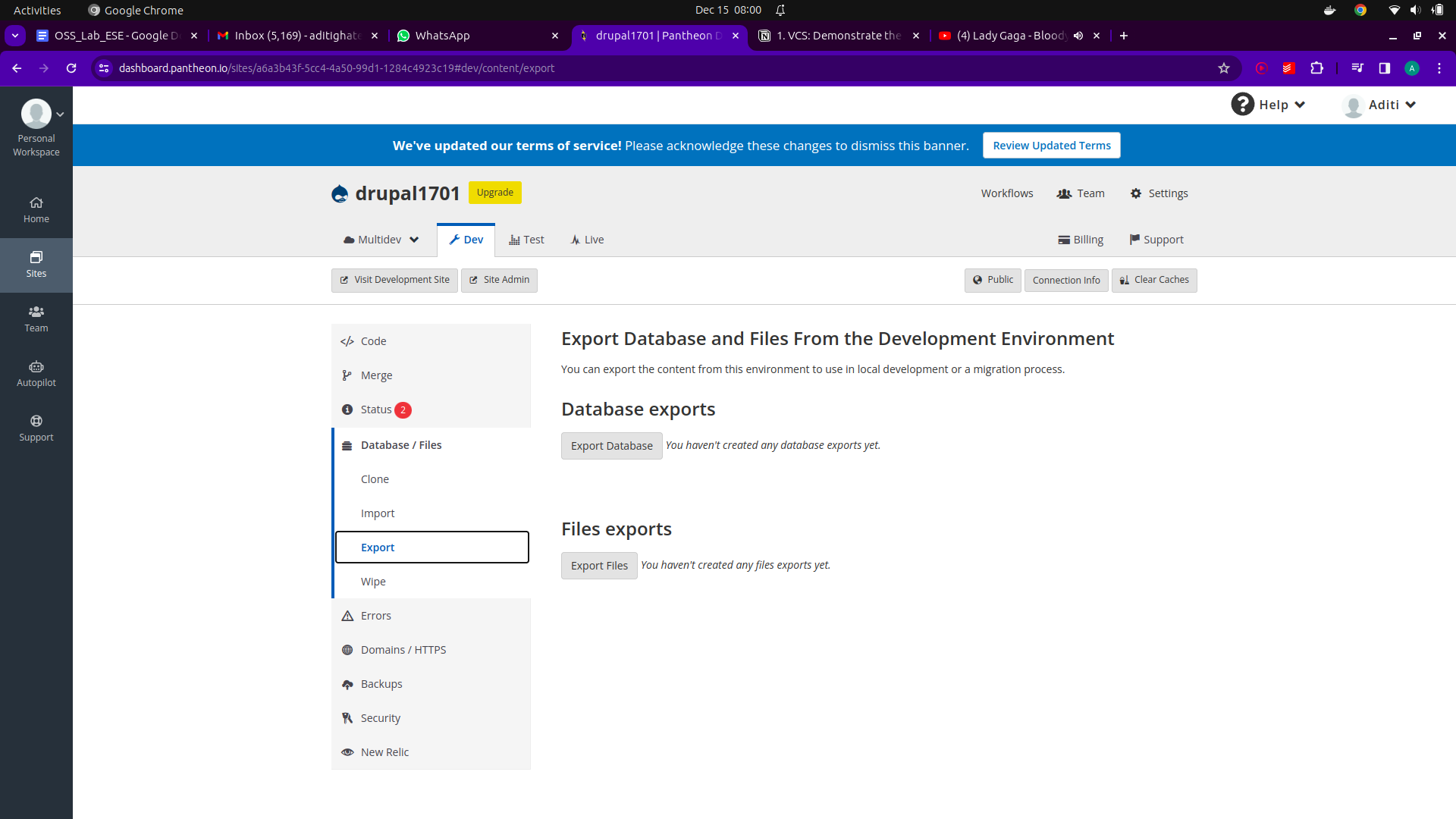
[**Bug tracking**](https://www.openproject.org/collaboration-software-features/#bug-tracking) **on any suitable open source (code) from internet.**

**Compare it with other Bug tracking tool. How bug tracking improves the quality of code (on answer sheet).**

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

| Drupal is a powerful content management system (CMS) that allows users to build and manage websites with flexibility and scalability. Let's walk through some of the key features of Drupal and demonstrate how it manages content for a client.  Drupal Features:  User Management:  Drupal provides a robust user management system, allowing administrators to create and manage user accounts with different roles and permissions.  Users can have specific roles like anonymous user, authenticated user, editor, and administrator.  Content Types:  One of Drupal's strengths is its flexibility in handling different content types. You can define custom content types with specific fields and configurations.  For example, you can create content types for articles, blog posts, events, or any other type of content relevant to the website.  Content Creation and Editing:  Users with the appropriate permissions can easily create, edit, and manage content through a user-friendly interface.  The WYSIWYG editor allows content creators to format text, add media, and create engaging content without needing technical expertise.  Taxonomy:  Drupal includes a powerful taxonomy system that allows you to categorize and tag content. This enhances the organization and navigation of the site.  You can create vocabularies and attach them to content types, enabling a structured way to classify content.  Themes and Templates:  Drupal supports theming, allowing users to change the look and feel of the site easily.  Users can choose from a variety of themes or create custom themes to match the website's design requirements.  Modules and Extensions:  Drupal has a modular architecture, and users can extend its functionality by installing modules.  There are numerous contributed modules available to add features like SEO optimization, social media integration, and more.  Responsive Design:  Drupal is designed to create websites that are responsive and accessible across various devices, ensuring a positive user experience.  Content Management in Drupal:  Create Users:  As an administrator, navigate to the user management section.  Add new users, assign roles, and configure permissions based on the user's responsibilities.  Create Content:  Define content types (e.g., articles, pages) and configure fields for each type.  Users can then create new content items, filling in the fields and using the WYSIWYG editor.  Manage Taxonomy:  Create and manage vocabularies to classify content.  Assign taxonomy terms to content items to create a structured content organization.  Themes and Appearance:  Choose a theme or install a new one.  Customize the theme settings to match the website's branding and design requirements.  Comparison with Joomla:  Drupal:  Known for its flexibility and scalability, making it suitable for complex websites.  Strong developer community and extensive documentation.  Steeper learning curve but offers more customization options.  Joomla:  User-friendly and suitable for small to medium-sized websites.  Easier to learn for beginners.  Offers a balance between features and ease of use.  In summary, Drupal is a powerful CMS suitable for large and complex websites, offering extensive customization options and scalability. Joomla, on the other hand, is more user-friendly and suitable for smaller projects. The choice between them depends on the specific needs and technical expertise of the user or development team. |
| --- |

****

****

**20. CMS: Demonstrate the use/features of CMS software: "Media Wiki” from bitnami or official website. Create users and show how wiki manages contents for a client. Also configure the working of core features of wiki. Can we use wiki as alternative to writing journal Compare it with other CMS (on answer sheet)**

| MediaWiki Features and Demonstration:  MediaWiki is a popular open-source wiki software known for being the engine behind Wikipedia. Let's explore its features and demonstrate how it manages content for a client:  User Management:  MediaWiki allows for the creation of user accounts with different access levels.  Users can be assigned roles such as anonymous users, registered users, and administrators.  Content Management:  MediaWiki is designed for collaborative content creation and editing.  Users can create, edit, and link pages easily using wiki markup or a WYSIWYG editor.  Version Control:  MediaWiki tracks revisions of each page, allowing users to view and revert to previous versions.  This feature is crucial for maintaining a history of changes and tracking contributions.  Categories and Tags:  Content organization is facilitated through categories and tags.  Users can categorize pages, making it easier to navigate and find relevant information.  Templates:  MediaWiki supports the use of templates, allowing users to create standardized content structures.  Templates enhance consistency and make it easier to update information across multiple pages.  Media Integration:  Users can easily embed images, videos, and other media into wiki pages.  This is useful for creating rich and visually engaging content.  Wiki Content Management:  Create Users:  Navigate to the user management section and create new user accounts.  Assign appropriate roles and permissions based on user requirements.  Create and Edit Pages:  Users can create new pages by simply linking to a non-existent page.  Edit existing pages using wiki markup or the visual editor.  Version History:  Each page maintains a history of revisions.  Users can compare versions and revert to previous states if needed.  Categories and Tags:  Categorize pages to organize content thematically.  Use tags to enhance searchability and content discoverability.  Templates:  Create templates for standardized content structures.  Insert templates into pages to maintain consistency across the wiki.  Wiki as an Alternative to Journal Writing:  Pros:  Collaborative Editing: Multiple users can contribute to a wiki, making it suitable for collaborative projects or journals.  Version Control: Keeps a detailed history of changes, allowing users to track edits over time.  Ease of Organization: Categories, tags, and templates make it easy to organize and structure content.  Cons:  Limited Formatting: Wiki markup may be limiting for users accustomed to more advanced formatting options.  Public Accessibility: Wikis are often designed for public access, which may not be suitable for private journaling.  Comparison with Other CMS:  MediaWiki vs. Confluence:  MediaWiki: Open-source, widely used, and free. Strong community support.  Confluence: Commercial product by Atlassian. Offers advanced features, tighter integration with Jira, and better support but comes with a cost.  MediaWiki vs. DokuWiki:  MediaWiki: Feature-rich, suitable for larger projects. More complex setup.  DokuWiki: Lightweight, simpler to set up and use. Better for smaller projects or personal use.  In summary, MediaWiki is a powerful and flexible wiki platform with features tailored for collaborative content creation. Its suitability as an alternative to journal writing depends on specific use cases and preferences. |
| --- |

**24. Project Management tool: Demonstrate the use/features of Project Management tool: “Open Atrium” for managing. 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 it with other Project Management tool (on answer sheet)x`**

| Open Atrium Features:  Project Planning and Scheduling:  Open Atrium provides tools for creating project plans, defining milestones, and scheduling tasks.  Users can set due dates, assign tasks, and visualize project timelines.  Product Roadmap and Release Planning:  The platform supports the creation of product roadmaps, allowing teams to plan and communicate the strategic direction of their projects.  Release planning features help in organizing and scheduling product releases.  Task Management and Team Collaboration:  Open Atrium includes task management features where users can create, assign, and track the progress of tasks.  Collaboration tools, such as discussion forums, allow for team communication and sharing of ideas.  Agile and Scrum:  Open Atrium can be customized to support Agile and Scrum methodologies.  Features like task boards, user stories, and sprint planning can be implemented for Agile project management.  Time Tracking, Cost Reporting, and Budgeting:  Time tracking functionalities allow team members to log the time spent on tasks.  Cost reporting and budgeting tools help in monitoring project expenses and staying within budget.  Bug Tracking:  Open Atrium likely includes bug tracking features to log, prioritize, and track the resolution of software issues.  Demonstration on an Open Source Project:  To demonstrate the features of Open Atrium, you can use a suitable open-source project from platforms like GitHub. For example, select a project written in a programming language such as Python or JavaScript. Here's how you might apply Open Atrium features:  Project Planning:  Create a project plan within Open Atrium, defining tasks, assigning responsibilities, and setting due dates.  Product Roadmap and Release Planning:  Use Open Atrium to create a visual representation of the product roadmap.  Plan and schedule upcoming releases, incorporating features and milestones.  Task Management and Collaboration:  Create tasks within Open Atrium for specific development or testing activities.  Assign tasks to team members and use collaboration features for discussions.  Agile and Scrum:  If the open-source project follows Agile or Scrum methodologies, adapt Open Atrium to support sprint planning, backlog management, and task boards.  Time Tracking and Budgeting:  Log time spent on tasks to demonstrate time tracking capabilities.  Utilize budgeting features to set project budgets and monitor expenses.  Bug Tracking:  Log and prioritize bugs within Open Atrium.  Track the resolution status, associated discussions, and any adjustments made to the project plan.  Comparison with Other Project Management Tools:  Compare Open Atrium with other project management tools such as Jira, Asana, or Trello. Consider factors like:  Ease of use and user interface.  Integration capabilities with other tools and platforms.  Flexibility and customization options.  Pricing and licensing models.  Support for specific project management methodologies (Agile, Scrum, Kanban).  Selecting the right tool depends on the specific needs, preferences, and workflows of the project team. Always check the latest information and updates from official sources for the most accurate and current details. |
| --- |

**13. Configure and demonstrate the use of FTP and Telnet.  
Show the imp steps and file name of configurations. (on answer sheet)**

| Step 1: Install and Configure FTP Server  Installation:  Install an FTP server software. For example, vsftpd (Very Secure FTP Daemon) on Linux.  bash  Copy code  sudo apt-get update  sudo apt-get install vsftpd  Configuration File:  Edit the vsftpd configuration file. The configuration file is typically located at /etc/vsftpd.conf.  bash  Copy code  sudo nano /etc/vsftpd.conf  Configure settings such as anonymous access, user permissions, etc.  Sample Configuration:  conf  Copy code  anonymous\_enable=NO  local\_enable=YES  write\_enable=YES  chroot\_local\_user=YES  Save and exit the file.  Restart FTP Server:  Restart the FTP server to apply the changes.  bash  Copy code  sudo service vsftpd restart  Step 2: Create Users  Create Users:  Use the adduser command to create new users.  bash  Copy code  sudo adduser user1  sudo adduser user2  Follow the prompts to set passwords and additional information.  Step 3: Create Public Folder  Create Public Folder:  As the admin (root or a user with sudo privileges), create a shared folder. For example, /home/admin/public.  bash  Copy code  sudo mkdir /home/admin/public  Set permissions for the folder.  bash  Copy code  sudo chmod 755 /home/admin/public  sudo chown admin:admin /home/admin/public  Share Files:  Copy files to the public folder for sharing.  bash  Copy code  sudo cp /path/to/file.txt /home/admin/public  Adjust file permissions if necessary.  Step 4: Test FTP Access  FTP Client:  Use an FTP client (e.g., FileZilla) to connect to the FTP server.  Host: Your server's IP address or domain  Username: User1, User2, or any created user  Password: The password set during user creation  Port: Default is 21  Transfer Files:  Use the FTP client to upload/download files to/from the server.  These steps provide a basic configuration for FTP on a Linux server. Ensure that you adapt the instructions based on your specific server environment and requirements. |
| --- |

| **Telnet:**  1. In one machine/terminal, configure the server for telnet  sudo apt install telnetd xinetd  2. Check if it is running  sudo systemctl status xinetd.service  3. If is not active/running  sudo systemctl start xinetd.service  4. Create Telnet file  sudo nano /etc/xinetd.d/telnet  Write below in the file  service telnet  {  disable = no  flags = REUSE  socket\_type = stream  wait = no  user = root  server = /usr/sbin/in.telnetd  log\_on\_failure += USERID  }  5. Then save and close the file and restart xinetd.service as follows:  sudo systemctl restart xinetd.service  6. Telnet server uses port 23 for listening to the incoming connections. Therefore, you will  need to open this port in your firewall. Run the command below to do so :  sudo ufw allow 23  7. Note the ip address  (Type ifconfig in terminal and get the IP ->)10.10.13.226 //in my case  8. Open new terminal which would be the client  Now you can connect to your Telnet server from another machine (where the Telnet  client is installed). On your client machine, use the following command syntax to connect  to the Telnet server:  telnet 10.10.13.226 |
| --- |

**25.** **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).**

Step 1: **Create a New Directory**

Open your terminal and navigate to the location where you want to create the directory for your experiment.

**Step 2: Create Flask Application**

Create a file named app.py in the directory with the following content:

| # app.py  from flask import Flask  app = Flask(\_\_name\_\_)  @app.route('/')  def hello\_world():  return 'Hello, Flask!' |
| --- |

Step 3: Create Dockerfile for Flask Application

Create a file named Dockerfile in the same directory with the following content:

| # Dockerfile  FROM python:3.9-slim  WORKDIR /app  COPY . /app  RUN pip install --no-cache-dir Flask  EXPOSE 5000  CMD ["python", "app.py"] |
| --- |

Step 4: **Build Docker Image for Flask Application**  
Build the Docker image for the Flask application:  
**docker build -t flask-hello-world .**

Step 5: **Create Docker Network**Create a Docker network to allow communication between containers:  
**docker network create my-network**

Step 6**: Run Flask Application Container  
docker run -d --network my-network --name flask-app flask-hello-world**

Step 7: **Pull and Run WordPress Container**Pull the official WordPress image and run the WordPress container, connecting it to the same network:

**docker pull wordpress  
docker run -d --network my-network --name wordpress-container wordpress**

Step 8**: Create Nginx Configuration**Create a file named nginx.conf in the LoadBalancingExperiment directory with the following content:

| # nginx.conf  events {}  http {  upstream backend {  server flask-app:5000; # Flask application  server wordpress-container:80; # WordPress container  }  server {  listen 80;  location / {  proxy\_pass http://backend;  }  }  } |
| --- |

**Step 9: Run Nginx Container**

Run the Nginx container, linking it to the Flask and WordPress containers within the my-network network:

**docker run -d --network my-network -p 80:80 --name nginx-container -v $(pwd)/nginx.conf:/etc/nginx/nginx.conf:ro nginx**

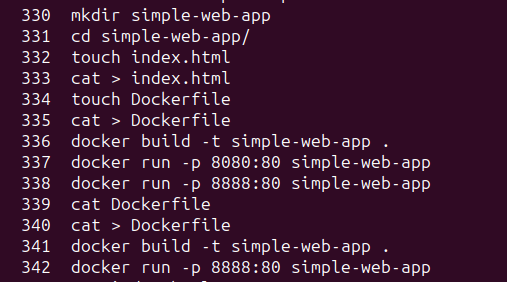
Step 10: **Access Load-Balanced Application**  
Visit http://localhost in your web browser. Nginx, acting as a reverse proxy, will distribute the requests between the Flask and WordPress applications.

**Takeaway:**

| In this experiment, we learned how to create a simple Python Flask application, containerize it using Docker, and set up a Docker network. We extended the experiment by introducing a WordPress application and an Nginx container to act as a reverse proxy for load balancing between the Flask and WordPress containers. The step-by-step process involved creating Dockerfiles, building Docker images, running containers, and utilizing Docker networks to enable communication between containers. We also explored basic Nginx configuration for load balancing. This hands-on experience reinforced fundamental Docker concepts, such as containerization, networking, and the role of reverse proxies in orchestrating and distributing traffic across multiple applications. The experiment serves as a foundational exploration into the practical aspects of containerized application deployment and orchestration. |
| --- |

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

Commands:



1. Create a directory simple-web-app
2. cd simple-web-app
3. Create a html file in the same folder.

index.html:

| <!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</h2>  <form action="/login" 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>  <input type="submit" value="Login">  </form>  </body>  </html> |
| --- |

1. Create a Dockerfile in the same folder.

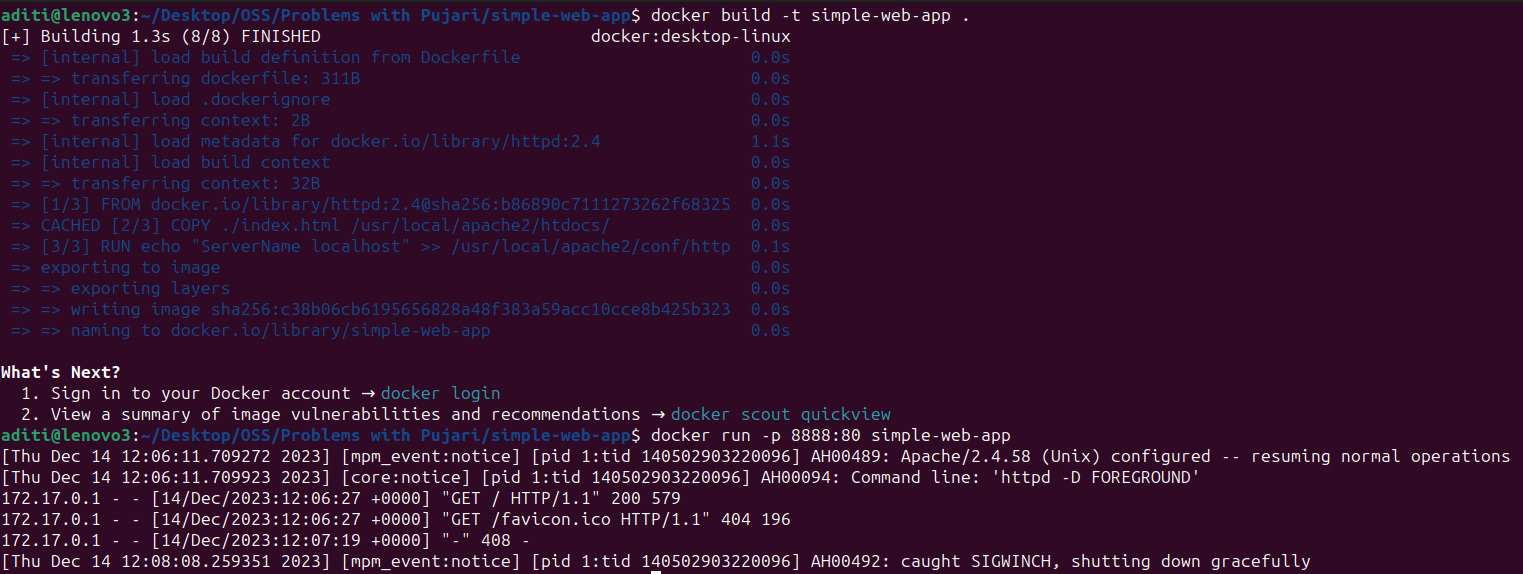
Dockerfile:

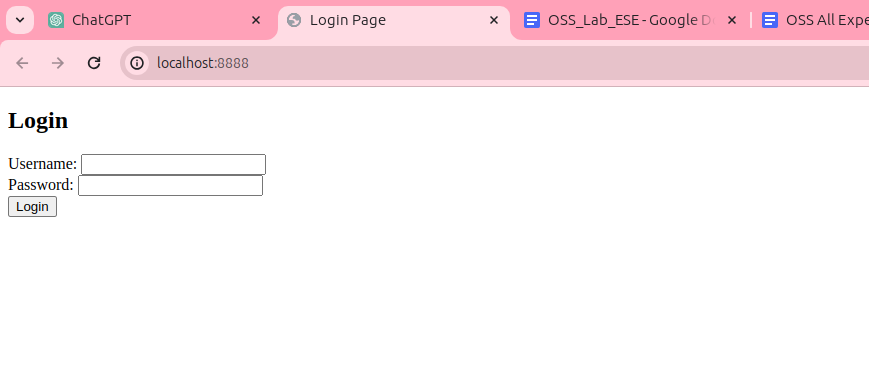
| # Use the official Apache image as the base image  FROM httpd:2.4  # Copy the HTML file to the Apache web server's document root  COPY ./index.html /usr/local/apache2/htdocs/  # Suppress AH00558 warning  RUN echo "ServerName localhost" >> /usr/local/apache2/conf/httpd.conf |
| --- |

1. Build the docker image

**docker build -t simple-web-app**

1. Run the application on available port  
   **docker run -p 8888-80 simple-web-app**

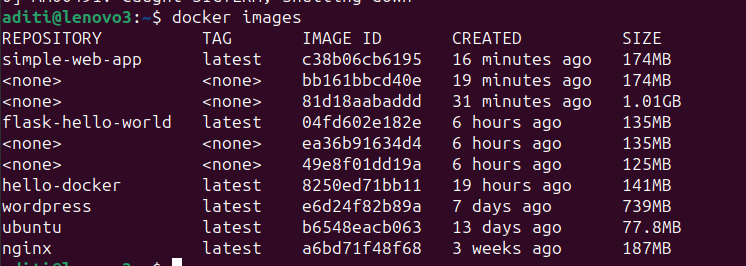
****

****

**23. Docker: Run the Docker container from a recently created image and run the container at port number 80 in the host system.**

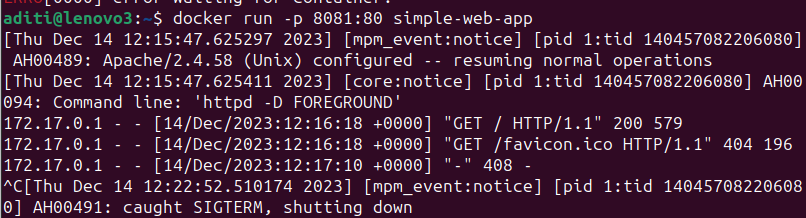
**28. Docker: Run the Docker container from recently created image and run the container at port number 80 in host system**

1. List out the available docker images on the device  
   **docker images**

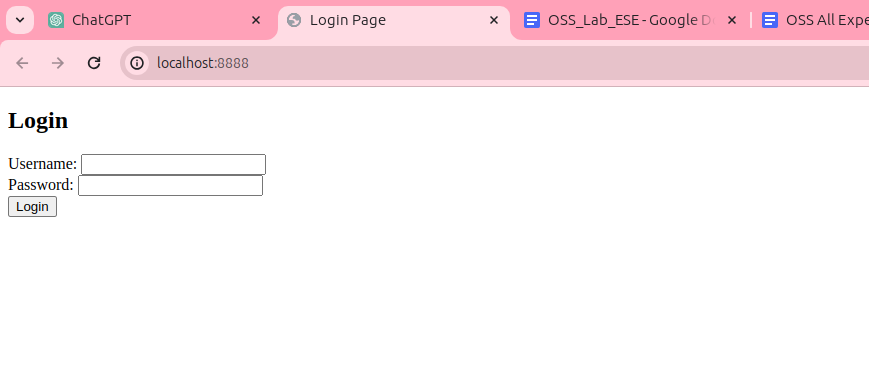
****

1. To run a Docker container from a recently created image and expose it on port 80(any available port) of the host system, you can use the following command.

**docker run -p 8081-80 simple-web-app**

****

1. Output: Same as above 43rd.

****

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

1. Create a directory where you want to perform the experiment.

**mkdir ExpB24**

1. Create a python file

| # calculator.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 "Cannot divide by zero"  return x / y  # Test the functions  num1 = 10  num2 = 5  print(f"Addition: {add(num1, num2)}")  print(f"Subtraction: {subtract(num1, num2)}")  print(f"Multiplication: {multiply(num1, num2)}")  print(f"Division: {divide(num1, num2)}") |
| --- |

1. Create a Dockerfile

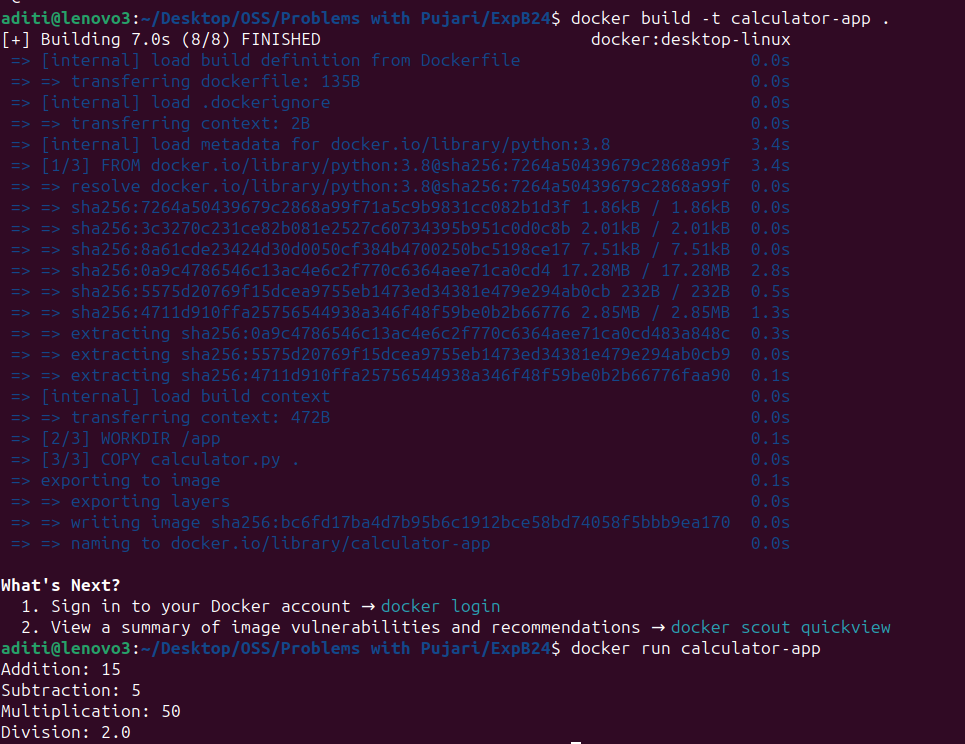
| # Dockerfile  FROM python:3.8  WORKDIR /app  COPY calculator.py .  CMD ["python", "calculator.py"] |
| --- |

This Dockerfile uses the official Python 3.8 image as the base image, sets the working directory to /app, copies the calculator.py script into the container, and specifies the command to run the Python script.

1. Build the docker image:

**docker build -t calculator-app .**

1. Run the docker container:



**25. Docker: Run the Docker container with created image for c/java/python**

1. Create a directory where you want to perform the experiment

**mkdir ExpB25**

1. Create a C-program:

| #include <stdio.h>  int main() {  printf("Hello from C!\n");  return 0;  } |
| --- |

1. Create a Dockerfile

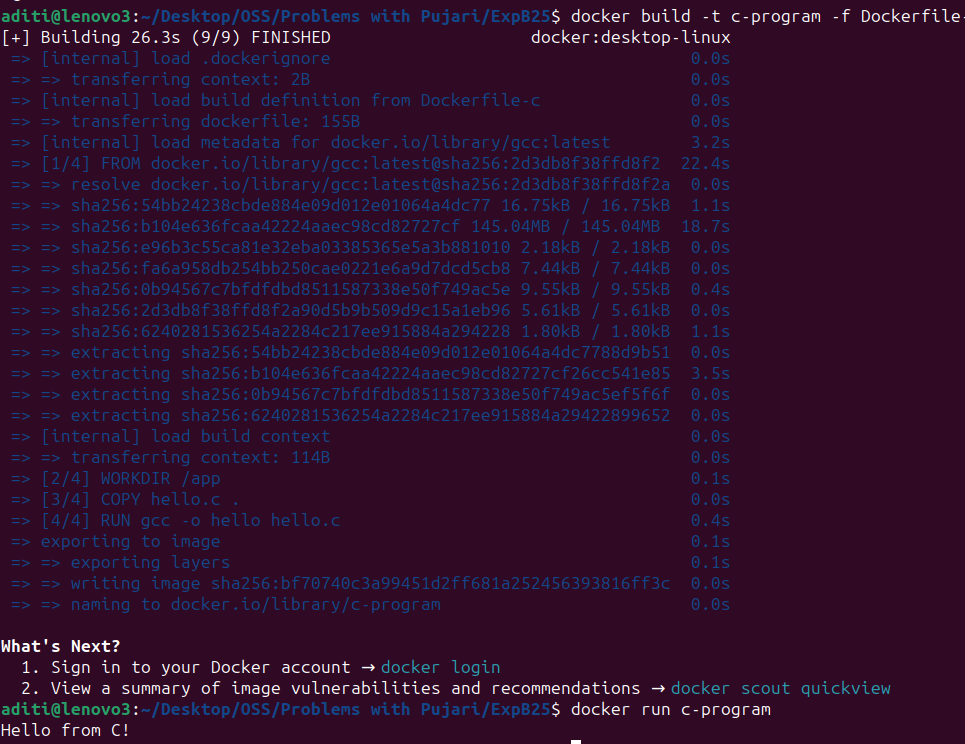
| # Dockerfile for C program  FROM gcc:latest  WORKDIR /app  COPY hello.c .  RUN gcc -o hello hello.c  CMD ["./hello"] |
| --- |

1. Build the docker image

**docker build -t c-program -f Dockerfile-c .**

1. Run the docker container.

**docker run c-program**

****

**26. Docker: Create a simple web application using LAMP Stack on docker container.**

1. Create a directory where you want to perform the experiment.

**mkdir ExpB26**

**cd ExpB26**

1. Create a Dockerfile for Apache/PHP Container.

| # Dockerfile for Apache/PHP  FROM php:7.4-apache  # Install mysqli extension for PHP  RUN docker-php-ext-install mysqli |
| --- |

1. Create a Dockerfile for the MySQL container.

| # Dockerfile for MySQL  FROM mysql:5.7  # Environment variables for MySQL  ENV MYSQL\_ROOT\_PASSWORD=root  ENV MYSQL\_DATABASE=mydatabase  ENV MYSQL\_USER=myuser  ENV MYSQL\_PASSWORD=mypassword |
| --- |

1. Create a **docker-compose.yml** File.  
   This Docker Compose file defines two services: web (Apache/PHP) and db (MySQL).

| version: '3'  services:  web:  build: .  ports:  - "8080:80"  volumes:  - ./html:/var/www/html  depends\_on:  - db  db:  build:  context: .  dockerfile: Dockerfile-mysql  ports:  - "3306:3306"  environment:  MYSQL\_ROOT\_PASSWORD: root  MYSQL\_DATABASE: mydatabase  MYSQL\_USER: myuser  MYSQL\_PASSWORD: mypassword |
| --- |

The volumes section mounts the ./html directory from your host into the /var/www/html directory in the container. This is where your PHP files can be placed.

1. Create a PHP file.

| <?php  $conn = new mysqli("db", "myuser", "mypassword", "mydatabase");  if ($conn->connect\_error) {  die("Connection failed: " . $conn->connect\_error);  }  echo "Connected to MySQL successfully";  $conn->close();  ?> |
| --- |

1. Build and run the Docker Containers.

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

1. **Create a directory for your project and navigate into it:**

**mkdir ExpB27**

**cd ExpB27**

**mkdir web-app**

**cd web-app**

1. Create a web application

index.html:

| <!-- index.html -->  <!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>  <h1>Login Page</h1>  <form>  <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>  <input type="submit" value="Login">  </form>  </body>  </html> |
| --- |

1. Create a Dockerfile

| # Dockerfile  FROM nginx:latest  COPY . /usr/share/nginx/html |
| --- |

This Dockerfile uses the official Nginx image and copies the contents of the current directory into the default Nginx web root.

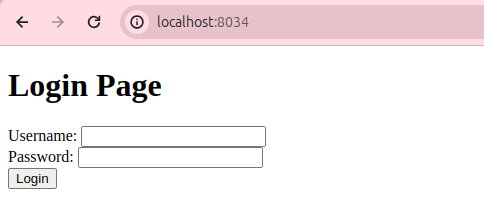
1. Build the Docker image

**docker build -t my-web-app .**

1. Run the Docker Container.

**docker run -p 8034:80 my-web-app**

http://localhost:8034



**29. Create a simple Hello-world python flask application and create the docker image of that Flask application.**

**36. \*Docker: A. Create a simple Hello-world python flask application and create the docker image of that Flask application.**

Step 1: Install Docker on Ubuntu  
Make sure Docker is installed on your Ubuntu system. If not, you can install it using the following commands:  
**sudo apt update  
sudo apt install docker.io  
sudo systemctl start docker  
sudo systemctl enable docker**Step 2: Create a Simple Flask Application  
Create a directory for your Flask application and navigate into it:  
**mkdir flask\_hello\_world  
cd flask\_hello\_world**

Now, create a file named app.py with the following content:

| from flask import Flask  app = Flask(\_\_name\_\_)  @app.route('/')  def hello\_world():  return 'Hello, World!'  if \_\_name\_\_ == '\_\_main\_\_':  app.run(host='0.0.0.0', port=5000) |
| --- |

Step 3: Create a Dockerfile

In the same directory, create a file named Dockerfile without any file extension. This file will define the Docker image configuration:

| # Use the official Python image as the base image  FROM python:3.9-slim  # Set the working directory  WORKDIR /app  # Copy the current directory contents into the container at /app  COPY . /app  # Install Flask  RUN pip install --no-cache-dir Flask  # Make port 5000 available to the world outside this container  EXPOSE 5000  # Define environment variable  ENV NAME World  # Run app.py when the container launches  CMD ["python", "app.py"] |
| --- |

Step 4: Build the Docker Image

Now, in the same directory where your Dockerfile is located, build the Docker image using the following command:

**docker build -t flask-hello-world** .

This command will use the Dockerfile to create an image named flask-hello-world.

Step 5: Run the Docker Container

After the image is built, you can run a container based on that image:

**docker run -p 5000:5000 flask-hello-world**

Step 6: Access the Flask Application

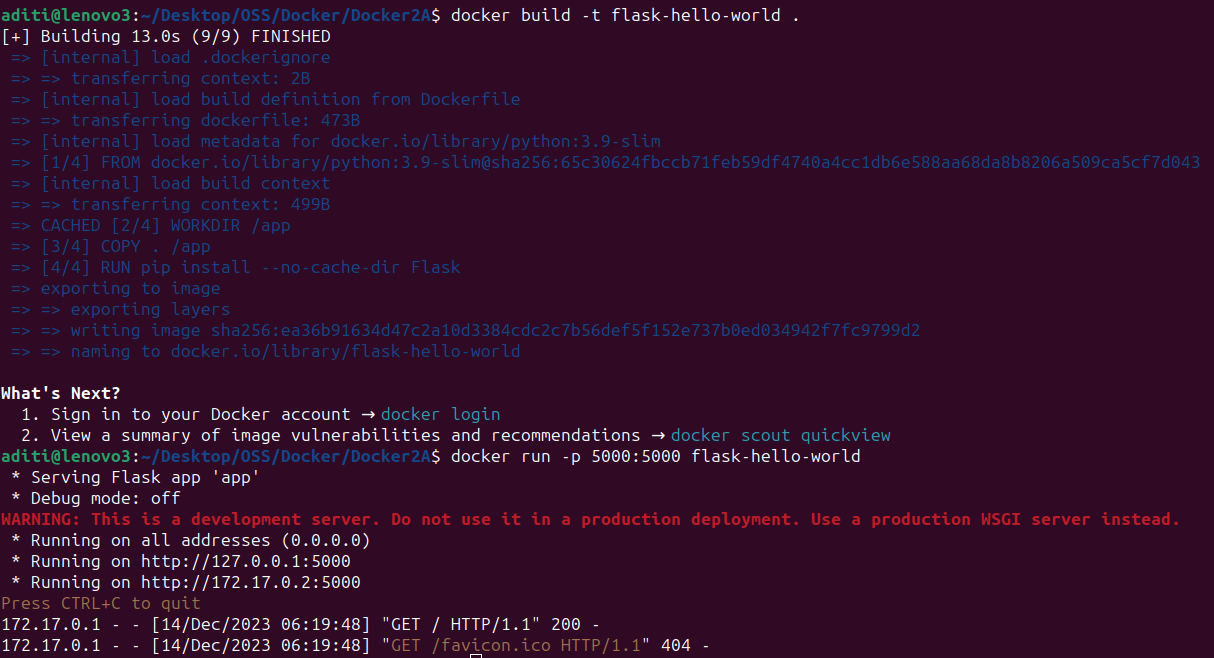
Open a web browser and navigate to:

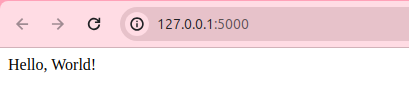
**http://localhost:5000**

You should see the "Hello, World!" message.

That's it! You've successfully created a simple Hello World Python Flask application and created a Docker image for it on Ubuntu.

**Output:**

****

****

**Takeaway:**

| In this exercise, we learned how to create a simple "Hello, World!" Python Flask application and containerize it using Docker. We started by writing a basic Flask app in the app.py file, then created a Dockerfile specifying the Python base image, setting the working directory, copying the application files, and installing Flask. We built a Docker image using the docker build command and ran a Docker container with the docker run command, exposing the application on port 5000. We accessed the running Flask application by visiting http://localhost:5000 in the browser. This hands-on experience introduced the fundamental concepts of creating Docker images for Python applications, handling dependencies, and running applications in isolated containers, emphasizing the importance of creating production-ready configurations for real-world deployments. |
| --- |

**30. Run the docker container from recently created image and run that docker container to 5000 port of host system.  
32. Docker: Create a Docker image of simple web application from using HTTP web server at port 5000 in host.  
36 B. Run the docker container from recently created image and run that docker container to 5000 port of host system.**

Certainly! If you've already built the Docker image named flask-hello-world, you can follow these steps to run a container from that image and map the container's port 5000 to the host system's port 5000:

1. Open a Terminal Window.

2.Navigate to the Project Directory:

Change the current working directory to the one where your Dockerfile and app.py are located

# Run the Docker container

**docker run -p 5000:5000 flask-hello-world**

This command runs a new container from the flask-hello-world image and maps the container's port 5000 to the host system's port 5000.

The Flask application inside the container should now be accessible at http://localhost:5000 on your host system.

Keep in mind that if the container with the same name is already running or the port 5000 on your host system is in use, you might need to stop the existing container or choose a different port.

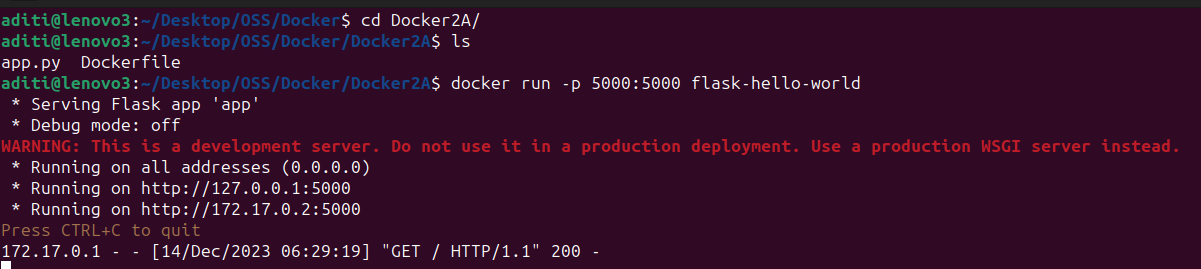
To stop the container, you can open a new terminal and use the following command:

**docker ps.**This command will display a list of running containers along with their container IDs. Identify the container ID associated with your Flask application and use the following command to stop it:

**docker stop <container\_id>**

Replace <container\_id> with the actual container ID. Once the existing container is stopped, you can re-run the docker run command to start a new container.

**Output:**

****

**Takeaway:**

| In this exercise, we learned how to run a Docker container from a previously created image for a simple Python Flask application. We used the docker run command to start the container, mapping port 5000 from the container to port 5000 on the host system. This allowed us to access the Flask application by visiting http://localhost:5000 in a web browser. The hands-on experience reinforced the fundamental concepts of containerization, emphasizing the portability and isolation of applications encapsulated within Docker containers. We also observed how to interact with Docker containers using basic terminal commands, such as starting and stopping containers. |
| --- |

**31. Docker: Pull the LAMP Stack container from docker hub and host a web application of your own.**

1. Pull LAMP Stack Container:

Open a terminal and run the following command to pull the LAMP stack container from Docker Hub:

**docker pull bitnami/lamp:latest**

**35. With the help of Docker-compose deploy the ‘Wordpress’ and ‘Mysql’ container and access the front end of ‘Wordpress’**

Below are step-by-step instructions to deploy WordPress and MySQL containers using Docker Compose and access the WordPress frontend:

Step 1: **Install Docker and Docker Compose**

Make sure you have Docker and Docker Compose installed on your system. You can download and install them from the official Docker website.

Make sure Docker is installed on your Ubuntu system. If not, you can install it using the following commands:

| **sudo apt update**  **sudo apt install docker.io**  **sudo systemctl start docker**  **sudo systemctl enable docker** |
| --- |

Step 2: **Create a Docker Compose file**

Create a file named docker-compose.yml in a directory of your choice. This file will define the services and their configurations.

| version: '3.8'  services:  wordpress:  image: wordpress  ports:  - "8080:80"  environment:  WORDPRESS\_DB\_HOST: mysql  WORDPRESS\_DB\_USER: wordpress  WORDPRESS\_DB\_PASSWORD: password  WORDPRESS\_DB\_NAME: wordpress  depends\_on:  - mysql  networks:  - my\_network  mysql:  image: mysql:5.7  environment:  MYSQL\_ROOT\_PASSWORD: root\_password  MYSQL\_DATABASE: wordpress  MYSQL\_USER: wordpress  MYSQL\_PASSWORD: password  volumes:  - mysql\_data:/var/lib/mysql  networks:  - my\_network  networks:  my\_network:  driver: bridge  volumes:  mysql\_data: |
| --- |

Step 3: **Run Docker Compose**

Open a terminal and navigate to the directory where your docker-compose.yml file is located. Run the following command:

**sudo docker-compose up -d**

This command will download the necessary images and start the containers in detached mode.

Step 4: **Access WordPress Frontend**

Once the containers are up and running, you can access the WordPress frontend by opening a web browser and navigating to:

**http://localhost:8080**

You should see the WordPress setup page. Follow the on-screen instructions to complete the WordPress installation.

Step 5: **Clean Up**

To stop and remove the containers, you can run the following command in the same directory as your docker-compose.yml file:

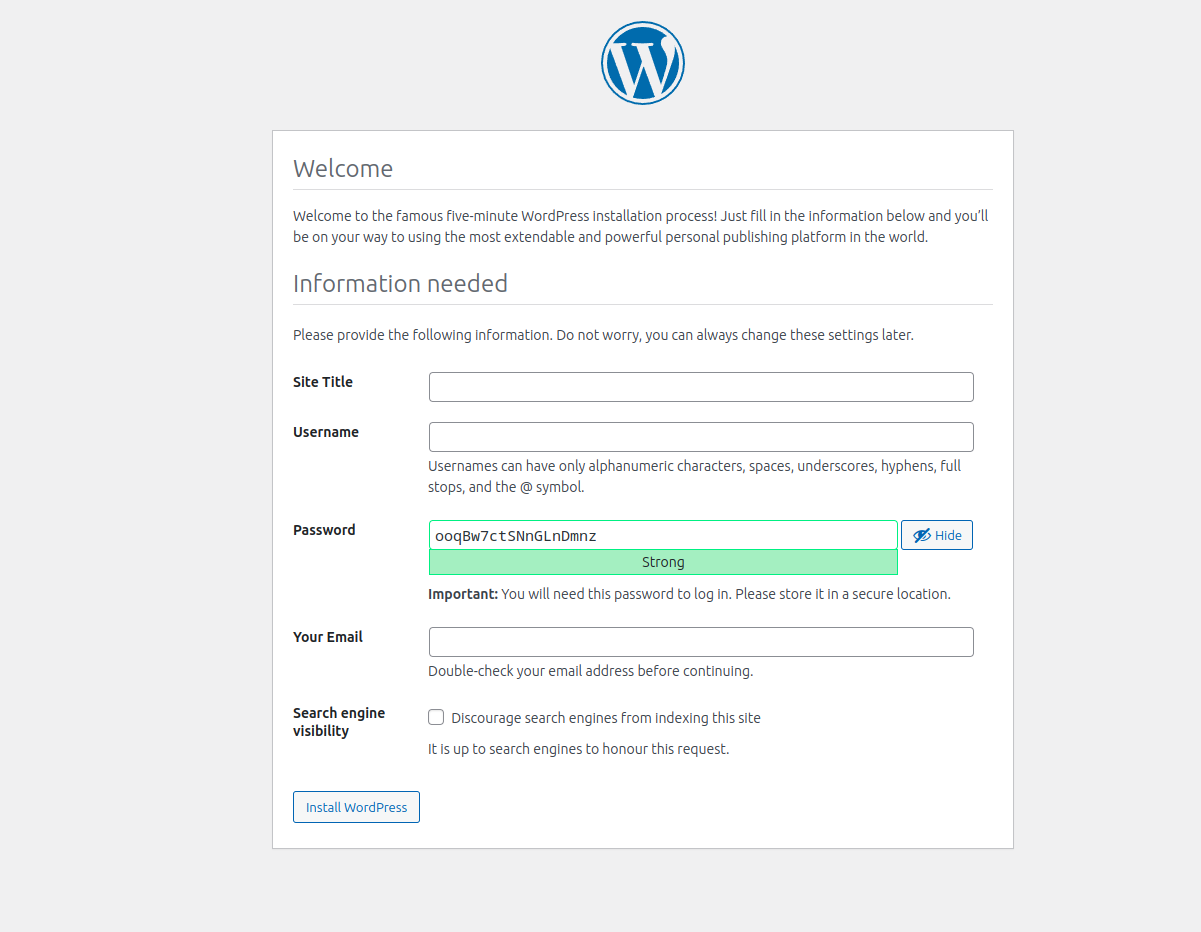
**docker-compose down**

This will stop and remove the containers, but it will keep the data volumes intact. If you want to remove the volumes as well, add the -v option:

**docker-compose down -v**

That's it! You have successfully deployed WordPress and MySQL containers using Docker Compose and accessed the WordPress frontend.

**Output:**

****

**Takeaway:**

| In this experiment, you utilized Docker Compose to orchestrate the deployment of a WordPress application and a MySQL database within containers. The docker-compose.yml file defined the services, their configurations, and established dependencies between them. Through the docker-compose up -d command, the containers were started in detached mode, with WordPress accessible at http://localhost:8080. You learned about configuring networks, volumes for data persistence, and encountered and resolved the "permission denied" error by adding the user to the docker group. This hands-on experience demonstrated the simplicity of managing multi-container applications and introduced fundamental concepts of containerization, emphasizing the importance of security considerations and data persistence in real-world scenarios. |
| --- |