Docker container:-

To uninstall docker which are pre install:-

sudo apt-get remove docker docker-engine docker.io

Update system by sudo apt-get install

Of some system command :- **sudo apt-get install docker-ce**

If this not work then:-

To get help command:- **docker install it will dive correct command**

To install docker- **sudo apt install docker.io**

For another packages:- Sudo snap install docker

To check weather the docker is install or not we pull docker image from docker hub by command

sudo docker run hello-world

To check it is pulled or not:- **sudo docker images**

It show docker images To see all pull images:-

**sudo docker ps -a**

### Experiment 28-

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

**Install the docker from above process.**

**Step 1**:- Create a python file containing all arithmetic operations. Name of file must be “**calculator.py**”

**Step 2**: - Now create the Docker file by name “**Dockerfile”.**

Add following content in it that file

# Dockerfile

**# Use an official Python runtime as a parent image FROM python:3.8**

# Set the working directory to /app WORKDIR /app

**# Copy the current directory contents into the container at /app COPY . /app**

# Install any needed packages specified in requirements.txt RUN pip install --no-cache-dir -r requirements.txt

**# Make port 80 available to the world outside this container EXPOSE 80**

# Define environment variable ENV NAME World

**# Run calculator.py when the container launches CMD ["python", "calculator.py"]**

**Step 3:-** Create a requirement file for some additional requirements for the python script use by us.

The name of file must be **“requirement.txt”**

**Step 4:-** Build the docker image by running the below command but condition is that all the file calculator,Docker,requirement must be in a single folder.

docker build -t calculator-app .

**Step 4:-** Now we run the docker image.

**docker run -it calculator-app**

**Experiment Number 29:-**

Run the Docker container with the created image .

**Step 1:-** There is no predefined container then create the container from above process (Experiment and create the docker image).

**Step 2:-** To run the docker image use command

“docker run -it calculator-app”

Use your own file name instead of a calculator.

**Experiment No 32**

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

**Step 1:-** Create the python file by name app.py

from flask import Flask app = Flask(\_\_name )

@app.route("/") def hello():

return "Hello, Docker!"

if name == "\_\_main ": app.run(host='0.0.0.0', port=80)

#### Step 2:-

**Create the docker file by name Dockerfile**

# Use an official Python runtime as a parent image FROM python:3.6

# Set the working directory to /app WORKDIR /app

# Copy the current directory contents into the container at /app COPY . /app

# Install any needed packages specified in requirements.txt RUN pip install -r requirements.txt

# Make port 80 available to the world outside this container EXPOSE 80

# Define environment variable ENV NAME World

# Run app.py when the container launches

CMD ["python", "app.py"]

**Step 4:-** Create the file by name **“requirements.txt”**

In file type **“flask”**

flask

**Sep 5:-** Run the command

“sudo docker build -t flask-app .”

**Step 6:-** Run the command to open on port 80:80

sudo docker run -p 80:80 flask-app

**In terminal of vs code it give one http:// address copy it and run it on browser Example http://172.17.0.2:80/**

# Experiment 33:-

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

**Repeat the above experiment again because it is done by flask only.**

**Experiment 34:-**

#### Run the docker container from recently created image and run that docker container to 5000 port of host system.

**Step 1:- Use the command to run**

**sudo docker build -t flask-app .**

**Step 2:- Ue this command to run on 5000**

# sudo docker run -p 5000:80 flask-app

**Then see in terminal the https:// link copy it and use it on the browser.**

# Experiment 35:-

**Exp19:-**

**Create two applications in two different docker containers. Push those applications and run to show the communications between two dockers.**

**Backend Application (Flask API)**

##### Create a directory for your project: mkdir docker\_communication\_demo cd docker\_communication\_demo

1. **Create a file named app.py for the Flask API:**

# app.py

from flask import Flask, jsonify app = Flask(\_\_name ) @app.route('/api/data')

def get\_data():

return jsonify({"message": "Hello from the backend!"}) if name == '\_\_main ':

app.run(debug=True, host='0.0.0.0', port=5000)

##### Create a Dockerfile for the backend:

# Dockerfile FROM python:3.8 WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt COPY . .

CMD ["python", "app.py"]

##### Create a requirements.txt file:

Flask>=2.1.5

Werkzeug>=2.0.2

1. **Build and run the backend Docker container:**

##### docker build -t backend-app .

**docker run -p 5000:5000 backend-app**

**Frontend Application (Flask Web App)**

* 1. **Create a directory for the frontend in** same directory in which backend directory are there:

##### mkdir frontend-app cd frontend-app

* 1. **Create a file named app.py for the Flask web app:**

# app.py

from flask import Flask, render\_template import requests

app = Flask(\_\_name )

backend\_url = "http://backend:5000" # This is the Docker service name @app.route('/')

def home():

response = requests.get(f"{backend\_url}/api/data")

data = response.json()

return render\_template('index.html', message=data['message']) if name == '\_\_main ':

app.run(debug=True, host='0.0.0.0', port=5001)

##### Create a Dockerfile for the frontend:

# Dockerfile FROM python:3.8 WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt COPY . .

CMD ["python", "app.py"]

* 1. **Create a requirements.txt file:**

**Flask>=2.1.5 requests==2.26.0**

* 1. **Create a directory named templates and add a file named index.html:**

<!-- templates/index.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Docker Communication Demo</title>

</head>

<body>

<h1>{{ message }}</h1>

</body>

</html>

* 1. **Build and run the frontend Docker container: docker build -t frontend-app .**

**docker run -p 5001:5001 --link backend frontend-app**

**Docker-compose File**

**Create a docker-compose.yml file in your project directory** along with frontend and backed directory **with the following content:**

version: '3' services:

backend: build:

context: ./docker\_communication\_demo ports:

- "5000:5000"

frontend: build:

context: ./frontend-app ports:

- "5001:5001"

depends\_on:

- backend

## Run command:-

**docker-compose up**

Exp :- 37

**Create a docker image of simple login form using Flask on port 7000. Step 1:-**

Create a file named app.py with the following code for a simple Flask login form:

# app.py

from flask import Flask, render\_template, request

app = Flask(\_\_name )

@app.route('/') def index():

return render\_template('login.html')

@app.route('/login', methods=['POST']) def login():

username = request.form['username'] password = request.form['password']

# Add your login logic here (for simplicity, we'll just print the credentials)

print(f"Username: {username}, Password: {password}")

return 'Login successful!'

if name == '\_\_main ':

app.run(debug=True, host='0.0.0.0', port=7000)

**Step 2:** Create HTML Template Create a folder named templates and inside it, create a file named login.html with the following content:

<!-- templates/login.html -->

<!DOCTYPE html>

<html>

<head>

<title>Login</title>

</head>

<body>

<h2>Login Form</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>

**Step 3:** Create a Dockerfile Create a file named Dockerfile in the same directory as your

app.py with the following content:

# Dockerfile FROM python:3.8

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt COPY . .

EXPOSE 7000

CMD ["python", "app.py"]

**Step 4:** Create a requirements.txt file in the same directory as your app.py Create a file named

requirements.txt with the following content:

Flask>=2.0.1 Werkzeug>=2.0.1

**Step 5:** Build the Docker Image Open a terminal, navigate to the directory containing your Dockerfile, app.py, templates, and requirements.txt files, and run the following command to build the Docker image:

docker build -t flask-login-app .

**Step 6:** Run the Docker Container After building the image, run the Docker container with the following command:

docker run -p 7000:7000 flask-login-app

Now, you should be able to access the simple login form at http://localhost:7000 in your web browser.

Exp :- 38

**Create a docker image of simple login form using django on port 6000.**

**Step 1: Create a Django Project**

# Create a directory for your project mkdir simple\_login\_django

**# Navigate to the project directory cd simple\_login\_django**

# Create a virtual environment (optional but recommended) python3 -m venv venv

**source venv/bin/activate # On Windows, use `venv\Scripts\activate`**

# Install Django pip install django

**# Create a Django project**

**django-admin startproject simplelogin**

#### Step 2: Create a Django App

# Navigate to the project directory cd simplelogin

**# Create a Django app**

**python manage.py startapp loginapp**

#### Step 3: Update loginapp/views.py

**Create a file named views.py inside the loginapp directory with the following content:**

from django.shortcuts import render from django.http import HttpResponse

def login(request):

return render(request, 'loginapp/login.html', {})

def success(request):

return HttpResponse("Successful Login!")

#### Step 4: Create loginapp/templates/loginapp/login.html

Create a templates directory inside the loginapp directory, and inside it, create a file named

**login.html with the following content:**

<!DOCTYPE html>

<html>

<head>

<title>Login Page</title>

</head>

<body>

<h2>Login</h2>

<form action="/success/" method="post">

{% csrf\_token %}

<label for="username">Username:</label>

<input type="text" name="username" id="username" required>

<br>

<label for="password">Password:</label>

<input type="password" name="password" id="password" required>

<br>

<input type="submit" value="Login">

</form>

</body>

</html>

#### Step 5: Update loginapp/urls.py

Create a file named urls.py inside the loginapp directory with the following content:

from django.urls import path from . import views

urlpatterns = [

path('login/', views.login, name='login'), path('success/', views.success, name='success'),

]

#### Step 6: Update simplelogin/urls.py

**Update the urls.py file inside the simplelogin directory with the following content:**

from django.contrib import admin

from django.urls import include, path

urlpatterns = [

path('admin/', admin.site.urls), path('', include('loginapp.urls')),

]

#### Update in the setting.py file with following content:-

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates', 'DIRS': [BASE\_DIR / 'loginapp' / 'templates'],

'APP\_DIRS': True, 'OPTIONS': {

'context\_processors': [ 'django.template.context\_processors.debug', 'django.template.context\_processors.request', 'django.contrib.auth.context\_processors.auth', 'django.contrib.messages.context\_processors.messages',

],

},

},

]

**Step 8: Dockerize the Application**

**Create a Dockerfile in the project root in the level of manage.py file with the following content:**

# Use an official Python runtime as a parent image FROM python:3.8-slim

# Set the working directory to /app WORKDIR /app

# Copy the current directory contents into the container at /app COPY . /app

# Install any needed packages specified in requirements.txt RUN pip install --no-cache-dir -r requirements.txt

# Make port 6000 available to the world outside this container EXPOSE 6000

# Define environment variable ENV NAME simplelogin

# Run app.py when the container launches

CMD ["python", "manage.py", "runserver", "0.0.0.0:6000"]

#### Step 9: Create a requirements.txt file

Create a file named requirements.txt in the project root in the level of manage.py file with the following content:

Django==3.2.5

#### Step 10: Build and Run the Docker Image

# Build the Docker image

**docker build -t simple-login-django .**

# Run the Docker container

**docker run -p 8000:6000 simple-login-django**

Open your web browser and navigate to http://localhost:8000/login/. You should see the login page. Enter any username and password, and you'll be redirected to the success page.

**Verify Directory Structure:**

Ensure that your directory structure is correct. The templates directory should be inside the

loginapp directory. simple\_login\_django/

├── loginapp/

│ ├── templates/

│ │ └── loginapp/

│ │ └── login.html

│ ├── init .py

│ ├── admin.py

│ ├── apps.py

│ ├── migrations/

│ ├── models.py

│ ├── tests.py

│ └── views.py

├── simplelogin/

│ ├── init .py

│ ├── asgi.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

├── venv/

└── Dockerfile

Exp:-39

**Create a container with ngnix web server and create one more container with mysql.**

#### Create the Nginx Container:

Step 1: Create an Nginx Dockerfile

Create a file named Dockerfile.nginx with the following content:

# Dockerfile.nginx FROM nginx:latest

# Copy custom Nginx configuration COPY nginx.conf /etc/nginx/nginx.conf # Expose port 80

EXPOSE 80

# Start Nginx

CMD ["nginx", "-g", "daemon off;"]

Step 2: Create an Nginx Configuration File

Create a file named nginx.conf with your custom Nginx configuration. For simplicity, you can start with a basic configuration:

# nginx.conf user nginx;

worker\_processes 1;

error\_log /var/log/nginx/error.log warn;

pid

/var/run/nginx.pid;

events {

worker\_connections 1024;

}

http {

include

/etc/nginx/mime.types;

default\_type application/octet-stream;

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request"' '$status $body\_bytes\_sent "$http\_referer" ' '"$http\_user\_agent" "$http\_x\_forwarded\_for"';

access\_log /var/log/nginx/access.log main;

sendfile

on;

keepalive\_timeout 65;

include /etc/nginx/conf.d/\*.conf;

}

**Step 3:** Build and Run the Nginx Container

docker build -t nginx-container -f Dockerfile.nginx .

**docker run -d -p 80:80 --name nginx-container nginx-container**

#### Create the MySQL Container:

Step 1: Create a MySQL Dockerfile

Create a file named Dockerfile.mysql with the following content:

# Dockerfile.mysql FROM mysql:latest

# Set environment variables

ENV MYSQL\_ROOT\_PASSWORD=root\_password \ MYSQL\_DATABASE=my\_database \ MYSQL\_USER=my\_user \ MYSQL\_PASSWORD=my\_password

# Expose port 3306 EXPOSE 3306

# Start MySQL

CMD ["mysqld"]

**Step 2:** Build and Run the MySQL Container

docker build -t mysql-container -f Dockerfile.mysql .

**docker run -d -p 3306:3306 --name mysql-container mysql-container**

#### Verify Containers

You can access the Nginx welcome page by visiting [http://localhost](http://localhost/) in your web browser.

#### View Running Containers

docker ps

**docker inspect mysql-container docker exec -it mysql-container bash**

mysql -u root -p

password= root\_password

**Exp:-40**

**Create a simple web form to insert the records in mysql data base.**

#### Step 1: Create a New Directory

**Create a new directory for your project. For example: mkdir lamp-web-form**

**cd lamp-web-form**

#### Step 2: Create Dockerfile for PHP and Apache

Create a file named Dockerfile in the project directory:

# Dockerfile

FROM php:7.4-apache

# Install MySQLi extension

RUN docker-php-ext-install mysqli COPY src/ /var/www/html/

EXPOSE 80

#### Step 3: Create the Source Directory

Create a directory named src in the project directory:

**mkdir src**

#### Step 4: Create PHP Script with Web Form

Inside the src directory, create a file named index.php with the following content:

<!-- src/index.php -->

<!DOCTYPE html>

<html>

<head>

<title>Simple PHP Web Form</title>

</head>

<body>

<h1>Web Form to Insert Records</h1>

<form action="insert.php" method="post">

<label for="name">Name:</label>

<input type="text" id="name" name="name" required><br>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required><br>

<input type="submit" value="Submit">

</form>

</body>

</html>

Inside the src directory, create a file named insert.php with the following content:

<!-- src/insert.php -->

<?php

$host = 'mysql';

$user = 'my\_user';

$password = 'my\_password';

$database = 'my\_database';

$conn = new mysqli($host, $user, $password, $database); if ($conn->connect\_error) {

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

}

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

$name = $\_POST["name"];

$email = $\_POST["email"];

$sql = "INSERT INTO users (name, email) VALUES ('$name', '$email')"; if ($conn->query($sql) === TRUE) {

echo "<p>New record inserted successfully</p>";

} else {

echo "Error: " . $sql . "<br>" . $conn->error;

}

}

$conn->close();

?>

#### Step 5: Create Dockerfile for MySQL

Create a file named Dockerfile.mysql in the project directory:

# Dockerfile.mysql FROM mysql:latest

# Set environment variables

ENV MYSQL\_ROOT\_PASSWORD=root\_password ENV MYSQL\_DATABASE=my\_database

ENV MYSQL\_USER=my\_user

ENV MYSQL\_PASSWORD=my\_password # Copy initialization SQL script

COPY init.sql /docker-entrypoint-initdb.d/ # Expose the MySQL port

EXPOSE 3306

Create a file named init.sql in the same directory as your Dockerfile.mysql with the following content:

-- init.sql

CREATE DATABASE IF NOT EXISTS my\_database;

USE my\_database;

CREATE TABLE IF NOT EXISTS users (

id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(255) NOT NULL,

email VARCHAR(255) NOT NULL

);

#### Step 6: Create Docker Compose File

Create a file named docker-compose.yml in the project directory:

version: '3'

services:

web:

build:

context: .

dockerfile: Dockerfile ports:

- "8080:80"

depends\_on:

- mysql environment:

MYSQL\_HOST: mysql MYSQL\_USER: root

MYSQL\_PASSWORD: root\_password MYSQL\_DATABASE: my\_database

mysql:

build:

context: .

dockerfile: Dockerfile.mysql ports:

- "3306:3306"

phpmyadmin:

image: phpmyadmin/phpmyadmin ports:

- "8081:80"

environment: PMA\_HOST: mysql PMA\_USER: root

PMA\_PASSWORD: root\_password

#### Step 7: Build and Run the Docker Containers

Run the following commands to build and run the Docker containers:

**docker-compose build docker-compose up -d**

Visit http://localhost:8080 in your web browser to access the web form

Visit http://localhost:8081 in your web browser to access the phpMyAdmin to see your data is inserted or not

Exp:-42

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

#### Step 1: Create the Java Application

Create a directory for your Java application and add a file named MyApp.java:

// MyApp.java

public class MyApp {

public static void main(String[] args) { System.out.println("Hello, Docker!");

}

}

#### Step 2: Create Dockerfile

In the same directory as your Java application, create a file named Dockerfile:

# Use the official Ubuntu base image FROM ubuntu:latest

# Install OpenJDK

RUN apt-get update && \

apt-get install -y openjdk-11-jdk # Set the working directory

WORKDIR /app

# Copy the Java application into the container COPY MyApp.java .

# Compile the Java application RUN javac MyApp.java

# Define the command to run the application

CMD ["java", "MyApp"]

#### Step 3: Build the Docker Image

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

#### Step 4: Run the Docker Container

**docker run my-java-app**

**43. Run a LAMP Stack Container at port 8080 and host media wiki site on native machine.**

**1] create docker-compose file:-**

# MediaWiki with MySQL version: '3'

services:

mediawiki:

image: mediawiki:1.38 restart: always networks:

* docker\_network ports:

- 8080:80

# volumes:

# - ./LocalSettings.php:/var/www/html/LocalSettings.php

# After initial setup, download LocalSettings.php to the same directory as # this yaml and uncomment the following line and use compose to restart

# the mediawiki service database:

image: mysql:8.0.29 restart: always networks:

* docker\_network

environment:

MYSQL\_DATABASE: wiki\_db

MYSQL\_ROOT\_PASSWORD: root

MYSQL\_USER: wikimedia

MYSQL\_PASSWORD: wikimedia

volumes:

- /var/lib/mysql

# phpmyadmin

phpmyadmin:

depends\_on:

- database

image: phpmyadmin/phpmyadmin

restart: always

ports:

- '8000:80'

environment:

PMA\_HOST: database

MYSQL\_ROOT\_PASSWORD: root

UPLOAD\_LIMIT: 64M

networks:

- docker\_network

networks:

docker\_network:

driver: bridge

1. Follow the installation instruction While database configuration Change database host:-

**Localhost to database Change database name:- wiki\_db**

Password:- root

When we install docker, docker info command will not run and give the following error. So to fix it write the command:

ERROR: permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/info": dial unix

/var/run/docker.sock: connect: permission denied

**Command:** sudo chmod 666 /var/run/docker.sock

If docker-compose not found: Then run following command:

sudo curl -L https://github.com/docker/compose/releases/download/1.21.0/docker- compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

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

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

Docker Compose file to be written for this experiment:

version: '3'

services:

# Database services for wordpress we use mysql mysql\_db:

container\_name: mysql\_container image: mysql:8.2

restart: always environment:

MYSQL\_ROOT\_PASSWORD: it

MYSQL\_DATABASE: wordpress\_db MYSQL\_USER: Ankur MYSQL\_PASSWORD: Ankur2003@

volumes:

* mysql:/var/lib/mysql wordpress:

depends\_on:

* mysql\_db

image: wordpress:latest restart: always

ports:

- "8080:80"

environment:

WORDPRESS\_DB\_HOST: mysql\_db:3306 WORDPRESS\_DB\_USER: Ankur

WORDPRESS\_DB\_PASSWORD: Ankur2003@

WORDPRESS\_DB\_NAME: wordpress\_db volumes:

- "./:/var/www/html"

volumes: mysql: {}

Command: sudo docker-compose up To Open Website: localhost

* 1. Create a simple Hello-world python flask application and create the docker image of that Flask application. Run application on port 5000.

Make A directory Named: Docker\_Flask.

Make 3 files: app.py, Dockerfile, requirements.txt

app.py:

from flask import Flask

import os

app = Flask(\_\_name ) @app.route("/")

def hello():

return "Flask inside Docker!!"

if name == "\_\_main ":

port = int(os.environ.get("PORT", 5000)) app.run(debug=True,host='0.0.0.0',port=port)

requirements.txt

flask

Dockerfile:

FROM python:3.6 COPY . /app WORKDIR /app

RUN pip install -r requirements.txt ENTRYPOINT ["python"]

CMD ["app.py"]

Then Run two commands:

docker build -t simple-flask-app:latest .

docker run -d -p 5000:5000 simple-flask-app Link: localhost:5000

21. **Pull the LAMP Stack container from docker hub and host a web application of your own. Push that image back to repository. Make use of database.**

<https://github.com/raptor-2001/php-dockerized-form>

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

Create a Directory: Python-App. A file app.py with content:

print("Hello World")

Change the program with arithmetic operations program. Dockerfile:

FROM python:3.9 WORKDIR /app COPY . /app EXPOSE 80

ENV NAME World

CMD ["python", "app.py"]

Build the image:

docker build -t my-python-app . docker run -p 4000:80 my-python-app

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

In this experiment, we need to perform load balancing between two applications/image,

{wordpress, flask application}.

Directory Structure:

Root Directory:

flask\_app

app.py Dockerfile requirements.txt

docker-compose.yaml nginx.conf

**Command to make application work:** docker-compose up localhost:5000 -> Flask Application

localhost:8080 -> Wordpress Application

localhost -> Any application depending upon balancer.

Create Flask Application similar to that of experiment 24. Content of Dockerfile:

events {}

http {

upstream backend { server flask:5000; server wordpress:80;

}

server {

listen 80;

location / {

proxy\_pass http://backend; 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;

}

}

}

**Content of docker-compose.yaml:**

version: '3'

services:

nginx:

image: nginx:latest ports:

- "80:80"

volumes:

* ./nginx.conf:/etc/nginx/nginx.conf depends\_on:
* flask
* wordpress

flask:

build:

context: ./flask\_app ports:

- "5000:5000"

mysql\_db:

container\_name: mysql\_container image: mysql:8.2

restart: always environment:

MYSQL\_ROOT\_PASSWORD: it

MYSQL\_DATABASE: wordpress\_db MYSQL\_USER: Ankur MYSQL\_PASSWORD: Ankur2003@

volumes:

* mysql:/var/lib/mysql wordpress:

depends\_on:

* mysql\_db

image: wordpress:latest restart: always

ports:

- "8080:80"

environment:

WORDPRESS\_DB\_HOST: mysql\_db:3306 WORDPRESS\_DB\_USER: Ankur WORDPRESS\_DB\_PASSWORD: Ankur2003@

WORDPRESS\_DB\_NAME: wordpress\_db volumes:

* "./:/var/www/html"

volumes: mysql: {}

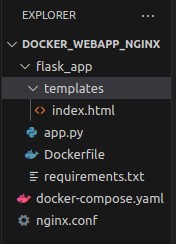
Command to run this experiment:

docker-compose up -d

Write localhost to access site.

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

We need to build a flask application with login details and using nginx web server, we need to create image.

Directory Structure:

**Content of flask application similar to that of experiment 24. Some changes in app.py and templates folder is added.**

app.py:

from flask import Flask, render\_template import os

app = Flask(\_\_name )

@app.route('/') def index():

return render\_template('index.html')

if name == '\_\_main ':

port = int(os.environ.get("PORT", 5000)) app.run(debug=True,host='0.0.0.0',port=port)

**Index.html:**

<!-- templates/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>

nginx.conf file:

events {}

http {

upstream backend { server flask:5000;

}

server {

listen 80;

location / {

proxy\_pass http://backend; 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;

}

}

}

**docker-compose.yaml:**

version: '3'

services: nginx:

image: nginx:latest ports:

- "80:80"

volumes:

* ./nginx.conf:/etc/nginx/nginx.conf depends\_on:
* flask

flask:

build:

context: ./flask\_app ports:

- "5000:5000"

Experiment 19: Docker communication

1. Create a back-end server in flask or NodeJs or any other :

const http=require("http"); const hostname="0.0.0.0"; const port=3000;

const server=http.createServer((req,res)=>{ console.log("Request of "+req.url,+" by "+req.method); res.end("Hello User from ProjectI");

})

server.listen(port,hostname,()=>{

console.log(`Server listening at +${hostname}:${port}`);

})

1. Dockerize it using Dockerfile:

# Use an official Node.js image as a base image FROM node:latest

# Set the working directory in the container WORKDIR /usr/src/app

# Copy the application code to the container COPY . .

# Install npm dependencies RUN npm install

# Specify the command to run on container start CMD ["npm", "start"]

EXPOSE 3000

1. After dockerizing it :

**use below command to get container id:**

docker ps -a

use below command to get container ip\_adress:

docker inspect ${container\_id}

1. Create another container in which image running should be alpine.

Use wget -qO- 172.17.0.2:3000

If message is displayed then assignment is completed .

Experiment No. 20 Youtrack :

In this experiment, we need to get a open source code and make a docker file of it and run it. <https://github.com/uniplug/youtrack-docker.git>

The above is the link I have used to take reference of open source code i.e youtrack. From the above link you will get the docker image, u need to run the docker image .

On running a container, we get a token on the terminal and that token is used on running

container IP address and our Youtrack is configured and ready to be used . e.g. 172.17.0.2

##### Docker Installation

* 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. sudo apt-get update
  3. sudo apt-get install ca-certificates curl gnupg
  4. sudo install -m 0755 -d /etc/apt/keyrings
  5. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o

/etc/apt/keyrings/docker.gpg

* 1. sudo chmod a+r /etc/apt/keyrings/docker.gpg
  2. echo \

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

$(. /etc/os-release && echo "$VERSION\_CODENAME") stable" | \ sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

* 1. sudo apt-get update
  2. 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.

* 1. sudo docker run hello-world

##### Create Docker image:

1. Go the the directory where the application files exists.
2. Create docker file gedit Dockerfile
3. Write this in Dockerfile FROM ubuntu

COPY . .

CMD ["echo","Hello,there"]

It means that the ubuntu is used as a VM to create the docker image.

COPY . . means it copies the entire content of the current directory to the current directory of the docker images. So, this copies everything present in our directory to the VM

CMD is used to give any instructions in the docker file

1. To build Docker image

Sudo docker build -t mydocker .

1. To run docker image Sudo docker run mydocker
2. To see all docker images Sudo docker images
3. To open terminal of the docker image Sudo docker run -it mydocker bash

It means interactive mode

##### If we want to build our dockerfile on a base images eg. nginx

* 1. Write this in dockerfile

It shows the location in the images where to copy

* 1. Then build Docker image

Sudo docker build -t mydocker .

* 1. Run docker container from docker image Sudo docker run -d -p 8080:80 mydocker

8080 is the host machine’s port from where we can access our application using web browser. It maps port 8080 in host to port 80 in container.

* 1. We can access our application from the web browser localhost:8080

And download our file by localhost:8080/1.py

**Docker Experiment example**

**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.**

1. Create HTML Form
2. Create docker file FROM httpd:latest

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

1. Build docker image

docker build -t simple-web-app .

1. Run docker container

docker run -d -p 80:80 simple-web-app

1. Access the form from localhost:80

#### FTP

1. sudo apt update
2. sudo apt install vsftpd
3. Sudo service vsftpd status
4. Sudo nano /etc/vsftpd.conf Uncomment write\_enable=YES ADD

user\_sub\_token=$USER local\_root=/home/$USER/ftp pasv\_min\_port=10000 pasv\_max\_port=10100 userlist\_enable=YES userlist\_file=/etc/vsftpd.userlist userlist\_deny=NO

1. sudo ufw allow from any to any port 20,21,10000:10100 proto tcp
2. sudo adduser ftpuser1 password : abcd
3. sudo mkdir /home/ftpuser1/ftp
4. sudo chown nobody:nogroup /home/ftpuser1/ftp
5. sudo chmod a-w /home/ftpuser1/ftp
6. sudo mkdir /home/ftpuser1/ftp/upload
7. sudo chown ftpuser1:ftpuser1 /home/ftpuser1/ftp/upload
8. echo "My FTP Server" | sudo tee /home/ftpuser1/ftp/upload/demo.txt
9. sudo ls -la /home/ftpuser1/ftp
10. echo "Adwait" | sudo tee -a /etc/vsftpd.userlist
11. sudo systemctl restart vsftpd
12. ifconfig

Take first ip address (inet ke baju ka)

1. Go to Other Locations on the PC and write ftp://your-ip-address
2. Login from the created ftpuser1
3. We see the files of the ftpuser1

#### 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

1. If is not active/running

sudo systemctl start xinetd.service

1. 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

}

1. Then save and close the file and restart xinetd.service as follows: sudo systemctl restart xinetd.service
2. 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

1. Note the ip address ->10.10.13.226 //in my case
2. 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

#### NFS

Letters a,b,c,d tell us the order of executing steps

Open 2 separate terminals for client and server

#### A Server

1. sudo apt update
2. sudo apt install nfs-kernel-server 3.sudo mkdir -p /mnt/nfs\_share

4.sudo chown -R nobody:nogroup /mnt/nfs\_share/ 5.sudo chmod 777 /mnt/nfs\_share/

6.sudo nano /etc/exports 7.sudo exportfs -a

1. sudo systemctl restart nfs-kernel-server
2. sudo ufw allow from 10.10.13.133 to any port nfs 10.udo ufw enable

11.sudo ufw status

#### B client

1. sudo apt update
2. sudo apt install nfs-common 3.sudo mkdir -p /mnt/nfs\_clientshare

4.sudo mount 10.10.13.133:/mnt/nfs\_share /mnt/nfs\_clientshare

#### C server

1. cd /mnt/nfs\_share
2. touch file1.txt file2.txt file3.txt

#### D client

1. ls -l /mnt/nfs\_clientshare/

**SVN**

###### SVN stands for Subversion. It is an open-source centralized version control system written in Java, licensed under Apache. Software developers use Subversion to maintain current and historical versions of files such as source code.

**Step 1: Install Apache2**

###### sudo apt update

* + sudo apt install apache2 apache2-utils

**We have installed Apache2 now let’s start and enable it.**

###### sudo systemctl start apache2.service

* + sudo systemctl enable apache2.service

##### We have successfully set-up and enable the HTTP web server. Let’s install SVN now.

**Step 2: Install SVN**

###### sudo apt-get install subversion libapache2-mod-svn subversion-tools libsvn-dev

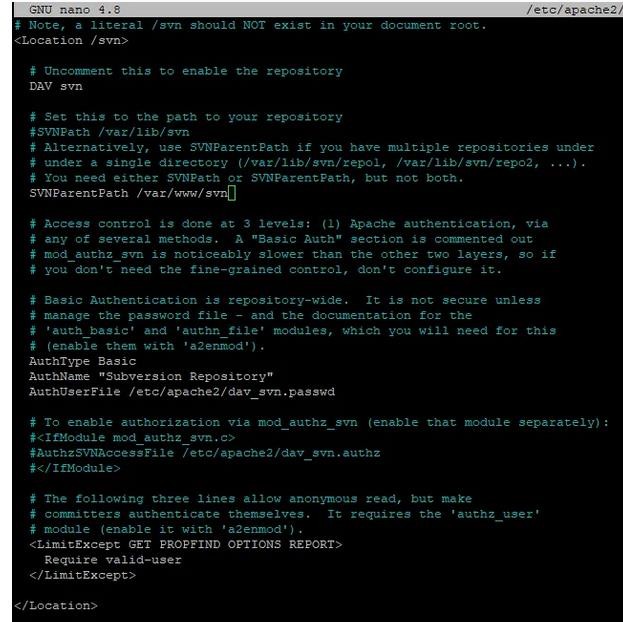
SVN and all dependencies are installed. Now enable Apache2 modules to run SVN to function.

* + sudo a2enmod dav
  + sudo a2enmod dav\_svn
  + sudo service apache2 restart

##### Step 3: Configure Apache2 with SVN

* + sudo nano /etc/apache2/mods-enabled/dav\_svn.conf

Make mentioned Changes/un-comment lines in the file.



**Let’s Create Repository Now**

* + sudo mkdir /var/www/svn
  + sudo svnadmin create /var/www/svn/project
  + sudo chown -R www-data:www-data /var/www/svn
  + sudo chmod -R 775 /var/www/svn

##### Step 4: Create SVN User Accounts

Use the below command to create a new SVN user(admin).

* + sudo htpasswd -cm /etc/apache2/dav\_svn.passwd admi**n** If you wish to create more users then use the below command
  + sudo htpasswd -m /etc/apache2/dav\_svn.passwd awais

We have successfully Installed and configure SVN let’s restart the Apache2 server and Test it. Restart Apache2 server with the below command.

* + sudo systemctl restart apache2.service

**Let’s Test It**

Open your browser and write the following in your URL bar.

* + localhost/svn/project

{to remove anything from www folder

**sudo rm -R** /var/www/wordpress/wp-content/themes/myFolder/\*

-R to recursively remove anything inside it (and deeper).

This also removes files (not just directories). ex.sudo rm -R /var/www/svn/project

}

**Debian Package**

[https://karthikkalyanaraman.medium.com/creating-debian-packages-](https://karthikkalyanaraman.medium.com/creating-debian-packages-cmake-e519a0186e87) [cmake-e519a0186e87](https://karthikkalyanaraman.medium.com/creating-debian-packages-cmake-e519a0186e87)

Note: build the directory structure with extreme care



#### after double clicking extract the data.tar.gz file and u’ll be able to see home folder inside that folder there will bw app

**Wordpress**