**RUNNING A DOCKER CONTAINER NGNIX**

**3. Pull the Nginx Docker Image**

Get the official Nginx image from Docker Hub:

bash

Copy code

sudo docker pull nginx

**4. Run the Nginx Container**

Now you can start a new Nginx container. Use this command:

bash

Copy code

sudo docker run --name mynginx -p 80:80 -d nginx

Explanation of the command:

* --name mynginx: Names the container "mynginx" (you can choose any name you like).
* -p 80:80: Maps port 80 on your host machine to port 80 in the container.
* -d: Runs the container in detached mode (in the background).
* nginx: Specifies the Docker image to use.

**5. Verify the Container is Running**

Check if the Nginx container is running:

bash

Copy code

sudo docker ps

You should see the container listed with the name "mynginx."

**6. Access Nginx**

Open your web browser and navigate to http://localhost. You should see the default Nginx welcome page.

run a web server Apache in docker containers to serve static content or proxy request to other services

**3. Pull the Apache Docker Image**

bash

Copy code

sudo docker pull httpd

**4. Run the Apache Container**

bash

Copy code

sudo docker run --name myapache -p 80:80 -d httpd

**5. Serve Static Content**

bash

Copy code

sudo docker run --name myapache -p 80:80 -v /path/to/your/static/files:/usr/local/apache2/htdocs -d httpd

**6. Proxy Requests to Other Services**

1. Create httpd.conf with proxy settings.
2. Save the file.
3. Run Apache with the custom configuration:

bash

Copy code

sudo docker run --name myapache -p 80:80 -v /path/to/your/httpd.conf:/usr/local/apache2/conf/httpd.conf -d httpd

**7. Verify Apache is Running**

bash

Copy code

sudo docker ps

**8. Access Your Apache Server**

Navigate to http://localhost in your web browser.

**9. Manage the Container**

* **Stop the container**:

bash

Copy code

sudo docker stop myapache

* **Start the container**:

bash

Copy code

sudo docker start myapache

* **Remove the container**:

bash

Copy code

sudo docker rm myapache

* **View logs**:

bash

Copy code

sudo docker logs myapache

docker container set of file manipulation with database connection

**1. Start Docker**

Make sure Docker is running on your system. You can start it with:

bash

Copy code

sudo service docker start

**2. Create and Run Containers**

* **Run a MySQL Container**

This command creates a MySQL container with a root password of 1234:

bash

Copy code

docker run -d --name mysql-container -e MYSQL\_ROOT\_PASSWORD=1234 mysql

* **Run an HTTP Server Container**

This command creates a simple HTTP server container:

bash

Copy code

docker run -d --name webserver httpd

**3. Check Container Details**

To see if your containers are running:

bash

Copy code

docker ps

To inspect network details:

bash

Copy code

docker network inspect bridge

**4. Access and Manipulate Files in a Container**

* **Create and Edit a File in a Container**

Start an Ubuntu container:

bash

Copy code

docker run -it --name ubuntu-container ubuntu

Install necessary tools and create a file:

bash

Copy code

apt-get update

apt-get install -y vim

vim /test.txt

Add content to /test.txt and save it.

* **Stop and Start the Container**

bash

Copy code

docker stop ubuntu-container

docker start ubuntu-container

* **Copy Files Between Container and Local System**
  + **Copy from Container to Local System:**

bash

Copy code

docker cp ubuntu-container:/test.txt .

* + **Create and Edit a Local File:**

bash

Copy code

vim test.txt

* + **Copy from Local System to Container:**

bash

Copy code

docker cp test.txt ubuntu-container:/test.txt

**5. Connect to the MySQL Database**

* **Install MySQL Client in the Webserver Container**

bash

Copy code

docker exec -it webserver /bin/bash

apt-get update

apt-get install -y mysql-client

* **Connect to MySQL**

Use the MySQL client to connect to the MySQL container:

bash

Copy code

mysql -h 172.17.0.2 -u root -p1234

Replace 172.17.0.2 with the IP address of your MySQL container if needed.

**Summary**

1. **Start Docker**.
2. **Run containers** for MySQL and an HTTP server.
3. **Check container status** and network settings.
4. **Access containers** to create/edit files and copy files between containers and your local system.
5. **Connect to the MySQL database** from another container.

docker compose in a single container we can create multiple files

**Create a Project Directory**

Create a directory for your Docker Compose project:

bash

Copy code

mkdir my-docker-project

cd my-docker-project

**3. Create a docker-compose.yml File**

Inside your project directory, create a docker-compose.yml file. This file will define your services, in this case, a single container with multiple files. Here’s an example configuration:

yaml

Copy code

version: '3.8'

services:

app:

image: ubuntu

container\_name: my\_app

volumes:

- ./files:/data

command: sleep infinity

This configuration does the following:

* Uses the ubuntu image.
* Names the container my\_app.
* Mounts a local directory ./files to the container’s /data directory.
* Keeps the container running indefinitely with sleep infinity (you can replace this with your actual command if needed).

**4. Create the Directory for Files**

Create a files directory inside your project directory. This directory will hold the files you want to use inside the container.

bash

Copy code

mkdir files

**5. Add Files to the files Directory**

Place any files you need into the files directory. For example, you might have:

bash

Copy code

echo "This is file1" > files/file1.txt

echo "This is file2" > files/file2.txt

**6. Start the Docker Compose Setup**

Use Docker Compose to start the container:

bash

Copy code

docker-compose up -d

The -d flag runs the container in detached mode.

**7. Verify the Setup**

Check that your container is running:

bash

Copy code

docker ps

You can also access the container’s shell to verify that the files are in place:

bash

Copy code

docker exec -it my\_app /bin/bash

Inside the container, navigate to /data to see the files:

bash

Copy code

cd /data

ls

cat file1.txt

**8. Stop and Remove the Container**

To stop and remove the container, use:

bash

Copy code

docker-compose down

setting up and using sitebroker for Website p\*\*\*\*\*\*\*\*\*\* testing

 **Clone the SiteBroker Repository**

bash

Copy code

git clone https://github.com/exploiter/sitebroker.git

 **Install Python and Required Modules**

* Install Python and pip:

bash

Copy code

sudo apt install python3 python3-pip

* Install the required Python modules:

bash

Copy code

pip3 install -r sitebroker/requirements.txt

 **Build and Run the Docker Image**

* Navigate to the sitebroker directory:

bash

Copy code

cd sitebroker

* Build the Docker image:

bash

Copy code

docker build -t sitebroker .

* Run the Docker container:

bash

Copy code

docker run -it --name sitebroker sitebroker

 **Using SiteBroker to Scan a Website**

* Enter the desired website URL when prompted, for example, github.com.
* Choose the required options for the scan based on what you want to do with the given website.
* Type your input and press Enter.

 **Access the SiteBroker Interface**

* You can access the SiteBroker interface via:

bash

Copy code

<http://localhost:8080/docker.html>

setting up a URL shortener application using docker

**1. Start Docker Daemon**

Ensure Docker is running:

bash

Copy code

sudo service docker start

**2. Create Project Directory**

Create and navigate to your project directory:

bash

Copy code

mkdir project-url-shortener

cd project-url-shortener

**3. Create Dockerfile**

Create a Dockerfile to define your application’s Docker image:

bash

Copy code

nano Dockerfile

Add the following content to Dockerfile:

dockerfile

Copy code

FROM python:3.8-slim

WORKDIR /app

COPY requirements.txt requirements.txt

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

RUN pip install gunicorn

EXPOSE 5000

CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]

**4. Create requirements.txt**

Create a requirements.txt file to list your Python dependencies:

bash

Copy code

nano requirements.txt

Add the following content to requirements.txt:

Copy code

flask

**5. Create app.py**

Create an app.py file for your Flask application:

bash

Copy code

nano app.py

Add the following content to app.py:

python

Copy code

from flask import Flask, request, redirect, render\_template

import hashlib

app = Flask(\_\_name\_\_)

url\_map = {}

@app.route('/', methods=['GET', 'POST'])

def index():

if request.method == 'POST':

original\_url = request.form['url']

short\_hash = hashlib.md5(original\_url.encode()).hexdigest()[:6]

url\_map[short\_hash] = original\_url

short\_url = request.host\_url + short\_hash

return render\_template('index.html', short\_url=short\_url)

return render\_template('index.html')

@app.route('/<short\_hash>')

def redirect\_to\_url(short\_hash):

original\_url = url\_map.get(short\_hash)

if original\_url:

return redirect(original\_url)

return 'URL not found', 404

if \_\_name\_\_ == '\_\_main\_\_':

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

**6. Create templates Directory and index.html**

Create a templates directory and an index.html file for your HTML template:

bash

Copy code

mkdir templates

nano templates/index.html

Add the following content to index.html:

html

Copy code

<!DOCTYPE html>

<html lang='en'>

<head>

<meta charset='UTF-8'>

<title>URL Shortener</title>

</head>

<body>

<h1>URL Shortener</h1>

<form method="POST">

<label for='url'>Enter URL: </label>

<input type='url' id='url' name='url' required>

<button type='submit'>Shorten</button>

</form>

{% if short\_url %}

<p>Shortened URL: <a href="{{ short\_url }}">{{ short\_url }}</a></p>

{% endif %}

</body>

</html>

**7. Build and Run the Docker Container**

Build the Docker image:

bash

Copy code

docker build -t url-shortener .

Run the Docker container:

bash

Copy code

docker run -d -p 5000:5000 --name url-shortener url-shortener

**Summary**

1. **Start Docker Daemon**: Ensure Docker is running.
2. **Create Project Directory**: Set up your project structure.
3. **Create Dockerfile**: Define your Docker image.
4. **Create requirements.txt**: List dependencies.
5. **Create app.py**: Define your Flask application.
6. **Create templates Directory and index.html**: Create the HTML template.
7. **Build and Run the Docker Container**: Build the image and run the container.

Now, you should be able to access your URL shortener application by navigating to http://localhost:5000 in your web browser.

setting of inter container communication using a docker bridge network on Ubuntu

**Create a Docker Network**

Create a custom Docker network that containers can use to communicate with each other:

bash

Copy code

sudo docker network create file-sharing

**2. Create a Docker Volume**

Create a Docker volume to share data between containers:

bash

Copy code

sudo docker volume create shared-data

**3. Run Containers Connected to the Network and Volume**

Run four containers connected to the file-sharing network and using the shared-data volume. Here’s how to do it:

**Container 1**

bash

Copy code

sudo docker run -d --name container1 --network file-sharing -v shared-data:/data alpine sh -c "while true; do sleep 3600; done"

**Container 2**

bash

Copy code

sudo docker run -d --name container2 --network file-sharing -v shared-data:/data alpine sh -c "while true; do sleep 3600; done"

**Container 3**

bash

Copy code

sudo docker run -d --name container3 --network file-sharing -v shared-data:/data alpine sh -c "while true; do sleep 3600; done"

**Container 4**

bash

Copy code

sudo docker run -d --name container4 --network file-sharing -v shared-data:/data alpine sh -c "while true; do sleep 3600; done"

**4. Create and Access Files in the Shared Volume**

**Create a File in the Shared Volume from Container 1**

Execute a command in container1 to create a file in the shared volume:

bash

Copy code

sudo docker exec container1 sh -c "echo 'Hello from container1' > /data/file-from-container1.txt"

**Access the File from Container 2**

Verify that the file created in container1 is accessible from container2:

bash

Copy code

sudo docker exec container2 cat /data/file-from-container1.txt

You should see the content "Hello from container1" displayed.

**Summary**

1. **Create a Docker Network**: Set up a custom network for container communication.
2. **Create a Docker Volume**: Create a volume for shared data.
3. **Run Containers**: Start containers connected to the network and volume.
4. **Create and Access Files**: Create a file in one container and access it from another.

This setup allows you to share data between containers using Docker volumes and ensure they can communicate over a custom network. If you need further details or encounter any issues, feel free to ask!

**wordpress**

**Create a Project Directory**

1. Open a terminal.
2. Create a new directory for your Docker project:

bash

Copy code

mkdir wordpress-docker

cd wordpress-docker

**3. Create the Docker Compose File**

1. In your project directory, create a docker-compose.yml file:

bash

Copy code

nano docker-compose.yml

(You can use any text editor you prefer; nano is just an example.)

1. Copy and paste the following content into the docker-compose.yml file:

yaml

Copy code

version: '3.8'

services:

db:

image: mysql:5.7

container\_name: mysql-db

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: wordpress

MYSQL\_DATABASE: wordpress

MYSQL\_USER: wordpress

MYSQL\_PASSWORD: wordpress

volumes:

- db-data:/var/lib/mysql

ports:

- "3306:3306"

wordpress:

image: wordpress:latest

container\_name: wordpress

restart: always

ports:

- "80:80"

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: wordpress

WORDPRESS\_DB\_PASSWORD: wordpress

WORDPRESS\_DB\_NAME: wordpress

volumes:

- wp-data:/var/www/html

depends\_on:

- db

volumes:

db-data:

wp-data:

1. Save and exit the file. (In nano, you can save by pressing CTRL+O, then Enter, and exit by pressing CTRL+X.)

**4. Start the Containers**

1. In your terminal, ensure you're in the project directory containing docker-compose.yml.
2. Run the following command to start the services:

bash

Copy code

sudo docker-compose up -d

The -d flag runs the containers in detached mode.

**5. Verify the Containers Are Running**

1. Check the status of your containers:

bash

Copy code

sudo docker-compose ps

You should see both wordpress and mysql-db containers listed.

**6. Access WordPress**

1. Open a web browser and navigate to http://localhost (or http://<your-server-ip> if you're using a remote server).
2. You should see the WordPress setup page. Follow the instructions to complete the installation.

**7. Stop and Remove Containers**

When you want to stop and remove the containers, you can use:

bash

Copy code

sudo docker-compose down

**8. View Logs**

To view logs for debugging or monitoring, use:

bash

Copy code

sudo docker-compose logs

THREATMAPPER CONTHINER SENSORS INSIDE ON UBUNTU A DOCKER (13)

**Create the Dockerfile**

First, create a Dockerfile that sets up ThreatMapper in an Ubuntu-based Docker container.

1. **Create a Project Directory**:

bash

Copy code

mkdir threatmapper-docker

cd threatmapper-docker

1. **Create the Dockerfile**:

bash

Copy code

nano Dockerfile

1. **Add the Following Content to Dockerfile**:

Dockerfile

Copy code

# Use the latest Ubuntu image

FROM ubuntu:latest

# Update package list and install curl

RUN apt-get update && \

apt-get install -y curl

# Install ThreatMapper

RUN curl -fsSL https://downloads.deepfence.io/deployment/scripts/install.sh | bash

# Set the default command to run ThreatMapper

ENTRYPOINT ["threatmapper-agent"]

Save and exit the file (in nano, use CTRL+O, then Enter, and CTRL+X to exit).

**2. Build the Docker Image**

Build the Docker image using the Dockerfile.

bash

Copy code

sudo docker build -t threatmapper-ubuntu .

**3. Run the ThreatMapper Container**

To run the ThreatMapper container, you need to provide your API key and console URL.

1. **Run the Container**:

bash

Copy code

sudo docker run -d \

--name threatmapper-agent \

-e MYMI\_CONSOLE\_URL=<your\_console\_url> \

-e MYMI\_API\_KEY=<your\_api\_key> \

threatmapper-ubuntu

Replace <your\_console\_url> with your ThreatMapper console URL and <your\_api\_key> with your API key.

**4. Verify the Docker Container**

To check if your ThreatMapper container is running:

1. **List Running Containers**:

bash

Copy code

sudo docker ps

1. **List All Containers (Including Stopped Ones)**:

bash

Copy code

sudo docker ps -a

1. **View Logs (if needed)**:

bash

Copy code

sudo docker logs threatmapper-agent

**5. Clean Up**

To stop and remove the ThreatMapper container and Docker image:

1. **Stop and Remove the Container**:

bash

Copy code

sudo docker stop threatmapper-agent

sudo docker rm threatmapper-agent

1. **Remove the Docker Image**:

bash

Copy code

sudo docker rmi threatmapper-ubuntu

**Start Docker Service**

Ensure Docker is running:

bash

Copy code

sudo service docker start

**setting up a Firewall within a docker container and control network traffic**

**2. Create and Run a Docker Container**

Create and start a Docker container with Ubuntu:

bash

Copy code

sudo docker run -it --name my-container ubuntu:latest

**3. Install Firewall Tools Inside the Container**

Once inside the container, update the package list and install firewall tools (iptables and ufw):

bash

Copy code

apt update

apt install -y iptables ufw

**4. Configure Firewall Rules Inside the Container**

1. **List Current Rules**:

bash

Copy code

iptables -L

1. **Add Rules**:

Allow TCP traffic on port 80:

bash

Copy code

iptables -A INPUT -p tcp --dport 80 -j ACCEPT

Drop all other incoming traffic (make sure this is configured carefully based on your needs):

bash

Copy code

iptables -A INPUT -j DROP

1. **Save Firewall Rules**:

You need to save the rules if you want them to persist after a restart. For iptables:

bash

Copy code

iptables-save > /etc/iptables/rules.v4

1. **Configure ufw (Uncomplicated Firewall)**:
   * Enable ufw:

bash

Copy code

ufw enable

* + Allow port 80/tcp:

bash

Copy code

ufw allow 80/tcp

* + Set default policies:

bash

Copy code

ufw default deny incoming

ufw default allow outgoing

* + Check the status:

bash

Copy code

ufw status

**5. Create a firewall.sh Script**

Create a script to automate firewall configuration:

1. **Create the Script**:

bash

Copy code

nano /usr/local/bin/firewall.sh

1. **Add the Following Content to firewall.sh**:

bash

Copy code

#!/bin/bash

iptables -A INPUT -p tcp --dport 80 -j ACCEPT

iptables -A INPUT -j DROP

ufw enable

ufw allow 80/tcp

ufw default deny incoming

ufw default allow outgoing

Save and exit the file (in nano, use CTRL+O, then Enter, and CTRL+X to exit).

1. **Make the Script Executable**:

bash

Copy code

chmod +x /usr/local/bin/firewall.sh

**6. Modify the Dockerfile**

To include the firewall setup in your Docker image, modify the Dockerfile:

1. **Create or Modify Dockerfile**:

bash

Copy code

nano Dockerfile

1. **Add the Following Content**:

Dockerfile

Copy code

FROM ubuntu:latest

RUN apt-get update && \

apt-get install -y iptables ufw

# Copy the firewall script into the container

COPY firewall.sh /usr/local/bin/firewall.sh

# Make the script executable

RUN chmod +x /usr/local/bin/firewall.sh

# Set the script as the entrypoint

ENTRYPOINT ["/usr/local/bin/firewall.sh"]

Save and exit the file.

**7. Build and Run the Docker Image**

1. **Build the Docker Image**:

bash

Copy code

sudo docker build -t my-secure-container .

1. **Run the Docker Container**:

bash

Copy code

sudo docker run -it --name secure-container my-secure-container

**8. Control Network Traffic on the Host**

To control network traffic from the Docker host:

1. **Allow or Drop Traffic to/from a Container**:

Allow traffic on port 80:

bash

Copy code

sudo iptables -A DOCKER-USER -d <container-ip> -p tcp --dport 80 -j ACCEPT

Drop all traffic:

bash

Copy code

sudo iptables -A DOCKER-USER -d <container-ip> -j DROP

1. **Save Host Firewall Rules**:

bash

Copy code

sudo iptables-save > /etc/iptables/rules.v4