Dockerize 4 tier application such as Nginx,Postgres-db,Phyton,Redis

set up a Dockerized Four-Tier application with Nginx, PostgreSQL, Python (Flask application), and Redis, we can use Docker Compose to define and manage the containers. Below is a step-by-step guide to help you get started:

Installation instructions

1. Launch amazon ubuntu server in aws

2. ssh to ubuntu to install packages

$ ssh -i <key.pem> ubuntu@<ip-address>

3. Update and Upgrade linux machine

$ sudo apt update

$ sudo apt upgrade

install docker

1. Next, install a few prerequisite packages which let apt use packages over HTTPS:

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

2. Then add the GPG key for the official Docker repository to your system:

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

3. Add the Docker repository to APT sources:

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

4. Update your existing list of packages again for the addition to be recognized:

$ sudo apt update

5. Make sure you are about to install from the Docker repo instead of the default Ubuntu repo:

$ apt-cache policy docker-ce

6. You'll see output like this, although the version number for Docker may be different:

Output of apt-cache policy docker-ce
docker-ce:
  Installed: (none)
  Candidate: 5:20.10.14~3-0~ubuntu-jammy
  Version table:
    5:20.10.14~3-0~ubuntu-jammy 500
      500 https://download.docker.com/linux/ubuntu jammy/stable amd64 Packages
    5:20.10.13~3-0~ubuntu-jammy 500
      500 https://download.docker.com/linux/ubuntu jammy/stable amd64 Packages
Notice that docker-ce is not installed, but the candidate for installation is from the Docker repository for Ubuntu 22.04 (jammy).

7. Finally, install Docker:

$ sudo apt install docker-ce

8. Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running:

$ sudo systemctl status docker

The output should be similar to the following, showing that the service is active and running:

Output
● docker.service - Docker Application Container Engine
    Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
    Active: active (running) since Fri 2022-04-01 21:30:25 UTC; 22s ago
TriggeredBy: ● docker.socket
      Docs: https://docs.docker.com
  Main PID: 7854 (dockerd)
     Tasks: 7
    Memory: 38.3M
       CPU: 340ms
    CGroup: /system.slice/docker.service
            └─7854 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Docker Compose Installation

1. To install docker compose on Ubuntu Linux, execute the following commands one after the another

$ sudo curl -L "https://github.com/docker/compose/releases/download/v2.20.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

2. give Execute Permision " +x " docker-compose file for running

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

3. Check the docker-compose version by running following command,

$ docker-compose --version

4. If you want to avoid typing sudo whenever you run the docker command, add your username to the docker group:

$ sudo usermod -aG docker ${USER}

$ newgrp docker

5. Clone 4_tier_project Nginx-webserver,Phyton,Postgres-db,Redis

$ cd /home/ubuntu

$ git clone https://github.com/rehankhanmrk/dockerize-4-tier-project.git

$ cd dockerize-4-tier-project

Application Code:

Create a directory named app in your project directory to store your Python application code. Inside the app directory, you'll need to have a Flask application that connects to the PostgreSQL database and uses Redis for caching.
You'll also need a requirements.txt file to specify the Python dependencies.

6. show Dockerfile

$ cat Dockerfile

7. Dockerfile

```
FROM tiangolo/uwsgi-nginx-flask:python3.8

WORKDIR /app

COPY ./app/requirements.txt /app/
RUN pip3 install -r /app/requirements.txt

COPY ./app /app
```

8. show index.html file

$ cat app/templates/index.html

index.html file

```
<body style="background-color: black;">
<center style="background-color: blue;">
<h1> Wish-List App From Rehan DevOps For Tech_bridge</h1>

<b> Create your wishlist record here</b>

<form action="/save" method="POST">

        <label for="username">Username</label>
        <input name="username" id="username" required><br><br>

        <label for="place">Favorite Place</label>
        <input name="place" id="place" required><br><br>

        <label for="food">Favorite Food</label>
        <input name="food" id="food" required><br><br>

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

</form>

<b>Type in the username and fetch the WishList</b>
<form action="/get" method="POST">
        <label for="username">Username</label>
        <input name="username" id="username" required><br><br>
        <input type="submit" value="Submit">
</form>

{% if get %}
<font color="blue">The Wishlist for <b>{{username}}</b> is <b>{{place}}</b> and <b>{{food}}</b></font>
<small>{{msg}}</small>
{% endif %}

{% if no_record %}
<font color="red">{{msg}}</font>
{% endif %}

{% if user_exists %}
<font color="red">User <b>{{username}}</b> already exists with the wishlist <b>{{place}}</b> and <b>{{food}}</b></font>
<small>{{msg}}</small>
{% endif %}

{% if saved %}
<font color="green"> Successfully saved data for <b>{{username}}</b> with Wishlist <b>{{place}}</b> and
<b>{{food}}</b>!!!</font>
{% endif %}

</center>
<br>
<a href="/keys">Keys</a>
{% if keys %}
        <h3> Keys </h3>
        <small>(From DataBase)</small>
        <ul>
```

```
        {% for username in usernames %}
                <li>{{username}}</li>
        {% endfor %}
        </ul>
{% endif %}


</body>
```

9. Here's a Flask application example (main.py):

$ cat app/main.py

```
from flask import Flask, request, render_template
import redis

app = Flask(__name__)

# postgresql://username:password@host:port/database
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://hello_flask:hello_flask@db:5432/hello_flask_dev'

from models import db, UserFavs

db.init_app(app)
with app.app_context():
   # To create / use database mentioned in URI
   db.create_all()
   db.session.commit()

red = redis.Redis(host='redis', port=6379, db=0)

@app.route("/")
def main():
   return render_template('index.html')

@app.route("/save", methods=['POST'])
def save():
   username = str(request.form['username']).lower()
   place = str(request.form['place']).lower()
   food = str(request.form['food']).lower()

   # check if data of the username already exists in the redis
   if red.hgetall(username).keys():
      print("hget username:", red.hgetall(username))
      # return a msg to the template, saying the user already exists(from redis)
      return render_template('index.html', user_exists=1, msg='(From Redis)', username=username,
place=red.hget(username,"place").decode('utf-8'), food=red.hget(username,"food").decode('utf-8'))

   # if not in redis, then check in db
   elif len(list(red.hgetall(username)))==0:
```

```
    record =  UserFavs.query.filter_by(username=username).first()
    print("Records fecthed from db:", record)

    if record:
      red.hset(username, "place", place)
      red.hset(username, "food", food)
      # return a msg to the template, saying the user already exists(from database)
      return render_template('index.html', user_exists=1, msg='(From DataBase)', username=username,
place=record.place, food=record.food)

  # if data of the username doesnot exist anywhere, create a new record in DataBase and store in Redis also
  # create a new record in DataBase
  new_record = UserFavs(username=username, place=place, food=food)
  db.session.add(new_record)
  db.session.commit()

  # store in Redis also
  red.hset(username, "place", place)
  red.hset(username, "food", food)

  # cross-checking if the record insertion was successful into database
  record =  UserFavs.query.filter_by(username=username).first()
  print("Records fetched from db after insert:", record)

  # cross-checking if the insertion was successful into redis
  print("key-values from redis after insert:", red.hgetall(username))

  # return a success message upon saving
  return render_template('index.html', saved=1, username=username, place=red.hget(username, "place").decode('utf-8'),
food=red.hget(username, "food").decode('utf-8'))

@app.route("/keys", methods=['GET'])
def keys():
        records = UserFavs.query.all()
        names = []
        for record in records:
                names.append(record.username)
        return render_template('index.html', keys=1, usernames=names)


@app.route("/get", methods=['POST'])
def get():
        username = request.form['username']
        print("Username:", username)
        user_data = red.hgetall(username)
        print("GET Redis:", user_data)

        if not user_data:
                record = UserFavs.query.filter_by(username=username).first()
                print("GET Record:", record)
                if not record:
                        print("No data in redis or db")
                        return render_template('index.html', no_record=1, msg=f"Record not yet defined for
{username}")
                red.hset(username, "place", record.place)
```

```
                red.hset(username, "food", record.food)
                return render_template('index.html', get=1, msg="(From DataBase)",username=username,
place=record.place, food=record.food)
        return render_template('index.html',get=1, msg="(From Redis)", username=username,
place=user_data[b'place'].decode('utf-8'), food=user_data[b'food'].decode('utf-8'))
```

10. This docker-compose.yml file defines the same structure as mentioned earlier with Nginx, Python (application), PostgreSQL (database), and Redis (cache) services.

11. show docker-compose.yml file

$ cat docker-compose.yml

docker-compose.yml file

```
version: '3'
services:
  nginx:
    build: ./
    environment:
      - PYTHONBUFFERED=1
    ports:
      - "80:80"
    links:
      - redis
      - db
  redis:
    image: redis:6.0.8
    ports:
      - "6379:6379"
  db:
    image: postgres:12-alpine
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      - POSTGRES_USER=hello_flask
      - POSTGRES_PASSWORD=hello_flask
      - POSTGRES_DB=hello_flask_dev
    ports:
      - "5432:5432"

volumes:
  postgres_data:
```

12. run this Docker Compose file, navigate to the directory containing the docker-compose.yml file and execute the following command:

$ docker-compose up -d

13. check docker-compose running condition

$ docker ps

14. OR

$ docker-compose ps

then access your application through the Nginx web server at http://publicip

http://12.23.43.44

With this setup, you'll have a Dockerized Four-Tier application running with Nginx serving as the web tier, Python handling the application tier, PostgreSQL as the data tier, and Redis as the cache tier.

11. Visit your website HTTPS://<your website>
Enjoy Your website serve

Support 🙏😃
Thanks for your support :)