docker build -t welcome-app .// docker build --no-cache -t welcome-app .
//docker build -t inamul78/welcome-app .

docker images.. Running docker image
docker run -p 5000:5000 welcome-app

dockers ps will give container id and container names
docker stop f3264e938e6c

image removal
docker image rm -f inamul78/welcome-app:latest

#Docker compose
docker compose down
docker compose up --build

📝 Flask + Docker Host URLs (Simple Notes)

1️⃣ What does `0.0.0.0` mean?

- `0.0.0.0` means **listen on all network interfaces**
- It allows the app to accept requests from:
    - localhost
    - Docker network
    - other containers

📌 In Docker, this is mandatory

```
app.run(host="0.0.0.0", port=5000)
```

2️⃣ Why do we see multiple URLs?

When Flask starts inside Docker, it shows **all reachable addresses**.

Example:

```
Running on http://127.0.0.1:5000
Running on http://172.17.0.2:5000
```

3️⃣ Meaning of each URL

- 🔹 **`127.0.0.1:5000`**
  - Local machine (host)
  - Also called **localhost**
  - ✅ This is what we open in browser

- 🔹 **`172.17.0.2:5000`**
  - Docker container's **internal IP**
  - Exists only inside Docker network
  - ❌ Not used in browser normally

4️⃣ How Docker connects them (important)

Command:

```
docker run -p 5000:5000 welcome-app
```

Meaning:

```
Host port 5000  →  Container port 5000
```

So:

```
localhost:5000  →  Docker  →  Flask app
```

5️⃣ Why not bind Flask to `127.0.0.1`?

❌ This will NOT work in Docker:

```
app.run()
```

Because:

- `127.0.0.1` inside container ≠ your laptop

✅ Correct way:

```
app.run(host="0.0.0.0")
```

6️⃣ Which URL should YOU use?

✔ Always use:

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

7️⃣ One-line exam answer

Two URLs appear because Flask runs inside Docker's private network and Docker maps it to localhost.

🔑 Key Takeaway (remember this)

- `0.0.0.0` → listen everywhere
- `127.0.0.1` → local machine
- Docker bridges both using `-p`

✅ CORRECT FIX (DO THIS)

🔹 Step 1: Check you have `requirement.txt`

You already do:

`requirement.txt`

Open it and make sure it contains:

```
numpy
flask
```

(or whatever your app uses)

🔹 Step 2: Create / Fix your `Dockerfile`

Inside **Section_46 Docker**, your `Dockerfile` should look like this:

```
FROM python:3.10-slim
```

```
WORKDIR /app

COPY requirement.txt .

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

COPY . .

EXPOSE 5000

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

⚠️ Important:

- `pip install` happens **before** running app
- This installs numpy inside the image

🔹 Step 3: Rebuild the Docker image (MANDATORY)

Your old image does NOT have numpy.

Run:

```
docker build -t welcome-app .
```

Wait until it finishes.

🔹 Step 4: Run the container again

```
docker run -p 5000:5000 welcome-app
```

✅ Now the error should be **GONE**.

🧪 Verify it’s working

Open browser:

http://localhost:5000

📝 Flask + Docker Host URLs (Simple Notes)

1️⃣ What does 0.0.0.0 mean?

- `0.0.0.0` means **listen on all network interfaces**
- It allows the app to accept requests from:
  - localhost
  - Docker network
  - other containers

📌 In Docker, this is mandatory

```
app.run(host="0.0.0.0", port=5000)
```

2️⃣ Why do we see multiple URLs?

When Flask starts inside Docker, it shows **all reachable addresses**.

Example:

```
Running on http://127.0.0.1:5000
Running on http://172.17.0.2:5000
```

3️⃣ Meaning of each URL

🔹 **127.0.0.1:5000**

- Local machine (host)
- Also called **localhost**
- ✅ This is what we open in browser

🔹 **172.17.0.2:5000**

- Docker container's **internal IP**
- Exists only inside Docker network
- ❌ Not used in browser normally

4️⃣ How Docker connects them (important)

Command:

```
docker run -p 5000:5000 welcome-app
```

Meaning:

Host port 5000 → Container port 5000

So:

localhost:5000 → Docker → Flask app

5️⃣ Why not bind Flask to 127.0.0.1?

❌ This will NOT work in Docker:

```
app.run()
```

Because:

- 127.0.0.1 inside container ≠ your laptop

✅ Correct way:

```
app.run(host="0.0.0.0")
```

6️⃣ Which URL should YOU use?

✔ Always use:

http://localhost:5000

7️⃣ One-line exam answer

> **Two URLs appear because Flask runs inside Docker's private network and Docker maps it to localhost.**

🔑 Key Takeaway (remember this)

- 0.0.0.0 → listen everywhere
- 127.0.0.1 → local machine
- Docker bridges both using -p

If you want, I can also give:

- **Docker networking diagram**
- **Interview Q&A**
- **One-page Docker + Flask cheatsheet**

Just tell me 👍

docker image and docker conatner differnce as we delete docker conatiner docker iamge ins not deleted what are they and how tehr are related??

This is a **very important Docker concept**.
I'll explain it in **simple notes + analogy + commands**, so it sticks forever.

🐳 Docker Image vs Docker Container (Simple Notes)

1️⃣ What is a Docker Image?

👉 A **Docker image** is a **blueprint / template**.

- Read-only
- Contains:
  - OS layer (Linux)
  - Python
  - Libraries (numpy, pandas, etc.)
  - Your app code
- Cannot run by itself

📌 **Think:** Image = *class / recipe*

Example:

docker images

2️⃣ What is a Docker Container?

👉 A **Docker container** is a **running instance of an image**.

- Created from an image

- Has:
    - Running process
    - Temporary writable layer
- Can be started, stopped, deleted

📌 **Think:** Container = *object / cooked dish*

Example:

```
docker ps
```

3️⃣ Relationship (MOST IMPORTANT)

```
Docker Image  →  Docker Container
   (template)     (running copy)
```

- One image ➜ many containers
- Containers **depend on image**
- Image does **not depend on containers**

4️⃣ Why deleting a container does NOT delete the image?

Because:

- Image is the **source**

- Container is just a **copy**

Deleting a copy does not delete the original file.

📌 Just like:

- Deleting a **running program**
- Does not delete the **software installer**

5️⃣ Visual analogy (easy to remember)

Real World	Docker
Recipe	Image
Cooked food	Container
Class	Image
Object	Container
App installer	Image
Running app	Container

6️⃣ Commands summary (important)

🔹 Build image

```
docker build -t welcome-app .
```

🔹 Run container

```
docker run welcome-app
```

🔹 List images

```
docker images
```

🔹 List containers

```
docker ps -a
```

🔹 Delete container

```
docker rm <container_id>
```

👉 Image stays ✅

🔹 Delete image (manual)

```
docker rmi welcome-app
```

⚠️ Only possible if no container is using it.

7️⃣ Why Docker works like this?

Because it allows:

- Fast startup
- Reusability

- Multiple containers from one image
- Easy scaling

Example:

```
docker run welcome-app
docker run welcome-app
docker run welcome-app
```

➡ 3 containers from **same image**

8️⃣ One-line interview answer

> **A Docker image is a read-only blueprint, while a container is a running instance of that image. Deleting a container does not delete the image.**

🔑 Final takeaway

- Image = **what to run**
- Container = **running thing**
- Delete container ➜ image stays
- Delete image ➜ container cannot run

```
import time

import redis
from flask import Flask
```

```
app= Flask(__name__)
cache = redis.Redis(host='redis',port=6379)


def get_hit_count():
    retries =5
    while True:
        try:
            cache.reset_retry_count()
            return cache.incr('hits')
        except redis.exceptionsConnectionError as exc:
            if retries ==0:
                raise exc
            retires -=1
            time.sleep(0.5)

@app.route('/')
def hello():
    count = get_hit_count()
    return 'Hellp Krish I have been seen {} times .\n'.format(count)
```

🧠 Big Picture (start here)

This app does **one simple thing**:

> Every time you open the website, it **counts how many times the page was visited**, using **Redis**.

- Flask → web server
- Redis → counter storage
- Docker → runs Flask and Redis as separate services

📦 What each part does

1️⃣ Imports

```
import time
import redis
from flask import Flask
```

- `time` → to wait before retrying
- `redis` → Python client to talk to Redis server
- `Flask` → web framework

2️⃣ Create Flask app

```
app = Flask(__name__)
```

This creates your web application.

3️⃣ Connect to Redis

```
cache = redis.Redis(host='redis', port=6379)
```

🔴 This line is **VERY IMPORTANT**

- `redis` is **NOT localhost**
- It is the **service name** of Redis in Docker (usually from `docker-compose.yml`)
- Port `6379` is Redis default port
  👉 Flask container talks to Redis container using **Docker networking**.

4️⃣ The counter logic

```
def get_hit_count():
    retries = 5
```

This function:

- Tries to increase a counter in Redis
- Retries if Redis is not ready yet

🔁 Retry loop

```
while True:
    try:
        return cache.incr('hits')
```

- `cache.incr('hits')` means:

  "Increase the value of key `hits` by 1 and return it"

  Example in Redis:

```
hits = 1
hits = 2
hits = 3
```

If Redis works → function returns immediately.

❌ If Redis is not ready

```
except redis.exceptions.ConnectionError as exc:
```

- Happens when Redis container hasn't started yet

⏳ Retry mechanism

```
if retries == 0:
```

```
    raise exc
retries -= 1
time.sleep(0.5)
```

- Try again up to **5 times**
- Wait **0.5 seconds** between tries
- If still failing → crash the app

5️⃣ Flask route

```
@app.route('/')
def hello():
```

This means:

> When user opens `/` (homepage), run this function

🔢 Get visit count

```
count = get_hit_count()
```

- Calls Redis
- Increments counter
- Gets visit number

📤 Send response

```
return 'Hello Krish I have been seen {} times.\n'.format(count)
```

Example output:

```
Hello Krish I have been seen 7 times.
```

🔄 How everything works together (VERY IMPORTANT)

Step-by-step flow:

1. Browser opens `http://localhost:5000`
2. Flask route `/` is called
3. `get_hit_count()` runs
4. Redis increments `hits`
5. Count is returned
6. Flask sends response to browser

🧱 Where Docker comes in

Usually you run this with **Docker Compose**:

```
services:
  web:
    build: .
    ports:
      - "5000:5000"
  redis:
    image: redis
```

- Flask container → service name `web`
- Redis container → service name `redis`
- `host='redis'` works because Docker provides DNS

❗ Small MISTAKES in your code (important)

❌ Typo 1

```
cache.reset_retry_count()
```

❌ This function does NOT exist in Redis
👉 **Remove this line**

❌ Typo 2

```
except redis.exceptionsConnectionError as exc:
```

❌ Missing dot

✅ Correct:

```
except redis.exceptions.ConnectionError as exc:
```

❌ Typo 3

```
retires -=1
```

❌ Wrong variable name

✅ Correct:

```
retries -= 1
```

✅ Corrected version (clean)

```
def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)
```

📝 Simple exam-style summary

- Flask handles web requests
- Redis stores visit count
- `incr()` increases counter
- Retry logic waits for Redis
- Docker connects Flask and Redis using service names

🔑 One-line takeaway

> **Flask shows the page, Redis remembers how many times it was visited, and Docker lets them talk to each other.**

```
# ------------------------------
#1️⃣ Base Image
# ------------------------------
# Use a lightweight Linux image that already has Python 3.10 installed
# This saves us from installing Python manually
FROM python:3.10-slim


# ------------------------------
#2️⃣ Working Directory
# ------------------------------
# Create a folder named /code inside the container
# All commands will now run from this directory
WORKDIR /code


# ------------------------------
#3️⃣ Environment Variables for Flask
# ------------------------------
# Tell Flask which Python file is the main application
ENV FLASK_APP=app.py

# Make Flask listen on all network interfaces
# This is REQUIRED so Docker can expose the app to the browser
ENV FLASK_RUN_HOST=0.0.0.0


# ------------------------------
#4️⃣ Copy Project Files
# ------------------------------
# Copy everything from the current project folder (host)
# into the /code directory inside the container
# This includes app.py, requirements.txt, templates, etc.
COPY . .


# ------------------------------
#5️⃣ Install Python Dependencies
# ------------------------------
# Install all libraries listed in requirements.txt
# These are installed INSIDE the container (not your local system)
RUN pip install -r requirements.txt
```

```
# ------------------------------
#6️⃣ Expose Port
# ------------------------------
# Inform Docker that this container uses port 5000
# (Actual port mapping happens in `docker run -p`)
EXPOSE 5000


# ------------------------------
#7️⃣ Run the Application
# ------------------------------
# This command runs automatically when the container starts
# Equivalent to running: flask run
CMD ["flask", "run"]
```

✅ Correct & COMMENTED docker-compose.yml

Save this as **docker-compose.yml**

```
version: "3.8"   # Docker Compose file version

services:

  # -------------------------
  # Flask Web Application
  # -------------------------
  web:
    build: .              # Build image using Dockerfile in current
folder
    ports:
      - "8000:5000"       # Host port 8000 → Container port 5000
    depends_on:
      - redis             # Start redis before web
      - mysql             # Start mysql before web
```

```
  # -------------------------
  # Redis Service
  # -------------------------
  redis:
    image: redis          # Pull official Redis image from Docker Hub
    ports:
      - "6379:6379"       # Optional: expose Redis to host

  # -------------------------
  # MySQL Database
  # -------------------------
  mysql:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: mydb
    ports:
      - "3306:3306"
```

🔗 How all services connect (MAGIC PART)

Service	Hostname to use
Flask	`web`
Redis	`redis`
MySQL	`mysql`

Example in Python:

```
redis.Redis(host="redis", port=6379)

mysql.connect(host="mysql", user="root", password="root")
```

👉 No IP addresses needed.

▶️ How to run everything

From folder containing `docker-compose.yml`:

```
docker-compose up --build
```

Now you get:

- Flask app
- Redis
- MySQL
  all running together 🎉

🧠 Mental Model (remember this)

Thing	Meaning
Dockerfile	How to build ONE app
Image	Blueprint
Container	Running app
Docker Compose	Run MANY containers together
services	Individual containers
service name	Network hostname

📝 One-line exam / interview answer

Docker Compose is a tool used to define and run multi-container Docker applications using a single YAML file.