

# MACHINE LEARNING OPERATIONS

UNLIMITED



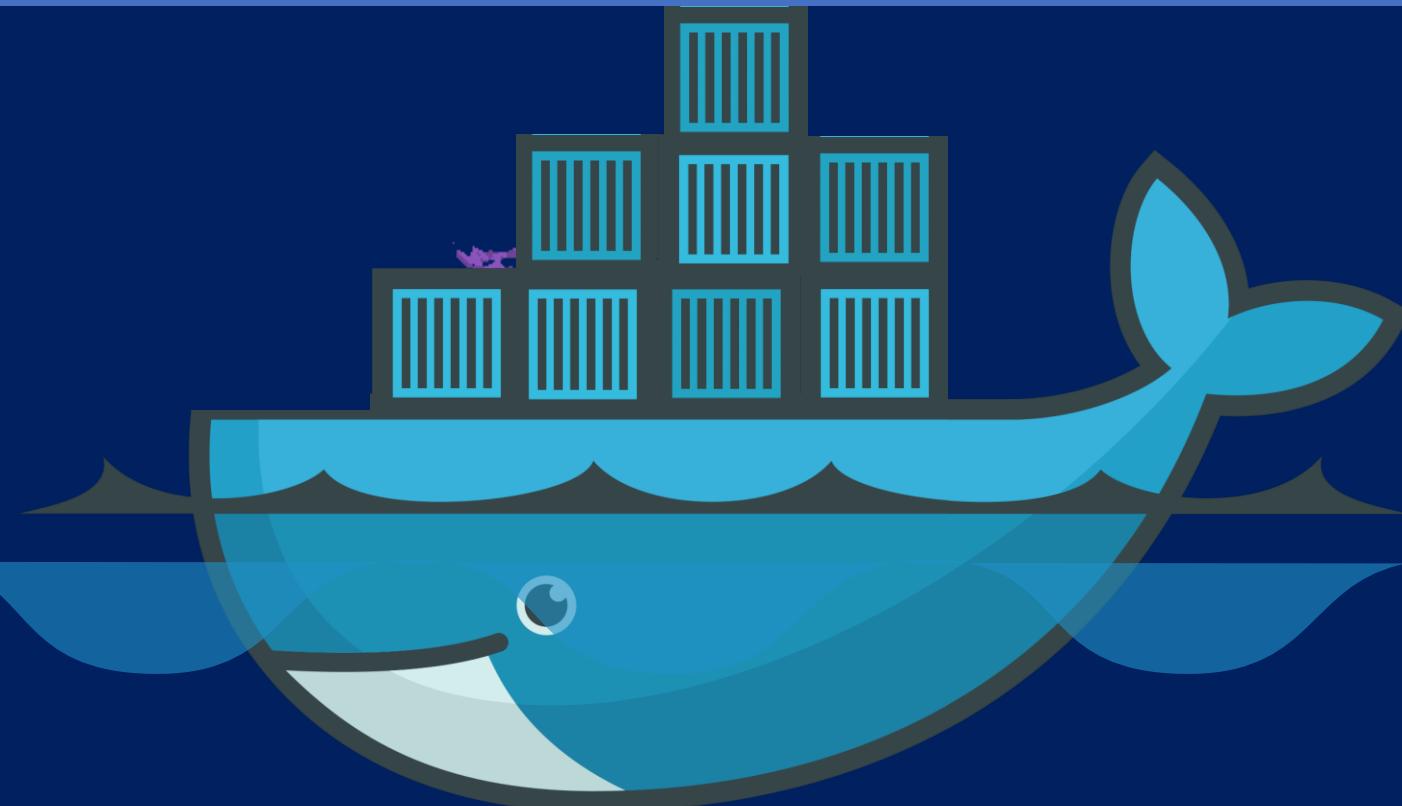
WEEK 11



Presented by **Asst. Prof. Dr. Tuchsanai Ploysuwan**

# Dockerfile and Docker Image

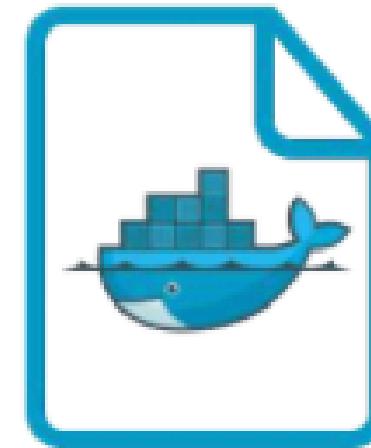
---



# Dockerfile

---

- Dockerfile is instructions to build Docker Image
  - How to run commands
  - Add files or directories
  - Create environment variables
  - What process to run when launching container
- Result from building Dockerfile is Docker Image



Dockerfile

# Sample Dockerfile

```
FROM node:14.15.0-alpine3.12 ← OS + System Packages
COPY . /nodejs/.           ← Source Code
WORKDIR /nodejs
RUN npm install             ← Library Dependencies
ENV VERSION 1.0            ← Configuration
EXPOSE 8081
CMD ["node", "/nodejs/main.js"]
```

## Dockerfile reference

FROM <image:tag>	Sets the base image for subsequent instructions.
RUN <command>	Execute any commands in image.
CMD <command> <param1> <param2>	Sets the command to be executed when running the image.
LABEL <key>=<value> ...	Adds metadata to an image.
EXPOSE <port>	Informs Docker that the container listens on the specified network ports at runtime.
ENV <key> <value>	Sets the environment variable.
COPY <src> <dest>	Copies new files from source to the filesystem of the container at the path destinations.
ENTRYPOINT <command> <param1> <param2>	Command line arguments will be appended after all elements in an exec form ENTRYPOINT.
VOLUME ["/data"]	Sets mounted volumes from native host or other containers.
WORKDIR <path>	Sets the working directory for any commands.

# How to create my own image?

Dockerfile

```
FROM Ubuntu  
  
RUN apt-get update  
RUN apt-get install python  
  
RUN pip install flask  
RUN pip install flask-mysql  
  
COPY ./opt/source-code  
  
ENTRYPOINT FLASK_APP=/opt/source-code/app.py flask run
```

1. OS - Ubuntu

2. Update apt repo

3. Install dependencies using apt

4. Install Python dependencies using pip

5. Copy source code to /opt folder

6. Run the web server using “flask” command

```
docker build -t tuchsanai/my-custom-app .
```

```
docker push tuchsanai/my-custom-app
```



# Dockerfile

Dockerfile

INSTRUCTION

ARGUMENT

Dockerfile

FROM Ubuntu

RUN apt-get update

RUN apt-get install python

RUN pip install flask

RUN pip install flask-mysql

COPY ./opt/source-code

ENTRYPOINT FLASK\_APP=/opt/source-code/app.py flask run

Start from a base OS or another image

Install all dependencies

Copy source code

Specify Entrypoint

# Layered architecture

Dockerfile

```
FROM Ubuntu

RUN apt-get update && apt-get -y install python

RUN pip install flask flask-mysql

COPY ./opt/source-code

ENTRYPOINT FLASK_APP=/opt/source-code/app.py flask run
```

docker build Dockerfile -t mmumshad/my-custom-app



```
root@osboxes:/root/simple-webapp-docker # docker history mmumshad/simple-webapp
IMAGE          CREATED      CREATED BY
1a45ba829f10  About an hour ago /bin/sh -c #(nop) ENTRYPOINT ["/bin/sh" ... 0B
37d37ed8fe99  About an hour ago /bin/sh -c #(nop) COPY file:29b92853d73898... 229B
d6aaebf8ded0  About an hour ago /bin/sh -c pip install flask flask-mysql 6.39MB
e4c055538e60  About an hour ago /bin/sh -c apt-get update && apt-get insta... 306MB
ccc7a11d65b1  2 weeks ago   /bin/sh -c #(nop) CMD ["/bin/bash"] 0B
<missing>     2 weeks ago   /bin/sh -c mkdir -p /run/systemd && echo '...' 7B
<missing>     2 weeks ago   /bin/sh -c sed -i 's/^#\s*/(deb.*universe\... 2.76kB
<missing>     2 weeks ago   /bin/sh -c rm -rf /var/lib/apt/lists/* 0B
<missing>     2 weeks ago   /bin/sh -c set -xe  && echo '#!/bin/sh' >... 745B
<missing>     2 weeks ago   /bin/sh -c #(nop) ADD file:39d3593ea220e68... 120MB
```

# Docker build output

```
root@osboxes:/root/simple-webapp-docker # docker build .
Sending build context to Docker daemon 3.072kB
Step 1/5 : FROM ubuntu
--> ccc7a11d65b1
Step 2/5 : RUN apt-get update && apt-get install -y python python-setuptools python-dev
--> Running in a7840dbfad17
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:4 http://security.ubuntu.com/ubuntu xenial-security/universe Sources [46.3 kB]
Get:5 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Get:6 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [440 kB]
Step 3/5 : RUN pip install flask flask-mysql
--> Running in a4a6c9190ba3
Collecting flask
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Collecting flask-mysql
  Downloading Flask_SQLAlchemy-1.4.0-py2.py3-none-any.whl
Removing intermediate container a4a6c9190ba3
Step 4/5 : COPY app.py /opt/
--> e7cdab17e782
Removing intermediate container faaaaaf63c512
Step 5/5 : ENTRYPOINT FLASK_APP=/opt/app.py flask run --host=0.0.0.0
--> Running in d452c574a8bb
--> 9f27c36920bc
Removing intermediate container d452c574a8bb
Successfully built 9f27c36920bc
```

# failure



Layer 1. Base Ubuntu Layer

Layer 2. Changes in apt packages

Layer 3. Changes in pip packages

Layer 4. Source code

Layer 5. Update Entrypoint with “flask” command

```
docker build Dockerfile -t mmumshad/my-custom-app
```

```
root@osboxes:/root/simple-webapp-docker # docker build .
Sending build context to Docker daemon 5.12kB
Step 1/5 : FROM ubuntu
--> ccc7a11d65b1
Step 2/5 : RUN apt-get update && apt-get install -y python python-pip
--> Using cache
--> e4c055538e60
Step 3/5 : RUN pip install flask
--> Running in aacdaccd7403
Collecting flask
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Removing intermediate container aacdaccd7403
Step 4/5 : COPY app.py /opt/
--> af41ef57f6f3
Removing intermediate container a49cc8befc8f
Step 5/5 : ENTRYPOINT FLASK_APP=/opt/app.py flask run --host=0.0.0.0
--> Running in 3d745ff07d5a
--> 910416d360b6
Removing intermediate container 3d745ff07d5a
Successfully built 910416d360b6
```

# LAB 1 :

The screenshot shows a GitHub repository interface for a lab titled "01\_LAB\_CMD\_EntryPoint".

**Repository Overview:**

- Code tab:** Selected.
- Issues, Pull requests, Actions, Projects, Security, Insights, Settings:** Available tabs.
- Branch:** main
- Commit:** Tuchsanai · d · 906b441 · 14 minutes ago · History
- File List:** Dockerfile-CMD, Dockerfile-CMD-ENTRYPOINT, Dockerfile-ENTRYPOINT, README.md
- Last commit message:** ff
- Last commit date:** 18 minutes ago

**README.md Content:**

### Objective CMD and ENTRYPOINT

The goal of this lab is to understand:

- How `CMD` and `ENTRYPOINT` instructions define a container's default behavior.
- The difference in behavior when using them separately and together.
- How to override the default behavior defined by these instructions.

### Prerequisites

- Docker installed on your machine.

### Lab Setup

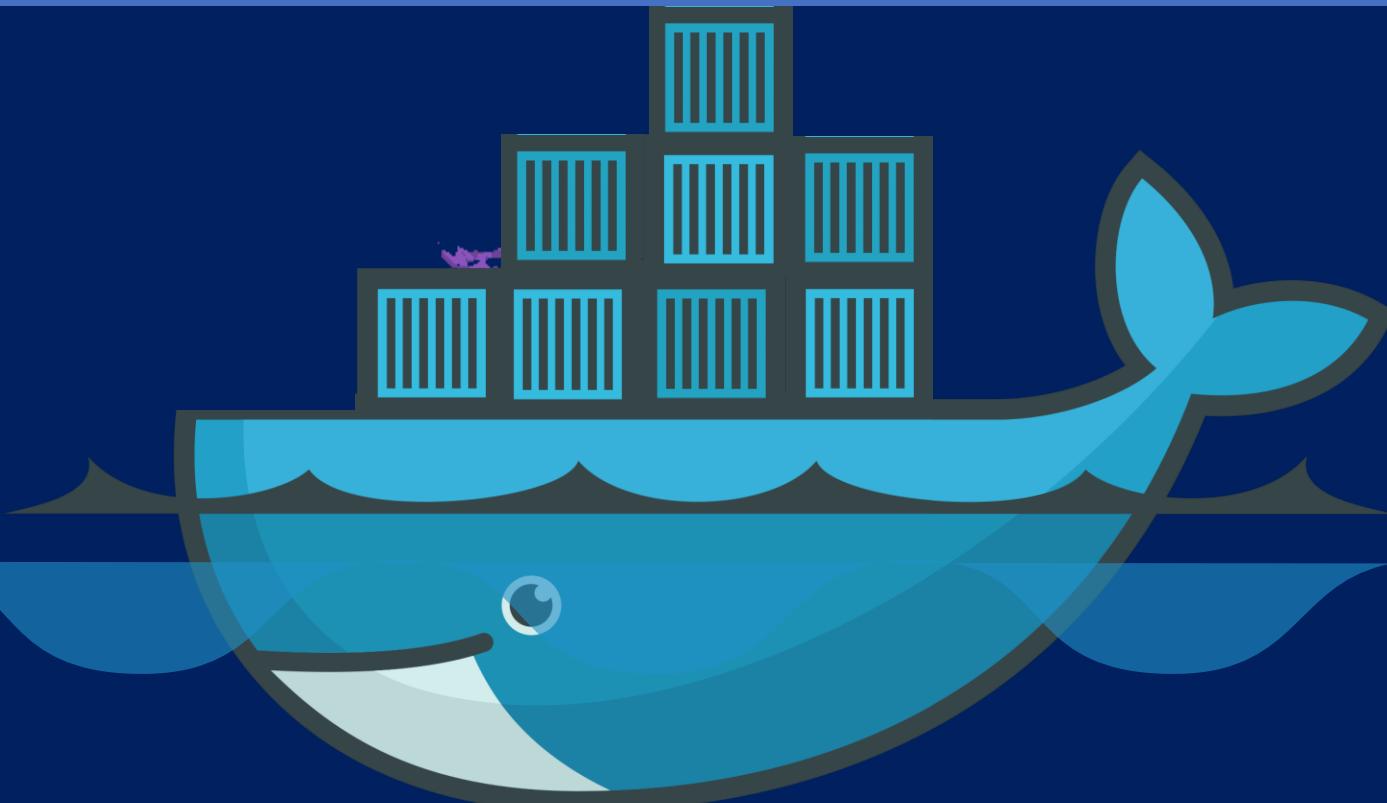
#### create directory

```
mkdir LAB0_Week11
cd LAB0_Week11
```

#### git clone branch dev

# Docker registry

---



**hub.docker.com**

**docker hub**

**Develop faster. Run anywhere.**

Docker Hub is the world's easiest way to create, manage, and deliver your team's container applications.

Search Docker Hub

1

**Spotlight**

- CLOUD DEVELOPMENT**  
Build up to 39x faster with Docker Build Cloud  
Introducing Docker Build Cloud: A new solution to speed up build times and improve developer productivity
- AI/ML DEVELOPMENT**  
LLM Everywhere: Docker and Hugging Face  
Set up a local development environment for Hugging Face with Docker
- SOFTWARE SUPPLY CHAIN**  
Take action on prioritized insights  
Bridge the gap between development workflows and security needs

**Featured Solutions**

- Kubescape
- Ambassador Telepresence
- grafana/grafana
- opensearchproject/open...

**Sign in**

Sign in to Docker to continue to Docker Hub.

Username or email address xxxx

Continue

OR

Continue with Google

Continue with GitHub

2

**hub.docker.com**

**dockerhub** Explore Repositories Organizations

Search Docker Hub

tuchsanai

Create repository

tuchsanai / pytorch\_jupyterlab\_ubuntu22.04  
Contains: Image | Last pushed: 3 months ago

tuchsanai / kubirstapp  
Contains: Image | Last pushed: 5 months ago

tuchsanai / custom-nginx-registry  
Contains: Image | Last pushed: 11 months ago

3

Create An Organization

Create and manage users and grant access to your repositories.

**hub.docker.com**

**dockerhub** Explore Repositories Organizations

Search Docker Hub

tuchsanai

Create repository

tuchsanai / pytorch\_jupyterlab\_ubuntu22.04  
Contains: Image | Last pushed: 3 months ago

tuchsanai / kubirstapp  
Contains: Image | Last pushed: 5 months ago

tuchsanai / custom-nginx-registry  
Contains: Image | Last pushed: 11 months ago

What's New

My Profile

**My Account**

Billing

Sign out

Organization

Create and manage users and grant access to your repositories.

4

hub.docker.com/settings/security

5

Search Docker Hub

tuchsanai Joined April 4, 2022

General Security Default Privacy Notifications Convert Account Deactivate Account

**Access Tokens**

It looks like you have not created any access tokens. Docker Hub lets you [create tokens](#) to authenticate access. Treat personal access tokens as alternatives to your password. [Learn more](#)

New Access Token

**Two-Factor Authentication**

Two-factor authentication is not enabled yet. Two-factor authentication adds an extra layer of security to your account by requiring more than just a password to sign in. [Learn more](#)

Enable Two-Factor Authentication

Account Settings / Security

tuchsanai user

General Security Default Privacy Notifications Convert Account Deactivate Account

**Copy Access Token**

When logging in from your Docker CLI client, use this token as a password. [Learn more](#)

ACCESS TOKEN DESCRIPTION me

ACCESS PERMISSIONS Read, Write, Delete

To use the access token from your Docker CLI client:

1. Run `docker login -u [REDACTED]`
2. At the password prompt, enter the personal access token.

dckr\_pat\_d0rTt42tz[REDACTED]QnZchI-kI

WARNING: This access token will only be displayed once. It will not be stored and cannot be retrieved. Please be sure to save it now.

Copy and Close

Enable Two-Factor Authentication

6:22:31

# Image

**image:** docker.io/nginx/nginx

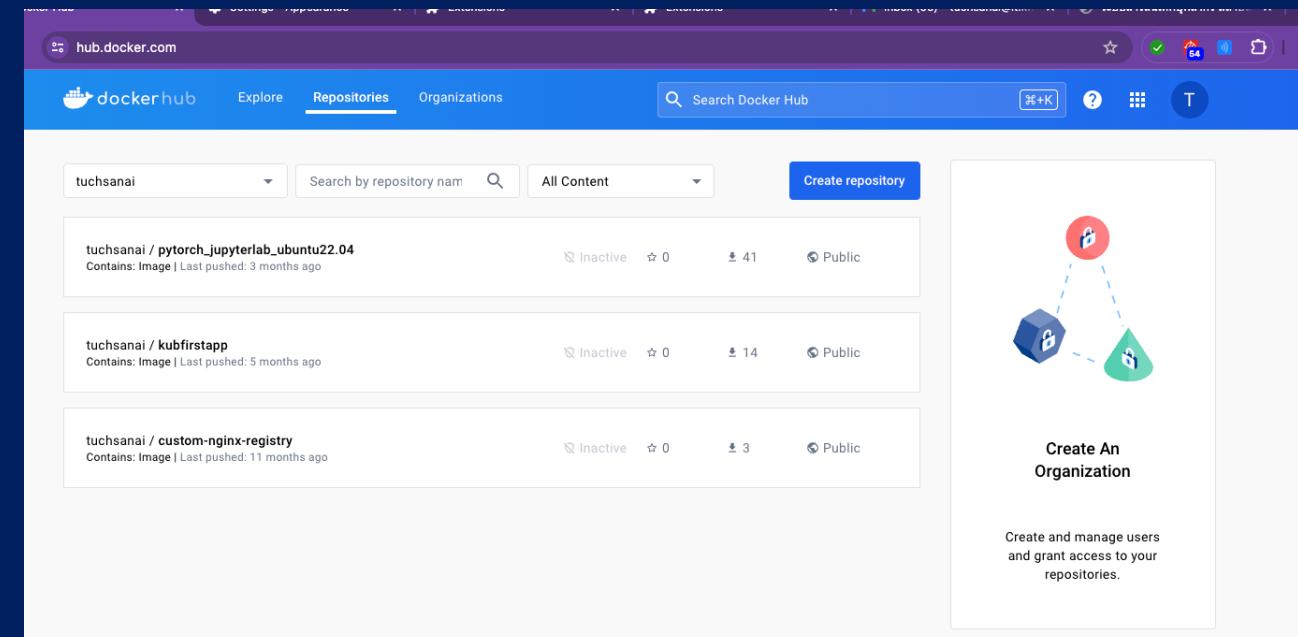


Registry

User

Image : tag

docker.io  
Docker Hub



# Private Registry

► docker login hub.docker.com

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to <https://hub.docker.com> to create one.

**Username:** registry-user

**Password:**

**WARNING!** Your password will be stored unencrypted in /home/vagrant/.docker/config.json.

Login Succeeded

► docker run private-registry.io/apps/internal-app

# Deploy Private Registry

```
docker build -t <your-dockerhub-username>/<image_repository_name>:tag .
```

```
docker push <your-dockerhub-username>/<image_repository_name>:tag
```

```
docker pull <your-dockerhub-username>/<image_repository_name>:tag
```

# LAB 2:

The screenshot shows a GitHub repository interface for a lab titled "02\_Lab\_docker\_registry".

**Repository Details:**

- Owner: Tuchsanai
- Created: 1fcb90e · 39 minutes ago
- Last commit message: History

**File Structure:**

Name	Last commit message	Last commit date
..		
images	11	39 minutes ago
Dockerfile	dd	6 hours ago
index.html	dd	6 hours ago
readme.md	11	39 minutes ago

**Content Preview:**

**readme.md**

**Lab docker registry**

**Step 1: Login to Docker Hub**

Run the following command to login to Docker Hub:

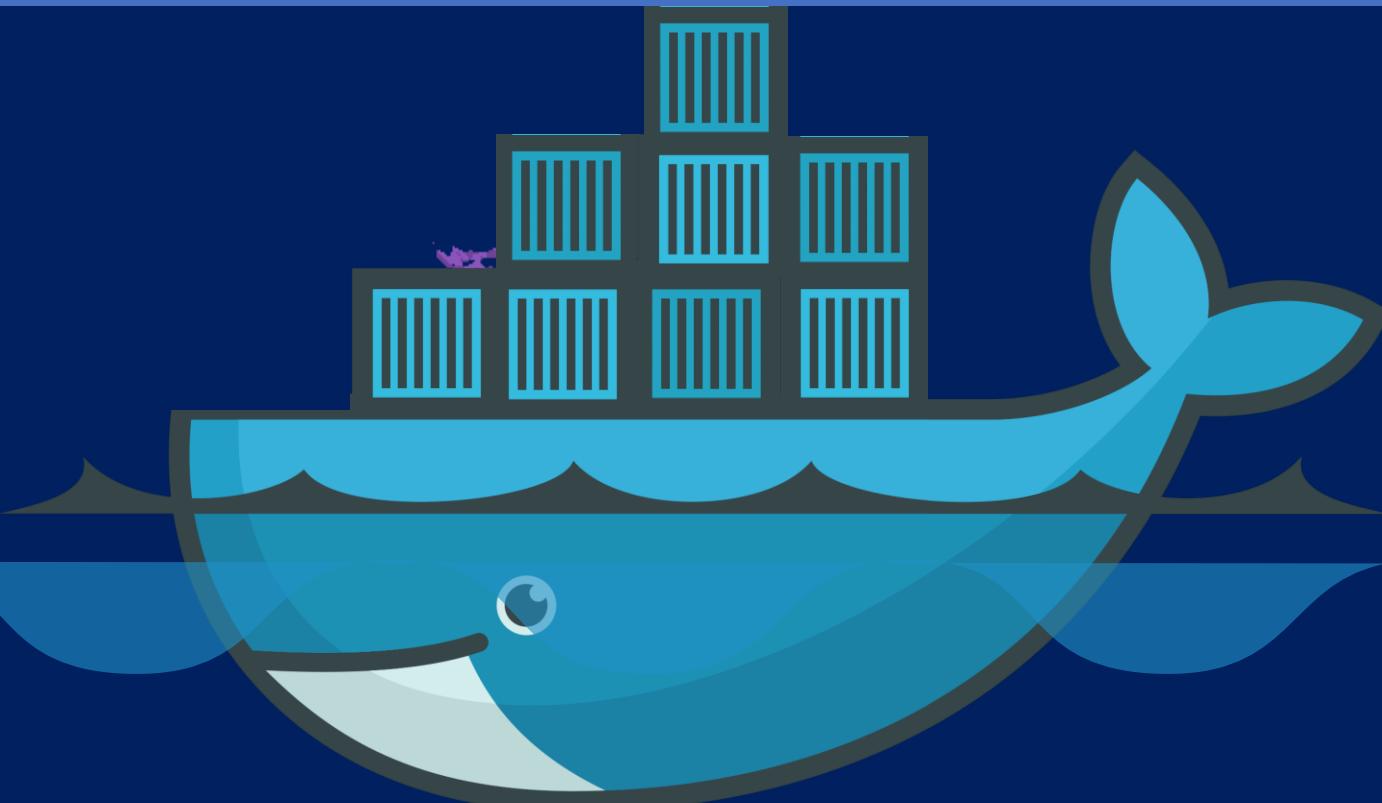
```
docker login -u yourusername
```

#### Enter your password when prompted.

**Step 2:**

# Docker Network

---



# Default networks



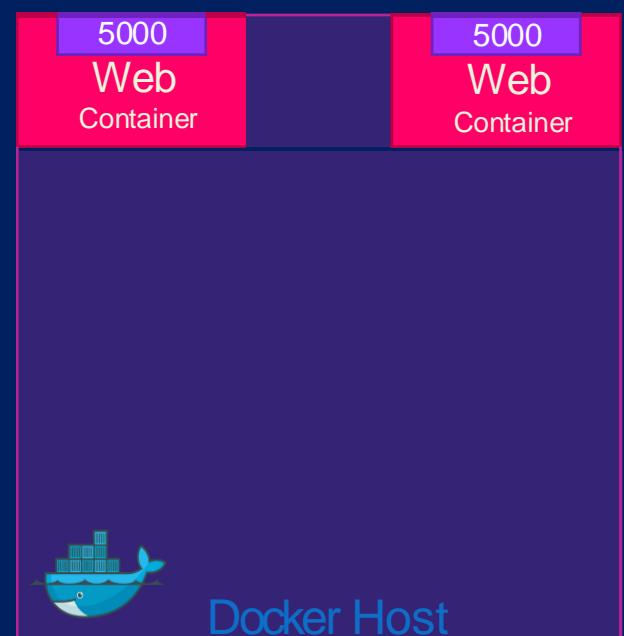
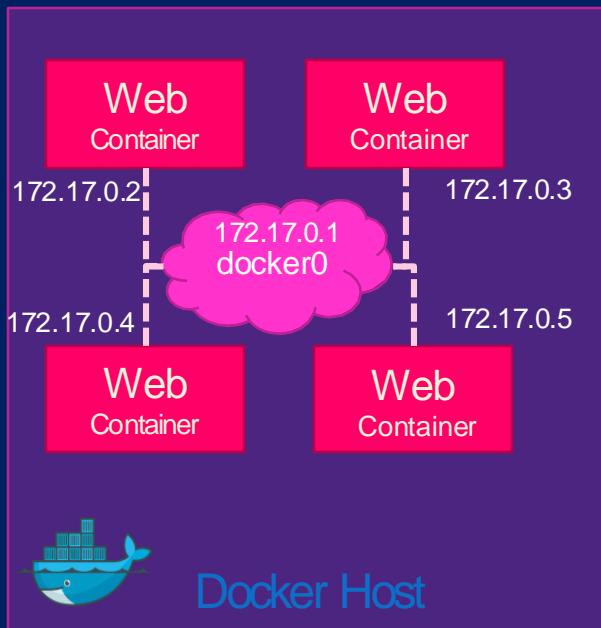
`docker run ubuntu`



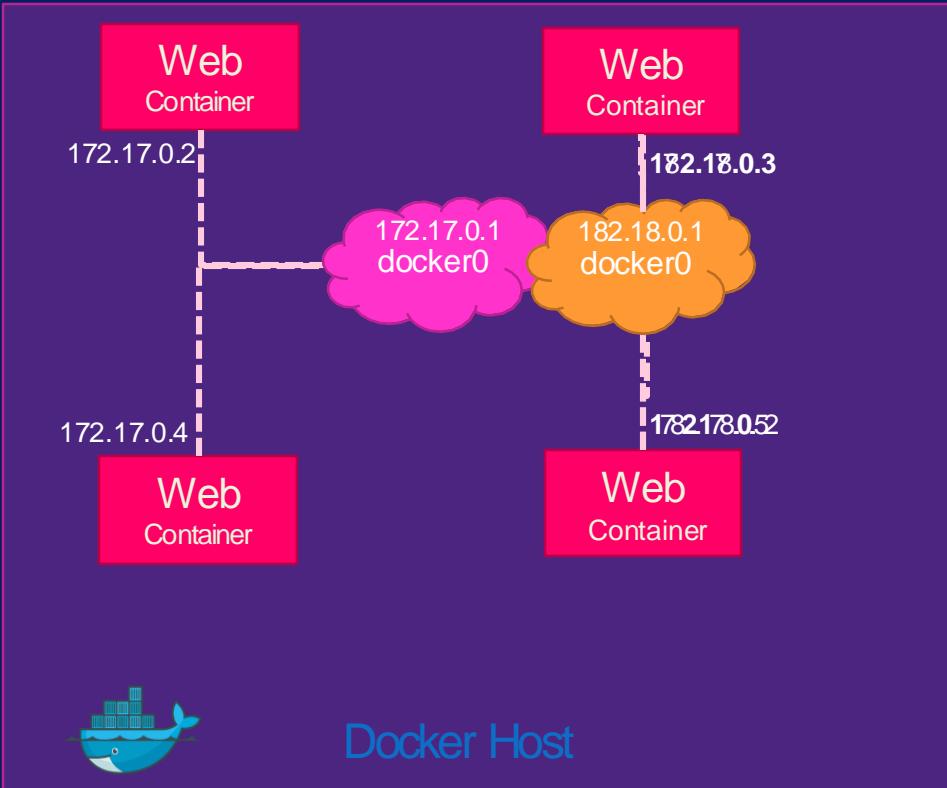
`docker run Ubuntu--network=none`



`docker run Ubuntu--network=host`



# User-defined networks



```
docker network create \
--driver bridge \
--subnet 182.18.0.0/16 custom-
isolated-network
```

```
docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
dba0fb9370fe	bridge	bridge	local
46d476b87cd9	customer-isolated-network	bridge	local
6de685cec1ce	docker_gwbridge	bridge	local
e29d188b4e47	host	host	local
mmrho7vsb9rm	ingress	overlay	swarm
d9f11695f0d6	none	null	local
d371b4009142	simplewebappdocker_default	bridge	local

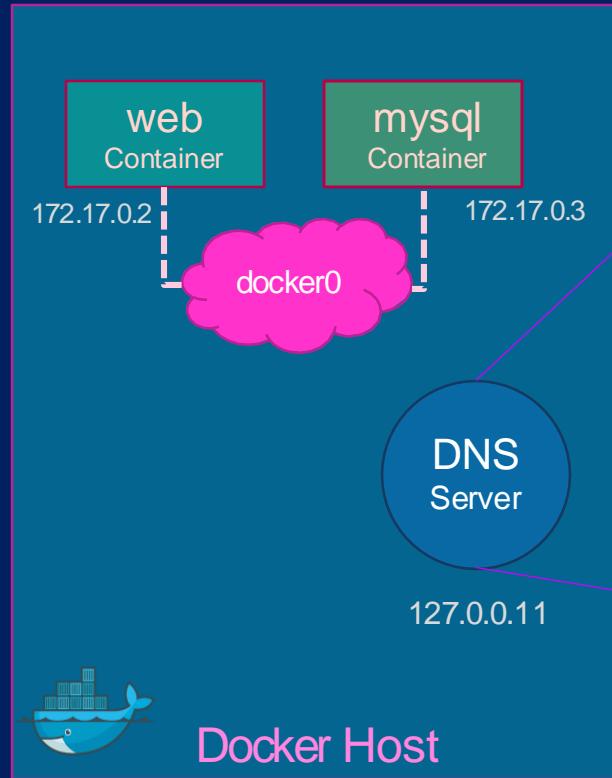
# Inspect Network

```
▶ docker inspect blissful_hopper
```

```
[  
 {  
   "Id": "35505f7810d17291261a43391d4b6c0846594d415ce4f4d0a6ffbf9cc5109048",  
   "Name": "/blissful_hopper",  
   "NetworkSettings": {  
     "Bridge":  
       "",  
       "Gateway": "172.17.0.1",  
       "IPAddress": "172.17.0.6",  
       "MacAddress": "02:42:ac:11:00:06",  
       "Networks": {  
         "bridge": {  
           "Gateway": "172.17.0.1",  
           "IPAddress": "172.17.0.6",  
           "MacAddress": "02:42:ac:11:00:06",  
         }  
       }  
     }  
   }  
 ]
```

# Embedded DNS

```
mysql.connect(    mysql    )
```



Host	IP
web	172.17.0.2
mysql	172.17.0.3

# LAB 3:

## Lab Docker Network with Subnet

This guide explains how to create a lab Docker network with a specific subnet using a busybox container.

1. Create a new Docker network named "lab\_network" with a specific subnet, e.g., 192.168.100.0/24, using the following command:

```
docker network create --subnet 192.168.100.0/24 lab_network
```

2. To confirm that the "lab\_network" has been created and to view its settings, run the following command:

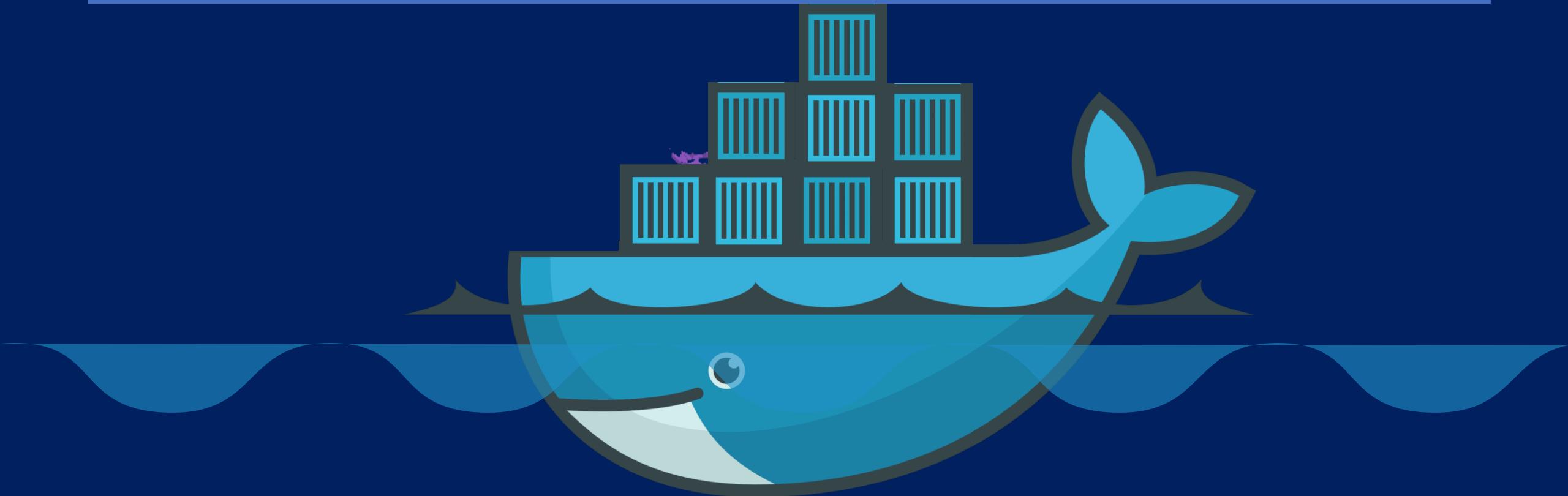
```
docker network inspect lab_network
```

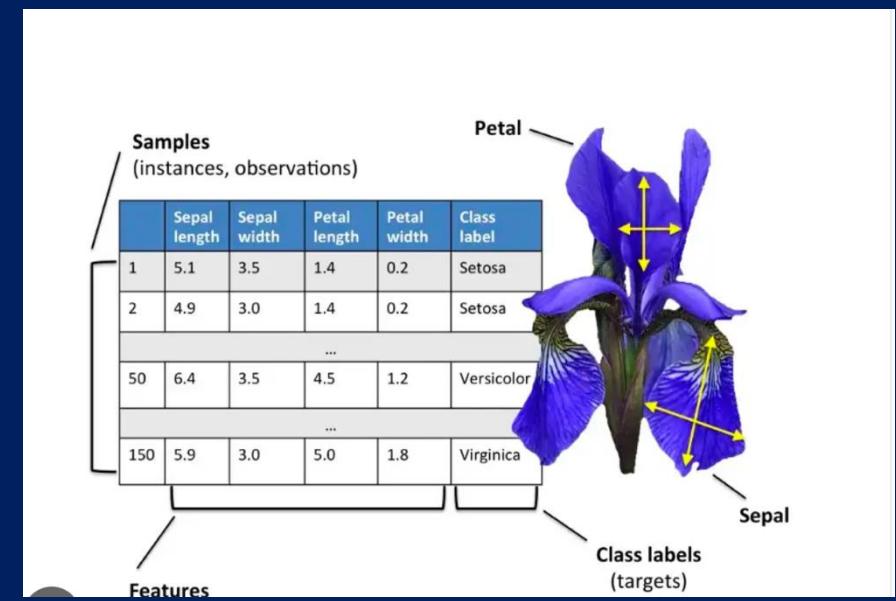
3. Run a new container with the busybox image and connect it to the "lab\_network" using the following command:

```
docker run --name busybox_container --network lab_network busybox
```

# LAB 4. ML With Gradio

---



**iris setosa**petal  
sepal**iris versicolor**petal  
sepal**iris virginica**petal  
sepal

<b>Id</b>	<b>SepalLengthCm</b>	<b>SepalWidthCm</b>	<b>PetalLengthCm</b>	<b>PetalWidthCm</b>	<b>Species</b>
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
13	4.8	3	1.4	0.1	Iris-setosa
14	4.3	3	1.1	0.1	Iris-setosa
15	5.8	4	1.2	0.2	Iris-setosa
16	5.7	4.4	1.5	0.4	Iris-setosa
17	5.4	3.9	1.3	0.4	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa
19	5.7	3.8	1.7	0.3	Iris-setosa

## Problem 1: Classification with the Iris Dataset

Objective: Learn how to classify the Iris dataset using `RandomForestClassifier` and `GradientBoostingClassifier`.

```
# Step 1: Setup and Import Libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report
```

1] ✓ 10.4s

```
# Step 2: Load the Iris Dataset
# Load the Iris dataset
iris_data = load_iris()
X_iris, y_iris = iris_data.data, iris_data.target

print(X_iris.shape, y_iris.shape)
```

3] ✓ 0.0s

(150, 4) (150,)

Generate + Code + Markdown

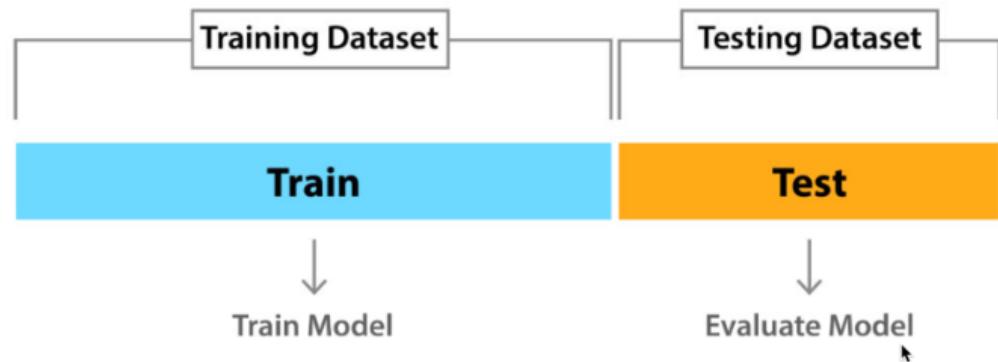
X\_iris[0:20]

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3]])
```

y\_iris[0:20]

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

## DATASET



```
# Step 3: Split the Data into Training and Test Sets
# Split Iris dataset
X_iris_train, X_iris_test, y_iris_train, y_iris_test = train_test_split(
    X_iris, y_iris, test_size=0.3, random_state=42)
```

```
# Step 4: Train the Classifiers
# Train classifiers on Iris dataset
rf_iris = RandomForestClassifier(random_state=42)
gb_iris = GradientBoostingClassifier(random_state=42)

rf_iris.fit(X_iris_train, y_iris_train)
gb_iris.fit(X_iris_train, y_iris_train)
```

```
GradientBoostingClassifier(random_state=42)
```

```
# Step 5: Evaluate the Models
# Evaluate on Iris dataset
y_iris_pred_rf = rf_iris.predict(X_iris_test)
y_iris_pred_gb = gb_iris.predict(X_iris_test)

print('Iris Dataset - Random Forest')
print(f'Accuracy: {accuracy_score(y_iris_test, y_iris_pred_rf):.2f}')
print(classification_report(y_iris_test, y_iris_pred_rf))
```

```
# Step 5: Evaluate the Models
# Evaluate on Iris dataset
y_iris_pred_rf = rf_iris.predict(X_iris_test)
y_iris_pred_gb = gb_iris.predict(X_iris_test)

print('Iris Dataset - Random Forest')
print(f'Accuracy: {accuracy_score(y_iris_test, y_iris_pred_rf):.2f}')
print(classification_report(y_iris_test, y_iris_pred_rf))

print('Iris Dataset - Gradient Boosting')
print(f'Accuracy: {accuracy_score(y_iris_test, y_iris_pred_gb):.2f}')
print(classification_report(y_iris_test, y_iris_pred_gb))
```

Iris Dataset - Random Forest

Accuracy: 1.00

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

Iris Dataset - Gradient Boosting

Accuracy: 1.00

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

## Project Overview

- **Backend:** FastAPI serving ML predictions
  - Port: 8087
  - Models: RandomForestClassifier, GradientBoostingClassifier
  - Dataset: Iris flower dataset
- **Frontend:** Gradio interface for input and visualization
  - Port: 8085
  - Features: Sliders for measurements, dual model predictions
- **Containerization:** Docker with separate images for frontend and backend

### Visual Representation (Text-Based Diagram)



## Project Structure

```
student-lab/
├── backend/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── app.py
└── frontend/
    ├── Dockerfile
    ├── requirements.txt
    └── interface.py
README.md
```

## Frontend: frontend/interface.py

```
import gradio as gr
import requests

# Backend URL (adjusted for Docker linking)
BACKEND_URL = "http://backend:8087/predict"

def predict_iris(sepal_length, sepal_width, petal_length, petal_width):
    # Prepare data for API call
    data = {
        "sepal_length": sepal_length,
        "sepal_width": sepal_width,
        "petal_length": petal_length,
        "petal_width": petal_width
    }

    # Make request to backend
    try:
        response = requests.post(BACKEND_URL, json=data)
        response.raise_for_status()
        predictions = response.json()
        return (
            f"Random Forest: {predictions['random_forest']}",
            f"Gradient Boosting: {predictions['gradient_boosting']}"
        )
    except Exception as e:
        return f"Error: {str(e)}", f"Error: {str(e)}"

# Gradio interface
interface = gr.Interface(
    fn=predict_iris,
    inputs=[
        gr.Slider(4.0, 8.0, step=0.1, label="Sepal Length (cm)"),
        gr.Slider(2.0, 4.5, step=0.1, label="Sepal Width (cm)"),
        gr.Slider(1.0, 7.0, step=0.1, label="Petal Length (cm)"),
        gr.Slider(0.1, 2.5, step=0.1, label="Petal Width (cm)")
    ],
    outputs=[
        gr.Textbox(label="Random Forest Prediction"),
        gr.Textbox(label="Gradient Boosting Prediction")
    ],
    title="Iris Species Classifier",
    description="Adjust the sliders to input flower measurements and see predictions from two ML models."
)

if __name__ == "__main__":
    interface.launch(server_name="0.0.0.0", server_port=8085)
```

## Backend: backend/app.py

```
from fastapi import FastAPI
from pydantic import BaseModel
import numpy as np
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

app = FastAPI()

# Load and train models at startup
iris = load_iris()
X, y = iris.data, iris.target

# Train RandomForestClassifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X, y)

# Train GradientBoostingClassifier
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X, y)

# Input data model
class IrisInput(BaseModel):
    sepal_length: float
    sepal_width: float
    petal_length: float
    petal_width: float

@app.get("/health")
def health_check():
    return {"status": "healthy"}

@app.post("/predict")
def predict(data: IrisInput):
    # Convert input to array
    input_data = np.array([[data.sepal_length, data.sepal_width,
                           data.petal_length, data.petal_width]])

    # Get predictions
    rf_pred = rf_model.predict(input_data)[0]
    gb_pred = gb_model.predict(input_data)[0]

    # Map predictions to Iris species
    species = {0: "setosa", 1: "versicolor", 2: "virginica"}

    return {
        "random_forest": species[rf_pred],
        "gradient_boosting": species[gb_pred]
    }

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8087)
```

## frontend/requirements.txt

```
gradio  
requests
```

## Frontend: frontend/Dockerfile

```
FROM python:3.9-slim  
  
WORKDIR /app  
  
COPY requirements.txt .  
RUN pip install --no-cache-dir -r requirements.txt  
  
COPY interface.py .  
  
EXPOSE 8085  
  
CMD ["python", "interface.py"]
```

## Backend: backend/requirements.txt

```
uvicorn  
fastapi  
scikit  
numpy  
pydantic
```

## Backend: backend/Dockerfile

```
FROM python:3.9-slim  
  
WORKDIR /app  
  
COPY requirements.txt .  
RUN pip install --no-cache-dir -r requirements.txt  
  
COPY app.py .  
  
EXPOSE 8087  
  
CMD ["python", "app.py"]
```

Build the backend image:

```
cd backend
docker build -t iris-backend .
cd ..
```



Build the frontend image:

```
cd frontend
docker build -t iris-frontend .
cd ..
```



### 3. Run the Containers

Run the backend container:

```
docker run -d --name backend -p 8087:8087 iris-backend
```



Run the frontend container (linked to backend):

```
docker run -d --name frontend -p 8085:8085 --link backend iris-frontend
```



# Iris Species Classifier

Adjust the sliders to input flower measurements and see predictions from two ML models.

Sepal Length (cm) 5.1

Sepal Width (cm) 3.5

Petal Length (cm) 3.6

Petal Width (cm) 0.6

**Clear** **Submit**

## Random Forest Prediction

Random Forest: setosa

## Gradient Boosting Prediction

Gradient Boosting: versicolor

**Flag**

# Docker compose

```
docker run yourname/simple-webapp
```

```
docker run mongodb
```

```
docker run redis:alpine
```

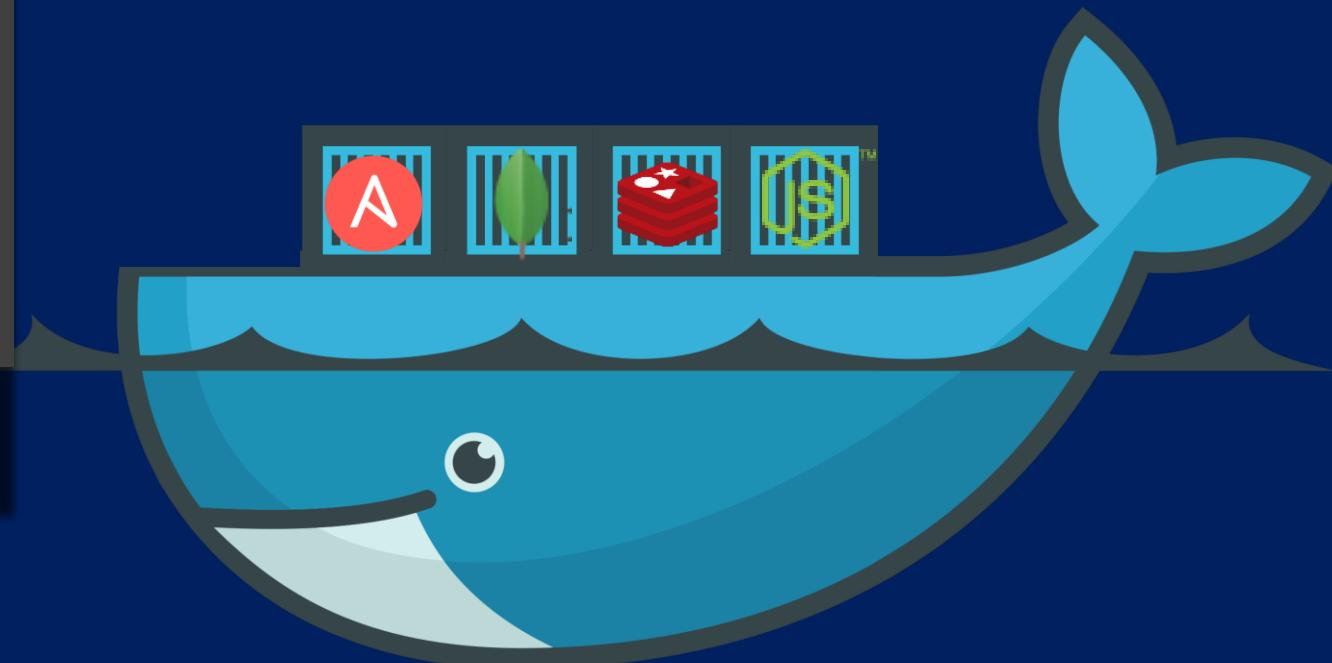
```
docker run ansible
```

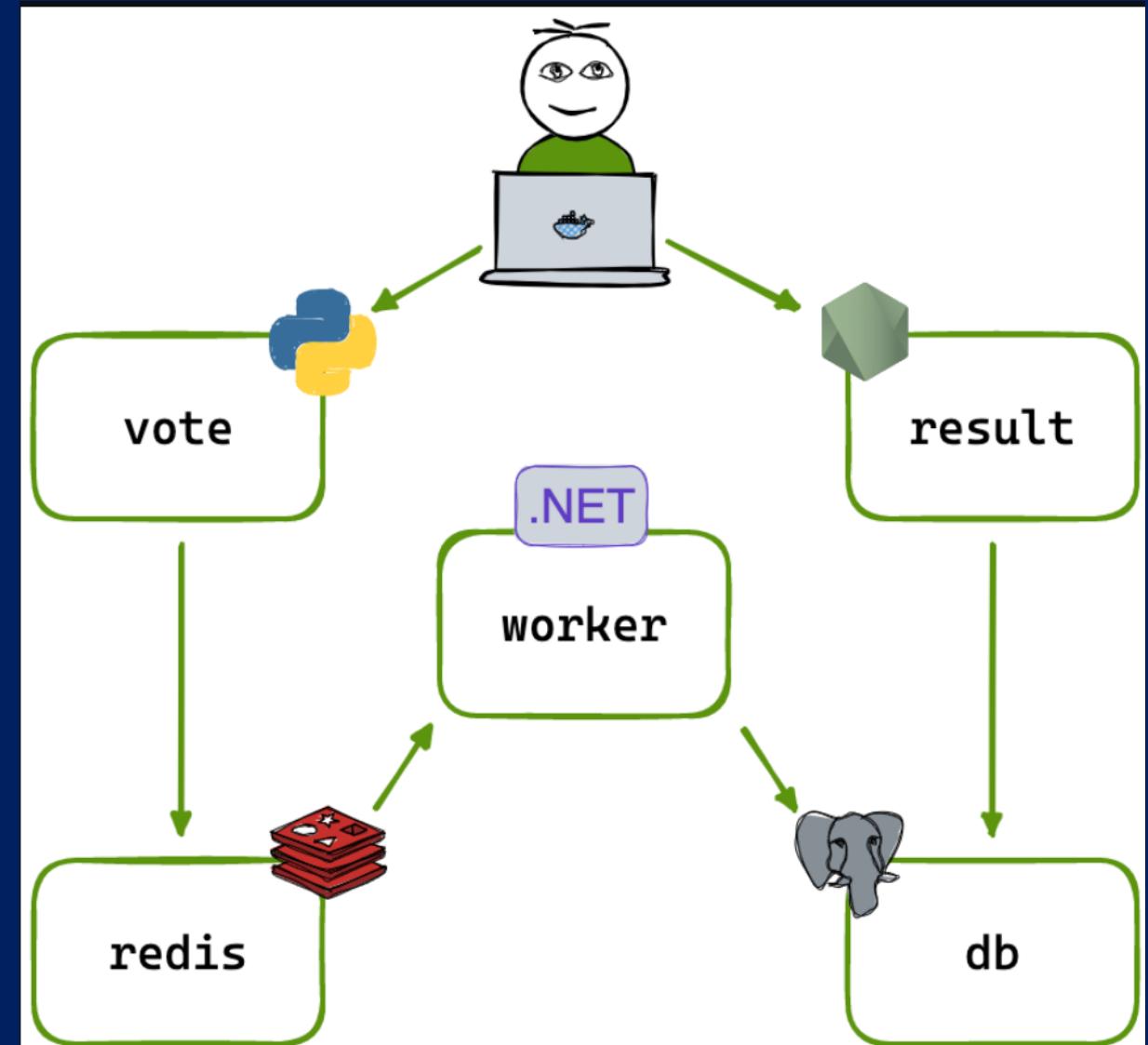
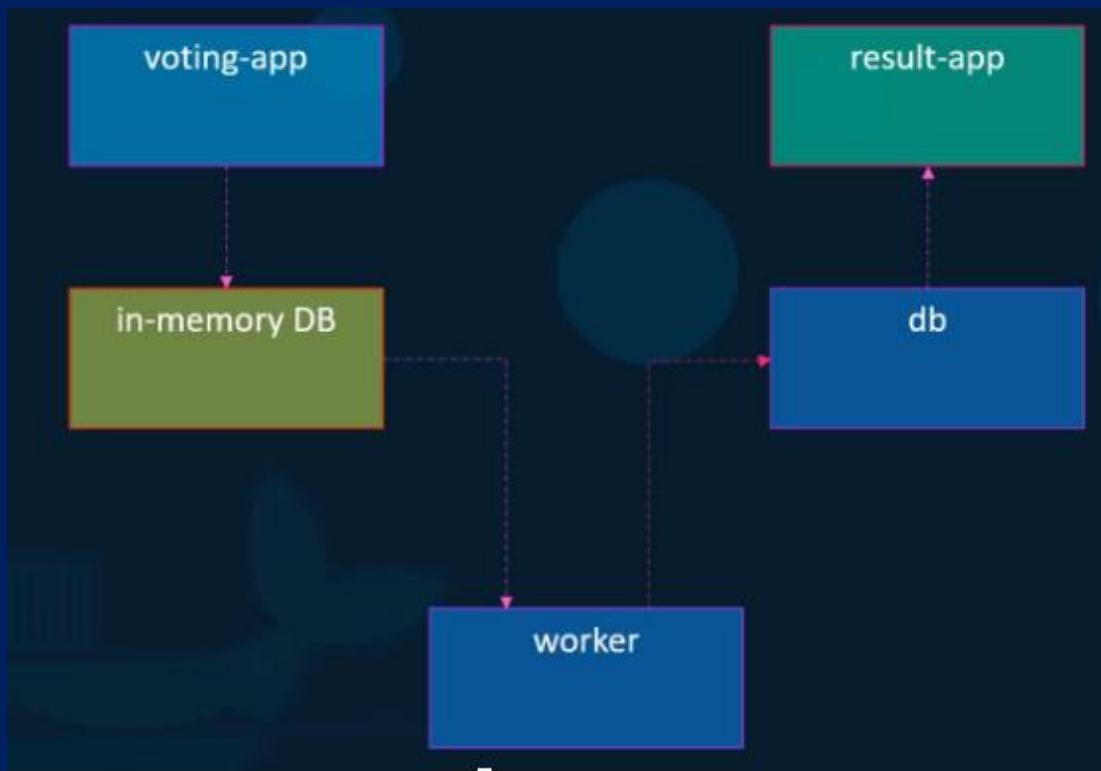
```
docker-compose.yml
```

```
services:  
  web:  
    image: "yourname/simple-webapp"  
  database:  
    image: "mongodb"  
  messaging:  
    image: "redis:alpine"  
  orchestration:  
    image: "ansible"
```

```
docker-compose up
```

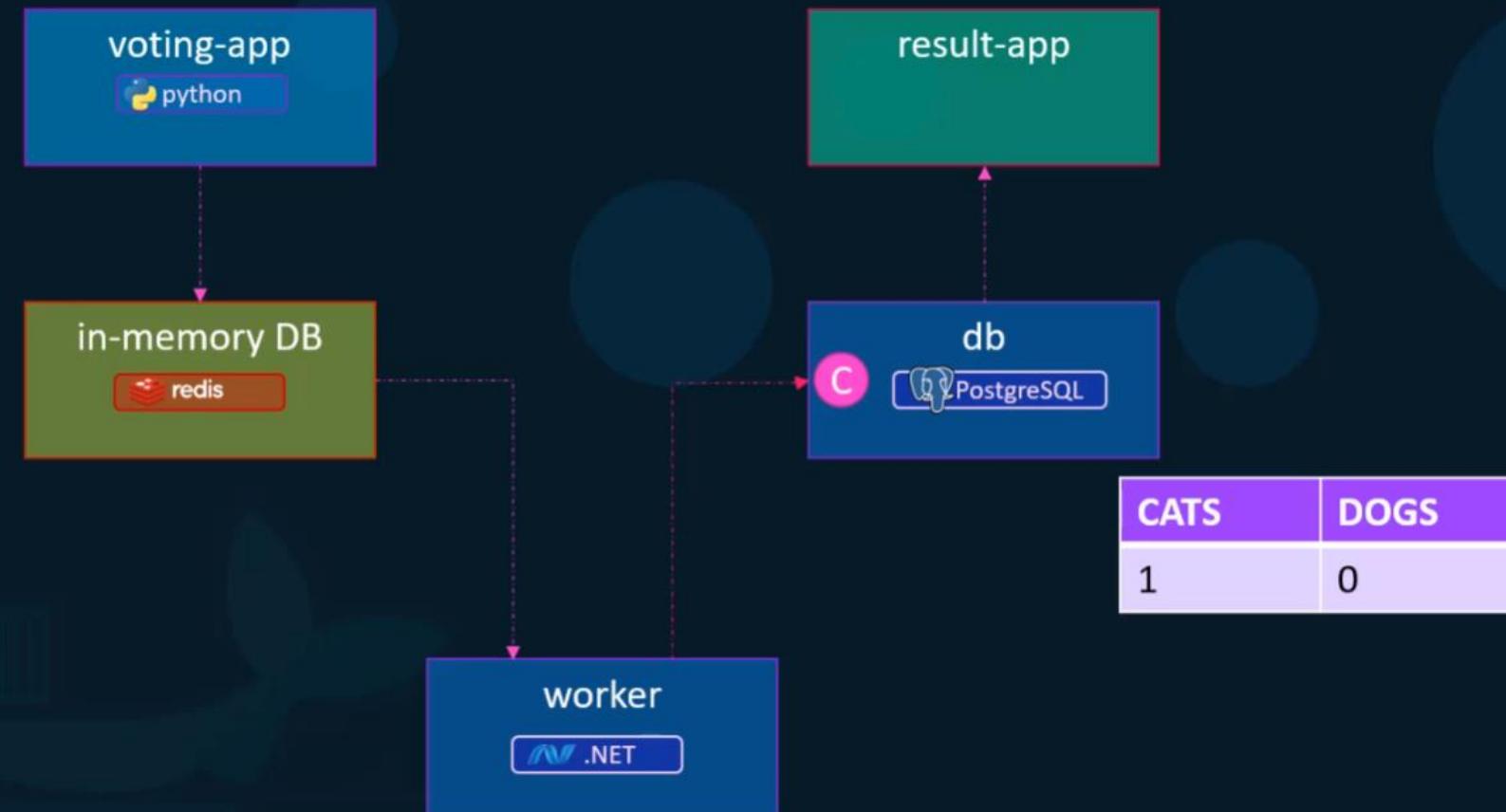
Public Docker registry - dockerhub



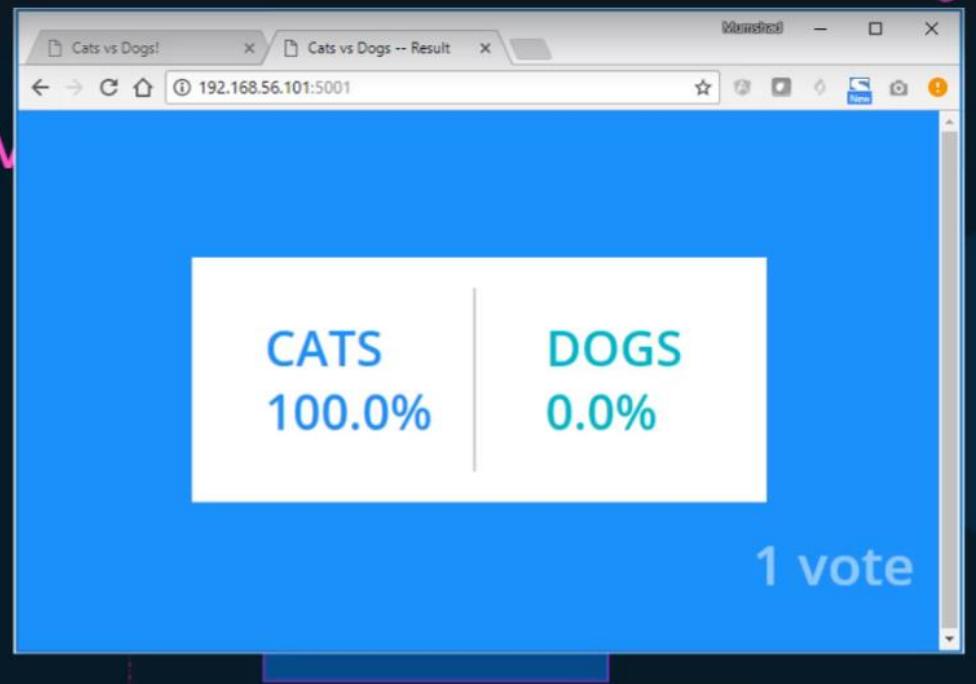
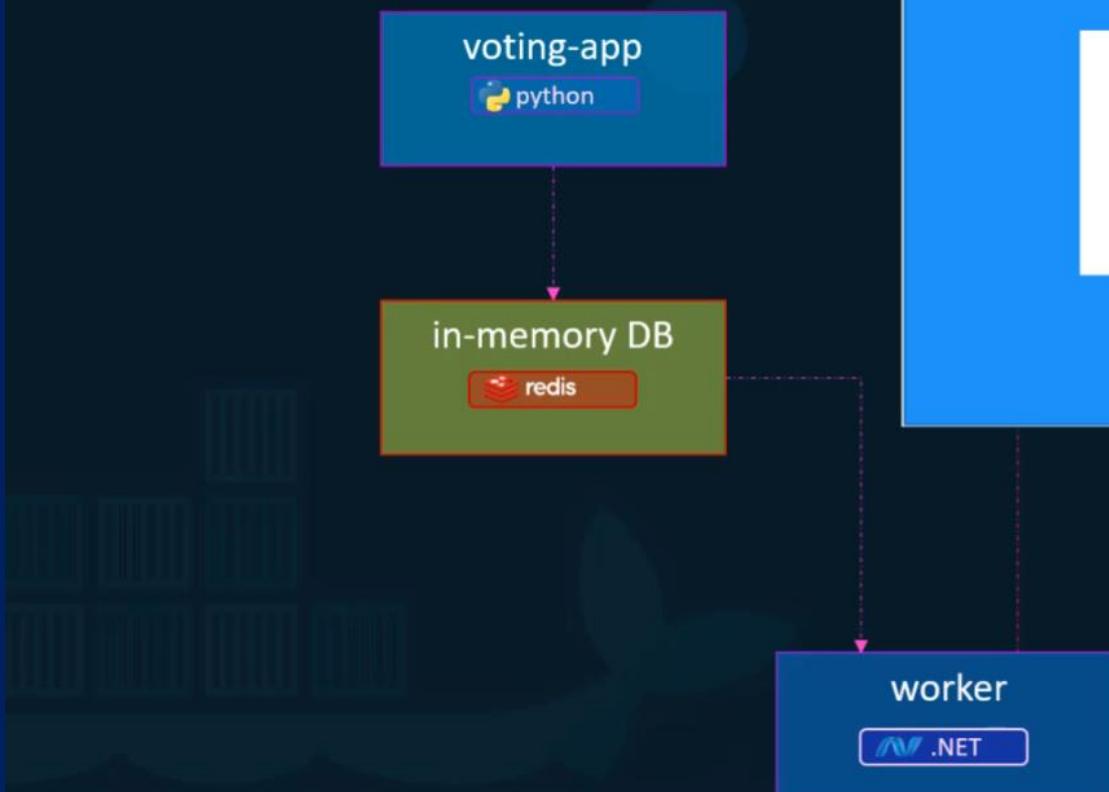


- A front-end web app in [Python](#) which lets you vote between two options
- A [Redis](#) which collects new votes
- A [.NET](#) worker which consumes votes and stores them in...
- A [Postgres](#) database backed by a Docker volume
- A [Node.js](#) web app which shows the results of the voting in real time

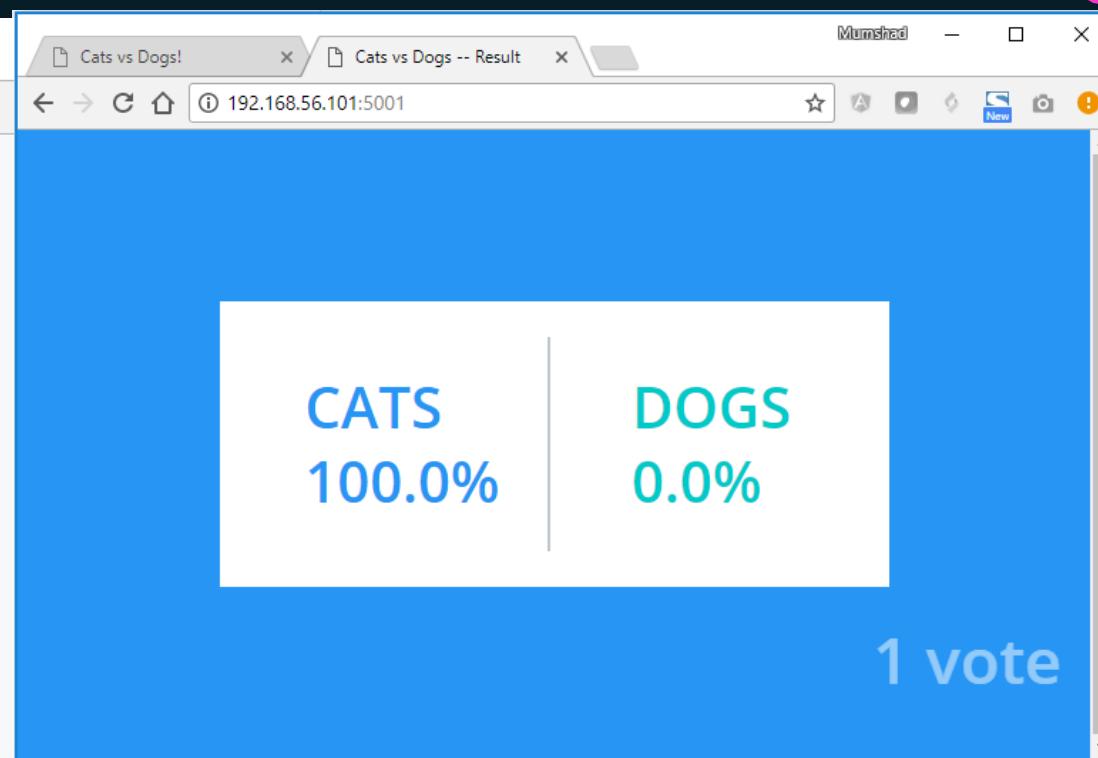
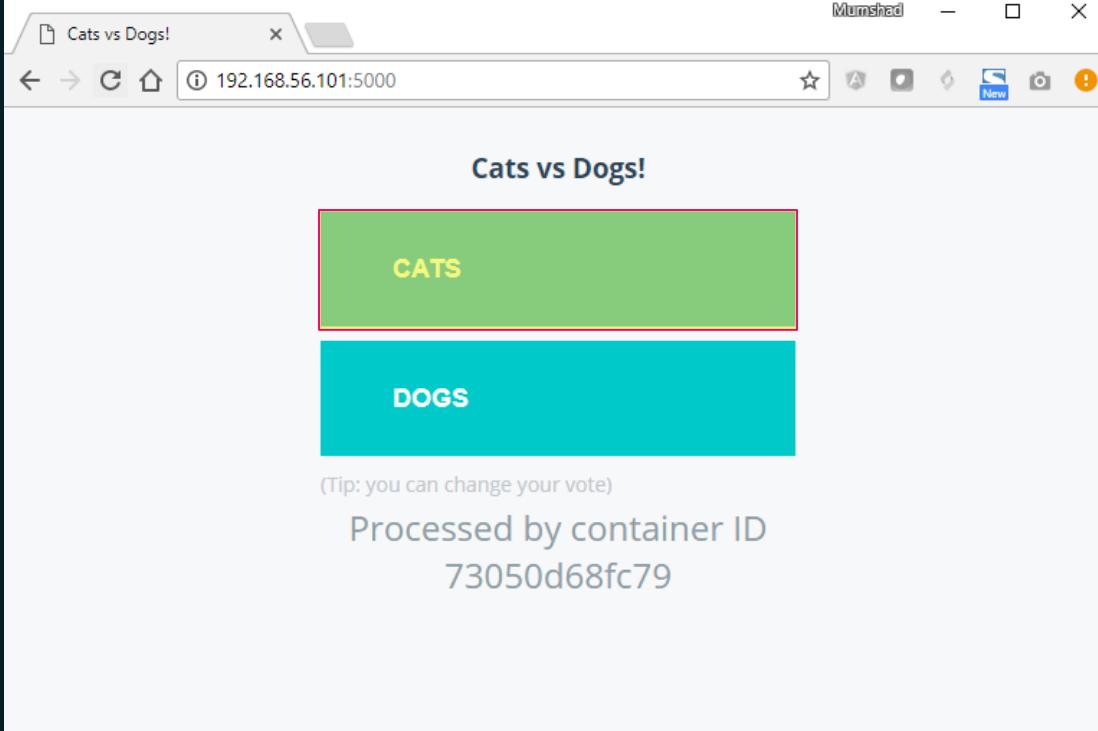
# Sample application – voting application



# Sample application – voting



CATS	DOGS
1	0



CATS	DOGS
1	0



# docker run

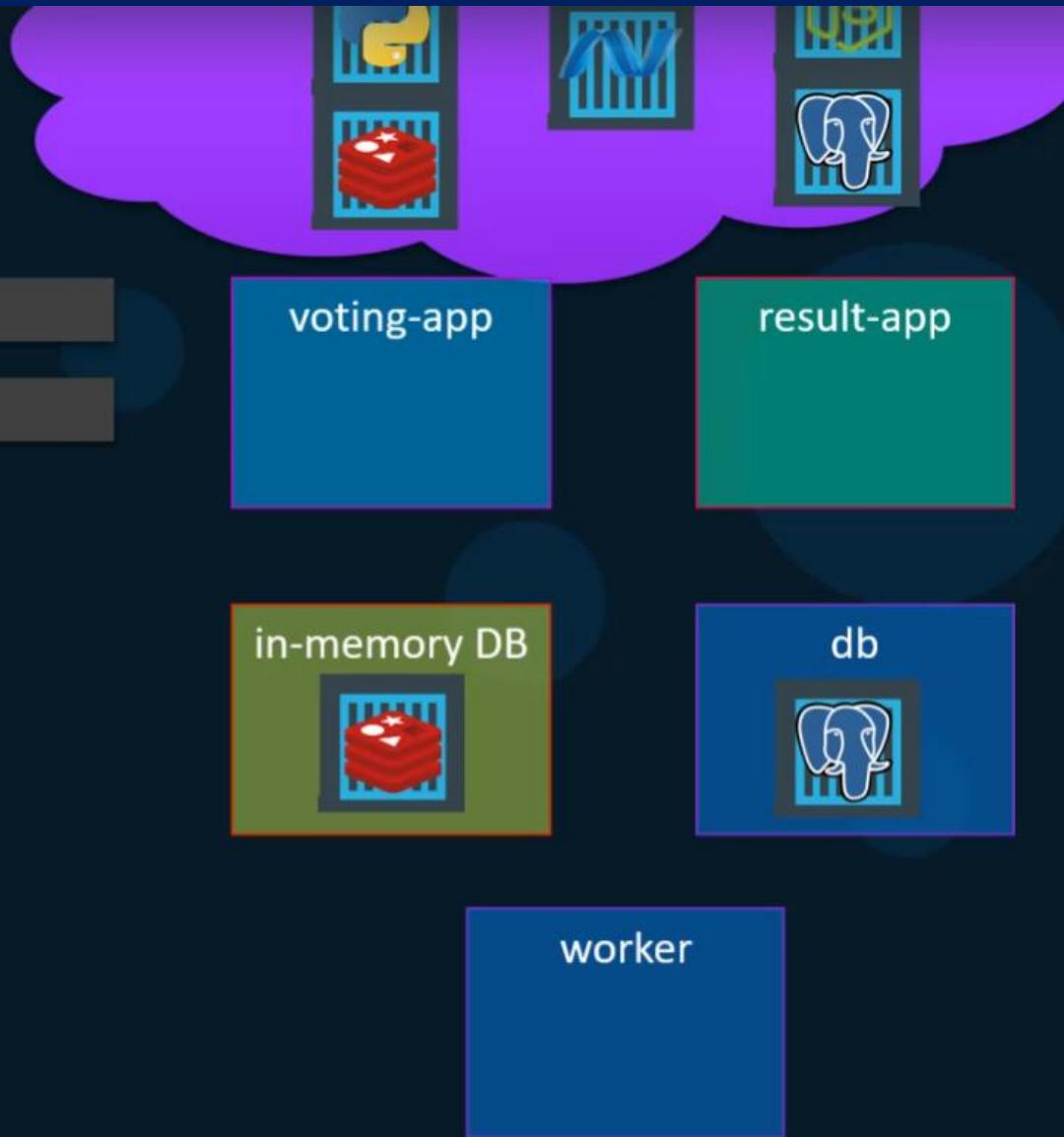
```
docker run -d --name=redis redis
```



# docker run

```
docker run -d --name=redis redis
```

```
docker run -d --name=db postgres
```



# docker run

```
docker run -d --name=redis redis
```

```
docker run -d --name=db postgres
```

```
docker run -d --name=vote -p 5000:80 voting-app
```



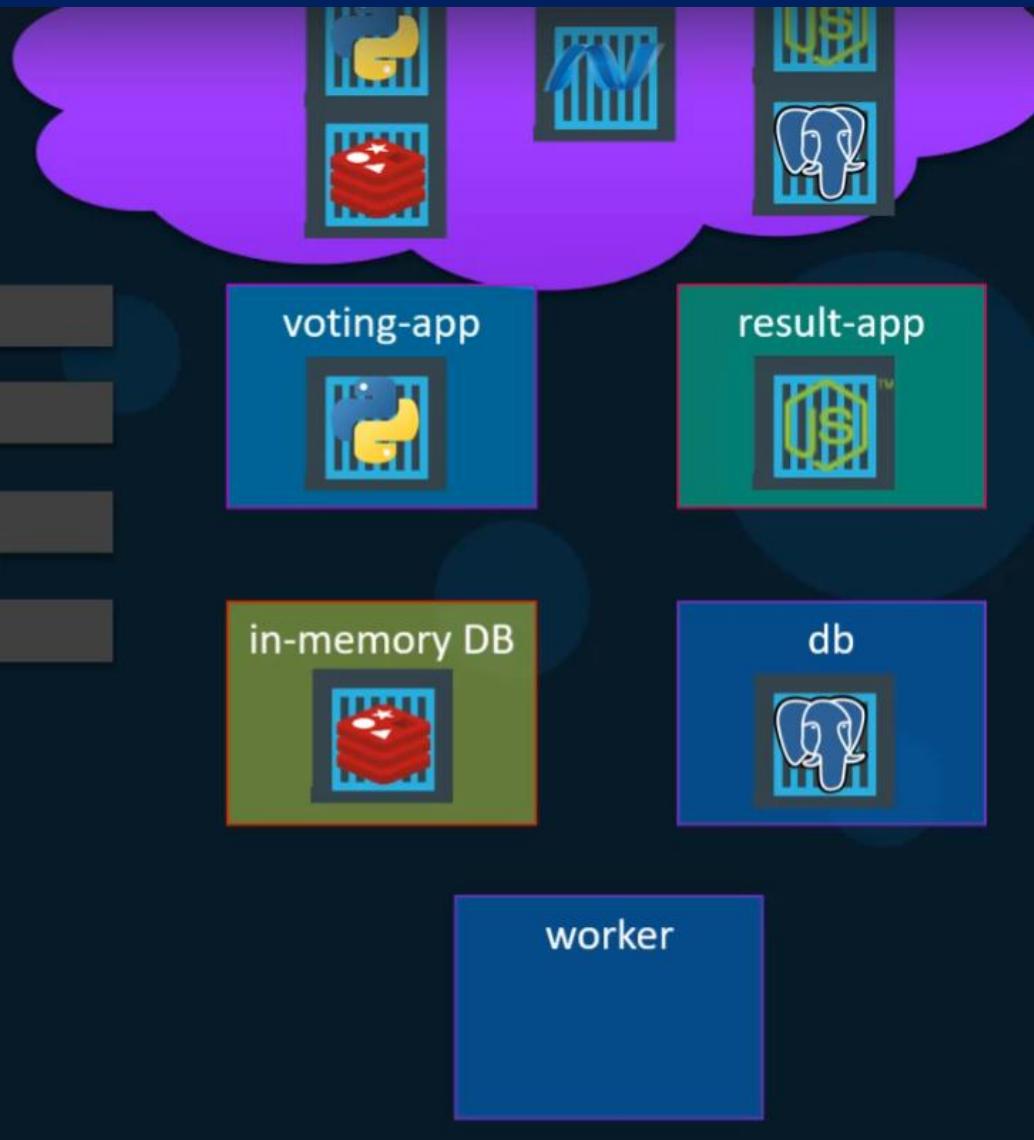
# docker run

```
docker run -d --name=redis redis
```

```
docker run -d --name=db postgres
```

```
docker run -d --name=vote -p 5000:80 voting-app
```

```
docker run -d --name=result -p 5001:80 result-app
```



# docker run

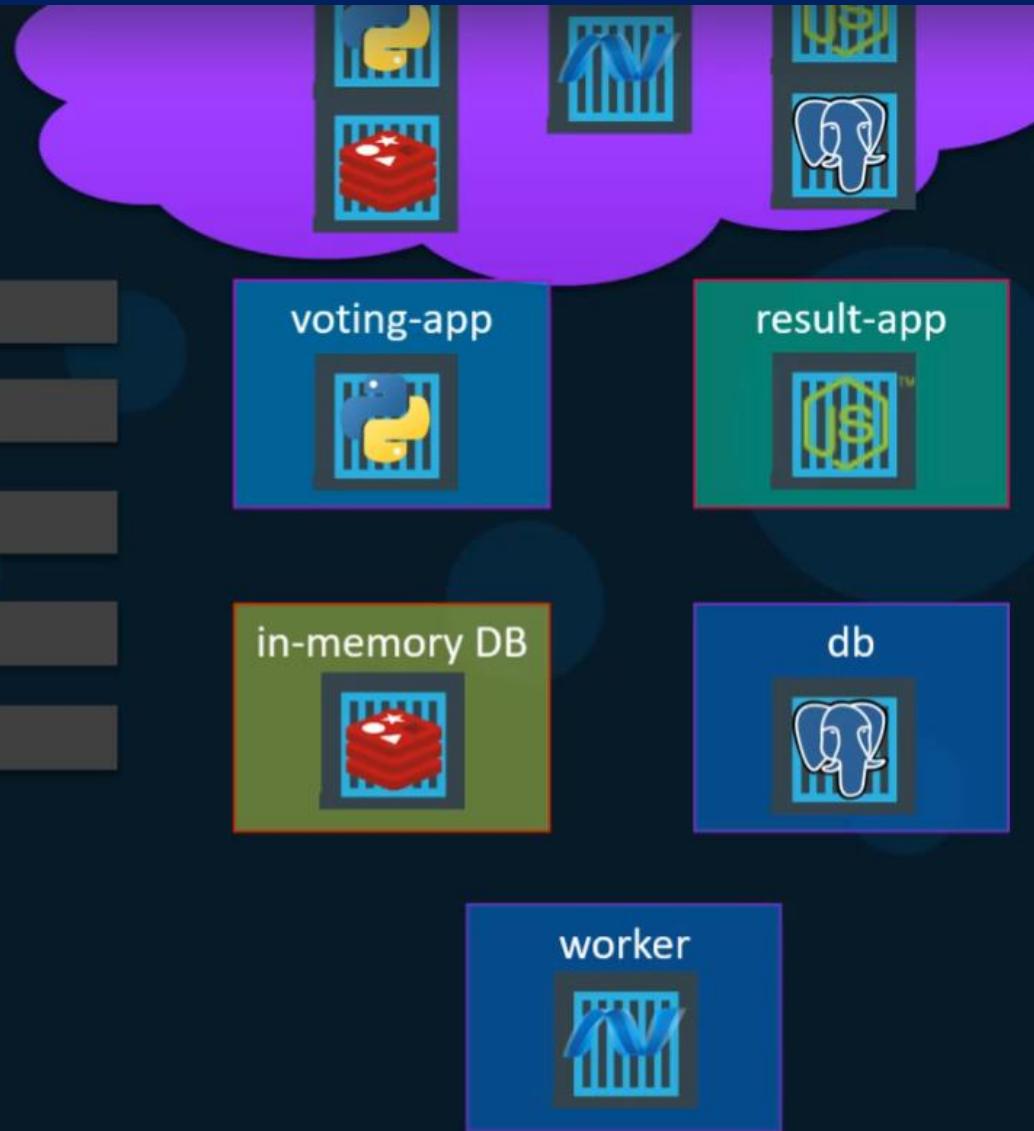
```
docker run -d --name=redis redis
```

```
docker run -d --name=db postgres
```

```
docker run -d --name=vote -p 5000:80 voting-app
```

```
docker run -d --name=result -p 5001:80 result-app
```

```
docker run -d --name=worker worker
```



# docker run

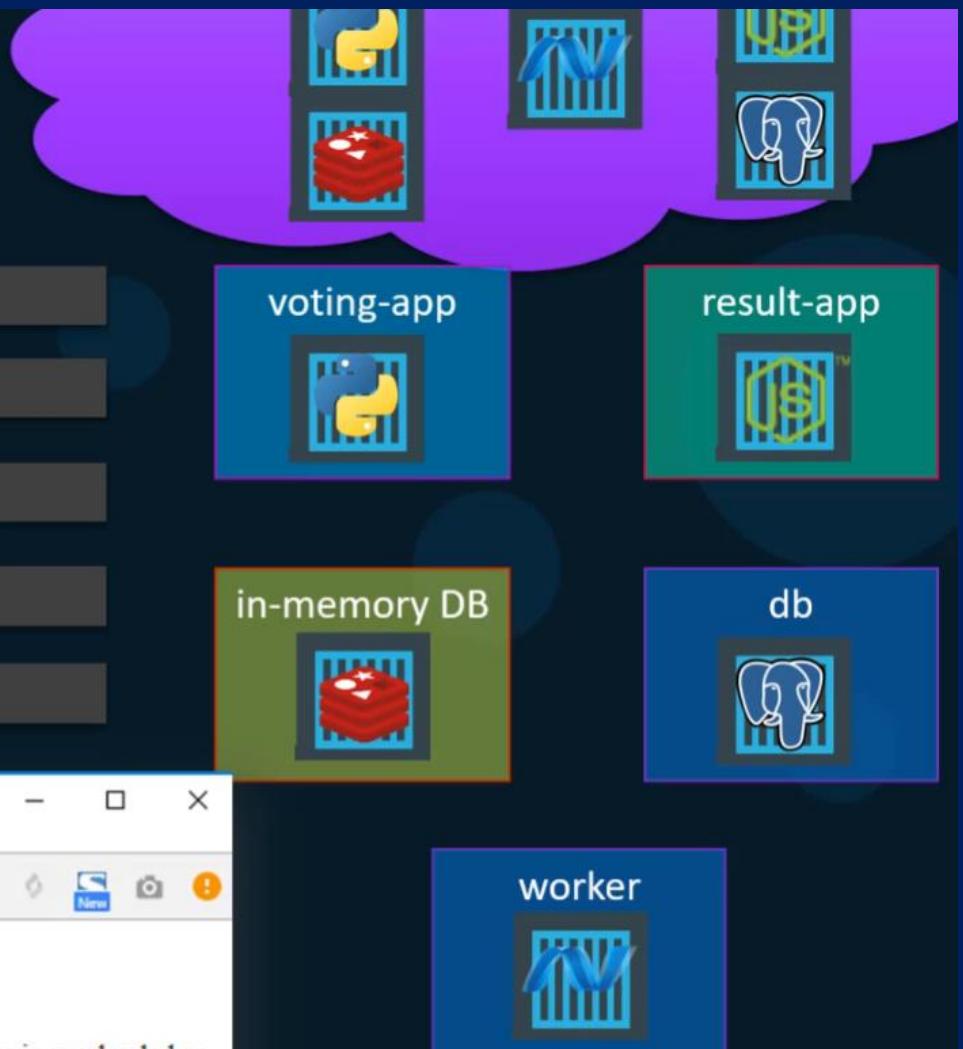
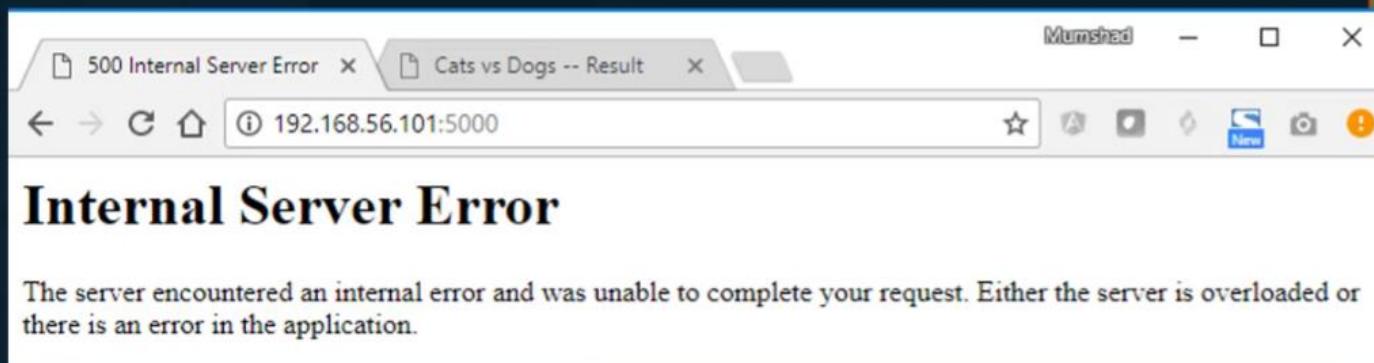
```
docker run -d --name=redis redis
```

```
docker run -d --name=db postgres
```

```
docker run -d --name=vote -p 5000:80 voting-app
```

```
docker run -d --name=result -p 5001:80 result-app
```

```
docker run -d --name=worker worker
```



# docker run --links

```
docker run -d --name=redis redis
```

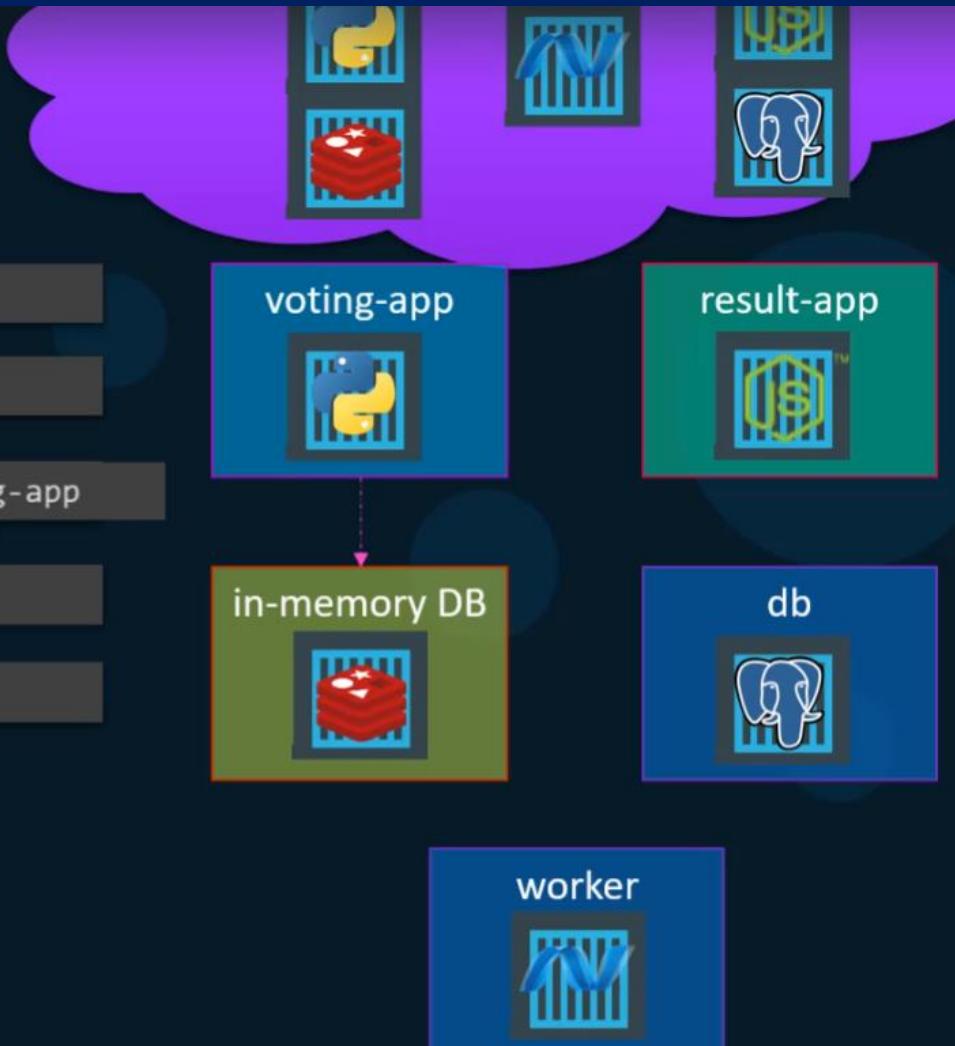
```
docker run -d --name=db postgres
```

```
docker run -d --name=vote -p 5000:80 --link redis:redis voting-app
```

```
docker run -d --name=result -p 5001:80 result-app
```

```
docker run -d --name=worker worker
```

```
def get_redis():
    if not hasattr(g, 'redis'):
        g.redis = Redis(host="redis", db=0, socket_timeout=5)
    return g.redis
```



# docker run --links

```
docker run -d --name=redis redis
```

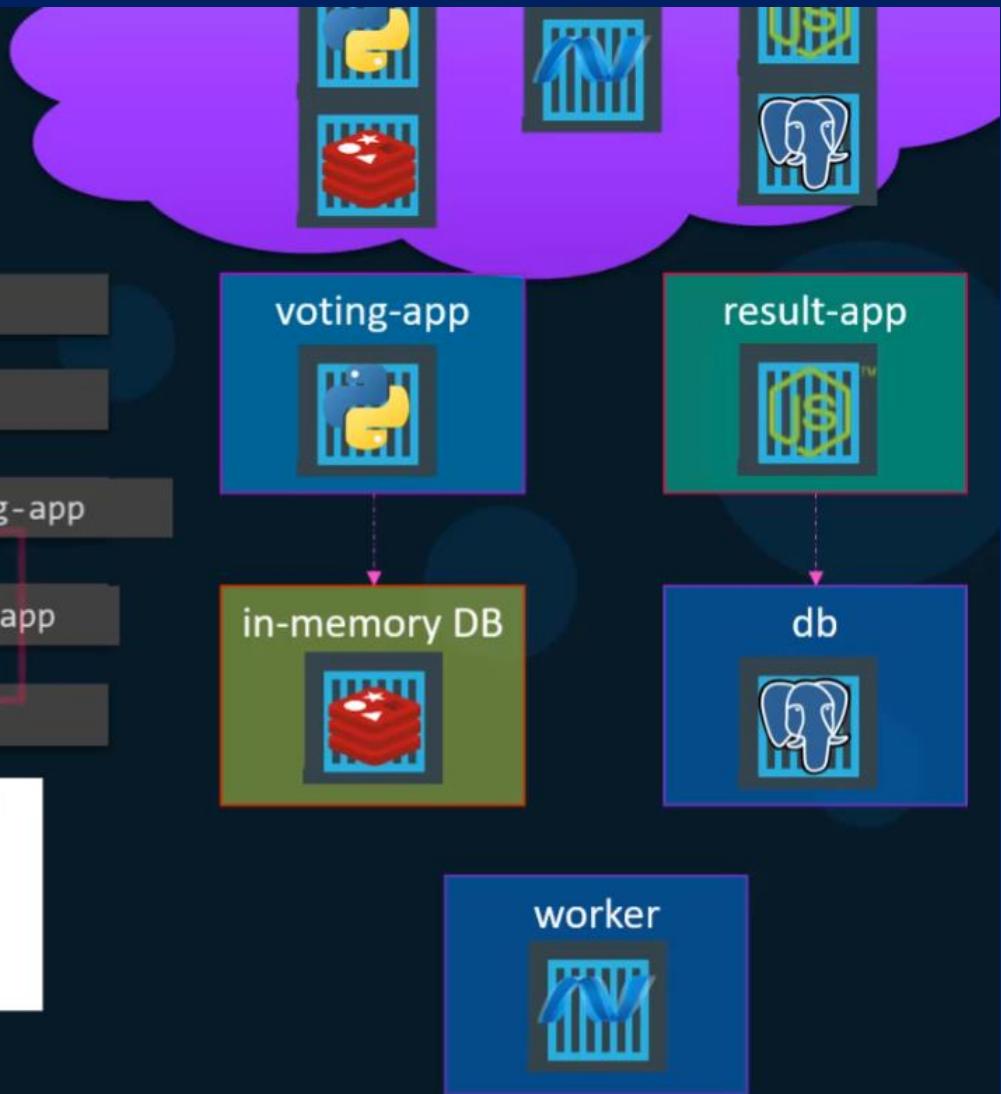
```
docker run -d --name=db postgres
```

```
docker run -d --name=vote -p 5000:80 --link redis:redis voting-app
```

```
docker run -d --name=result -p 5001:80 --link db:db result-app
```

```
docker run -d --name=worker worker
```

```
pg.connect('postgres://postgres@db/postgres', function(err, client, done) {  
  if (err) {  
    console.error("Waiting for db");  
  }  
  callback(err, client);  
});
```



# docker run --links

```
docker run -d --name=redis redis
```

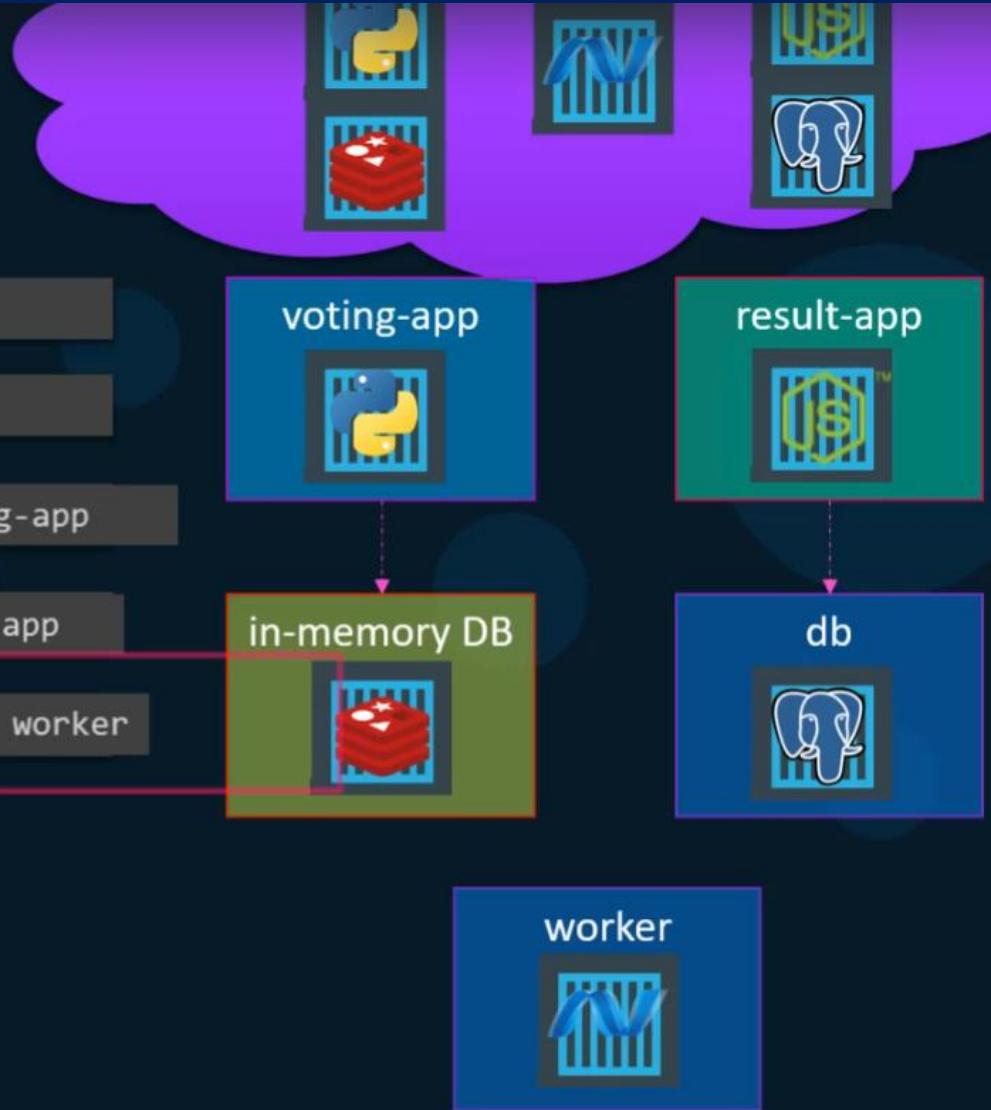
```
docker run -d --name=db postgres
```

```
docker run -d --name=vote -p 5000:80 --link redis:redis voting-app
```

```
docker run -d --name=result -p 5001:80 --link db:db result-app
```

```
docker run -d --name=worker --link db:db --link redis:redis worker
```

```
try {  
    Jedis redis = connectToRedis("redis");  
    Connection dbConn = connectToDB("db");  
  
    System.out.println("Watching vote queue");
```



# Docker compose

```
docker run -d --name=redis redis
```

```
docker run -d --name=db postgres:9.4
```

```
docker run -d --name=vote -p 5000:80 --link redis:redis voting-app
```

```
docker run -d --name=result -p 5001:80 --link db:db result-app
```

```
docker run -d --name=worker --link db:db --link redis:redis worker
```

```
docker-compose.yml
```

```
redis:  
  image: redis  
db:  
  image: postgres:9.4  
vote:  
  image: voting-app  
  ports:  
    - 5000:80  
  links:  
    - redis  
result:  
  image: result-app  
  ports:  
    - 5001:80  
  links:  
    - db  
worker:  
  image: worker  
  links:  
    - redis  
    - db
```

# Docker compose - build

docker-compose.yml

```
redis:  
  image: redis  
  
db:  
  image: postgres:9.4 vote:  
  
    image: voting-app
```

```
  ports:  
    - 5000:80
```

```
  links:  
    - redis
```

```
result:  
  image: result  
  ports:  
    - 5001:80
```

```
  links:  
    - db
```

```
worker:  
  image: worker  
  links:  
    - db  
    - redis
```

docker-compose.yml

```
redis:  
  image: redis  
  
db:  
  image: postgres:9.4 vote:  
  
    build: ./vote
```

```
  ports:  
    - 5000:80
```

```
  links:  
    - redis
```

```
result:  
  build: ./result  
  ports:  
    - 5001:80
```

```
  links:  
    - db
```

```
worker:  
  build: ./worker  
  links:  
    - db  
    - redis
```

SOFTWARE-DEVELOPMENT-TOOLS-AND-ENVIRONMENTS / Week10 / Lab\_docker\_compose / vote /

Name	Last commit message
..	0
static/stylesheets	0
templates	0
Dockerfile	0
app.py	0
requirements.txt	0

# Docker compose - versions

docker-compose.yml

```
redis:  
    image: redis  
  
db:  
    image: postgres:9.4  
  
vote:  
    image: voting-app  
    ports:  
        - 5000:80  
    links:  
        - redis
```

version: 1

docker-compose.yml

```
version: 2  
  
services:  
    redis:  
        image: redis  
  
    db:  
        image: postgres:9.4  
  
    vote:  
        image: voting-app  
        ports:  
            - 5000:80  
        depends_on:  
            - redis
```

version: 2

docker-compose.yml

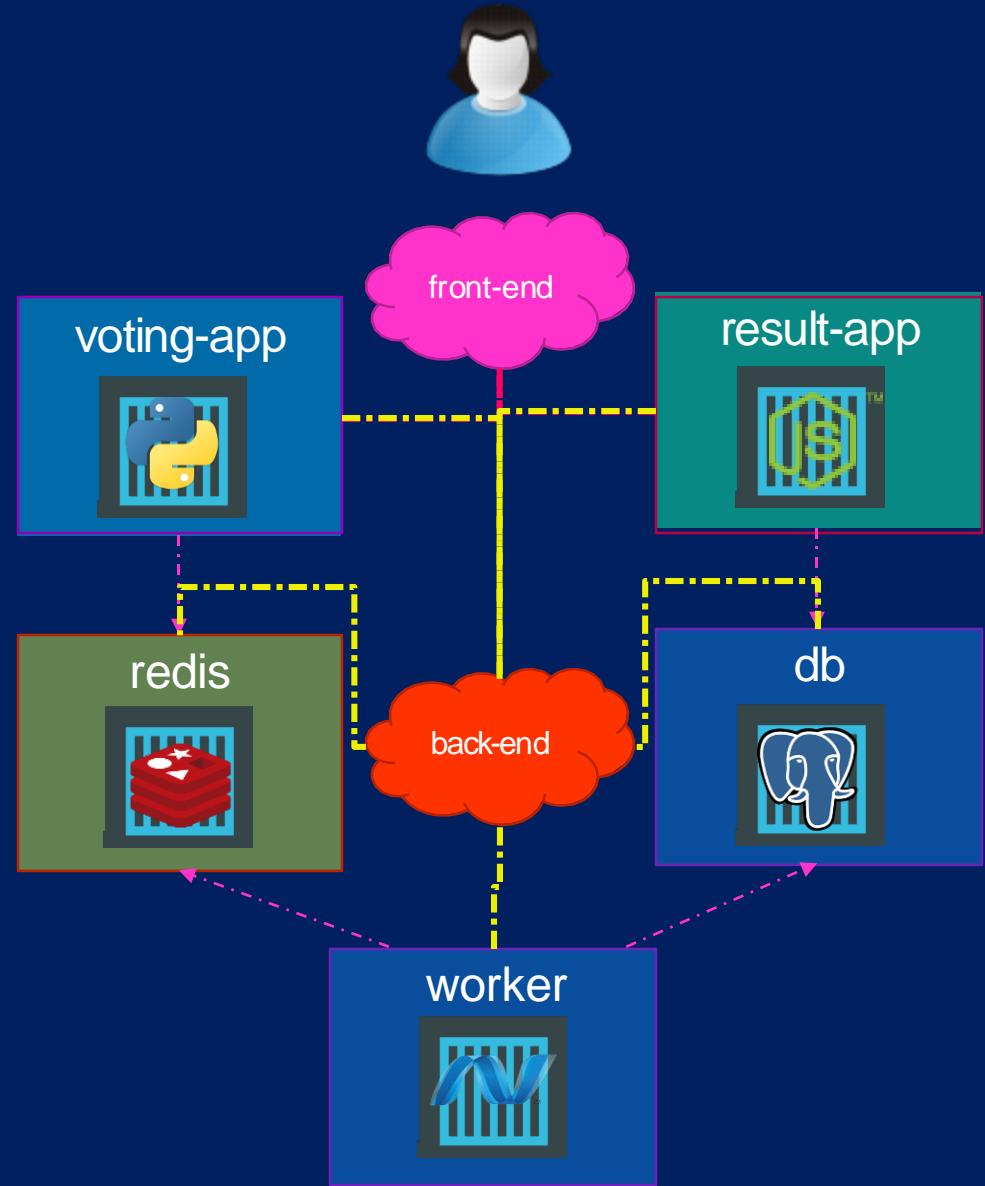
```
version: 3  
  
services:  
    redis:  
        image: redis  
  
    db:  
        image: postgres:9.4  
  
    vote:  
        image: voting-app  
        ports:  
            - 5000:80
```

version: 3

# Docker compose

docker-compose.yml

```
version: 2
services:
  redis:
    image: redis
    networks:
      - back-end
  db:
    image: postgres:9.4
    networks:
      - back-end
  vote:
    image: voting-app
    networks:
      - front-end
      - back-end
  result:
    image: result
    networks:
      - front-end
      - back-end
networks:
  front-end:
  back-end:
```



## **docker-compose up**

This command is used to start the containers defined in the docker-compose.yml file. If the containers don't exist, they will be created.

```
docker-compose up
```

## **docker-compose down**

This command is used to stop and remove the containers defined in the docker-compose.yml file.

```
docker-compose down
```

To delete all networks, images, and volumes with Docker Compose, you can use the following commands:

```
docker-compose down --rmi all --volumes --remove-orphans
```

- The `--rmi all` flag tells Docker to remove all images associated with the containers.
- The `--volumes` flag tells Docker to remove any volumes associated with the containers.
- The `--remove-orphans` flag tells Docker to remove any containers that were not defined in the docker-compose.yml file.

```
docker network prune  
docker volume prune
```

- The second command (`docker network prune`) will remove any unused networks.
- The third command (`docker volume prune`) will remove any unused volumes.

# LAB 5 :

Screenshot of a GitHub repository for a "Example Voting App".

**Repository Structure:**

- Code
- Issues
- Pull requests
- Actions
- Projects
- Security
- Insights
- Settings

main · DevTools / 02\_Docker / Week11 / 06\_Lab\_docker\_compose /

Last commit message: c84c4bf · 14 minutes ago · History

Name Last commit message Last commit date

- ..
- .github
- healthchecks
- k8s-specifications
- result
- seed-data
- vote
- worker
- ignore
- LICENSE
- MAINTAINERS
- README.md
- architecture.excalidraw.png
- docker-compose.yml

README.md

**Example Voting App**

A simple distributed application running across multiple Docker containers.

**Getting started**

The diagram illustrates the architecture of the Example Voting App. It features a front-end web application (represented by a person icon with a laptop) that interacts with a .NET worker container. The .NET worker container also connects to a Redis database and a PostgreSQL database (db). The Redis database is shown with a red cube icon, and the PostgreSQL database is shown with a grey elephant icon.

- A front-end web app in [Python](#) which lets you vote between two options
- A [Redis](#) which collects new votes
- A [.NET](#) worker which consumes votes and stores them in...
- A [Postgres](#) database backed by a Docker volume
- A [Node.js](#) web app which shows the results of the voting in real time



