



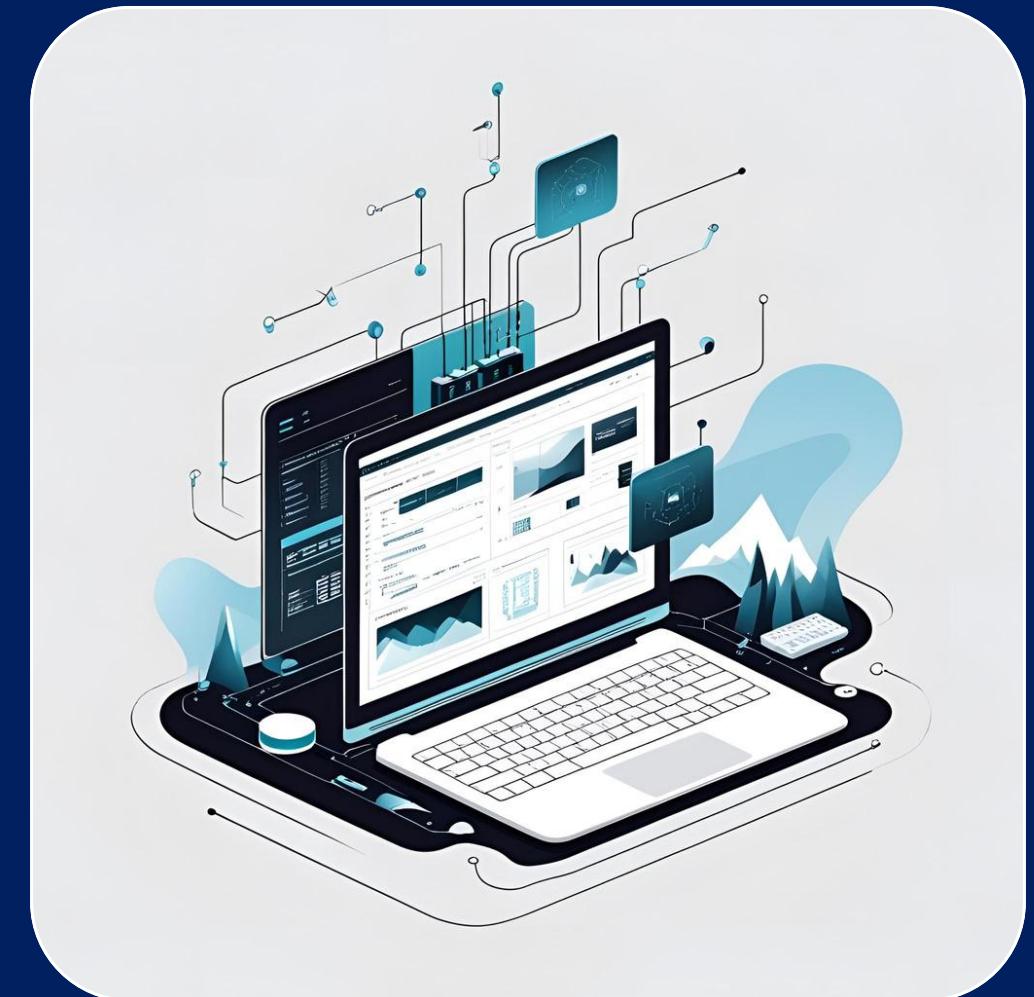
WEEK 11

# SOFTWARE DEVELOPMENT TOOLS AND ENVIRONMENTS

UNLIMITED

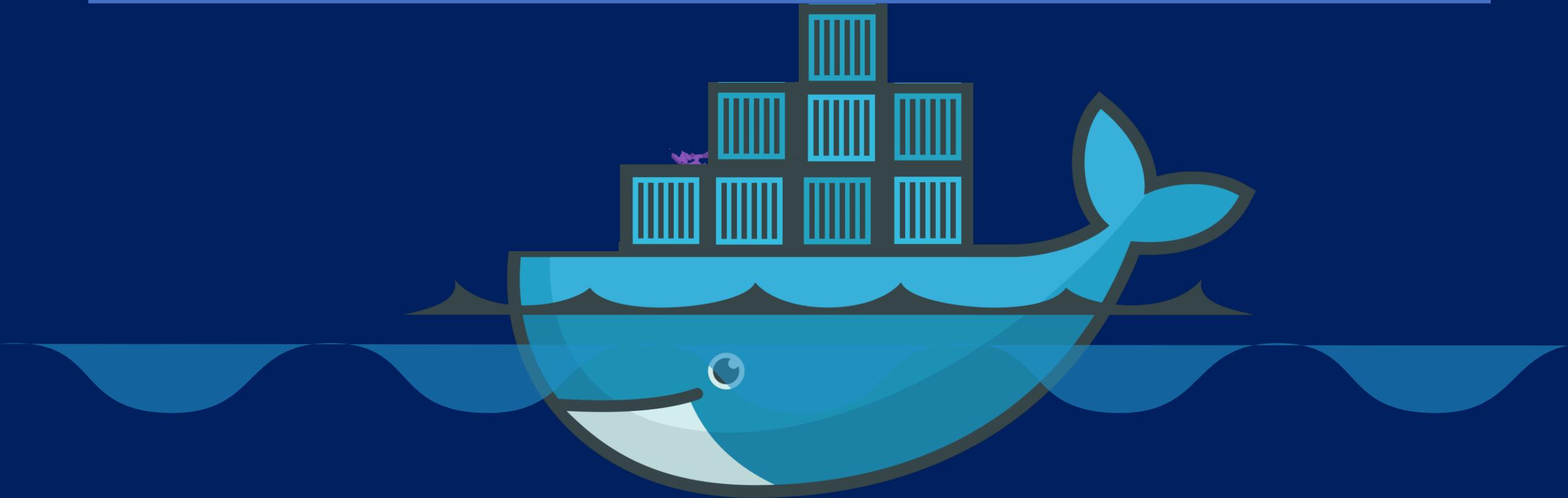
Presented by

Asst. Prof. Dr. Tuchsanai Ploysuwan



# Dockerfile and Docker Image

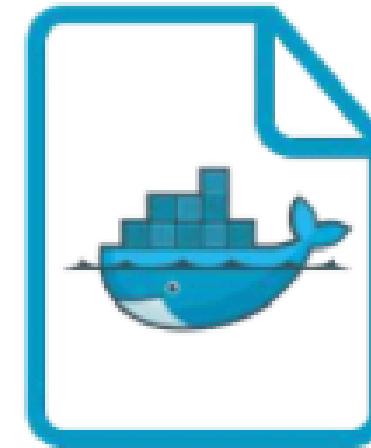
---



# 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.



## : ความแตกต่างระหว่าง CMD และ ENTRYPOINT ใน Dockerfile

### ◆ CMD (Command)

- ใช้กำหนด ค่าเริ่มต้น (default command/arguments) ที่จะรันเมื่อ container start
- สามารถถูก override ได้ เวลา run docker run
- มีได้เพียง 1 ค่าเท่านั้น (ถ้ามีหลายบรรทัด จะขับบรรทัดสุดท้าย)

ตัวอย่าง:

dockerfile

ကัดลอกโค้ด

```
FROM ubuntu:20.04
CMD ["echo", "Hello from CMD"]
```

Run:

bash

ကัดลอกโค้ด

```
docker run myimage
# Output: Hello from CMD
```

```
docker run myimage echo "Override CMD"
# Output: Override CMD
```



## : ความแตกต่างระหว่าง CMD และ ENTRYPOINT ใน Dockerfile

### ◆ CMD (Command)

- ใช้กำหนด ค่าเริ่มต้น (default command/arguments) ที่จะรันเมื่อ container start
- สามารถถูก override ได้ เวลา run docker run
- มีได้เพียง 1 ค่าเท่านั้น (ถ้ามีหลายบรรทัด จะขับรถหัดสุดท้าย)

ตัวอย่าง:

dockerfile

ကัดลอกโค้ด

```
FROM ubuntu:20.04
CMD ["echo", "Hello from CMD"]
```

Run:

bash

ကัดลอกโค้ด

```
docker run myimage
# Output: Hello from CMD
```

```
docker run myimage echo "Override CMD"
# Output: Override CMD
```

## ◆ ENTRYPPOINT

- ใช้กำหนด **คำสั่งหลัก** (main command) ที่ container ต้องรันเสมอ
- ไม่สามารถ override ได้ง่าย (arguments จะถูก append ต่อท้าย)
- เหมาะสมกับการสร้าง executable container

ตัวอย่าง:

dockerfile

ကัดลอกโคด

```
FROM ubuntu:20.04
ENTRYPOINT ["echo", "Hello from ENTRYPPOINT"]
```

Run:

bash

ကัดลอกโคด

```
docker run myimage
# Output: Hello from ENTRYPPOINT
```

```
docker run myimage World
# Output: Hello from ENTRYPPOINT World
```

- ◆ CMD + ENTRYPOINT ใช้ร่วมกัน

- ENTRYPOINT = main program
- CMD = default arguments

ตัวอย่าง:

dockerfile

ดูโค้ด

```
FROM ubuntu:20.04
ENTRYPOINT ["echo"]
CMD ["Hello from CMD"]
```

Run:

bash

ดูโค้ด

```
docker run myimage
# Output: Hello from CMD

docker run myimage World
# Output: World
```

# 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 Dockerfile -t mmumshad/my-custom-app
```

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



# Dockerfile

Dockerfile

INSTRUCTION

ARGUMENT

Dockerfile

FROM Ubuntu

Start from a base OS or another image

RUN apt-get update

RUN apt-get install python

Install all dependencies

RUN pip install flask

RUN pip install flask-mysql

COPY ./opt/source-code

Copy source code

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

Specify Entrypoint

# Layered architecture

Dockerfile

```
FROM Ubuntu

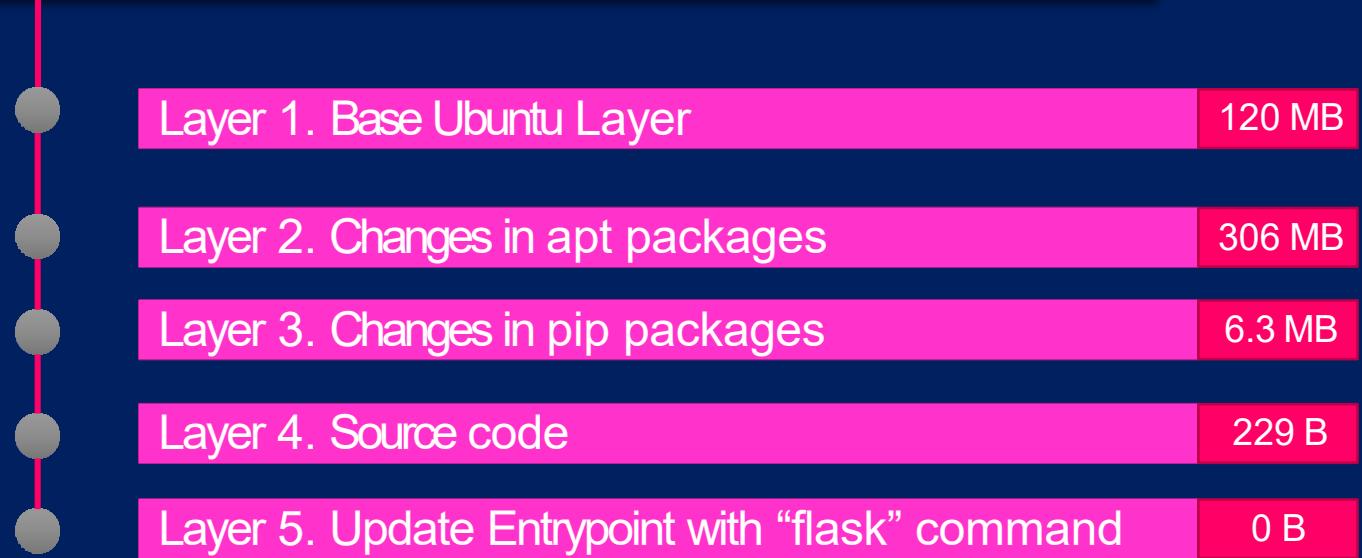
RUN apt-get update && apt-get -y install python

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 0 and LAB 1 :

GitHub repository: DevTools / 02\_Docker / Week11 / 00\_LAB\_CMD\_EntryPoint /

Name	Last commit message	Last commit date
Dockerfile-CMD	main edit	1 minute ago
Dockerfile-ENTRYPOINT	main edit	1 minute ago
readme.md	dd	4 minutes ago
readme.md		

### Objective

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

```
git clone -b dev https://github.com/Tuchsanai/DevTools.git
```

```
cd DevTools/02_Docker/Week11/00_LAB_CMD_EntryPoint/
```

GitHub repository: DevTools / 02\_Docker / Week11 / 01\_Lab\_Dockerfile /

Name	Last commit message	Last commit date
Dockerfile	ddd	1 hour ago
jupyter.png	dd	2 hours ago
readme.md	d	1 hour ago
requirements.txt	dd	2 hours ago
readme.md		

### LAB Basic example of a Dockerfile for creating a lab environment with Python, Jupyter, and some essential data science libraries:

A Dockerfile is a script that contains instructions to build a Docker image. Here's a basic example of a Dockerfile for creating a lab environment with Python, Jupyter, and some essential data science libraries:

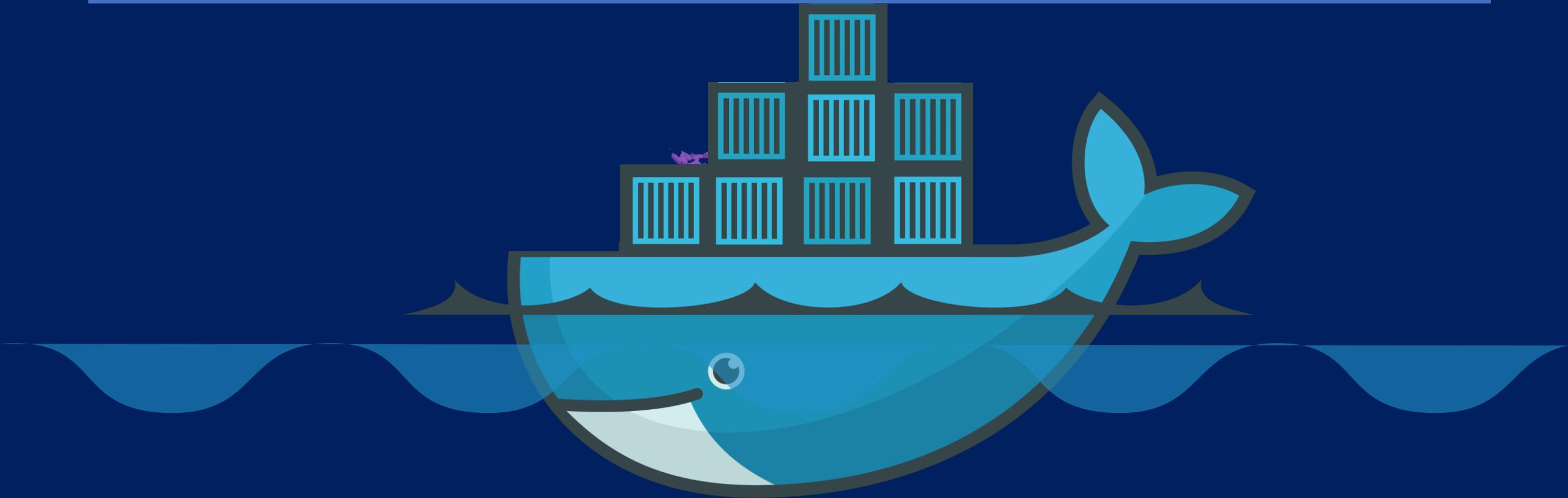
notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions.

jupyter

Name	Last Modified	File size
Dockerfile	3 นาทีก่อน	509 B
readme.md	3 นาทีก่อน	1.45 kB
requirements.txt	3 นาทีก่อน	16 B

# Docker registry

---



**1**

The Docker Hub homepage features a blue header with the Docker logo and navigation links for "Sign In" and "Sign up". A red circle highlights the "Sign In" button. Below the header is a search bar labeled "Search Docker Hub". The main content area includes sections for "Spotlight" and "Featured Solutions", each displaying cards for various Docker services like Docker Build Cloud, LLM Everywhere, Docker and Hugging Face, Docker Scout, Kubescape, Ambassador Telepresence, Grafana, and OpenSearch.

**2**

A screenshot of the Docker sign-in page at [login.docker.com/u/login/identifier?state=hKf02SB1aWVWM3RPaGfIandSaWRXcJQ1Mff6505JQBrnMZDKFur3VuaxZlcnNhC1sb2dpbqN0aWTZIEdyQnpTU...](https://login.docker.com/u/login/identifier?state=hKf02SB1aWVWM3RPaGfIandSaWRXcJQ1Mff6505JQBrnMZDKFur3VuaxZlcnNhC1sb2dpbqN0aWTZIEdyQnpTU...). It shows a "Sign in" form with fields for "Username or email address" (containing "xxxx") and "Password". There are also "Continue" buttons for Google and GitHub.

**3**

A screenshot of the Docker Hub user profile for "tuchsanai". The top navigation bar includes "Explore", "Repositories", and "Organizations". The "Repositories" tab is active. The main content shows three public repositories: "tuchsanai / pytorch\_jupyterlab\_ubuntu22.04", "tuchsanai / kubfirstapp", and "tuchsanai / custom-nginx-registry". To the right, there's a "Create An Organization" section with a "Create repository" button.

**4**

A screenshot of the same Docker Hub user profile for "tuchsanai", but with a context menu open over the user icon. The menu items are "What's New", "My Profile", "My Account" (which is highlighted with a red circle), and "Billing". Below the menu, a "Sign out" link is visible. The "Organization" section at the bottom right is partially visible.

hub.docker.com/settings/security

5

tuchsanai

Joined April 4, 2022

General  
Security (circled)  
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

6

tuchsanai

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

Copy and Close

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.

Enable Two-Factor Authentication

# Image

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



Registry

User

Image : tag

docker.io  
Docker Hub

Screenshot of the Docker Hub web interface (hub.docker.com) showing a user's repositories. The user is "tuchsanai".

Repository	Last pushed	Stars	Forks	Status
tuchsanai / pytorch_jupyterlab_ubuntu22.04	3 months ago	0	41	Public
tuchsanai / kubirstapp	5 months ago	0	14	Public
tuchsanai / custom-nginx-registry	11 months ago	0	3	Public

The interface includes a search bar, navigation tabs (Explore, Repositories, Organizations), and a "Create repository" button. On the right, there are sections for "Create An Organization" and "Create and manage users and grant access to your repositories".

# 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 '02\_Lab\_docker\_registry'. The repository was created by 'Tuchsanai' 39 minutes ago. The file structure includes a 'readme.md' file and several files in the 'images' folder: 'Dockerfile', 'index.html', and 'readme.md'. The 'readme.md' file is currently being edited.

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

**readme.md**

**Lab docker registry**

**Step 1: Login to Docker Hub**

Run the following command to login to Docker Hub:

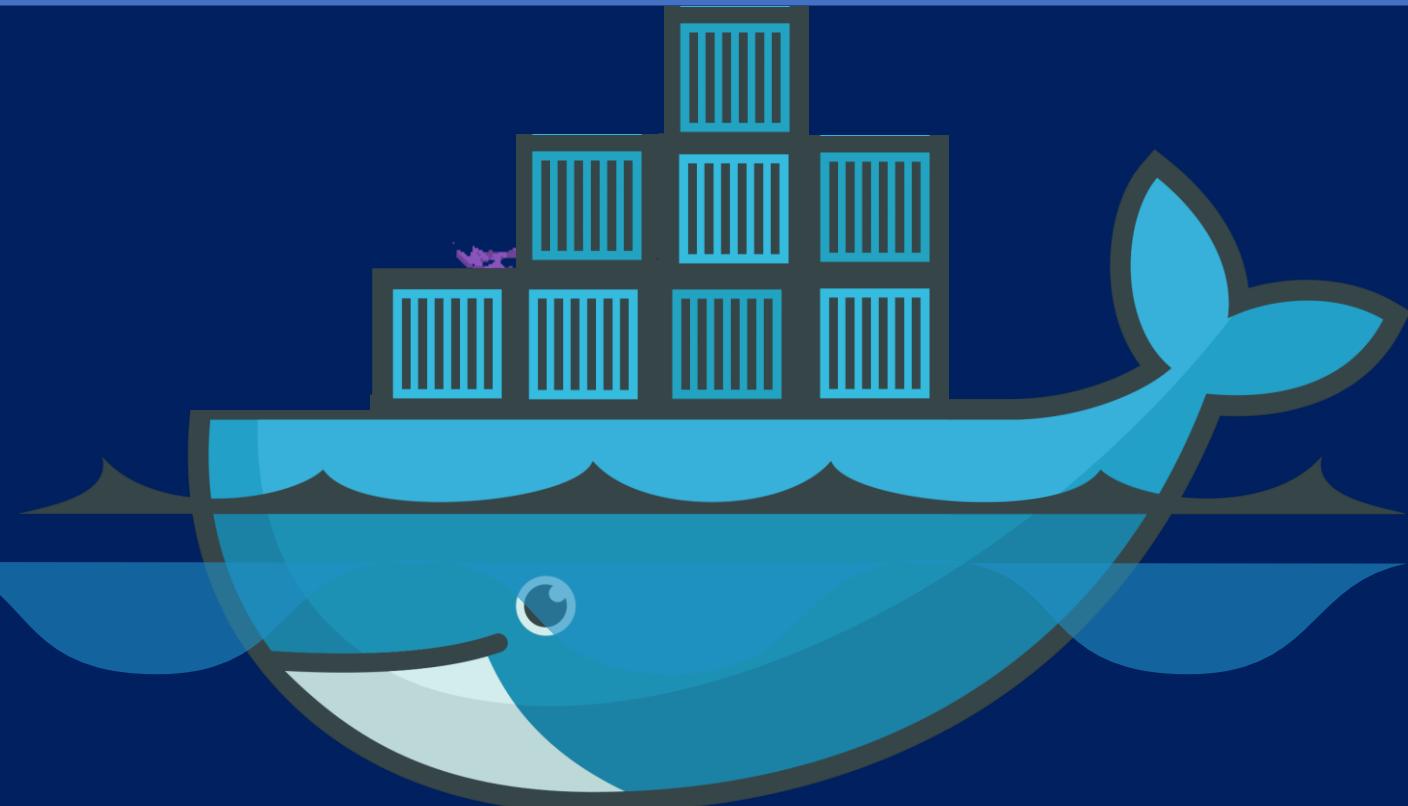
```
docker login -u yourusername
```

#### Enter your password when prompted.  
.....

**Step 2:**

# Docker Network

---



# Default networks



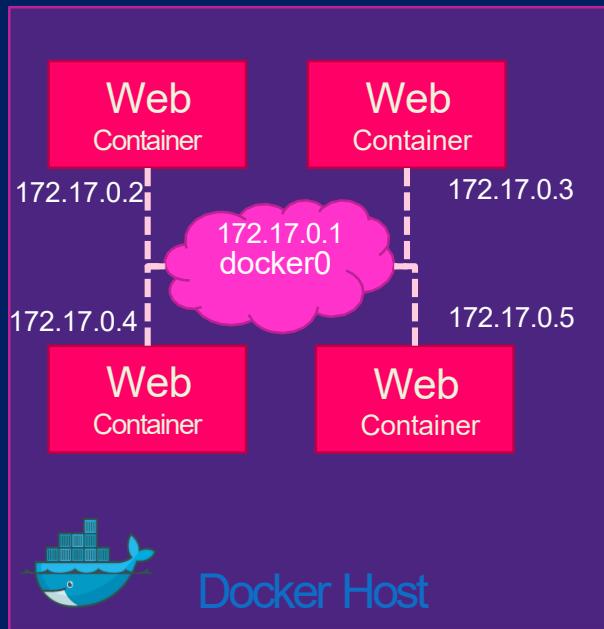
`docker run ubuntu`



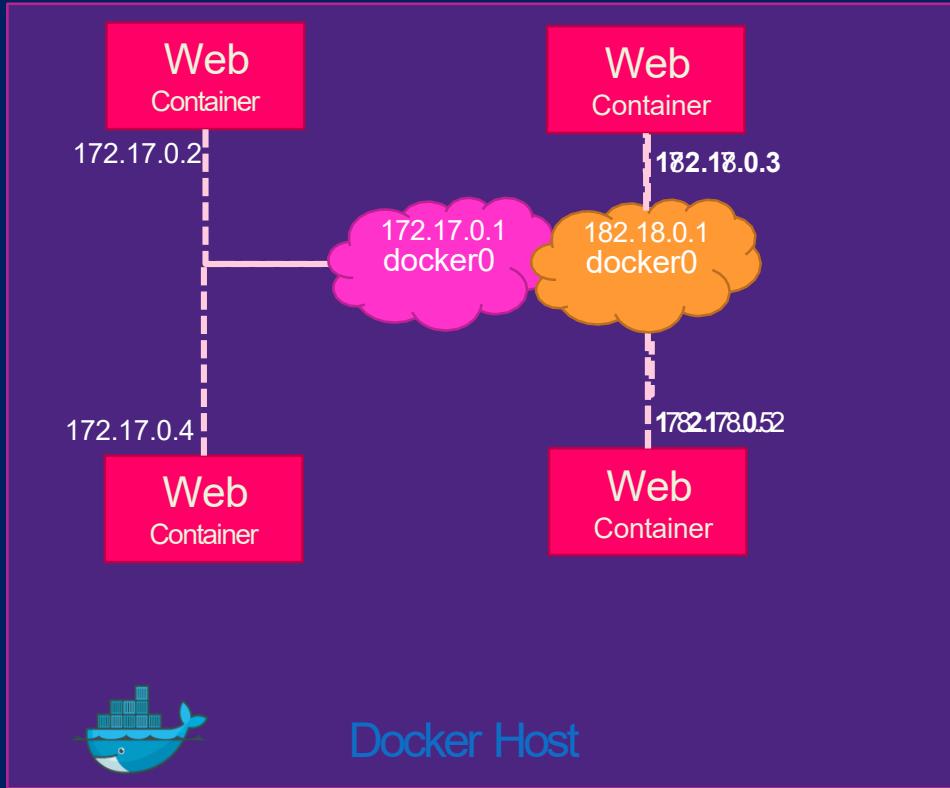
`docker run Ubuntu --network=none`



`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	simplewebappdockernet	bridge	local

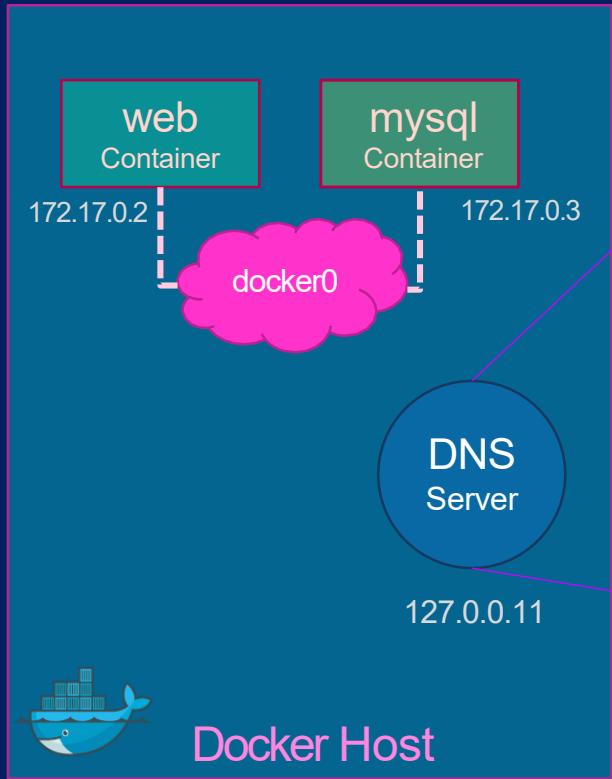
# Inspect Network

```
▶ 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:

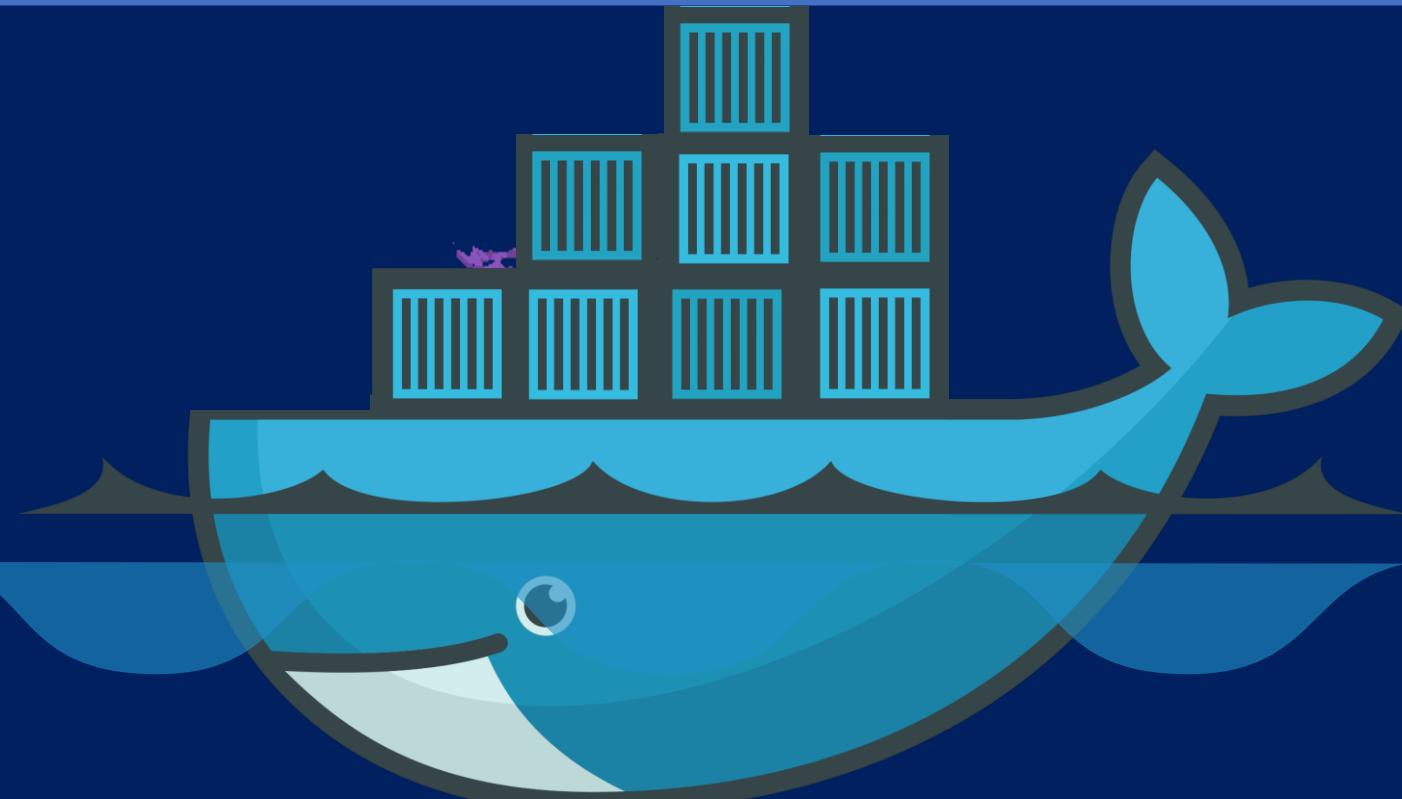
```
docker network inspect lab_network
```

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
```

# Docker compose

---



# Docker compose

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

```
docker run mongodb
```

```
docker run redis:alpine
```

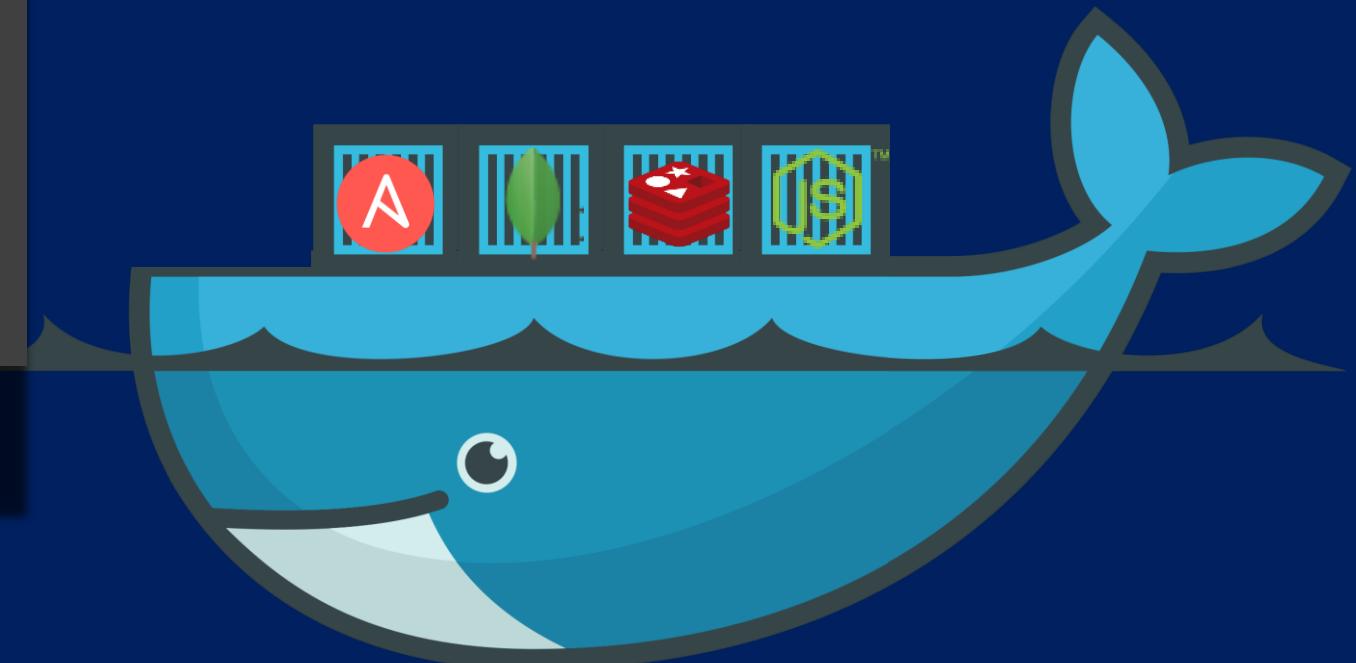
```
docker run ansible
```

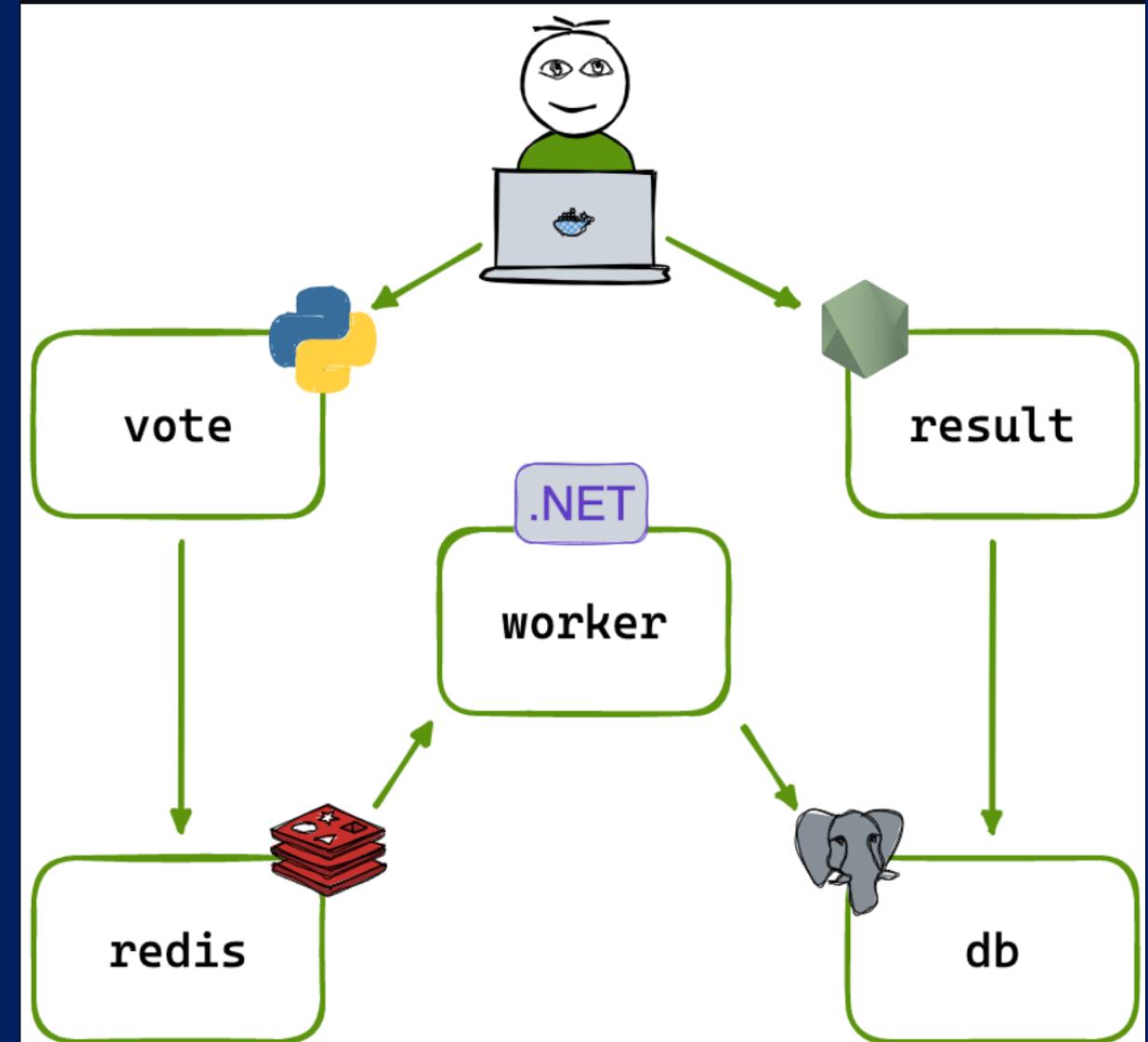
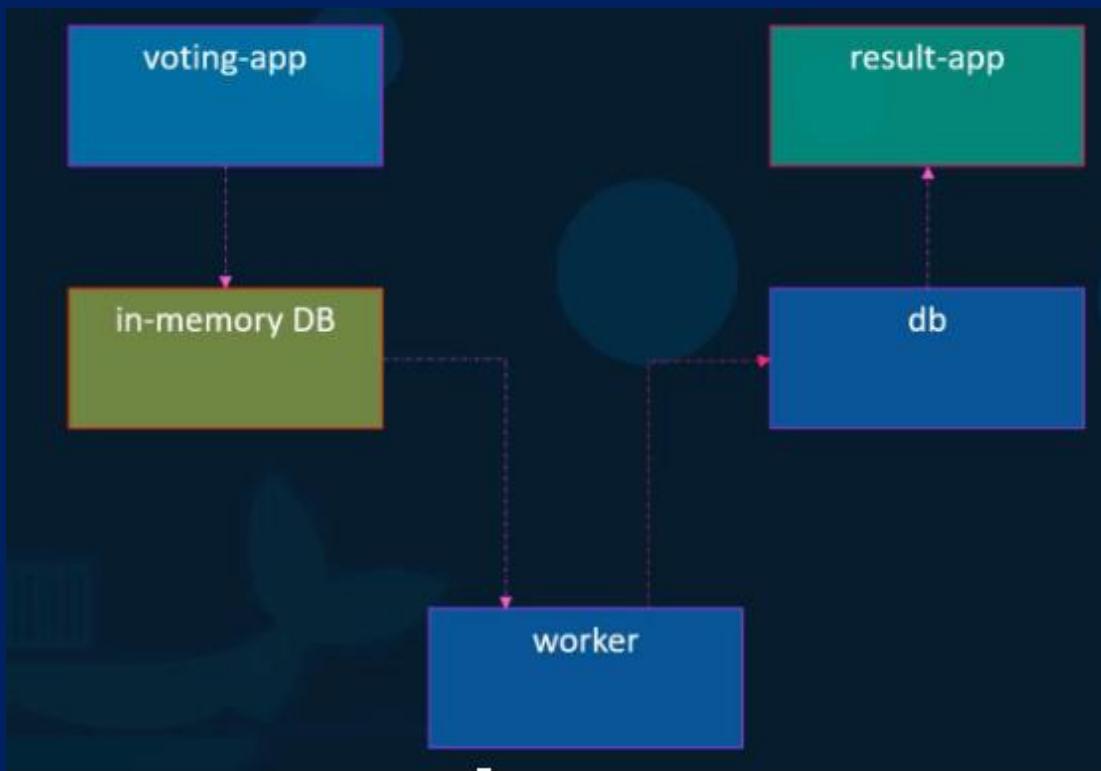
```
docker-compose.yml
```

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

```
docker-compose up
```

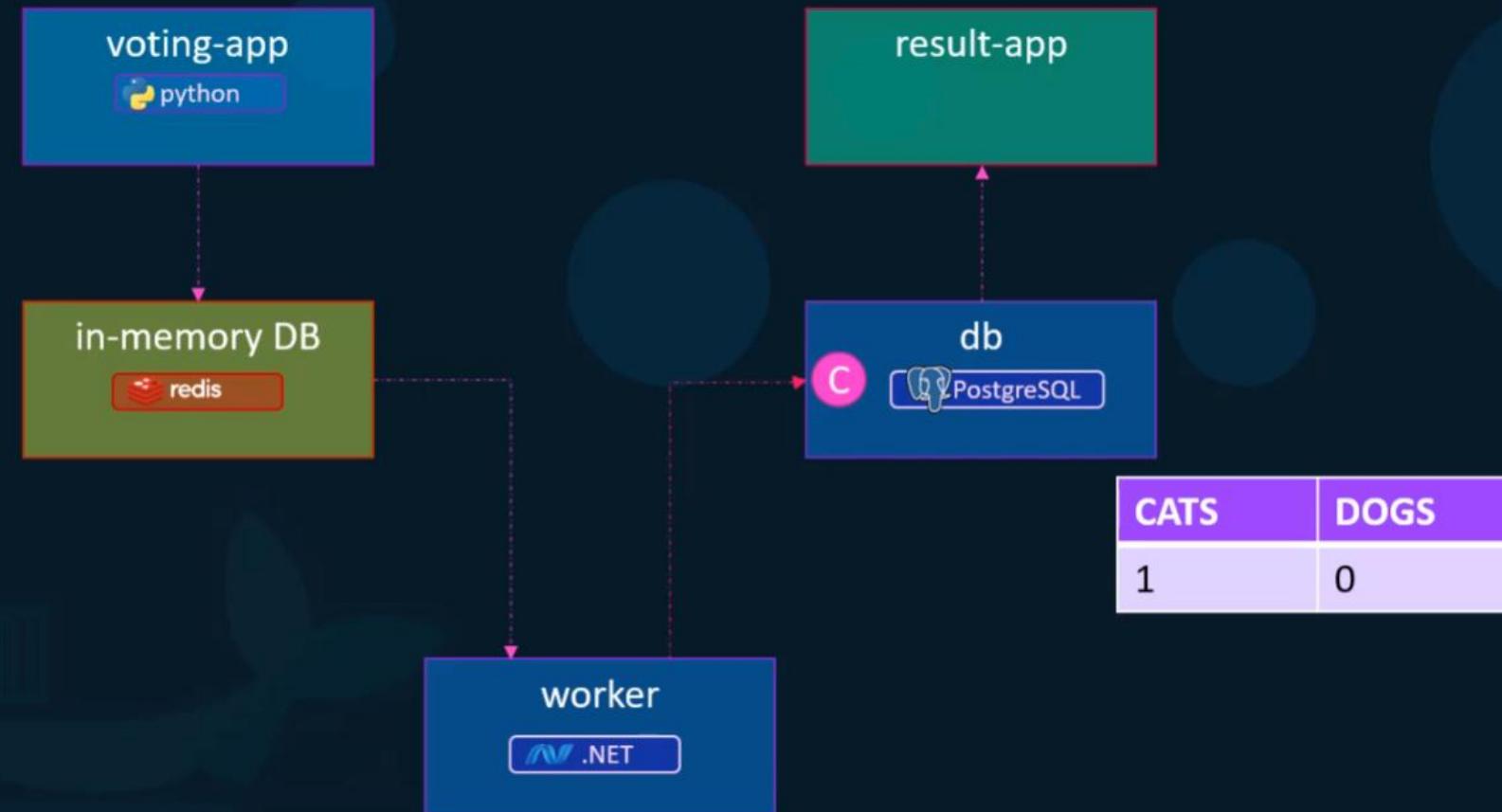
Public Docker registry - dockerhub



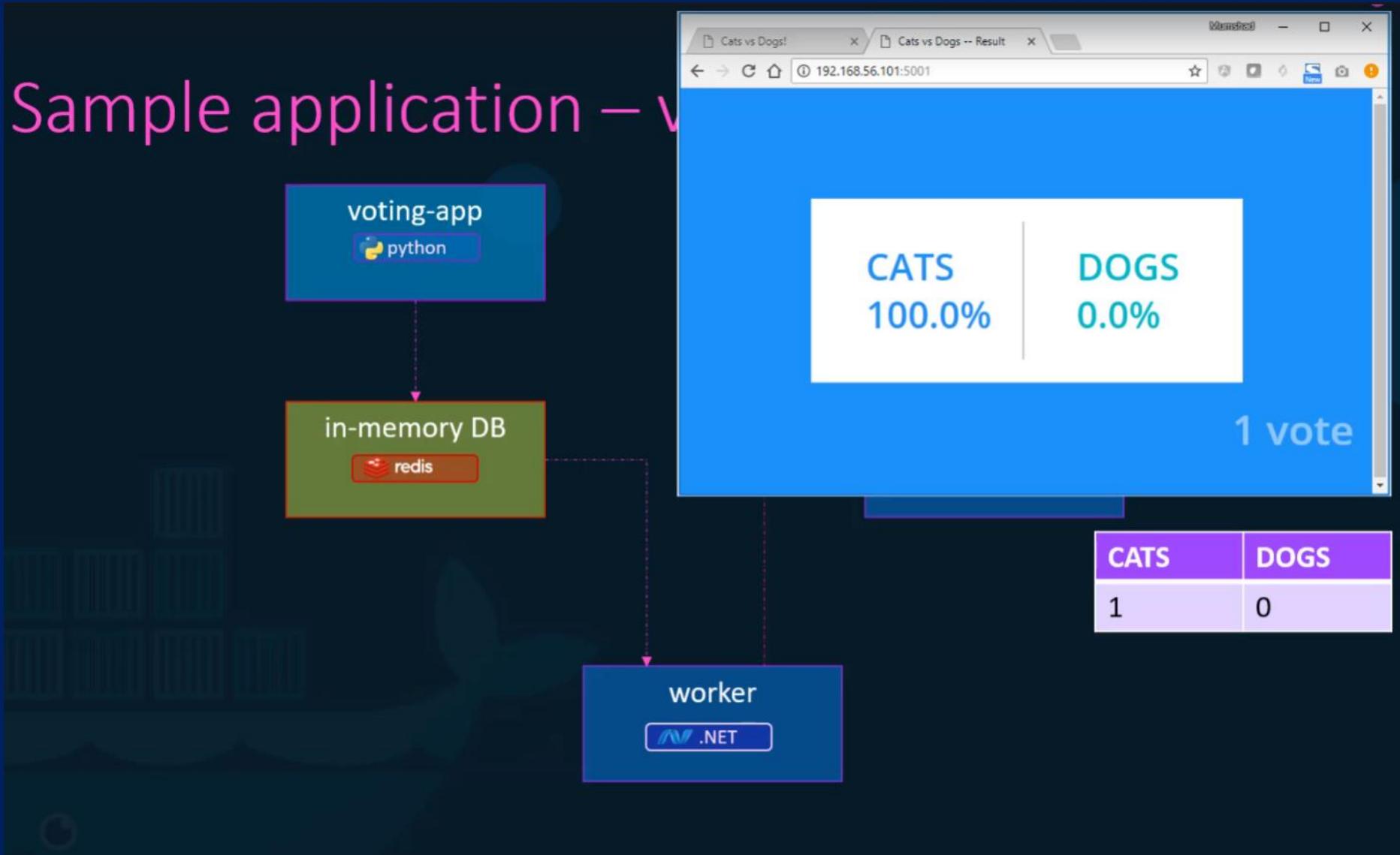


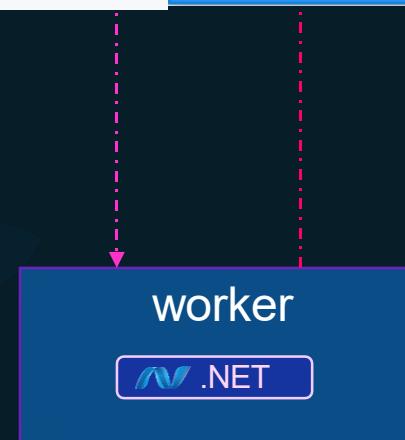
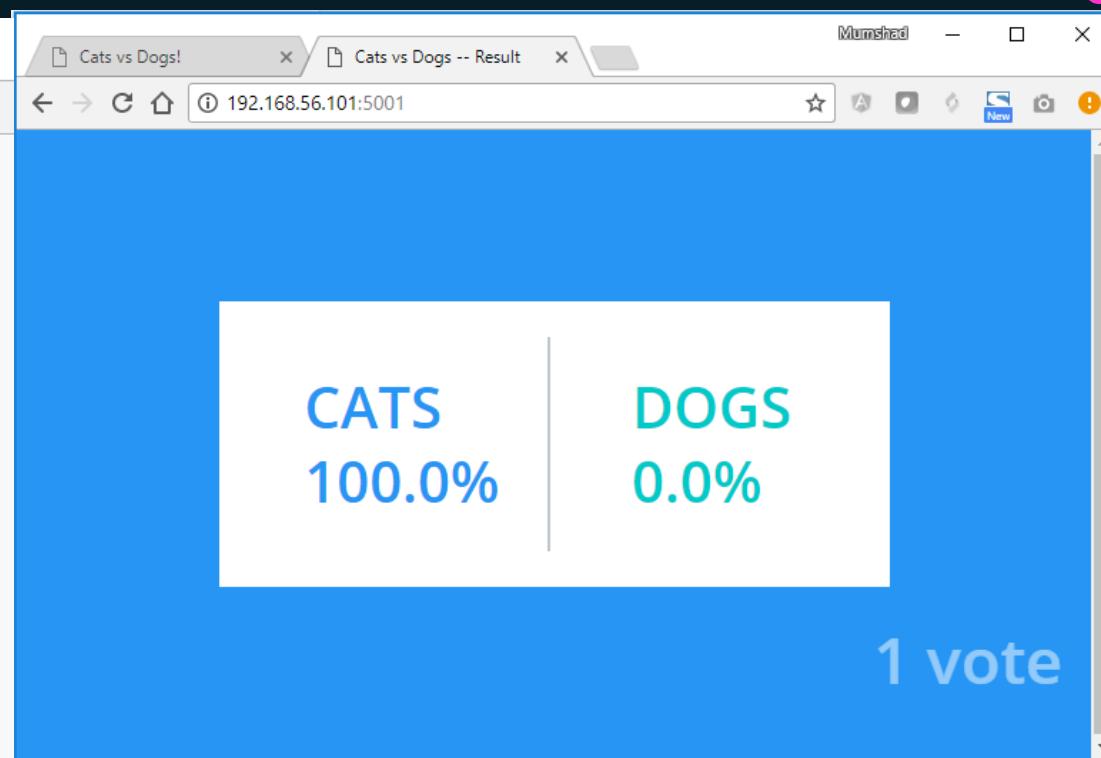
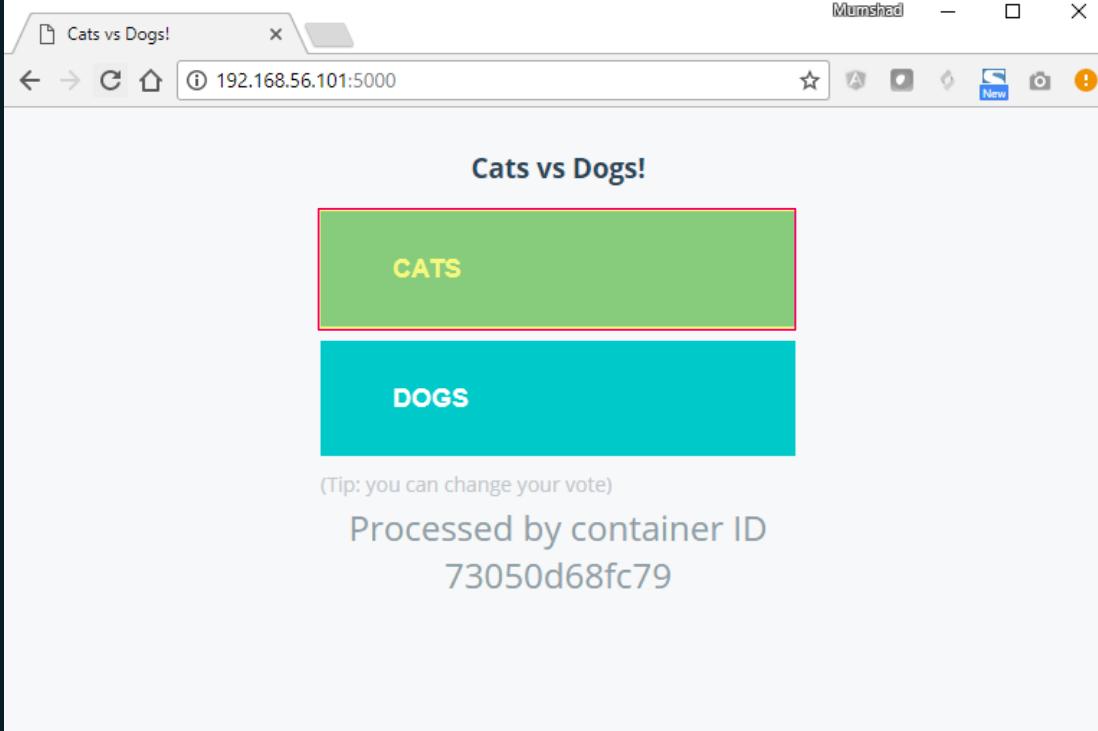
- A front-end web app in [Python](#) which lets you vote between two options
- A [Redis](#) which collects new votes
- A [.NET](#) worker which consumes votes and stores them in...
- A [Postgres](#) database backed by a Docker volume
- A [Node.js](#) web app which shows the results of the voting in real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

# Sample application – voting application



# Sample application – voting

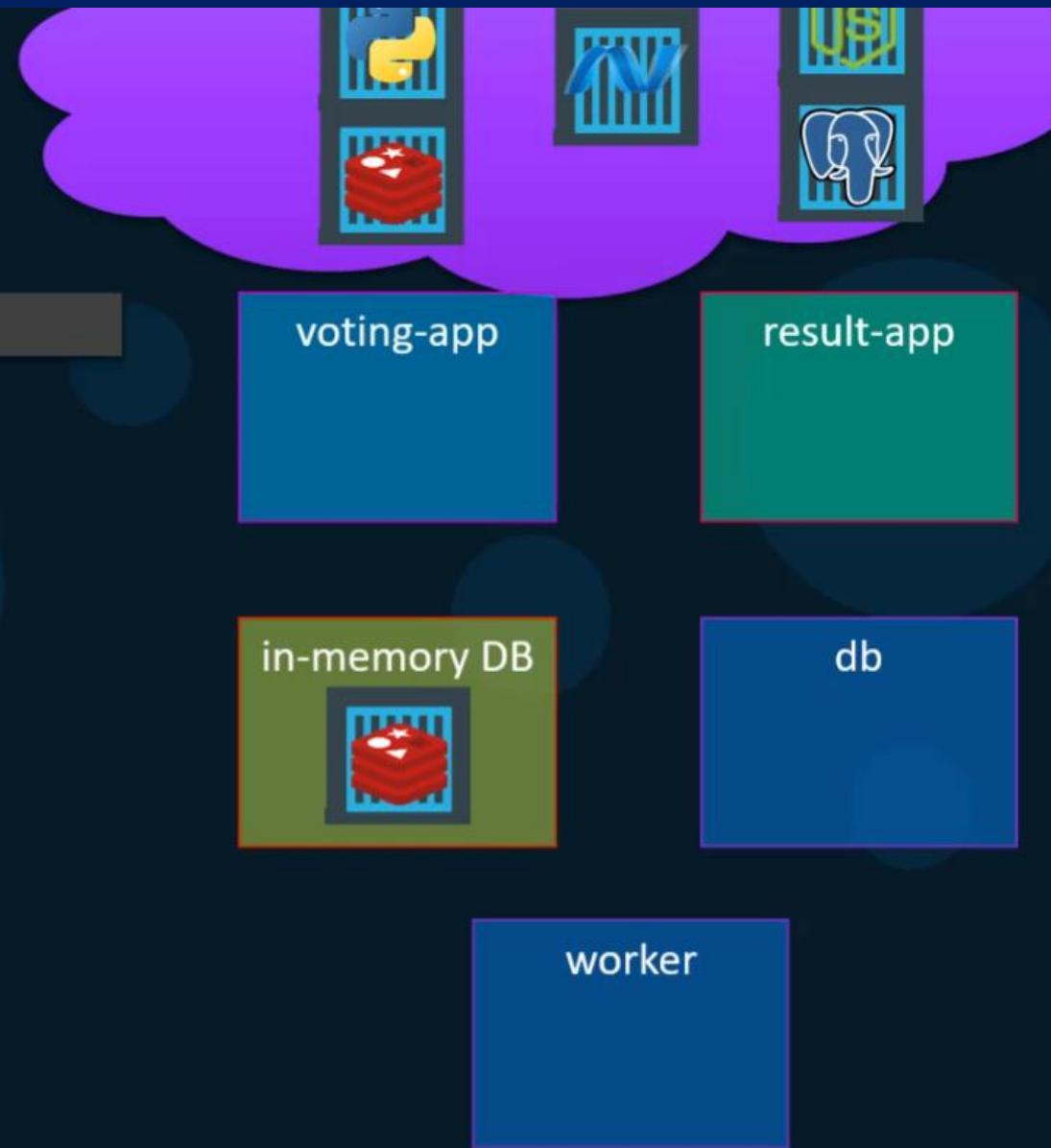




CATS	DOGS
1	0

# docker run

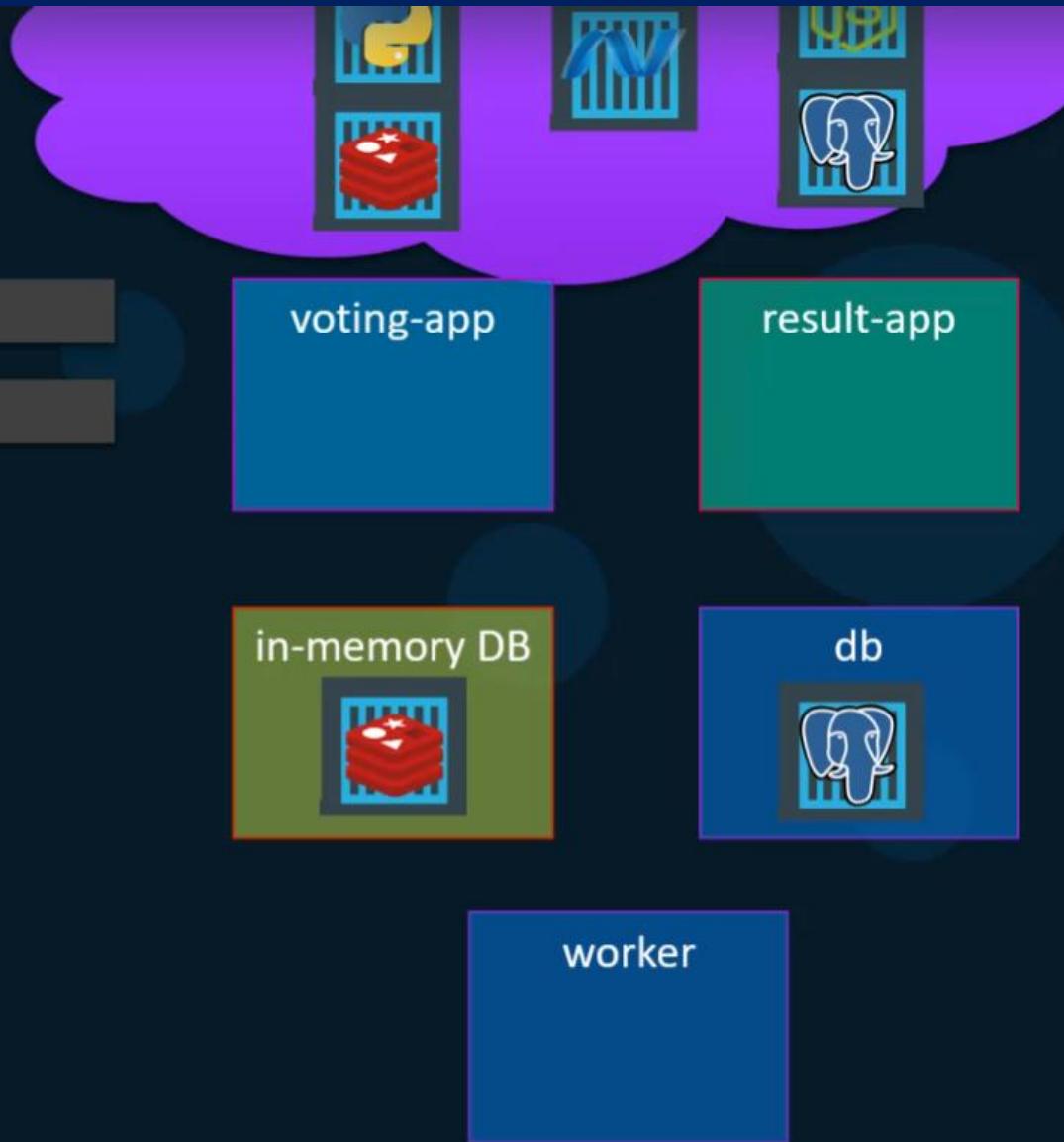
```
docker run -d --name=redis redis
```



# docker run

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

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

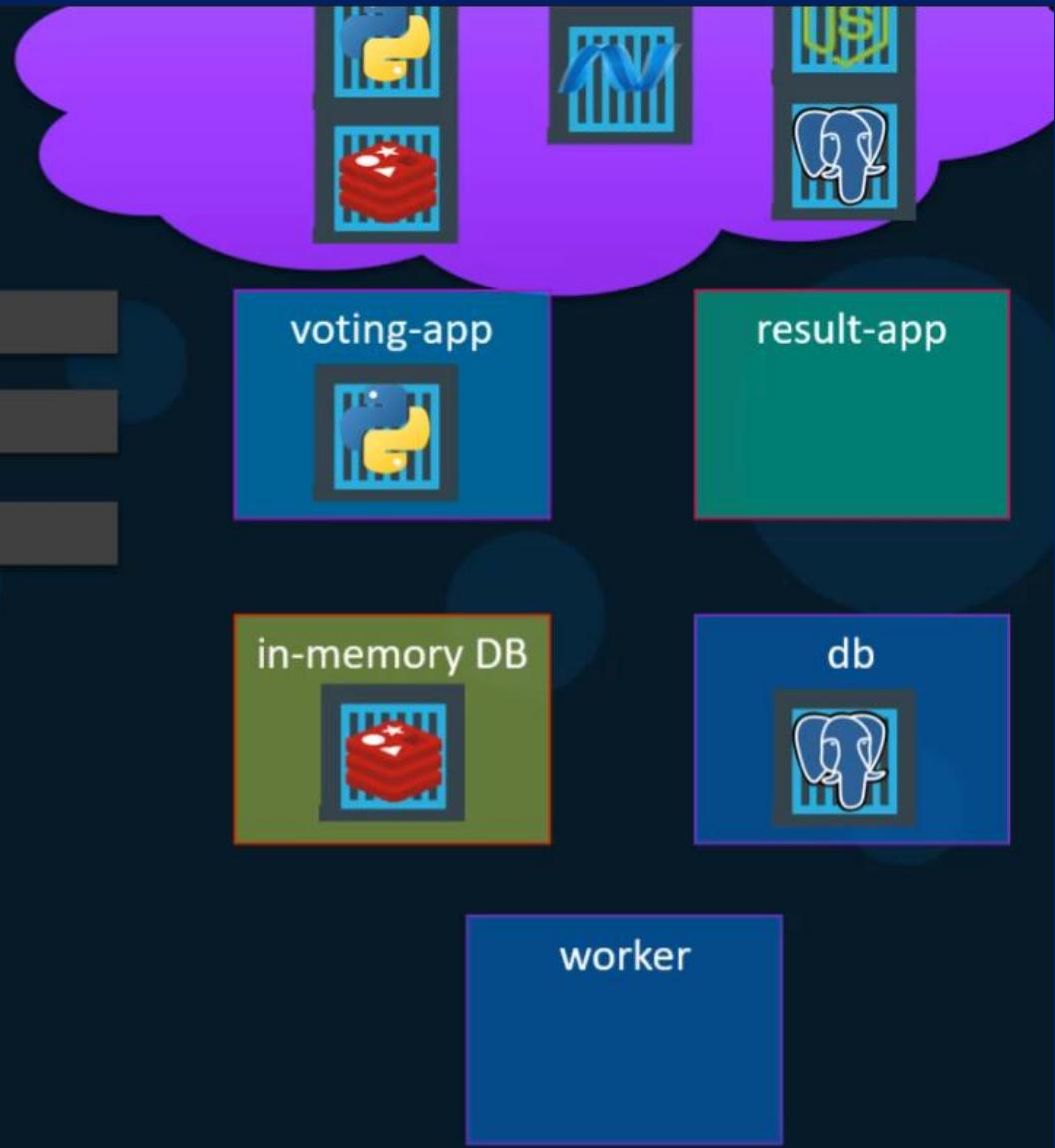


# docker run

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

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

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



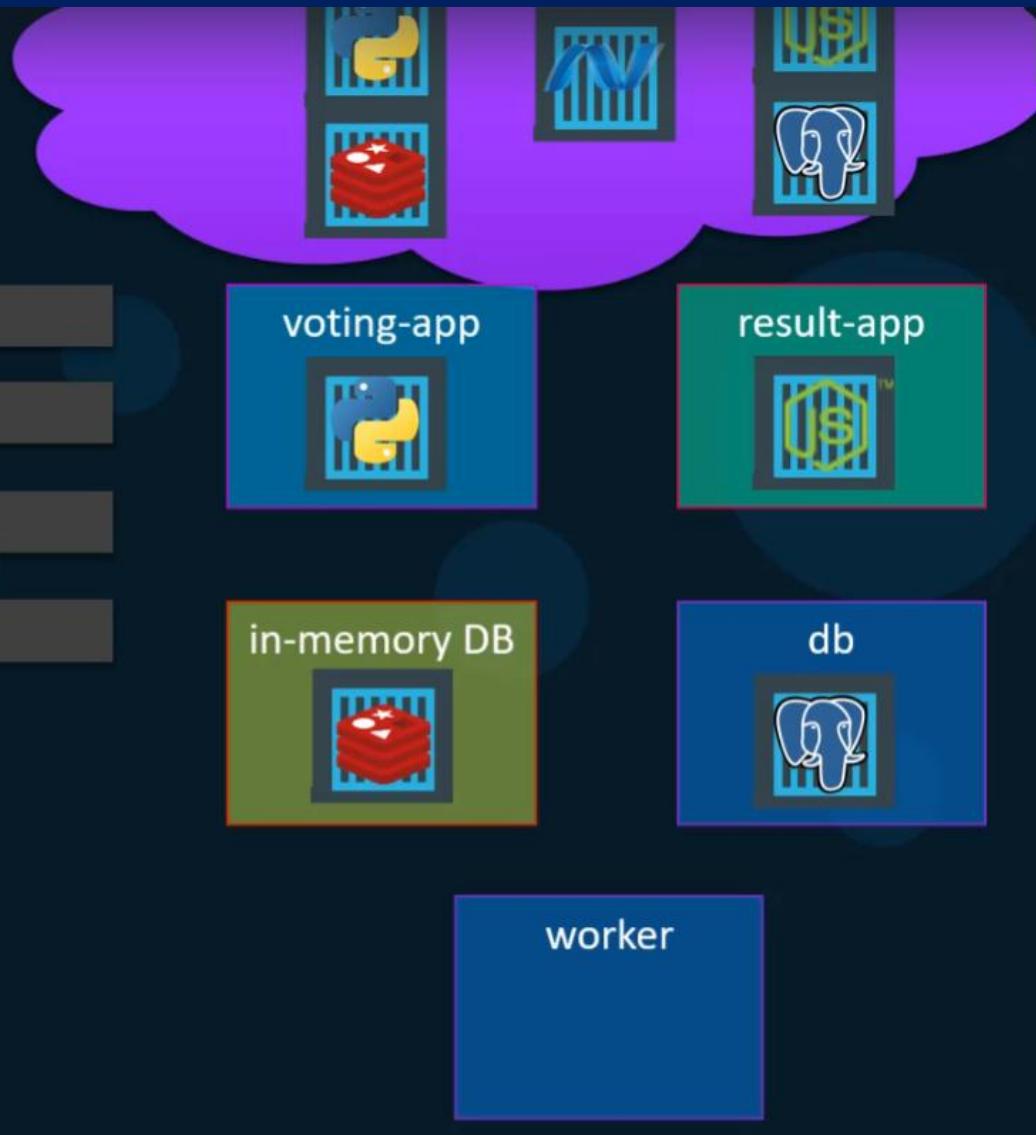
# docker run

```
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

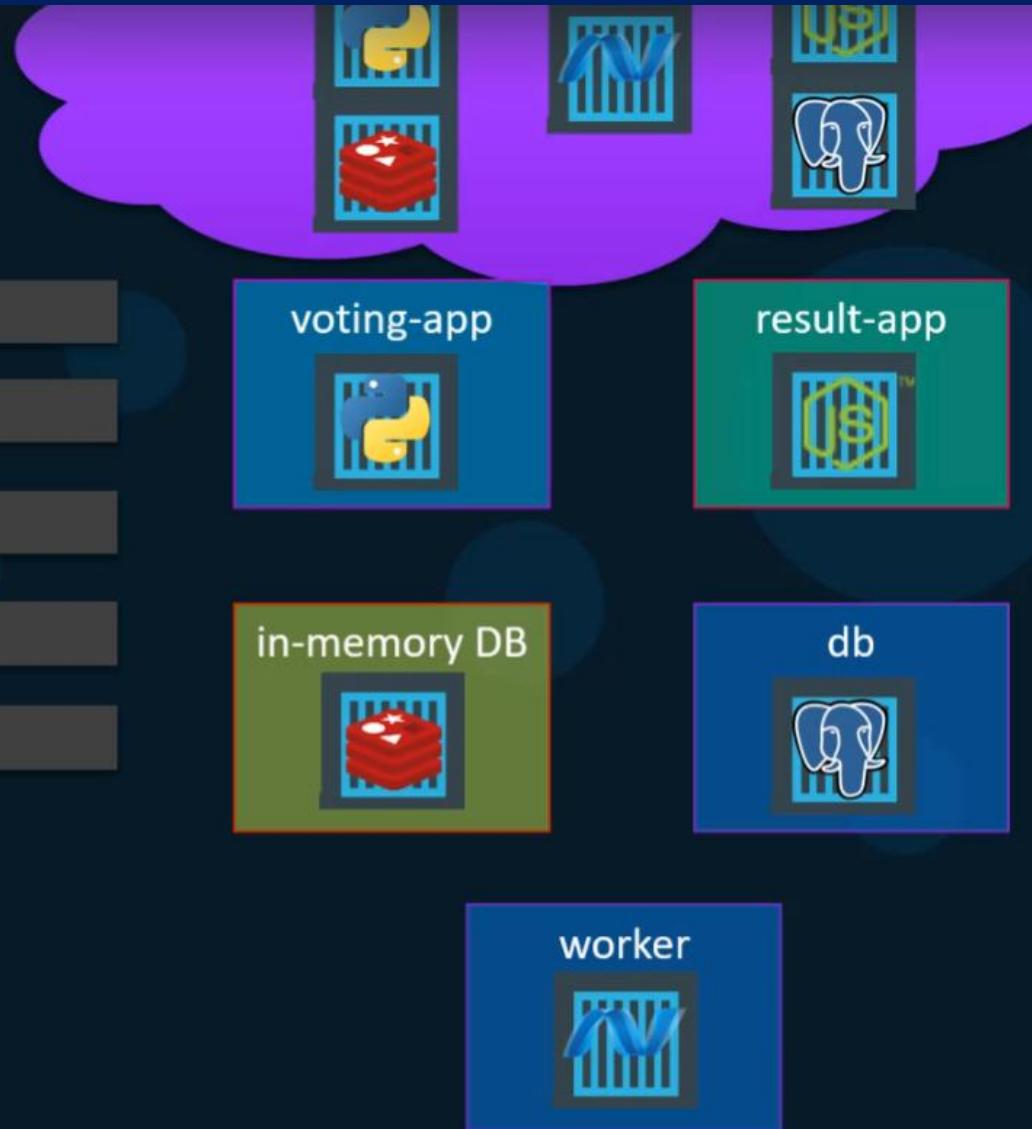
```
docker run -d --name=redis redis
```

```
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

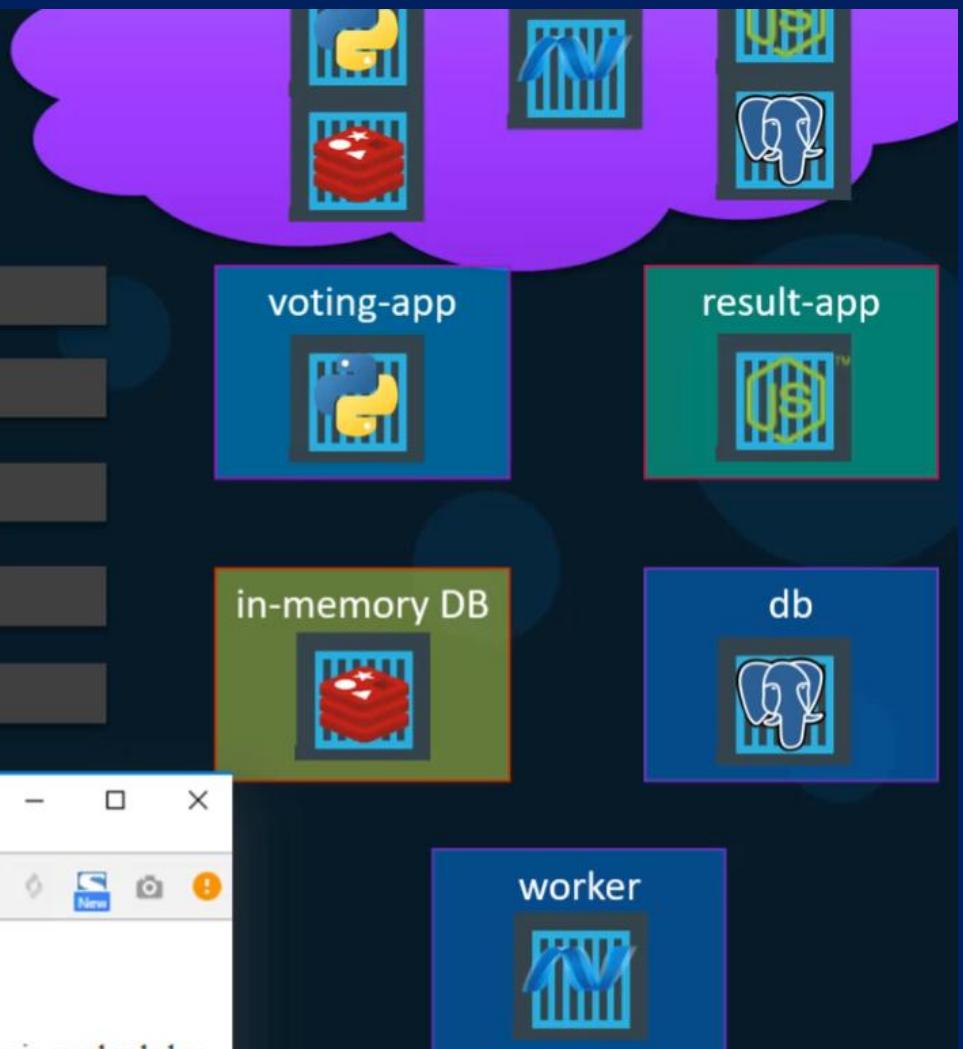
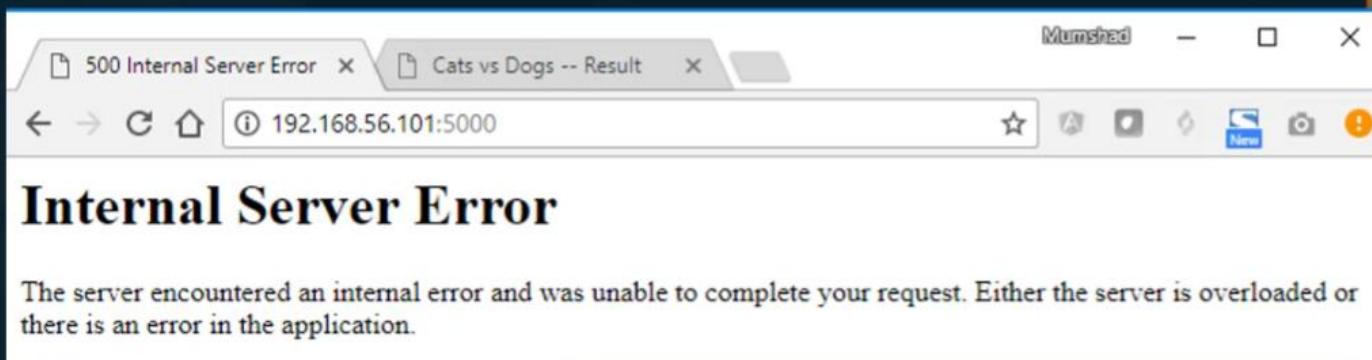
```
docker run -d --name=redis redis
```

```
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
```

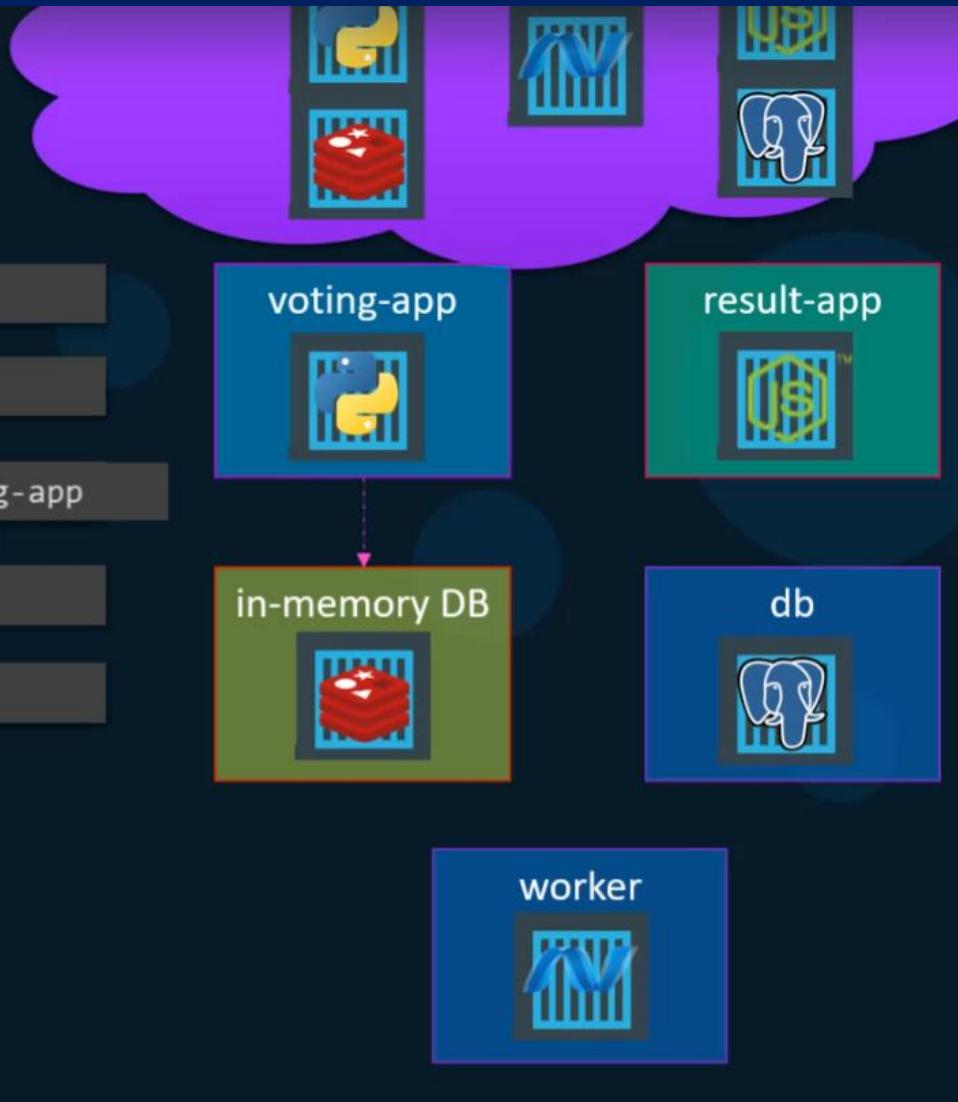
```
docker run -d --name=db postgres
```

```
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
```

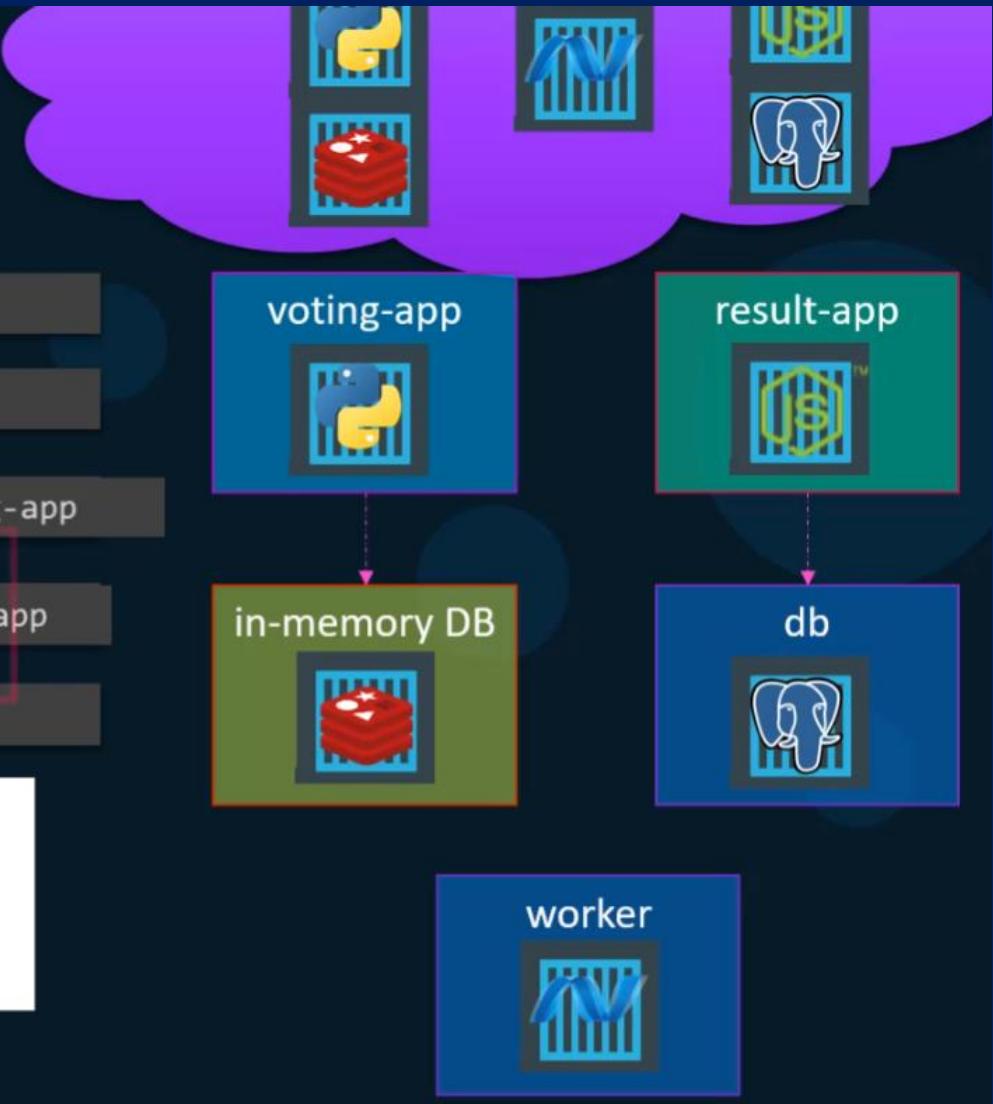
```
docker run -d --name=db postgres
```

```
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
```

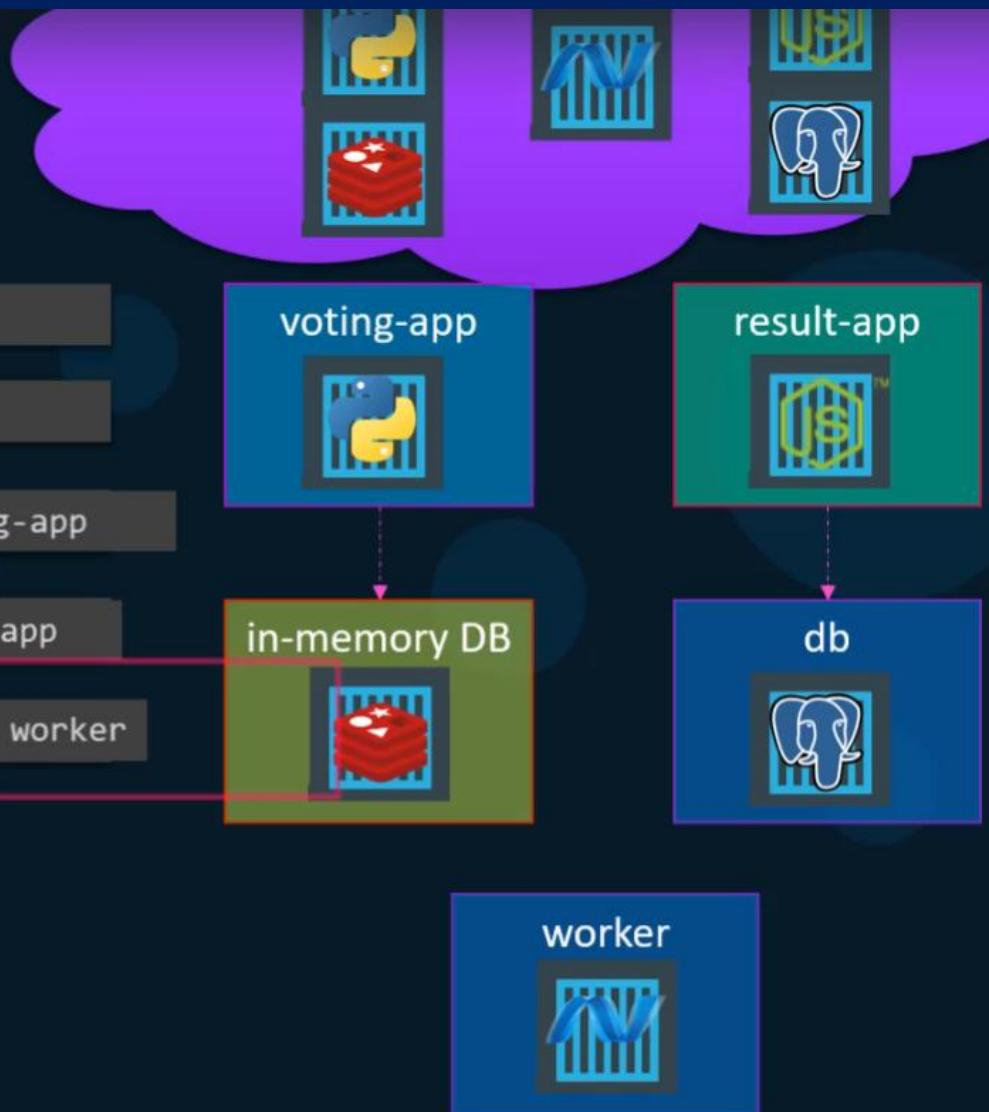
```
docker run -d --name=db postgres
```

```
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 /

Tuchsanai 0

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

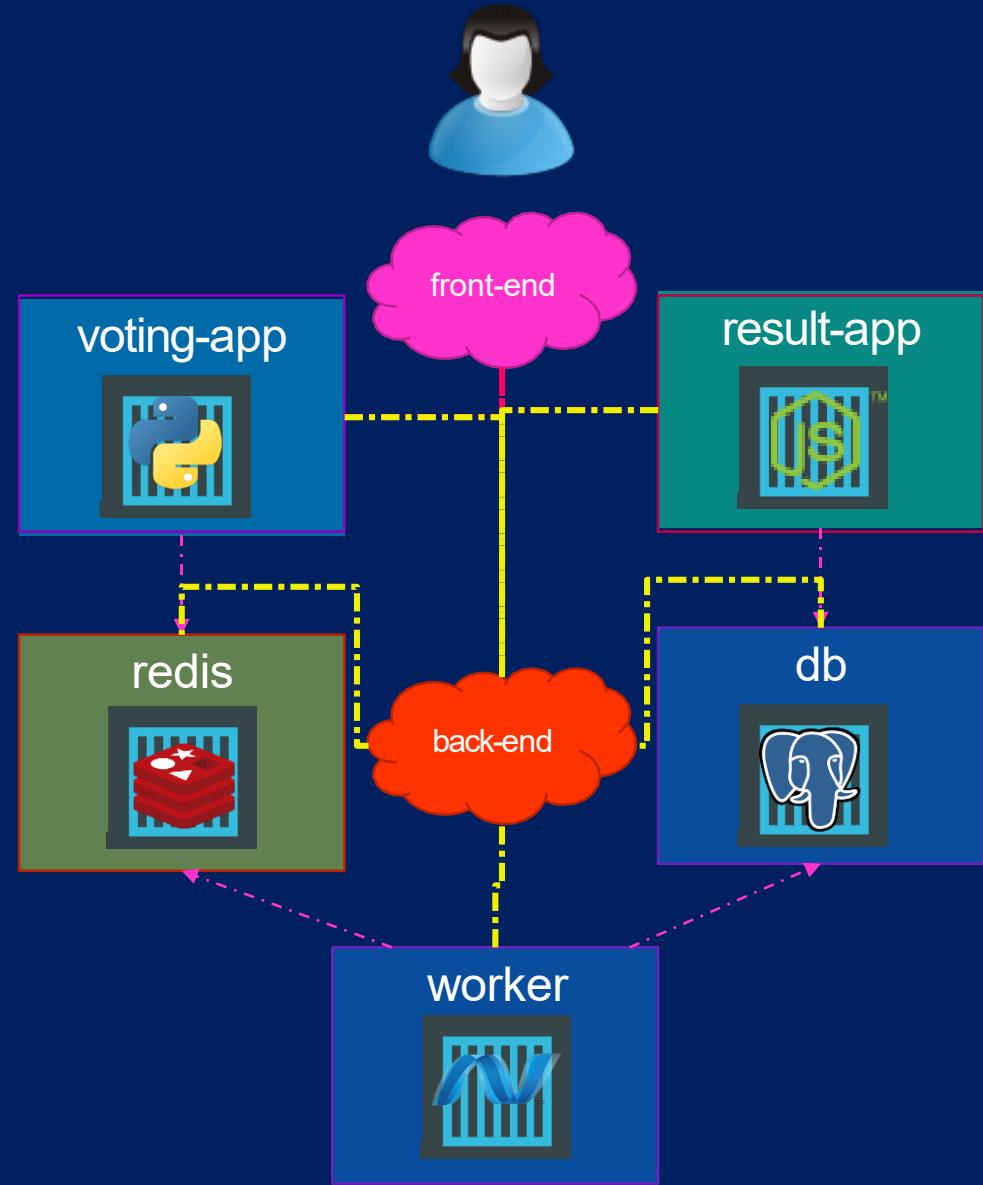
```
version: 3  
services:  
    redis:  
        image: redis  
  
    db:  
        image: postgres:9.4  
  
    vote:  
        image: voting-app  
        ports:  
            - 5000:80
```

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:
```



```
redis:
  image: redis:alpine
  volumes:
    - "./healthchecks:/healthchecks"
  healthcheck:
    test: /healthchecks/redis.sh
    interval: "5s"
  networks:
    - back-tier

db:
  image: postgres:15-alpine
  environment:
    POSTGRES_USER: "postgres"
    POSTGRES_PASSWORD: "postgres"
  volumes:
    - "db-data:/var/lib/postgresql/data"
    - "./healthchecks:/healthchecks"
  healthcheck:
    test: /healthchecks/postgres.sh
    interval: "5s"
  networks:
    - back-tier
```

```
vote:
  build: ./vote
  # use python rather than gunicorn for local dev
  command: python app.py
  depends_on:
    redis:
      condition: service_healthy
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost"]
    interval: 15s
    timeout: 5s
    retries: 3
    start_period: 10s
  volumes:
    - ./vote:/app
  ports:
    - "5000:80"
  networks:
    - front-tier
    - back-tier

result:
  build: ./result
  # use nodemon rather than node for local dev
  entrypoint: nodemon server.js
  depends_on:
    db:
      condition: service_healthy
  volumes:
    - ./result:/app
  ports:
    - "5001:80"
    - "5858:5858"
  networks:
    - front-tier
    - back-tier
```

## postgres.sh

```
#!/bin/bash
set -eo pipefail

host="$(hostname -i || echo '127.0.0.1')"
user="${POSTGRES_USER:-postgres}"
db="${POSTGRES_DB:-$POSTGRES_USER}"
export PGPASSWORD="${POSTGRES_PASSWORD:-}"

args=(
    # force postgres to not use the local unix socket (test "external" connectivity)
    --host "$host"
    --username "$user"
    --dbname "$db"
    --quiet --no-align --tuples-only
)

if select="$(echo 'SELECT 1' | psql "${args[@]}")" && [ "$select" = '1' ]; then
    exit 0
fi

exit 1
```

## redis.sh

```
1  #!/bin/sh
2  set -eo pipefail
3
4  host="$(hostname -i || echo '127.0.0.1')"
5
6  if ping="$(redis-cli -h "$host" ping)" && [ "$ping" = 'PONG' ]; then
7      exit 0
8  fi
9
10 exit 1
```

## **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 4 :

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

The repository structure is as follows:

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

Files listed in the main directory:

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

README.md content:

### Example Voting App

A simple distributed application running across multiple Docker containers.

#### Getting started

The architecture diagram illustrates the flow of data in the voting application:

- A user interface (laptop icon) interacts with a "vote" component (blue hexagon).
- The "vote" component interacts with a ".NET worker" component (purple rounded rectangle).
- The ".NET worker" component interacts with a "redis" component (red cube).
- The "redis" component interacts with a "db" component (elephant icon).
- The "db" component interacts with a "result" component (green hexagon).
- The "result" component provides feedback to the user interface.

List of components and their descriptions:

- 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