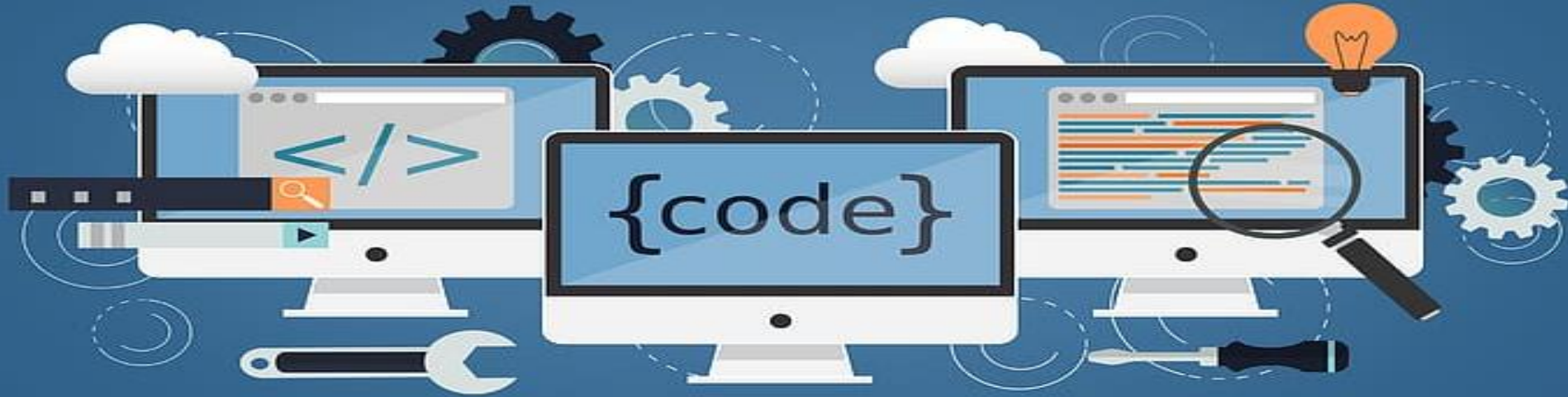


# DES424

## Cloud-based Application Development

### Lecture 11: Docker Compose and Tutorial 2



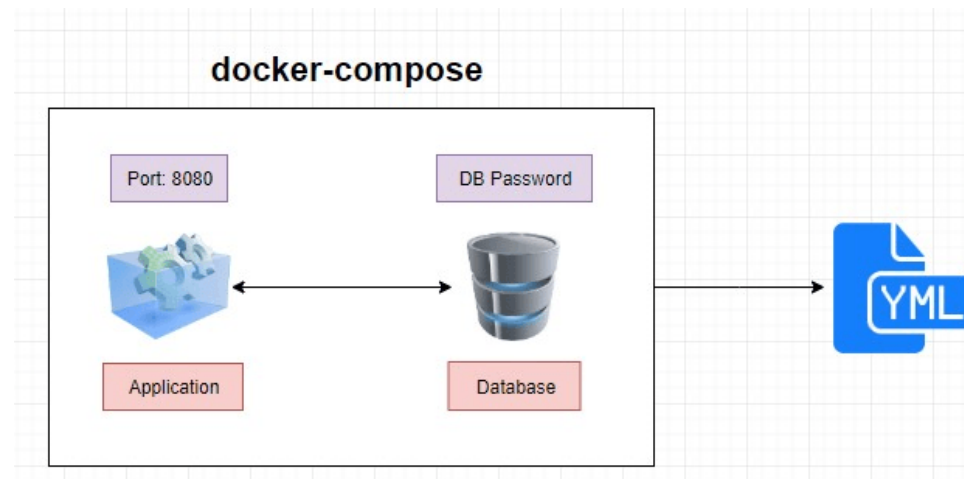
Dr. Apichon Witayangkurn  
([apichon@siit.tu.ac.th](mailto:apichon@siit.tu.ac.th))

# Tutorial Roadmap

- Storage with Docker Volume
- Storage with Bind Host
- Create custom build web image with code from GitHub
- Dockerfile with build argument
- Dockerfile with environment variable
- Share your image on Docker Hub
- Deploy image on Amazon Elastic Beanstalk
- A Node.js application using Nginx, Docker and Redis
- Portainer – Docker Management
- Dashboard for monitoring dockers
- Voting App with docker-compose

# Docker Compose

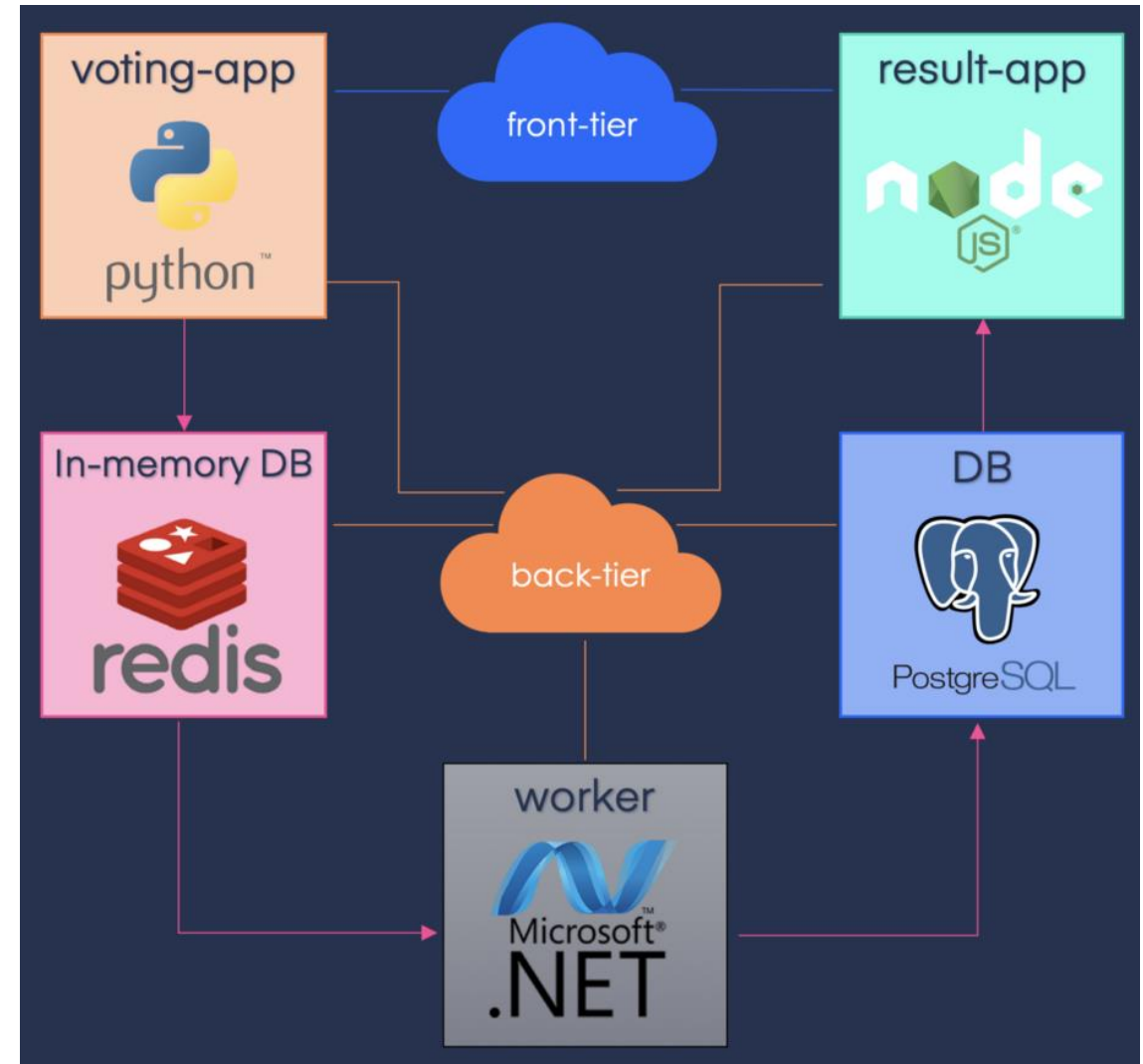
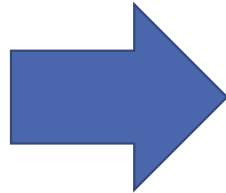
- It is a tool that assists in **defining** and **sharing multi-container applications**. By using Compose, we can **define** the services in a **YAML file**, as well as spin them up and tear them down with one **single command**.
- Using Compose is basically a three-step process:
  1. **Define** your **app's environment** with a Dockerfile so it can be reproduced anywhere.
  2. **Define** the **services** that **make up your app in docker-compose.yml** so they can be run together in an isolated environment.
  3. **Run docker compose up** and the Docker compose command starts and runs your entire app.



# Docker Compose

docker-compose.yml

```
version: 3
services:
  redis:
    image: redis
    networks:
      - back-tier
  db:
    image: postgres:9.4
    networks:
      - back-tier
  vote:
    build: ./vote
    ports:
      - 5000:80
    networks:
      - front-tier
      - back-tier
  result:
  worker:
networks:
  front-tier:
  back-tier:
```



# YAML for Docker Compose

- Docker Compose works by applying many rules declared within a **single docker-compose.yml** configuration file.
- In **compose file** (yml), it usually contain at least **one service** and optionally **volumes** and **network**.

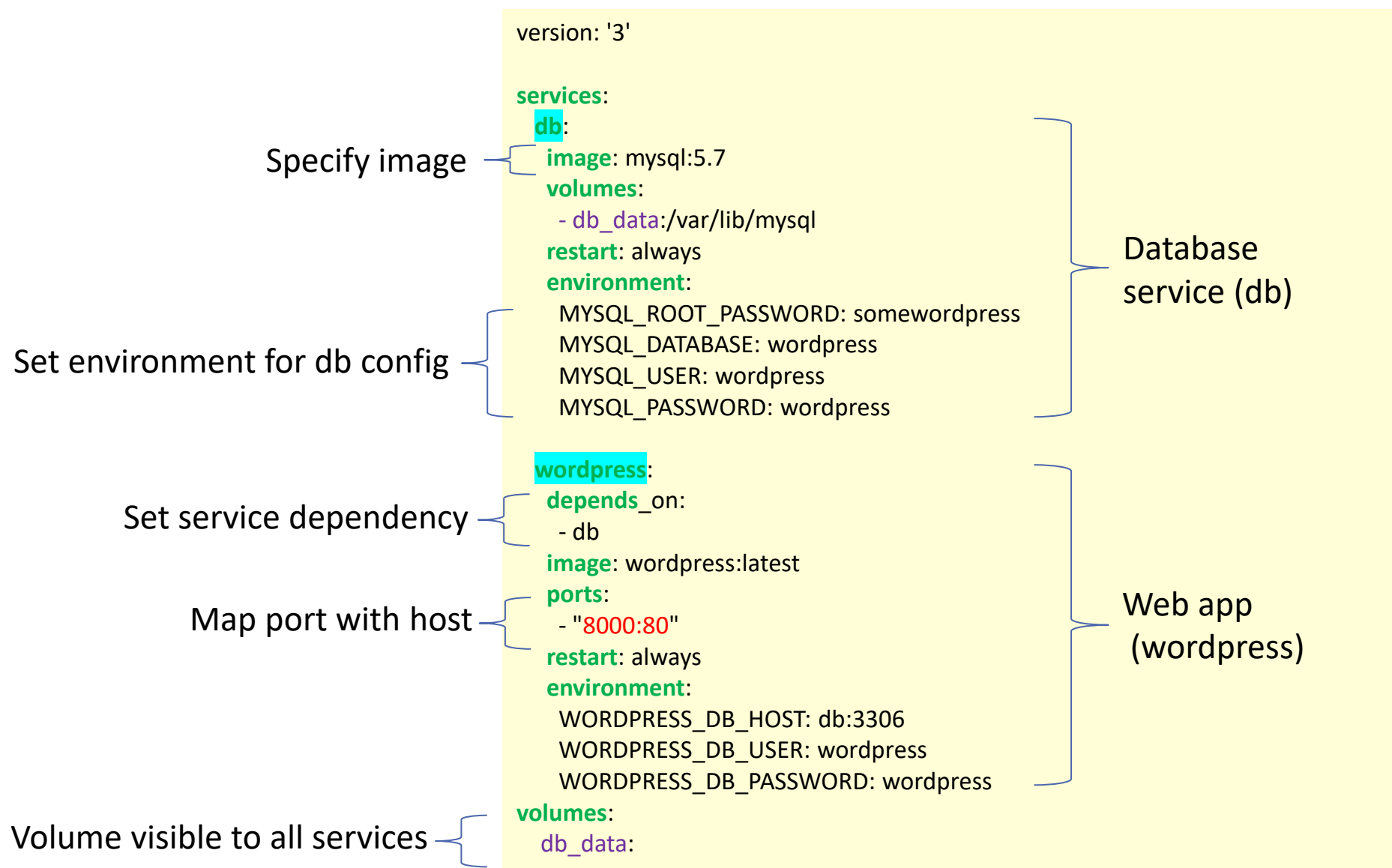
Services refer to containers' configuration

Define physical areas of disk space shared  
between the host and a container

Define the network communication between containers,  
and between a container and the host

```
version: '3'
services:
  ...
volumes:
  ...
networks:
  ...
```

# Example YAML for Docker Compose



# Docker Compose : Services

- Services refer to containers' configuration.
- For example, let's take a dockerized **web application** consisting of a **front end**, a **back end**, and a **database**, then split those components into three images and define them as three different services in the configuration.
- Service can be image from Docker Hub or build via Dockerfile

```
services:  
  frontend:  
    image: my-vue-app  
    ...  
  backend:  
    image: my-springboot-app  
    ...  
  db:  
    image: postgres  
    ...
```

```
services:  
  my-service:  
    image: ubuntu:latest  
    ...
```

**Pulling an Image**

```
services:  
  my-custom-app:  
    build: /path/to/dockerfile/  
    ...
```

**Building an Image**

```
services:  
  my-custom-app:  
    build: https://github.com/my-company/my-project.git  
    image: my-project-image  
    ...
```

# Docker Compose : Networking

- A service can **communicate with another service** on the **same network** by simply referencing it by **container name** and **port** (for example **network-example-service:80**), provided that we've made the port accessible through the **expose** keyword:

```
services:
  network-example-service:
    image: karthequian/helloworld:latest
    expose:
      - "80"
```

To reach a container from the host, the ports must be exposed declaratively through the ports keyword, which also allows us to choose if exposing the port differently in the host:

```
services:
  network-example-service:
    image: karthequian/helloworld:latest
    ports:
      - "80:80"
    ...
  my-custom-app:
    image: myapp:latest
    ports:
      - "8080:3000"
    ...
  my-custom-app-replica:
    image: myapp:latest
    ports:
      - "8081:3000"
    ...
```



# Docker Compose : Networking

- We can also create custom network aligning with system architecture.

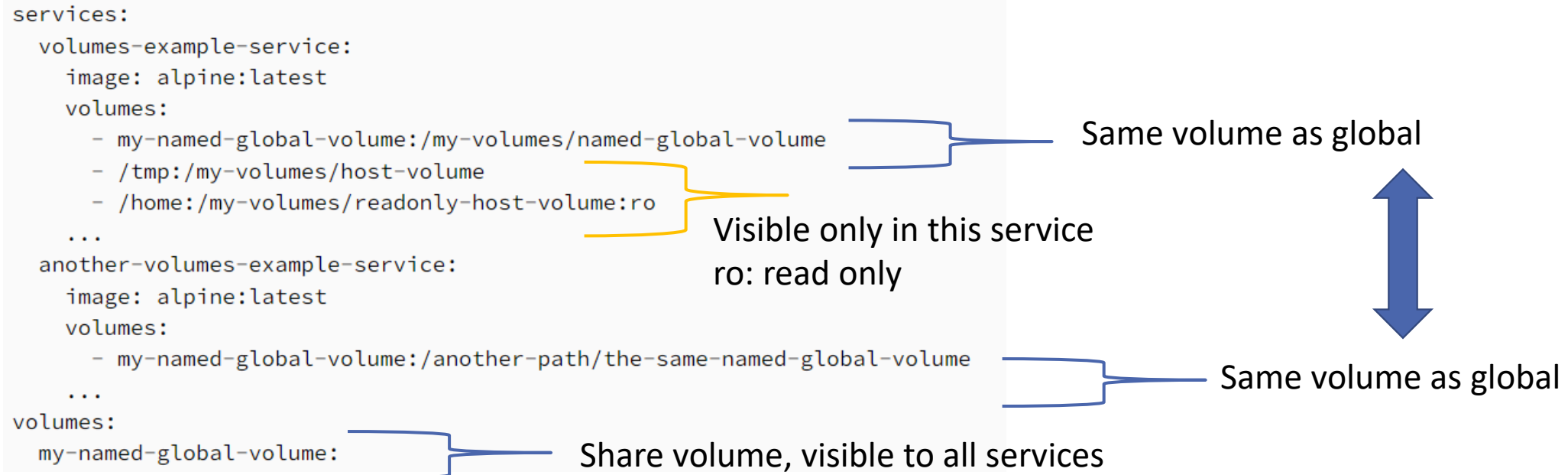
```
services:
  webapp:
    image: my-vue-app
    ports:
      - "80:80"
    networks:
      - frontend_net
      - backend_net
  db:
    image: postgres
    networks:
      - backend_net
networks:
  frontend_net:
  backend_net:
```

**frontend\_net** allow access from outside  
**backend\_net** use internally with database

Only **backend\_net** for provide service to internal network

# Docker Compose : Volume

- There are two types of volumes: **named**, and **host ones**.
- Docker manages **named volumes**, **automatically** mounting them in **self-generated directories** in the host. **Host volumes** also allow us to specify an **existing folder in the host**.
- We can configure **host volumes at the service level** and **named volumes in the outer level** of the configuration, in order to make the latter **visible to other containers** and not only to the one they belong:



# Docker Compose : Lifecycle Management

- To start for first time:

```
docker-compose up
```

- After the first time, however, we can simply use start to start the services:

```
docker-compose start
```

- In case our file has a different name than the default one (**docker-compose.yml**), we can exploit the -f and --file flags to specify an alternate file name:

```
docker-compose -f custom-compose-file.yml start
```

- Compose can also run in the background as a daemon when launched with the -d option:

```
docker-compose up -d
```

- Shutdown

```
docker-compose stop
```

```
docker-compose down <- delete everything
```

# Tutorial Roadmap

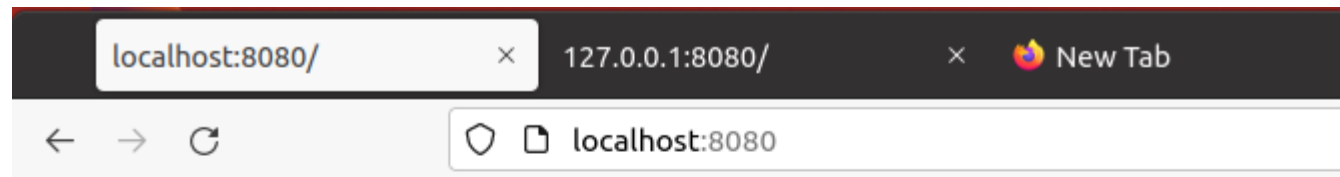
- Storage with Docker Volume
- Storage with Bind Host
- Create custom build web image with code from GitHub
- Dockerfile with build argument
- Dockerfile with environment variable
- Share your image on Docker Hub
- Deploy image on Amazon Elastic Beanstalk
- A Node.js application using Nginx, Docker and Redis
- Portainer – Docker Management
- Dashboard for monitoring dockers
- Voting App with docker-compose

# Persistent storage: with docker volume

- Run your docker with docker volume named "**myvol\_data**".

```
docker run --name myweb_v -d -p 8080:80 -v myvol_data:/usr/share/nginx/html mytestweb
```

- If everything is ok, you should be able to access the website: localhost:8080.



**My Name: Apichon Wi**

Id: 521561651561

# Persistent storage: with docker volume

- Try change web content from bash terminal
- Open another terminal and run docker exec command to enter shell of container

```
docker exec -it myweb_v bash
```

- Go to data path of nginx: `cd /usr/share/nginx/html`

```
apichon@ubuntu:~/Desktop$ docker exec -it myweb_v bash
root@57885fd4aaf5:/# cd /usr/share/nginx/html
root@57885fd4aaf5:/usr/share/nginx/html# ls
50x.html  index.html
root@57885fd4aaf5:/usr/share/nginx/html#
```

- Edit index.html by adding school information but your don't have text editor

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My Name: Your name</h1>
    <p>Id: Your ID</p>
    <p>My School: SIIT </p>
  </body>
</html>
```

# Persistent storage: with docker volume

- Since we did not include any text editor in the container, we need to **install text editor**
- In the container shell, run apt-get command to install nano editor

```
apt-get update  
apt-get install nano
```

```
# then you can edit the file  
nano index.html
```

- Modify, save and refresh the browser



# Persistent storage: with docker volume

- Let try another way by **copying local file to volume** in the container
- Create testcopy.html in the local directory (adding page title)

```
<!DOCTYPE html>
<html>
<head><title>DES424</title></head>
  <body>
    <h1>My Name: Your name</h1>
    <p>Id: Your ID</p>
    <p>My School: SIIT</p>
  </body>
</html>
```

- Copy file to container and test access via browser (**myweb\_v** is container name)

```
docker cp /home/apichon/Desktop/testcopy.html myweb_v:/usr/share/nginx/html
```

```
apichon@ubuntu:~/Desktop$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
06aef91417bd   mytestweb     "/docker-entrypoint...." 12 minutes ago Up 12 minutes  0.0.0.0:8080->80/tcp, :::8080->80/tcp  myweb_v
apichon@ubuntu:~/Desktop$ pwd
/home/apichon/Desktop
apichon@ubuntu:~/Desktop$ docker cp /home/apichon/Desktop/testcopy.html myweb_v:/usr/share/nginx/html
```



# Next Step: run new instance with existing volume

- Stop and remove the previous container
- Check if your volume still exist

```
docker stop myweb_v
docker rm myweb_v
docker volume ls
```

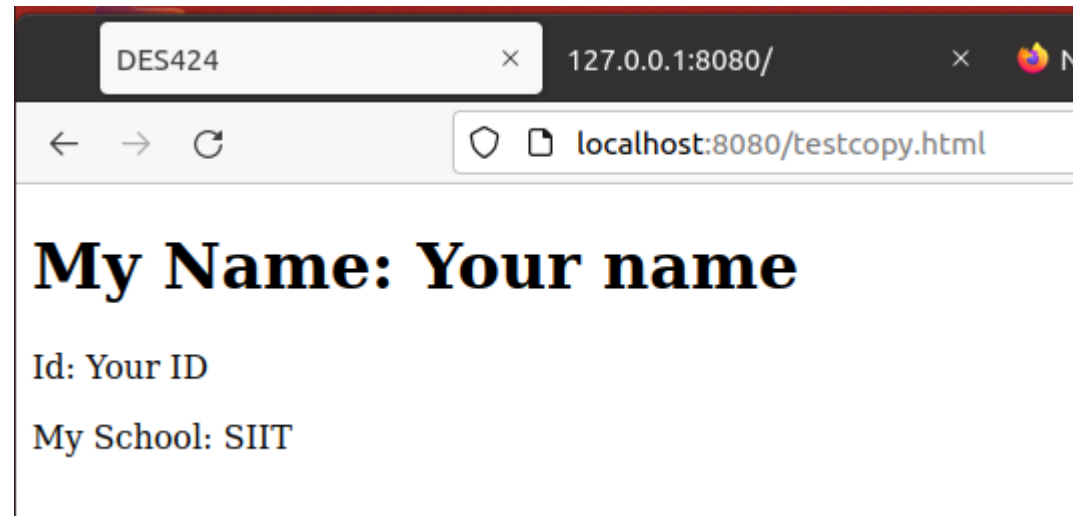
```
apichon@ubuntu:~/Desktop$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                               NAMES
57885fd4aaf5   mytestweb  "/docker-entrypoint..." 33 minutes ago Up 33 minutes  0.0.0.0:8080->80/tcp, :::8080->80/tcp myweb_v
apichon@ubuntu:~/Desktop$ docker stop myweb_v
myweb_v
apichon@ubuntu:~/Desktop$ docker rm myweb_v
myweb_v
apichon@ubuntu:~/Desktop$ docker volume ls
DRIVER      VOLUME NAME
local       myvol_data
```

# Next Step: run new instance with existing volume

- Run new container instance with the existing volume [myvol\_data] and see if your modified content still exist

```
docker run --name myweb_v -d -p 8080:80 -v myvol_data:/usr/share/nginx/html mytestweb
```

- Test access to **testcopy.html**

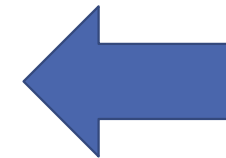
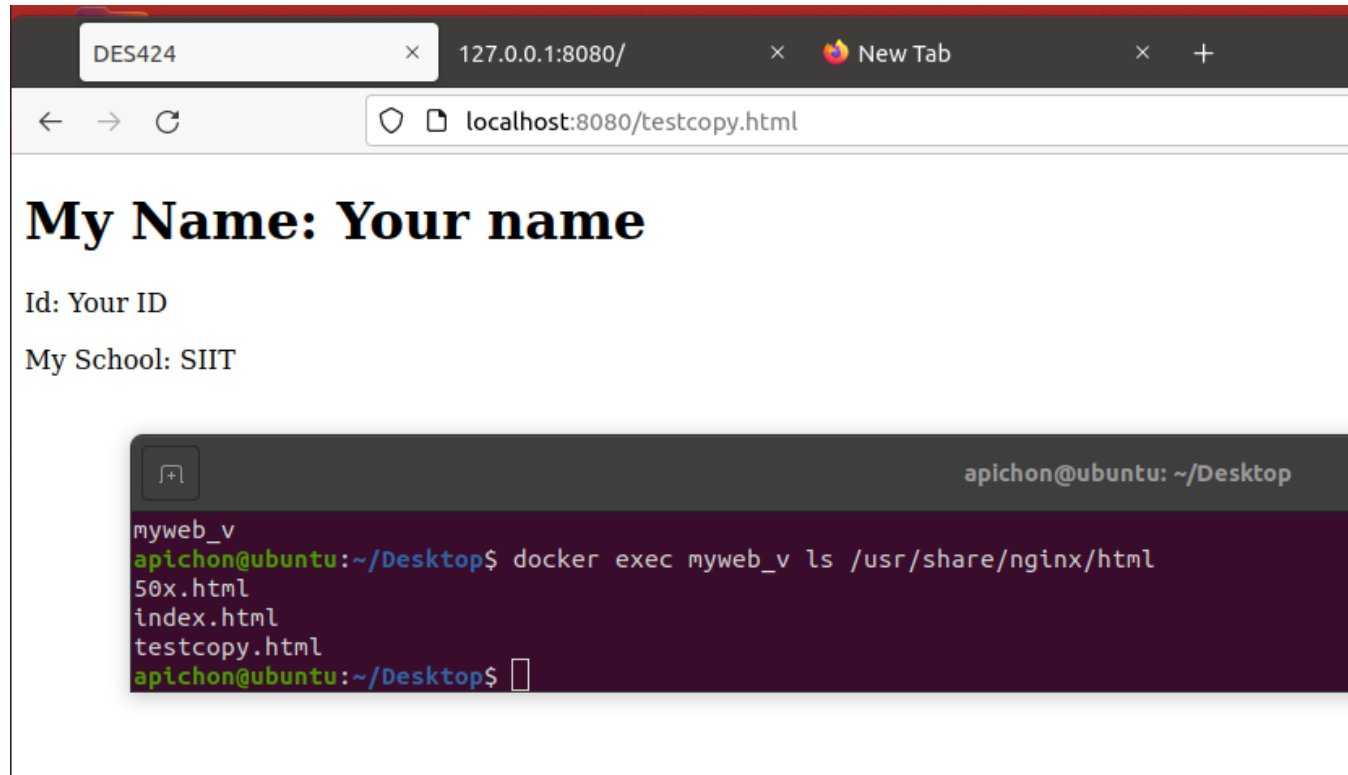


# Tutorial check point : 1

- In your container, list files in /usr/share/nginx/html

```
docker exec myweb_v ls /usr/share/nginx/html
```

- Capture screen with terminal and browser and paste it on classroom assignment



**Make sure to include both browser and terminal**

# Persistent storage: with bind mount

- You will create local directory and map that directory with a path in the container
- Create a directory named "my\_local\_vol" and create new file: index.html with the following content:

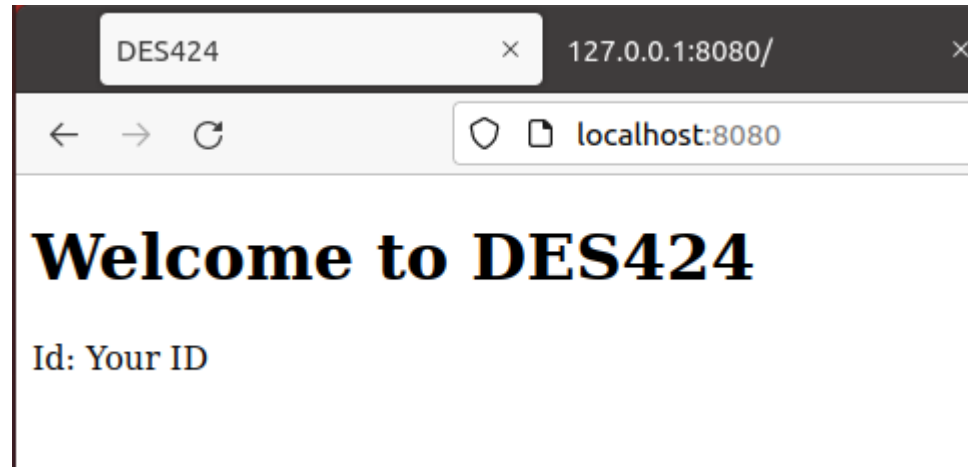
```
<!DOCTYPE html>
<html>
<head><title>DES424</title></head>
  <body>
    <h1>Welcome to DES424</h1>
    <p>Id: Your ID</p>
  </body>
</html>
```

- Run your docker image with mapping path to host: my\_local\_vol

```
docker run --name myweb_h -d -p 8080:80 -v /home/apichon/Desktop/my_local_vol:/usr/share/nginx/html mytestweb
```

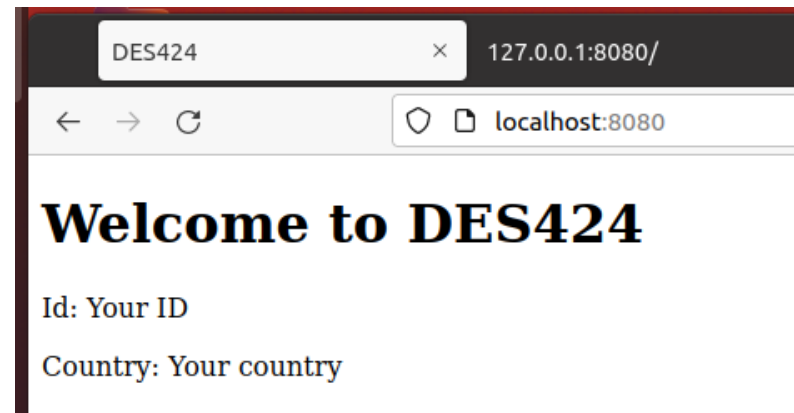
# Persistent storage: with bind mount

- Open browser with localhost:8080, you should see the content from your host file



- Let edit index.html by adding your country in your host file (my\_local\_vol)

```
<!DOCTYPE html>
<html>
<head><title>DES424</title></head>
  <body>
    <h1>Welcome to DES424</h1>
    <p>Id: Your ID</p>
    <p>Country: Your country</p>
  </body>
</html>
```

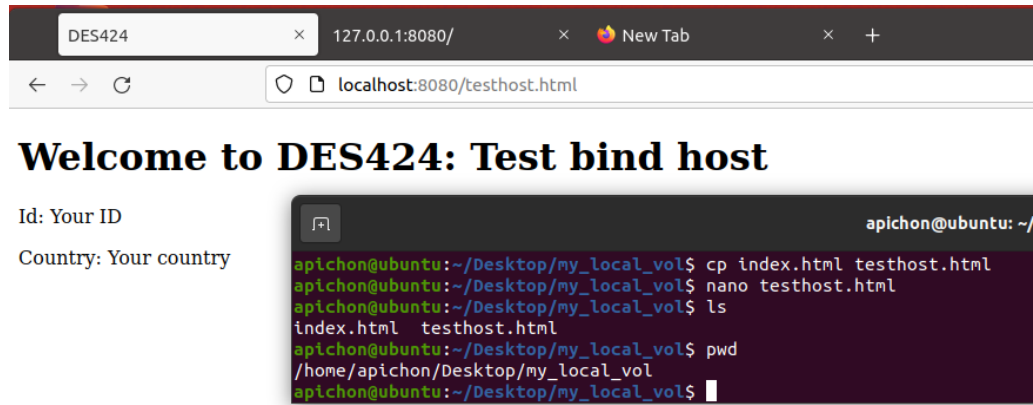


# Tutorial check point : 2

- Create new file: testhost.html in your host mapping directory (my\_local\_vol)

```
<!DOCTYPE html>
<html>
<head><title>DES424</title></head>
  <body>
    <h1>Welcome to DES424: Test bind host</h1>
    <p>Id: Your ID</p>
    <p>Country: Your country</p>
  </body>
</html>
```

- List files in directory and show the page on browser
- Capture screen with terminal and browser and paste it on classroom assignment



# Dockerfile: build with code from GitHub

- We will build a custom image for web server (nginx) with the code from GitHub
- Create directory name **my\_git\_docker** and enter that directory.
- Create a file named "Dockerfile" (no extension) with the following content and save it.

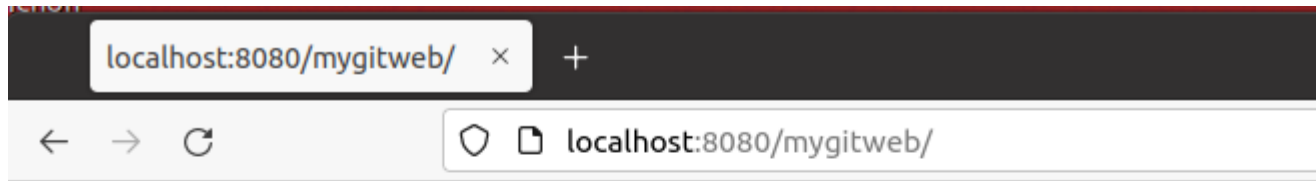
```
FROM nginx
LABEL MAINTAINER "Your name"

RUN apt-get update && apt-get install -y git
RUN mkdir /mycode && cd /mycode && git clone https://github.com/apichonsiit/testgit.git
RUN cp -r /mycode/testgit/src /usr/share/nginx/html/mygitweb
```

- **testgit** is your project did in previous assignment
- Run command to build image: `docker build -t mygitweb .`

# Dockerfile: build with code from GitHub

- Run your image: `docker run --name myweb_g -d -p 8080:80 mygitweb`
- Check your web on browser



**My First Static Web App**

**My Name: Apichon Wi**

Id: 521561651561

**Test adding message**



# Tutorial check point : 3

- In your git homework project, **add logo image** to index.html in /src and **change your name to use argument** instead.

```
<!DOCTYPE html>
<html><body>
  <h1>My Name: <MY_NAME></h1>
  <p>Id: <MY_ID></p>
  
</body></html>
```

- <MY\_NAME> and <MY\_ID> will be replaced with argument passing when running container
- New Dockerfile

```
FROM nginx
LABEL MAINTAINER "Your name"
ARG MY_NAME=change my name
ARG MY_ID=change my id
RUN apt-get update && apt-get install -y git sed
RUN mkdir /mycode && cd /mycode && git clone https://github.com/apichonsiit/testgit.git
RUN cp -r /mycode/testgit/src /usr/share/nginx/html/mygitweb
RUN export MY_NAME=$MY_NAME && sed -i 's/<MY_NAME>/"${MY_NAME}"/gi' /usr/share/nginx/html/mygitweb/index.html
RUN export MY_ID=$MY_ID && sed -i 's/<MY_ID>/"${MY_ID}"/gi' /usr/share/nginx/html/mygitweb/index.html
```

# Tutorial check point : 3

- Run command to build image:

```
docker build --no-cache --build-arg MY_NAME=Apichon --build-arg MY_ID=123456 -t mygitweb:1.1 .
```

- Capture screen with terminal and browser and paste it on classroom assignment



# Dockerfile: passing environment value

- We will build a custom image and accept custom environment value at runtime.
- Create new file on your project: check4.html

```
<!DOCTYPE html>
<html>
<body>
  <h1>Welcome to DES424: Check Point 4</h1>
  <p>My Name: Your Name</p>
  <p>My ID: Your ID</p>
  <p>Host: <MY_HOST></p>
</body>
</html>
```

- Create new Dockerfile

```
FROM nginx
LABEL MAINTAINER "Your name"
ENV MY_HOST localhost
RUN apt-get update && apt-get install -y git sed
RUN mkdir /mycode && cd /mycode && git clone https://github.com/apichonsiit/testgit.git
RUN cp -r /mycode/testgit/src /usr/share/nginx/html/mygitweb
CMD sed -i 's/< MY_HOST >/"$MY_HOST"/gi' /usr/share/nginx/html/mygitweb/check4.html
```

# Dockerfile: passing environment value

- We will build a custom image and accept custom environment value at runtime.
- Create new file on your project: check4.html

```
<!DOCTYPE html>
<html>
<body>
  <h1>Welcome to DES424: Check Point 4</h1>
  <p>My Name: Your Name</p>
  <p>My ID: Your ID</p>
  <p>Host: <MY_HOST></p>
</body>
</html>
```

- We need to run **sed** to replace text at runtime, hence, we need to use **CMD** but as inspected the image, it CMD was used for start nginx daemon. We will then need to create a shell script to run those two command.
- Create **dotask.sh** with the following scripts:

```
sed -i 's/<MY_HOST>/'"$MY_HOST"'/gi' /usr/share/nginx/html/mygitweb/check4.html
nginx -g 'daemon off;'
```

# Dockerfile: passing environment value

- Create new Dockerfile

```
FROM nginx
LABEL MAINTAINER "Your name"
ENV MY_HOST localhost
COPY dotask.sh /
RUN apt-get update && apt-get install -y git sed
RUN mkdir /mycode && cd /mycode && git clone https://github.com/apichonsiit/testgit.git
RUN cp -r /mycode/testgit/src /usr/share/nginx/html/mygitweb
CMD ["sh", "/dotask.sh"]
```

- Build Dockerfile with tag: 1.4 and run container

```
docker build --no-cache -t mygitweb:1.4
docker run -d --name myweb_g -p 8080:80 mygitweb:1.4
```



# Tutorial check point : 4

- Run container with passing MY\_HOST:

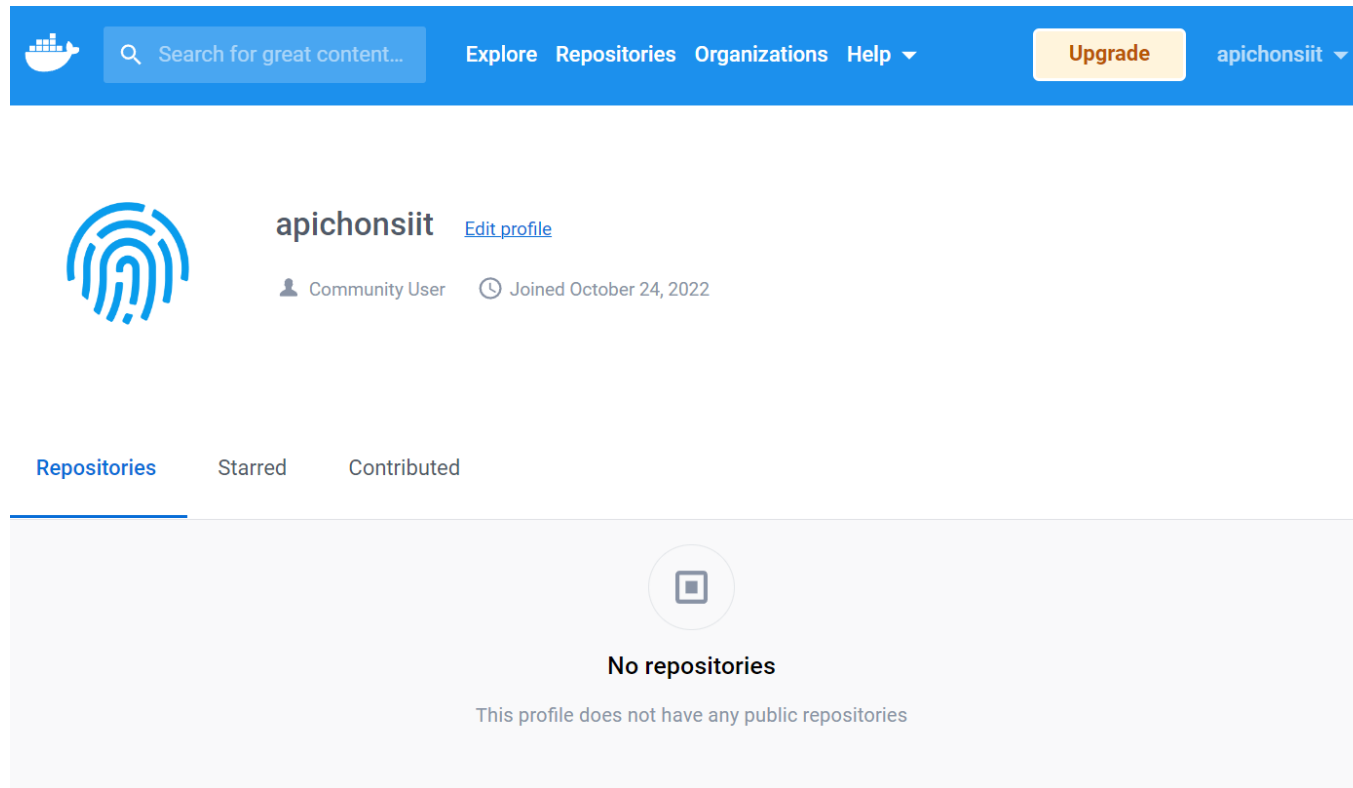
```
docker run -d --name myweb_g -e MY_HOST=MyDockerOnVM -p 8080:80 mygitweb:1.4
```

- Capture screen with terminal and browser and paste it on classroom assignment



# Push your custom image to Docker Hub

- Create an account on docker hub ([hub.docker.com](https://hub.docker.com))
- During the registration, select personal plan



# Push your custom image to Docker Hub

- In terminal, login to docker hub via command line (change username to yours)

```
docker login --username=apichonsiit
```

- It will prompt for password.

```
apichon@ubuntu:~/Desktop/check4$ docker login --username=apichonsiit
Password:
WARNING! Your password will be stored unencrypted in /home/apichon/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
apichon@ubuntu:~/Desktop/check4$
```

- Once login succeeded, try build and image again by specify docker hub registry (use same image with check point 4)

```
docker build --no-cache -t apichonsiit/mygitweb:1.4 .
```



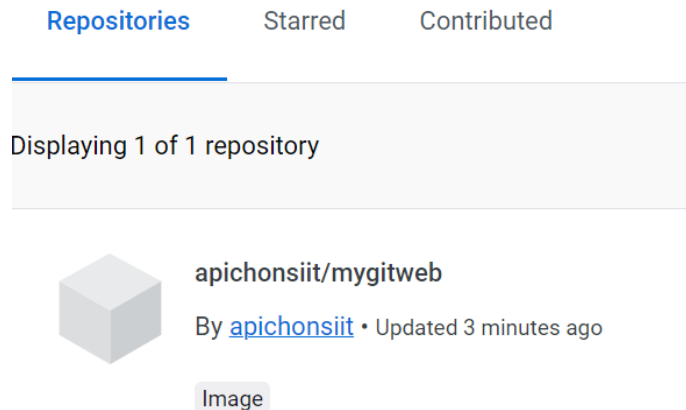
# Push your custom image to Docker Hub

- Run push command to upload image to docker hub (change username to yours)

```
docker push apichonsiit/mygitweb:1.4
```

```
apichon@ubuntu:~/Desktop/check4$ docker push apichonsiit/mygitweb:1.4
The push refers to repository [docker.io/apichonsiit/mygitweb]
c3d6fd7bb675: Pushed
e07b3b16ca1e: Pushed
c9a73ac26ba7: Pushed
f689d8732adb: Pushed
d6a3537fc36a: Mounted from library/nginx
819eb3a45632: Mounted from library/nginx
5eda6fa69be4: Mounted from library/nginx
6f4f3ce1dca0: Mounted from library/nginx
58a06a0d345c: Mounted from library/nginx
fe7b1e9bf792: Mounted from library/nginx
1.4: digest: sha256:7237bc0c17c666a3d72512cfc1ad01843cfffaf6f59ef30dec5ed05380b2cf720 size: 2405
```

- You can check your image (repository on Docker Hub)



# Deploy image on Amazon Elastic Beanstalk

- Go to AWS Academy lab learner and service for Amazon Elastic Beanstalk

Compute

## Amazon Elastic Beanstalk

### End-to-end web application management.

Amazon Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS.

**Get started**

Easily deploy your web application in minutes.

[Create application](#)

**Pricing**

There's no additional charge for Elastic Beanstalk. You pay for Amazon Web Services resources that we create to store and run your web application, like Amazon S3 buckets and Amazon EC2 instances.

**Get started**

You simply upload your code and Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, and automatic scaling to web application health monitoring, with ongoing fully managed patch and security updates. [Learn more](#)

**Benefits and features**

**Easy to get started**

Elastic Beanstalk is the simplest way to deploy and run your web application on Amazon Web Services. Elastic Beanstalk automatically handles the deployment details of capacity provisioning, load balancing, automatic scaling, and web

**Complete resource control**

You have the freedom to select the Amazon Web Services resources, such as Amazon EC2 instance types, that are optimal for your web application. Additionally, Elastic Beanstalk lets you manage and retain full control over the

**Getting started**

[Launch a web application](#)

- Click on create application

# Deploy image on Amazon Elastic Beanstalk

- Environment: Web server
- Define application name: mydockerweb
- Select Platform

Step 1  
Configure environment

Step 2  
Configure service access

Step 3 - optional  
Set up networking, database, and tags

Step 4 - optional  
Configure instance traffic and scaling

Step 5 - optional  
Configure updates, monitoring, and logging

Step 6  
Review

## Configure environment [Info](#)

### Environment tier [Info](#)

Amazon Elastic Beanstalk has two types of environment tiers to support different types of web applications.

☒ Web server environment  
Run a website, web application, or web API that serves HTTP requests. [Learn more](#)

☐ Worker environment  
Run a worker application that processes long-running workloads on demand or per

### Application information [Info](#)

Application name

Maximum length of 100 characters.

► Application tags (optional)

### Platform [Info](#)

Platform type

☒ Managed platform  
Platforms published and maintained by Amazon Elastic Beanstalk. [Learn more](#)

☐ Custom platform  
Platforms created and owned by you. This option is unavailable if you have no platforms.

Platform

Platform branch

Platform version

# Deploy image on Amazon Elastic Beanstalk

- Create Dockerrun.aws.json
- Change to your docker image

```
{
  "AWSEBDockerrunVersion": "1",
  "Image": {
    "Name": "apichonsiit/mygitweb:1.4",
    "Update": "true"
  },
  "Ports": [
    {
      "ContainerPort": 80,
      "HostPort": 80
    }
  ]
}
```

# Deploy image on Amazon Elastic Beanstalk

- Upload file created the previous slide
- Select single instance in Present and process with Next.

### Application code [Info](#)

☐ Sample application

☐ Existing version  
Application versions that you have uploaded.

☒ Upload your code  
Upload a source bundle from your computer or copy one from Amazon S3.

Version label  
Unique name for this version of your application code.

mytestapp

Source code origin. Maximum size 500 MB

☒ Local file

Upload application

☒ File name: **Dockerrun.aws.json**  
File must be less than 500MB max file size

☐ Public S3 URL

### Presets [Info](#)

Start from a preset that matches your use case or choose custom configuration to unset recommended values and use the service's default values.

Configuration presets

☒ Single instance (free tier eligible)

☐ Single instance (using spot instance)

☐ High availability

☐ High availability (using spot and on-demand instances)

☐ Custom configuration

Cancel

# Deploy image on Amazon Elastic Beanstalk

- Config Service Access, Skip to Review and submit
- Wait until deploy complete and health show ok
- Click link to access web

## Configure service access [Info](#)

### Service access

IAM roles, assumed by Elastic Beanstalk as a service role, and EC2 instance profiles allow Elastic Beanstalk to create and manage your environment. Both the IAM role and instance profile must be attached to IAM managed policies that contain the required permissions. [Learn more](#)

#### Service role

- ☐ Create and use new service role
- ☒ Use an existing service role

#### Existing service roles

Choose an existing IAM role for Elastic Beanstalk to assume as a service role. The existing IAM role must have the required IAM managed policies.

LabRole



#### EC2 key pair

Select an EC2 key pair to securely log in to your EC2 instances. [Learn more](#)

vockey



#### EC2 instance profile

Choose an IAM instance profile with managed policies that allow your EC2 instances to perform required operations.

LabInstanceProfile



Cancel

Skip to review

Previous

Next

[Elastic Beanstalk](#) > [Environments](#) > Mydockerweb2-env

## Mydockerweb2-env [Info](#)

### Environment overview

#### Health

✓ Ok

URL for your app

#### Domain

Mydockerweb2-env.eba-bn2cxmxk.us-east-1.elasticbeanstalk.com

#### Environment ID

e-xzbbvei7rk

#### Application name

mydockerweb2

#### Events

#### Health

#### Logs

#### Monitoring

#### Alarms

#### Managed updates

#### Tags

### Events (11) [Info](#)

Filter events by text, property or value

#### Time

#### Type

#### Details

October 31, 2023 00:00:09 (UTC+7)

INFO

Successfully launched environment: Mydockerweb2-env

# Deploy image on Amazon Elastic Beanstalk

- Access to app portal under configuration -> Environment Properties
- We will add environment variable: MY\_HOST = AWS Beanstalk
- Then apply

Environment properties

The following properties are passed in the application as environment properties. [Learn more](#)

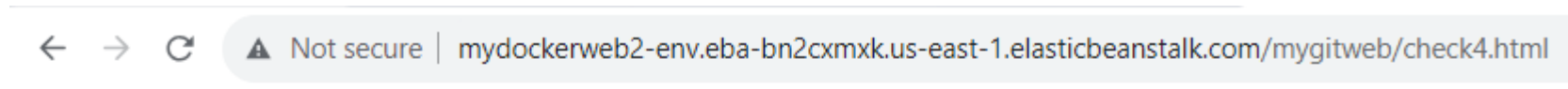
Name	Value	
MY_HOST	AWS Beanstalk	Remove

Add environment property

Cancel Continue Apply

# Tutorial check point : 5

- Access to your container app
- Capture screen with terminal and browser and paste it on classroom assignment



## Welcome to DES424: Check Point 4

My Name: Your Name

My ID: Your ID

Host: AWS Beanstalk



# Tutorial Roadmap

- Storage with Docker Volume
- Storage with Bind Host
- Create custom build web image with code from GitHub
- Dockerfile with build argument
- Dockerfile with environment variable
- Share your image on Docker Hub
- Deploy image on Azure App Service
- A Node.js application using Nginx, Docker and Redis
- Portainer – Docker Management
- Dashboard for monitoring dockers
- Voting App with docker-compose

# Install docker-compose

- Install binary file of docker-compose
- change permission of the file
- Check if it is working

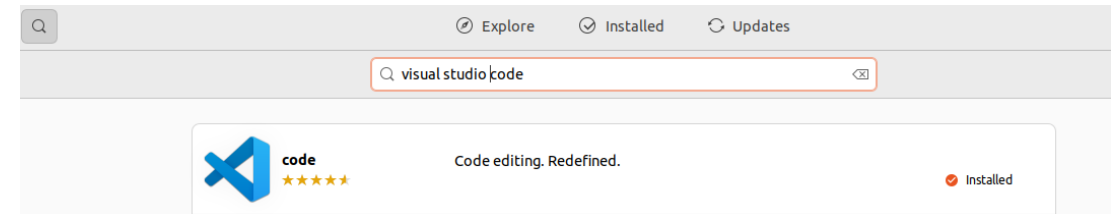
```
sudo curl -SL https://github.com/docker/compose/releases/download/v2.12.2/docker-compose-linux-x86_64 -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version
```

```
apichon@ubuntu:~$ sudo curl -SL https://github.com/docker/compose/releases/download/v2.12.2/docker-compose-linux-x86_64 -o /usr/local/bin/docker-compose
[sudo] password for apichon:
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left    Speed
  0     0     0     0     0     0      0     0  --:--:--  --:--:--  --:--:--     0
100 42.8M  100 42.8M    0     0 3240k      0  0:00:13  0:00:13  --:--:-- 2633k
```

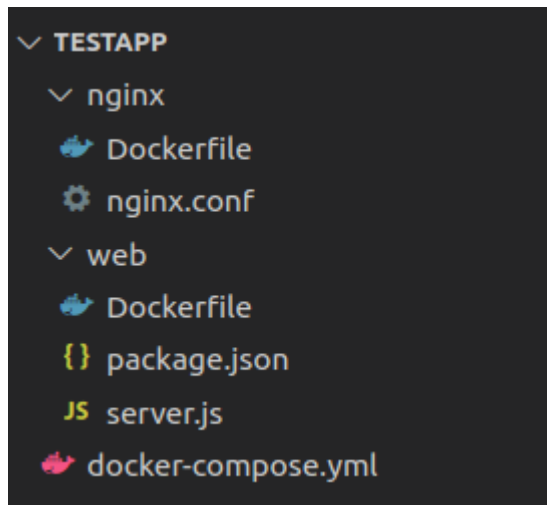
```
apichon@ubuntu:~$ sudo chmod +x /usr/local/bin/docker-compose
apichon@ubuntu:~$ docker-compose --version
Docker Compose version v2.12.2
```

# A Node.js application using Nginx, Docker and Redis

- You will see how to containerize a NodeJS web application with Redis database and Nginx as a reverse proxy in front of NodeJS app using Docker. We are going to build the following project structure
- You may install Vs code on Ubuntu.



## Project structure



- Create project structure similar to the figure.

# Step1: Create a Docker compose file

- Create an empty file with the below content and save it by name - "**docker-compose.yml**"
- The compose file defines an application with **four services redis, nginx, web1 and web2**.
- When deploying the application, docker-compose maps port 80 of the web service container to **port 80** of the host as specified in the file.

```
docker-compose.yml
1  services:
2    redis:
3      image: 'redislabs/redismod'
4      ports:
5        - '6379:6379'
6    web1:
7      restart: on-failure
8      build: ./web
9      hostname: web1
10     ports:
11       - '81:5000'
12    web2:
13      restart: on-failure
14      build: ./web
15      hostname: web2
16      ports:
17        - '82:5000'
18    nginx:
19      build: ./nginx
20      ports:
21        - '80:80'
22      depends_on:
23        - web1
24        - web2
25
```

## Step 2. Create nginx directory and add the below files:

- Under nginx folder, create nginx.conf and Dockerfile

```
nginx > ⚙ nginx.conf
1  upstream loadbalancer {
2      server web1:5000;
3      server web2:5000;
4  }
5
6  server {
7      listen 80;
8      server_name localhost;
9      location / {
10         proxy_pass http://loadbalancer;
11     }
12 }
13
```

```
nginx > 🐳 Dockerfile > ...
1  FROM nginx:1.21.6
2  RUN rm /etc/nginx/conf.d/default.conf
3  COPY nginx.conf /etc/nginx/conf.d/default.conf
```

# Step 3. Create a web directory and add the below files:

- Under web folder, create package.json, server.js and Dockerfile

```
web > JS server.js > ...
1  const os = require('os');
2  const express = require('express');
3  const app = express();
4  const redis = require('redis');
5  const redisClient = redis.createClient({
6    host: 'redis',
7    port: 6379
8  });
9
10 app.get('/', function(req, res) {
11   redisClient.get('numVisits', function(err, numVisits) {
12     numVisitsToDisplay = parseInt(numVisits) + 1;
13     if (isNaN(numVisitsToDisplay)) {
14       numVisitsToDisplay = 1;
15     }
16     res.send(os.hostname() + ': Number of visits is: ' + numVisitsToDisplay);
17     numVisits++;
18     redisClient.set('numVisits', numVisits);
19   });
20 });
21
22 app.listen(5000, function() {
23   console.log('Web application is listening on port 5000');
24 });
```

```
web > {} package.json > ...
1  {
2    "name": "web",
3    "version": "1.0.0",
4    "description": "Running Node.js and Express.js on Docker",
5    "main": "server.js",
6    "scripts": {
7      "start": "node server.js"
8    },
9    "dependencies": {
10     "express": "^4.17.2",
11     "redis": "3.1.2"
12   },
13   "author": "",
14   "license": "MIT"
15 }
```

```
web > Dockerfile > ...
1  FROM node:14.17.3-alpine3.14
2
3  WORKDIR /usr/src/app
4
5  COPY ./package.json ./
6  RUN npm install
7  COPY ./server.js ./
8
9  CMD ["npm", "start"]
10
```

# Step 4. Deploy the application

- Deploy the full-fledged app using docker-compose

```
$ docker-compose up -d
```

```
[+] Running 5/5
:: Network testapp_default    Created
:: Container testapp-web2-1    Started
:: Container testapp-redis-1   Started
:: Container testapp-web1-1    Started
:: Container testapp-nginx-1   Started
```

```
apichon@ubuntu:~/Desktop/test-compose/testapp$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
137f47610f4d	testapp-nginx	"/docker-entrypoint...."	20 seconds ago	Up 17 seconds	0.0.0.0:80->80/tcp, :::80->80/tcp	testapp-nginx-1
b3afa333c5a6	testapp-web1	"docker-entrypoint.s..."	20 seconds ago	Up 18 seconds	0.0.0.0:81->5000/tcp, :::81->5000/tcp	testapp-web1-1
bb48f3d554c6	testapp-web2	"docker-entrypoint.s..."	20 seconds ago	Up 18 seconds	0.0.0.0:82->5000/tcp, :::82->5000/tcp	testapp-web2-1
6fad19e5313a	redislabs/redismod	"redis-server --load..."	20 seconds ago	Up 18 seconds	0.0.0.0:6379->6379/tcp, :::6379->6379/tcp	testapp-redis-1

# Step 5. Monitoring Redis keys

- Install redis-tools and run redis-cli

```
sudo apt-get install redis-tools  
redis-cli
```

```
% redis-cli  
127.0.0.1:6379> monitor  
OK  
1646485507.290868 [0 172.24.0.2:34330] "get" "numVisits"  
1646485507.309070 [0 172.24.0.2:34330] "set" "numVisits" "5"  
1646485509.228084 [0 172.24.0.2:34330] "get" "numVisits"  
1646485509.241762 [0 172.24.0.2:34330] "set" "numVisits" "6"  
1646485509.619369 [0 172.24.0.4:52082] "get" "numVisits"  
1646485509.629739 [0 172.24.0.4:52082] "set" "numVisits" "7"  
1646485509.990926 [0 172.24.0.2:34330] "get" "numVisits"  
1646485509.999947 [0 172.24.0.2:34330] "set" "numVisits" "8"  
1646485510.270934 [0 172.24.0.4:52082] "get" "numVisits"  
1646485510.286785 [0 172.24.0.4:52082] "set" "numVisits" "9"  
1646485510.469613 [0 172.24.0.2:34330] "get" "numVisits"  
1646485510.480849 [0 172.24.0.2:34330] "set" "numVisits" "10"  
1646485510.622615 [0 172.24.0.4:52082] "get" "numVisits"  
1646485510.632720 [0 172.24.0.4:52082] "set" "numVisits" "11"
```



## Step 6. Testing the app

- After the application starts, navigate to `http://localhost:80` in your web browser or run

```
curl localhost:80
```

```
curl localhost:80
curl localhost:80
web1: Total number of visits is: 1
```

```
curl localhost:80
web1: Total number of visits is: 2
```

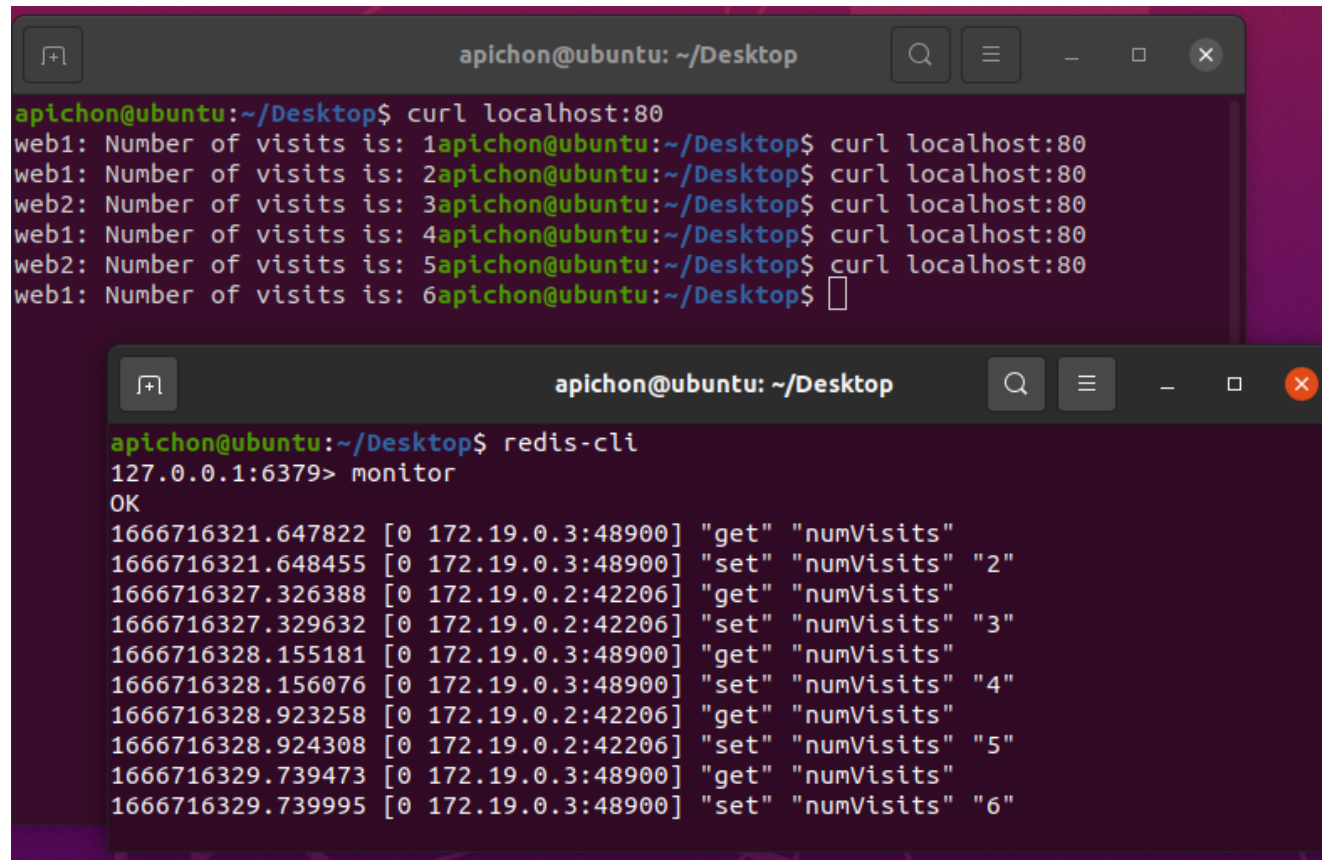
```
$ curl localhost:80
web2: Total number of visits is: 3
```

```
$ curl localhost:80
web2: Total number of visits is: 4
```

```
apichon@ubuntu:~/Desktop$ redis-cli
127.0.0.1:6379> monitor
OK
1666716321.647822 [0 172.19.0.3:48900] "get" "numVisits"
1666716321.648455 [0 172.19.0.3:48900] "set" "numVisits" "2"
1666716327.326388 [0 172.19.0.2:42206] "get" "numVisits"
1666716327.329632 [0 172.19.0.2:42206] "set" "numVisits" "3"
1666716328.155181 [0 172.19.0.3:48900] "get" "numVisits"
1666716328.156076 [0 172.19.0.3:48900] "set" "numVisits" "4"
1666716328.923258 [0 172.19.0.2:42206] "get" "numVisits"
1666716328.924308 [0 172.19.0.2:42206] "set" "numVisits" "5"
1666716329.739473 [0 172.19.0.3:48900] "get" "numVisits"
1666716329.739995 [0 172.19.0.3:48900] "set" "numVisits" "6"
```

# Tutorial check point : 6

- Capture screen as following and paste it on classroom assignment



The image shows two terminal windows from a user named 'apichon' on an 'ubuntu' machine, located in the '~/Desktop' directory. The top window shows a series of 'curl localhost:80' commands being executed, with the output of each command showing the number of visits for 'web1' and 'web2' increasing from 1 to 6. The bottom window shows a 'redis-cli' session where the 'monitor' command is used to observe Redis operations. The output shows a sequence of 'get' and 'set' commands for the key 'numVisits', with values increasing from 1 to 6.

```
apichon@ubuntu: ~/Desktop
apichon@ubuntu:~/Desktop$ curl localhost:80
web1: Number of visits is: 1apichon@ubuntu:~/Desktop$ curl localhost:80
web1: Number of visits is: 2apichon@ubuntu:~/Desktop$ curl localhost:80
web2: Number of visits is: 3apichon@ubuntu:~/Desktop$ curl localhost:80
web1: Number of visits is: 4apichon@ubuntu:~/Desktop$ curl localhost:80
web2: Number of visits is: 5apichon@ubuntu:~/Desktop$ curl localhost:80
web1: Number of visits is: 6apichon@ubuntu:~/Desktop$

apichon@ubuntu: ~/Desktop
apichon@ubuntu:~/Desktop$ redis-cli
127.0.0.1:6379> monitor
OK
1666716321.647822 [0 172.19.0.3:48900] "get" "numVisits"
1666716321.648455 [0 172.19.0.3:48900] "set" "numVisits" "2"
1666716327.326388 [0 172.19.0.2:42206] "get" "numVisits"
1666716327.329632 [0 172.19.0.2:42206] "set" "numVisits" "3"
1666716328.155181 [0 172.19.0.3:48900] "get" "numVisits"
1666716328.156076 [0 172.19.0.3:48900] "set" "numVisits" "4"
1666716328.923258 [0 172.19.0.2:42206] "get" "numVisits"
1666716328.924308 [0 172.19.0.2:42206] "set" "numVisits" "5"
1666716329.739473 [0 172.19.0.3:48900] "get" "numVisits"
1666716329.739995 [0 172.19.0.3:48900] "set" "numVisits" "6"
```

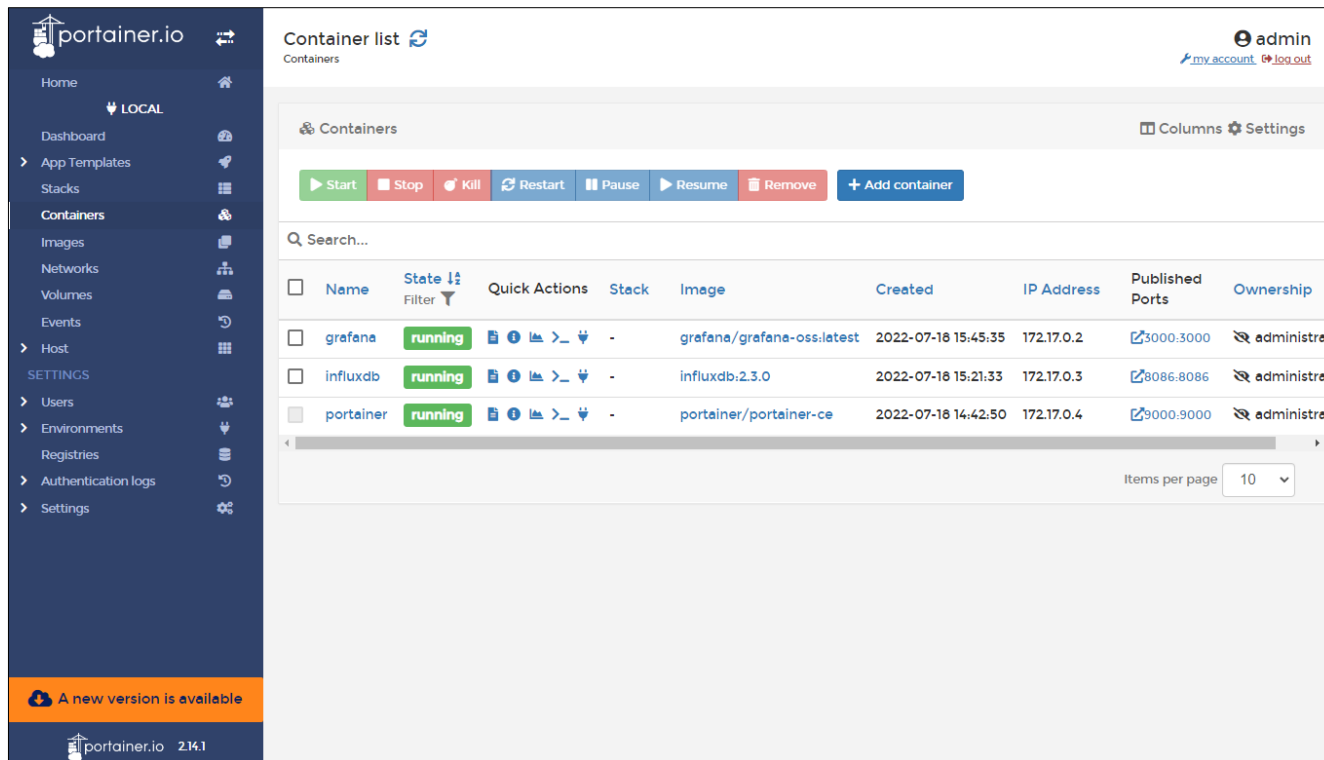
# Tutorial Roadmap

- Storage with Docker Volume
- Storage with Bind Host
- Create custom build web image with code from GitHub
- Dockerfile with build argument
- Dockerfile with environment variable
- Share your image on Docker Hub
- Deploy image on Azure App Service
- A Node.js application using Nginx, Docker and Redis
- Portainer – Docker Management
- Dashboard for monitoring dockers
- Voting App with docker-compose

# Portainer : GUI-based Container Management

<https://www.portainer.io/>

- **Portainer** is a popular Docker **UI** that helps you **visualize your containers, images, volumes** and **networks**. Portainer helps you take control of the Docker resources on your machine, avoiding lengthy terminal commands.



# Install Portainer

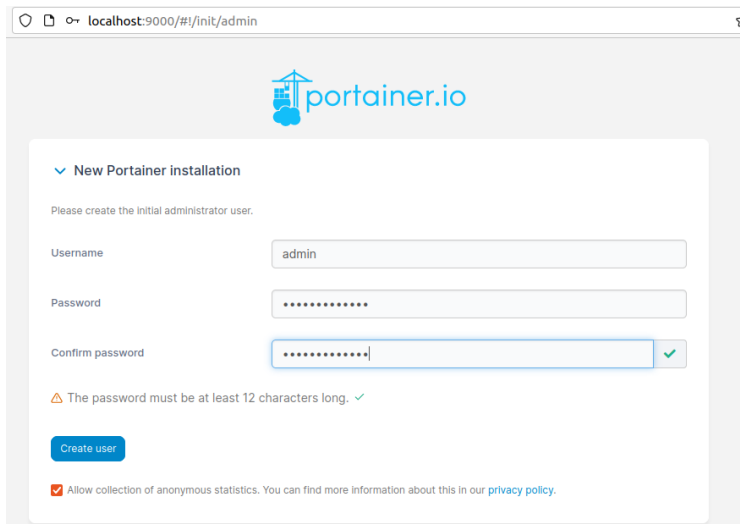
- Create docker volume for Portainer and run the container

```
docker volume create portainer_data
```

```
docker run -d -p 9000:9000 --name=portainer --restart=unless-stopped -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce
```

```
apichon@ubuntu:~/Desktop$ docker volume create portainer_data
portainer_data
apichon@ubuntu:~/Desktop$ docker run -d -p 9000:9000 --name=portainer --restart=unless-stopped -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce
```

- Then, access to portainer web via localhost:9000
- For the firsts time, you will be asked to create an account



localhost:9000/#/init/admin

portainer.io

▼ New Portainer installation

Please create the initial administrator user.

Username: admin

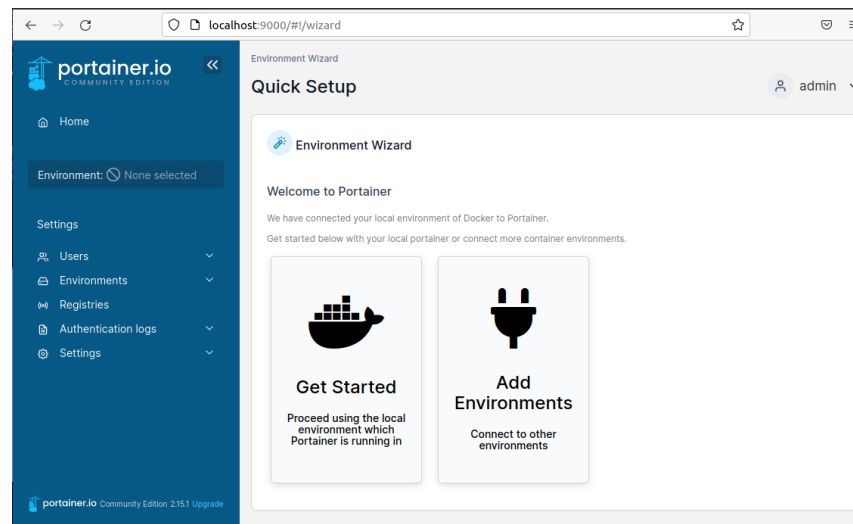
Password: .....

Confirm password: ..... ✓

⚠ The password must be at least 12 characters long. ✓

Create user

☒ Allow collection of anonymous statistics. You can find more information about this in our [privacy policy](#).



# Portainer Web: select docker daemon

- Click on home and select local, that will link to your current local docker daemon.

The screenshot displays the Portainer Web interface. On the left is a dark blue sidebar with the 'portainer.io' logo and 'COMMUNITY EDITION' text. Below the logo is a 'Home' button. Further down is a section 'Environment: None selected'. At the bottom of the sidebar is a 'Settings' section with a list of items: Users, Environments, Registries, Authentication logs, and Settings, each with a dropdown arrow.

The main content area has a light gray header with 'Environments' and 'Home' (with a refresh icon). In the top right corner of the main area is a user profile icon labeled 'admin' with a dropdown arrow.

The main content area features a section titled 'Environments' with a sub-header 'Click on an environment to manage'. Below this is a 'Refresh' button and a search bar labeled 'Search by name, group, tag, status, URL...'. There are several filter buttons: Platform, Connection Type, Status, Tags, Groups, Agent Version, and a 'Clear all' button. To the right of these filters are 'Sort By' and a list icon.

The main content area displays a single environment entry for 'local'. It includes a Docker icon, the name 'local', a green 'up' status indicator, and a timestamp '2022-10-25 18:19:40'. Below this, it shows system statistics: '1 stack', '11 containers', '5' running (green power icon), '6' stopped (red power icon), '0' pending (green heart icon), '0' failed (red heart icon), '4 volumes', and '29 Images'. At the bottom of the statistics, it shows '2 CPU', '8.3 GB RAM', '0 GPU', and 'No tags'. To the right of the statistics, it shows 'Group: Unassigned' with an edit icon, 'Standalone 20.10.19', and the socket path '/var/run/docker.sock'.

At the bottom right of the main content area, there is a 'Items per page' dropdown menu set to '10'.

# Portainer Web: Dashboard

- You can see information of your docker including images, containers, volume and networks.

The screenshot displays the Portainer Web Dashboard interface. On the left is a dark blue sidebar with the Portainer.io logo and a navigation menu. The main content area is light gray and titled 'Environment summary' and 'Dashboard'. It features a table of environment details and a grid of resource usage cards.

**Environment summary**

**Dashboard**

admin

Environment info	
Environment	local 2 8.3 GB - Standalone 20.10.19
URL	/var/run/docker.sock
GPU	none
Tags	-

**1**  
Stack

**29**  
Images 7.4 GB

**4**  
Networks

**11**  
Containers  
5 running 0 healthy 0 stopped 0 unhealthy

**4**  
Volumes

**0**  
GPUs

# Portainer Web: containers

- Select containers, it will list all the containers in dockers.

The screenshot displays the Portainer Web interface. On the left is a dark blue sidebar with the Portainer.io logo and a navigation menu. The main area is titled 'Containers' and 'Container list'. It features a table of containers with columns for Name, State, Filter, Quick Actions, Stack, Image, Created, IP Address, GPUs, Published Ports, and Ownership. The table lists several containers, including 'brave\_austin', 'charming\_hamilton', 'cranky\_lederberg', 'infallible\_roentgen', 'myweb\_g', 'portainer', 'sweet\_lichterman', 'testapp-nginx-1', 'testapp-redis-1', and 'testapp-web1-1'. The 'portainer' container is in a 'running' state, while the others are 'exited'. At the bottom right, there is a pagination control showing 'Items per page' set to 10 and a page indicator for page 1 of 2.

portainer.io  
COMMUNITY EDITION

Home

local

Dashboard

App Templates

Stacks

Containers

Images

Networks

Volumes

Events

Host

Settings

Users

Environments

Registries

Authentication logs

Settings

Containers

Container list

admin

Containers

Search...

Start Stop Kill Restart Pause Resume Remove Add container

<input type="checkbox"/>	Name ↓↑	State ↓↑	Filter	Quick Actions	Stack ↓↑	Image ↓↑	Created ↓↑	IP Address ↓↑	GPUs	Published Ports	Ownership ↓↑
<input type="checkbox"/>	brave_austin	exited			-	386321406592	2022-10-23 09:57:31	-	none	-	administrators
<input type="checkbox"/>	charming_hamilton	exited			-	84817084c2db	2022-10-23 09:56:35	-	none	-	administrators
<input type="checkbox"/>	cranky_lederberg	exited			-	84817084c2db	2022-10-23 09:54:56	-	none	-	administrators
<input type="checkbox"/>	infallible_roentgen	exited			-	317b06bdddea	2022-10-23 10:13:53	-	none	-	administrators
<input type="checkbox"/>	myweb_g	exited			-	mygitweb:1.4	2022-10-23 22:38:27	-	none	-	administrators
<input type="checkbox"/>	portainer	running			-	portainer/portainer-ce	2022-10-25 18:14:39	172.17.0.2	none	9000:9000	administrators
<input type="checkbox"/>	sweet_lichterman	exited			-	23122ec0b1f6	2022-10-23 11:30:48	-	none	-	administrators
<input type="checkbox"/>	testapp-nginx-1	running			testapp	testapp-nginx	2022-10-25 09:44:43	172.19.0.5	none	80:80	administrators
<input type="checkbox"/>	testapp-redis-1	running			testapp	redislabs/redismod	2022-10-25 09:44:43	172.19.0.3	none	6379:6379	administrators
<input type="checkbox"/>	testapp-web1-1	running			testapp	testapp-web1	2022-10-25 09:44:43	172.19.0.2	none	81:5000	administrators

Items per page 10 < 1 2 >



# Portainer Web: container detail

- Select container, it will list the detail of that container.

Containers > testapp-redis-1

## Container details

admin

### Actions

[Start](#) [Stop](#) [Kill](#) [Restart](#) [Pause](#) [Resume](#) [Remove](#) [Recreate](#) [Duplicate/Edit](#)


### Container status

ID	6c5ab7d117307e24d1431e0a385bb4a9edd076a94227c8af420830f74725e68d
Name	testapp-redis-1 <a href="#">🔗</a>
Status	🟢 Running for 13 minutes
Created	2022-10-25 09:44:43
Start time	2022-10-25 18:14:12
Container webhook ⓘ	<input type="checkbox"/> <a href="#">Business Edition Feature</a>

[Logs](#) [Inspect](#) [Stats](#) [Console](#) [Attach](#)


# Portainer Web: container detail

- Select container, it will list the detail of that container.


 Container details		
IMAGE	<a href="#">redislabs/redismod@sha256:88923bcac4adbf325ce6b4a1c9fc91daf2438950f4186242bbf4dfc43cef3dd2</a>	
PORT CONFIGURATION	0.0.0.0:6379 → 6379/tcp :::6379 → 6379/tcp	
CMD	<code>--loadmodule /usr/lib/redis/modules/redisai.so --loadmodule /usr/lib/redis/modules/redisearch.so --loadmodule /usr/lib/redis/modules/redisgraph.so --loadmodule /usr/lib/redis/modules/redistimeseries.so --loadmodule /usr/lib/redis/modules/rejson.so --loadmodule /usr/lib/redis/modules/redisbloom.so --loadmodule /usr/lib/redis/modules/redisgears.so Plugin /var/opt/redislabs/modules/rg/plugin/gears_python.so</code>	
ENTRYPOINT	redis-server	
ENV	LD_LIBRARY_PATH	/usr/lib/redis/modules
	PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
	REDISGEARS_MODULE_DIR	/var/opt/redislabs/lib/modules
	REDISGEARS_PY_DIR	/var/opt/redislabs/modules/rg
	REDISGRAPH_DEPS	libgomp1 git
	com.docker.compose.config-hash	9ff71648172ddaf66c8e7a82e91658af85c1979b05a668b4c1fadd12b12e6409
	com.docker.compose.container-number	1
	com.docker.compose.depends_on	
	com.docker.compose.image	sha256:88923bcac4adbf325ce6b4a1c9fc91daf2438950f4186242bbf4dfc43cef3dd2

# Portainer Web: container detail

- Select container, it will list the detail of that container.

 **Volumes**

Host/volume	Path in container
<a href="#">3c7d68ac5ce7667d0a3d9b885b28eb60df6c6e51f327e5118976e95785a1bacf</a>	/data

 **Connected networks**

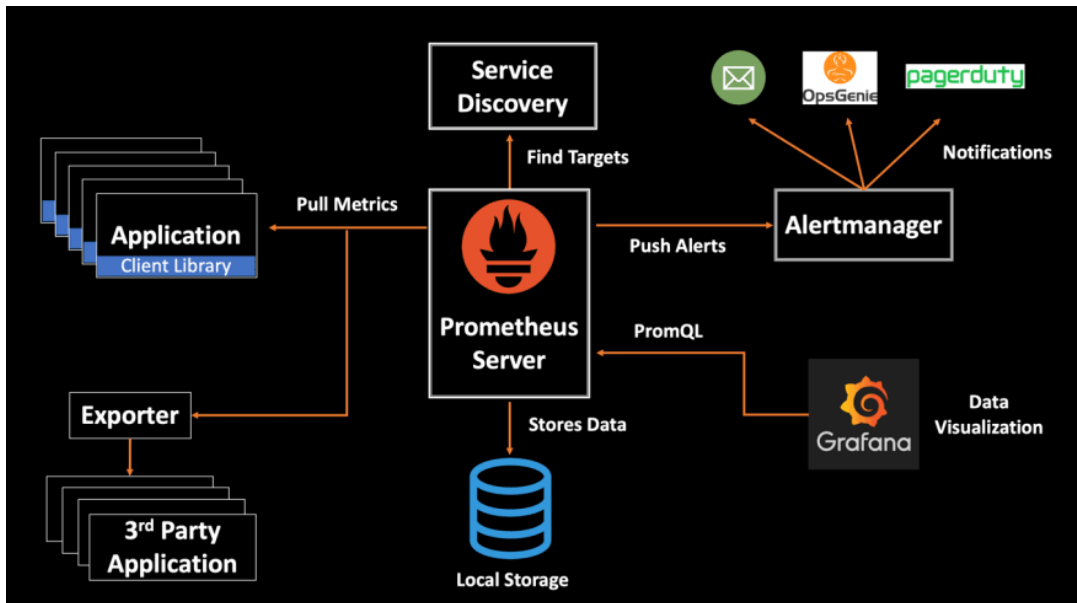
Join a network  Join network

Network	IP Address	Gateway	MAC Address	Actions
<a href="#">testapp_default</a>	172.19.0.3	172.19.0.1	02:42:ac:13:00:03	<button>Leave network</button>

Items per page 10

# Let try setup dashboard for monitoring dockers

- **Grafana** is a leading time-series, an **open-source platform for visualization** and monitoring. It allows you to query, visualize, set alerts, and understand metrics no matter where they are stored. You can create amazing dashboards in Grafana to visualize and monitor the metrics.
- **Prometheus** is an **open-source time-series monitoring system** for machine-centric and highly dynamic service-oriented architectures. It can literally monitor everything. It **integrates with Grafana very smoothly**.

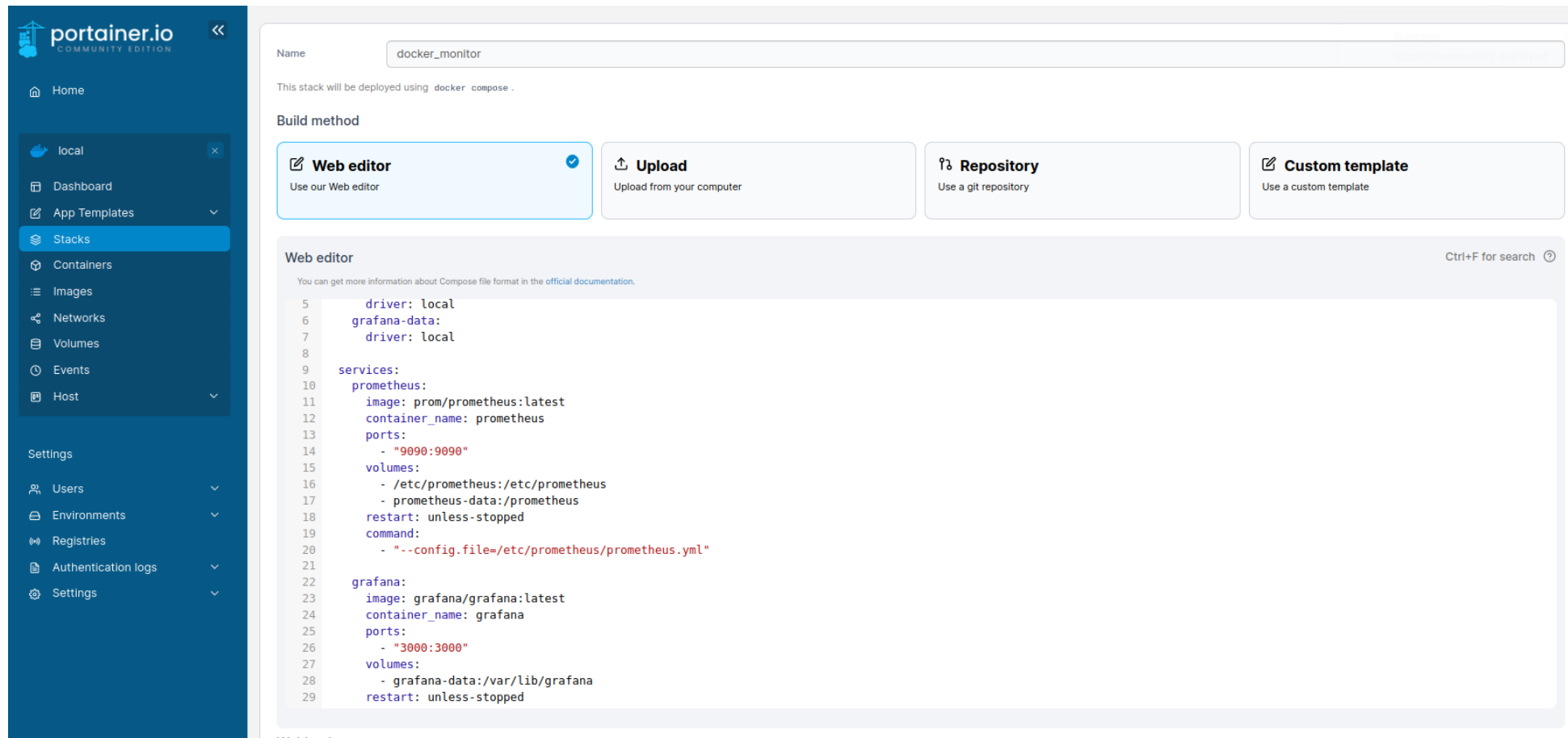


The **Prometheus server** consists of:

- **Time Series Database** that stores all the metric data like current CPU usage, memory usage etc.
- **Data Retrieval Worker** is responsible for all the data pulling activities from applications, services, servers etc. and pushing them into the database.
- **HTTP Server API** meant to accept queries for the stored data. The Server API is used to display the data in a dashboard or a Web UI.

# Step 1: Deploy Prometheus and Grafana

- Create a new stack and define or paste the content of your docker-compose file in a box web editor.



# Step 1: Deploy Prometheus and Grafana

- Copy all text from left to right to web editor of stack

```
version: '3'

volumes:
  prometheus-data:
    driver: local
  grafana-data:
    driver: local

services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    ports:
      - "9090:9090"
    volumes:
      - /etc/prometheus:/etc/prometheus
      - prometheus-data:/prometheus
    restart: unless-stopped
    command:
      - "--config.file=/etc/prometheus/prometheus.yml"

  node_exporter:
    image: quay.io/prometheus/node-exporter:latest
    container_name: node_exporter
    command:
      - '--path.rootfs=/host'
    pid: host
    restart: unless-stopped
    volumes:
      - '/:/host:ro,rslave'
```

```
cadvisor:
  image: google/cadvisor:latest
  container_name: cadvisor
  # ports:
  # - "8080:8080"
  volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:ro
    - /sys:/sys:ro
    - /var/lib/docker:/var/lib/docker:ro
    - /dev/disk:/dev/disk:ro
  devices:
    - /dev/kmsg
  restart: unless-stopped

grafana:
  image: grafana/grafana:latest
  container_name: grafana
  ports:
    - "3000:3000"
  volumes:
    - grafana-data:/var/lib/grafana
  restart: unless-stopped
```

## Step 2: Configure Prometheus

- Create directory (if not exist): /etc/prometheus
- Create file in that directory: nano /etc/prometheus/prometheus.yml

```
global:
  scrape_interval: 15s # By default, scrape targets every 15 seconds.

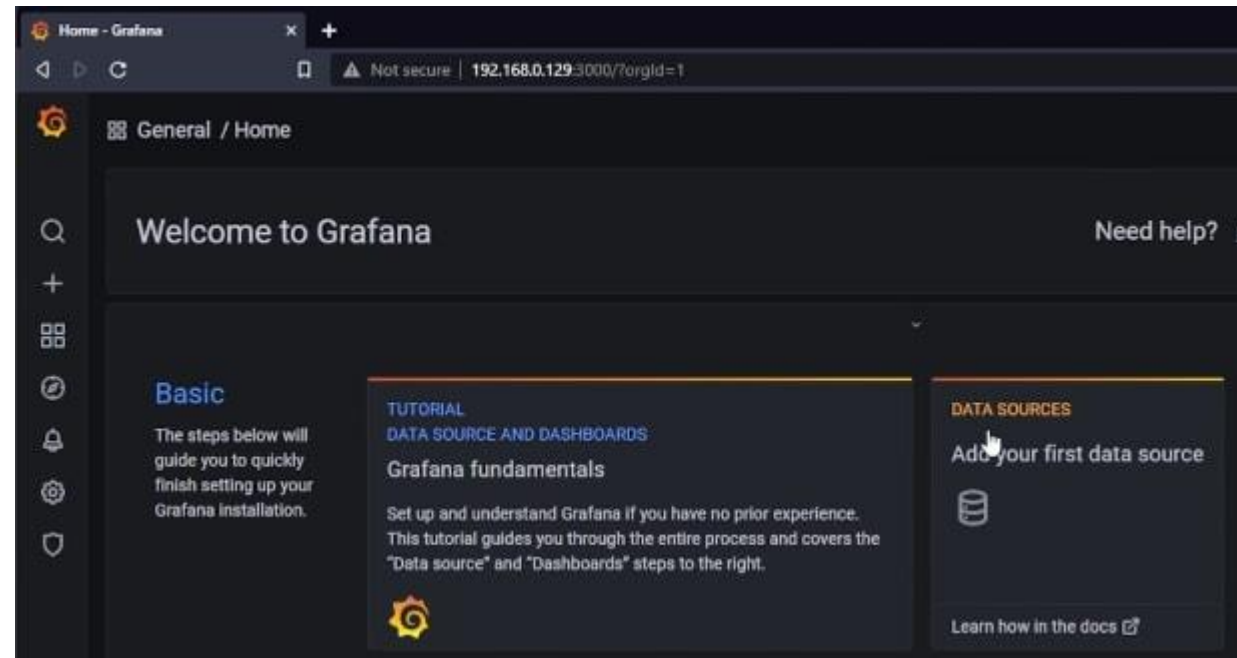
# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'
    # Override the global default and scrape targets from this job every 5 seconds.
    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:9090']
  # Example job for node_exporter
  - job_name: 'node_exporter'
    static_configs:
      - targets: ['node_exporter:9100']

  # Example job for cadvisor
  - job_name: 'cadvisor'
    static_configs:
      - targets: ['cadvisor:8080']
```

- You may need to restart prometheus container

# Step 3: Configure Grafana

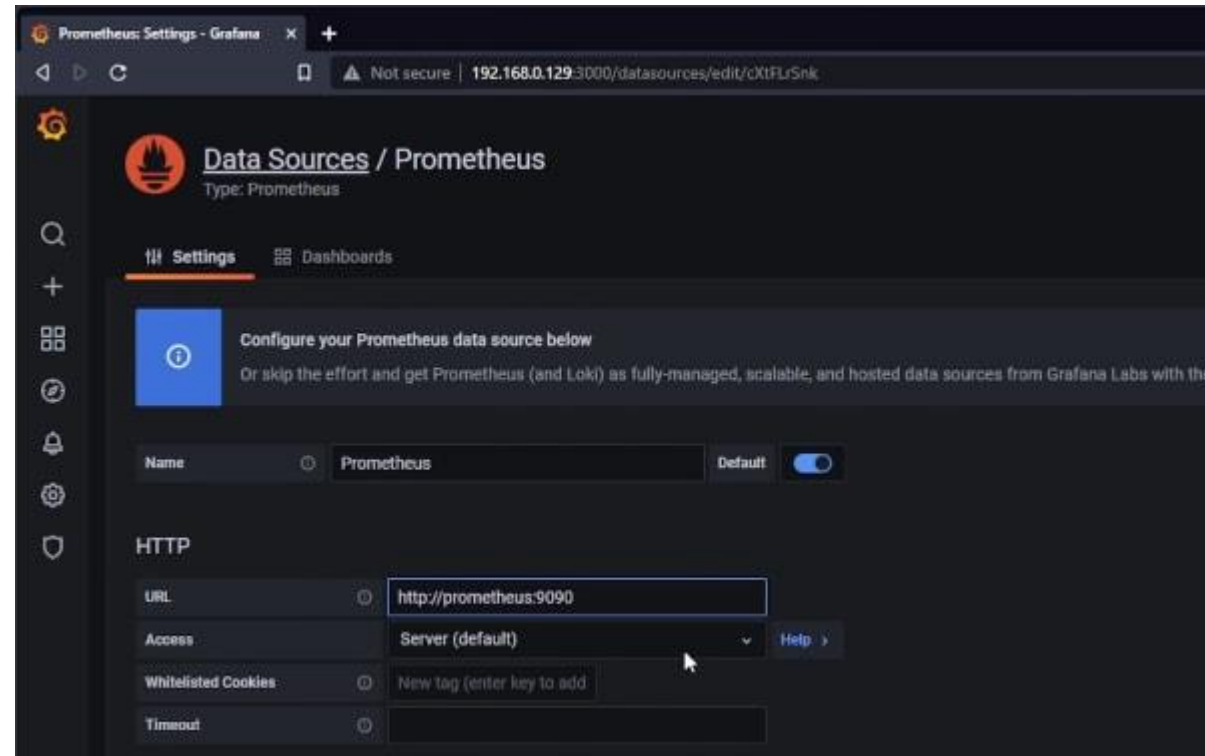
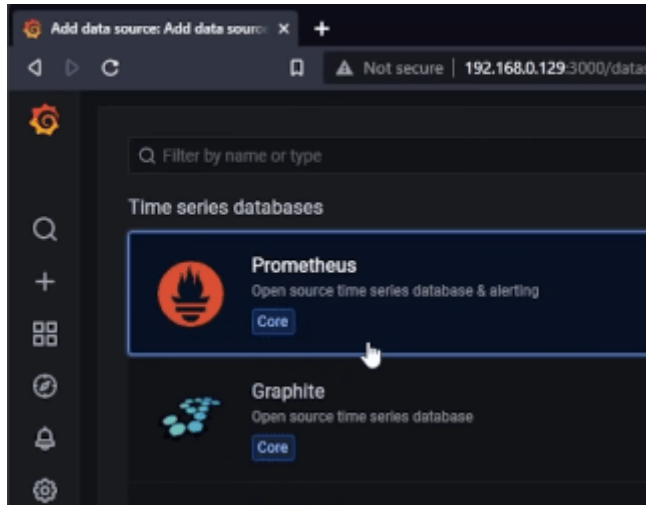
- In the Grafana page we built above at **localhost:3000**. Here we log in with the user and password as admin.
- Then select data sources to add prometheus





# Step 3: Configure Grafana

- select prometheus
- At the setting we need to reconfigure the HTTP tab at the URL is <http://prometheus:9090>



# Step 4: Import Grafana Dashboards

- In the Grafana homepage <https://grafana.com> at the Dashboards tab we will see there are many dashboards. In this tutorial we will learn about node exporters and cadvisor.

All dashboards » Node Exporter Full

**Node Exporter Full** by rtraile

DASHBOARD

Last updated: a month ago

Start with Grafana Cloud and the new FREE tier. Includes 10K series Prometheus or Graphite Metrics and 50gb Loki Logs

Downloads: 12707847

Reviews: 56

★★★★★

Add your review!

Overview Revisions Reviews

Nearly all default values exported by Prometheus node exporter graphed.

Only requires the default job\_name: node, add as many targets as you need in '/etc/prometheus/prometheus.yml'.

```
- job_name: node
  static_configs:
    - targets: ['localhost:9100']
```

Recommended for prometheus-node-exporter the arguments '--collector.systemd --collector.processes' because the graph uses some of their metrics.

Since revision 16, for prometheus-node-exporter v0.18 or newer. Since revision 12, for prometheus-node-exporter v0.16 or newer.

Available on github: <https://github.com/rfrail3/grafana-dashboards.git>

Get this dashboard:

1860

Copy ID to Clipboard

Download JSON

How do I import this dashboard?

Dependencies:

- GRAFANA 7.3.7
- GAUGE
- GRAPH
- PROMETHEUS 1.0.0
- SINGLESTAT

Server Monitoring (1860)

All dashboards » Cadvisor exporter

**Cadvisor exporter** by kokorinav

DASHBOARD

Simple exporter for cadvisor only

Last updated: a year ago

Start with Grafana Cloud and the new FREE tier. Includes 10K series Prometheus or Graphite Metrics and 50gb Loki Logs

Downloads: 1196

Reviews: 2

★★★★★

Add your review!

Overview Revisions Reviews

Simple set of several graphs. Just for getting Cadvisor's metrics only

Set includes:

- CPU Usage by container
- Memory Usage
- Memory Cached
- Received Network Traffic
- Sent Network Traffic

And a table with info:

- Label
- Working dir
- Service
- Registry Image
- Instance
- Name

Get this dashboard:

14282

Copy ID to Clipboard

Download JSON

How do I import this dashboard?

Dependencies:

- GRAFANA 7.4.5
- GRAPH
- PROMETHEUS 1.0.0
- TABLE

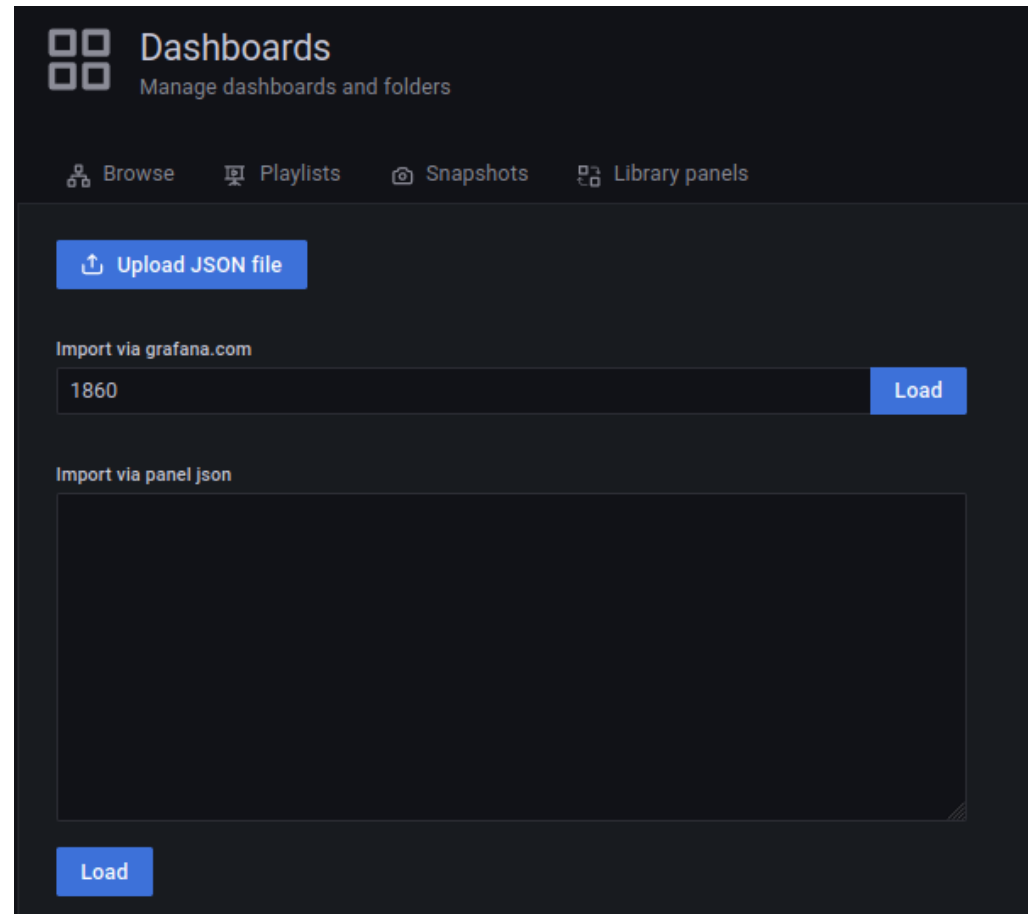
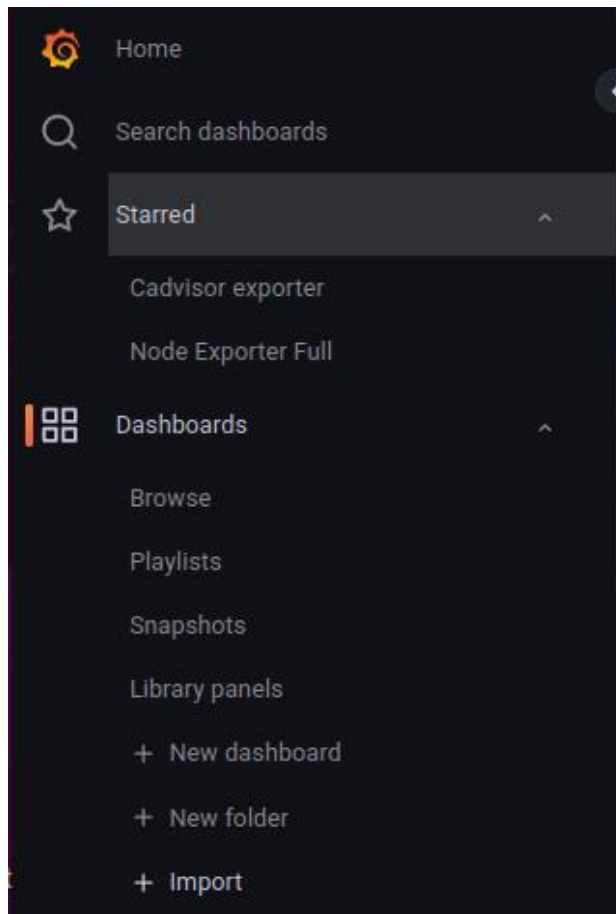
Data Sources:

- PROMETHEUS

Docker Monitoring (14282)

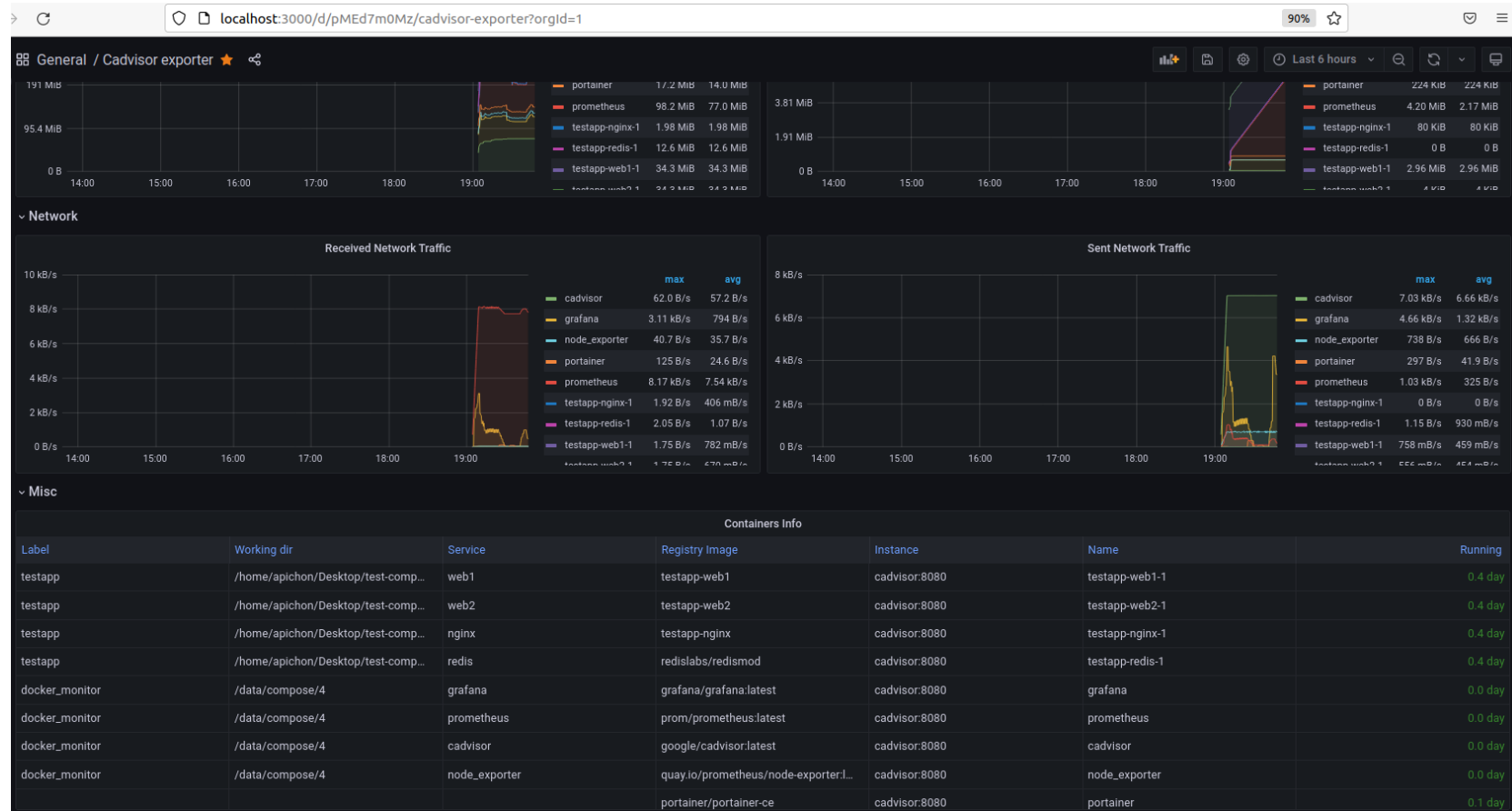
# Step 4: Import Grafana Dashboards

- On Dashboards Menu, select import
- Input dashboard id from website and click load



# Tutorial check point : 7

- Access to cadvisor-export dashboard
- Capture screen of the dashboard



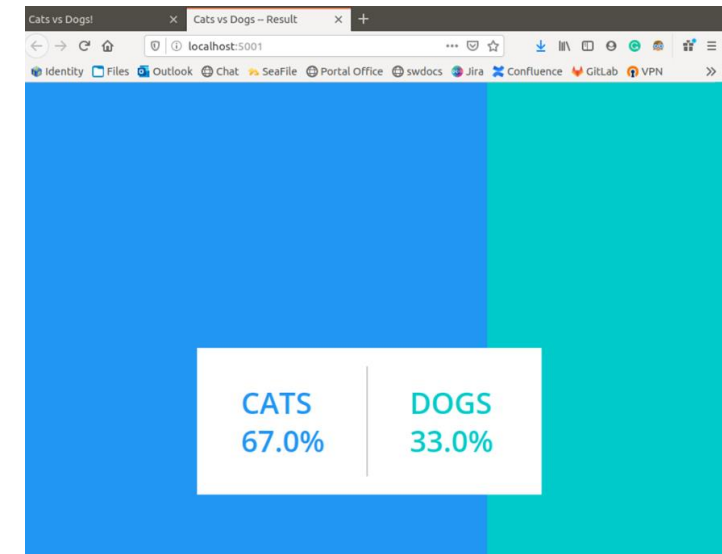
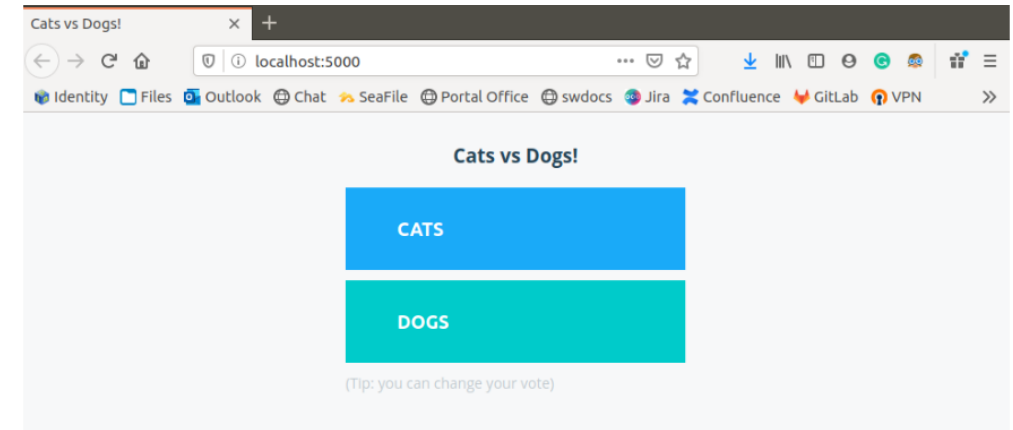
# Tutorial Roadmap

- Storage with Docker Volume
- Storage with Bind Host
- Create custom build web image with code from GitHub
- Dockerfile with build argument
- Dockerfile with environment variable
- Share your image on Docker Hub
- Deploy image on Azure App Service
- A Node.js application using Nginx, Docker and Redis
- Portainer – Docker Management
- Dashboard for monitoring dockers
- **Voting App with docker-compose**

# Next: Voting App with docker-compose

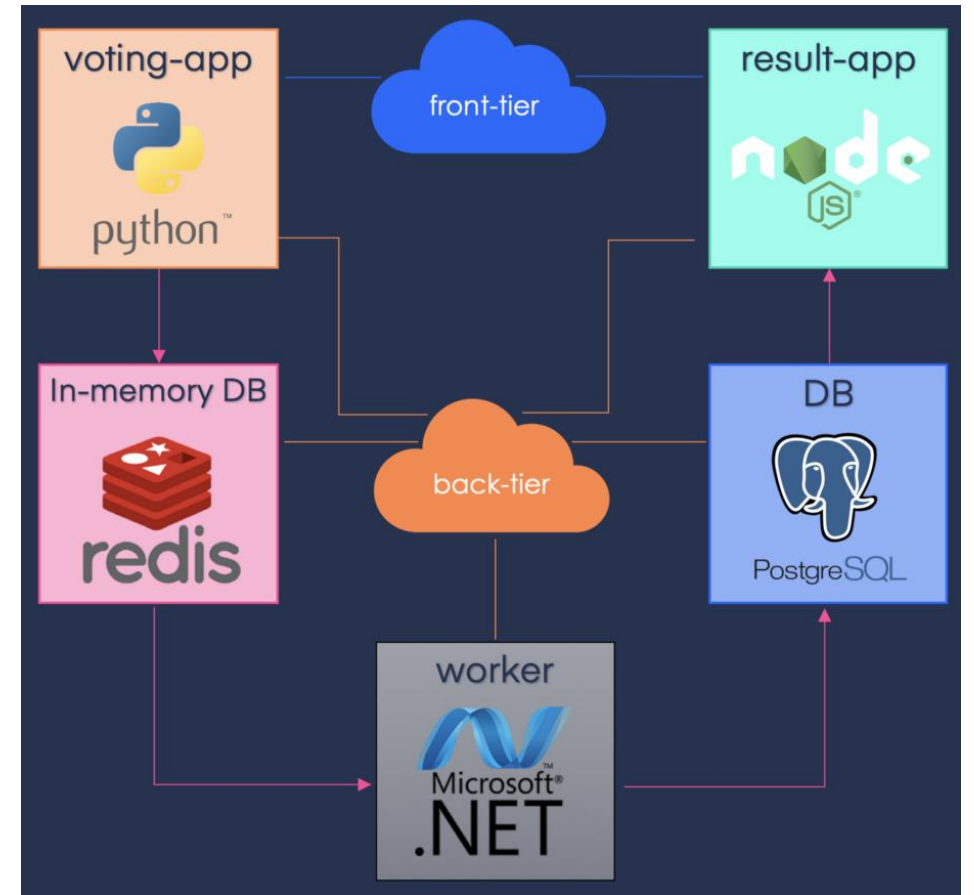
We will deploy the existing voting app from GitHub

- **Voting-app** : **Front-end** of the application written in python flask framework, which allows the users to cast their votes.
- **Redis** : An in-memory data structure store, used as a temporary database to **store the votes** casted by the users.
- **Worker** : service written in .NET, that retrieves the **votes data from redis and stores** it into **PostgreSQL** DB service.
- **PostgreSQL** DB : PostgreSQL **DB** used as persistent storage database.
- **Result-app** : service written in node js, displays the **voting results** to the user



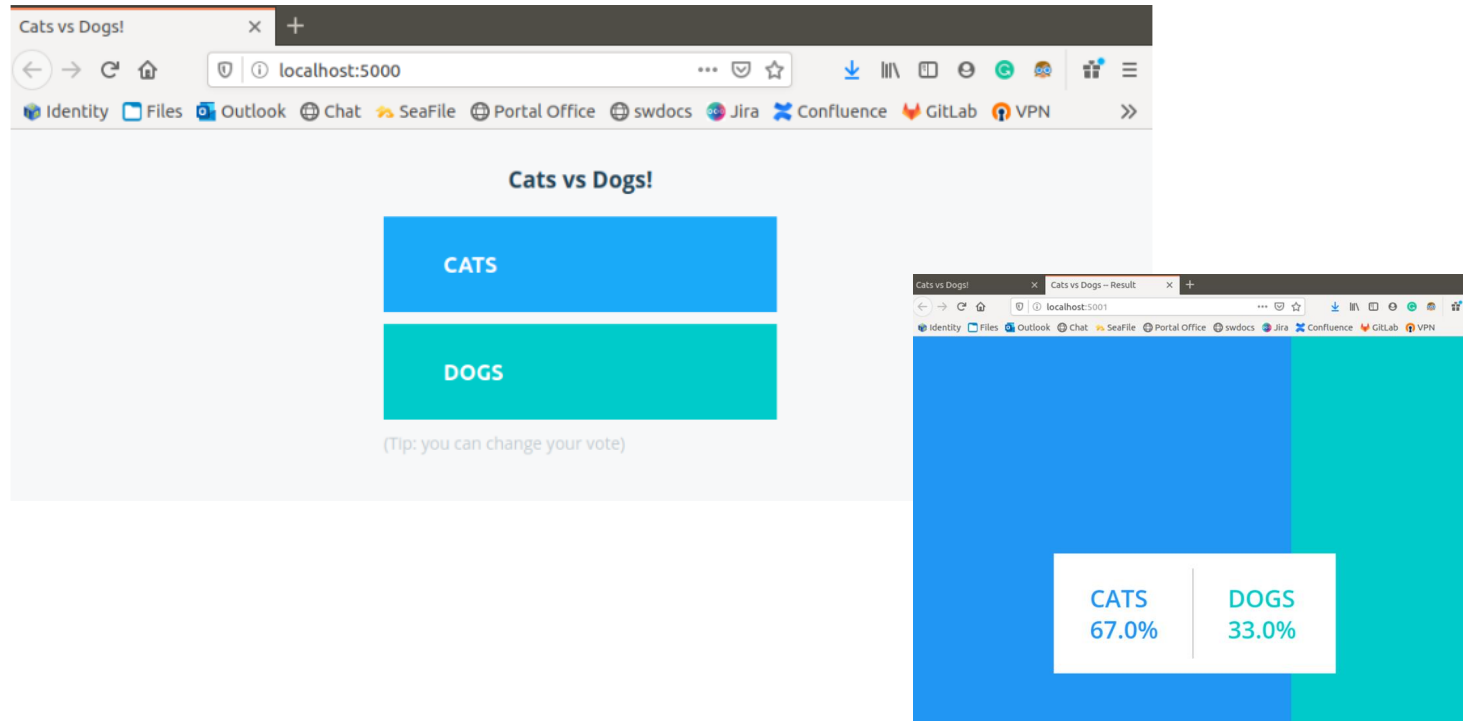
## We will deploy the existing voting app from GitHub

- **Voting-app** : **Front-end** of the application written in python flask framework, which allows the users to cast their votes.
- **Redis** : An in-memory data structure store, used as a temporary database to **store the votes** casted by the users.
- **Worker** : service written in .NET, that retrieves the **votes data from redis and stores** it into **PostgreSQL** DB service.
- **PostgreSQL** DB : PostgreSQL **DB** used as persistent storage database.
- **Result-app** : service written in node js, displays the **voting results** to the user



# Tutorial check point : 8

- Get code from GitHub and deploy
- <https://github.com/dockersamples/example-voting-app>
- Capture screen with current running container, vote web and result web





Done.

