Docker Containers

Waheed Iqbal

DevOps (Fall 2023)

Department of Data Science, FCIT, University of the Punjab
Lahore, Pakistan

Install Docker Engine on Ubuntu

- You need 64-bit version of one of these Ubuntu versions:
  - Ubuntu Lunar 23.04
  - Ubuntu Kinetic 22.10
  - Ubuntu Jammy 22.04 (LTS)
  - Ubuntu Focal 20.04 (LTS)
- Check your Ubuntu Release and Code Name using:
  - `cat /etc/*release*`
- [Install using the convenience script](#) (Click the link)
- The following two commands will install the docker on your Ubuntu machine

```
curl -fsSL https://get.docker.com -o get-docker.sh
DRY_RUN=1 sudo sh ./get-docker.sh
```


```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ DRY_RUN=1 sudo sh ./get-docker.sh
```

Install Docker Engine on Ubuntu (Cont.)

- If the installation works well, you see the following output:

```
 Version:           0.19.0
 GitCommit:         de40ad0

================================================================================

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
         documentation for details: https://docs.docker.com/go/attack-surface/

================================================================================
```

Post Installation Steps

If you want to run docker commands without sudo, add your user to the docker

To create the `docker` group and add your user:

1. Create the `docker` group.

```
$ sudo groupadd docker
```

2. Add your user to the `docker` group.

```
$ sudo usermod -aG docker $USER
```

3. Log out and log back in so that your group membership is re-evaluated.

Docker Commands

- docker run is a command used in the Docker command-line interface (CLI) to run a command in a new container.

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

- OPTIONS are various options that can be passed to the command, such as specifying ports to publish, setting environment variables, and mounting volumes.
- IMAGE is the image you want to use to create the container.
- COMMAND is the command you want to run inside the container.
- ARG are any arguments to be passed to the command.

```
wiqbal@wiqbal-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~$ sudo docker run nginx
Unable to find image 'nginx:latest' locally

latest: Pulling from library/nginx
8740c948ffd4: Downloading [================================>                  ]  19.59MB/31.4MB
d2c0556a17c5: Downloading [=================>                                  ]  8.657MB/25.47MB
c8b9881f2c6a: Download complete
693c3ffa8f43: Download complete
8316c5e80e6d: Download complete
b2fe3577faa4: Download complete
```

Docker Commands (Cont.)

- docker ps is a command used in the Docker command-line interface (CLI) to list all running containers on a system.

```
docker ps
CONTAINER ID   IMAGE   COMMAND                  CREATED          STATUS          PORTS     NAMES
796856ac413d   nginx   "nginx -g 'daemon of…"   7 seconds ago    Up 6 seconds    80/tcp    silly_sammet
```

- The docker ps -a command lists all containers, both running and stopped, on a system.

```
docker ps -a
CONTAINER ID   IMAGE   COMMAND                  CREATED          STATUS                     NAMES
796856ac413d   nginx   "nginx -g 'daemon of…"   7 seconds ago    Up 6 seconds               silly_sammet
cff8ac918a2f   redis   "docker-entrypoint.s…"   6 seconds ago    Exited (0) 3 seconds ago   relaxed_aryabhata
```

- Docker stop [NAME/ID of Container] stops the running container

```
docker stop silly_sammet
silly_sammet
```

Docker Commands (Cont.)

- Docker start [NAME/ID of Container] starts the stopped container. It will restore the file system for the container too.
- docker rm is a command used in the Docker command-line interface (CLI) to remove one or more containers.

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

```
docker rm silly_sammet
silly_sammet

docker ps -a
CONTAINER ID   IMAGE    COMMAND                  CREATED          STATUS
cff8ac918a2f   redis    "docker-entrypoint.s…"   6 seconds ago    Exited (0) 3 seconds ago
```

Docker Commands (Cont.)

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

```
docker rm silly_sammet
silly_sammet

docker ps -a
CONTAINER ID   IMAGE    COMMAND                  CREATED          STATUS
cff8ac918a2f   redis    "docker-entrypoint.s…"   6 seconds ago    Exited (0) 3 seconds ago
```

Docker Commands (Cont.)

- docker images will list local images, their repository, tags, and their size.

```
docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              f68d6e55e065        4 days ago          109MB
redis               latest              4760dc956b2d        15 months ago       107MB
ubuntu              latest              f975c5035748        16 months ago       112MB
alpine              latest              3fd9065eaf02        18 months ago       4.14MB
```

- docker rmi removes images from the host node.

```
docker rmi nginx
Untagged: nginx:latest
Untagged: nginx@sha256:96fb261b66270b900ea5a2c17a26abbfabe95506e73c3a3c65869a6dbe83223a
Deleted: sha256:f68d6e55e06520f152403e6d96d0de5c9790a89b4cfc99f4626f68146fa1dbdc
Deleted: sha256:1b0c768769e2bb66e74a205317ba531473781a78b77feef8ea6fd7be7f4044e1
Deleted: sha256:34138fb60020a180e512485fb96fd42e286fb0d86cf1fa2506b11ff6b945b03f
Deleted: sha256:cf5b3c6798f77b1f78bf4e297b27cfa5b6caa982f04caeb5de7d13c255fd7a1e
```

Docker Commands (Cont.)

- docker run downloads image from docker registry locally and then run it.

```
docker run nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
fc7181108d40: Already exists
d2e987ca2267: Pull complete
0b760b431b11: Pull complete
Digest:
sha256:96fb261b66270b900ea5a2c17a26abbfabe95506e73c3a3c65869a6dbe83223a
Status: Downloaded newer image for nginx:latest
```

- docker pull only downloads image from docker registry locally.

```
docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
fc7181108d40: Pull complete
d2e987ca2267: Pull complete
0b760b431b11: Pull complete
Digest:
sha256:96fb261b66270b900ea5a2c17a26abbfabe95506e73c3a3c65869a6dbe83223a
Status: Downloaded newer image for nginx:latest
```

Docker Commands (Cont.)

- docker exec is a command used to run a command inside an existing running container.

```
docker exec -it container_name command
```

- -i option stands for interactive, this allows you to interact with the container
- -t option allows you to have a pseudo-terminal
- container_name is the name or ID of the container to connect to
- command is the command you want to run inside the container

```
docker exec -it mycontainer /bin/bash
```

Docker Commands (Cont.)

What happened here?

```
docker exec distracted_mcclintock cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.18.0.2      538d037f94a7
```

Docker Commands (Cont.)

- Docker by default run some command inside. If there is no command defined the container terminates right after running
- For example **<u>docker run ubuntu</u>** will run ubuntu container but terminate as we do not define any command and there is no default command for this image

Docker Commands (Cont.)

The docker inspect command is used to get detailed information about Docker objects such as images, containers, networks, and volumes. By default it returns data in the form of a JSON array.

```
docker inspect blissful_hopper
```


```
[
    {
        "Id": "35505f7810d17291261a43391d4b6c0846594d415ce4f4d0a6ffbf9cc5109048",
        "Name": "/blissful_hopper",
        "Path": "python",
        "Args": [
            "app.py"
        ],
        "State": {
            "Status": "running",
            "Running": true,
        },

        "Mounts": [],
        "Config": {
           "Entrypoint": [
                "python",
                "app.py"
            ],
        },
        "NetworkSettings": {..}
      }
]
```

Docker Commands (Cont.)

The docker logs command shows information logged by a running container.

```
docker logs blissful_hopper
 This is a sample web application that displays a colored background.
 A color can be specified in two ways.

 1. As a command line argument with --color as the argument. Accepts one of
red,green,blue,blue2,pink,darkblue
 2. As an Environment variable APP_COLOR. Accepts one of
red,green,blue,blue2,pink,darkblue
 3. If none of the above then a random color is picked from the above list.
 Note: Command line argument precedes over environment variable.


No command line argument or environment variable. Picking a Random Color =blue
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

Docker Commands (Cont.)

```
$ docker run --name mynginx1 -p 80:80 -d nginx
```

where:

- `mynginx1` is the name of the created container based on the NGINX image
- the `-d` option specifies that the container runs in detached mode: the container continues to run until stopped but does not respond to commands run on the command line.
- the `-p` option tells Docker to map the ports exposed in the container by the NGINX image (port `80`) to the specified port on the Docker host. The first parameter specifies the port in the Docker host, the second parameter is mapped to the port exposed in the container

The command returns the long form of the container ID: `fcd1fb01b14557c7c9d991238f2558ae2704d129cf9fb97bb4fadf673a58580d`. This form of ID is used in the name of log files.

Verify that the container was created and is running with the `docker ps` command:

```
$ docker ps
CONTAINER ID  IMAGE         COMMAND                 CREATED         STATUS        ...
fcd1fb01b145  nginx:latest  "nginx -g 'daemon of  16 seconds ago  Up 15 seconds ...

    ... PORTS               NAMES
    ... 0.0.0.0:80->80/tcp mynginx1
```

Tags

- Docker Images are managed by tags.
- The default tag for every image is latest. However, we can also use specific tags.
- Tags are mostly used to refer a specific version.

```
docker run redis

Using default tag: latest
latest: Pulling from library/redis
f5d23c7fed46: Pull complete
Status: Downloaded newer image for redis:latest

1:C 31 Jul 2019 09:02:32.624 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 31 Jul 2019 09:02:32.624 # Redis version=5.0.5, bits=64, commit=00000000, modified=0, pid=1, just started
1:M 31 Jul 2019 09:02:32.626 # Server initialized

docker run redis:4.0
```

TAG

```
Unable to find image 'redis:4.0' locally
4.0: Pulling from library/redis
e44f086c03a2: Pull complete
Status: Downloaded newer image for redis:4.0

1:C 31 Jul 09:02:56.527 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 31 Jul 09:02:56.527 # Redis version=4.0.14, bits=64, commit=00000000, modified=0, pid=1, just started
1:M 31 Jul 09:02:56.530 # Server initialized
```

Docker Hub

docker hub | Search Docker Hub | Explore | Pricing | Sign In | Register

Filters

Products

- [ ] Images
- [ ] Extensions
- [ ] Plugins

Trusted Content

- [ ] Docker Official Image
- [ ] Verified Publisher
- [ ] Sponsored OSS

Operating Systems

- [ ] Linux
- [ ] Windows

Architectures

- [ ] ARM
- [ ] ARM 64
- [ ] IBM POWER
- [ ] IBM Z
- [ ] PowerPC 64 LE
- [ ] x86

1 - 25 of 10,000 available results.

Suggested

alpine DOCKER OFFICIAL IMAGE
Updated 15 days ago
1B+ Downloads 9.6K Stars
A minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size!
Linux 386 PowerPC 64 LE riscv64 IBM Z x86-64 ARM ARM 64

busybox DOCKER OFFICIAL IMAGE
Updated 19 days ago
1B+ Downloads 2.9K Stars
Busybox base image.
Linux IBM Z 386 mips64le PowerPC 64 LE riscv64 ARM x86-64 ARM 64

nginx DOCKER OFFICIAL IMAGE
Updated 13 days ago
1B+ Downloads 10K+ Stars
Official build of Nginx.
Linux ARM 64 386 mips64le PowerPC 64 LE IBM Z x86-64 ARM

ubuntu DOCKER OFFICIAL IMAGE
Updated 22 days ago
1B+ Downloads 10K+ Stars
Ubuntu is a Debian-based Linux operating system based on free software.

Volume Mapping

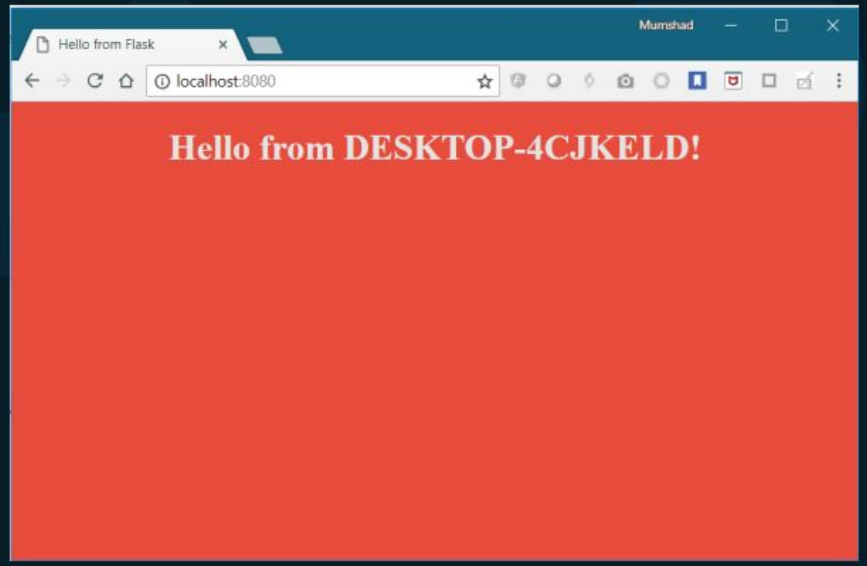
Once we remove/delete docker, the file system of the docker also destroyed. We can map a local directory to the container for avoid it.

- docker run –v /opt/datadir:/var/lib/mysql mysql
  - /opt/datadir is local directory in the host node
  - /var/lib/mysql is directory in the container

Docker Environment Variable

app.py

```python
import os
from flask import Flask

app = Flask(__name__)

…
…

color = "red"

@app.route("/")
def main():
    print(color)
    return render_template('hello.html', color=color)


if __name__ == "__main__":
    app.run(host="0.0.0.0", port="8080")
```

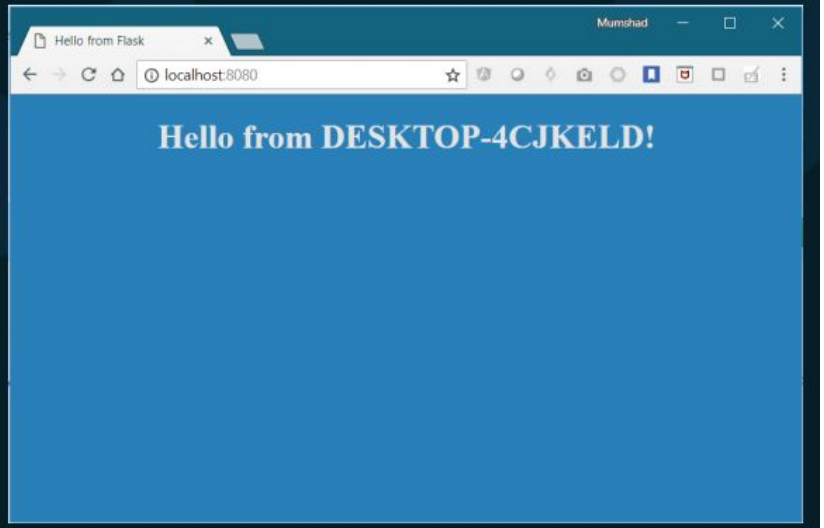



```
python app.py
```

Docker Environment Variable (Cont.)

app.py

```
import os
from flask import Flask

app = Flask(__name__)

…
…

color = os.environ.get('APP_COLOR')

@app.route("/")
def main():
    print(color)
    return render_template('hello.html', color=color)


if __name__ == "__main__":
    app.run(host="0.0.0.0", port="8080")
```



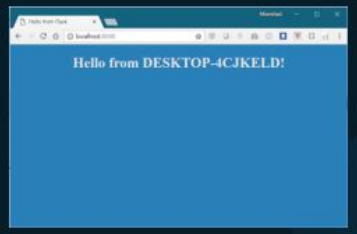

```
export APP_COLOR=blue; python app.py
```

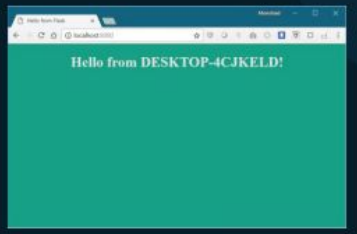
Docker Environment Variable (Cont.)

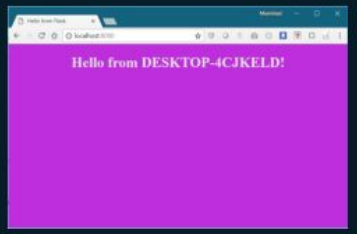
```
docker run -e APP_COLOR=blue    simple-webapp-color
```

```
docker run  -e APP_COLOR=green simple-webapp-color
```

```
docker run  -e APP_COLOR=pink simple-webapp-color
```







Docker Environment Variable (Cont.)

```
docker inspect blissful_hopper
```


```
[
    {
        "Id": "35505f7810d17291261a43391d4b6c0846594d415ce4f4d0a6ffbf9cc5109048",
        "State": {
            "Status": "running",
            "Running": true,
        },

        "Mounts": [],
        "Config": {
            "Env": [
                "APP_COLOR=blue",
                "LANG=C.UTF-8",
                "GPG_KEY=0D96DF4D4110E5C43FBFB17F2D347EA6AA65421D",
                "PYTHON_VERSION=3.6.6",
                "PYTHON_PIP_VERSION=18.1"
            ],
            "Entrypoint": [
                "python",
                "app.py"
            ],
      }
]
```

Task: Getting Comfortable With Dockers

1. Install docker engine on your machine
2. Run **jenkins** container on your machine
3. Access it through internal IP on the default port
4. Re-run the container with port mapping and access it through external IP and a custom port

Building Custom Docker Images

- A Dockerfile is a script that contains instructions for building a Docker image.
- It is used to automate the process of creating a container image.
- It typically includes information about the base image, any additional software that needs to be installed, and any configurations or environment variables that need to be set.
- The basic syntax for Dockerfile is:
  INSTRUCTION arguments

Dockerfile

The most commonly used instructions in Dockerfile are:

- FROM: specifies the base image for the build
- RUN: runs a command to install software or make other changes to the image
- COPY: copies files or directories from the host machine to the image
- ENV: sets environment variables
- EXPOSE: specifies the ports that the container will listen on
- CMD: specifies the command that will be run when a container is started from the image
- ENTRYPOINT: instruction sets the command that will be executed when the container is started from the image.

Unlike the CMD instruction, the ENTRYPOINT instruction does not get overridden when additional command-line arguments are passed to the docker run command

Dockerfile (Cont.)

Dockerfile

INSTRUCTION ARGUMENT

Dockerfile

```
FROM Ubuntu

RUN apt-get update
RUN apt-get install python

RUN pip install flask
RUN pip install flask-mysql

COPY . /opt/source-code

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

Start from a base OS or another image

Install all dependencies

Copy source code

Specify Entrypoint

Dockerfile (Cont.)

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
```

Layer 1. Base Ubuntu Layer	120 MB
Layer 2. Changes in apt packages	306 MB
Layer 3. Changes in pip packages	6.3 MB
Layer 4. Source code	229 B
Layer 5. Update Entrypoint with “flask” command	0 B

```
root@osboxes:/root/simple-webapp-docker # docker history mmumshad/simple-webapp
IMAGE          CREATED          CREATED BY                                      SIZE      COMMENT
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
```

1. Create app.py and Dockerfile in a folder
2. Then create a subfolder templates and then under that hello.html
3. Run the following command:
   docker build . -t myimage
4. If successfully image created, check docker images and then run a container from this image.
5. Check IP and access the URL on web browser
6. Change the code to read background color from environment variable, rebuild the image, and run the container.

Dockerfile

```
FROM ubuntu

# Install any needed packages specified in requirements.txt

RUN apt-get update

RUN apt-get install -y python3 python3-pip

RUN pip install flask

COPY . /app

# Make port 8001 available to the world outside this container

EXPOSE 8001

ENV FLASK_APP=/app/app.py

ENTRYPOINT ["flask", "run", "--host=0.0.0.0", "--port=8001"]
```

app.py

```
import os
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def hello_world():
    background_color = 'pink'  #

    return render_template('hello.html',
background_color=background_color)

if __name__ == '__main__':
    app.run(host="0.0.0.0", port="8001")
```

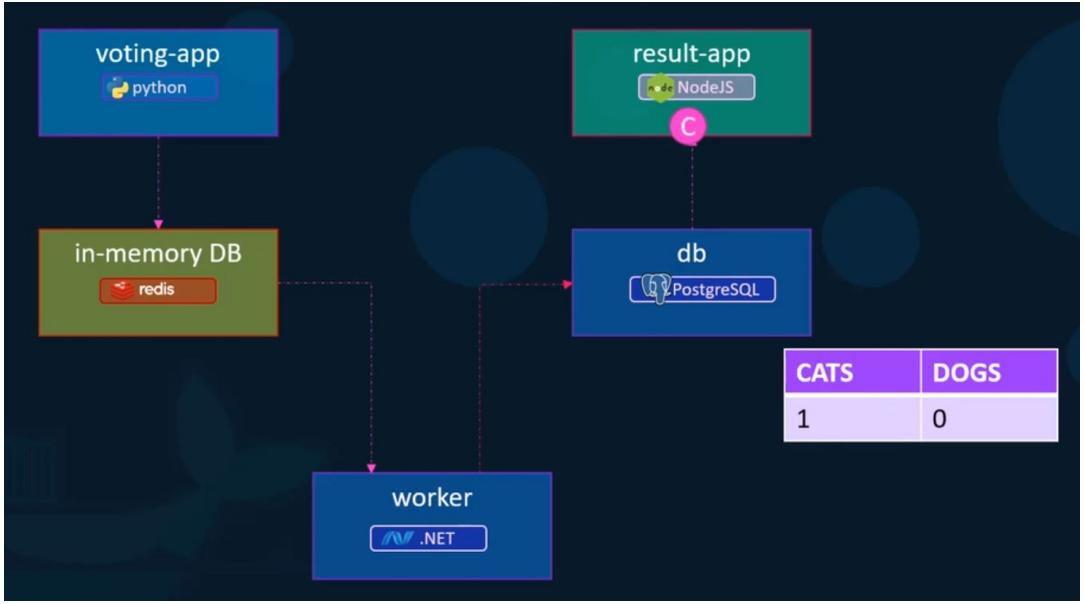
templates/hello.html

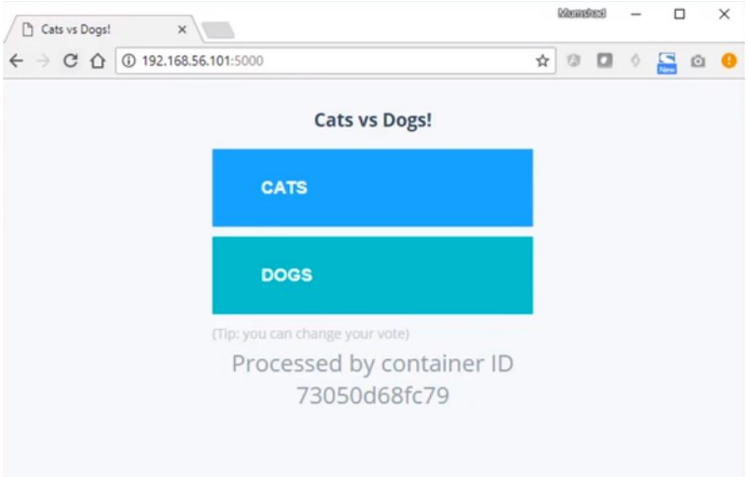
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Hello World</title>
    <style>
        body {
            background-color: {{ background_color }};
        }
    </style>
</head>
<body>
    <h1>Hello, World!</h1>
</body>
</html>
```

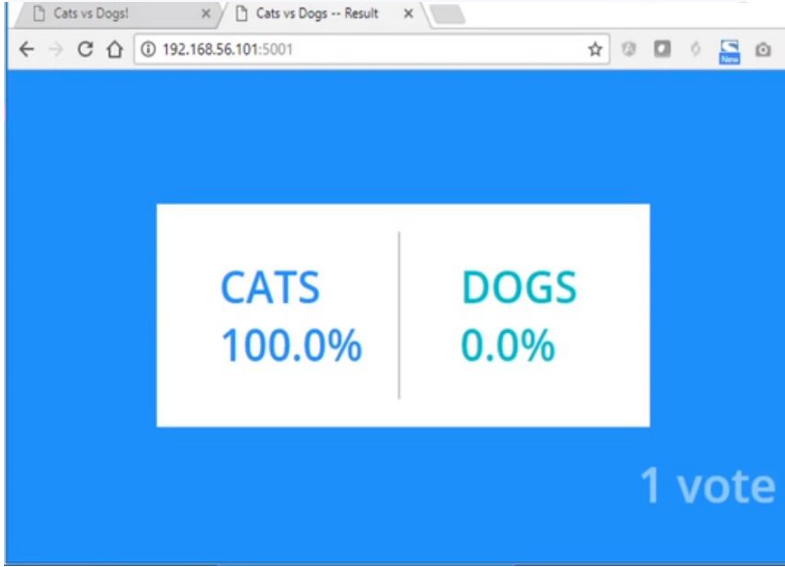
Docker Compose

- Docker Compose is a tool for defining and running multi-container Docker applications.
- It is used to define the services, networks and volumes for the application, and then starts and stops all of the containers with a single command.
- The configuration for the application is defined in a docker-compose.yml file, which specifies the services, their dependencies, and their configuration.
- This allows developers to easily manage and scale their applications, and makes it easy to run the application in different environments.
- It also provides a way to run multiple container applications together, by grouping all the services in one compose file.

Docker Compose (Cont.)

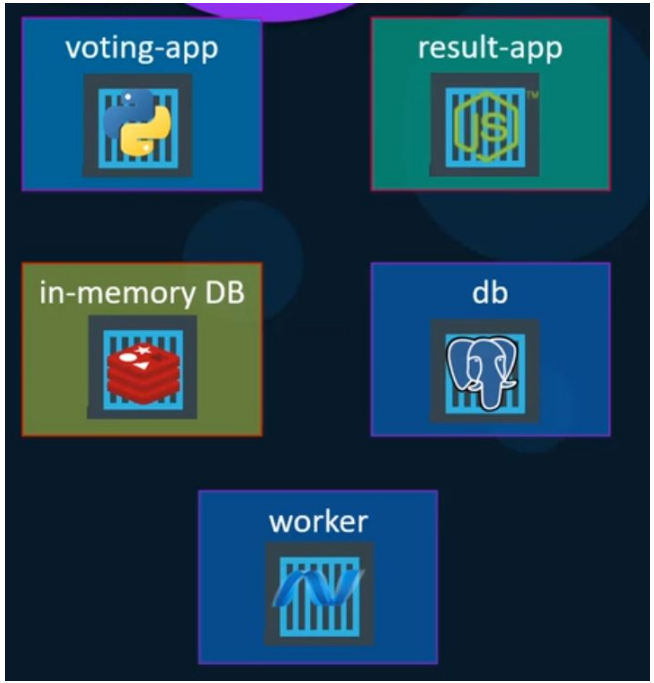






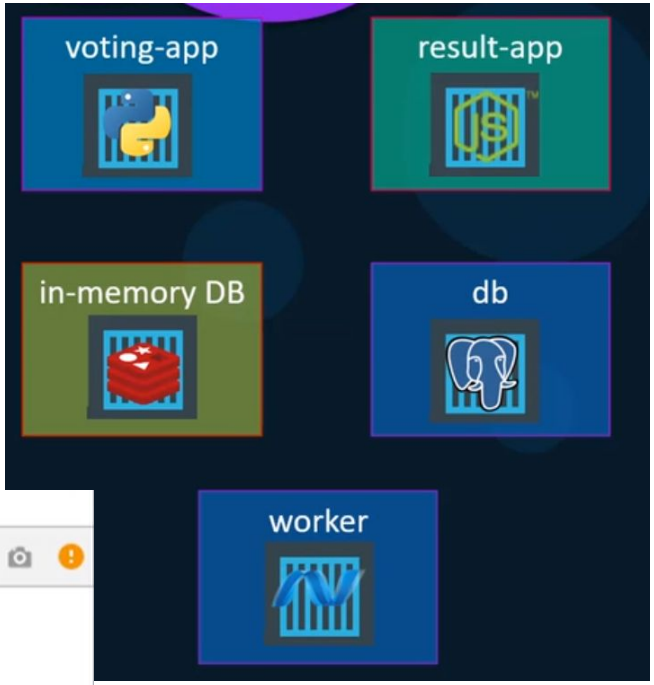
Docker Compose (Cont.)

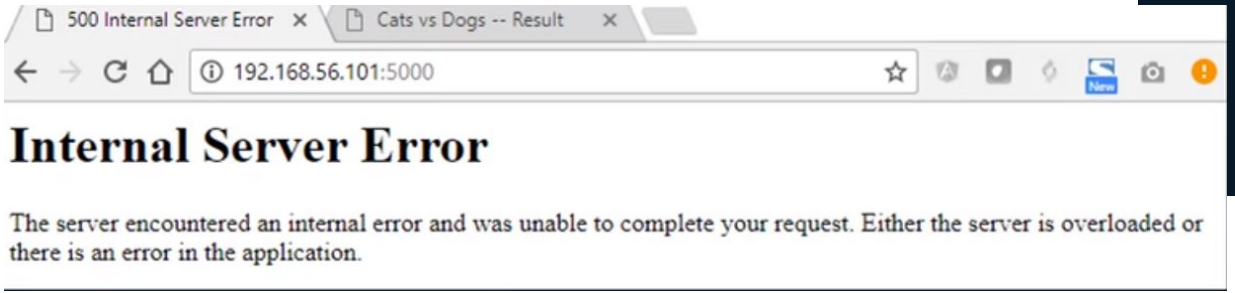
```
docker run -d --name=redis redis

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 Compose (Cont.)

```
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 Compose (Cont.)

```
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
```

```
/app # cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      redis 89cd8eb563da
```

- The --link option in docker run allows you to link one container to another. Syntax:

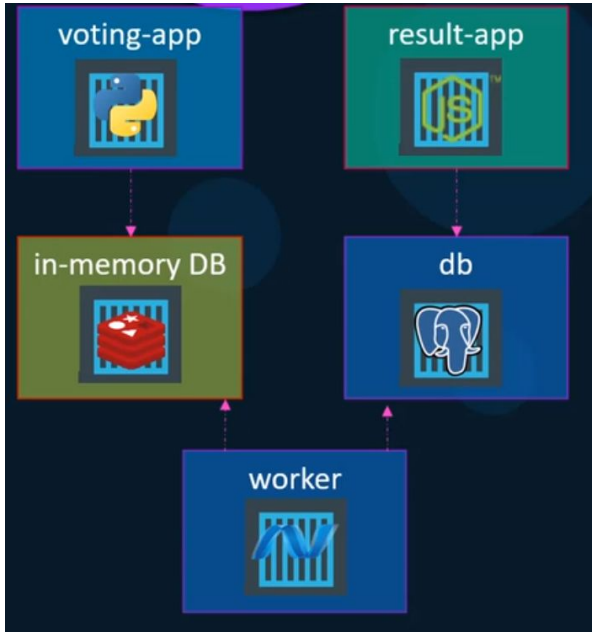
  docker run --link <name/id of container to link>:<alias in the running container> <image name>

- --link redis:redis in the example is linking voting-app with redis container whereas redis is also used alias in voting-app to access the redis container.

Docker Compose (Cont.)

```
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.err.println("Watching vote queue");
```



Docker Compose (Cont.)

```
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
```

db:db = db

- First time, you need to install docker-compose. You can do using the following two commands:
  - sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
  - sudo chmod +x /usr/local/bin/docker-compose

```
docker-compose up
```

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
```

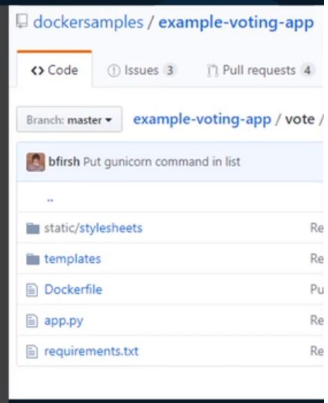
Docker Compose (Cont.)

docker-compose.yml

```yaml
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

```yaml
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
```



If the image is not in the registry. We can instruct docker compose to build the image from a local folder which contain source code of the app and Dockerfile

Docker Compose (Cont.)

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
```

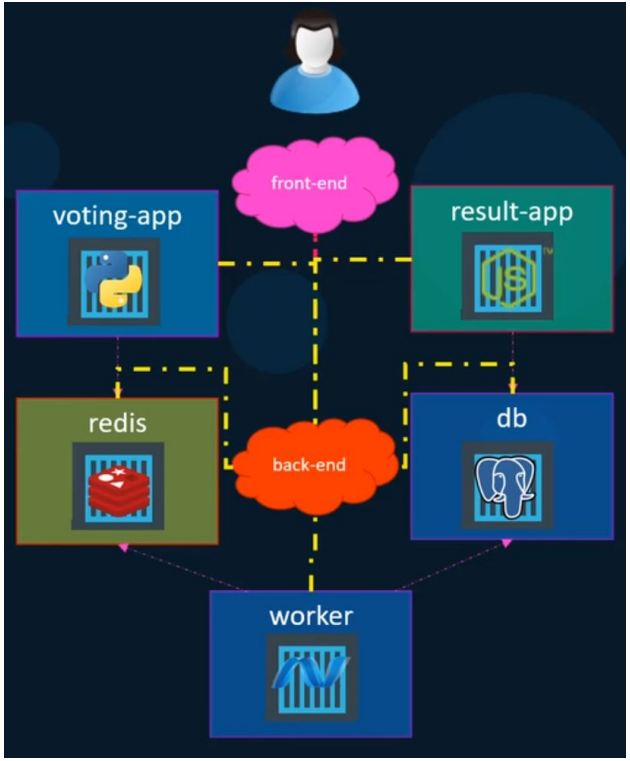
version: 3

The latest is version 3. All the features of version 2 are available in it. We do not need to specify links and it can support multiple networks. We mostly use version 2.

Docker Compose (Cont.)

docker-compose.yml

```yaml
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:
```



Assignment

- Run voting app using docker swarm. The code is available at: https://github.com/dockersamples/example-voting-app

The submission details and deadline will be announced on classroom!

Credit

Part of slides are taken from KodeKloud!