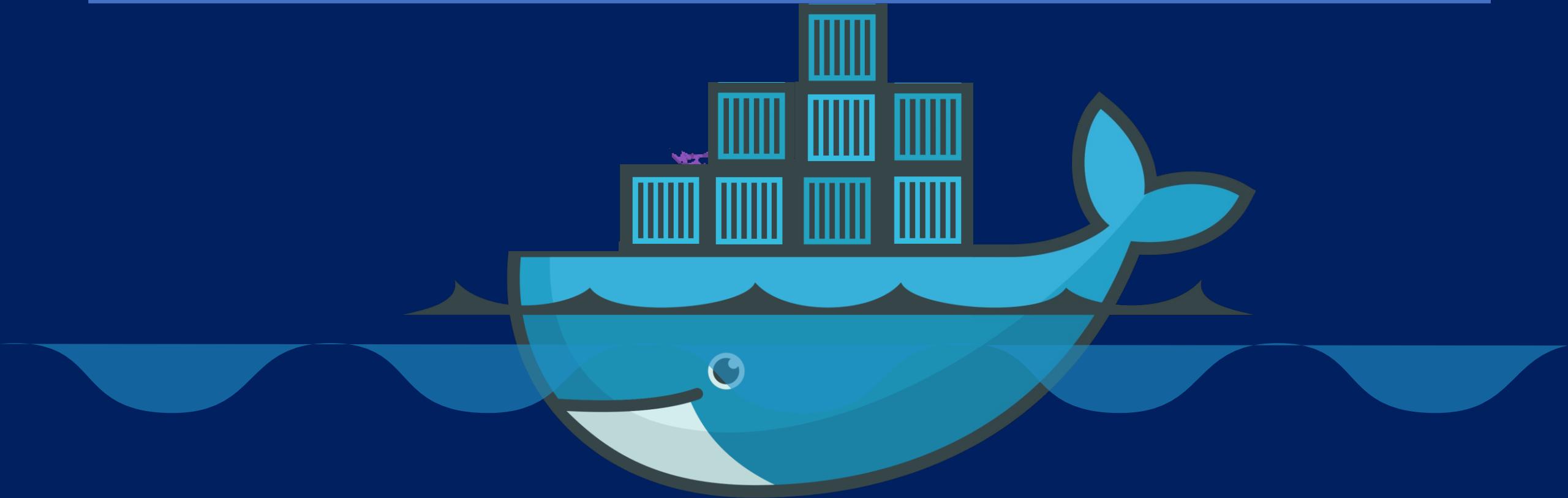




Week 9 : SOFTWARE DEVELOPMENT TOOLS AND ENVIRONMENTS

Docker Command



ps – list containers

```
▶ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
796856ac413d	nginx	"nginx -g 'daemon off..."	7 seconds ago	Up 6 seconds	80/tcp	silly_sammet

```
▶ docker ps -a
```

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

STOP – stop a container

```
▶ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
796856ac413d	nginx	"nginx -g 'daemon off..."	7 seconds ago	Up 6 seconds	80/tcp	silly_sammet

```
▶ docker stop silly_sammet
```

```
silly_sammet
```

```
▶ docker ps -a
```

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

To stop all running containers

```
▶ docker stop $(docker ps -aq)
```

docker ps -aq lists all container IDs on the Docker host, whether they are running or stopped.

Rm – Remove a container

```
▶ docker rm silly_sammet
```

```
silly_sammet
```

```
▶ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
cff8ac918a2f	redis	"docker-entrypoint.s..."	6 seconds ago	Exited (0) 3 seconds ago	relaxed_aryabhatta

To delete all containers

```
▶ docker rm $(docker ps -aq)
```

This command uses the docker ps command with the -aq flags to list all running and stopped containers in your system, and then passes the list of container IDs to the docker rm command to remove them all.

```
▶ docker container prune -f
```

This command will remove all stopped containers, including their networks and volumes. The --force flag is used to skip the confirmation prompt and delete the containers directly.

images – List images

▶ 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

rmi – Remove images

```
▶ docker rmi nginx
```

```
Untagged: nginx:latest
Untagged: nginx@sha256:96fb261b66270b900ea5a2c17a26abbfabef95506e73c3a3c65869a6dbe83223a
Deleted: sha256:f68d6e55e06520f152403e6d96d0de5c9790a89b4cf99f4626f68146fa1dbdc
Deleted: sha256:1b0c768769e2bb66e74a205317ba531473781a78b77feef8ea6fd7be7f4044e1
Deleted: sha256:34138fb60020a180e512485fb96fd42e286fb0d86cf1fa2506b11ff6b945b03f
Deleted: sha256:cf5b3c6798f77b1f78bf4e297b27cfa5b6caa982f04caeb5de7d13c255fd7a1e
```

! Delete all dependent containers to remove image

To delete all images

```
▶ docker rmi -f $(docker images -aq)
```

To clean up unused images.

```
▶ docker image prune -f
```

LAB 1 : docker : ps stop rm

1. สร้าง container 2 ตัวชื่อ container1 และ container2 โดยใช้ image nginx ด้วยคำสั่ง

```
css
```

 Copy code

```
docker run -d --name container1 nginx  
docker run -d --name container2 nginx
```

2. ใช้คำสั่ง docker ps เพื่อตรวจสอบ container ทั้งสองว่ากำลังทำงานอยู่หรือไม่

```
docker ps
```

ผลลัพธ์:

```
bash
```

 Copy code

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
e5e5f5d14306	nginx	"/docker-entrypoint...."	2 seconds ago	Up 1 sec
77f7f291059c	nginx	"/docker-entrypoint...."	7 seconds ago	Up 6 sec

3. ใช้คำสั่ง docker stop เพื่อหยุดการทำงานของ container2

```
vbnet
docker stop container2
Copy code
```

4. ใช้คำสั่ง docker ps เพื่อตรวจสอบ container ทั้งสองว่าขึ้นอยู่หรือไม่

```
docker ps
Copy code
```

ผลลัพธ์:

```
bash
CONTAINER ID   IMAGE      COMMAND           CREATED          STATUS
77f7f291059c   nginx      "/docker-entrypoint..."  2 minutes ago   Up 2 minu
Copy code
```

5. ใช้คำสั่ง docker rm เพื่อลบ container2 ออกจาก Docker Host

```
bash
docker rm container2
Copy code
```

6. ใช้คำสั่ง docker ps เพื่อตรวจสอบ container ว่าลบออกไปหรือไม่

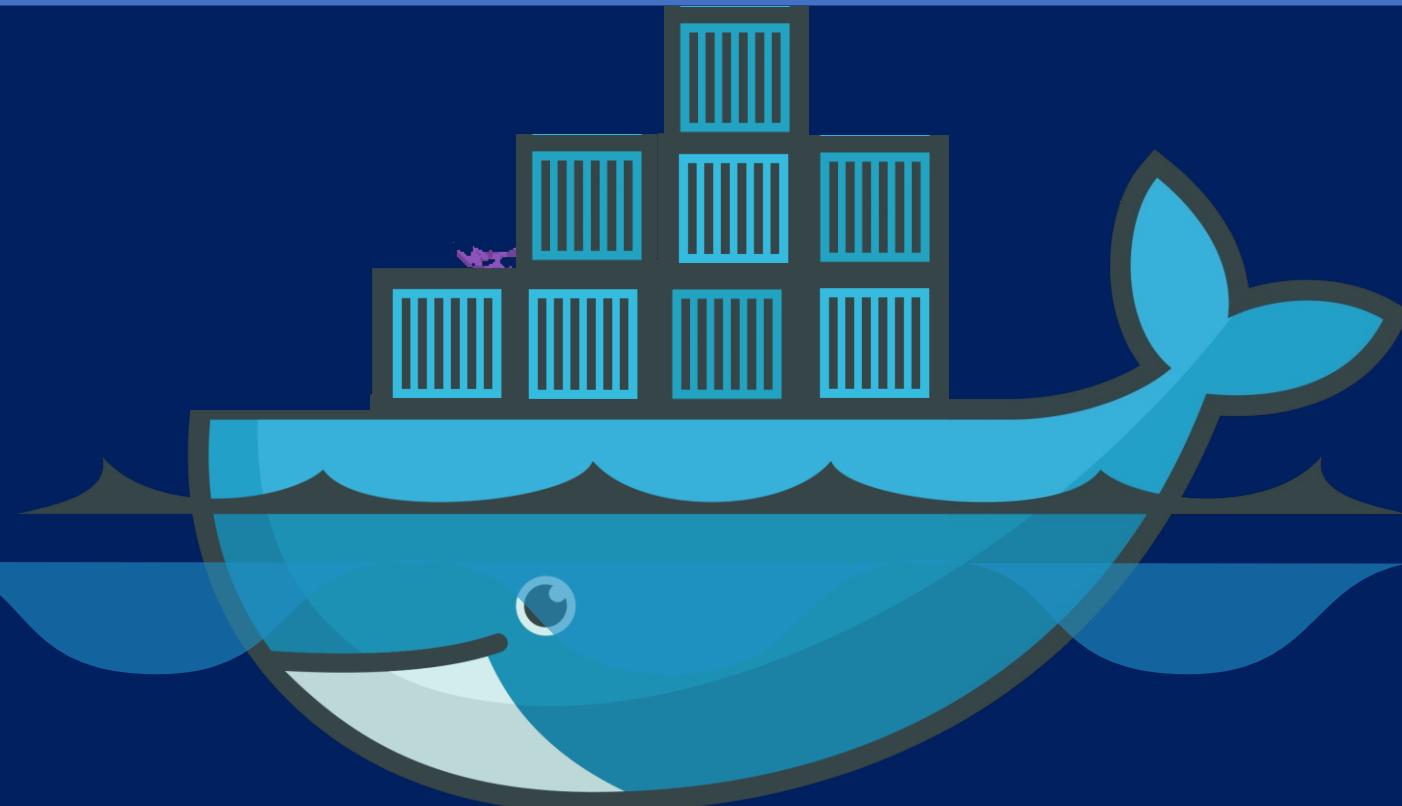
```
docker ps
Copy code
```

ผลลัพธ์:

```
bash
CONTAINER ID   IMAGE      COMMAND           CREATED          STATUS
77f7f291059c   nginx      "/docker-entrypoint..."  4 minutes ago   Up 4 minu
Copy code
```

จะเห็นได้ว่า container2 ถูกลบออกจาก Docker Host แล้ว

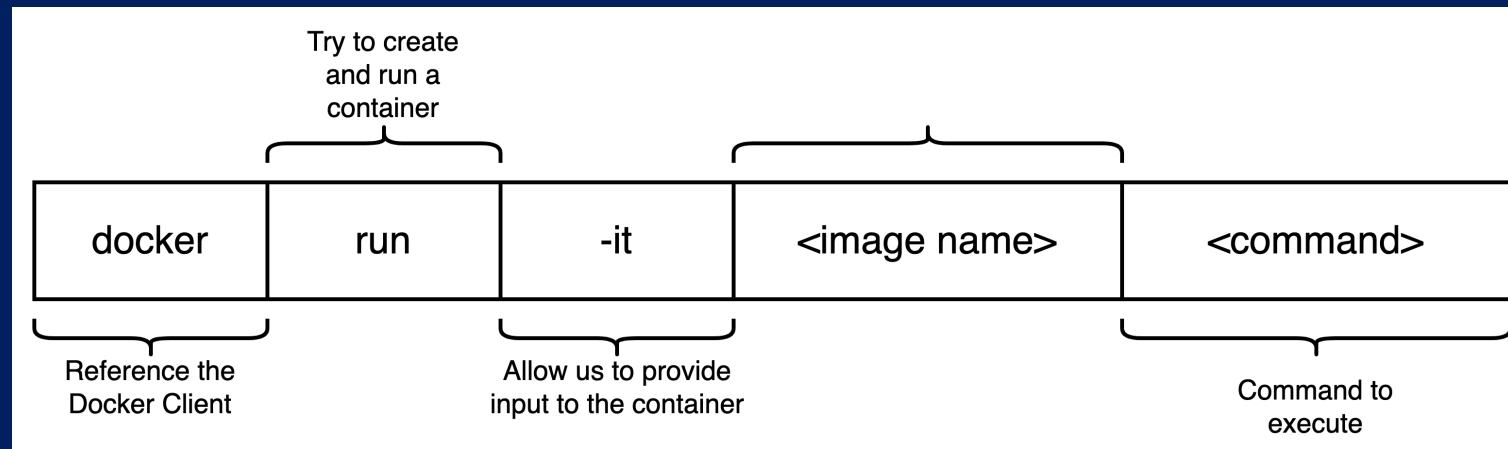
More Docker run and Docker exec



Run – stdin

```
docker pull ubuntu  
docker run -it ubuntu sh
```

```
[# ls  
bin dev home media opt root sbin sys usr  
boot etc lib mnt proc run srv tmp var  
[# echo hello world  
hello world  
# ]
```



Run – attach and detach

```
docker run -d --name lab nginx
```

```
latest: Pulling from library/nginx
66dbba0fb1b5: Pull complete
6a4b1f0b5a90: Pull complete
16ea4daad357: Pull complete
646b2422838c: Pull complete
c6036fb71e57: Pull complete
dc0e78f15ad0: Pull complete
Digest: sha256:aa0afebbb3cfa473099a62c4b32e9b3fb73ed23f2a75a65ce1d4b4f55a5c2ef2
Status: Downloaded newer image for nginx:latest
252b74d97da0dd09d01cc50358ea86766894ba016dc8c03d22a01399d4298eb6
```

```
docker run --name lab nginx
```

```
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
66dbba0fb1b5: Pull complete
6a4b1f0b5a90: Pull complete
16ea4daad357: Pull complete
646b2422838c: Pull complete
c6036fb71e57: Pull complete
dc0e78f15ad0: Pull complete
2023/03/10 03:40:11 [notice] 1#1: start worker processes
2023/03/10 03:40:11 [notice] 1#1: start worker process 29
2023/03/10 03:40:11 [notice] 1#1: start worker process 30
2023/03/10 03:40:11 [notice] 1#1: start worker process 31
2023/03/10 03:40:11 [notice] 1#1: start worker process 32
2023/03/10 03:40:11 [notice] 1#1: start worker process 33
```

Run – tag

```
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 # o000o000o000o Redis is starting o000o000o000o
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 # o000o000o000o Redis is starting o000o000o000o
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
```

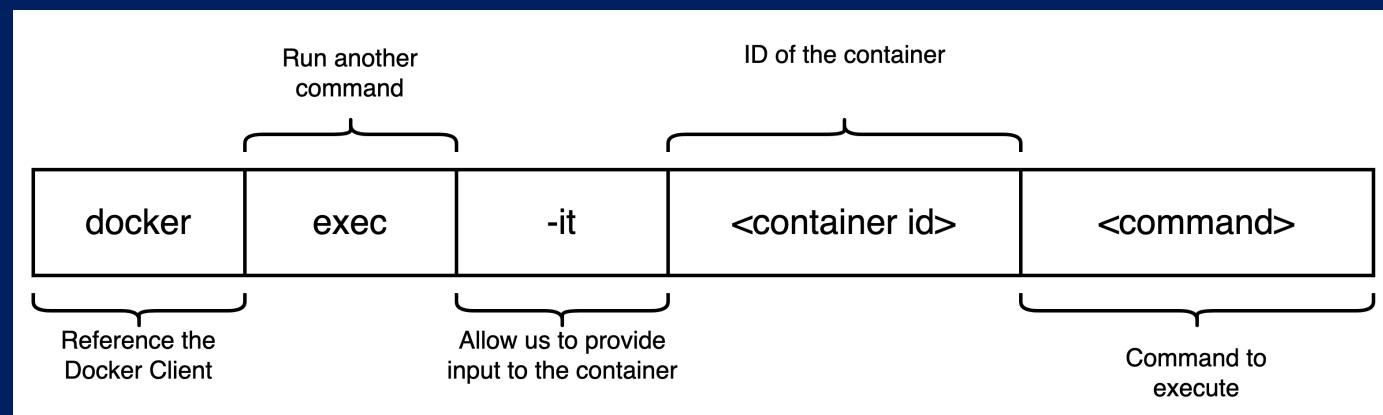
Exec— execute a command

```
▶ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
538d037f94a7	ubuntu	"sleep 100"	6 seconds ago	Up 4 seconds	distracted_mcclintock

```
▶ 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
```



Inspect Container

```
▶ 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": {...}  
   }  
 }]
```

Container Logs

```
▶ 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)
```

LAB 2 :

2.1 Run – stdin

```
python
Copy code
docker run -it ubuntu /bin/bash
```

คำสั่งดังกล่าวจะสร้าง Docker container จาก Ubuntu image และเปิด shell ของ Ubuntu ให้คุณ
สามารถเข้าถึงและใช้งานได้ทันที หากไม่มี Ubuntu image อยู่ในเครื่องของคุณ ระบบจะดาวน์โหลด
Ubuntu image มาให้อัตโนมัติก่อนเริ่มทำงานใน Docker container นี้

2.2 Run – stdin with Dockerfile

git clone

```
git clone https://github.com/Tuchsanai/devpot_week9.git  
cd devpot_week9/Lab2_stdin_with_Dockerfile
```

go to Directory

```
cd devpot_week9
```

build Docker image with docker build

```
docker build -t myubuntu .
```

docker run with -it option

```
docker run -it myubuntu
```

devpot_week9 / Lab2_stdin_with_Dockerfile /

Tuchsanai zzz

Name

..

Dockerfile

readme.md

FROM ubuntu

```
RUN apt-get update && apt-get install -y \  
git \  
wget  
["/bin/bash"]
```

docker environment variables

Environment Variables

app.py

```
import os
from flask import Flask

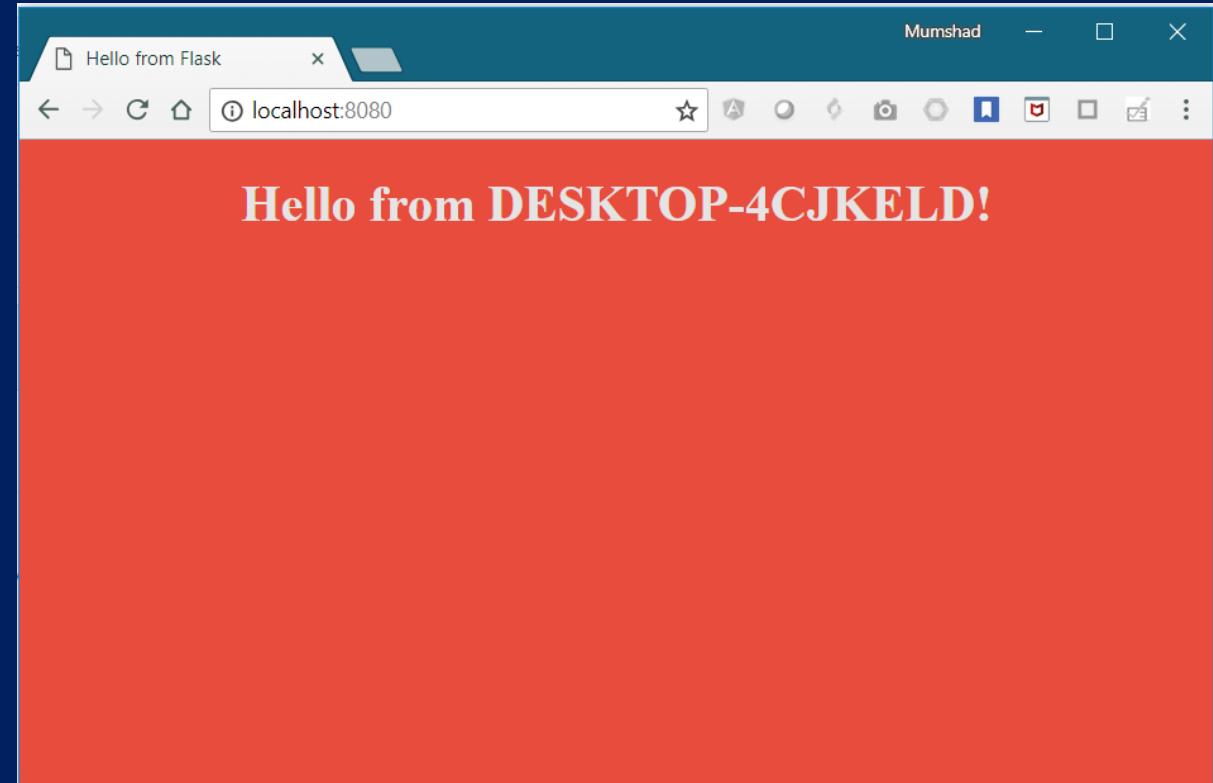
app = Flask(__name__)

...
...

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

Environment Variables

app.py

```
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")
```

Environment Variables

app.py

```
import os
from flask import Flask

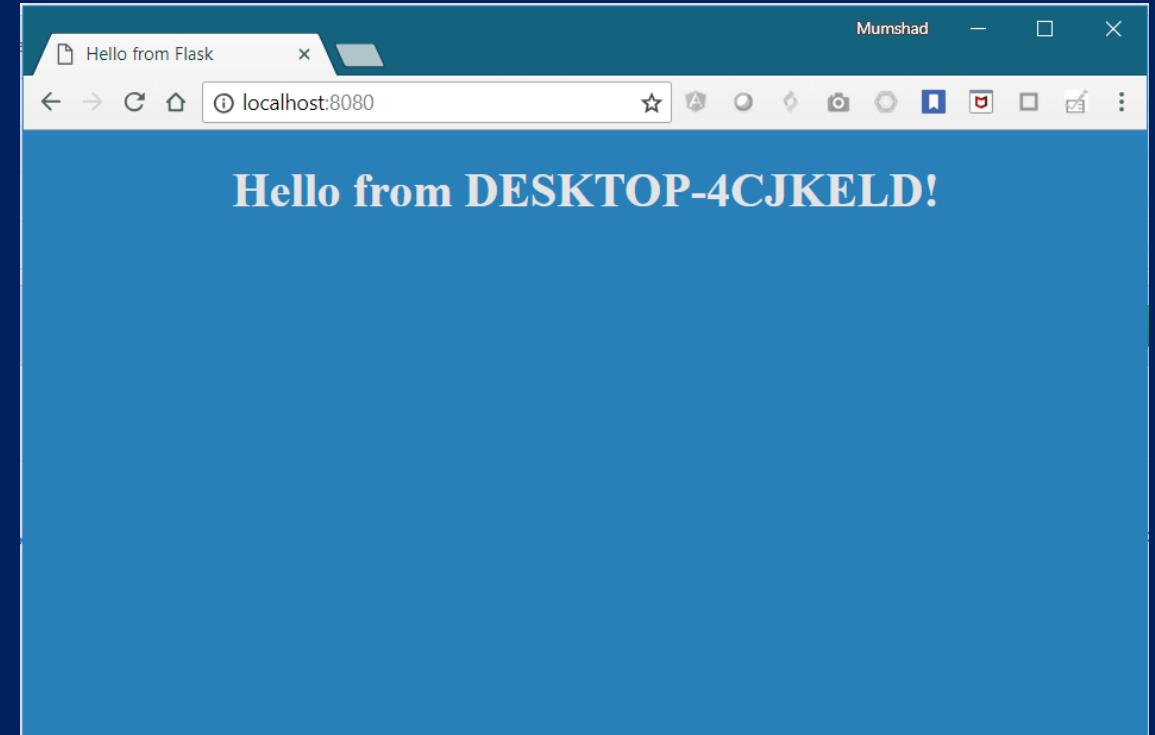
app = Flask(__name__)

...
...

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

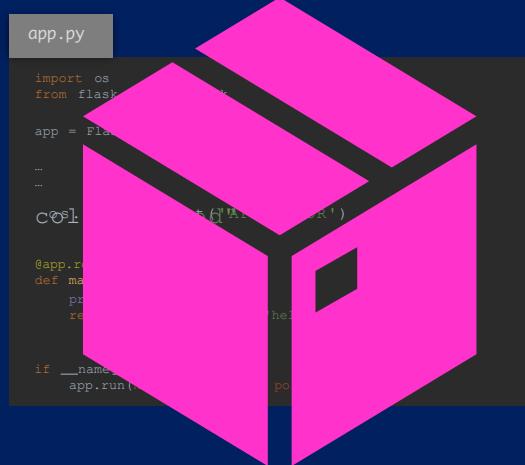
@app.route("/")
def main():
    print(color)
    return render_template('hello.html', color=color)

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

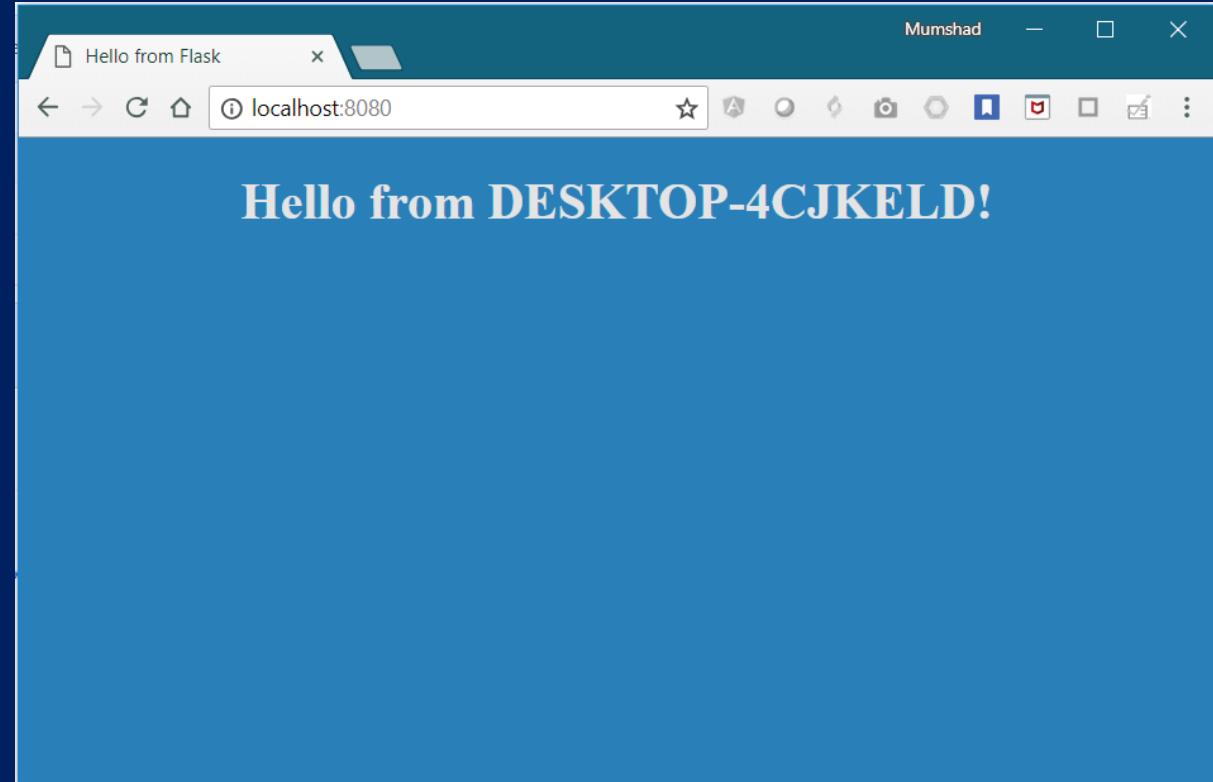


▶ export APP_COLOR=blue; python app.py

ENV Variables in Docker



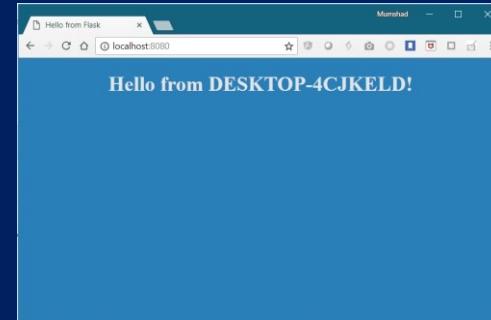
```
app.py
import os
from flask import Flask
app = Flask(__name__)
...
COSL
@app.route('/')
def main():
    print("Hello from DESKTOP-4CJKELD!")
    return "Hello from DESKTOP-4CJKELD!"
if __name__ == "__main__":
    app.run()
```



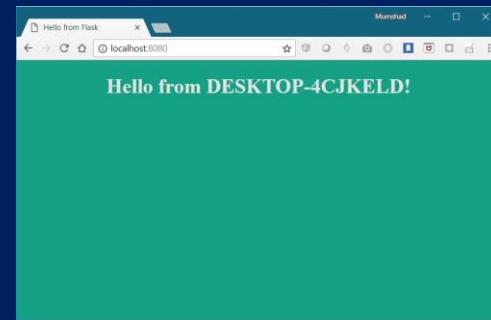
► docker run -e APP_COLOR=blue

ENV Variables in Docker

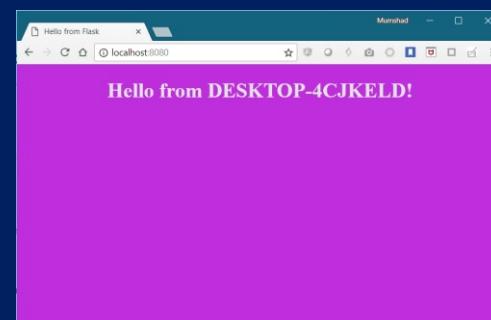
```
▶ docker run -e APP_COLOR=blue simple-webapp-color
```



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



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



Inspect Environment Variable

```
▶ 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"  
     ],  
   },  
 }]
```

d o c k e r
images

What am I containerizing?



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

Install all dependencies

RUN apt-get install python

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

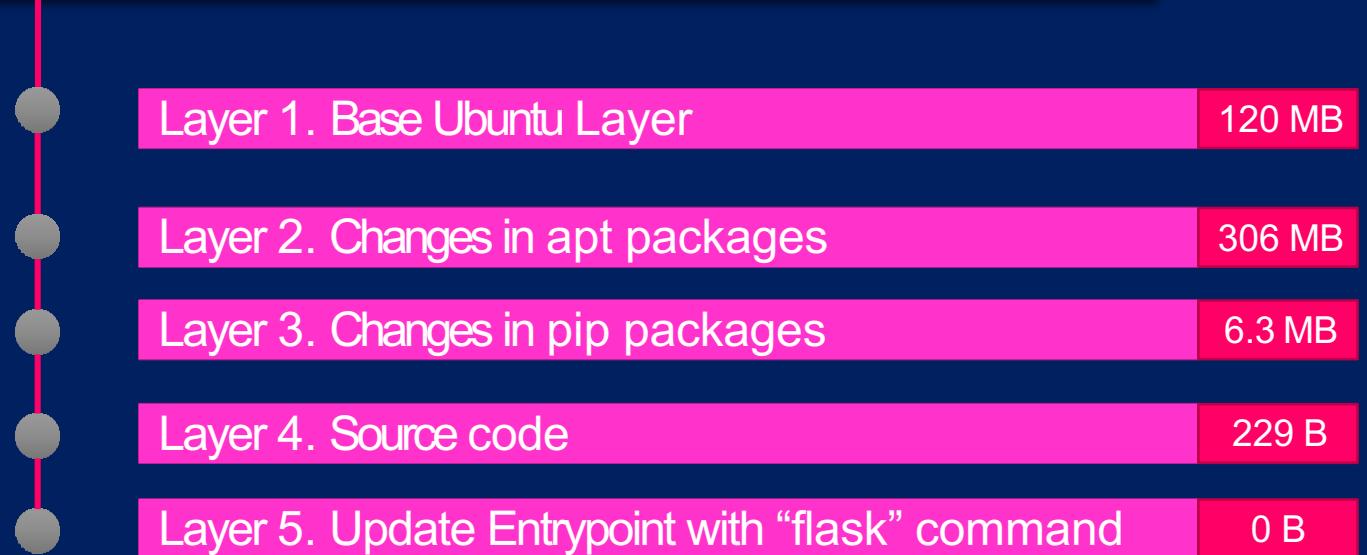
Specify Entrypoint

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

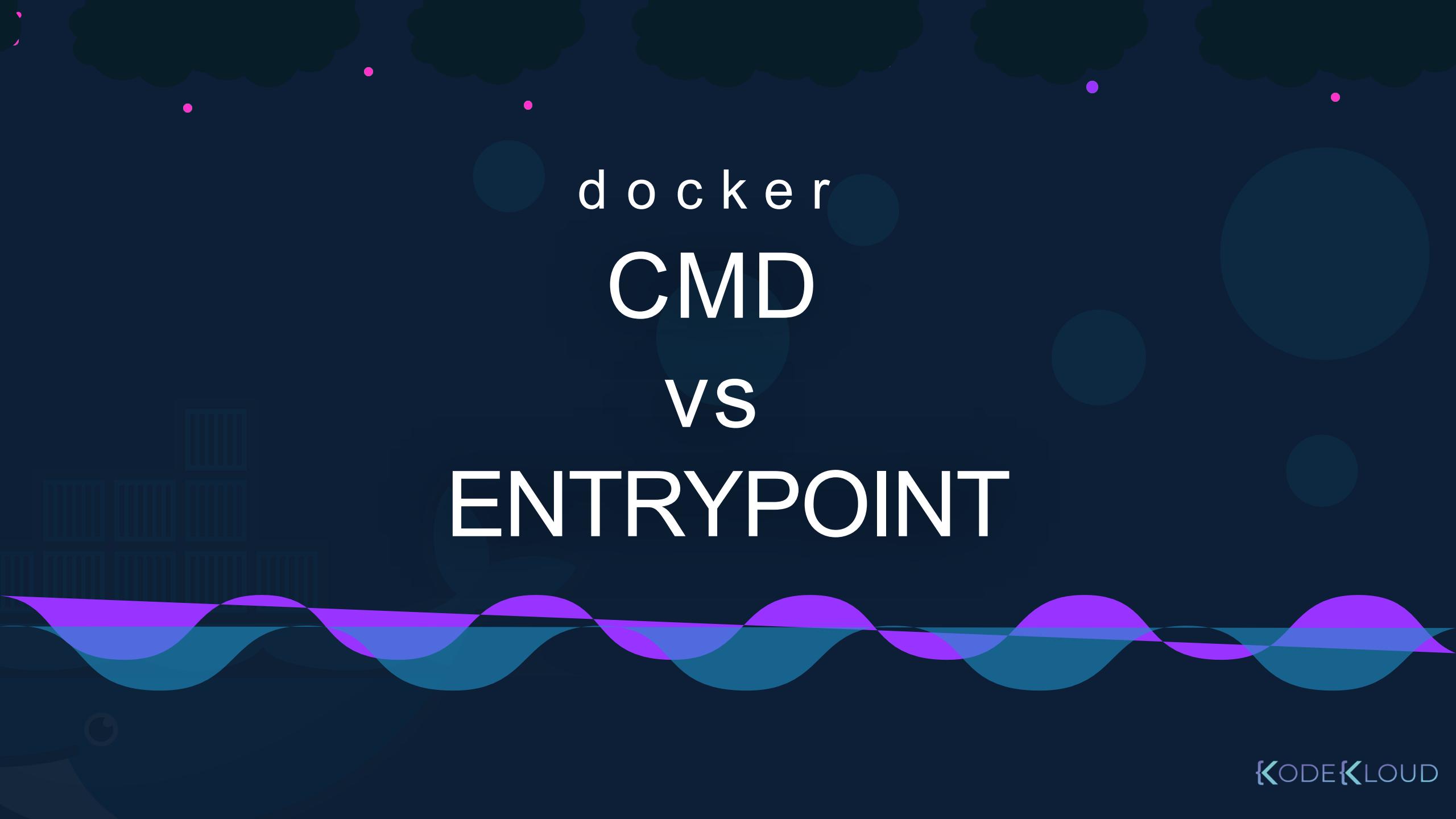
```
docker build Dockerfile -t mmumshad/my-custom-app
```

```
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
```

What can you containerize?



Containerize Everything!!!



docker CMD vs ENTRYPOINT

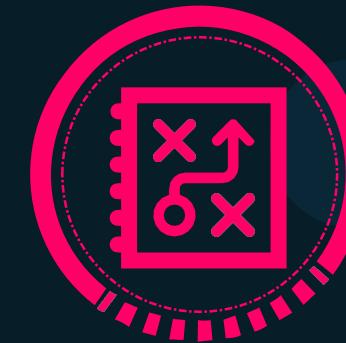
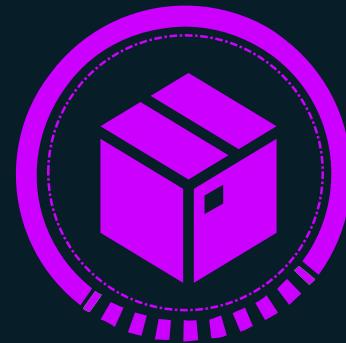
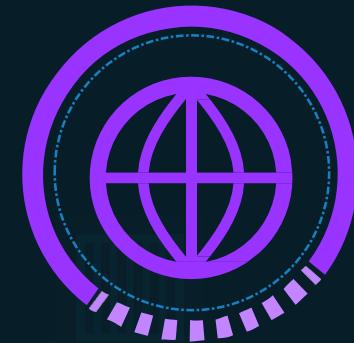
```
▶ docker run ubuntu
```

```
▶ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
--------------	-------	---------	---------	--------	-------

```
▶ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
45aacca36850	ubuntu	"/bin/bash"	43 seconds ago	Exited (0) 41 seconds ago	



```
# Install Nginx.  
RUN \  
    add-apt-repository -y ppa:nginx/stable && \  
    apt-get update && \  
    apt-get install -y nginx && \  
    rm -rf /var/lib/apt/lists/* && \  
    echo "\ndaemon off;" >> /etc/nginx/nginx.conf && \  
    chown -R www-data:www-data /var/lib/nginx  
  
# Define mountable directories.  
VOLUME ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/cor  
  
# Define working directory.  
WORKDIR /etc/nginx  
  
# Define default command.  
CMD ["nginx"]
```

```
ARG MYSQL_SERVER_PACKAGE_URL=https://repo.mysql.com/yum/mysql-8.0-community/docker/x86_64/mysql-community-release-el7-3.noarch.rpm  
ARG MYSQL_SHELL_PACKAGE_URL=https://repo.mysql.com/yum/mysql-tools-community/el/7/x86_64/mysql-tools-community-release-el7-1.noarch.rpm  
  
# Install server  
RUN rpmkeys --import https://repo.mysql.com/RPM-GPG-KEY-mysql \  
    && yum install -y $MYSQL_SERVER_PACKAGE_URL $MYSQL_SHELL_PACKAGE_URL libpwquality \  
    && yum clean all \  
    && mkdir /docker-entrypoint-initdb.d  
  
VOLUME /var/lib/mysql  
  
COPY docker-entrypoint.sh /entrypoint.sh  
COPY healthcheck.sh /healthcheck.sh  
ENTRYPOINT ["/entrypoint.sh"]  
HEALTHCHECK CMD /healthcheck.sh  
EXPOSE 3306 33060  
CMD ["mysqld"]
```

```
# Pull base image.  
FROM ubuntu:14.04  
  
# Install.  
RUN \  
    sed -i 's/# \(.*multiverse$\)/\1/g' /etc/apt/sources.list && \  
    apt-get update && \  
    apt-get -y upgrade && \  
    apt-get install -y build-essential && \  
    apt-get install -y software-properties-common && \  
    apt-get install -y byobu curl git htop man unzip vim wget && \  
    rm -rf /var/lib/apt/lists/*  
  
# Add files.  
ADD root/.bashrc /root/.bashrc  
ADD root/.gitconfig /root/.gitconfig  
ADD root/.scripts /root/.scripts  
  
# Set environment variables.  
ENV HOME /root  
  
# Define working directory.  
WORKDIR /root  
  
# Define default command.  
CMD ["bash"]
```

▶ docker run ubuntu [COMMAND]

▶ docker run ubuntu sleep 5

FROM Ubuntu

CMD sleep 5

CMD command param1

CMD ["command", "param1"]

CMD sleep 5

CMD ["sleep", "5"]

CMD ["sleep 5"]



```
▶ docker build -t ubuntu-sleeper .
```

```
▶ docker run ubuntu-sleeper
```



```
FROM Ubuntu
```

```
CMD sleep 5
```

Command at Startup: sleep 10

```
▶ docker run ubuntu-sleeper sleep 10
```

```
FROM Ubuntu
```

```
ENTRYPOINT ["sleep", "10"]
```

Command at Startup:

```
▶ docker run ubuntu-sleeper 10
```

Command at Startup:

```
▶ docker run ubuntu-sleeper  
sleep: missing operand  
Try 'sleep --help' for more information.
```

```
FROM Ubuntu
```

```
ENTRYPOINT ["sleep"]
```

```
CMD ["5"]
```

Command at Startup:

```
▶ docker run ubuntu-sleeper 10
```

Command at Startup:

```
▶ docker run --entrypoint sleep2.0 ubuntu-sleeper 10
```

Command at Startup:

d o c k e r
networking

Default networks



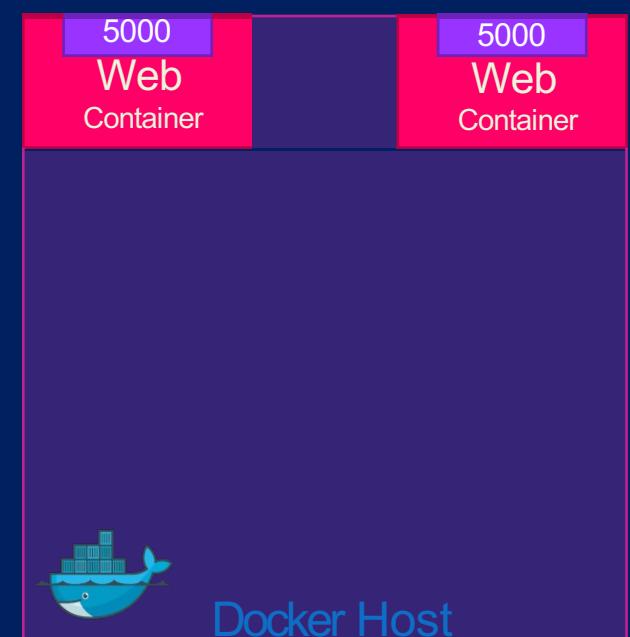
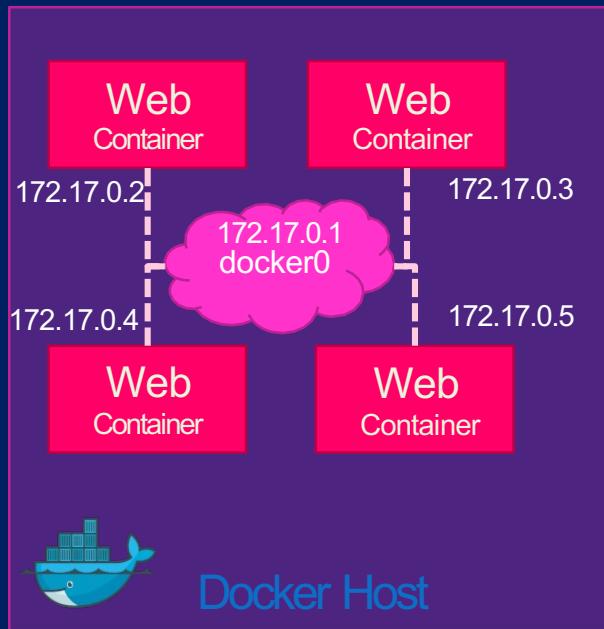
```
docker run ubuntu
```



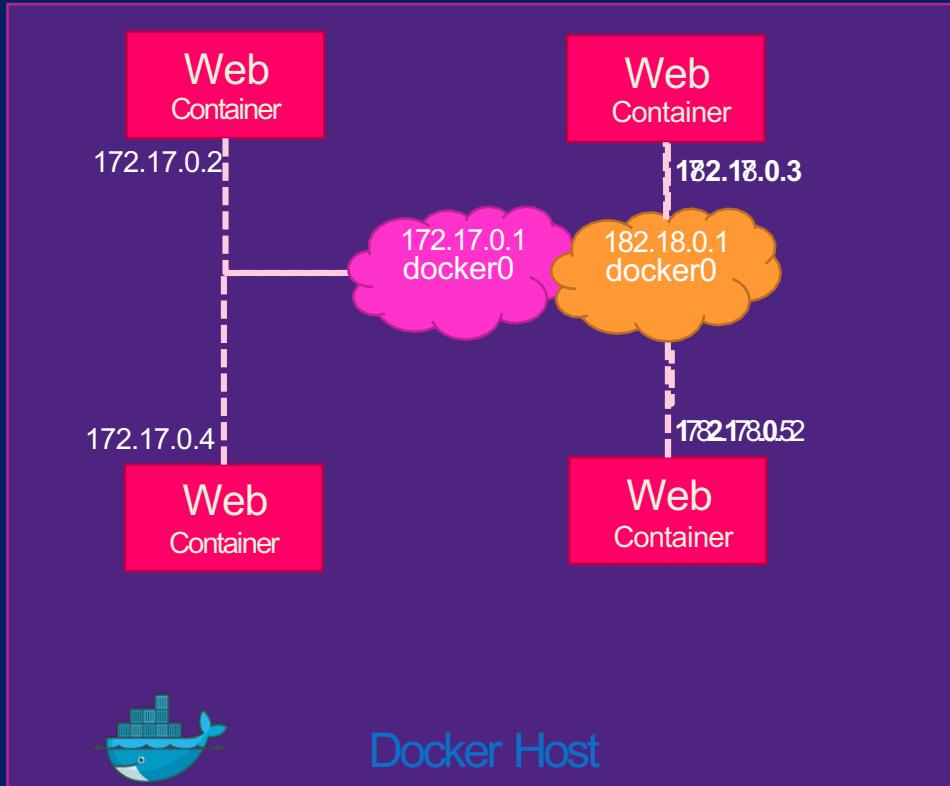
```
docker run Ubuntu --network=none
```



```
docker run Ubuntu --network=host
```



User-defined networks



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

```
docker network ls
```

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

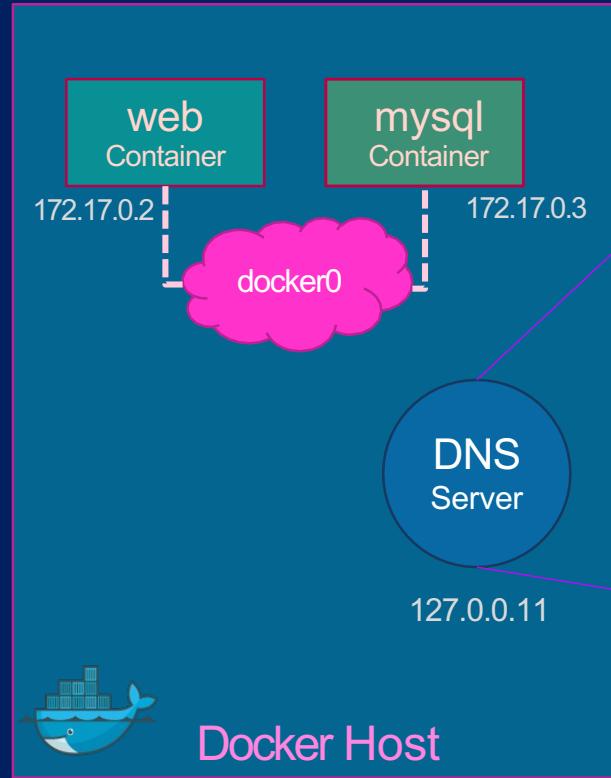
Inspect Network

```
▶ docker inspect blissful_hopper
```

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

Embedded DNS

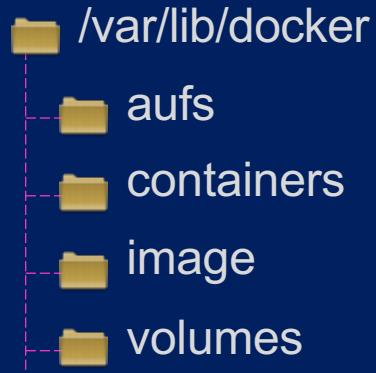
```
mysql.connect( mysql )
```



Host	IP
web	172.17.0.2
mysql	172.17.0.3

d o c k e r
storage

File system



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
```

Dockerfile2

```
FROM Ubuntu

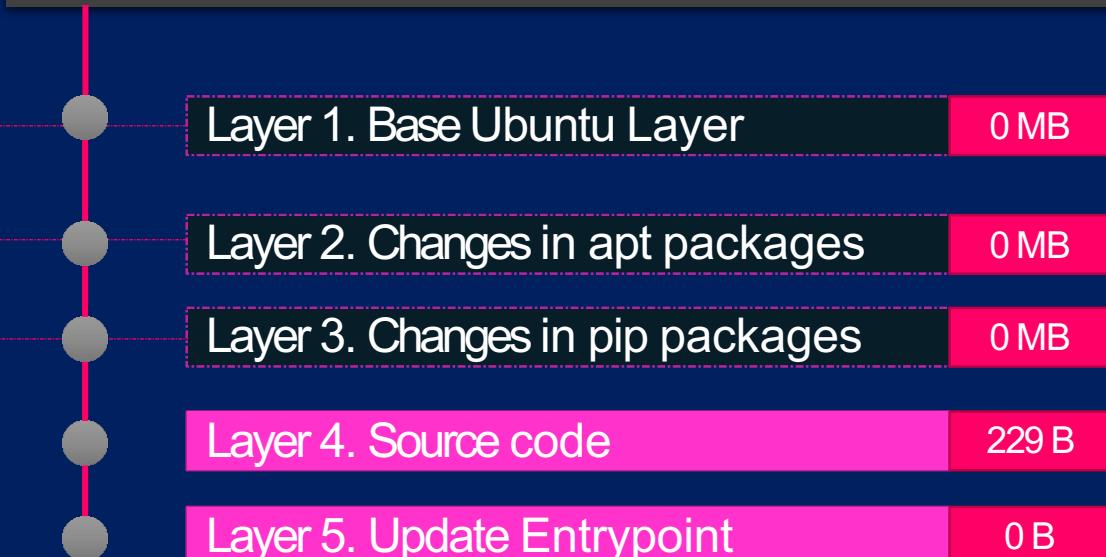
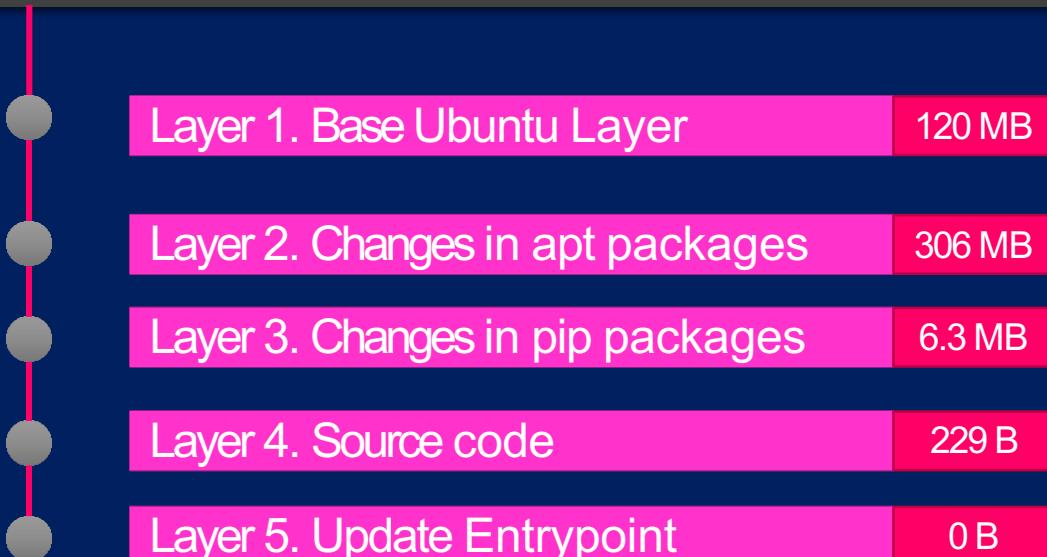
RUN apt-get update && apt-get -y install python

RUN pip install flask flask-mysql

COPY app2.py /opt/source-code

ENTRYPOINT FLASK_APP=/opt/source-code/app2.py flask run
```

```
docker build Dockerfile2 -t mmumshad/my-custom-app-2
```



Layered architecture

Container Layer

ReadWrite

Layer 6. Container Layer

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

Image Layers

Read Only

Layer 5. Update Entrypoint with “flask” command

Layer 4. Source code

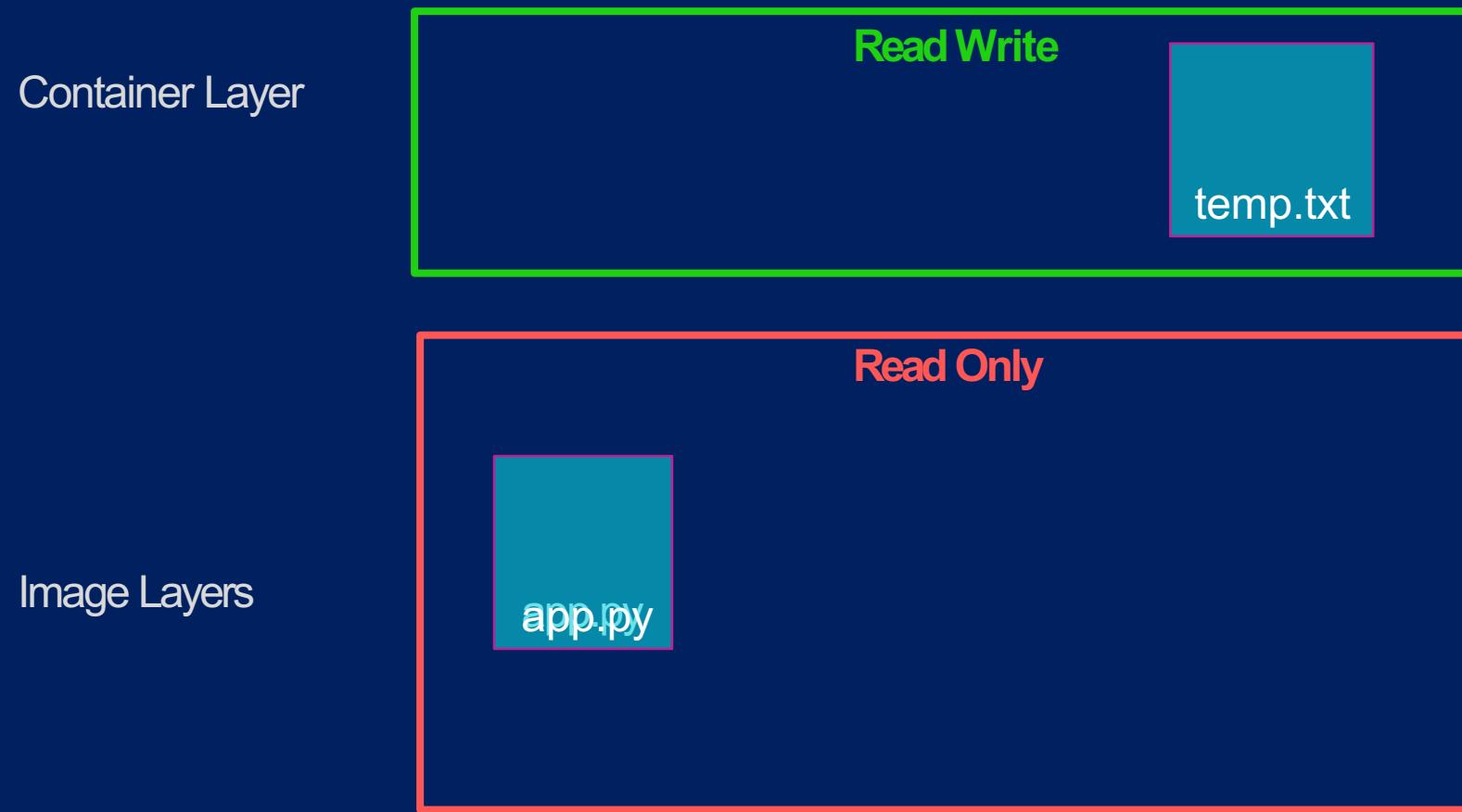
Layer 3. Changes in pip packages

Layer 2. Changes in apt packages

Layer 1. Base Ubuntu Layer

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

COPY-ON-WRITE



volumes

```
docker volume create data_volume
```

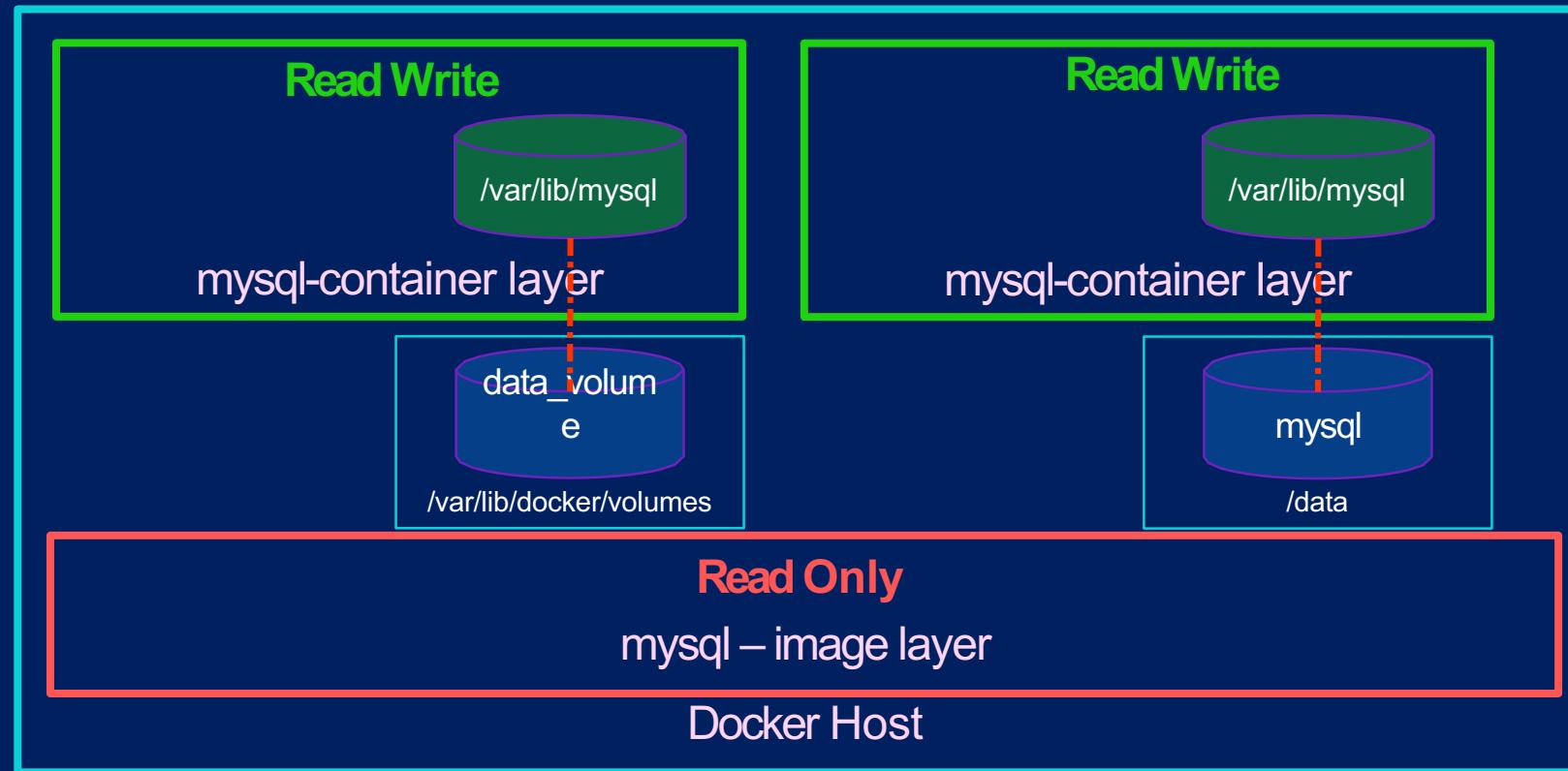


```
docker run -v data_volume:/var/lib/mysql mysql
```

```
docker run -v data_volume2:/var/lib/mysql mysql
```

```
docker run -v /data/mysql:/var/lib/mysql mysql
```

```
docker run \
--mount type=bind,source=/data/mysql,target=/var/lib/mysql mysql
```



Storage drivers

- AUFS
- ZFS
- BTRFS
- Device Mapper
- Overlay
- Overlay2

d o c k e r
compose

Docker compose

```
docker run mmumshad/simple-webapp
```

```
docker run mongodb
```

```
docker run redis:alpine
```

```
docker run ansible
```

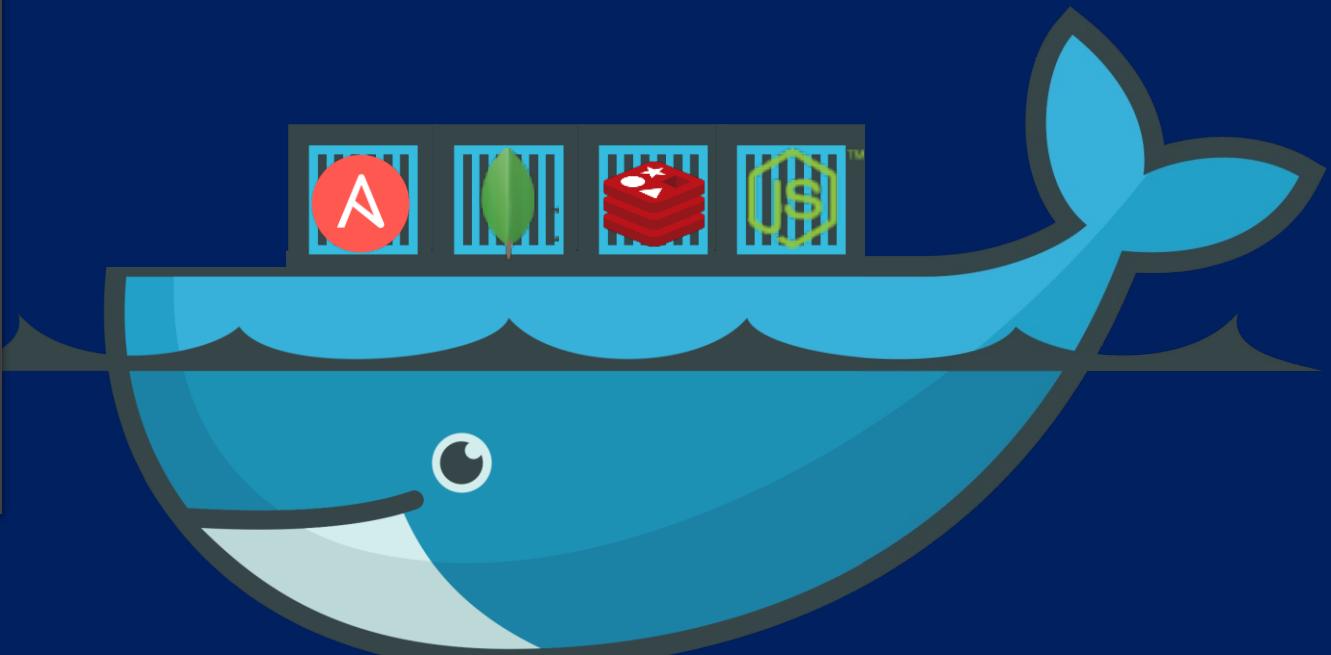
```
docker-compose.yml
```

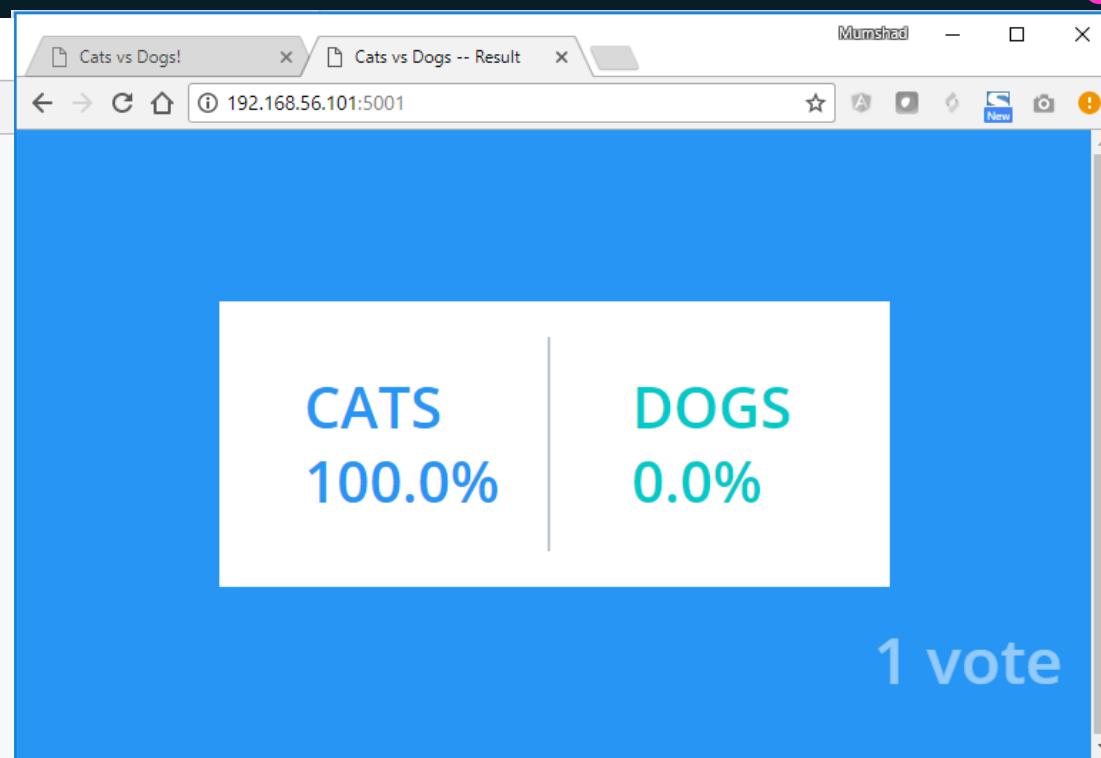
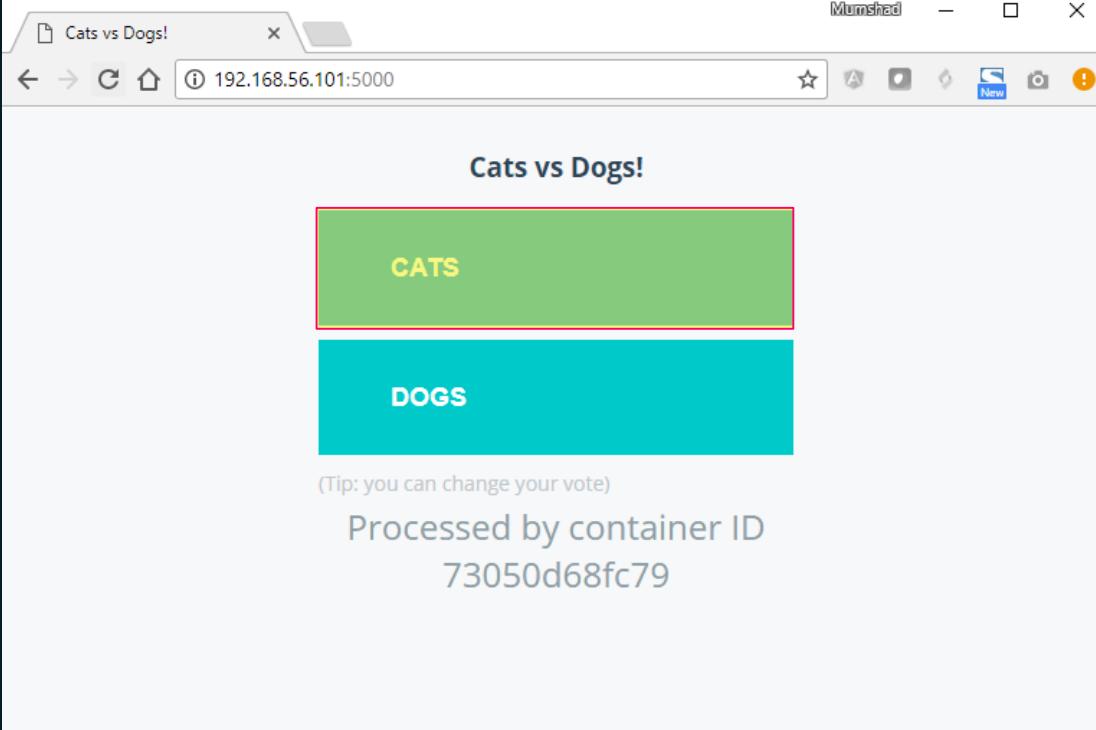
```
services:  
  web:  
    image: "mmumshad/simple-webapp"  
  database:  
    image: "mongodb"  
  messaging:  
    image: "redis:alpine"  
  orchestration:  
    image: "ansible"
```

```
docker-compose up
```



Public Docker registry - dockerhub





CATS	DOGS
1	0

docker run --links

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

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

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

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

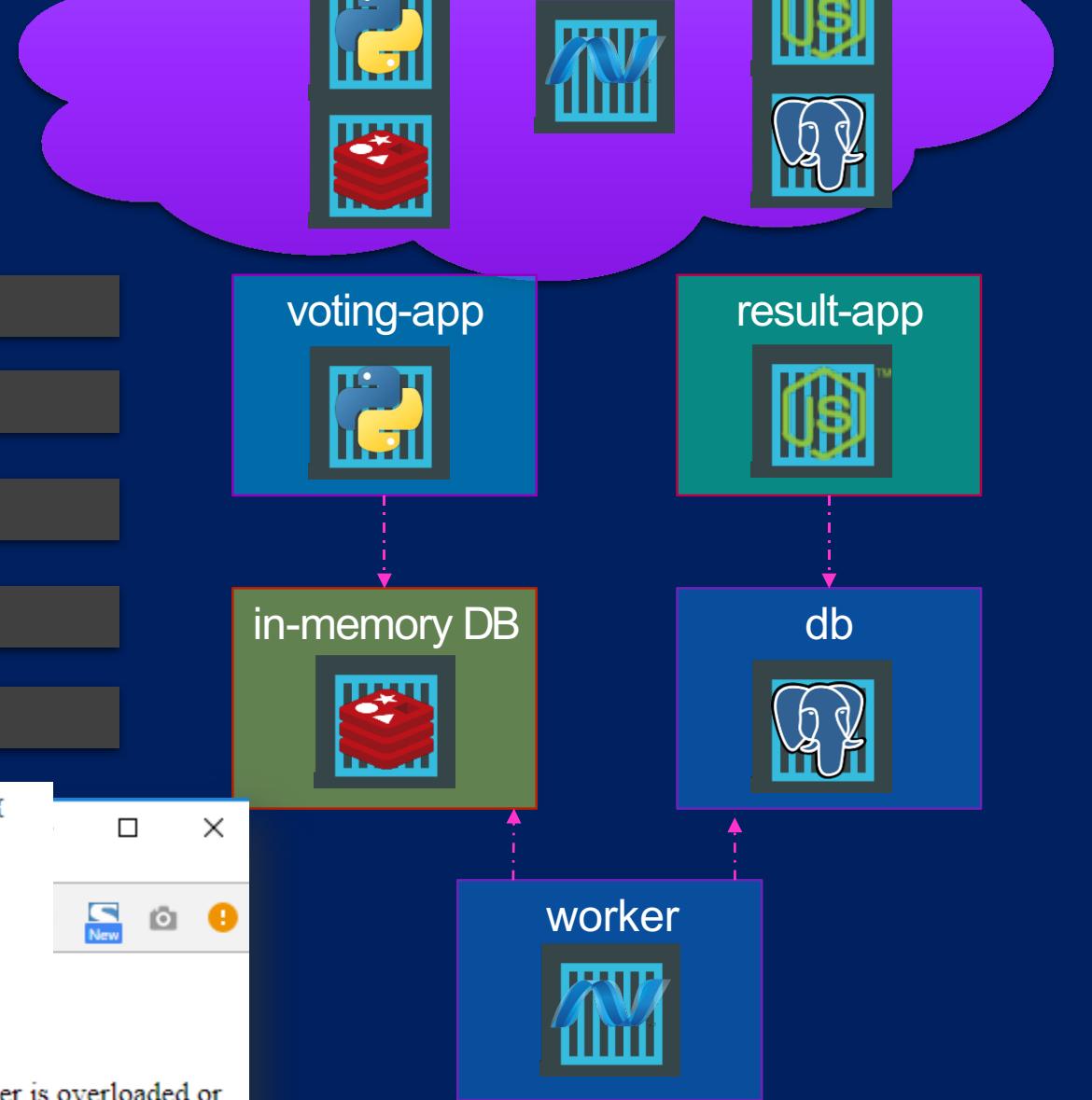
```
docker run -d --name=worker --link db:db --link redis:redis
```

The terminal shows the following commands:

```
500 Internal Server Error
try {
    Jedis redis = connectToRedis("redis");
    Connection dbConn = connectToDB("db");
    System.out.println("Watching vote queue");
}
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
172.17.0.3      ebcae9eb46bf
```

A Java code snippet is displayed in the terminal:

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



Deprecation Warning

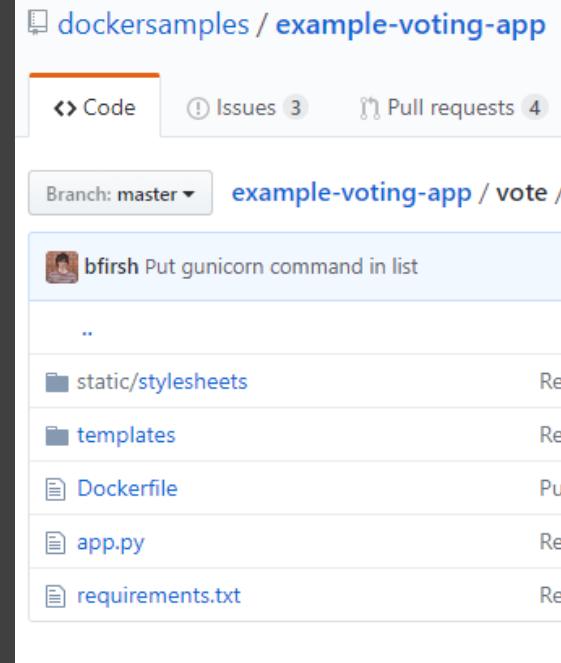
Docker compose - build

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
```



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

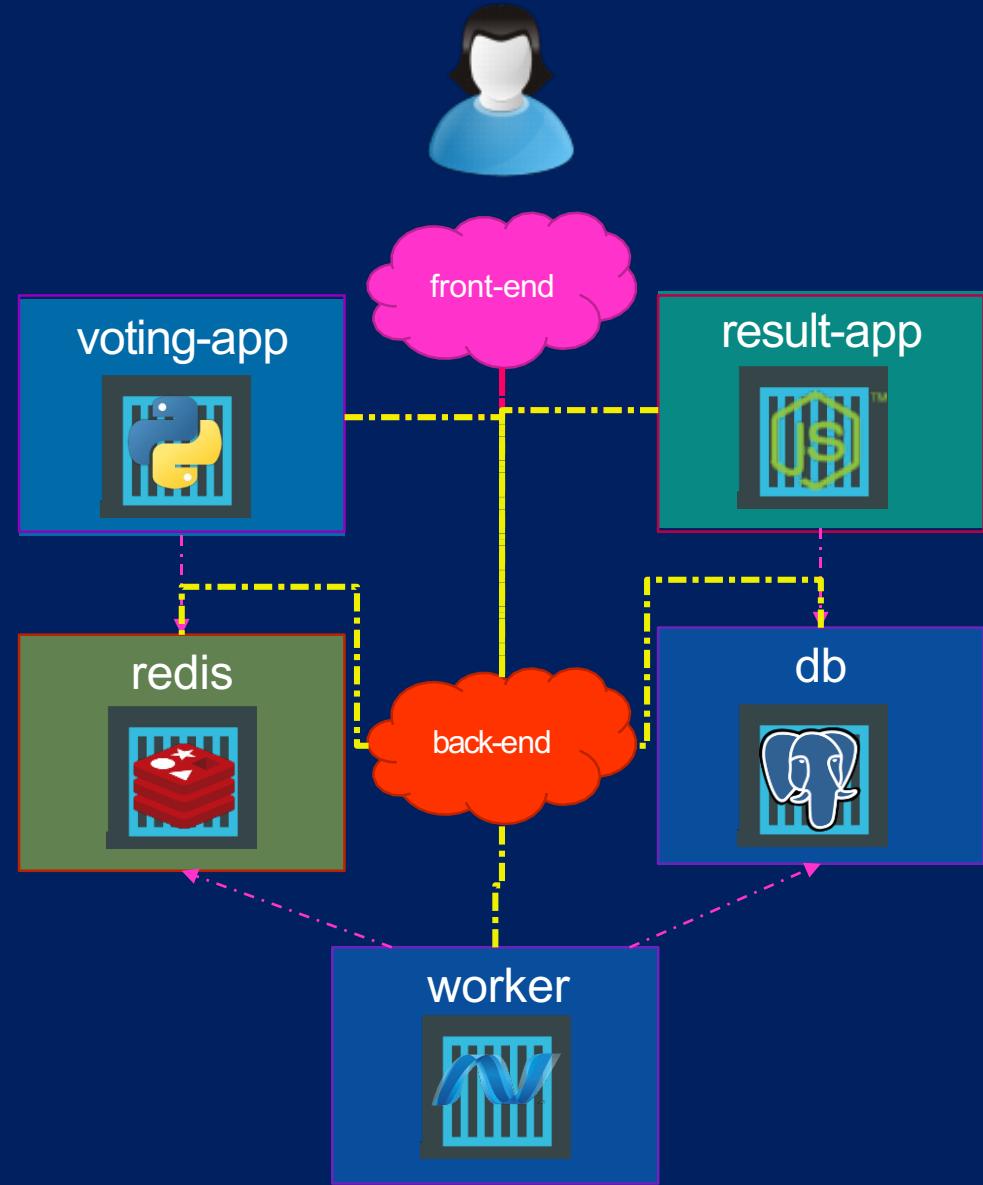
```
version: 3  
services:
```

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:
```



d o c k e r
registry

Image

```
▶ docker run nginx
```

Image

docker.io
Docker Hub

image: docker.io/nginx/nginx



Registry User/
Account Repository Image/
Repository

gcr.io/kubernetes-e2e-test-images/dnsutils

Private Registry

```
▶ docker login private-registry.io
```

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 run -d -p 5000:5000 --name registry registry:2
```

```
▶ docker image tag my-image localhost:5000/my-image
```

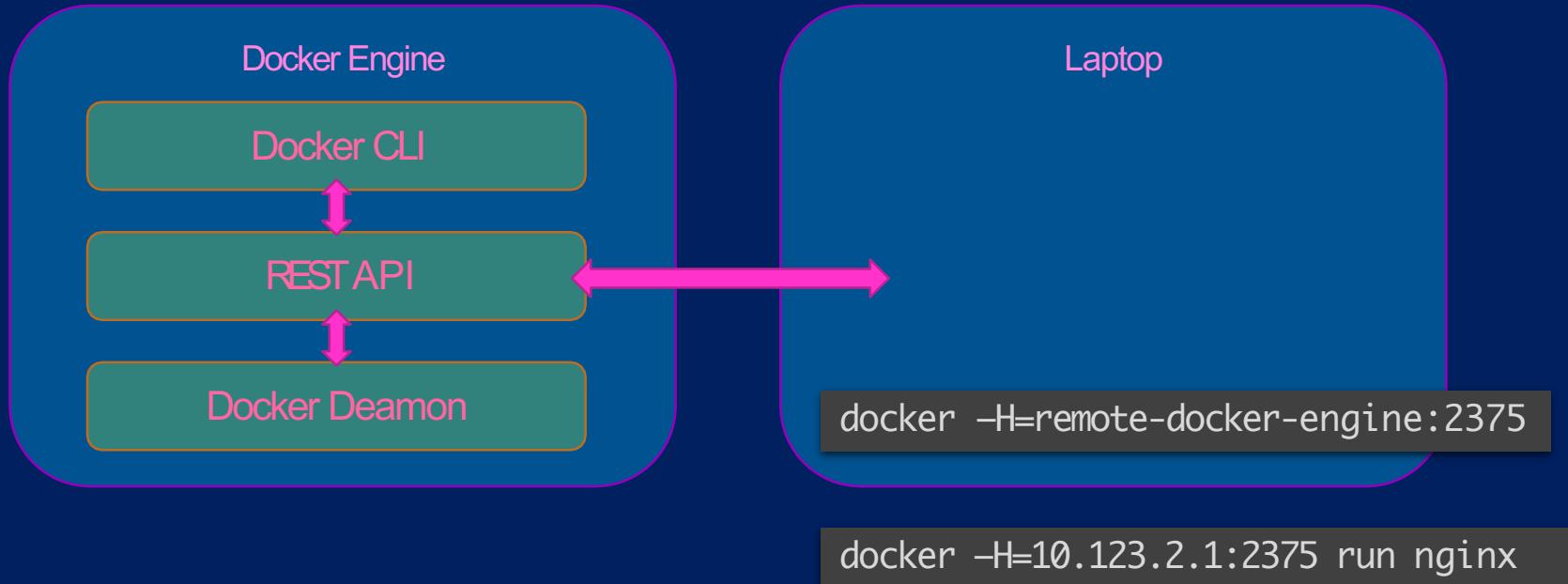
```
▶ docker push localhost:5000/my-image
```

```
▶ docker pull localhost:5000/my-image
```

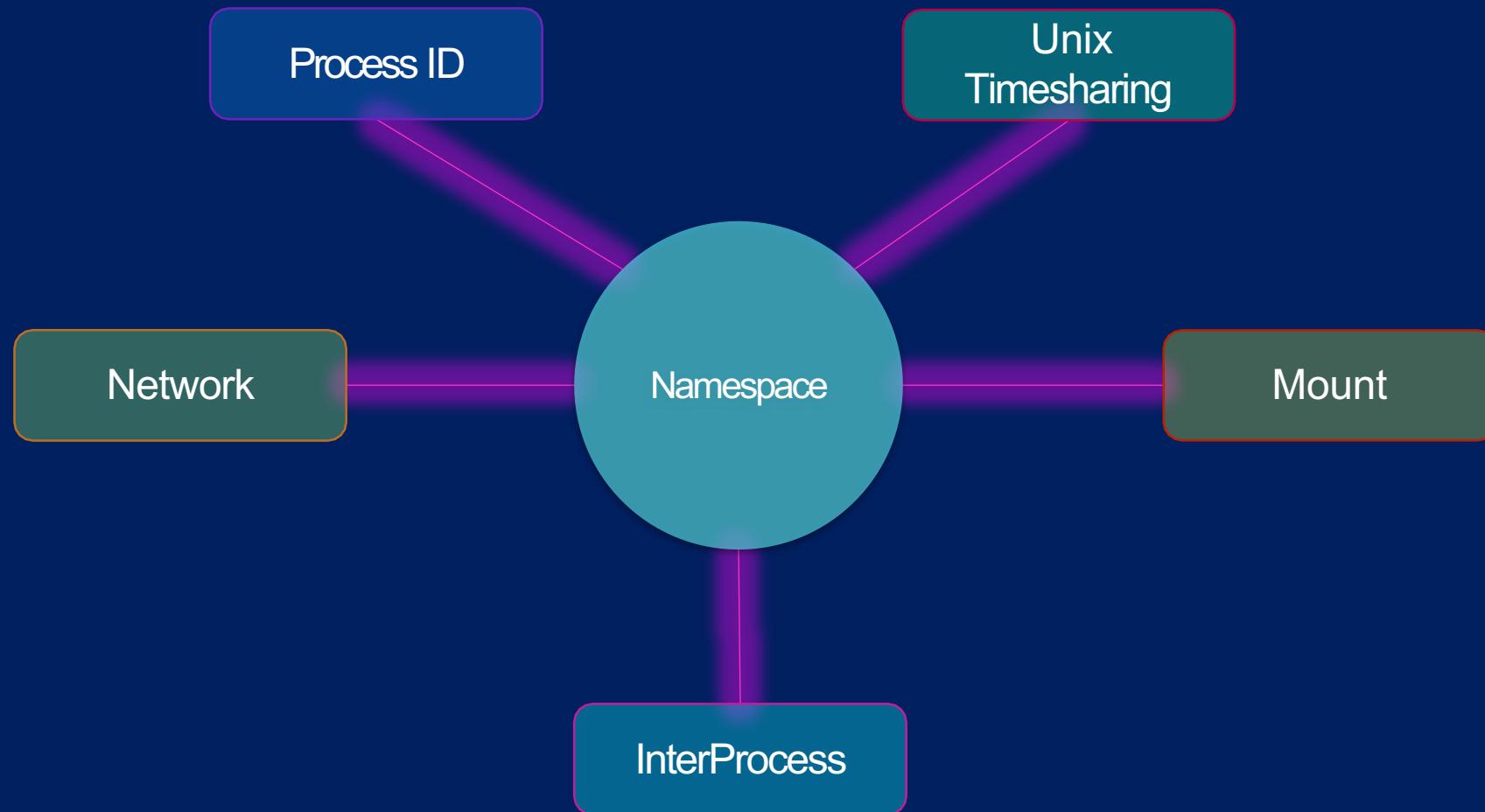
```
▶ docker pull 192.168.56.100:5000/my-image
```


d o c k e r
engine

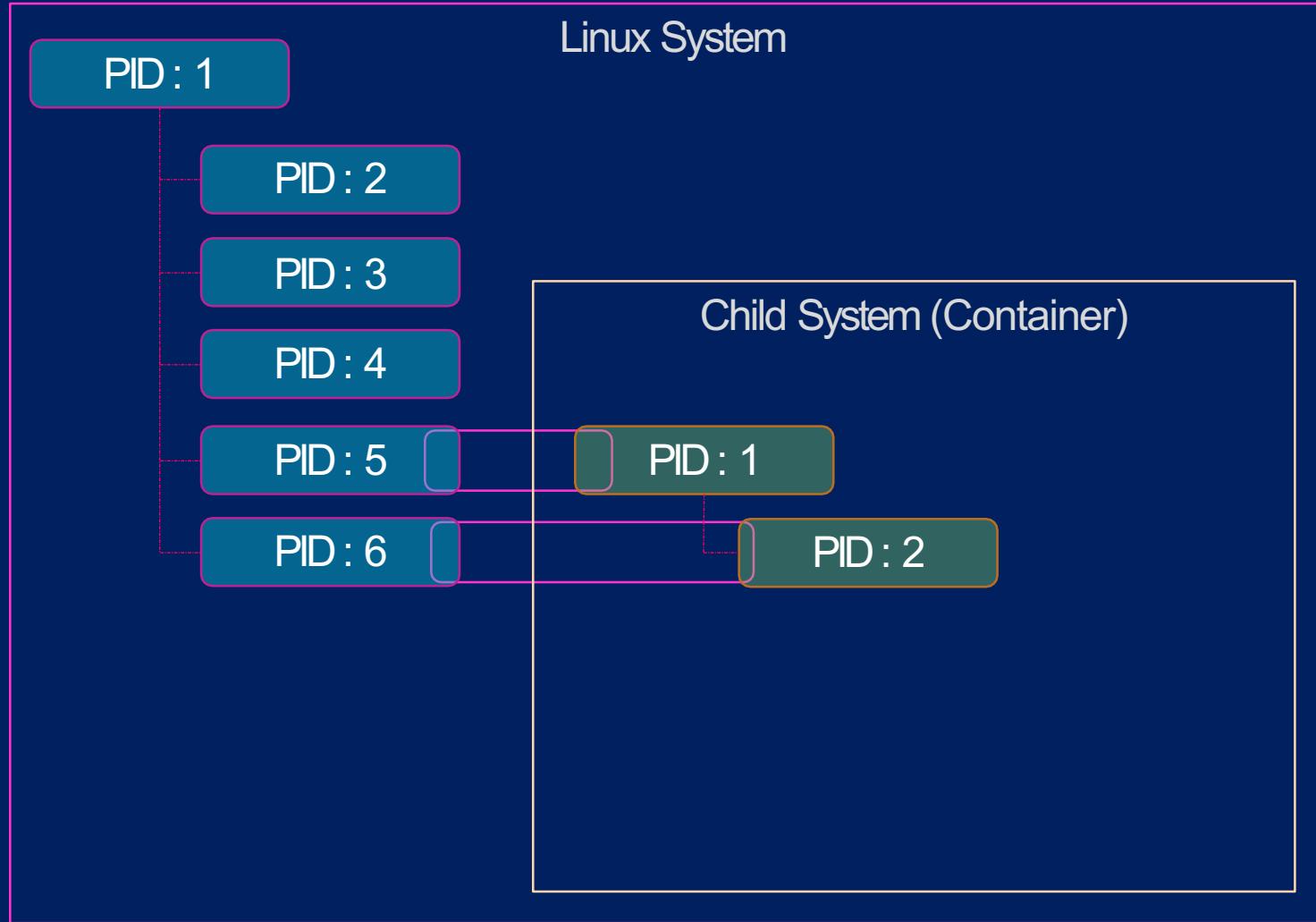
Docker Engine



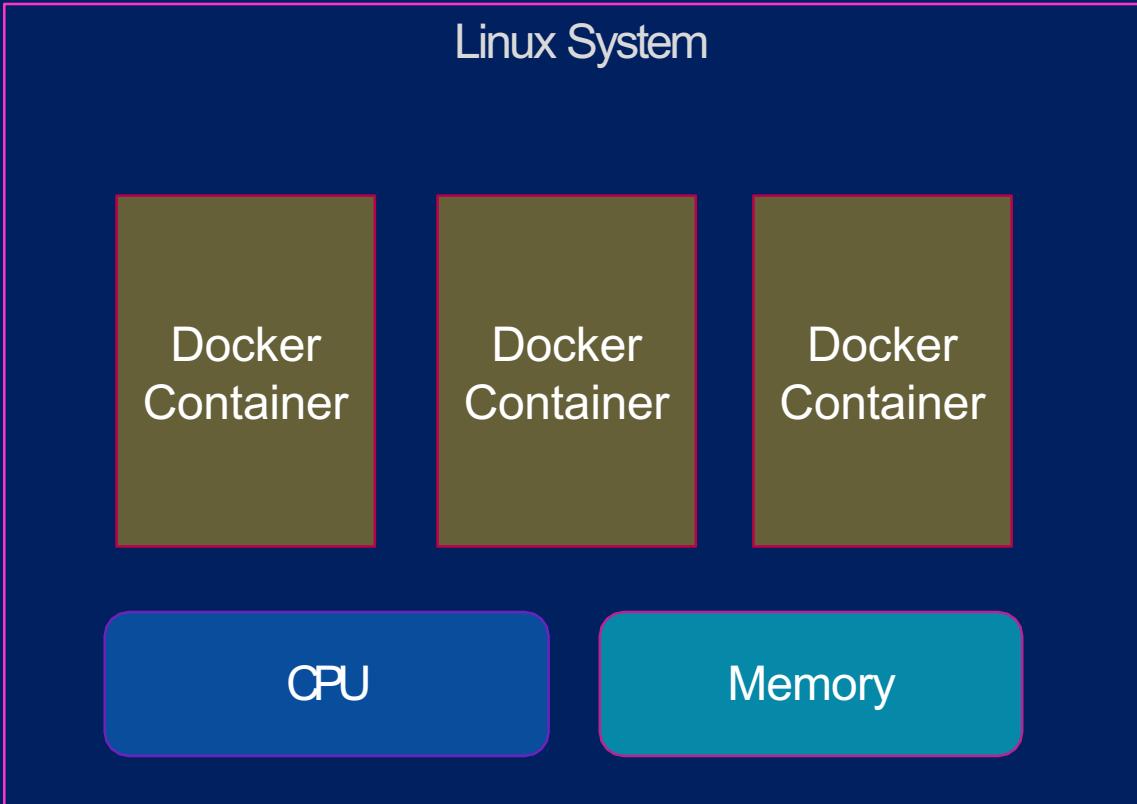
containerization



Namespace - PID



cgroups



```
docker run --cpus=.5 ubuntu
```

```
docker run --memory=100m ubuntu
```


d o c k e r

On Windows

Docker on windows

1. Docker on Windows using Docker Toolbox
2. Docker Desktop for Windows

1. Docker toolbox

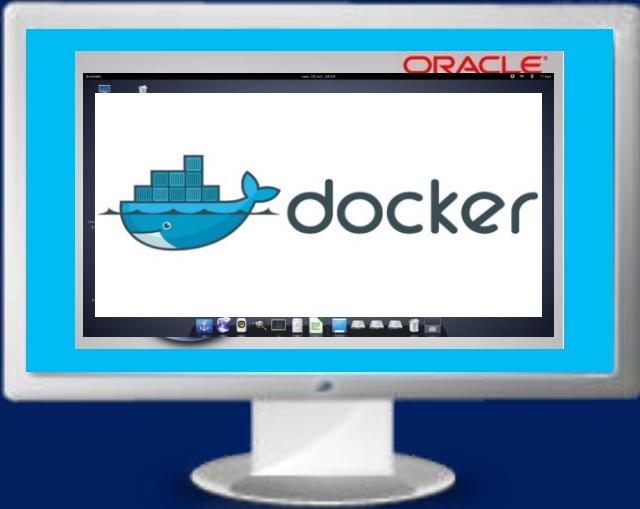


- 64-bit operating
- Windows 7 or higher.
- Virtualization is enabled



- Oracle Virtualbox
- Docker Engine
- Docker Machine
- Docker Compose
- Kitematic GUI

2. Docker Desktop for Windows



Support: Windows 10 Enterprise/Professional Edition
Windows Server 2016

Linux Containers (Default)
Or
Windows Containers

About Docker

Discover Docker Enterprise Edition

Settings

Check for Updates

Diagnose and Feedback...

Switch to Windows containers...

Docker Store

Documentation

Kitematic

Sign in / Create Docker ID...

Repositories

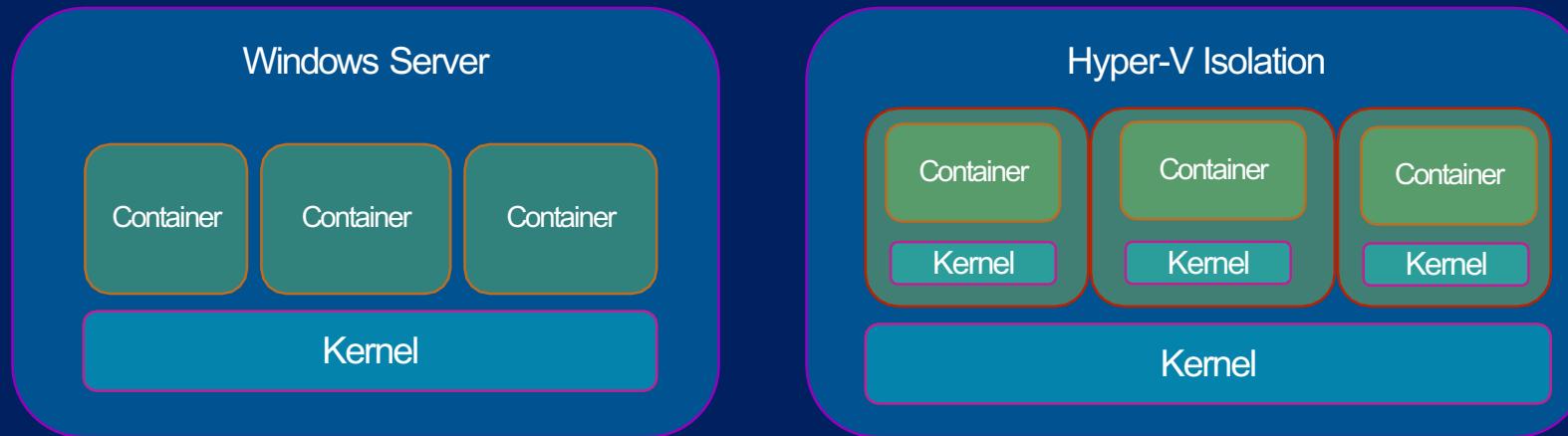
Kubernetes

Restart...

Quit Docker

Windows containers

Container Types:



Base Images:

- Windows Server Core
- Nano Server

Support

- Windows Server 2016
- Nano Server
- Windows 10 Professional and Enterprise (Hyper-V Isolated Containers)

VirtualBox Or Hyper-V

d o c k e r

On Mac

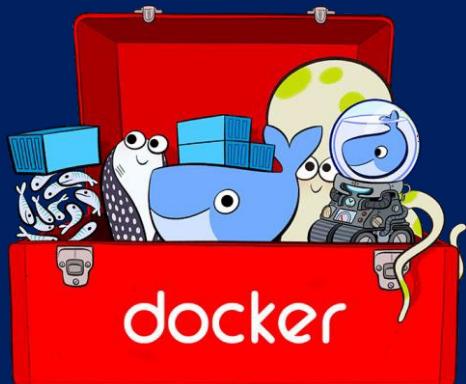
Docker on Mac

1. Docker on Mac using Docker Toolbox
2. Docker Desktop for Mac

1. Docker toolbox

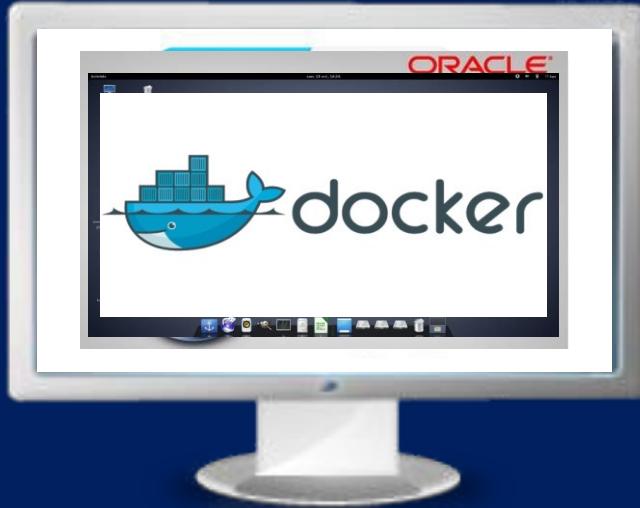


- macOS 10.8 “Mountain Lion” or newer



- Oracle Virtualbox
- Docker Engine
- Docker Machine
- Docker Compose
- Kitematic GUI

2. Docker Desktop for Mac



Support: macOS Sierra 10.12 or newer
Mac Hardware - 2010 model or newer

Linux Containers

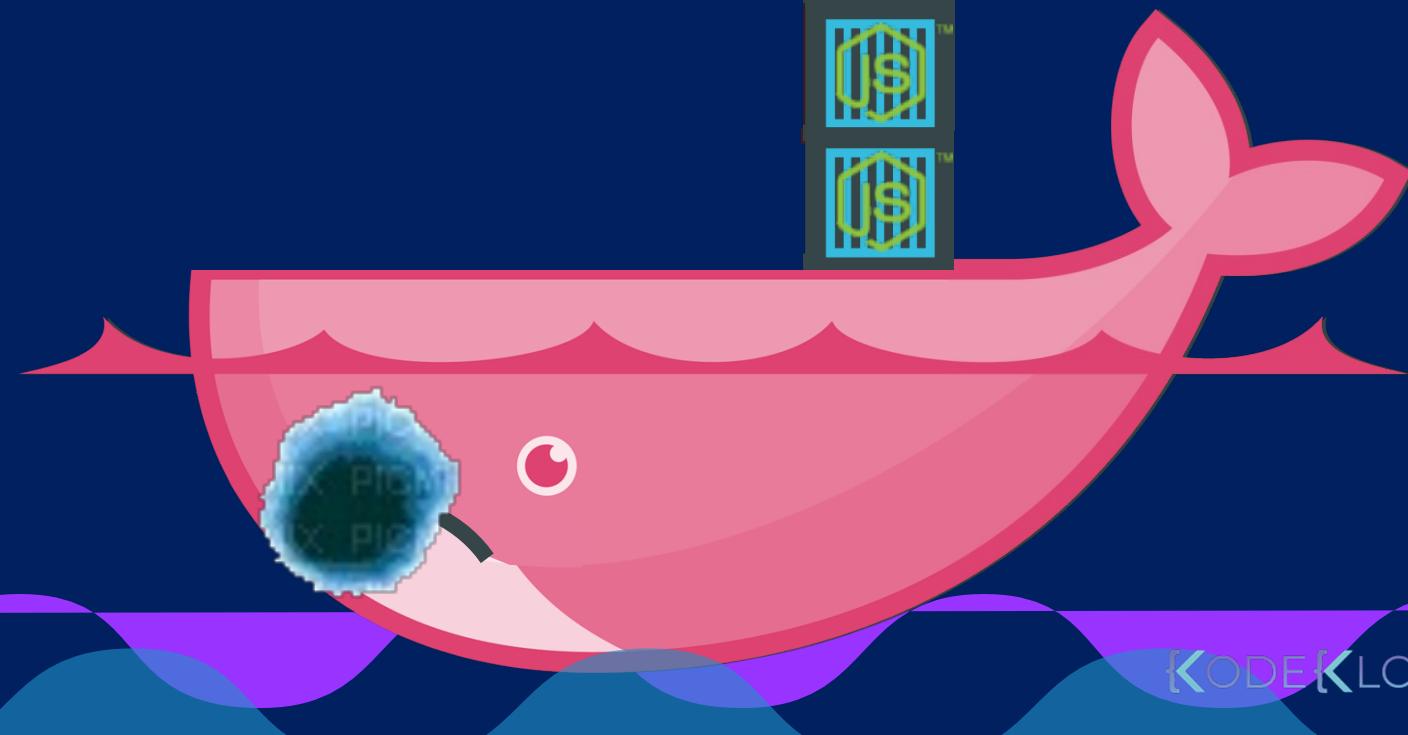
container orchestration

Why Orchestrate?



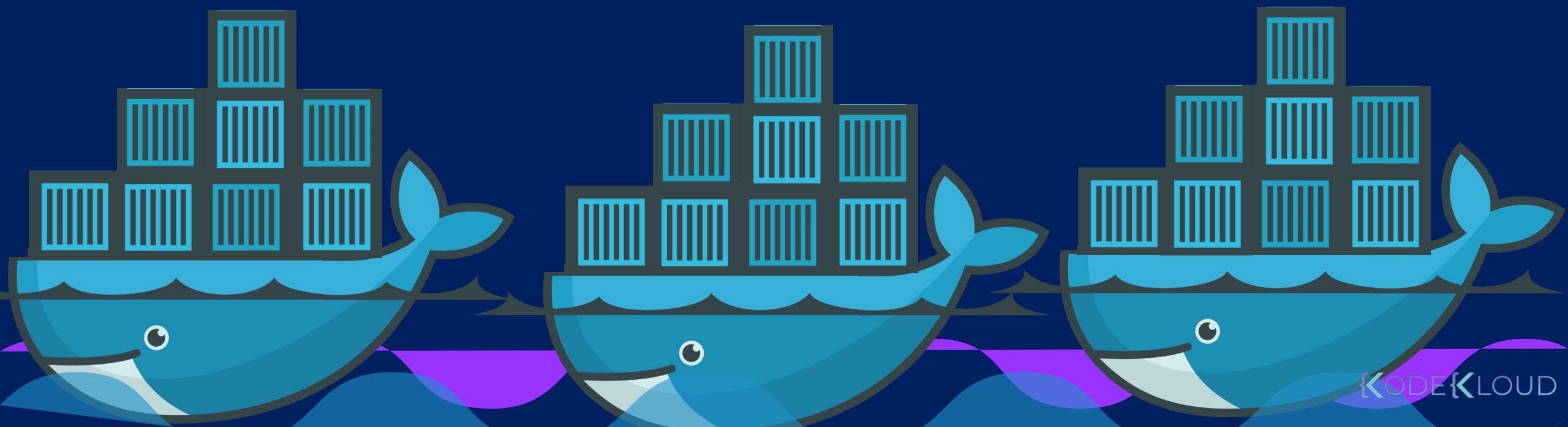
Public Docker registry - dockerhub

```
docker run nodejs  
docker run nodejs  
docker run nodejs  
docker run nodejs
```



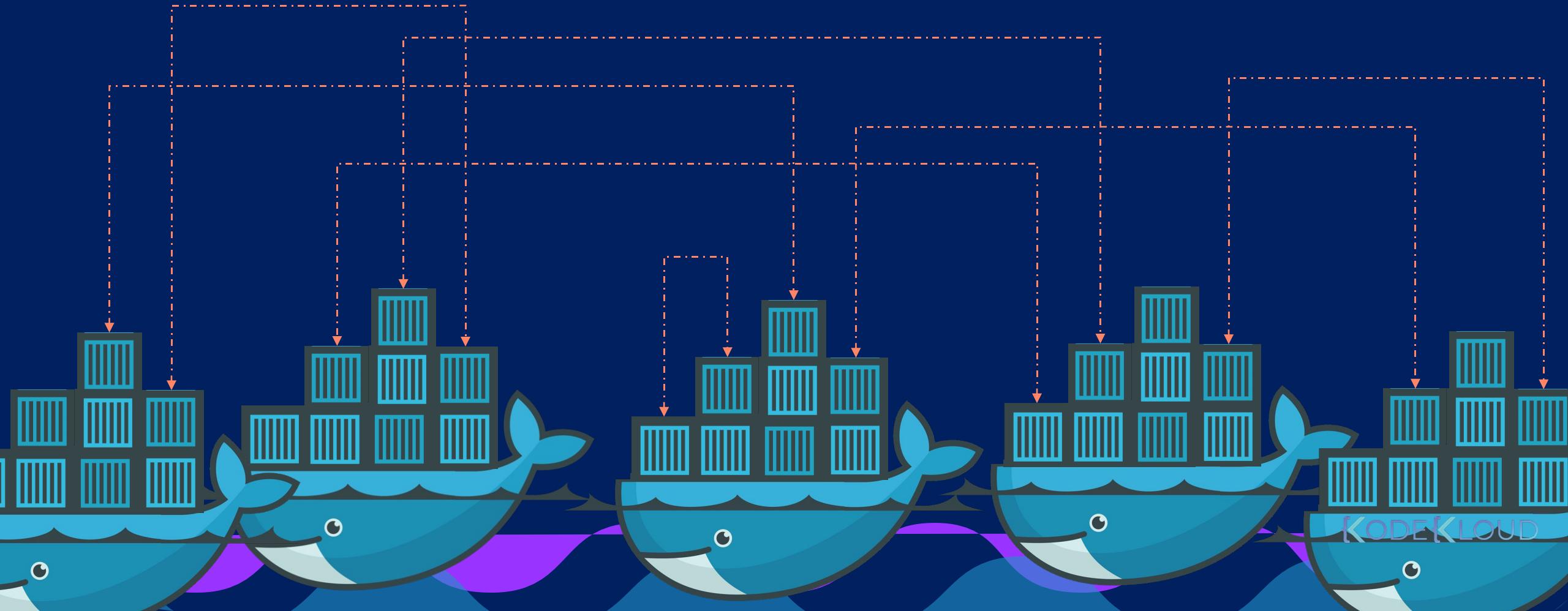
Container Orchestration

```
docker service create --replicas=100 nodejs
```



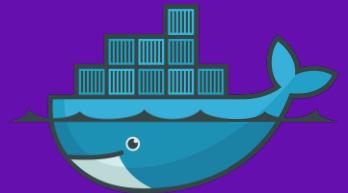
Container Orchestration

```
docker service create --replicas=100 nodejs
```



d o c k e r swarm

Solutions



Docker Swarm

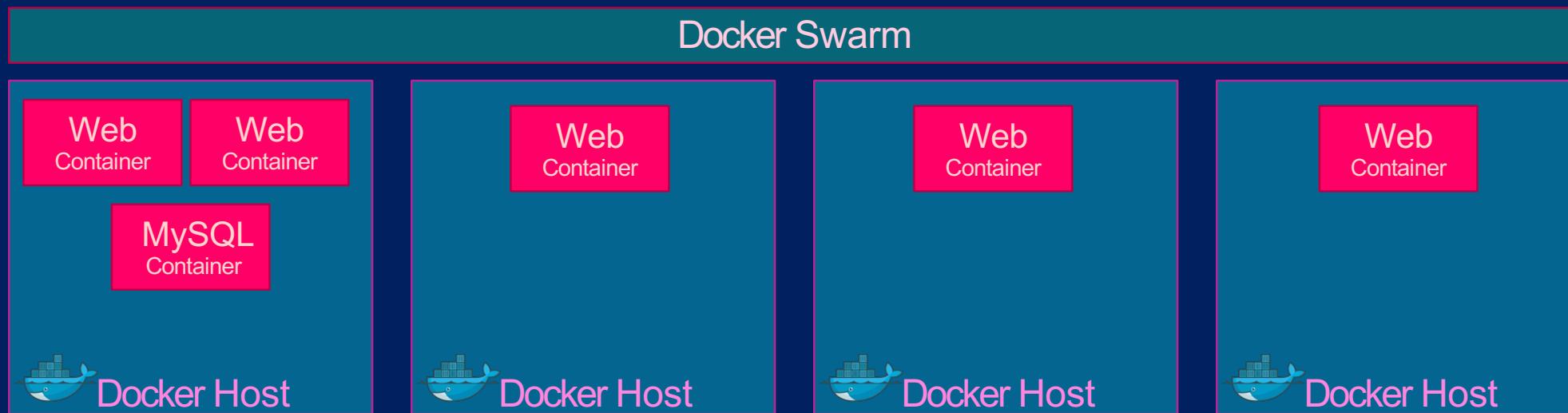


kubernetes



MESOS

Docker swarm



Setup swarm

Swarm Manager

```
docker swarm init
```



Node
Worker

```
docker swarm join  
--token <token>
```



Node
Worker

```
docker swarm join  
--token <token>
```



Node
Worker

```
docker swarm join  
--token <token>
```



```
root@osboxes:/root/simple-webapp-docker # docker swarm init --advertise-addr 192.168.1.12  
Swarm initialized: current node (0j76dum2r56p1xfne4ub1ps2c) is now a manager.
```

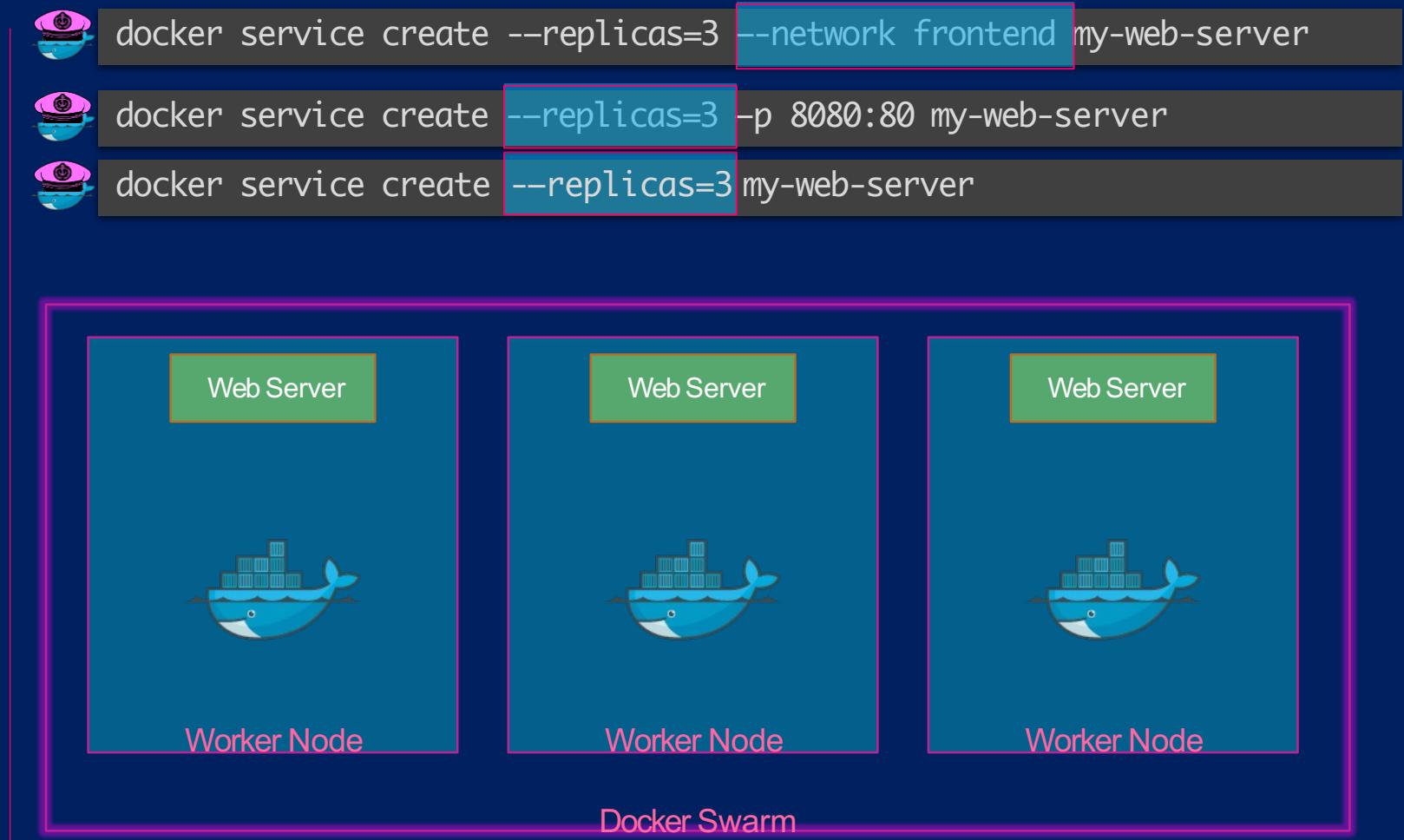
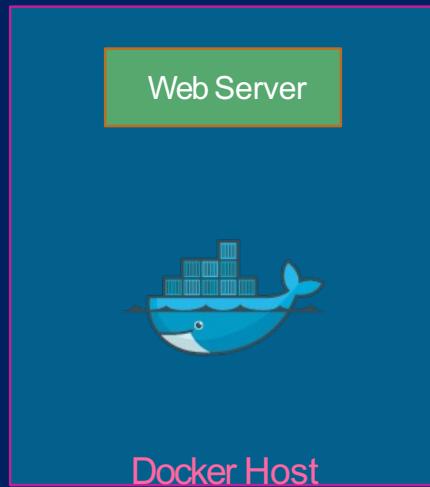
To add a worker to this swarm, run the following command:

```
docker swarm join --token SWMTKN-1-35va8b3fi5krpdskefqqxgttmulw3z828daucri7y526ne0sgu-2eek9qm33d4lxzoq6we9i8izp 192.16  
8.1.12:2377
```

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

Docker service

```
docker run my-web-server
```



kubernetes





```
docker run my-web-server
```



```
kubectl rolling-update my-web-server --rollback
```

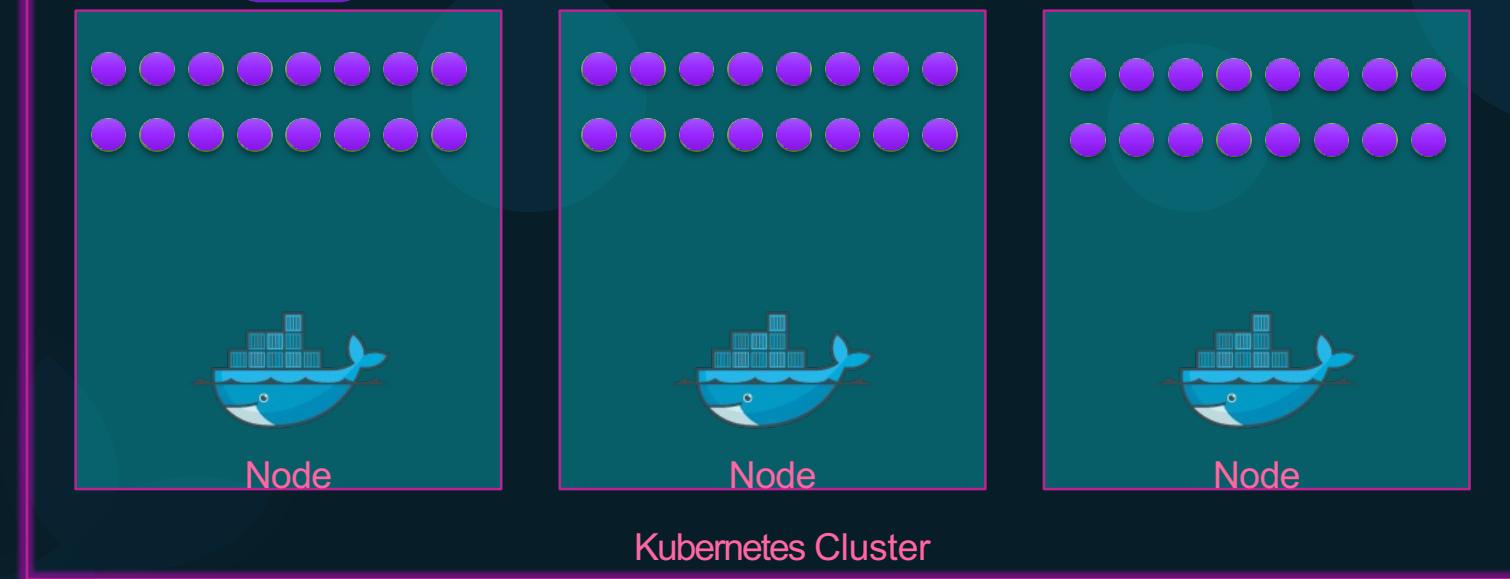
```
kubectl rolling-update my-web-server --image=web-server:2
```

```
kubectl scale --replicas=2000 my-web-server
```

```
kubectl run --replicas=1000 my-web-server
```

A

B



Security

POD AutoScalers

Cluster AutoScalers

Network

weaveworks

flannel

cilium

SCALEIO

vmware
NSX



Storage

ceph

GlusterFS

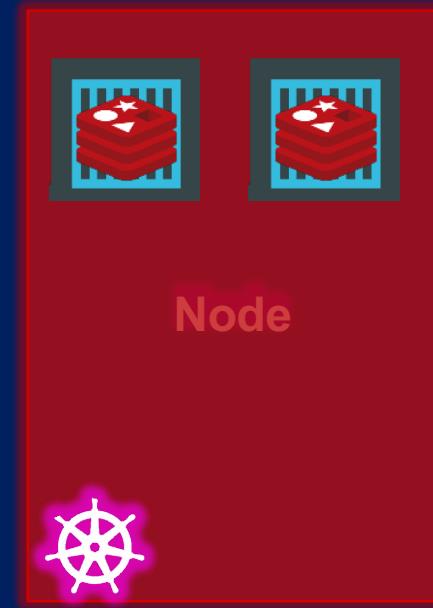
Amazon
Web Services

Red Hat OpenShift

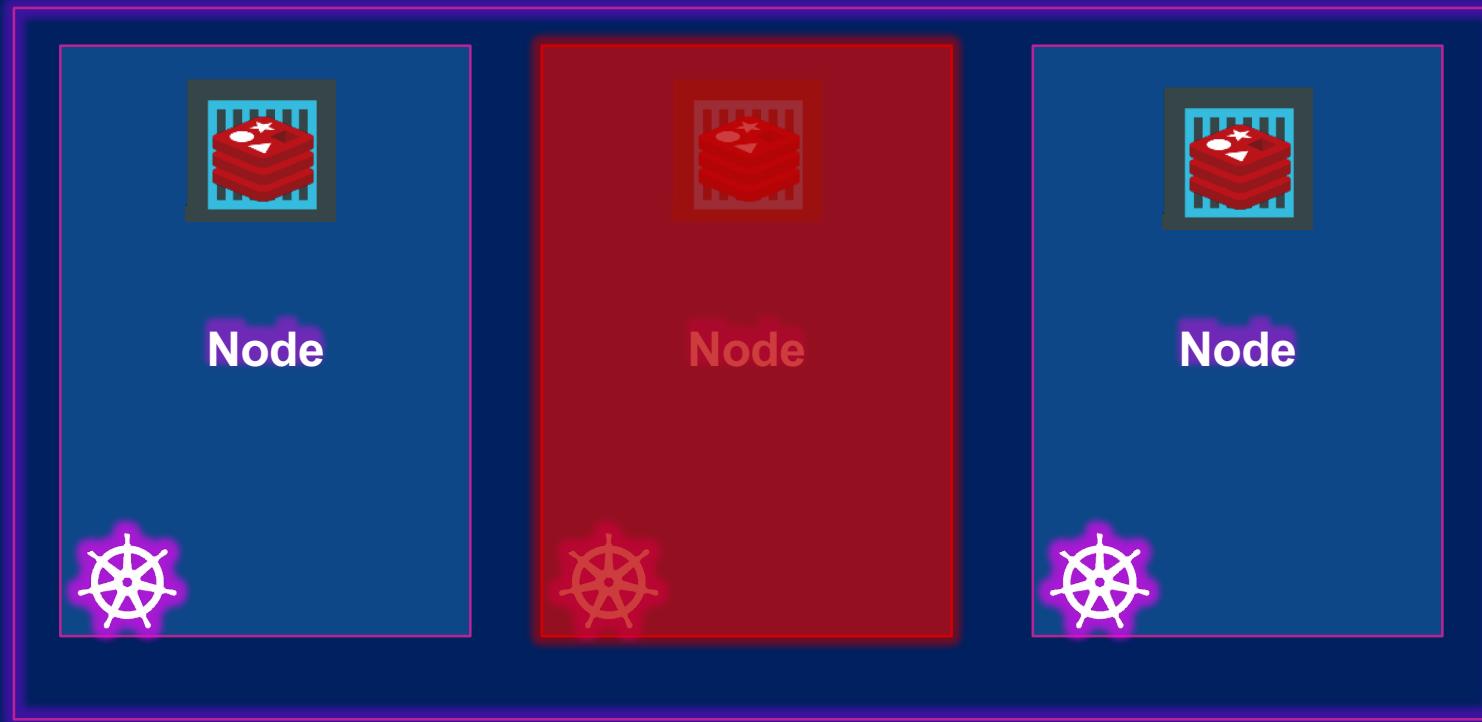
Flocker
by ClusterHQ

KODE KLOUD

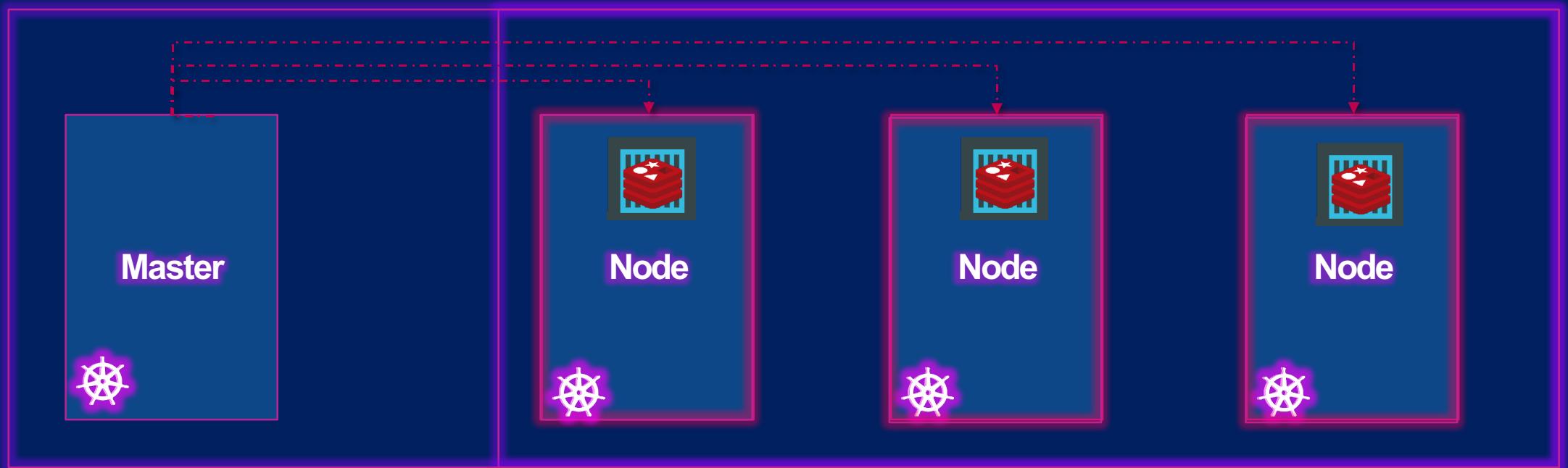
Nodes (Minions)



Cluster



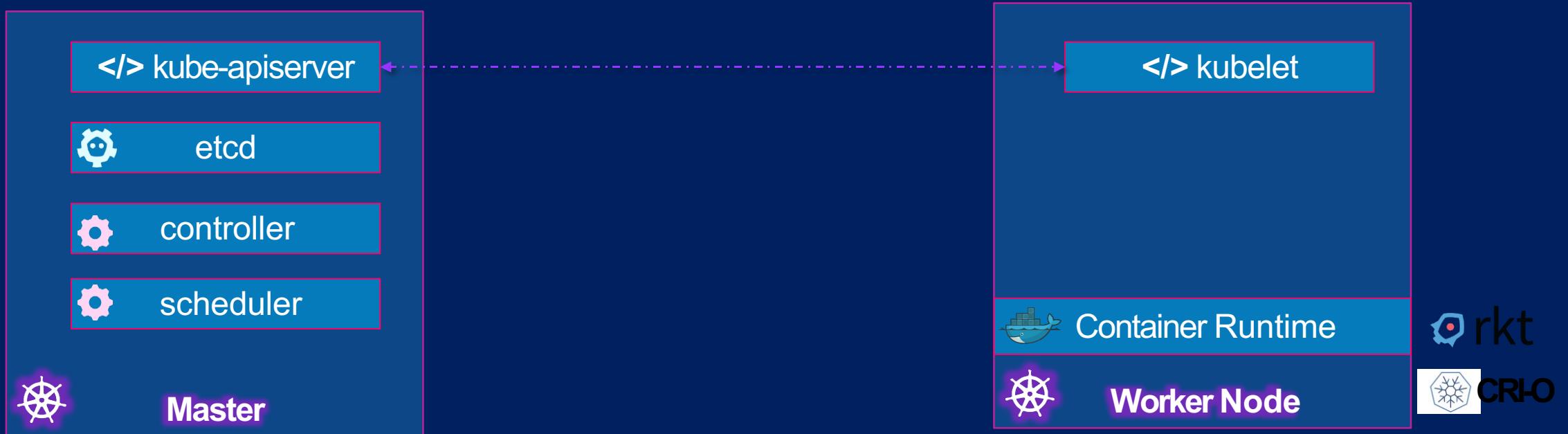
Master



Components



Master vs Worker Nodes



kubectl

```
kubectl run hello-minikube
```



```
kubectl cluster-info
```



```
kubectl get nodes
```



```
kubectl run my-web-app --image=my-web-app --replicas=100
```


CONCLUSION

A large, white, sans-serif font word "CONCLUSION" is centered on a dark blue background. Behind the letters, there are three stylized fireworks: one on the left in gold, one in the center in green and blue, and one on the right in yellow. There are also small, colorful confetti-like dots scattered around the text.

