

# DOCKER

Puripat Rattanayuonyong

Docker คือ “Software Container”  
ที่เป็นการสร้าง “สภาพแวดล้อมเฉพาะ” ให้กับซอฟต์แวร์  
ต่าง ๆ และทำให้ซอฟต์แวร์เหล่านั้นทำงานบนเครื่องไม่  
ก็ได้จะได้ผล เมื่อนั้นกัน

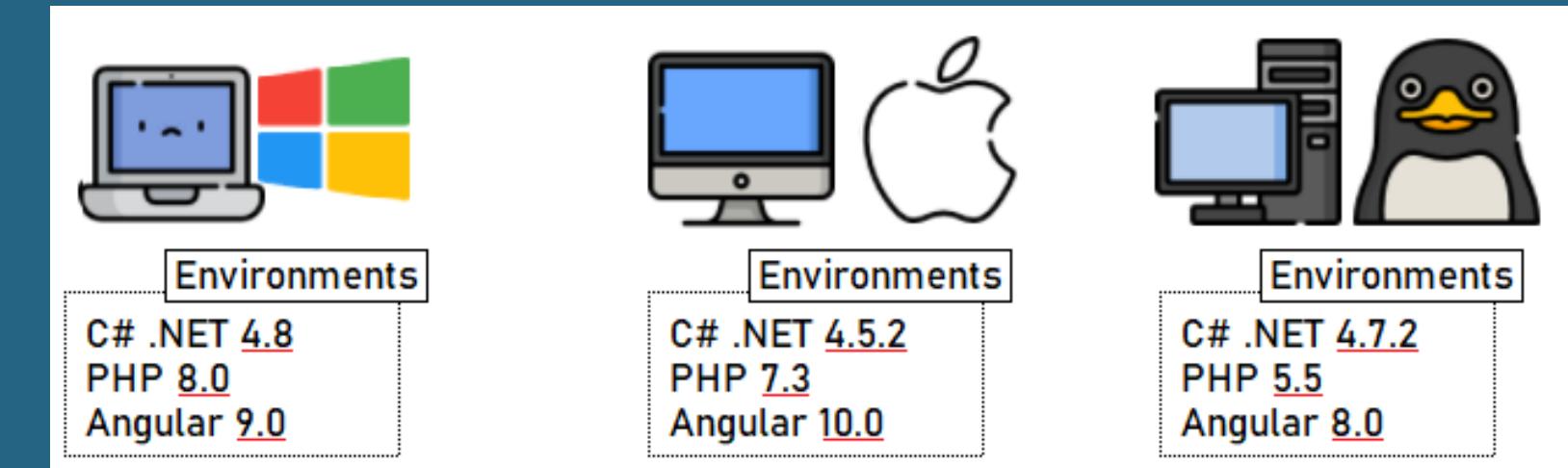
# ปัญหาที่เราเจอกับบ่อโยต

## เสียเวลาติดตั้ง Application

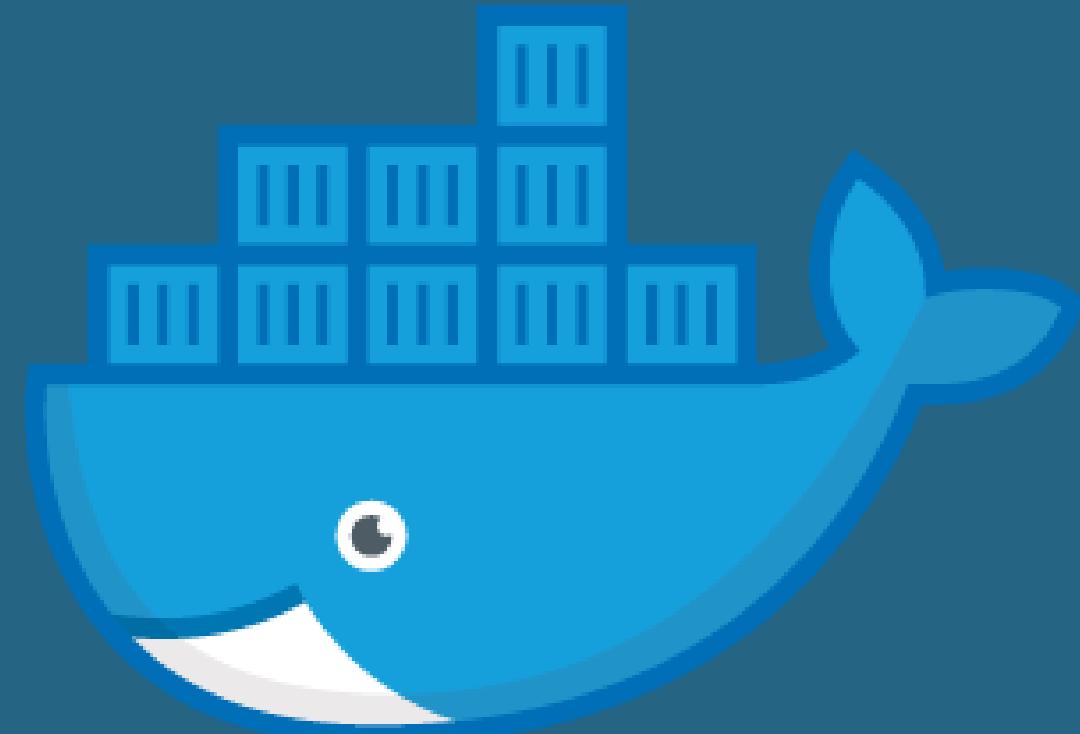
- nginx, PHP, MySQL, Wordpress

## เครื่อง Dev กับ Production ไม่เหมือนกัน

- OS Windows, PHP 8.0, Angular 9.0
- OS MacOS, PHP 7.3, Angular 10.0



# ปัญหาหน้าไป เมื่อมี DOCKER



docker®

Docker มีความสามารถในการ “ห่อ” application ต่างๆ ให้อยู่ในรูปของ “container” ที่เราสามารถนำไปติดตั้งที่เครื่องไหนก็ได้ที่มี Docker รันอยู่

# RUN ON CENTOS 7

```
[root@mgrl pre1]# docker compose up -d
[+] Running 7/7
  ● Network pre1_elasticsearch-net    Created
  ● Network pre1_mongo-net          Created
  ● Container elasticsearch        Healthy
  ● Container mongo                Healthy
  ● Container logstash             Started
  ● Container cron                 Started
  ● Container sdp-la               Started
[root@mgrl pre1]#
```

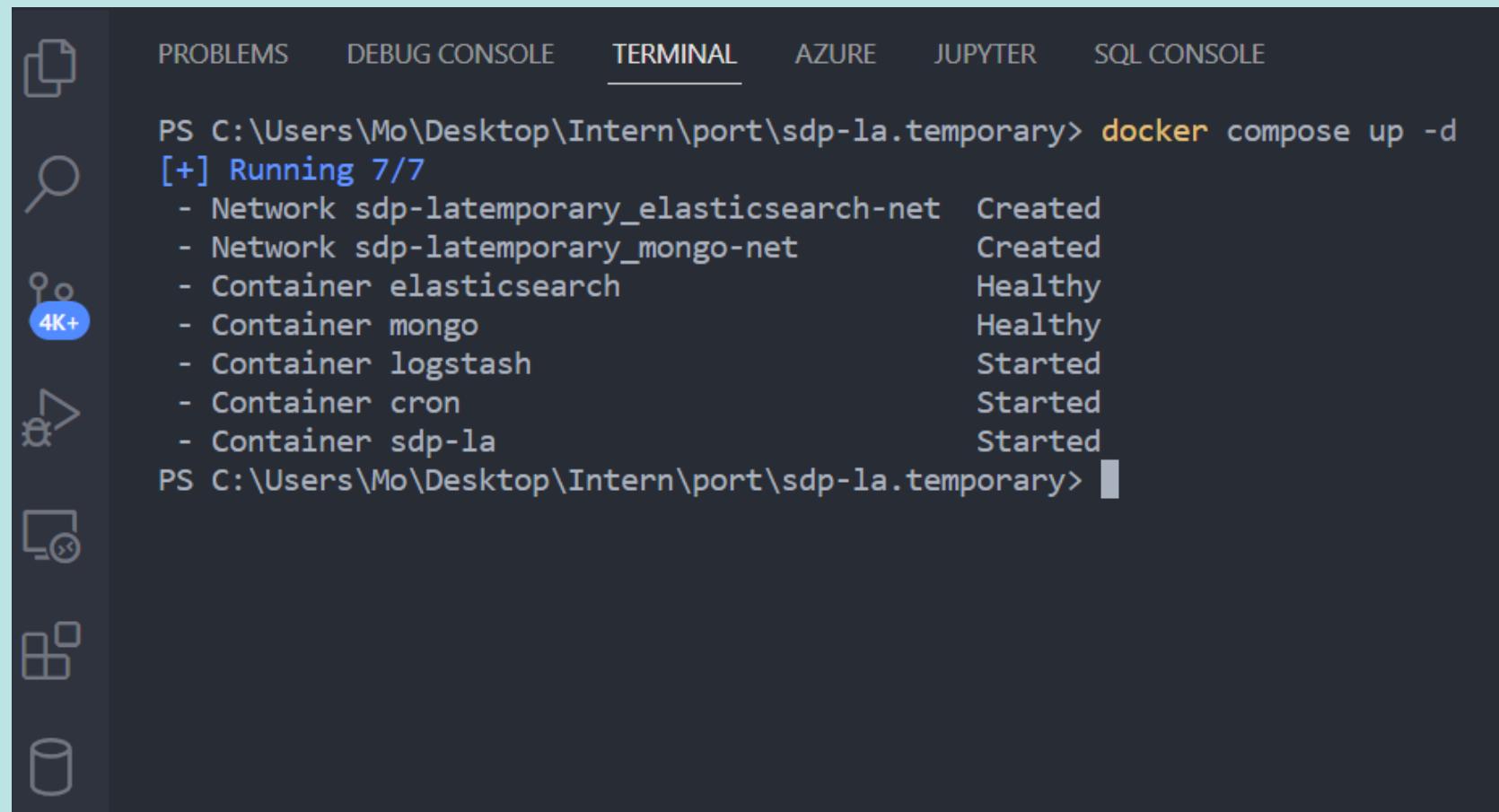


Username

Password

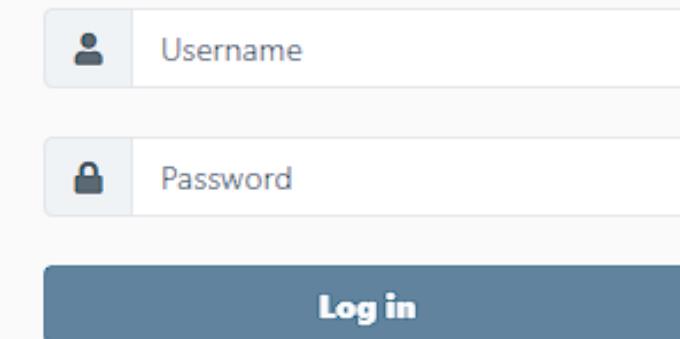
**Log in**

# RUN ON WINDOW 10



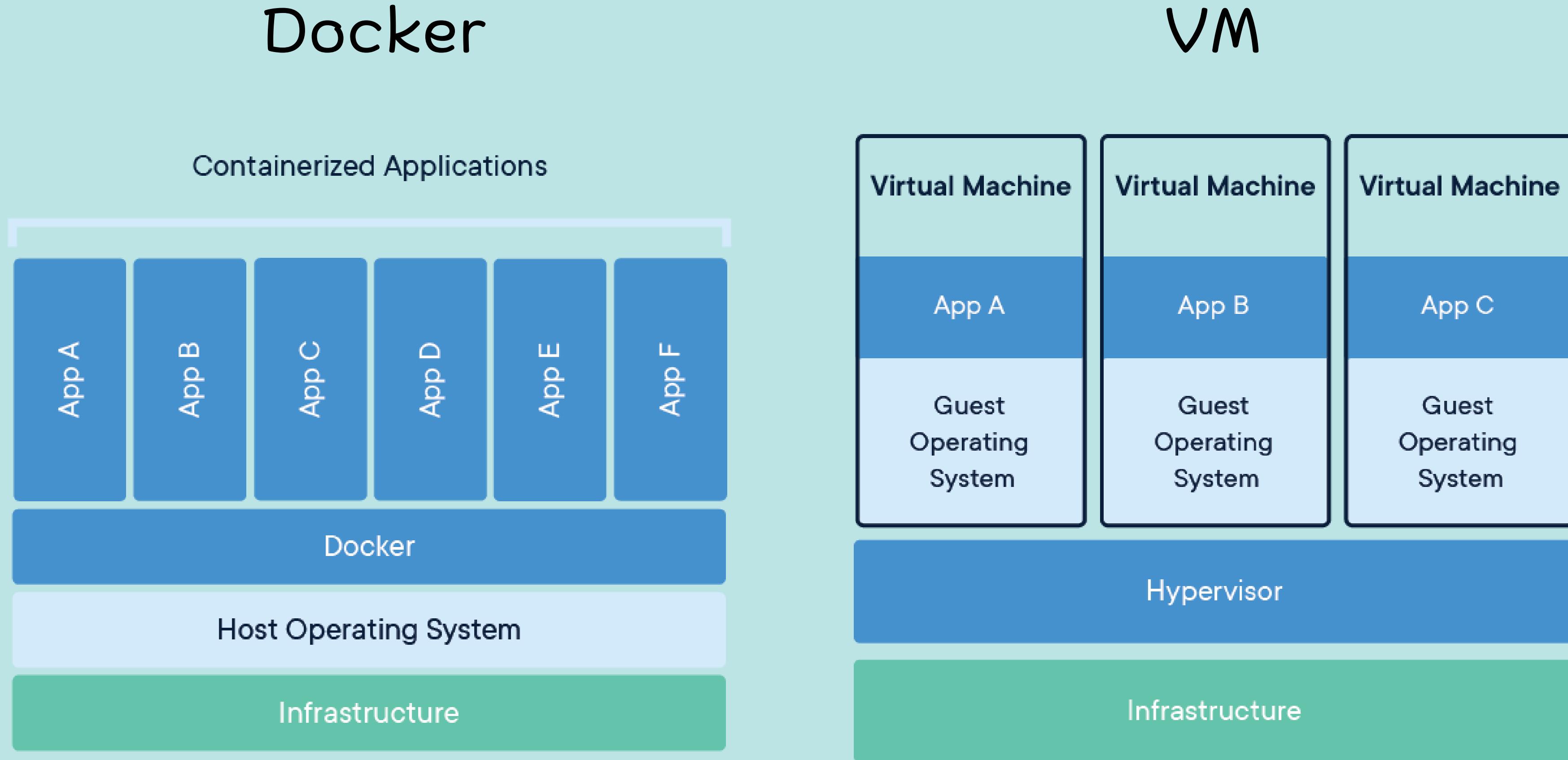
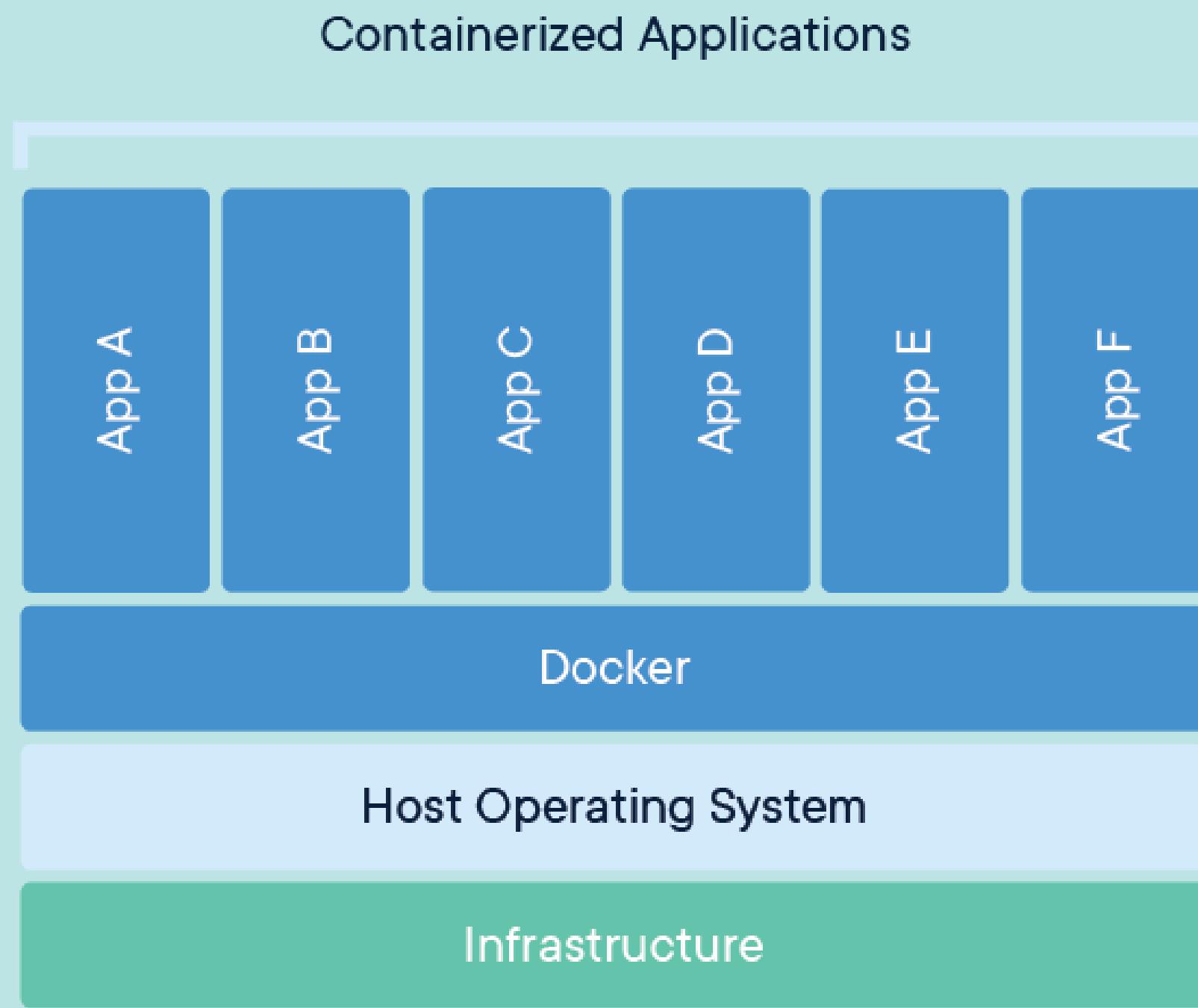
PROBLEMS DEBUG CONSOLE TERMINAL AZURE JUPYTER SQL CONSOLE

```
PS C:\Users\Mo\Desktop\Intern\port\sdp-la.temporary> docker compose up -d
[+] Running 7/7
- Network sdp-latemporary_elasticsearch-net  Created
- Network sdp-latemporary_mongo-net          Created
- Container elasticsearch                   Healthy
- Container mongo                         Healthy
- Container logstash                      Started
- Container cron                          Started
- Container sdp-la                        Started
PS C:\Users\Mo\Desktop\Intern\port\sdp-la.temporary>
```

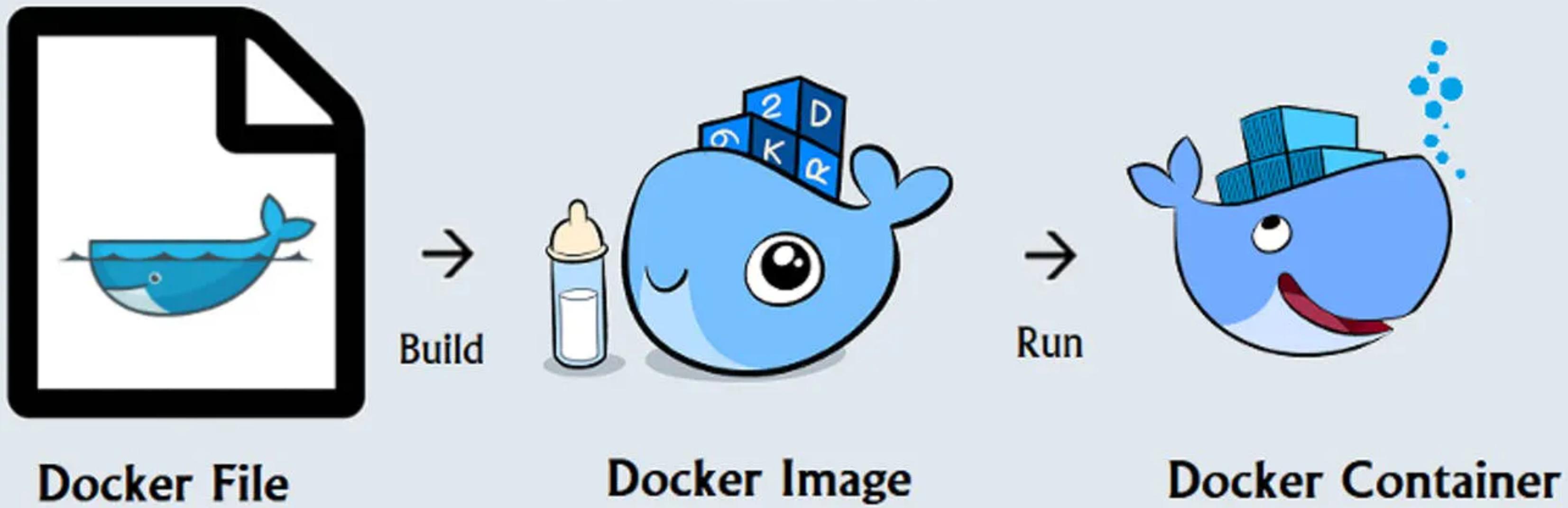


The login form consists of three fields: a "Username" field with a user icon, a "Password" field with a lock icon, and a blue "Log in" button.

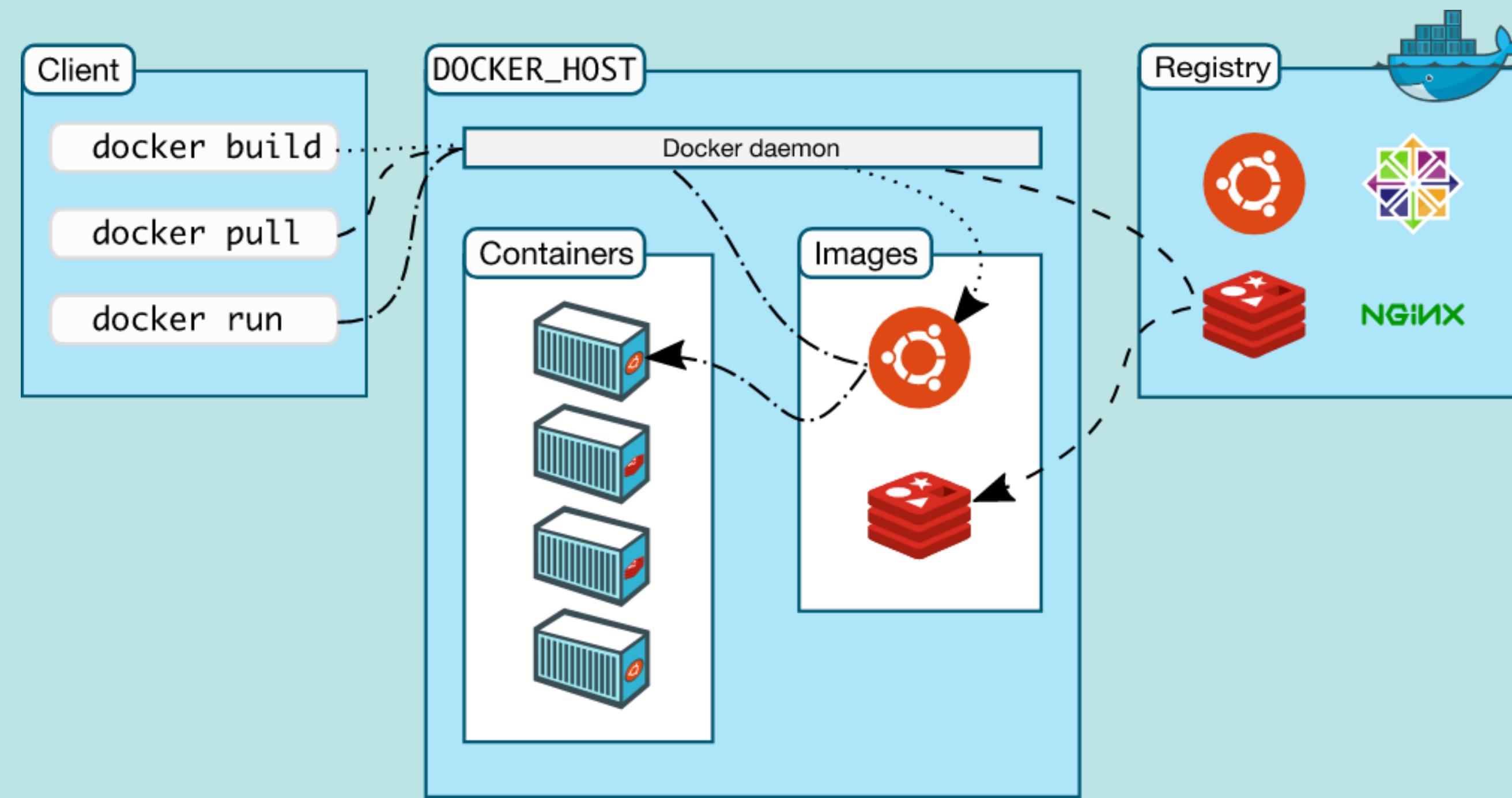
# Docker



## HOW TO CREATE DOCKER IMAGE BY USING DOCKER FILE



# DOCKER ARCHITECTURE



# Dockerfile คืออะไร

ชุดคำสั่งที่เราเตรียมไว้กับ Docker เอาไปสร้างเป็น Image โดยชุดคำสั่งพวกนี้สามารถกำหนดได้หมดเลย ว่า เราจะให้มันติดตั้งอะไรให้เราบาง

# Dockerfile

```
FROM ubuntu:20.04
```

เป็นคำสั่งที่กำหนดว่าจะใช้ base image ตัวไหน

```
WORKDIR /usr/src/app
```

กำหนด working directory

```
COPY . .
```

copy ไฟล์จาก directory ของ local

```
RUN apt-get update && \
```

จะให้กำจัดร่องน้ำเริ่มรัน container

```
apt-get -y upgrade && \
```

```
apt-get install -y build-essential && \
```

```
apt-get install -y byobu curl git htop man unzip vim wget && \
```

```
rm -rf /var/lib/apt/lists/*
```

```
EXPOSE 5000
```

กำหนด port ของ container

```
CMD ["echo", "Hello Docker"]
```

คำสั่งนี้จะ run ตอนที่ทุกอย่างเสร็จสิ้นแล้ว

# Dockerfile

## CMD VS ENTRYPOINT

### CMD

```
FROM centos:8.1.1911  
CMD ["echo", "Hello Docker"]
```

```
$ docker run <image-id>  
Hello Docker  
$ docker run <image-id> hostname  
# hostname is exec to override CMD  
244af....
```

### ENTRYPOINT

```
FROM centos:8.1.1911  
ENTRYPOINT ["echo", "Hello Docker"]
```

```
$ docker run <image-id>  
Hello Docker  
$ docker run <image-id> hostname  
# hostname as parameter to exec  
Hello Docker hostname
```

# Dockerfile

CMD + ENTRYPOINT

```
FROM centos:8.1.1911
ENTRYPOINT ["echo", "Hello"]
CMD ["Docker"]
```

docker run <image-id>

**Hello Docker**

docker run <image-id> Ben

**Hello Ben**

# Docker Build

เมื่อเราเขียน Docker File ขึ้นมาแล้ว เราจะสามารถ Build มันเป็น Docker Image ได้ด้วยคำสั่ง

```
docker build -t <image-name>:<tag> <directory>
```

```
[root@localhost demo]# docker build -t demo:v1 .
Sending build context to Docker daemon 124.4MB
Step 1/6 : FROM ubuntu:20.04
20.04: Pulling from library/ubuntu
d7bfe07ed847: Pull complete
```

# Docker image คืออะไร

Docker image เป็นตัวต้นแบบของ container ซึ่งภายในจะประกอบด้วย application ต่างๆ ที่มีการติดตั้งไว้เพื่อใช้งานสำหรับ service นั้นๆ รวมทั้งมีการ config ค่าต่างๆ ไว้เรียบร้อยแล้ว

# Docker image

เราสามารถดูข้อมูลเกี่ยวกับ image ได้ด้วยคำสั่ง  
docker images / docker image ls

```
[root@wrk1 demo]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
demo	latest	fe5b7f552a4f	About a minute ago	466MB
mo0303/tastfile	latest	6b535be690f8	3 days ago	113MB
sdp-la	latest	2b04177ab415	4 days ago	674MB
sdp-latempory_sdp-la	latest	a75fe32924fd	5 days ago	674MB
sdp-latempory_cron	latest	c56e0d72eaed	5 days ago	79MB
sdp-backend	latest	863f46c5493e	5 days ago	794MB

# Docker image

docker images / docker image ls แสดง image ก็งหมดที่มีของเรา

docker rmi (image name) ลบ image

docker image inspect ใช้ดูรายละเอียดของ image

docker image build ใช้สร้าง image จาก Dockerfile

# Docker pull

เราสามารถนำ Docker image จาก Docker Hub Repository มาใช้ได้  
ด้วยคำสั่ง docker pull image\_name

```
[root@wrk1 demo]# docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
b85a868b505f: Pull complete
f4407baf103: Pull complete
4a7307612456: Pull complete
935cecace2a0: Pull complete
8f46223e4234: Pull complete
fe0ef4c895f5: Pull complete
Digest: sha256:10f14ffa93f8dedf1057897b745e5ac72ac5655c299dade0aa434c71557697ea
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
You have new mail in /var/spool/mail/root
[root@wrk1 demo]# |
```

# Docker push

นอกจากเราจะสามารถ pull image ได้แล้ว เรา ก็สามารถ push image ขึ้นไปที่ Docker Hub Repository ได้เช่นกัน โดยใช้คำสั่ง  
docker push <tag>

```
[root@wrk1 demo]# docker login
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: mo0303
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@wrk1 demo]# docker tag demo mo0303/testdemo:v1
[root@wrk1 demo]# docker push mo0303/testdemo:v1
The push refers to repository [docker.io/mo0303/testdemo]
e84fed6fe5fb: Pushed
e6ec3abf388a: Pushed
086214bd8f54: Pushed
af7ed92504ae: Mounted from library/ubuntu
v1: digest: sha256:1eb515e8198c88368582f13c97a777c15e351916804f0af3308b1d2a6b92e5fe size: 1156
You have new mail in /var/spool/mail/root
[root@wrk1 demo]#
```

# Docker local registry

นอกนี้จากเราจะสามารถ push/pull image จาก Docker Hub Repository ได้แล้ว เราสามารถ push/pull image ไปที่ local registry ได้เช่นกัน

```
[root@wrk1 demo]# docker run -d -p 5000:5000 --restart=always --name registry registry:2
Unable to find image 'registry:2' locally
2: Pulling from library/registry
2408cc74d12b: Already exists
ea60b727a1ce: Pull complete
c87369050336: Pull complete
e69d20d3dd20: Pull complete
fc30d7061437: Pull complete
Digest: sha256:bedef0f1d248508fe0a16d2cacea1d2e68e899b2220e2258f1b604e1f327d475
Status: Downloaded newer image for registry:2
1086b183090e5a79a8e5808a87b8d09c677e22e6587b73673c71a63c231cbfed
You have new mail in /var/spool/mail/root
[root@wrk1 demo]# docker tag demo localhost:5000/my-demo
[root@wrk1 demo]# docker push localhost:5000/my-demo
Using default tag: latest
The push refers to repository [localhost:5000/my-demo]
e84fed6fe5fb: Pushed
e6ec3abf388a: Pushed
086214bd8f54: Pushed
af7ed92504ae: Pushed
latest: digest: sha256:1eb515e8198c88368582f13c97a777c15e351916804f0af3308b1d2a6b92e5fe size: 1156
[root@wrk1 demo]# docker pull localhost:5000/my-demo
Using default tag: latest
latest: Pulling from my-demo
Digest: sha256:1eb515e8198c88368582f13c97a777c15e351916804f0af3308b1d2a6b92e5fe
Status: Image is up to date for localhost:5000/my-demo:latest
localhost:5000/my-demo:latest
[root@wrk1 demo]#
```

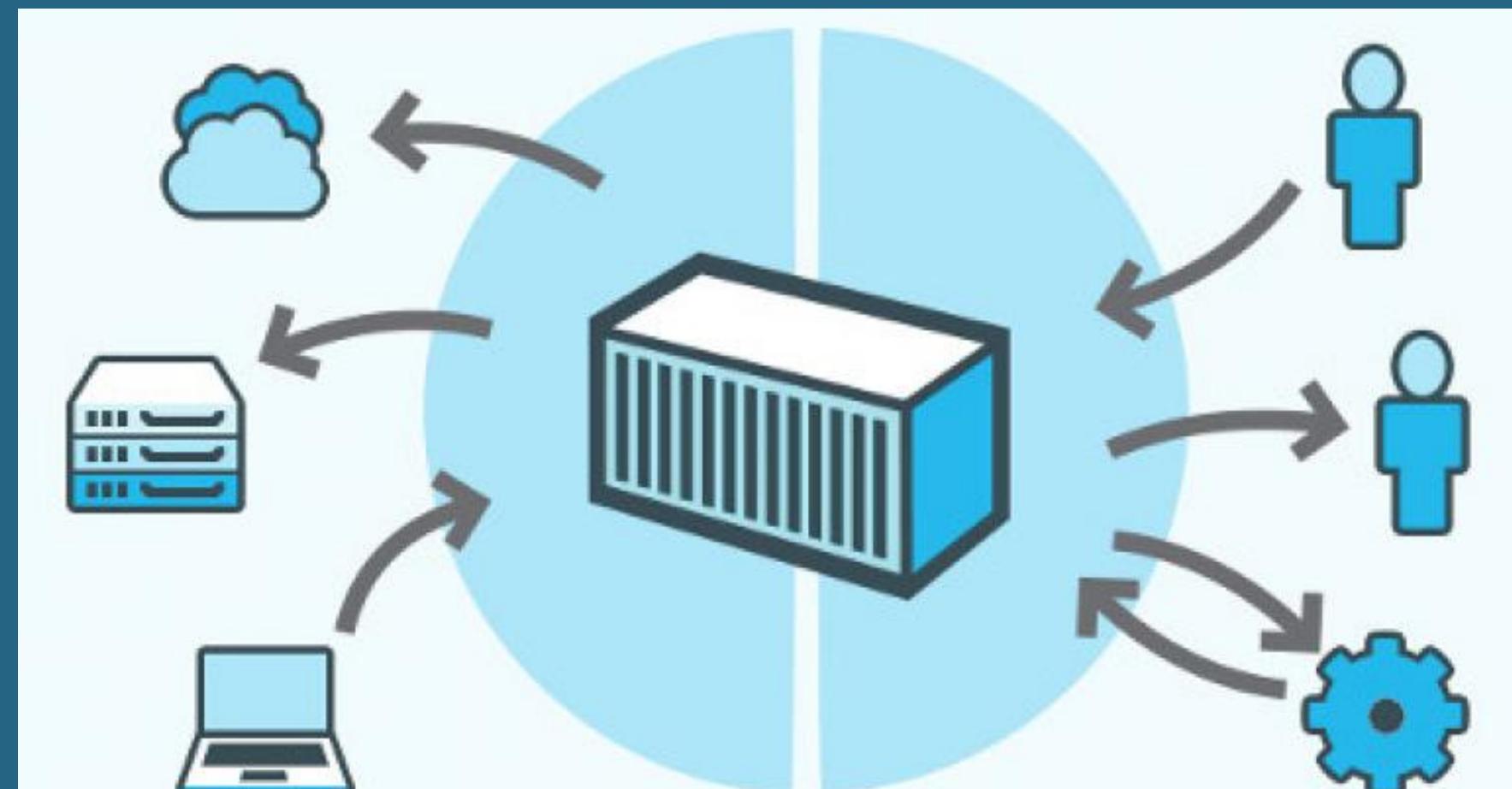
# Docker run

เมื่อเราได้ image จากการ build Dockerfile หรือ pull image จาก Docker Hub แล้ว เราสามารถ Run Docker Image ขึ้นมาเป็น Container ได้ด้วยคำสั่ง docker run image\_name

```
[root@mgr1 demo]# docker run demo
Hello Docker
[root@mgr1 demo]# |
```

# Docker container คืออะไร

Docker container เปรียบเสมือนกล่อง ซึ่งนำ docker image มาติดตั้ง เพื่อให้สามารถใช้งาน service ที่ต้องการจาก image นั้นๆ ได้



# Docker exec

เราสามารถ run command ใน Container ที่กำลัง Run อยู่ ด้วยคำสั่ง  
docker exec -it image\_name <command>

```
[root@wrk1 demo]# docker run --name nginxdemo -d -p80:80 nginx
190d362077f3f35f76e5044df24cc0c5d0411d84ddf60dc1fa41c47fb62b342a
[root@wrk1 demo]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
190d362077f3        nginx              "/docker-entrypoint..."   3 seconds ago     Up 2 seconds      0.0.0.0:80->80/tcp, :::80->80/tcp
1086b183090e        registry:2       "/entrypoint.sh /etc..."  45 minutes ago   Up 45 minutes    0.0.0.0:5000->5000/tcp, :::5000->5000/tcp
0c3d1030a062        node:12.18.2    "docker-entrypoint.s..."  6 days ago       Up 6 days
[root@wrk1 demo]# docker exec -it nginxdemo /bin/bash
root@190d362077f3:/# |
```

# Docker run

เราสามารถสั่ง Run พร้อมกับเข้าไปใน Container ได้ด้วยคำสั่ง  
docker run -it image\_name <command>

```
[root@mgr1 demo]# docker run -it nginx /bin/bash
root@ae04b15c4482:/# echo "Hello Demo" > Demo.html
root@ae04b15c4482:/# ls
Demo.html  boot  docker-entrypoint.d  etc  lib   media  opt   root  sbin  sys  usr
bin        dev   docker-entrypoint.sh  home  lib64  mnt   proc  run   srv  tmp  var
root@ae04b15c4482:/# cat Demo.html
Hello Demo
root@ae04b15c4482:/# exit
exit
[root@mgr1 demo]# |
```

# Docker cp

เราสามารถ copy file หรือ folder จากเครื่อง host machine ไปยัง container ได้ด้วยคำสั่ง

```
docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-  
docker cp [OPTIONS] SRC_PATH|- CONTAINER:DEST_PATH
```

```
[root@wrk1 demo]# docker exec -it ubuntu /bin/bash  
root@bb7763575202:/# ls  
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var  
root@bb7763575202:/# exit  
exit  
[root@wrk1 demo]# ls  
demo.html  demomulti  Dockerfile  page.html  
[root@wrk1 demo]# docker cp ./ ubuntu:.  
[root@wrk1 demo]# docker exec -it ubuntu /bin/bash  
root@bb7763575202:/# ls  
Dockerfile  boot      demomulti  etc      lib      lib64      media      opt      proc      run      srv      tmp      var  
bin          demo.html  dev        home    lib32    libx32    mnt      page.html  root    sbin    sys    usr  
root@bb7763575202:/# exit
```

# Docker save & load

docker save เป็นการ save docker image ให้อยู่ในรูปแบบไฟล์  
โดยใช้คำสั่ง docker save [OPTIONS] IMAGE [IMAGE...]

docker load เป็นการ extract file เป็น docker image  
โดยใช้คำสั่ง docker load -i (path file image)

```
[root@wrk1 demo]# docker save -o demo.tar.gz demo
You have new mail in /var/spool/mail/root
[root@wrk1 demo]# ls
demo.html  demomulti  demo.tar.gz  Dockerfile  page.html
[root@wrk1 demo]# docker load -i demo.tar.gz
Loaded image: demo:latest
```

# BEFORE bindmount

```
[root@wrk1 demo]# docker run --name beforemount -p 80:80 -d nginx
4b4ca344bdc1e5b14b5306f9142a4d3eacd9b5534f2636d72058e6d34f815478
[root@wrk1 demo]# |
```

# BEFORE bindmount

**Welcome to nginx!**

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org).  
Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

# AFTER bindmount

เราสามารถสั่ง Run พร้อมกับการ mount file หรือ folder จากเครื่อง host machine ไปยัง container

```
[root@wrk1 demo]# ls
demo.html  demomulti  Dockerfile  page.html
[root@wrk1 demo]# pwd
/root/demo
[root@wrk1 demo]# docker run --name demomount -p80:80 -v $PWD/page.html:/usr/share/nginx/html/index.html -d
nginx
8eb3fbc53939cff94b3fb92b1bc05f06e597453ebffff903fa50bc6b76b0be51
[root@wrk1 demo]#
```

# AFTER bindmount

## CSS Layout Float

In this example, we have created a header, two columns/boxes and a footer. On smaller screens, the columns will stack on top of each other.

Resize the browser window to see the responsive effect (you will learn more about this in our next chapter - HTML Responsive.)

## Cities

[London](#)  
[Paris](#)  
[Tokyo](#)

### London

London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.

Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.

Footer

# Docker attach

เราสามารถแบบ terminal เข้าไปใน container ที่กำลัง Run อยู่ ด้วยคำสั่ง docker attach container

```
[root@wrk1 ~]# docker attach ubuntu
root@bb7763575202:/# read escape sequence
[root@wrk1 ~]# docker attach beforemount
192.168.10.254 - - [29/Jun/2022:04:45:48 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT
10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0 Safari/537.36" "-"
^C2022/06/29 04:45:58 [notice] 1#1: signal 2 (SIGINT) received, exiting
2022/06/29 04:45:58 [notice] 26#26: exiting
2022/06/29 04:45:58 [notice] 26#26: exit
2022/06/29 04:45:58 [notice] 28#28: exiting
2022/06/29 04:45:58 [notice] 28#28: exit
2022/06/29 04:45:58 [notice] 25#25: exiting
2022/06/29 04:45:58 [notice] 25#25: exit
2022/06/29 04:45:58 [notice] 27#27: exiting
2022/06/29 04:45:58 [notice] 27#27: exit
2022/06/29 04:45:58 [notice] 1#1: signal 17 (SIGCHLD) received from 26
2022/06/29 04:45:58 [notice] 1#1: worker process 25 exited with code 0
2022/06/29 04:45:58 [notice] 1#1: worker process 26 exited with code 0
2022/06/29 04:45:58 [notice] 1#1: signal 29 (SIGIO) received
2022/06/29 04:45:58 [notice] 1#1: signal 17 (SIGCHLD) received from 27
2022/06/29 04:45:58 [notice] 1#1: worker process 27 exited with code 0
2022/06/29 04:45:58 [notice] 1#1: signal 29 (SIGIO) received
2022/06/29 04:45:58 [notice] 1#1: signal 17 (SIGCHLD) received from 28
2022/06/29 04:45:58 [notice] 1#1: worker process 28 exited with code 0
2022/06/29 04:45:58 [notice] 1#1: exit
```

# Docker attach vs Docker exec

docker attach จะแสดง output ของคำสั่ง CMD  
docker exec จะสามารถรัน command ที่เราต้องการได้

```
[root@wrk1 ~]# docker attach ubuntu
root@bb7763575202:/# read escape sequence
You have new mail in /var/spool/mail/root
[root@wrk1 ~]# docker exec -it ubuntu echo "Hello Docker exec"
Hello Docker exec
[root@wrk1 ~]# |
```

# Docker logs

เราสามารถดู logs ของ Container ได้ด้วยคำสั่ง  
docker logs <container-id> <option>

```
[root@wrk1 demo]# docker logs nginxdemo -f
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/06/27 10:54:33 [notice] 1#1: using the "epoll" event method
2022/06/27 10:54:33 [notice] 1#1: nginx/1.23.0
2022/06/27 10:54:33 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2022/06/27 10:54:33 [notice] 1#1: OS: Linux 3.10.0-1160.66.1.el7.x86_64
2022/06/27 10:54:33 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2022/06/27 10:54:33 [notice] 1#1: start worker processes
2022/06/27 10:54:33 [notice] 1#1: start worker process 31
2022/06/27 10:54:33 [notice] 1#1: start worker process 32
2022/06/27 10:54:33 [notice] 1#1: start worker process 33
2022/06/27 10:54:33 [notice] 1#1: start worker process 34
```

# Docker inspect

เราสามารถใช้คำสั่ง docker inspect container\_name เพื่อดูรายละเอียดต่างๆ ของ container ได้

```
[root@wrk1 ~]# docker inspect registry
[
  {
    "Id": "1086b183090e5a79a8e5808a87b8d09c677e22e6587b73673c71a63c231cbfed",
    "Created": "2022-06-27T10:08:55.7251017Z",
    "Path": "/entrypoint.sh",

    "Mounts": [
      {
        "Type": "volume",
        "Name": "ca9c45312a56d17050b1baebb863a8ca26aa2be5021a6aa4bb82582dc623ecf6",
        "Source": "/var/lib/docker/volumes/ca9c45312a56d17050b1baebb863a8ca26aa2be5021a6aa4bb82582dc623ecf6

        "Destination": "/var/lib/registry",
```

# Docker volume

เมื่อเรา run container และใช้งาน ข้อมูลจะถูกเก็บไว้บน container

ถ้าเราทำการ stop และ remove container นี้ก็จะไปข้อมูลจะหายไปหมด

volume เปรียบเสมือนการจองพื้นที่ใน docker host เพื่อกำการเก็บ file หรือ

ข้อมูลต่างๆที่เราต้องการ แล้วให้ container อ้างอิงมาหา

# Docker volume

docker volume ls |ເພື່ອແສດງ volume ກີ່ນິກັ້ງຮມດ

```
[root@wrk1 ~]# docker volume ls
DRIVER      VOLUME NAME
local      0f778d5252bb96a1bdee789b005a34baf9137b6600a01867b0541dffed9d9a84
local      1bf75e94d02c5ba9a2d764b559135ec3ea2dfb65e7169d82dfef512277739e00
local      1c5f5a31fc12ea5db0cd193260542aad9fbf8110fc0528f950859579dc8d027b
local      2ceec2dd40207db8038a39dbd62e94c7b25478aa002dd2d439968e6bb76858cc
local      8a063141d8219b17794b57b7dc6f0b4131ee09515a43d0f68f4027150d54ad
local      9f36b6a7ad1435e85bfd0c38a2271faff8e46fc0ab962881f1a189f8be13e20d
local      29b729ec4f54c80ac08ebff287cbc8809436a6ed0b2a8d77f8660e33c4bc9ea6
local      38ac8fdd07eda0eee57d970ba5499c401b88030188c4d6490f76d87fa9abef65
local      62a8cb68de29e3dc34bae9862af7637362f4ea019bdd44e4ff67a00a3a5c333b
local      90cdd4493ee90eff5199492add5140c6e96d569b2fa911234b680b3b4c464f96
local      3401b3f19f1c9cb049e7a993cdce9462b67934c90cbf3040d13e38237fa2ad17
local      7783b779a964b9e89138cf905459552871f6a47dcc52367ff70f9292d8e53a78
local      685249b519ded53453db39a845cc0159212f8b09f155b9a7f02c68d9d4b5e225
local      716884948ade1b25bd6df639b28dde95b0d021f8ff2efe5188d9fd102cf81
local      a71cb011736710cc59208ee795009020b697de3e360751b16957594453cad25b
local      ae1f0068d24ffcce6b409927c84b7cb06a7501e8112a04c09ff19b8930538684
local      b448649ef1d77659b40791573c47667a81a3ae7fdbb36a4e04caa2320ff237de
local      b68006410d76eb4975ead046060e637003c0a3b1cd2f100a40718fc7d84ad80b
local      c75c751590f6380e98adf8ba0766386b45e8bb46e05e3e9c4bad586eb27eac23
local      c0421e667eba420a9afec9d103b9080f5122da5db33f631695010e8d2f813114
local      ca9c45312a56d17050b1baebb863a8ca26aa2be5021a6aa4bb82582dc623ecf6
```

# Docker volume

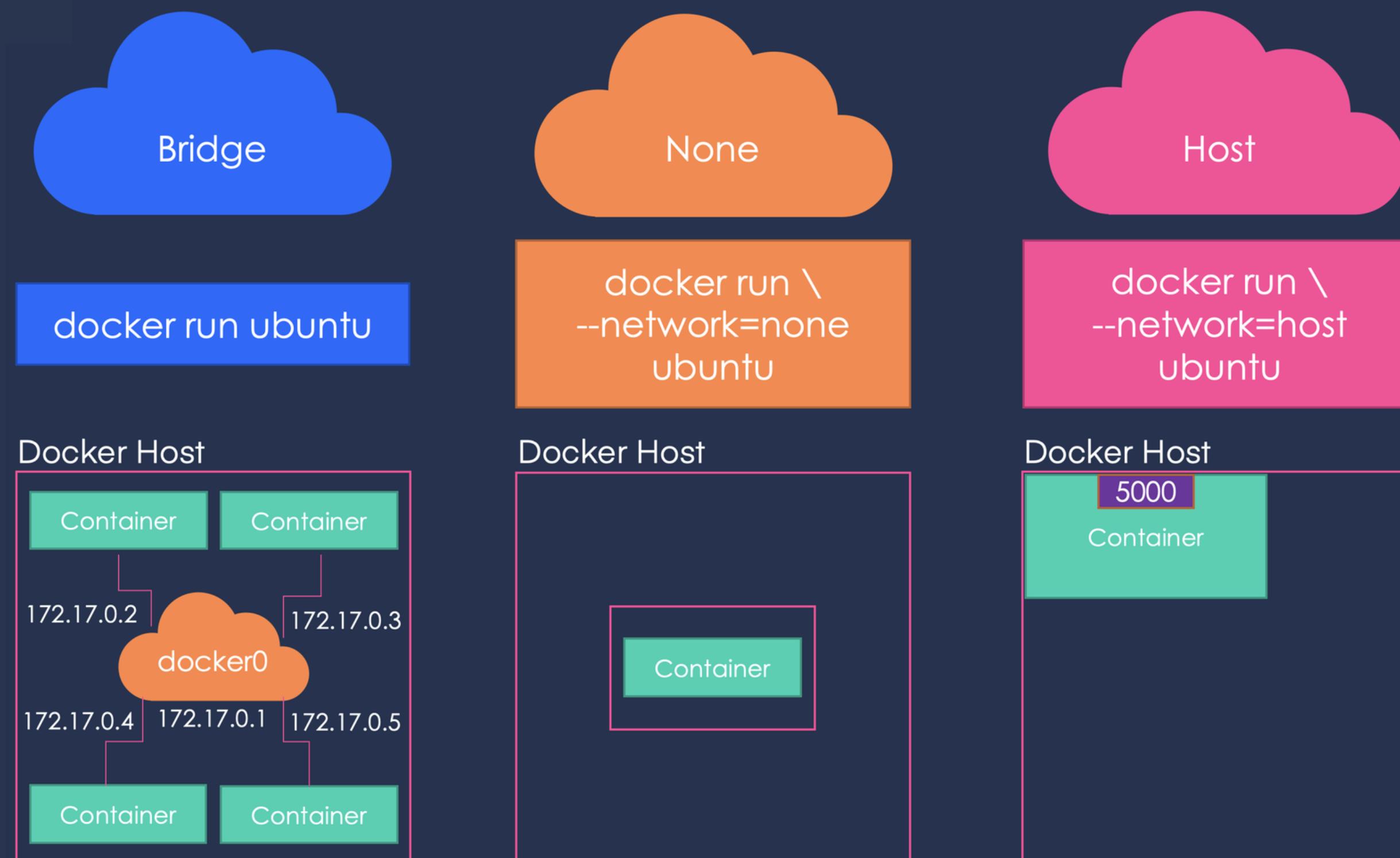
docker volume ls เพื่อแสดง volume ที่มีทั้งหมด

docker volume create เป็นคำสั่งเพื่อสร้าง volume ให้ container

docker volume inspect (Volume\_name) แสดงรายละเอียดของ volume

docker volume rm (Volume\_name) ลบ volume ที่กำหนด

# Docker network



# Docker network

docker network ls เพื่อแสดง network ที่มีอยู่

docker network create [OPTIONS] NETWORK เป็นคำสั่งเพื่อสร้าง network

docker network inspect เป็นคำสั่งเพื่อแสดงรายละเอียดของ network

docker network rm ลบ network ที่กำหนด

# Docker network

```
[root@wrk1 ~]# docker inspect registry
```

```
"Networks": {  
    "bridge": {  
        "IPAMConfig": null,  
        "Links": null,  
        "Aliases": null,  
        "NetworkID": "d631d7350ee41e4a9d9f45439b463be370937fc5fb1af434efd5a498a3b6609f",  
        "EndpointID": "77c9df65f0529275a2e403b827bbff84eaff920715af2d1bae0a93cd220e9d8c",  
        "Gateway": "172.17.0.1",  
        "IPAddress": "172.17.0.3",  
        "IPPrefixLen": 16,  
        "IPv6Gateway": "",  
        "GlobalIPv6Address": "",  
        "GlobalIPv6PrefixLen": 0,  
        "MacAddress": "02:42:ac:11:00:03",  
        "DriverOpts": null  
    }  
}
```

```
[root@wrk1 ~]# docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
d631d7350ee4	bridge	bridge	local

# Docker network

```
[root@wrk1 ~]# docker network create --driver=bridge --subnet=172.28.0.0/16 --ip-range=172.28.5.0/24 --gateway=172.28.5.254 mo-net
3e37218f417611a8077e015518f2629b46dafe35d7cae255a89e94adea39fde2
You have new mail in /var/spool/mail/root
[root@wrk1 ~]# docker network ls
NETWORK ID      NAME        DRIVER      SCOPE
d631d7350ee4    bridge      bridge      local
c0eedeb1bc1b    docker_gwbridge  bridge      local
4e097e2c85b1    host        host        local
moaxyenrgh0x    ingress     overlay     swarm
c8023ef05a6d    localnet     bridge      local
3e37218f4176    mo-net      bridge      local ←
c51763ce912f    net-mo       bridge      local
e7cf3ebbe859    node-multistage-docker_default  bridge      local
a94bc7d05307    none        null        local
acf5c9e454b9    sdp-la_default  bridge      local
e0bde6629b2b    sdp-la_elasticsearch-net  bridge      local
[root@wrk1 ~]# |
```

# Docker network

```
[root@wrk1 ~]# docker network create -d bridge NetworkA
0c3a41d1dee6a6de570b55f8fbb20f0cd66336d296efccf1ee1d55cfb2478e7d
You have new mail in /var/spool/mail/root
[root@wrk1 ~]# docker network create -d bridge NetworkB
7001fe84f7b6ed67d01f44f2eb4094092baeb9c94fe74c2f1ad94d95cc4b211d
[root@wrk1 ~]# docker network ls
NETWORK ID      NAME                DRIVER   SCOPE
0c3a41d1dee6    NetworkA           bridge   local
7001fe84f7b6    NetworkB           bridge   local
d631d7350ee4    bridge             bridge   local
c0eedeb1bc1b    docker_gwbridge   bridge   local
4e097e2c85b1    host               host    local
moaxyenrgh0x    ingress            overlay  swarm
c8023ef05a6d    localnet           bridge   local
3e37218f4176    mo-net             bridge   local
c51763ce912f    net-mo              bridge   local
e7cf3ebbe859    node-multistage-docker_default bridge   local
a94bc7d05307    none               null    local
acf5c9e454b9    sdp-la_default    bridge   local
e0bde6629b2b    sdp-la_elasticsearch-net bridge   local
[root@wrk1 ~]# |
```

# Docker network

```
[root@wrk1 ~]# docker run -dit --name demoNetwork1 --network NetworkA alpine /bin/sh  
3e33b5720dbe186ff73806f6f2e6c422622268544bd1b3242658099f8c1e41f
```

```
[root@wrk1 ~]# docker run -dit --name demoNetwork2 --network NetworkB alpine /bin/sh  
a6c43a854fe88e4021bf8ab3a5b83c3a25cc621384110234e49c65518e06211d
```

```
[root@wrk1 ~]# docker network inspect NetworkA
```

```
"Containers": {  
    "3e33b5720dbe186ff73806f6f2e6c422622268544bd1b3242658099f8c1e41f": {  
        "Name": "demoNetwork1", ←  
        "EndpointID": "22d849411e8b9fd13567613609ae713d2a15ac6d47d0930a63b6717aa3e1bbad",  
        "MacAddress": "02:42:ac:1d:00:02",  
        "IPv4Address": "172.29.0.2/16",  
        "IPv6Address": ""  
    }  
}
```

```
[root@wrk1 ~]# docker network inspect NetworkB
```

```
"Containers": {  
    "a6c43a854fe88e4021bf8ab3a5b83c3a25cc621384110234e49c65518e06211d": {  
        "Name": "demoNetwork2", ←  
        "EndpointID": "5bc3c9bec562c7acf8d744f368c4d9b1a641bcad984df0a478c757f7f291bde4",  
        "MacAddress": "02:42:ac:1e:00:02",  
        "IPv4Address": "172.30.0.2/16",  
        "IPv6Address": ""  
    }  
}
```

# Docker network

```
[root@wrk1 ~]# docker exec -it demoNetwork1 /bin/sh
/ # 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
172.29.0.2      3e33b5720dbe
/ # ping demoNetwork2
ping: bad address 'demoNetwork2'
/ # exit
[root@wrk1 ~]# docker exec -it demoNetwork2 /bin/sh
/ # 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
172.30.0.2      a6c43a854fe8
/ # ping demoNetwork1
ping: bad address 'demoNetwork1'
```

# Docker network

```
[root@wrk1 ~]# docker network create -d bridge NetworkZ  
bad30b9f706c5c818dd8023bd268ef9286f347a74a8abef9dfe4803b38559c20  
[root@wrk1 ~]# docker network connect NetworkZ demoNetwork1  
[root@wrk1 ~]# docker network connect NetworkZ demoNetwork2  
[root@wrk1 ~]# docker network inspect NetworkZ  
"Containers": {  
    "3e33b5720dbe186ff73806f6f2e6c4226222268544bd1b3242658099f8c1e41f": {  
        "Name": "demoNetwork1",  
        "EndpointID": "bdaeaf0ae3dd8884005f5f211ee8a4a61858e78c6c288e8b542e4bcf84c43577",  
        "MacAddress": "02:42:c0:a8:10:02",  
        "IPv4Address": "192.168.16.2/20",  
        "IPv6Address": ""  
    },  
    "a6c43a854fe88e4021bf8ab3a5b83c3a25cc621384110234e49c65518e06211d": {  
        "Name": "demoNetwork2",  
        "EndpointID": "20bd6a5e8e1bae5995a23c2b032cf7e1300b0c5d4ecec8573919654bd3f50ef4",  
        "MacAddress": "02:42:c0:a8:10:03",  
        "IPv4Address": "192.168.16.3/20",  
        "IPv6Address": ""  
    }  
}
```

# Docker network

```
[root@wrk1 ~]# docker exec -it demoNetwork1 /bin/sh
/ # ping demoNetwork2
PING demoNetwork2 (192.168.16.3): 56 data bytes
64 bytes from 192.168.16.3: seq=0 ttl=64 time=0.129 ms
64 bytes from 192.168.16.3: seq=1 ttl=64 time=0.165 ms
64 bytes from 192.168.16.3: seq=2 ttl=64 time=0.141 ms
64 bytes from 192.168.16.3: seq=3 ttl=64 time=0.137 ms
^C
--- demoNetwork2 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.129/0.143/0.165 ms
/ # exit
You have new mail in /var/spool/mail/root
[root@wrk1 ~]# docker exec -it demoNetwork2 /bin/sh
/ # ping demoNetwork1
PING demoNetwork1 (192.168.16.2): 56 data bytes
64 bytes from 192.168.16.2: seq=0 ttl=64 time=0.088 ms
64 bytes from 192.168.16.2: seq=1 ttl=64 time=0.175 ms
64 bytes from 192.168.16.2: seq=2 ttl=64 time=0.133 ms
64 bytes from 192.168.16.2: seq=3 ttl=64 time=0.129 ms
^C
--- demoNetwork1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.088/0.131/0.175 ms
/ # exit
```

# Docker network

```
[root@wrk1 ~]# docker run -dit --name demoNetwork3 --network NetworkC alpine /bin/sh  
f129438ca39853bf3623ddc0c5361ec4a3d61f16ab3acb1c65c33f73d0675ba7  
[root@wrk1 ~]# docker network connect NetworkB demoNetwork3  
[root@wrk1 ~]# docker exec -it demoNetwork3 /bin/sh  
/ # ping demoNetwork2  
PING demoNetwork2 (172.30.0.2): 56 data bytes  
64 bytes from 172.30.0.2: seq=0 ttl=64 time=0.228 ms  
64 bytes from 172.30.0.2: seq=1 ttl=64 time=0.110 ms  
64 bytes from 172.30.0.2: seq=2 ttl=64 time=0.130 ms  
^C  
--- demoNetwork2 ping statistics ---  
3 packets transmitted, 3 packets received, 0% packet loss  
round-trip min/avg/max = 0.110/0.156/0.228 ms  
/ # ping demoNetwork1  
ping: bad address 'demoNetwork1'
```

# Docker network

```
[root@wrk1 ~]# docker network disconnect NetworkB demoNetwork3
[root@wrk1 ~]# docker network connect NetworkZ demoNetwork3
[root@wrk1 ~]# docker exec -it demoNetwork3 /bin/sh
/ # ping demoNetwork1
PING demoNetwork1 (192.168.16.2): 56 data bytes
64 bytes from 192.168.16.2: seq=0 ttl=64 time=0.121 ms
64 bytes from 192.168.16.2: seq=1 ttl=64 time=0.176 ms
64 bytes from 192.168.16.2: seq=2 ttl=64 time=0.173 ms
^C
--- demoNetwork1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.121/0.156/0.176 ms
/ # ping demoNetwork2
PING demoNetwork2 (192.168.16.3): 56 data bytes
64 bytes from 192.168.16.3: seq=0 ttl=64 time=0.163 ms
64 bytes from 192.168.16.3: seq=1 ttl=64 time=0.101 ms
64 bytes from 192.168.16.3: seq=2 ttl=64 time=0.142 ms
^C
--- demoNetwork2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.101/0.135/0.163 ms
```

# Docker multistage build

ກາສ build ແບບປົກຕົວ

```
FROM golang:1.11.1

RUN mkdir /app

ADD . /app

WORKDIR /app

#Create executable for Go code

RUN CGO_ENABLED=0 GOOS=linux go build -o main ./...

CMD ["./main"]
```

# Docker multistage build

main.go

```
package main

import "fmt"

func main() {
    fmt.Println("hello world")
}
~
```

# Docker multistage build

ນາສ build ໂບບປກຕີ

```
[root@wrk1 demomulti]# docker build . -f SingleStageDockerfile -t demosinglestagebuild
Sending build context to Docker daemon 5.632kB
Step 1/6 : FROM golang:1.11.1
--> 45e48f60e268
Step 2/6 : RUN mkdir /app
--> Using cache
--> bf0aab1d6053
Step 3/6 : ADD . /app
--> Using cache
--> f0a7aa6c4901
Step 4/6 : WORKDIR /app
--> Using cache
--> dfe168775a80
Step 5/6 : RUN CGO_ENABLED=0 GOOS=linux go build -o main ./...
--> Using cache
--> cdf3d90cf5e5
Step 6/6 : CMD ["./main"]
--> Using cache
--> f0b8963bd73b
Successfully built f0b8963bd73b
Successfully tagged demosinglestagebuild:latest
[root@wrk1 demomulti]# |
```

# Docker multistage build

ນາສ build ແບບປກຕີ

```
[root@wrk1 demomulti]# docker run demosinglestagebuild
hello world
[root@wrk1 demomulti]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
demosinglestagebuild	latest	f0b8963bd73b	About a minute ago	778MB

# Docker multistage build

## multi-stage build

```
FROM golang:1.11.1 AS builder
RUN mkdir /app
ADD . /app
WORKDIR /app
# Create executable for Go code
RUN CGO_ENABLED=0 GOOS=linux go build -o main ./...
# Second stage build which will execute the binary
FROM alpine:latest AS production
COPY --from=builder /app .
CMD [".main"]
~
```

# Docker multistage build

## multi-stage build

```
[root@wrk1 demomulti]# docker build . -f multistageDockerfile -t demomultistagebuild
Sending build context to Docker daemon 5.632kB
Step 1/8 : FROM golang:1.11.1 AS builder
--> 45e48f60e268
Step 2/8 : RUN mkdir /app
--> Using cache
--> bf0aab1d6053
Step 3/8 : ADD . /app
--> Using cache
--> f0a7aa6c4901
Step 4/8 : WORKDIR /app
--> Using cache
--> dfe168775a80
Step 5/8 : RUN CGO_ENABLED=0 GOOS=linux go build -o main ./...
--> Using cache
--> cdf3d90cf5e5
Step 6/8 : FROM alpine:latest AS production
--> e66264b98777
Step 7/8 : COPY --from=builder /app .
--> def6eb9428db
Step 8/8 : CMD ["./main"]
--> Running in 1bbb6138f586
Removing intermediate container 1bbb6138f586
--> 482b3fcc67b3
Successfully built 482b3fcc67b3
Successfully tagged demomultistagebuild:latest
```

# Docker multistage build

multi-stage build ນັ້ນຈ່າຍລົດບາດຂອງ image ສຸດກໍາຍໄດ້ອຍ່າງມໍາຄາລ

```
[root@wrk1 demomulti]# docker run demomultistagebuild  
hello world
```

```
You have new mail in /var/spool/mail/root
```

```
[root@wrk1 demomulti]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
demomultistagebuild	latest	482b3fcc67b3	About a minute ago	7.43MB
demosinglestagebuild	latest	f0b8963bd73b	5 minutes ago	778MB

# .dockerignore

ไฟล์ .dockerignore จะเป็นไฟล์ที่เอาไว้ระบุว่า สิ่งไหนบ้างที่จะไม่ต้องถูกรวมเข้าไปใน context เวลาที่ถูก build ซึ่งจะประโยชน์สองอย่างคือ

1. ถ้าไฟล์ที่ไม่เกี่ยวกับ context หรือไฟล์ที่มีขนาดใหญ่มากๆ ก็จะไม่ต้องเสียเวลาเข้าไปอ่านใน context ซึ่งจะทำให้สร้าง image ได้เร็วขึ้น
2. ทางด้านความปลอดภัย สามารถเลือกได้ว่าไฟล์ที่ค่อนข้าง sensitive (เช่น .env, .git, secret) จะไม่ถูกรวมเข้าไปใน context ตอน build ด้วย

```
node_modules  
build  
.dockerignore  
Dockerfile  
Dockerfile.prod
```

# Docker compose คืออะไร

Docker compose คือ script คำสั่ง ที่เอาไว้สร้าง container หลายๆ อันขึ้นมาพร้อมกัน โดยการสร้างไฟล์ที่ชื่อว่า docker-compose.yml เพื่อใช้งานการควบคุมการสร้าง container และกำหนด option ต่างๆ ที่ใช้งานการรัน container แต่ละตัว

# Docker compose

```
version: '3'
services:
  db:
    image: mysql:5.7
    volumes:
      - db_database:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
    healthcheck:
      test: mysql -uroot -pwordpress -e 'show databases;'
      interval: 10s
      timeout: 10s
      retries: 3
      start_period: 20s
    networks:
      - wordpress-db
  wordpress:
    depends_on:
      db:
        condition: service_healthy
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
    networks:
      - wordpress-db
  volumes:
    db_database:
  networks:
    wordpress-db:
      driver: bridge
```

# HealthCheck

เป็นการตรวจเช็คสถานะของ container ว่าสามารถ  
ทำงานได้ปกติอย่างที่ต้องการได้หรือไม่

test	เป็นการระบุคำสั่งที่จะดำเนินการและเป็นการ healthcheck ของ container
interval	เป็นการระบุเวลาระหว่างการ health check ใบແຕ່ລະครັ້ງ
timeout	เป็นการระบุจำนวนวินาทีที่รอให้คำสั่ง health check return exit code ก่อนจะประกาศว่า container as unhealthy
retries	เป็นการระบุจำนวนความล้มเหลวจากการ health check ก่อนประกาศว่า container as unhealthy
start-period	เป็นจำนวนวินาทีที่ container จะเป็นต้องเริ่มต้นก่อนการ health check

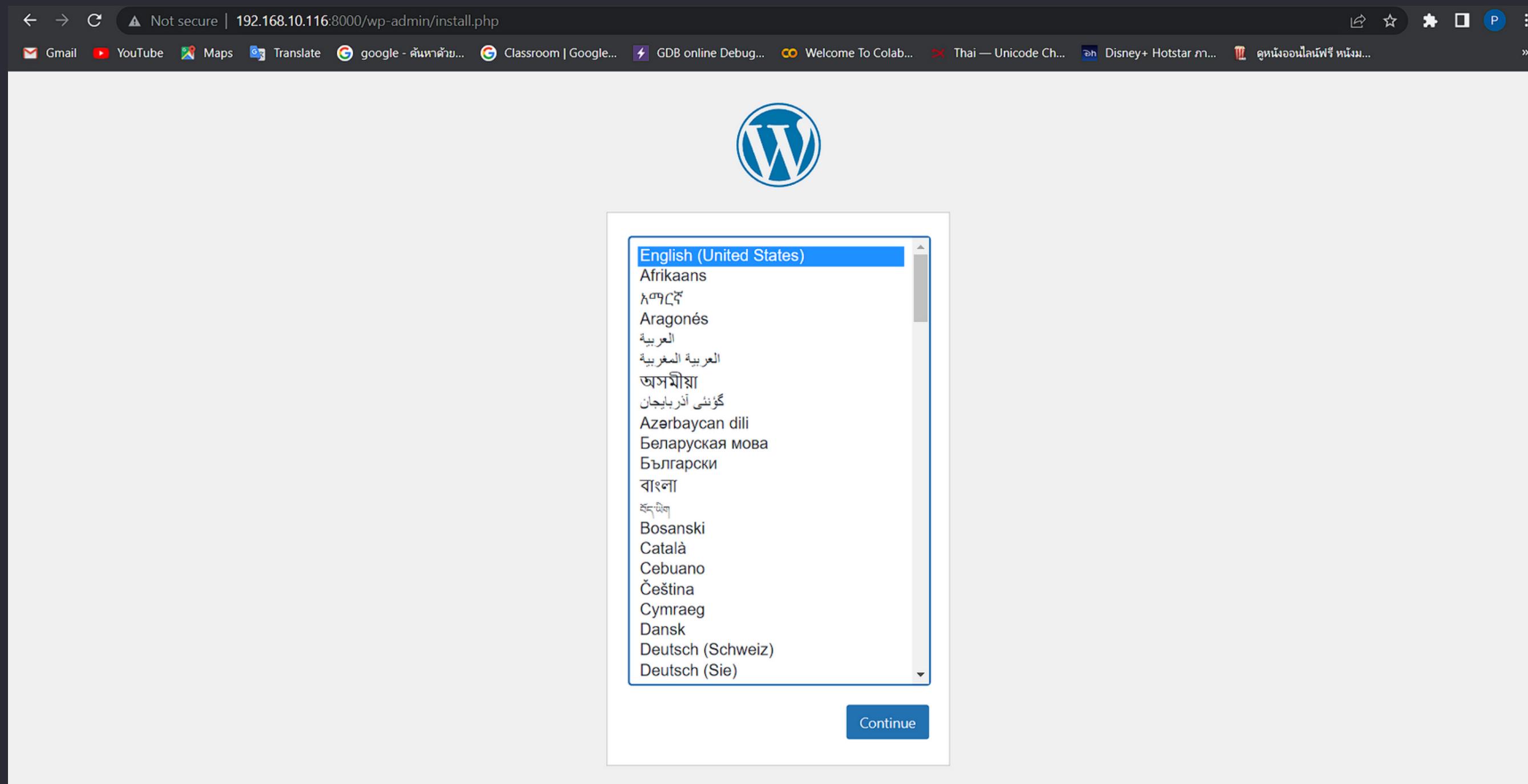
# Docker compose

เราสามารถ run docker-compose.yml ได้ด้วยคำสั่ง  
docker compose up

```
[root@localhost wordpress-mysql-compose]# docker compose up
[+] Running 3/3
  # Network wordpress-mysql-compose_wordpress-db    Created      0.3s
  # Container wordpress-mysql-compose-db-1          Created      0.1s
  # Container wordpress-mysql-compose-wordpress-1   Created      0.2s
Attaching to wordpress-mysql-compose-db-1, wordpress-mysql-compose-wordpress-1
wordpress-mysql-compose-db-1           | 2022-07-03 10:15:29+00:00 [Note] [Entrypoint]
: Entrypoint script for MySQL Server 5.7.38-1debian10 started.
wordpress-mysql-compose-db-1           | 2022-07-03 10:15:29+00:00 [Note] [Entrypoint]
: Switching to dedicated user 'mysql'
wordpress-mysql-compose-db-1           | 2022-07-03 10:15:29+00:00 [Note] [Entrypoint]
: Entrypoint script for MySQL Server 5.7.38-1debian10 started.
wordpress-mysql-compose-db-1           | 2022-07-03T10:15:29.519799Z 0 [Warning] TIMES
TAMP with implicit DEFAULT value is deprecated. Please use --explicit_defaults_for_t
imestamp server option (see documentation for more details).wordpress-mysql-compose-
db-1           | 2022-07-03T10:15:29.521039Z 0 [Note] mysqld (mysqld 5.7.38) starting
as process 1 ...
wordpress-mysql-compose-db-1           | 2022-07-03T10:15:29.524255Z 0 [Note] InnoDB:
PUNCH HOLE support available
wordpress-mysql-compose-db-1           | 2022-07-03T10:15:29.524283Z 0 [Note] InnoDB:
Mutexes and rw_locks use GCC atomic builtins
wordpress-mysql-compose-db-1           | 2022-07-03T10:15:29.524294Z 0 [Note] InnoDB:
```

# Docker compose

จากบันได browser ip:<port กี่กำหนดไว้ใน docker compose>



# Docker compose

เราสามารถ stop docker compose ได้ด้วยคำสั่ง `ctrl + c`  
และลบ container ที่สร้างจาก `docker-compose.yml` ด้วยคำสั่ง  
`docker compose down`

```
[root@localhost wordpress-mysql-compose]# docker compose down
[+] Running 3/3
  # Container wordpress-mysql-compose-wordpress-1    Removed      0.0s
  # Container wordpress-mysql-compose-db-1           Removed      0.0s
  # Network wordpress-mysql-compose_wordpress-db   Removed      0.3s
[root@localhost wordpress-mysql-compose]# |
```

# Docker compose

docker compose up เป็นคำสั่งเพื่อ create และ run docker-compose file

docker compose down เป็นคำสั่งเพื่อ stop และ remove container

docker compose images เป็นคำสั่งเพื่อดูรายชื่อของ image ที่ docker compose

docker compose run เป็นคำสั่งเพื่อ run 1 service ที่ docker compose file

# ຕົວອຢ່າງ

## ນໍາ sorce code ຂອງຝຶ່ງ Frontend ມາ Run ໃນ Docker

The screenshot shows a web application interface for managing licenses. At the top, there are four summary boxes:

- Type SLG: 5 License (just minute)
- Type LA: 3 License (just minute)
- License Request: 7 License (on 15 March, 2022)
- Summary: Active 3 License, Nearly Expire 1 License, Expire 2 License (on 15 March, 2019)

Below these is a section titled "License List" containing a table with the following columns: Customer ID, Certificate ID, Type, Storage, and Expired. The table lists 16 rows of license information.

Customer ID	Certificate ID	Type	Storage	Expired
00000000000000000000000000000000	LG-LOGGERMA200156532	LA	1073741824	2022-12-13T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA200156536	LA	1073741824	2022-12-13T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA200156567	LA	1073741824	2022-12-13T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA200156753	LA	1073741824	2022-12-13T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA200156887	LA	1073741824	2022-12-13T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA2001565998	SLG	1073741824	2022-12-13T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA2001566000	SLG	1073741824	2022-12-15T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA2001566020	SLG	1073741824	2022-12-15T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA2001566065	SLG	1073741824	2022-12-15T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA2001566775	SLG	1073741824	2022-12-15T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA2001566785	SLG	1073741824	2022-12-15T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA2001566768	SLG	1073741824	2022-12-15T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA2001566769	SLG	1073741824	2022-12-15T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA2001566777	SLG	1073741824	2023-12-15T11:09:21Z
00000000000000000000000000000000	LG-LOGGERMA2001566778	SLG	1073741824	2023-12-15T11:09:21Z

# REFERENCE:

[HTTPS://DOCS.DOCKER.COM/GET-STARTED/OVERVIEW](https://docs.docker.com/get-started/overview)

[HTTPS://BLOG.SKOOLDIO.COM/WHAT-IS-DOCKER/](https://blog.skooldio.com/what-is-docker/)

[HTTPS://GITHUB.COM/DOCKER](https://github.com/docker)

# THANK YOU

An illustration of a large blue whale swimming towards the left. On its back, a small city with several buildings and a green roof is perched. The whale has a white patch on its lower jaw and a dark eye. It is set against a teal background. In the top corners, there are stylized white bubbles and a school of four small fish with black stripes. The bottom corners feature white coral reef illustrations.