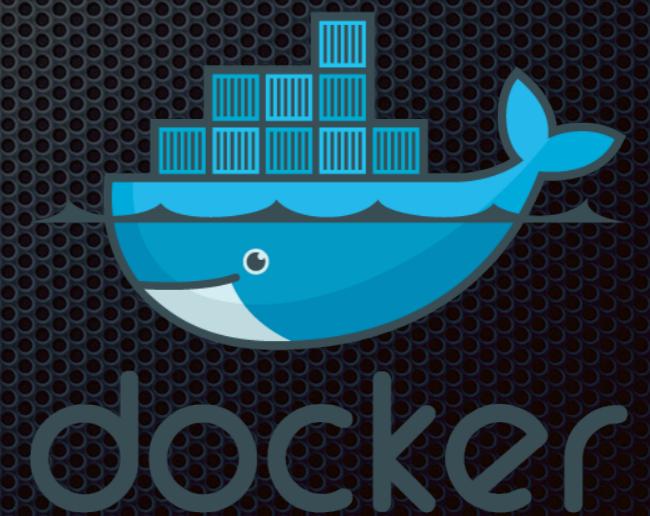


# Introduction to Docker

Krerk Piromsopa, Ph. D.

Department of Computer Engineering  
Chulalongkorn University



*Some slides from [docker.io](https://www.docker.io)*

# Goal

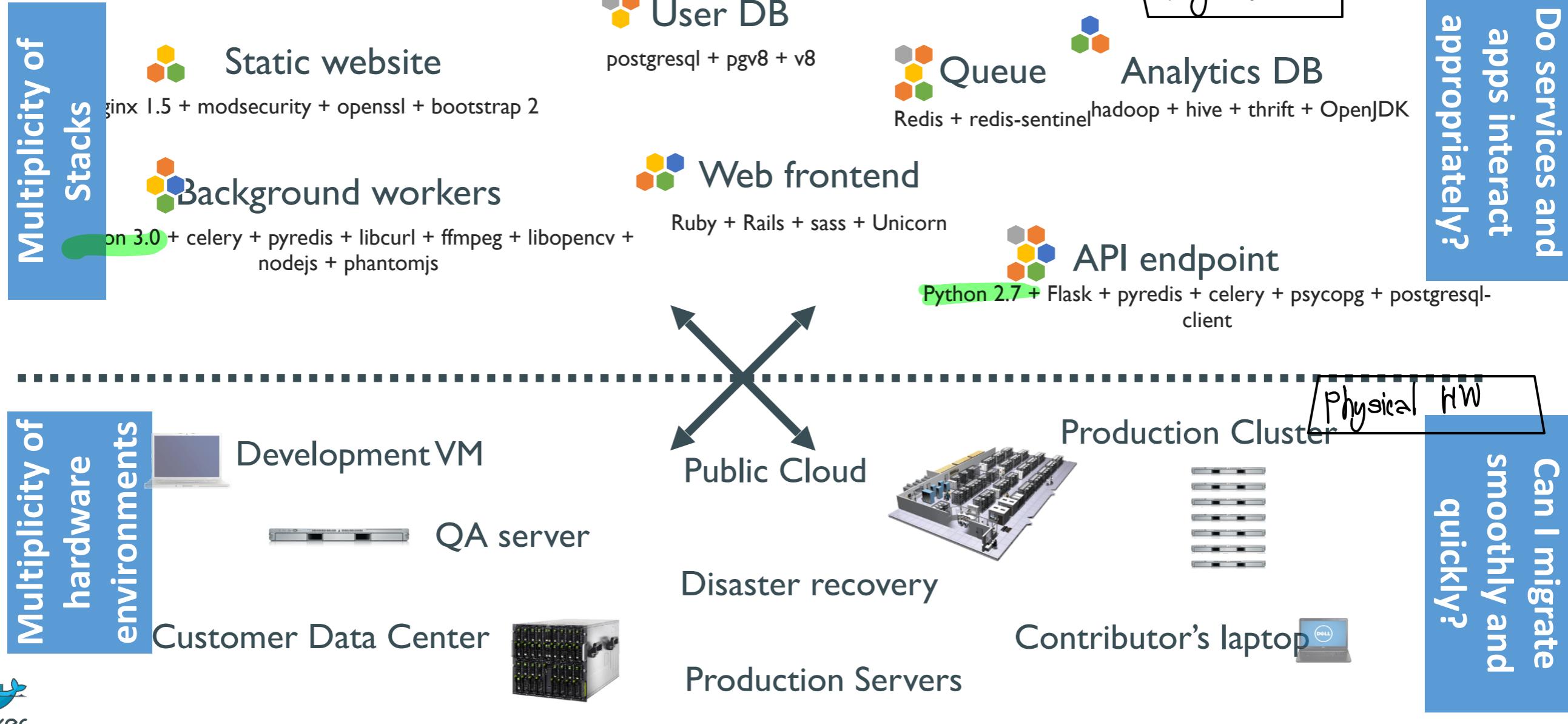
- What/Why/How?
- Docker BASIC
  - Image, Registry
- Docker Compose

# What/Why Docker?

# The Challenge

ຫຍນຂອງຮູ້ອຳນວຍ  
sizing database Segment ສະໜັບສະໜັດ  
ຈາກ 3 ຕາມປິ່ນ 10,000 ດົນ ໂຮງຮາກເປົ້າຍາ  
ຮູ້ຕົກລົງເປັນລາຍງົກຈົກໃຈ ດັ່ງລາຍລະອຽດ?

ເຫັນ SW ຊື່ dependencies ຕ່າງກົນ ເພີ້ນ Python 2.7 ຂັ້ນ 3



(ຕາງໆ docker នີ້ຍຸດຕິນໍ້າ ອານພໍ່ໄວ້) install ກ່າວນີ້, ຈົນຄ່ອງອະນຸມາວະ Peak

# The Matrix From Hell

	<b>Static website</b>	?	?	?	?	?	?	?
	<b>Web frontend</b>	?	?	?	?	?	?	?
	<b>Background workers</b>	?	?	?	?	?	?	?
	<b>User DB</b>	?	?	?	?	?	?	?
	<b>Analytics DB</b>	?	?	?	?	?	?	?
	<b>Queue</b>	?	?	?	?	?	?	?
	<b>Development VM</b>	<b>QA Server</b>	<b>Single Prod Server</b>	<b>Onsite Cluster</b>	<b>Public Cloud</b>	<b>Contributor's laptop</b>	<b>Customer Servers</b>	



# Cargo Transport Pre-1960

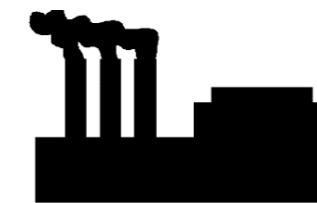
ឧបន៍ដែលត្រូវបានការសម្រេចក្នុងពាណិជ្ជកម្ម

Multiplicity of Goods



Do I worry about how goods interact (e.g. coffee beans next to spices)

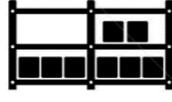
Multiplicity of methods for transporting/storing



Can I transport quickly and smoothly (e.g. from boat to train to truck)

# Also a matrix from hell

---

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

# Solution: Intermodal Shipping Container



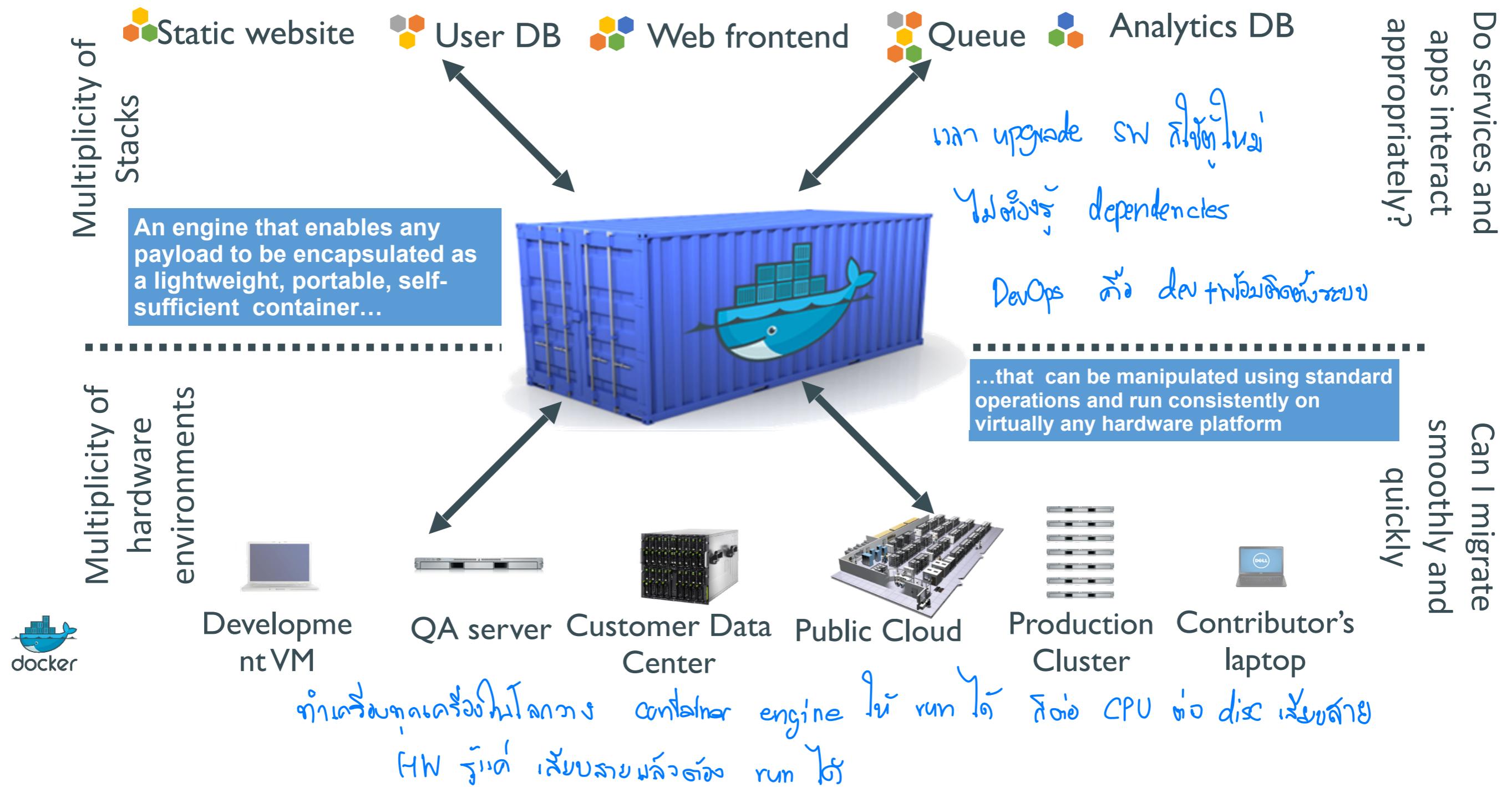
ក្នុងការអនុវត្តន៍ ទំនាក់ទំនង និង ផលិតផល នឹងការប្រើប្រាស់  
ដោយសារព័ត៌មាន និង ការគ្រប់គ្រង នៃ ការអនុវត្តន៍ និង ផលិតផល  
នៅក្នុងការអនុវត្តន៍ ទំនាក់ទំនង និង ផលិតផល នឹងការប្រើប្រាស់  
ដោយសារព័ត៌មាន និង ការគ្រប់គ្រង នៃ ការអនុវត្តន៍ និង ផលិតផល

ការអនុវត្តន៍ និង ការគ្រប់គ្រង នៃ ការអនុវត្តន៍ និង ផលិតផល

Do I worry about how goods interact (e.g. coffee beans next to spices)?

Can I transport quickly and smoothly (e.g. from boat to train to truck)?

# Docker is a shipping container system for code



# Docker eliminates the matrix from Hell

	<b>Static website</b>						
	<b>Web frontend</b>						
	<b>Background workers</b>						
	<b>User DB</b>						
	<b>Analytics DB</b>						
	<b>Queue</b>						
	<b>Development VM</b>	<b>QA Server</b>	<b>Single Prod Server</b>	<b>Onsite Cluster</b>	<b>Public Cloud</b>	<b>Contributor's laptop</b>	<b>Customer Servers</b>



HN အေဂါ ရဲမဲ ၏ မျှနှုန်းမြတ်စွာ။



JAVA → Write once run anywhere

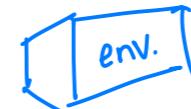
ឧសភាស្ត្រ Write once debug anywhere

# Why Developers Care

## • Build once... (finally) run anywhere\*

- A clean, safe, hygienic and portable runtime environment for your app.
- No worries about missing dependencies, packages and other pain points during subsequent deployments. *→ មិនមែនការងារ version ឡើង*
- Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying
- Automate testing, integration, packaging... anything you can script
- Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.
- Cheap, zero-penalty containers to deploy services? A VM without the overhead of a VM? Instant replay and reset of image snapshots? That's the power of Docker

motto



container គ្រប់រាយក្នុងវា run ឬន windows, mac os, linux

docker តាមអាជីវកម្ម

\* With the 0.7 release, we support any x86 server running a modern Linux kernel (3.2+ generally. 2.6.32+ for RHEL 6.5+, Fedora, & related)



# Why Devops Cares? *run dev infra op*

---

- **Configure once...run anything**

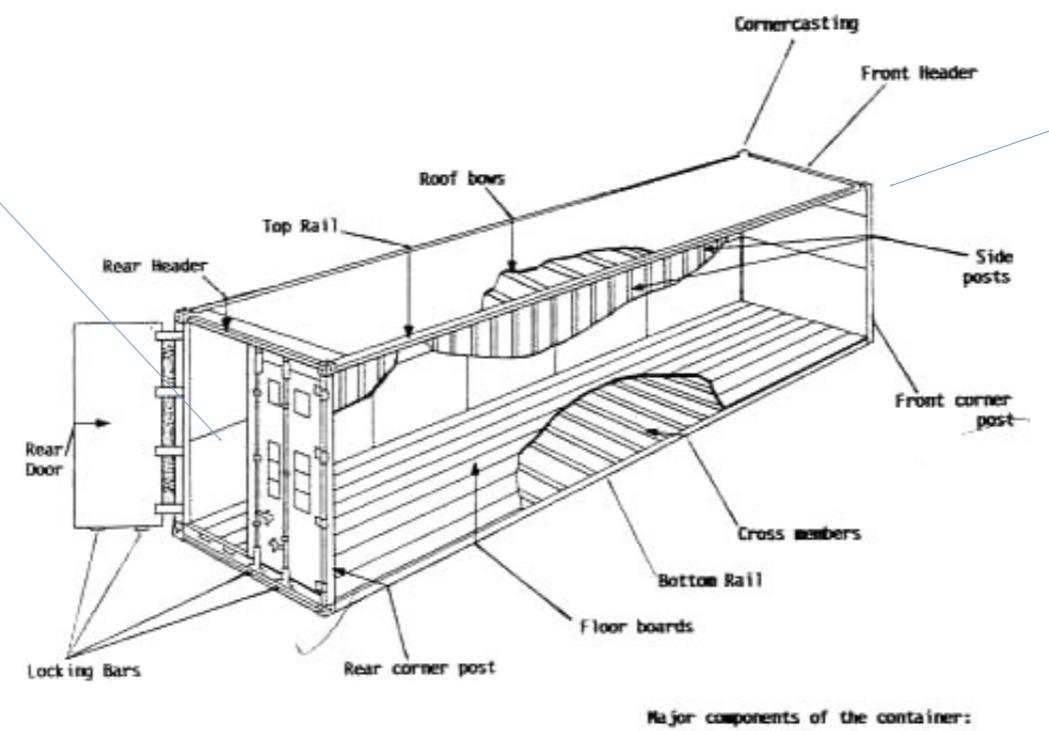
- Make the entire lifecycle more efficient, consistent, and repeatable
- Increase the quality of code produced by developers.
- Eliminate inconsistencies between development, test, production, and customer environments
- Support segregation of duties
- Significantly improves the speed and reliability of continuous deployment and continuous integration systems
- Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VMs



# Why it works—separation of concerns

In dev & abstract របៀបរបស់linux

- Dan the Developer
  - Worries about what's "inside" the container *សម្រាប់ខ្លួន*
    - His code
    - His Libraries
    - His Package Manager
    - His Apps
    - His Data
  - All Linux servers look the same



- Oscar the Ops Guy
  - Worries about what's "outside" the container *និងរយៈដទៃ*
    - Logging
    - Remote access
    - Monitoring
    - Network config
  - All containers start, stop, copy, attach, migrate, etc. the same way

# More technical explanation

---

## WHY

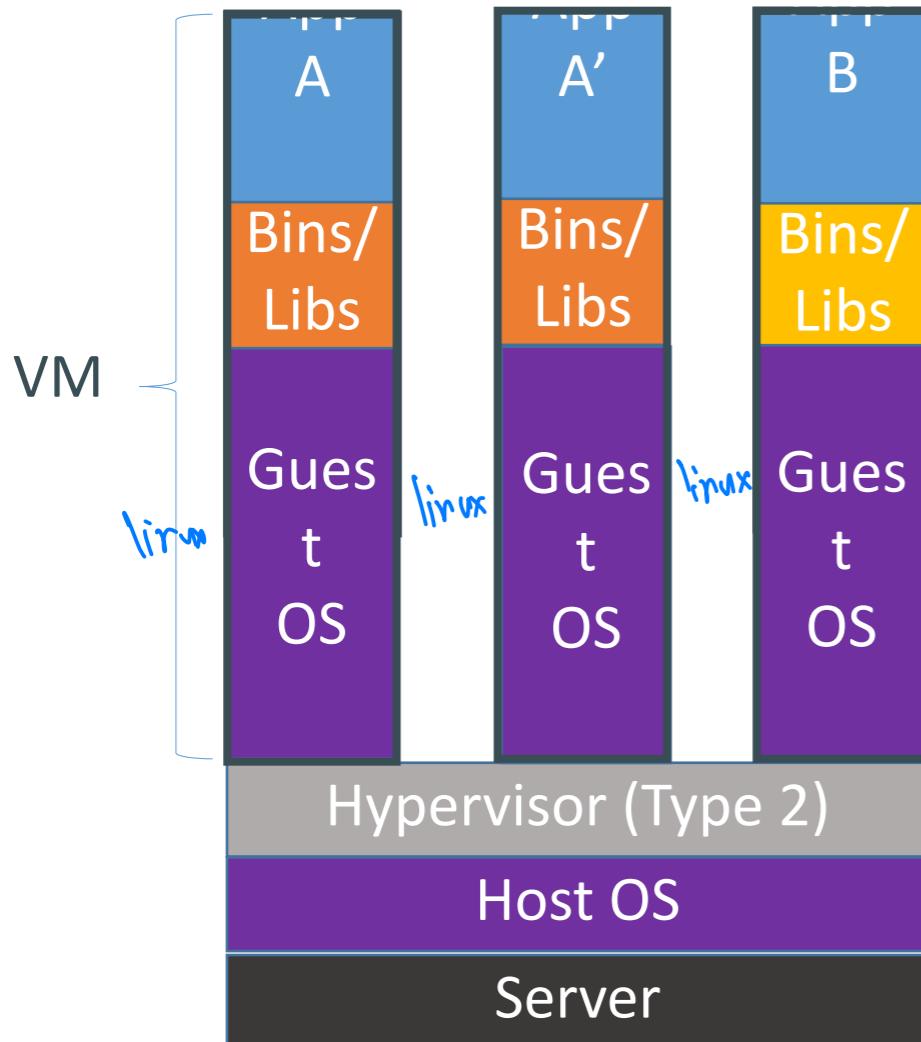
- Run everywhere *�່ານແຈ້ງຈຸດ linux > 2.6.32*
  - Regardless of kernel version (2.6.32+)
  - Regardless of host distro
  - Physical or virtual, cloud or not
  - Container and host architecture must match\*
- Run anything
  - If it can run on the host, it can run in the container
  - i.e. if it can run on a Linux kernel, it can run

## WHAT

- High Level—It's a lightweight VM
  - Own process space
  - Own network interface
  - Can run stuff as root
  - Can have its own /sbin/init (different from host)
  - <<machine container>>
- Low Level—It's chroot on steroids
  - Can also *not* have its own /sbin/init
  - Container=isolated processes
  - Share kernel with host
  - No device emulation (neither HVM nor PV) from host
  - <<application container>>



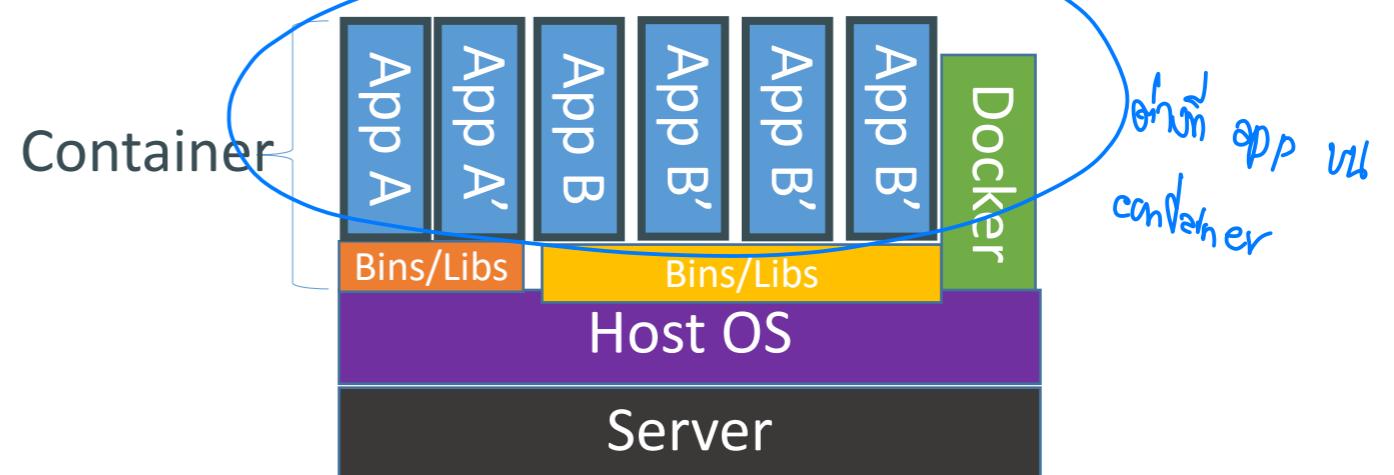
# Containers vs. VMs



Install OS ຖុកចិត្តបែង ផ្តល់ disc ខាងក្រោម

Containers are isolated,  
but share OS and, where  
appropriate, bins/libraries

...result is significantly faster  
deployment, much less overhead,  
easier migration, faster restart



Container នឹង Host ម៉ោង នូវ disc ដែលគាំទ្រ

ពន្លាខ្លួច Host ទៅ គ្រប់គ្រង



# Why are Docker containers lightweight?

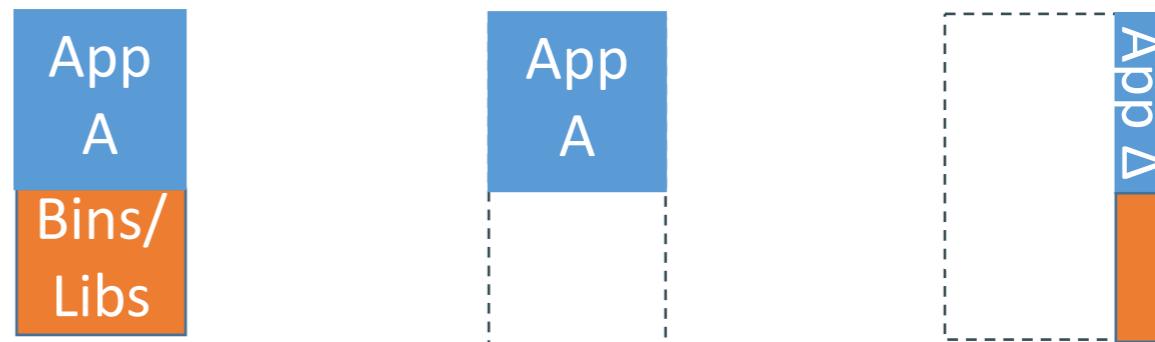
VMs



Every app, every copy of an app, and every slight modification of the app requires a new virtual server

ต้อง copy ทุกอย่าง ไม่ใช่แค่ไฟล์

Containers



Original App  
(No OS to take up space, resources, or require restart)

layer ล้ำเก็บ copy เดียว

Copy of App  
No OS. Can Share bins/libs

Modified App

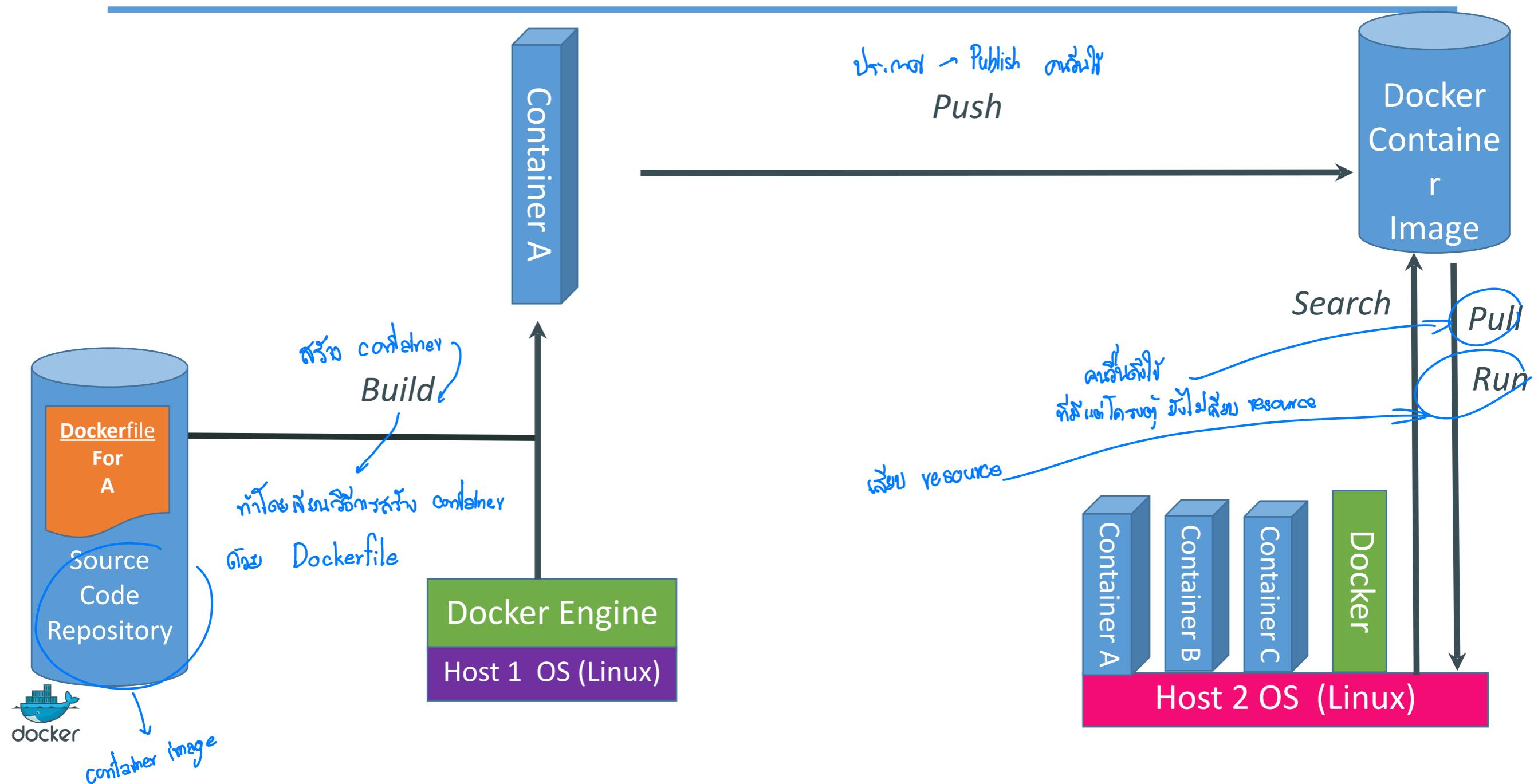
Copy on write capabilities allow us to only save the diffs  
Between container A and container A'

layer file system เก็บส่วนต่างๆ file เอง

start vm จันทร์ minute container หล่อจันทร์

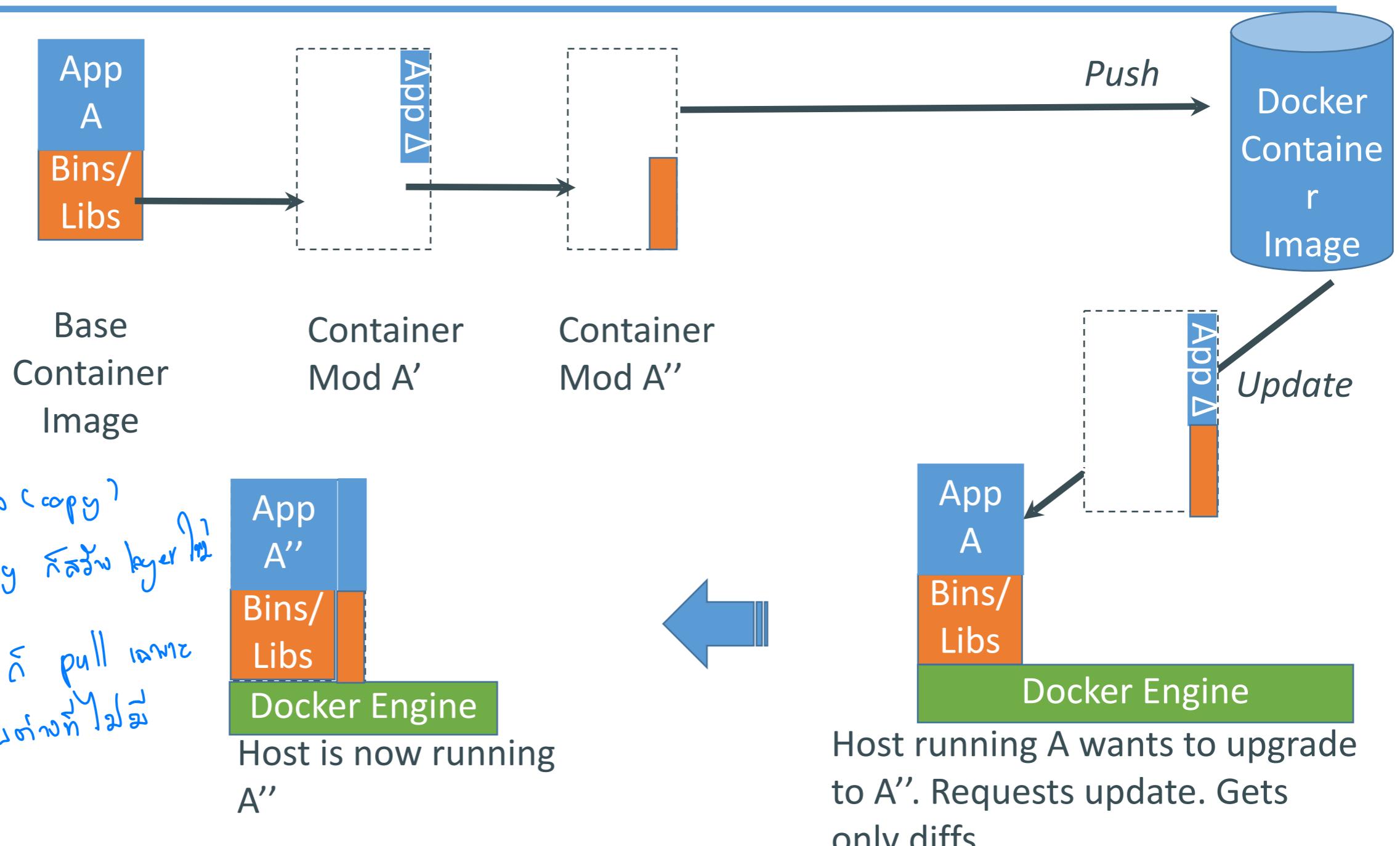


# What are the basics of the Docker system?



# Changes and Updates

ເກືອນ pull git + fetch



# Docker Basic

Container ជីន stateless (ទៅ ឬ ត្រូវ ស្វែងរក្សាយក្រឹង) ឬ ការរៀបចំ, ផលិត និង storage

ឯកសារ មិន មាន ការពារ ឬ ការរៀបចំ

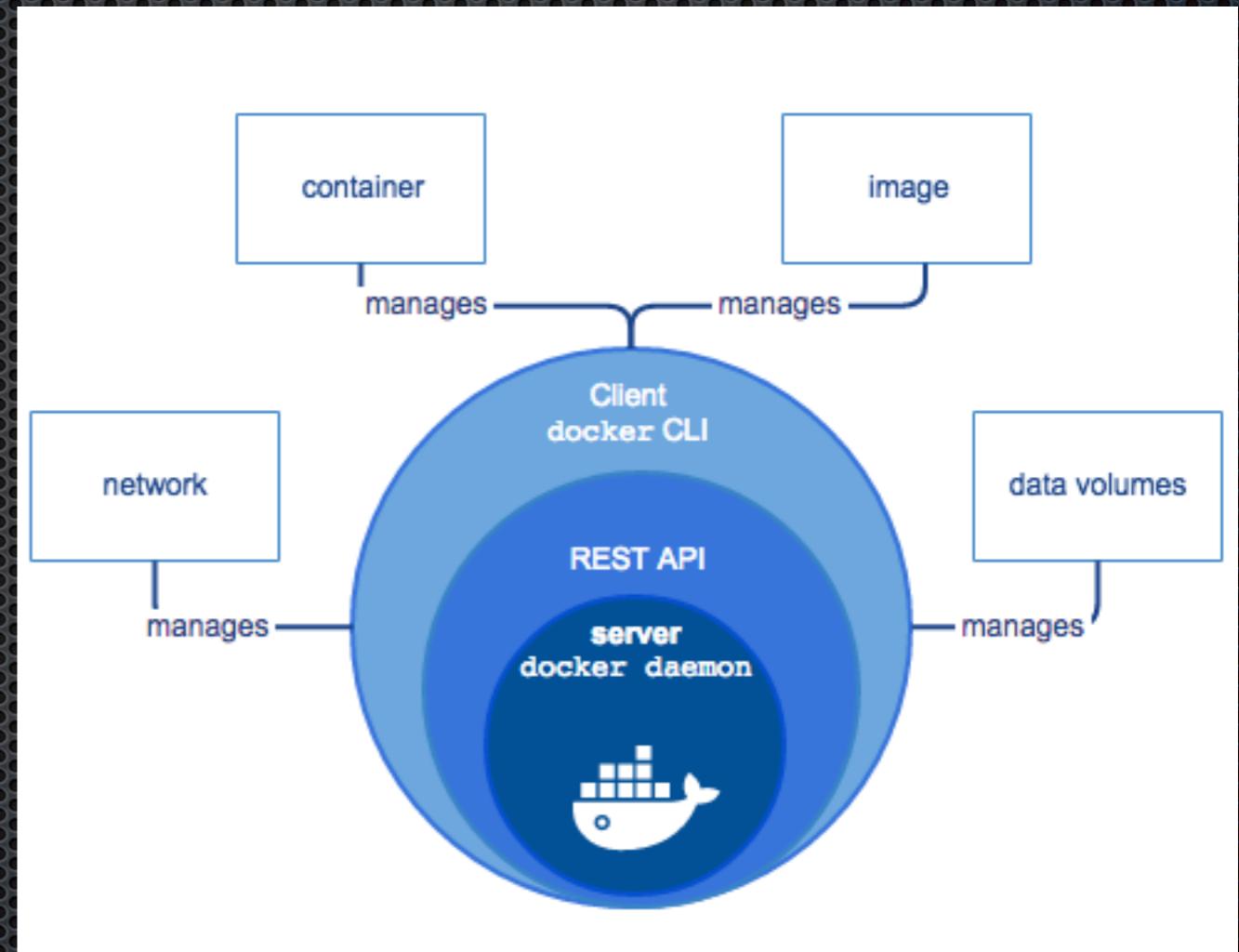
- Docker abstracts:

▪ Storage សារណ៍រួម ឬ ឯកសារ

- Network

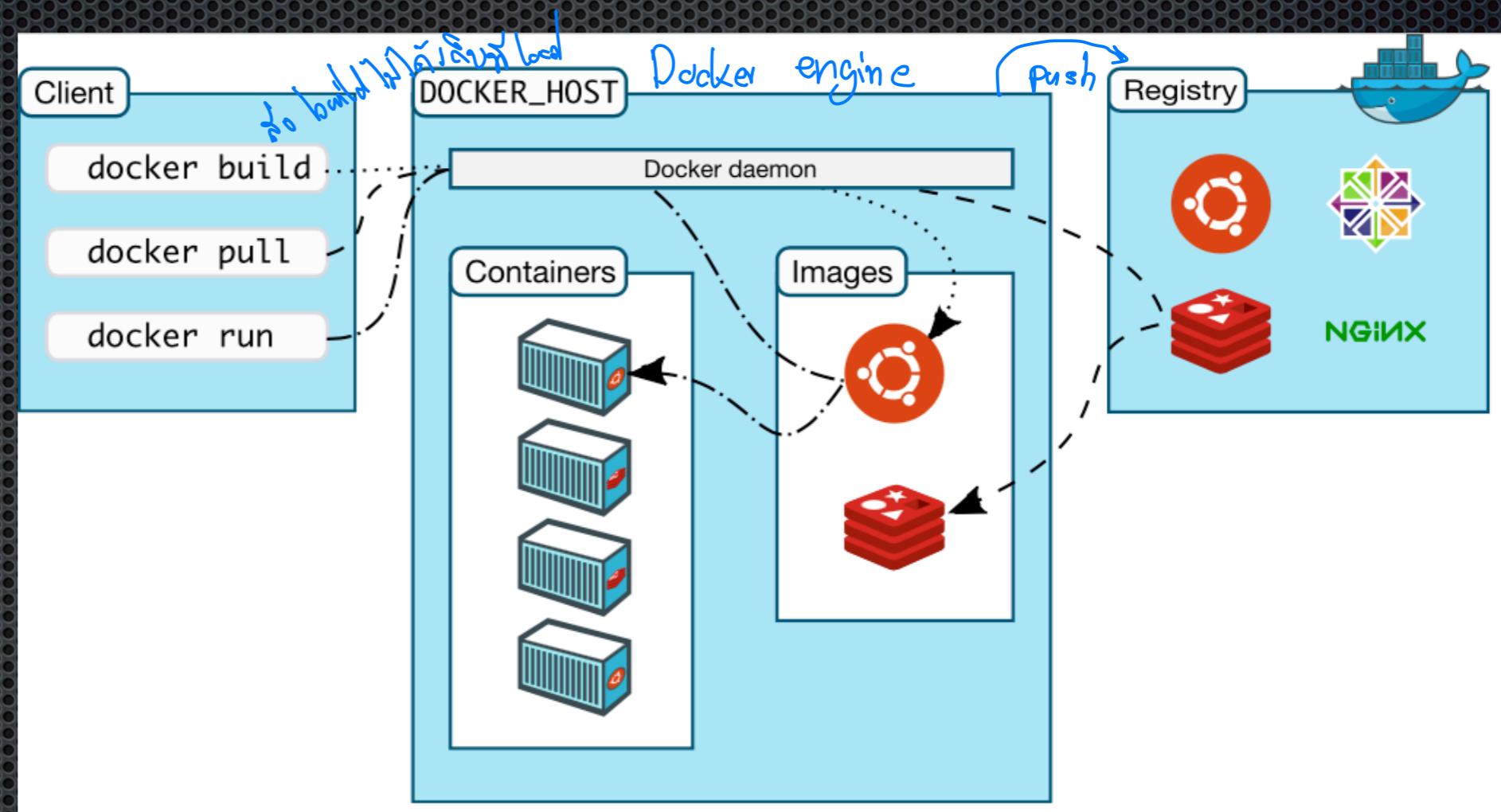
▪ CPU, RAM ឬ ការគ្រប់គ្រង

- Management  
(start/stop)



# Docker Architecture

- ❖ daemon
- ❖ client
- ❖ Inside
  - ❖ images
  - ❖ containers
  - ❖ registry
- ❖ services



# Docker Compose

ໄຟຣີເຈົ້າໃຫ້ມານິຍາກັງມາ ເພື່ອ db, ອີ່ app server ມີ docker compose ສົ່ງ run ຖ້າຕົວໆ  
service composition ນະຍົງ component ພາບກັນທຳມາດົາ

- Compose is a tool for defining and running multi-container Docker applications.
  - Use a compose file to configure services.
  - Deploy (create and start) with a single command

# Swarm Mode

107 > 1 docker engine ມາຈີນ ກົນ  
ສໍາມາດຮັບຮັດ

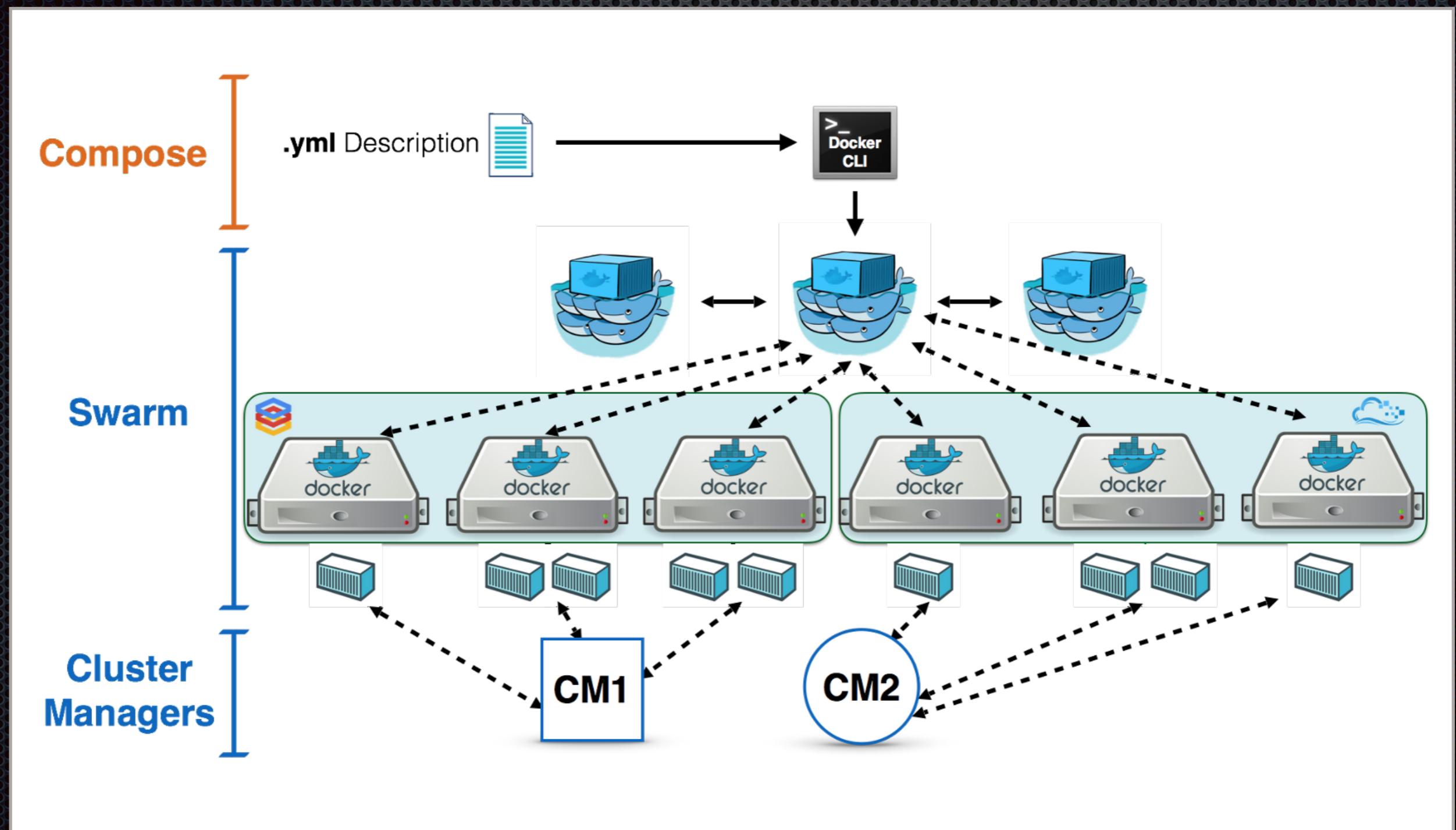
ຮັບຮັດ docker ວິທີໂອຟຸນຸງ

- Cluster management or orchestration features for Docker Engine.
- Managers and Workers
- Services
- Definitions of tasks to execute on managers and workers
- Ingress load balancing ( we will revisit this later)
- PublishedPort

Deprecated  
(In favor of Kubernetes)

ຕາມນີ້ຈະ Kube ສິນ້ ສວມ

# Anatomy of Swarm



# Dockerfile

# Comment  
INSTRUCTION arguments

```
# FROM [--platform=<platform>] <image> [AS <name>]
FROM python:3.9 FROM=container កែវនេះជា ឲ្យ base តារាង នៃ python base Linux
# LABEL
LABEL MAINTAINER="Kerk Piromspa" LABEL = បានយកចូល
# RUN [shell commands] — /bin/sh -c
RUN echo world      RUN ព័ត៌មាននៅលើបណ្តុះបណ្តាល
RUN ["/bin/bash","-c","echo hello"]    linux ក្រុង
# CMD ["executable","param1","param2"] - default commands
# CMD command param1 param2
CMD ["/usr/local/bin/python3", "main.py"] cmd ដែលធ្វើឡើ (start) នៅក្នុងការគាំទ្រ
# EXPOSE network ports ឲ្យ PORT នេះ
EXPOSE 80/tcp
EXPOSE 80/udp
```

# Dockerfile (ctd.)

```
# ENV MY_VAR my-value          នៃ docker run -e myname  
ENV MYNAME "Kerk"             ក្រឡាបាយនៃក្នុងការបង្កើត និងការសម្រាប់ក្នុងការអនុវត្ត  
# ADD [--chown=<user>:<group>] [--chmod=<perms>] [--checksum=<checksum>] <src>... <dest>  
#ADD [--chown=<user>:<group>] [--chmod=<perms>] ["<src>",... "<dest>"]  
ADD main.* /app               នាំទូទាត់ការអនុវត្តន៍  
#COPY [--chown=<user>:<group>] [--chmod=<perms>] <src>... <dest>  
#COPY [--chown=<user>:<group>] [--chmod=<perms>] ["<src>",... "<dest>"]  
COPY main.* /app  
#ENTRYPOINT ["executable", "param1", "param2"] បញ្ជាក់ការអនុវត្តន៍  
# WORKDIR /path/to/workdir    ផ្លូវការក្នុងការអនុវត្តន៍  
# VOLUME ["/data"]           ទូទាត់ការអនុវត្តន៍
```

# ENTRYPOINT vs. CMD

No ENTRYPPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT ["exec_entry", "p1_entry"]
No CMD	<i>error, not allowed</i>	/bin/sh -c exec_entry p1_entry
CMD ["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	cmd នៃវានេរណ៍នឹង entrypoint /bin/sh -c exec_entry p1_entry
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

ADD សម្រាប់ pull នៅ url

copy តែចាប់ពី local minify

# Example

- First try to create the simple python app and containerize it with Dockerfile
- docker build -t my\_demo .
- docker run my\_demo

ກໍລິນາດຈົກ  
ກໍລິນາດຈົກ

Dockerfile

```
FROM python:3.9
LABEL MAINTAINER="Krerk"

RUN mkdir /app
COPY main.py /app
RUN chmod +x /app/main.py
WORKDIR /app

CMD ["./main.py"]
```

Main.py

```
#!/usr/local/bin/python3
print('This is my secret program.')
```

# Publish image to registry

- Create a local registry to publish to..  
(we will remove later)
  - docker run --name registry -p 50000:5000 registry:2
  - #test with  
curl http://localhost:5000/v2/
- Push
  - docker tag my\_demo localhost:5000/v2/my\_demo
  - docker push localhost:5000/v2/my\_demo

Give it a try.

# Prerequisite

- docker client, docker desktop, or docker engine
- pull  
adminer  
mariadb:10.1  
php:7.3-alpine, php:8.2-alpine

# Scenario 1: command line

- Run a PHP script using docker engine.

- docker pull php:7.3-alpine

- docker run \  
-v \$PWD:/usr/src/myapp \  
-w /usr/src/myapp \  
php:7.3-alpine \  
php HelloWorld.php

```
HelloWorld.php  
<?php  
echo "PHP ".phpversion()."\\n";  
echo "Hello, World\\n";  
?>
```

- Now let's replace it with php:8.2-alpine

# Scenario 2 : service

Host (Mac OSX)

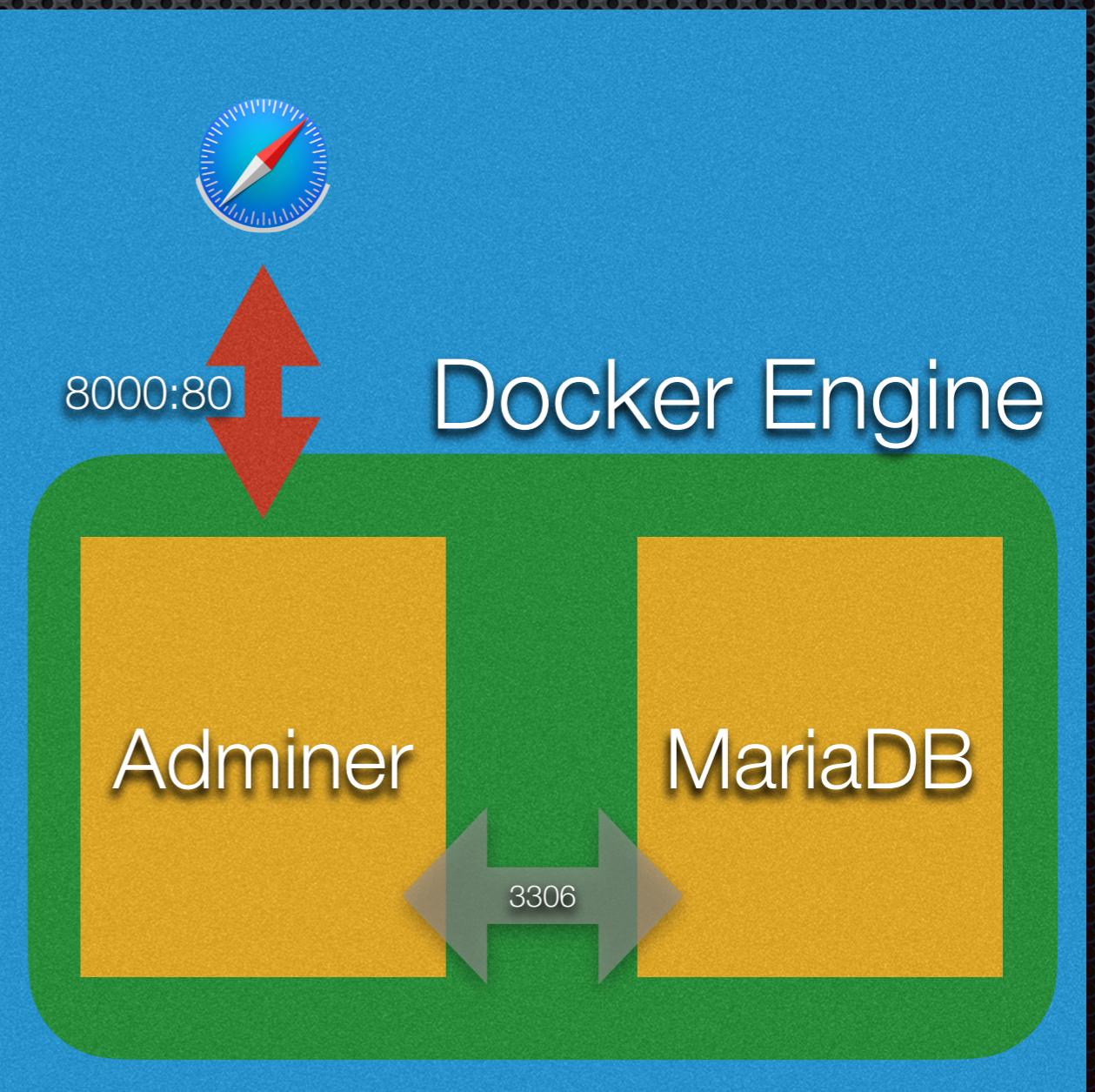
MariaDB (mariadb:10.1)

```
docker run --name ex-app-db \  
-e MYSQL_ROOT_PASSWORD=exroot \  
-d mariadb:10.1
```

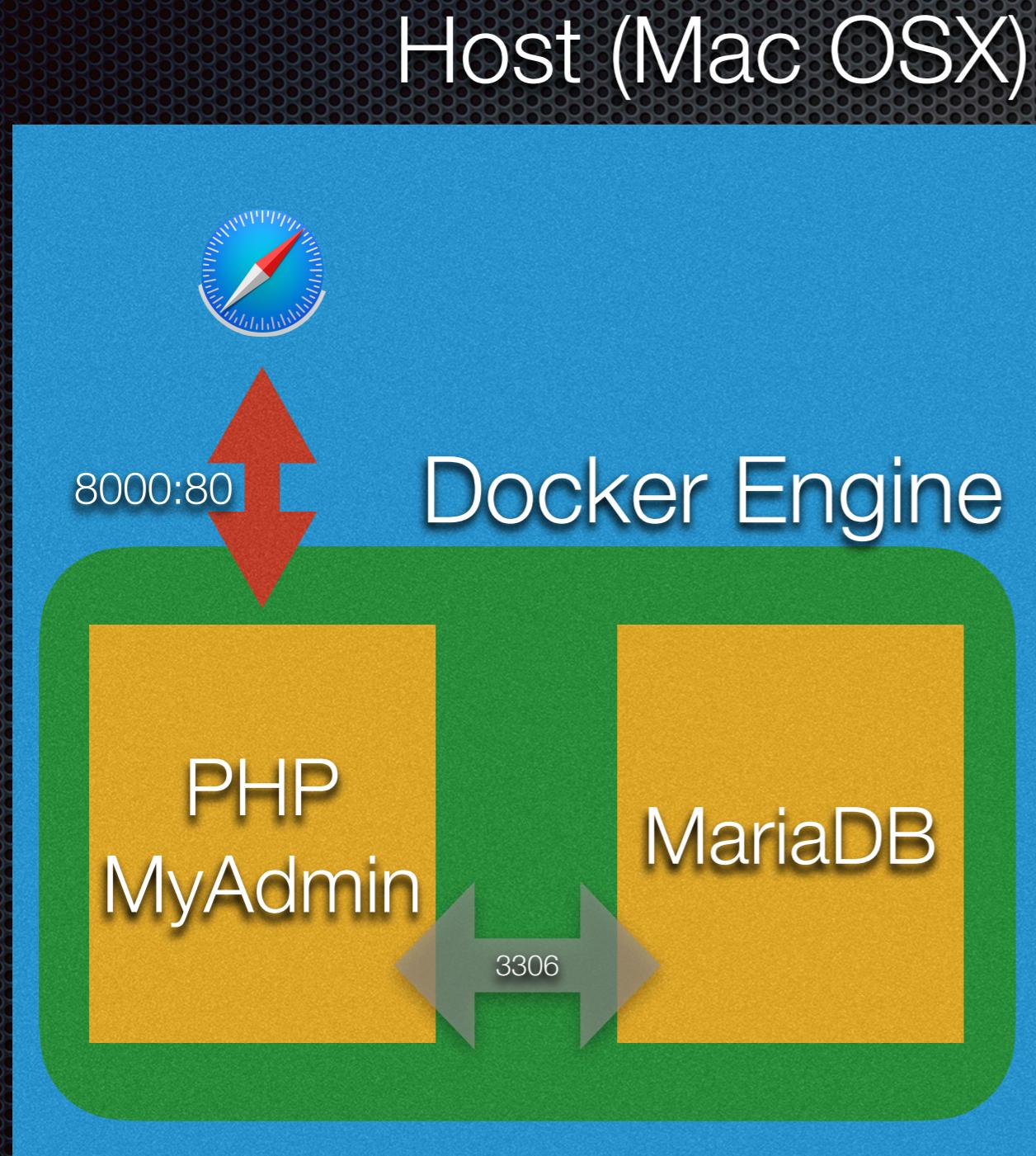
adminer 2 dmin or dñv mari

```
docker run --name ex-app-myadmin \  
--link ex-app-db:db \  
-d -p 8000:80 Adminer db
```

browse to <http://localhost:8000>



# Use docker compose



docker-compose.yaml

```
version: "3.0"
services:
  db:
    image: mariadb:10.1
    container_name: ex-app-db
    volumes:
      - ./data/mysql:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: exroot
  phpmyadmin:
    links:
      - db
    image: phpmyadmin/phpmyadmin
    container_name: ex-app-myadmin
    restart: always
    ports:
      - 8000:80
```

docker compose up -d

запущен в фоне

# Challenge/Activity

- In your team, create a local repository.
- Create a simple program (in your favorite programming language) that will read a value from an environment variable “secret\_user” and print “Hello, \$secret\_user” to the screen
- Containerize it (ie. Make a Dockerfile and publish to your local repository).
- Ask your friend to pull and run it from your local repository (ie. `docker run -e secret_user=Krerk my_repo.localhost:5000/demo` )

# References

- <https://docs.docker.com/engine/reference/builder/>
- <https://docs.docker.com/compose/compose-file/>
- <https://docs.docker.com/engine/swarm/stack-deploy/>

Not used

# Scenario 3 : Docker Swarm

index.php

- Work as a group (2-4 persons)
- Make a swarm of dockers.
- Deploy 4 instances of php application.  
(Let's ignore the database for now)

```
Hello from ..“  
<?php  
echo $_SERVER['SERVER_ADDR'];  
?>  
”..
```

# Steps

- To start a manager
  - docker swarm init
- To join a master (run on workers)
  - docker swarm join —token [token from master] [host:port]
- See your cluster
  - docker node ls

# Deploy a service/stack

- Deploy with
  - `docker stack deploy --compose-file docker-compose.yaml \ mydemo`
- Use docker service command to see your service
  - `docker stack services mydemo`
- Scale with
  - `docker service scale <SERVICE-ID>=<NUMBER-OF-TASKS>`

# More

- Delete stack with
  - `docker stack rm mydemo`
- Delete service with
  - `docker service rm [servicename]`
  - eg. `docker service rm registry`
- Drain a node (to remove)
  - `docker node update --availability drain [worker]`
- Leave the swarm
  - `docker swarm leave`

# Scenario 4: DB App (Optional)

- Deploy
  - 1 instance of mariadb
  - 4 instances of phpmyadmin on docker swarm

# Notes

- It is also possible to specify the number of replicas in the compose file.
- Now, we can also use kubernetes to manage docker swarm. (not in this workshop).
- This will use kubernetes deployment yaml.

# Notes

- docker vs. docker-compose
  - docker to manage single container
  - docker-compose to manage multiple containers
- docker service vs. docker stack
  - docker service to manage a single service
  - docker stack to manage group of services