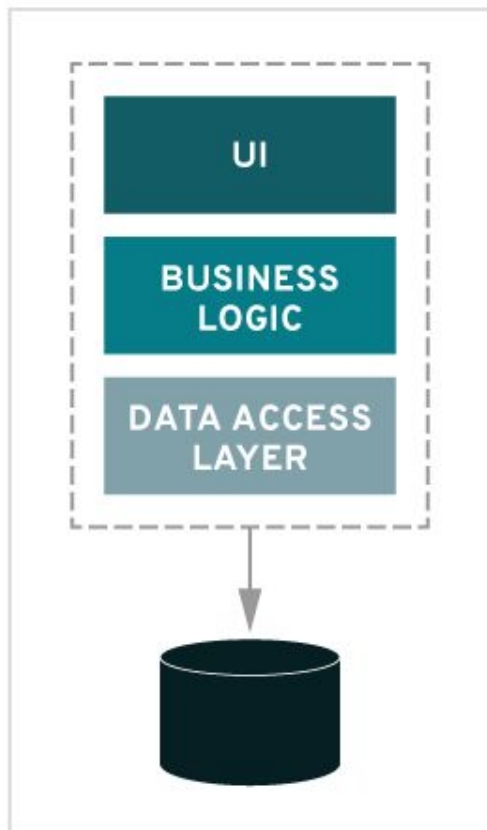


# Microservice & Basic Docker

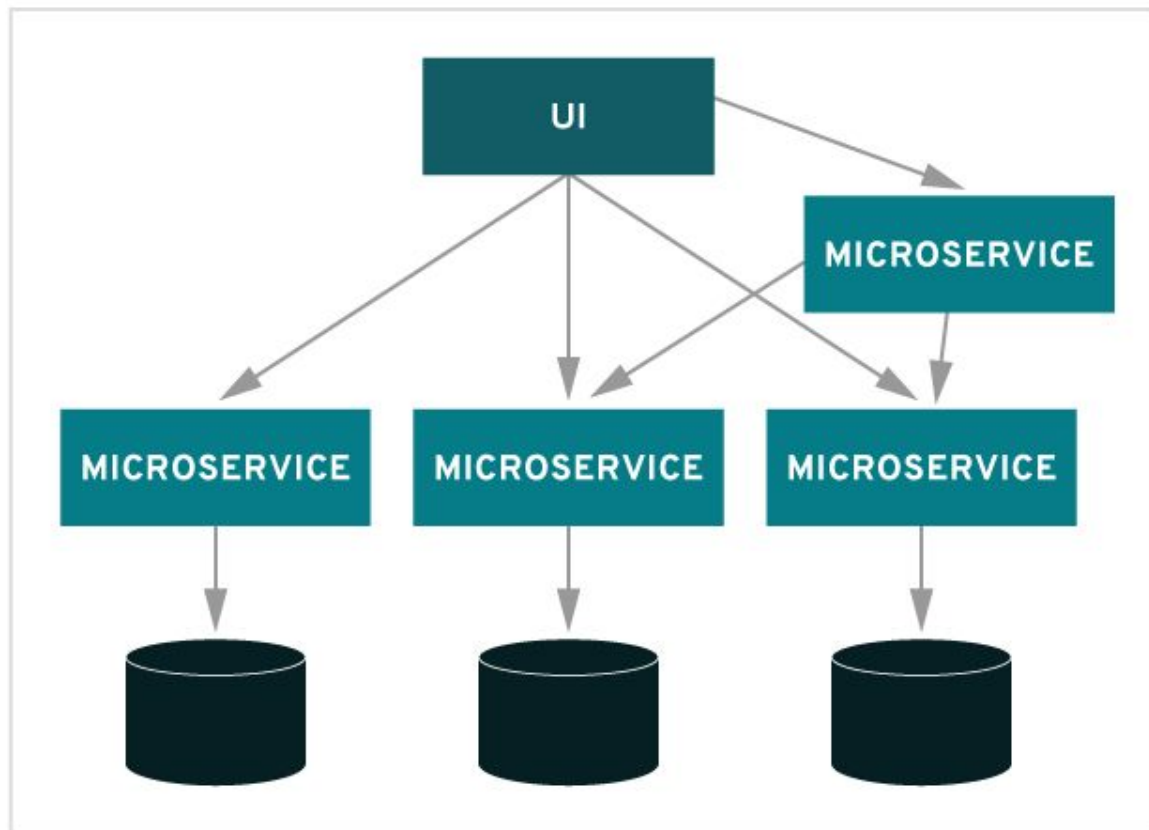
Ridnarong Promya (INO)

# MONOLITHIC



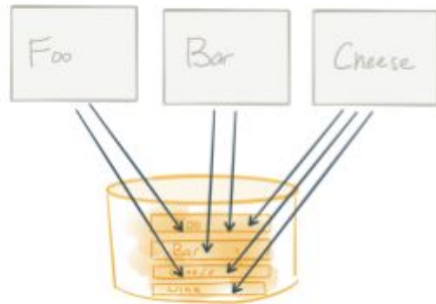
VS.

# MICROSERVICES

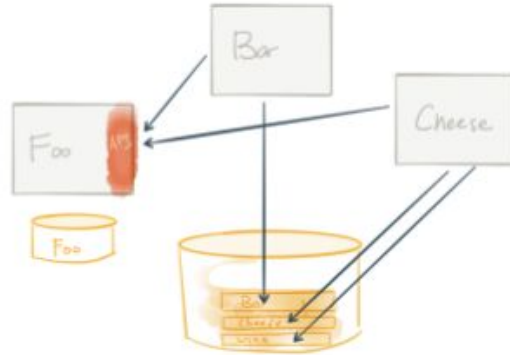


# How to transfer Monolith to Microservice

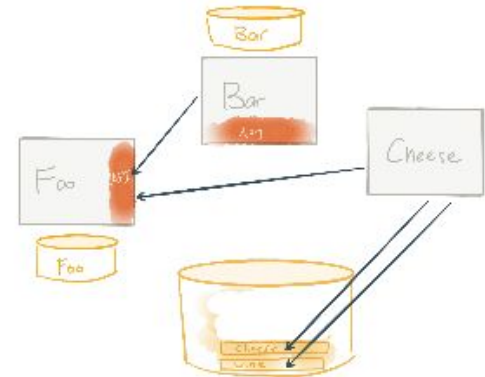
Identify modules



Break out API

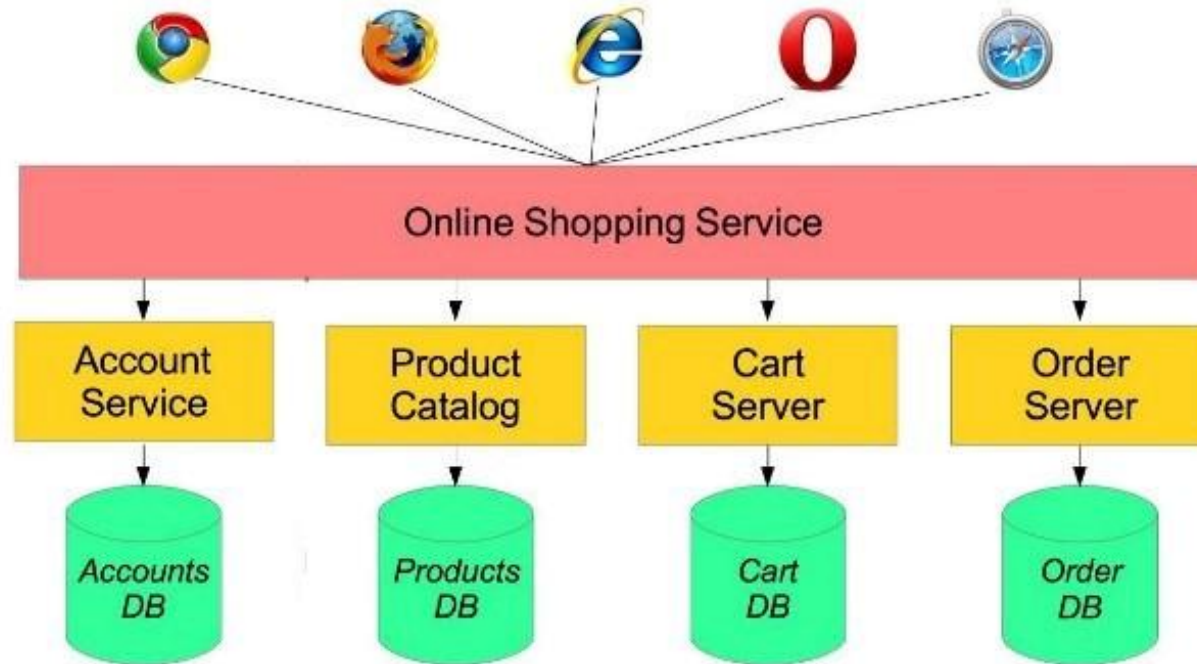


Rinse, repeat



<http://blog.christianposta.com/microservices/low-risk-monolith-to-microservice-evolution/>

# Microservice Architecture - Shopping online



# THE PATH TO MODERN APP DEV

## A DIGITAL DARWINISM

RE-ORG TO  
DEVOPS

SELF-SERVICE  
ON-DEMAND  
INFRA

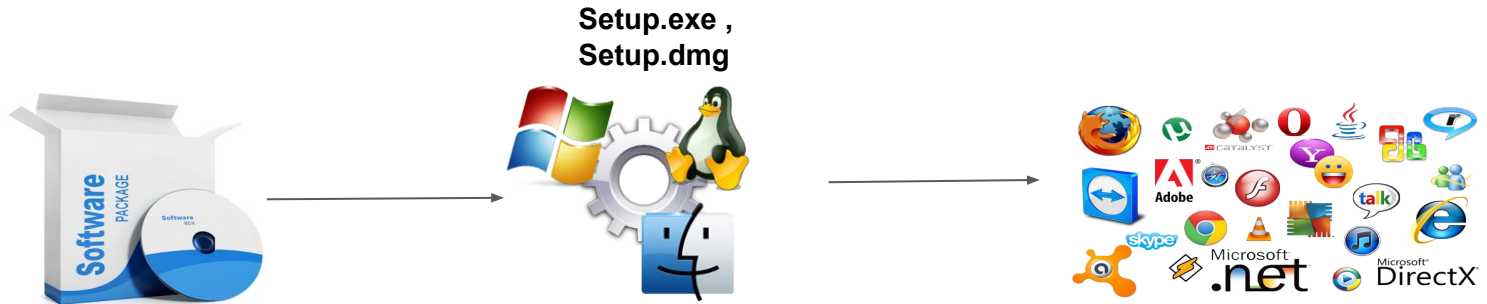
AUTOMATION

CONTINUOUS  
DELIVERY

ADVANCED  
DEPLOYMENT  
TECHNIQUES

MICROSERVICES  
.....  
FAST  
MONOLITH

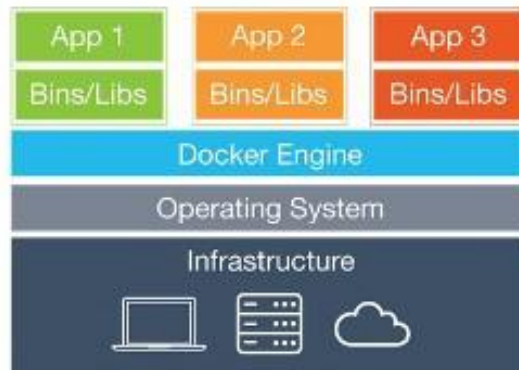
# บทบาทเรื่อง Container



# ทบทวนเรื่อง Container



Virtual Machines



Containers

# Container Benefit

- Isolation : สามารถแบ่งทรัพยากร สำหรับการทำงานแต่ละส่วน
- Standard : มีกลุ่มมาตรฐานสำหรับ Container Runtime และ Container Image Format
- Unify : สามารถนำไป Run บนที่ใดก็ได้ ไม่ว่าจะถูกสร้างที่ไหน ด้วย OS อะไร
- Portable : โปรแกรมจะถูกจัดเตรียมในรูปแบบของ Container Image
- Extensible : สามารถเริ่มต้นจาก Base Image ที่มีอยู่แล้ว
- Sharing : สามารถแบ่งปัน Container Image ไว้บน Repository





# ทบทวน docker ก่อนเริ่มใช้งาน

## Application

คือ สิ่งที่เราต้องการจะใช้งาน เช่น Apache, PHP, MySQL, WordPress, Laravel เป็นต้น

## Docker image

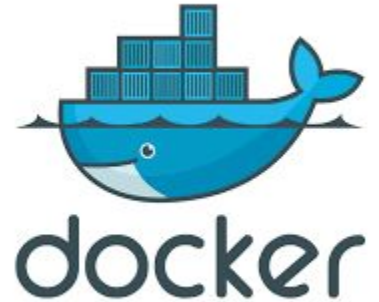
คือ ไฟล์ image ที่บรรจุ application ช่างต้นเอาไว้ โดยไฟล์นี้ได้มาจากการ build ไฟล์ **Dockerfile** ที่ใส่ command ของ Linux สำหรับติดตั้ง application ดังกล่าวเอาไว้

## Docker container

คือ กล่องที่บรรจุ application ที่พร้อมทำงาน ซึ่งเป็นผลมาจากการรัน Docker image

## Docker Hub

คือ ที่ๆ เราสามารถอัปโหลด image ที่ตัวเองสร้างขึ้นไปเพื่อแบ่งปันให้คนอื่นได้ใช้ด้วย หรือเราจะดาวน์โหลด image ที่คนอื่นทำไว้ดีแล้วมา ใช้งานเลยก็ได้



# ทบทวน docker ก่อนเริ่มใช้งาน



## ORCHESTRATE

Initialize swarm mode and listen on a specific interface  
`docker swarm init --advertise-addr 10.1.0.2`

Join an existing swarm as a manager node  
`docker swarm join --token <manager-token> 10.1.0.2:2377`

Join an existing swarm as a worker node  
`docker swarm join --token <worker-token> 10.1.0.2:2377`

List the nodes participating in a swarm  
`docker node ls`

Create a service from an image exposed on a specific port and deploy 3 instances  
`docker service create --replicas 3 -p 80:80 --name web nginx`

List the services running in a swarm  
`docker service ls`

Scale a service  
`docker service scale web=5`

List the tasks of a service  
`docker service ps web`

## BUILD

Build an image from the Dockerfile in the current directory and tag the image  
`docker build -t myapp:1.0 .`

List all images that are locally stored with the Docker engine  
`docker images`

Delete an image from the local image store  
`docker rmi alpine:3.4`

## SHIP

Pull an image from a registry  
`docker pull alpine:3.4`

Retag a local image with a new image name and tag  
`docker tag alpine:3.4 myrepo/myalpine:3.4`

Log in to a registry (the Docker Hub by default)  
`docker login my.registry.com:8000`

Push an image to a registry  
`docker push myrepo/myalpine:3.4`

## RUN

```
docker run
  --rm remove container automatically after it exits
  -it connect the container to terminal
  --name web name the container
  -p 5000:80 expose port 5000 externally and map to port 80
  -v ~/dev:/code create a host mapped volume inside the container
  alpine:3.4 the image from which the container is instantiated
  /bin/sh the command to run inside the container
```

Stop a running container through SIGTERM  
`docker stop web`

Stop a running container through SIGKILL  
`docker kill web`

Create an overlay network and specify a subnet  
`docker network create --subnet 10.1.0.0/24 --gateway 10.1.0.1 -d overlay mynet`

List the networks  
`docker network ls`

List the running containers  
`docker ps`

Delete all running and stopped containers  
`docker rm -f $(docker ps -aq)`

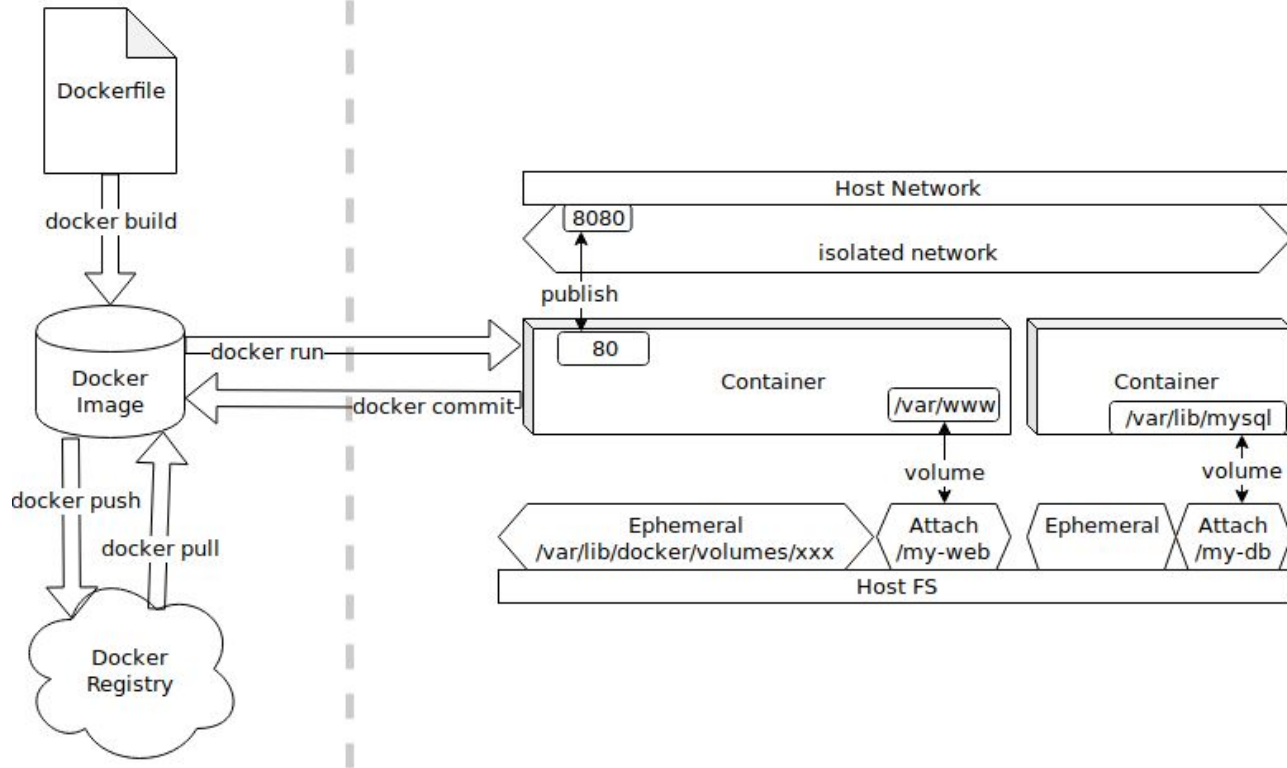
Create a new bash process inside the container and connect it to the terminal  
`docker exec -it web bash`

Print the last 100 lines of a container's logs  
`docker logs --tail 100 web`

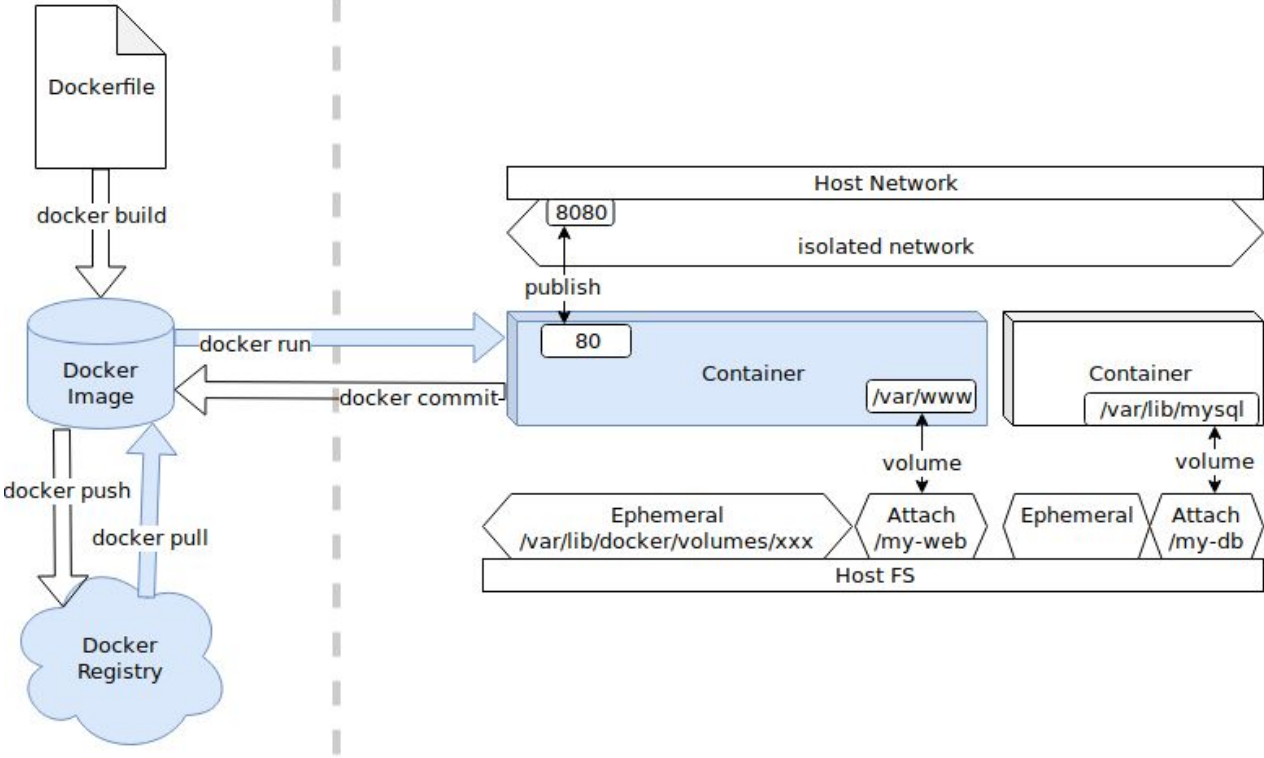
# Docker Commands

<code>docker run [OPTIONS] IMAGE [COMMAND] [ARG...]</code>	<code>docker pull [OPTIONS] NAME[:TAG @DIGEST]</code>
<code>docker inspect [OPTIONS] NAME ID [NAME ID...]</code>	<code>docker push [OPTIONS] NAME[:TAG]</code>
<code>docker ps [OPTIONS]</code>	<code>docker rmi [OPTIONS] IMAGE [IMAGE...]</code>
<code>docker build [OPTIONS] PATH   URL   -</code>	<code>docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]</code>
<code>docker start [OPTIONS] CONTAINER [CONTAINER...]</code>	<code>docker logs [OPTIONS] CONTAINER</code>
<code>docker stop [OPTIONS] CONTAINER [CONTAINER...]</code>	<code>docker exec [OPTIONS] CONTAINER COMMAND [ARG...]</code>
<code>docker rm [OPTIONS] CONTAINER [CONTAINER...]</code>	<code>docker help [command]</code>

# Container Development Process



# Docker In action



.....Coffee Break.....



# The Twelve-Factor App: 12 ประเด็นพื้นฐานสู่การพัฒนา SaaS Cloud ที่ยั่งยืน

<https://12factor.net/>

## THE TWELVE FACTORS

### I. Codebase

One codebase tracked in revision control, many deploys

### II. Dependencies

Explicitly declare and isolate dependencies

### III. Config

Store config in the environment

### IV. Backing services

Treat backing services as attached resources

### V. Build, release, run

Strictly separate build and run stages

### VI. Processes

Execute only one process per instance of an application

### VII. Port binding

Export services via port binding

### VIII. Concurrency

Scale out via the process model

### IX. Disposability

Maximize robustness with fast startup and graceful shutdown

### X. Dev/prod parity

Keep development, staging, and production as similar as possible

### XI. Logs

Treat logs as event streams

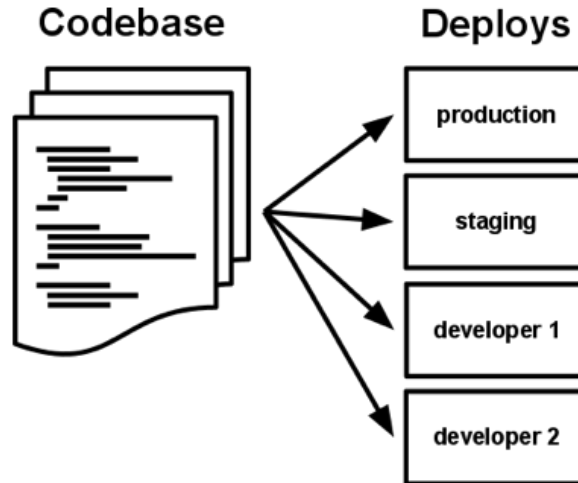
### XII. Admin processes

Run admin / management tasks as one-off processes

# 1. Codebase

หนึ่ง Codebase จัดเก็บเพียง หนึ่ง application จัดเก็บ code ด้วย Version Control เช่น Git หรือ SVN

แต่ หนึ่ง Codebase สามารถ deploys ได้หลายที่





## 2. Dependencies

เป้าหมายคือ การแยก Dependency ต่างๆ ออกจากระบบงาน  
ระบุ package และ เลขเวอร์ชันที่ใช้ในงานใน application อย่างชัดเจนใน Codebase

ผลที่ได้ ทำให้สามารถ deploy ระบบงานได้บ่อยและง่ายขึ้น

# 3. Config

เก็บ config ต่าง ๆ ใน Environment Variables

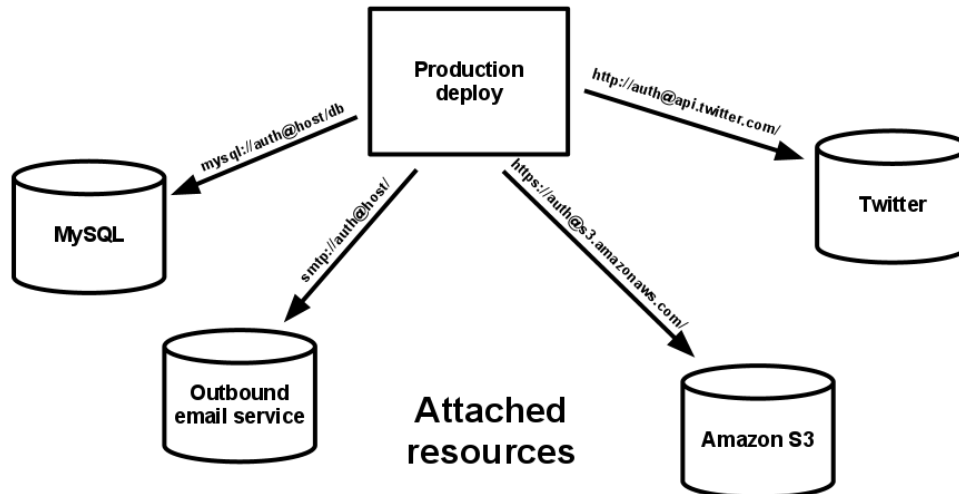
ทำให้สามารถย้าย code ไปทำงานต่าง platform ได้ โดยแก้ Environment ตาม platform เหล่านั้น

แล้ว configuration มีอะไรบ้าง

- ข้อมูลการใช้งาน database และ service ต่าง ๆ
- สิ่งที่ต้องกำหนดสำหรับการ deploy ในแต่ละ environmet
- อื่น ๆ ที่ต้องเปลี่ยนแปลงตาม environment เช่น dev , test , staging , production

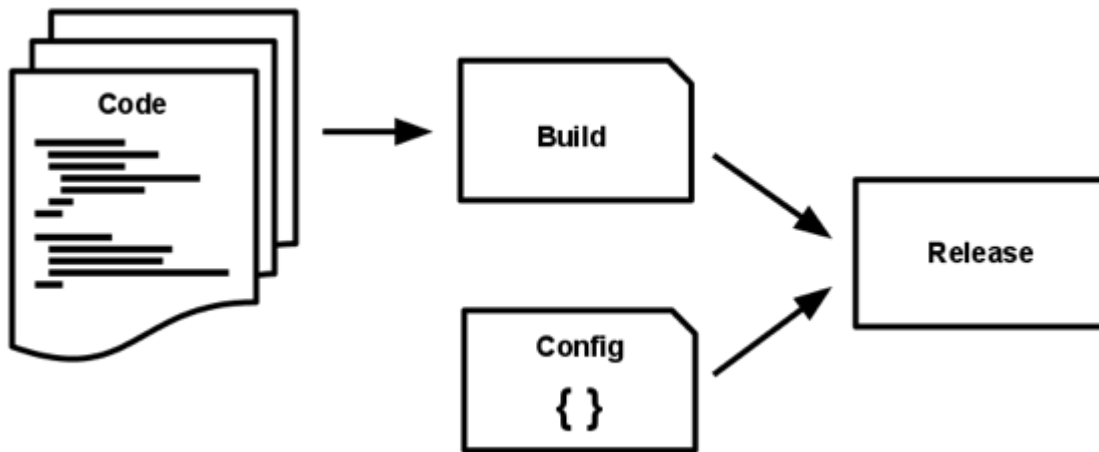
## 4. Backing service

Backing service คือ บริการที่ application ของเราไปเรียกใช้ เช่น Database ซึ่ง Service เหล่านี้ ต้องสามารถเข้าถึงได้ง่าย เช่น ผ่าน url และเปลี่ยนแปลงได้ง่าย ทำให้สามารถถูกแทนที่ตลอดเวลา โดยการแทนที่พวกนี้ ต้องไม่กระทบกับ code ของเรา



## 5. Build, Release, Run

แบ่งการ deploy ออกเป็น Build, Release และ Run Stages อย่างชัดเจน



เพื่อผลักระงานที่ซับซ้อนให้กับ Build Stage เนื่องจากยังไม่ได้นำไปใช้งานจริง หากเกิดข้อผิดพลาดก็นำมาแก้ไขได้ง่าย ส่วน การ run ก็จะต้องออกแบบให้ใช้เวลาให้น้อยที่สุด

## 6. Processes

อย่าจัดเก็บข้อมูลบนโปรเซส คือ ห้ามจำสถานะบนตัวมัน

การทำงานแบบ Stateless จะทำให้สามารถ scale ได้ง่ายขึ้น

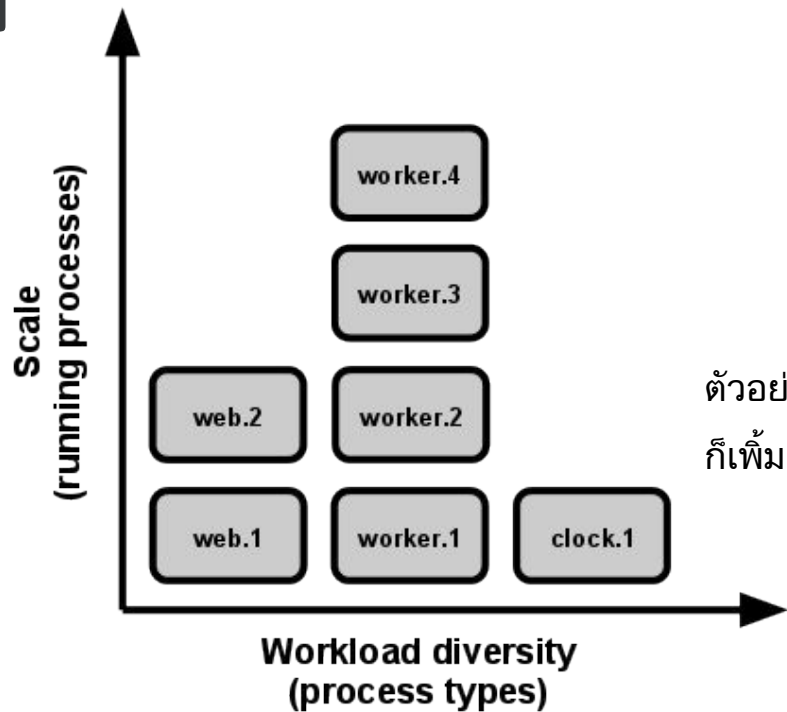
## 7. Port binding

เพื่อให้ application สามารถโยกย้ายข้ามแพลตฟอร์มได้ง่าย จึงไม่ควรผูกติดกับ Webserver

ควรทำ service เพื่อให้บริการผ่าน PORT ของ HTTP

## 8. Concurrency

ขยายระบบด้วยการใช้ process model คือ งานไหนที่ต้องการใช้มากขึ้นก็เพิ่มงานส่วนนั้นเข้าไป



ตัวอย่างเช่น มีงานเกี่ยวกับการประมวลผลมาก ก็เพิ่ม worker เข้าไป

## 9. Disposability

Application ควรจะมี startup time ที่ไว และต้องมีการจัดการข้อมูลให้เรียบร้อย  
เมื่อมีการ shutdown



# 10. Dev/prod parity

**ทำสภาพแวดล้อมของ development Staging/Production ให้เหมือนกันมากที่สุด**

Time gab

ทำที่ละน้อย deploy ทันที เพื่อให้ development กับ Staging/Production  
แตกต่างกันน้อยที่สุด

Personal gab

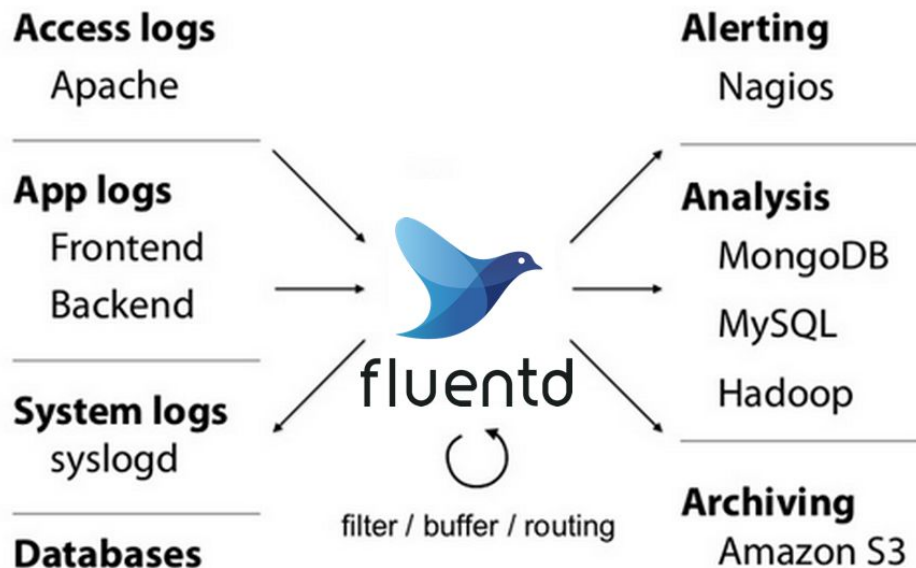
คนเขียนโค้ด เป็นคนเดียวกันกับคน deploy

Tools gab

ใช้งานซอฟต์แวร์ ให้ตรงกับที่ production ใช้

# 11. Logs

สามารถตรวจดู Logs ตามเหตุการณ์ที่เราสนใจได้ภายหลังและสามารถนำข้อมูลของ Logs ไปประมวลผลอีกที



## 12. Admin processes

มีคำสั่งสำหรับการทำงานด้าน Admin/Management

โดยคำสั่งนี้จะสร้างโปรเซสการทำงานขึ้นมาใหม่

ไม่เกี่ยวข้องกับ โปรเซสของ application

โปรเซสการ management จะถูกทำลายหลังการใช้งานเสร็จสิ้น

# เตรียมเครื่องให้พร้อม

- Develop environment [Python]
  - Windows: <https://docs.djangoproject.com/en/2.0/howto/windows/>
  - Linux: <https://docs.djangoproject.com/en/2.0/topics/install/>
- Operation environment [Docker]
  - <https://www.docker.com/community-edition>
- Windows
- Ubuntu

# Lunch



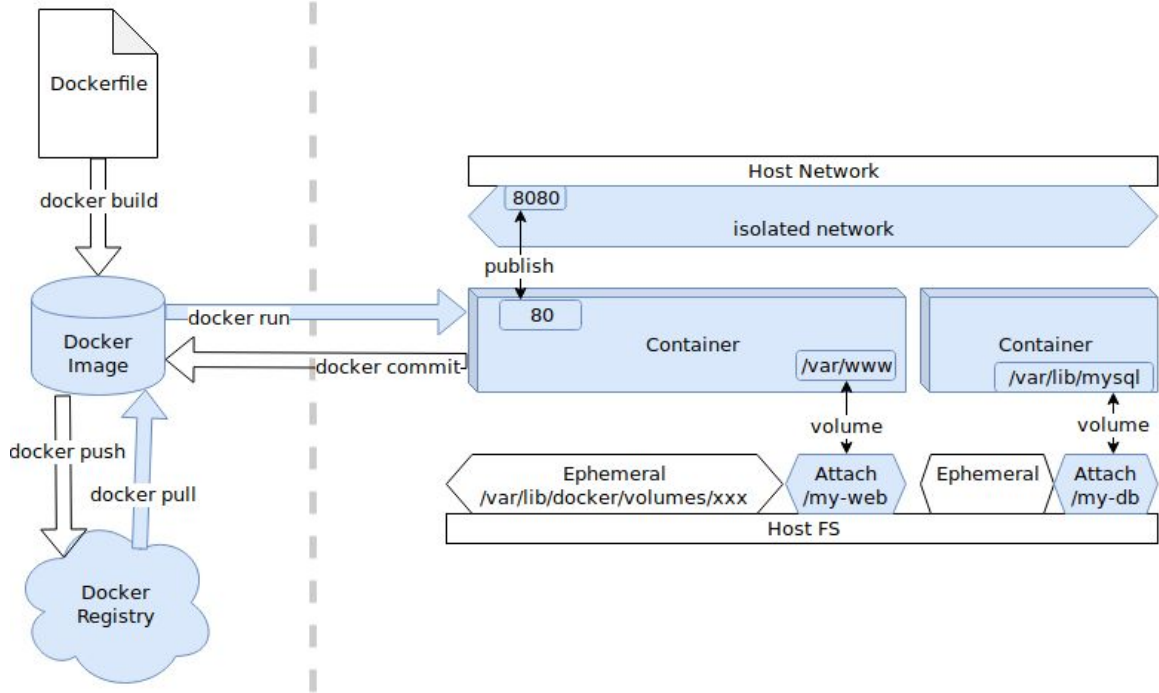
# Exercise

## Static web

- nginx  
[https://hub.docker.com/\\_/nginx/](https://hub.docker.com/_/nginx/)
- coderaiser/cloudcmd  
<https://hub.docker.com/r/coderaiser/cloudcmd>

## Classic 3-tier Application

- wordpress  
[https://hub.docker.com/\\_/wordpress/](https://hub.docker.com/_/wordpress/)
- mariadb  
[https://hub.docker.com/\\_/mariadb/](https://hub.docker.com/_/mariadb/)



## 1. Static web




```
$ mkdir my-web
$ echo "Hello, world" > my-web/index.html
$ docker run -v "${PWD}/my-web"/:/usr/share/nginx/html -p 8080:80 -it --rm nginx
$ docker run -v /c/User/Nectec/my-web:/usr/share/nginx/html -p 8080:80 -it --rm nginx
$ docker run -v c:/User/Nectec/my-web:/usr/share/nginx/html -p 8080:80 -it --rm nginx
$ docker run -t --rm -v "${PWD}/my-web"/:/root -it --rm -p 8000:8000 coderaiser/cloudcmd
```

## 2. Classic 3-tier Application

```
$ docker network create my-wordpress
$ mkdir db-data
$ docker run -it --rm -e MYSQL_ROOT_PASSWORD=my-secret-pw -v "${PWD}/db-data":/var/lib/mysql --name mysql --network my-wordpress mariadb
$ mkdir wp-content
$ docker run -e WORDPRESS_DB_PASSWORD=my-secret-pw -v "${PWD}/wp-content":/var/www/html/wp-content/ --network my-wordpress -it --rm -p 8000:80 wordpress
```

# Docker Image มาจากที่ไหน

<https://hub.docker.com/explore/>

Search		Explore	Help	Sign up	Sign in
Explore Official Repositories					
	nginx official	7.6K STARS	10M+ PULLS	>	DETAILS
	alpine official	2.9K STARS	10M+ PULLS	>	DETAILS
	httpd official	1.4K STARS	10M+ PULLS	>	DETAILS






# การได้มาของ Docker image (..ที่ดี..)

- การใช้ Docker Image ที่มีอยู่แล้ว
- Build จาก Dockerfile
- จัดการ Dependencies อย่างเหมาะสม ของแต่ละ Applications
- สามารถ Config ขณะ Runtime ได้
- แยก Application แล้วผูกเข้าด้วยกันด้วย link
- เก็บข้อมูลไว้ใน Persistent volume ภายนอก
- จัดการ Log/Monitoring อย่างเหมาะสม

# การใช้ Docker image ที่มีอยู่แล้ว

- มองหา software ใน <https://hub.docker.com/>
- ดูจาก official + vendor + pull + last push + git
- นำมาใช้เลย แบบ config ขณะ run
- นำมาเป็น base ใน Dockerfile

 <a href="#">wordpress</a> official	2.0K STARS	10M+ PULLS
 <a href="#">bitnami/wordpress</a> public   automated build	59 STARS	1M+ PULLS
 <a href="#">centurylink/wordpress</a> public   automated build	15 STARS	1M+ PULLS

# ศึกษา Image อย่าง เข้าใจ

OFFICIAL REPOSITORY



Last pushed: 4 days ago

## Supported tags and respective Dockerfile links

- 7.2.0-cli-stretch, 7.2-cli-stretch, 7-cli-stretch, cli-stretch, 7.2.0-stretch, 7.2-stretch, 7-stretch, stretch, 7.2.0-cli, 7.2-cli, 7-cli, cli, 7.2.0, 7.2, 7, latest ([7.2/stretch/cli/Dockerfile](#))
- 7.2.0-apache-stretch, 7.2-apache-stretch, 7-apache-stretch, apache-stretch, 7.2.0-apache, 7.2-apache, 7-apache, apache ([7.2/stretch/apache/Dockerfile](#))
- 7.2.0-fpm-stretch, 7.2-fpm-stretch, 7-fpm-stretch, fpm-stretch, 7.2.0-fpm, 7.2-fpm, 7-fpm, fpm ([7.2/stretch/fpm/Dockerfile](#))
- 7.2.0-zts-stretch, 7.2-zts-stretch, 7-zts-stretch, zts-stretch, 7.2.0-zts, 7.2-zts, 7-zts, zts ([7.2/stretch/zts/Dockerfile](#))
- 7.2.0-cli-alpine3.7, 7.2-cli-alpine3.7, 7-cli-alpine3.7, cli-alpine3.7, 7.2.0-alpine3.7, 7.2-alpine3.7, 7-alpine3.7, alpine3.7 ([7.2/alpine3.7/cli/Dockerfile](#))
- 7.2.0-fpm-alpine3.7, 7.2-fpm-alpine3.7, 7-fpm-alpine3.7, fpm-alpine3.7 ([7.2/alpine3.7/fpm/Dockerfile](#))
- 7.2.0-zts-alpine3.7, 7.2-zts-alpine3.7, 7-zts-alpine3.7, zts-alpine3.7 ([7.2/alpine3.7/zts/Dockerfile](#))
- 7.2.0-cli-alpine3.6, 7.2-cli-alpine3.6, 7-cli-alpine3.6, cli-alpine3.6, 7.2.0-alpine3.6, 7.2-alpine3.6, 7-alpine3.6, alpine3.6, 7.2.0-cli-alpine, 7.2-cli-alpine, 7-cli-alpine, cli-alpine, 7.2.0-alpine, 7.2-alpine, 7-alpine, alpine ([7.2/alpine3.6/cli/Dockerfile](#))
- 7.2.0-fpm-alpine3.6, 7.2-fpm-alpine3.6, 7-fpm-alpine3.6, fpm-alpine3.6, 7.2.0-fpm-alpine, 7.2-fpm-alpine, 7-fpm-alpine, fpm-alpine ([7.2/alpine3.6/fpm/Dockerfile](#))
- 7.2.0-zts-alpine3.6, 7.2-zts-alpine3.6, 7-zts-alpine3.6, zts-alpine3.6, 7.2.0-zts-alpine, 7.2-zts-alpine, 7-zts-alpine, zts-alpine ([7.2/alpine3.6/zts/Dockerfile](#))

## With Apache

More commonly, you will probably want to run PHP in conjunction with Apache httpd. Conveniently, there's a version of the PHP container that's packaged with the Apache web server.

### Create a Dockerfile in your PHP project

```
FROM php:7.0-apache
COPY src/ /var/www/html/
```

Where `src/` is the directory containing all your PHP code. Then, run the commands to build and run the Docker image:

```
$ docker build -t my-php-app .
$ docker run -d --name my-running-app my-php-app
```

We recommend that you add a custom `php.ini` configuration. COPY it into `/usr/local/etc/php` by adding one more line to the Dockerfile above and running the same commands to build and run:

```
FROM php:7.0-apache
COPY config/php.ini /usr/local/etc/php/
COPY src/ /var/www/html/
```

Where `src/` is the directory containing all your PHP code and `config/` contains your `php.ini` file.

### Without a Dockerfile

If you don't want to include a `Dockerfile` in your project, it is sufficient to do the following:

```
$ docker run -d -p 80:80 --name my-apache-php-app -v "$PWD":/var/www/html php:
```

---

# Build จาก Dockerfile

- แนะนำมากกว่า docker commit
- องค์ความรู้ อยู่ในรูปแบบของ text เป็นเอกสารได้ในตัว
- ทำซ้ำ และปรับแต่งได้
- ใช้ version control ได้
- ใช้งาน automate build

```
1 FROM php:5.6-apache
2
3 # install the PHP extensions we need
4 RUN set -ex; \
5     \
6     apt-get update; \
7     apt-get install -y \
8         libjpeg-dev \
9         libpng-dev \
10        ; \
11    rm -rf /var/lib/apt/lists/*; \
12    \
13    docker-php-ext-configure gd --with-png-dir=/usr --with-jpeg-dir=/usr; \
14    docker-php-ext-install gd mysqli opcache
```

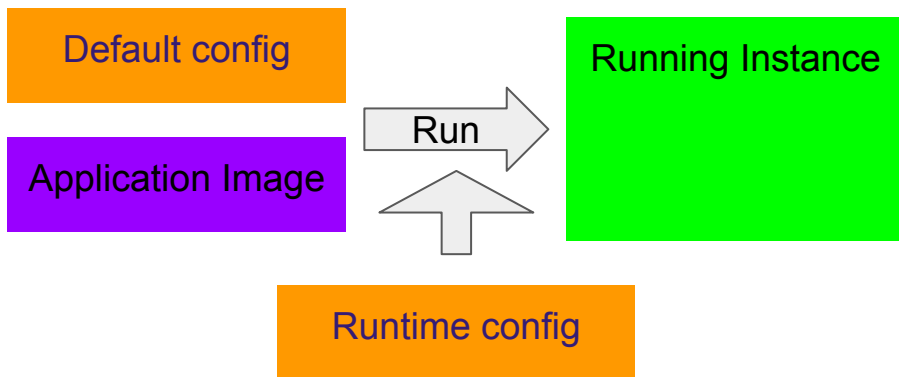
# จัดการ Dependencies อย่างไร

- แยก lib ออกจาก code
- ภาษาสมัยใหม่มักจะมีตัวจัดการ Dependencies
  - เช่น PHP:Composer, Java:Maven, Python:pip, JS:npm
  - รายการของ dependencies จะอยู่ในรูปแบบของไฟล์
  - สะดวกใช้เวลา build



# Config at runtime

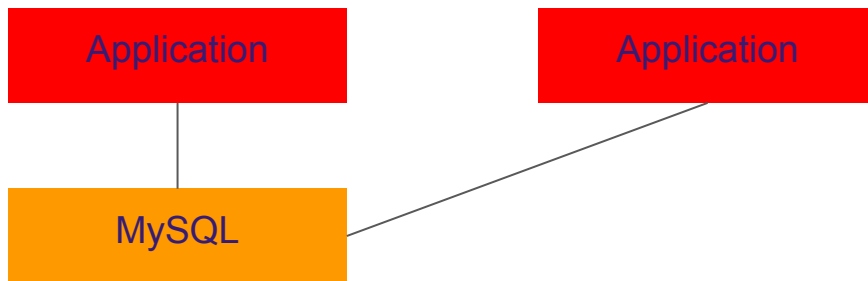
- ควรแยก Config ออกจาก code เช่น DB Access Credential
- ผ่าน Environment Variable [docker run -e XXX=xxx]
- ผ่าน config file [docker run -v /myconfig:/etc/app/conf]
- Customized at runtime



- `-e WORDPRESS_DB_HOST=...` (defaults to the
- `-e WORDPRESS_DB_USER=...` (defaults to "ro
- `-e WORDPRESS_DB_PASSWORD=...` (defaults environment variable from the linked `mysql` cont
- `-e WORDPRESS_DB_NAME=...` (defaults to "w
- `-e WORDPRESS_TABLE_PREFIX=...` (default default table prefix in `wp-config.php`)
- `-e WORDPRESS_AUTH_KEY=...` , `-e WORDPI`  
`WORDPRESS_LOGGED_IN_KEY=...` , `-e WORD`  
`WORDPRESS_AUTH_SALT=...` , `-e WORDPRES`  
`WORDPRESS_LOGGED_IN_SALT=...` , `-e WOR`  
random SHA1s)

# Linked Container หลายอันเข้าด้วยกัน

- แยกการทำงาน ระหว่าง application กับ backend service เชื่อมกันผ่าน --link
- หลักการ “หนึ่งอัน ทำหนึ่งอย่าง-Do one thing at a time”
- หากต้องการ scale สามารถ scale ได้เฉพาะบาง container





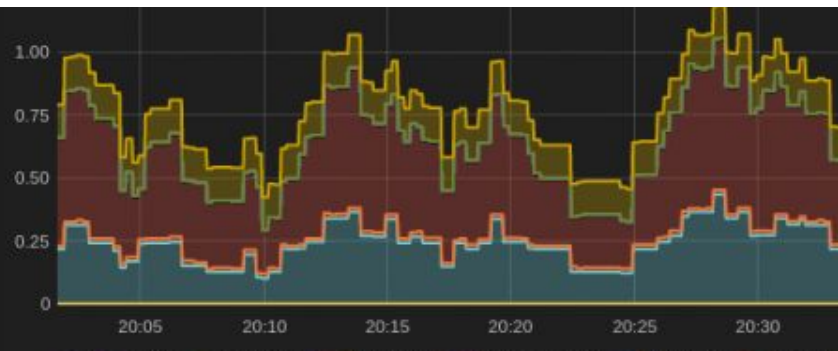
# แยกข้อมูลไว้โดย Persistent volume

- ผ่านการใช้ `-v <host path>:<container path>`
- ควรแยก User data ออกจาก application
  - ลบ container แล้ว data ยังอยู่
  - สะดวกในการย้าย Application
  - เหมาะกับทำงานบน cluster (Kubernetes, Docker)

# จัดการ Log/monitoring อย่างเหมาะสม

- Log สำคัญมาก เพราะ เราไม่สามารถ ssh เข้าไปเพื่อตรวจสอบได้ ต้องอาศัยข้อมูลจาก Log
- Log ผ่าน STDOUT แทนที่ Log file
- ช่วยให้จัดการเรื่อง Log ได้ง่ายขึ้น ดูผ่าน docker logs หรือ ใช้ log shipper ได้ง่าย ลดการจัดการไฟล์ Log ทำ centralize log ได้
- มี health check สำหรับ process ที่ทำงานเคียงข้างกัน ทำให้เรานำ monitoring tools ไป check การทำงานได้

```
04:31:08,873 INFO [org.jboss.as.server] (Serve
"keycloak-server.war")
04:31:08,962 INFO [org.jboss.as.server] (Contr
04:31:08,965 INFO [org.jboss.as] (Controller B
anagement
04:31:08,965 INFO [org.jboss.as] (Controller B
04:31:08,966 INFO [org.jboss.as] (Controller B
84ms - Started 546 of 882 services (604 service
04:35:42,187 WARN [org.jboss.as.domain.manager
```



# Dockerfile quick reference

Keyword	Description
FROM	Base image สำหรับเริ่มต้น
MAINTAINER	ข้อมูลผู้เขียน Dockerfile
RUN	รัน command ใน container แล้วบันทึก ลงใน image
CMD	Command ที่จะทำงานเมื่อเริ่ม container
EXPOSE	ข้อมูลเกี่ยวกับการ bind port
ENV	ตั้งค่า environment variable
ADD	Copy file, folder, URL ไปยัง image
COPY	Copy file, folder ไปยัง image

Keyword	Description
ENTRYPOINT	Command ที่จะทำงานเมื่อเริ่ม container ทำงานก่อน CMD
VOLUME	ข้อมูลเกี่ยวกับโฟลเดอร์ที่สามารถ จัดเก็บ ภายนอกได้
USER	เปลี่ยน user ที่ทำงาน
WORKDIR	เปลี่ยน folder ที่ทำงาน
ARG	ตัวแปรที่อนุญาตให้ส่งผ่าน --build-arg ขณะ build image

ตัวอย่างการสร้าง Image

# เรียนรู้จากมืออาชีพ Wordpress

<https://github.com/docker-library/wordpress/blob/43d32697c6862dcb48ca520e87e1e0fb585aee03/php5.6/apache/Dockerfile>

OFFICIAL REPOSITORY

wordpress 

Last pushed: 2 days ago

## Supported tags and respective Dockerfile links

- 4.9.1-php5.6-apache , 4.9-php5.6-apache , 4-php5.6-apache , php5.6-apache , 4.9.1-php5.6 , 4.9-php5.6 , 4-php5.6 , php5.6 ([php5.6/apache/Dockerfile](#))

```
1 FROM php:5.6-apache
2
3 # install the PHP extensions we need
4 RUN set -ex; \
5     \
6     apt-get update; \
7     apt-get install -y \
8         libjpeg-dev \
9         libpng-dev \
10        ; \
11    rm -rf /var/lib/apt/lists/*; \
12    \
13    docker-php-ext-configure gd --with-png-dir=/usr --with-jpeg-dir=/usr; \
14    docker-php-ext-install gd mysqli opcache
15 # TODO consider removing the *-dev deps and only keeping the necessary lib* packages
16
17 # set recommended PHP.ini settings
18 # see https://secure.php.net/manual/en/opcache.installation.php
19 RUN { \
20     echo 'opcache.memory_consumption=128'; \
21     echo 'opcache.interned_strings_buffer=8'; \
22     echo 'opcache.max_accelerated_files=4000'; \
23     echo 'opcache.revalidate_freq=2'; \
24     echo 'opcache.fast_shutdown=1'; \
25     echo 'opcache.enable_cli=1'; \
26 } > /usr/local/etc/php/conf.d/opcache-recommended.ini
27
28 RUN a2enmod rewrite expires
```

การใช้ base image จาก image ที่ใกล้เคียง

การใช้ RUN แบบหลายๆ command ช่วยเรื่อง image layers

การแก้ไขไฟล์โดยใช้ echo

Enable apache module

```
30 VOLUME /var/www/html
31
32 ENV WORDPRESS_VERSION 4.9.1
33 ENV WORDPRESS_SHA1 892d2c23b9d458ec3d44de59b753adb41012e903
34
35 RUN set -ex; \
36     curl -o wordpress.tar.gz -fSL "https://wordpress.org/wordpress-${WORDPRESS_VERSION}.tar.gz"; \
37     echo "$WORDPRESS_SHA1 *wordpress.tar.gz" | sha1sum -c -; \
38 # upstream tarballs include ./wordpress/ so this gives us /usr/src/wordpress
39     tar -xzf wordpress.tar.gz -C /usr/src/; \
40     rm wordpress.tar.gz; \
41     chown -R www-data:www-data /usr/src/wordpress
42
43 COPY docker-entrypoint.sh /usr/local/bin/
44
45 ENTRYPOINT ["docker-entrypoint.sh"]
46 CMD ["apache2-foreground"]
```

เตรียม Volume สำหรับ mount จาก  
ภายนอก

ใช้ shell script ช่วยเช่นเรื่อง permission  
และการ set ค่าจาก ENV

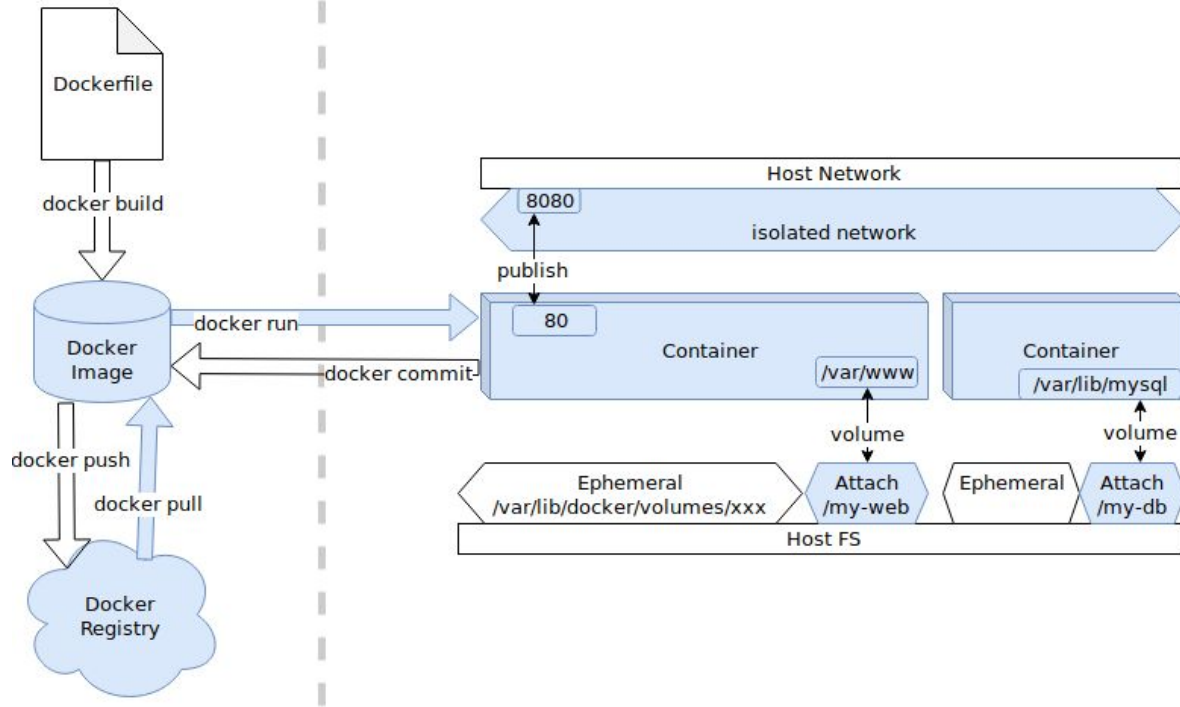
บางโปรแกรมต้องใช้คำสั่งเฉพาะเพื่อ run  
เป็น foreground

.....Coffee Break.....

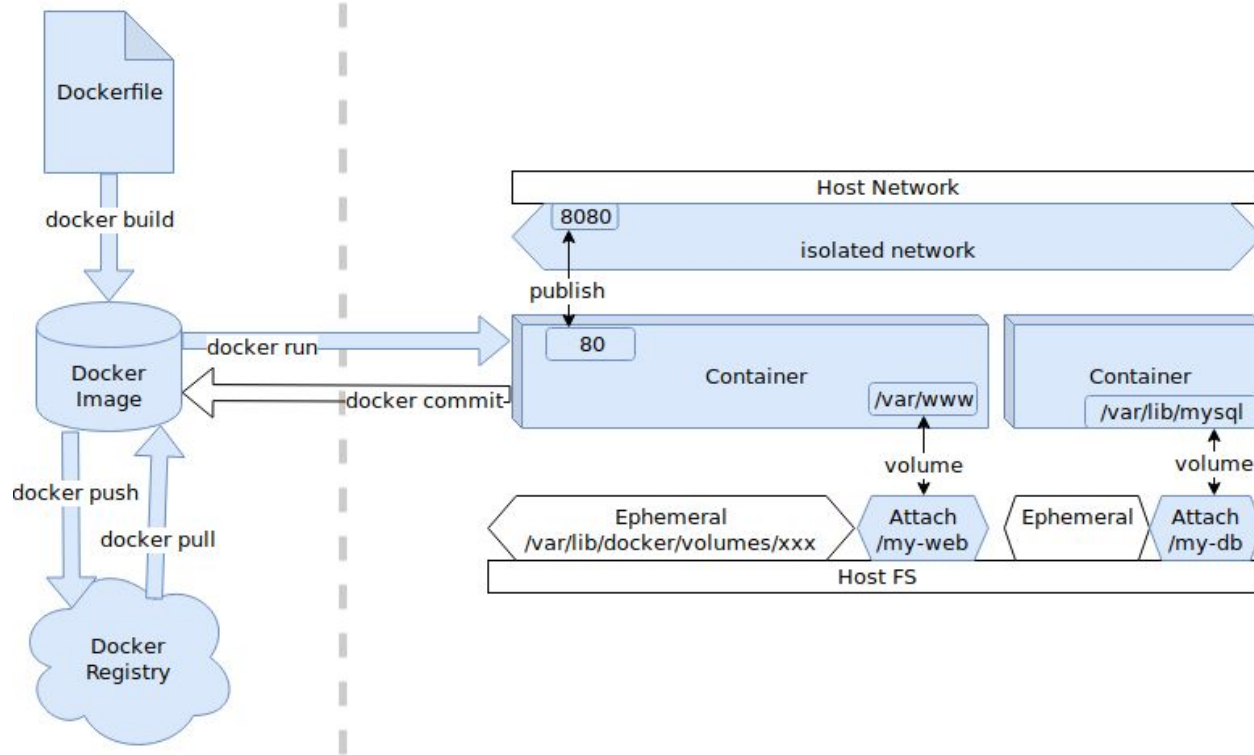




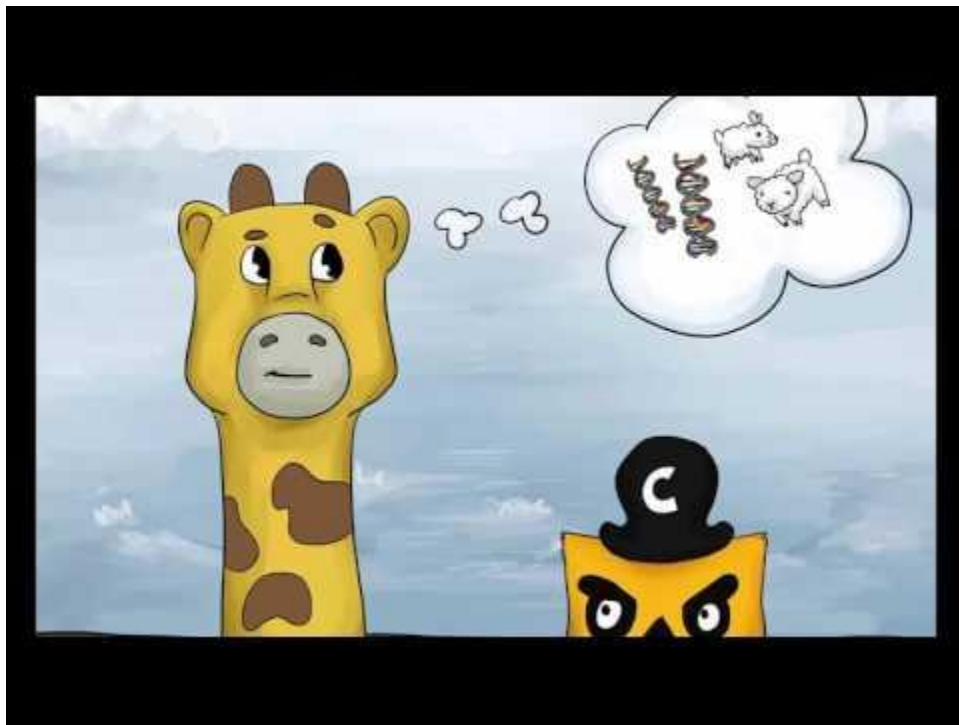
# Hand on: Create new Django application



# Hand on: Docker build



# Kubernetes Concept



The Illustrated Children's Guide to Kubernetes

# Compute



## Pod

เป็นหน่วยที่เล็กที่สุดในการจัดการด้วย Kubernetes เปรียบเสมือนเครื่องคอมพิวเตอร์ 1 เครื่อง โดยจะมีทรัพยากรเป็นของตัวเอง ซึ่งประกอบด้วย Container ตั้งแต่ 1 container ขึ้นไปทำงานอยู่ภายใน



## Deployment

เป็นส่วนที่ดูแล Life cycle ของ Pod โดย Kubernetes จะดูแลให้ การทำงานของ Pod เป็นไปตามที่ระบุไว้ใน Spec เช่น จำนวน Relicas, Health check, Restart

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

# Network



## Service

เป็นช่องทางในติดต่อผ่านเครือข่าย ซึ่ง Kubernetes จะสร้าง Cluster IP ที่สามารถติดต่อได้ทั้ง Cluster และสามารถทำการค้นพบได้ผ่าน DNS โดยการใส่ selector ผ่าน label



## Network Policy

สามารถกำหนดการเข้าถึง Service ด้วยการกำหนดกฎของเครือข่าย

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

# Storage



## Persistent Volume

เป็นพื้นที่จัดเก็บข้อมูลถาวร ที่สร้างขึ้นโดยอาจจะใช้ storage backend รูปแบบต่างๆ ซึ่งสร้างโดยผู้ดูแลระบบ หรือสร้างโดยอัตโนมัติผ่าน Persistent Volume Claim และ Storage Class



## Persistent Volume Claim

เป็นคำขอสร้าง พื้นที่จัดเก็บข้อมูล โดยระบบจะไปจับคู่กับ Persistent Volume ที่มีอยู่ หรือสร้างขึ้นมาใหม่



## Storage Class

ชนิดของพื้นที่จัดเก็บข้อมูล ซึ่งถูกใช้ในการสร้างพื้นที่จัดเก็บข้อมูลตามที่ ร้องขออัตโนมัติได้

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: standard
```

# Config



## ConfigMap

จัดเก็บข้อมูลแบบ Key - Value สำหรับข้อมูลที่ไม่ใหญ่มากนัก และสามารถอ่านได้จากหลาย Pod โดยสามารถนำไปใช้ผ่านการ Mount เป็นไฟล์ หรือส่งเป็น Environment Variable ใน Pod ได้



## Secret

ใช้จัดเก็บข้อมูลเช่นเดียวกับ ConfigMap แต่ภายใน Kubernetes จะมีการจัดการแบบพิเศษเพื่อเก็บรักษาข้อมูลที่เป็นความลับ

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: my-conf
data:
  my.ini: |
    [client]
    port = 3306
    socket = /var/run/mysqld/mysql.sock
```



# Running classic 3-tier Application





## Log In

Username or email

Password

[Forgot Password?](#)

Log In



GitHub



Twitter



Facebook



Google



Instagram

New user? [Register](#)

© 2020