**Docker introduction**

* เป็น open-source ที่ช่วยจำลองสภาพแวดล้อม ในการ run service หรือ server
* มีรูปแบบการทำงานแบบ Agile มีความยืนหยุ่น และรวดเร็วในการทำงานและส่งมอบงาน
* หลักการสร้าง Container คือเพื่อจัดการ Library ต่างๆ และยังมีเครื่องมือในการจัดการ Version Control เพื่อให้ง่ายต่อการจัดการ และแก้ไขปรับปรุงได้ง่าย
* Docker เลยเป็นที่รู้จักในวงกว้างและเริ่มเข้ามาใช้งานมากยิ่งขึ้น และยังติดตั้งเพื่อใช้งานได้หลากหลายระบบปฏิบัติการไม่ว่า Linux, Windows, Mac

Docker vs vmware

* Docker ใช้เวลาไม่วินาทีในการ Boot
* ไม่เปลือง memory , disk

Docker images (หน้า 26)

* Repository คือ ที่เก็บ image หรือ แหล่งที่มาของ image พร้อมชื่อ
* Tag คือ เอาไว้บอก version ถ้าไม่ระบุ version ในการ pull image มา image ที่ pull ลงมาจะเป็น latest คือล่าสุด
* Image id คือ ค่า hash สามารใช้ในการ ลบ image ได้ เหมือนกับ repo
* Create คือ ระยะเวลาในการสร้าง image
* Size คือ ขนาดของ image

\*\*\* Image id กับ repo คือ ชื่อของ image สามารถนำมาใช้ในการ ลบ ได้

Docker container

* เหตุผลที่ใช้ cmder คือ เราสามารถใช้คำสั่ง linux รวมถึง path directory จะเหมือนกัน linux เลย เพราะ container ที่เรารัน เราจะอิง base linux (หน้า 30)
* Check date เพิ่ม -e TZ=Asia/Bangkok เข้าไปเพื่อ set datetime
* ทดสอบย่อย (หน้า 48)

docker run --name mariadb-container -e MYSQL\_ROOT\_PASSWORD=12345678 -e MARIADB\_DATABASE=maria\_db -p 3307:3306 -d mariadb:latest

Docker Volumes

* $(pwd) คือการเรียกใช้ path directory จะเหมือนกับคำสั่ง pwd ของ linux แต่ที่ใช้แบบนี้เพราะเป็นการเรียกใช้ตัวแปร ไม่อย่างงั้นจะมองว่าเป็นค่า string ธรรมดา ยกตัวอย่าง echo $(pwd) (หน้า 55)
* ทดสอบย่อย (หน้า 58)

docker run --name postgres-container -v $(pwd)/database:/var/lib/postgresql/data -p 5432:5432 -e POSTGRES\_PASSWORD=12345678 -d postgres:latest

Docker Network

* ทดสอบย่อย (หน้า 71)

docker run --name pgadmin-container -p 8888:80 -e PGADMIN\_DEFAULT\_EMAIL=admin@example.com -e PGADMIN\_DEFAULT\_PASSWORD=admin -d dpage/pgadmin4

Dockerfile

* FROM คือ การเลือกใช้ image
* WORDIR คือ การเลือก directory ที่จะให้ให้ map folder แรกใน กรณีที่ copy หรือให้ทำงานจะทำที่นี่ก่อน
* COPY ตามด้วยไฟล์ คือ การ copy file หรือ folder เข้าไปเพื่อ run build

การที่ต้อง copy file .json เข้าไปก่อนเพราะมีการ แก้ไขน้อย และสามารถเรียกใช้ package หรือ lib ที่อยู่ข้างในได้ เนื่องจากมีการ เก็บ cache จึงทำให้ลดขั้นตอนการ download package น้อยลง

อาจจะไม่ต้องพูด ( แต่ถ้าเอาไปไว้หลัง เมื่อ run npm install จะทำการ download package ใหม่ทั้งหมด จึงทำให้ใช้เวลาในการ Run build นานขึ้น )

* RUN คือ คำสั่งที่ใช้ในการ run คำสั่งต่างๆ ตามโจทย์จะเป็นการ run npm install
* COPY . . คือการ copy ทั้งหมดที่อยู่ใน directory นี้ เข้าไปใน directory ปลายทางตาม wordir ที่ได้กำหนดไว้
* RUN สุดท้ายคือการ npm run build
* Layers การเรียงการทำงานเป็น Layer นั้นมีผลต่อ ระยะเวลา กับ ขนาดในการ build เพราะ ที่เอา copy package.json กัย package-lock.json ขึ้นก่อนเนื่องจาก lib จะมีการเปลี่ยนแปลงที่น้อยมาก ดังนั้นจะสามารถ ดึงข้อมูลจาก cache มาใช้ได้เลยใช้เวลาน้อย แต่ถ้าเอา step build project ก่อนตัว package หรือ lib จะต้องทำการ load ใหม่จึงทำให้ใช้เวลาใน build นานขึ้น หรืออาจจะทำให้ขนาดใหญ่ขึ้น
* อีกอย่างนึงการใช้คำสั่ง Run หลายบรรทัดก็จะทำให้ ขนาดของ image ใหญ่ขึ้นอีกด้วย

ตัวอย่าง 1 ตัวอย่างนี้จะทำให้ ขนาดเล็กกว่า ( single run )

RUN apk update && apk upgrade && \ apk add --no-cache \ python3 \ nodejs \ git

ตัวอย่างที่ 2 ตัวอย่างนี้จะทำให้ ขนาด image ใหญ่ขึ้น ( multiple run )

RUN apk update

RUN apk upgrade

RUN apk add python3

RUN apk add nodejs

RUN apk add git

* ดังนั้นการ กำหนด Layer ในการทำงานนั้นจึงมีผลมากในการ run build

**Dockerfile**

**1 stage multi stage**

#STAGE 1 build Angular

FROM node:18.19.0-alpine AS builder

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm install

COPY . .

RUN npm run build

#STAGE 2 deploy nginx

FROM nginx:1.23.3-alpine

#COPY nginx.conf /etc/nginx/nginx.conf

COPY --from=builder /app/dist/<name\_project>/browser /usr/share/nginx/html/

#STAGE 1 build Angular

FROM node:18.19.0-alpine

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm install

COPY . .

RUN npm run build

Docker-compose

* docker compose run Dockerfile angular

services:

angular:

container\_name: angular\_multi-stage

image: multi-stage:1.0.0

build:

dockerfile: Dockerfile

context: .

environment:

- TZ=Asia/Bangkok

ports:

- 8801:80

* create docker-compose.yml install mysqldb

version: '3.9'

services:

db:

image: mysql:latest

container\_name: mysqldb

environment:

MYSQL\_ROOT\_PASSWORD: rootpassword

MYSQL\_DATABASE: wordpressdb

MYSQL\_USER: myuser

MYSQL\_PASSWORD: mypassword

volumes:

- ./db\_data:/var/lib/mysql

ports:

- "3306:3306"

networks:

- web\_network

networks:

web\_network:

name: wlamp

driver: bridge

* docker-compose.yml install phpMyAdmin

phpmyadmin:

container\_name: myadmin

image: phpmyadmin/phpmyadmin

restart: always

environment:

- PMA\_HOST=db

- PMA\_USER=root

- PMA\_PASSWORD=rootpassword

ports:

- 7770:80

networks:

- web\_network

depends\_on:

- db

* docker-compose.yml install WordPress

wordpress:

depends\_on:

- db

image: wordpress

restart: always

networks:

- web\_network

volumes:

- ./wordpress:/var/www/html

ports:

- 8800:80

environment:

- WORDPRESS\_DB\_HOST=db:3306

- WORDPRESS\_DB\_USER=myuser

- WORDPRESS\_DB\_PASSWORD=mypassword

- WORDPRESS\_DB\_NAME=wordpressdb