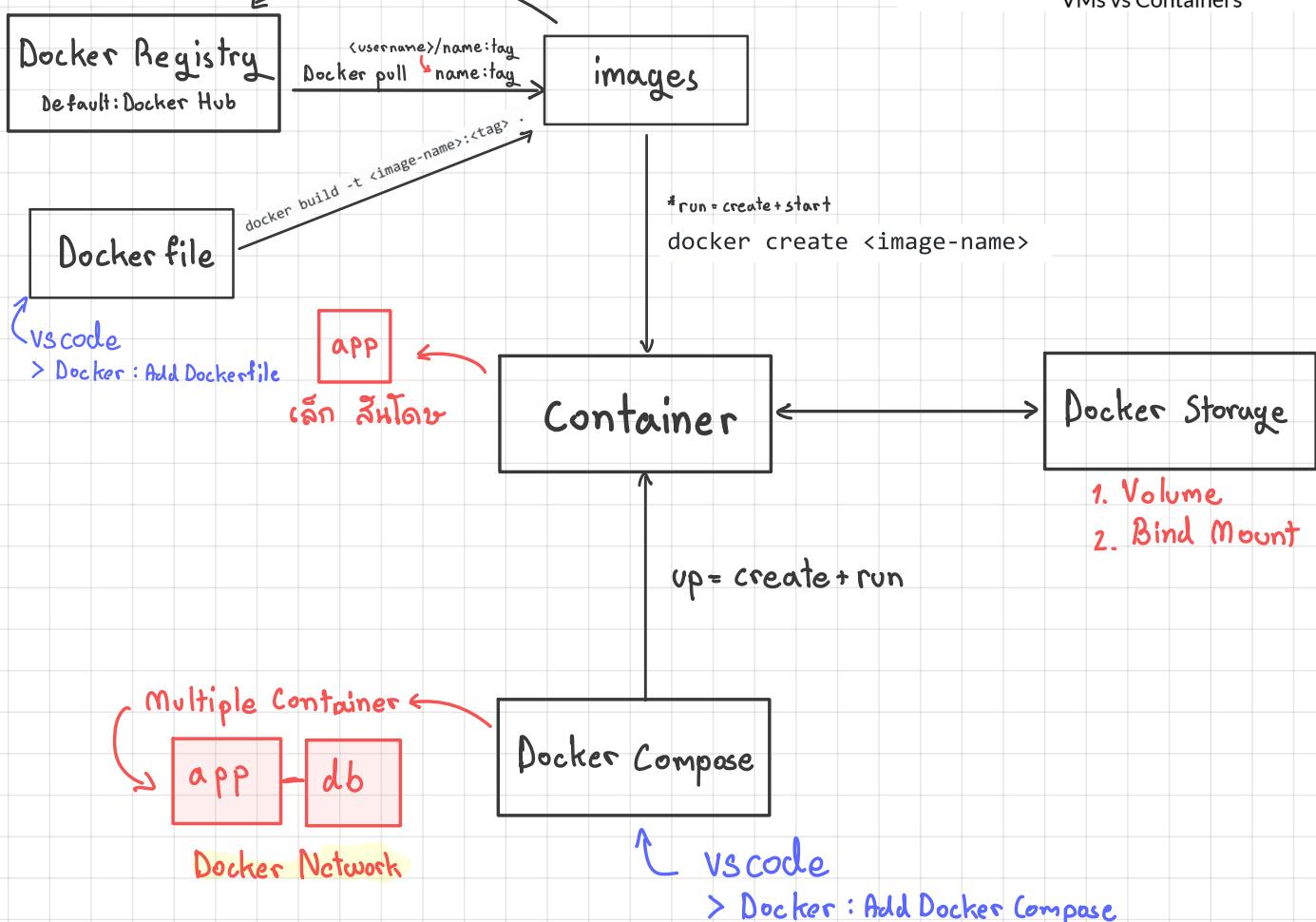


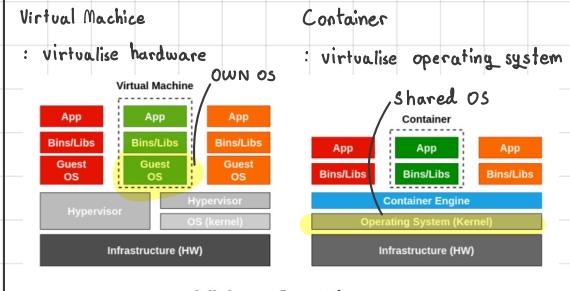
Docker → Create Container  
RUN Container

## 1. Create Container

`docker tag <image>:tag <username>/name:<tag>`  
`docker push <username>/name:<tag>`



## Container ជីវិតណា?



VMs vs Containers

--name abc ชื่อ container -d = detach Mode  
 -p 3333:3000, map port Host A → B ของ Container

### Docker Command

คำสั่ง	คำอธิบาย
docker build -t <image-name>:<tag>	สร้าง Docker image จาก Dockerfile ที่อยู่ใน directory ปัจจุบัน
docker create <image-name>	สร้าง container จาก image แต่ยังไม่เริ่มรัน
docker start <container-name>	เริ่ม container ที่หยุดอยู่
docker run <options> <image-name>	สร้างและรัน container ใหม่จาก image ที่กำหนด
docker ps	แสดงสถานะของ containers ที่รันอยู่ในตอนนี้, ซึ่ง
docker images	แสดงรายการ images ที่มีอยู่ในเครื่อง, รายวะ เว็บไซด์ size
docker logs <container-name>	ดู logs ของ container ที่กำหนด
docker stop <container-name>	หยุดการทำงานของ container ที่กำหนด
docker rm <container-name> --force	ลบ container ที่หยุดการทำงานแล้ว
docker rmi <image-name>	ลบ image ที่กำหนด
docker ps -a รวมไว้รัน	แสดงรายการ containers ทั้งหมด (รันอยู่หรือไม่)
docker exec -it <container-name> <command>	รันคำสั่งใน container ที่กำหนดในโหมด interactive

docker tag <image>:<tag> <username>/name:<tag>  
 docker push <username>/name:<tag>

### Docker Compose Command

คำสั่ง	คำอธิบาย
docker-compose up	เริ่มต้น services ทั้งหมดตามที่ระบุใน docker-compose.yml (ปกติจะรันในโหมด foreground)
docker-compose up -d	เริ่มต้น services ในโหมด background (detached)
docker-compose down	หยุดและลบ containers, networks, volumes ตามที่ระบุใน docker-compose.yml
docker-compose build	สร้างหรือ rebuild images ตามที่ระบุใน docker-compose.yml
docker-compose logs	ดู logs ของ services ทั้งหมด
docker-compose ps	แสดงสถานะของ containers ที่รันอยู่จาก docker-compose.yml
docker-compose exec <service-name> <command>	รันคำสั่งใน container ของ service ที่ระบุจาก docker-compose.yml
docker-compose run <service-name> <command>	รันคำสั่งใน container ใหม่ของ service ที่ระบุโดยไม่ต้องเริ่ม services ทั้งหมด
docker-compose restart	รีสตาร์ท services ทั้งหมดที่กำหนดใน docker-compose.yml
docker-compose scale <service-name>=<num>	กำหนดจำนวน instances ของ service ที่ระบุ (เฉพาะในบางเวอร์ชัน)
docker-compose up --build	สร้างและเริ่มต้น services ทั้งหมดพร้อม rebuild images

### Dockerfile

คำสั่ง	คำอธิบาย
FROM <image-name>	กำหนด base image ที่จะใช้สร้าง image ใหม่
WORKDIR <path>	กำหนด directory ที่จะใช้เป็น working directory ใน container
COPY <source> <destination>	คัดลอกไฟล์หรือโฟลเดอร์จากเครื่อง host ไปยัง container
RUN <command>	รันคำสั่งใน container ขณะสร้าง image
CMD ["command"]	คำสั่งที่จะรันเมื่อ container เริ่มทำงาน
EXPOSE <port>	เปิดพอร์ตที่ container จะรับการเชื่อมต่อจากภายนอก app Host

## Dockerfile

### 1. Basic Syntax

- FROM: ระบุ base image ที่ใช้

```
FROM python:3.9-alpine
```

Base image , Service

- WORKDIR: เปลี่ยน directory ภายใน container

```
WORKDIR /app
```

- COPY: คัดลอกไฟล์จากเครื่อง host ไปยัง container

```
COPY . /app
```

- RUN: รันคำสั่งใน container ตอนสร้าง image

```
RUN pip install -r requirements.txt
```

Command

- CMD: กำหนดคำสั่งที่จะรันเมื่อ container เริ่มทำงาน

```
CMD ["python", "app.py"]
```

→ python app.py

- EXPOSE: เปิดพอร์ตที่ container จะรับการเชื่อมต่อ

```
EXPOSE 8000
```

### 2. Example of a Simple Dockerfile

```
# Set the base image
FROM python:3.11-alpine

# Set the working directory
WORKDIR /app

# Copy the local directory content to the container
COPY . /app

# Install dependencies
RUN pip install -r requirements.txt

# Expose the application port
EXPOSE 8000

# Run the application
CMD ["python", "app.py"]
```

ENV NODE\_ENV = production ← ตอนนี้ร่าง

docker build --build-arg NODE\_ENV=production -t f3:latest .

(หรือก็จะนําไป compose)

## Docker Compose Configuration (docker-compose.yml)

### 1. Basic Example

```
version: "3"
services:
  web:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - .:/app
    environment:
      - ENV=production
```

try. fast API (python)

Compose:

```
services:
  myapp:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "9000:7000"
```

Dockerfile:

```
# ใช้ Base Image เป็น Python 3.12 เวอร์ชัน Slim (ขนาดเล็ก เหมาะสำหรับการ Deploy)
FROM python:3.12.0-slim

# กำหนด Directory หลักภายใน Container เป็น /app
WORKDIR /app/

# คัดลอกไฟล์ requirements.txt จากเครื่อง Host ไปยัง Container
COPY requirements.txt /app/
```

```
# ติดตั้ง Dependencies ตามที่ระบุไว้ใน requirements.txt โดยไม่เก็บ Cache เพื่อลดขนาด Image
RUN pip install --no-cache-dir -r requirements.txt
```

```
# แสดงรายการไฟล์ใน Directory /app (สำหรับ Debugging ตรวจสอบไฟล์ที่ถูกคัดลอกมา)
RUN ls
```

```
# คัดลอกไฟล์เดอร์ app ทั้งหมดจากเครื่อง Host ไปยัง Directory มัจฉะนั้นของ Container (/app)
COPY app/ .
```

```
# เปิด Port 7000 เพื่อให้ Container สามารถรับการเชื่อมต่อจากภายนอกได้
EXPOSE 7000
```

```
# คำสั่งเริ่มต้นเมื่อ Container ทำงาน: รัน Uvicorn เซิร์ฟเวอร์ด้วยพารามิเตอร์ที่ระบุ
```

```
CMD ["uvicorn", "--host", "0.0.0.0", "--port", "7000", "--reload", "main:app"]
```

### 2. Multi-Service Example

```
version: "3"
services:
  web:
    build: ./web
    ports:
      - "8000:8000"
  db:
    image: postgres:alpine
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: appdb
```

### 3. Using Volumes and Networks

```
version: "3"
services:
  web:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - web-data:/app
    networks:
      - app-network

  db:
    image: postgres:alpine
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: appdb
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - app-network

  volumes:
    web-data: ← ชื่อ
    db-data:

networks:
  app-network:
    driver: bridge
```

user : app

Port:Map กับ dockerfile

สร้าง volume

(bind ≠ volume)

# Docker Storage Cheat Sheet

ประเภท	คำอธิบาย	คำสั่ง
Bind Mounts	เชื่อมโยงไฟล์หรือโฟลเดอร์จากเครื่อง host กับ container	docker run -v /host/path:/container/path my-image
Volumes	ใช้สำหรับเก็บข้อมูลที่ไม่หายไปเมื่อ container หยุดหรือถูกลบ	docker run -v volume_name:/container/path my-image
Networks	เชื่อมต่อ container ผ่านเครือข่ายที่กำหนด เช่น bridge, overlay	docker network create network_name

## รายละเอียดเพิ่มเติม

## 1. Docker Bind Mounts

- ໃຊ້ແໜ່ງໄຟລ໌/ໂຟລ໌ເດວຍຈາກເຄື່ອງ host ໄປຢັ້ງ container
  - ຂອບເຖິງ: ແກ້ໄຂໄຟລ໌ໃນ container ທີ່ຮູ້ host ຈະເຫັນຜລທັນທີ
  - ຂອບເສີຍ: ຂອບເສີຍ path ມີນ host ເປົ້າຢືນ ອາຈສັງຜລຕ່ວງການທີ່ຖືກຕະກຳ
  - **ຄໍາສັ່ງ:**

```
docker run -v /path/to/host/directory:/path/in/container my-image
```

## 2. Docker Volumes

- ใช้จัดการข้อมูลที่ต้องการเก็บระหว่าง container หรือเพื่อให้ข้อมูลไม่หายไปเมื่อ container หยุด/ถูกลบ
  - ข้อดี: Docker จัดการข้อมูลให้, ข้อมูลไม่หายเมื่อ container หยุด
  - ข้อเสีย: เข้าถึงข้อมูลจากเครื่อง host ได้ยาก
  - คำสั่ง:

- สร้าง volume:

```
docker volume create my_volume
```

- ໃຊ້ volume ໃນ container:

```
docker run -v my volume:/path/in/container my-image
```

- ลง volume:

```
docker volume rm my_volume
```

### 3. Docker Networks

- ໃນເຊື່ອມຕົວ container ກັບ network ເຊັ່ນ bridge, host, overlay
  - ຂໍາຍໃຫ້ containers ສາມາຄົດຕິດຕ່ອງກັນໄດ້
  - ຄ໏າສັ່ງ:

- #### ○ สร้าง network:

```
docker network create my_network
```

- เชื่อมต่อ container กับ network:

```
docker run --network my_network my-image
```

- คุณ network:

```
docker network rm my_network
```

