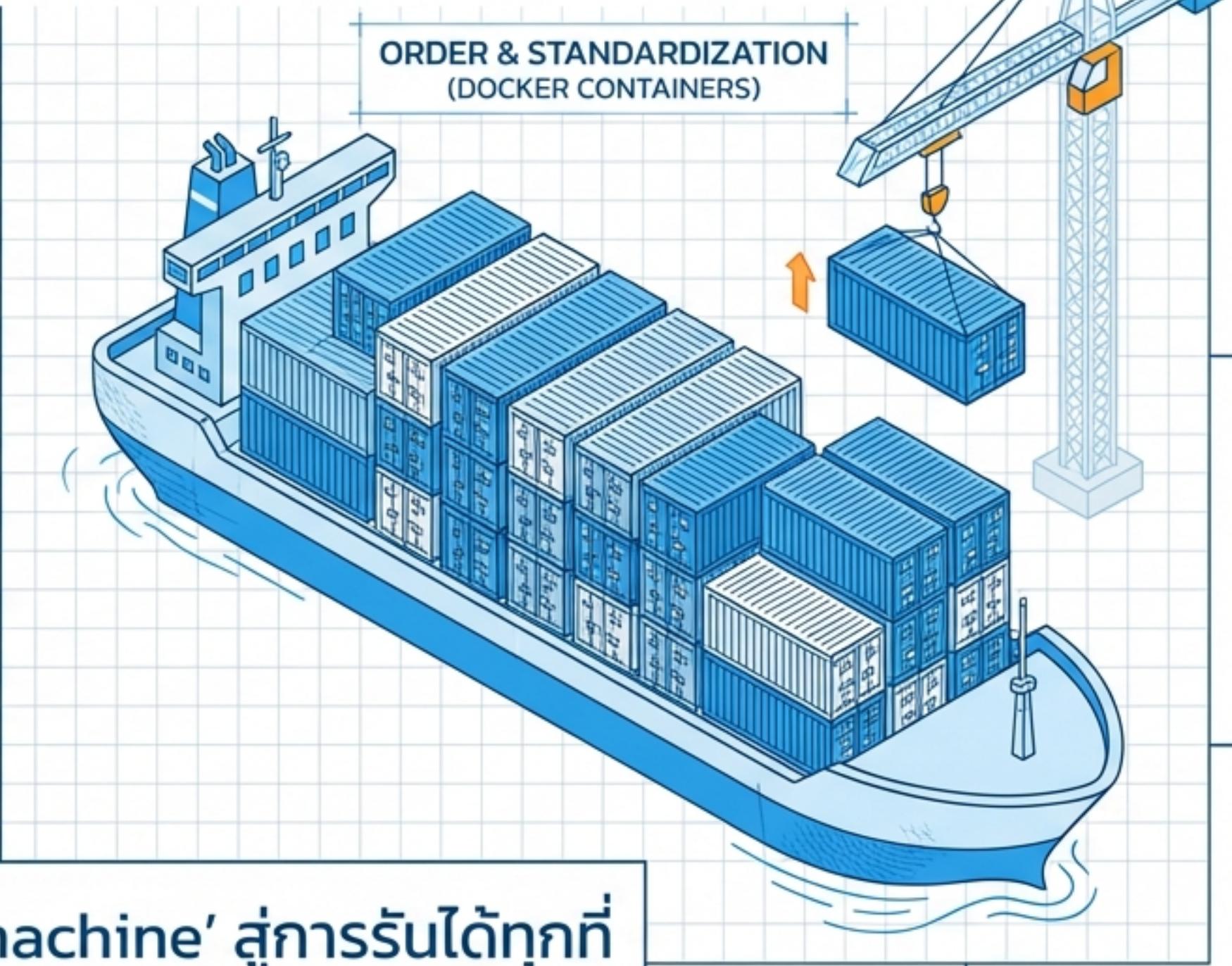
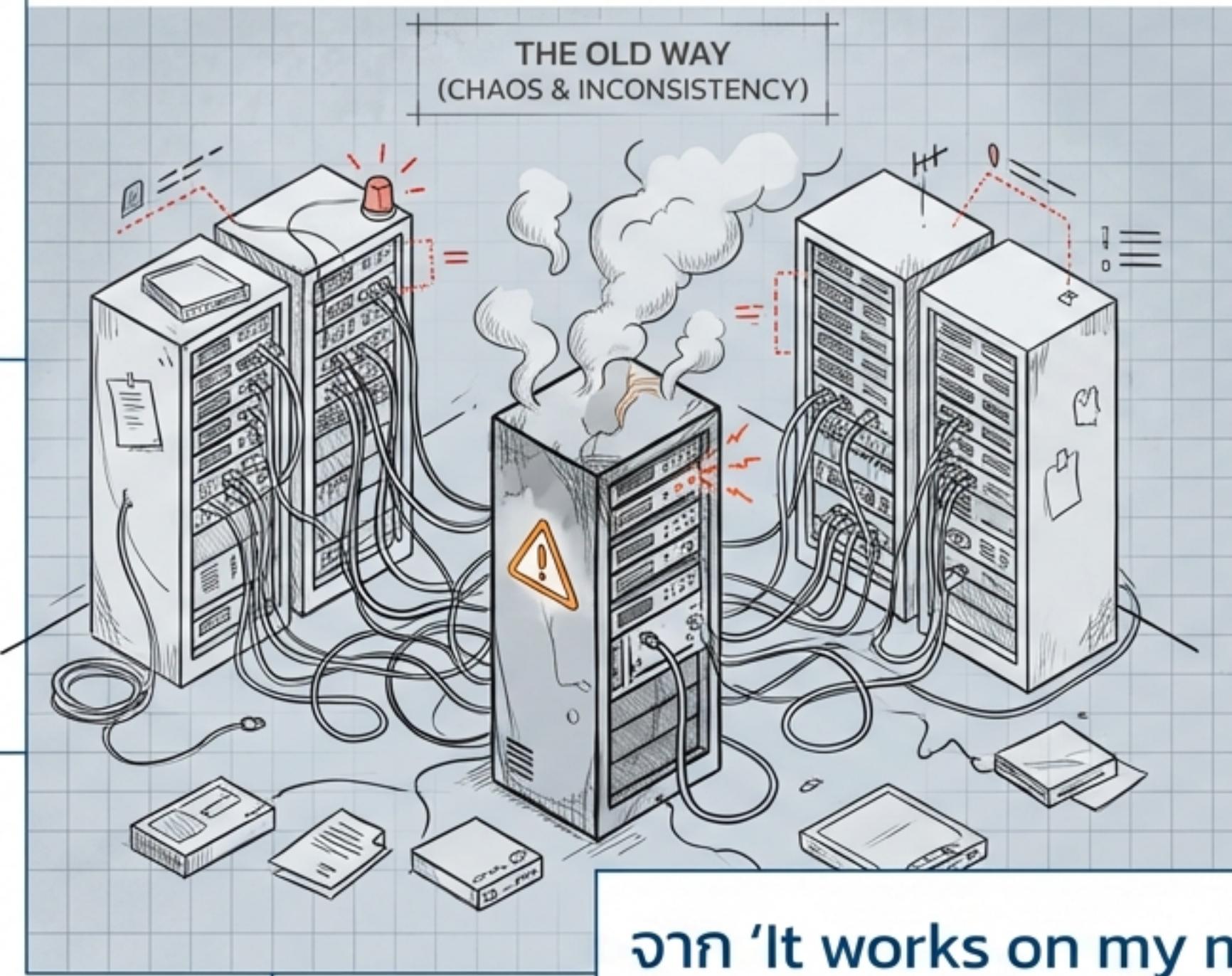


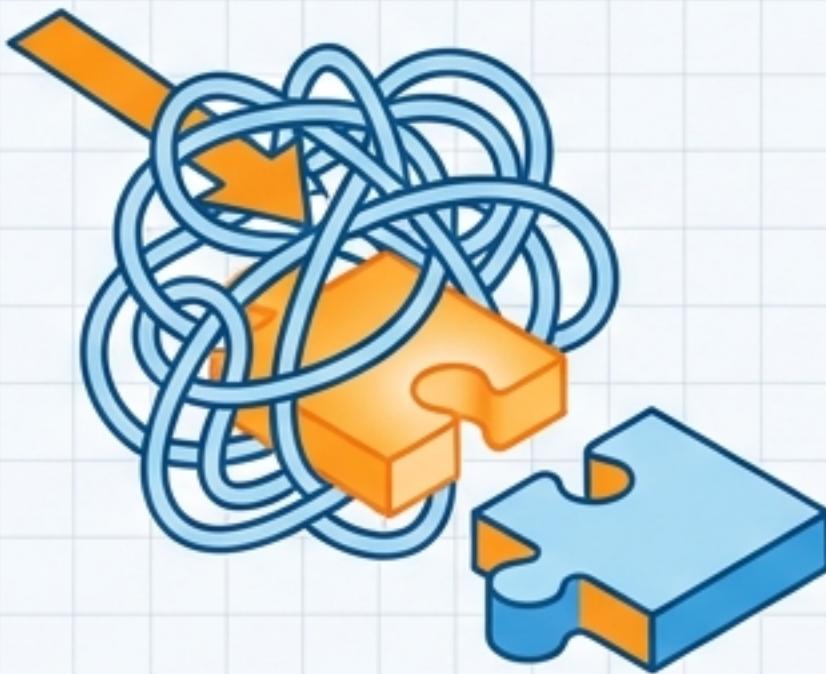
Docker & Docker Compose Essentials

สรุปคร่าวๆในสไลด์เดียว



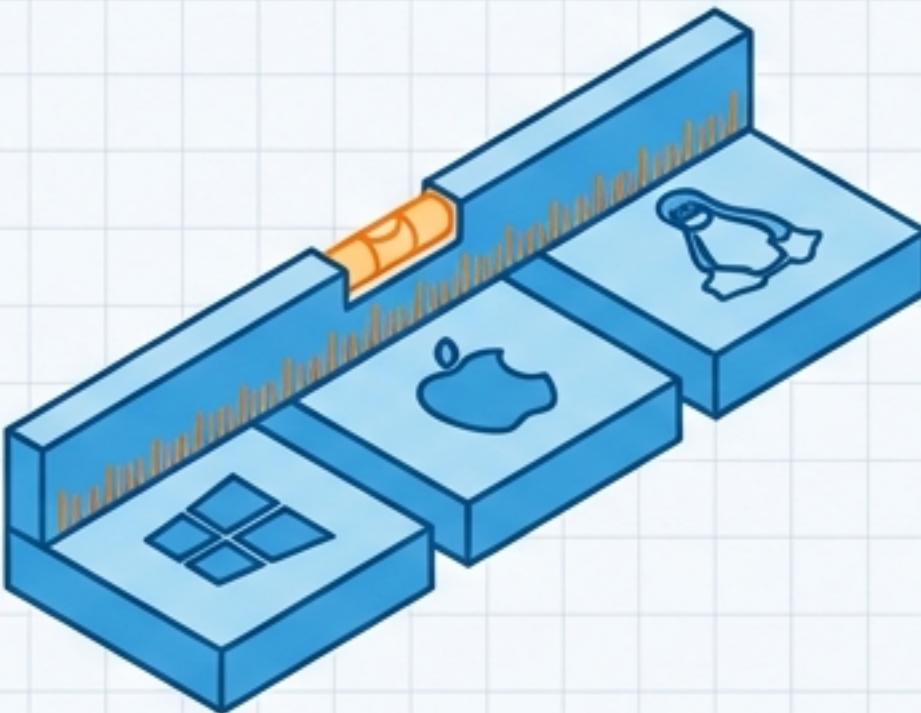
จาก 'It works on my machine' สู่การรันได้ทุกที่

ทำไมต้อง Docker?



Dependency Hell

แก้ปัญหา Library ตีกัน
หรือ version ไม่ตรงกัน



Standardization

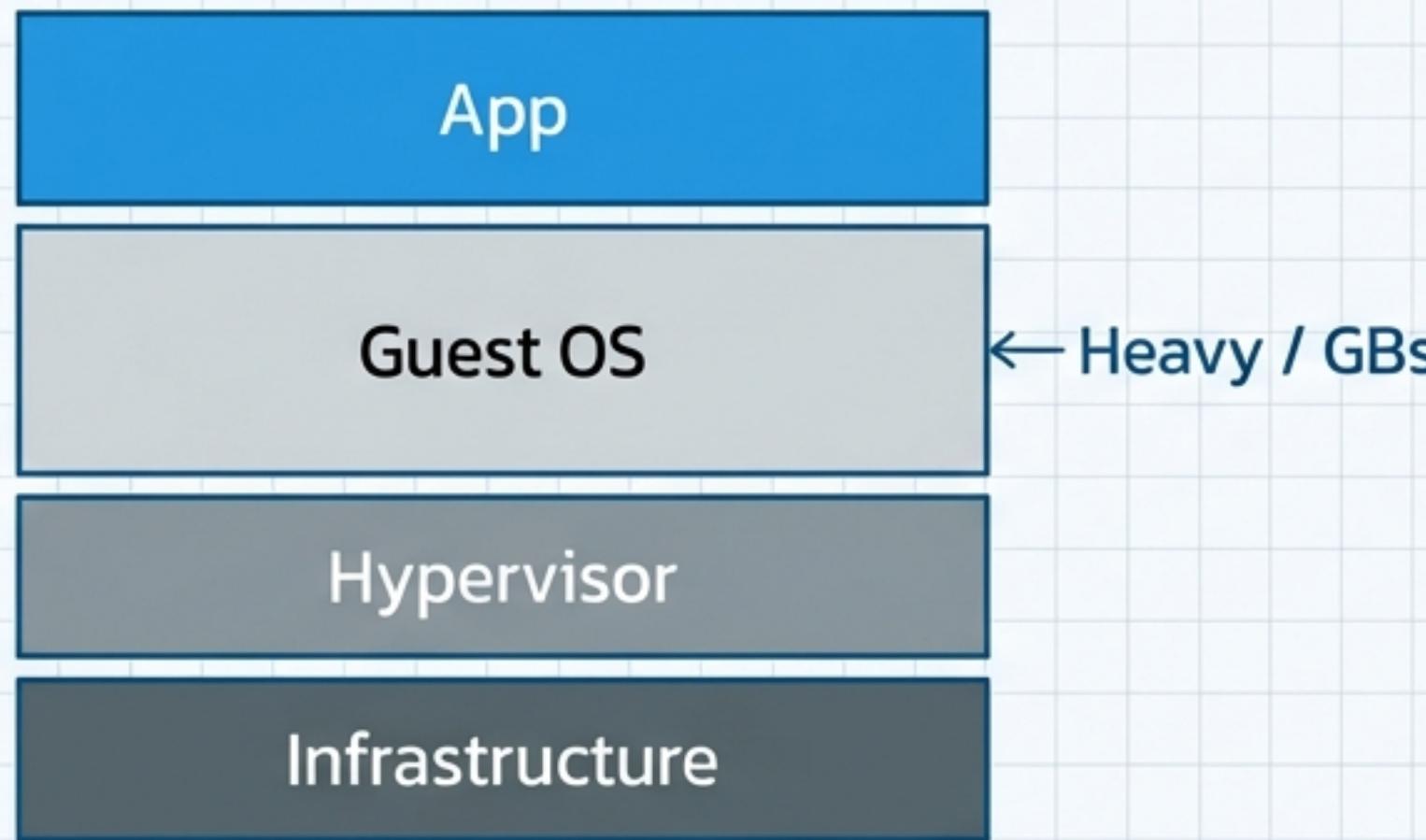
ไม่ต้องปวดหัวกับการลงโปรแกรม
ที่ต่างกันใน Windows/Mac



Isolation

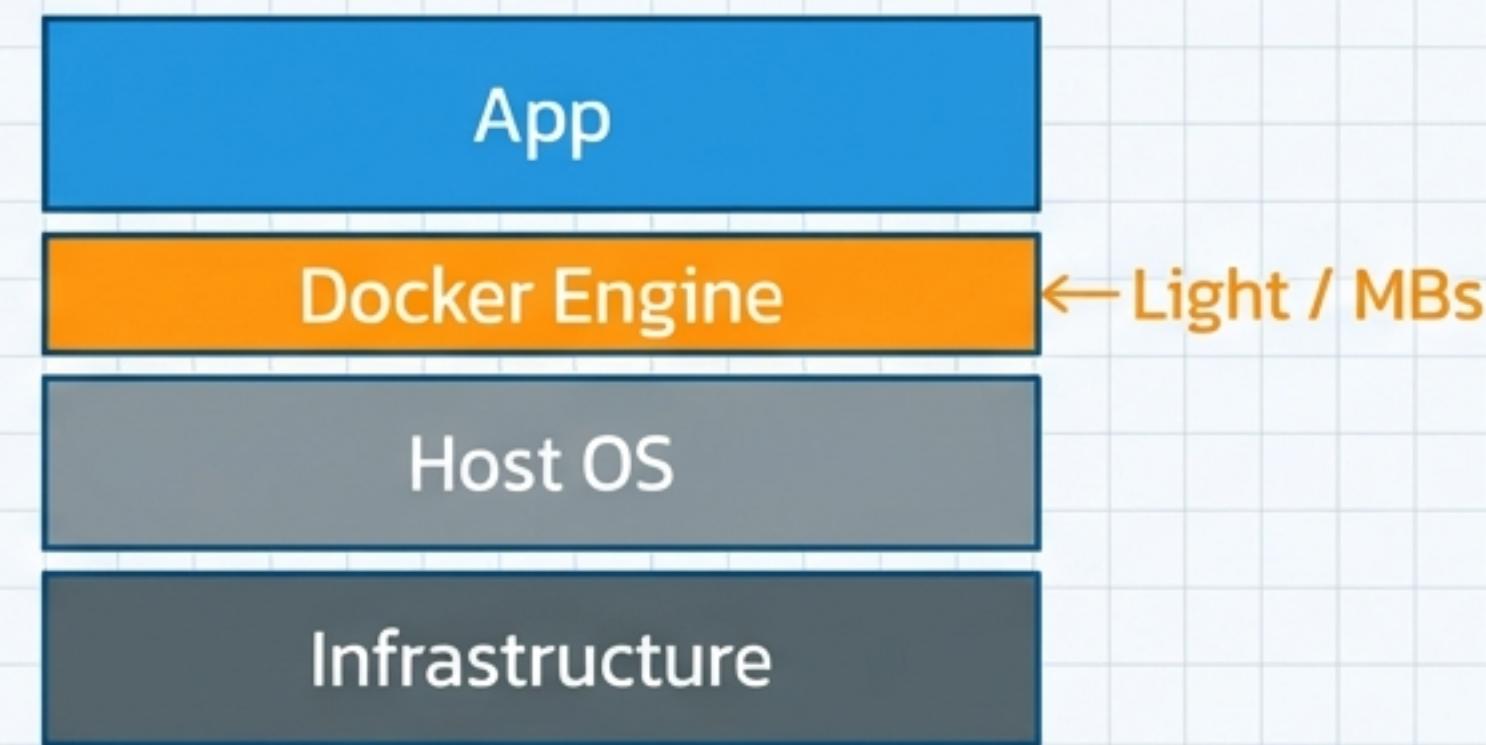
แยก Environment ของแต่ละ
Service ออกจากกันชัดเจน

Docker ต่างจาก Virtual Machine (VM) อย่างไร



Virtual Machine

จำลอง Hardware + OS เต็มรูปแบบ
ช้า (Boot เป็นนาที)
ขนาดเป็น GB



Docker Container

จำลองแค่ Application Layer
เร็ว (Boot เป็น millisecond)
ขนาดเป็น MB

2 คำศัพท์หัวใจสำคัญ: Image และ Container

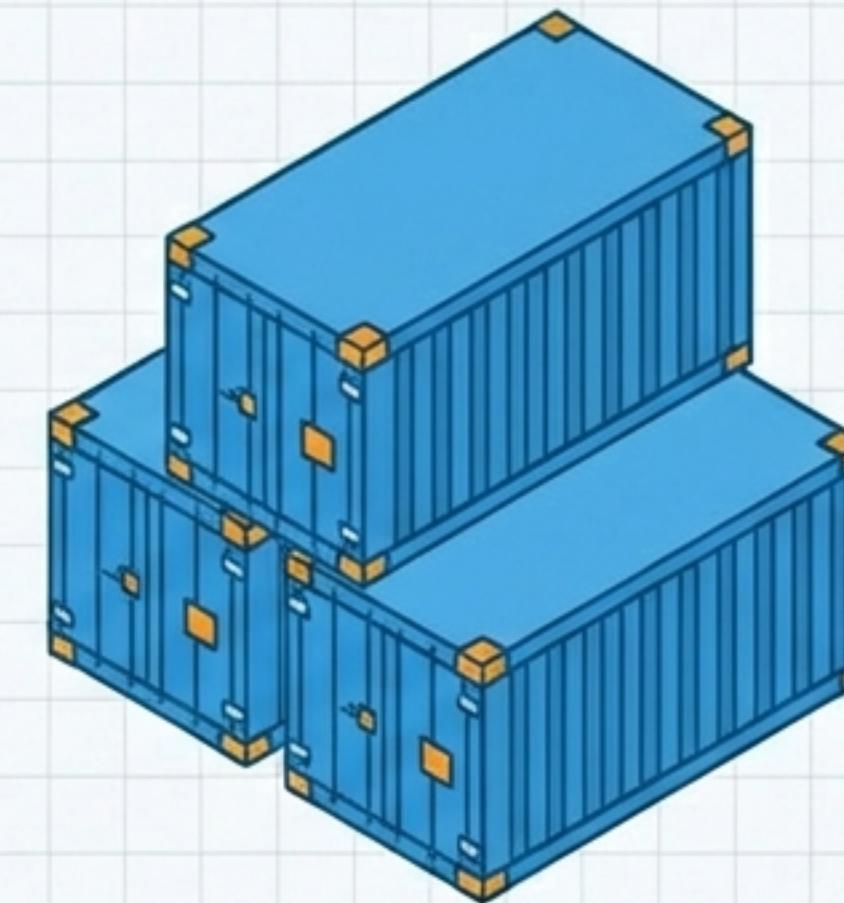
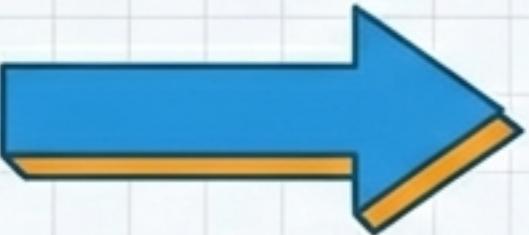
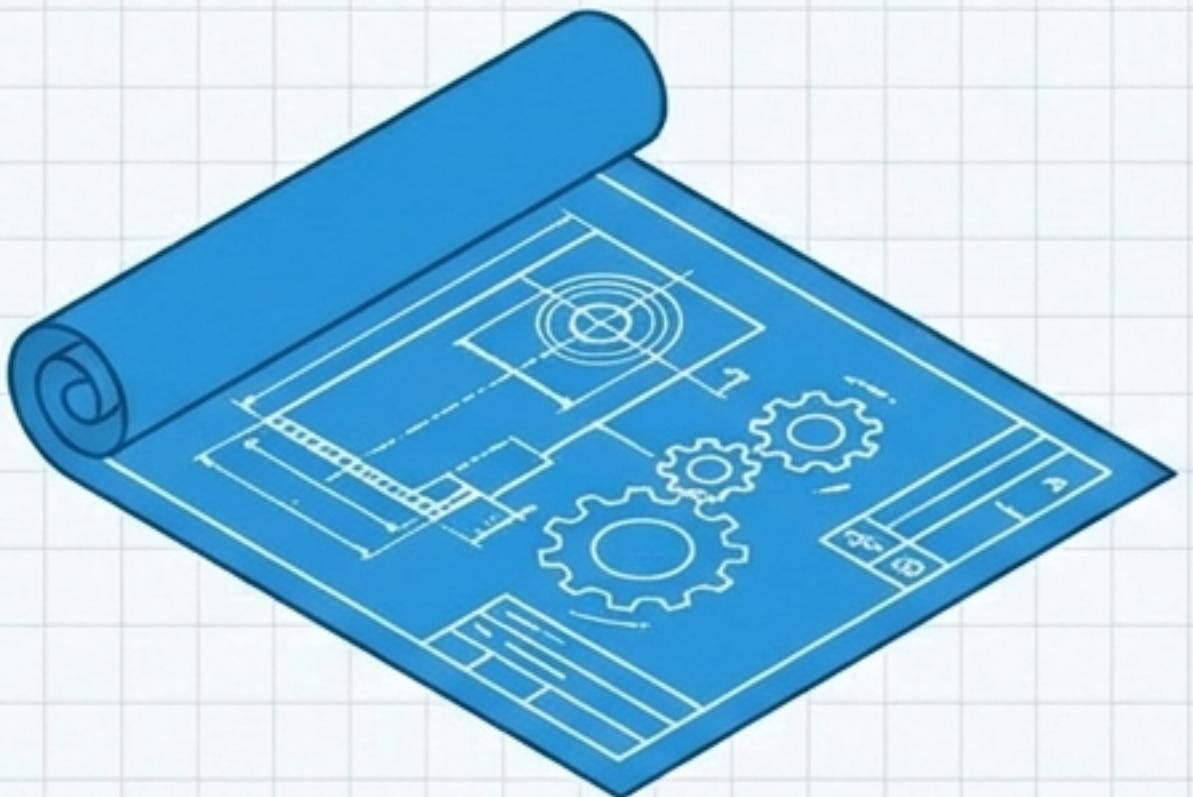
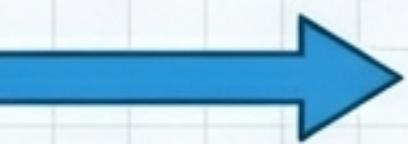


Image (The Blueprint)

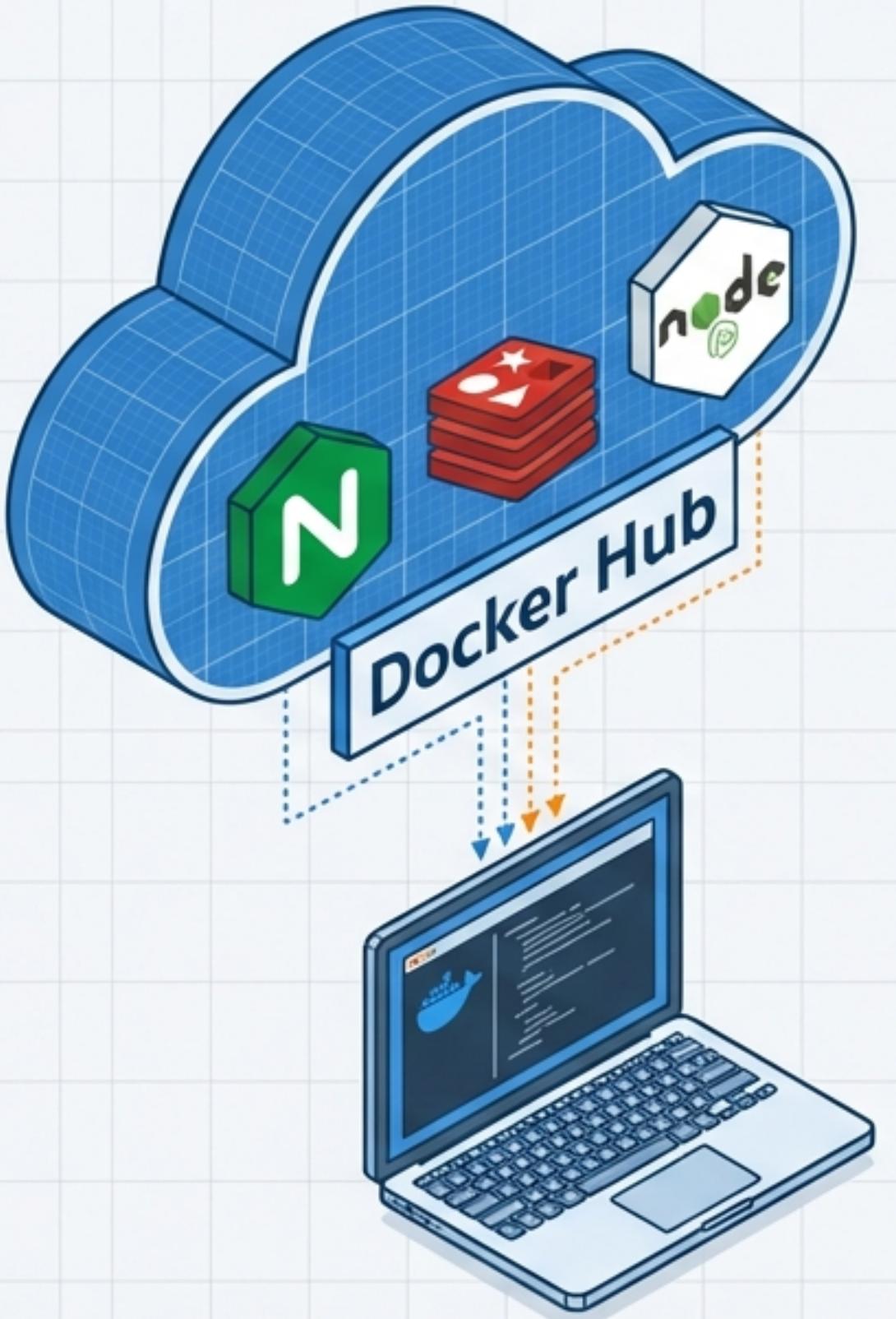
ตัวต้นแบบ (Artifact) ที่รวม
Code + Libraries + Config ไว้
แก้ไขไม่ได้ (Read-only)
เปรียบเสมือนไฟล์ติดตั้ง



Container (The Building)

สิ่งที่รันขึ้นมาจาก Image
เปรียบเสมือนโปรแกรมที่กำลังทำงาน
เราสามารถสร้างหลาย Container จาก 1 Image ได้

Image มาจากไหน?



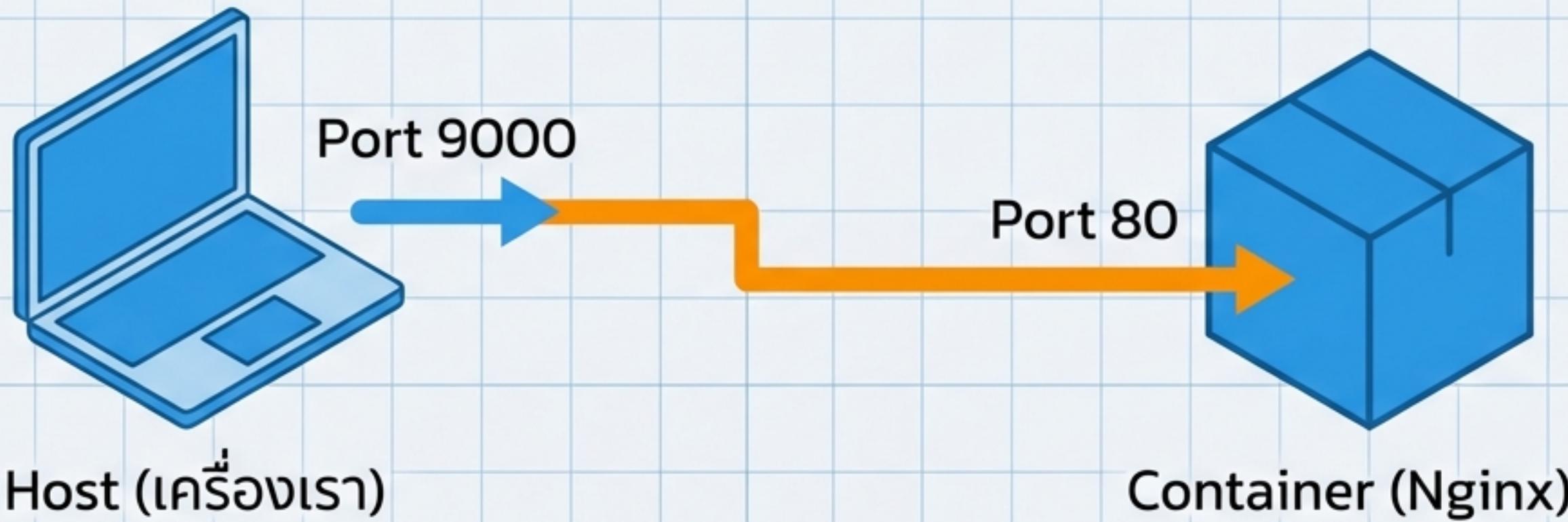
- **Docker Hub**
Public Registry ที่ใหญ่ที่สุด (Like GitHub for Images)
มี Official Images ที่ดูแลโดยผู้พัฒนา
- **Private Registry**
ที่เก็บ Image ส่วนตัวของบริษัท (เช่น AWS ECR, Google CR) เพื่อความปลอดภัย
- **Versioning (Tags)**
Image จะมี 'Tag' ระบุเวอร์ชัน เช่น
`redis:4.0`, `node:19-alpine`, หรือ `latest`

คำสั่งพื้นฐานที่ต้องรู้

TERMINAL

- > **docker pull [image]** ดาวน์โหลด Image จาก Registry ลงเครื่อง
- > **docker run [image]** สร้างและเริ่ม Container ใหม่จาก Image
- > **docker ps** ดูรายชื่อ Container ที่กำลังทำงานอยู่ (Active)
- > **docker stop [id]** หยุดการทำงานของ Container
- > **docker ps -a** ดู Container กั้งหมด (รวมที่หยุดไปแล้ว)

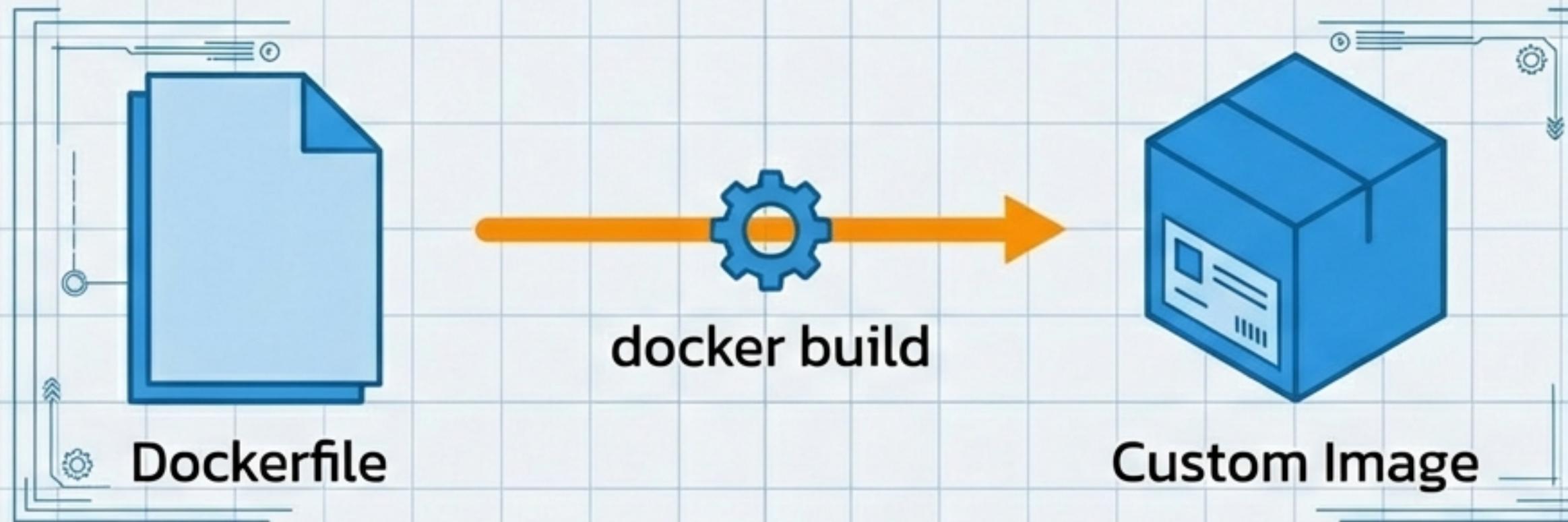
Port Mapping (การเชื่อมต่อ กับ โลกภายนอก)



```
docker run -p 9000:80 nginx
```

- **-p 9000:80**: เชื่อม Port 9000 ของเครื่องเรา (Host) ไปยัง Port 80 ของ Container
- **ผลลัพธ์**: เข้าเว็บผ่าน localhost:9000 ได้เลย

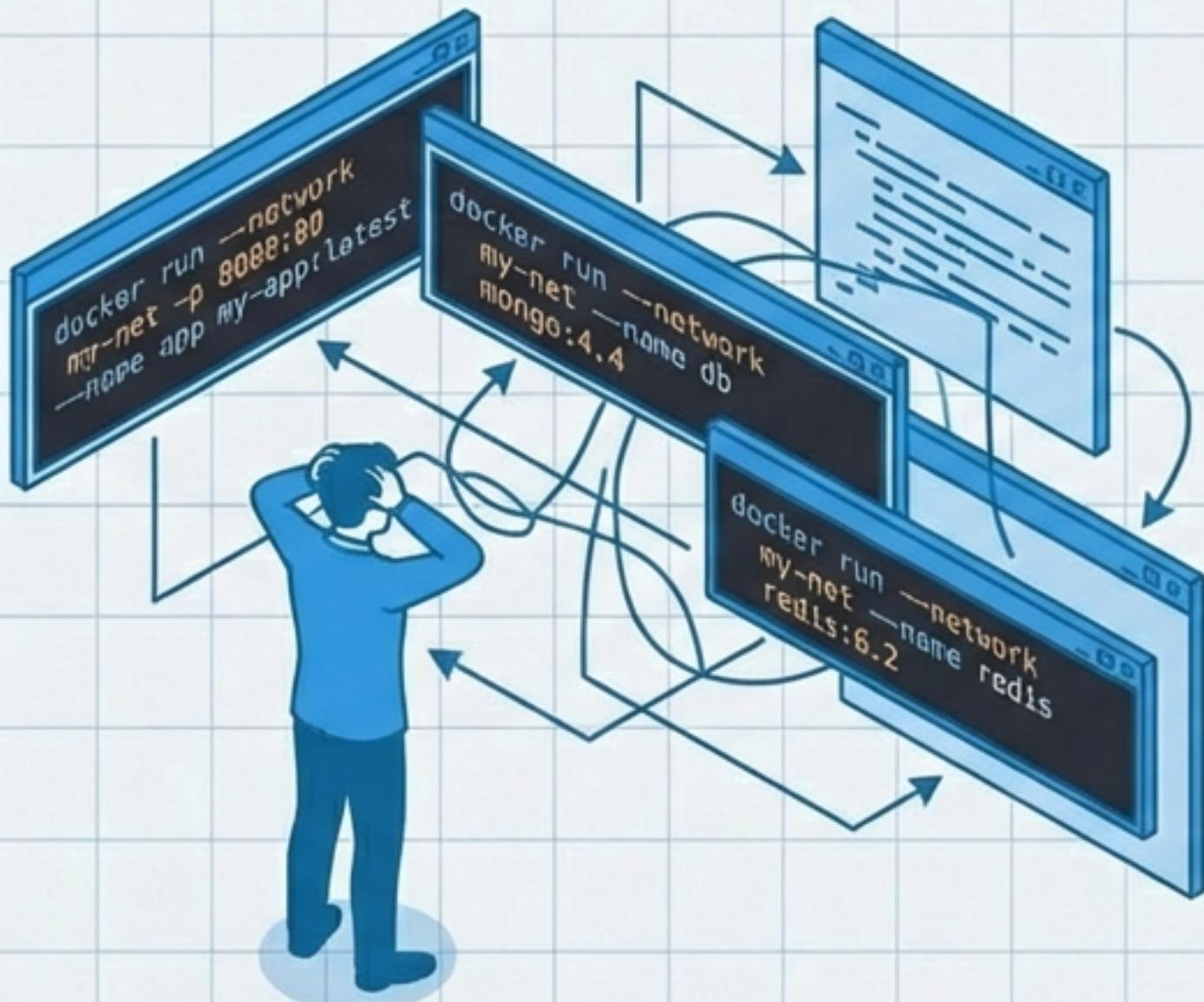
สร้าง Image ของตัวเองด้วย Dockerfile



Dockerfile : คู่มือการสร้าง Image

- **FROM** : เลือก Base Image (เช่น node:19-alpine)
- **COPY** : ก็อปปี้ Code จากเครื่องเราเข้าไปใน Image
- **RUN** : รันคำสั่งติดตั้ง (เช่น npm install) ตอน Build
- **CMD** : คำสั่งเริ่มต้นเมื่อ Container ถูกรัน (เช่น node server.js)

ปัญหาของการรันหลาย Container



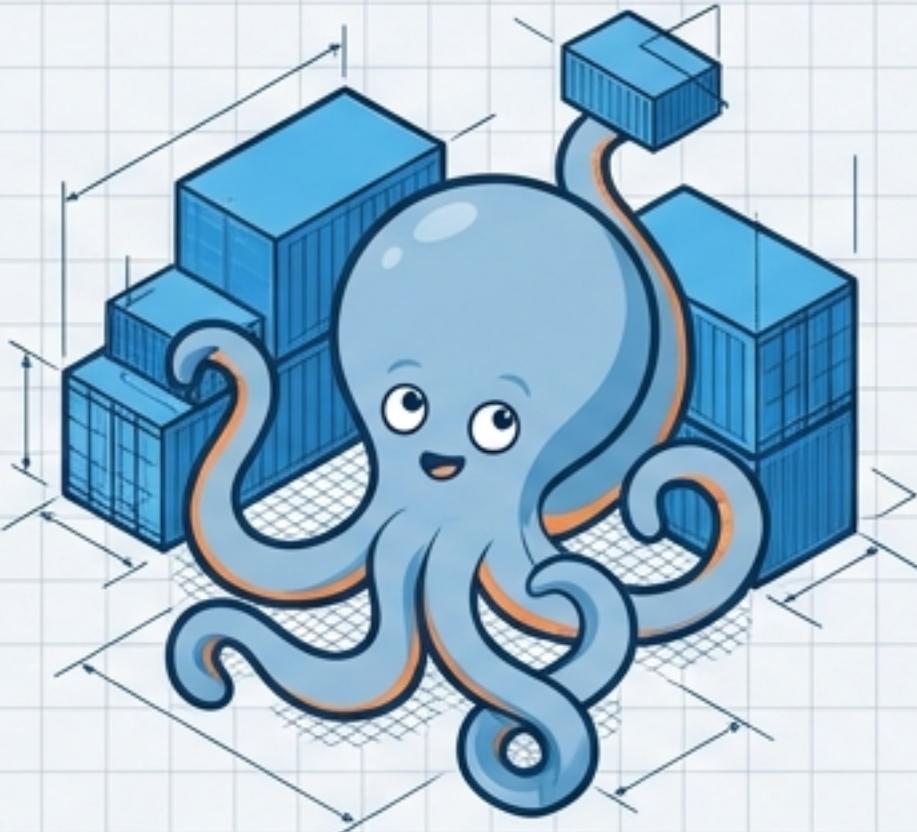
- แอปพลิเคชันจริงไม่ได้มีแค่ Service เดียว (App + DB + Redis)
- ต้องจำคำสั่ง `docker run` ยาวๆ หลายบรรทัด
- ต้องจัดการ Network ให้ Container คุยกันเองวุ่นวาย

Solution: ต้องมีตัวช่วยจัดการ (Orchestration)

Docker Compose คืออะไร

One File

กำหนดทุกอย่างในไฟล์
`docker-compose.yaml`



Docker Compose

One Command

สั่งรันทุก Service
ด้วยคำสั่งเดียว

Consistent Environment

จำลองสภาพแวดล้อมทั้งหมด (App + DB) ให้เหมือนกันทั้งทีม

โครงสร้างของ docker-compose.yaml

```
version: "3.8"
services:
  mongodb:
    image: mongo
    ports:
      - "27017:27017"
  web-app:
    build: .
    ports:
      - "3000:3000"
  environment:
    - DB_HOST=mongodb
```

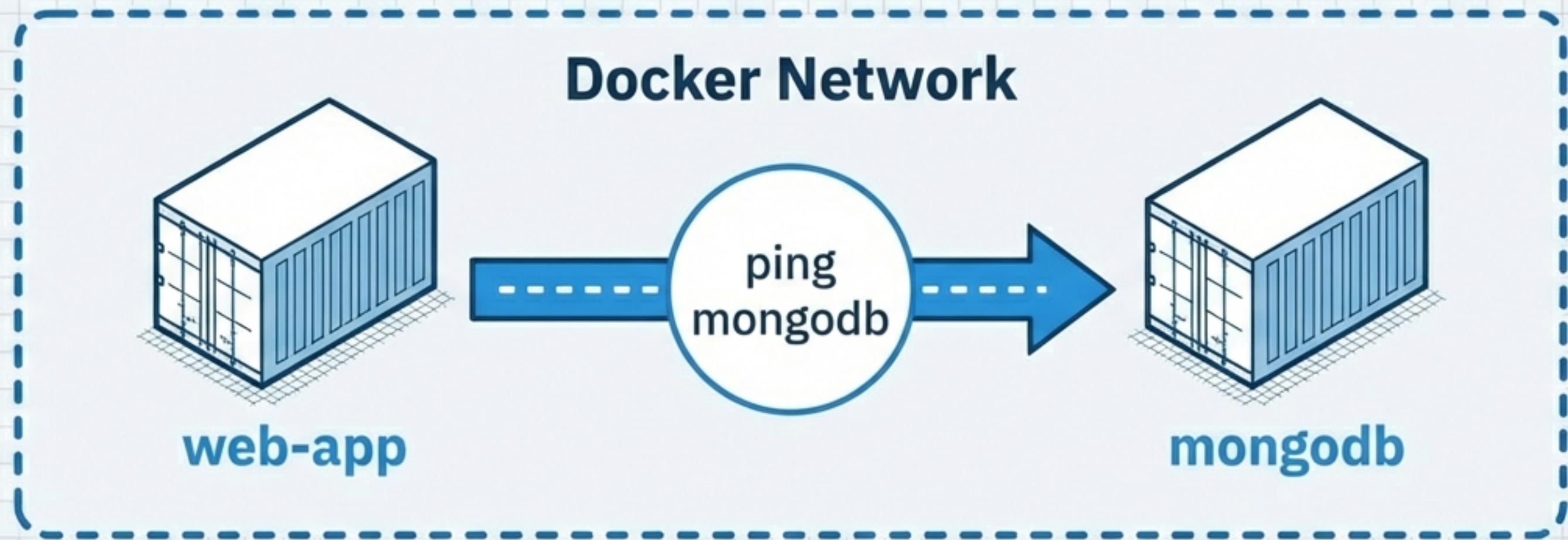
version: ระบุเวอร์ชันไฟล์

services: รายชื่อ Container
(mongodb, web-app)

image/build: ใช้ Image กลาง
หรือ Build จาก Dockerfile

environment: ใส่ค่า Config
หรือ Password

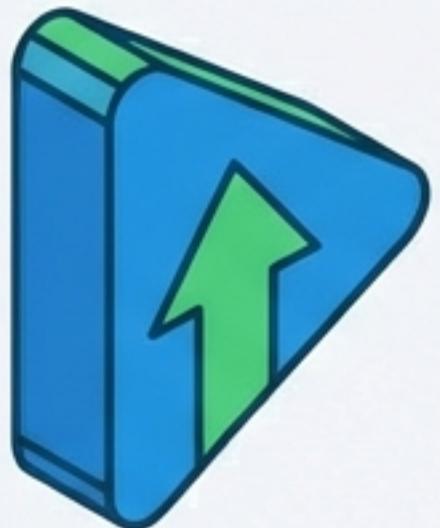
Networking แบบอัตโนมัติ



- **Service Discovery:** Container คุยกันได้โดยใช้ 'ชื่อ Service' เป็น Hostname ไม่ต้องใช้ IP Address
- **Example:** Web App เชื่อมต่อ Database ด้วย URL `mongodb://mongodb:27017`

ควบคุมทุกอย่างด้วยคำสั่งเดียว

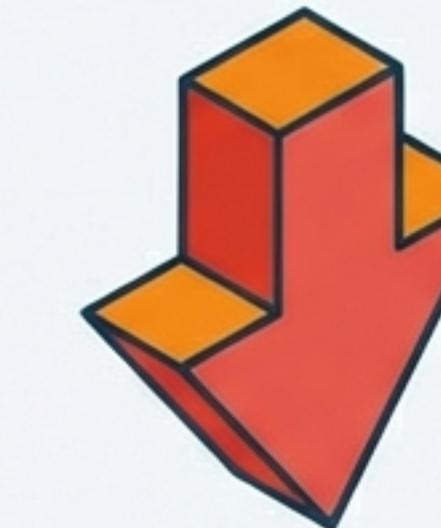
Docker Blue



`docker-compose up`

สร้าง Network, สร้าง Container,
และเริ่มทำงาน
(ใช้ `-d` เพื่อรันเบื้องหลัง)

Construction Orange



`docker-compose down`

หยุดและ **ลบ** Container + Network
ทิ้งทั้งหมด (Clean slate)

Note: `docker-compose start/stop` หยุดชั่วคราวแต่ไม่ลบ Container

ความปลอดภัยและข้อมูล

Security



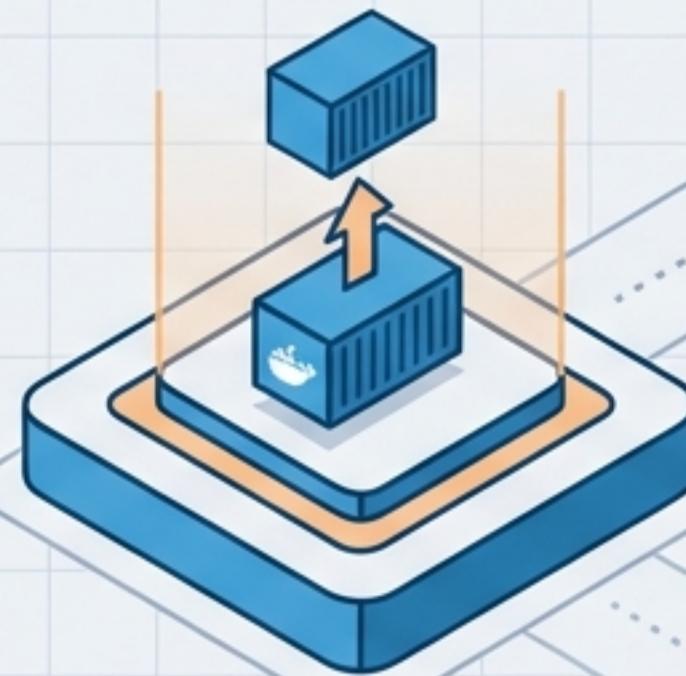
Environment Variables: อย่าใส่ Password ในไฟล์ YAML ตรงๆ! ให้ใช้ syntax `${MONGO_PASSWORD}` และเก็บค่าจริงไว้ในไฟล์ `.env`

Persistence



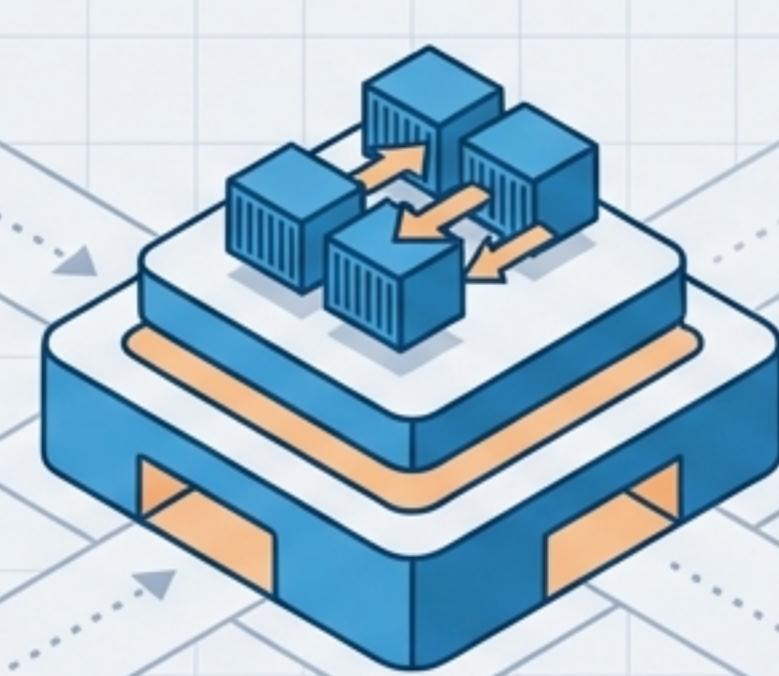
Volumes: Container พังข้อมูลหาย! ต้องใช้ **Volumes** ผูก Folder เพื่อเก็บข้อมูล Database ภาชนะ

สรุปและก้าวต่อไป



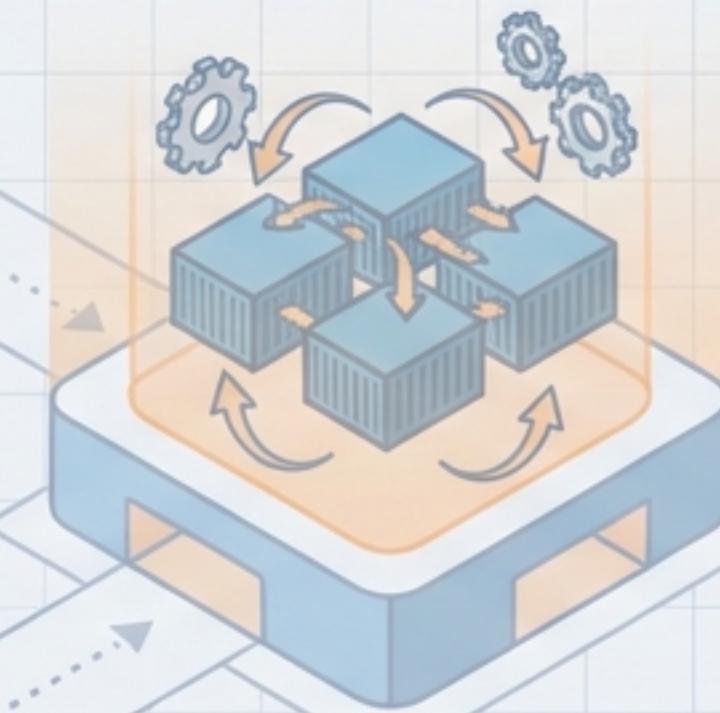
Docker

สร้าง Package (Image)
และรัน (Container)



Docker Compose

จัดการหลาย Container
ในเครื่อง Dev



Kubernetes

Production Scale &
Auto-healing (Next Step)

Standardize your dev, simplify your ops.