# Docker & Dockerization

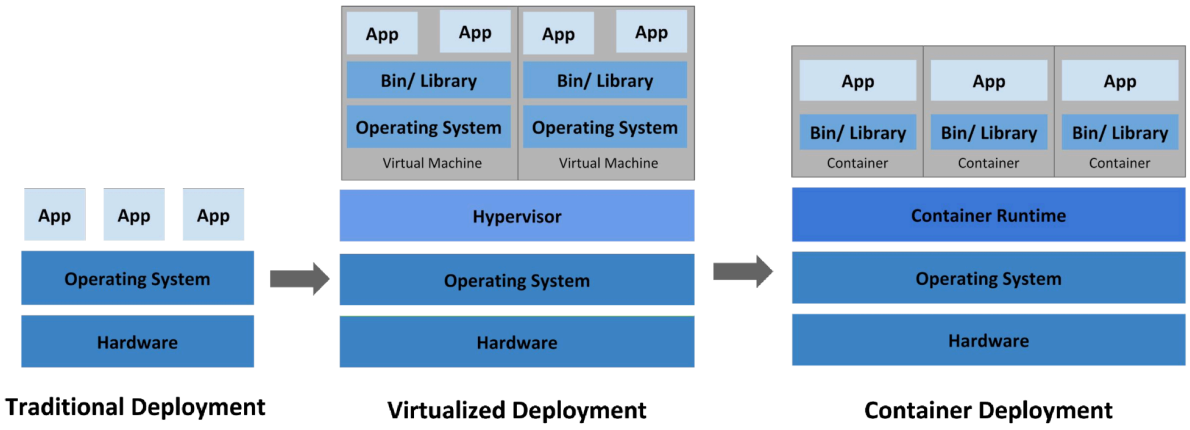
# Table of Contents

# Intro to Docker

---

เครื่องมือแบบ open-source ที่ช่วยจำลองสภาพแวดล้อม (environment) ในการรัน service หรือ server ตามหลักการสร้าง container เพื่อจัดการกับ library ต่างๆ อีกทั้งยังช่วยจัดการในเรื่องของ version control เพื่อง่ายต่อการจัดการกับปัญหาต่างๆ ที่เกิดขึ้น ซึ่งในปัจจุบันในโลกของการพัฒนา software มีรูปแบบการทำงานแบบ agile ที่เน้นความรวดเร็วในการส่งมอบงานในแต่ละขั้นตอน Docker จึงเป็นที่รู้จักในวงกว้างและเริ่มเข้ามามีบทบาทอย่างมากในโลกของการพัฒนา software อีกทั้งยังเป็นเครื่องมือที่จำเป็นสำหรับการทำ DevSecOps

## ทำไมต้องใช้ Docker

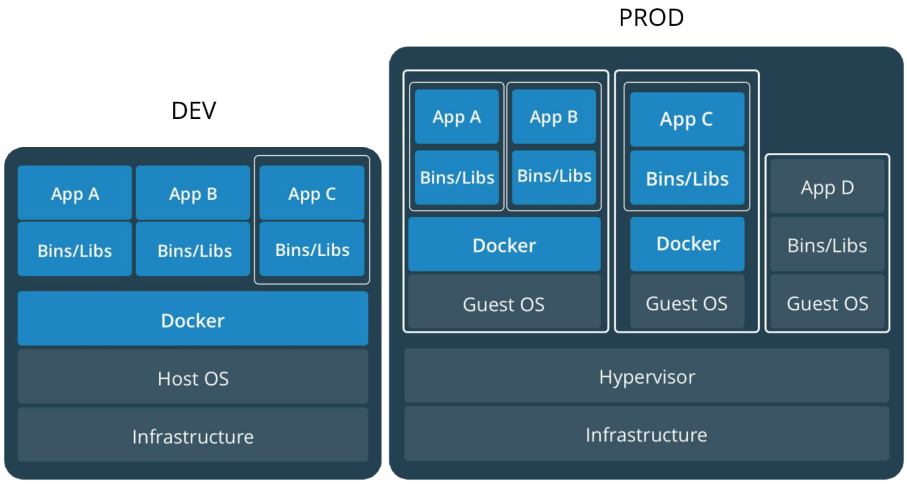



1. **Traditional Deployment** ในยุคที่เราใช้ Physical server 1 เครื่อง ในการ deploy และเพื่อความคุ้มค่า Physical server ที่เรามีจะถูกใช้ในการลง application หลายๆ อันพร้อมๆ กัน ซึ่งทำให้เกิดปัญหา element ของแต่ละ application ตีกัน เช่น application แต่ละตัวมีการลง JAVA ซึ่งเป็นในเครื่องเรามี JAVA หลายๆ เวอร์ชัน 1.5, 1.6, 1.8 ทำให้เวลา run มีปัญหาเกิดขึ้น ทั้งการ maintenance และปัญหาในการเลือกเวอร์ชัน
2. **Virtualized Deployment** เป็นยุคที่มีการเกิดขึ้นของ software hypervisor ซึ่ง concept คือการจำลองเครื่อง Physical server ขึ้นมา เรียกว่า Virtual Machine ทั้ง CPU, memory, hard disk, hardware ต่างๆ ขึ้นมาเสมือนคอมพิวเตอร์เลย ปรับสเปคปรับความเร็วต่างๆ ได้ตามงบประมาณที่เรามี ซึ่งส่วนใหญ่ก็จะสร้าง VM ขึ้นมาหลายๆ เครื่อง ให้แต่ละเครื่องเพียงพอต่อการ run application แต่ละตัว แต่ปัญหา

คือจะเกิดปัญหา Overhead ทั้งในเรื่องการจำลอง hardware ต่างๆ ทำให้ทำงานได้ช้าลง 2-5 เท่า รวมถึง Overhead ในกรณีที่เรามีการ run ใน environment ที่ใกล้เคียงกัน จะทำให้เปลือง resource ไปโดยใช่เหตุ

3. **Container Deployment** ซึ่งในยุคนี้จะพูดถึงการสร้าง Container ขึ้นมาเพื่อขัง resource สร้างกำแพงขึ้นมาแบ่ง resource ทำให้เราสามารถใช้ resource ได้อย่างมีประสิทธิภาพมากขึ้น เลือก container ไปใช้กับแต่ละ application ได้ดียิ่งขึ้น ทำให้ปัญหา overhead ลดลง และเครื่องมือในการสร้าง Container ที่นิยมก็คือ Docker

## Container แตกต่างจาก Virtual Machine อย่างไร?




เนื่องจาก Docker ทำงานอยู่บน Container เป็นหลัก จัดการเรื่องทรัพยากรต่างๆ ของเครื่องต่อจาก OS อีกที ต่างจาก VM ที่แบ่งทรัพยากรและ OS อย่างชัดเจนเลยตั้งแต่แรก ถ้าสังเกตจากภาพด้านล่างจะเห็นว่า Docker จะรันบน OS แต่ถ้าเป็น VM จะรันบน Virtualization Hypervisor โดยตรงเลยแล้วค่อยรัน OS บน VM ซึ่งส่งผลให้การทำงานของ VM มี overhead มากกว่าการ run service ด้วย container รวมถึงในแง่ของการใช้ทรัพยากรอย่าง CPU, memory, disk อีกด้วย

## ข้อดีของ Docker

1. Portability ทดสอบ container ที่เดียวสามารถ deploy ได้ทุกที่ที่มี docker รันอยู่โดยไม่ต้องกลัวว่าจะไม่สามารถรันได้
2. Performance เนื่องจาก container ไม่ได้มีการบรรจุ OS เข้าไปด้วย นั่นหมายความว่า docker นั่นจะมีขนาดเล็กกว่า VM ทำให้ขนาดเล็ก, build ได้เร็วกว่า รวมถึงการรันได้มีประสิทธิภาพดีกว่าด้วย
3. Agility ด้วย portability และ performance ช่วยให้เหมาะสมกับการทำ agile process รวมถึงเหมาะกับการทำ CI/CD อีกด้วย ช่วยให้ compile, build, test ได้ดียิ่งขึ้น
4. Scalability เราสามารถสร้าง container ใหม่ ได้ตามความต้องการของ application ที่ scale ได้ โดยใช้เวลาอันสั้น

## ข้อเสียของ Docker

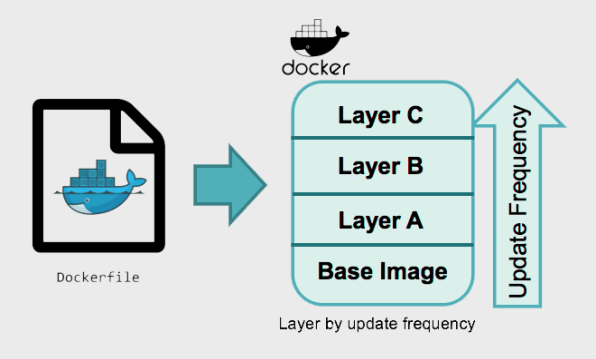
1. เนื่องจากการรัน docker ไม่ได้รัน OS ใหม่ทั้งหมดเป็นเพียงแค่การจำลอง env ทำให้อาจเกิดการโจมตีที่ OS หลักผ่านทาง docker ได้และอาจกระทบกับ container ตัวอื่นๆ
2. ตอนเริ่มแรกที่ docker ถูกสร้าง มันถูกออกแบบมาเพื่อรองรับการรันบน linux เท่านั้น ที่เราสามารถรัน docker บน window,mac ได้นั้นเพราะ เมื่อเราลง docker ใน window,mac จะมีการสร้าง VM ที่เป็น linux เพื่อมารัน docker อีกที ทำให้ประสิทธิภาพการทำงานอาจจะไม่สามารถทำได้สูงสุดเท่ารันบน linux
3. Docker ไม่เหมาะกับการจัดการ resource บนเครื่องใหญ่ๆ หรือไม่เหมาะกับโปรแกรมที่ออกแบบมาเพื่อทำงานบน VM
4. Learning curve ที่สูง เนื่องจากเป็นการทำงานเกี่ยวกับ OS, network รวมถึงการจัดการทรัพยากรต่างๆ ทำให้ต้องอาศัยเวลาการเรียนรู้ที่ค่อนข้างสูง แต่ทาง docker ก็มี tool ใหม่ๆ ออกมาช่วยเหลือให้ใช้งานได้ง่ายยิ่งขึ้น แต่การที่จะใช้งาน docker ได้อย่างช่ำชองจำเป็นต้องเรียนรู้ tools อื่นเพื่อใช้ในการประกอบด้วย

# Docker Concept

---

## Image

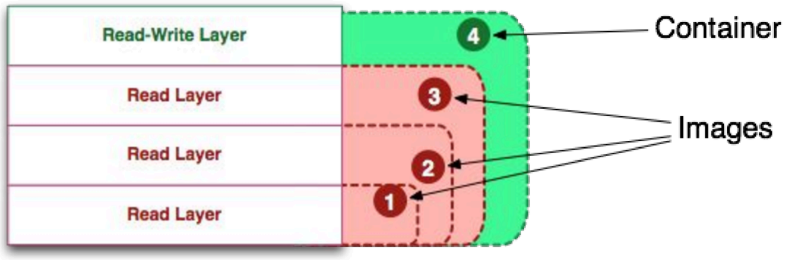
ต้นแบบของ Container ข้างในจะเป็น Linux ที่มีการติดตั้ง Application และมีการ Configuration เอาไว้แล้ว ซึ่งเกิดมาจากการ build ไฟล์ Dockerfile ขึ้นมาเป็น image




```
FROM node:14.14.0-alpine3.12   <-- OS + System Packages
COPY . /nodejs/.               <-- Source Code
WORKDIR /nodejs
RUN npm install                <-- Library Dependencies
ENV VERSION 1.0                <-- Configuration
EXPOSE 8081
CMD ["node", "/nodejs/main.js"]
```

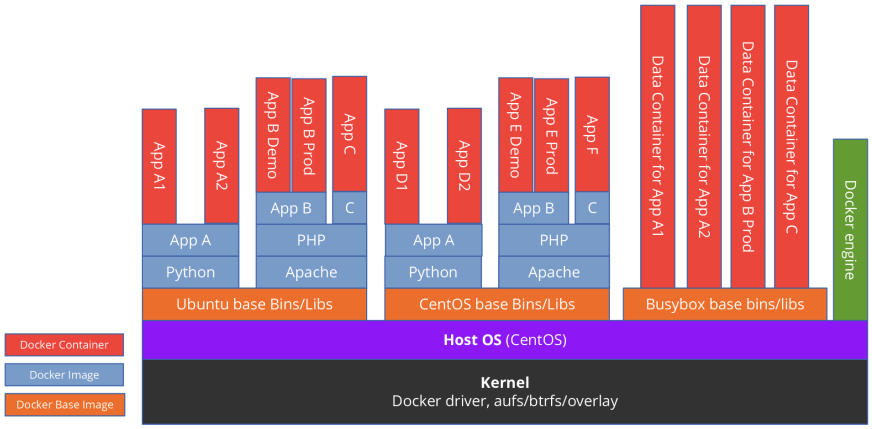
## Container

container จะถูกสร้างมาจาก Docker Image ที่เป็นต้นแบบ เกิดเป็น container จะได้ Service หรือ Application ที่สามารถเรียกใช้งานได้ทันที



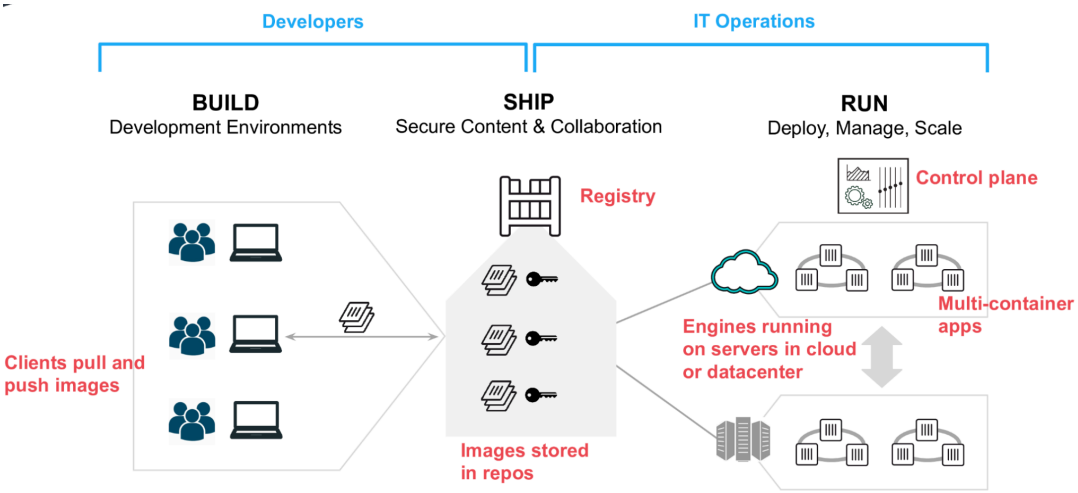

***Docker Image is a class***

***Docker Container is a instance of class***

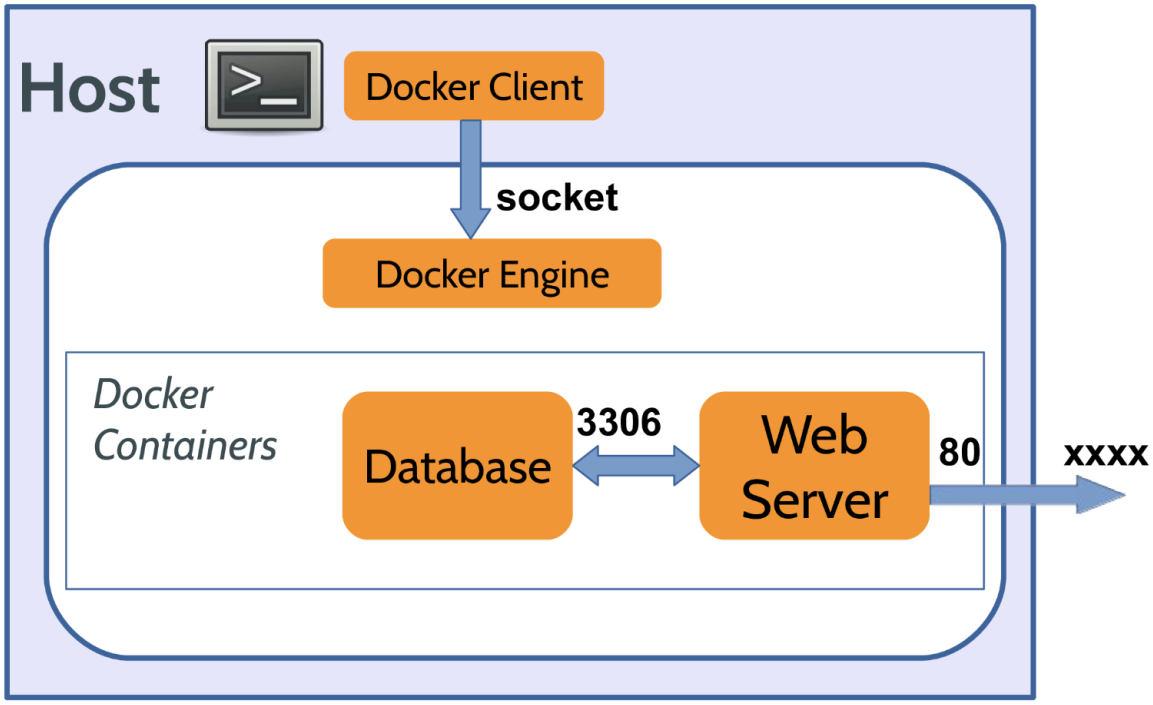



## Registry

เราสามารถสร้าง Docker Image แล้วนำไปเก็บรวบรวมไว้บน server (ลักษณะเดียวกับการเก็บ Source Code ไว้บน Github) โดย Docker registry ณ ปัจจุบันก็มีให้เลือกใช้งานได้หลากหลายโดยมี Docker Hub เป็น Docker registry หลักในการเรียกใช้(pull) Docker Image และนอกจากนี้ยังมีผู้ให้บริการ docker registry เจ้าอื่นๆด้วย เช่น Gitlab, Quay.io, Google Cloud เป็นต้น



# Docker Architecture




Docker เป็น Architecture เป็น client server หมายความว่า command ที่เราพิมพ์เช่น docker run, docker build ตัว client จะไปสั่งให้ engine ทำงาน โดยในกรอบของ Docker Container จะมี environment (Network, Namespace) เป็นของตัวเอง หากต้องการที่จะเข้าถึง web server จะเข้าถึงไม่ได้เนื่องจาก web server จะมี ip เป็นของตัวเองที่แยกออกไป ดังนั้นหากเราต้องการที่จะเข้าถึง web server จึงจำเป็นจะต้องทำ Port Forwarding

# Docker Command

---

| Command | Describe |
|---|---|
| docker ps | List all running containers |
| docker stop container-id | Stop the container which is running |
| docker start container-id | Start the container which is stopped |
| docker restart container-id | Restart the container which is running |
| docker port container-id | List port mappings of a specific container |
| docker rm container-id | Remove the stopped container |
| docker rm -f container-id | Remove the running container forcefully |
| docker pull image-info | Pull the image from docker hub repository |