CS438

Container

Lecture S3

1st Semester 2021

Kasidit Chanchio

Department of Computer Science

เนื้อหาเรียบเรียงจาก https://www.youtube.com/watch?v=sK5i-N34im8&t=211s

คอนเทนเนอร์

- คือการประมวลผลแบบเสมือนอย่างเบา (lightweight VM) ที่สร้างภาพเสมือนของการใช้งานคอนเทนเนอร์เหมือนกับเป็นเครื่องจริงเครื่องหนึ่ง ที่ผู้ใช้สามารถ
  - รันแอพพลิเคชั่น
  - ประมวลผลเป็นซูเปอร์ยูเซอร์
  - รันเซอร์วิส
  - ติดตั้งแพคเกจ
  - กำหนดค่าเน็ตเวิร์ค เช่นไอพีแอดเดรส
  - มีอุปกรณ์เครือข่าย (เสมือน) เป็นของตนเอง

คอนเทนเนอร์

- แต่คอนเทนเนอร์ไม่ใช่วีเอ็ม
- คอนเทนเนอร์เป็นกลุ่มของโพรเซสที่รันอยู่บนโฮสโอเอส
- คอนเทนเนอร์สร้างภาพเสมือนของการแยกกลุ่มของโพรเซสออกจากกลุ่มอื่น และแต่ละกลุ่มจะมองไม่เห็นโพรเซสในกลุ่มอื่น
- โฮสโอเอสเห็นโพรเซสในคอนเทนเนอร์ แต่โพรเซสในคอนเทนเนอร์จะเห็นเฉพาะโพรเซสที่อยู่ในคอนเทนเนอร์เดียวกันเท่านั้น
- ไม่สามารถรันต่างโอเอสจากโฮสได้ และไม่สามารถมีเคอร์เนิลโมดูลแยกจากของโฮสได้ และทุกคอนเทนเนอร์แชร์เคอร์เนิลร่วมกัน
- แต่ละคอนเทนเนอร์มีโพรเซสที่มีพีไอดี 1 แต่ไม่ใช้ init ไม่มี cron

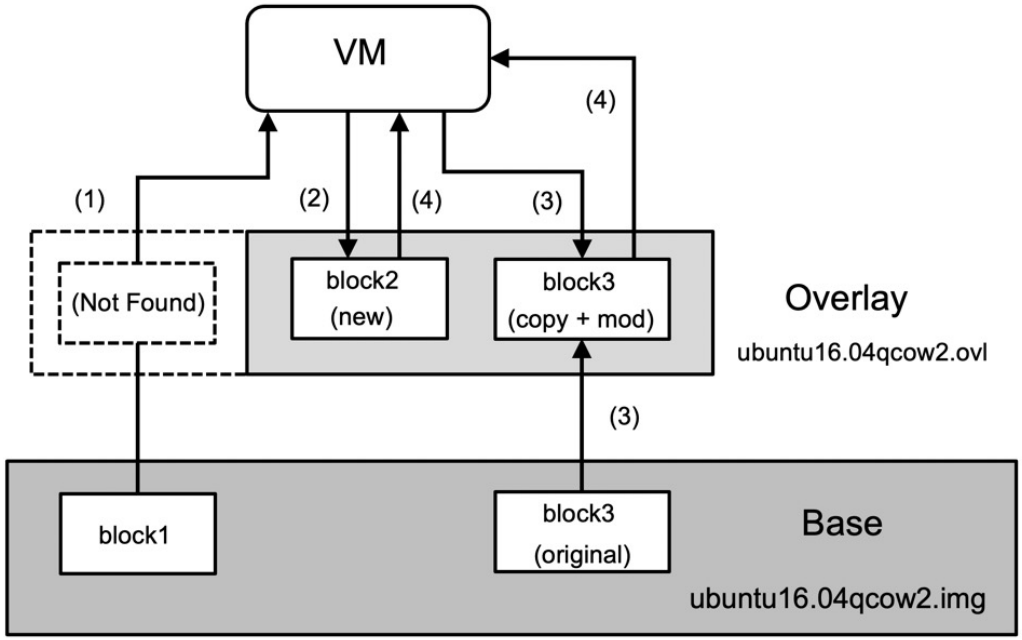
คอนเทนเนอร์

- คอนเทนเนอร์ สร้างขึ้นจากความสามารถของเคอร์เนิล 3 อย่างได้แก่
- ซีกรุ๊ป (cgroup) อนุญาตให้เราวัดและกำหนดปริมาณการใช้งานทรัพยากรสำหรับกลุ่มของโพรเซส เช่นหน่วยความจำ ซีพียู บล็อกไอโอ เน็ตเวิร์คไอโอ
- เนมสเปซ (name space) อนุญาตให้มีการสร้างอินสแตนซ์ของความสามารถของเคอร์เนิลให้มีมากกว่าหนึ่งอินสแตนซ์ โดยที่กลุ่มของโพรเซสในหนึ่งอินสแต๊นซ์จะเห็นเฉพาะโพรเซสในกลุ่มและไม่เห็นโพรเซสในอินสแต๊นซ์อื่น
- ก็อปปี้ออนไร้ท์ (Copy on Write) (CoW) อิมเมจ อนุญาตให้กลุ่มของโพรเซสมีไฟล์ซิสเต็มอิมเมจของตนเอง ที่มองเห็นได้เฉพาะโพรเซสในกลุ่ม

ก็อปปี้ออนไร้ท์ (Copy on Write) อิมเมจ

- อนุญาตให้ผู้ใช้สร้างอิมเมจ อย่างรวดเร็วโดย ดาวน์โหลดเฉพาะอิมเมจที่จำเป็นต่อการใช้งาน แทนที่จะสร้างหรือดาวน์โหลดไฟล์ซิสเต็มขนาดใหญ่มาทีเดียว
- เบสอิมเมจ (base) คือข้อมูลตั้งต้น (อาจเป็นส่วนใหญ่ที่มักจะไม่เปลี่ยน)
- โอเวอร์เลย์ คือข้อมูลการเปลี่ยนแปลงที่เกิดกับเบสอิมเมจ หรือโอเวอร์เลย์อิมเมจชั้นก่อนหน้า
- โอเวอร์เลย์ สามารถมีได้หลายชั้น โดยมีโอเวอร์เลย์อื่นเป็นเบส
- การเปลี่ยนแปลงในปัจจุบัน จะเกิดขึ้นบนวีเอ็มชั้นบนสุด

หลักการก็อปปี้ออนไร้ท์



1. ถ้าหา block ใน ovl ไม่พบก็จะมองหาใน base (recursive)
2. ถ้ามีการเพิ่มข้อมูลใหม่จะเพิ่มใน ovl
3. การเปลี่ยนแปลงข้อมูล จะก็อปปี้บล็อกข้อมูลจากเบสมาที่ ovl ก่อนแล้วจึงเปลี่ยนค่าใน ovl

ก็อปปี้ออนไร้ท์ (Copy on Write) อิมเมจ

- คอนเทนเนอร์จะติดตามความเปลี่ยนแปลงที่เกิดขึ้นในอิมเมจเสมอ
- มีระบบจัดการข้อมูลแบบ CoW หลายแบบ ได้แก่
  - AUFS และ Overlay เป็นการทำ COW ในระดับไฟล์
  - device mapper เป็นการทำ COW ในระดับบล็อกข้อมูล
  - ZFS, BTRFS เป็นการทำ COW ในระดับไฟล์ซิสเต็ม
- เนื่องจากอิมเมจมีขนาดเล็ก ทำให้สร้าง และบูทได้อย่างรวดเร็ว

คอนเทนเนอร์

- การใช้งาน ซีกรุ๊ป เนมสเปซ และก็อปปี้ออนไร้ท์ อิมเมจ นั้นจะใช้ร่วมกันหรือแยกจากกันก็ได้
  - ถ้าต้องการแค่จำกัดปริมาณการใช้งานทรัยากร ก็ใช้แค่ ซีกรุ๊ป
  - ถ้าต้องการแค่แยก เน็ตเวิร์คสแตก (Network Stack) ของโฮสและกลุ่มของโพรเซสออกจากกัน เช่นในการทำ Network Virtualization เราสามารถสร้างเร้าเตอร์หลายเร้าเตอร์บนเครื่องโฮส โดยที่แต่ละเร้าเตอร์มี เน็ตเวิร์คเนมสเปซของตนเอง
  - ถ้าต้องการใช้ debugger เพื่อดูการทำงานของโพรเซสแต่ไม่ต้องการให้ debugger ใช้ทรัพยากรที่จำกัดของคอนเทนเนอร์ก็ ให้โพรเซส debugger อยู่ในเนมสเปซของคอนเทนเนอร์ แต่ไม่ให้อยู่ในซีกรุ๊ป
  - ย้ายอินเตอร์เฟสจากคอนเทนเนอร์หนึ่งไปอีกคอนเทนเนอร์หนึ่งได้

แคพาบิลิตี้ (Capabilities) และความปลอดภัย

- แคพาบิลิตี้ (capabilities) อนุญาตให้ผู้ใช้กำหนดความสามารถในการกำหนดความเป็น root และ non-root เพื่อบริหารจัดการทรัพยากรแบบละเอียด (fine grain) แทนที่จะให้เป็น root และ non-root ของทั้งระบบ ยกตัวอย่างเช่น คอนเทนเนอร์บางคอนเทนเนอร์อาจทำหน้าที่จัดการวีพีเอ็น เราสามารถใช้ capability เฉพาะเพื่อกำหนดให้มันสามารถจัดการ อินเตอร์เฟสที่เกี่ยวข้องกับวีพีเอ็นได้ เป็นต้น
- ผู้ใช้อาจใช้ซอฟต์แวร์ SELinux หรือ AppArmour เพื่อเพิ่มความปลอดภัยในการใช้งานคอนเทนเนอร์

คอนเทนเนอร์

- LXC สร้างจาก namespace และ cgroups แต่ไม่มีเครื่องมือบริหารจัดการอิมเมจที่ง่ายต่อการใช้งาน
- Docker engine ประกอบไปด้วย
  - daemon process ให้บริการ Rest API
  - ใช้ namespace และ cgroups และ cow image
  - ใช้ libcontainer เพื่อรันโพรเซส
  - จัดการ container image และการสร้างและจัดการอิมเมจ
- runc เป็น simplify version ของ docker ที่ใช้ libcontainer
- อื่นๆ OpenVZ Jail/Zone

Docker คอนเทนเนอร์

- เป็นพลตฟอร์มสำหรับการ
  - พัฒนา แอพพลิเคชั่น
  - ส่งมอบ แอพพลิเคชั่น และ
  - รันแอพพลิเคชั่น
- ในอดีต การพัฒนา ส่งมอบ และติดตั้ง ใช้เวลานานเนื่องจากสภาพแวดล้อมของการพัฒนาซอฟต์แวร์กับสภาพแวดล้อมในการติดตั้งและใช้งานจริงต่างกัน
- ผู้ใช้สามารถพัฒนา และทดสอบ แอพพลิเคชั่นได้ภายในคอนเทนเนอร์
- หลังจากนั้นสามารถ deploy ได้บนทุกที่ เช่น cloud หรือ server

Docker คอนเทนเนอร์

- Docker แก้ปัญหาดังกล่าวด้วยการมองแพคเกตเป็น COW อิมเมจ และมีเครื่องมือสำหรับ upload และ download อิมเมจเป็นส่วนๆที่ทำได้อย่างรวดเร็ว
- อิมเมจที่ใช้มีซอฟต์แวร์และส่วนประกอบของเคอร์เนิลและซอฟต์แวร์ที่เกี่ยวข้องที่เหมาะสมสำหรับรันแอพพลิเคชั่น ทำให้สามารถดาวน์โหลดทุกอย่างมาได้ทั้งหมดและสามารถ deploy แอพพลิเคชั่นใน production environment ได้โดยไม่ขาดซอฟต์แวร์ที่จำเป็น
- Docker เป็น container ซึ่งใช้กลุ่มของโพรเซสในโอเอสเพื่อรันแอพพลิเคชั่น จึงมีความรวดเร็วในการติดตั้งและใช้งานโดยเฉพาะเมื่อต้องการรันหลายอินสแต๊นซ์พร้อมกัน

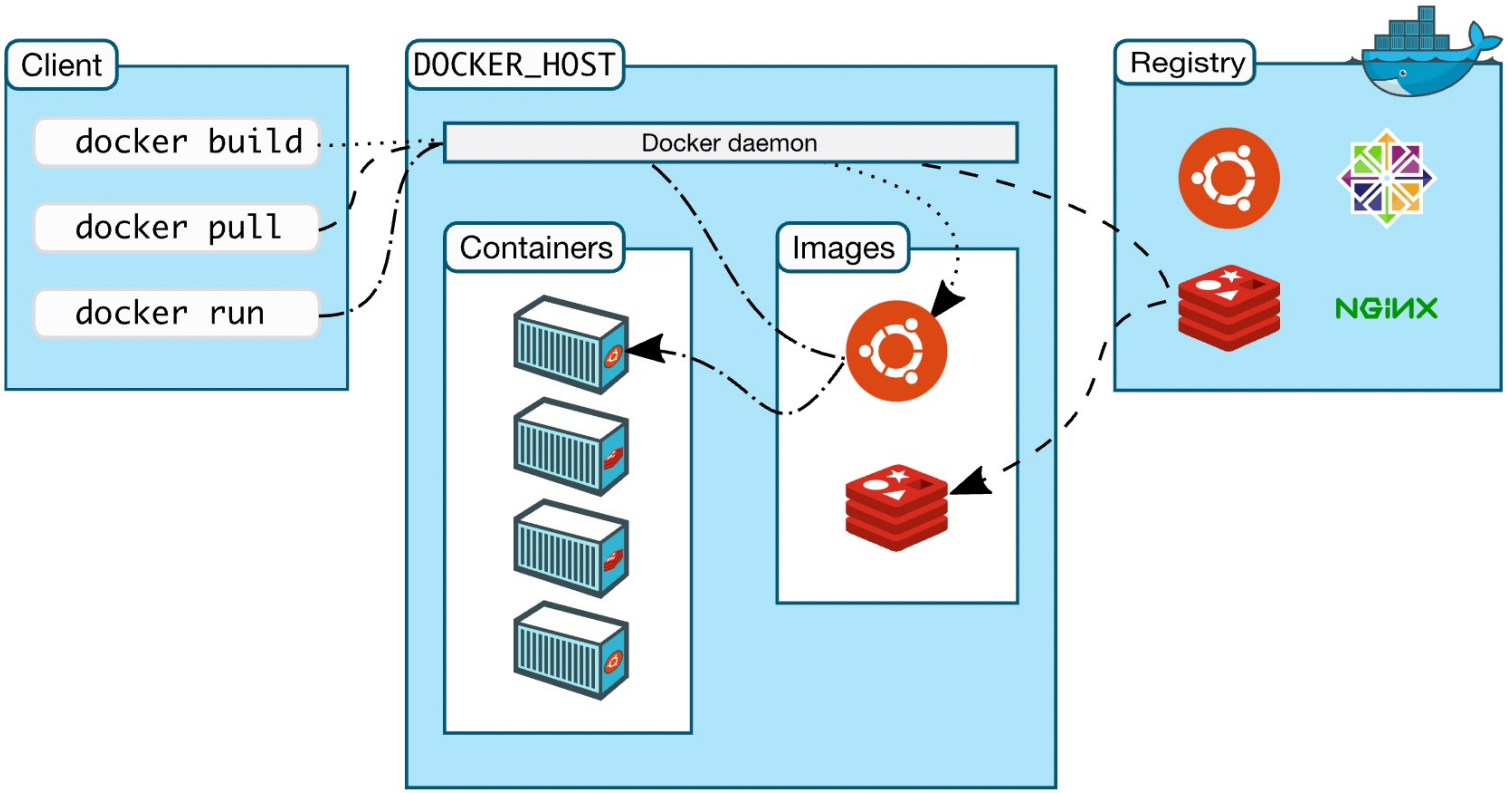
Docker Use-Case

- Developer พัฒนาโปรแกรมและแชร์โปรแกรมกับทีม โดยใช้ docker container ในสภาพแวดล้อมพัฒนาโปรแกรม (development environment)
- หลังจากพัฒนาเสร็จก็ใช้ docker ติดตั้งแอพพิเคชั่นในสภาพแวดล้อมสำหรับทดสอบ (test environment)
- ถ้าพบ bugs ก็พัฒนาโปรแกรมต่อและใช้ docker ติดตั้งแอพพลิเคชั่นเพื่อทดสอบต่อ
- เมื่อทดสอบเสร็จแล้ว ก็ใช้ docker ติดตั้งแอพพลิเคชั่นในสภาพแวดล้อมใช้งานจริง (Deployment environment)

สถาปัตยกรรมของ Docker คอนเทนเนอร์

- ใช้สถาปัตยกรรมแบบ Client และ Server
- docker client สั่งงานผ่าน docker daemon ด้วย Rest API บน UNIX socket หรือ Network
- ทั้งสองอาจอยู่บนเครื่องเดียวกันหรือต่างเครื่องกันก็ได้
- docker client สามารถติดต่อ docker daemon ได้หลาย daemon
- docker compose เป็น client อีกชนิดหนึ่งทีอนุญาตให้ผู้ใช้งาน เซ็ตของ container

สถาปัตยกรรมของ Docker คอนเทนเนอร์

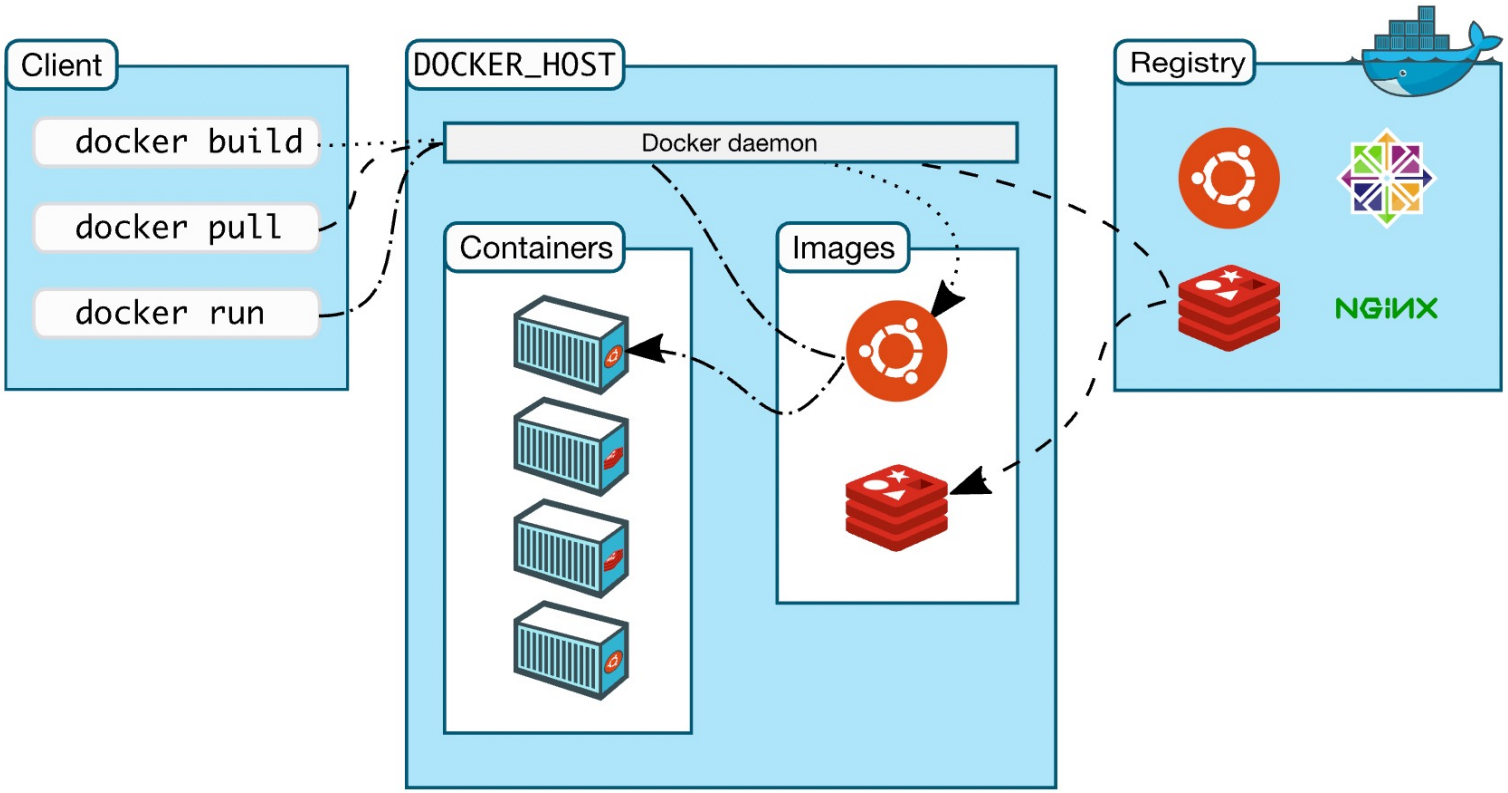


ภาพจาก https://docs.docker.com/get-started/overview/

สถาปัตยกรรมของ Docker คอนเทนเนอร์

- Docker daemon (dockerd) รอรับ Docker API และจัดการ docker object เช่น image และ container และ network และ voliumes นอกจากนั้น dockerd สามารถติดต่อกับ dockerd อื่นเพื่อให้บริการผู้ใช้ได้ด้วย
- docker client (docker) เป็นโปรแกรมที่ผู้ใช้เรียกใช้เพื่อสั่งงาน dockerd ด้วย Docker API ยกตัวอย่างเช่นผู้ใช้สามารถออกคำสั่ง docker run ซึ่งจะถูกส่งไปให้ dockerd ดำเนินการ

สถาปัตยกรรมของ Docker คอนเทนเนอร์



ภาพจาก https://docs.docker.com/get-started/overview/

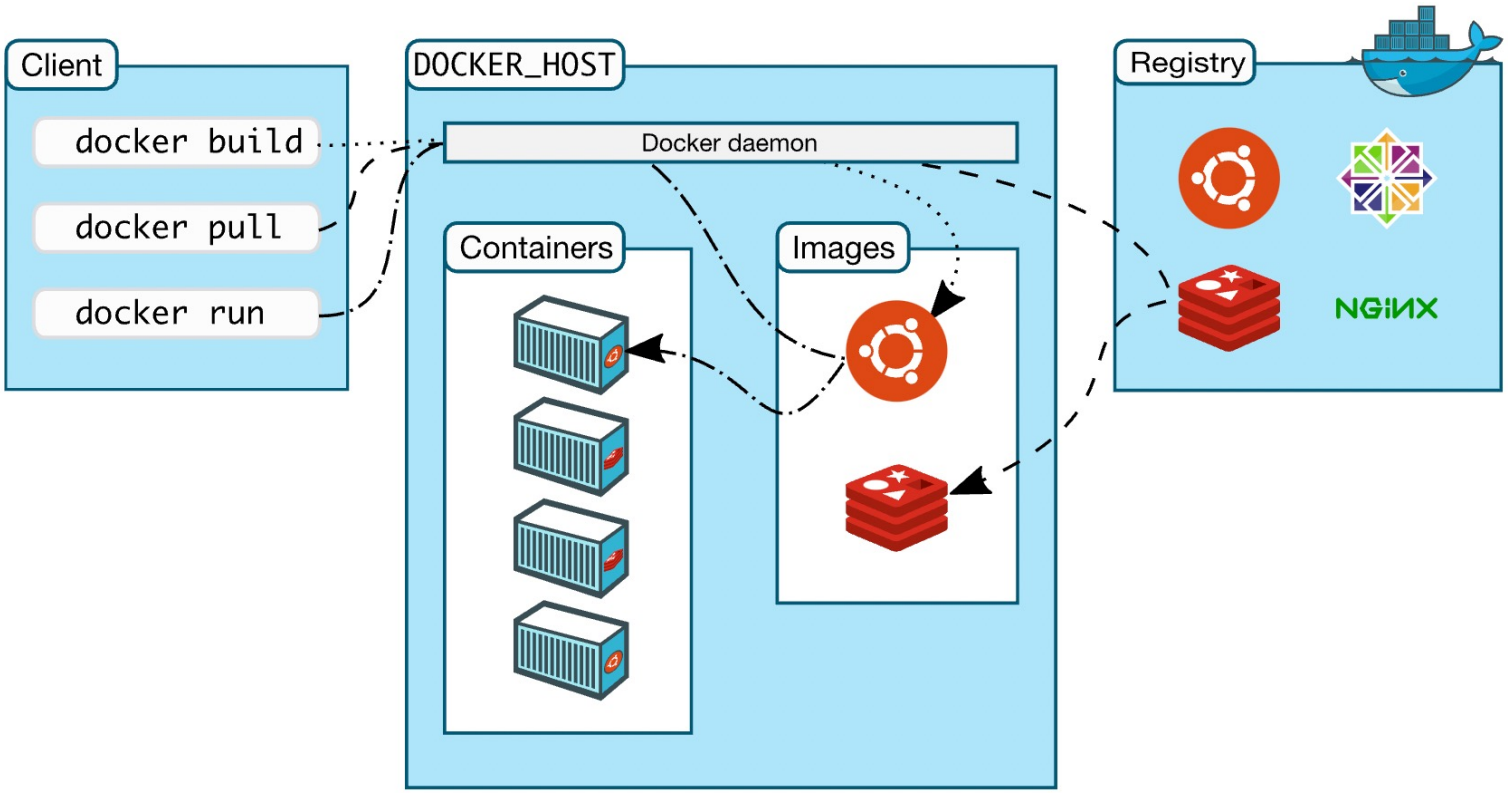
สถาปัตยกรรมของ Docker คอนเทนเนอร์

- docker registry (registry) ใช้เก็บ docker อิมเมจ มีทั้งแบบที่เป็น registry สาธารณะ (public) เช่น docker hub และส่วนตัว (provate) ที่ผู้ใช้สร้างขึ้นใช้เองได้
  - คำสั่ง docker pull และ docker run จะดึงอิมเมจจาก docker hub มาติดตั้งบนเครื่องของผู้ใช้
  - คำสั่ง docker push อัพโหลดอิมเมจไปสู่ registry

Docker Objects

- เมื่อผู้ใช้ใช้งาน docker ผู้ใช้จะปฏิบัติงานอยู่บน docker objects ได้แก่
  - docker image
  - docker container
  - docker local daemon
  - docker volumes
  - docker networks
  - etc.

สถาปัตยกรรมของ Docker คอนเทนเนอร์



ภาพจาก https://docs.docker.com/get-started/overview/

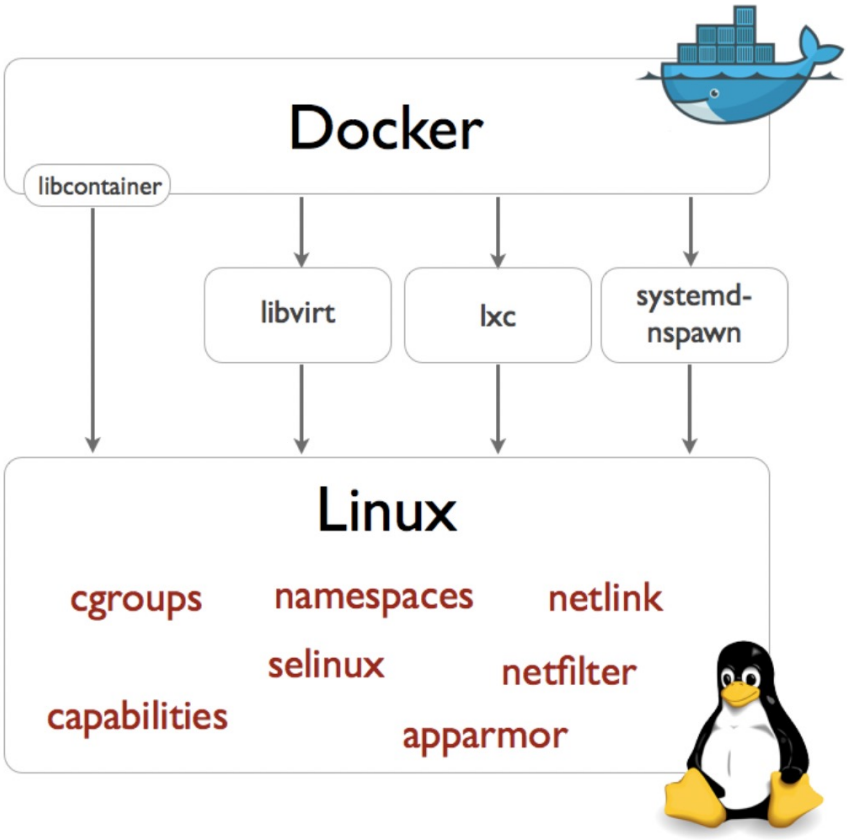
Docker Image

- อิมเมจ (image) เป็น template ของอิมเมจ ที่มีชุดคำสั่งสำหรับสร้าง docker container ในนั้น
- อิมเมจมักจะถูกสร้างขึ้นมาจาก อิมเมจอื่น เช่น อิมเมจของผู้ใช้อาจถูกสร้างขึ้นมาจาก ubuntu อิมเมจ ซึ่งส่วนที่สร้างเพิ่มเติมของผู้ใช้ก็จะมีซอฟต์แวร์และข้อมูลสำหรับการประมวลผลแอพพลิเคชั่นของผู้ใช้
- ผู้ใช้สามารถสร้างอิมเมจของตนเองโดยสร้าง Dockerfile ซึ่งจะประกอบไปด้วยคำสั่งสำหรับสร้างอิมเมจ โดยจะสร้างเป็น layer
- เมื่อเปลี่ยนแปลงเพื่อสร้างอิมเมจใหม่ จะมีเฉพาะ layer ที่เปลี่ยนเท่านั้นที่ถูกนำมาสร้างอิมเมจใหม่ ด้วยเหตุนี้การสร้างอิมเมจจึงทำได้อย่างรวดเร็ว

Docker Container

- คอนเทนเนอร์ คืออินสแตนซ์ของอิมเมจที่ผู้ใช้สามารถสั่งให้ประมวลผลได้ ผู้ใช้สามารถ CREATE START STOP MOVE และ DELETE คอนเทนเนอร์ ได้
- ผู้ใช้สามารถเชื่อมต่อ container เข้ากับระบบเครือข่าย (network) หรือหน่วยเก็บข้อมูล (storage)
- ผู้ใช้สามารถสร้างอิมเมจใหม่จากสถานะของการประมวลผลในปัจจุบัน
- ผู้ใช้สามารถกำหนดระดับของการแยกกันของคอนเทนเนอร์จากคอนเทนเนอร์อื่น และจากโฮสคอมพิวเตอร์ได้
- พฤติกรรมการประมวลผลของคอมเทนเนอร์ขึ้นอยู่กับอิมเมจ และการกำหนดค่าเริ่มต้นตอนที่ผู้ใช้สร้างและรันคอนเทนเนอร์
- เมื่อลบคอนเทนเนอร์ การเปลี่ยนแปลงสถานการณ์ประมวลผลของคอนเทนเนอร์ที่ไม่ถูกเก็บในหน่วยเก็บข้อมูลจะถูกลบไป

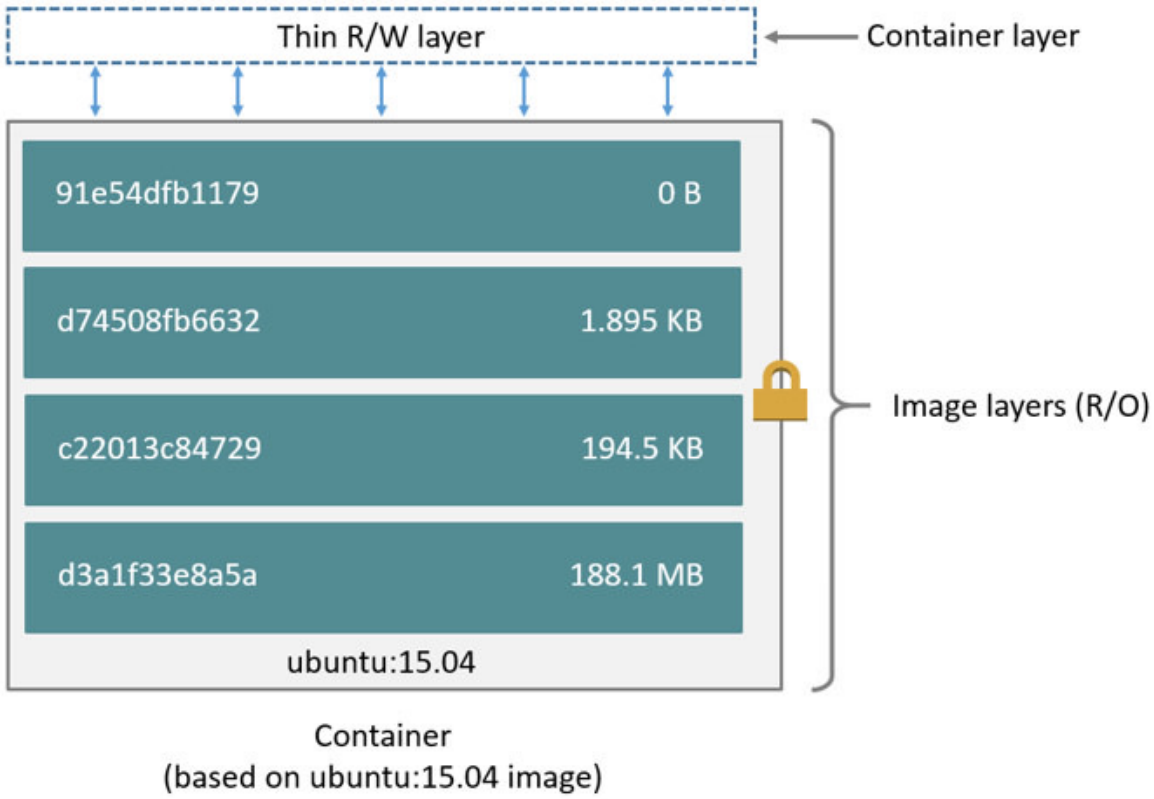
โครงสร้างของ Docker Container



- Dockerd เรียกใช้ library ชื่อ libcontainer เพื่อสั่งงาน Linux Kernel
- Libconatiner เรียกใช้ Linux API เพื่อจัดการ
  - Cgroups
  - Namespace
  - netfilter
  - etc.

ภาพจาก https://stackoverflow.com/questions/26181836/how-does-docker-share-resources

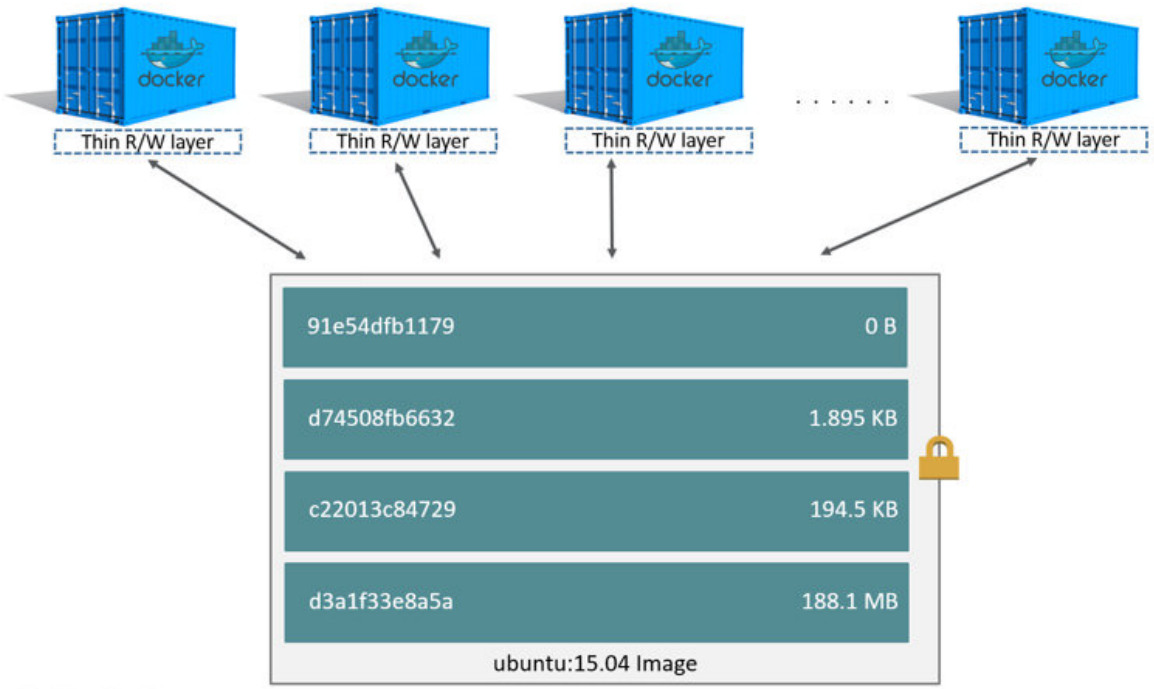
โครงสร้างของอิมเมจแบบ CoW ของ Docker (1)



- Dockerd จัดการเก็บอิมเมจเป็น layer
- จัดการ layer ด้วยหลักการ CoW
- แต่ละ layer เก็บการเปลี่ยนแปลงที่ไม่ใหญ่มาก ทำให้สามารถ pull และนำมาสร้าง image ใหม่ได้อย่างรวดเร็ว
- การจัดการเก็บจริงขึ้นอยู่กับ storage driver เช่น Overlay2, aufs, ZFS, Btrfs, etc.

ภาพจาก https://docs.docker.com/storage/storagedriver/

โครงสร้างของอิมเมจแบบ CoW ของ Docker (1)



- หลักการ CoW อนุญาตให้ หลาย container สามารถ share read-only layer ร่วมกันได้
- การเปลี่ยนแปลงจะถูกเก็บใน storage layer ชั้นบนสุด
- อิมเมจไฟล์ของปต่ละคอนเทนเนอร์อยู่ที่ /var/lib/docker/<storage driver>

ภาพจาก https://docs.docker.com/storage/storagedriver/

สรุป

- แนะนำคอนเทนเนอร์เบื้องต้น
- แนะนำหลักการ Copy On Write
- แนะนำโครงสร้างของ Docker Container เบื้องต้น

เนื้อหาเรียบเรียงจาก https://www.youtube.com/watch?v=sK5i-N34im8&t=211s