

# Docker Engine

Basic Docker

# Let's start

1. เตรียม Guest OS (Ubuntu Server 16.04.1 LTS)
  - เปิด Guest OS ด้วย VMWare Work Station หรือ Player.
  - กำหนด network ของ guest เป็นแบบ “NAT Network”
  - กำหนดจำนวน core ของ CPU เท่ากับ 1 CPU กำหนด memory ขนาด 512MB หรือ 1GB
2. Start Guest OS (ถ้ามี dialog ถามว่า copy มาหรือ move มา ให้เลือก “I Copied it”)
3. Login

```
username: docker  
password: pass@word1
```

4. ตรวจสอบ IP Address ของเครื่องด้วยคำสั่ง

```
$ ifconfig  
// ใช้ ip จาก ems33
```

5. เปิด SSH client จาก host OS เพื่อ secure shell ไปยัง guest OS

```
$ ssh docker@<guest_ip_address>  
$ password: ...
```

# Docker Basic Command

- ตรวจสอบ version ของ docker

```
$ docker --version
```

- Run docker container แรก ... Hello, World!

```
$ docker run hello-world
```

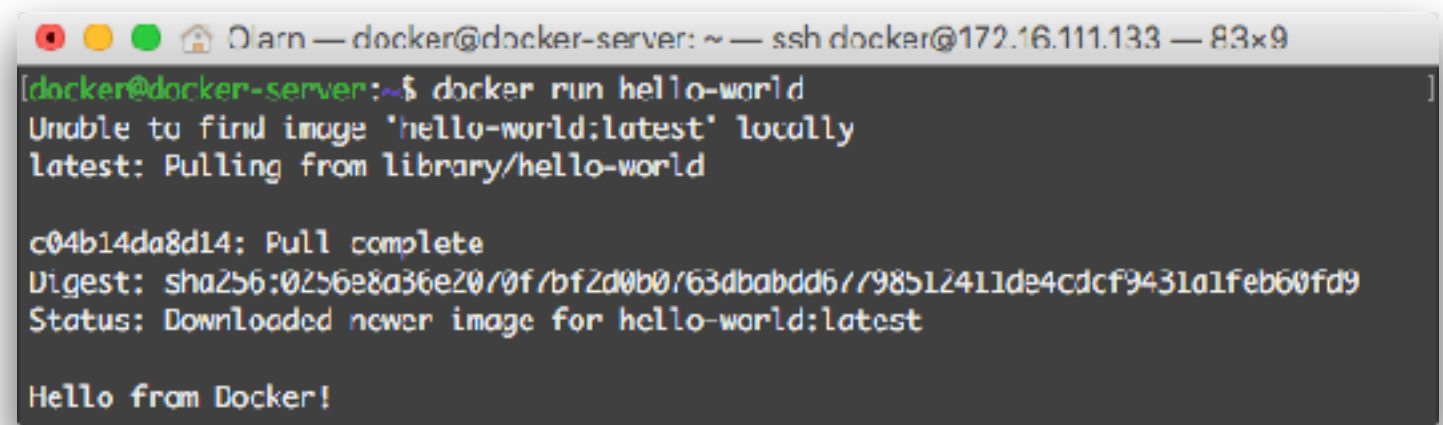
- ตรวจสอบ process ของ docker container

```
$ docker ps
```

```
$ docker ps -a
```

- ดูรายชื่อของ Docker Images

```
$ docker images
```

A terminal window titled 'Darn — docker@docker-server: ~ — ssh docker@172.16.111.133 — 83x9'. The terminal shows the command 'docker run hello-world' being executed. The output indicates that the image 'hello-world:latest' was not found locally and was pulled from the library. The pull is complete, and the container is running, outputting 'Hello from Docker!'.

```
[docker@docker-server:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world

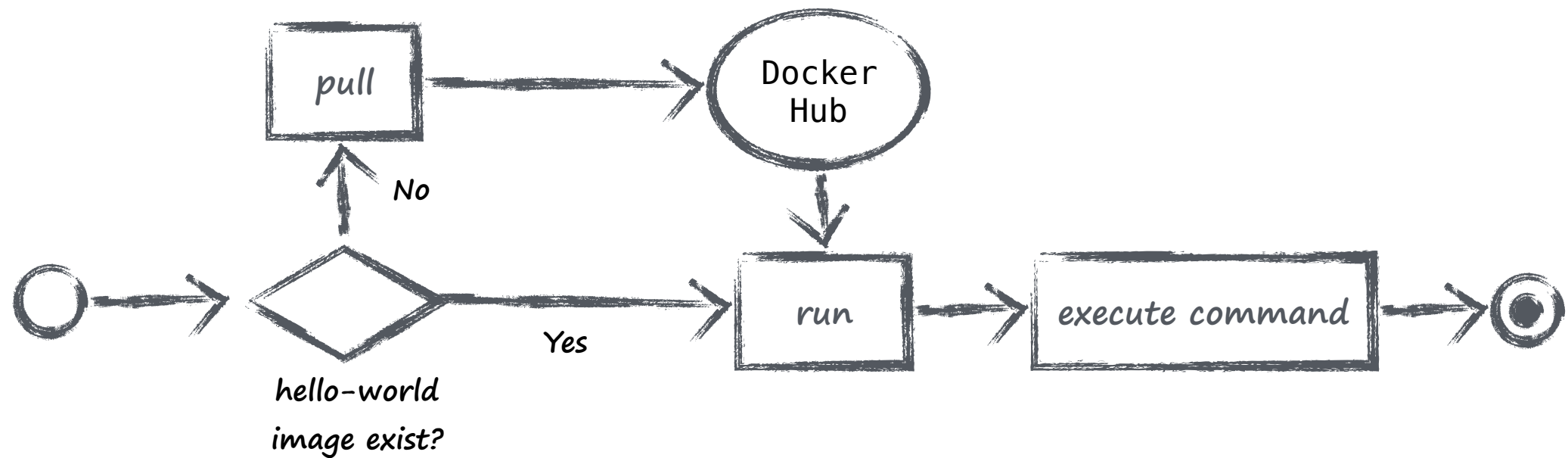
c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e20/0f/bf2d0b0/63dbabdd6/798512411de4cdcf9431a1feb60fd9
Status: Downloaded newer image for hello-world:latest

Hello from Docker!]
```

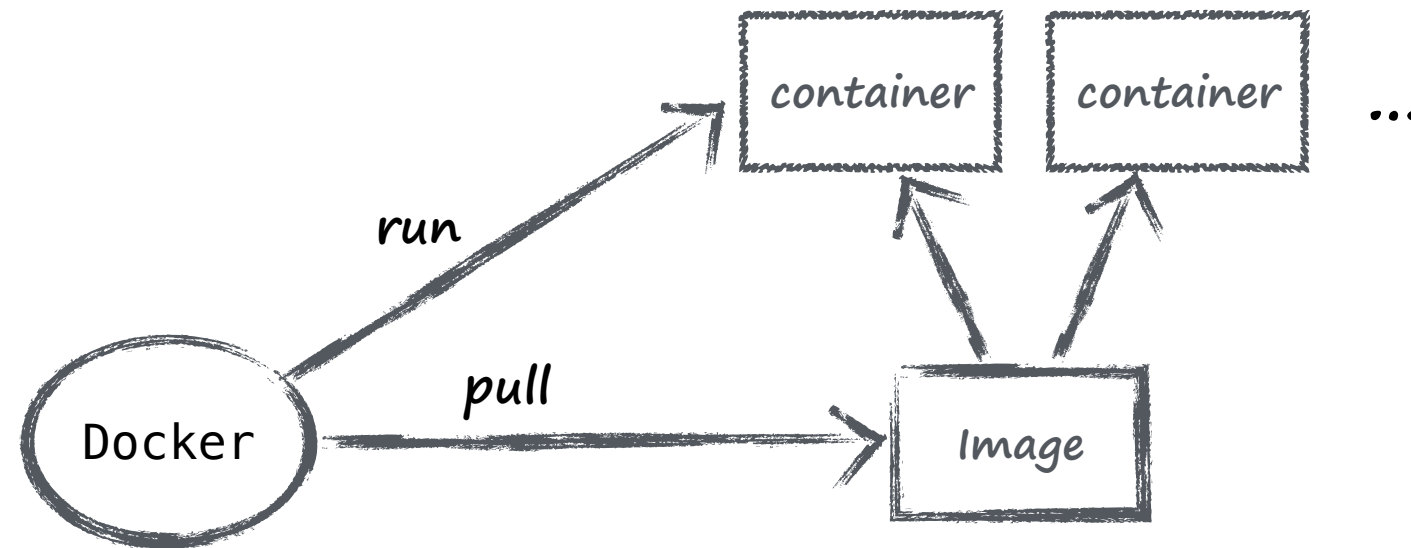
# 'docker run'

*Usage: docker [option] COMMAND [argument ...]*

```
$ docker run hello-world
```



# Container State



- long running process
- service
- daemonize

# Pull Image

- ค้นหา docker image

- Google
- Docker Hub
- Search Command

```
$ docker search ubuntu
```

- Pull

```
$ docker pull ubuntu
```

- *pull โดยระบุ tag*

```
$ docker pull ubuntu:14.04
```

- แสดงรายการ images

```
$ docker images
```

# Docker ps

- ตรวจสอบ process ของ container

```
$ docker ps
```

- ตรวจสอบ process ทั้งหมด ทั้งที่ทำงานอยู่และไม่ได้ทำงาน

```
$ docker ps -a
```

- เฉพาะเฉพาะ container ID

```
$ docker ps -q หรือ  
$ docker ps -aq
```

- แสดงพื้นที่ที่ container ใช้ด้วย

```
$ docker ps -a -s
```

หรือ

```
$ docker ps -as
```

- แสดงเฉพาะ container id ที่ exit ไปแล้ว

```
$ docker ps -aq
```

# Docker Container

- run ubuntu container

```
$ docker run ubuntu
```

- run container และ interactive กับ TTY (เหมือน ssh เข้าไปใน container)

```
$ docker run -i -t ubuntu  
root@<docker_container_id>:/#
```

- ปิด tty ของ container (เหมือน log off)

```
$ exit
```

- ลบ container ทันทีเมื่อ process ทำงานจบ

```
$ docker run -it --rm ubuntu
```



# Docker Run

- run ubuntu แบบ daemon

```
$ docker run -d -t -P --name myUbuntu ubuntu
```

โดยที่ -d คือ run แบบ daemon mode

-t คือ (optional) จำลอง TTY เพื่อให้ container ทำงานค้างไว้กรณีที่ไม่มี process หลัก

-P (p ใหญ่) คือ forward port ที่เปิดใน container ทั้งหมดโดยไม่ต้องกำหนดทีละ port

- ตรวจสอบ process

```
$ docker ps
```

- หยุดการทำงาน container

```
$ docker stop <container_id>
```

หรือผ่าน *docker name*

```
$ docker stop myUbuntu
```

- เริ่มการทำงาน container อีกครั้ง

```
$ docker start <container_id>
```

# Docker exec

- เป็นคำสั่งที่ใช้เพื่อ execute command ใน container เช่น ถ้ามี container ubuntu ทำงานอยู่ และเราต้องการ list file ภายใน folder /etc เราใช้คำสั่งดังนี้

```
$ docker exec myUbuntu ls -l /etc
```

- เราสามารถสั่ง list ของคำสั่งได้ เช่น

```
$ docker exec myUbuntu \  
touch test.txt; \  
echo hello world> test.txt; \  
tail test.txt
```

- ถ้าเราต้องการ ssh เข้าไปใน container ที่ทำงานอยู่ เราไม่จำเป็นต้องติดตั้ง ssh-server แต่ใช้คำสั่ง exec กับ bash เพื่อ interactive กับ container ได้โดยตรง

```
$ docker exec -it myUbuntu /bin/bash
```

# root

- user default ใน container นั้น เป็นได้หลากหลาย ขึ้นอยู่กับว่าเจ้าของ container นั้นกำหนดไว้เช่นไร
- เช่น กรณีที่เป็น ubuntu (official) user ที่ทำงานอยู่จะเป็น root แต่ถ้าเป็น jenkins (official) user ที่ทำงานอยู่จะเป็น jenkins
- ในบางกรณี เราต้องติดตั้งบางอย่างเพิ่มลงไป ใน container เช่น เมื่อ jenkins ต้องสั่งรัน unit test ของ node.js แต่บน container ไม่ได้ติดตั้ง node.js ไว้ เราต้องการสิทธิ์ root เพื่อติดตั้งโปรแกรม
- ในกรณีนี้ เราสามารถ exec โดยระบุ user ได้ เช่น

```
$ docker exec -it -u root myUbuntu /bin/bash
```

# Pause

- หยุดการทำงานชั่วคราว

```
$ docker pause <container_id>
```

- ทำงานต่อ

```
$ docker unpause <container_id>
```

# Docker Filter

- Filter

```
$ docker ps -f status=exited
```

```
$ docker ps -f status=running
```

- แสดงเฉพาะ container id

```
$ docker ps -a -q
```

- แสดงเฉพาะ container id ที่มี status เป็น exited

```
$ docker ps -aq -f status=exited
```

# Docker Commit

- Save container ไปเป็น image ใหม่

```
$ docker commit <container_id> new_image_name[:tag]
```

เช่น

```
$ docker commit 345 my_image
```

- ลอง list รายการ image ด้วยคำสั่ง

```
$ docker images
```

# Inspect & Stats

- ดู detail ของ docker container

```
$ docker inspect <container_id>
```

- Filter IP

```
$ docker inspect -f <params> <container_id> หรือ  
$ docker inspect --format=<params> <container_id>
```

เช่น

```
$ docker inspect -f {{.NetworkSettings.IPAddress}} 5a  
$ docker inspect --format='{{json .NetworkSettings}}' 5a
```

- ดู process ที่ทำงานอยู่ใน container

```
$ docker top <container_id>
```

- เราสามารถดู status ของ docker container ได้ด้วยคำสั่ง stats เช่น

```
$ docker stats
```

# Docker Remove

- ลบ container

```
$ docker rm <container_id>
```

- เราสามารถลบ container ได้ครั้งละหลายๆ container ได้ เช่น

```
$ docker rm container1 container2 container3
```

- ลบ container ทั้งหมดที่ status เป็น exited

```
$ docker rm $(docker ps -a -q -f status=exited)
```

- ลบ container แบบ force (ใช้กรณีที่ stop ไม่ได้)

```
$ docker rm -f $(docker ps -a -q)
```

- ลบ image

```
$ docker rmi <image_id>
```



# Labs

- สร้าง tomcat web server ด้วย docker
- note: curl localhost:8080

References :-

tomcat - [https://hub.docker.com/\\_/tomcat/](https://hub.docker.com/_/tomcat/)

# Volume

- mount volume ออกมานอก container
- ลบ container ข้อมูลไม่ถูกลบไปด้วย
- share ข้าม container ได้
- command

```
$ docker run -v <host_path>:<container_path> <container_id> [params]
```

เช่น

```
$ docker run -d --name node -v /home/docker/webapp:/webapp node
```

note: ถ้าใน parameter ไม่ได้ระบุ <container\_path> จะเป็นการ mount volume มาที่เครื่อง host ที่ docker กำหนดให้ (ดูรายละเอียดใน slide ถัดไป)

- การระบุ path บน windows จะใช้ Linux style เช่น

```
$ docker run -dt --name myubuntu -v /c/webapp:/webapp ubuntu
```

# Volume

- เมื่อสร้างหรือ mount volume แล้ว เราสามารถดูได้ว่า volume ที่ถูก mount อยู่ที่ไหน โดยดูจากข้อมูลที่ได้จากการ inspect

```
$ docker inspect myubuntu
```

เช่น

```
...
"Mounts": [
  {
    "Name": "fac362...80535",
    "Source": "/var/lib/docker/volumes/fac362...80535/_data",
    "Destination": "/webapp",
    "Driver": "local",
    "Mode": "",
    "RW": true,
    "Propagation": ""
  }
]
...
```

- สังเกตว่า Source จะเป็น /var/lib/... เพราะตอน run เราไม่ได้กำหนด source บน host
- เราสามารถ mount เฉพาะ file ได้ ไม่ต้อง mount ทั้ง folder ก็ได้
- เราสามารถกำหนด -v ได้มากกว่า 1 volume ตอน run เช่น

# Labs

- Lab 1 : run 2 container

```
$ docker run -d -t -v /tmp/data:/data ubuntu
```

- Lab 2 : mount volume ออกมานอก container

```
$ docker pull mongo
```

```
$ docker run -d -p 27017:27017 \
-v /tmp/data:/data/db mongo
```