# Docker

## Information security club course II

Chih-Hsuan Yang(SCC)

National Sun Yat-sen University

March 7, 2021
v0.1

# Outline

1. Container
   - Basic
   - Comparing Containers and Virtual Machines
   - Status
2. Dockerfile
   - Introduction
   - Write a Dockerfile
   - Write a Docker-compose
3. Docker commands
   - Basic commands
   - Exercises
4. Tiny project
   - Image deployment
   - X Window forwarding
   - Maybe: Cross platform
5. Advanced issues
6. References

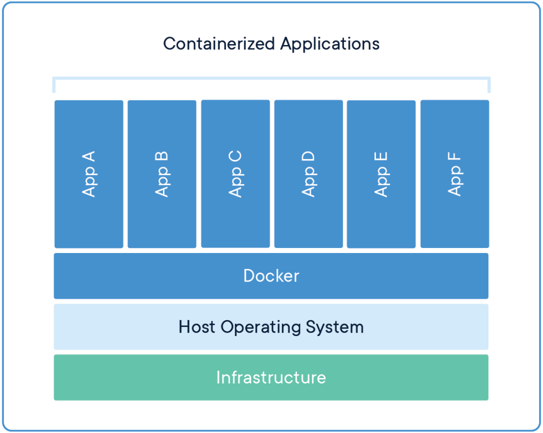
# Before the speech

- The ‘Learning Corner TA’ of operating system class
  - 18:00 → 21:00 (Thur.)
  - EC1013
  - From 25nd March 2021
- The other time for OS problems
  - 18:00 → 21:00 (Fri.)
  - EC3034
  - Send an email before you come.
- zxc25077667@protonmail.com
- We learn more only if you ask.

# Container

# Big idea

Containerize

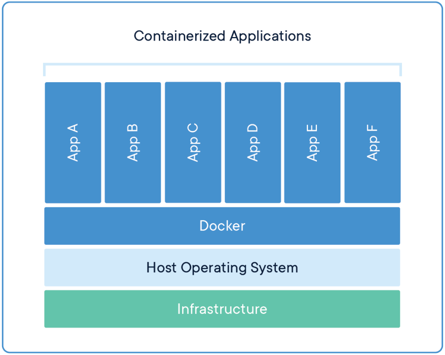


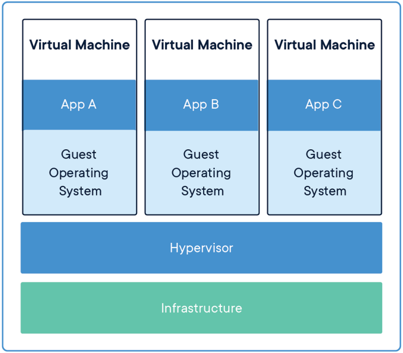
[1]

# Big idea

[2]

# Compare with virtule machines

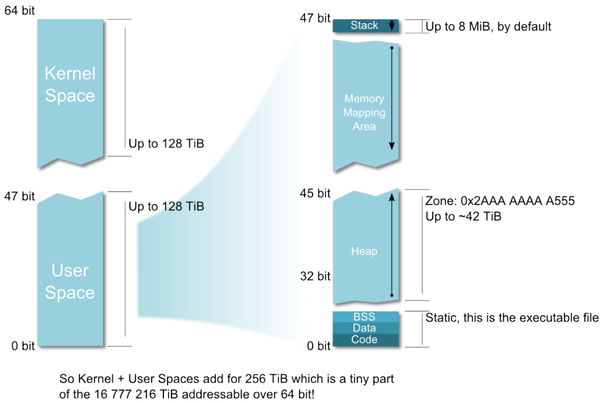




Wait, what is Hypervisor?

# So, what is share kernel?

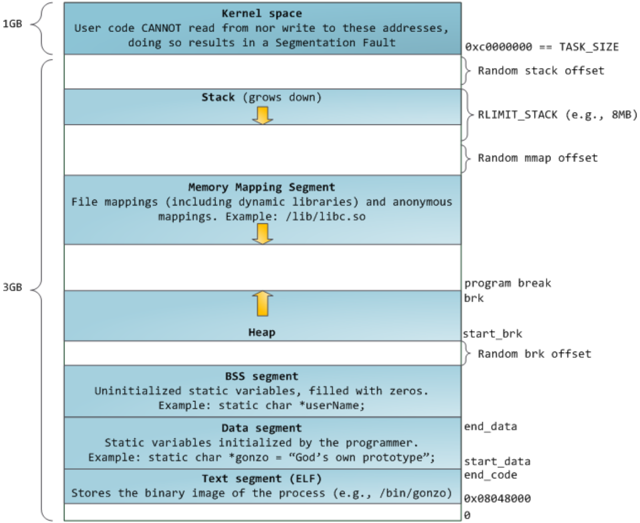
Let's recall the OS 101 course



[3]

# So, what is share kernel?

## The 32-bits memory layout


1GB
Kernel space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault
0xc0000000 == TASK_SIZE
Random stack offset
Stack (grows down)
RLIMIT_STACK (e.g., 8MB)
Random mmap offset
Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so
3GB
program break
brk
Heap
start_brk
Random brk offset
BSS segment
Uninitialized static variables, filled with zeros. Example: static char *userName;
end_data
Data segment
Static variables initialized by the programmer. Example: static char *gonzo = "God's own prototype";
start_data
end_code
Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)
0x08048000
0

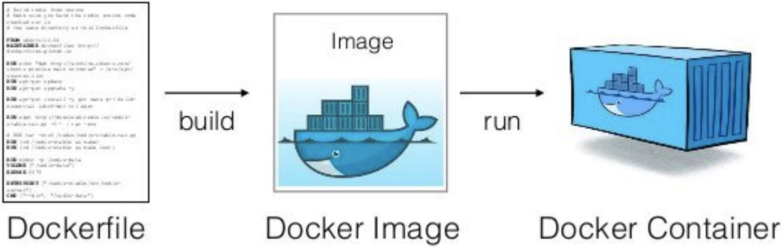
[4]

# What is HYPERVISOR?

Virtual machine monitor

Monitor? Will introduce in the OS.

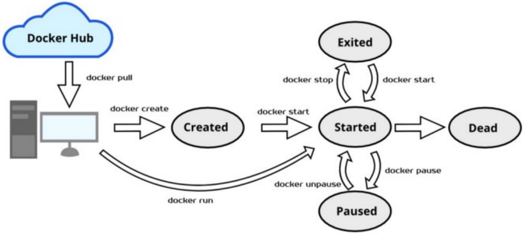
# Image and Container



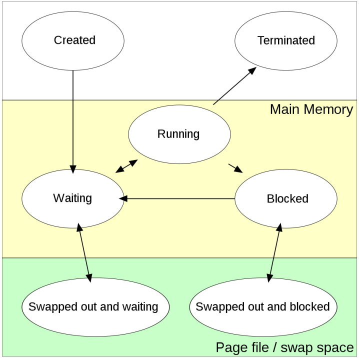
[5]

Like a program in execution is called a process.
An image in execution is called a container.

# Lifecycle



(a) Container's Lifecycle



(b) Process's Lifecycle

[6, 7]

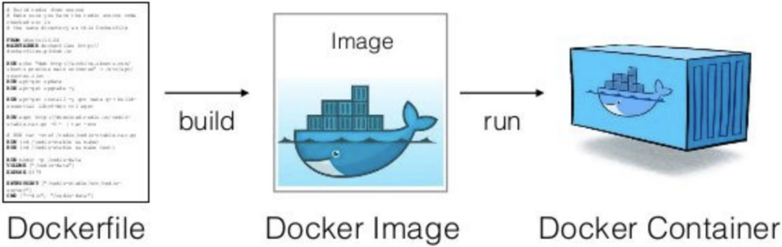
# Dockerfile

# What is Dockerfile?

Definition:

A text document that contains all the commands a user could call on the command line to assemble an image.

# Image and Container again



[5]

Like a program in execution is called a process.
An image in execution is called a container.

# DockerHub

A public **images** hub. We push/pull **images** from it by default. We push an image and pull an image rather than a container.

Make sense, right?

# Dockerfile 101

1. FROM
2. RUN, CMD, ENTRYPOINT
3. EXPOSE
4. ENV
5. ADD, COPY
6. VOLUME
7. USER
8. WORKDIR
9. ONBUILD

https://docs.docker.com/engine/reference/builder/

# Dockerfile's Lab

```
FROM ubuntu

ENV KFC=EGG_TART

RUN apt update && apt install -y x11vnc xvfb firefox

RUN useradd -m user1 --uid=1000 && \
    echo "user1:Ch@ng3_m3" | chpasswd
USER user1:1000
WORKDIR /home/user1

RUN bash -c 'echo "firefox" >> /home/user1/.bashrc' && \
    mkdir ~/.vnc && \
    x11vnc -storepasswd nsysuisc ~/.vnc/passwd

EXPOSE 5900
CMD    ["x11vnc", "-forever", "-usepw", "-create"]
```

src/dockerfile101/Dockerfile

# Docker-compose 101

```
version: "3.9"
services:

  redis:
    image: redis:alpine
    ports:
      - "6379"
    networks:
      - frontend
    deploy:
      replicas: 2
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
```

src/docker–compose101/example.yml

https://docs.docker.com/compose/compose-file/compose-file-v3/

# Docker-compose's Lab

```
#!/bin/bash

git clone https://github.com/docker/example-voting-app.git
cd example-voting-app

# Initialize docker swarm
sudo docker swarm init

# Deploy
sudo docker stack deploy --compose-file docker-stack.yml vote
```

src/docker–compose101/auto_build.sh

# Docker-compose's Lab requirements

```bash
#!/bin/bash

# Install docker compose
sudo curl -L "https://github.com/docker/compose/releases/download/1.28.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

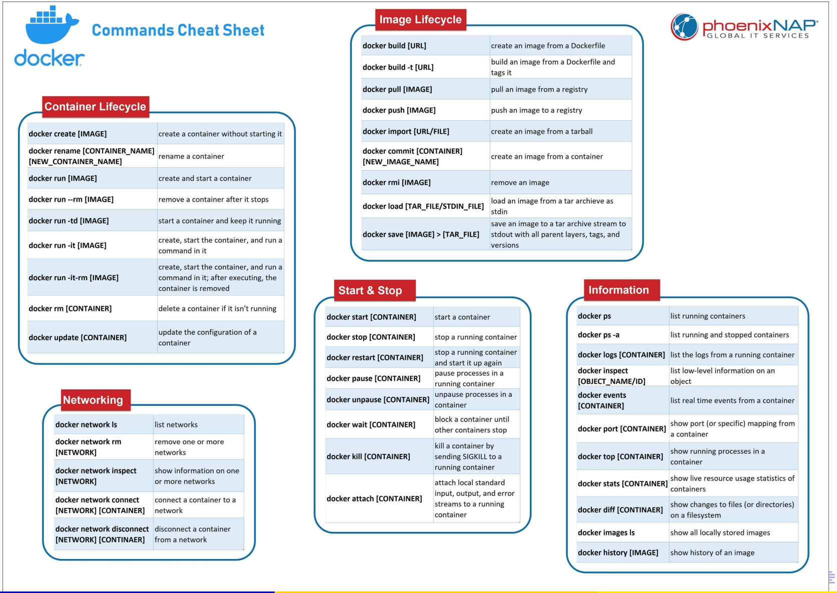
src/docker–compose101/install_docker_compose.sh

### Stop all containers:

sudo docker stop $(sudo docker ps -a -q)
sudo docker stack rm <name>

# Docker commands

# Basic



## Commands Cheat Sheet

### Container Lifecycle

| | |
|---|---|
| docker create [IMAGE] | create a container without starting it |
| docker rename [CONTAINER_NAME] [NEW_CONTAINER_NAME] | rename a container |
| docker run [IMAGE] | create and start a container |
| docker run --rm [IMAGE] | remove a container after it stops |
| docker run -td [IMAGE] | start a container and keep it running |
| docker run -it [IMAGE] | create, start the container, and run a command in it |
| docker run -it-rm [IMAGE] | create, start the container, and run a command in it; after executing, the container is removed |
| docker rm [CONTAINER] | delete a container if it isn't running |
| docker update [CONTAINER] | update the configuration of a container |

### Networking

| | |
|---|---|
| docker network ls | list networks |
| docker network rm [NETWORK] | remove one or more networks |
| docker network inspect [NETWORK] | show information on one or more networks |
| docker network connect [NETWORK] [CONTAINER] | connect a container to a network |
| docker network disconnect [NETWORK] [CONTINAER] | disconnect a container from a network |

### Image Lifecycle

| | |
|---|---|
| docker build [URL] | create an image from a Dockerfile |
| docker build -t [URL] | build an image from a Dockerfile and tags it |
| docker pull [IMAGE] | pull an image from a registry |
| docker push [IMAGE] | push an image to a registry |
| docker import [URL/FILE] | create an image from a tarball |
| docker commit [CONTAINER] [NEW_IMAGE_NAME] | create an image from a container |
| docker rmi [IMAGE] | remove an image |
| docker load [TAR_FILE/STDIN_FILE] | load an image from a tar archieve as stdin |
| docker save [IMAGE] > [TAR_FILE] | save an image to a tar archive stream to stdout with all parent layers, tags, and versions |

### Start & Stop

| | |
|---|---|
| docker start [CONTAINER] | start a container |
| docker stop [CONTAINER] | stop a running container |
| docker restart [CONTAINER] | stop a running container and start it up again |
| docker pause [CONTAINER] | pause processes in a running container |
| docker unpause [CONTAINER] | unpause processes in a container |
| docker wait [CONTAINER] | block a container until other containers stop |
| docker kill [CONTAINER] | kill a container by sending SIGKILL to a running container |
| docker attach [CONTAINER] | attach local standard input, output, and error streams to a running container |

### Information

| | |
|---|---|
| docker ps | list running containers |
| docker ps -a | list running and stopped containers |
| docker logs [CONTAINER] | list the logs from a running container |
| docker inspect [OBJECT_NAME/ID] | list low-level information on an object |
| docker events [CONTAINER] | list real time events from a container |
| docker port [CONTAINER] | show port (or specific) mapping from a container |
| docker top [CONTAINER] | show running processes in a container |
| docker stats [CONTAINER] | show live resource usage statistics of containers |
| docker diff [CONTINAER] | show changes to files (or directories) on a filesystem |
| docker images ls | show all locally stored images |
| docker history [IMAGE] | show history of an image |

[8]

# Full cheat sheet

Next page reference: [9]

## 1. Containers

A lightweight virtual OS that run processes in full isolation.

### 1.1 Lifecycle

- docker create creates a container but does not start it.
- docker rename allows the container to be renamed.
- docker run creates and starts a container in one operation.
- docker rm deletes a container.
- docker update updates a container's resource limits.
- docker run --rm : remove the container after it stops.
- docker run -v $HOSTDIR:$DOCKERDIR: map the directory ($HOSTDIR) on the host to a
- docker container ($DOCKERDIR).
- docker rm –v: remove the volumes associated with the container.
- docker run --log-driver=syslog : run docker with a custom log driver.

### 1.2 Starting and Stopping

- docker start starts a container so it is running.
- docker stop stops a running container.
- docker restart stops and starts a container.
- docker pause pauses a running container, "freezing" it in place.
- docker unpause will unpause a running container.
- docker wait blocks until running container stops.

docker kill sends a SIGKILL to a running container.

docker attach will connect to a running container.

### 1.3 CPU Constraints

CPU can be limited either using a percentage over all CPUs, or by using specific cores.

- -c or cpu-shares: 1024 means 100% of the CPU, so if we want the container to take 50% of all CPU cores, we should specify 512 for instance, docker run -ti --c 512 ...cpuset-cpus
- : use only some CPU cores, for instance, docker run -ti --cpuset-cpus=0,4,6 ...

### 1.4 Memory Constraints

Memory can be limited using –m flag, for instance, docker run -it -m 300M ubuntu:14.04 /bin/bash

### 1.5 Capabilities

cap-add and cap-drop: Add or drop linux capabilities.

- Mount a FUSE based filesystem:
  - docker run --rm -it --cap-add SYS_ADMIN --device /dev/fuse sshfs
- Give access to a single device:
  - docker run -it --device=/dev/ttyUSB0 debian bash
- Give access to all devices:
  - docker run -it --privileged -v /dev/bus/usb:/dev/bus/usb debian bash

### 1.6 Info

- docker ps shows running containers.
- docker logs gets logs from container. (You can use a custom log driver, but logs is only available for json-fileand journald in 1.10).
- docker inspect looks at all the info on a container (including IP address).
- docker events gets events from container.
- docker port shows public facing port of container.
- docker top shows running processes in container.
- docker stats shows containers' resource usage statistics.
- docker diff shows changed files in the container's FS.
- docker ps –ashows running and stopped containers

### 1.7 Import / Export

- docker cp copies files or folders between a container and the local filesystem.
- docker export turns container filesystem into tarball archive stream to STDOUT.

### 1.8 Executing Commands

docker exec to execute a command in container.

## 2. Images

A template or blueprint for docker containers.

### 2.1 Lifecycle

- docker images shows all images.
- docker import creates an image from a tarball.
- docker build creates image from Dockerfile.
- docker commit creates image from a container, pausing it temporarily if it is running.
- docker rmi removes an image.
- docker load loads an image from a tar archive as STDIN, including images and tags (as of 0.7).
- docker save saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions (as of 0.7).

### 2.2. Info

- docker history shows history of image.
- docker tag tags an image to a name (local or registry).

### 2.3. Cleaning up

- docker rmi remove specific images.
- docker-gc a toolto clean up images that are no longer used by any containers in a safe manner.

### 2.4. Load/Save image

- docker load < my_image.tar.gz load an image from file
- docker save my_image:my_tag | gzip > my_image.tar.gz save an existing image

### 2.5. Import/Export container

- cat my_container.tar.gz | docker import - my_image:my_tag import a container as an image from file
- docker export my_container | gzip > my_container.tar.gz export an existing container

## 3. Networks

A small def goes here

### 3.1. Lifecycle

- docker network create
- docker network rm

### 3.2. Info

- docker network ls
- docker network inspect

### 3.3. Connection

- docker network connect
- docker network disconnect

## 4. Registry & Repository

A repository is a hosted collection of tagged images that together create the file system for a container.
A registry is a host -- a server that stores repositories and provides an HTTP API for managing the uploading and downloading of repositories.
Docker.com hosts its own index to a central registry which contains a large number of repositories.

- docker login to login to a registry.
- docker logout to logout from a registry.
- docker search searches registry for image.
- docker pull pulls an image from registry to local machine.
- docker push pushes an image to the registry from local machine.

## 5. Volumes

Docker volumes are free-floating filesystems. They don't have to be connected to a particular container. You should use volumes mounted from data-only containers for portability.

### 5.1. Lifecycle

- docker volume create
- docker volume rm

### 5.2. Info

- docker volume ls
- docker volume inspect

## 6. Exposing ports

- docker run -p 127.0.0.1:$HOSTPORT:$CONTAINER-
- PORT --name CONTAINER -t docker_image mapping the container port to the host port using –p
- EXPOSE <CONTAINERPORT>expose port CONTAINERPORT at runtime (see dockerfile)
- docker port CONTAINER $CONTAINERPORT check the mapped port

## 7. Tips

### 7.1. Get IP address

- docker inspect some_docker_id | grep IPAddress | cut -d '"' -f 4

or install jq:

- docker inspect some_docker_id | jq -r '.[0].NetworkSettings.IPAddress'

or using a go template:

- docker inspect -f '{{ .NetworkSettings.IPAddress }}' <container_name>

### 7.2. Get port mapping

```
docker inspect -f '{{range $p, $conf := .NetworkSettings.Ports}} {{$p}} -> {{(index $conf 0).HostPort}} {{end}}' <containername>
```

### 7.3. Find containers by regular expression

```
for i in $(docker ps -a | grep "REGEXP_PATTERN" | cut -f1 -d" "); do echo $i; done
```

### 7.4. Get Environment Settings

```
docker run --rm ubuntu env
```

### 7.5. Kill running containers

```
docker kill $(docker ps -q)
```

### 7.6. Delete old containers

```
docker ps -a | grep 'weeks ago' | awk '{print $1}' | xargs docker rm
```

### 7.7. Delete stopped containers

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

### 7.8. Delete dangling images

```
docker rmi $(docker images -q -f dangling=true)
```

### 7.9. Delete all images

```
docker rmi $(docker images -q)
```

### 7.10. Delete dangling volumes

```
docker volume rm $(docker volume ls -q -f dangling=true)
```

# Exercises

# Tiny project

# Single-responsibility principle

*Why we should decompose this project into tiny-tiny parts?*

Wikipedia:

The single-responsibility principle (SRP) is a computer-programming principle that states that every class in a computer program should have responsibility over a single part of that program's functionality, which it should encapsulate. All of that module, class or function's services should be narrowly aligned with that responsibility [10].

# Our PWN Labs

`https://github.com/giantbranch/pwn_deploy_chroot`

Steps:

1. git clone https://github.com/giantbranch/pwn_deploy_chroot.git
2. Put the PWN binary in to the bin/
3. python initialize.py
4. sudo docker-compose up –build -d

# Wine with X11

```
FROM debian

COPY . .

RUN dpkg --add-architecture i386
RUN apt-get update && \
    apt install wine wine32 wine64 libwine libwine:i386
   fonts-wine -y

ENV DISPLAY :0
```

src/wine/Dockerfile

## Build

sudo docker build -t wine0 .

# Wine with X11

```
sudo docker run -it --rm --env="DISPLAY" \
    --volume="${XAUTHORITY:-${HOME}/.Xauthority}:/root/.Xauthority:ro" \
    --volume="/tmp/.X11-unix:/tmp/.X11-unix" \
    --volume="winehome:/home/wineuser" \
    -v "${pwd}":"/line" \
    --hostname="$(hostname)" wine0 \
    wine LineInst.exe
```

src/wine/run.sh

# Raspbian in Docker

We need a Qemu supervise it.

```
FROM navikey/raspbian-buster

COPY qemu-system-arm /usr/bin/qemu-system-arm

ENV TZ=Asia/Taipei

RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && \
    echo $TZ > /etc/timezone && \
    apt update -y && \
    apt install libapparmor-dev build-essential python3 tmux zsh vim curl git -y

ADD ./work /work
WORKDIR /work
```

src/raspbian/Dockerfile

# Toys

Docker-OSX: `https://github.com/sickcodes/Docker-OSX`
x11docker: `https://github.com/mviereck/x11docker`

# Advanced issues

# Some advanced issues

- Security
  - Privileged
  - Escape
- Scalability
  - SDN (Software defined network)
  - SDS (Software defined storage)
  - Dynamic infrastructure / Load balance
- CI/CD

# References

# References I

- *What is a Container?* URL: https://www.docker.com/resources/what-container.
- *packhelp*. URL: https://packhelp.com/plain-shipping-box/.
- *64-bit memory layout*. URL: https://www.berthon.eu/wiki/foss:wikishelf:linux:memory.
- *32-bit memory layout*. URL: https://unix.stackexchange.com/questions/31407/how-does-forking-affect-a-processs-memory-layout.
- *Build a Docker Image just like how you would configure a VM*. URL: https://medium.com/platformer-blog/practical-guide-on-writing-a-dockerfile-for-your-application-89376f88b3b5.
- *Introduction To The Docker Life Cycle*. URL: https://medium.com/faun/introduction-to-docker-life-cycle-3bf3aeba883.
- *Process state, wiki*. URL: https://en.wikipedia.org/wiki/Process_state.
- *List Of Docker Commands: Cheat Sheet*. URL: https://phoenixnap.com/kb/list-of-docker-commands-cheat-sheet.
- *Docker Commands - Complete Cheat Sheet*. URL: https://linoxide.com/linux-how-to/docker-commands-cheat-sheet/.

# References II

- *Single-responsibility principle*. URL: https://en.wikipedia.org/wiki/Single-responsibility_principle.