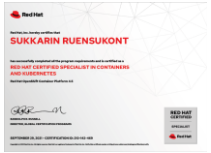

Sukkarin Ruensukont
MFEC Public Company Limited

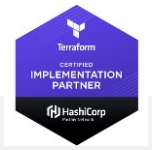


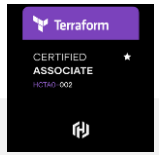


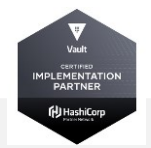


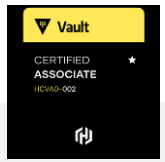






# Sukkarin Ruensukont

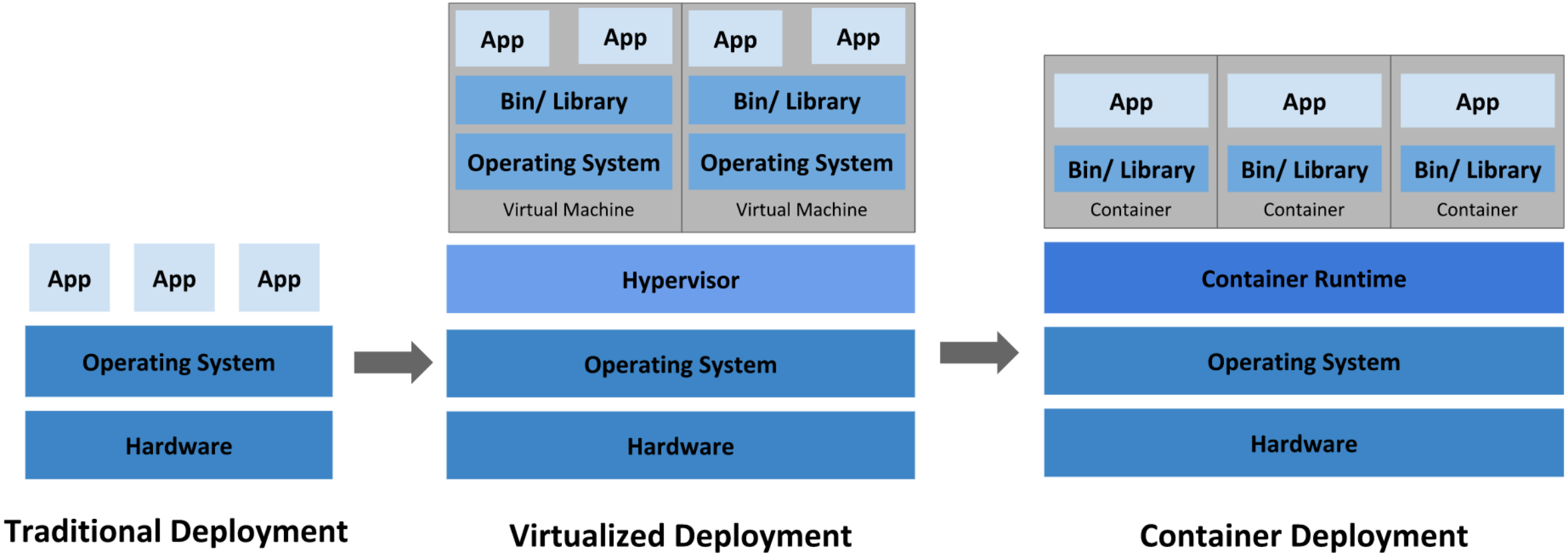
sukkarin@mfec.co.th

นั่งเล่น NGINX


MFEC
Public Company Limited

# Container Technology

# History of Deployment




https://kubernetes.io/docs/concepts/overview/

# Container and Docker

## Container technology

- Docker
- CRI-O
- CoreOS rkt
- Turbo Containers
- Intel Clear Containers
- lmctfy

# Container Technology

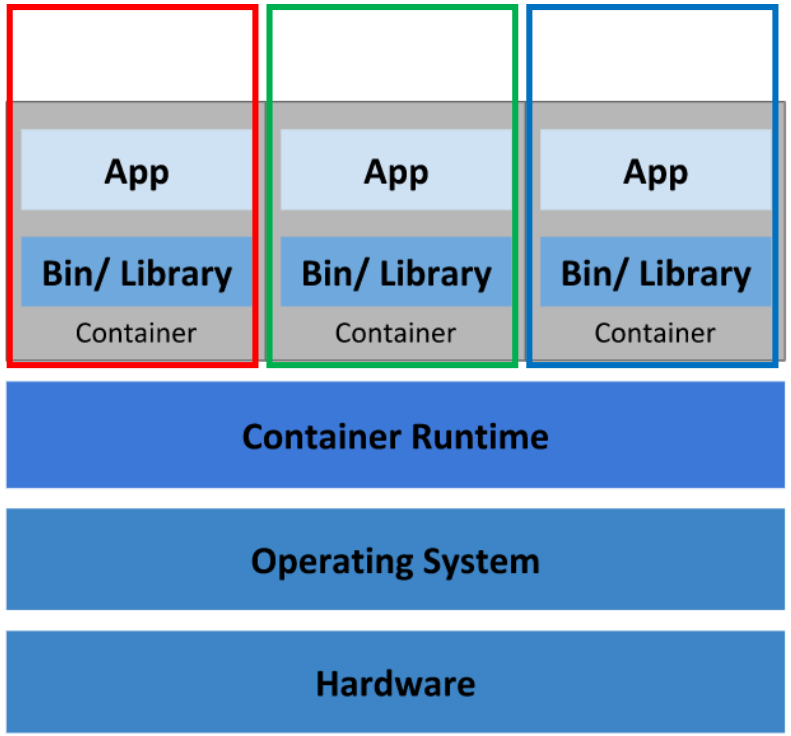
- Run “Process”
- Container process is separated from host process
- Container process is separated from other Container’s process

# Container Technology

```
/usr/bin/containerd
|  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.
|  |   \_ nginx: master process nginx -g daemon off;
|  |   |    \_ nginx: worker process
|  |   \_ mysqld
|  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.
|      \_ nginx: master process nginx -g daemon off;
|          \_ nginx: worker process
```

# Namespace

- Docker uses a technology called namespaces to provide the isolated workspace called the container. When you run a container, **Docker creates a set of namespaces for that container**




**Container Deployment**

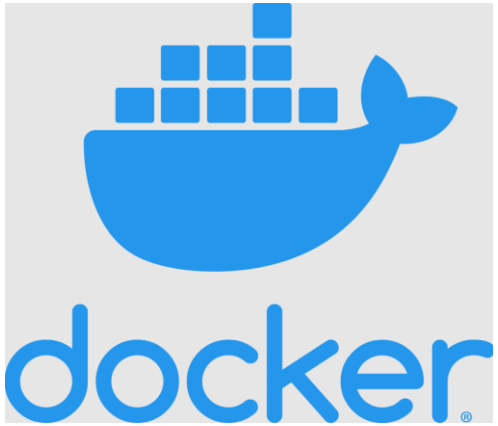
# Namespaces

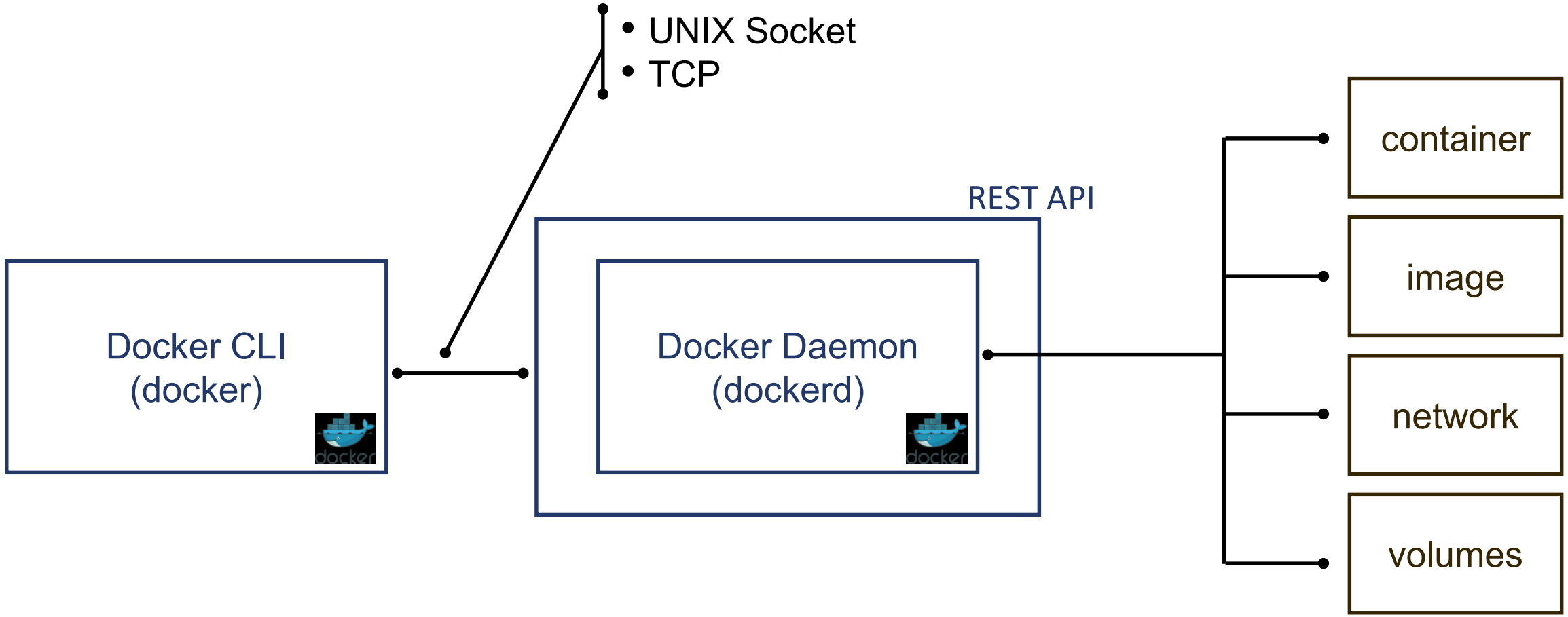
- pid: Process isolation
- net: Network interfaces
- ipc: InterProcess Communication
- mnt: Mount
- uts: Unix Timesharing System(kernel and version identifiers)

# Docker

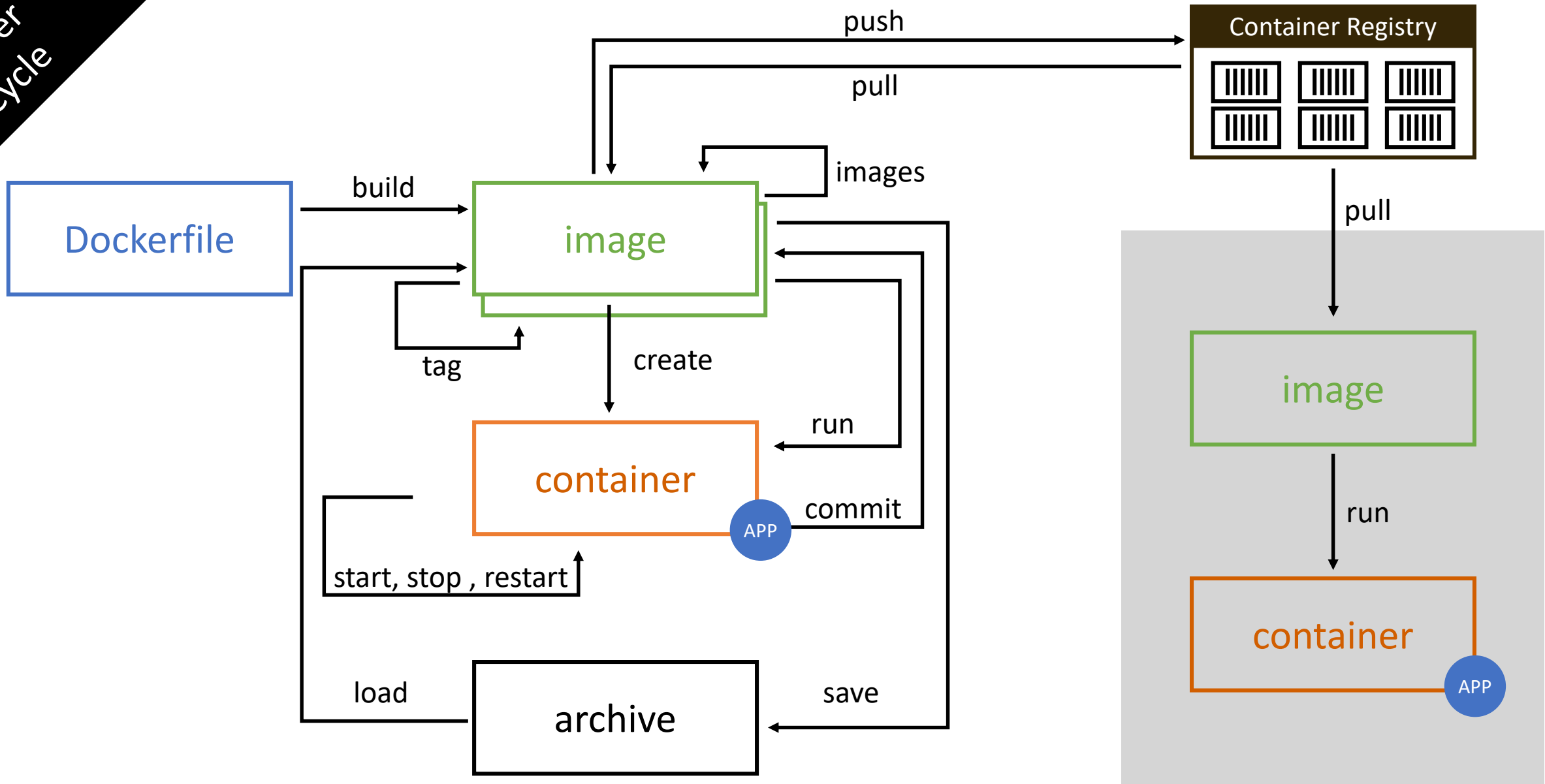
- Docker is written in the Go programming language and takes advantage of several features of the Linux kernel to deliver its functionality.

- Docker uses a technology called namespaces to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container.




• UNIX Socket
• TCP
REST API
Docker CLI
(docker)
Docker Daemon
(dockerd)
container
image
network
volumes

# Container Life Cycle



# Docker Commands

Action in DOCKER

info, login, logout, manifest, search
system, version


image push
image pull

container start
container stop
container kill
container restart
container pause
container unpause
container exec

container attach
container cp
container inspect
container prune
container rename
container rm
container update

container

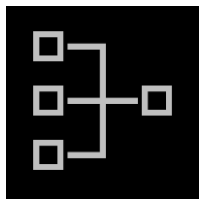
container export
container create
container run
container commit
image import

image

image load
image save
image tag
image rm
image history

image ls
image prune
image inspect

container ls
container port
container stats
container logs
Container top



network

volume

image build

Dockerfile

default: docker.io

default: latest

```
<image registry>/library/<image name>:tag

<image registry>/library/<image name>@digest
```

<optional>

# docker image pull

**docker image pull** เป็นคำสั่งที่ใช้ดึง image ที่ต้องการจาก Container Image Registry มาเก็บไว้ที่เครื่องที่เรียกใช้งาน

```
]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE

]# docker image pull nginxdemos/nginx-hello:plain-text
plain-text: Pulling from nginxdemos/nginx-hello
89d9c30c1d48: Pull complete
fa1bcf7bbd3d: Pull complete
5a75ae0625b2: Pull complete
d6af3bb15c09: Pull complete
Digest: sha256:71b4a63b6d31ae846c431d0dc7134bc0ae490fe4698eebf3ed297668db3b2939
Status: Downloaded newer image for nginxdemos/nginx-hello:plain-text
docker.io/nginxdemos/nginx-hello:plain-text

]# docker image ls
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
nginxdemos/nginx-hello      plain-text          c93b70031dd4        7 months ago        21.2MB
```

default: docker.io

# docker image push

**docker image push** เป็นคำสั่งที่ใช้ส่ง image จาก เครื่องที่เรียกใช้งานไปที่เก็บที่ Container Image Registry

```
]# docker image ls
REPOSITORY                                    TAG                 IMAGE ID            CREATED             SIZE
nginxdemos/nginx-hello                        plain-text          c93b70031dd4        7 months ago        21.2MB
registry.example.com/damrongsak/nginx-hello   latest              c93b70031dd4        7 months ago        21.2MB

]# docker image push registry.example.com/damrongsak/nginx-hello
The push refers to repository [registry.example.com/damrongsak/nginx-hello]
3c9fe36a825b: Pushed
fc971f0b69a3: Pushed
d8258aad6f8a: Pushed
77cae8ab23bf: Pushed
latest: digest: sha256:71b4a63b6d31ae846c431d0dc7134bc0ae490fe4698eebf3ed297668db3b2939 size: 1153
```

default: latest

Action in DOCKER

info, login, logout, manifest, search
system, version


container start
container stop
container kill
container restart
container pause
container unpause
container exec

image push
image pull

container attach
container cp
container inspect
container prune
container rename
container rm
container update

container

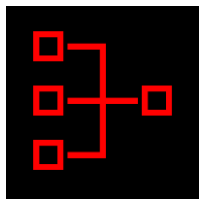
container export
container create
container run
container commit
image import

image

image load
image save
image tag
image rm
image history

image ls
image prune
image inspect

container ls
container ps
container port
container stats
container logs
Container top



network

volume

image build

Dockerfile

# docker image ls

**docker image ls** เป็นคำสั่งที่ใช้แสดงรายการ image ที่อยู่ในเครื่อง

```
]# docker image ls
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
nginx                     latest              c39a868aad02        8 days ago          133MB
busybox                   latest              f0b02e9d092d        4 weeks ago         1.23MB
centos                    latest              0d120b6ccaa8        3 months ago        215MB
nginxdemos/nginx-hello    latest              03472b12d393        10 months ago       21.2MB
busybox                   1.28                8c811b4aec35        2 years ago         1.15MB
]# docker image ls centos
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              latest              0d120b6ccaa8        3 months ago        215MB
]# docker image ls --digests centos
REPOSITORY          TAG                 DIGEST                                                                    IMAGE
ID                  CREATED             SIZE
centos              latest              sha256:76d24f3ba3317fa945743bb3746fbaf3a0b752f10b10376960de01da70685fbd
0d120b6ccaa8        3 months ago        215MB
```

Image ที่ถูกเก็บใน v2 เป็นต้นไปจะที่ค่าที่เป็น content-addressable identifier เรียกว่า digest ซึ่งเป็นค่าที่ถูกสร้างขึ้นสำหรับ image นั้น ๆ และจะไม่เปลี่ยนแปลง จึงใช้เป็นค่าที่จะระบุถึง image ตัวนั้น ๆ ได้เจาะจงถึงแม้ tag จะถูกเปลี่ยนแปลงไปก็ตาม

# docker image tag

**docker image tag** เป็นคำสั่งกำหนดค่าที่ใช้สำหรับอ้างอิงถึง image นั้น ๆ และยังใช้เป็นสื่อความหมายถึงความแตกต่างของ image ที่ชื่อเดียวกัน แต่มีความต่างกัน และ tag มันจะนิยมตั้งตาม version หรือ release ของ image ตัวนั้น
การ tag ยังใช้ประโยชน์เพื่อต้องการจะ push image ไปยัง private registry ได้ด้วย

```
]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              c39a868aad02        8 days ago          133MB
]# docker image tag nginx nginx:myversion
]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              c39a868aad02        8 days ago          133MB
nginx               myversion           c39a868aad02        8 days ago          133MB
]# docker image tag nginx:myversion repo.example.com/nginx:4.0
]# docker image ls
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
nginx                       latest              c39a868aad02        8 days ago          133MB
nginx                       myversion           c39a868aad02        8 days ago          133MB
repo.example.com/nginx      4.0                 c39a868aad02        8 days ago          133MB
```

# docker image prune

**docker image prune** เป็นคำสั่งที่ใช้ลบ image ออกจากเครื่อง โดย default หากไม่ระบุ option ก็จะสั่งลบแต่ dangling image เท่านั้น แต่ถ้าต้องการลบ image ทุกตัวที่ไม่มีการเรียกใช้งานอยู่ในขณะนั้นก็เพียงเพิ่ม option -a หรือ --all

```
]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              c39a868aad02        8 days ago          133MB
busybox             latest              f0b02e9d092d        4 weeks ago         1.23MB
centos              latest              0d120b6ccaa8        3 months ago        215MB
]# docker image prune -a
WARNING! This will remove all images without at least one container associated to them.
Are you sure you want to continue? [y/N] y
Deleted Images:
untagged: centos:latest
untagged: centos@sha256:76d24f3ba3317fa945743bb3746fbaf3a0b752f10b10376960de01da70685fbd
deleted: sha256:0d120b6ccaa8c5e149176798b3501d4dd1885f961922497cd0abef155c869566
### truncate output ####
untagged: busybox:latest
untagged: busybox@sha256:a9286defaba7b3a519d585ba0e37d0b2cbee74ebfe590960b0b1d6a5e97d1e1d
deleted: sha256:f0b02e9d092d905d0d87a8455a1ae3e9bb47b4aa3dc125125ca5cd10d6441c9f
deleted: sha256:d2421964bad195c959ba147ad21626ccddc73a4f2638664ad1c07bd9df48a675

Total reclaimed space: 349.2MB
```

# docker image inspect

**docker image inspect** เป็นคำสั่งที่ใช้แสดงข้อมูลรายละเอียดต่าง ๆ ของ image

```
]# docker image inspect centos
[
    {
        "Id": "sha256:0d120b6ccaa8c5e149176798b3501d4dd1885f961922497cd0abef155c869566",
        "RepoTags": [
            "centos:latest"
        ],
        "RepoDigests": [
            "centos@sha256:76d24f3ba3317fa945743bb3746fbaf3a0b752f10b10376960de01da70685fbd"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2020-08-10T18:19:49.837885498Z",
        "Container": "3b04ecd9fb2d3f921f12d858edf5f3a6aa7c36c8e1e6f74bd32555fd4d7f7da2",
         "ContainerConfig": {
            "Hostname": "3b04ecd9fb2d",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
```

next >>

# docker image inspect


```
 "Architecture": "amd64",
        "Os": "linux",
        "Size": 215102299,
        "VirtualSize": 215102299,
        "GraphDriver": {
            "Data": {
                "MergedDir":
"/var/lib/docker/overlay2/f5d5717d8c7f6e44f0de673e7aa74a632f6004d3f042af5f4bcfe7d70c899677/merged",
                "UpperDir":
"/var/lib/docker/overlay2/f5d5717d8c7f6e44f0de673e7aa74a632f6004d3f042af5f4bcfe7d70c899677/diff",
                "WorkDir":
"/var/lib/docker/overlay2/f5d5717d8c7f6e44f0de673e7aa74a632f6004d3f042af5f4bcfe7d70c899677/work"
            },
            "Name": "overlay2"
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:291f6e44771a7b4399b0c6fb40ab4fe0331ddf76eda11080f052b003d96c7726"
            ]
        },
        "Metadata": {
            "LastTagTime": "0001-01-01T00:00:00Z"
        }
    }
]
```

# docker image history

**docker image history** เป็นคำสั่งที่ใช้แสดงข้อมูลรายละเอียดในการสร้าง image ทำให้ทราบ instruction ในการสร้าง image ได้

```
]# docker image history nginx
IMAGE          CREATED       CREATED BY                                      SIZE      COMMENT
bc9a0695f571   2 days ago    /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
<missing>      2 days ago    /bin/sh -c #(nop)  STOPSIGNAL SIGQUIT           0B
<missing>      2 days ago    /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>      2 days ago    /bin/sh -c #(nop)  ENTRYPOINT ["/docker-entr…   0B
<missing>      2 days ago    /bin/sh -c #(nop) COPY file:0fd5fca330dcd6a7…   1.04kB
<missing>      2 days ago    /bin/sh -c #(nop) COPY file:08ae525f517706a5…   1.95kB
<missing>      2 days ago    /bin/sh -c #(nop) COPY file:e7e183879c35719c…   1.2kB
<missing>      2 days ago    /bin/sh -c set -x     && addgroup --system -…   63.6MB
<missing>      2 days ago    /bin/sh -c #(nop)  ENV PKG_RELEASE=1~buster     0B
<missing>      2 days ago    /bin/sh -c #(nop)  ENV NJS_VERSION=0.4.4        0B
<missing>      2 days ago    /bin/sh -c #(nop)  ENV NGINX_VERSION=1.19.5     0B
<missing>      9 days ago    /bin/sh -c #(nop)  LABEL maintainer=NGINX Do…   0B
<missing>      9 days ago    /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>      9 days ago    /bin/sh -c #(nop) ADD file:d2abb0e4e7ac17737…   69.2MB
```

# docker image rm

**docker image rm** เป็นคำสั่งที่ใช้ลบ image ที่เก็บอยู่ในเครื่อง

```
]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              f643c72bc252        35 hours ago        72.9MB
nginx               latest              bc9a0695f571        2 days ago          133MB
centos              latest              0d120b6ccaa8        3 months ago        215MB
]# docker image rm ubuntu nginx
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:c95a8e48bf88e9849f3e0f723d9f49fa12c5a00cfc6e60d2bc99d87555295e4c
Deleted: sha256:f643c72bc25212974c16f3348b3a898b1ec1eb13ec1539e10a103e6e217eb2f1
Deleted: sha256:9386795d450ce06c6819c8bc5eff8daa71d47ccb9f9fb8d49fe1ccfb5fb3edbe
Deleted: sha256:3779241fda7b1caf03964626c3503e930f2f19a5ffaba6f4b4ad21fd38df3b6b
Deleted: sha256:bacd3af13903e13a43fe87b6944acd1ff21024132aad6e74b4452d984fb1a99a
Untagged: nginx:latest
Untagged: nginx@sha256:6b1daa9462046581ac15be20277a7c75476283f969cb3a61c8725ec38d3b01c3
Deleted: sha256:bc9a0695f5712dcaaa09a5adc415a3936ccba13fc2587dfd76b1b8aeea3f221c
Deleted: sha256:a6862ade3b91fdde2aa8a3d77fdcc95b1eb6c606be079c11b7f97f249d0e731d
Deleted: sha256:32bcbe3740b68d0625744e774b404140366c0c4a2b2eadf32280d66ba001b4fb
Deleted: sha256:2dc5e43f496e41a18c016904b6665454a53be22eb4dcc1b468d864b4e2d1f311
Deleted: sha256:5fe6a7c579cd9fbcfa604810974c4c0c16893f4c40bc801545607ebd0accea74
Deleted: sha256:f5600c6330da7bb112776ba067a32a9c20842d6ecc8ee3289f1a713b644092f8
]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              latest              0d120b6ccaa8        3 months ago        215MB
```

Action in DOCKER

info, login, logout, manifest, search
system, version


container start
container stop
container kill
container restart
container pause
container unpause
container exec

image push
image pull

container attach
container cp
container inspect
container prune
container rename
container rm
container update

container export
container create
container run
container commit
image import

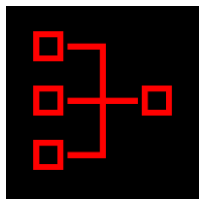
container

image

image load
image save
image tag
image rm
image history

image ls
image prune
image inspect

container ls
container port
container stats
container logs
Container top



network

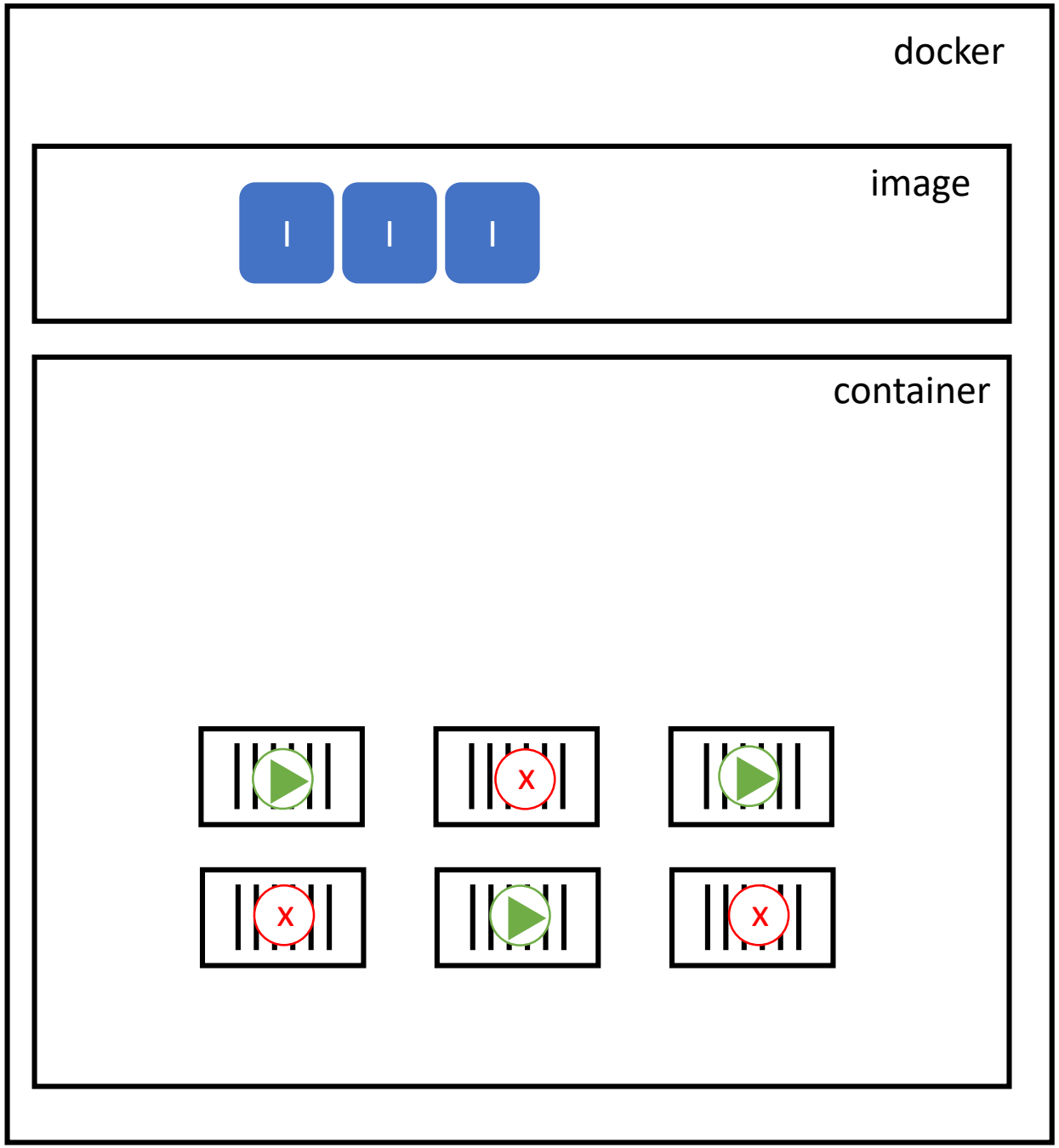
volume

image build

Dockerfile

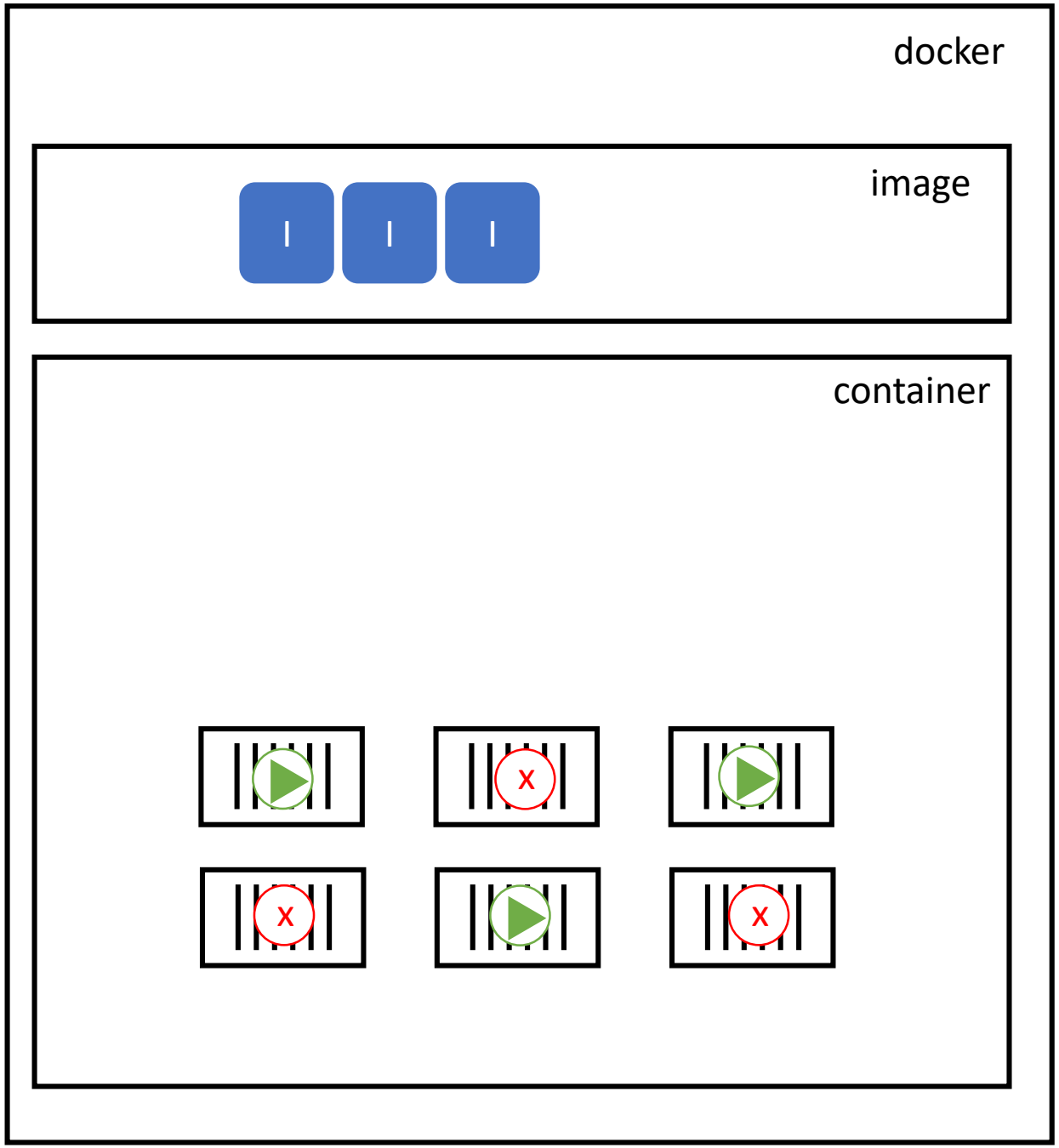
# docker container ls

```
]# docker container ls
```



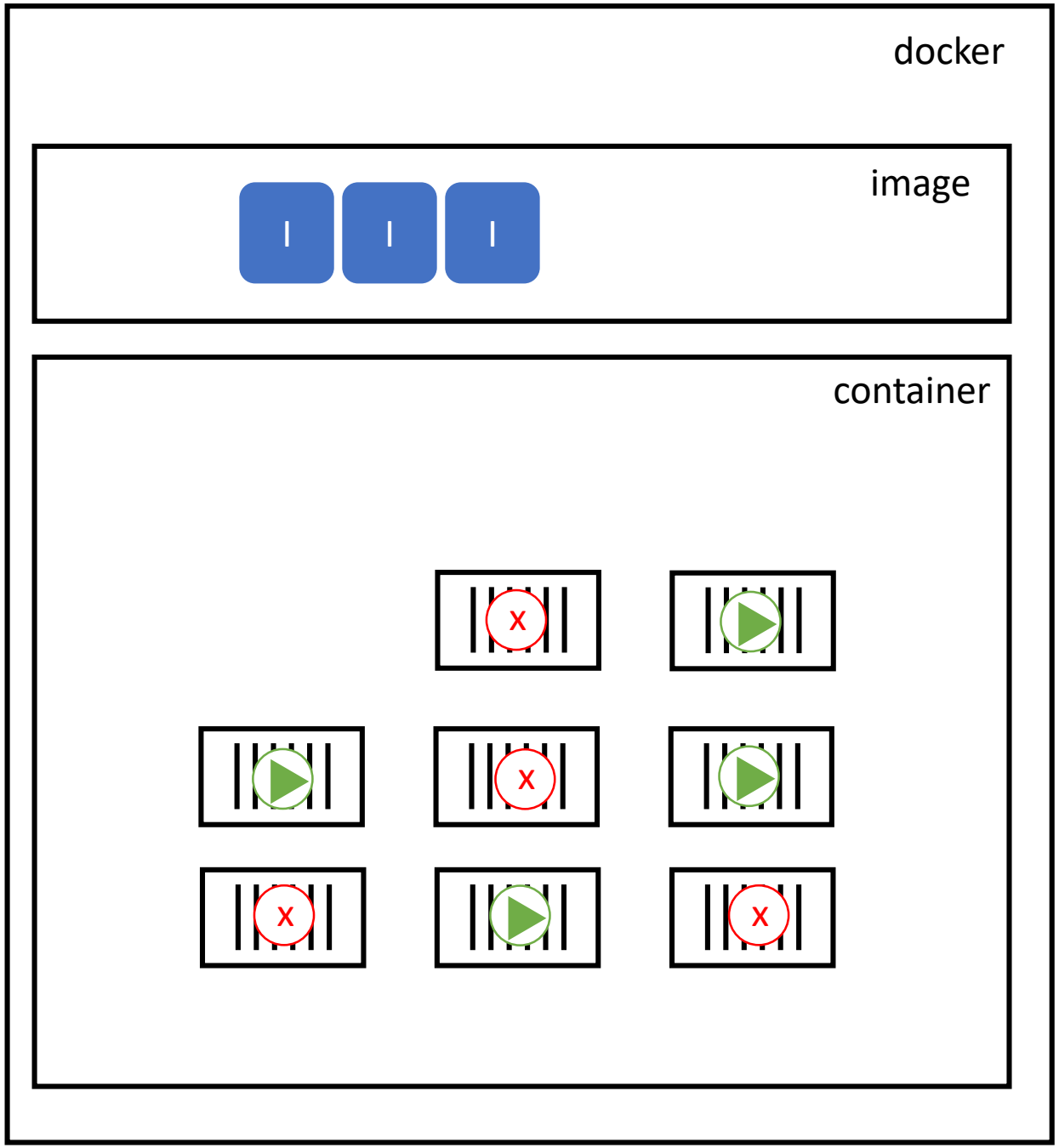
# docker container ls

```
]# docker container ls -a
```



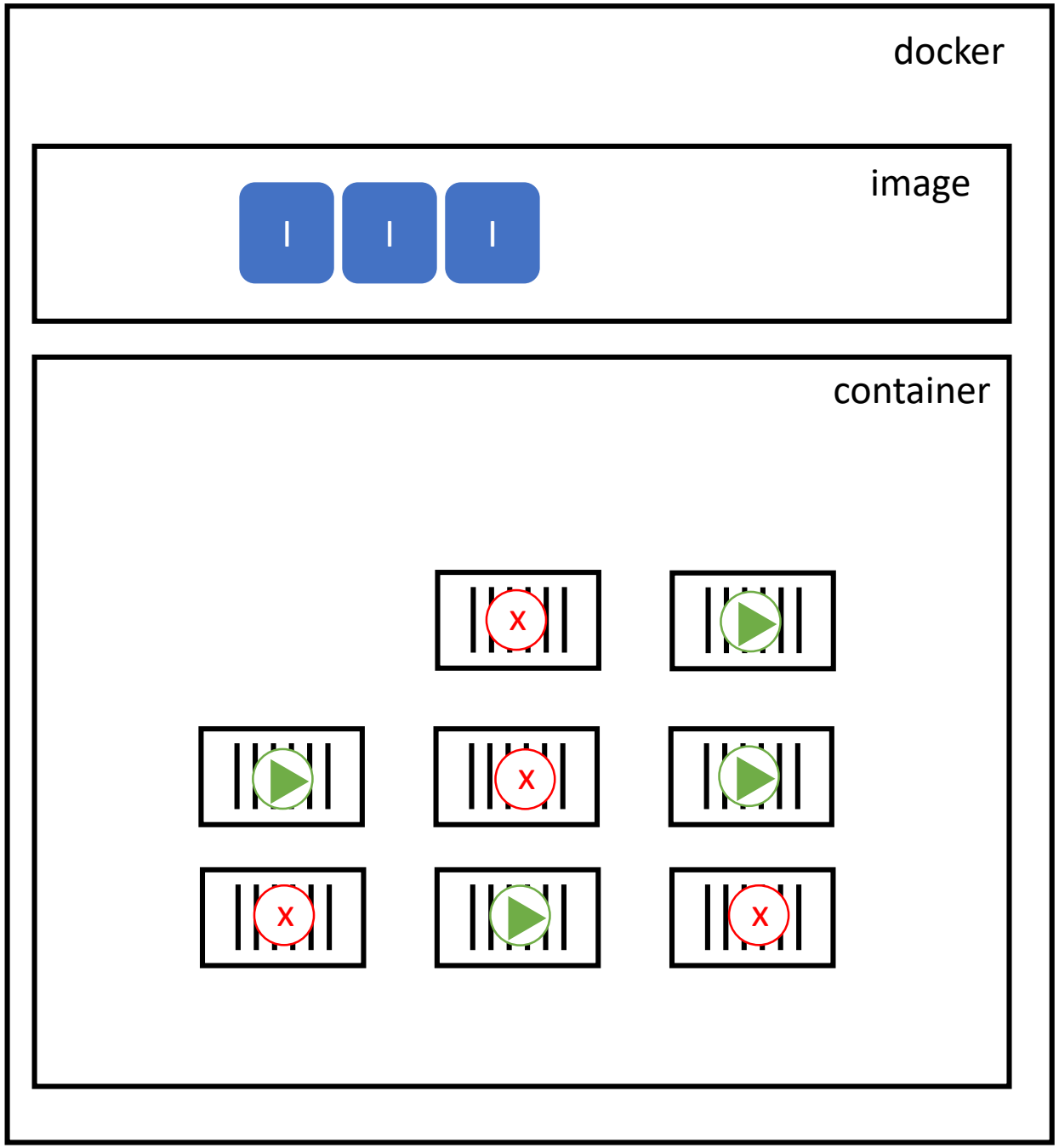
# docker container ls

```
]# docker container ls -l
```



# docker container ls

```
]# docker container ls -n 2
```



# docker container ls

**docker container ls** เป็นคำสั่งที่ใช้แสดง container ที่ถูกสร้างขึ้น

หากไม่มีการระบุ option คำสั่งจะแสดงเฉพาะ container ที่กำลังทำงานอยู่เท่านั้น

```
]# docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
227aaa315a81        nginx               "/docker-entrypoint.…"   6 minutes ago       Up 6 minutes
80/tcp              awesome_shtern
9e9aafd0b78f        nginx               "/docker-entrypoint.…"   6 minutes ago       Up 6 minutes
80/tcp              charming_lederberg
```

option -a คำสั่งจะแสดง container ทุกตัวที่มีอยู่ในเครื่อง

```
]# docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
ce2863e6168f        nginx               "/docker-entrypoint.…"   10 minutes ago      Exited (0) 9
minutes ago                         practical_kepler
227aaa315a81        nginx               "/docker-entrypoint.…"   10 minutes ago      Up 10 minutes
80/tcp              awesome_shtern
9e9aafd0b78f        nginx               "/docker-entrypoint.…"   10 minutes ago      Up 10 minutes
80/tcp              charming_lederberg
```

# docker container ls

option -l คำสั่งจะแสดง container ตัวล่าสุดที่ถูกสั่งสร้างขึ้นมา

```
]# docker container ls -l
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
ce2863e6168f        nginx               "/docker-entrypoint.…"   14 minutes ago      Exited (0) 13
minutes ago                         practical_kepler
```

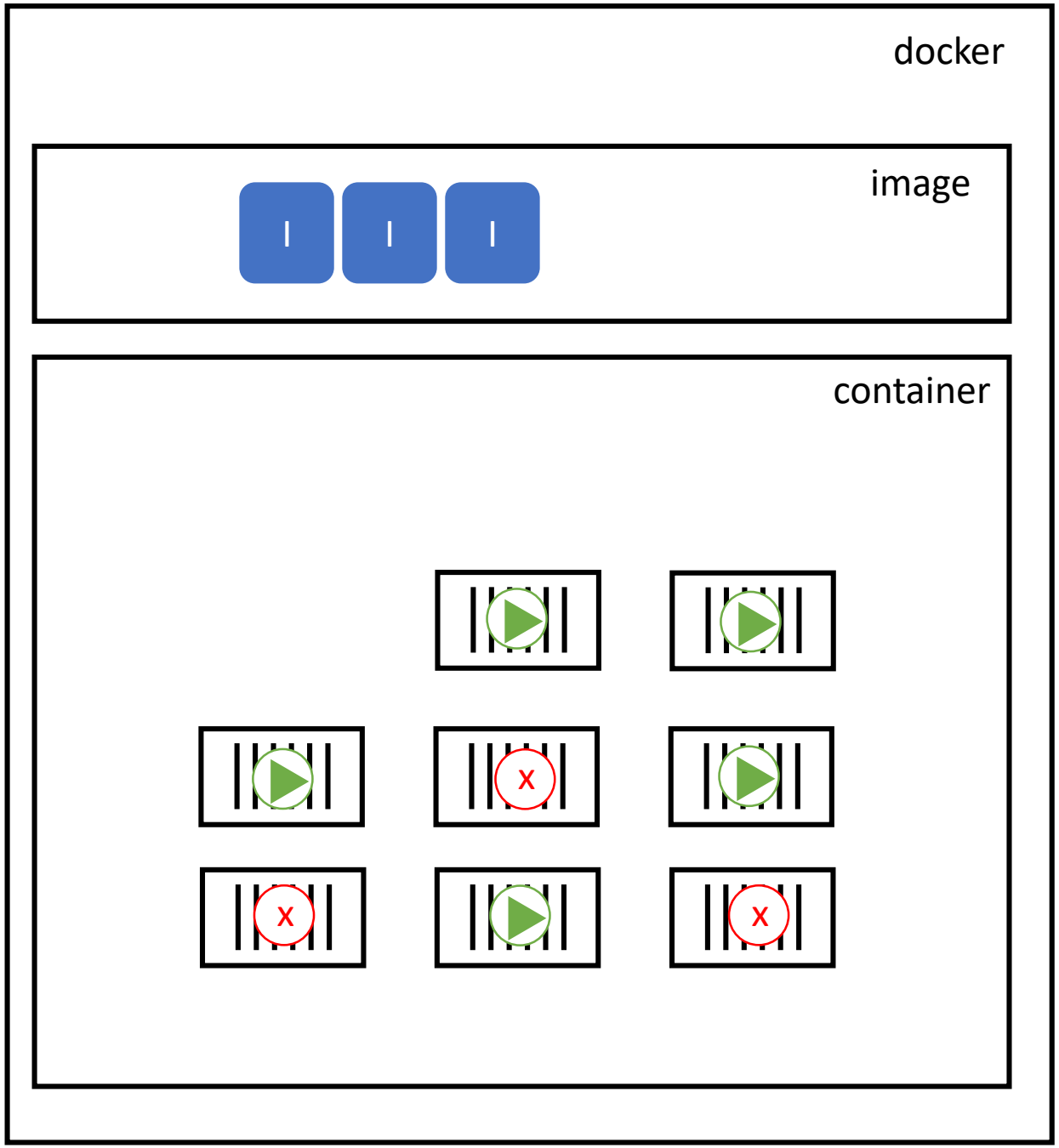
option -n N คำสั่งจะแสดง container N ตัวล่าสุดที่ถูกสั่งสร้างขึ้นมา

```
]# docker container ls -n 2
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
ce2863e6168f        nginx               "/docker-entrypoint.…"   16 minutes ago      Exited (0) 16
minutes ago                         practical_kepler
227aaa315a81        nginx               "/docker-entrypoint.…"   16 minutes ago      Up 16 minutes
80/tcp              awesome_shtern
```

# docker container run

```
]# docker container run -d nginx
```

คำสั่ง docker container run เป็นคำสั่งในการสร้าง container และ กำหนดให้ container เริ่มทำงานในคำสั่งเดียวกัน



# docker container run

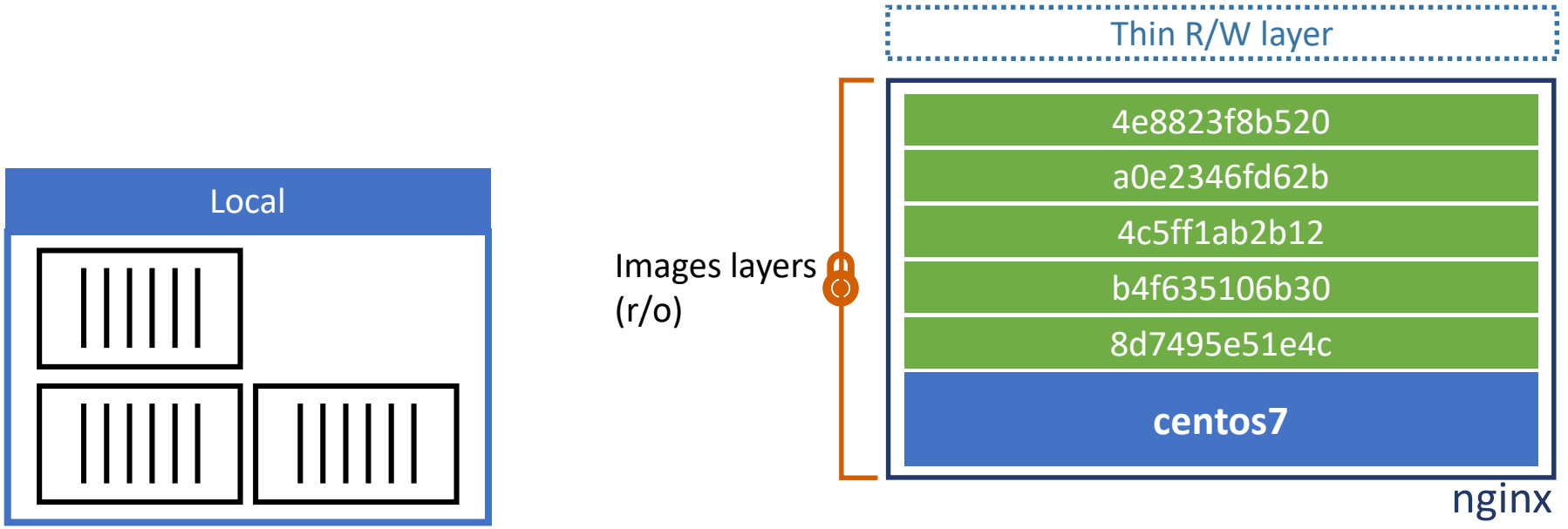
ส่งคำสั่งที่ต้องการไปทำงานใน container

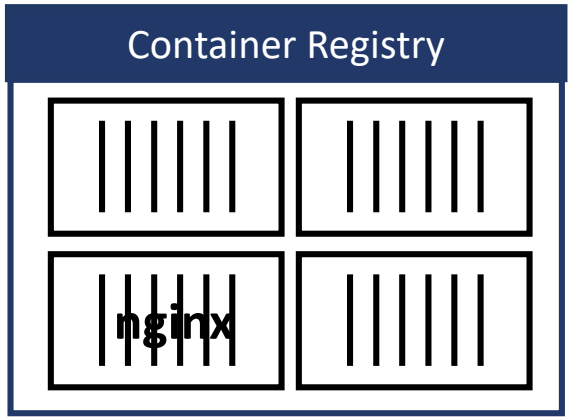
```
]# docker container run centos expr 5 + 2
7
]# docker container run centos expr 5 + 4
9
]# cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.1 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.1 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
]# docker container run centos cat /etc/redhat-release
CentOS Linux release 8.2.2004 (Core)
```

HOST

- ENTRYPOINT
- parameter in the command “docker run”
- CMD

```
]$ docker container run -d nginx
```



# docker container run

**sleep** - delay for a specified amount of time

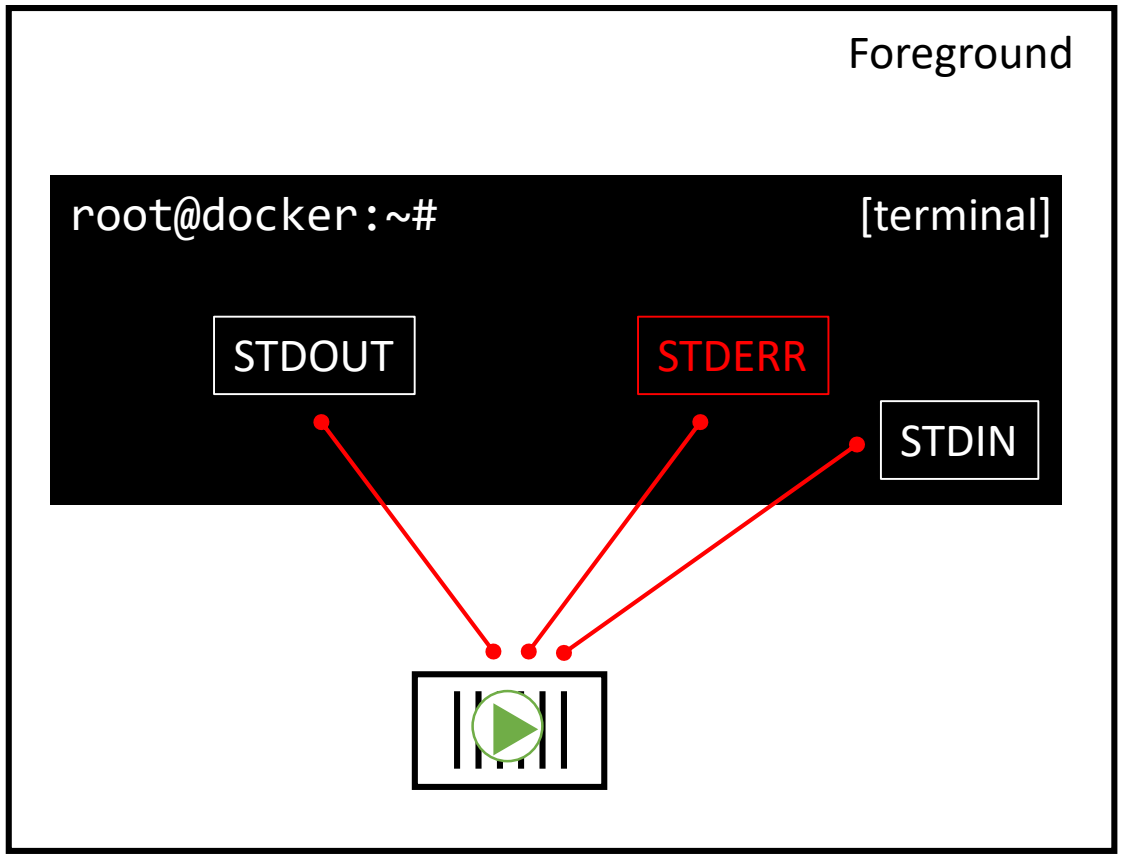
```
]# date; docker container run centos /bin/sh -c "echo Hello;sleep 20"; date
Sat 28 Nov 2020 12:02:04 AM +07
Hello
Sat 28 Nov 2020 12:02:25 AM +07
```

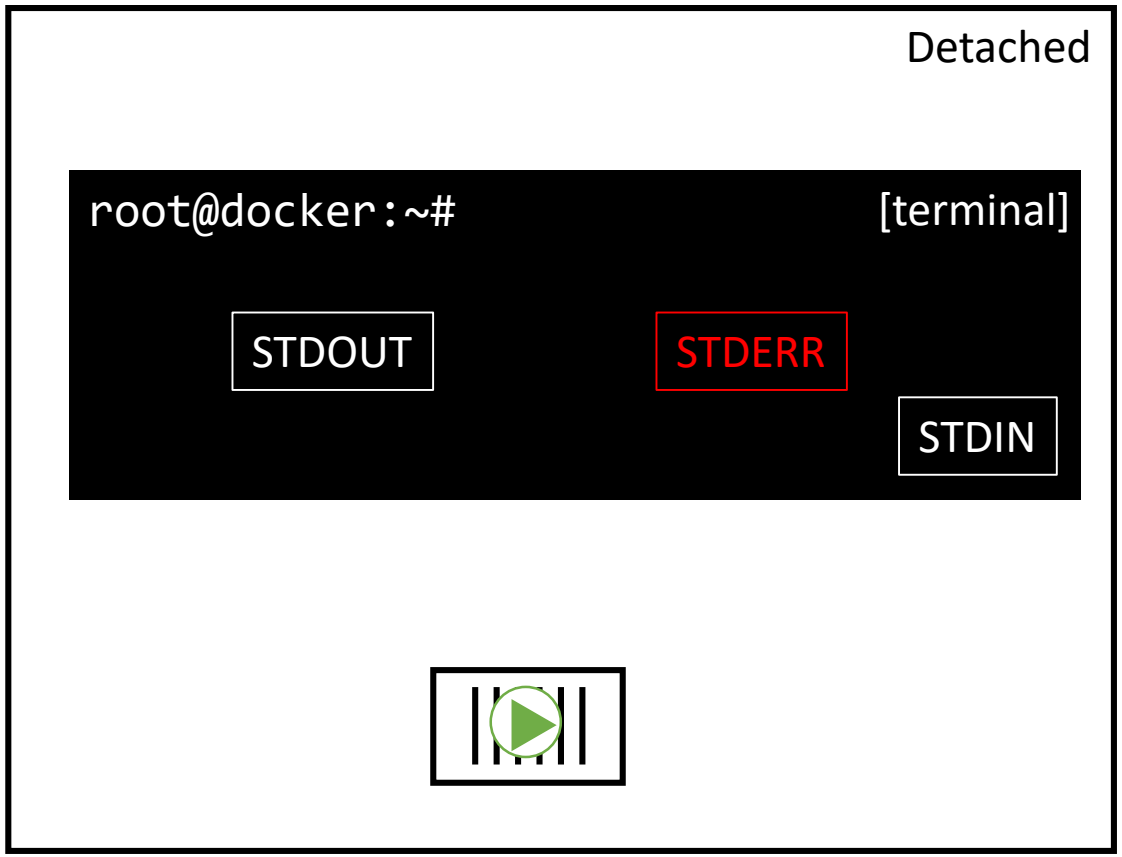
```
]# date; docker container run centos /bin/sh -c "while true; do  sleep 1; done"; date
```

จะเกิดอะไรขึ้นถ้าเรียกคำสั่งแบบนี้

# docker container run

## Detached vs Foreground



# docker container run

การสร้าง container ใน detached mode สามารถทำได้โดยการระบุ option -d=true หรือ -d และ container จะออกจาก detached mode เมื่อคำสั่งที่เรียกใช้งานใน container (root process) ทำงานเสร็จสิ้น หรือ หยุดการทำงาน

```
]# docker container run -d centos /bin/sh -c "while true; do  sleep 1; done"
55717e0d810ce9e4ef35fda81a5c10cefe29eddb1cb1d61f3a799b1521d6cf1c
]# docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
55717e0d810c        centos              "/bin/sh -c 'while t…"   7 seconds ago       Up 6 seconds
youthful_solomon
]# docker container run -d centos sleep 5
1e1734bcc8fc77dc72a5141e7c1c4c276d52f902e937bf138f98d34e8569d7cb
]# docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
1e1734bcc8fc        centos              "sleep 5"                2 seconds ago       Up 1 second
dreamy_mendeleev
55717e0d810c        centos              "/bin/sh -c 'while t…"   51 seconds ago      Up 50 seconds
youthful_solomon

]# docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED              STATUS
55717e0d810c        centos              "/bin/sh -c 'while t…"   About a minute ago   Up About a
minute              youthful_solomon
```

# docker container run

ในกรณีที่ต้องการทำงานกับ container ที่สร้างและเรียกใช้งาน โดยตรงด้วยตัวเอง ผ่าน terminal สามารถเลือกทำงานในรูปแบบนี้ด้วยการเลือก foreground mode และจำเป็นต้องเรียกใช้งานพร้อมกับ option ดังนี้

```
-a=[] : Attach to `STDIN`, `STDOUT` and/or `STDERR`
-t : Allocate a pseudo-tty
-i : Keep STDIN open even if not attached
```

* foreground mode เป็น Default mode ในการสร้าง container จึงไม่จำเป็นต้องมี option เพิ่มเติมในการกำหนด mode

```
]# docker container run -it centos /bin/bash
[root@aae6dff630ba /]# cat /etc/redhat-release
CentOS Linux release 8.2.2004 (Core)
[root@aae6dff630ba /]#
```

```
]# echo "cat /etc/redhat-release" | docker container run -i centos /bin/bash
CentOS Linux release 8.2.2004 (Core)
root@docker:~#
```

# docker container run

--rm เป็น option ที่สั่งเพิ่มเติม เพื่อให้ container ที่หยุดทำงาน ถูกลบโดยทันที ซึ่งโดยปกติ หากไม่ระบุ option --rm เมื่อ container ทำงานเสร็จเรียบร้อย container ก็จะยังอยู่ แต่มีสถานะเป็น Exited และสามารถถูกเรียกใช้งานอีกครั้งก็ได้

```
]# docker container run --rm centos expr 5 + 7
12
]# docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
]# docker run -d --rm nginx
0a948adb2052ca9c6efa4191096d1ee446065b7ba3e50af4f9cf3a1f5bc1005e
]# docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
0a948adb2052        nginx               "/docker-entrypoint.…"   6 seconds ago       Up 4 seconds
80/tcp              infallible_davinci
]# docker stop 0a948adb2052
0a948adb2052
]# docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
```

# docker container run

การระบุถึง container แต่ละตัวสามารถระบได้ 3 วิธีคือ UUID แบบยาว, UUID แบบสั้น และ ชื่อของ container เอง

- UUID จะถูกกำหนดโดย Docker Daemon
- UUID แบบสั้นจะเป็นตัวอักษร 12 ตัวแรกของ UUID แบบยาว
- หากไม่ระบุชื่อ Docker Daemon จะสุ่มคำออกมา 2 คำแล้วเชื่อด้วย _ (underscore)
- สามารถกำหนดชื่อของ container ได้ด้วย option --name

```
]# docker run -d --rm nginx
223d4307ab7550c899049ed55c2207d9ac343c71f039e20b4799a905904dcacd
]# docker run -d --rm --name mycontainer nginx
3250ef2f8a803c90b50ca68564a83dbef0aa74fece19e0c6a29b699627bb072c
]# docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
3250ef2f8a80        nginx               "/docker-entrypoint.…"   6 seconds ago       Up 5
seconds        80/tcp              mycontainer
223d4307ab75        nginx               "/docker-entrypoint.…"   18 seconds ago      Up 16
seconds        80/tcp              recursing_nightingale
```

**UUID stand for Universally Unique Identifiers**

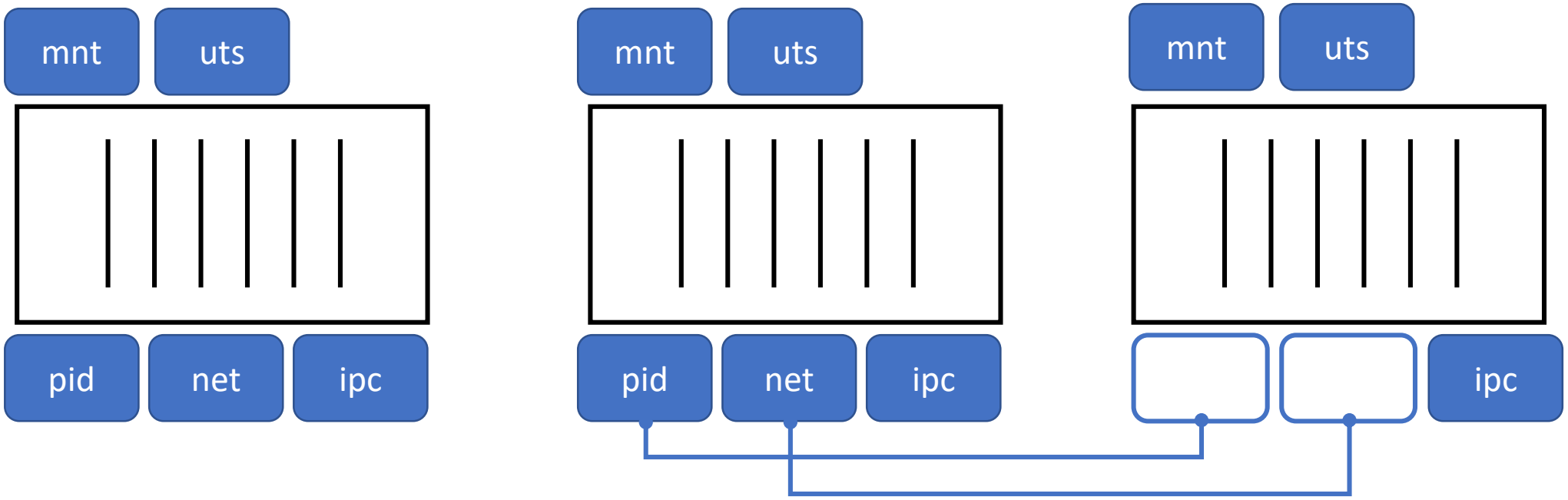
# docker container run

option --cidfile สามารถกำหนดให้เขียนค่า container ID แบบยาวไปที่ไฟล์ที่กำหนดได้

```
]# docker container run -d --cidfile mycid nginx
4581bea824b21ae4b1447166796bbaee6508d9b82de77f69ab19f127d3f618e0
]# cat mycid
4581bea824b21ae4b1447166796bbaee6508d9b82de77f69ab19f127d3f618e0
]# docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
4581bea824b2        nginx               "/docker-entrypoint.…"   22 seconds ago      Up 21
seconds       80/tcp              hungry_austin
```

# docker container run

Namespaces



# docker container run

โดยปกติการสร้างและเรียกใช้งาน container จะมีการสร้าง pid namespace สำหรับ container ขึ้นมาให้ใหม่เฉพาะตัว แต่ในบางกรณี หากต้องการให้ container ที่สร้างขึ้นไปใช้งานร่วมกับ container ตัวอื่น ๆ หรือ ใช้งานร่วมกัน host สามารถใช้ option --pid เพื่อระบุได้

```
--pid="" : Set the PID (Process) Namespace mode for the container,
           'container:<name|id>': joins another container's PID namespace
           'host': use the host's PID namespace inside the container
```

```
]# sleep 12345 &
[1] 18477

]# docker container run centos /bin/bash -c "ps aux | grep 12345"
root           1  0.0  0.1  11884  2796 ?        Ss   18:39   0:00 /bin/bash -c ps aux | grep 12345
root           7  0.0  0.0  11884   236 ?        R    18:39   0:00 /bin/bash -c ps aux | grep 12345

]# docker container run --pid=host centos /bin/bash -c "ps aux | grep 12345"
root       18477  0.0  0.0   5476   528 ?        S    18:39   0:00 sleep 12345
root       18580  0.0  3.0 767016 59124 ?        Sl+  18:39   0:00 docker container run --pid=host
centos /bin/bash -c ps aux | grep 12345
root       18617  0.0  0.1  11884  2856 ?        Ss   18:39   0:00 /bin/bash -c ps aux | grep 12345
root       18657  0.0  0.0   9176  1076 ?        S    18:39   0:00 grep 12345
```

# docker container run

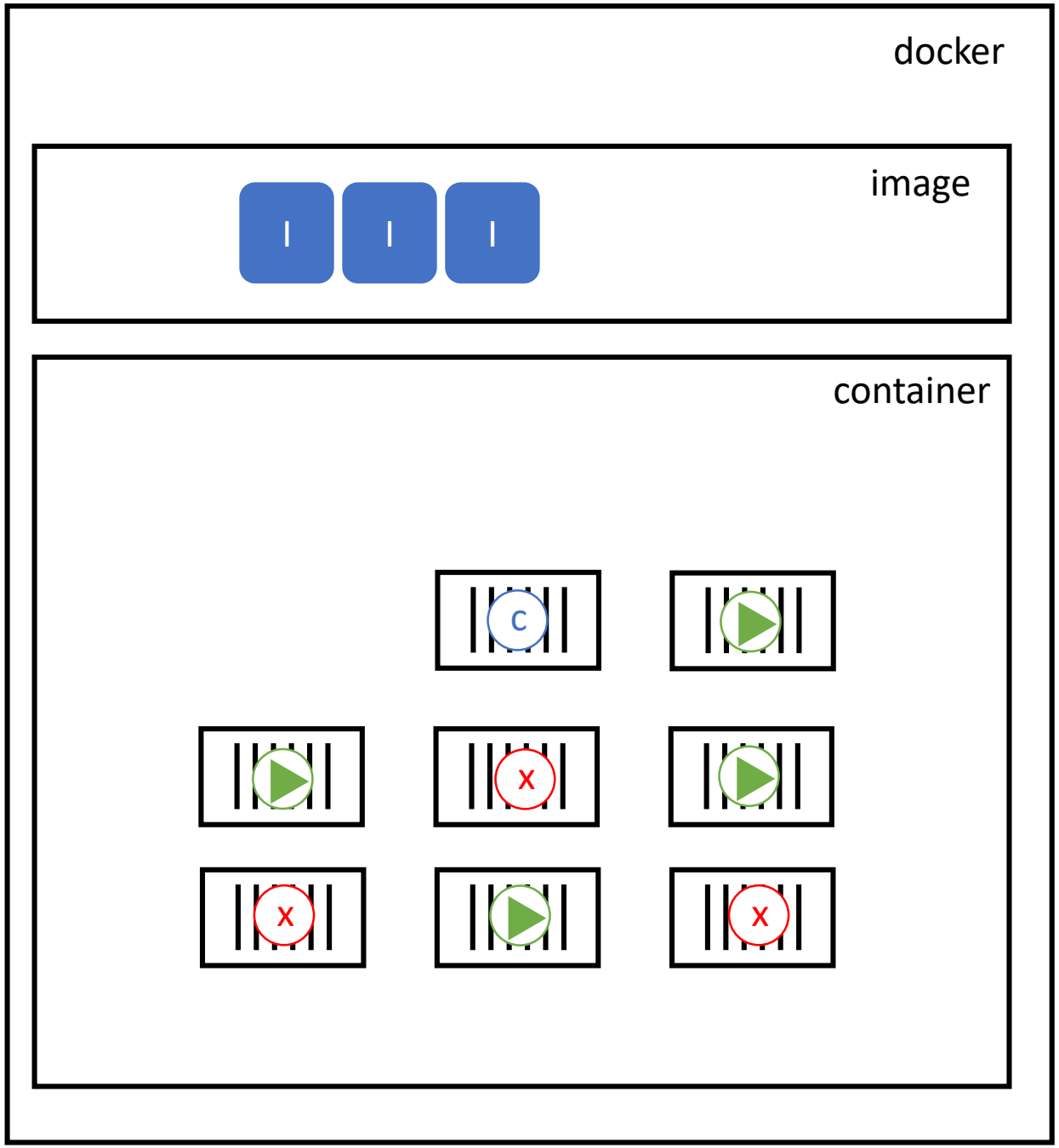
การกำหนดนโยบายในการ restart container โดยปกติหาก contianer หยุดทำงาน, docker daemon หยุดทำงาน แล้วเรียกขึ้นมาใช้งานอีกครั้ง หรือแม้กระทั้ง reboot เครื่อง สถานะของ container ก็จะหยุดทำงาน หากต้องการให้ container ทำงานอย่างต่อเนื่องสามารถใช้ --restart=<restart policy> เพื่อกำหนดการ start ของ container ในกรณีที่เกิดความผิดปกติได้

| Policy | Result |
|---|---|
| **no** | Do not automatically restart the container when it exits. This is the **default**. |
| **on-failure**[:max-retries] | Restart only **if the container exits with a non-zero exit status**. Optionally, limit the number of restart retries the Docker daemon attempts. |
| **always** | Always restart the container regardless of the exit status. When you specify always, the Docker daemon will try to restart the container indefinitely. The container will also always start on daemon startup, regardless of the current state of the container. |
| **unless-stopped** | Always restart the container regardless of the exit status, including on daemon startup, **except if the container was put into a stopped state before the Docker daemon was stopped**. |

# docker container create

```
]# docker container create nginx
```

docker container create เป็นคำสั่งที่ใช้สร้าง container จาก image ที่ระบุไว้ และสถานะของ container จะถูกระบุว่าเป็น “Created”



# docker container create

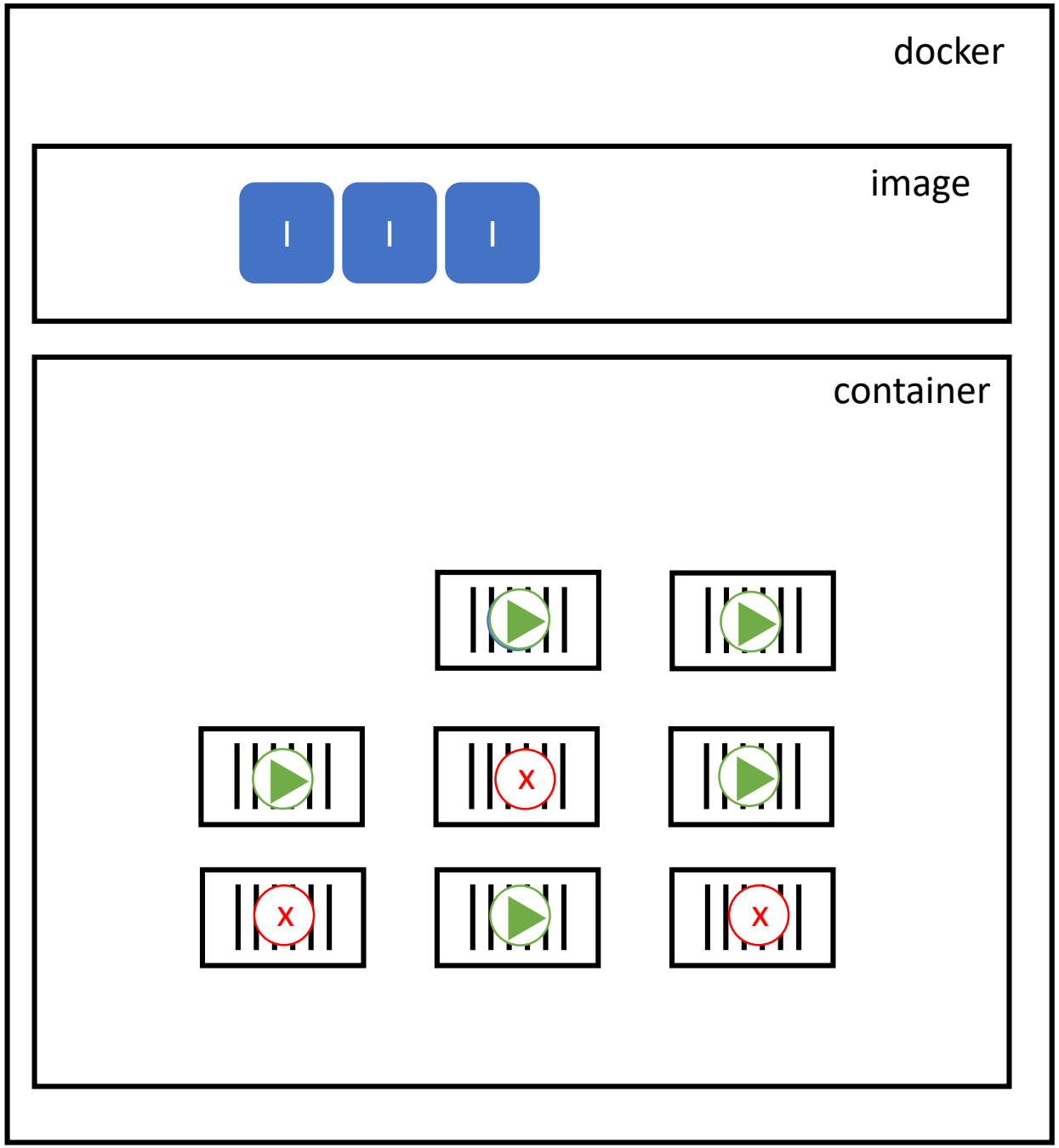
```
]# docker container create --name mynginx nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
852e50cd189d: Pull complete
571d7e852307: Pull complete
addb10abd9cb: Pull complete
d20aa7ccdb77: Pull complete
8b03f1e11359: Pull complete
Digest: sha256:6b1daa9462046581ac15be20277a7c75476283f969cb3a61c8725ec38d3b01c3
Status: Downloaded newer image for nginx:latest
9d2329de0b8e23d9d914954f24d5c954b1c3dcc91c5771bd70aa3d475185b6a7
```

```
]# docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
9d2329de0b8e        nginx               "/docker-entrypoint.…"   21 seconds ago      Created
mynginx
```

# docker container start

```
]# docker container start mynginx
```

เป็นคำสั่งที่สั่งให้ container ที่ถูกสร้างขึ้นแล้ว เริ่มต้นทำงาน
สถานะของ container จะถูกเปลี่ยนเป็น Up ถ้ายังทำงานอยู่



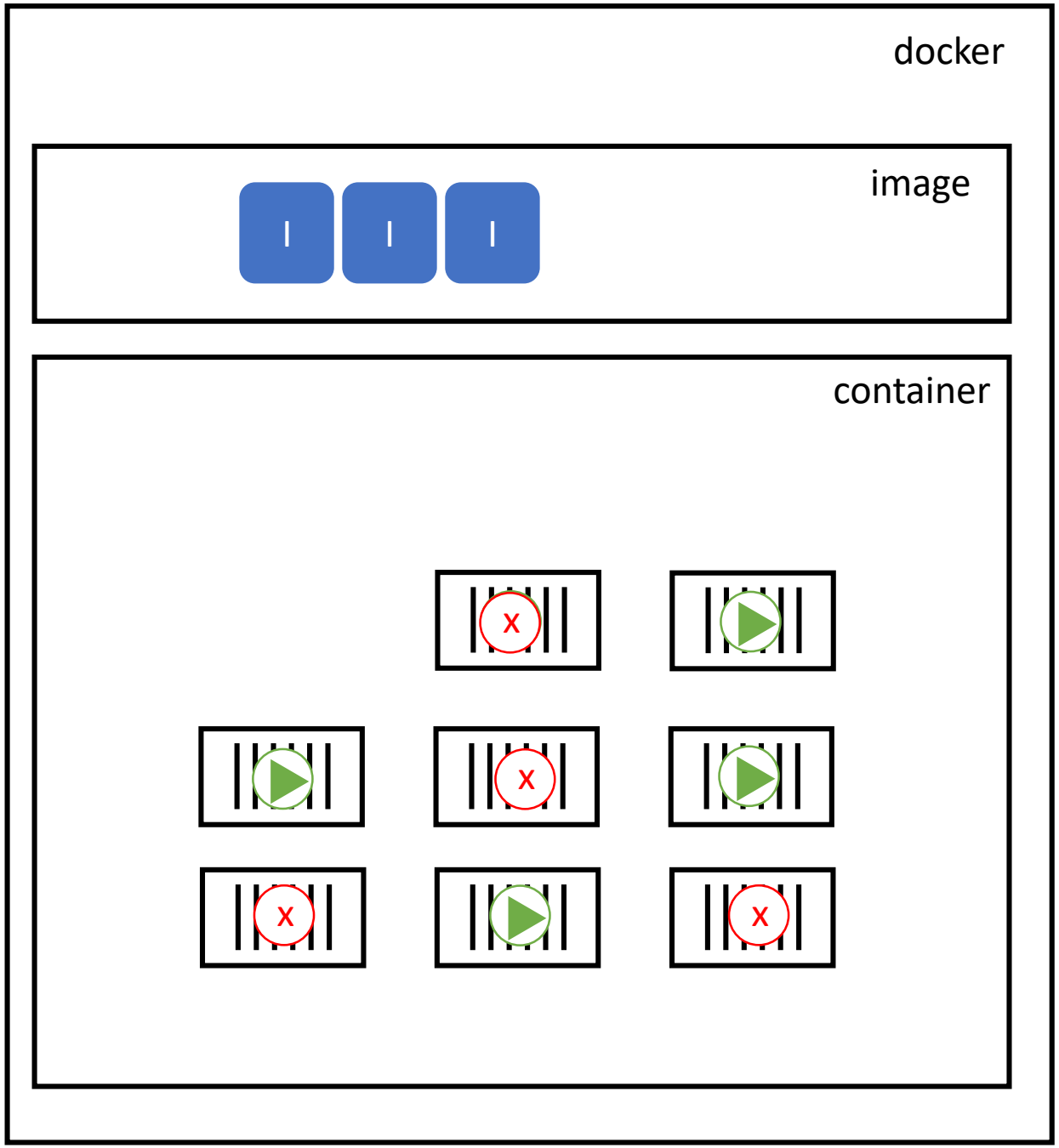
# docker container start

```
]# docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
9d2329de0b8e        nginx               "/docker-entrypoint.…"   21 seconds ago      Created
mynginx

]# docker container start mynginx
mynginx
]# docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
9d2329de0b8e        nginx               "/docker-entrypoint.…"   10 minutes ago      Up 4 seconds
80/tcp              mynginx
```

# docker container stop

```
]# docker container stop mynginx
```

เป็นคำสั่งที่สั่งให้ container หยุดทำงาน สถานะของ container ก็จะถูกเปลี่ยนเป็น Exited



# docker container stop

```
]# docker container start mynginx
mynginx
]# docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
9d2329de0b8e        nginx               "/docker-entrypoint.…"   10 minutes ago      Up 4 seconds
80/tcp              mynginx
```

กำหนดให้รอการหยุดทำงานของ process (SIGTERM) เป็นจำนวน N วินาที หากเกินเวลาที่กำหนด process จะถูกส่งสัญญา kill (SIGKILL) ให้หยุดทำงาน

```
]# docker container stop -t 30 mynginx
mynginx
]# docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
9d2329de0b8e        nginx               "/docker-entrypoint.…"   18 minutes ago      Exited (0) 6
seconds ago                       mynginx
```

# docker container restart

**docker container restart == docker container stop + docker container start**

```
]# docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
9d2329de0b8e        nginx               "/docker-entrypoint.…"   34 minutes ago      Up 2 minutes
80/tcp              mynginx
]# docker container restart mynginx
mynginx
]# docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
9d2329de0b8e        nginx               "/docker-entrypoint.…"   35 minutes ago      Up 6 seconds
80/tcp              mynginx
```

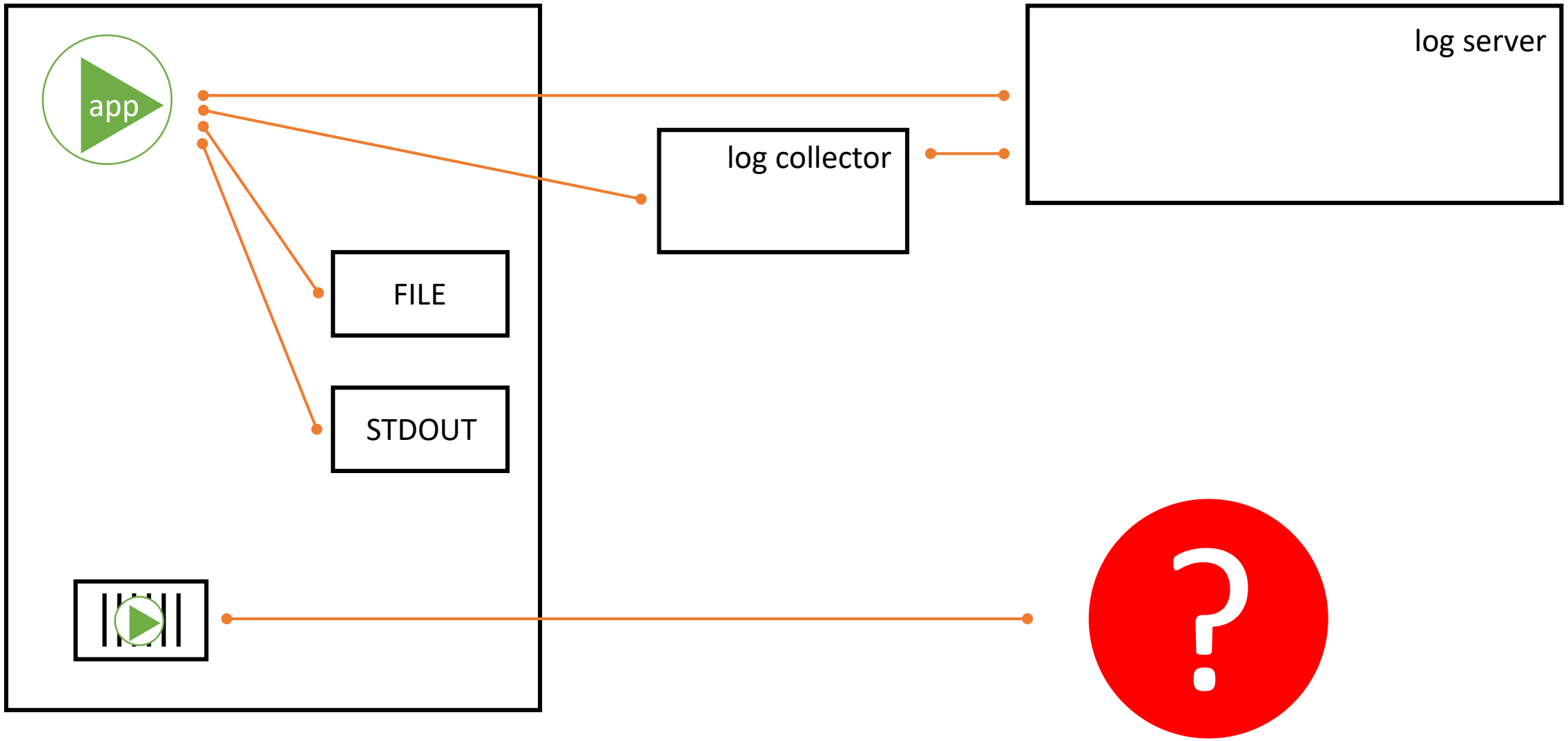
# docker container exec

docker container exec เป็นคำสั่งที่ใช้เรียกคำสั่งภายใน container ที่กำลังทำงานอยู่ มักนิยมใช้เรียก shell เพื่อเข้าไปตรวจสอบการทำงานของ container ซึ่งต้องระบุ option -it

```
]# docker container exec -d mynginx-centos sleep 2200
]# docker container exec -it mynginx-centos /bin/bash
bash-4.2$ ps  fax
    PID TTY      STAT   TIME COMMAND
     49 pts/0    Ss     0:00 /bin/bash
     64 pts/0    R+     0:00  \_ ps fax
     44 ?        Ss     0:00 sleep 2200
      1 ?        Ss     0:00 nginx: master process nginx -g daemon off;
     26 ?        S      0:00 nginx: worker process
     27 ?        S      0:00 nginx: worker process
bash-4.2$
```

# docker container logs

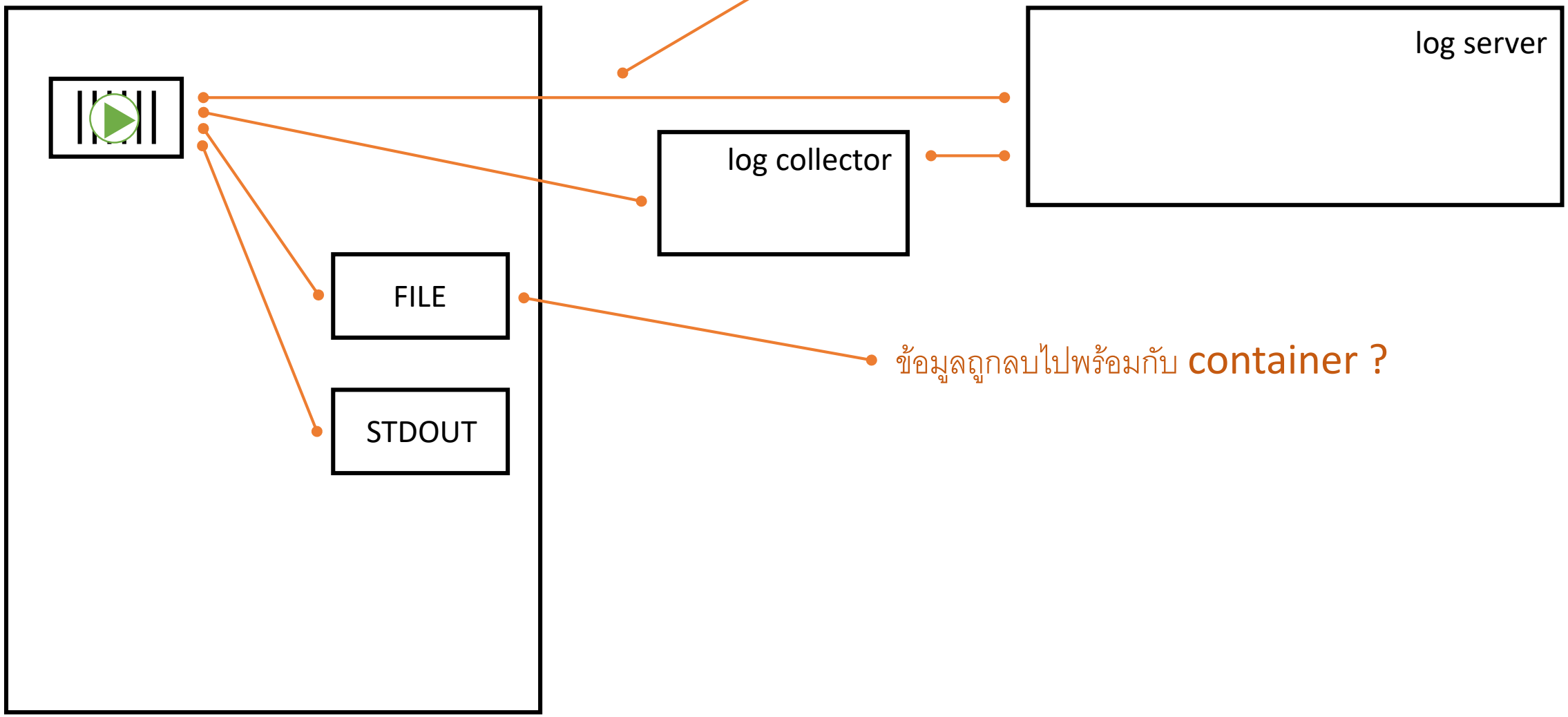
แนวทางในการจัด log ที่เกิดจากการทำงานของ application



# docker container logs

แนวทางในการจัด log ที่เกิดจากการทำงานของ application

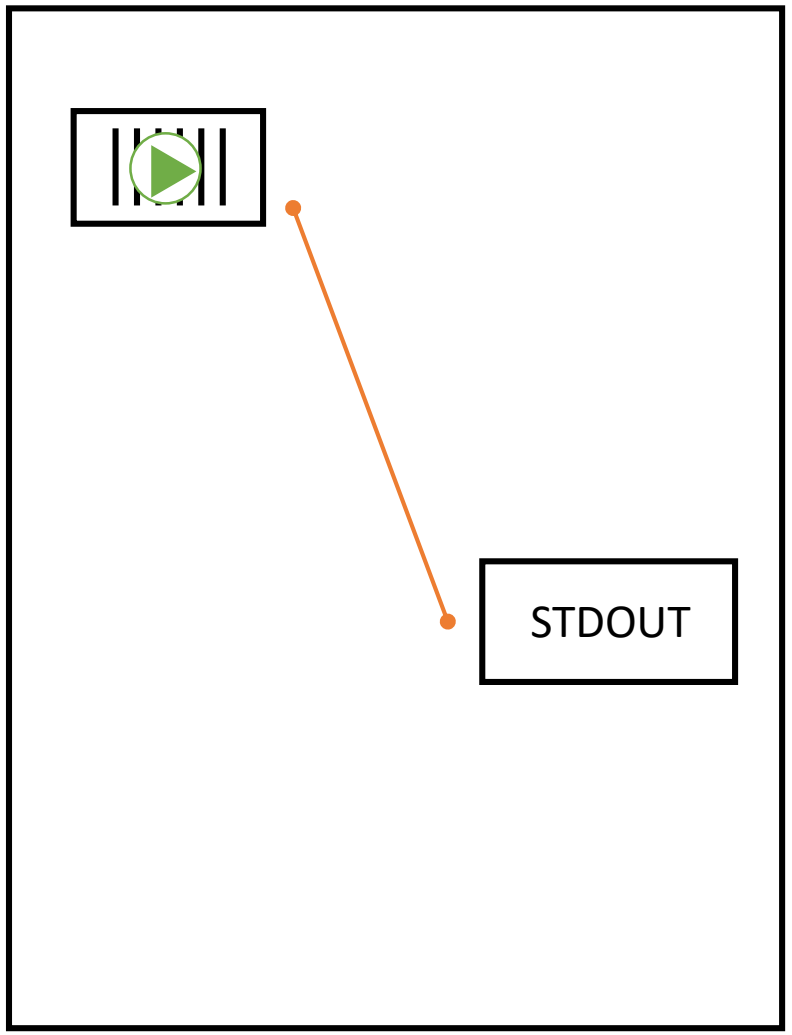
จำเป็นต้องติดตั้ง log shipping agent ใน container ?




ข้อมูลถูกลบไปพร้อมกับ container ?

# docker container logs

แนวทางในการจัด log ที่เกิดจากการทำงานของ application




```
]# docker logs mynginx
```

คำสั่ง docker logs ใช้ในการเรียกดูข้อความโปรแกรมใน container ส่งออกมาทาง STDOUT และ STDERR

# docker container logs

หากไม่ระบุ option ใด ๆ จะแสดงข้อความทั้งหมดที่ถูกส่งออกมาที่ STDOUT ของ container

```
]# docker container logs mynginx-centos
172.17.0.1 - - [28/Nov/2020:06:54:26 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:54:30 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:54:30 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:54:30 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:54:31 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:54:31 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0”
]#
```

option -f กำหนดให้แสดงข้อความอย่างต่อเนื่อง หากมีข้อความใหม่ที่ส่งออกมาที่ STDOUT

```
]# docker container logs -f mynginx-centos
172.17.0.1 - - [28/Nov/2020:06:54:26 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:54:30 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:54:30 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:54:30 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:54:31 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:54:31 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0”
```

# docker container logs

option --tail N กำหนดให้แสดงข้อความ N บรรทัดสุดท้าย

```
]# docker container logs --tail 5 mynginx-centos
172.17.0.1 - - [28/Nov/2020:06:56:34 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:56:34 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:56:34 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:56:35 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:06:56:35 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
```

option --since N กำหนดให้แสดงข้อความช่วงเวลาที่กำหนด เช่น 5 นาทีสุดท้าย

```
]# docker container logs --since 5m  mynginx-centos
172.17.0.1 - - [28/Nov/2020:07:03:21 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:07:03:21 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:07:03:21 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:07:03:21 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:07:03:22 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:07:08:10 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
172.17.0.1 - - [28/Nov/2020:07:08:11 +0000] "GET / HTTP/1.1" 200 105 "-" "curl/7.68.0"
```

# docker container stats

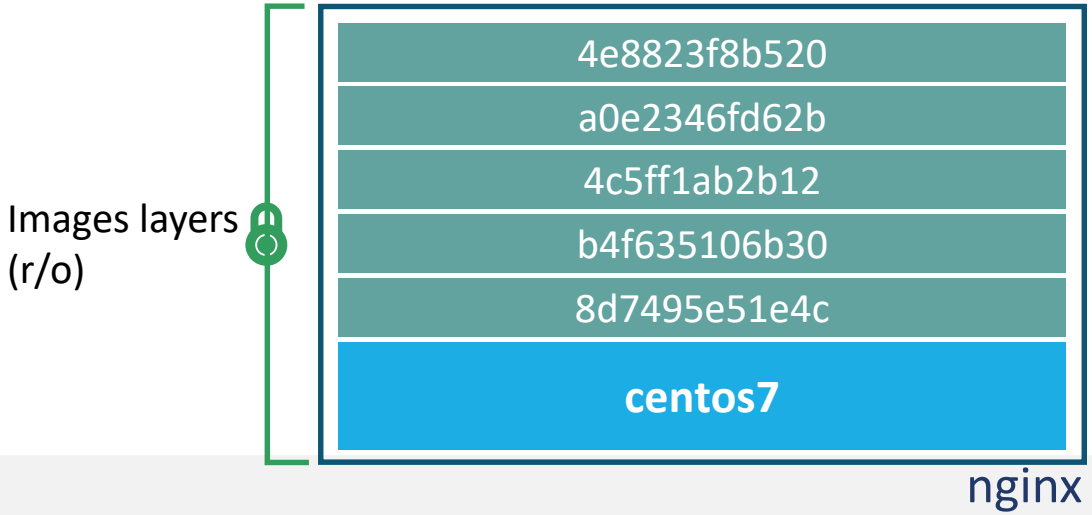
docker container stats เป็นคำสั่งที่ใช้แสดงสถิติการใช้ทรัพยากรของเครื่อง เช่น %CPU, Memory Usage, %Memory, Network Usage, Block I/O ของ container ที่กำลังทำงานอยู่

```
]# docker container stats
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT     MEM %
NET I/O             BLOCK I/O           PIDS
50b61cb6b7be        charming_curie      0.00%               1.773MiB / 1.872GiB   0.09%
1.54kB / 0B         0B / 0B             2
dabd9bfc1640        elastic_pike        0.00%               1.711MiB / 1.872GiB   0.09%
1.68kB / 0B         0B / 0B             2
6f50c8018a54        crazy_faraday       0.00%               3.156MiB / 1.872GiB   0.16%
1.68kB / 0B         0B / 4.1kB          3
1f50fa065b5a        nostalgic_swanson   0.00%               2.992MiB / 1.872GiB   0.16%
1.68kB / 0B         1.04MB / 8.19kB     2
7b553958a5c2        mynginx-centos      0.00%               5.633MiB / 1.872GiB   0.29%
14kB / 15.4kB       4.1kB / 0B          3
```

# Docker Images

# Images and Layers

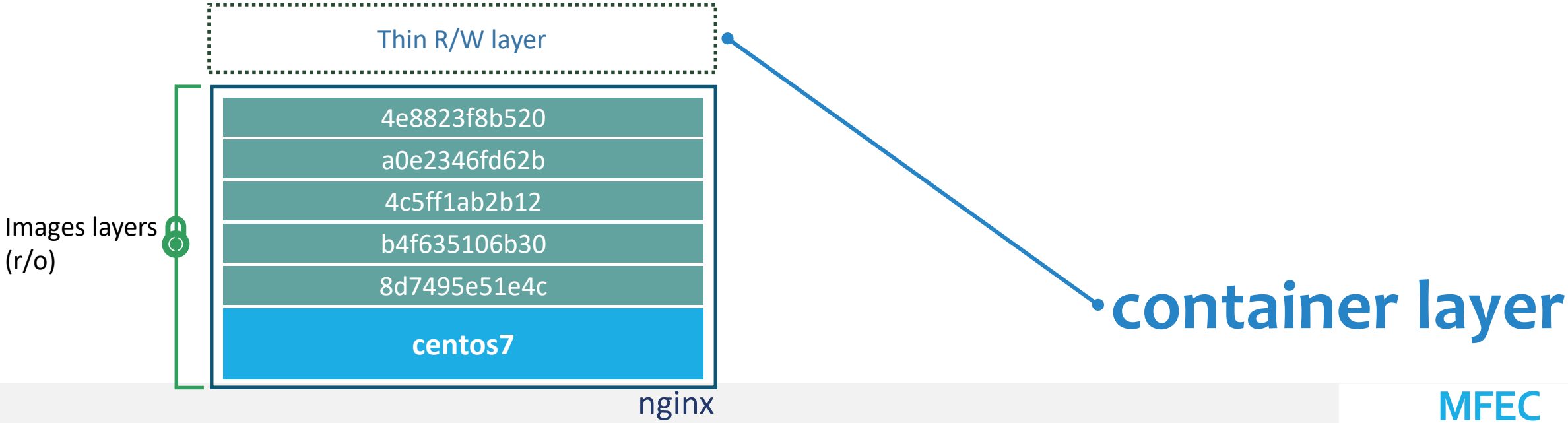
- Docker image มีลักษณะเป็นชั้น ( layer) ที่เรียงกันตามลำดับที่มีความสัมพันธ์ซึ่งกันและกัน
- แต่ละชั้นใน image จะเก็บความแตกต่างจากชั้นที่ถูกจัดเก็บมาก่อนหน้าชั้นของตัวเอง
- แต่ละชั้นจะถูกสร้างจาก ชุดคำสั่งในแต่ละบรรทัดที่ถูกระบุไว้ใน Dockerfile
- ข้อมูลที่ถูกเก็บแต่ละชั้นใน image จะถูกำหนดให้อ่านได้อย่างเดียว (Read Only)




```
FROM centos7
RUN yum install –y nginx
COPY index.html /var/www/html/
EXPOSE 80
VOLUME /var/www/html
CMD [“nginx”, “-g”, “daemon”]
```

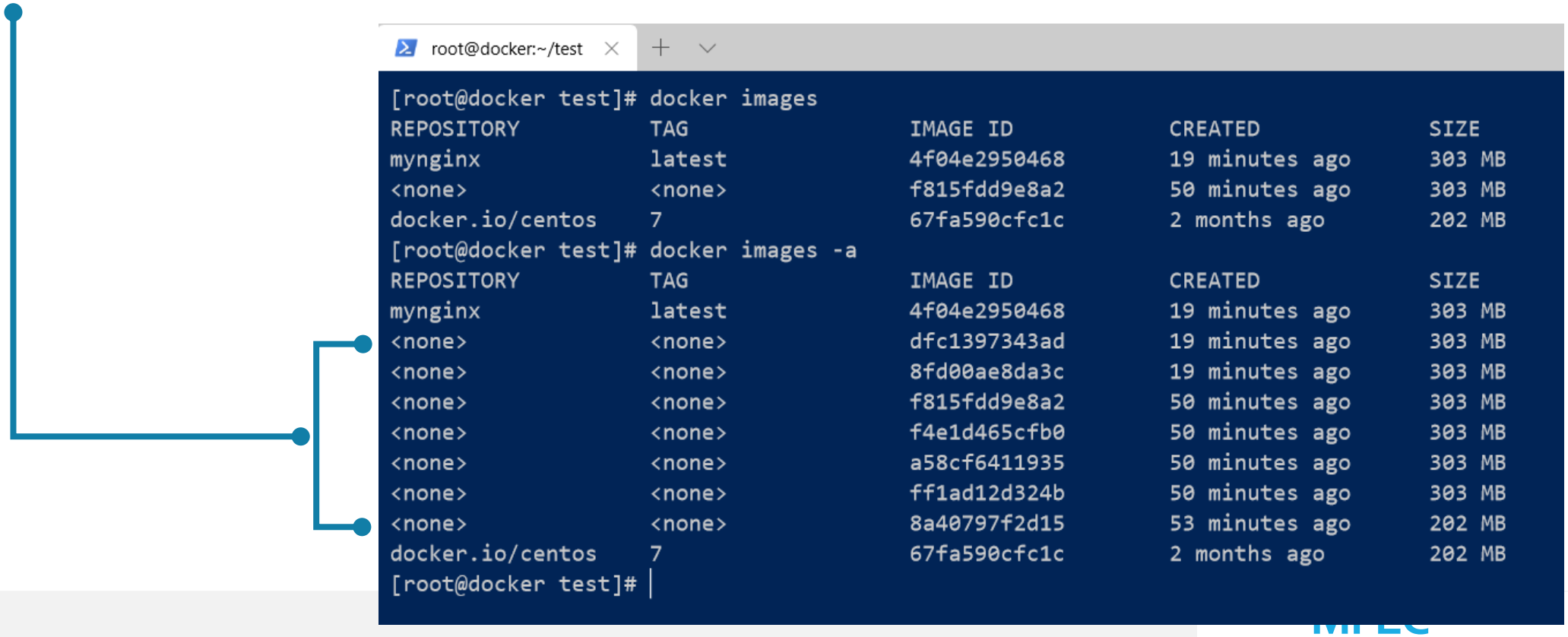
# Images and Layers

- เมื่อ container ที่ถูกสร้างจาก image ถูกสั่งให้สร้างขึ้น  container engine ก็จะสร้างชั้นใหม่เพิ่มขึ้นอีกชั้นหนึ่ง ที่สามารถเขียนข้อมลในชั้นข้อมูลชั้นนี้ได้
- ข้อมูลที่เกิดขึ้นในการทำงานของ container เช่นการสร้างไฟล์ขึ้นมาใหม่, การเปลี่ยนแปลงข้อมูลเดิมในไฟล์ที่มีอยู่ หรือการลบไฟล์จะถูกในบันทึกความเปลี่ยนแปลงในขั้นนี้



# Images and Layers > What is <none>:<none>?

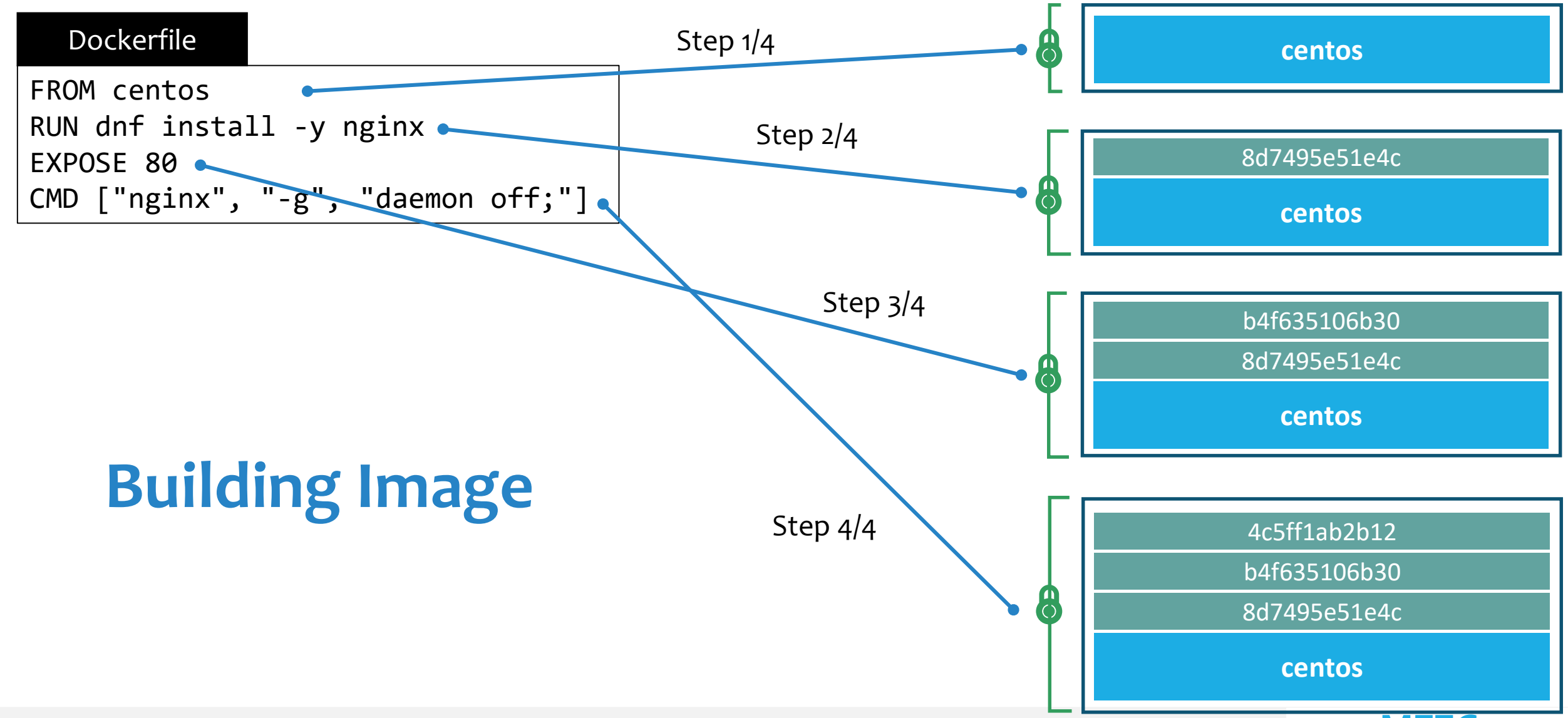
- Intermediate layers
- Dangling Images



# Images and Layers > Intermediate Layers

Dockerfile

```
FROM centos
RUN dnf install -y nginx
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```




## Building Image

# Images and Layers > Intermediate Layers

ในการสร้าง image จาก Dockerfile ชุดคำสั่งในแต่ละบรรทัดจะสร้างชั้นของความเปลี่ยนแปลงขึ้นมา และสร้างเป็น image ใหม่ และ image ที่ถูกสร้างขึ้นในแต่ละชั้นจะยังถูกเก็บไว้ เพื่อถูกใช้ประโยชน์เป็น cache ในการสร้าง image ใหม่ทำให้เพิ่มความเร็วในการสร้าง image ซ้ำ ซึ่งแต่ละชั้นเรียกว่า Intermediate Layer

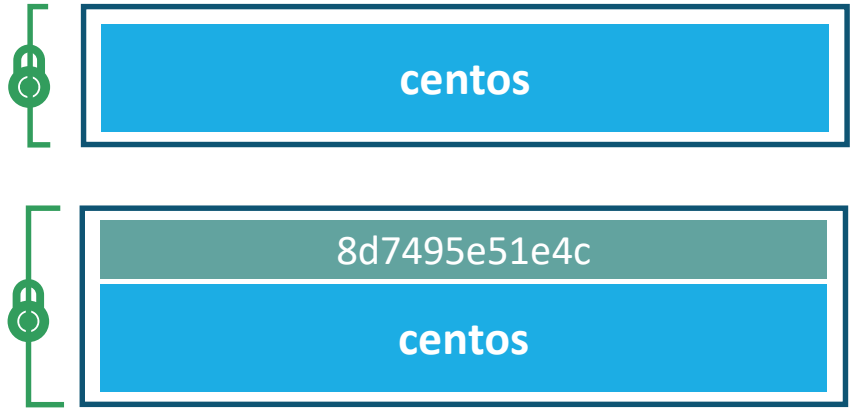
```
]# docker image ls -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              20404a198250        5 minutes ago       289MB
mynginx             latest              d347f53f2a7c        5 minutes ago       289MB
<none>              <none>              8f92fd776161        5 minutes ago       289MB
<none>              <none>              d2cf031034ce        11 minutes ago      289MB
<none>              <none>              5b07d713bd14        11 minutes ago      289MB
<none>              <none>              3e124ff4f165        16 minutes ago      289MB
centos              latest              0d120b6ccaa8        2 weeks ago         215MB
```

```
]# docker image history mynginx
IMAGE               CREATED             CREATED BY                                      SIZE
d347f53f2a7c        36 seconds ago      /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
20404a198250        36 seconds ago      /bin/sh -c #(nop)  EXPOSE 80                    0B
8f92fd776161        38 seconds ago      /bin/sh -c dnf install -y nginx                 74.4MB
0d120b6ccaa8        2 weeks ago         /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  LABEL org.label-schema.sc…   0B
<missing>           2 weeks ago         /bin/sh -c #(nop) ADD file:538afc0c5c964ce0d…   215MB
```

# Images and Layers > Dangling Images

Dockerfile

```
FROM centos
RUN dnf install -y nginx
EXPOSE 80  EXPOSE 8080
CMD ["nginx", "-g", "daemon off;"]
```

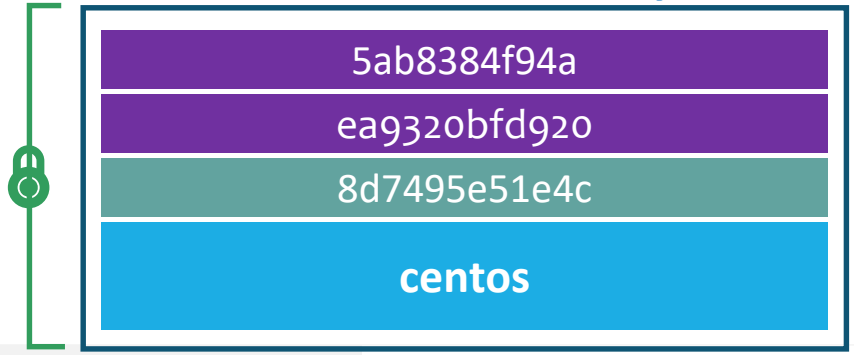



New Image

ยังไม่ถูกกำหนดเป็น dangling image เพราะยังถูกอ้างถึงใน image ในลำดับถัดไปอยู่



Dangling Image

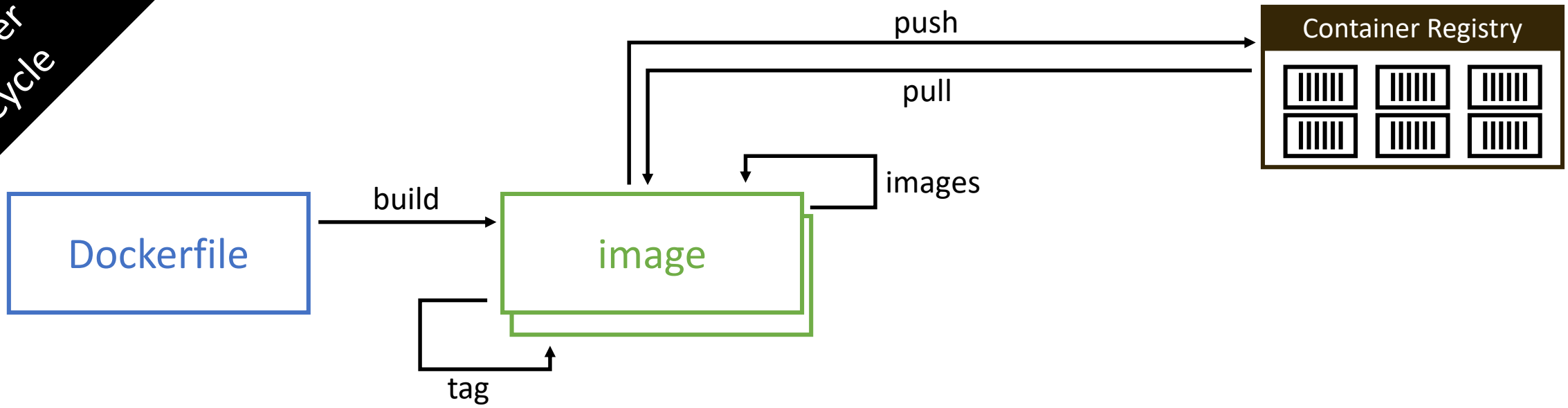

MFEC
Public Company Limited

# Images and Layers > Dangling Images

ถ้ามีการสร้าง image ใหม่จาก Dockerfile เดิมที่เคยมีการสร้าง image เรียบร้อยแล้ว แต่มีการเปลี่ยนแปลงบางชุดคำสั่งใน Dockerfile ดังนั้น จะเกิด image ที่ไม่ได้ถูกอ้างถึงและไม่ถูกเรียกใช้งาน ในการสร้าง image ในครั้งถัดไป ทำให้มีการใช้งานพื้นที่ดิสก์โดยไม่จำเป็นเกิดขึ้น

```
]# docker image ls -f dangling=true
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              d347f53f2a7c        About an hour ago   289MB
<none>              <none>              5b07d713bd14        2 hours ago         289MB
]# docker image rm 5b07d713bd14
Deleted: sha256:5b07d713bd147d019d6007132c18d7018e209625c0bf4cbf96e83d6a11763fda
Deleted: sha256:d2cf031034ce5ba802d4d15c9909ddfd38af9a0cff0d42b9798158799b07c824
Deleted: sha256:3e124ff4f1651cdbc159e423edd8e8b983b9491f398b78108a6fd6f81749fbb3
Deleted: sha256:2a81b54215eeb734302eaad493a963108a578319df1ca7f3158960227f841191
[root@docker lab]# docker image ls -f dangling=true
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              d347f53f2a7c        2 hours ago         289MB
```

Container Life Cycle




FROM centos:7

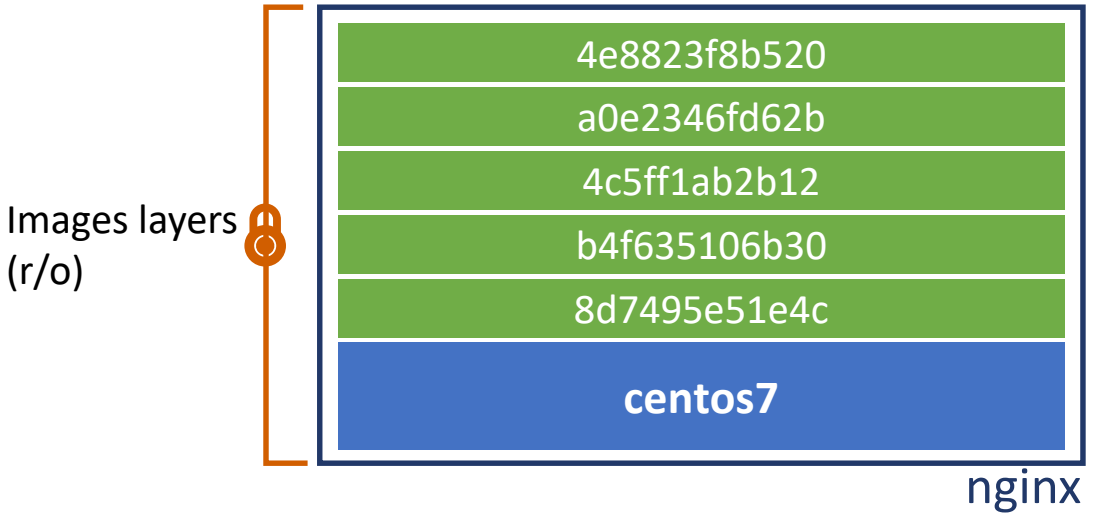
RUN yum install –y nginx

COPY index.html /var/www/html/

EXPOSE 80

VOLUME /var/www/html

CMD [“nginx”, “-g”, “daemon”]



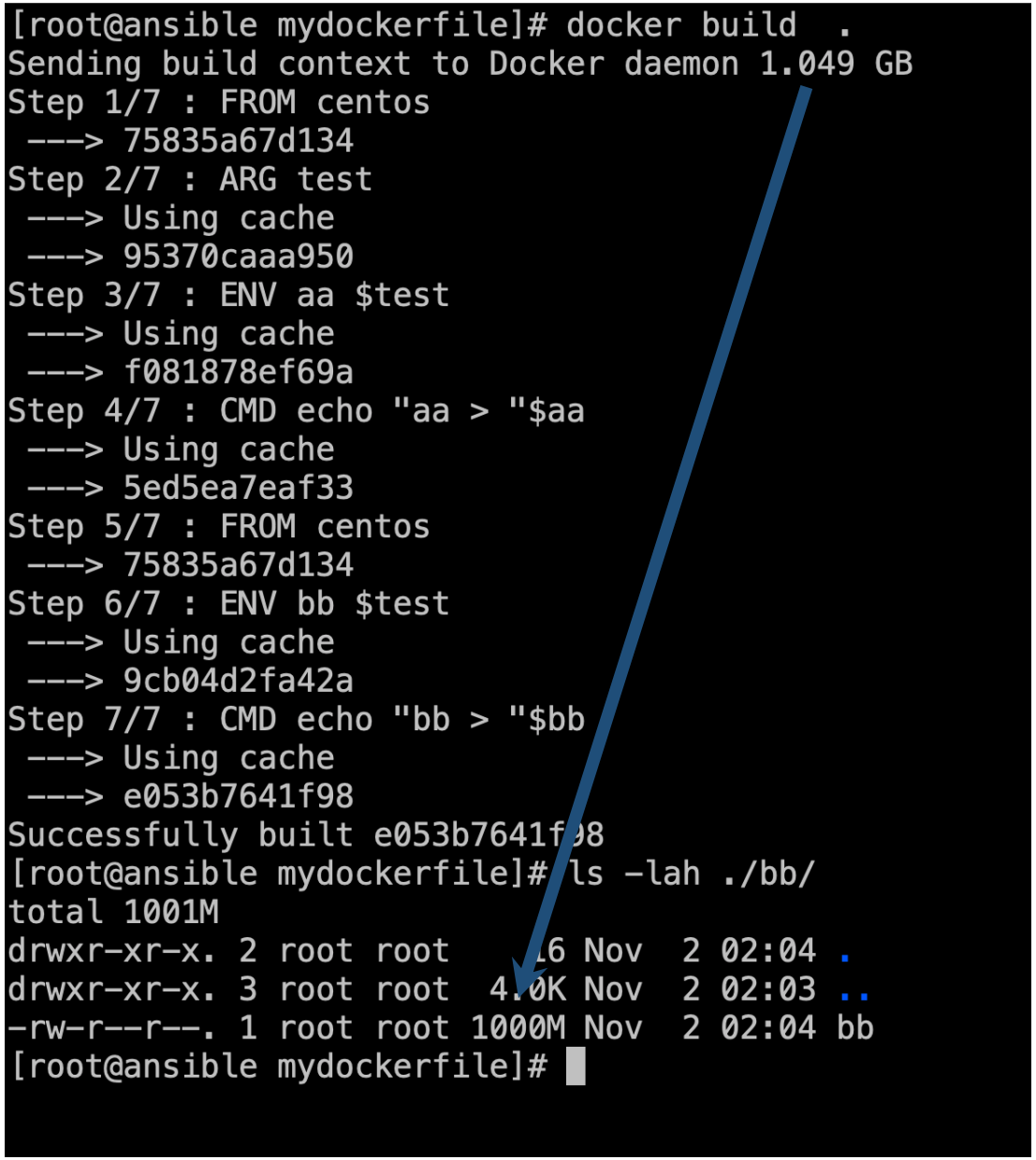
# docker BUILD

Docker can build images automatically by reading the instructions from a **Dockerfile**. A Dockerfile is a **text document** that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an **automated build** that executes several command-line instructions in succession.

The docker build command builds an image from a Dockerfile and a *context*. The **build's context is the set of files at a specified location PATH or URL**. The PATH is a directory on your local filesystem. The URL is a Git repository location.

**A context is processed recursively**. So, a PATH includes any subdirectories and the URL includes the repository and its submodules.

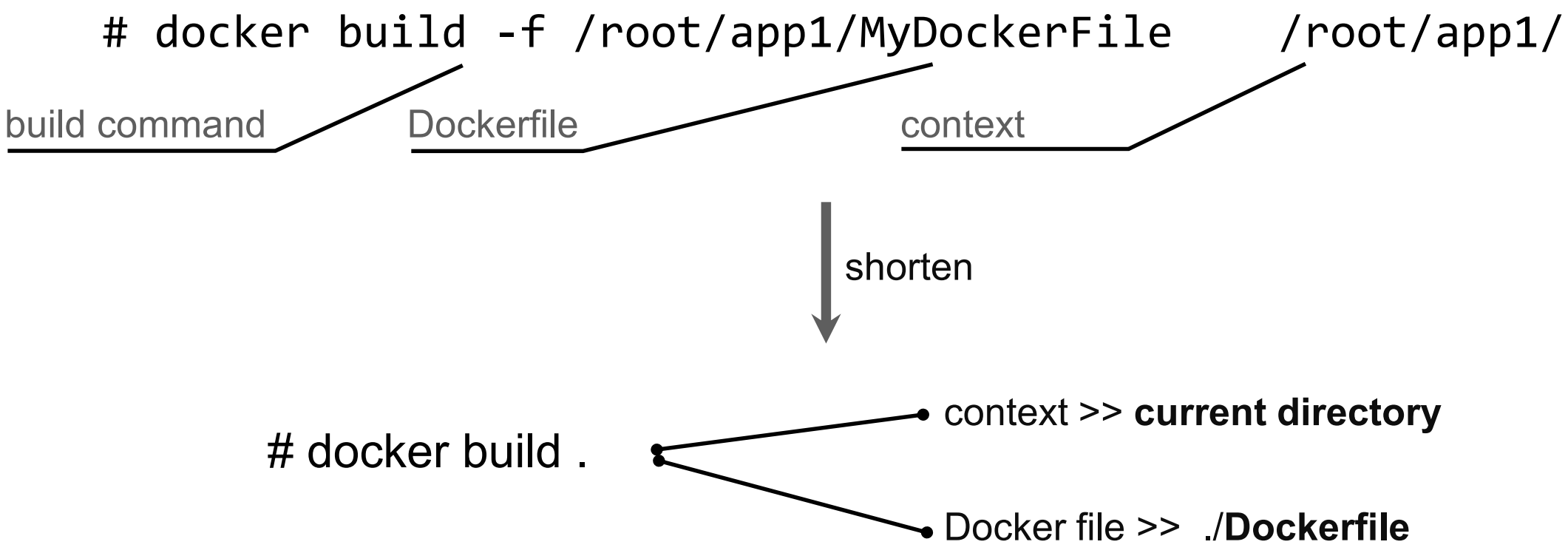
**The build is run by the Docker daemon**, not by the CLI. The first thing a build process does is **send the entire context (recursively) to the daemon**. In most cases, it's best to start with an empty directory as context and keep your Dockerfile in that directory. Add only the files needed for building the Dockerfile.



To use a file in the build context, the Dockerfile refers to the file specified in an instruction, for example, a COPY instruction. To increase the build's performance, **exclude files and directories** by adding a **.dockerignore** file to the context directory.

```
[root@ansible mydockerfile]# cat .dockerignore
myfile
[root@ansible mydockerfile]# ls -lah ./myfile/
total 1001M
drwxr-xr-x. 2 root root    21 Nov  3 00:54 .
drwxr-xr-x. 3 root root  4.0K Nov  3 00:54 ..
-rw-r--r--. 1 root root 1000M Nov  2 02:04 mydummy
[root@ansible mydockerfile]# docker build .
Sending build context to Docker daemon 17.41 kB
Step 1/7 : FROM centos
 ---> 75835a67d134
Step 2/7 : ARG test
 ---> Using cache
 ---> 95370caaa950
Step 3/7 : ENV aa $test
 ---> Using cache
 ---> f081878ef69a
Step 4/7 : CMD echo "aa > "$aa
 ---> Using cache
 ---> 5ed5ea7eaf33
Step 5/7 : FROM centos
 ---> 75835a67d134
Step 6/7 : ENV bb $test
 ---> Using cache
 ---> 9cb04d2fa42a
Step 7/7 : CMD echo "bb > "$bb
 ---> Using cache
 ---> e053b7641f98
Successfully built e053b7641f98
[root@ansible mydockerfile]#
```

The Dockerfile is called Dockerfile and located in the root of the context. You use the -f flag with docker build to point to a Dockerfile anywhere in your file system.

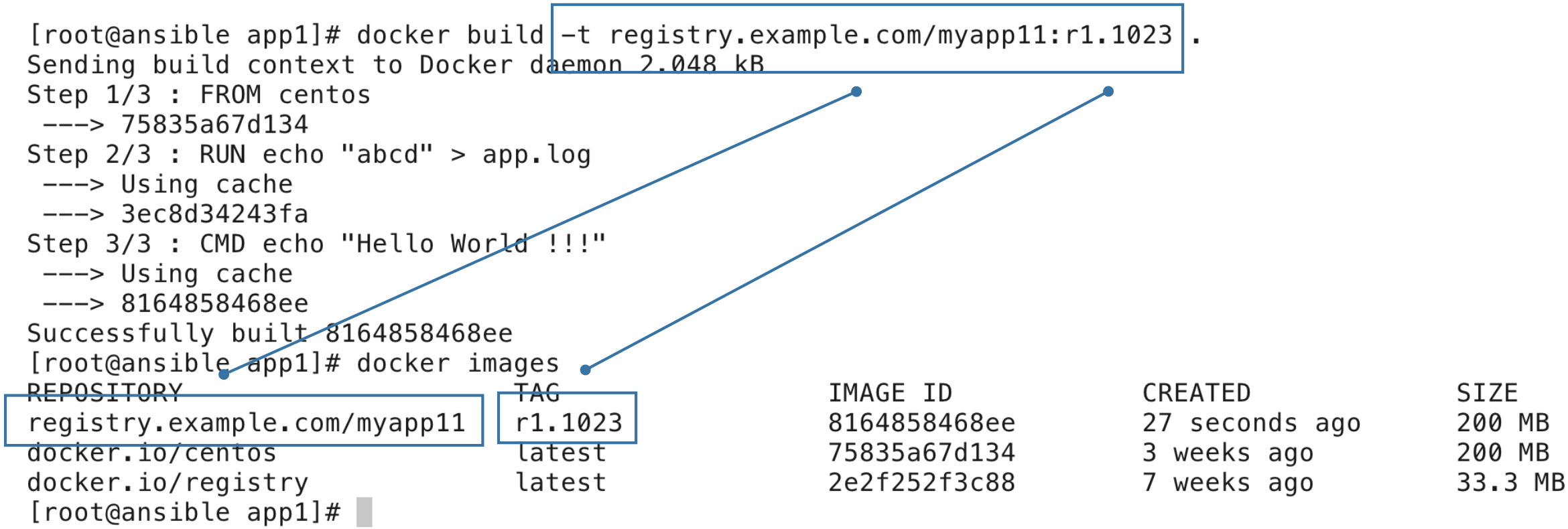


You can specify a repository and tag at which to save the new image if the build succeeds

```
[root@ansible app1]# docker build -t myapp1 .
Sending build context to Docker daemon 2.048 kB
Step 1/2 : FROM centos
 ---> 75835a67d134
Step 2/2 : CMD echo "Hello World"
 ---> Using cache
 ---> 94a6513b4a9d
Successfully built 94a6513b4a9d
[root@ansible app1]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
myapp1              latest              94a6513b4a9d        5 minutes ago       200 MB
docker.io/centos    latest              75835a67d134        3 weeks ago         200 MB
```

You can specify a **repository and tag** at which to save the new image if the build succeeds

```
[root@ansible app1]# docker build -t registry.example.com/myapp11:r1.1023 .
Sending build context to Docker daemon 2.048 kB
Step 1/3 : FROM centos
 ---> 75835a67d134
Step 2/3 : RUN echo "abcd" > app.log
 ---> Using cache
 ---> 3ec8d34243fa
Step 3/3 : CMD echo "Hello World !!!"
 ---> Using cache
 ---> 8164858468ee
Successfully built 8164858468ee
[root@ansible app1]# docker images
REPOSITORY                      TAG                 IMAGE ID            CREATED             SIZE
registry.example.com/myapp11    r1.1023             8164858468ee        27 seconds ago      200 MB
docker.io/centos                latest              75835a67d134        3 weeks ago         200 MB
docker.io/registry              latest              2e2f252f3c88        7 weeks ago         33.3 MB
[root@ansible app1]#
```

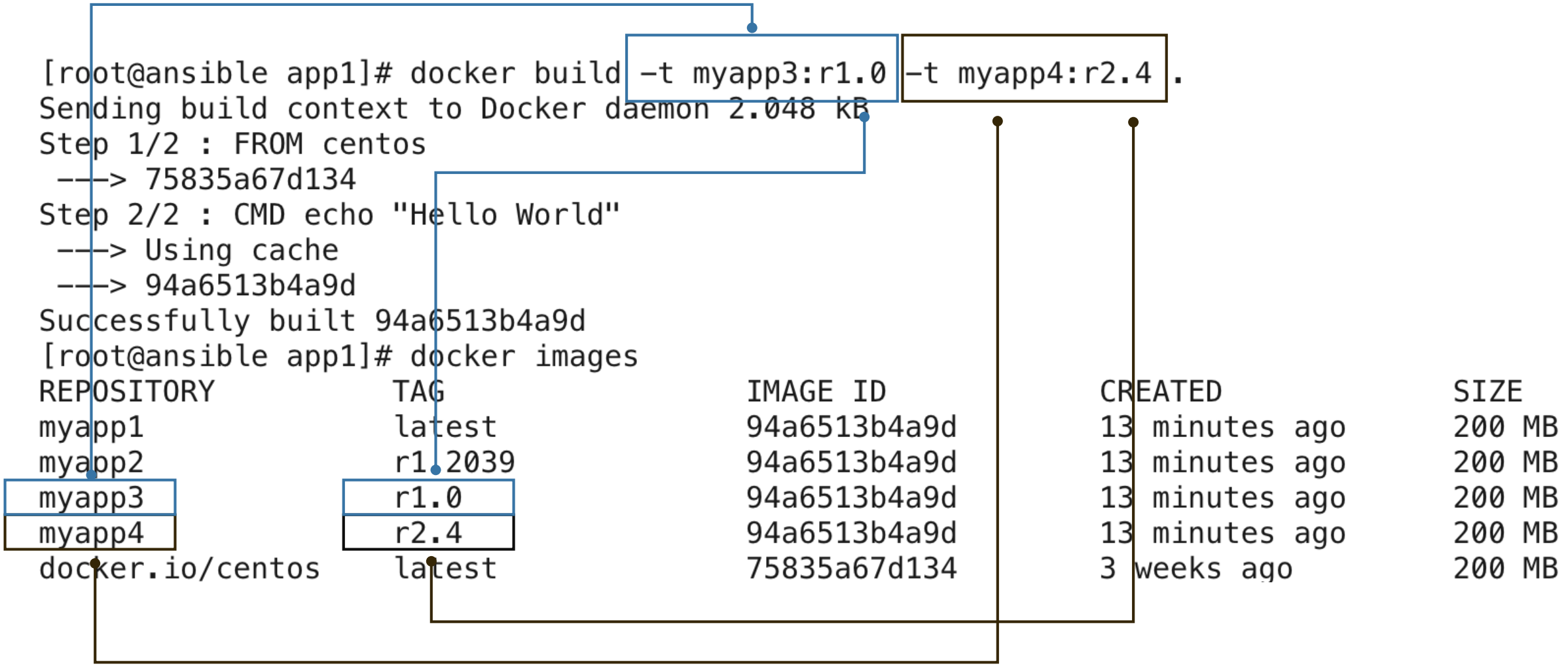


Why use **repository and tag** when build?

```
[root@ansible app1]# docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
myapp21                       latest              c82f8bdd81b2        19 seconds ago      200 MB
registry.example.com/myapp22  la                                      19 seconds ago      200 MB
docker.io/centos              la                                      3 weeks ago         200 MB
docker.io/registry            latest              2e2f252f3c88        7 weeks ago         33.3 MB
[root@ansible app1]# docker push registry.example.com/myapp21
The push refers to a repository [registry.example.com/myapp21]
An image does not exist locally with the tag: registry.example.com/myapp21
[root@ansible app1]# docker push registry.example.com/myapp22
The push refers to a repository [registry.example.com/myapp22]
9f0b5ae04617: Pushed
f972d139738d: Layer already exists
latest: digest: sha256:92ff683fddc006d4ba655b832081079a9f21c3ac0a1bd68aa76ca30f10670fa6 size: 736
[root@ansible app1]#
```

Cannot push to registry.example.com

To tag the image into **multiple repositories** after the build, **add multiple -t parameters** when you run the build command



Before the Docker daemon runs the instructions in the Dockerfile, it performs a preliminary validation of the Dockerfile and returns an error if the syntax is incorrect.

```
[root@ansible app1]# cat Dockerfile
FROM centos
CMDX echo "Hello World !!"
[root@ansible app1]# docker build .
Sending build context to Docker daemon 2.048 kB
Error response from daemon: Unknown instruction: CMDX
[root@ansible app1]#
```

Docker will re-use the intermediate images (cache), to accelerate the docker build process significantly.

First time build

```
[root@ansible app1]# docker build .
Sending build context to Docker daemon 2.048 kB
Step 1/3 : FROM centos
 ---> 75835a67d134
Step 2/3 : RUN echo "abcd" > app.log
 ---> Running in b3d18b445eb4

 ---> 0161c339d0e1
Removing intermediate container b3d18b445eb4
Step 3/3 : CMD echo "Hello World !!!"
 ---> Running in 8c508abc3d1f
 ---> 1e451f3eff9b
Removing intermediate container 8c508abc3d1f
Successfully built 1e451f3eff9b
[root@ansible app1]# docker build .
Sending build context to Docker daemon 2.048 kB
Step 1/3 : FROM centos
 ---> 75835a67d134
Step 2/3 : RUN echo "abcd" > app.log
 ---> Using cache
 ---> 0161c339d0e1
Step 3/3 : CMD echo "Hello World !!!"
 ---> Using cache
 ---> 1e451f3eff9b
Successfully built 1e451f3eff9b
[root@ansible app1]#
```

```
[root@ansible app1]# docker build .
Sending build context to Docker daemon 2.048 kB
Step 1/3 : FROM centos
 ---> 75835a67d134
Step 2/3 : RUN echo "abcd" > app.log
 ---> Using cache
 ---> 0161c339d0e1
Step 3/3 : CMD echo "Hello World !!!"
 ---> Using cache
 ---> 1e451f3eff9b
Successfully built 1e451f3eff9b
```

Force no cache

```
[root@ansible app1]# docker build . --no-cache
Sending build context to Docker daemon 2.048 kB
Step 1/3 : FROM centos
 ---> 75835a67d134
Step 2/3 : RUN echo "abcd" > app.log
 ---> Running in 4a8245cc5046

 ---> 93dd24757d49
Removing intermediate container 4a8245cc5046
Step 3/3 : CMD echo "Hello World !!!"
 ---> Running in 5b9045a4bb8d
 ---> 0f9b0f23bc1a
Removing intermediate container 5b9045a4bb8d
Successfully built 0f9b0f23bc1a
[root@ansible app1]#
```

# Dockerfile Format

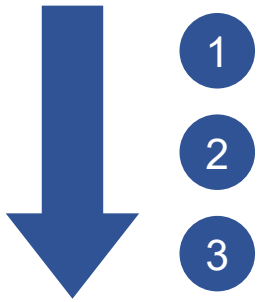
case insensitive

```
# Comment
INSTRUCTION arguments
```

convention is for them to be UPPERCASE to distinguish them from arguments more easily

Docker runs instructions in a Dockerfile in order.

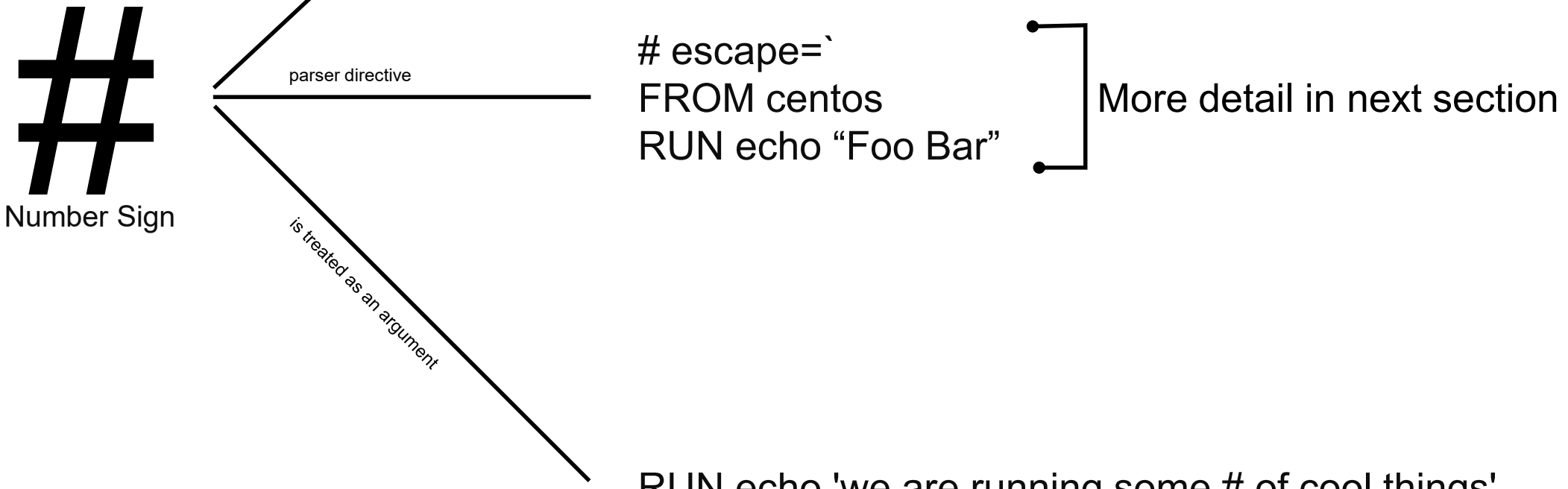
must start with a `FROM` instruction




Dockerfile

```
FROM centos
RUN echo “Foo Bar” >
app.log
CMD echo "Hello World !!!"
```

# Dockerfile Format



# Dockerfile Format

parser directive

Parser directives are **optional**, and affect the way in which subsequent lines in a Dockerfile are handled. Parser directives do not add layers to the build, and will not be shown as a build step. Parser directives are written as a **special type of comment** in the form **# directive=value**. A single directive may only be used once.

"**escape**" is the only supported directive

As of November 2018

Parser directive

```
# directive=value1
# directive=value2

FROM ImageName
```

Comment

```
FROM ImageName
# directive=value
```

# Dockerfile Format

escape

the default escape

# escape=\ (backslash)

Or

# escape=` (backtick)

scape character to ` is especially useful on Windows, where \ is the directory path separator. ` is consistent with Windows PowerShell.

```
FROM microsoft/nanoserver
COPY testfile.txt c:\\
RUN dir c:\
```

would be interpreted as an escape for the newline

COPY testfile.txt c:\RUN dir c:

```
# escape=`

FROM microsoft/nanoserver
COPY testfile.txt c:\
RUN dir c:\
```

Environment variables (declared with the **ENV** statement) can also be used in certain instructions **as variables** to be interpreted by the Dockerfile. Escapes are also handled for including variable-like syntax into a statement literally.

Environment variables are notated in the Dockerfile either with **$variable_name** or **${variable_name}**. They are treated equivalently and the brace syntax is typically used to address issues with variable names with no whitespace, like ${foo}_bar.

```
FROM centos
ENV MYVAR=ThisIsMyVAR
RUN echo "\$MYVAR IS "${MYVAR} > /app.log
RUN echo "\$MYVAR with _Additional1 IS "${MYVAR}_Additional1 >>  /app.log
RUN echo "\$MYVAR with Additional2 IS "${MYVAR}Addtional2 >>  /app.log
ENV MYVAR2=${MYVAR}AssignToMYVAR2
RUN echo "" >> /app.log
RUN echo "Assign New value to \$MYVAR2 >> "${MYVAR2} >> /app.log
CMD cat /app.log
[root@ansible app1]# docker run my
$MYVAR IS ThisIsMyVAR
$MYVAR with _Additional1 IS ThisIsMyVAR_Additional1
$MYVAR with Additional2 IS ThisIsMyVARAddtional2

Assign New value to $MYVAR2 >> ThisIsMyVARAssignToMYVAR2
```

# Environment replacement

What is $def values in 2nd line?

```
ENV abc=hello
ENV abc=bye def=$abc
ENV ghi=$abc
```

$def's value is hello

```
FROM centos
ENV abc=hello
RUN echo "\$abc is "${abc} > /app.log
ENV abc=bye def=$abc
RUN echo "\$abc is "${abc} >> /app.log
RUN echo "\$def is "${def} >> /app.log
ENV ghi=$abc
RUN echo "\$ghi is "${ghi} >> /app.log
CMD cat /app.log
[root@ansible app1]# docker run my
$abc is hello
$abc is bye
$def is hello
$ghi is bye
```

**Environment variable substitution will use the same value for each variable throughout the entire instruction.**

# **Initialize** a new build stage

FROM

A Dockerfile **must start with a FROM** instruction. The image can be any valid image – it is especially easy to start by **pulling an image** from the Public Repositories.

```
FROM <image> [AS <name>]
Or
FROM <image>[:<tag>] [AS <name>]
Or
FROM <image>[@<digest>] [AS <name>]
```

Images that use the v2 or later format have a content-addressable identifier called a **digest**. As long as the input used to generate the image is unchanged, the digest value is predictable.

# FROM

```
FROM <image> [AS <name>]
Or
FROM <image>[:<tag>] [AS <name>]
Or
FROM <image>[@<digest>] [AS <name>]
```

- Set the **Base Image** for subsequent instructions.
- FROM can appear **multiple times** within a single Dockerfile to create multiple images or use one build stage as a dependency for another.
- The last image ID output by the commit before each new FROM instruction. Each FROM instruction clears any state created by previous instructions.
- The **tag or digest values are optional**. If you omit either of them, the builder assumes a latest tag by default. The builder returns an error if it cannot find the tag value.

# RUN

RUN <command> (shell form, the command is run in a shell, which by default is /bin/sh -c on Linux or cmd /S /C on Windows)

RUN ["executable", "param1", "param2"] (exec form)

- **The RUN instruction will execute any commands in a new layer** on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.
- Layering RUN instructions and generating commits conforms to the core concepts of Docker where commits are cheap and containers can be created from any point in an image's history, much like source control.
- The exec form makes it possible to avoid shell string munging, and to RUN commands using a base image that does not contain the specified shell executable.
- The default shell for the shell form can be changed using the SHELL command.
- In the shell form **you can use a \ (backslash) to continue a single RUN instruction** onto the next line.

```
RUN /bin/bash -c 'source $HOME/.bashrc; \
    echo $HOME'
==
RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'
```

# CMD

CMD ["executable","param1","param2"] (exec form, this is the preferred form)

CMD ["param1","param2"] (as default parameters to ENTRYPOINT)

CMD command param1 param2 (shell form)

There can only be one CMD instruction in a Dockerfile. If you list more than one CMD then only the last CMD will take effect.

- The main purpose of a CMD is **to provide defaults for an executing container.** These defaults can include an executable, **or they can omit the executable, in which case you must specify an ENTRYPOINT instruction as well.**
- When used in the shell or exec formats, the CMD instruction sets the command to be executed when running the image.
- If you use the shell form of the CMD, then the <command> will execute in /bin/sh -c:

```
FROM ubuntu
CMD echo "This is a test." | wc -
```

If you want to run your <command> without a shell then you must express the command as a JSON array and give the full path to the executable. This array form is the preferred format of CMD. Any additional parameters must be individually expressed as strings in the array:

**If the user specifies arguments to docker run then they will override the default specified in CMD.**

# ENTRYPOINT

```
ENTRYPOINT ["executable", "param1", "param2"] (exec form, preferred)
ENTRYPOINT command param1 param2 (shell form)
```

- An ENTRYPOINT allows you to configure a container that will run as an executable.
- Command line arguments to docker run <image> will be appended after all elements in an exec form ENTRYPOINT, and will override all elements specified using CMD. **This allows arguments to be passed to the entry point, i.e., docker run <image> -d will pass the -d argument to the entry point**. You can override the ENTRYPOINT instruction using the docker run --entrypoint flag.
- The shell form prevents any CMD or run command line arguments from being used, but has the disadvantage that your ENTRYPOINT will be started as a subcommand of /bin/sh -c, which does not pass signals. This means that the executable will not be the container's PID 1 - and will not receive Unix signals - so your executable will not receive a SIGTERM from docker stop <container>.
- **Only the last ENTRYPOINT instruction in the Dockerfile will have an effect.**

```
FROM ubuntu
ENTRYPOINT ["top", "-b"]
CMD [“-c”]

FROM ubuntu
ENTRYPOINT exec top -b
```

# LABEL

```
LABEL <key>=<value> <key>=<value> <key>=<value> ...
```

- The LABEL instruction **adds metadata to an image.** A LABEL is a key-value pair. To include spaces within a LABEL value, use quotes and backslashes as you would in command-line parsing.

```
LABEL "com.example.vendor"="ACME Incorporated"
LABEL com.example.label-with-value="foo"
LABEL version="1.0"
LABEL description="This text illustrates \
that label-values can span multiple lines.”
```

# EXPOSE

EXPOSE <port> [<port>/<protocol>...]

- The EXPOSE instruction informs **Docker that the container listens on the specified network ports at runtime**. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.
- The EXPOSE instruction **does not actually publish the port**. It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published. To actually publish the port when running the container, use the -p flag on docker run to publish and map one or more ports, or **the -P flag to publish all exposed ports and map them to high-order ports**.
- By default, EXPOSE assumes TCP. You can also specify UDP

EXPOSE 80/udp
To expose on both TCP and UDP, include two lines:
EXPOSE 80/tcp
EXPOSE 80/udp
In this case, if you use -P with docker run, the port will be exposed once for TCP and once for UDP. Remember that -P uses an ephemeral high-ordered host port on the host, so the port will not be the same for TCP and UDP.
Regardless of the EXPOSE settings, you can override them at runtime by using the -p flag. For example
docker run -p 80:80/tcp -p 80:80/udp ...

# ENV

```
ENV <key> <value>
ENV <key>=<value>
```

- The ENV instruction sets the environment variable <key> to the value <value>. This value will be in the environment for all subsequent instructions in the build stage and can be replaced inline in many as well.
- The ENV instruction has two forms. The first form, **ENV <key> <value>, will set a single variable to a value**. The **entire string after the first space will be treated as the <value>** - including whitespace characters. The value will be interpreted for other environment variables, so quote characters will be removed if they are not escaped.
- The second form, **ENV <key>=<value> ..., allows for multiple variables to be set at one time**. Notice that the second form uses the equals sign (=) in the syntax, while the first form does not. Like command line parsing, quotes and backslashes can be used to include spaces within values.

```
ENV myName="John Doe" myDog=Rex\ The\ Dog \
    myCat=fluffy
and
ENV myName John Doe
ENV myDog Rex The Dog
ENV myCat fluffy
```

The environment variables set using ENV will persist when a container is run from the resulting image. You can **view the values using docker inspect**, and change them using **docker run --env <key>=<value>**.

# COPY

```
COPY [--chown=<user>:<group>] <src>... <dest>
COPY [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

- The COPY instruction **copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.**
- **Multiple <src>** resources may be specified but the paths of files and directories will be interpreted as relative to the source of the context of the build.
- Each <src> may contain **wildcards** and matching will be done using Go's filepath.Match rules.
- The **<dest> is an absolute path,** or a path relative to **WORKDIR**, into which the source will be copied inside the destination container.
- All new files and directories are created with a **UID and GID of 0**, unless the optional --chown flag specifies a given username, groupname, or UID/GID combination to request specific ownership of the copied content. The format of the **--chown flag allows for either username and groupname strings or direct integer UID and GID in any combination.** Providing a username without groupname or a UID without GID will use the same numeric UID as the GID. If a username or groupname is provided, the container's root filesystem /etc/passwd and /etc/group files will be used to perform the translation from name to integer UID or GID respectively.

```
COPY hom* /mydir/       # adds all files starting with "hom"
COPY hom?.txt /mydir/    # ? is replaced with any single character, e.g., "home.txt"
COPY test relativeDir/   # adds "test" to `WORKDIR`/relativeDir/
COPY test /absoluteDir/  # adds "test" to /absoluteDir/COPY --chown=55:mygroup files* /somedir/
COPY --chown=bin files* /somedir/
COPY --chown=1 files* /somedir/
COPY --chown=10:11 files* /somedir/
```

# ADD

```
ADD [--chown=<user>:<group>] <src>... <dest>
ADD [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

- The ADD instruction **copies new files, directories or remote file URLs** from <src> and adds them to the filesystem of the image at the path <dest>.
- Multiple <src> resources may be specified but if they are files or directories, their paths are interpreted as relative to the source of the context of the build.
- Each <src> may contain wildcards and matching will be done using
- All new files and directories are created with a UID and GID of 0, unless the optional --chown flag specifies a given username, groupname, or UID/GID combination to request specific ownership of the content added. The format of the --chown flag allows for either username and groupname strings or direct integer UID and GID in any combination. Providing a username without groupname or a UID without GID will use the same numeric UID as the GID. If a username or groupname is provided, the container's root filesystem /etc/passwd and /etc/group files will be used to perform the translation from name to integer UID or GID respectively.

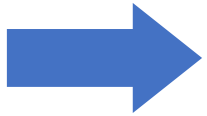
```
ADD test relativeDir/          # adds "test" to `WORKDIR`/relativeDir/
ADD test /absoluteDir/         # adds "test" to /absoluteDir/
ADD hom* /mydir/        # adds all files starting with "hom"
ADD hom?.txt /mydir/    # ? is replaced with any single character, e.g., “home.txt"
ADD --chown=55:mygroup files* /somedir/
ADD --chown=bin files* /somedir/
ADD --chown=1 files* /somedir/
ADD --chown=10:11 files* /somedir/
```

Docker suggests that it is often not efficient to copy from a URL using ADD, and it is best practice to use other strategies to include the required remote files.

Because **image size matters**, using ADD to fetch packages from remote URLs is strongly discouraged; you should use curl or wget instead. That way you **can delete the files you no longer need** after they've been extracted and you don't have to add another layer in your image.

—Dockerfile Best Practices

```
ADD http://example.com/big.tar.xz /usr/src/things/
RUN tar -xJf /usr/src/things/big.tar.xz -C /usr/src/things
RUN make -C /usr/src/things all
```



```
RUN mkdir -p /usr/src/things \
    && curl -SL http://example.com/big.tar.xz \
    | tar -xJC /usr/src/things \
    && make -C /usr/src/things all
```

For other items (files, directories) that do not require ADD's tar auto-extraction capability, you should always use COPY.

# VOLUME

```
VOLUME ["/data"]
```

- The VOLUME instruction **creates a mount point with the specified name and marks it as holding externally mounted volumes** from native host or other containers. The value can be a JSON array, VOLUME ["/var/log/"], or a plain string with multiple arguments, such as VOLUME /var/log or VOLUME /var/log /var/db.
- The docker run command initializes the newly created volume with any data that exists at the specified location within the base image.

```
FROM ubuntu
RUN mkdir /myvol
RUN echo "hello world" > /myvol/greeting
VOLUME /myvol
```

# USER

```
USER <user>[:<group>] or
USER <UID>[:<GID>]
```

- The USER instruction **sets the user name** (or UID) and optionally the user group (or GID) to use **when running the image and for any RUN, CMD and ENTRYPOINT** instructions that follow it in the Dockerfile.

```
FROM centos
RUN useradd mytest
USER mytest
CMD id
```

# WORKDIR

```
WORKDIR /path/to/workdir
```

- The WORKDIR instruction **sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD** instructions that follow it in the Dockerfile. If the WORKDIR doesn't exist, it will be created even if it's not used in any subsequent Dockerfile instruction.
- The WORKDIR instruction can be used multiple times in a Dockerfile. If a relative path is provided, it will be relative to the path of the previous WORKDIR instruction.

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

The output of the final pwd command in this Dockerfile would be /a/b/c.

The WORKDIR instruction can resolve environment variables previously set using ENV. You can only use environment variables explicitly set in the Dockerfile.

```
ENV DIRPATH /path
WORKDIR $DIRPATH/$DIRNAME
RUN pwd
```

The output of the final pwd command in this Dockerfile would be /path/$DIRNAME

# ARG

ARG <name>[=<default value>]

- The ARG instruction defines a variable that **users can pass at build-time to the builder with the docker build command using the --build-arg <varname>=<value> flag.** If a user specifies a build argument that was not defined in the Dockerfile, the build outputs a warning.
- [Warning] One or more build-args [foo] were not consumed.
- A Dockerfile may include one or more ARG instructions.

- **Default values** An ARG instruction can optionally include a default value.

```
FROM busybox
ARG user1=someuser
ARG buildno=1
... If an ARG instruction has a default value and if there is no value passed at build-time, the builder uses the default.
```

# ARG

- Scope
  - An ARG variable definition comes into effect from the line on which it is defined in the Dockerfile not from the argument's use on the command-line or elsewhere.
  - An ARG instruction goes out of scope at the end of the build stage where it was defined. To use an arg in multiple stages, each stage must include the ARG instruction.

```
FROM centos
RUN useradd user1; useradd user2;mkdir /mytest; chmod 777 /mytest
WORKDIR /mytest
USER ${user:-user1}
RUN id > cmd.log ; chmod 777 cmd.log
ARG user
USER $user
RUN id >> cmd.log
CMD cat /mytest/cmd.log

$ docker build --build-arg user=user2 -t testcmd .
```

# ARG

- Using ARG variables
  - You can use an ARG or an ENV instruction to specify variables that are available to the RUN instruction. Environment variables defined using the ENV instruction always override an ARG instruction of the same name. Consider this Dockerfile with an ENV and ARG instruction.
  - This behavior is similar to a shell script where a locally scoped variable overrides the variables passed as arguments or inherited from environment, from its point of definition.

```
FROM centos
ARG mydata
ENV mydata ${mydata:-thisisdefault}
CMD echo $mydata

$docker build --build-arg mydata=user1 -t testarg2 .
$ docker run testarg2
user1

$  docker build -t testarg2 .
$  docker run testarg2
thisisdefault
```

# ARG

- Predefined ARGs

  Docker has a set of predefined ARG variables that you can use without a corresponding ARG instruction in the Dockerfile.
  - HTTP_PROXY
  - http_proxy
  - HTTPS_PROXY
  - https_proxy
  - FTP_PROXY
  - ftp_proxy
  - NO_PROXY
  - no_proxy

```
FROM centos
RUN echo $http_proxy > /bb.txt
RUN echo $test_arg >> /bb.txt
CMD cat /bb.txt

$ docker build --build-arg http_proxy=http://proxy.example.com:8080 --build-arg
test_arg=mytest  -t testarg3 .

[Warning] One or more build-args [test_arg] were not consumed
```

# ONBUILD

ONBUILD [INSTRUCTION]

- The ONBUILD **instruction adds to the image a trigger instruction to be executed at a later time, when the image is used as the base for another build.** The trigger will be executed in the context of the downstream build, as if it had been inserted immediately after the FROM instruction in the downstream Dockerfile.
- Triggers are cleared from the final image after being executed. In other words they are not inherited by “grand-children” builds.

```
FROM centos
RUN echo 1 > /aa.txt
RUN echo 2 >> /aa.txt
ONBUILD RUN echo 3 >> /aa.txt
RUN echo 4 >> /aa.txt
CMD cat /aa.txt

$ docker build -t testonbuild .
$ docker run testonbuild
1
2
4
```

```
FROM testonbuild
RUN echo "new line" >> /aa.txt
CMD cat /aa.txt

$ docker build -t testonbuild2 .
Step 1/3 : FROM testonbuild
# Executing 1 build trigger...
$ docker run testonbuild2
1
2
4
3
new line
```

# SHELL

```
SHELL ["executable", "parameters"]
```

- The SHELL instruction allows the default shell used for the shell form of commands to be **overridden**. The default shell on Linux is ["/bin/sh", "-c"], and on Windows is ["cmd", "/S", "/C"]. The SHELL instruction must be written in JSON form in a Dockerfile.
- The SHELL instruction is particularly useful on Windows where there are two commonly used and quite different native shells: cmd and powershell, as well as alternate shells available including sh.
- The SHELL instruction can appear multiple times. Each SHELL instruction overrides all previous SHELL instructions, and affects all subsequent instructions.

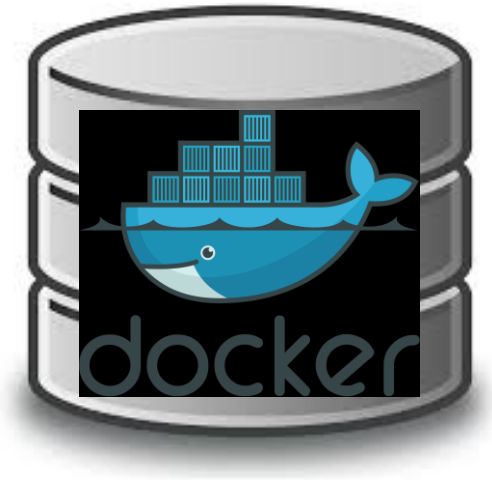
```
FROM microsoft/windowsservercore
# Executed as cmd /S /C echo default
RUN echo default
# Executed as cmd /S /C powershell -command Write-Host default
RUN powershell -command Write-Host default
# Executed as powershell -command Write-Host hello
SHELL ["powershell", "-command"]
RUN Write-Host hello
# Executed as cmd /S /C echo hello
SHELL ["cmd", "/S", "/C"]
RUN echo hello
```

# Docker Storage

# Managing data in Docker

By default all files created inside a container are stored on a writable container layer.

**The data doesn't persist when that container no longer exists**, and it can be difficult to get the data out of the container if another process needs it.

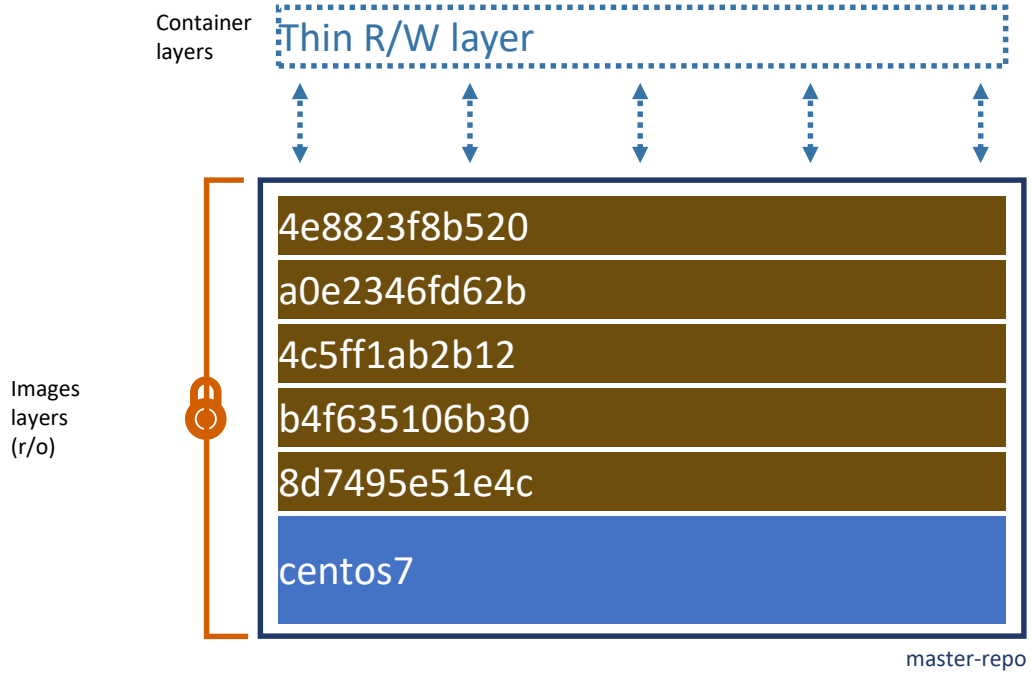



A container's writable layer is tightly coupled to the host machine where the container is running. **You can't easily move the data somewhere else.**

Writing into a container's writable layer **requires a storage driver to manage the filesystem.** The storage driver provides a union filesystem, using the Linux kernel. **This extra abstraction reduces performance** as compared to using data volumes, which write directly to the host filesystem.

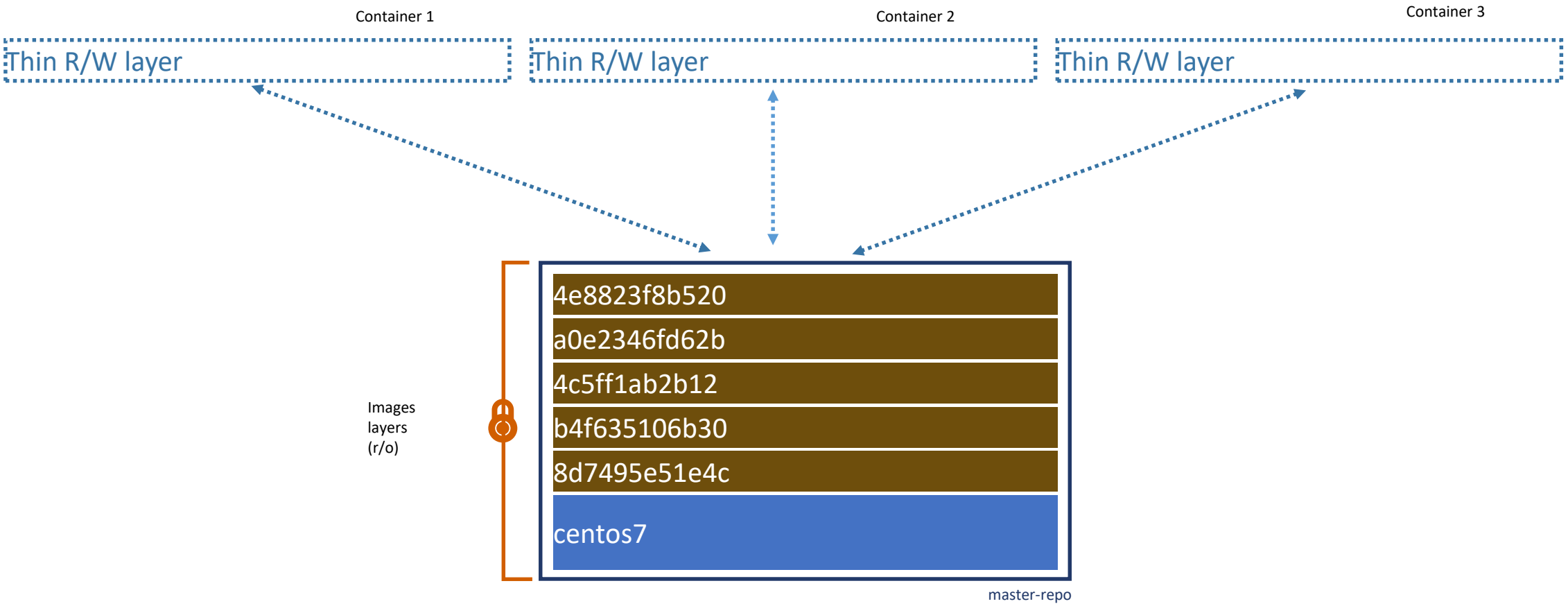
# Images and layers

```
TycheMini:container-repositry drs$ cat master-repo
FROM centos:centos7
LABEL maintainer="Damrongsak Reetanon <damrongs@gmail.com>"


RUN yum install -y epel-release && \
    yum install -y wget openssl sed  nginx &&\
    yum -y autoremove &&\
    yum clean all
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```



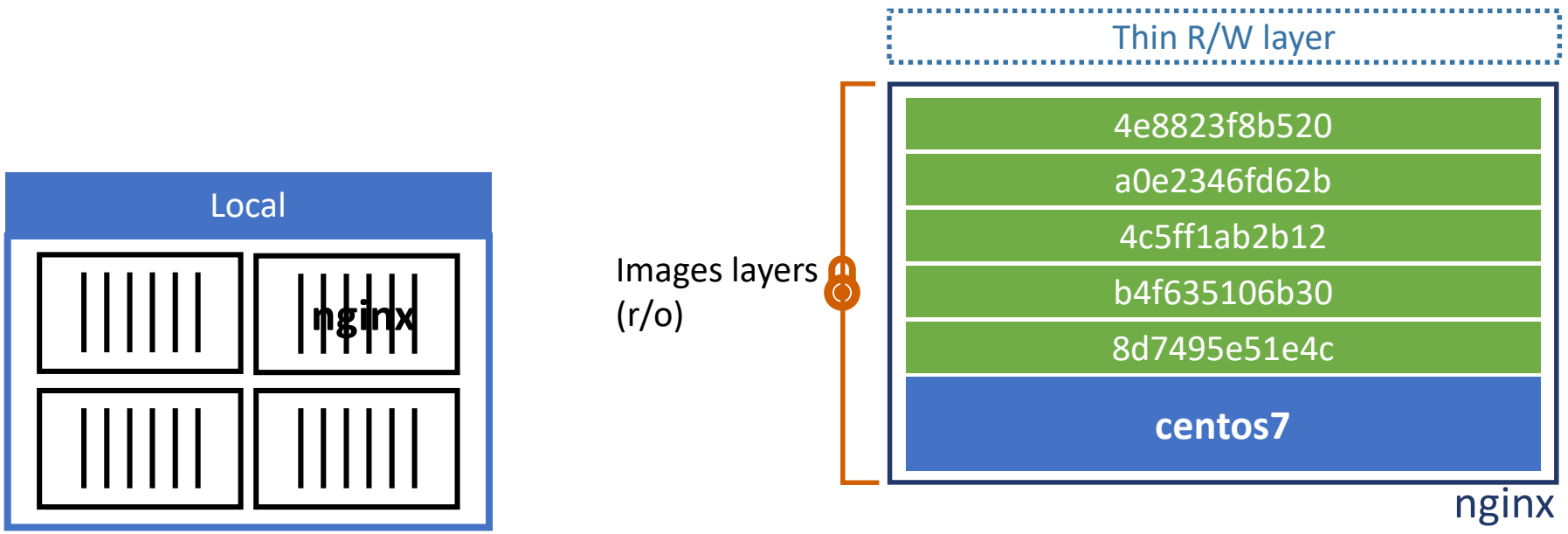

```
TycheMini:~ drs$ docker history master-repo
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
4e8823f8b520        4 weeks ago         /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
a0e2346fd62b        4 weeks ago         /bin/sh -c #(nop)  EXPOSE 80                    0B
4c5ff1ab2b12        4 weeks ago         /bin/sh -c #(nop) COPY file:06273b59b8e3d6bd…   2.51kB
b4f635106b30        4 weeks ago         /bin/sh -c yum install -y epel-release &&   …   82.7MB
8d7495e51e4c        4 weeks ago         /bin/sh -c #(nop)  LABEL maintainer=Damrongs…   0B
9f38484d220f        5 months ago        /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>           5 months ago        /bin/sh -c #(nop)  LABEL org.label-schema.sc…   0B
<missing>           5 months ago        /bin/sh -c #(nop) ADD file:074f2c974463ab38c…   202MB
```

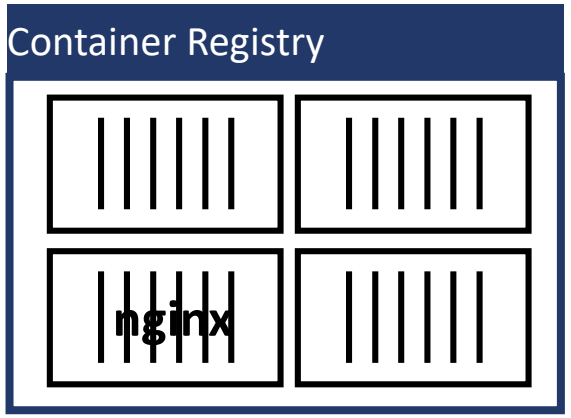
# Images and layers



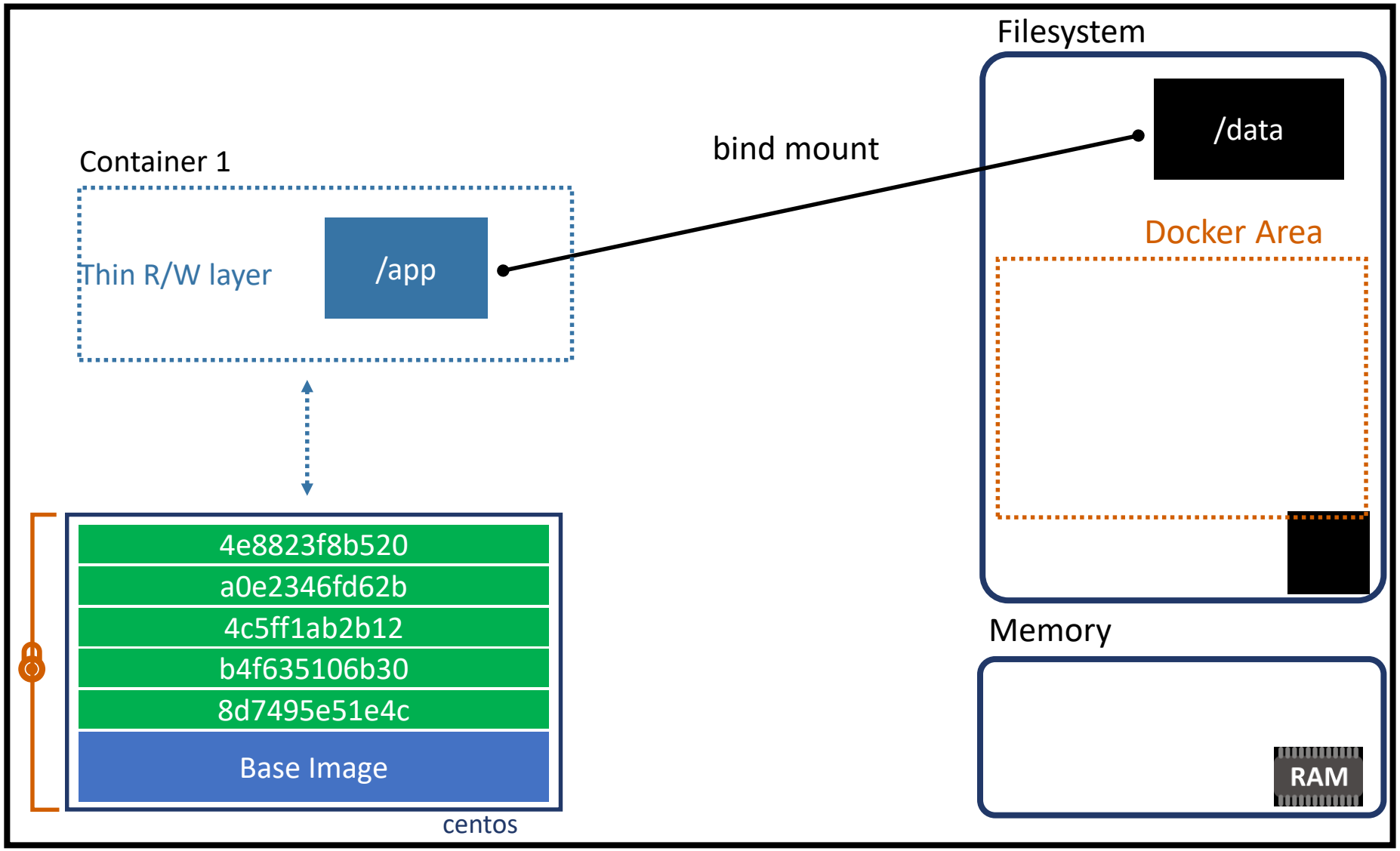
HOST

```
]$ docker run -d  --name myweb nginx

]$ docker stop myweb
]$ docker rm myweb
```

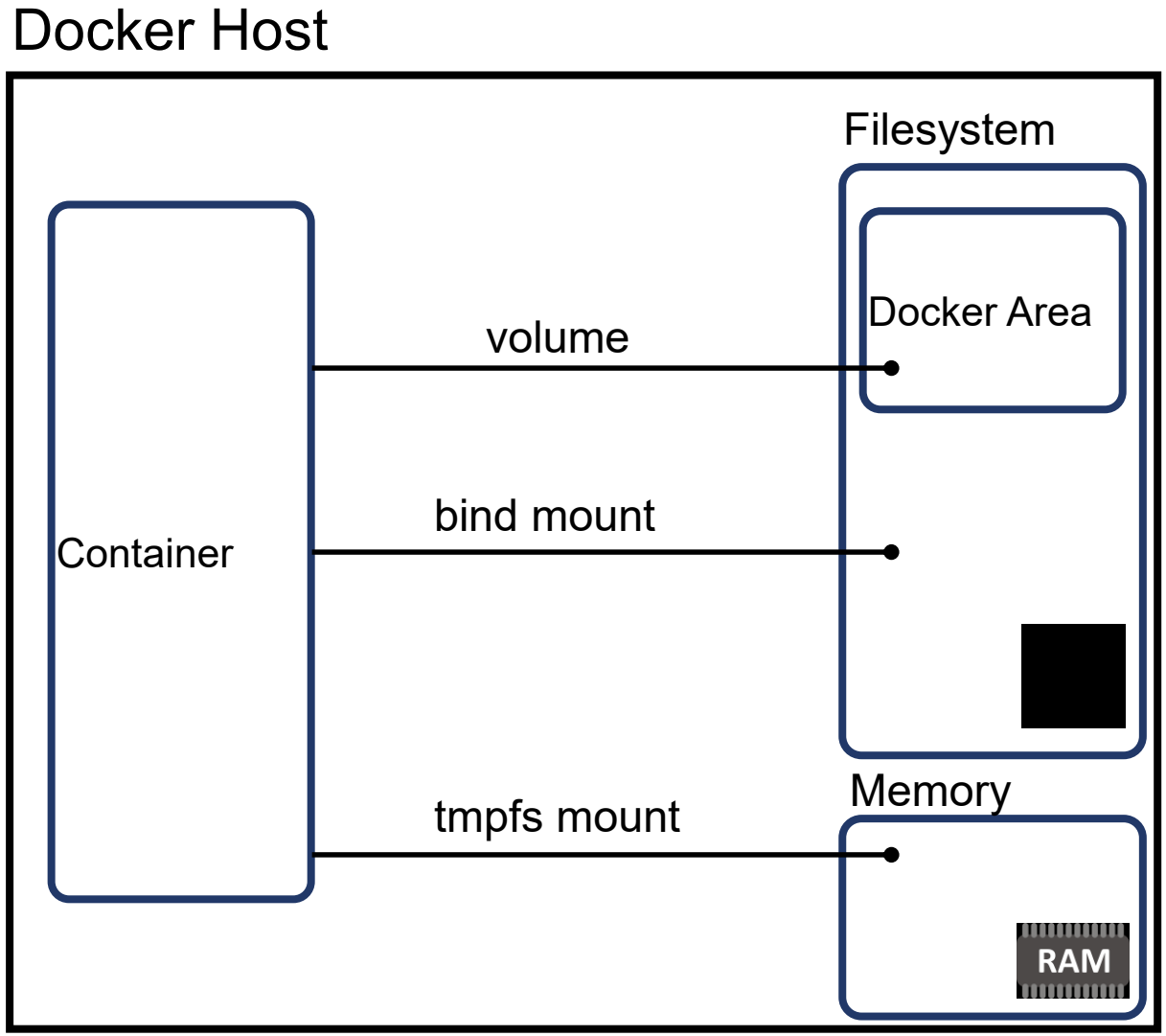


# bind mount


Docker Host
Filesystem
/data
bind mount
Container 1
Thin R/W layer
/app
Docker Area
4e8823f8b520
a0e2346fd62b
4c5ff1ab2b12
b4f635106b30
8d7495e51e4c
Base Image
centos
Memory
RAM

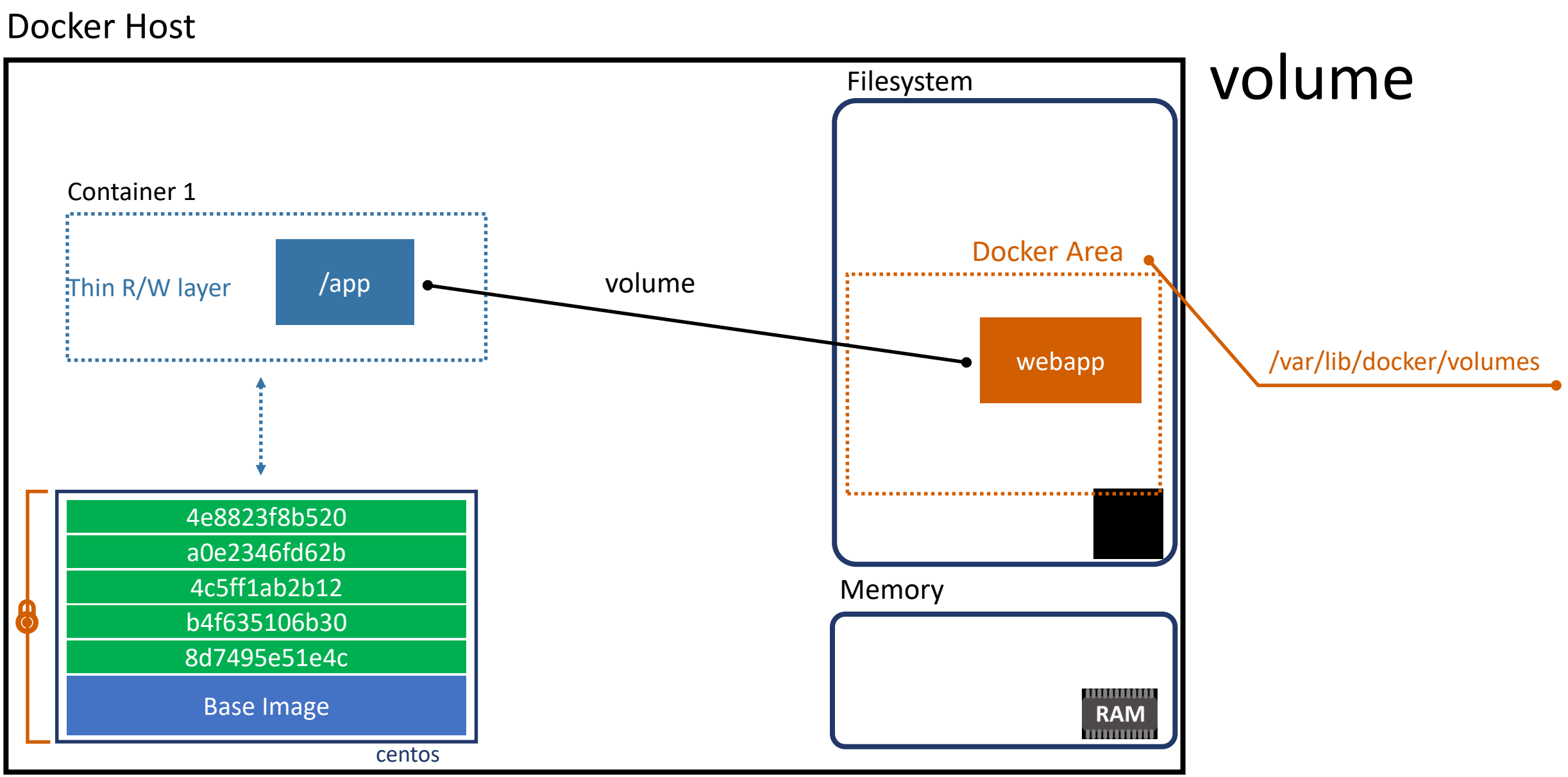
# Choose the right type of mount :: Bind mounts



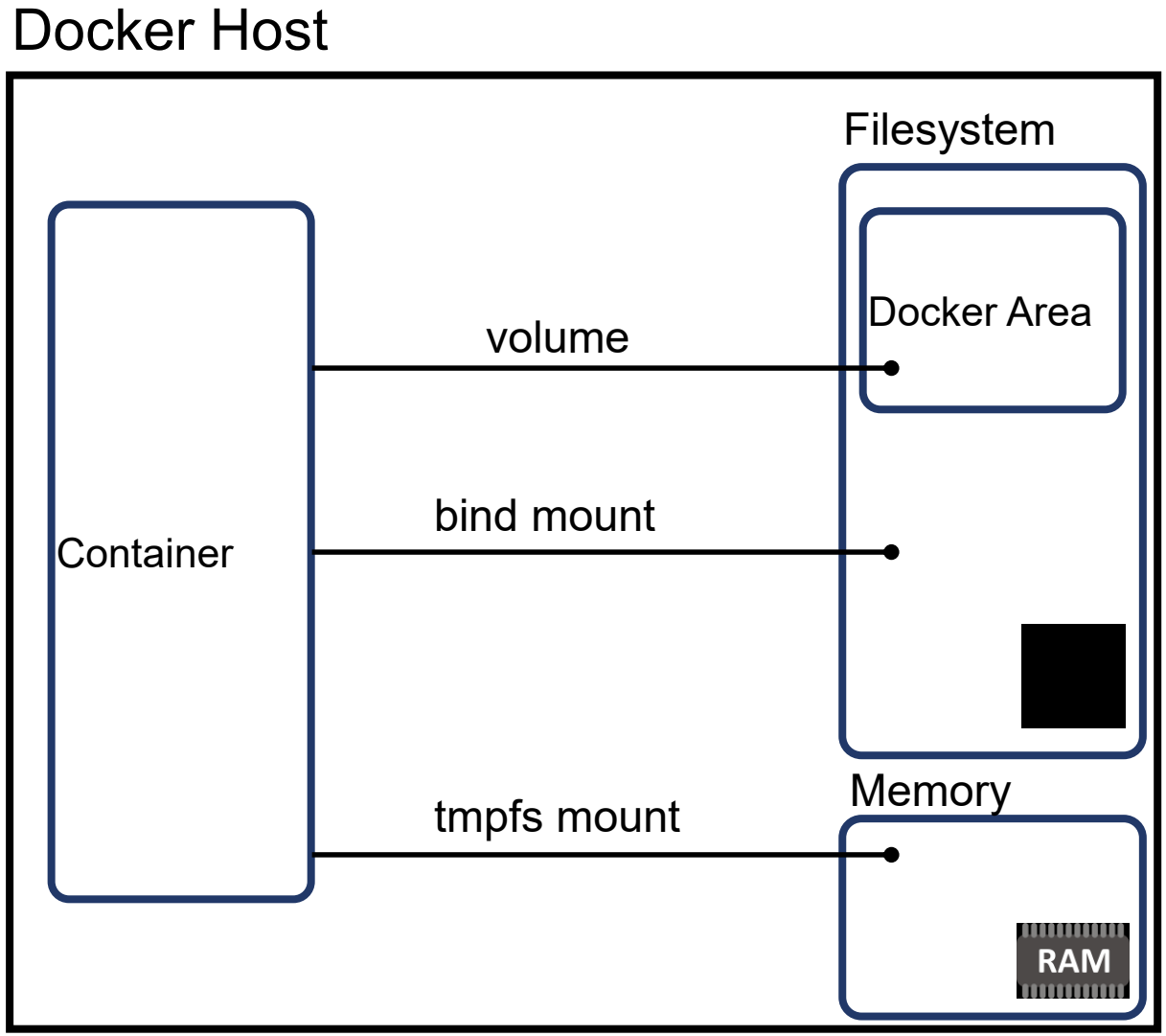

Bind mounts may be **stored anywhere on the host system**. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.

# Choose the right type of mount :: Bind mounts

```
[root@docker ~]# docker run -it --rm --mount type=bind,source=/tmp/,target=/xyz centos
/bin/bash
[root@50e4f5a7dcf8 /]# echo "Hi World" >> /xyz/bar.txt
[root@50e4f5a7dcf8 /]# exit
exit


[root@docker ~]# docker run -it --rm -v /tmp:/xyz centos /bin/bash
[root@4a01f2f2cefb /]# echo "This is a dog." >> /xyz/bar.txt
[root@4a01f2f2cefb /]# exit
exit


[root@docker ~]# cat /tmp/bar.txt
Hi World
This is a dog.
[root@docker ~]#
```


Docker Host
volume
Filesystem
Container 1
Thin R/W layer
/app
volume
Docker Area
webapp
/var/lib/docker/volumes
4e8823f8b520
a0e2346fd62b
4c5ff1ab2b12
b4f635106b30
8d7495e51e4c
Base Image
centos
Memory
RAM

# Choose the right type of mount :: Volumes




Volumes are stored in a part of the host filesystem which is **managed by Docker**(/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. **Volumes are the best way to persist data in Docker.**

- Volumes are easier to back up or migrate than bind mounts.
- You can manage volumes using Docker CLI commands or the Docker API.
- Volumes work on both Linux and Windows containers.
- Volumes can be more safely shared among multiple containers.
- Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
- New volumes can have their content pre-populated by a container.

# Choose the right type of mount :: Volumes

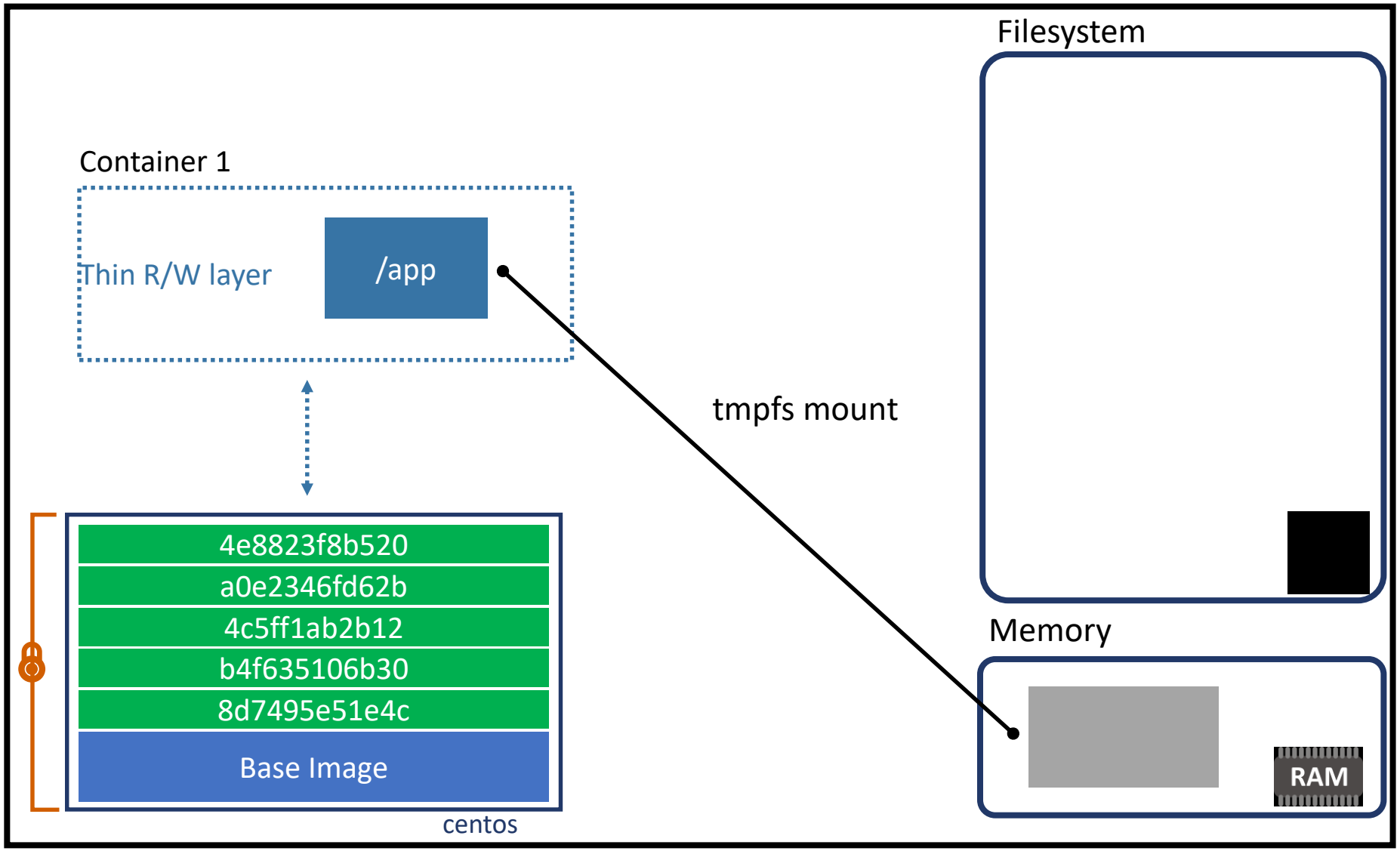
```
[root@docker ~]# docker volume create myvol
myvol
[root@docker ~]# docker volume inspect myvol
[
    {
        "CreatedAt": "2019-08-26T14:12:08+07:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/myvol/_data",
        "Name": "myvol",
        "Options": {},
        "Scope": "local"
    }
]
```

```
[root@docker ~]# cat
/var/lib/docker/volumes/myvol/_data/foo.txt
Hello World
```

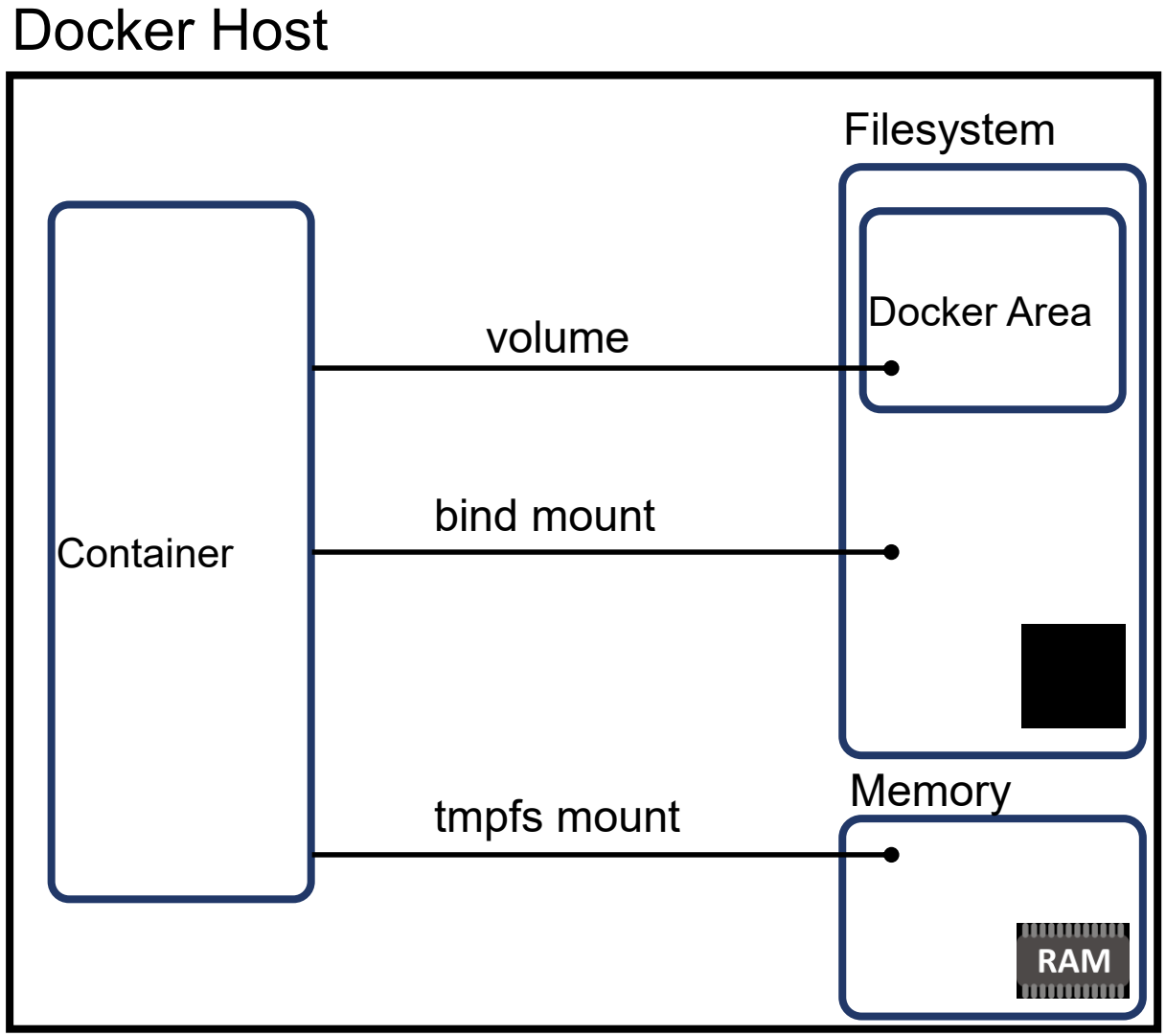
```
[root@docker ~]# docker run -it --rm -v myvol:/abc centos /bin/bash
[root@428d7798876b /]# echo "Hello World" >> /abc/foo.txt
```

```
[root@docker ~]# docker run -it --rm --mount source=myvol,target=/abc centos /bin/bash
[root@ec76c45fd8e3 /]# cat /abc/foo.txt
Hello World
```

# tmpfs mount


Docker Host
Filesystem
Container 1
Thin R/W layer
/app
tmpfs mount
4e8823f8b520
a0e2346fd62b
4c5ff1ab2b12
b4f635106b30
8d7495e51e4c
Base Image
centos
Memory
RAM

# Choose the right type of mount :: tmpfs mounts



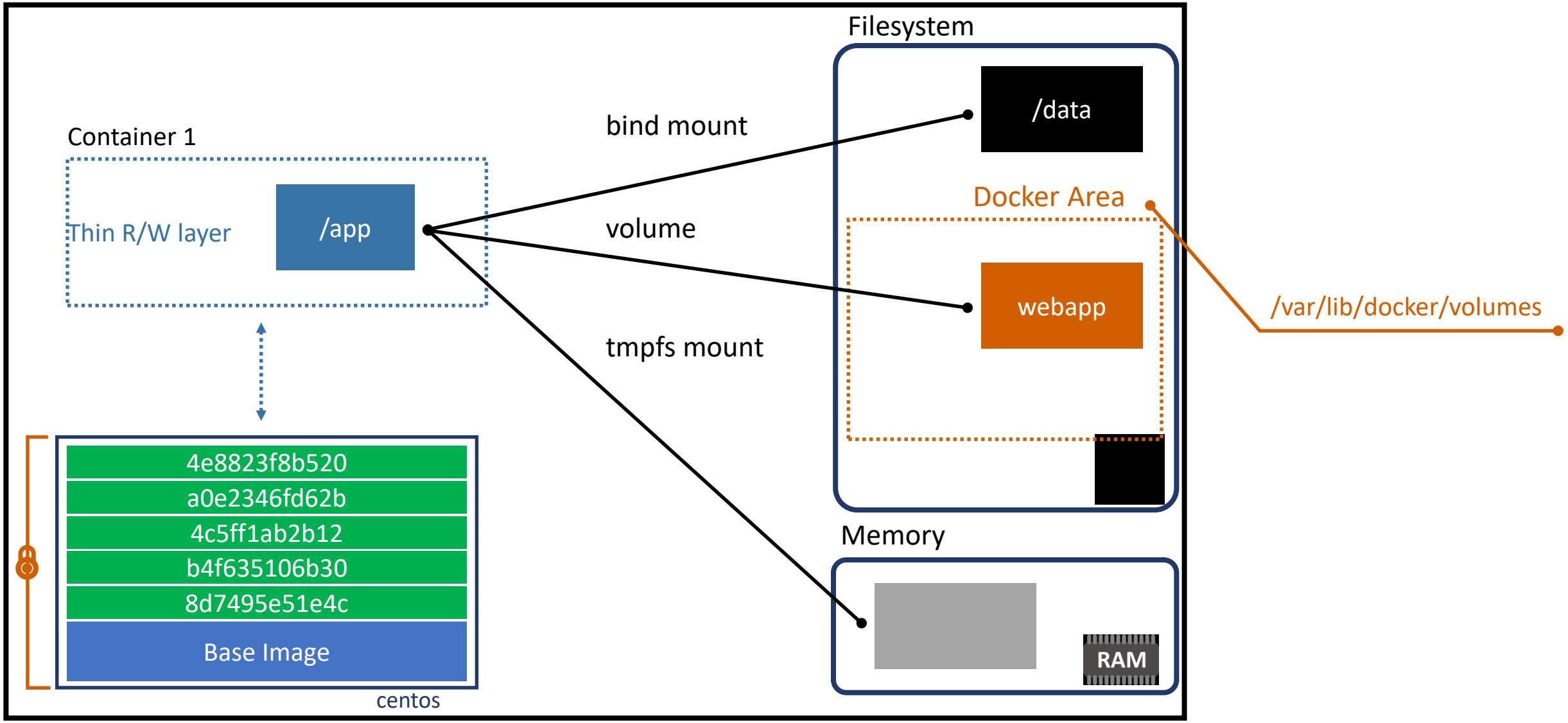

tmpfs mounts **are stored in the host system's memory** only, and are never written to the host system's filesystem.

# Choose the right type of mount :: tmpfs mounts

```
[root@docker ~]# docker run -it --rm --tmpfs /memdrive centos /bin/bash


[root@ae75d4f50280 /]# dd if=/dev/zero of=/memdrive/mem.txt bs=5k count=200
200+0 records in
200+0 records out
1024000 bytes (1.0 MB) copied, 0.00126122 s, 812 MB/s
[root@ae75d4f50280 /]#
```

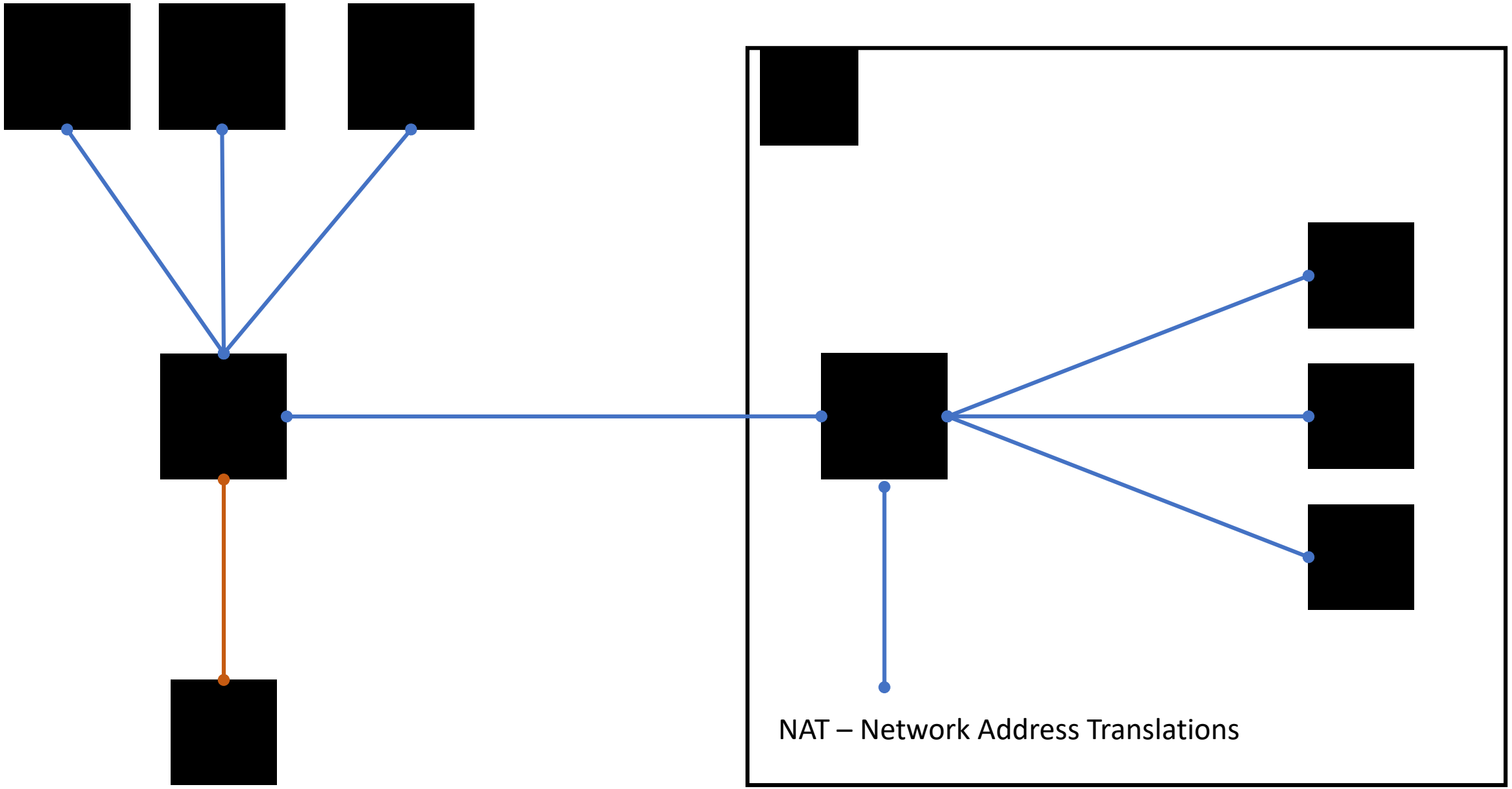
```
[root@docker ~]# docker run -it --rm --mount type=tmpfs,destination=/memdrive centos
/bin/bash

[root@e11383050105 /]#  dd if=/dev/zero of=/memdrive/mem.txt bs=5k count=200
200+0 records in
200+0 records out
1024000 bytes (1.0 MB) copied, 0.00105322 s, 972 MB/s
[root@e11383050105 /]#
```

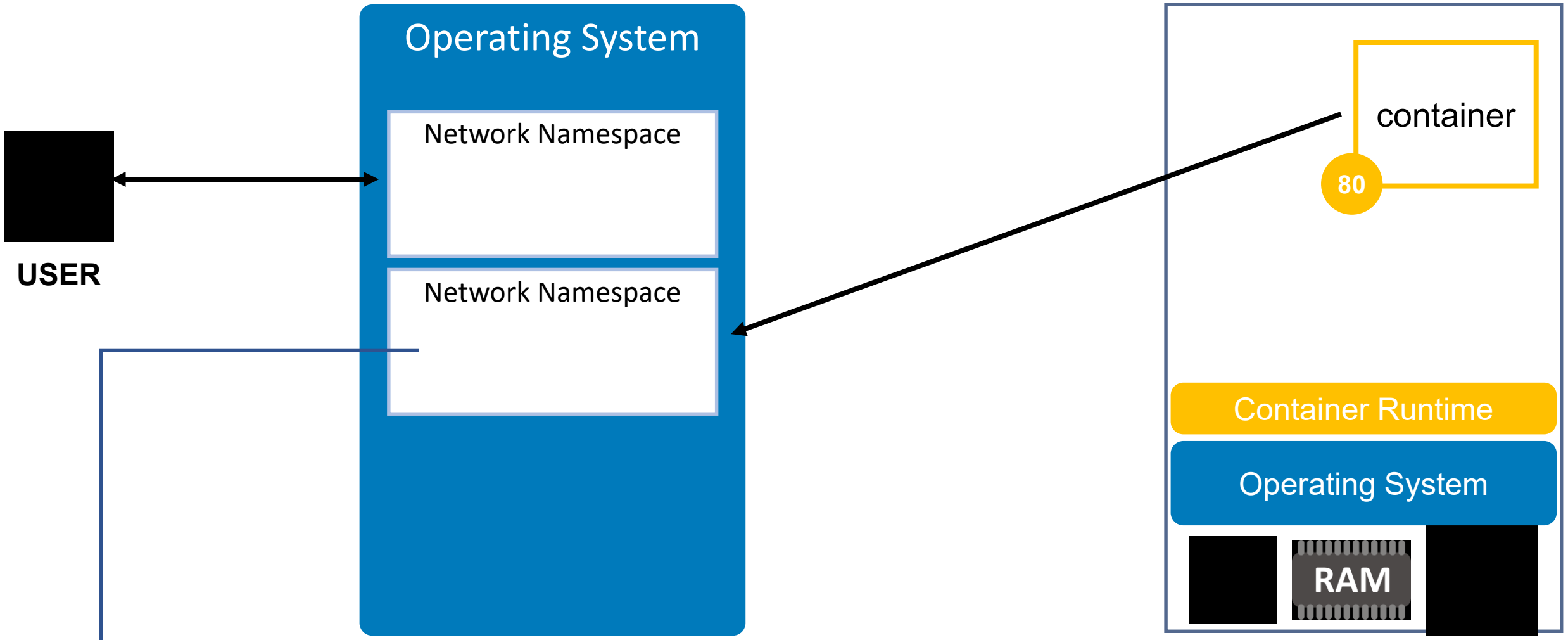
# Docker Host


Filesystem
/data
Container 1
bind mount
Thin R/W layer
/app
volume
Docker Area
webapp
/var/lib/docker/volumes
tmpfs mount
4e8823f8b520
a0e2346fd62b
4c5ff1ab2b12
b4f635106b30
8d7495e51e4c
Base Image
centos
Memory
RAM

# Docker Networking

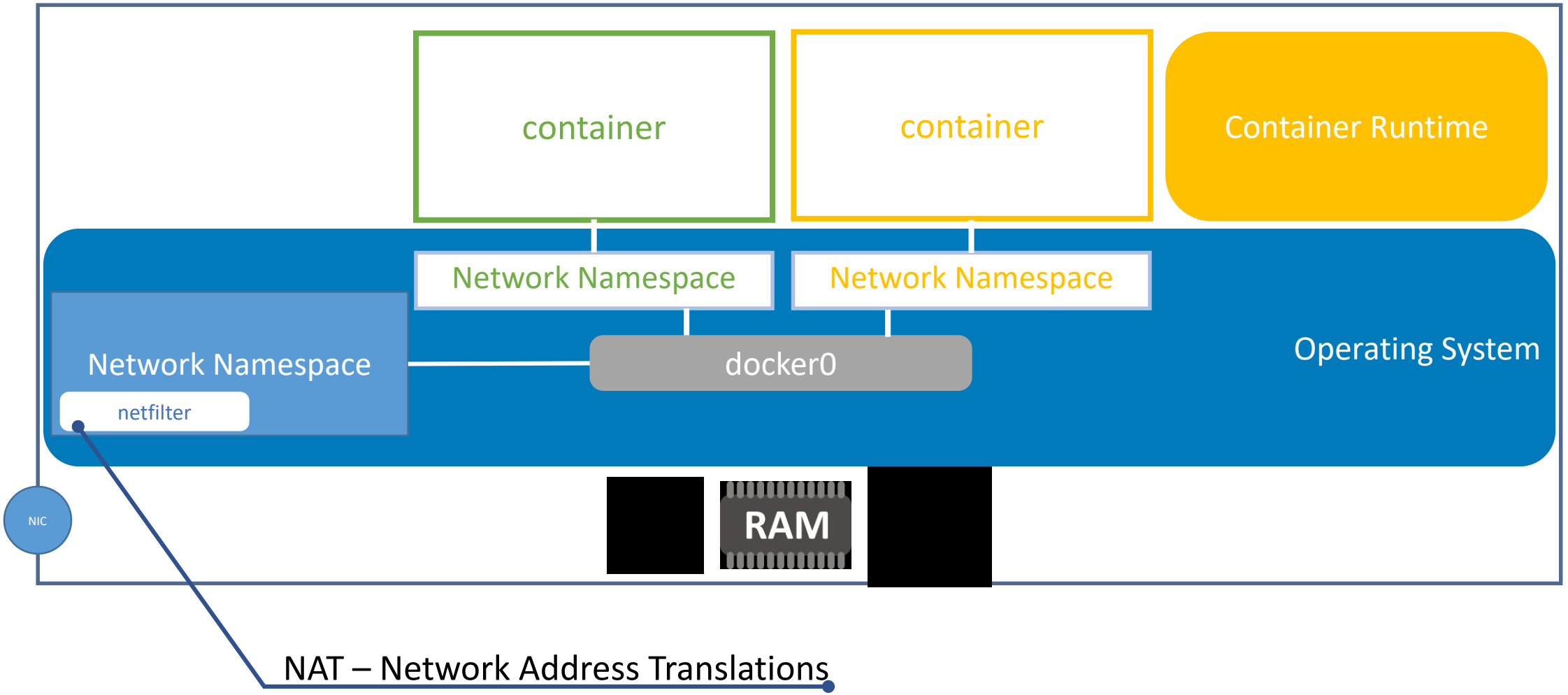
Think about Home Internet



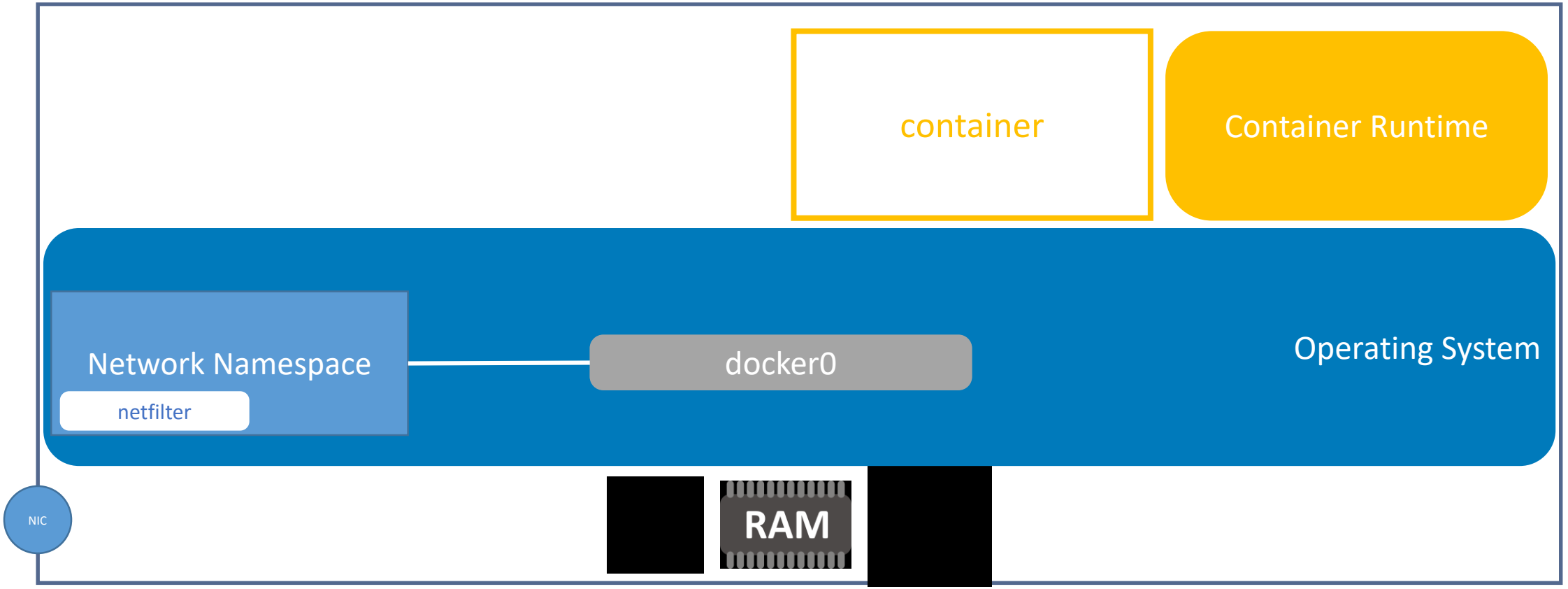



- Network namespaces virtualize the network stack. On creation a network namespace contains only a loopback interface.
- Each network interface (physical or virtual) is present in exactly 1 namespace and can be moved between namespaces.
- Each namespace will have a private set of IP addresses, its own routing table, socket listing, connection tracking table, firewall, and other network-related resources.

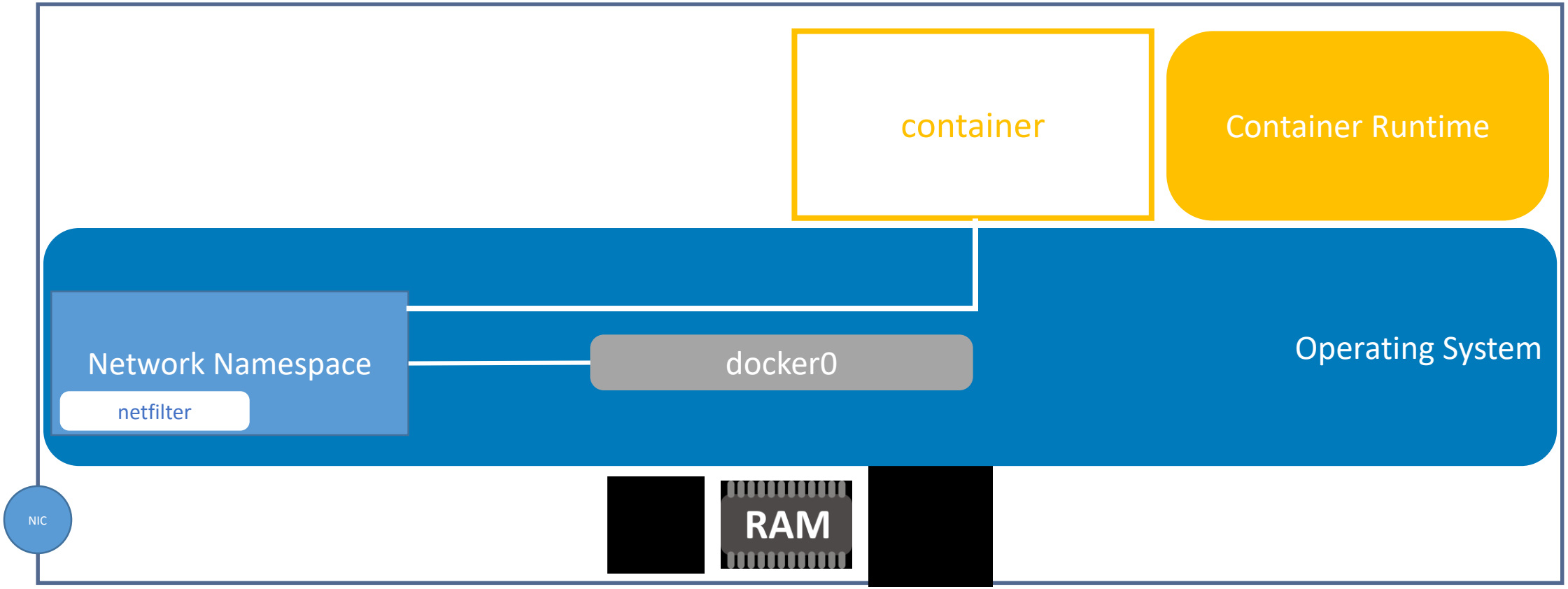
# Network in Docker (bridge – default)



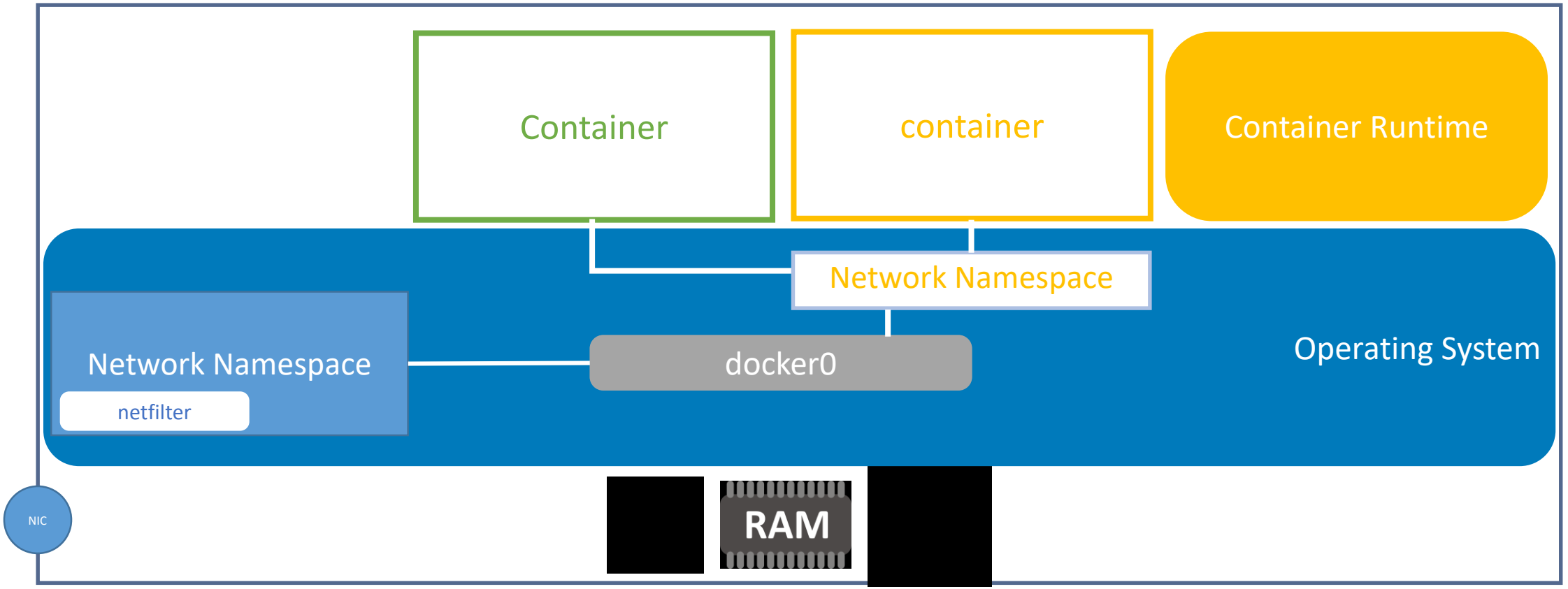
# Network in Docker (none)



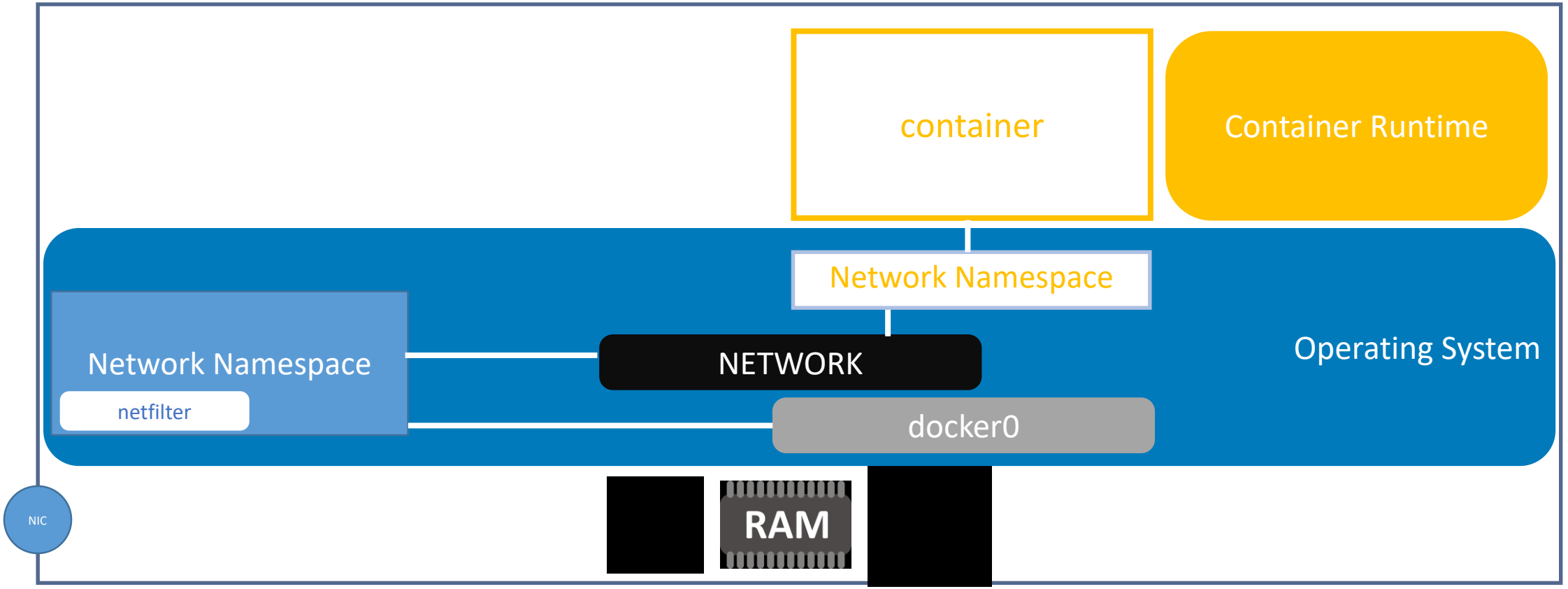
# Network in Docker (host)



# Network in Docker (container:<name|id>)

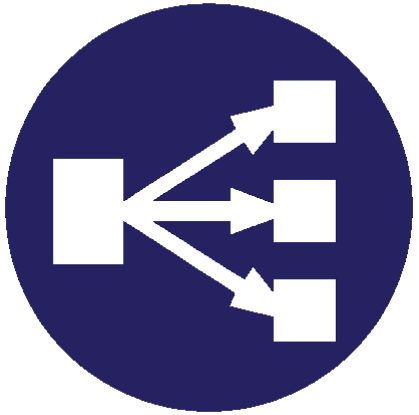


# Network in Docker (NETWORK)



# Supported Network in Docker

| Network | Description |
| --- | --- |
| **none** | No networking in the container. |
| **bridge** (default) | Connect the container to the bridge via veth interfaces. |
| **host** | Use the host's network stack inside the container. |
| **container**:<name\|id> | Use the network stack of another container, specified via its name or id. |
| **NETWORK** | Connects the container to a user created network (using docker network create command) |



# Network settings

- All containers have networking enabled and they can make **any outgoing connections**.
- The operator can completely **disable networking with** *docker run --network none* which disables all incoming and outgoing networking.
- **Publishing ports and linking to other containers only works with the default (bridge)**. The linking feature is a legacy feature. You should always prefer using Docker network drivers over linking.
- Your container will use the same DNS servers as the host by default, but you can **override this with --dns**
- The MAC address is generated using the IP address allocated to the container. You can set the container's MAC address explicitly by **providing a MAC address via the --mac-address** parameter

```
--dns=[]           : Set custom dns servers for the container
--network="bridge" : Connect a container to a network
                      'bridge': create a network stack on the default Docker bridge
                      'none': no networking
                      'container:<name|id>': reuse another container's network stack
                      'host': use the Docker host network stack
                      '<network-name>|<network-id>': connect to a user-defined network
--add-host=""      : Add a line to /etc/hosts (host:IP)
--mac-address=""   : Sets the container's Ethernet device's MAC address
--ip=""            : Sets the container's Ethernet device's IPv4 address
--ip6=""           : Sets the container's Ethernet device's IPv6 address
```

# Network settings

```
[root@docker ~]# docker run --rm -d --network host --name my_nginx nginx
53bccbfaadd879892237993ff60585dae4208487a4699bd1c46f6bed87322f39
[root@docker ~]#
[root@docker ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
53bccbfaadd8        nginx               "nginx -g 'daemon of…"   6 seconds ago       Up 6 seconds                            my_nginx
[root@docker ~]#
[root@docker ~]# docker inspect -f "{{ .NetworkSettings.Networks.host.IPAddress }}" 53bccbfaadd8

[root@docker ~]# curl http://10.211.55.38
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[root@docker ~]#
```

IP Address is not assigned to the container.

Browse to http service at Docker host

```
# docker run --rm -d --network host --name my_nginx nginx
```

# Published ports

By default, when you create a container, it does not publish any of its ports to the outside world. To make a port available to services outside of Docker, or to Docker containers which are not connected to the container's network, use the --publish or -p flag. This creates a firewall rule which maps a container port to a port on the Docker host.

| Flag value | Description |
| --- | --- |
| -p 8080:80 | Map TCP port 80 in the container to port 8080 on the Docker host. |
| -p 192.168.1.100:8080:80 | Map TCP port 80 in the container to port 8080 on the Docker host for connections to host IP 192.168.1.100. |
| -p 8080:80/udp | Map UDP port 80 in the container to port 8080 on the Docker host. |
| -p 8080:80/tcp -p 8080:80/udp | Map TCP port 80 in the container to TCP port 8080 on the Docker host, and map UDP port 80 in the container to UDP port 8080 on the Docker host. |

# Docker Compose

# Docker Compose

- Compose is a tool for defining and running multi-container Docker applications.
- With Compose, you use a YAML file to configure your application's services.
- Then, with a single command, you create and start all the services from your configuration.

# Compose v2

- Compose V2 integrates compose functions into the Docker platform, continuing to support most of the previous docker-compose
- Using “docker compose” instead of “docker-compose”

# Use case: Development environments

- Ability to run an application in an isolated environment and interact with it is crucial
- Document and configure all of the application's service dependencies (databases, queues, caches, web service APIs, etc)
- Convenient way to get started on a project, recude a multi-page "developer getting started guide"

# Use case: Automated Testing Environments

- Automated test suite for Continuous Deployment and Continuous Integration

# Use case: Production – Single host deployment

- Compose has traditionally been focused on development and testing workflows, but with each release we're making progress on more production-oriented features.

# Use case: Production – Single host deployment

- **Removing any volume bindings** for application code, so that code stays inside the container and can't be changed from outside
- Binding to **different ports** on the host
- Setting **environment variables** differently, such as reducing the verbosity of logging, or to specify settings for external services such as an email server
- Specifying a **restart policy** like restart: always to avoid downtime
- Adding **extra services** such as a log aggregator

# Use case: Production – Single host deployment

consider defining an additional Compose file  to apply along with the original Compose file

```
docker compose -f docker-compose.yml -f production.yml up -d
```

# Using compose

- Define your app's environment with a **Dockerfile** so it can be reproduced anywhere.

- Define the services that make up your app in **docker-compose.yml** so they can be run together in an isolated environment.

- Run **docker compose up** and the Docker compose command starts and runs your entire app

# Compose application model

- The Compose specification allows one to define a platform-agnostic container based application. Such an application is designed as a set of containers which have to both run together with adequate shared resources and communication channels.

# Compose application model

- **Service:** an abstract concept implemented on platforms by running the same container image (and configuration) one or more times.

- **Network:** a platform capability abstraction to establish an IP route between containers within services connected together
  **Services communicate with each other through Networks**

- **Volume:** describes such a persistent data as a high-level filesystem mount with global options
  **Services store and share persistent data into Volumes**

## Special thanks

to p'Jub Damrongsak for inspirations, materials and references in this Class

# ThankYou


Sukkarin Ruensukont

sukkarin@mfec.co.th

นั่งเล่น NGINX