

docker

สถาบัน ไอเอ็มซี

บริษัท สยามชานาญกิจ จำกัด และเพื่อนพ้องน้องพี่





Somkiat Puisungnoen

Somkiat Puisungnoen

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Intro

Software Craftsmanship

Software Practitioner at สยามช่างนาญกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Somkiat Puisungnoen 15 mins · Bangkok · ⚙️

Java and Bigdata





Page

Messages

Notifications 3

Insights

Publishing Tools

Settings

Help ▾



somkiat.cc

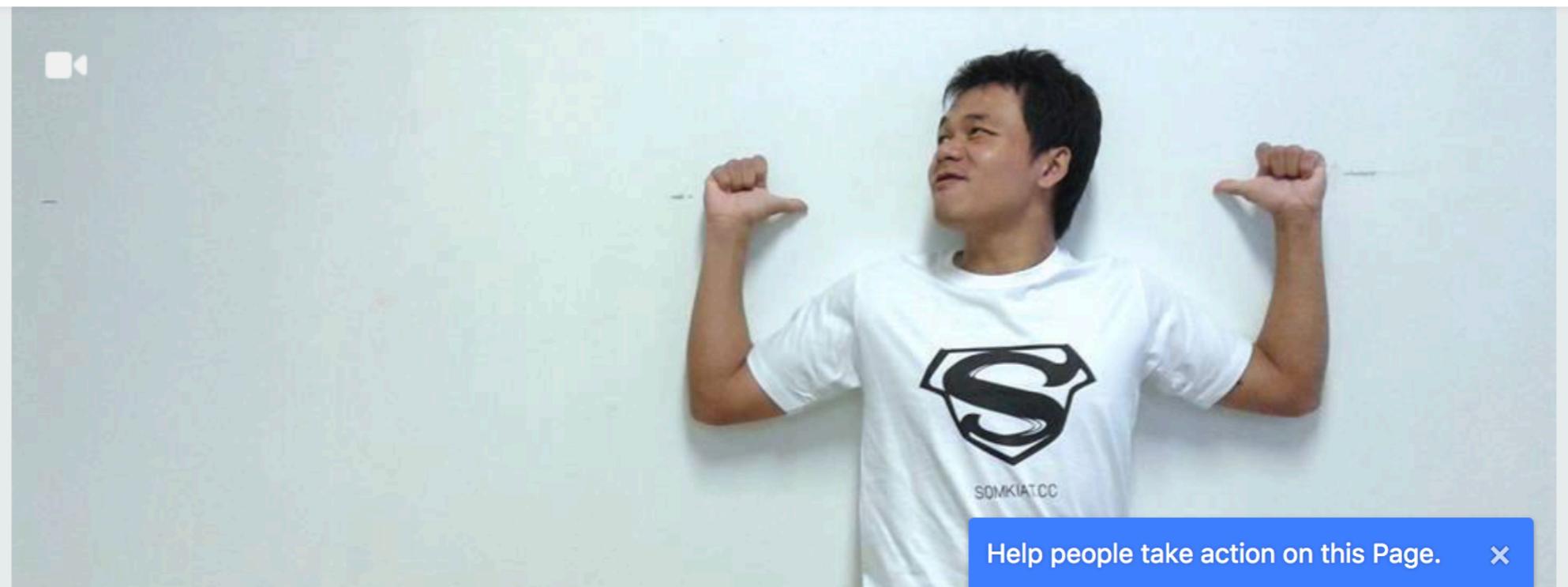
@somkiat.cc

Home

Posts

Videos

Photos



Agenda

Introduction

Basic of Docker

Building containers

Running web apps with Docker

Docker automation



Introduction

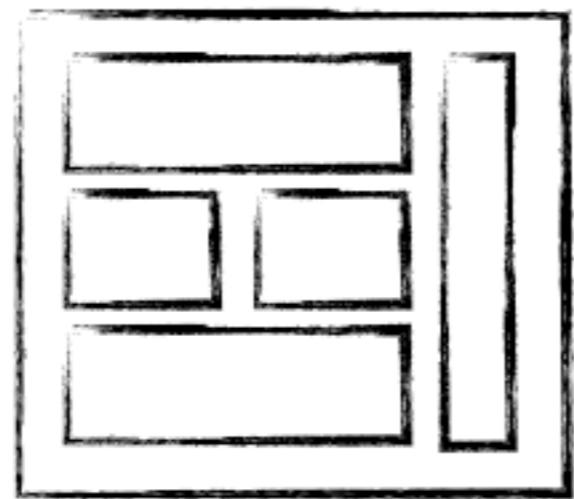


Why docker ?

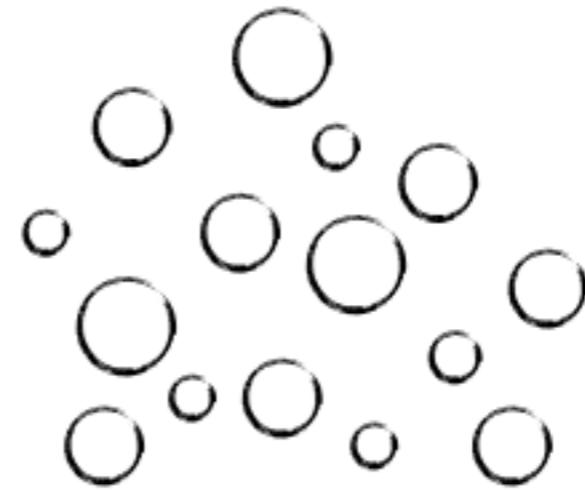


Why docker ?

Software industry has changed !!



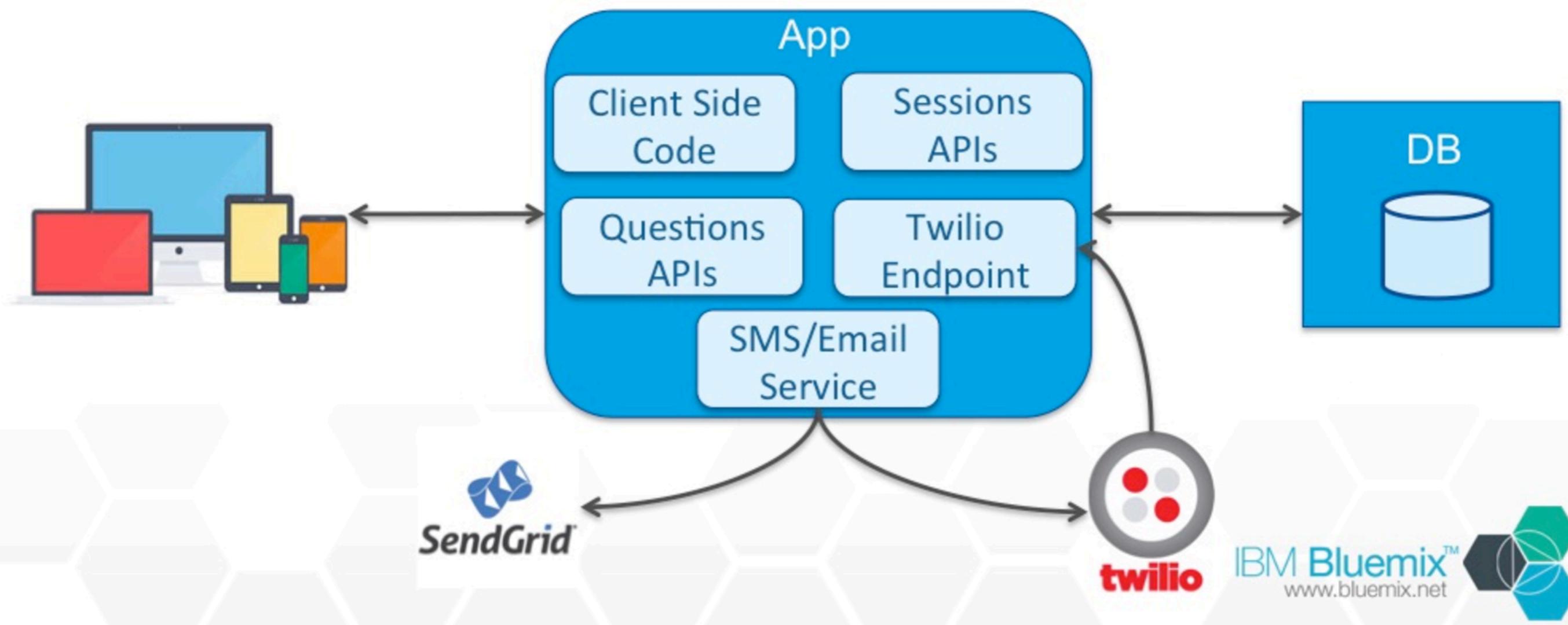
MONOLITHIC/LAYERED



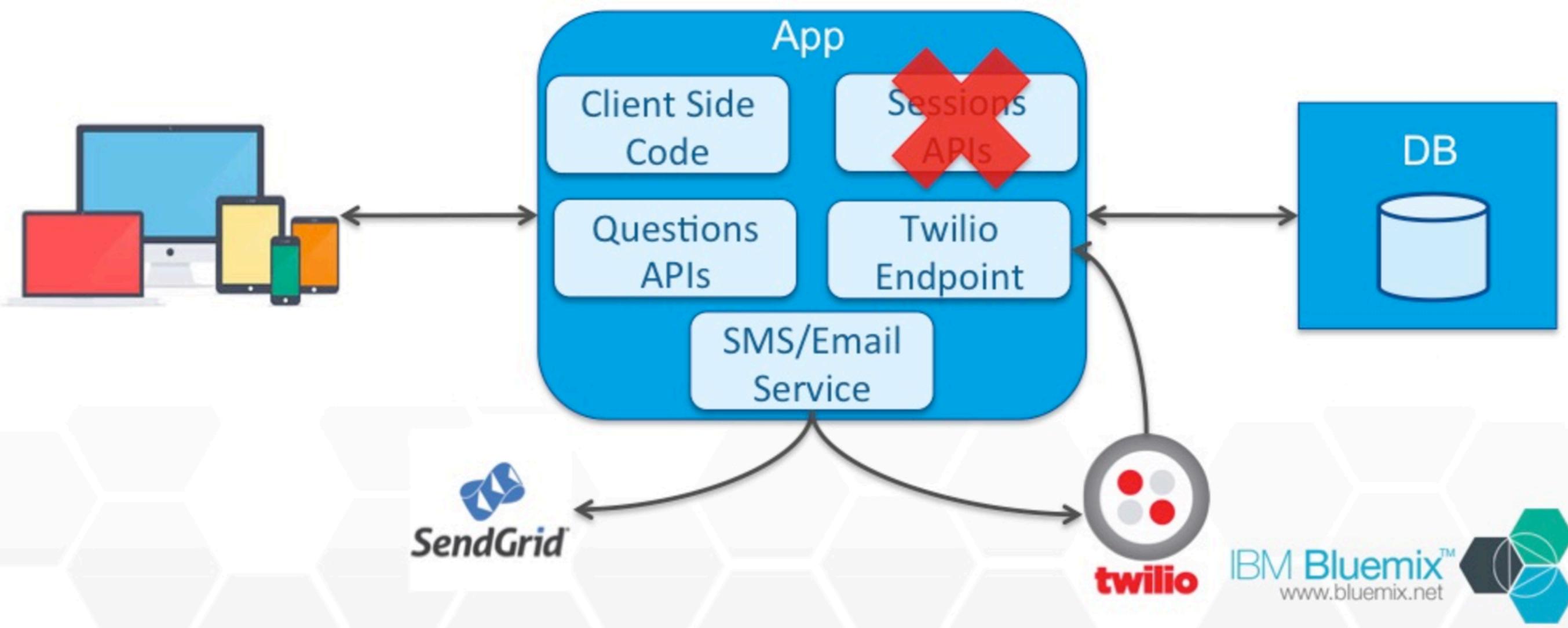
MICRO SERVICES



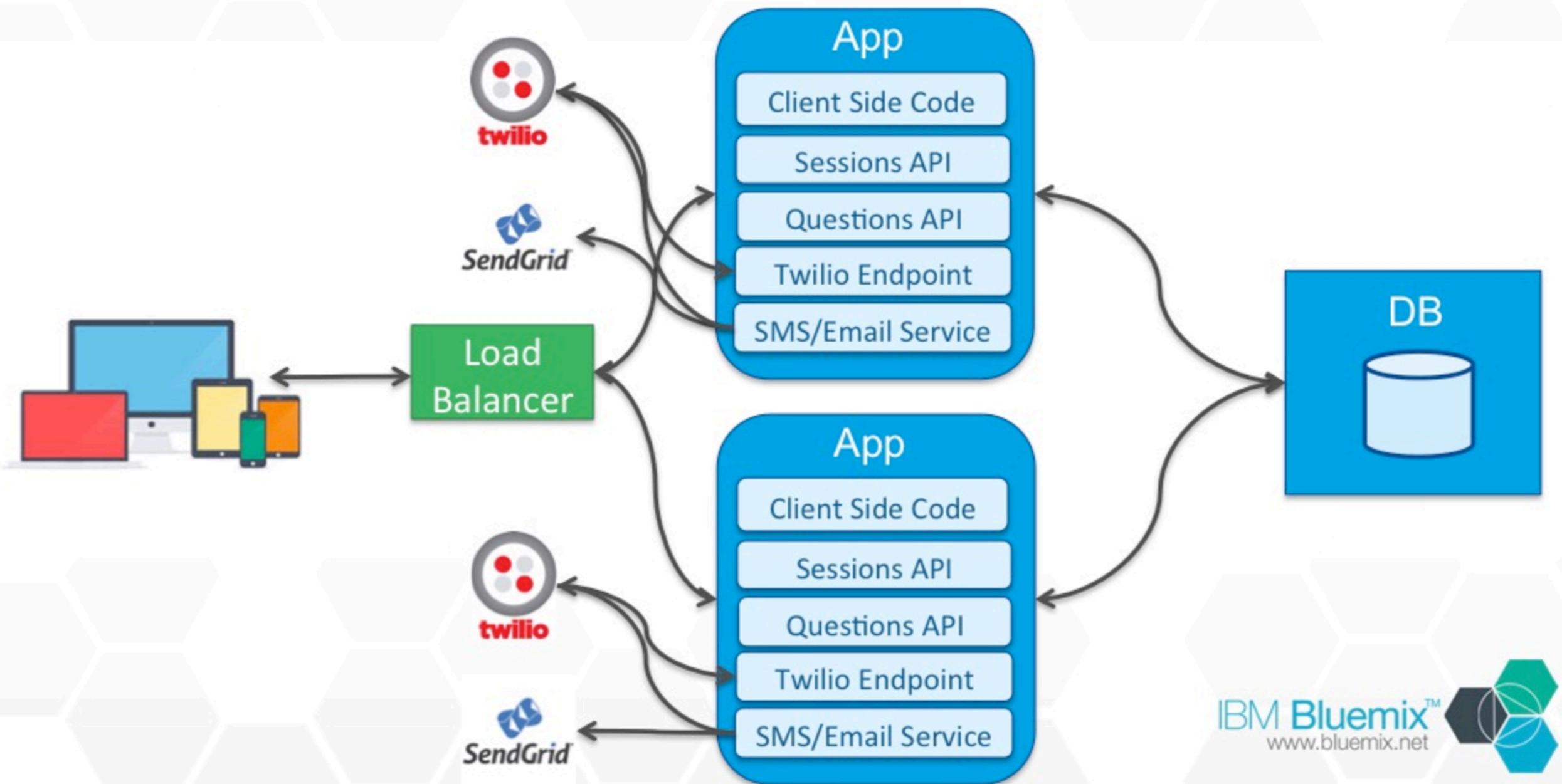
Monolith architecture



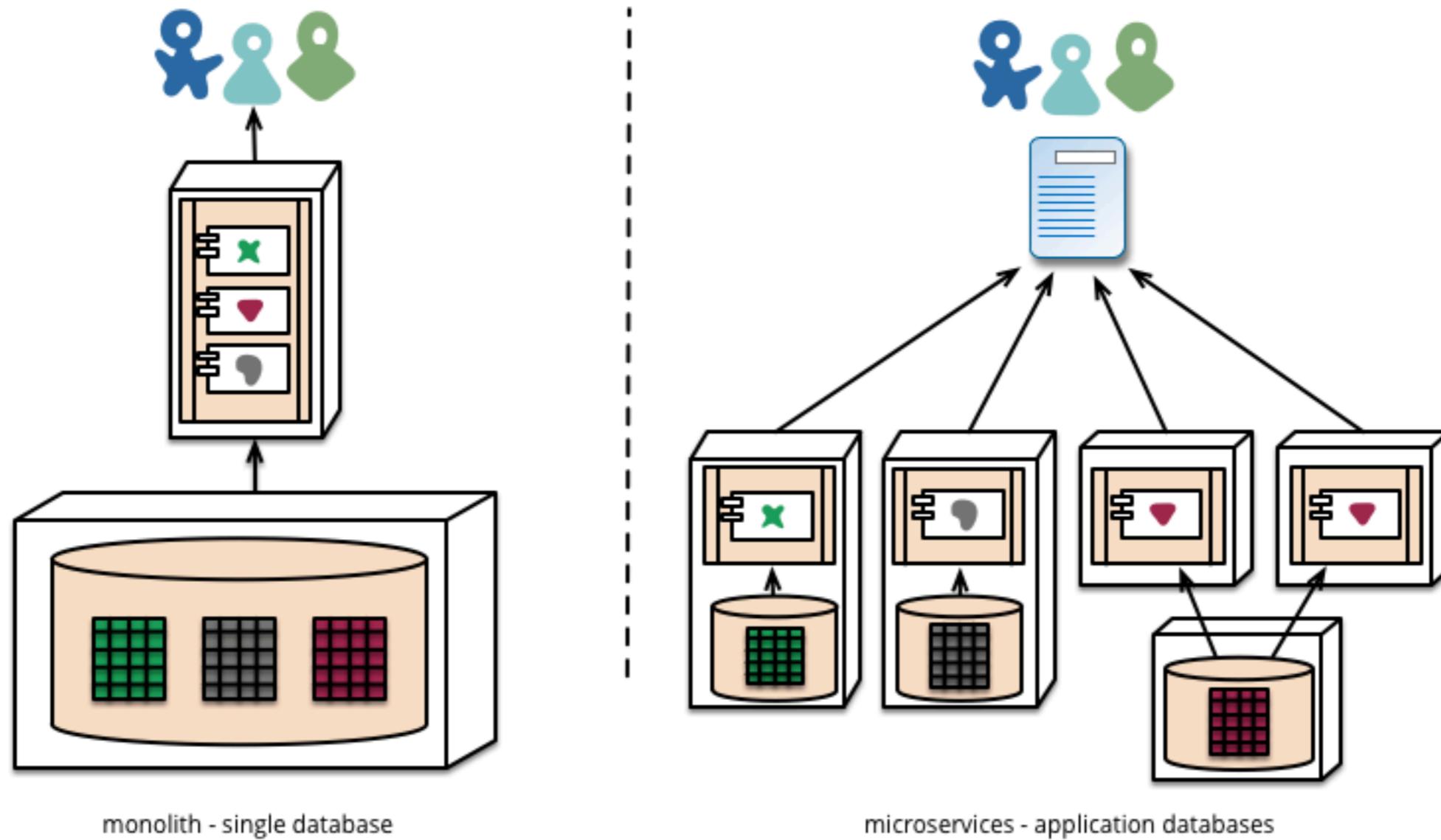
Failure in Monolith !!



Scale Monolith !!



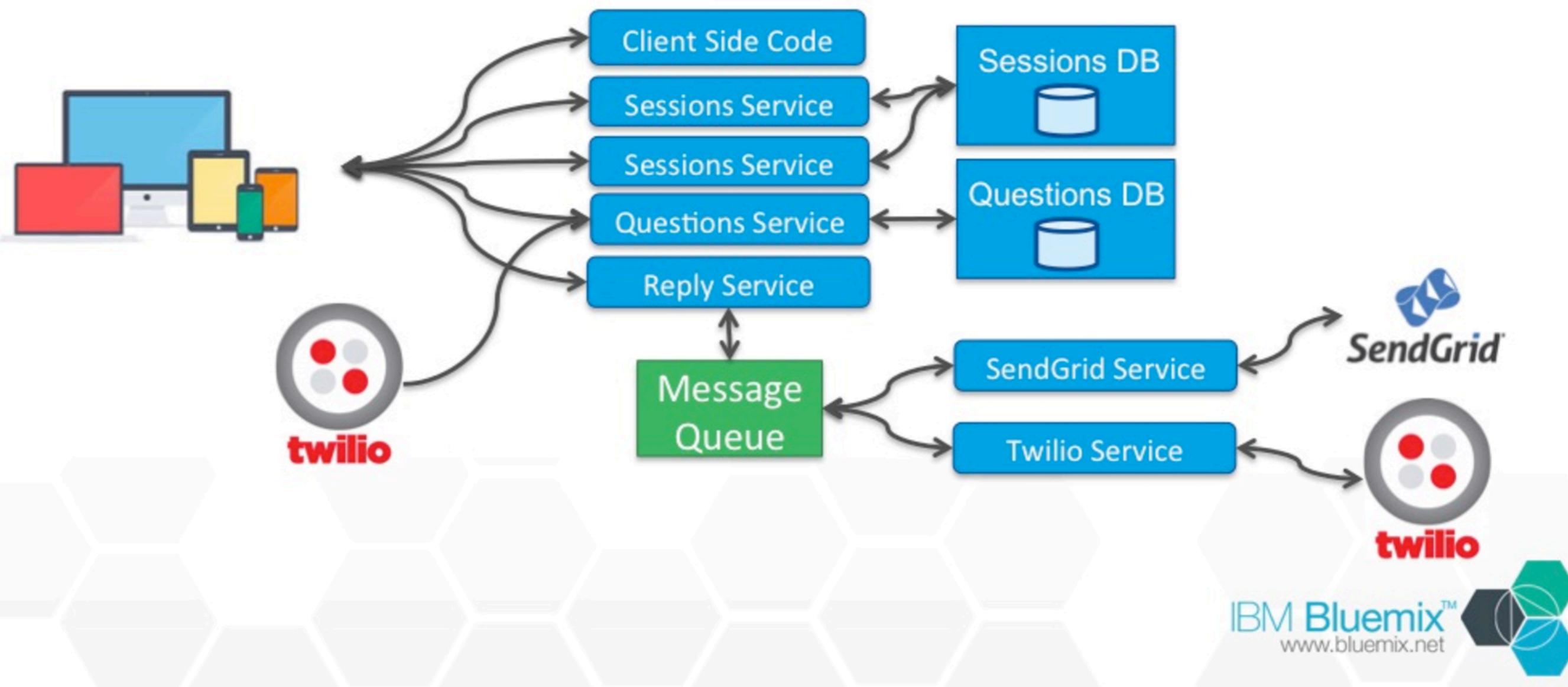
Microservice architecture



<https://martinfowler.com/articles/microservices.html>



Microservice architecture

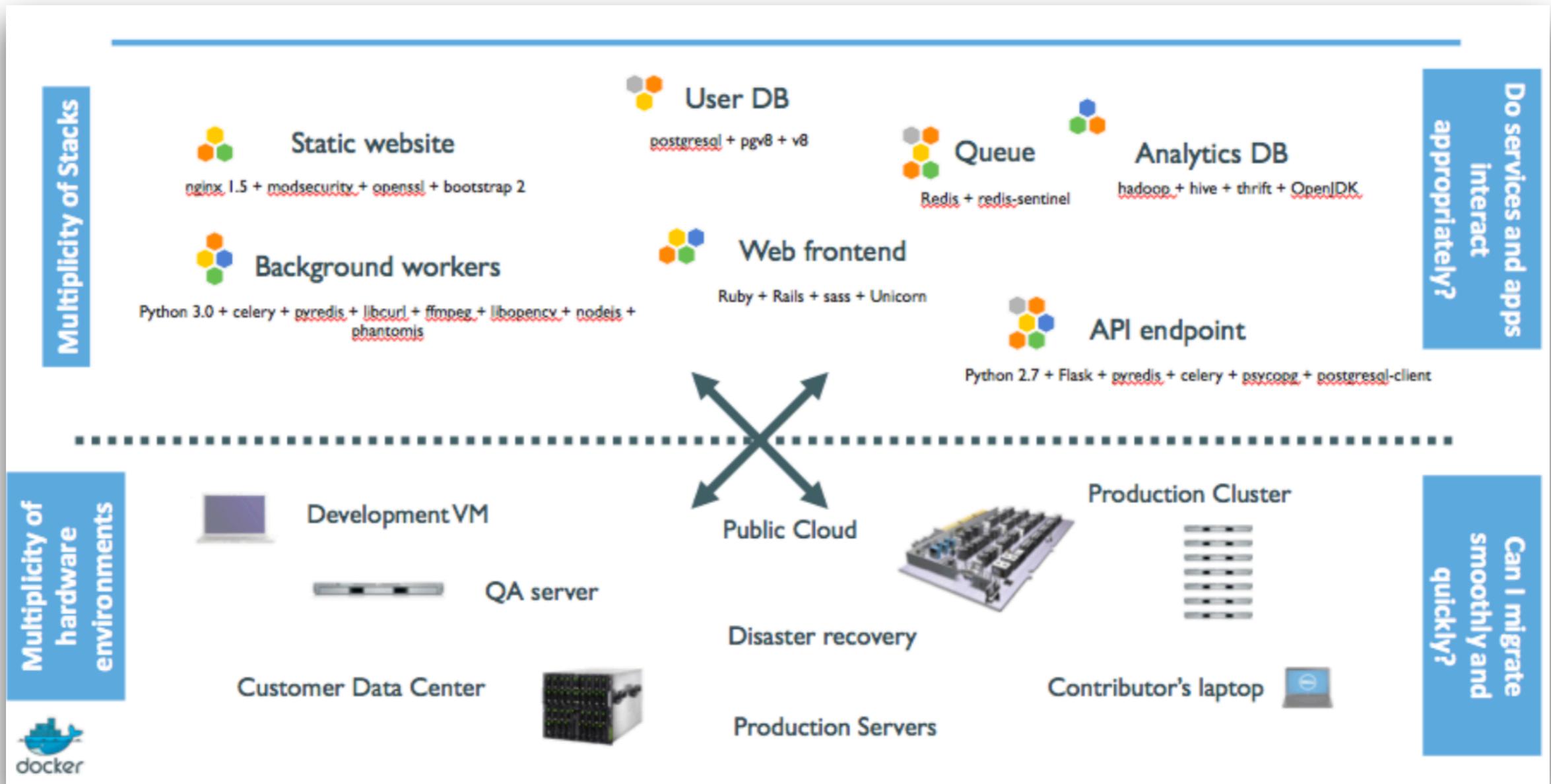


Why docker ?

We have problem !!



Problem

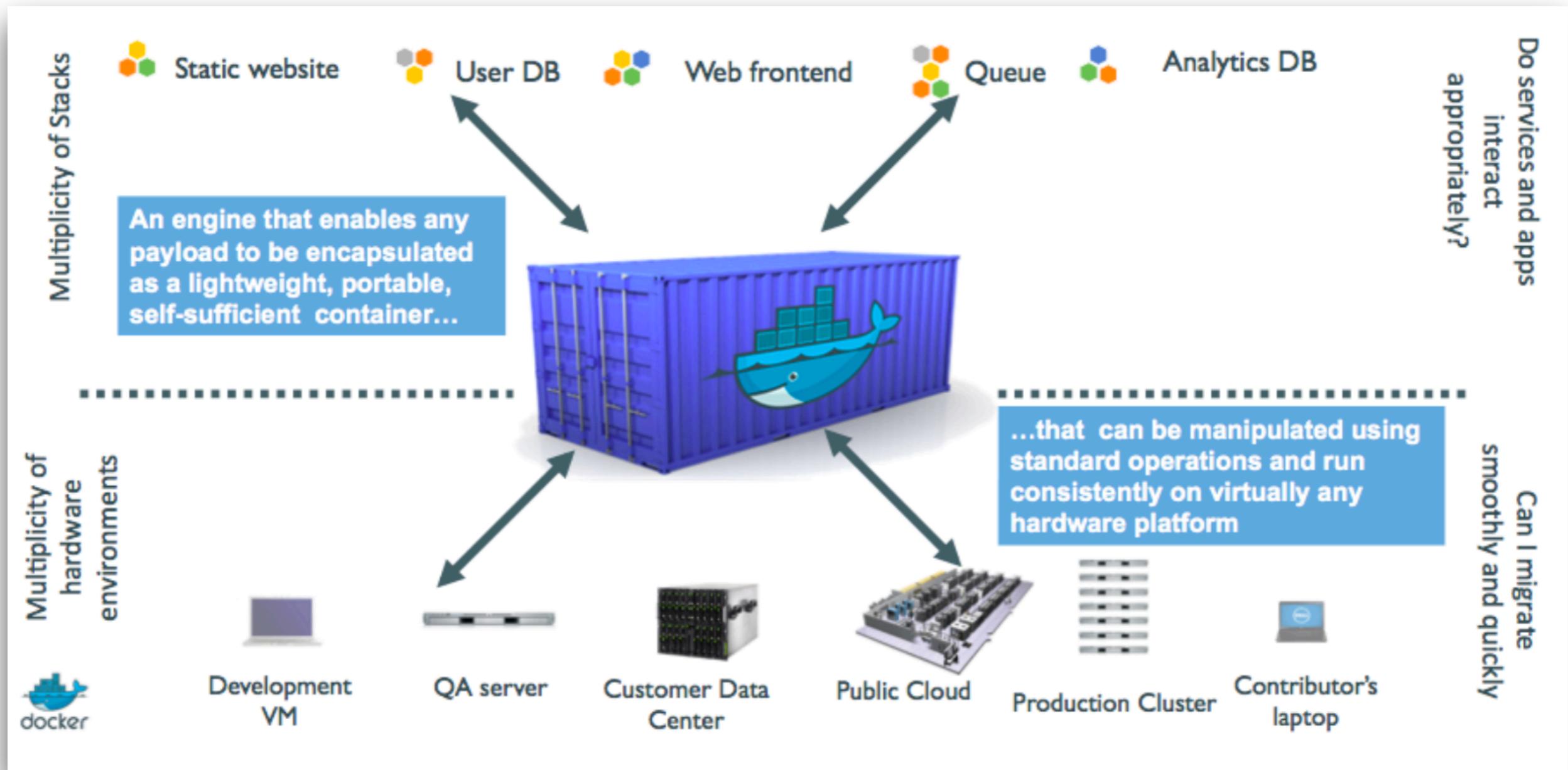


Problem

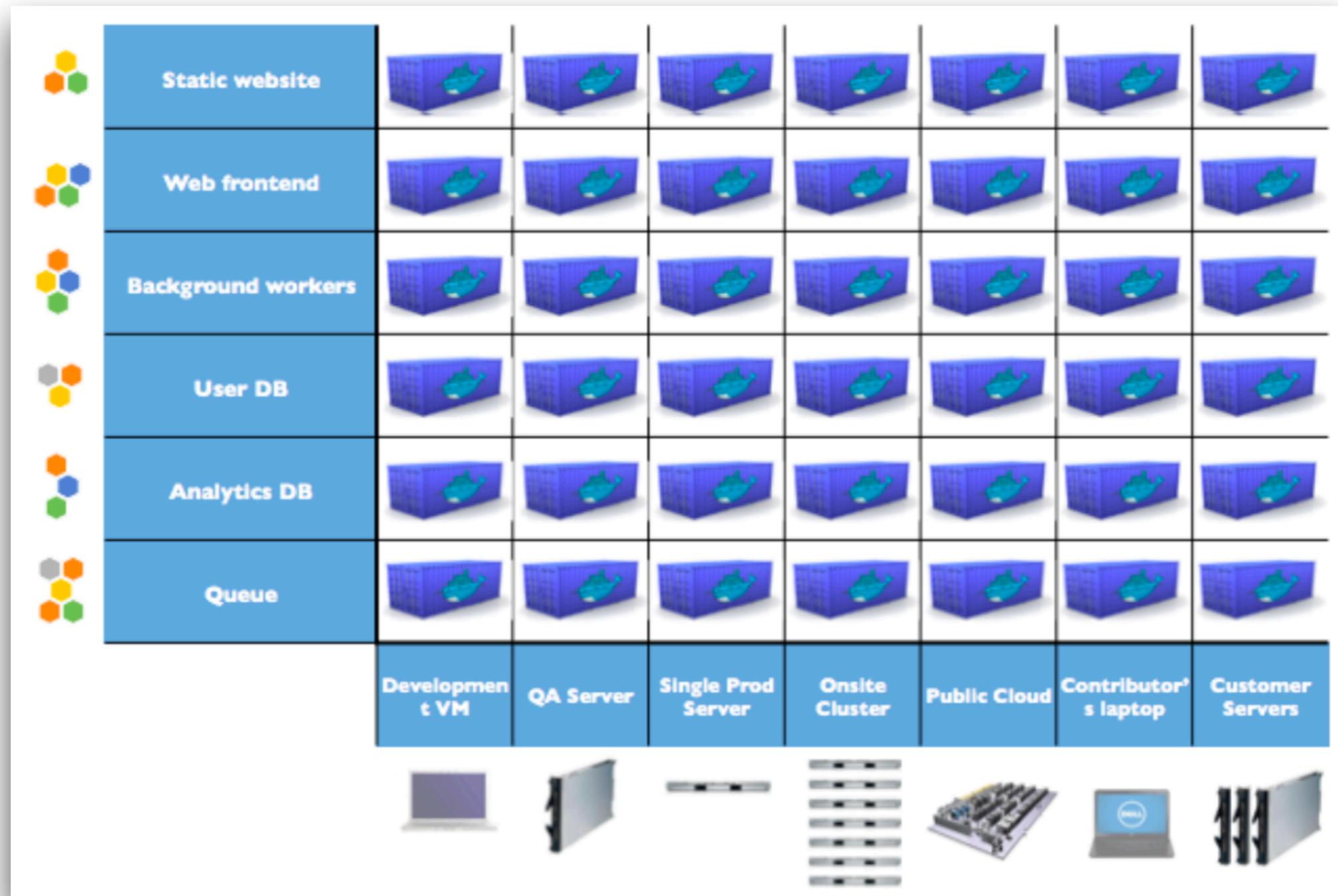
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers



Solution



Solution

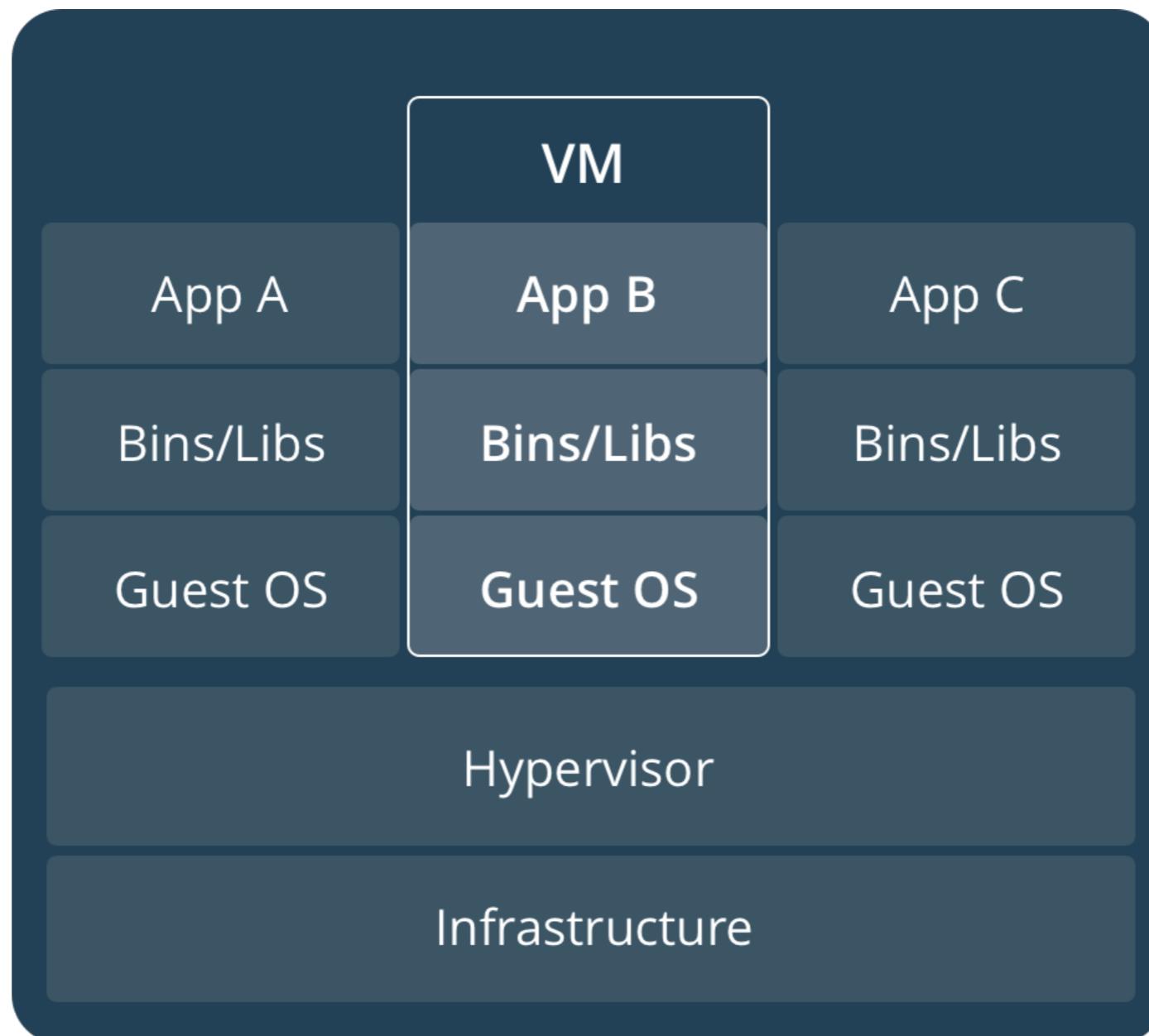


Container vs. Virtual machine

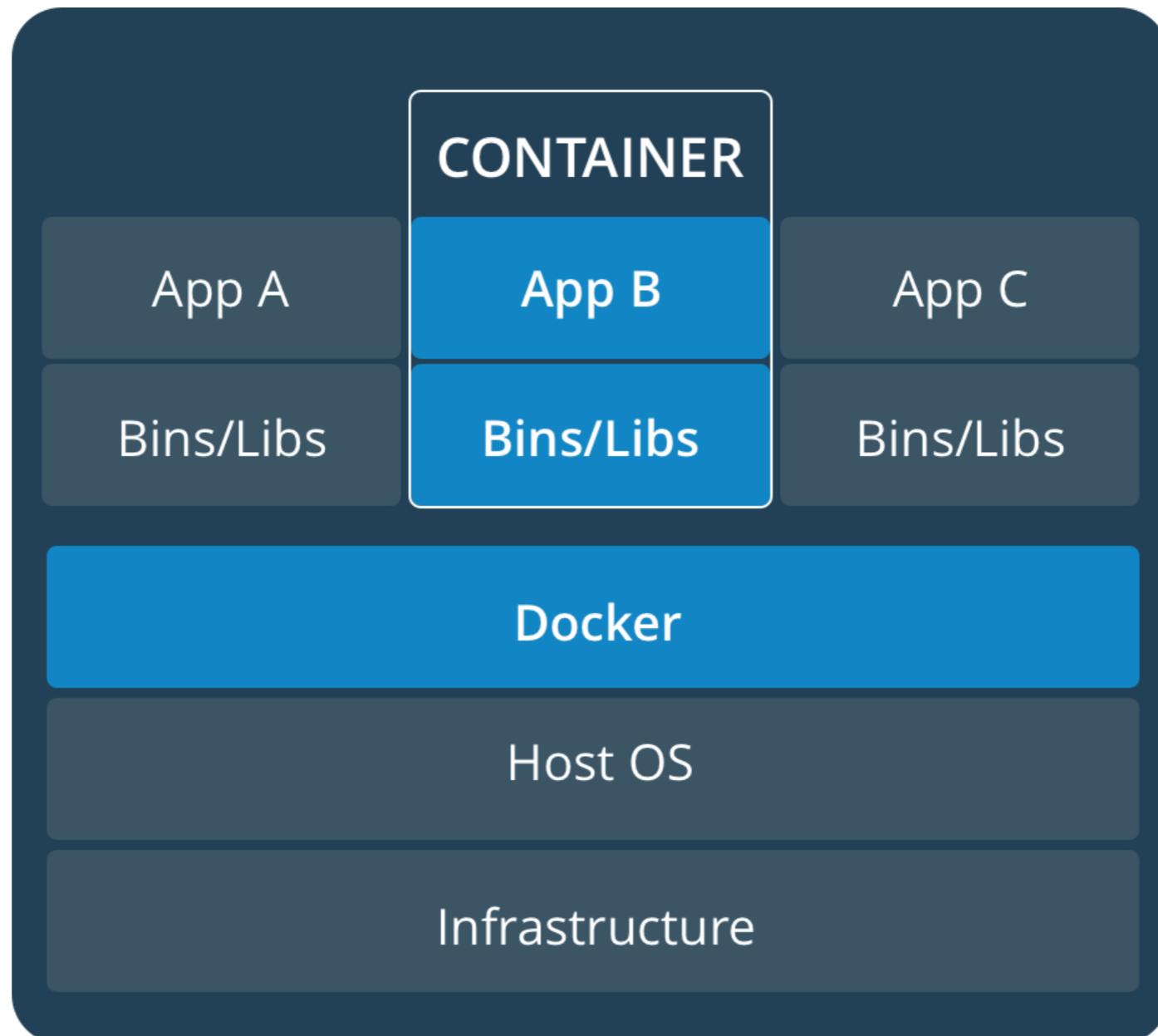
<https://www.docker.com/what-container>



Virtual Machine



Container



Virtual Machine

Abstraction of physical hardware

Full copy of OS and libraries

Slow to boot



Container

Abstraction at the app level

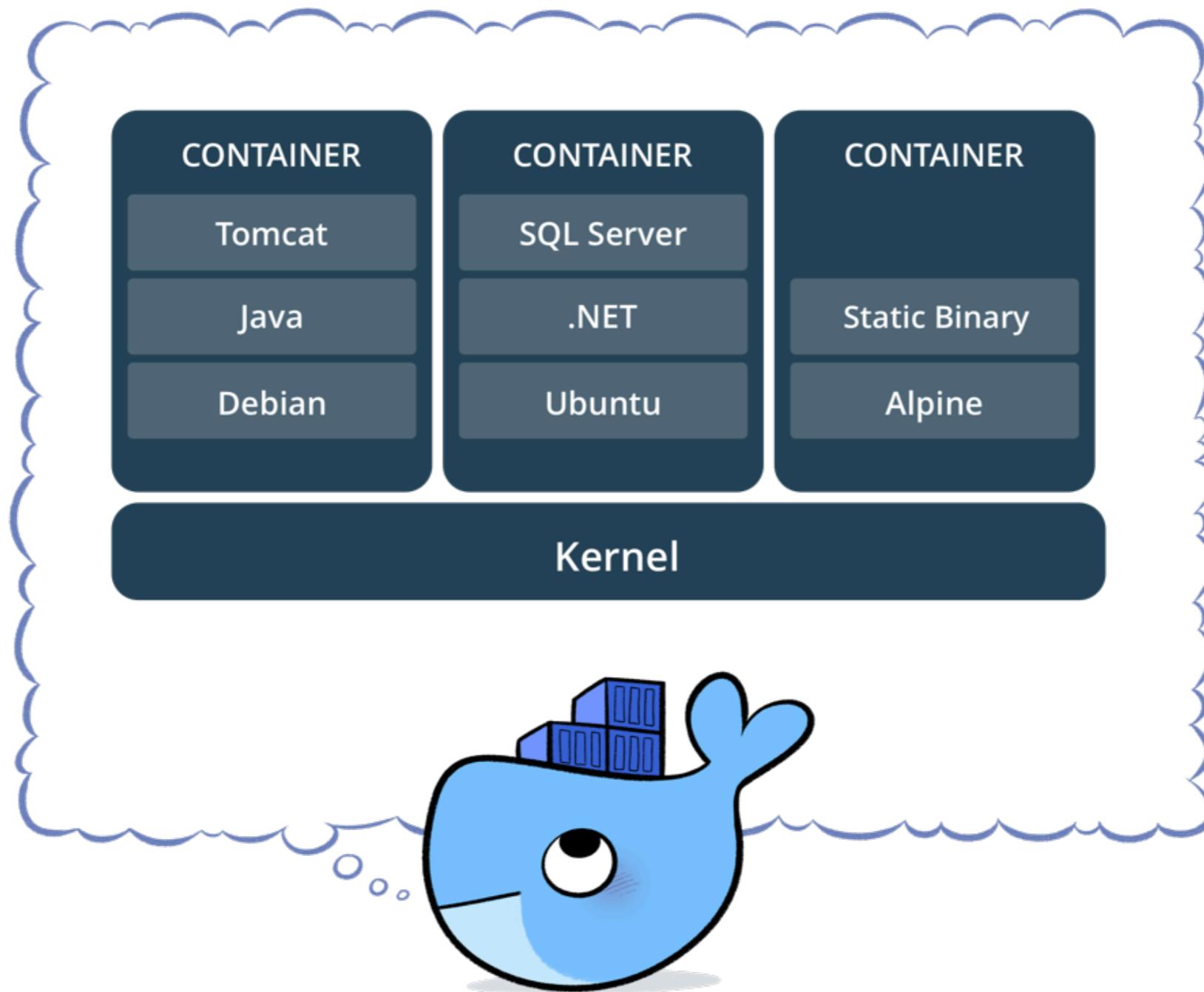
Share OS kernel

Isolate process

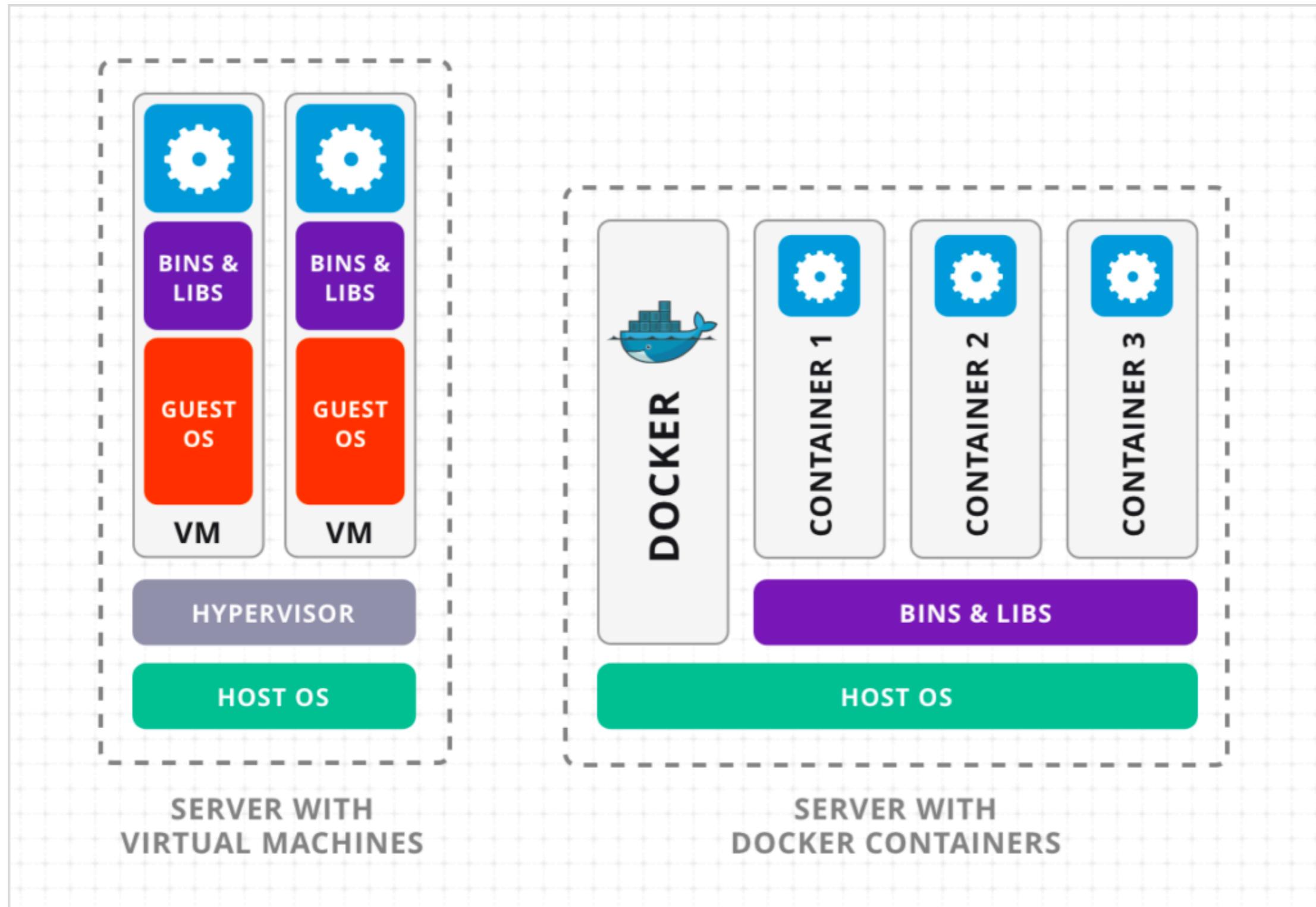
Less space than Virtual Machine



Docker



Container vs Virtual machine



Container vs Virtual machine

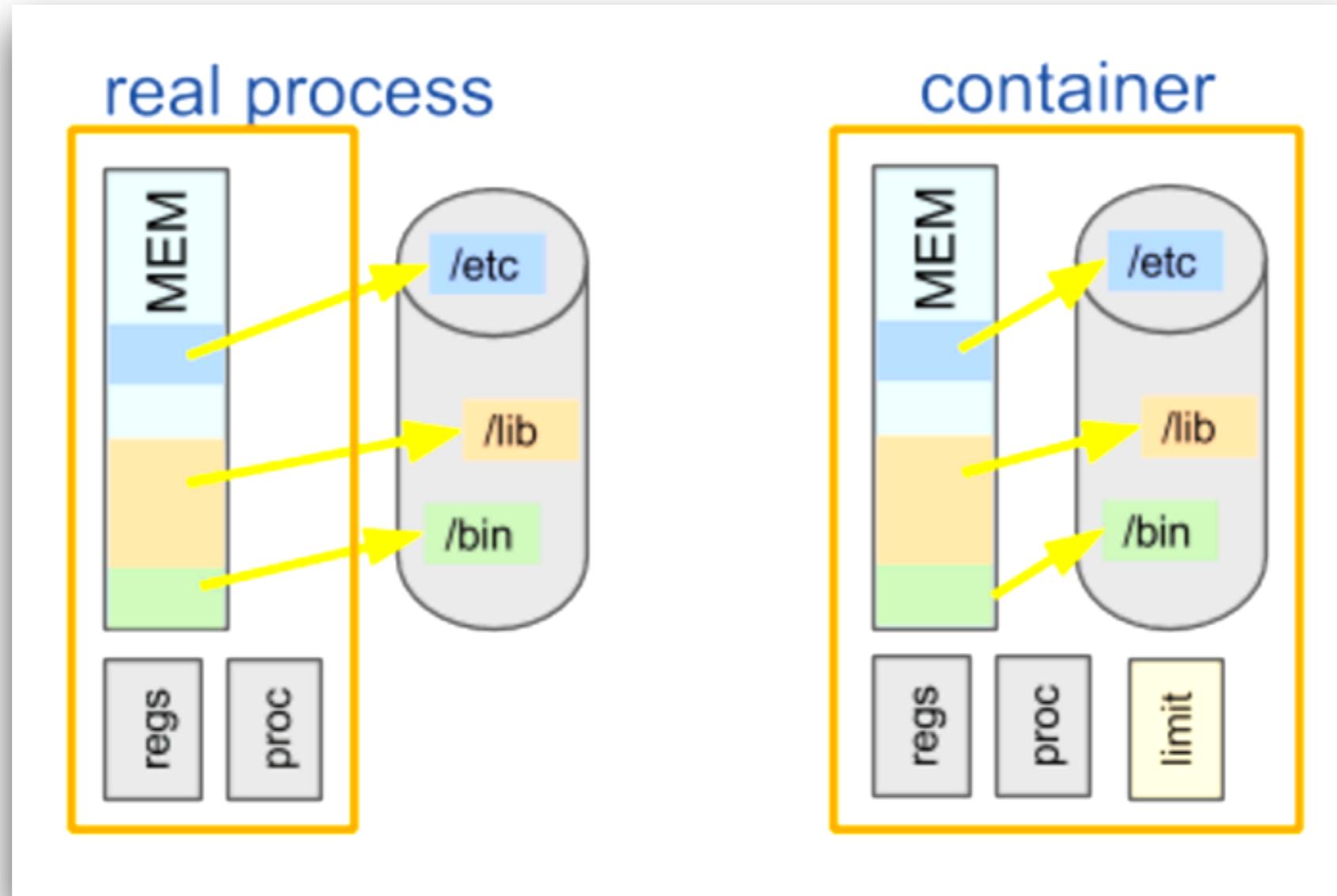
	VM	Docker
การใช้งาน resources	มาก	น้อย
ความเร็วของการ start/ boot	ช้า	เร็ว
ความเร็วในการทำงาน	ช้า	เร็ว
พื้นที่จัดเก็บ	มาก	น้อย



Container vs. Process



Container vs. Process



First container with Docker



Verify

\$docker version

```
Client: Docker Engine - Community
Version:           19.03.8
API version:      1.40
Go version:       go1.12.17
Git commit:       afacb8b
Built:            Wed Mar 11 01:21:11 2020
OS/Arch:          darwin/amd64
Experimental:    false

Server: Docker Engine - Community
Engine:
  Version:          19.03.8
  API version:     1.40 (minimum version 1.12)
  Go version:      go1.12.17
  Git commit:      afacb8b
  Built:           Wed Mar 11 01:29:16 2020
  OS/Arch:         linux/amd64
  Experimental:   false
containerd:
  Version:          v1.2.13
  GitCommit:        7ad184331fa3e55e52b890ea95e65ba581ae3429
runc:
  Version:          1.0.0-rc10
  GitCommit:        dc9208a3303feef5b3839f4323d9beb36df0a9dd
docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```



Version format of Docker

YY.MM.<no. updated>

Stable and Edge release

19.03.8

Year=2019

Month=March

Update Bug/security fix=8



Docker more information

\$docker info



Hello docker

\$ docker container run hello-world

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
78445dd45222: Pull complete
Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
Status: Downloaded newer image for hello-world:latest
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

```
https://cloud.docker.com/
```

For more examples and ideas, visit:

```
https://docs.docker.com/engine/userguide/
```



Hello again

```
$ docker container run -it ubuntu bash
```



Welcome to container

```
root@c20e8c218664:/# curl google.com  
bash: curl: command not found
```



Check installed packaged

```
root@c20e8c218664:/# dpkg -l | wc -l  
103
```



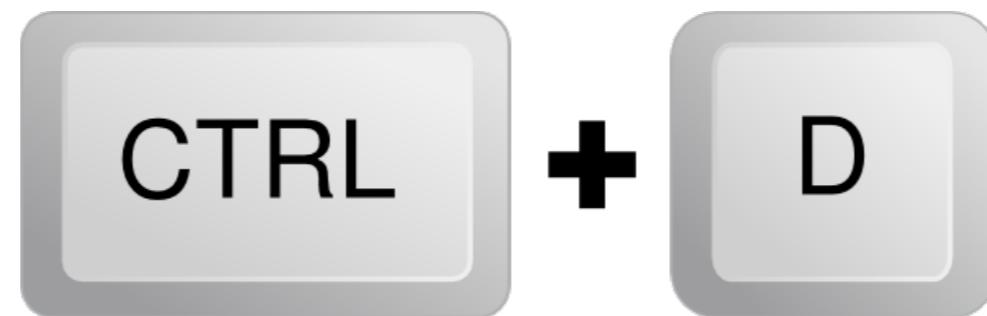
Install package in container

```
root@c20e8c218664:/# apt update  
root@c20e8c218664:/# apt install curl  
root@c20e8c218664:/# curl google.com
```



Exit from container

```
root@c20e8c218664:/# exit
```



Start another container

\$ docker container run -it ubuntu bash

\$ curl google.com

bash: curl: command not found



This is container



Basic of Docker

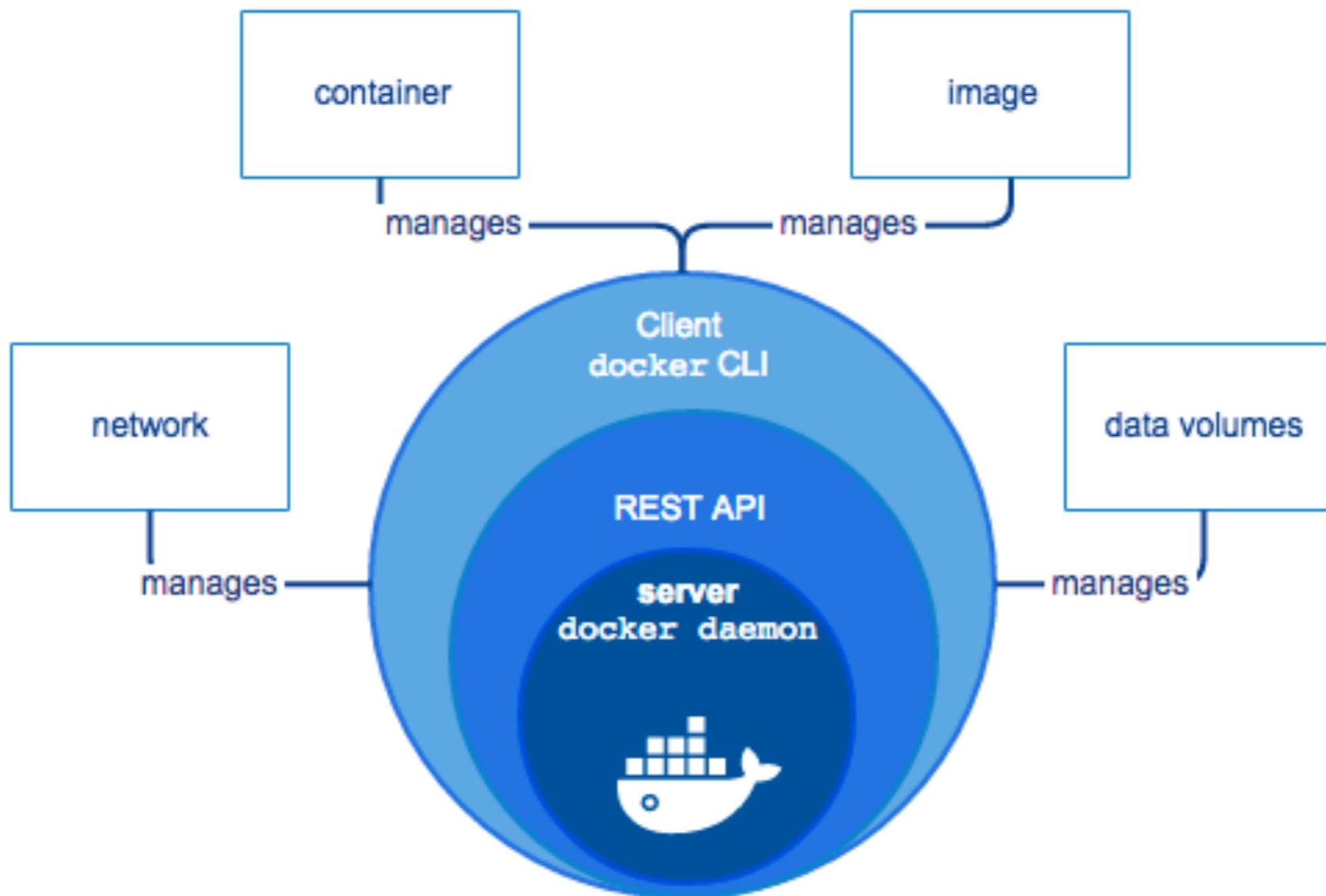


Basic of Docker

**Image
Container
Dockerfile
Registry
Network
Volume**



Basic of Docker



<https://docs.docker.com/get-started/overview/>



Image vs Container

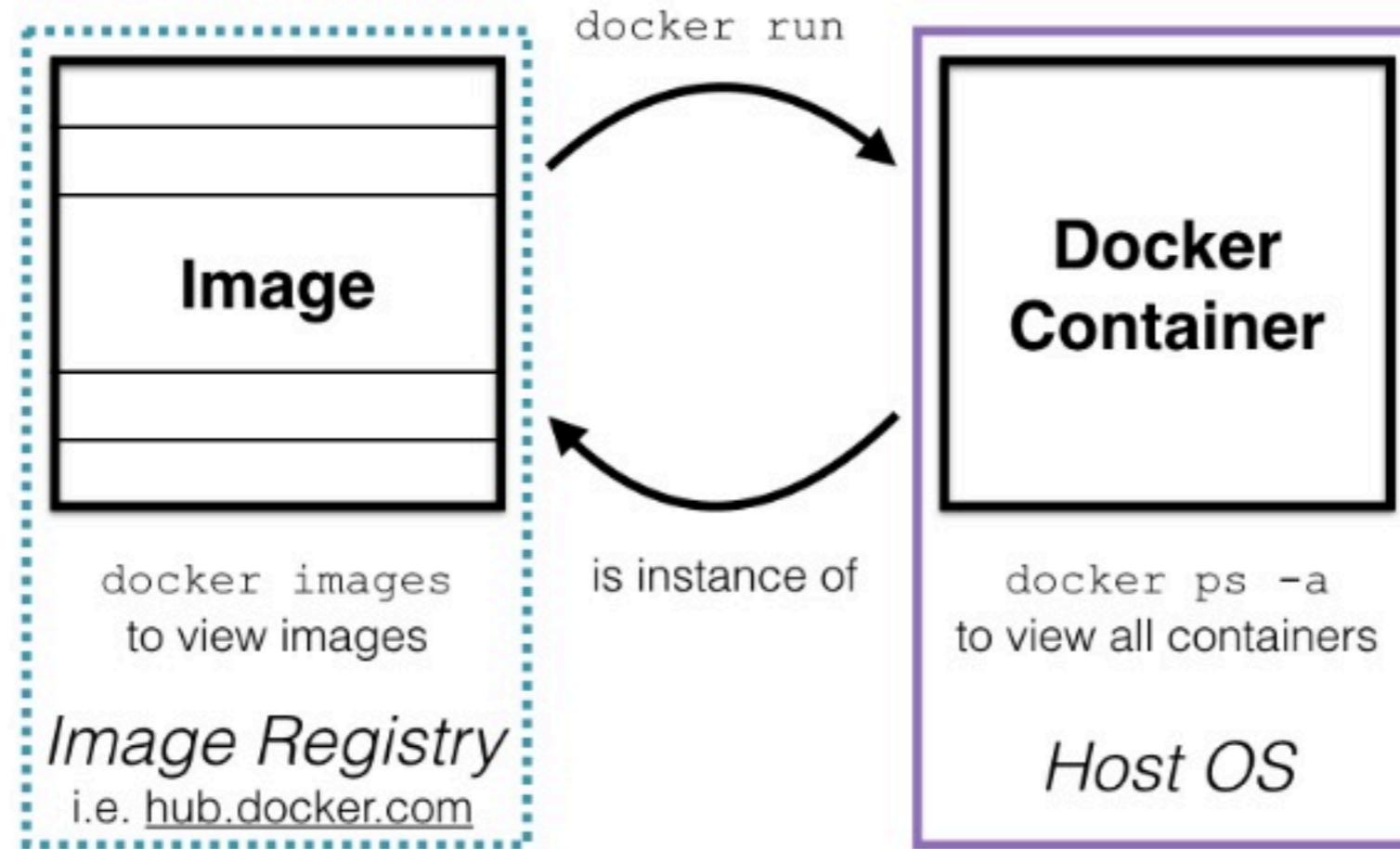


Image vs Container

Image like template/blueprint

Containers are created from image



Image vs Container

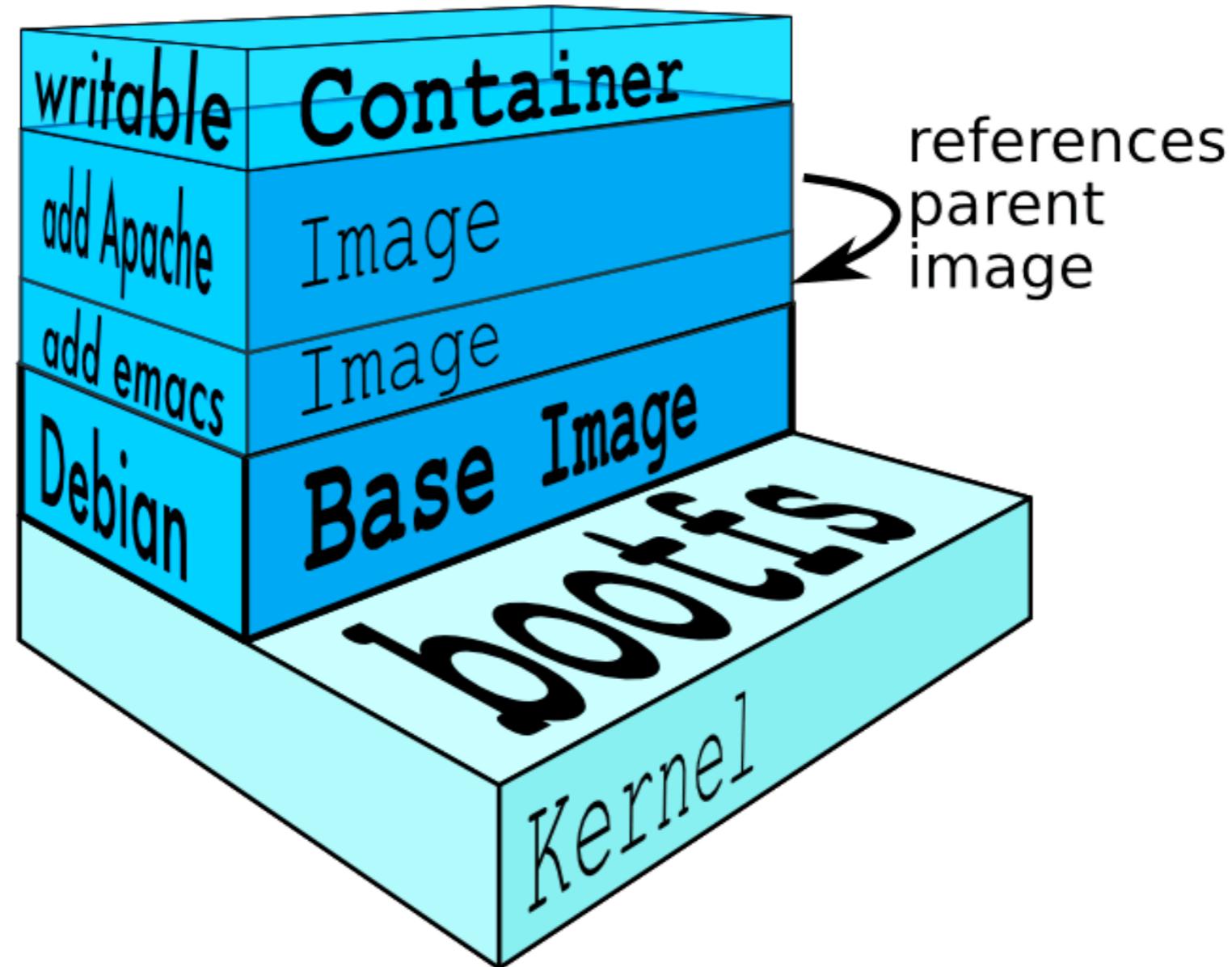
Image = class

Containers = instance

Layer = inheritance



Docker image vs container



Docker image

Collection of files and some meta data

Made of **layers**

Each layer can add/change/remove files

Image can share layers

Read-only file system

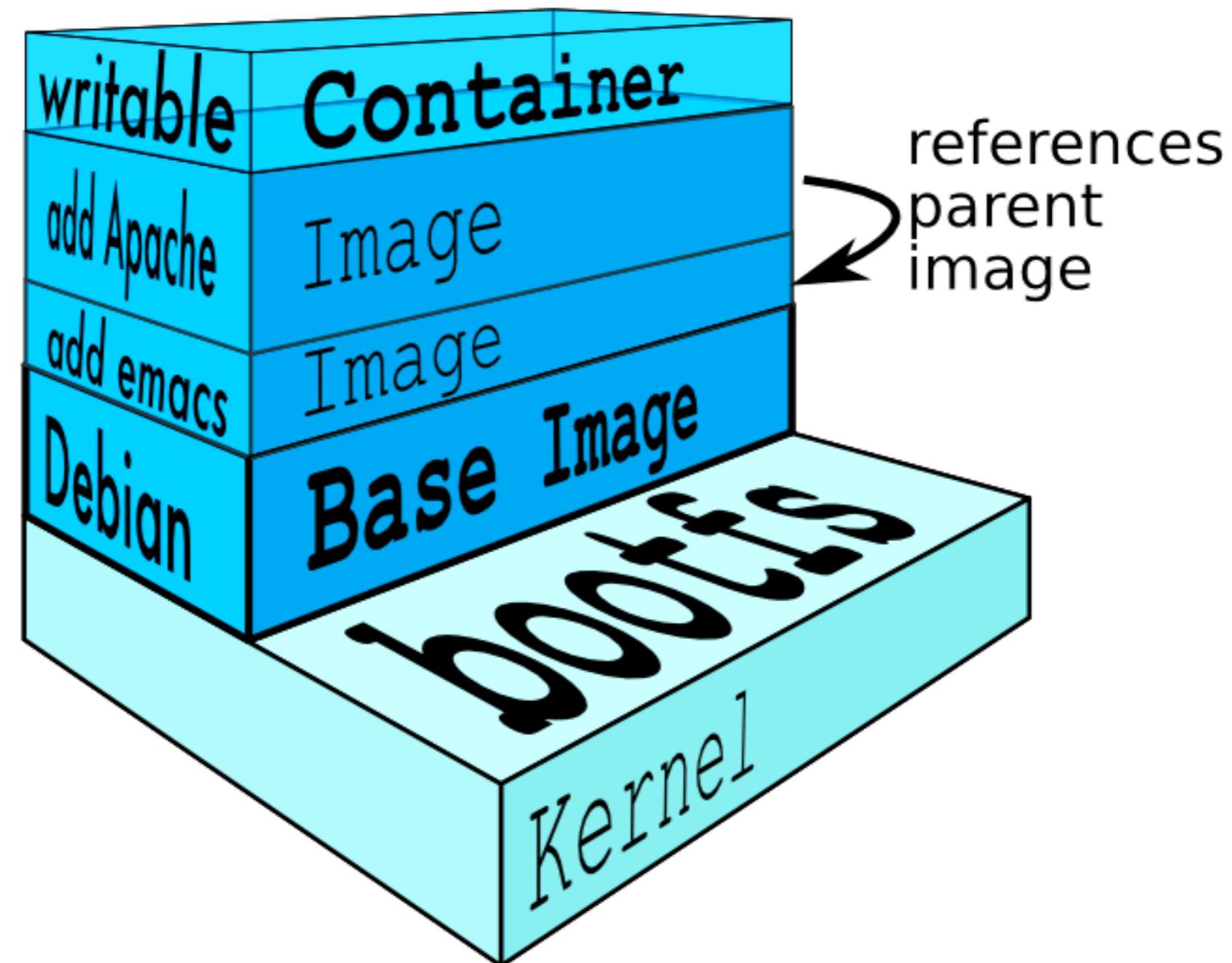


How do we change image ?

We do not !!



How do we change image ?

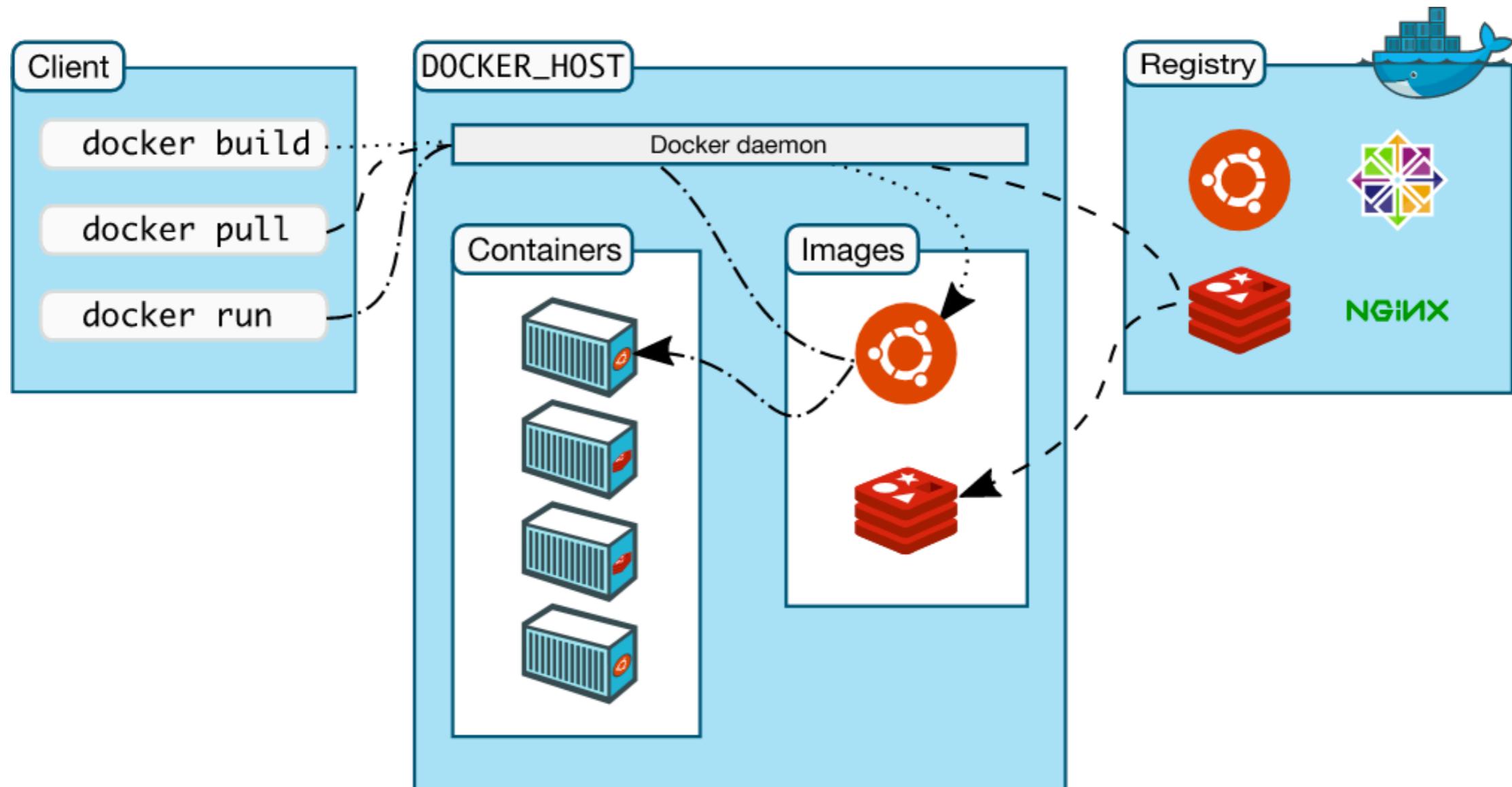


How do we change image ?

1. Create a new container from image
2. Changes in container
3. Transform into new layer
4. Create a new image on top the old image



How docker works ?



<https://docs.docker.com/get-started/overview/>



Docker commands



Docker commands

Running container

Running process

Clean up



Management commands

\$docker image

\$docker container

\$docker network

\$docker service

\$docker volume

\$docker system



Commands

\$docker build

\$docker commit

\$docker pull

\$docker run

\$docker rmi

\$docker rm



More commands

\$docker



Image command



List all images

\$docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	4a415e366388	2 weeks ago	3.99 MB
ubuntu	latest	0ef2e08ed3fa	2 weeks ago	130 MB
ubuntu	xenial	0ef2e08ed3fa	2 weeks ago	130 MB
ubuntu	trusty	7c09e61e9035	2 weeks ago	188 MB
hello-world	latest	48b5124b2768	2 months ago	1.84 kB

Old way command



List all images

\$ docker image ls -a

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	4a415e366388	2 weeks ago	3.99 MB
ubuntu	latest	0ef2e08ed3fa	2 weeks ago	130 MB
ubuntu	xenial	0ef2e08ed3fa	2 weeks ago	130 MB
ubuntu	trusty	7c09e61e9035	2 weeks ago	188 MB
hello-world	latest	48b5124b2768	2 months ago	1.84 kB



Search images

\$docker search <image name>

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED [OK]
divyakumarjain/nginx-proxy	nginx-proxy	1		
juanheredia/ngixkoken		0		
dai1219/nginx		0		
vinithmenon28/admatic-nginx		0		
bowlingx/docker-nginx-php-v8		0		
almanacproject/nginx-proxy-smqueryapi		0		
fangufl/docker-static-nginx		0		
eferlo/nginx		0		
slashn/nginxiptest2		0		
gymnae/webserverbase	Web server image with glidenlabs/alpine:ed...	0		[OK]
nomiad/ngixsample	test	0		
codexpage/ngixx		0		
jerideng/nginx		0		
yavinenanana/nginx		0		

<https://hub.docker.com/>



Download images (old)

\$docker pull <image name>

\$docker run <image name>



Download images (new)

\$docker image pull <image name>

\$docker container run <image name>



Docker with Ubuntu

OFFICIAL REPOSITORY

[ubuntu](#) 

Last pushed: 16 days ago

[Repo Info](#) [Tags](#)

Short Description

Ubuntu is a Debian-based Linux operating system based on free software.

Docker Pull Command

`docker pull ubuntu`

Full Description

Supported tags and respective [Dockerfile](#) links

- `12.04.5`, `12.04`, `precise-20170214`, `precise` ([precise/Dockerfile](#))
- `14.04.5`, `14.04`, `trusty-20170214`, `trusty` ([trusty/Dockerfile](#))
- `16.04`, `xenial-20170214`, `xenial`, `latest` ([xenial/Dockerfile](#))
- `16.10`, `yakkety-20170224`, `yakkety`, `rolling` ([yakkety/Dockerfile](#))
- `17.04`, `zesty-20170224`, `zesty`, `devel` ([zesty/Dockerfile](#))

https://hub.docker.com/_/ubuntu/



Pull image

\$docker image pull ubuntu:latest

```
latest: Pulling from library/nginx
Digest: sha256:52a189e49c0c797cf5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
Status: Image is up to date for nginx:latest
[MacBook-Pro-2% docker pull ubuntu:latest
latest: Pulling from library/ubuntu
d54efb8db41d: Downloading [=====] 23.87 MB/50.43 MB
f8b845f45a87: Download complete
e8db7bf7c39f: Download complete
9654c40e9079: Download complete
6d9ef359eaaa: Download complete
```



Pull image

```
$ docker image pull ubuntu:latest
```

Tag



Image and tags

Image can have **tags**

Tags define image variants

Default of tag is **:latest**

:latest tag can be updated frequently !!



Pull image with specified tag

```
$ docker image pull ubuntu:xenial
```

Specified tag



Remove image

\$docker image rm <id>



Remove all image

```
$docker image rmi $(docker image ls -a -q)
```



Remove all image

```
$docker image rm <id/name>
```



Remove all image

\$docker image prune



Docker image command

\$docker image

```
Usage: docker image COMMAND

Manage images

Options:
  --help  Print usage

Commands:
  build      Build an image from a Dockerfile
  history    Show the history of an image
  import     Import the contents from a tarball to create a filesystem image
  inspect    Display detailed information on one or more images
  load       Load an image from a tar archive or STDIN
  ls         List images
  prune     Remove unused images
  pull       Pull an image or a repository from a registry
  push       Push an image or a repository to a registry
  rm        Remove one or more images
  save      Save one or more images to a tar archive (streamed to STDOUT by default)
  tag       Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Run 'docker image COMMAND --help' for more information on a command.
```



Container command



List all containers

```
$docker container ps -a
```



List all containers

```
$docker container ps -a
```



Docker container ps

See latest container that was started

\$docker container ps -l

Show only container id

\$docker container ps -q



View log in container

\$docker container logs <id>



View log in container

\$docker container logs <id> --follow



Tail log

\$docker container logs --tail <no. line> <id>



Tail log in real time

```
$ docker container logs \
  --tail <no. line> \
  --follow \
<id>
```



Stop/remove all containers

```
$ docker container stop $(docker ps -a -q)
```

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



Remove all stopped container

\$docker container prune



Docker container command

\$docker container

```
Usage: docker container COMMAND

Manage containers

Options:
    --help  Print usage

Commands:
    attach      Attach to a running container
    commit      Create a new image from a container's changes
    cp          Copy files/folders between a container and the local filesystem
    create      Create a new container
    diff        Inspect changes to files or directories on a container's filesystem
    exec        Run a command in a running container
    export      Export a container's filesystem as a tar archive
    inspect    Display detailed information on one or more containers
    kill        Kill one or more running containers
    logs        Fetch the logs of a container
    ls          List containers
    pause       Pause all processes within one or more containers
    port        List port mappings or a specific mapping for the container
    prune      Remove all stopped containers
    rename     Rename a container
    restart    Restart one or more containers
    rm         Remove one or more containers
    run         Run a command in a new container
    start      Start one or more stopped containers
    stats      Display a live stream of container(s) resource usage statistics
    stop       Stop one or more running containers
    top        Display the running processes of a container
    unpause   Unpause all processes within one or more containers
    update     Update configuration of one or more containers
    wait       Block until one or more containers stop, then print their exit codes

Run 'docker container COMMAND --help' for more information on a command.
```



Container process



Container run process

Foreground

Interactive

Background



Let's start

\$docker run

\$docker container run

https://docs.docker.com/engine/reference/commandline/container_run/#options



Docker with nginx

OFFICIAL REPOSITORY

[nginx](#) 

Last pushed: 20 days ago

[Repo Info](#) [Tags](#)

Short Description

Official build of Nginx.

Docker Pull Command 

```
docker pull nginx
```

Full Description

Supported tags and respective [Dockerfile](#) links

- `1.11.10`, `mainline`, `1`, `1.11`, `latest` ([mainline/jessie/Dockerfile](#))
- `1.11.10-alpine`, `mainline-alpine`, `1-alpine`, `1.11-alpine`, `alpine` ([mainline/alpine/Dockerfile](#))
- `1.10.3`, `stable`, `1.10` ([stable/jessie/Dockerfile](#))
- `1.10.3-alpine`, `stable-alpine`, `1.10-alpine` ([stable/alpine/Dockerfile](#))

https://hub.docker.com/_/nginx/



Foreground

\$ docker container run nginx

```
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
693502eb7dfb: Pull complete
6decb850d2bc: Pull complete
c3e19f087ed6: Pull complete
Digest: sha256:52a189e49c0c797cf5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
Status: Downloaded newer image for nginx:latest
```



Foreground

\$docker container run jpetazzo/clock

```
Thu Mar 23 15:30:40 UTC 2017
Thu Mar 23 15:30:41 UTC 2017
Thu Mar 23 15:30:42 UTC 2017
Thu Mar 23 15:30:43 UTC 2017
Thu Mar 23 15:30:44 UTC 2017
Thu Mar 23 15:30:45 UTC 2017
Thu Mar 23 15:30:46 UTC 2017
Thu Mar 23 15:30:47 UTC 2017
```



Background

\$docker container run -d nginx

-d = --detach

Run container in background and print container ID



Interactive

```
$ docker container run -i -t nginx bash
```

-i = --interactive

-t = --tty



Start/Stop container

\$docker container **start** <id/name>

\$docker container **stop** <id/name>

\$docker container **restart** <id/name>



Interactive without exited !!

```
$ docker container run -i -t nginx bash
```

Ctrl+p + Ctrl+q.



Access to container

\$ docker container run -i -t

\$ docker container exec -i -t



Access to start container

\$ docker container run -i -t <id> bash

\$ docker container run -i -t <name> bash



Access to existing container

\$ docker container exec -i -t <id> bash

\$ docker container exec -i -t <name> bash



Delete all containers

\$docker container ?



Remove after exited

\$docker container run **--rm** nginx

\$docker container run **--rm -d** nginx

\$docker container run **--rm -it** nginx bash



Start nginx with name

```
$ docker container run \  
  --name hello-nginx \  
  -d nginx
```



Rename container

```
$docker container rename <old> <new>
```



Remove container

```
$ docker container stop hello-nginx
```

```
$ docker container rm hello-nginx
```



Monitoring container

\$docker container top

\$docker container inspect

\$docker container stats

\$docker container logs



Monitoring tools

Docker dashboard

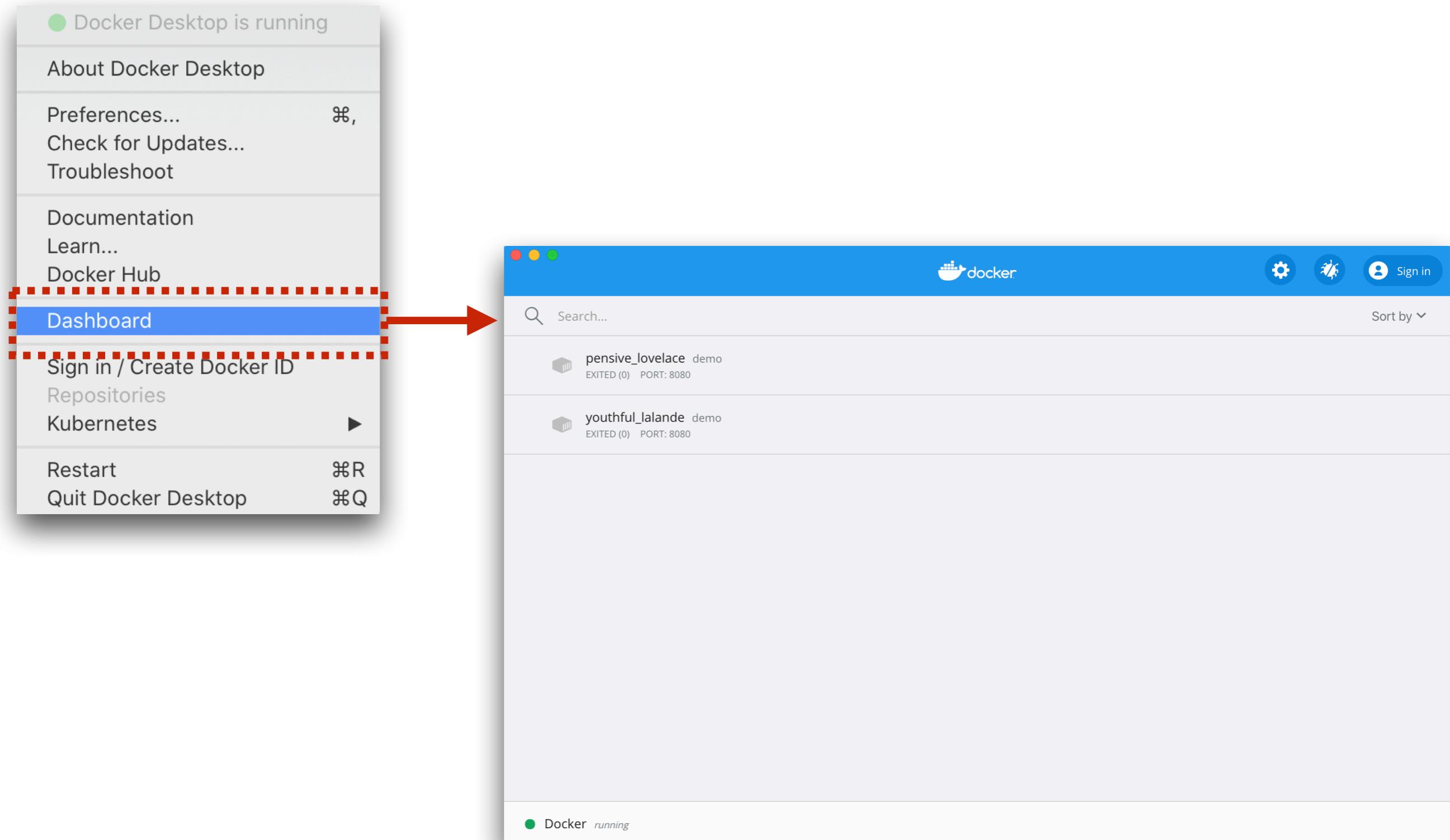
cAdvisor

Fluentd + Grafana

<https://github.com/up1/course-introduction-docker/tree/master/workshop/monitoring>



Docker Dashboard



cAdvisor



cAdvisor

<https://github.com/google/cadvisor>



What happen in docker container run ?

1. Find image locally in image cached
2. Find image in remote repository (Docker Hub)
3. Download with specified version (Latest)
4. Create new container from image
5. Assign virtual IP on private network to container
6. Open port on host and forward to port in container
7. Start container by using CMD, ENTRY point



Building image with interactive



Step to build

1. Install softwares in a container
2. Create new image
3. Create a container from new image
4. Share new image



Create a new container

```
$docker container run -it ubuntu
```



Install wget in container

```
/#apt update && apt install wget
```



Inspect the changes

\$docker container diff <id>

```
C ./wh..wh.plnk
A ./wh..wh.plnk/98.272433
C /etc
A /etc/ca-certificates
A /etc/ca-certificates/update.d
A /etc/ca-certificates.conf
C /etc/ld.so.cache
A /etc/ssl
A /etc/ssl/certs
A /etc/ssl/certs/00673b5b.0
```



Commit and run image

\$ docker container commit <id> <new>

\$ docker container run it <new> bash

#/wget somkiat.cc



Tagging image

\$docker container tag <id> <tag name>

\$docker container commit <id> <tag name>



Manual process = **bad**
Automated process = **good**



Automate build process with Dockerfile



Building image with Dockerfile



Dockerfile

Build recipe for a Docker image

Contain series of instructions

Use **docker build** command



First Dockerfile

```
FROM ubuntu  
RUN apt-get update  
RUN apt-get install -y wget
```



Build image !!

\$docker image build -t first_image .

```
Sending build context to Docker daemon 2.048 kB
Step 1/3 : FROM ubuntu
--> 0ef2e08ed3fa
Step 2/3 : RUN apt-get update
--> Running in 6c598d2946b7
Get:1 http://archive.ubuntu.com/ubuntu xenial InR
Get:2 http://archive.ubuntu.com/ubuntu xenial-upd
Get:3 http://archive.ubuntu.com/ubuntu xenial-sec
Get:4 http://archive.ubuntu.com/ubuntu xenial/main
Get:5 http://archive.ubuntu.com/ubuntu xenial/repo
```



Run image !!

```
$ docker container run -it first_image bash
```



History of image

Show all layers of image

\$docker image history <image name>

IMAGE	CREATED	CREATED BY	SIZE
1813d5ecf658	4 minutes ago	/bin/sh -c apt-get install -y wget	7.35 MB
2930e9a322d6	5 minutes ago	/bin/sh -c apt-get update	40.1 MB
0ef2e08ed3fa	3 weeks ago	/bin/sh -c #(nop) CMD ["/bin/bash"]	0 B
<missing>	3 weeks ago	/bin/sh -c mkdir -p /run/systemd && echo '...'	7 B
<missing>	3 weeks ago	/bin/sh -c sed -i 's/^#\s*\(\deb.*universe\ ...)	1.9 kB
<missing>	3 weeks ago	/bin/sh -c rm -rf /var/lib/apt/lists/*	0 B
<missing>	3 weeks ago	/bin/sh -c set -xe && echo '#!/bin/sh' >...	745 B
<missing>	3 weeks ago	/bin/sh -c #(nop) ADD file:efb254bc677d66d...	130 MB



CMD and ENTRYPOINT

Setting **default command**
to run in a container



Define a default command

Execute wget to get public ip
from ifconfig.me

```
$wget -O- -q http://ifconfig.me/ip
```



Add CMD to Dockerfile

```
FROM ubuntu  
RUN apt-get update  
RUN apt-get install -y wget  
  
CMD wget -O- -q http://ifconfig.me/ip
```



Build image and test

```
$ docker image build -t ifconfig .
```

```
$ docker container run ifconfig
```



We need ...

\$docker container run ifconfig somkiat.cc



Add ENTRYPOINT to Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y wget
ENTRYPOINT ["wget", "-O-", "-q"]
```



CMD + ENTRYPOINT

CMD define the base command

ENTRYPOINT define the default parameters



Add ENTRYPOINT to Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y wget
ENTRYPOINT ["wget", "-O-", "-q"]
CMD http://ifconfig.me/ip
```



Build image and test

```
$ docker image build -t ifconfig .
```

```
$ docker container run ifconfig
```

```
$ docker container run ifconfig somkiat.cc
```



Override ENTRYPPOINT

```
$ docker container run -it \  
  --entrypoint bash \  
  ifconfig
```



Dockerfile usage summary

Instructions are **executed in order**

Each instruction **creates a new layer** in images

Instructions are **cached**

FROM instruction must be the first

Comment with **#**

One CMD, One ENTRYPOINT



FROM instruction

Specified the source image to build

Must be the first instruction in Dockerfile



FROM instruction

FROM ubuntu

FROM ubuntu:16

FROM somkiat/hello

FROM localhost:5000/hello



MAINTAINER instruction

Who write this Docker file

It's optional but recommended

MAINTAINER somkiat pui <somkiat@gmail.com>



RUN instruction

Execute a command

Record changes made to the file system

Install libraries, packages and files



RUN instruction

It's can be specified in 2 ways

RUN apt-get update

RUN [“apt-get”, “update”]



RUN will NOT

Record state of processes

Automatically start deamons



EXPOSE instruction

Tell docker what ports are to be published

All ports are private by default

EXPOSE 8080



EXPOSE instruction

EXPOSE 8080



Publish by port

```
$docker run -p 8080:8080
```

Even if it was not declared with EXPOSE



Publish all port from EXPOSE

```
$ docker run -P
```



ADD instruction

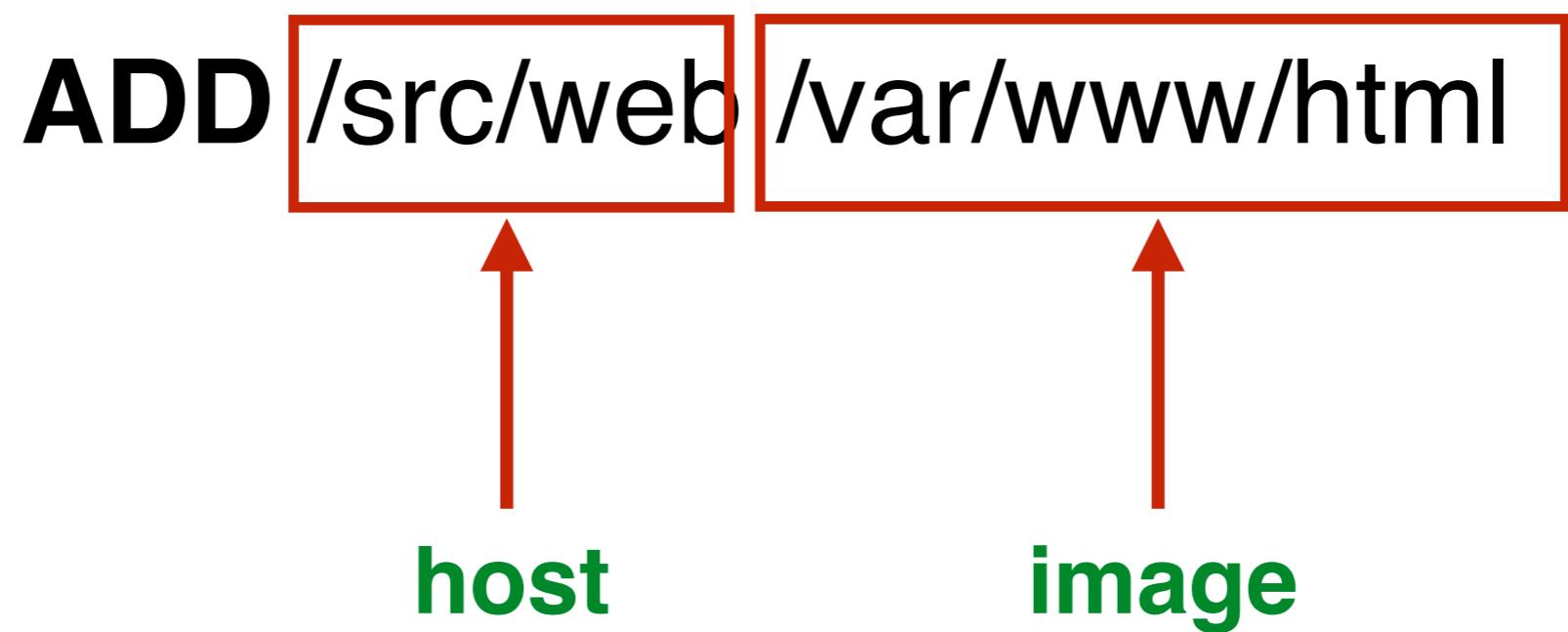
Add files and content from host into image

```
ADD /src/web /var/www/html
```



ADD instruction

Add files and content from host into image



ADD remote files

ADD http://example.com/web /opt



ADD instruction

ADD is cached

If the local source is zip/tar
it's will be unpacked to the destination



VOLUME instruction

Create data volume mount point

```
VOLUME [“/var/data”]
```



VOLUME instruction

Share and reuse between containers

Share volume with a **stopped** container

Data volume persist until all containers referencing them are destroyed



WORKDIR instruction

Set the working directory for subsequent instructions

It also affects **CMD** and **ENTRYPOINT**

WORKDIR /opt/web



ENV instruction

Specified environment variables

```
ENV WEB_PORT 8080
```

```
$docker run -e WEB_PORT=8080
```



Best practice for write Dockerfile

https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/



Building an efficient Dockerfile

Each line in a Dockerfile creates a new layer

Combine multiple similar commands into one

&& to continue commands

\ to wrap lines



Bad

```
RUN apt-get install -y build-essential  
RUN apt-get install -y python2.7  
RUN apt-get install -y python2.7-dev  
RUN apt-get install -y python-setuptools  
RUN apt-get install -y python-software-properties  
RUN apt-get install -y python-pip
```



Good

```
RUN apt-get install -y build-essential python2.7 python2.7-dev  
python-setuptools python-software-properties python-pip
```



Building an efficient Dockerfile

Build your Dockerfile to take
advantage of **Docker's caching system**



Building an efficient Dockerfile

ADD dependency list by themselves
to avoid reinstalling unchanged dependencies



BAD Dockerfile

```
FROM ubuntu:latest
MAINTAINER somkiat <somkiat@gmail.com>
RUN apt-get update
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q \
    python-all python-pip
ADD ./webapp /opt/webapp/
WORKDIR /opt/webapp
RUN pip install -qr requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```



Try to build image

```
$ docker image build -t bad .
```



Steps to build image

Sending build context to Docker daemon 5.632 kB

Step 1/9 : FROM ubuntu:latest

Step 2/9 : MAINTAINER somkiat <somkiat@gmail.com>

Step 3/9 : RUN apt-get update

Step 4/9 : RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q

Step 5/9 : ADD ./webapp /opt/webapp/

Step 6/9 : WORKDIR /opt/webapp

Step 7/9 : RUN pip install -qr requirements.txt

Step 8/9 : EXPOSE 5000

Step 9/9 : CMD python app.py



Layers of image

\$docker history <name>

IMAGE	CREATED	CREATED BY	SIZE
2429ccdcab68	3 minutes ago	/bin/sh -c #(nop) CMD ["python" "app.py"]	0 B
10484080db5d	3 minutes ago	/bin/sh -c #(nop) EXPOSE 5000/tcp	0 B
ee1108d1334c	3 minutes ago	/bin/sh -c pip install -qr requirements.txt	5.88 MB
fad25ba092a3	3 minutes ago	/bin/sh -c #(nop) WORKDIR /opt/webapp	0 B
03be4d94e086	3 minutes ago	/bin/sh -c #(nop) ADD dir:d42b0abdbfc069d1...	615 B
78edb3715e4f	3 minutes ago	/bin/sh -c DEBIAN_FRONTEND=noninteractive ...	278 MB
937cd364ce48	15 minutes ago	/bin/sh -c apt-get update	40.1 MB
fb769fba0cd	17 minutes ago	/bin/sh -c #(nop) MAINTAINER somkiat <som...	0 B
0ef2e08ed3fa	3 weeks ago	/bin/sh -c #(nop) CMD ["/bin/bash"]	0 B
<missing>	3 weeks ago	/bin/sh -c mkdir -p /run/systemd && echo '...	7 B
<missing>	3 weeks ago	/bin/sh -c sed -i 's/^#\s*/(deb.*universe\.../	1.9 kB
<missing>	3 weeks ago	/bin/sh -c rm -rf /var/lib/apt/lists/*	0 B
<missing>	3 weeks ago	/bin/sh -c set -xe && echo '#!/bin/sh' >...	745 B
<missing>	3 weeks ago	/bin/sh -c #(nop) ADD file:efb254bc677d66d...	130 MB



Try to build image again

\$ docker image build -t bad .

```
Sending build context to Docker daemon 6.656 kB
Step 1/9 : FROM ubuntu:latest
--> 0ef2e08ed3fa
Step 2/9 : MAINTAINER somkiat <somkiat@gmail.com>
--> Using cache
--> fbd769fba0cd
Step 3/9 : RUN apt-get update
--> Using cache
--> 937cd364ce48
Step 4/9 : RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q      python-all python-pip
--> Using cache
--> 78edb3715e4f
Step 5/9 : ADD ./webapp /opt/webapp/
--> 1c5da1b5f7ad
Removing intermediate container 029dfdd4ca2e
Step 6/9 : WORKDIR /opt/webapp
--> d48dffabc01b
Removing intermediate container 24f776e6e96f
Step 7/9 : RUN pip install -qr requirements.txt
--> Running in 1f6f1b49d0d9
```



Caching system

\$docker image build -t bad .

```
Sending build context to Docker daemon 6.656 kB
Step 1/9 : FROM ubuntu:latest
--> 0ef2e08ed3fa
Step 2/9 : MAINTAINER somkiat <somkiat@gmail.com>
--> Using cache
--> fbd769fba0cd
Step 3/9 : RUN apt-get update
--> Using cache
--> 937cd364ce48
Step 4/9 : RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q      python-all python-pip
--> Using cache
--> 78edb3715e4f
Step 5/9 : ADD ./webapp /opt/webapp/
--> 1c5da1b5f7ad
Removing intermediate container 029dfdd4ca2e
Step 6/9 : WORKDIR /opt/webapp
--> d48dffabc01b
Removing intermediate container 24f776e6e96f
Step 7/9 : RUN pip install -qr requirements.txt
--> Running in 1f6f1b49d0d9
```



Try to change code !!

\$docker image build -t bad .

```
Step 5/9 : ADD ./webapp /opt/webapp/
--> 47e33571ebaa
Removing intermediate container 04dd8cea0b7f
Step 6/9 : WORKDIR /opt/webapp
--> a53b54997d05
Removing intermediate container 5f2ac16b2fca
Step 7/9 : RUN pip install -qr requirements.txt
--> Running in f91e4bb81831
--> 68c2b7dba5f5
Removing intermediate container f91e4bb81831
Step 8/9 : EXPOSE 5000
--> Running in 7263f2ee0981
--> b1feb795684a
Removing intermediate container 7263f2ee0981
Step 9/9 : CMD python app.py
--> Running in 40fbda4aba40
--> ede1da1b7c0a
```



Caching is destroyed !!

\$docker image build -t bad .

```
Step 5/9 : ADD ./webapp /opt/webapp/
--> 47e33571ebaa
Removing intermediate container 04dd8cea0b7f
Step 6/9 : WORKDIR /opt/webapp
--> a53b54997d05
Removing intermediate container 5f2ac16b2fca
Step 7/9 : RUN pip install -qr requirements.txt
--> Running in f91e4bb81831
--> 68c2b7dba5f5
Removing intermediate container f91e4bb81831
Step 8/9 : EXPOSE 5000
--> Running in 7263f2ee0981
--> b1feb795684a
Removing intermediate container 7263f2ee0981
Step 9/9 : CMD python app.py
--> Running in 40fbda4aba40
--> ede1da1b7c0a
```



BAD Dockerfile

The dependencies are reinstalled every time



BAD Dockerfile

```
FROM ubuntu:latest
MAINTAINER somkiat <somkiat@gmail.com>
RUN apt-get update
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q \
    python-all python-pip
ADD ./webapp /opt/webapp/
WORKDIR /opt/webapp
RUN pip install -qr requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```



Fixed Dockerfile

Add dependencies as a separate step (Cached)



Fixed Dockerfile

```
FROM ubuntu:latest
MAINTAINER somkiat <somkiat@gmail.com>
RUN apt-get update
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q \
    python-all python-pip
ADD ./webapp/requirements.txt /tmp/requirements.txt
RUN pip install -qr /tmp/requirements.txt
ADD ./webapp /opt/webapp/
WORKDIR /opt/webapp
EXPOSE 5000
CMD ["python", "app.py"]
```



Clean up after process !!

Remove unnecessary file/directory

Reduce size of container



Example

Update dependencies in container

```
FROM debian:jessie
```

```
RUN DEBIAN_FRONTEND=noninteractive apt-get update
```

```
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y vim
```



Create image

```
$docker build -t start .
```



History of image (Layer)

\$docker history start

IMAGE	CREATED	CREATED BY	SIZE
8dc5716e6e33	29 minutes ago	/bin/sh -c DEBIAN_FRONTEND=noninteractive ...	28.7 MB
7f1673f232ef	29 minutes ago	/bin/sh -c DEBIAN_FRONTEND=noninteractive ...	9.88 MB
8cedef9d7368	4 days ago	/bin/sh -c #(nop) CMD ["/bin/bash"]	0 B
<missing>	4 days ago	/bin/sh -c #(nop) ADD file:4eedf861fb567ff...	123 MB



Create container and update

```
$ docker container -it --name os01 start bash
```

```
./#apt-get update
```



Diff in container

\$docker diff start

```
C /tmp
C /var
C /var/lib
C /var/lib/apt
D /var/lib/apt/.wh...opq
C /var/lib/apt/lists
C /var/lib/apt/lists/deb.debian.org_dists_jessie-updates_InRelease
C /var/lib/apt/lists/deb.debian.org_dists_jessie-updates_main_binary-amd64_Packages.gz
C /var/lib/apt/lists/deb.debian.org_dists_jessie_Release
C /var/lib/apt/lists/deb.debian.org_dists_jessie_Release.gpg
C /var/lib/apt/lists/deb.debian.org_dists_jessie_main_binary-amd64_Packages.gz
C /var/lib/apt/lists/lock
C /var/lib/apt/lists/partial
C /var/lib/apt/lists/security.debian.org_dists_jessie_updates_InRelease
C /var/lib/apt/lists/security.debian.org_dists_jessie_updates_main_binary-amd64_Packages.gz
```



Size of data in container

```
$docker ps -s  
--format 'table {{.Names}}\t{{.Image}}\t{{.Size}}'
```

NAMES	IMAGE	SIZE
os01	os01	9.88 MB (virtual 172 MB)



Should be remove !!

\$rm -rf /var/lib/apt

NAMES	IMAGE	SIZE
os01	os01	0 B (virtual 162 MB)



Update Dockerfile

Add remove/clean up process

```
FROM debian:jessie
```

```
RUN DEBIAN_FRONTEND=noninteractive apt-get update \
&& apt-get install -y vim \
&& rm -rf /var/lib/apt
```



History of image (Layer)

\$docker history start

IMAGE	CREATED	CREATED BY	SIZE
6722968e68ec	6 hours ago	/bin/sh -c DEBIAN_FRONTEND=noninteractive ...	28.7 MB
8cedef9d7368	4 days ago	/bin/sh -c #(nop) CMD ["/bin/bash"]	0 B
<missing>	4 days ago	/bin/sh -c #(nop) ADD file:4eedf861fb567ff...	123 MB



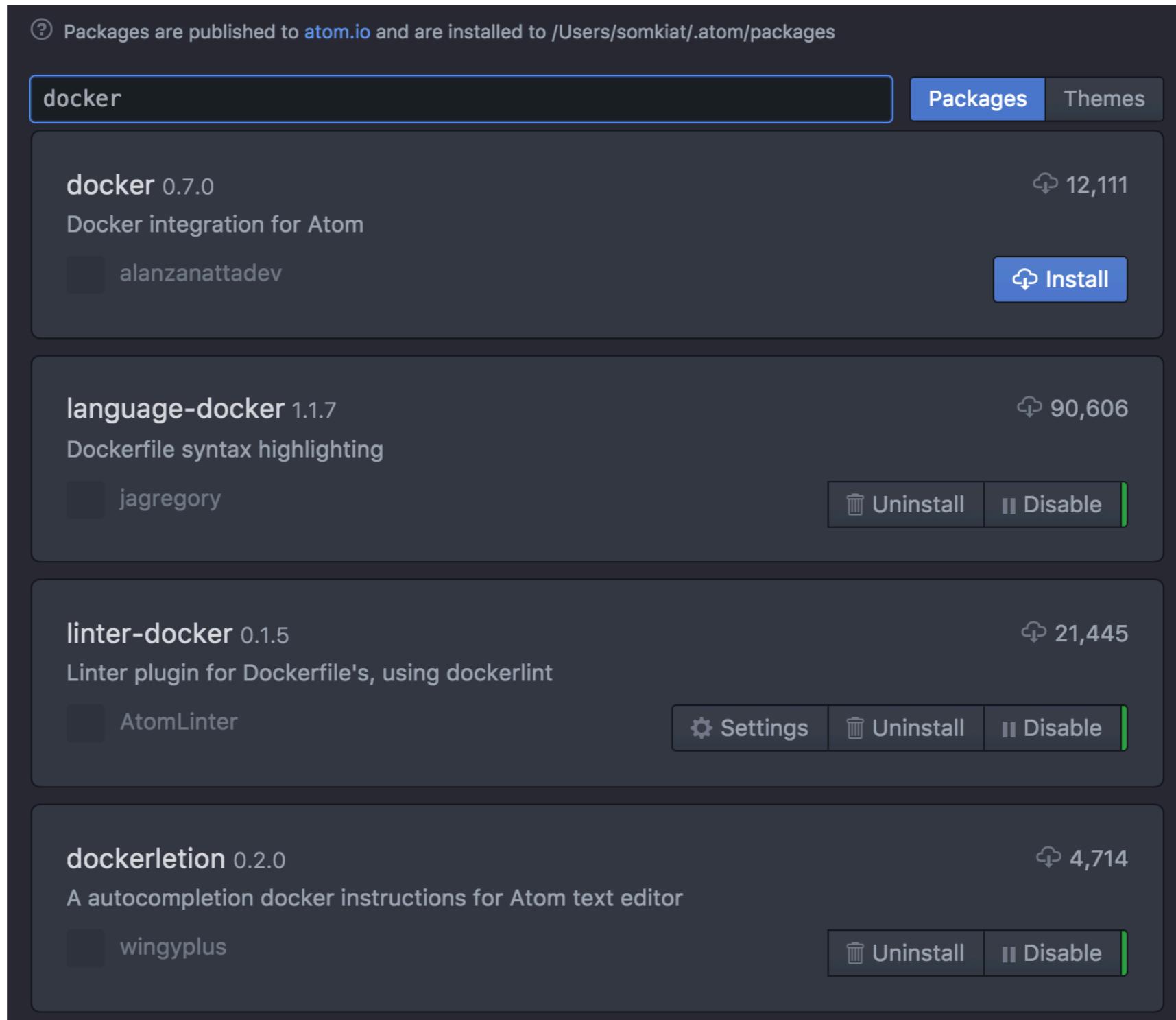
Using .dockerignore

Exclude unused files and directories

<https://docs.docker.com/engine/reference/builder/#dockerignore-file>



Tips Atom with Dockerfile



Basic of Container networking



Networking

Run network service in container

Manipulate container networking

Find IP's container



Command line with network

\$docker container run -p

\$docker container port



Default Networking

Each container connect to private virtual network that called “**bridge**”

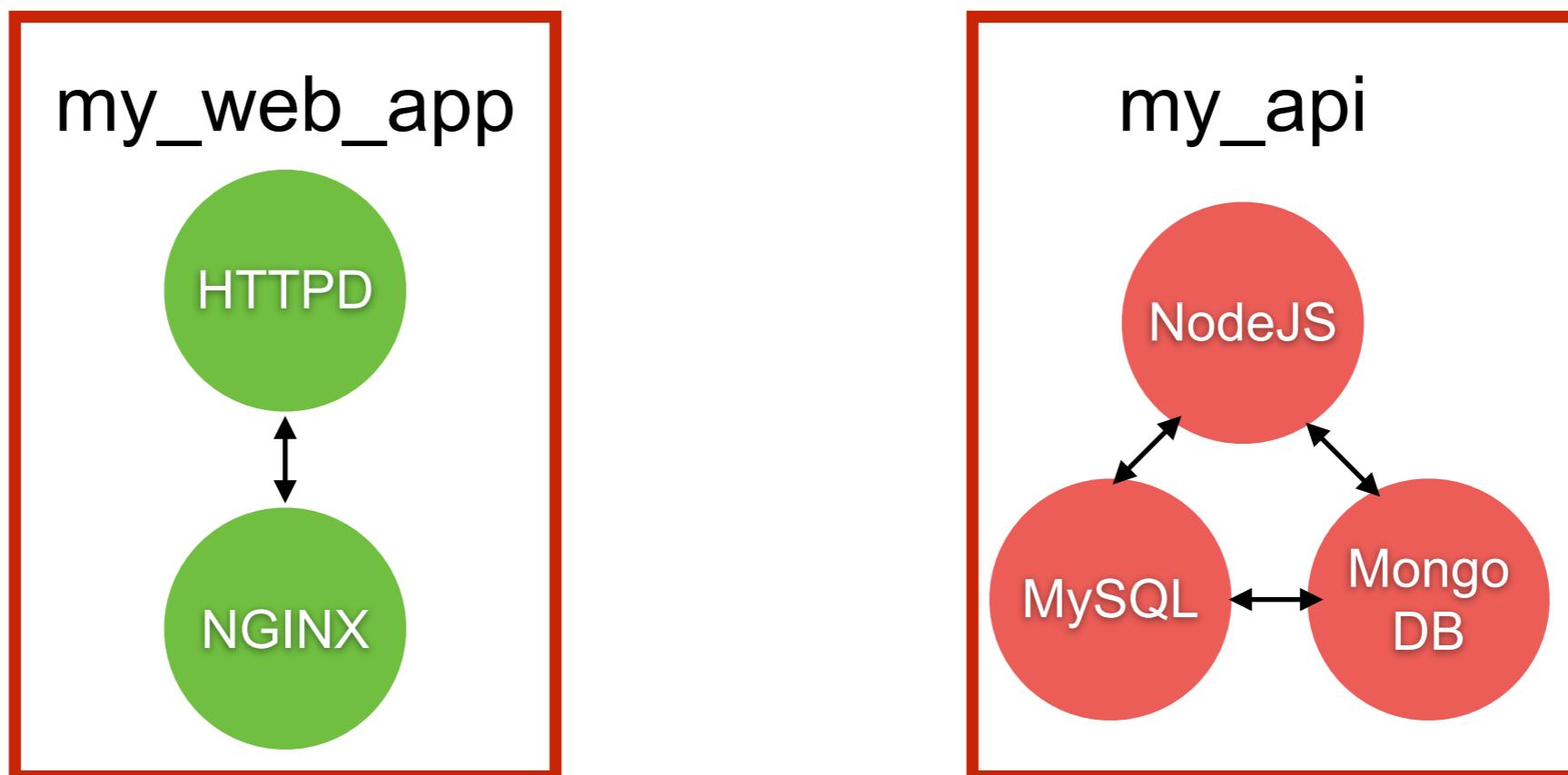
All containers on virtual network can talk to each other

Attach containers to more than one virtual network



Default Networking

Better solution, must create a new virtual network for each app/container



Skip virtual network

To use HOST IP

--net=host

--network host

Only on Linux host !!

<https://docs.docker.com/network/host/>



Disable network !!

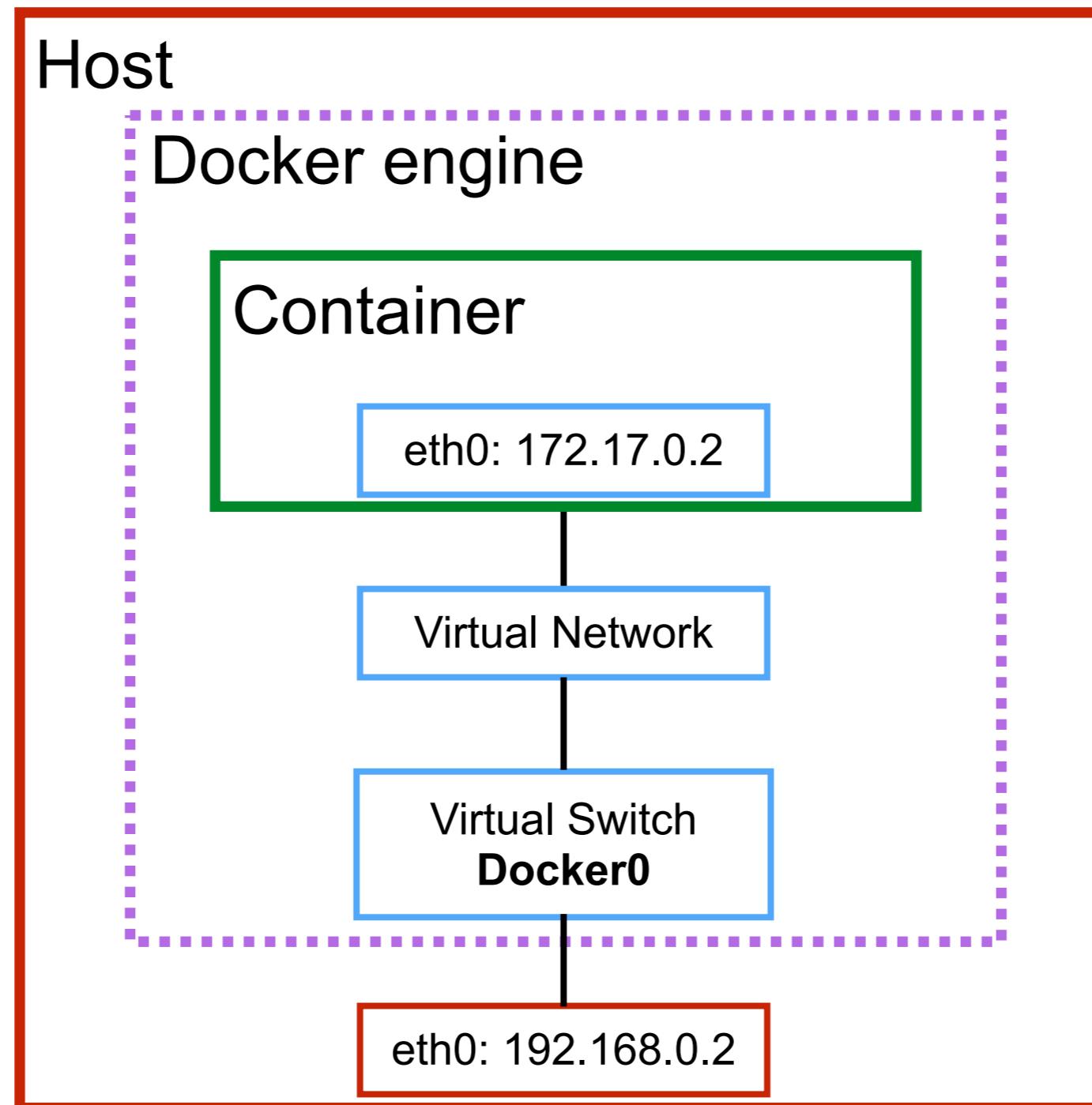
Use **--network none**

<https://docs.docker.com/network/none/>



Default Network

Use network bridge, Auto IP address



Bridge network

\$docker network inspect bridge

```
"Name": "bridge",
"Id": "634697547d0adb6f38f78f574aed8f91fe5818e7fc7bdf7b1dba142a5f7817bd",
"Created": "2020-05-28T03:02:21.079537505Z",
"Scope": "local",
"Driver": "bridge",
"EnableIPv6": false,
"IPAM": {
    "Driver": "default",
    "Options": null,
    "Config": [
        {
            "Subnet": "172.17.0.0/16",
            "Gateway": "172.17.0.1"
        }
    ]
},
"Internal": false,
"Attachable": false,
"Ingress": false,
"ConfigFrom": {
    "Network": ""
},
"ConfigOnly": false,
"Containers": {},
"Options": {
    "com.docker.network.bridge.default_bridge": "true",
    "com.docker.network.bridge.enable_icc": "true",
    "com.docker.network.bridge.enable_ip_masquerade": "true",
    "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
    "com.docker.network.bridge.name": "docker0",
    "com.docker.network.driver.mtu": "1500"
},
```

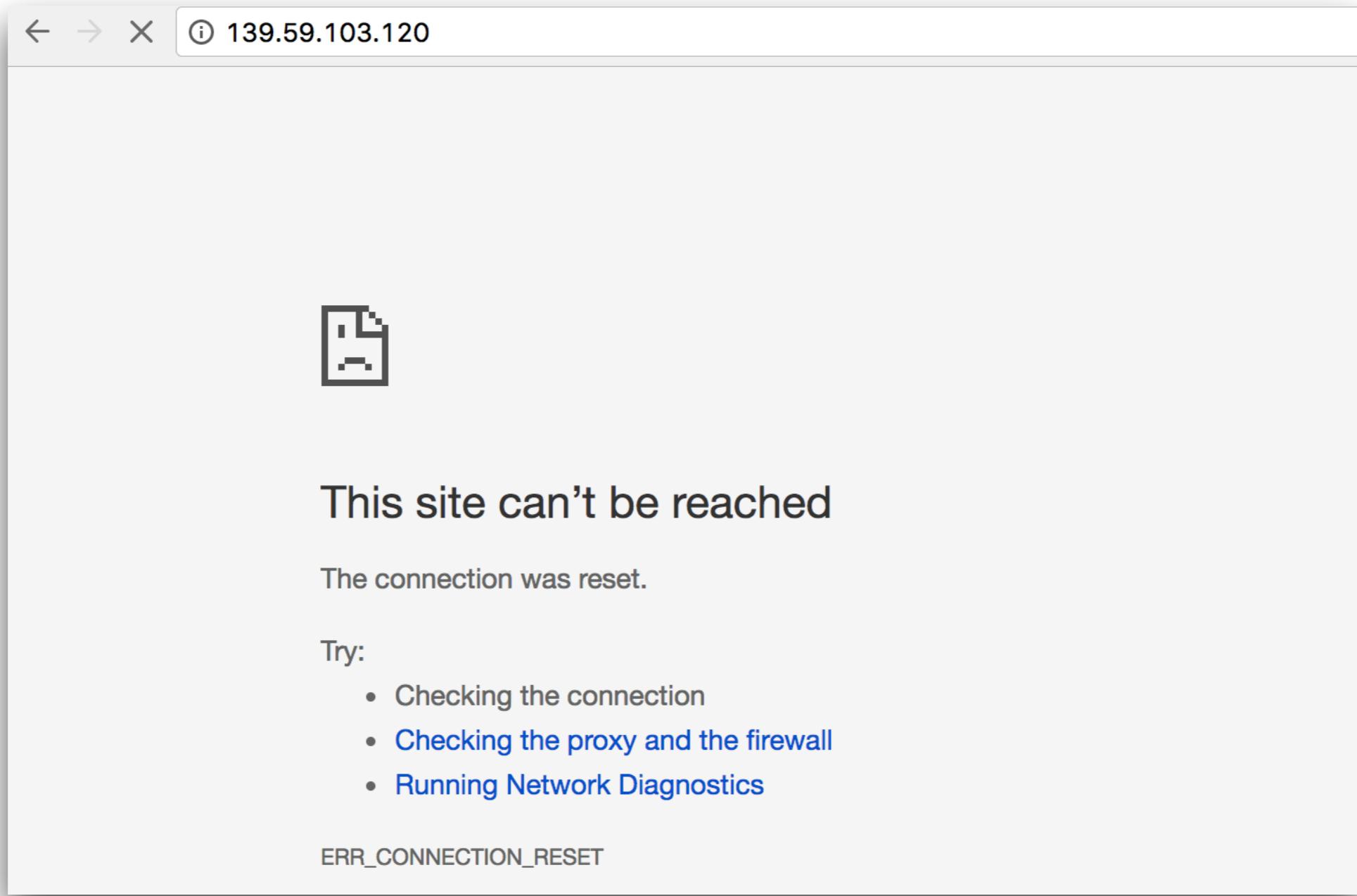


Start nginx with name

```
$ docker container run \  
  --name hello-nginx \  
  -d nginx
```



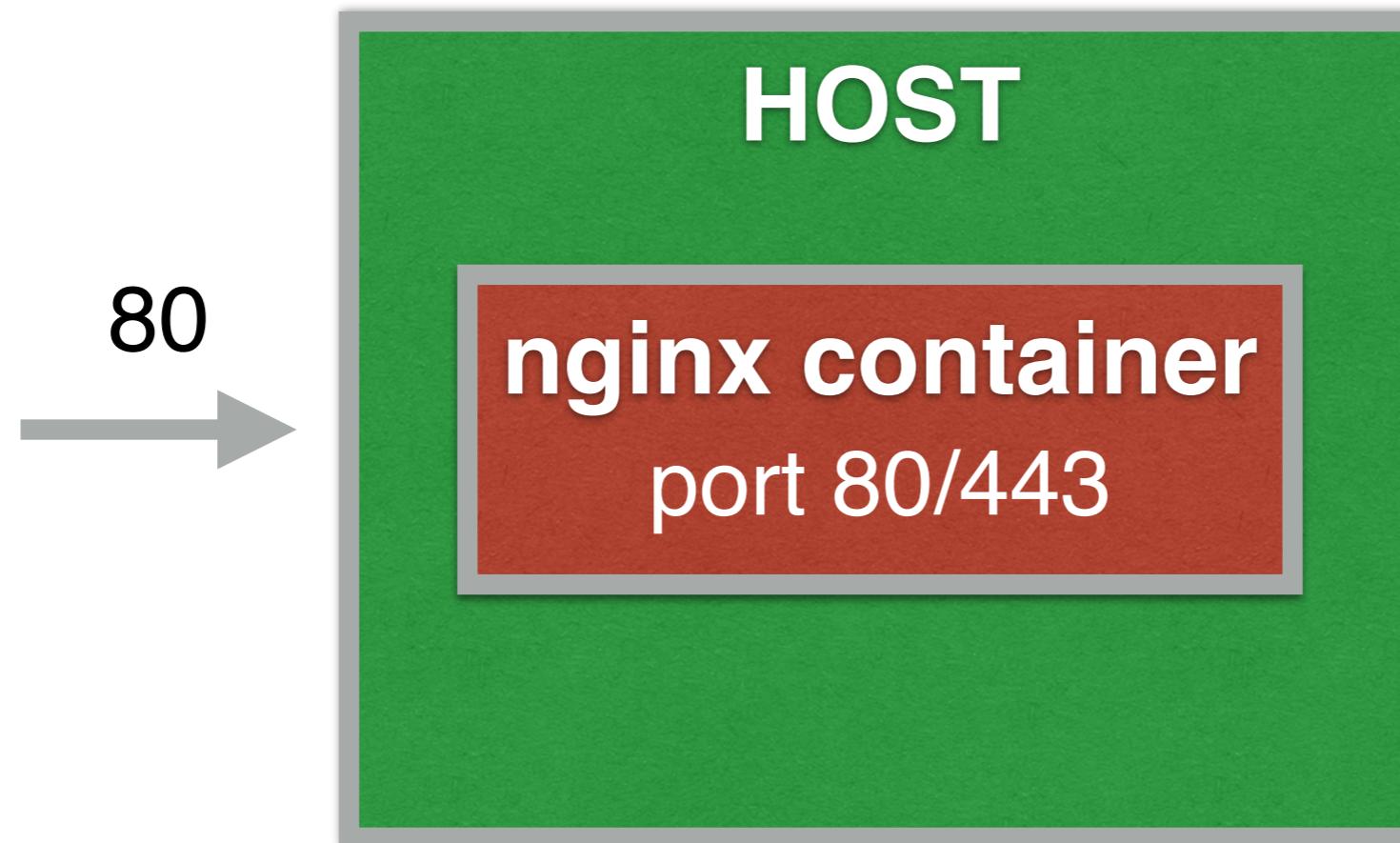
Try to access with port 80



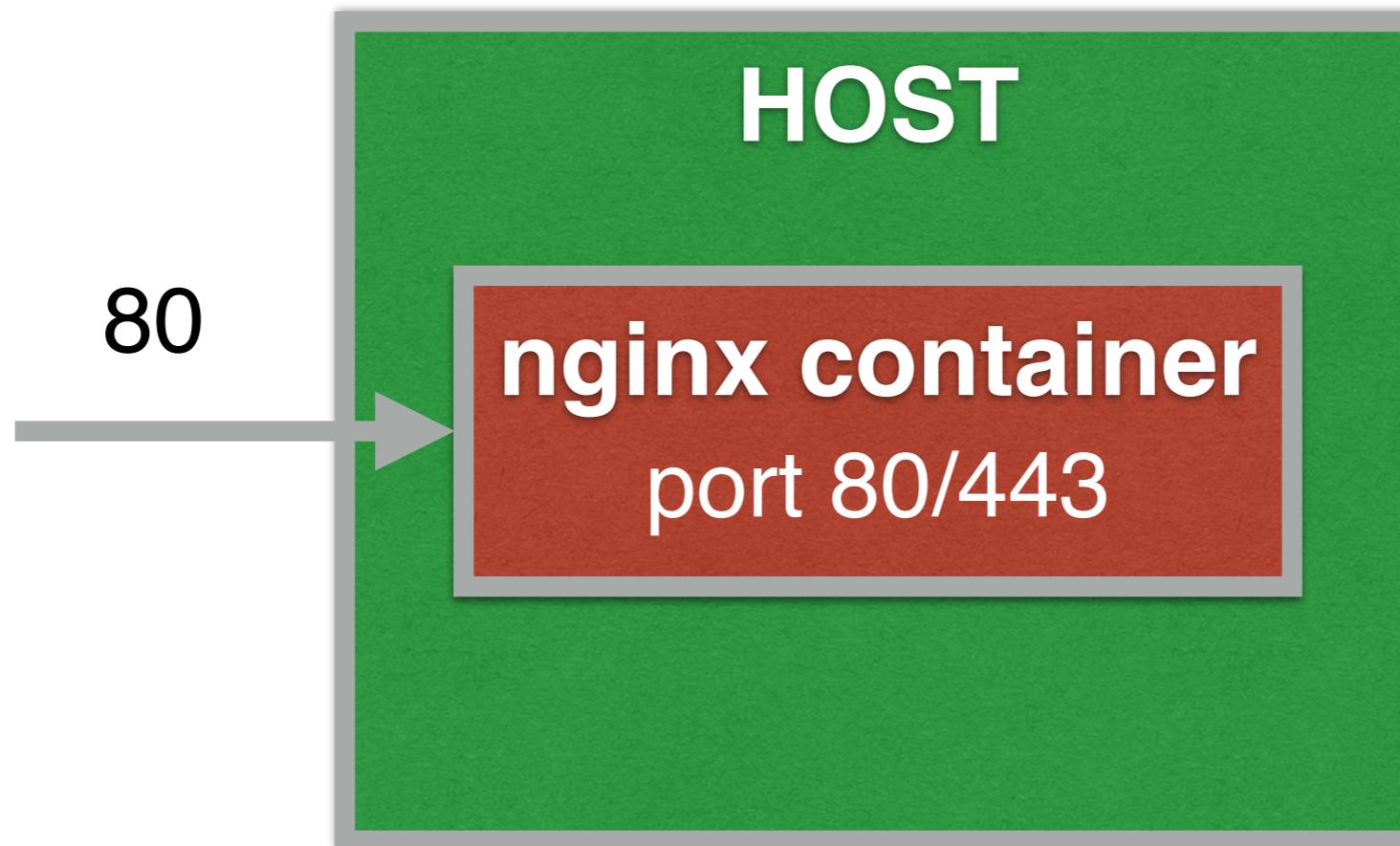
Why not ?



Port of Host and Container



Port of Host and Container



Publish all port

```
$ docker container run -d --name hello-nginx
```

-P nginx
--publish-all



See result

\$docker ps -l

NAMES	PORTS
test01	0.0.0.0:32774->80/tcp, 0.0.0.0:32773->443/tcp

Web server is running on port **80** inside the container

Port 80 is exposed on port **32774** on docker host



Custom result format

```
$ docker ps --format 'table {{.Names}}\t{{.Ports}}' -l
```

NAMES	PORTS
test01	0.0.0.0:32774->80/tcp, 0.0.0.0:32773->443/tcp

<https://docs.docker.com/engine/admin/formatting/>



Try to access



A screenshot of a web browser window. The address bar shows the URL `localhost:32774`. The main content area displays the Nginx welcome page with the heading **Welcome to nginx!**, a message about successful installation, and links for documentation and support.

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.



Publish by port

```
$ docker container run --name hello-nginx -d
```



See result

```
$docker ps --format 'table {{.Names}}\t{{.Ports}}' -l
```

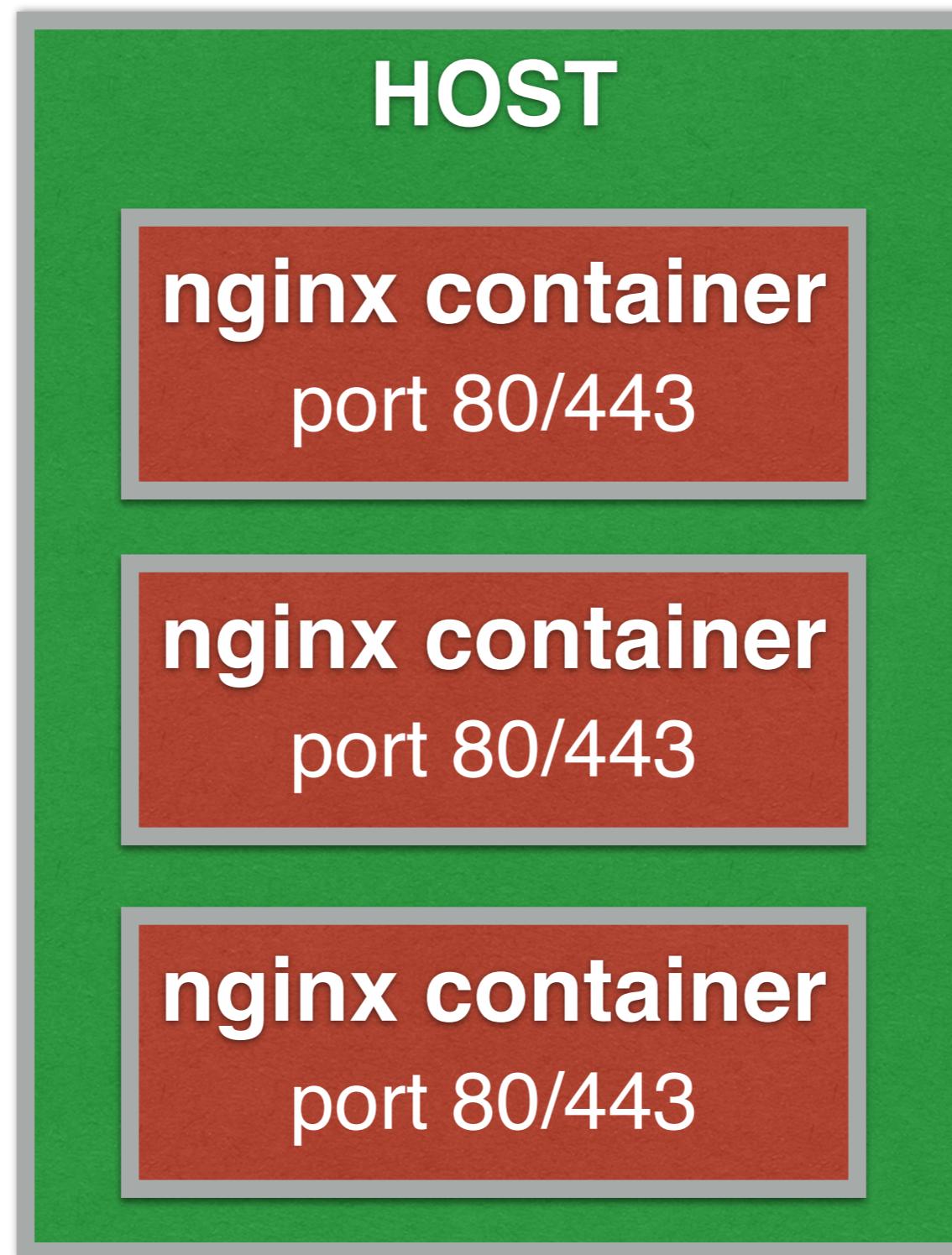
NAMES	PORTS
test01	0.0.0.0:80->80/tcp, 443/tcp



Try to access with port 80



More containers



Find port in each container

\$docker container port <id/name>



Connect to web server

\$curl localhost



Find IP's container

\$docker container inspect <id>

```
"NetworkSettings": {  
    "Bridge": "",  
    "SandboxID": "cd7aeeccb8020cb334f0ccd0601230f9927be15c88511d9ec0340af281e769eb5",  
    "HairpinMode": false,  
    "LinkLocalIPv6Address": "",  
    "LinkLocalIPv6PrefixLen": 0,  
    "Ports": [  
        "443/tcp": null,  
        "80/tcp": [  
            {"  
                "HostIp": "0.0.0.0",  
                "HostPort": "80"  
            }  
        ]  
    ],  
    "SandboxKey": "/var/run/docker/netns/cd7aeeccb8020",  
    "SecondaryIPAddresses": null,  
    "SecondaryIPv6Addresses": null,  
    "EndpointID": "3e63b14c92309aa0630a9356ca720ba3e0169be0d1128f0b4e3f6f33e7bad192",  
    "Gateway": "172.17.0.1",  
    "GlobalIPv6Address": "",  
    "GlobalIPv6PrefixLen": 0,  
    "IPAddress": "172.17.0.3",  
    "IPPrefixLen": 16,  
    "IPV6Gateway": "",  
    "MacAddress": "02:42:ac:11:00:03",  
    "Networks": {  
        "bridge": {  
            "IPAMConfig": null,  
            "Links": null,  
            "Aliases": null  
        }  
    }  
}
```



Find IP's container

```
$docker container inspect \  
--format '{{ .NetworkSettings.IPAddress }}' \  
<id>
```



Ping container with IP

\$ping <IP address>



Command line with network

\$docker network ls

\$docker network inspect

\$docker network create --driver

\$docker network connect

\$docker network disconnect



Docker network command

<https://docs.docker.com/engine/userguide/networking>



List all network

\$docker **network ls**

NETWORK ID	NAME	DRIVER	SCOPE
78054ed84305	bridge	bridge	local
2f402f2f8b98	host	host	local
43a5c01d994b	none	null	local



Create network

\$docker network **create** <name>

\$docker network inspect <name>



Connect network to container

```
$ docker network connect <name>  
      <container>
```



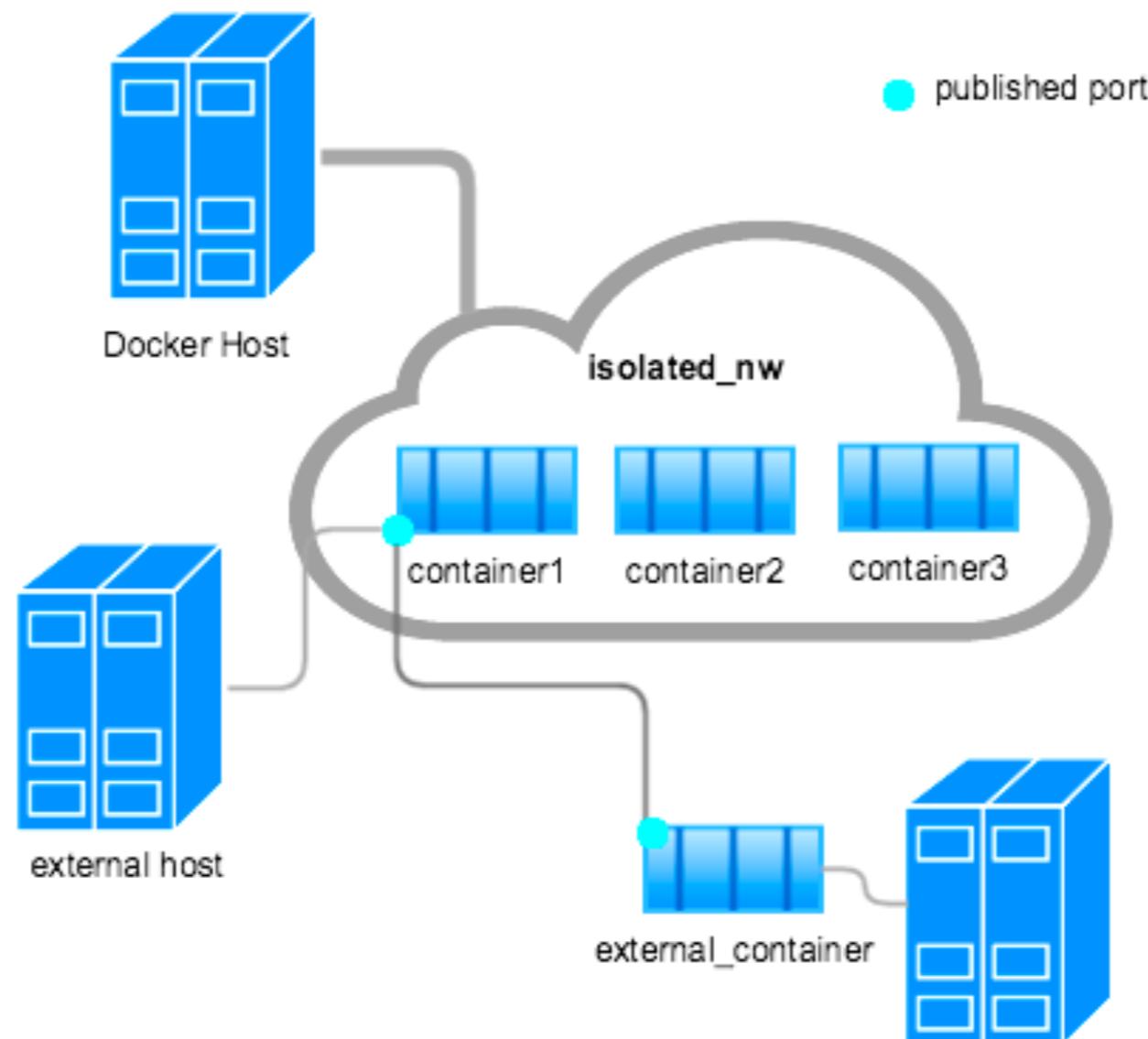
Remove network

\$docker network **rm** <name>

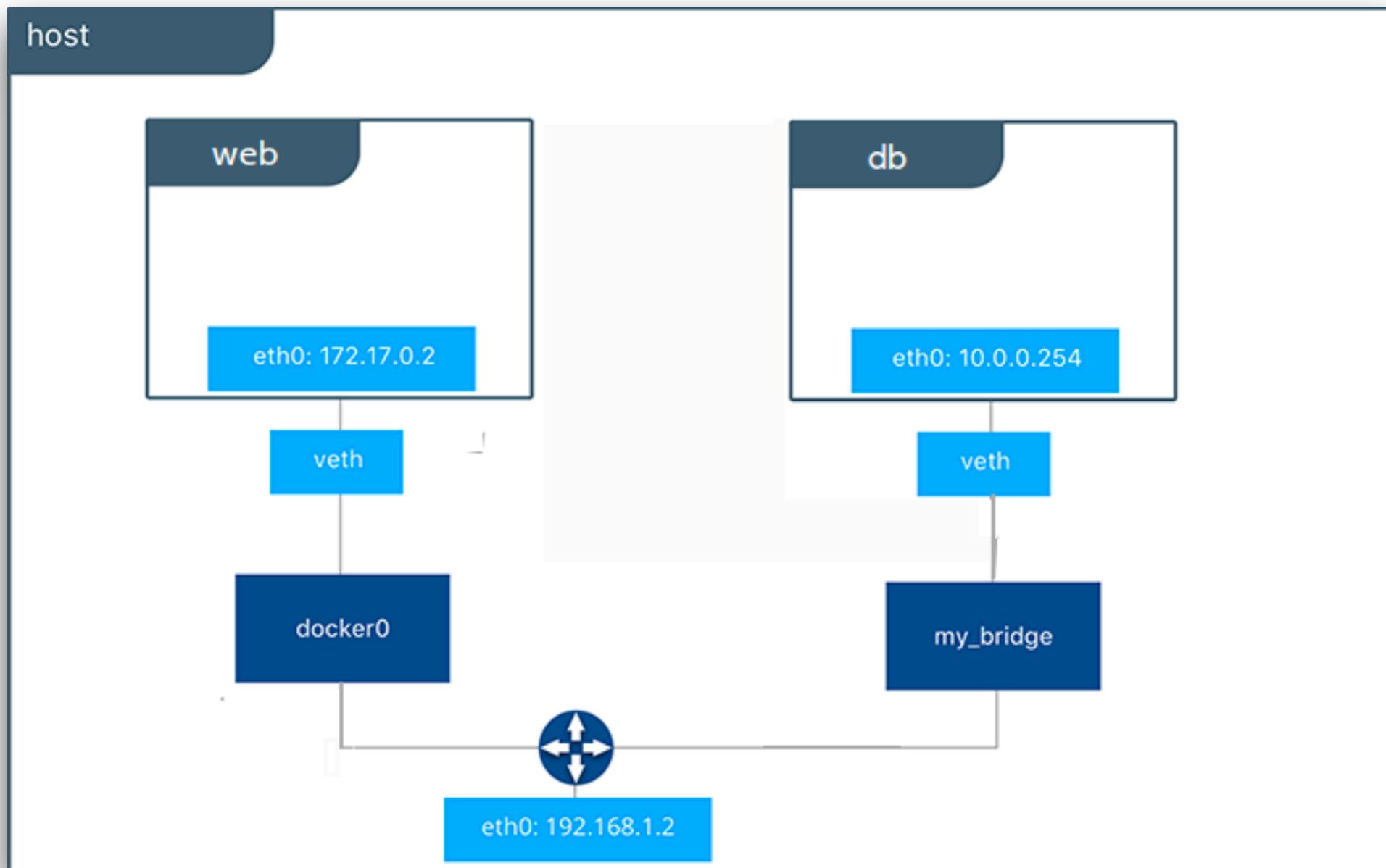
\$docker network prune



Isolated network



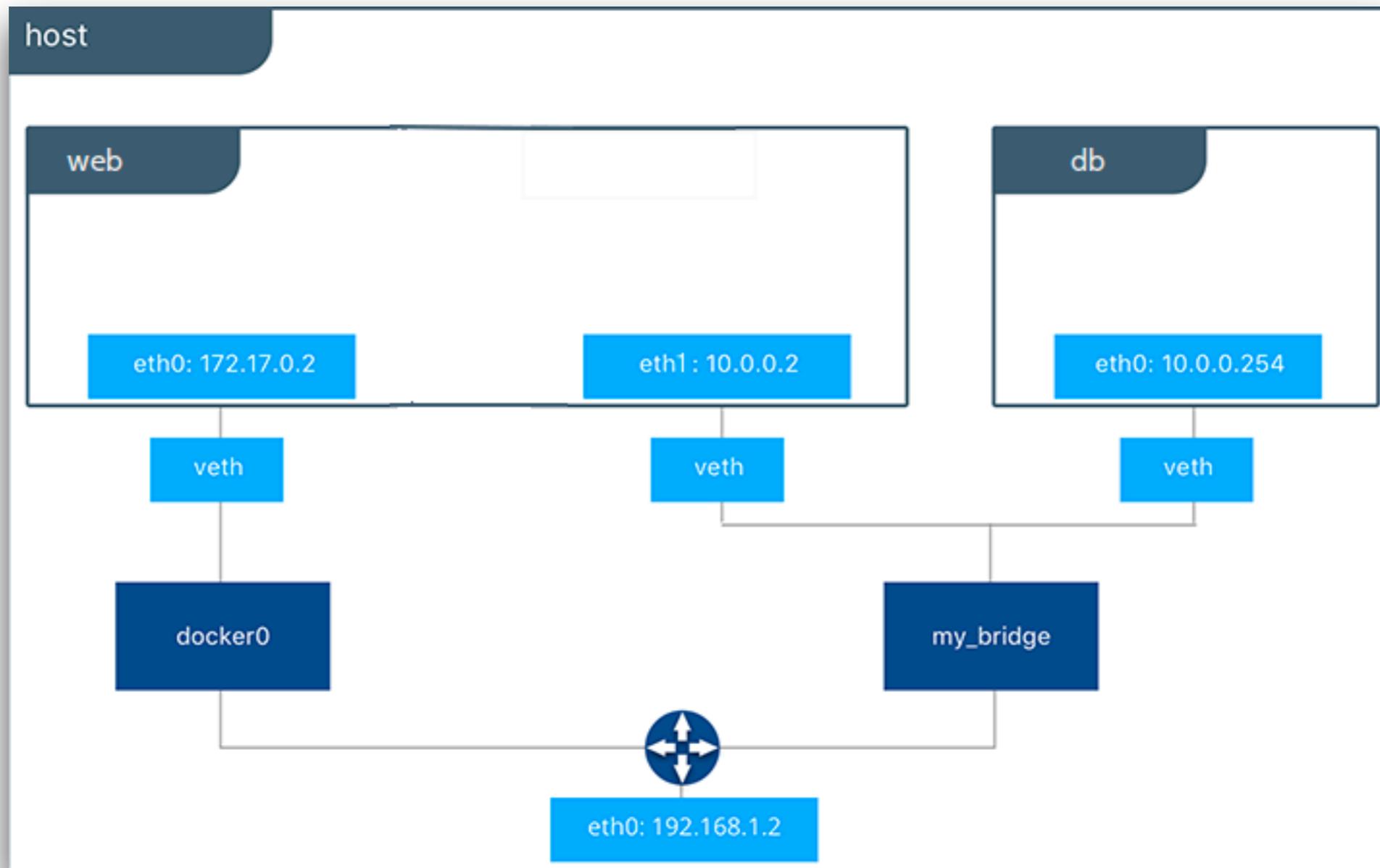
Isolated network of container



<https://docs.docker.com/engine/tutorials/networkingcontainers/>



Add network to container



Default Security

Manage your app/container with the new virtual network

Inter-communication between containers must never leave host

All externally exposed ports closed by default



DNS (Domain Name System)

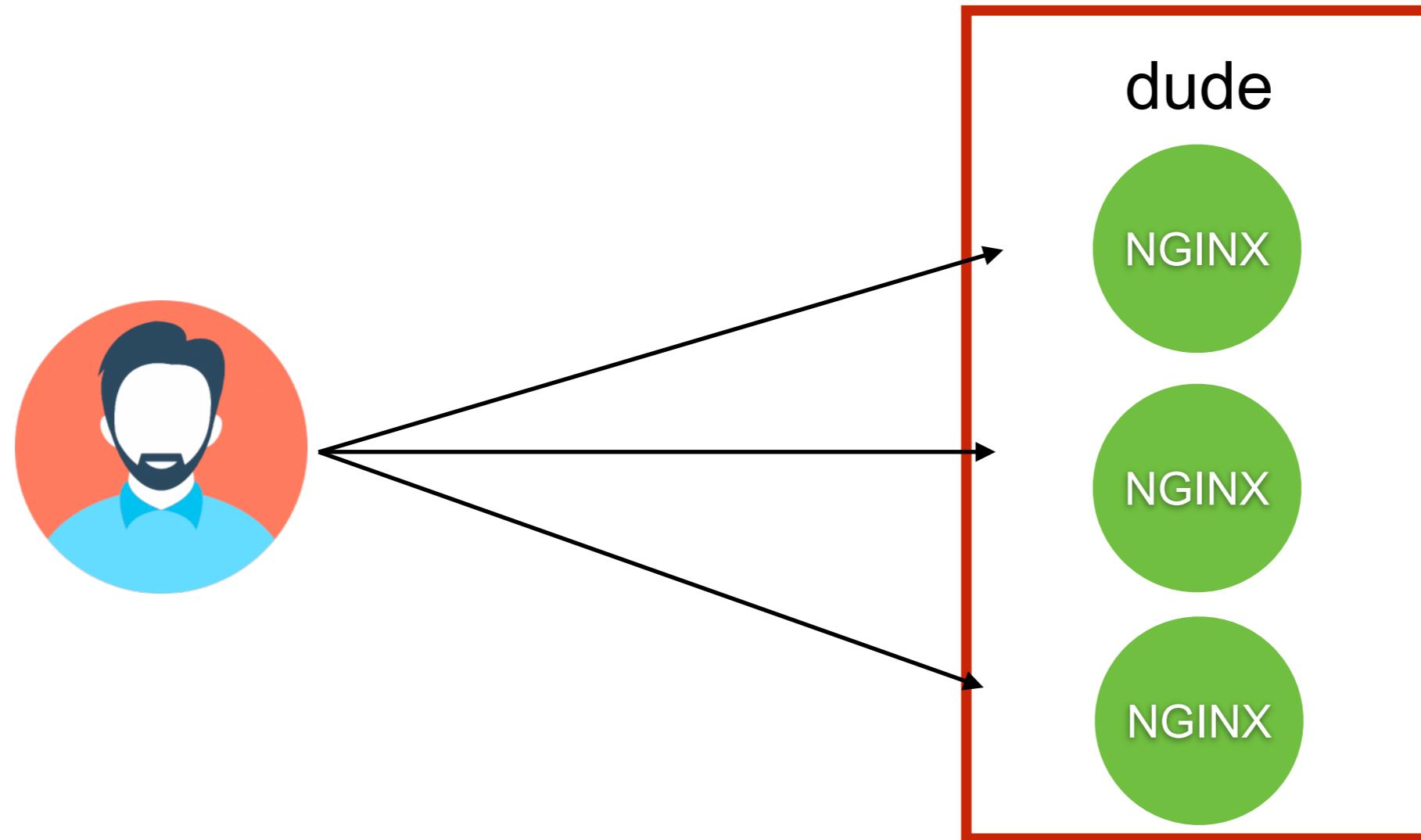
DNS is the key to make easy to communication

Use **--link** to enable DNS on default bridge network

Create new network is enabled DNS by default

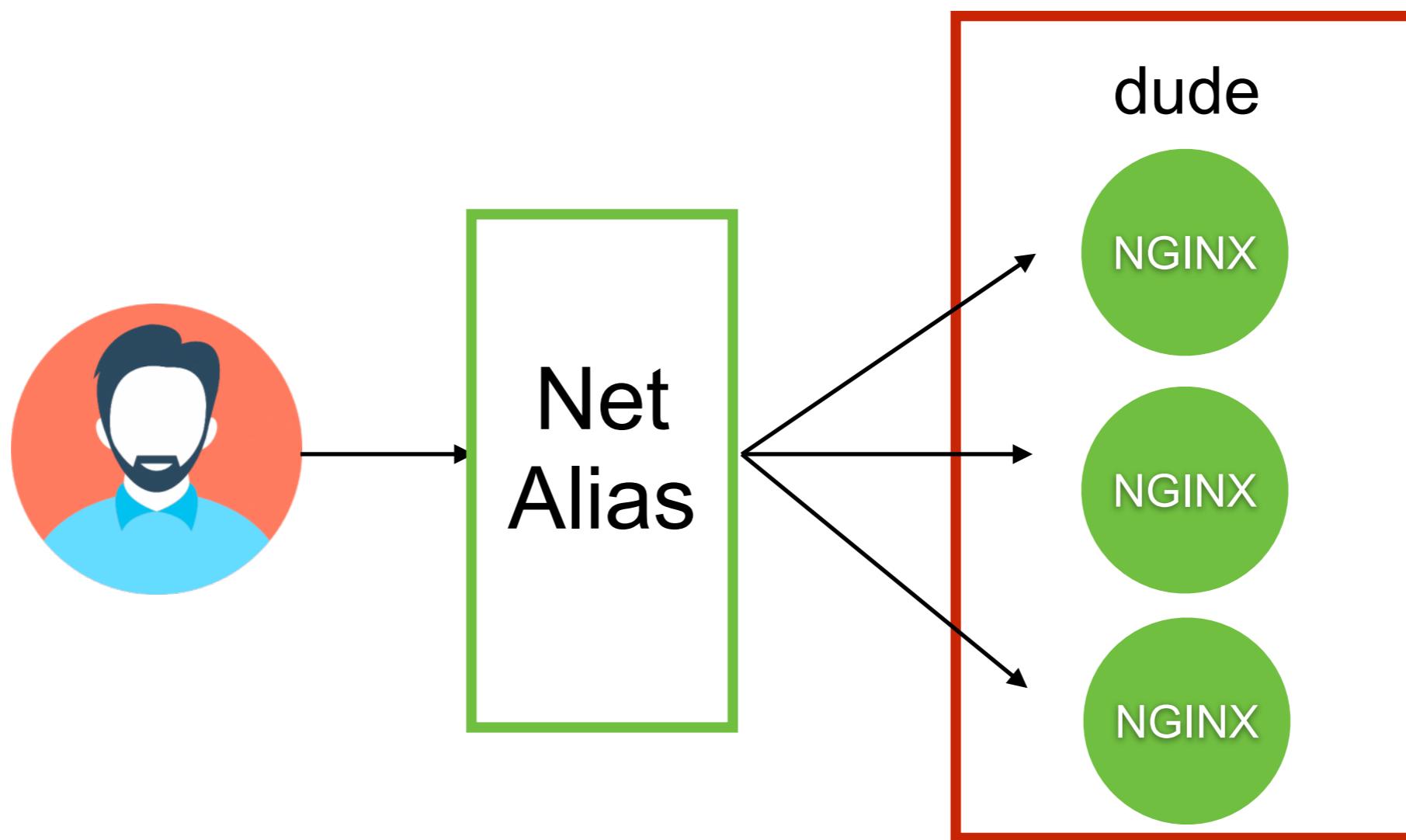


DNS Round Robin Testing ?



DNS Round Robin Testing

```
$docker container run -d --net dude  
--net-alias search nginx:alpine
```

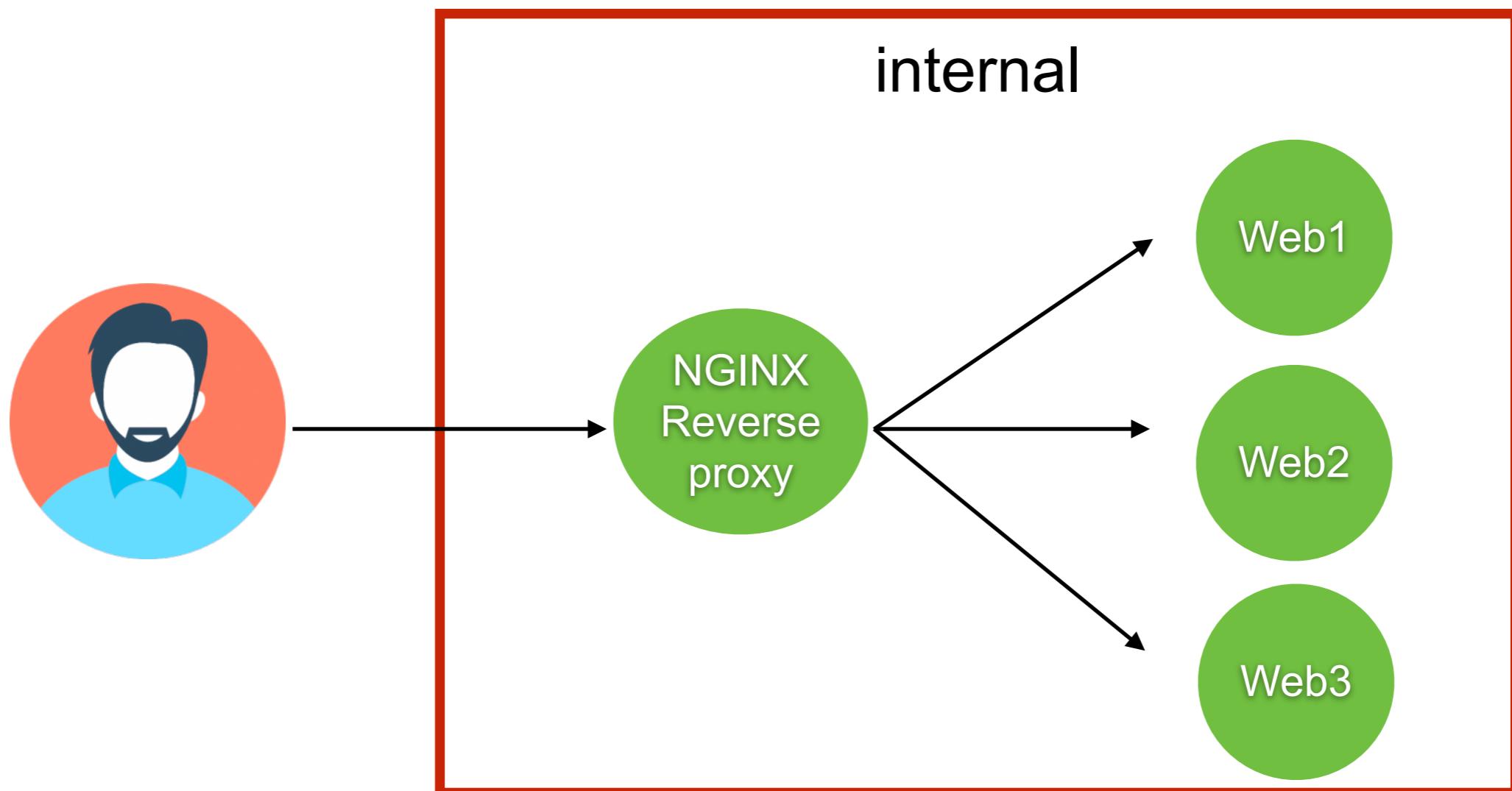


<https://docs.docker.com/engine/reference/commandline/run/>



Workshop

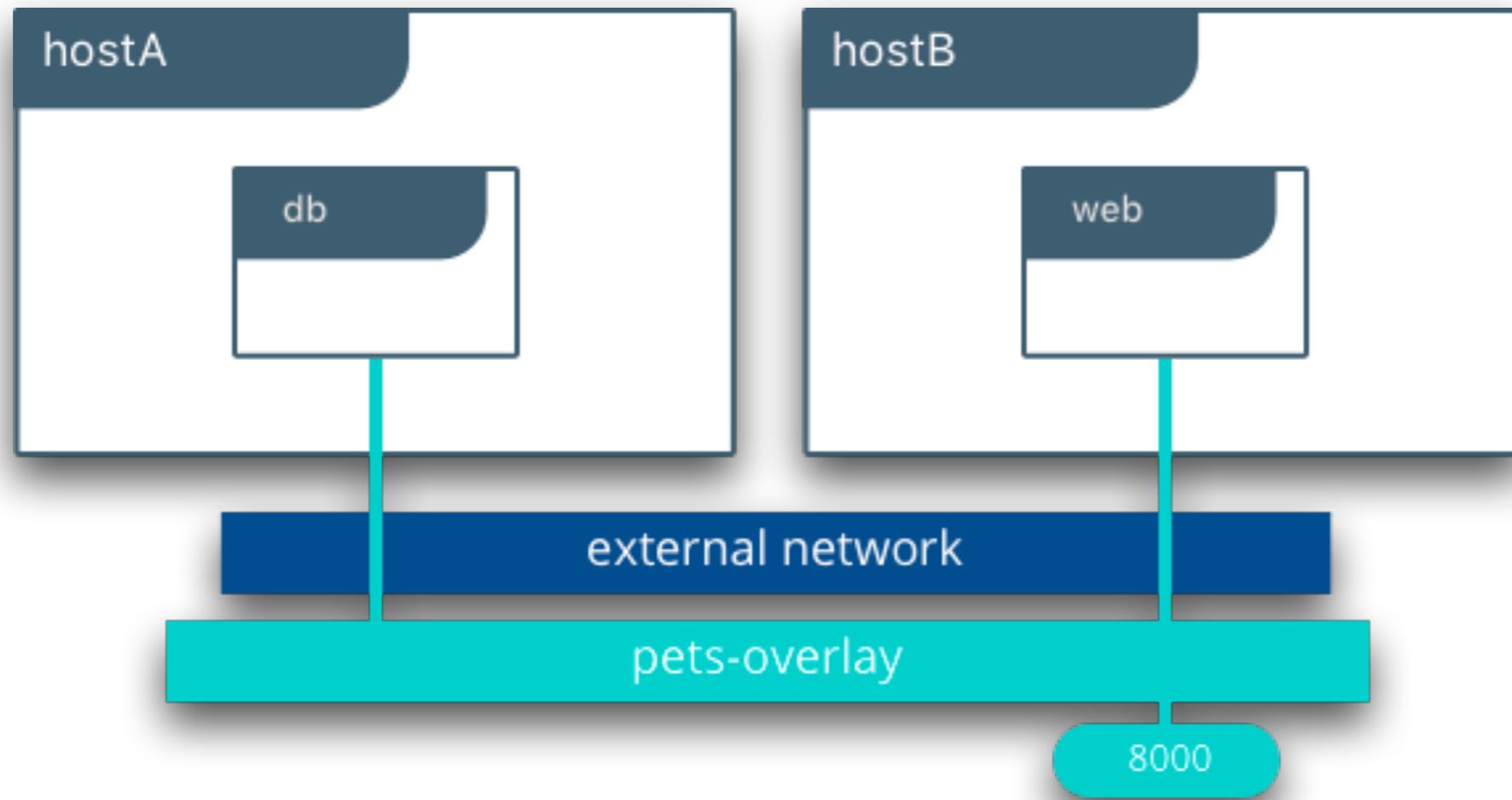
Try to create container and network



More Network Driver



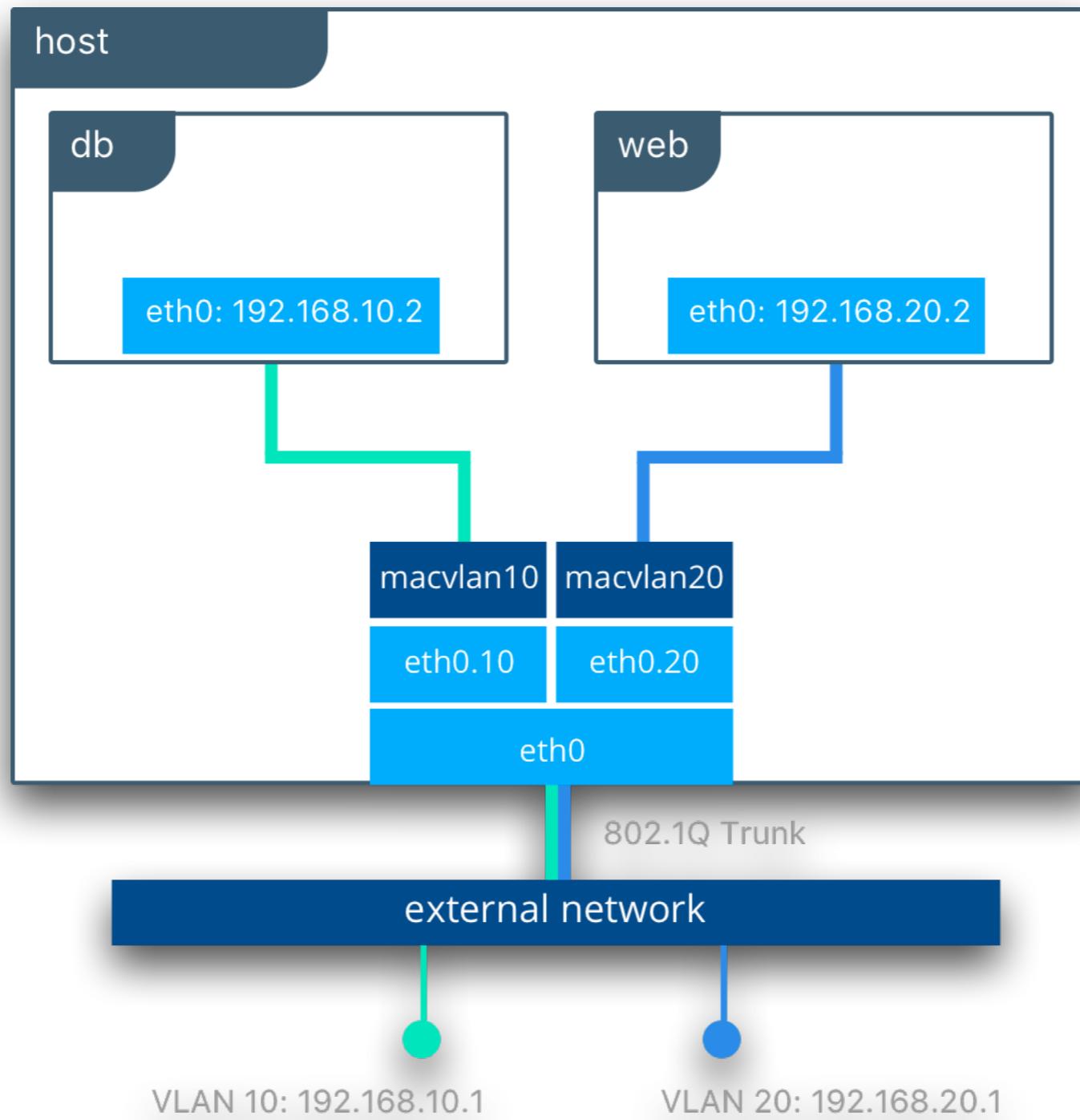
Overlay network driver



<https://docs.docker.com/network/overlay/>



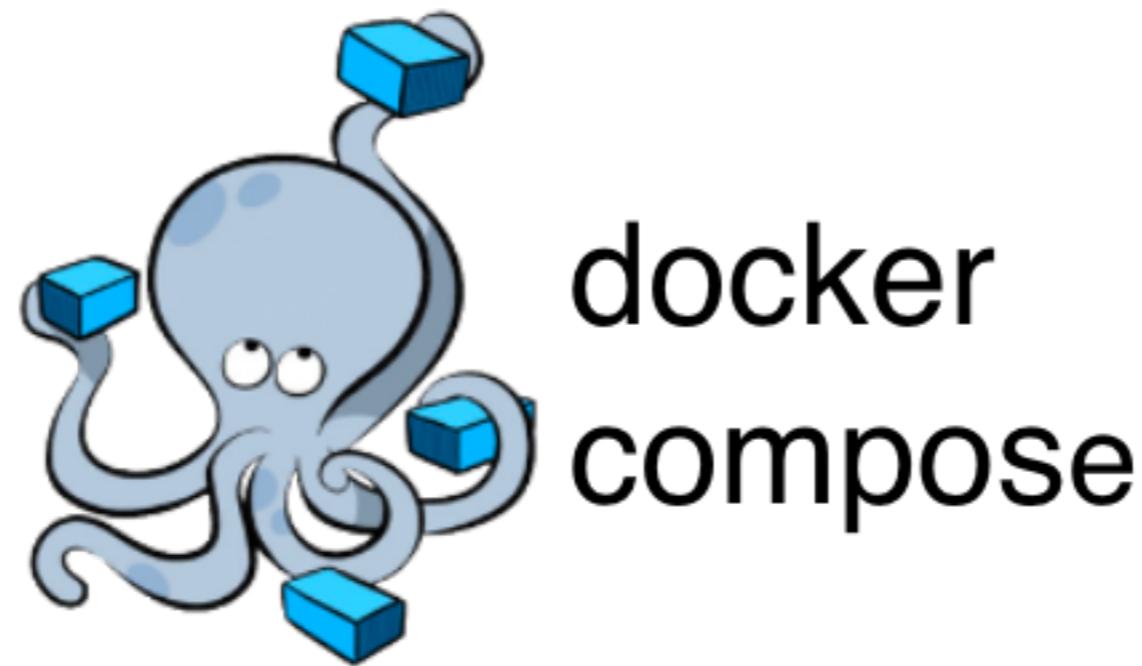
MACVLAN driver



<https://docs.docker.com/network/macvlan/>



Easier with Docker compose !!



Docker Storage

<https://docs.docker.com/storage/>



Docker Storage

Volumes

Bind mounts

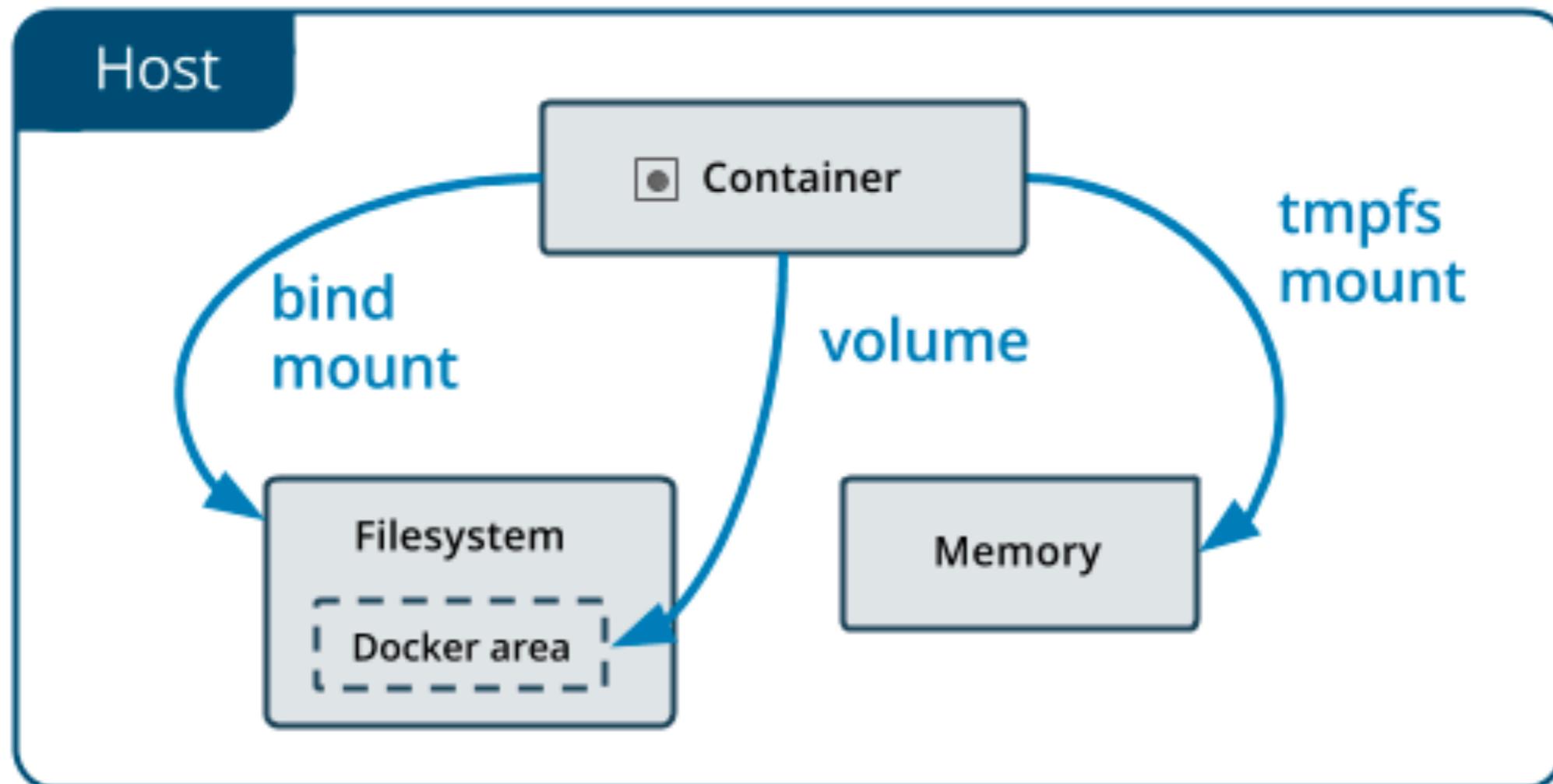
tmpfs mounts

Named pipes (from Microsoft)

<https://docs.docker.com/storage/>



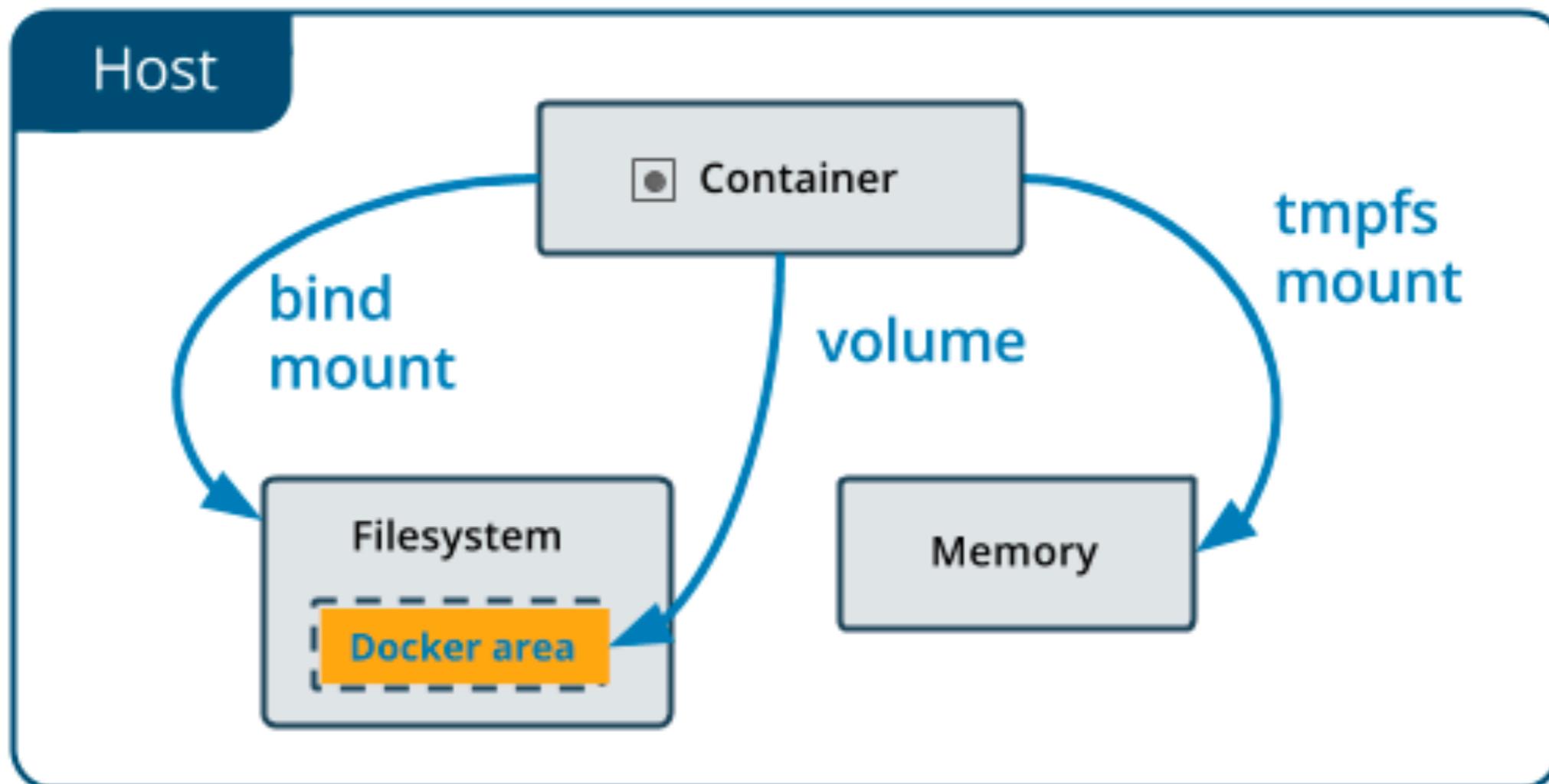
Docker Storage



<https://docs.docker.com/storage/>



Volumes



Volumes

Store data in a part of Host file system

/var/lib/docker/volumes on Linux

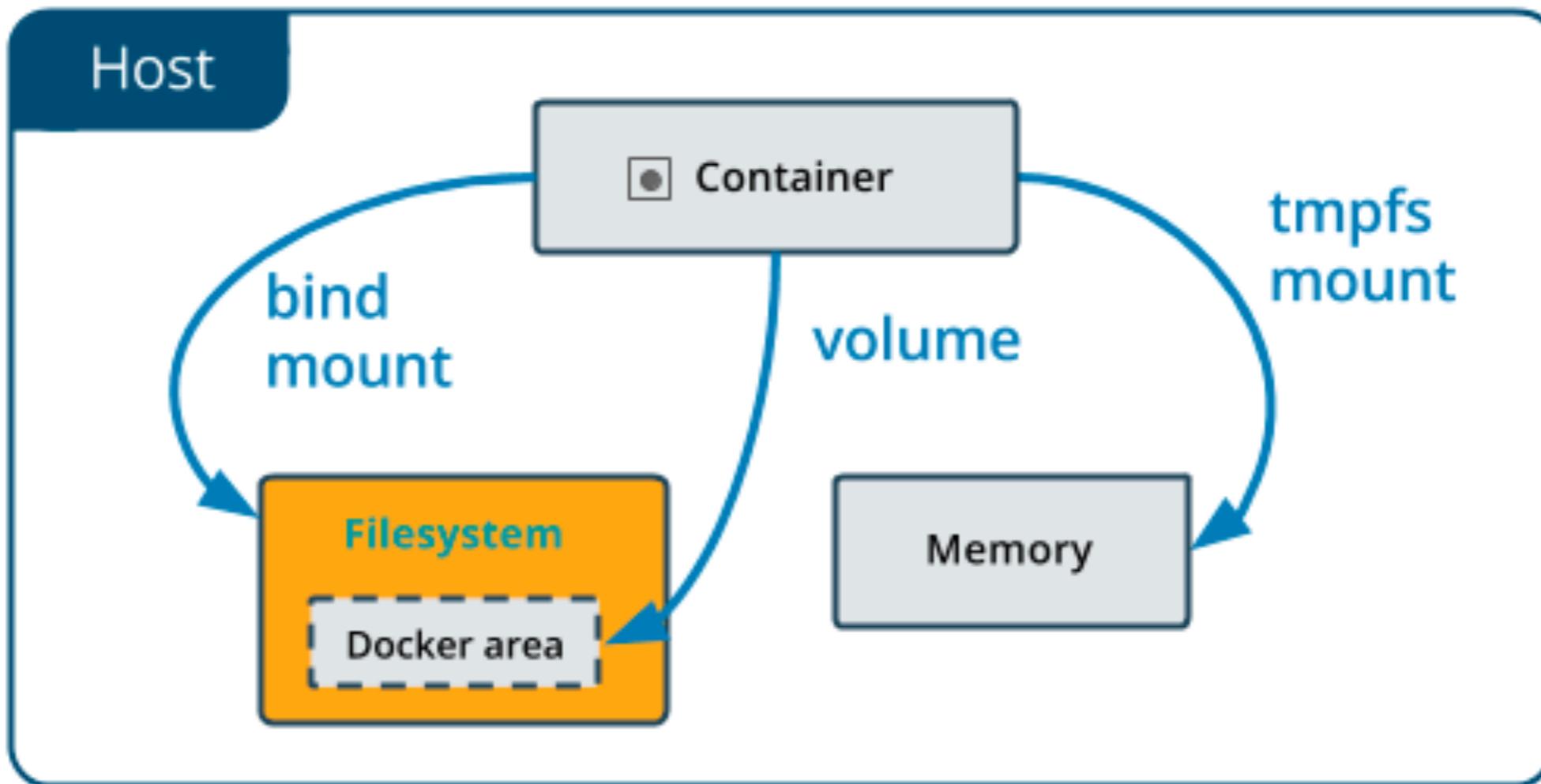
Can use Docker Volume Driver ...

Using -v, --mount in \$docker container

https://docs.docker.com/engine/extend/legacy_plugins/#volume-plugins



Bind mounts



Bind mounts

Store data on Host system

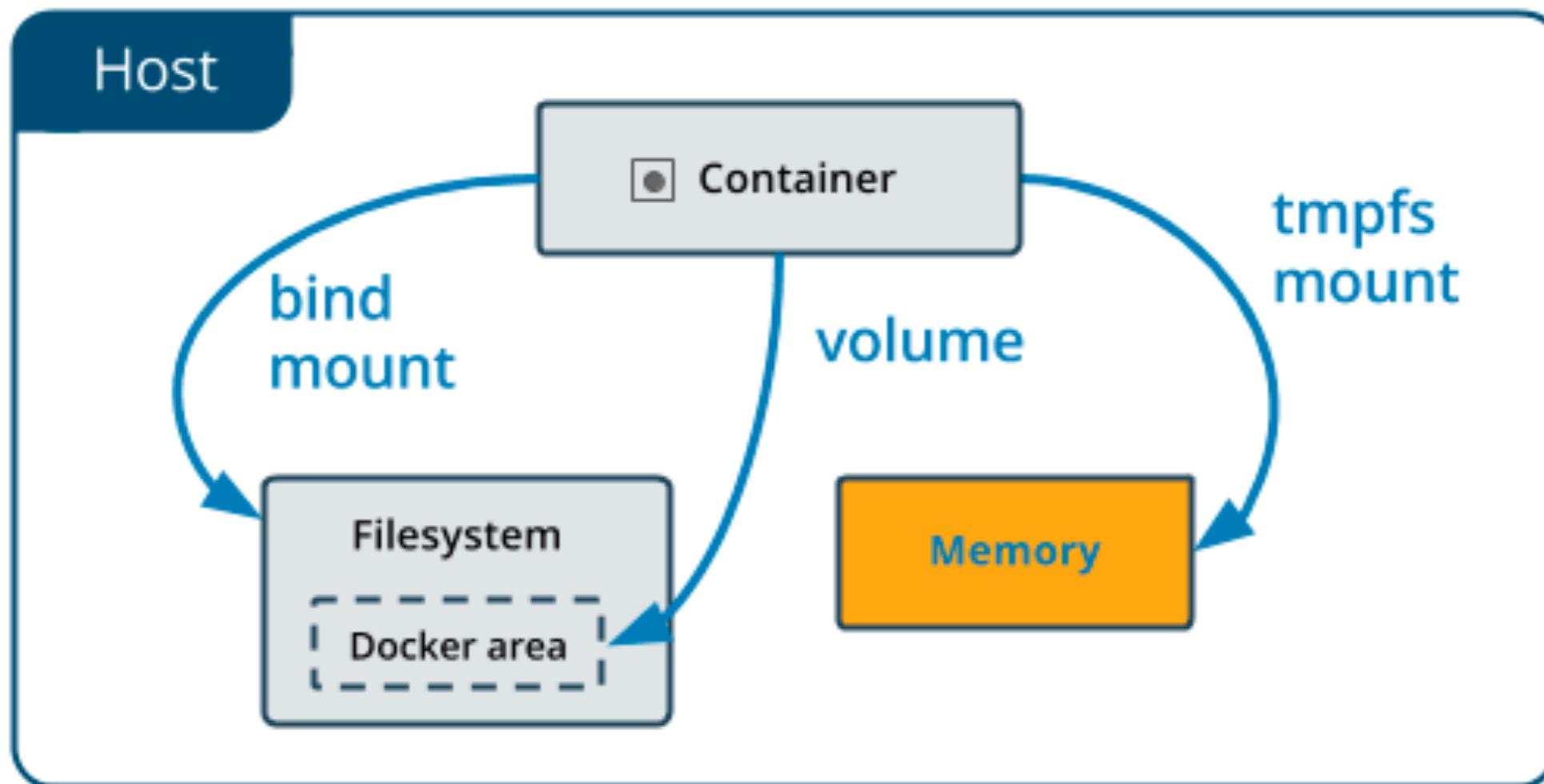
Use both files and directories

Both Host and Docker container can modify

Using -v, --mount in \$docker container



tmpfs mounts



tmpfs mounts

Store data in the Host's memory only
Never write on Host's file system

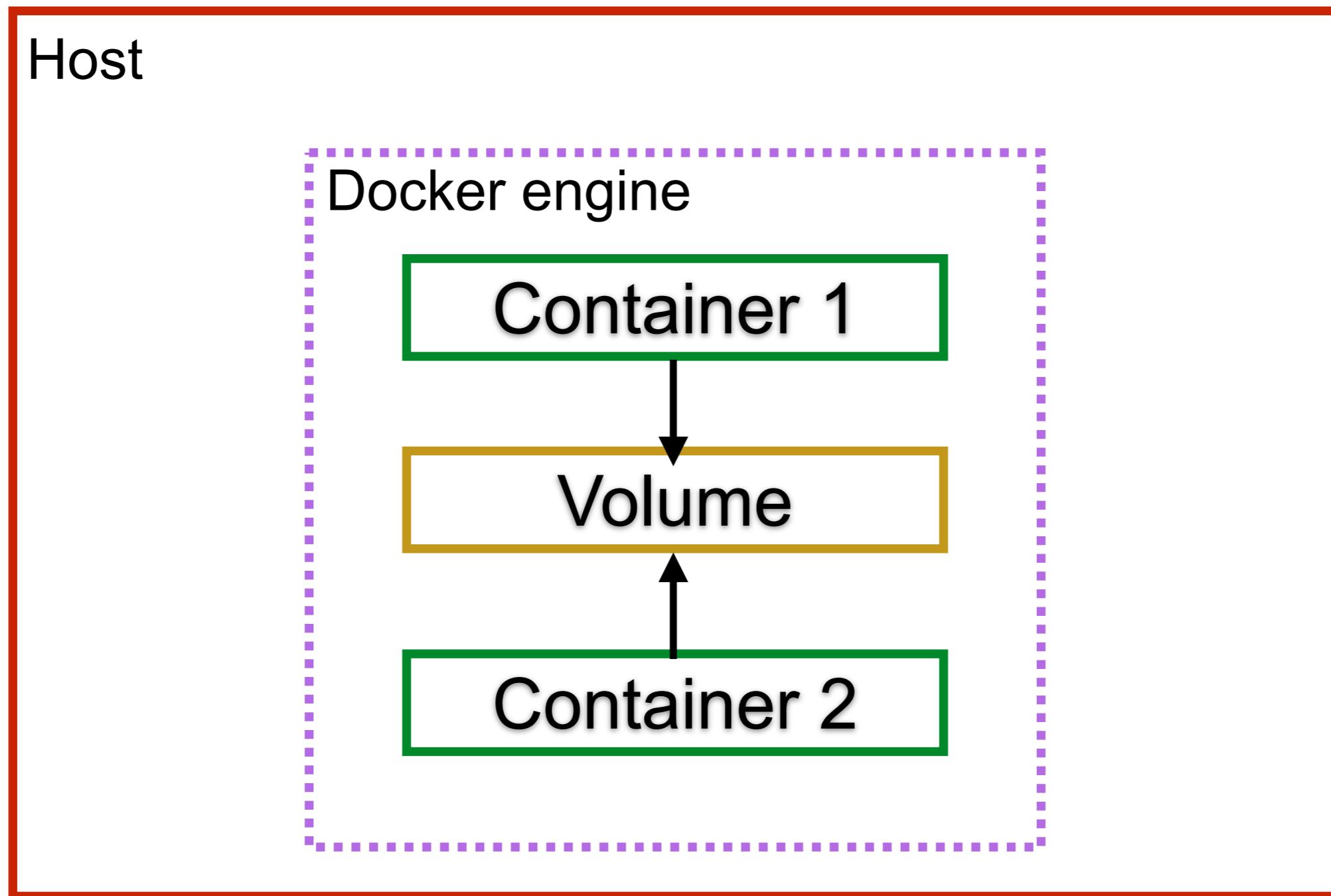
Using -v, --mount in \$docker container



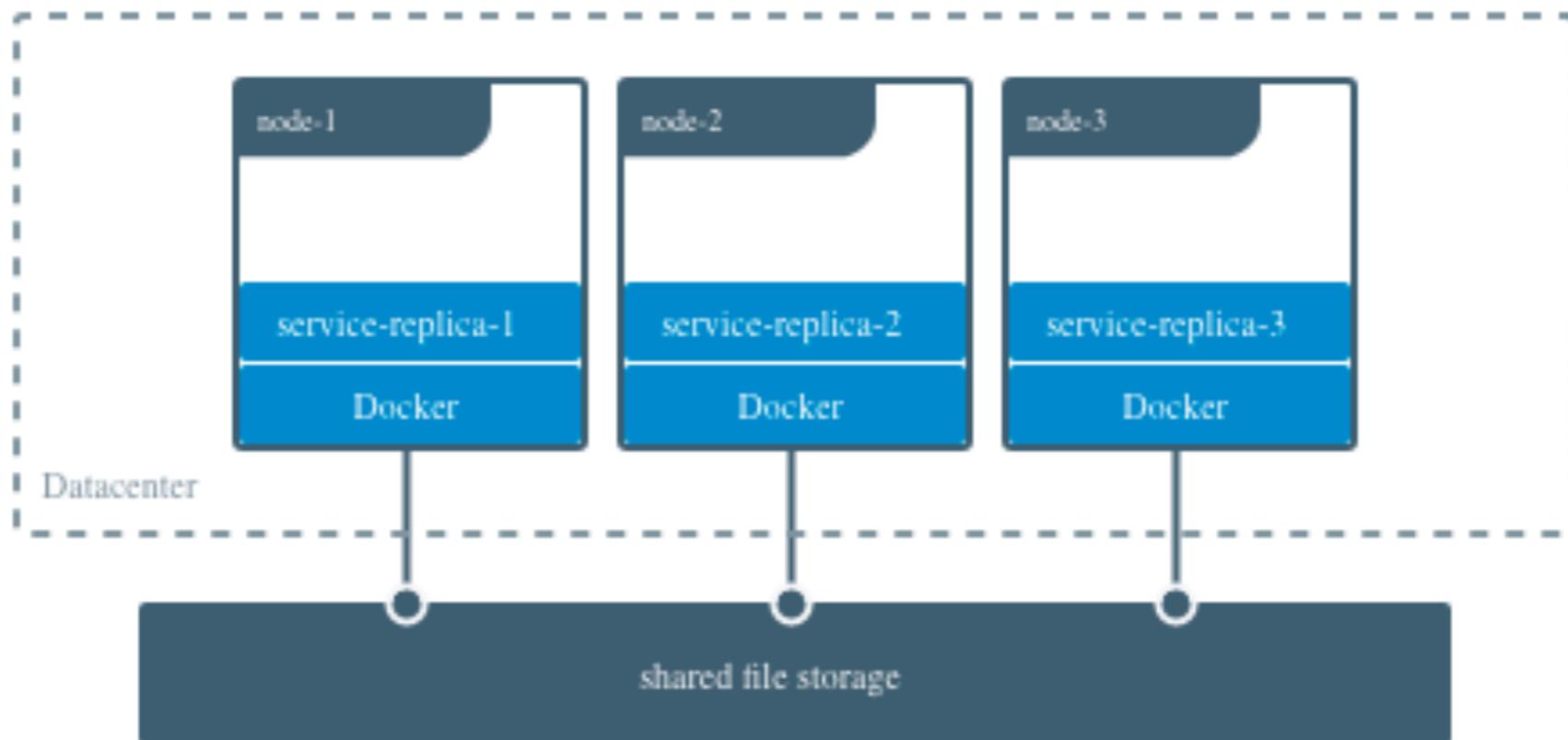
Workshop



Share data between container



Share data between machines

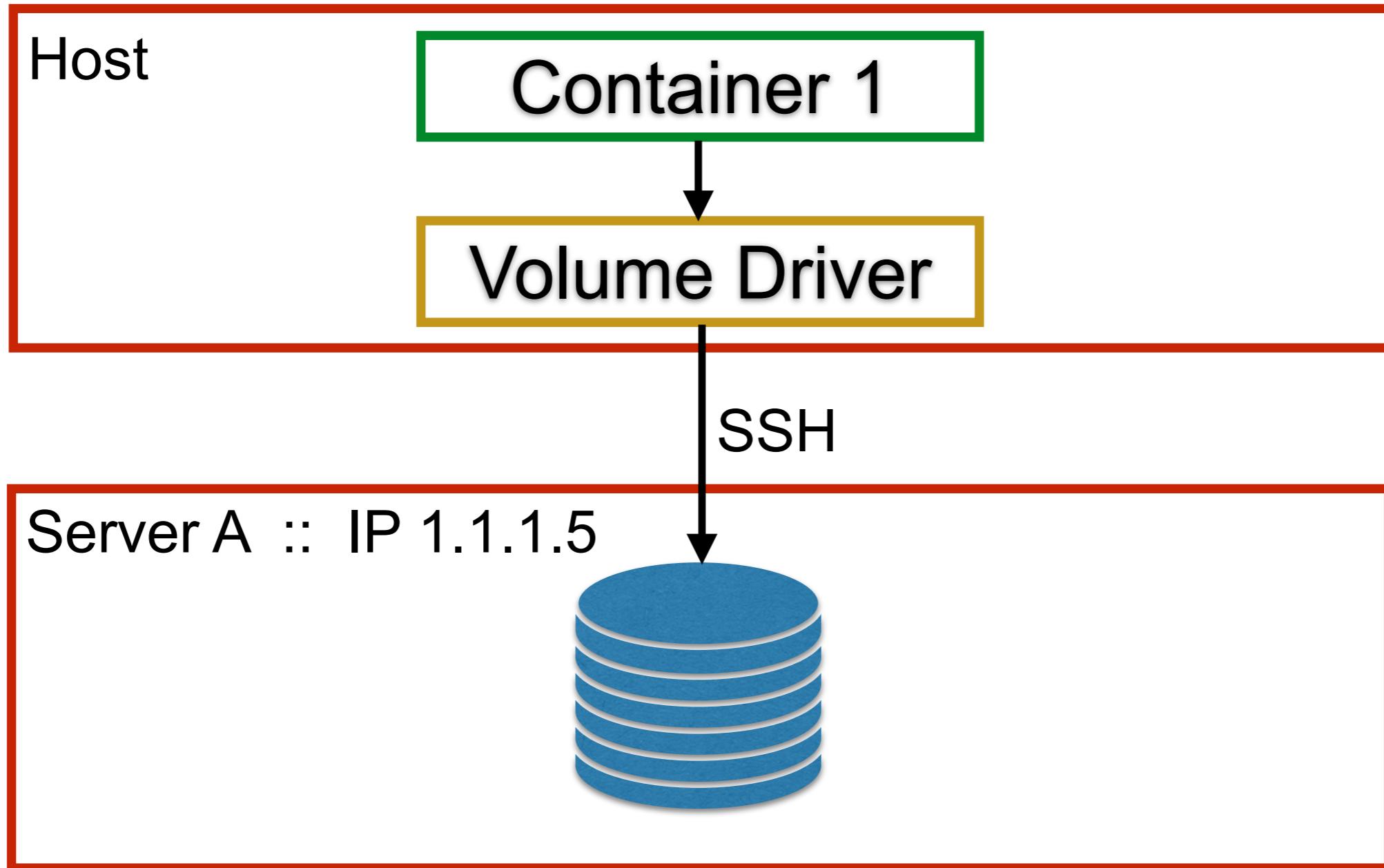


Using volume driver

<https://docs.docker.com/storage/volumes/#share-data-among-machines>



Example with sshfs



<https://docs.docker.com/storage/volumes/#use-a-volume-driver>



Working with Volumes

<https://docs.docker.com/engine/tutorials/dockervolumes>



Objectives

Create containers holding volumes

Share volumes across containers

Share host file/directory with containers



1. Volumes are special dir in container

1. Dockerfile
2. Use with command line



Dockerfile

VOLUME /var/lib/postgresql

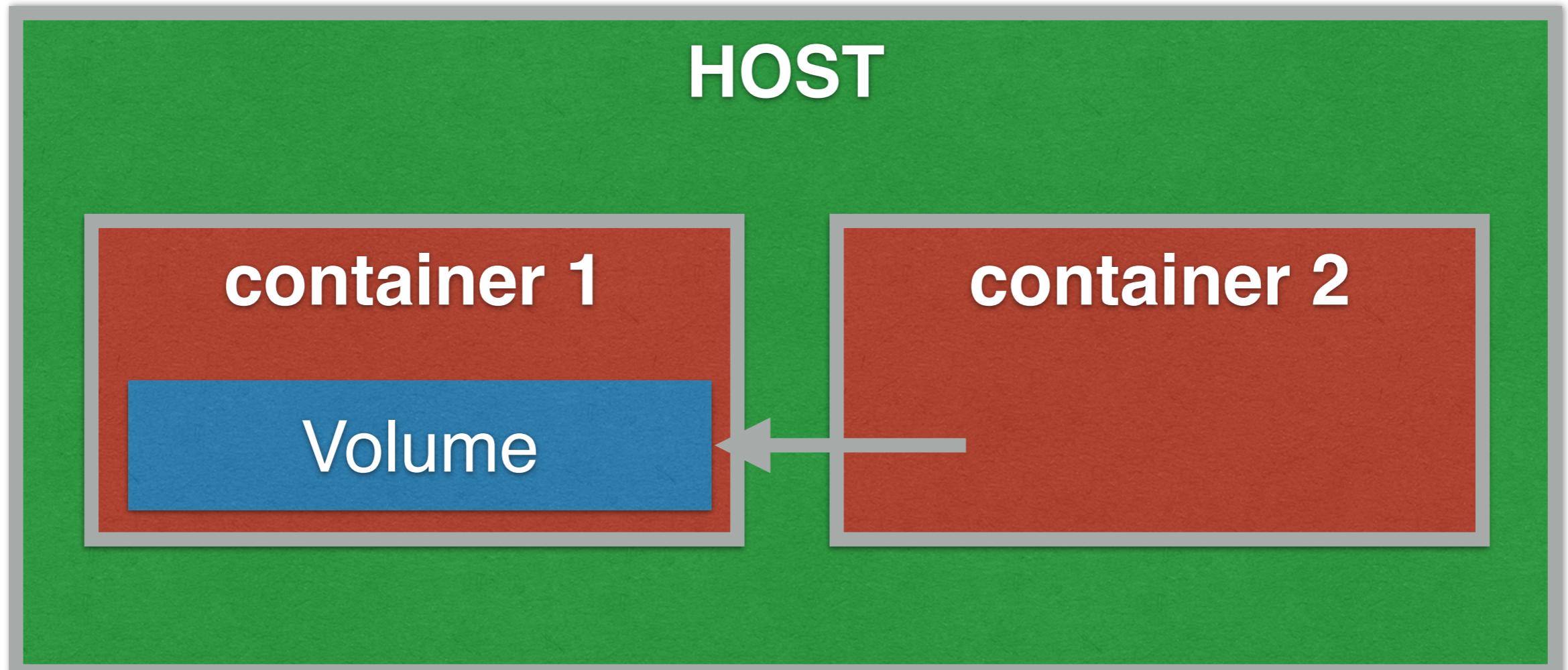


Command-line

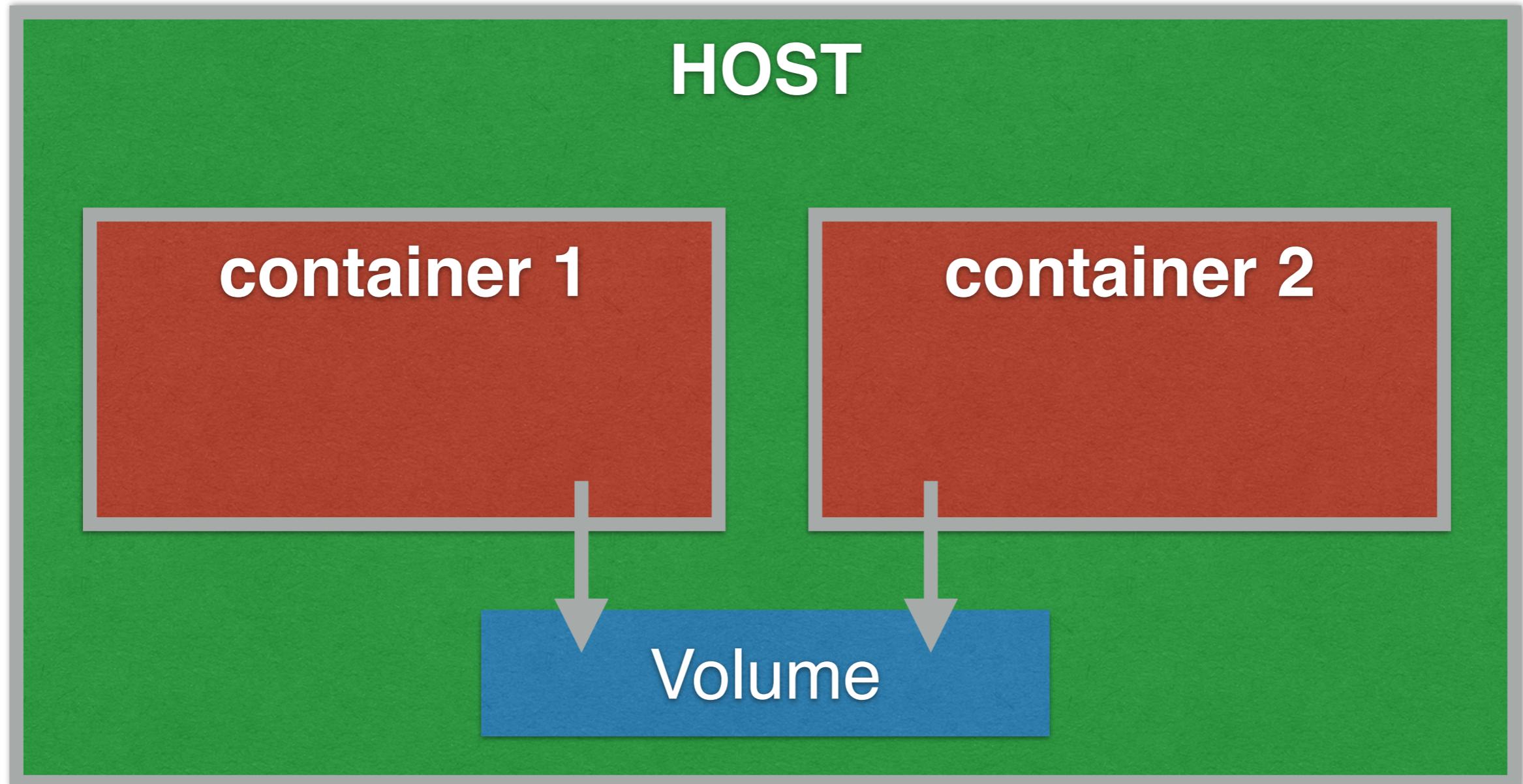
```
$ docker run -d -v /var/lib/postgresql \  
training/postgresql
```



2. Share volumes across containers



Real working



Create container 1

```
$docker container run -it \  
  --name container1 \  
  -v /var/log \  
  ubuntu bash
```



Edit file in container 1

```
:/# date >/var/log/current_date
```



Create container 2 with volume

```
$docker container run --name container2 \
--volumes-from container1 \
ubuntu cat /var/log/current_date
```



Volumes independent of containers

If Container is stopped

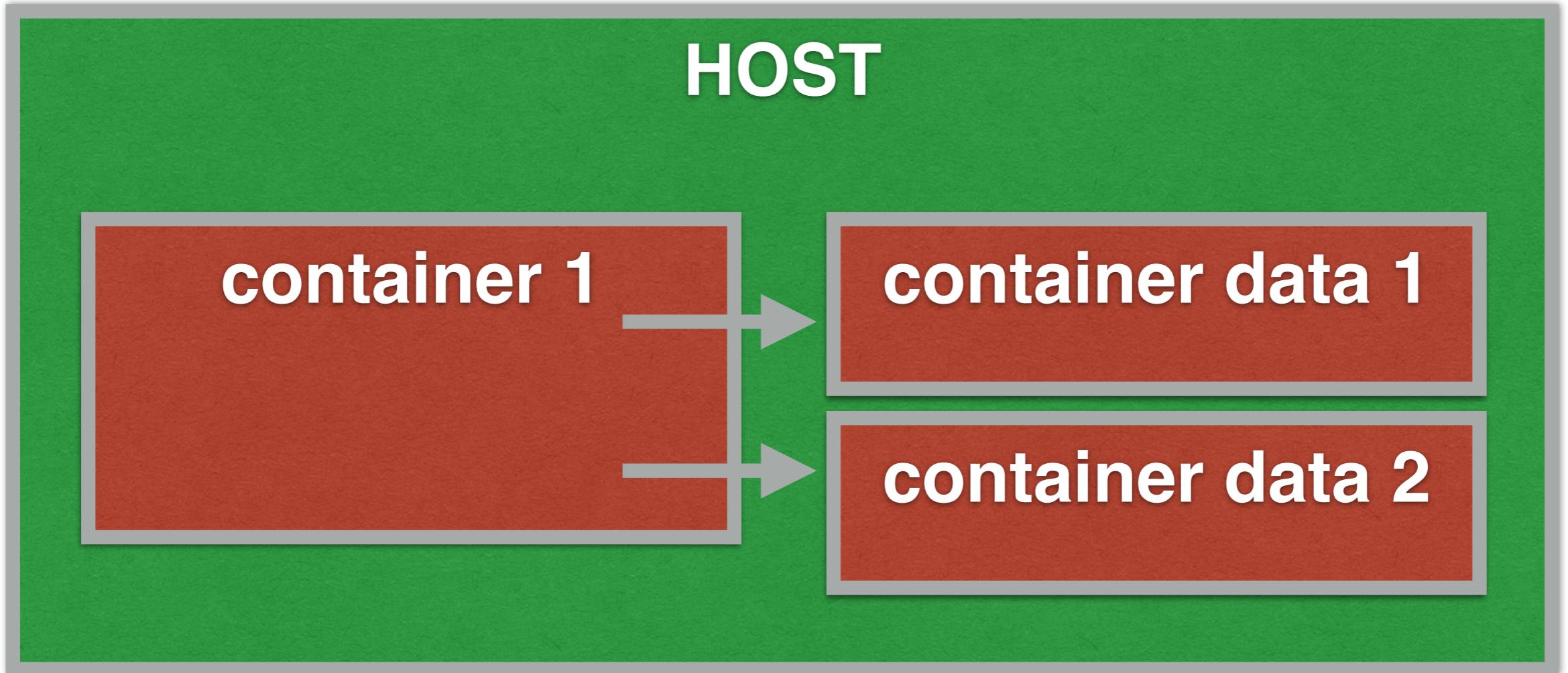
Volume still exist and available

Try to stop container 1 !!



3. Data container

Container for the purpose of referencing volumes



Create data containers with busybox

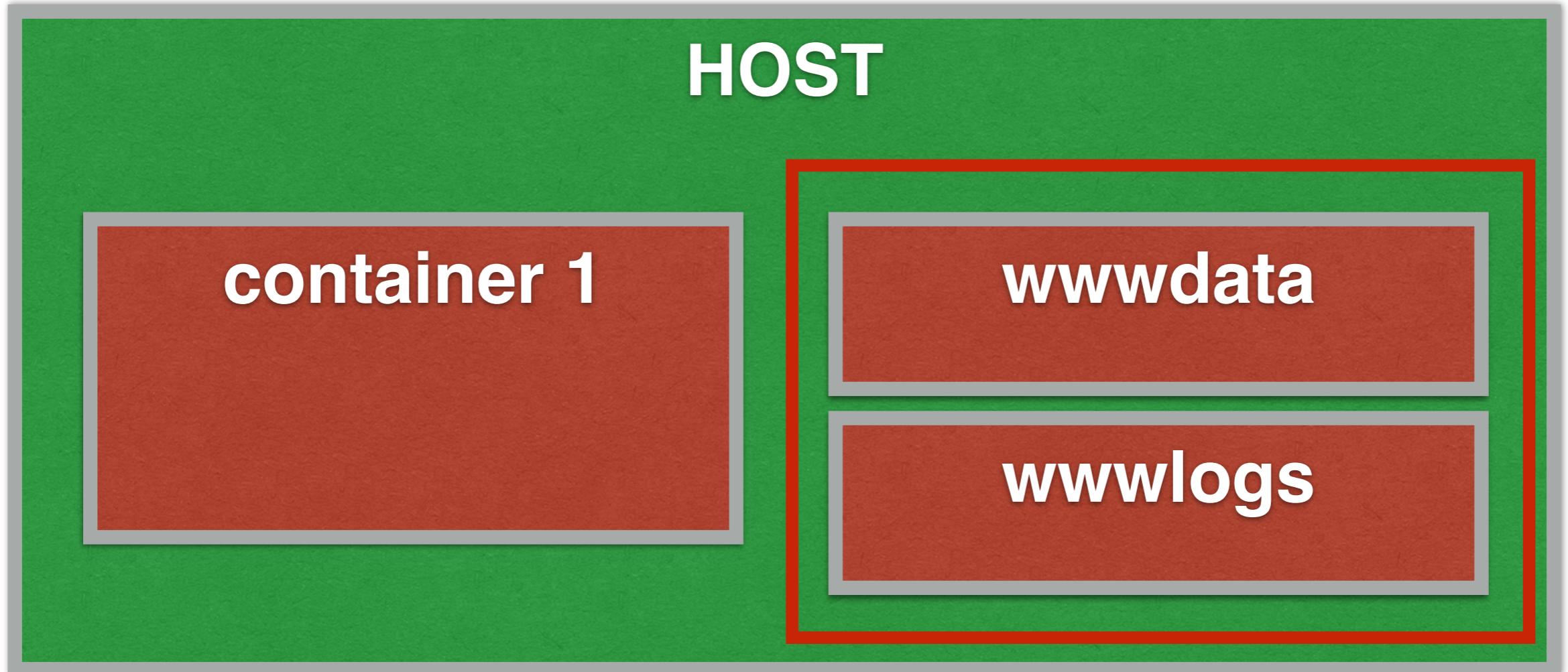
```
$ docker run --name wwwdata -v /var/lib/www busybox true
```

```
$ docker run --name wwwlogs -v /var/log/www busybox true
```

We have 2 containers



Data containers with busybox



Using data containers

```
$docker run -d --volumes-from wwwdata \  
--volumes-from wwwlogs webserver
```

```
$docker run -d --volumes-from wwwdata ftpserver
```

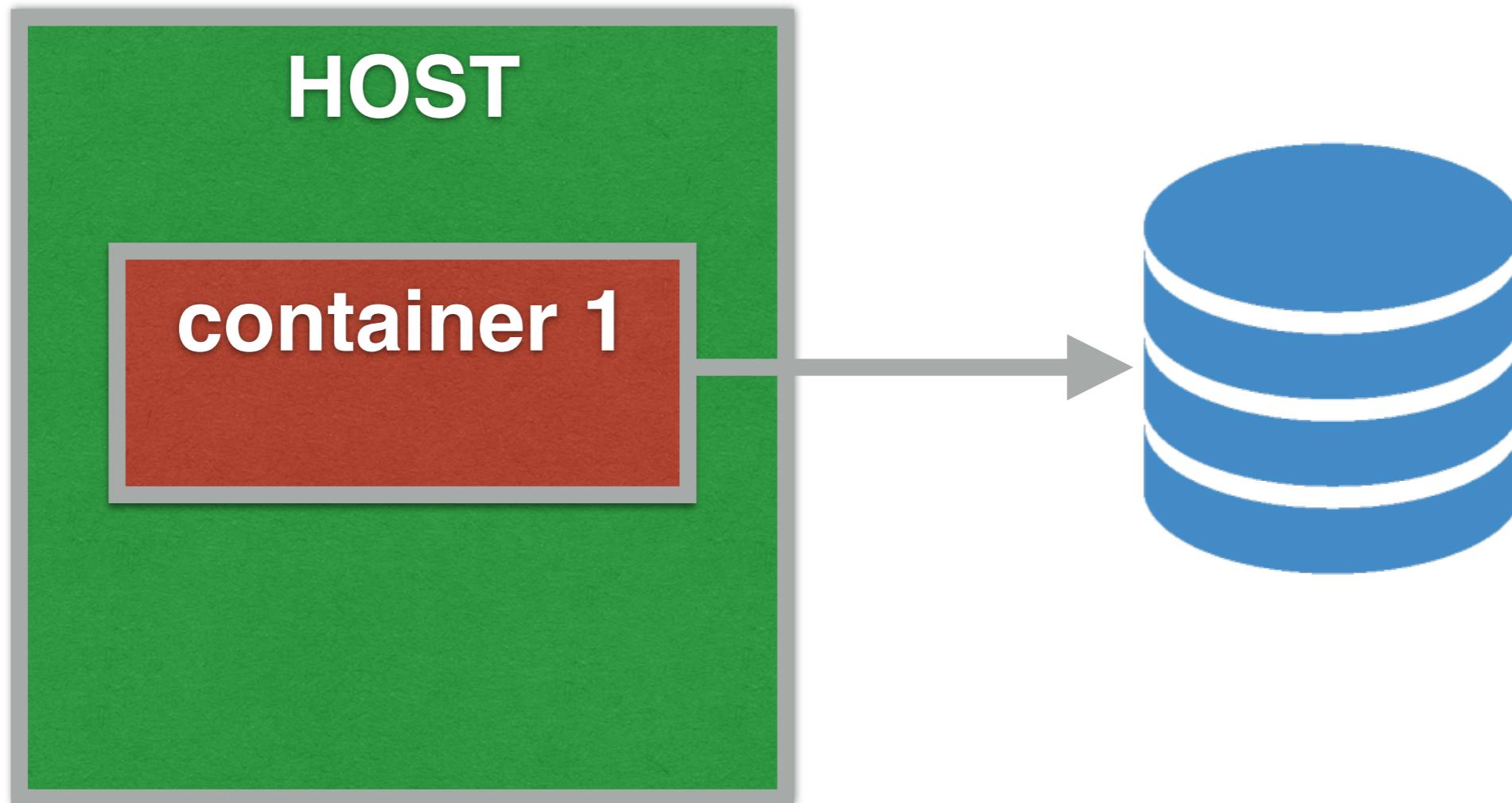
```
$docker run -d --volumes-from wwwlogs pipestash
```

How to test ?

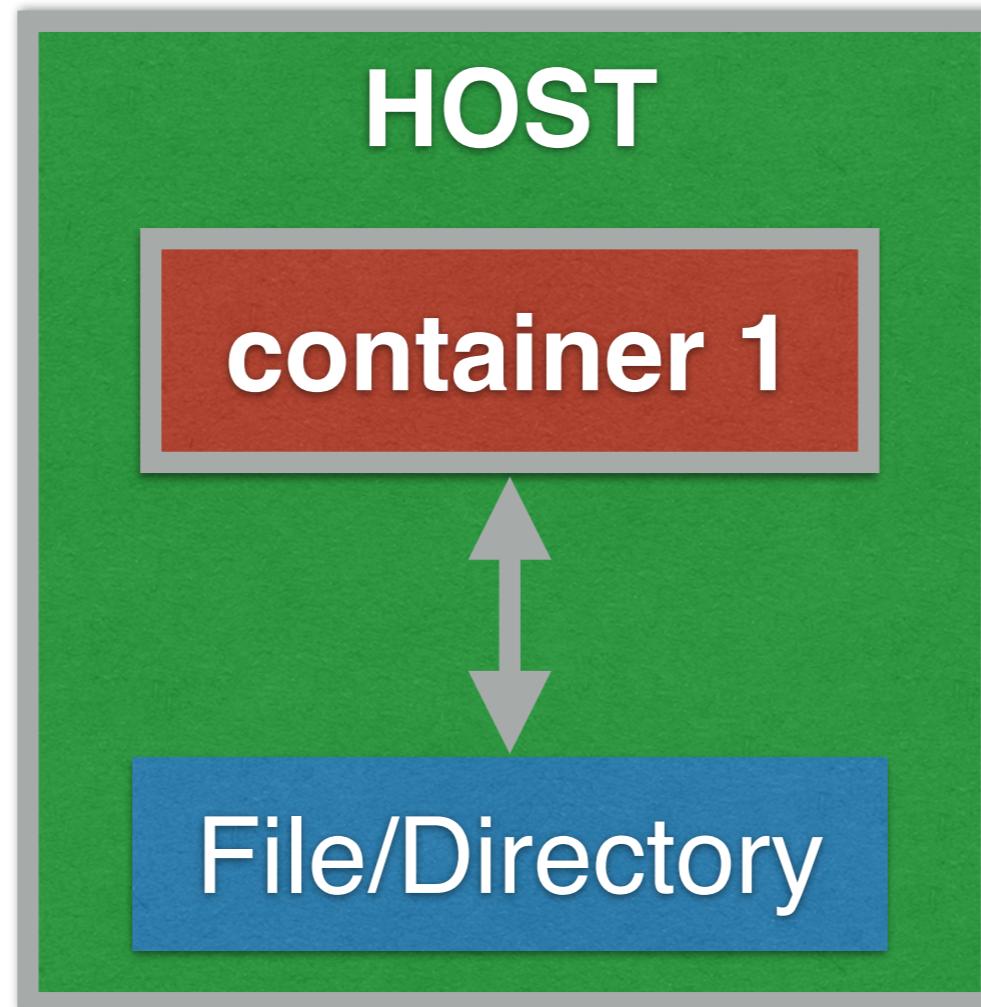


4. External storage

LVM, SAN, NFS, ZFS, Ceph, GlusterFS ... etc.



5. Share file/dir between host and container



Working with Nginx

OFFICIAL REPOSITORY

[nginx](#) 

Last pushed: 20 days ago

[Repo Info](#) [Tags](#)

Short Description

Official build of Nginx.

Docker Pull Command 

`docker pull nginx`

Full Description

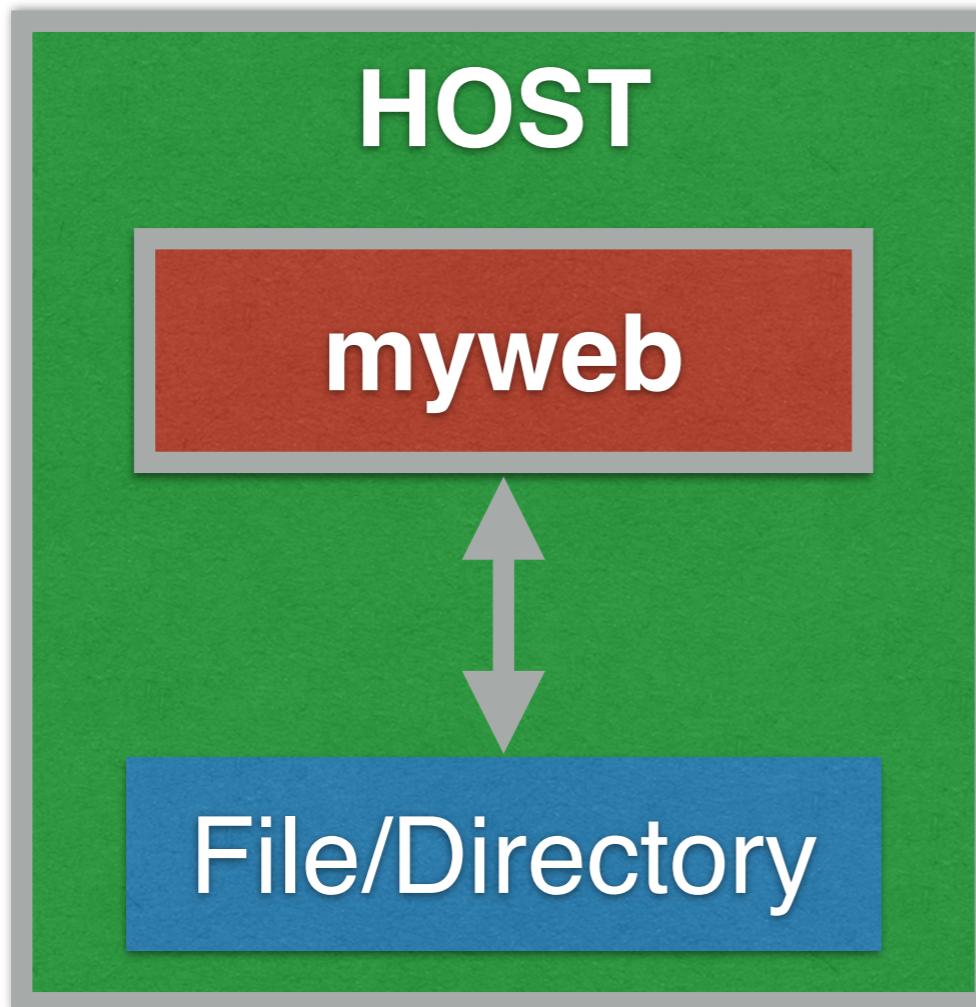
Supported tags and respective `Dockerfile` links

- `1.11.10`, `mainline`, `1`, `1.11`, `latest` ([mainline/jessie/Dockerfile](#))
- `1.11.10-alpine`, `mainline-alpine`, `1-alpine`, `1.11-alpine`, `alpine` ([mainline/alpine/Dockerfile](#))
- `1.10.3`, `stable`, `1.10` ([stable/jessie/Dockerfile](#))
- `1.10.3-alpine`, `stable-alpine`, `1.10-alpine` ([stable/alpine/Dockerfile](#))

https://hub.docker.com/_/nginx/



Working with Nginx



/usr/share/nginx/html

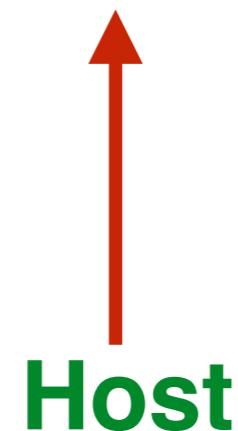
`$(pwd)/web`



Working with Nginx

```
$mkdir web
```

```
$docker container run -d -P --name myweb \  
-v $(pwd)/web:/usr/share/nginx/html \  
nginx
```



See publish port and test

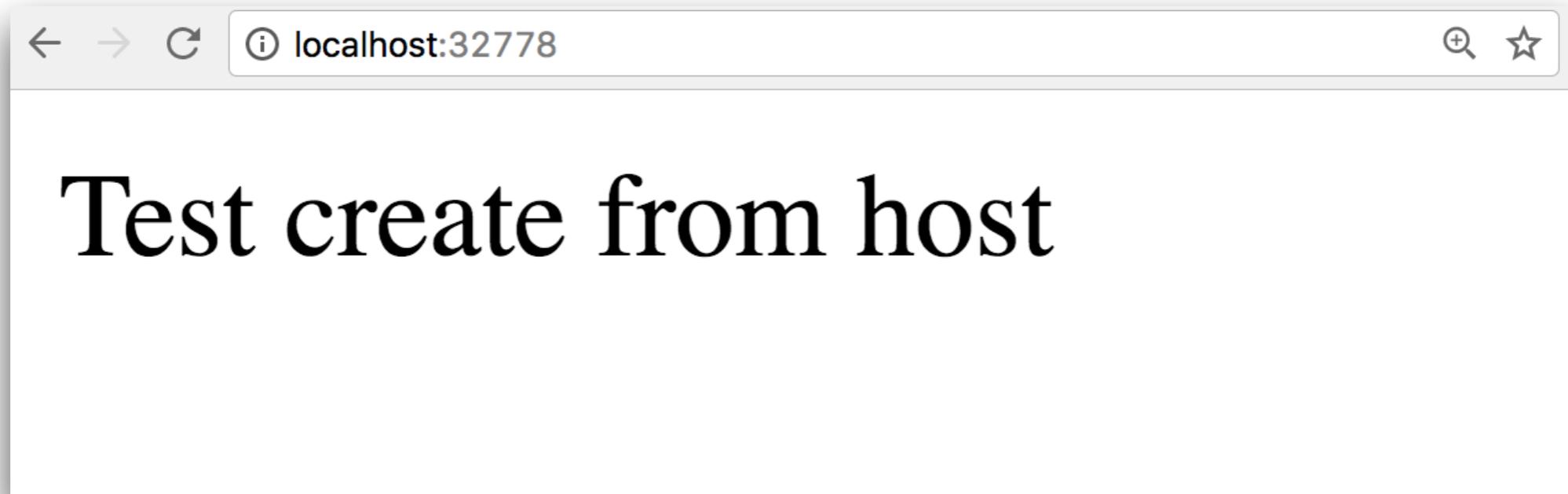
\$docker container port my web



Create new file in host

```
$cd web
```

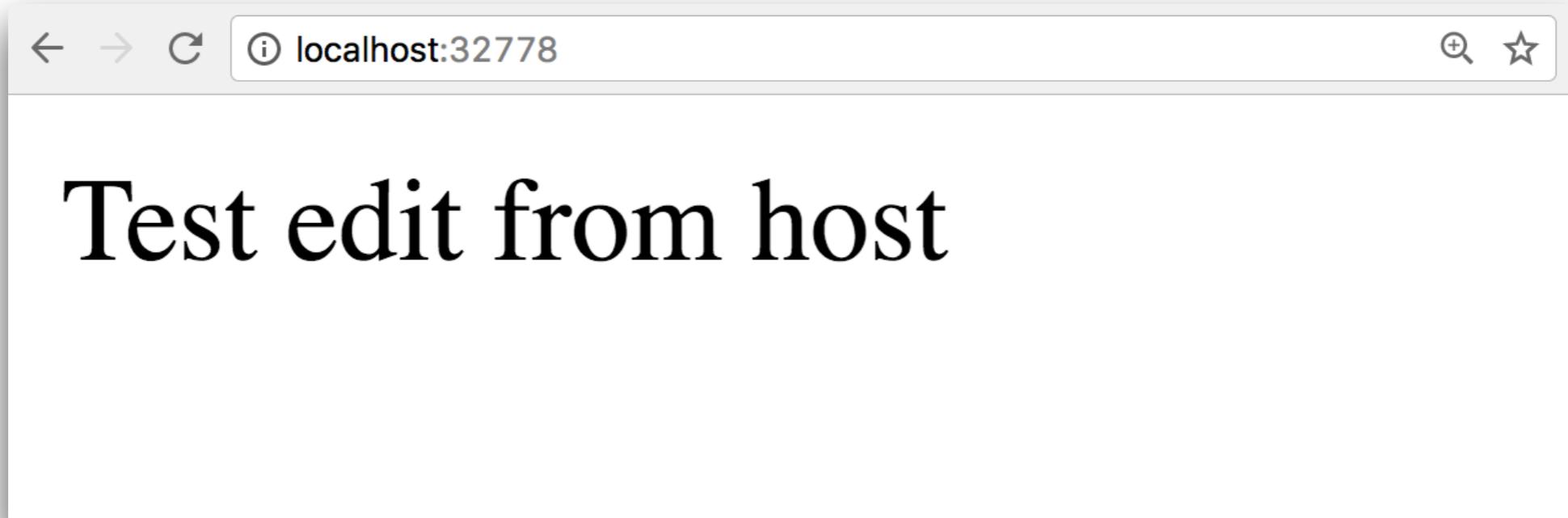
```
$echo "Test create from host" > index.html
```



Try to edit file in host

```
$cd web
```

```
$echo "Test edit from host" > index.html
```



Check volume in image

```
$docker image inspect <name>
```



Check volume in container

\$docker container inspect <id/name>

```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/Users/somkiat/data/slide/docker",
        "Destination": "/usr/share/nginx/html",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
],
```



Check volume in container

```
$docker container inspect <id/name>
```

```
$docker inspect --format='{{json .Mounts}}' <id>
```

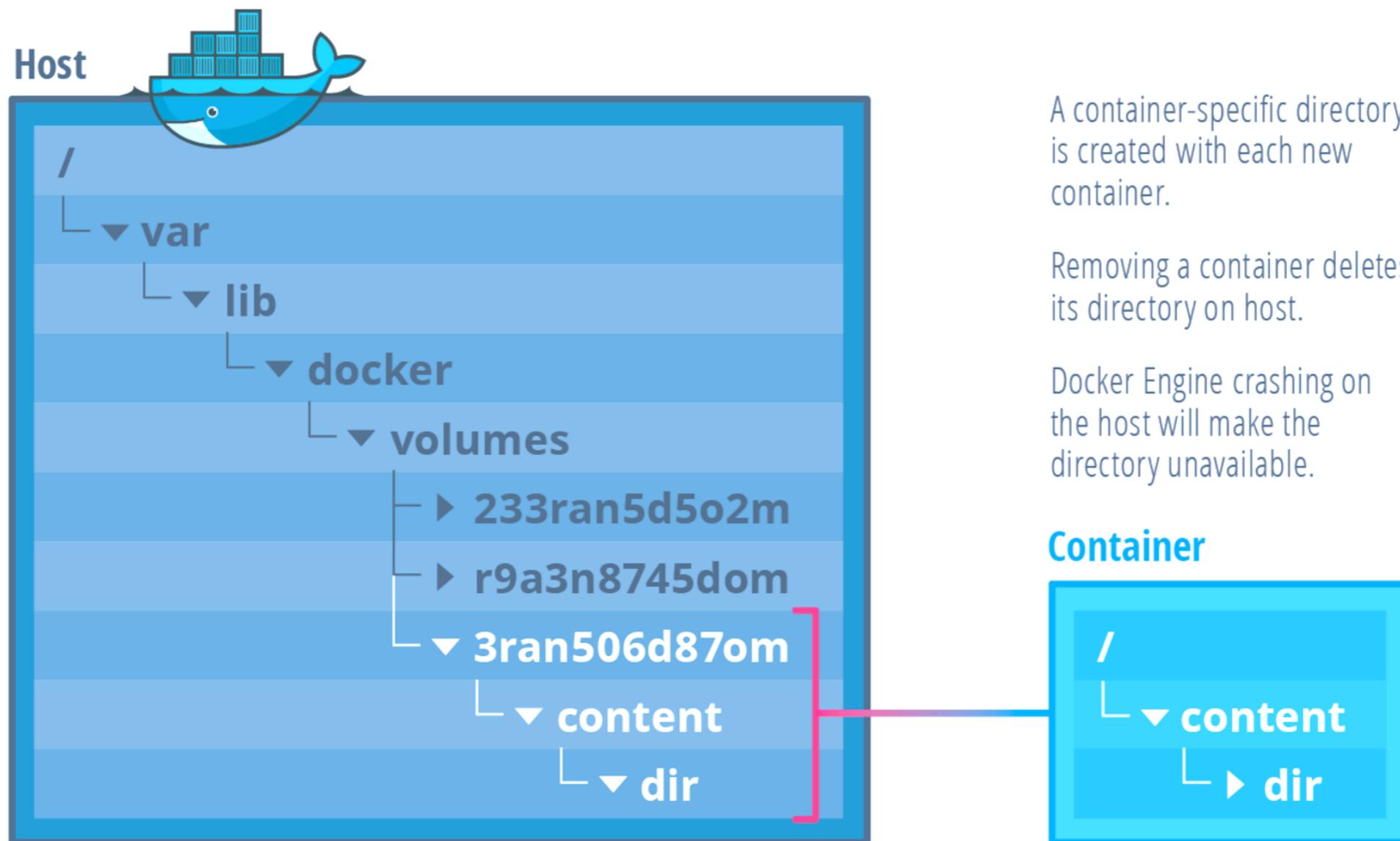
```
$docker inspect  
  --format='{{.HostConfig.VolumesFrom}}' <id>
```



Strategies to Manage Persistent Data

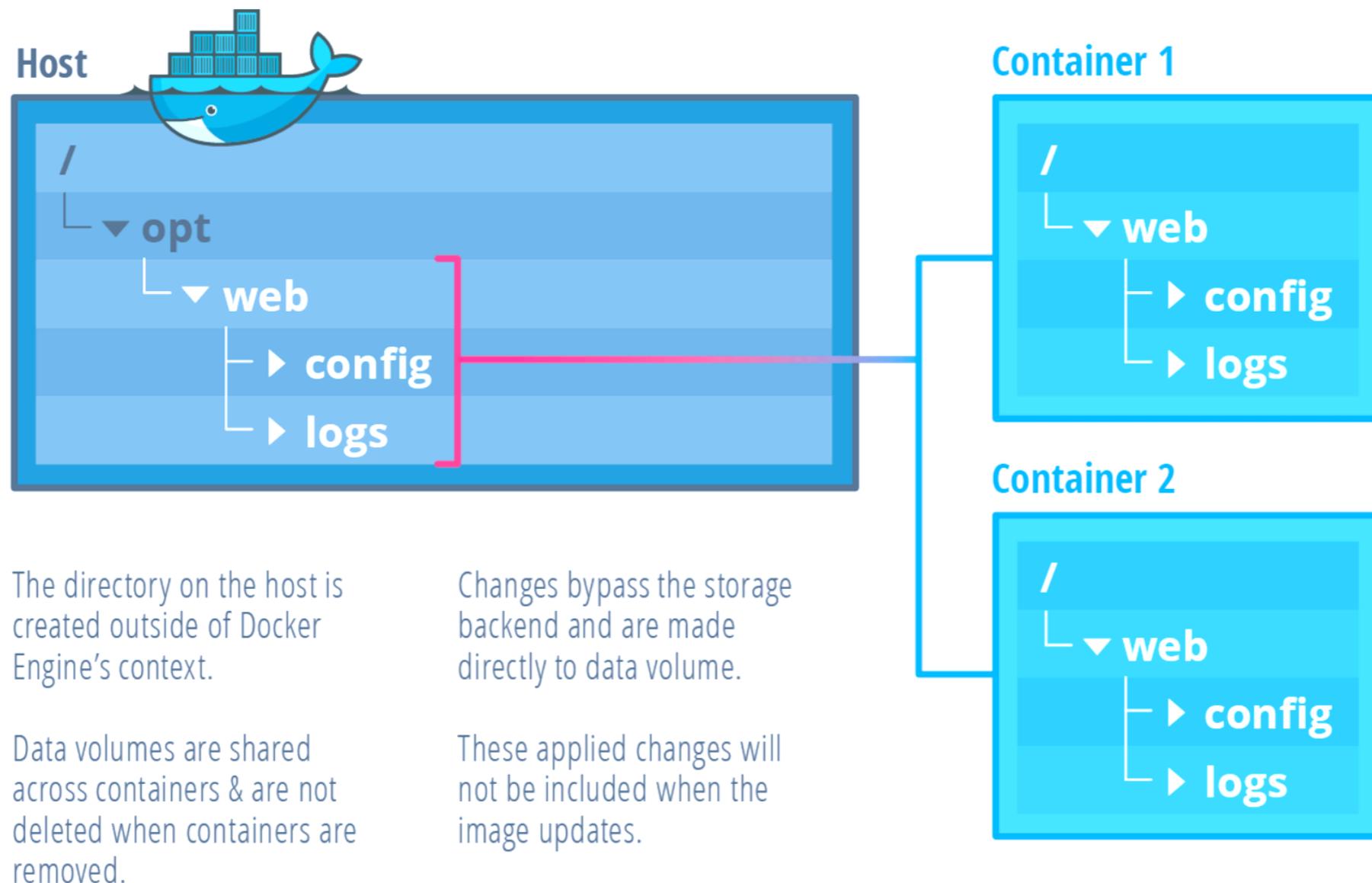


Host-based persistence per container



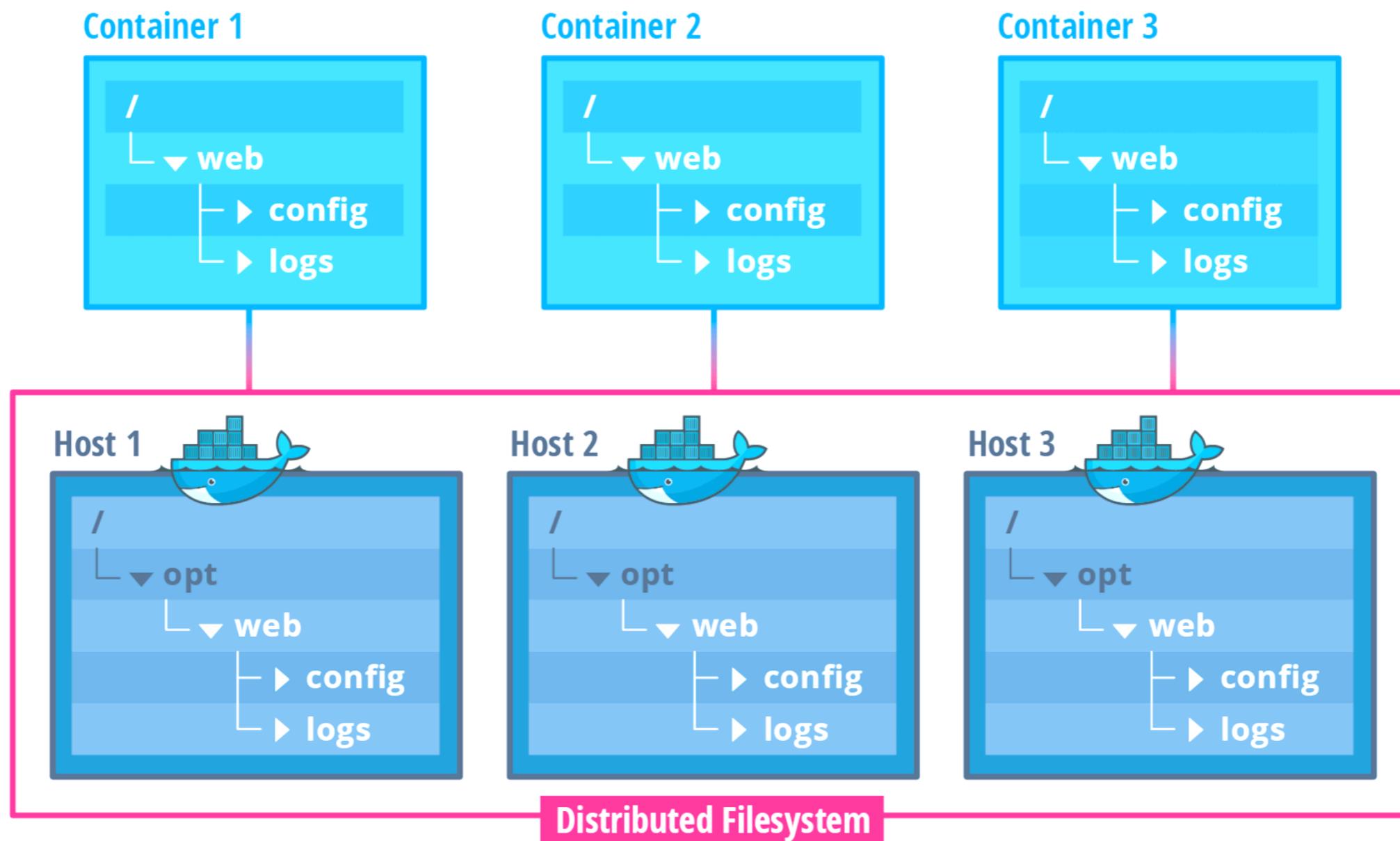
Host-based persistence

shared containers

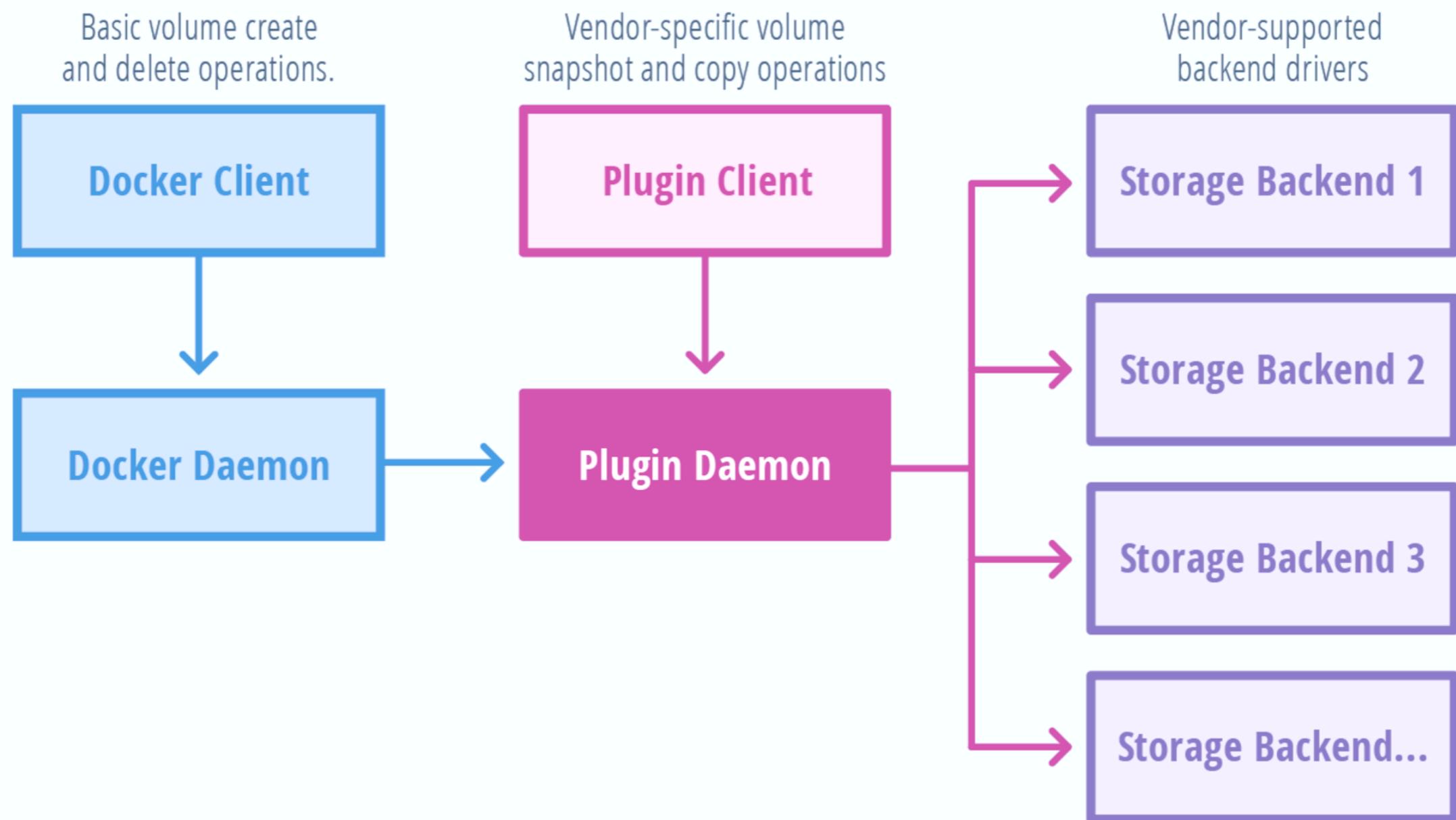


Multi-Host persistence

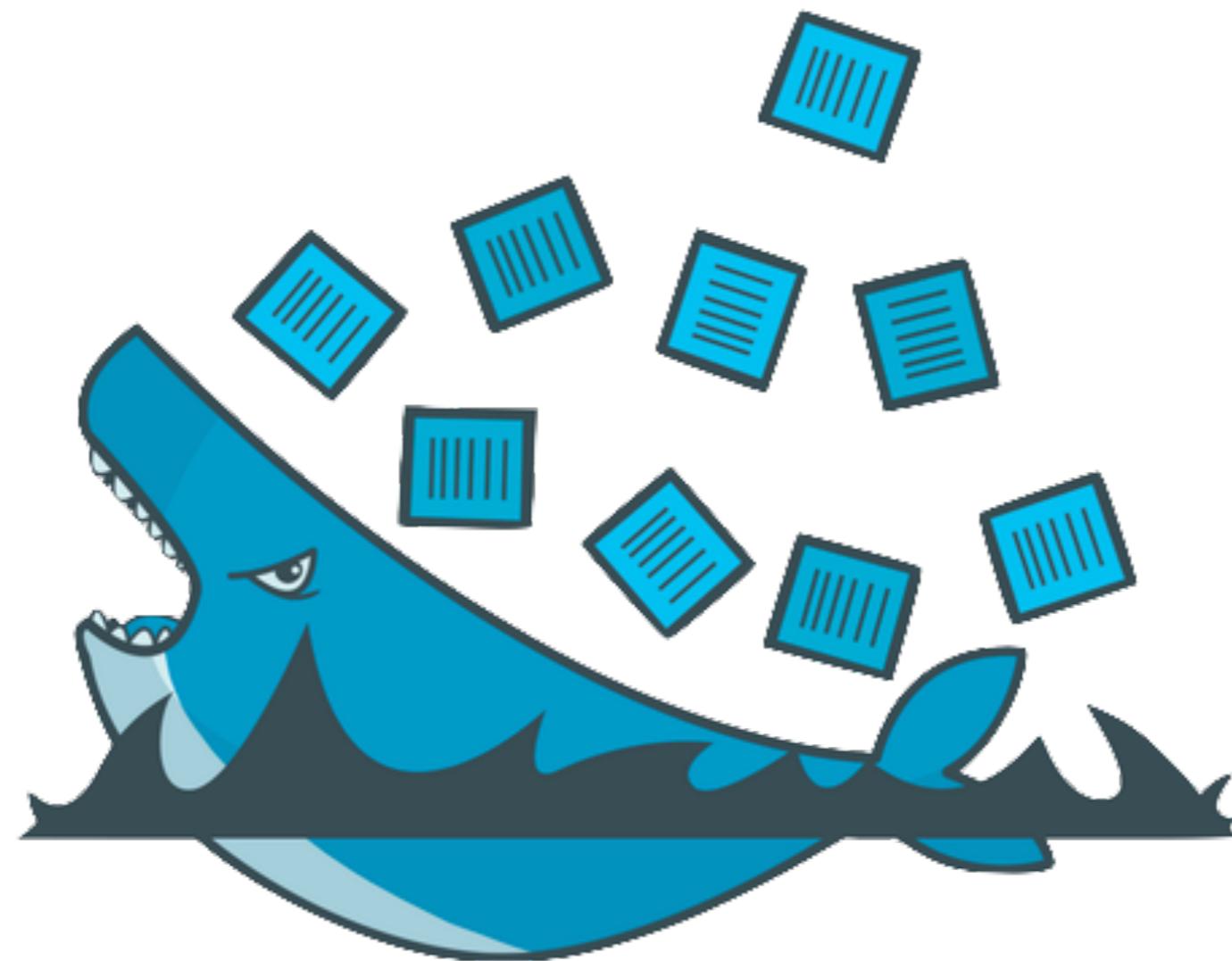
shared containers



Docker volume plugin



Workshop



Topics

Dockerfile

Multi-stage of Dockerfile

Working with environment variable

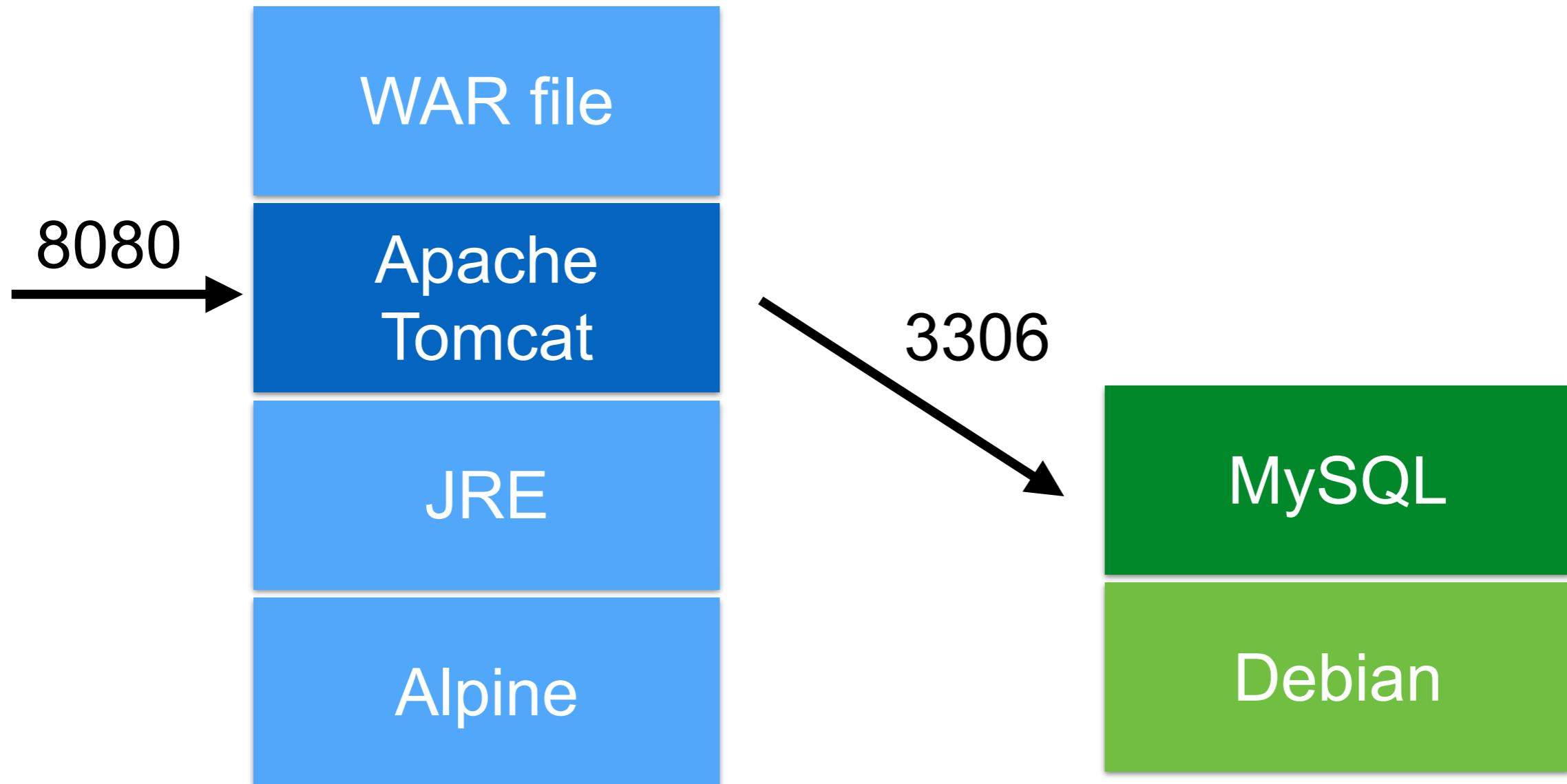
Working with volume

Working with Docker-compose

Working with Docker-swarm



Architecture



Source code

<https://github.com/up1/docker-workshop-java-mysql>



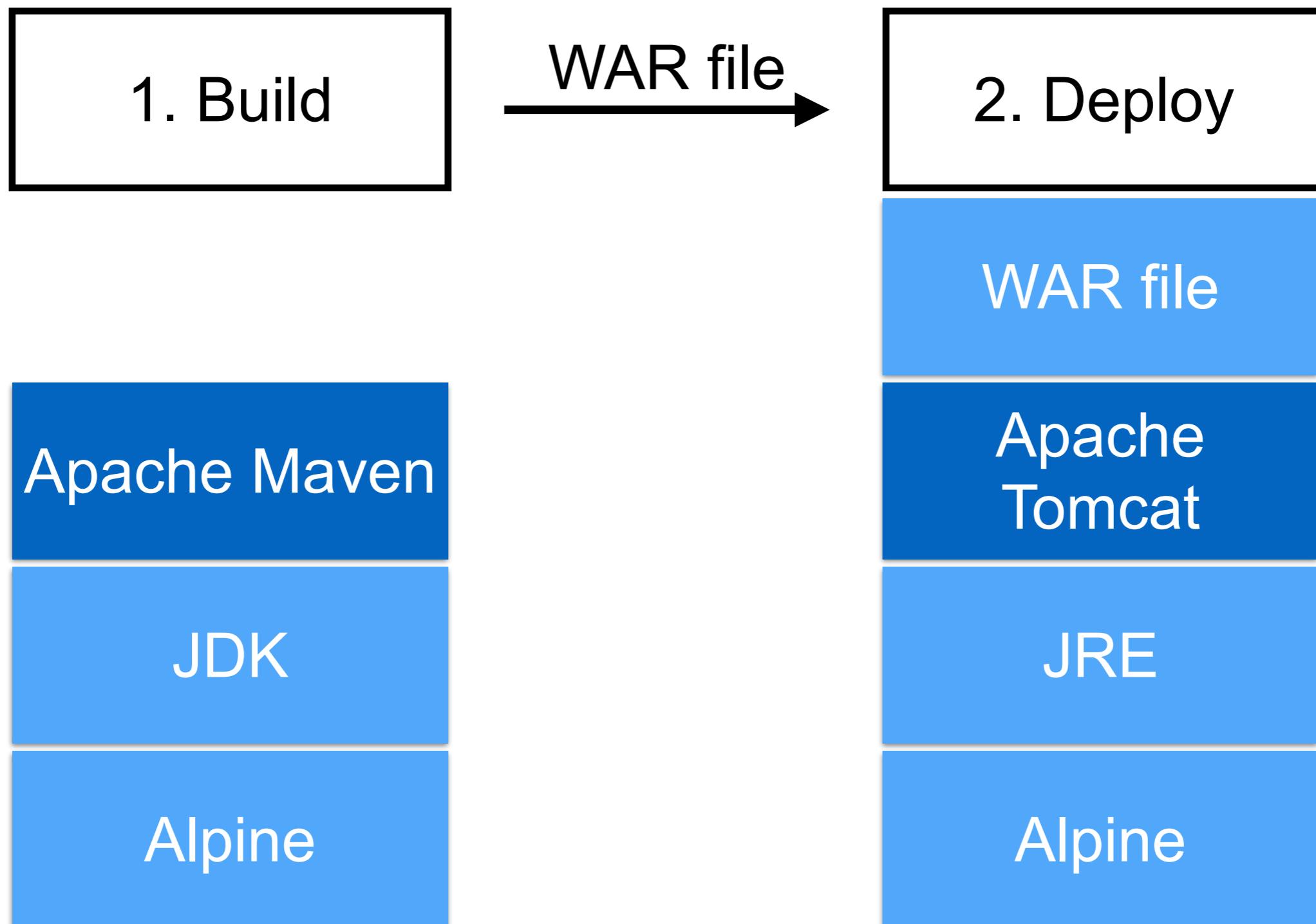
Web with Apache Tomcat



Step to deploy



Step to deploy



Build



Dockerfile_build

```
FROM maven:3.5.2-jdk-8-alpine
WORKDIR /src
COPY . /src

ENTRYPOINT ["mvn"]
CMD ["clean", "package"]
```



Build image of Build process

```
$ docker image build -t web_build:0.1  
-f Dockerfile_build .
```



Create build container

```
$ docker container run --rm  
-v "$HOME/.m2":/root/.m2  
-v $(pwd):/src  
web_build:0.1
```



Deploy



Dockerfile_deploy

```
FROM tomcat:9.0.1-jre8-alpine
COPY ./target/api.war /usr/local/tomcat/
webapps/api.war
```



Build image of Deploy process

```
$docker image build -t web_deploy:0.1  
-f Dockerfile_deploy .
```



Create deploy container

```
$docker container run -d  
-p 8080:8080  
web_deploy:0.1
```



Many step to build !!



Using multi-stage build

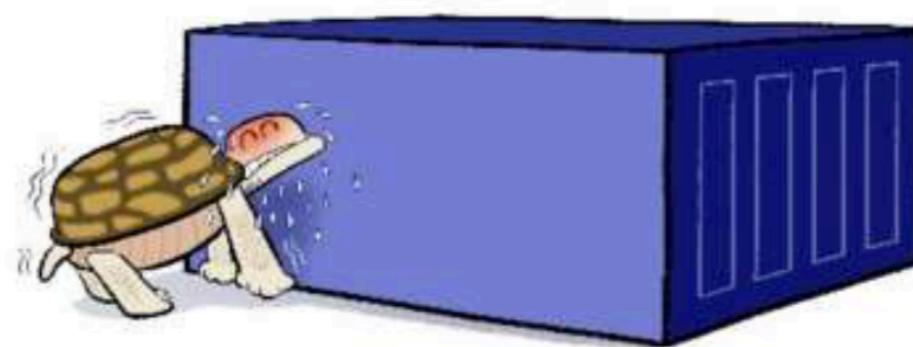
(Docker 17.05+)

<https://docs.docker.com/engine/userguide/eng-image/multistage-build/>



Build smaller images with multi-stage builds

First stage:
complete build environment



Second stage:
minimal runtime environment



One Dockerfile, one build

Dockerfile_api

```
FROM maven:3.5.2-jdk-8-alpine as builder
WORKDIR /src
COPY . /src
RUN mvn clean package
```

```
FROM tomcat:9.0.1-jre8-alpine
COPY --from=builder /src/target/api.war /usr/
local/tomcat/webapps/
```



Build image with multi-stage

```
$ docker image build -t web_api:0.1  
-f Dockerfile_api .
```



Downloading . . .



How to improve ?



Local maven server !!





Sonatype
Nexus



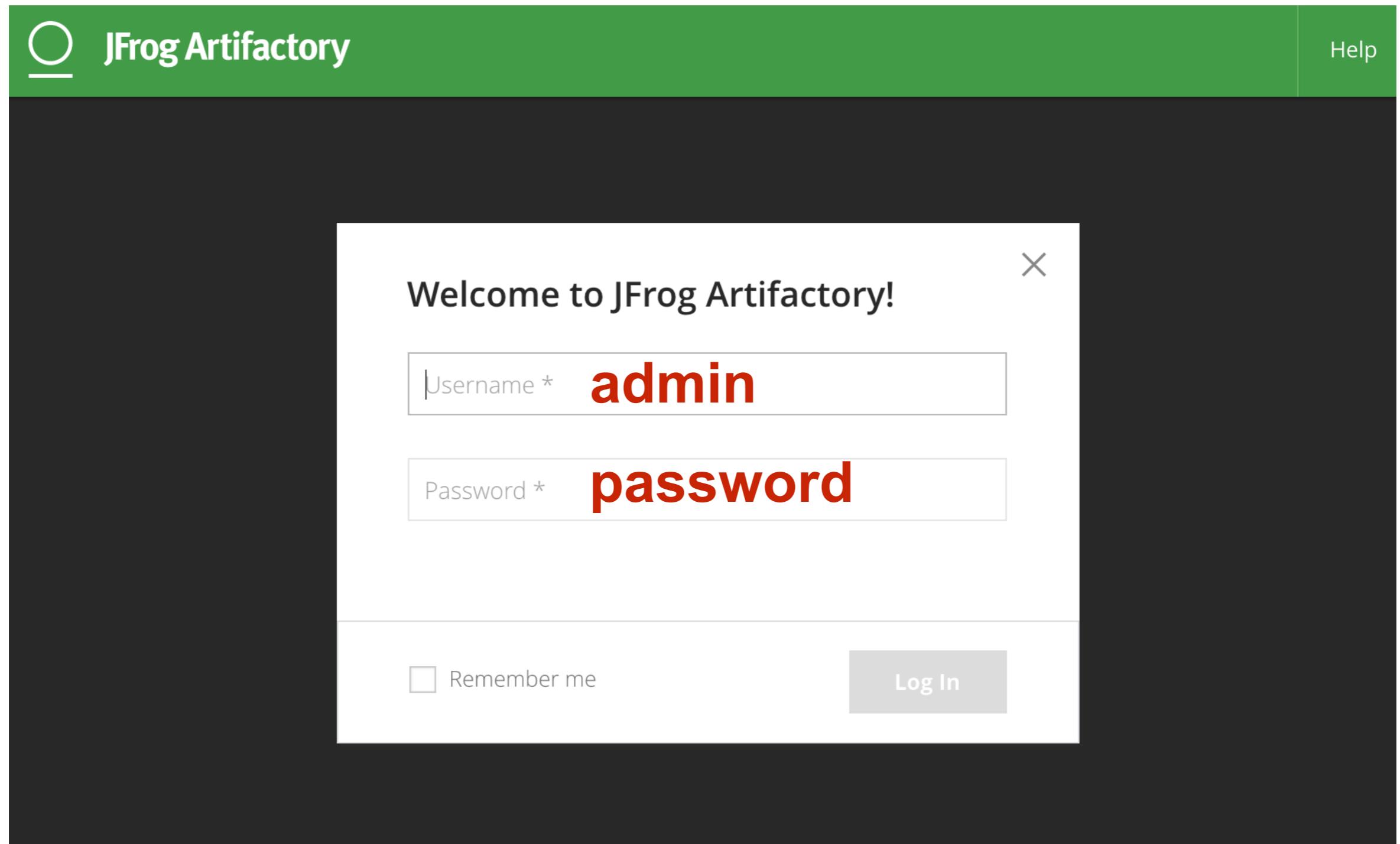
Frog Artifactory

```
$docker run --name artifactory -d  
-p 8081:8081  
docker.bintray.io/jfrog/artifactory-oss:latest
```

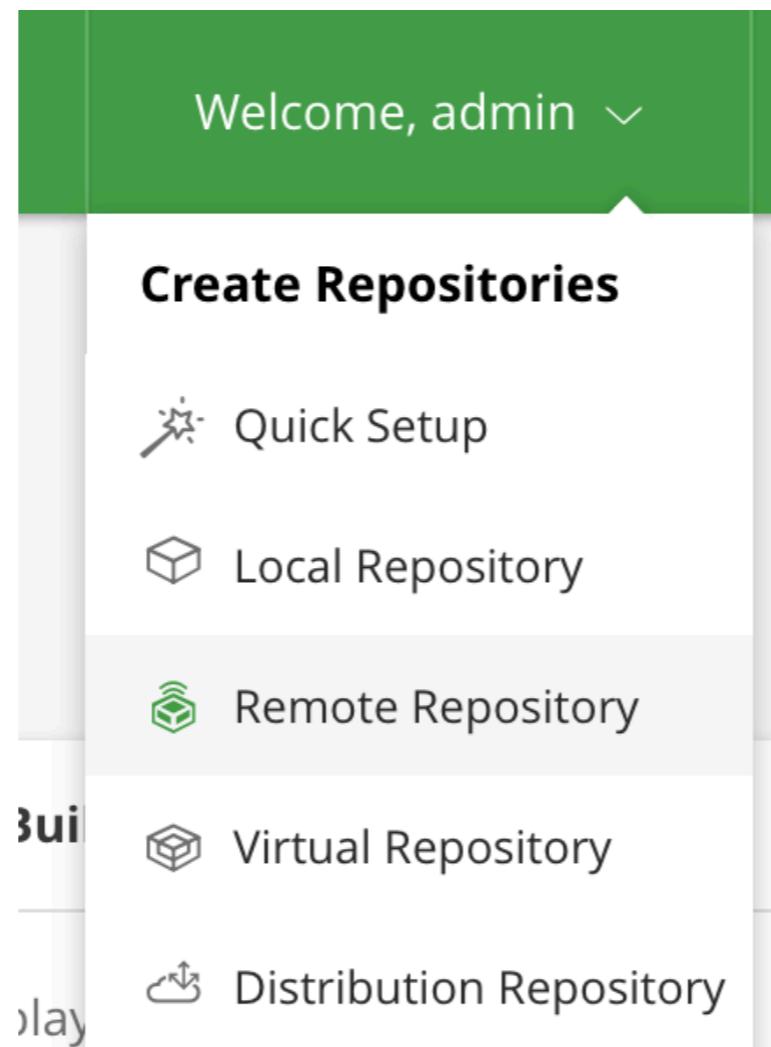
<https://www.jfrog.com/confluence/display/RTF/Installing+with+Docker>



Config Frog Artifactory



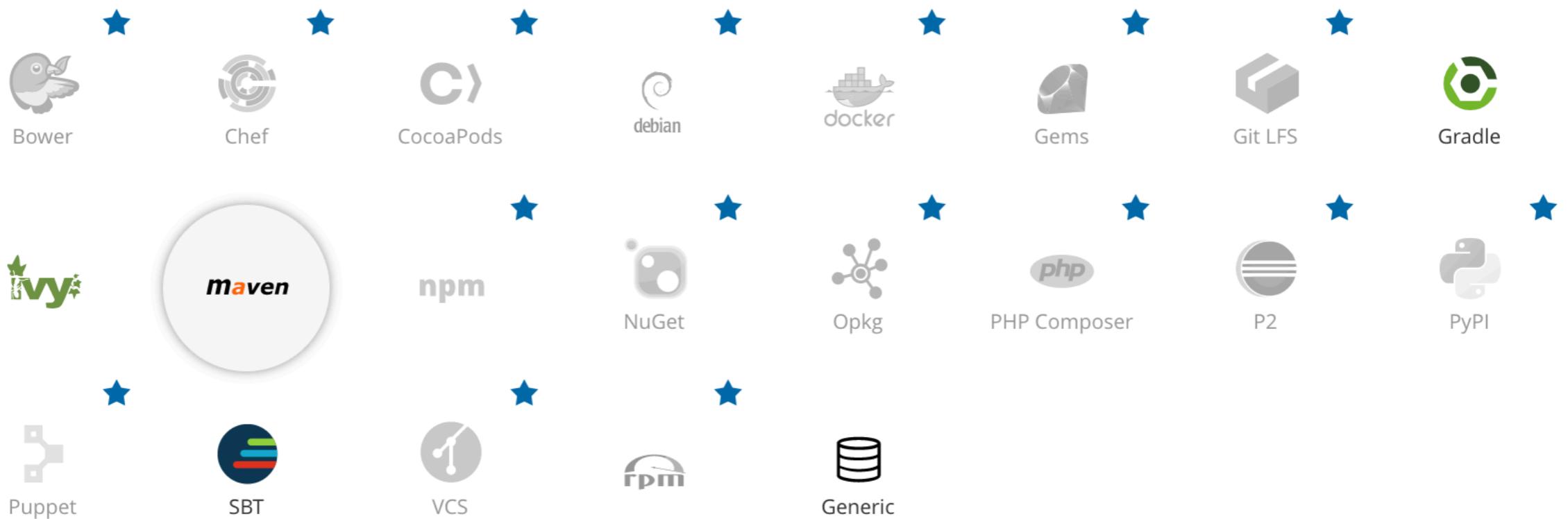
Create remote repository



Choose Maven

Select Package Type

X



Create remote repository

New Remote Repository

Basic Advanced Replications 

Package Type *

maven

Maven

Repository Key *

sample

URL *

https://jcenter.bintray.com

Test

General

Repository Layout

maven-2-default

Maven Settings

Checksum Policy 

Generate if absent

Cancel **Save & Finish**



Config maven

File /root/.m2/settings.xml

```
<mirrors>
  <mirror>
    <id>central</id>
    <name>central</name>
    <url>http://172.17.0.2:8081/artifactory/sample</url>
    <mirrorOf>*</mirrorOf>
  </mirror>
</mirrors>
```



Let's try again !!



Dockerfile_api

```
FROM maven:3.5.2-jdk-8-alpine as builder
WORKDIR /src
COPY settings.xml /root/.m2/settings.xml
COPY . /src
RUN mvn clean package
```

```
FROM tomcat:9.0.1-jre8-alpine
COPY --from=builder /src/target/api.war /usr/local/tomcat/webapps/
```



Build image with multi-stage

```
$ docker image build -t web_api:0.1  
-f Dockerfile_api .
```



Create container

```
$docker container run -d  
-p 8080:8080  
web_api:0.1
```



Database with MySQL



Docker image of MySQL

OFFICIAL REPOSITORY

[mysql](#) 

Last pushed: 8 days ago

[Repo Info](#) [Tags](#)

Short Description

MySQL is a widely used, open-source relational database management system (RDBMS).

Docker Pull Command

```
docker pull mysql
```

Full Description

Supported tags and respective [Dockerfile](#) links

- [8.0.3](#) , [8.0](#) , [8](#) ([8.0/Dockerfile](#))
- [5.7.20](#) , [5.7](#) , [5](#) , [latest](#) ([5.7/Dockerfile](#))
- [5.6.38](#) , [5.6](#) ([5.6/Dockerfile](#))
- [5.5.58](#) , [5.5](#) ([5.5/Dockerfile](#))

https://hub.docker.com/_/mysql/



Create container

```
$docker container run \
-d \
--name=my_database \
-e "MYSQL_ROOT_PASSWORD=mypassword"
\
-e "MYSQL_DATABASE=sample" \
-e "MYSQL_USER=user01" \
-e "MYSQL_PASSWORD=password" \
mysql:5.7.20
```



Insert data into mysql

Create file import.sql

```
USE sample;
```

```
CREATE TABLE USER (
    id INT(11),
    name char(60)
) ENGINE=INNODB;
```

```
INSERT INTO USER VALUES(1, 'Sample name 01');
INSERT INTO USER VALUES(2, 'Sample name 02');
INSERT INTO USER VALUES(3, 'Sample name 03');
```



Create Dockerfile_data

```
FROM mysql:5.7.20
COPY import.sql /docker-entrypoint-initdb.d/
```



Build image of mysql with data

```
$ docker image build -t mysql_data  
-f Dockerfile_data .
```

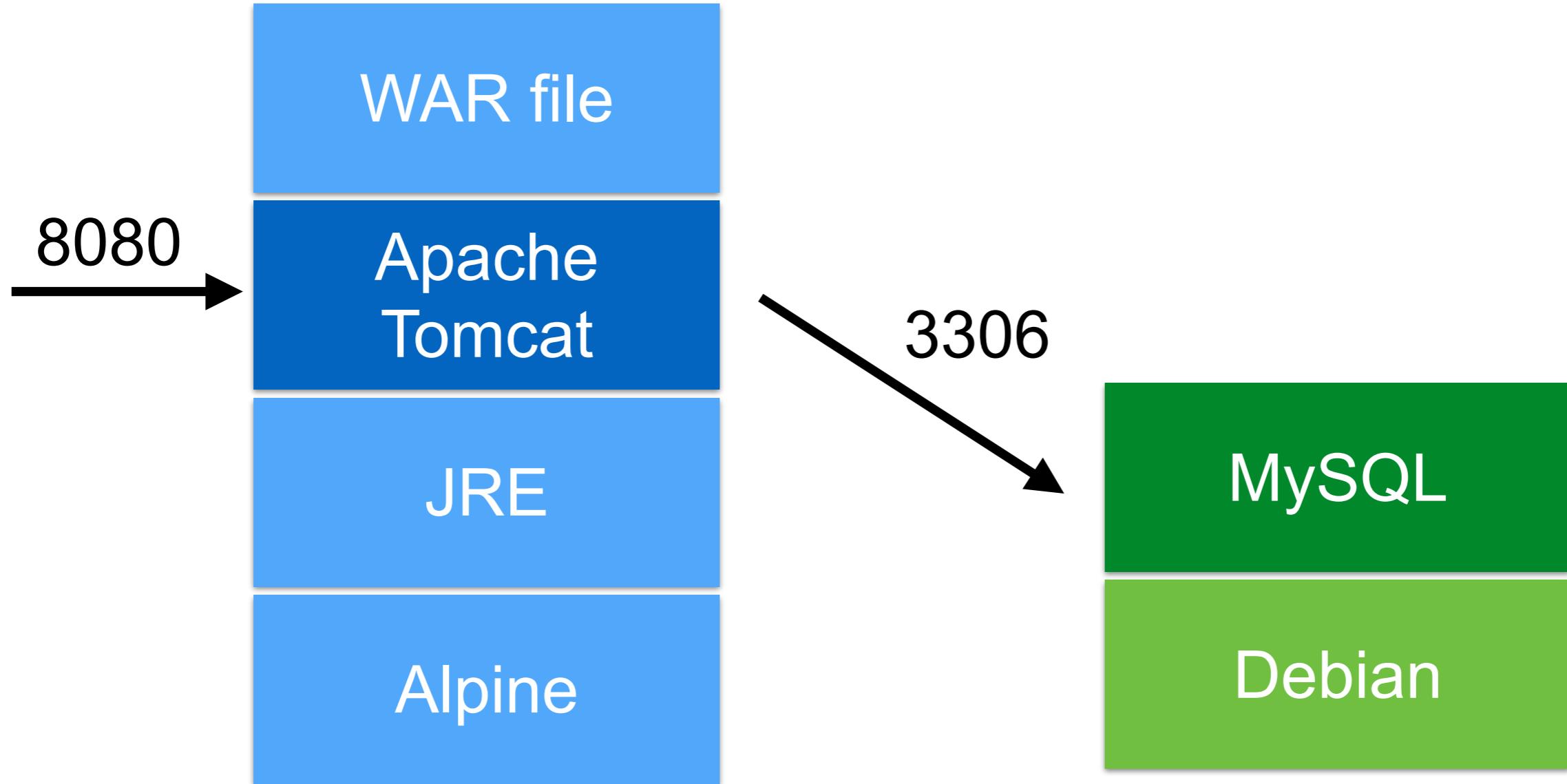


Create container again !!

```
$docker container run \
-d \
--name=my_database \
-e "MYSQL_ROOT_PASSWORD=mypassword"
\
-e "MYSQL_DATABASE=sample" \
-e "MYSQL_USER=user01" \
-e "MYSQL_PASSWORD=password" \
mysql_data
```



Architecture



Linking container



Create api container !!

```
$ docker container run
```

```
-d
```

```
-p 8080:8080
```

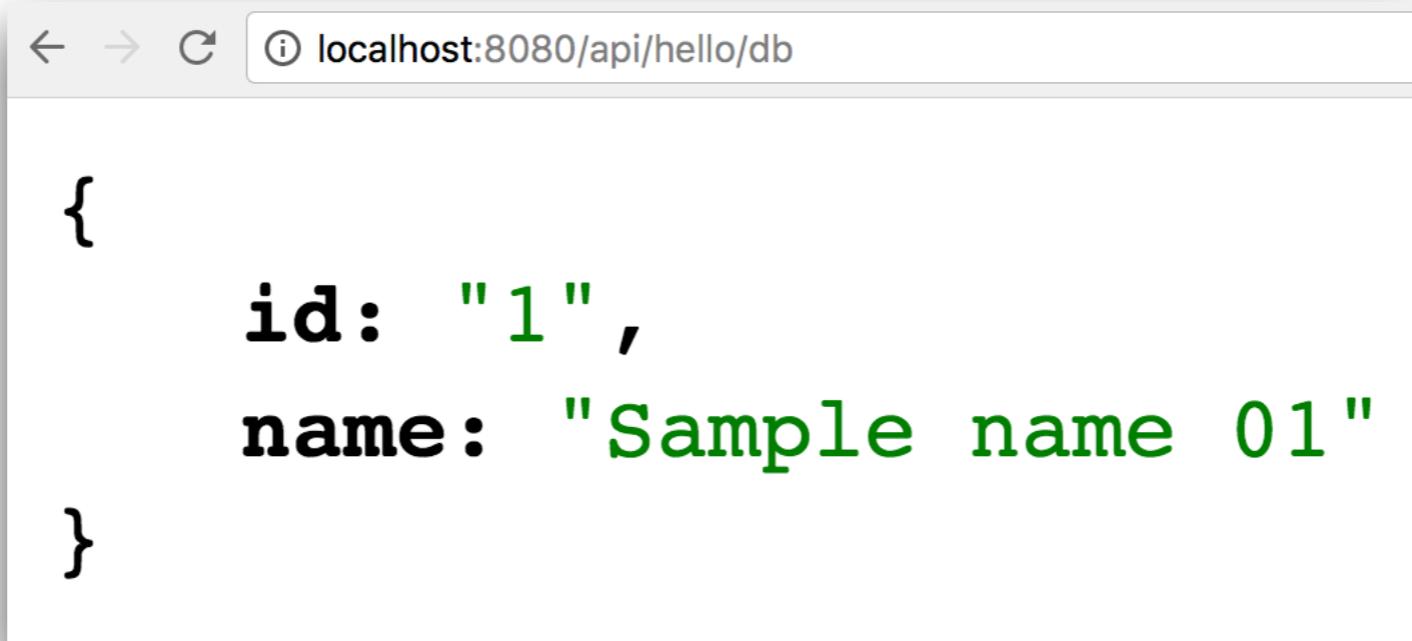
```
--link my_database
```

```
web_api:0.1
```



Testing with database

`http://localhost:8080/api/hello/db`



A screenshot of a web browser window. The address bar shows the URL `localhost:8080/api/hello/db`. The main content area displays the following JSON object:

```
{  
  id: "1",  
  name: "Sample name 01"  
}
```



Working with Environment variable



Using ENV in Dockerfile

```
FROM tomcat:9.0.1-jre8-alpine
ENV DATABASE_URL="jdbc:mysql://my_database/
sample?user=user01&password=password"
COPY --from=builder /src/target/api.war /usr/
local/tomcat/webapps/
```



Get ENV from Java

```
System.getenv("DATABASE_URL")
```

easy

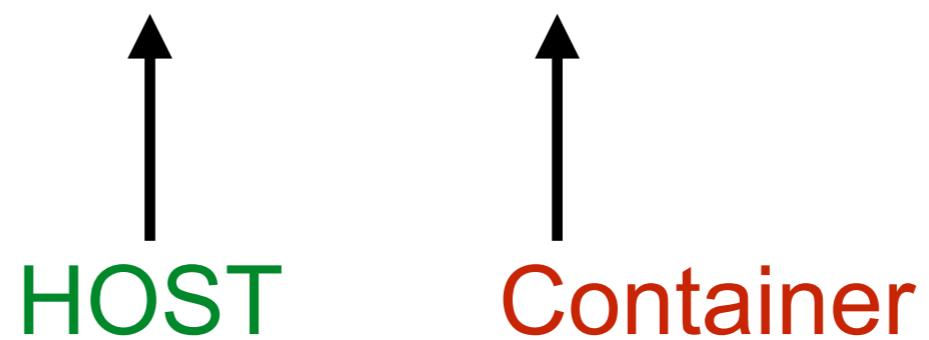


Working with Volume



Volume

-v \$PWD:/data



Named Volume

-v my_volume:/data

Docker volume

Container



Working with volume

\$docker volume ls

\$docker volume create my_volume

\$docker volume inspect my_volume



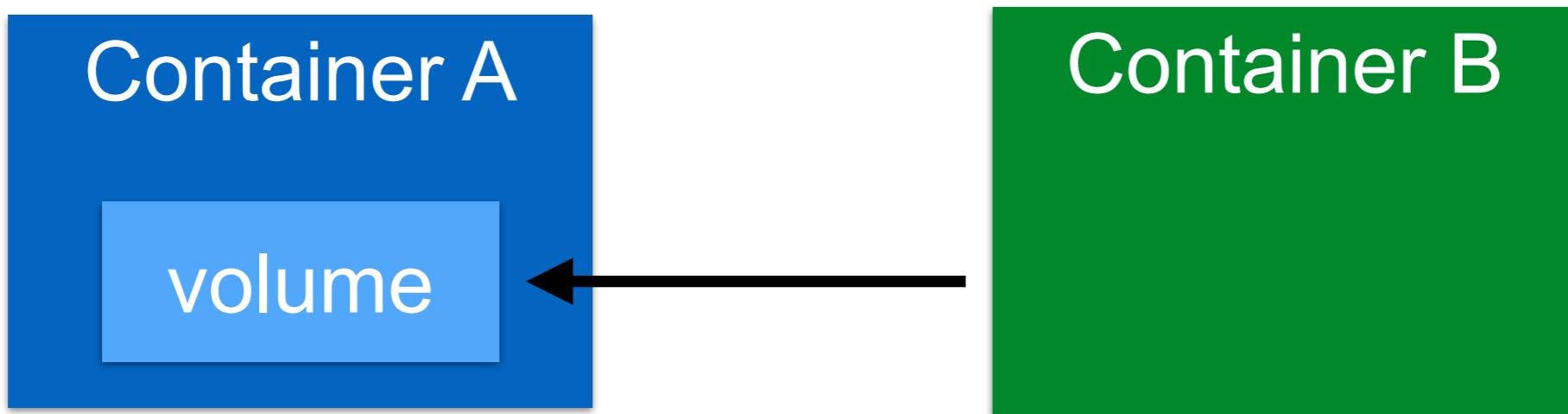
Remove volume

```
$docker volume prune
```

```
$docker volume rm my_volume
```



Volume from container



Create container A

```
$docker run ... -v /var/log --name A
```

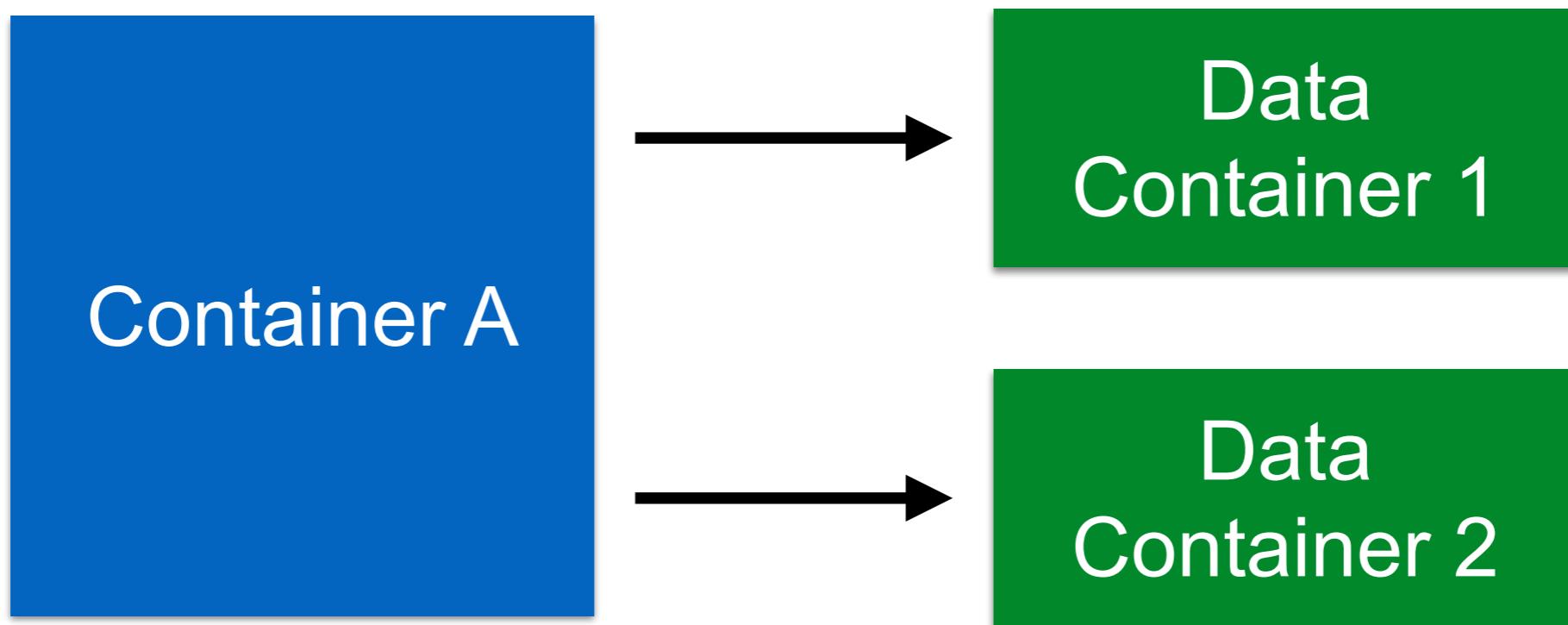


Create container B

```
$docker run ... -v /var/log --name A  
$docker run ... --volumes-from A
```



Volume/Data container



Move data out from mysql container ?



Inspect Image

\$docker inspect mysql:5.7.20

```
"Volumes": {  
    "/var/lib/mysql": {}  
},  
"WorkingDir": "",  
"Entrypoint": [  
    "docker-entrypoint.sh"  
],
```



Working with Docker-compose

<https://docs.docker.com/compose/>



Multiple containers app !!

Difficult to create/manage

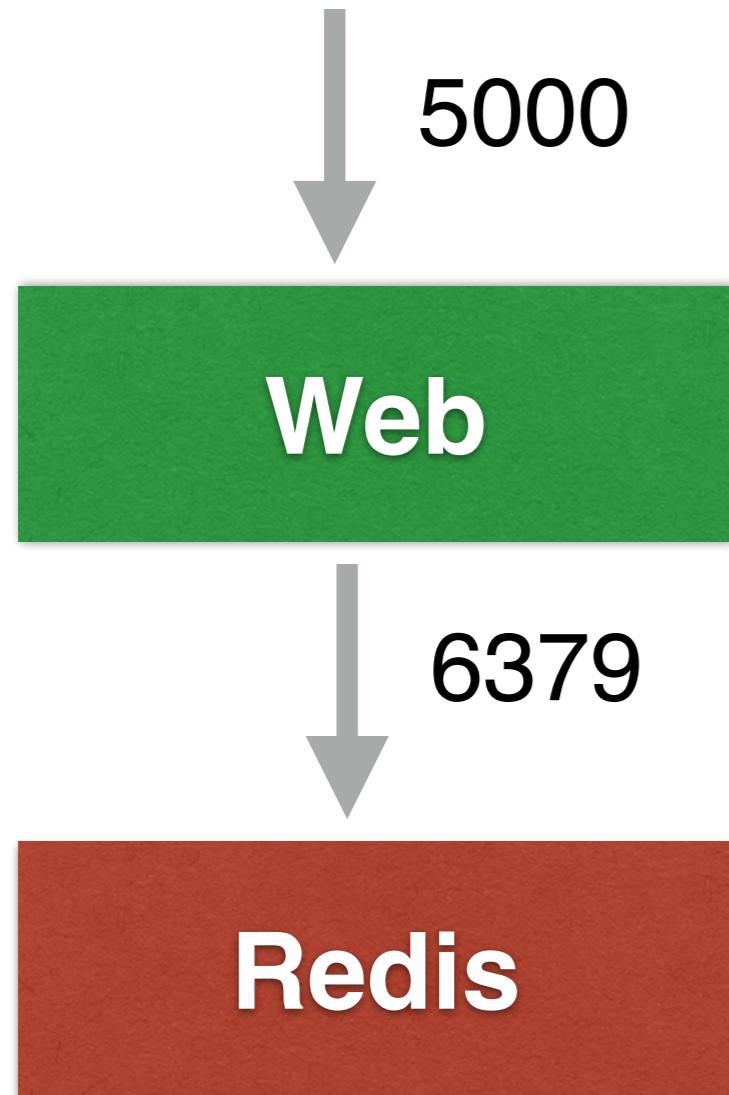


Multiple containers app !!

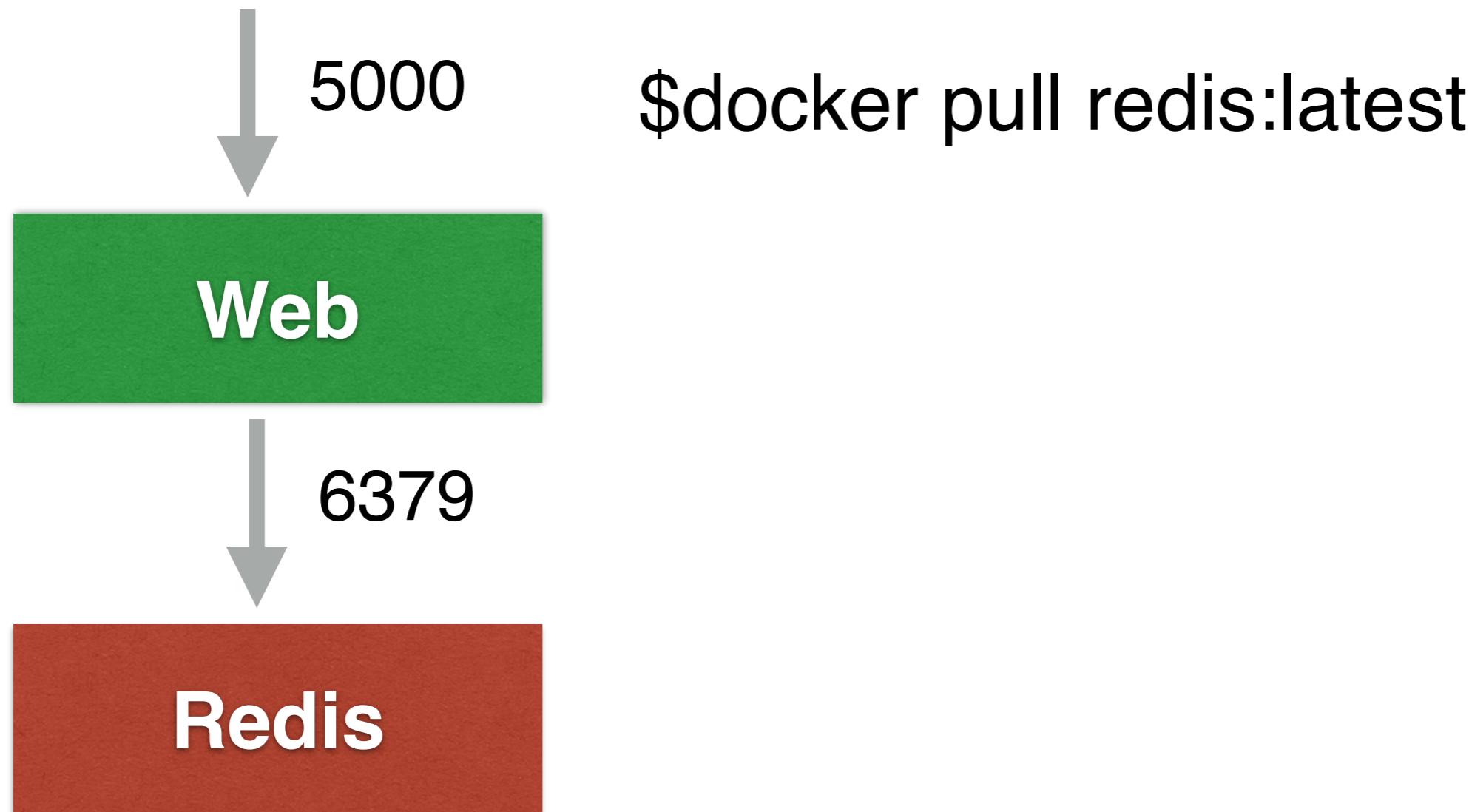
1. Build images from Dockerfile
2. Pull images from Hub/private/cache
3. Configuration and create containers
4. Start and stop containers
5. Stream their logs



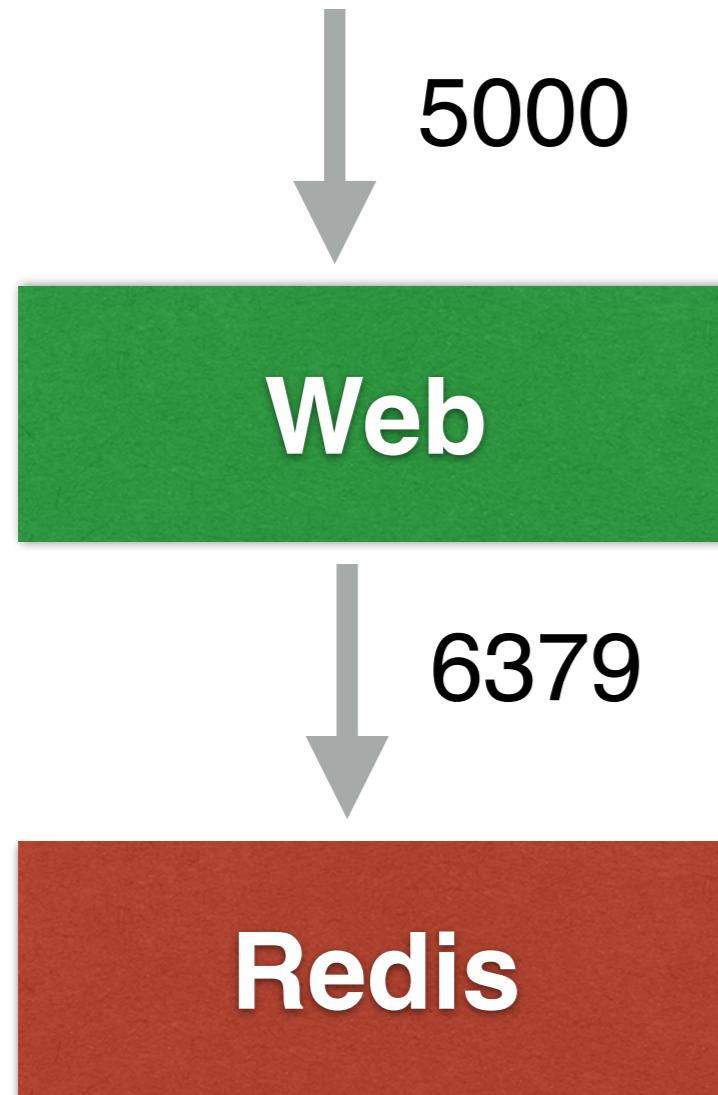
Multiple containers app !!



Multiple containers app !!



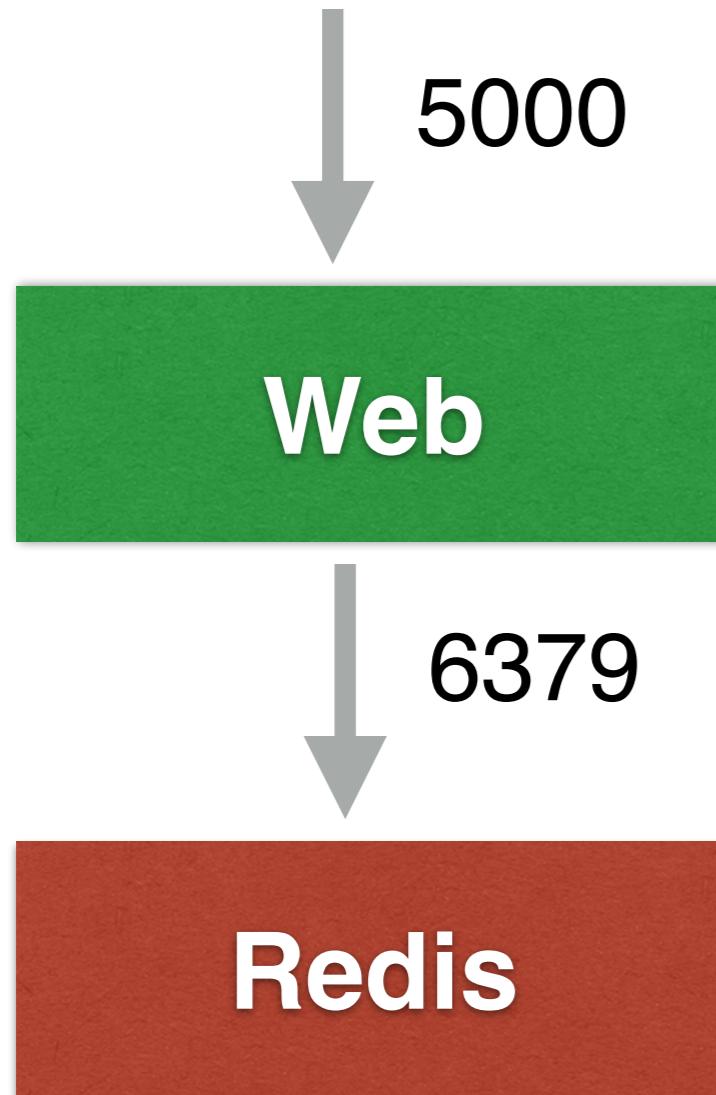
Multiple containers app !!



\$docker pull redis:latest
\$docker build -t web .



Multiple containers app !!



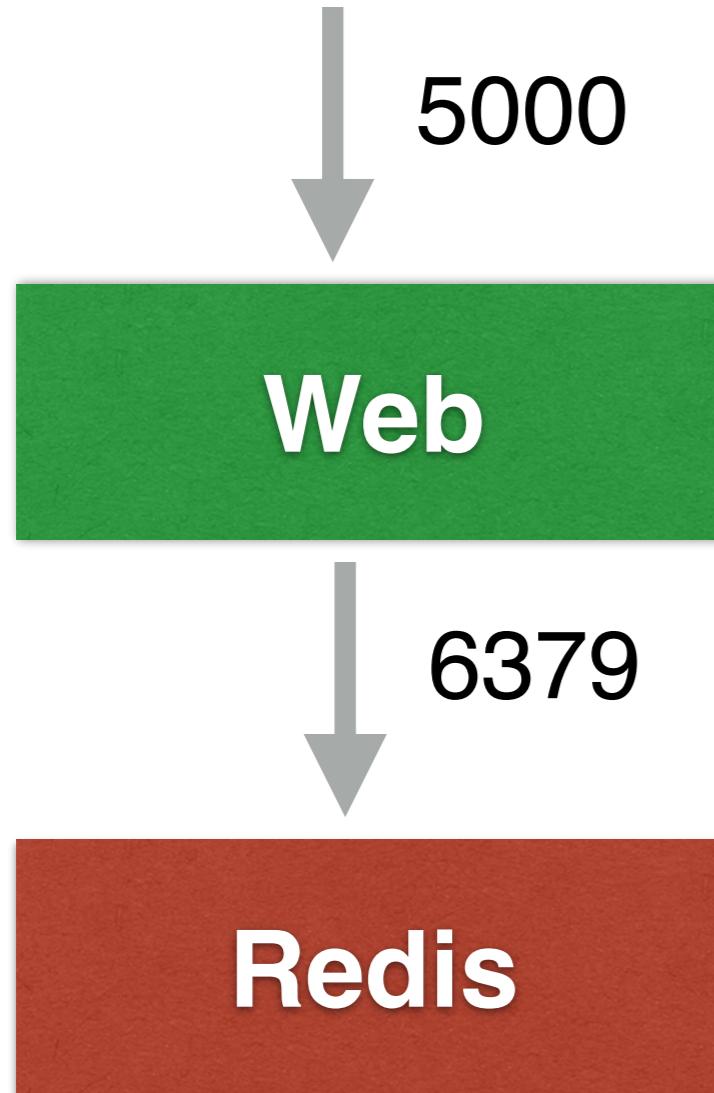
```
$docker pull redis:latest
```

```
$docker build -t web .
```

```
$docker run -d --name db redis \  
redis-server --appendonly yes
```

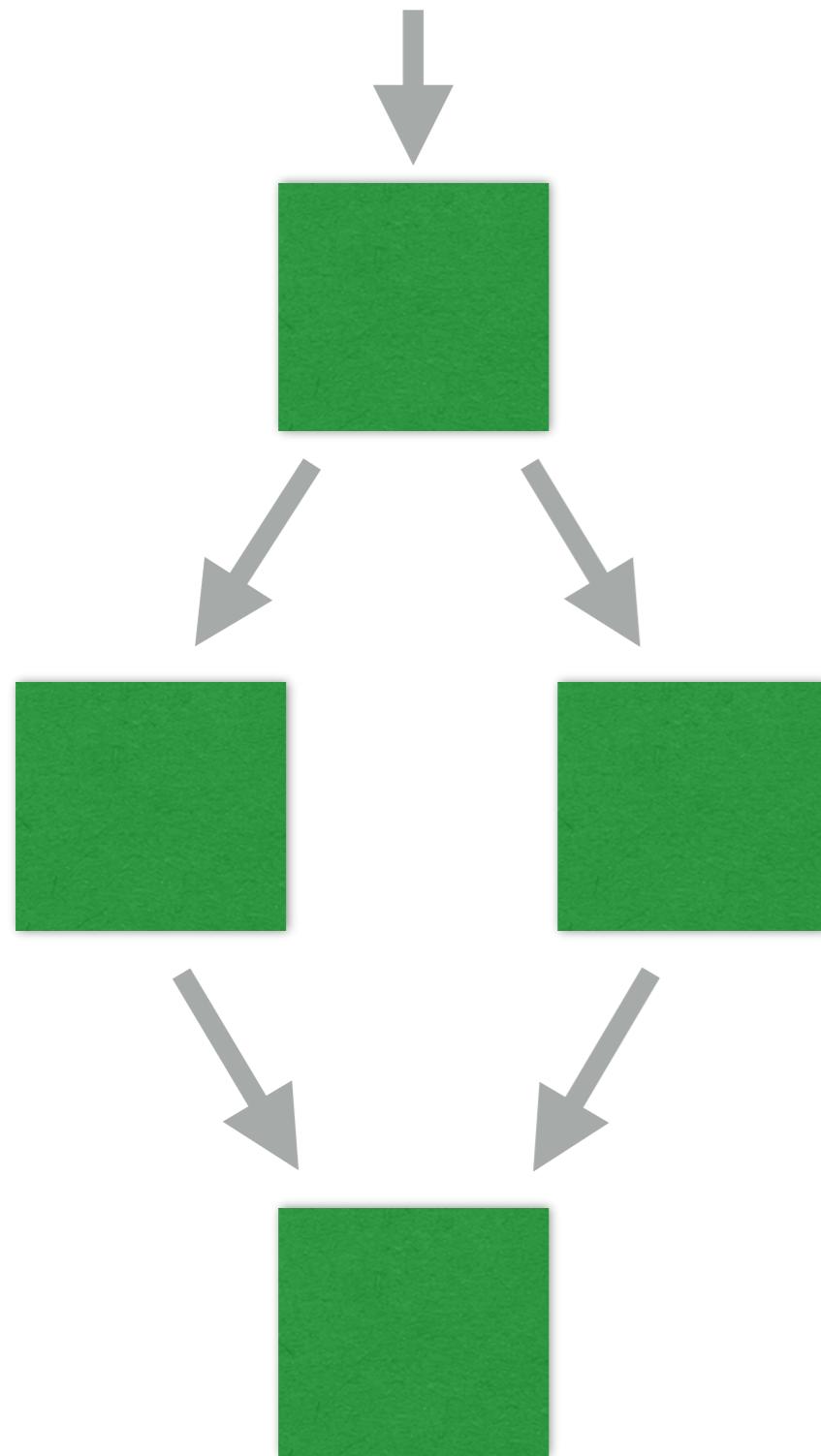


Multiple containers app !!

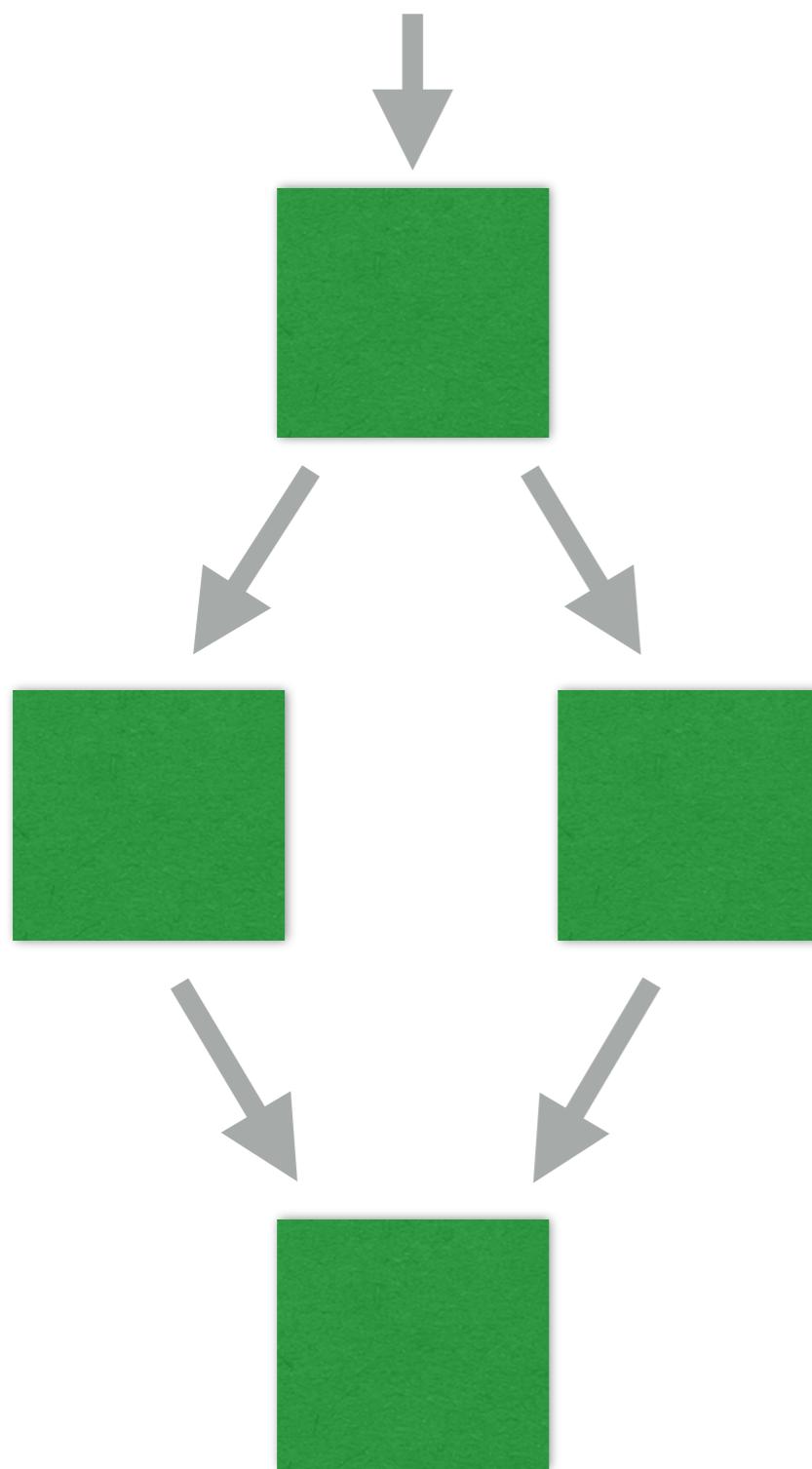


```
$docker pull redis:latest  
$docker build -t web .  
  
$docker run -d --name db redis  
redis-server --appendonly yes  
  
$docker run -d --name web --link  
db:db -p 5000:5000 -e  
REDIS_HOST=db -v $(pwd):/code  
web
```

Multiple containers app !!



Multiple containers app !!



\$docker pull

\$docker build

\$docker build

\$docker build

\$docker run

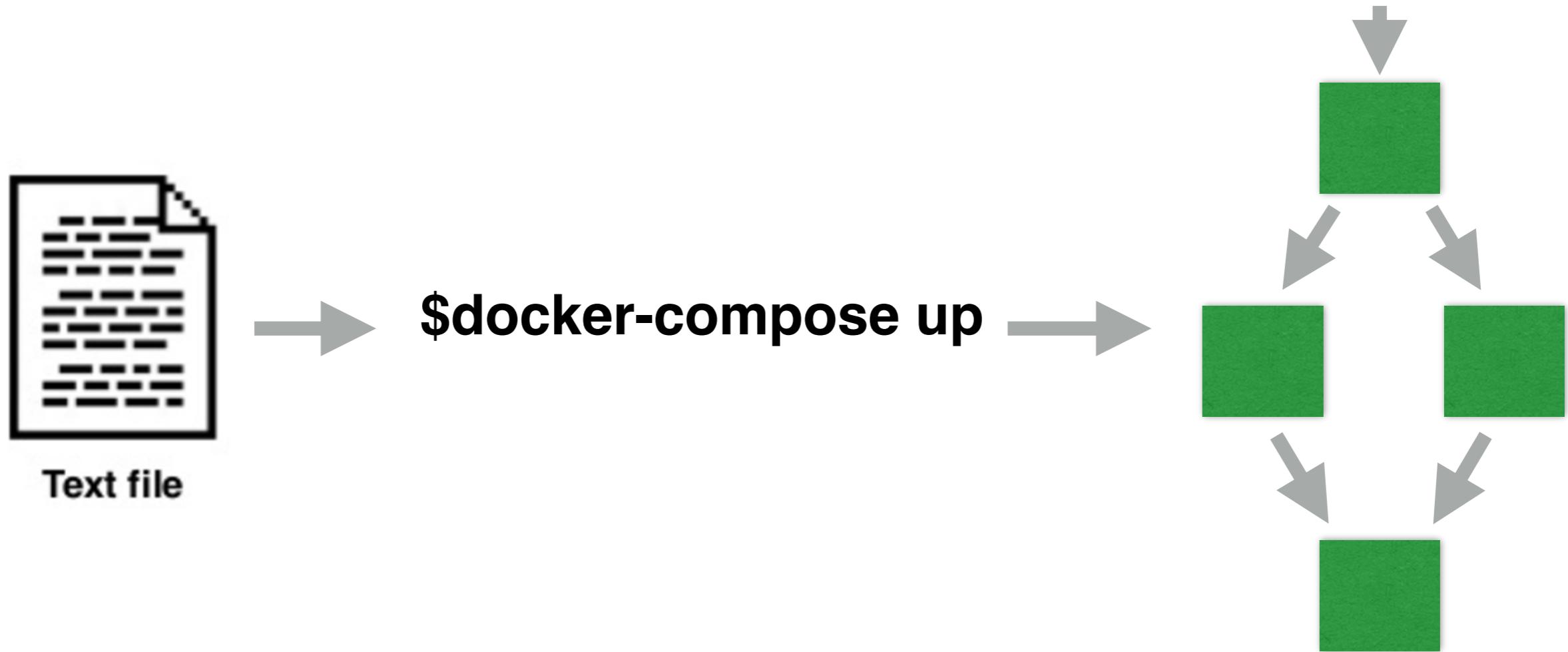
\$docker run

\$docker run



Docker compose

Running app in one command-line



docker-compose.yml

```
version: '3'  
services:  
  web:  
    container_name: web  
    build: .  
    ports:  
      - 80:5000  
    environment:  
      - REDIS_HOST=redis  
  
  redis:  
    container_name: redis  
    image: redis  
    links:  
      - web
```



Build and start

\$docker-compose build

\$docker-compose up -d



See all container

\$docker-compose ps

Name	Command	State	Ports
<hr/>			
redis	docker-entrypoint.sh redis ...	Up	6379/tcp
web	/bin/sh -c python app.py	Up	0.0.0.0:80->5000/tcp



Kill and remove

\$docker-compose kill

\$docker-compose rm

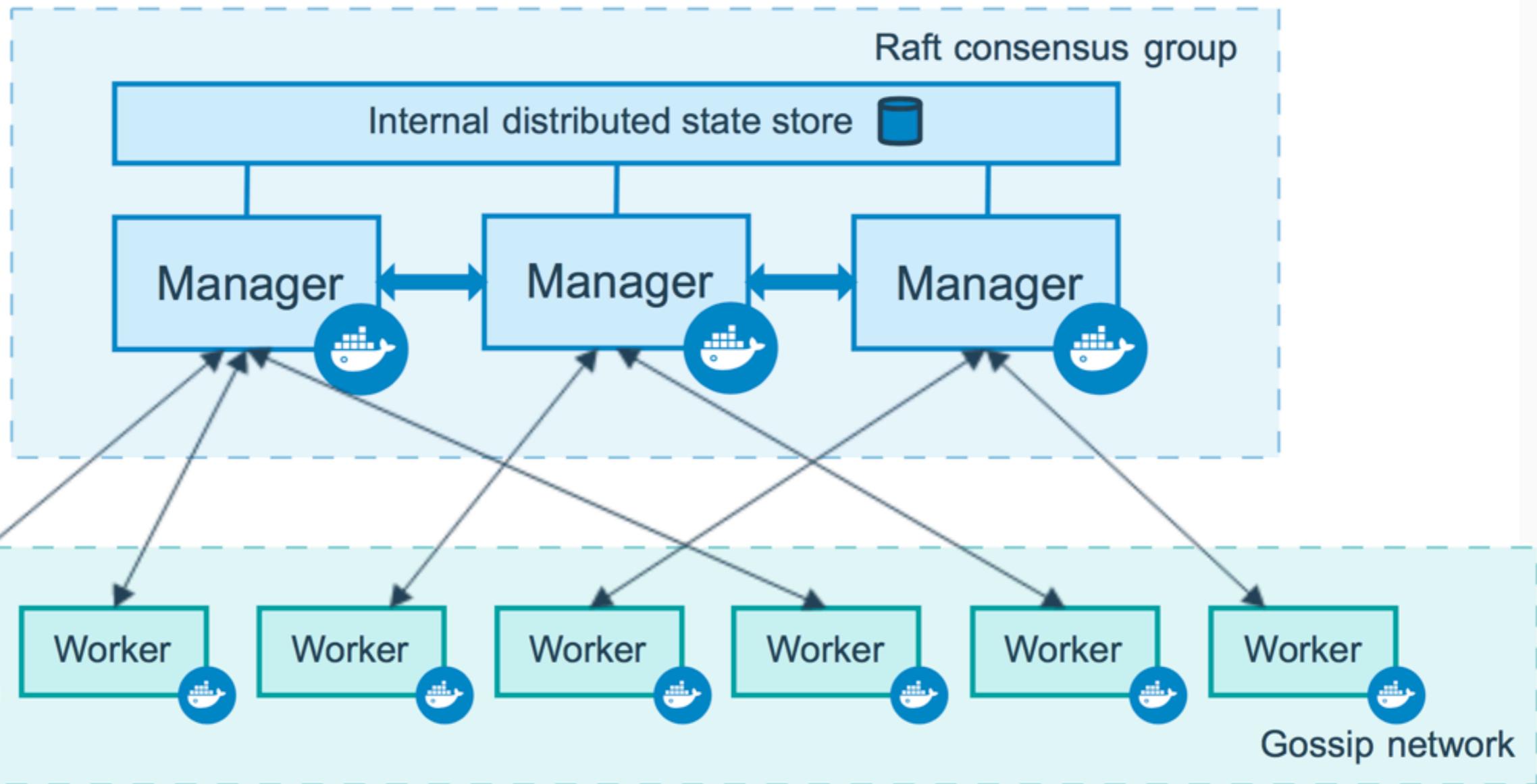
\$docker-compose down



Working with Docker swarm



Swarm



<https://docs.docker.com/engine/swarm/>



2 Types of Node

Manager node
Worker node



Step to Swarm

Docker-compose v3
Initial swarm cluster
Add manager node
Add worker node
Deploy and Scaling



Initial Swarm

```
$ docker swarm init
```

```
$ docker swarm join-token manager
```



Deploy stack to Swarm

```
$ docker stack deploy  
--compose-file=docker-compose-v2.yml  
my_demo
```



Scaling service

```
$ docker service ls
```

```
$ docker service scale my_demo_web=2
```



Goodbye Swarm

```
$ docker stack rm my_demo  
$ docker swarm leave
```



Development workflow



Objectives

Share code between host and container
Simple local development workflow



Workflow

1. Build image contain our dev environment
2. Start container from image + mount volume
3. Edit source code
4. Test my application
5. Repeat 3, 4
6. When done => commit/push code changes



Introduction to Docker Registry

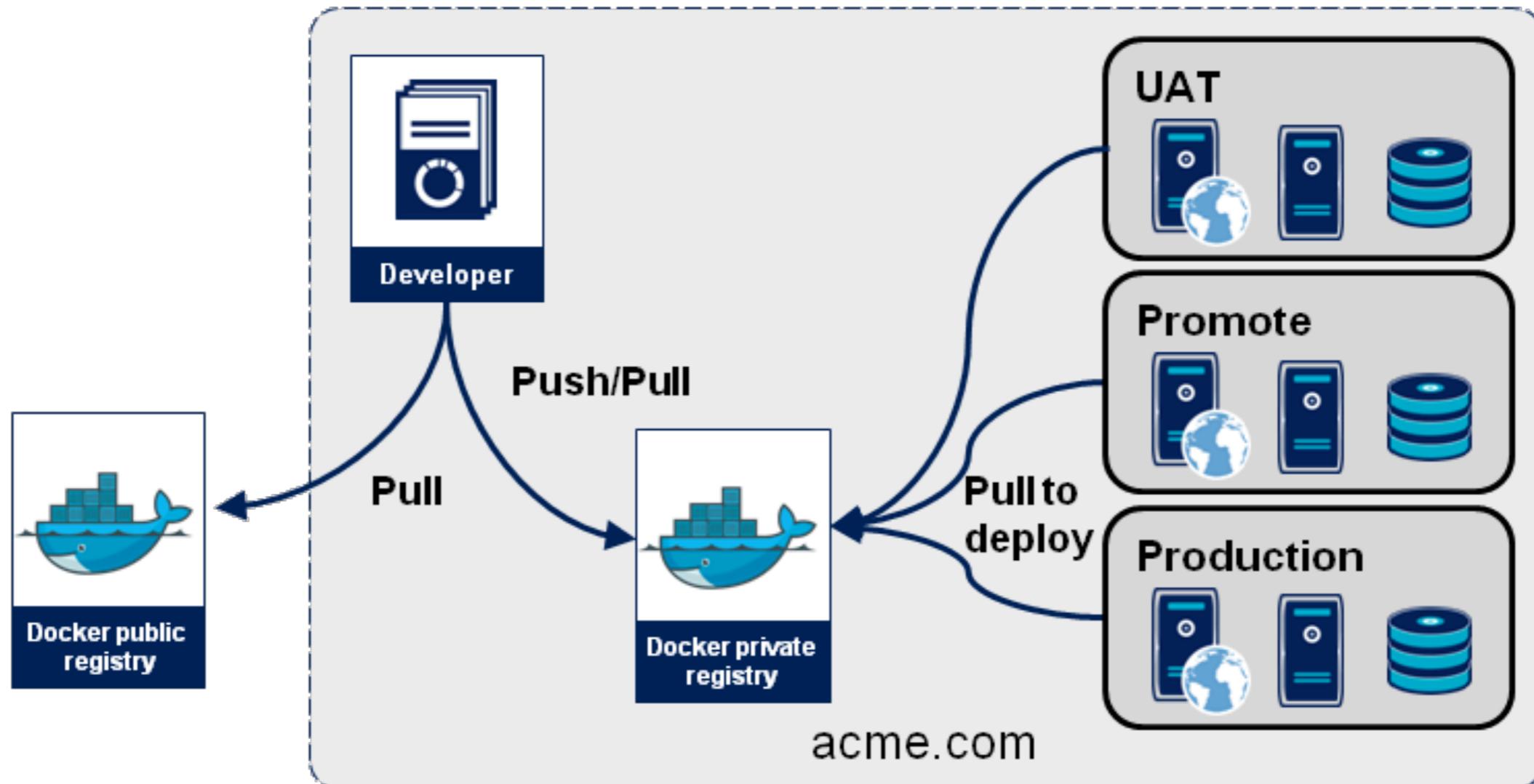


Docker registry

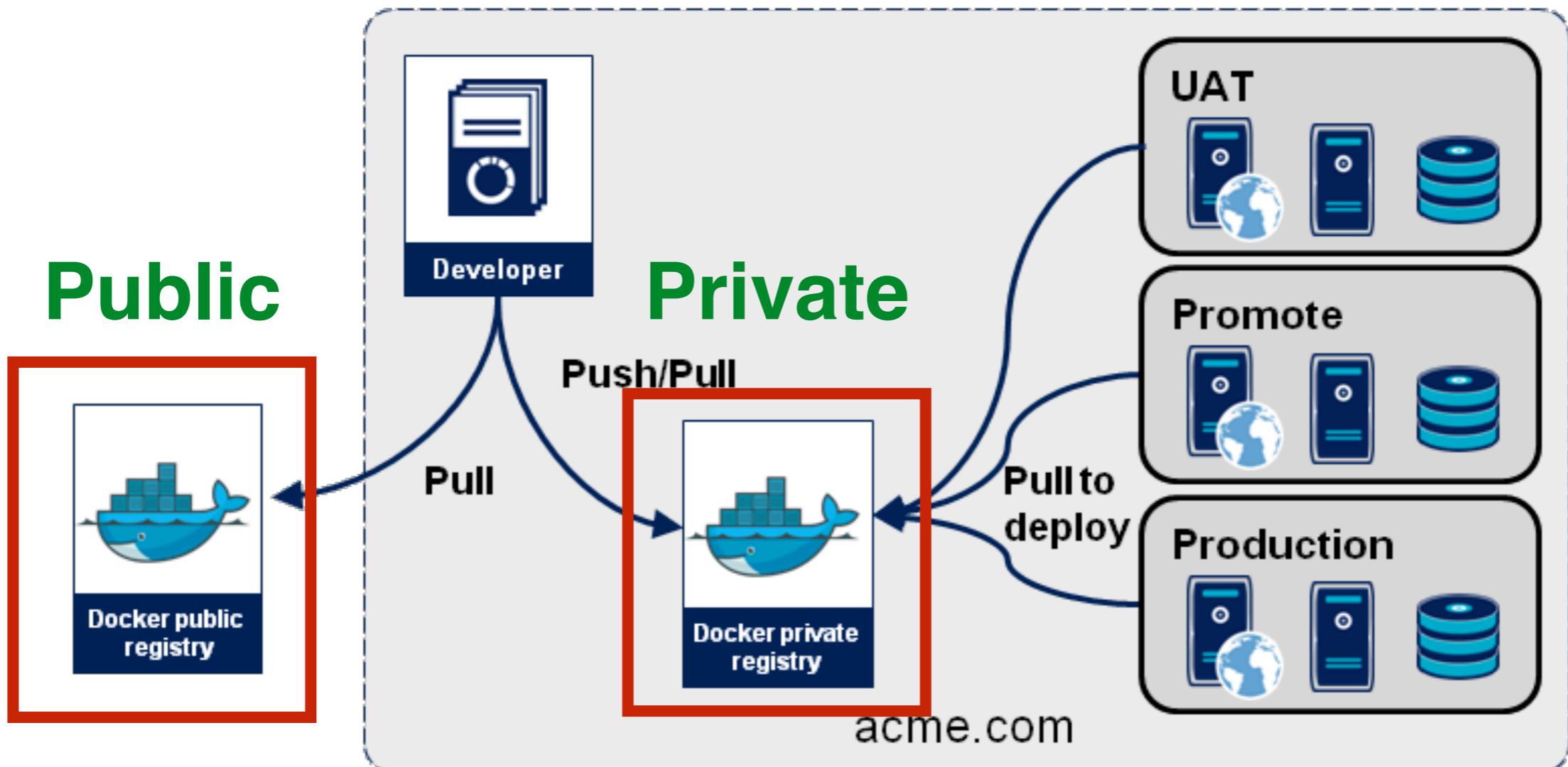
Storage and content delivery system
Holding docker images



Docker registry



Public and private registry



Docker Hub

Default registry of docker image

Contains official repositories from vendors

Support **public** and **private** repositories



Docker official repositories

<https://hub.docker.com/explore/>

The screenshot shows the Docker Hub 'Explore Official Repositories' page. At the top, there is a dark header bar with the Docker logo, a search bar, and navigation links for Dashboard, Explore, Organizations, Create, and a user profile for 'somkiat'. Below the header, the title 'Explore Official Repositories' is displayed in blue. The main content area lists three official Docker repositories in a grid format:

Repository	Stars	Pulls	Action
nginx / official	5.6K STARS	10M+ PULLS	DETAILS
redis / official	3.5K STARS	10M+ PULLS	DETAILS
busybox / official	963 STARS	10M+ PULLS	DETAILS



Introduction to Docker Hub



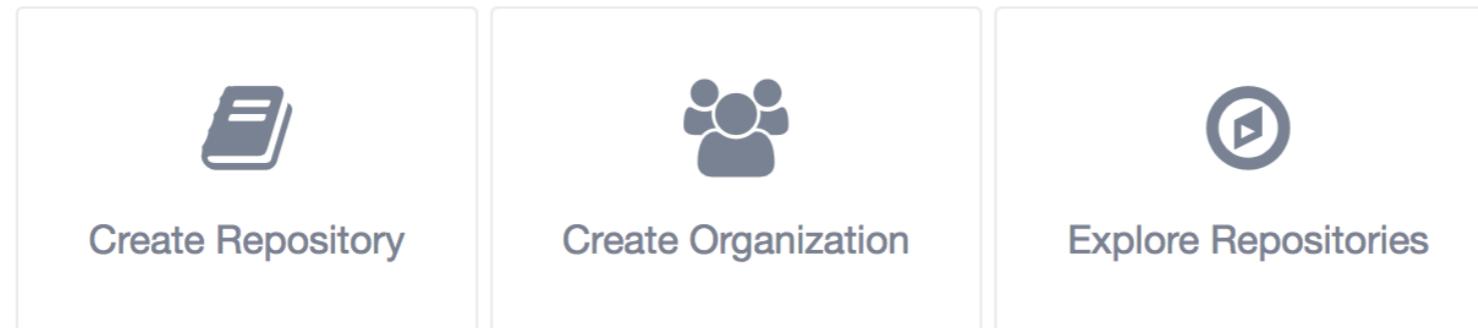
Docker Hub

<https://hub.docker.com/>

The screenshot shows the Docker Hub homepage. At the top is a dark blue header with the Docker logo, a search bar, and navigation links for Dashboard, Explore, Organizations, Create, and a user profile for 'somkiat'. Below the header is a light gray navigation bar with a dropdown for 'somkiat', links for 'Repositories', 'Stars', 'Contributed', and a message about private repositories.

Welcome to Docker Hub

Here are a few things to get you started.



Docker Store

<https://store.docker.com/>



364

บริษัท สยามชำนาญกิจ จำกัด และเพื่อนพ้องน้องพี่

Objectives

Register account on Docker Hub

Login to your account from command line

Create your own image

Push image to Docker Hub



Login to docker hub

\$docker login

```
Login with your Docker ID to push and pull images. If you don't have a Docker ID, head over to http://hub.docker.com
Username (somkiat):
```

```
Password:
```

```
Login Succeeded
```



Docker registry authentication credentials

\$cat ~/.docker/config.json

```
{  
  "auths": {  
    "https://index.docker.io/v1/": {  
      "auth": "c29ta2lhdDo0MjExMjEyM  
    }  
  }%  
}
```



Build image from scratch

```
$docker image build -t <account>/<image name>
```



Re-tag to existing image

```
$docker tag <image> <account>/<image>
```



Push image to docker hub

```
$docker push <account>/<image>
```



My public image

The screenshot shows a GitHub profile page for the user 'somkiat'. The top navigation bar includes links for Dashboard, Explore, Organizations, Create, and a user dropdown. Below the navigation, there's a search bar and a dropdown menu showing the user's repositories. The main section is titled 'Repositories' and features a large blue button to 'Create Repository +'. A search bar allows filtering by repository name. One repository is listed: 'somkiat/customphp' (public), which has 0 stars and 1 pull request. A 'DETAILS' button is next to it.

Dashboard Explore Organizations Create

Search

somkiat

Repositories Stars Contributed

Private Repositories: Using 0 of 1 Get more

Repositories

Type to filter repositories by name

Create Repository +

 somkiat/customphp
public

0 STARS 1 PULLS DETAILS



Pull image from docker hub

```
$docker pull <account>/<image>
```



Registry proxy cache

Store images locally

Reduce redundant images

Reduce bandwidth

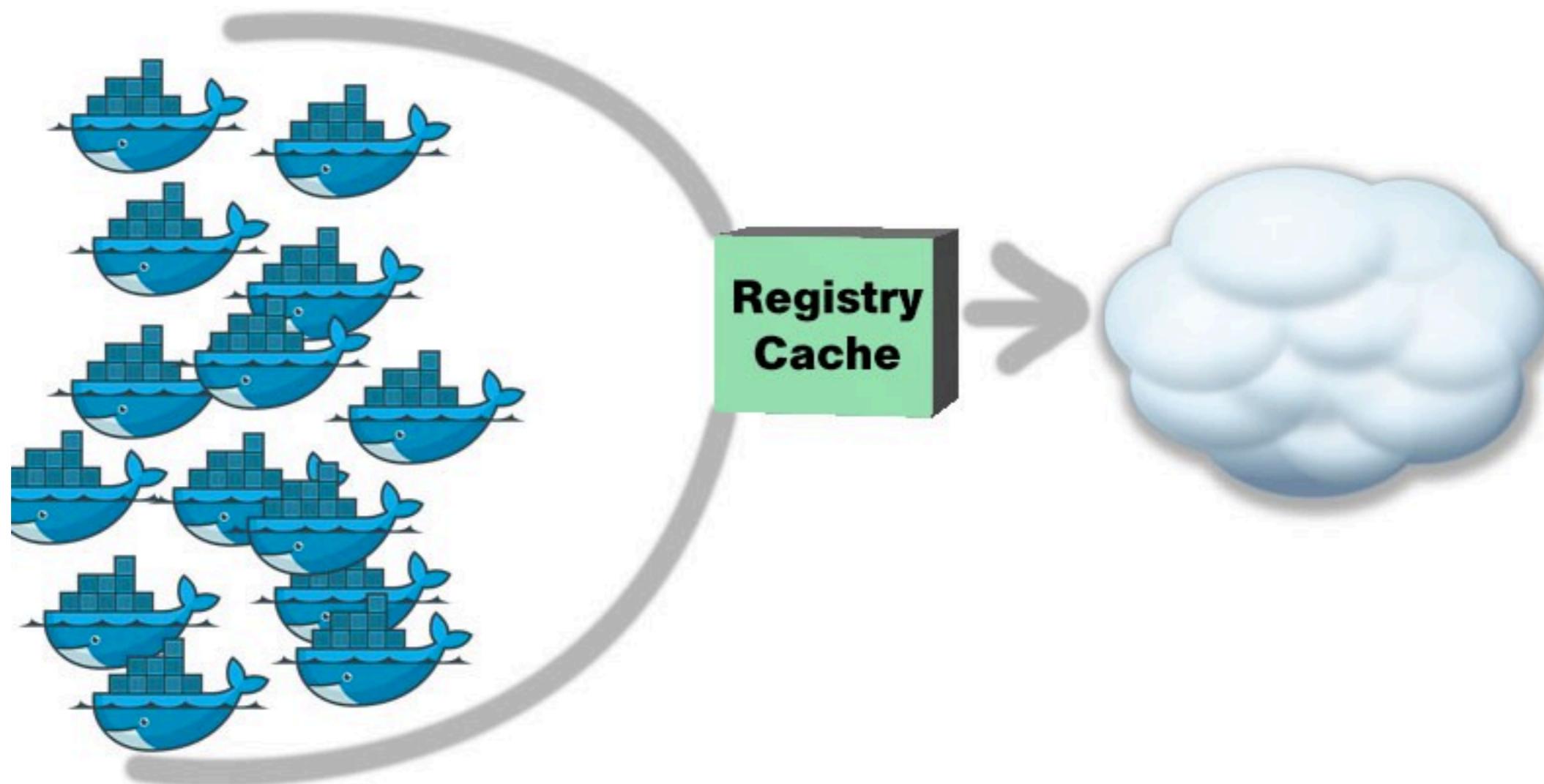
More speed !!

<https://docs.docker.com/registry/deploying/>

<https://blog.docker.com/2015/10/registry-proxy-cache-docker-open-source/>



Registry proxy cache



Start registry v2

```
$ docker container run -d --name registry \
-p 5000:5000 \
--restart=always \
registry:2
```

<https://docs.docker.com/registry/>



List of all images

`http://localhost:5000/v2/_catalog`



List of tag by image

http://localhost:5000/v2/<image name>/tags/list



The screenshot shows a browser window with the URL `localhost:5000/v2/customphp/tags/list` in the address bar. The page displays a JSON object representing the image's metadata:

```
{  
  "name": "customphp",  
  - "tags": [  
    "latest",  
    "0.1"  
  ]  
}
```



Stop registry v2

```
$docker stop registry  
$docker rm -v registry
```



Resources

<https://github.com/up1/course-introduction-docker>

