
# สถาบัน ไอเอ็มซี

# Learning path

Introduction

Basic of Docker

Building containers

Running web apps with Docker

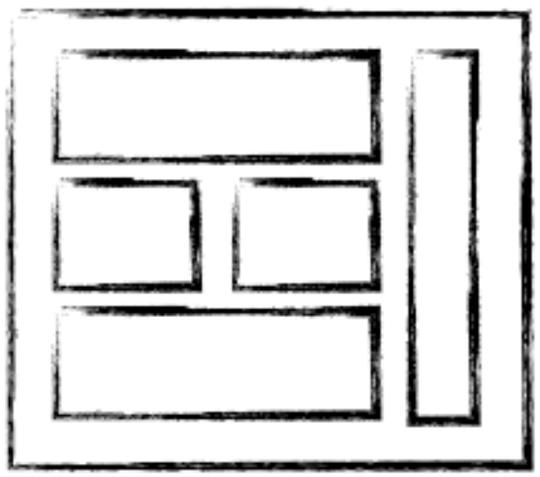
Docker automation

# Introduction

# Why docker ?

# Why docker ?

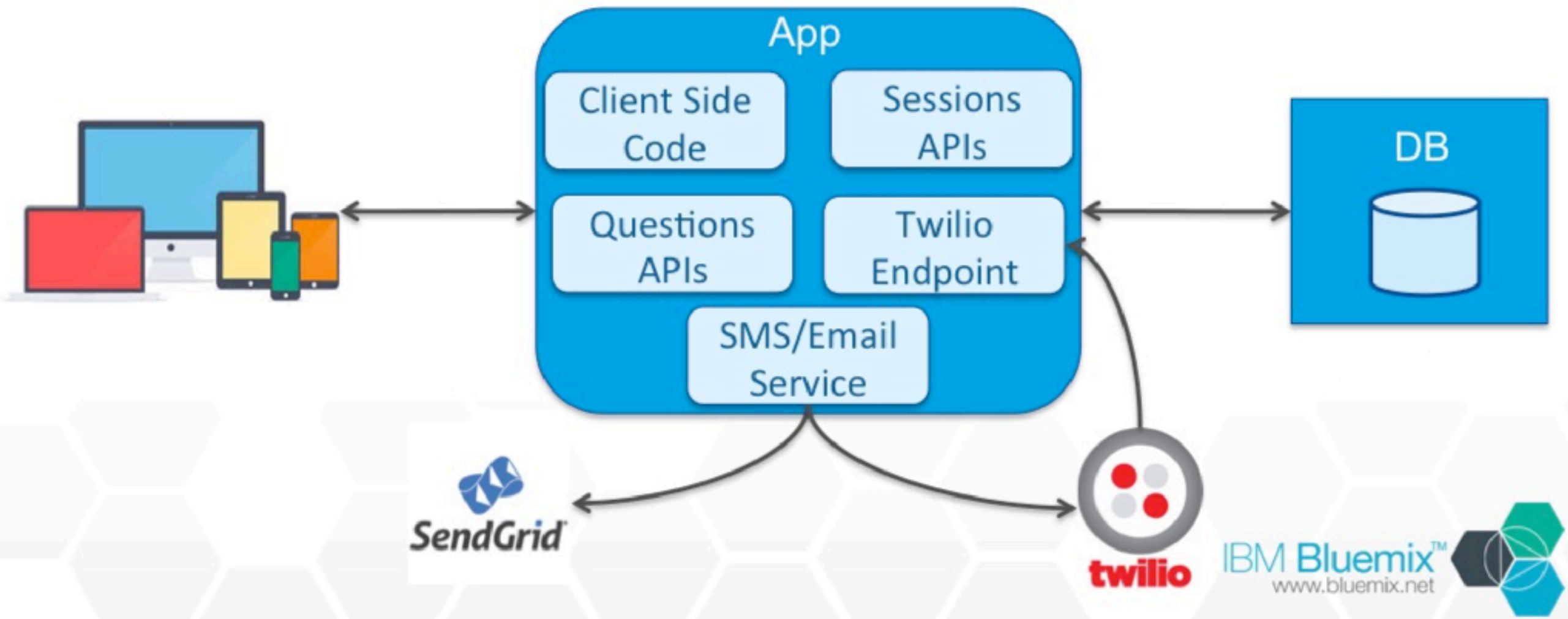
Software industry has changed !!



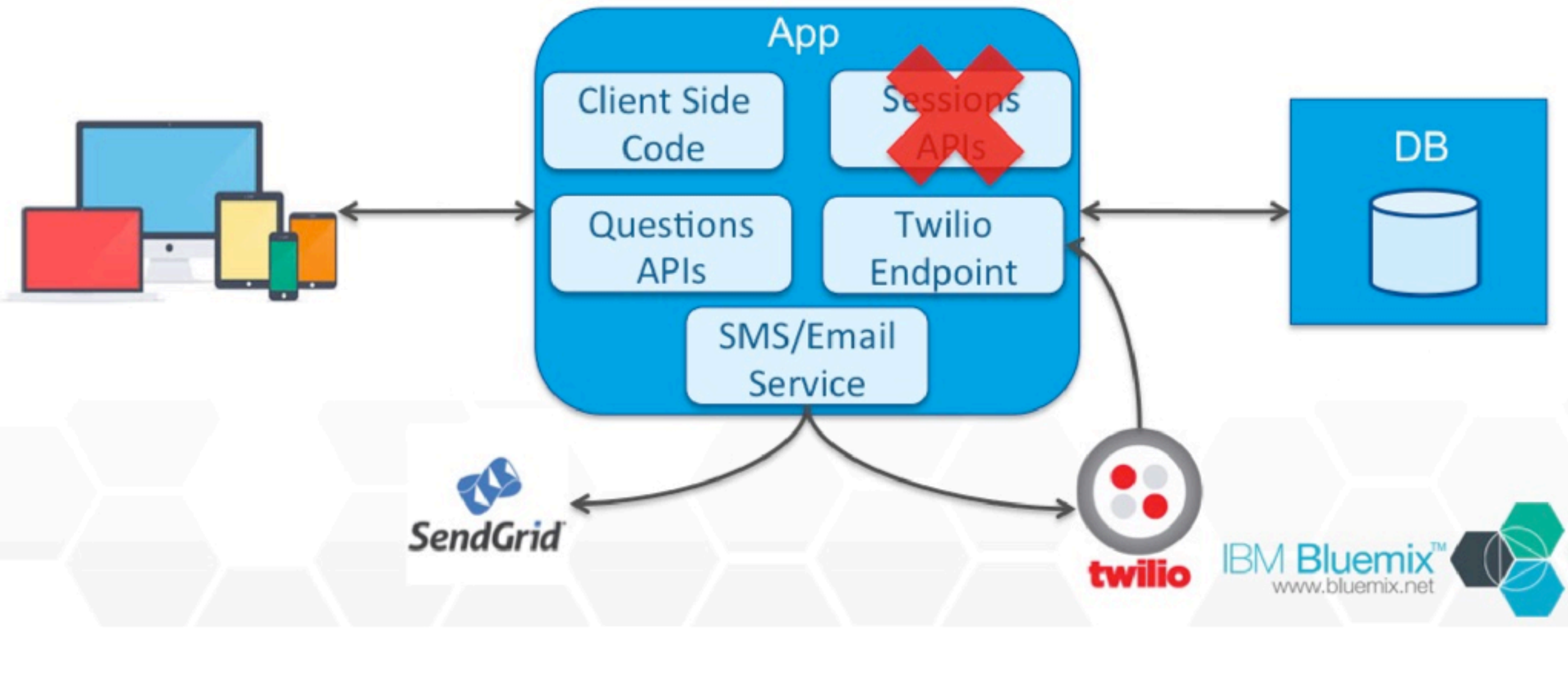
MONOLITHIC/LAYERED

MICRO SERVICES

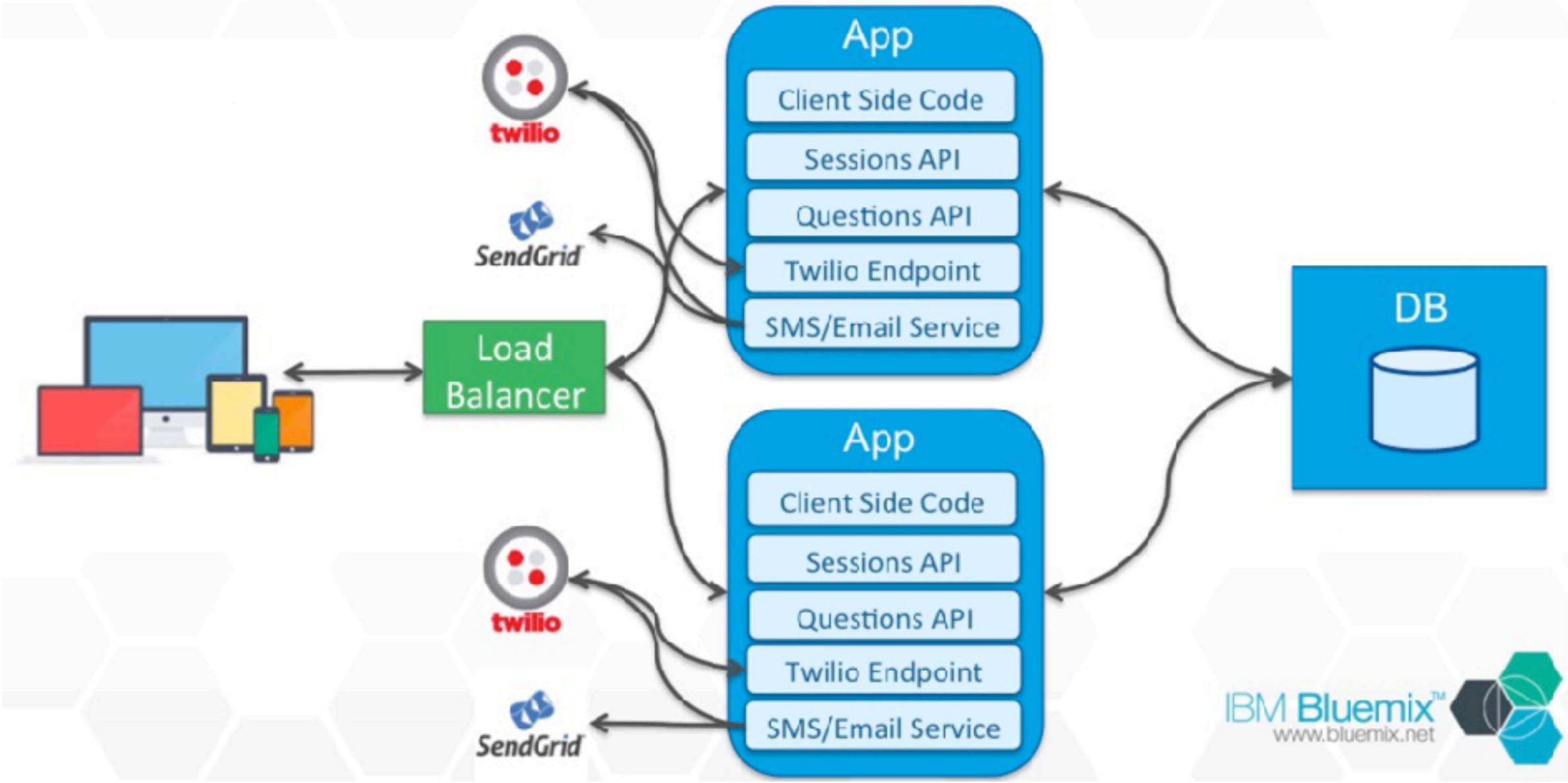
# Monolith architecture



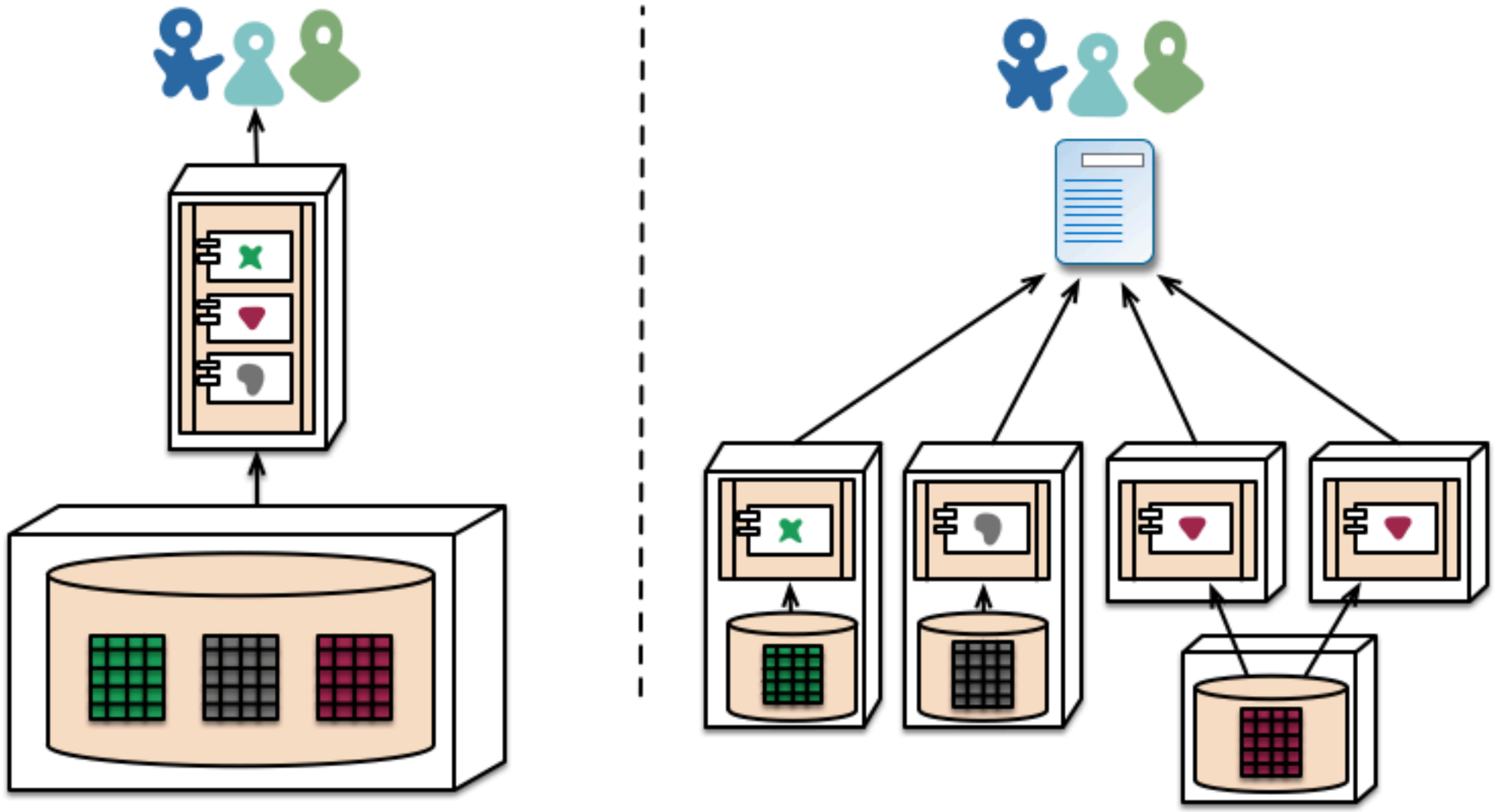
# Failure in Monolith !!



# Scale Monolith !!



# Microservice architecture

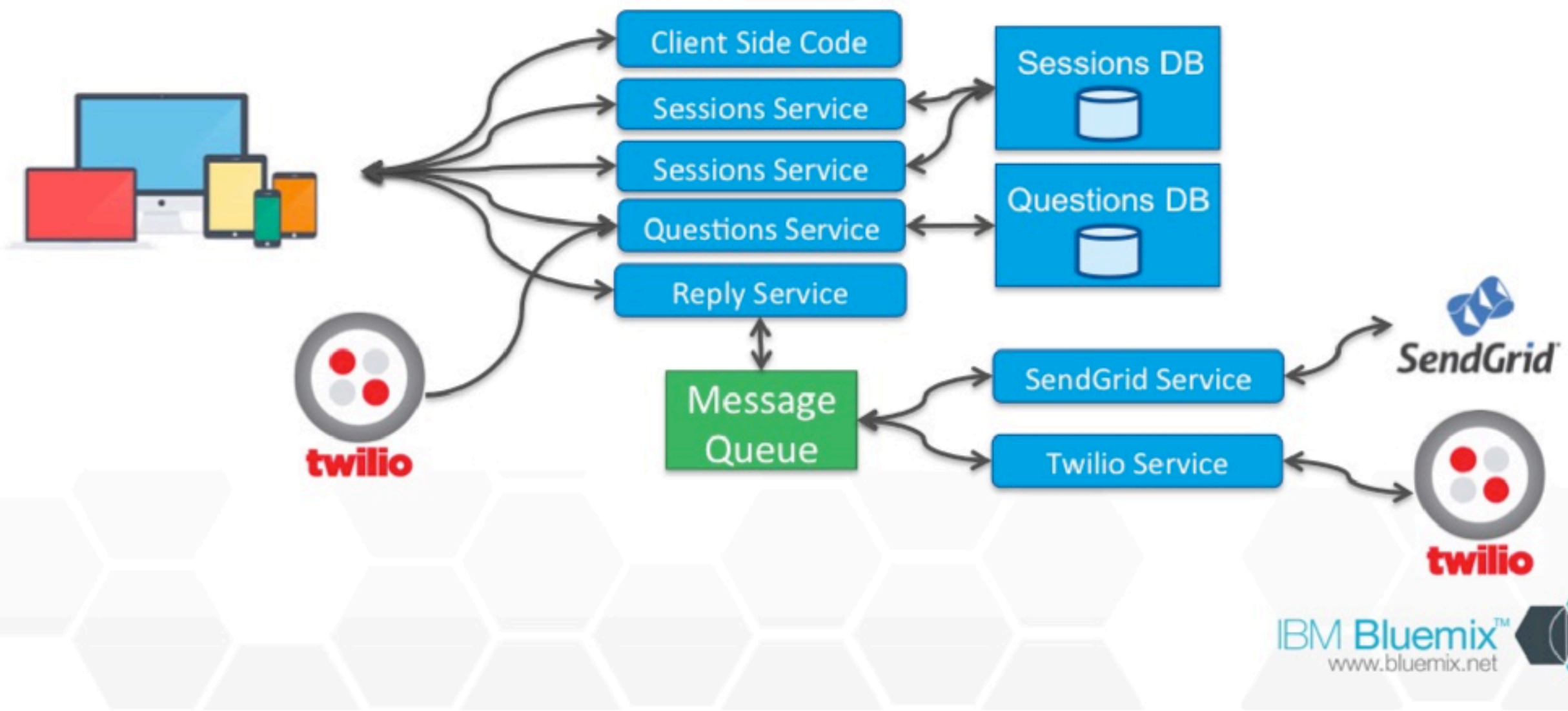


monolith - single database

microservices - application databases

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

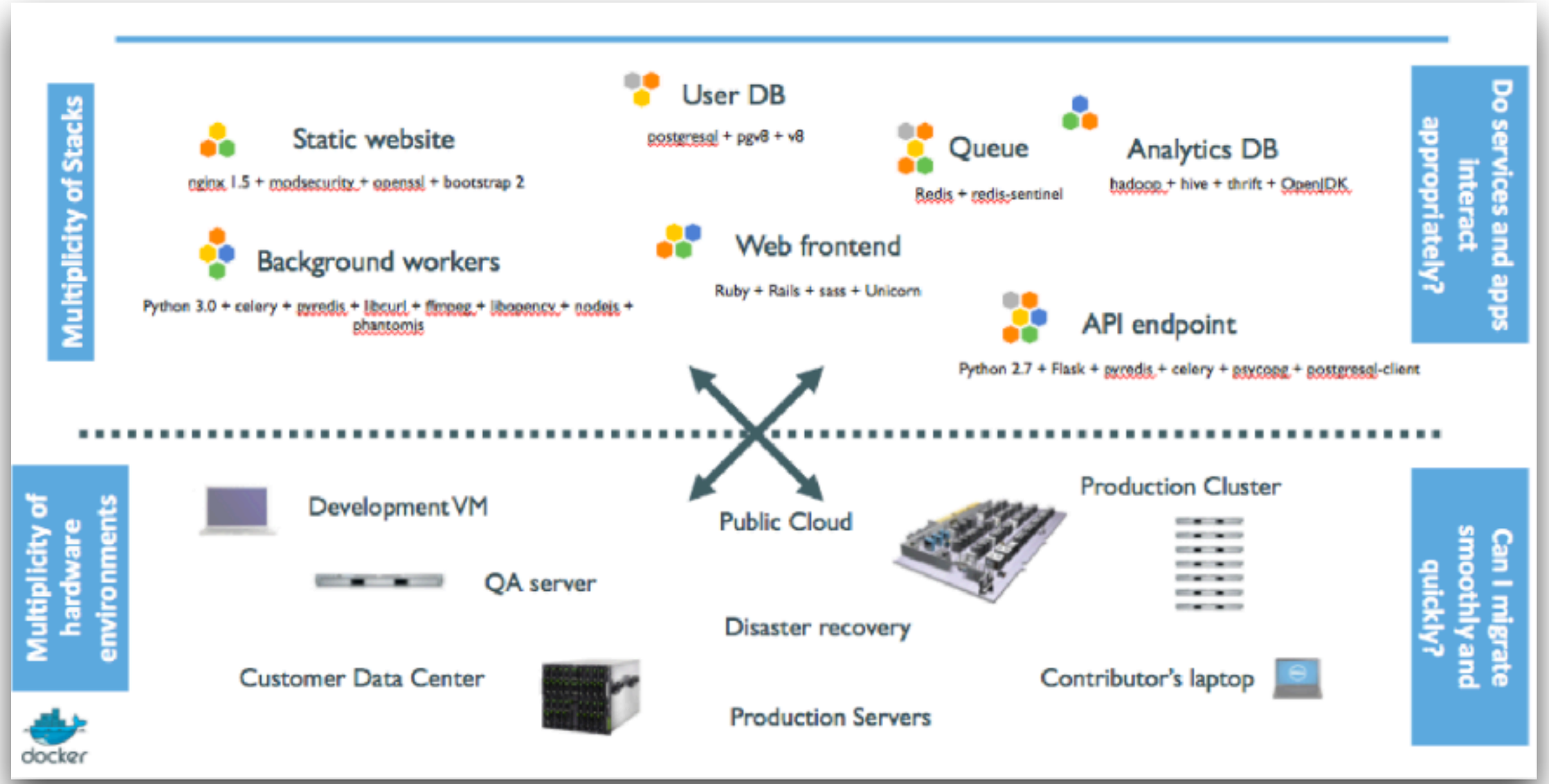
# Microservice architecture



# Why docker ?

We have problem !!

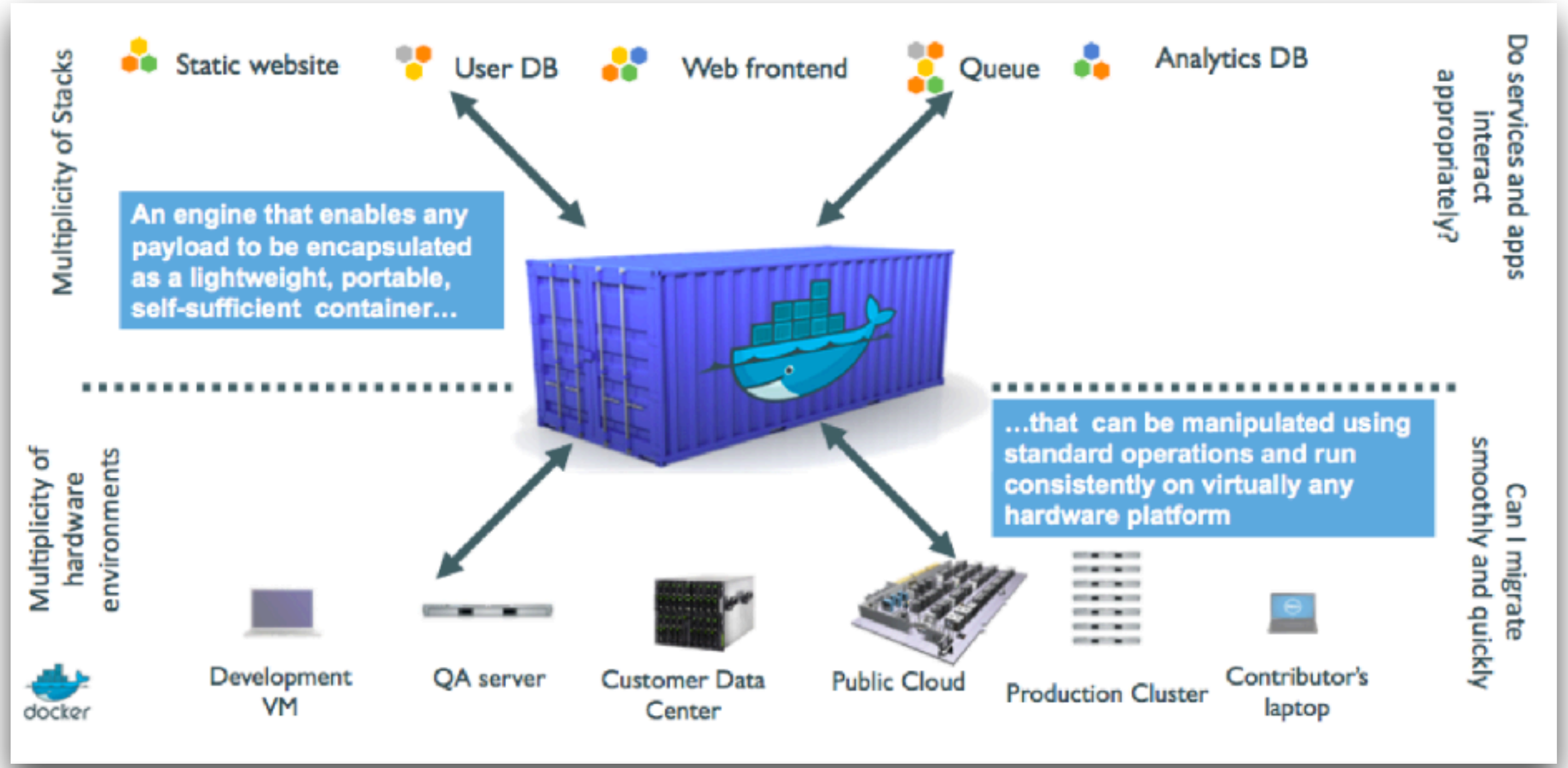
# Problem


Multiplicity of Stacks
Static website
nginx 1.5 + modsecurity + openssl + bootstrap 2
User DB
postgresql + pgv8 + v8
Queue
Redis + redis-sentinel
Analytics DB
hadoop + hive + thrift + OpenJDK
Background workers
Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs
Web frontend
Ruby + Rails + sass + Unicorn
API endpoint
Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client
Do services and apps interact appropriately?
Multiplicity of hardware environments
Development VM
QA server
Customer Data Center
Public Cloud
Disaster recovery
Production Servers
Production Cluster
Contributor's laptop
Can I migrate smoothly and quickly?
docker

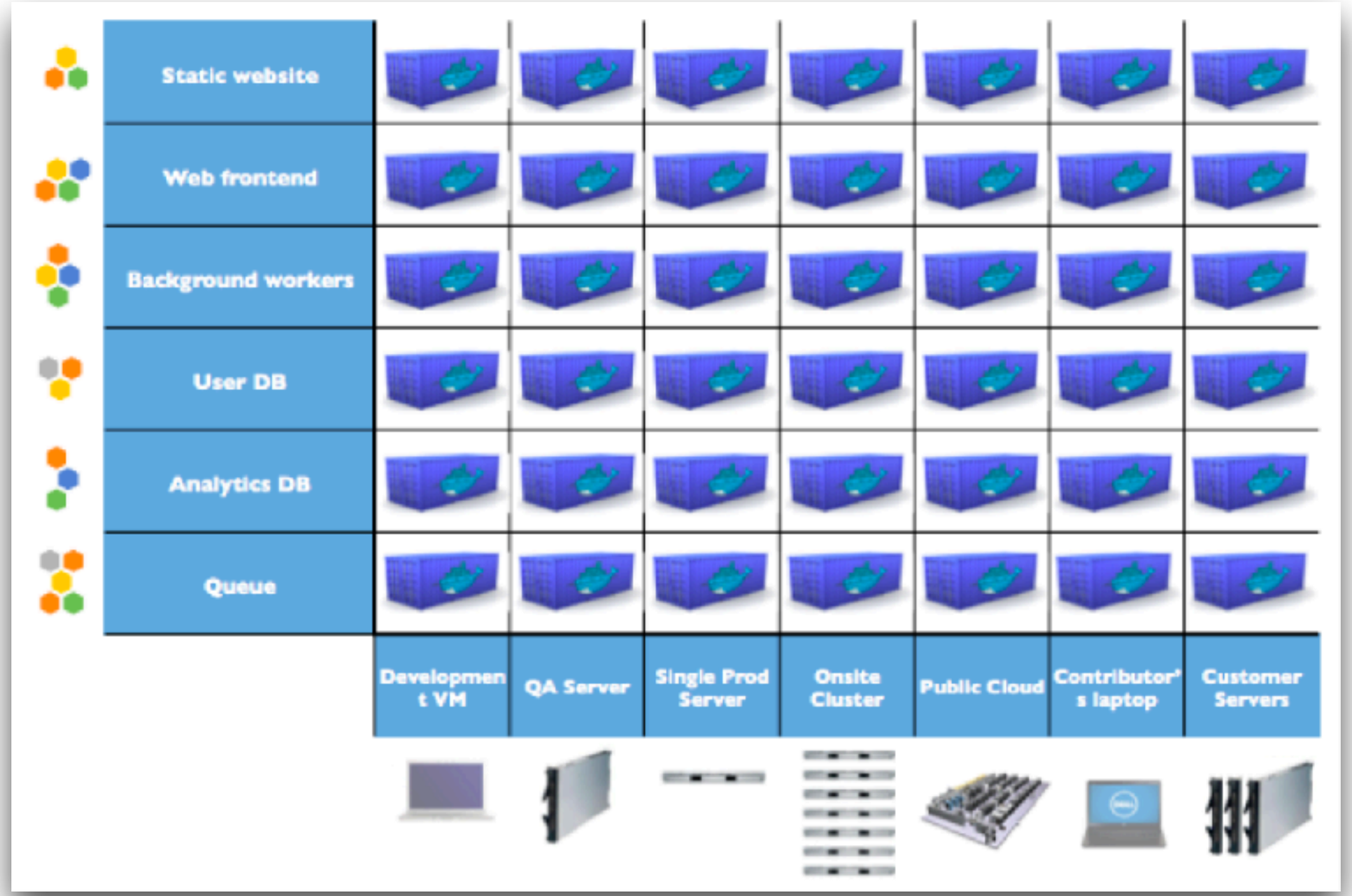
# Problem

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Static website | ? | ? | ? | ? | ? | ? | ? |
| Web frontend | ? | ? | ? | ? | ? | ? | ? |
| Background workers | ? | ? | ? | ? | ? | ? | ? |
| User DB | ? | ? | ? | ? | ? | ? | ? |
| Analytics DB | ? | ? | ? | ? | ? | ? | ? |
| Queue | ? | ? | ? | ? | ? | ? | ? |
| | Developmen t VM | QA Server | Single Prod Server | Onsite Cluster | Public Cloud | Contributor' s laptop | Customer Servers |

# Solution


Multiplicity of Stacks
Static website
User DB
Web frontend
Queue
Analytics DB
Do services and apps interact appropriately?
An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container...
...that can be manipulated using standard operations and run consistently on virtually any hardware platform
Multiplicity of hardware environments
Can I migrate smoothly and quickly
docker
Development VM
QA server
Customer Data Center
Public Cloud
Production Cluster
Contributor's laptop

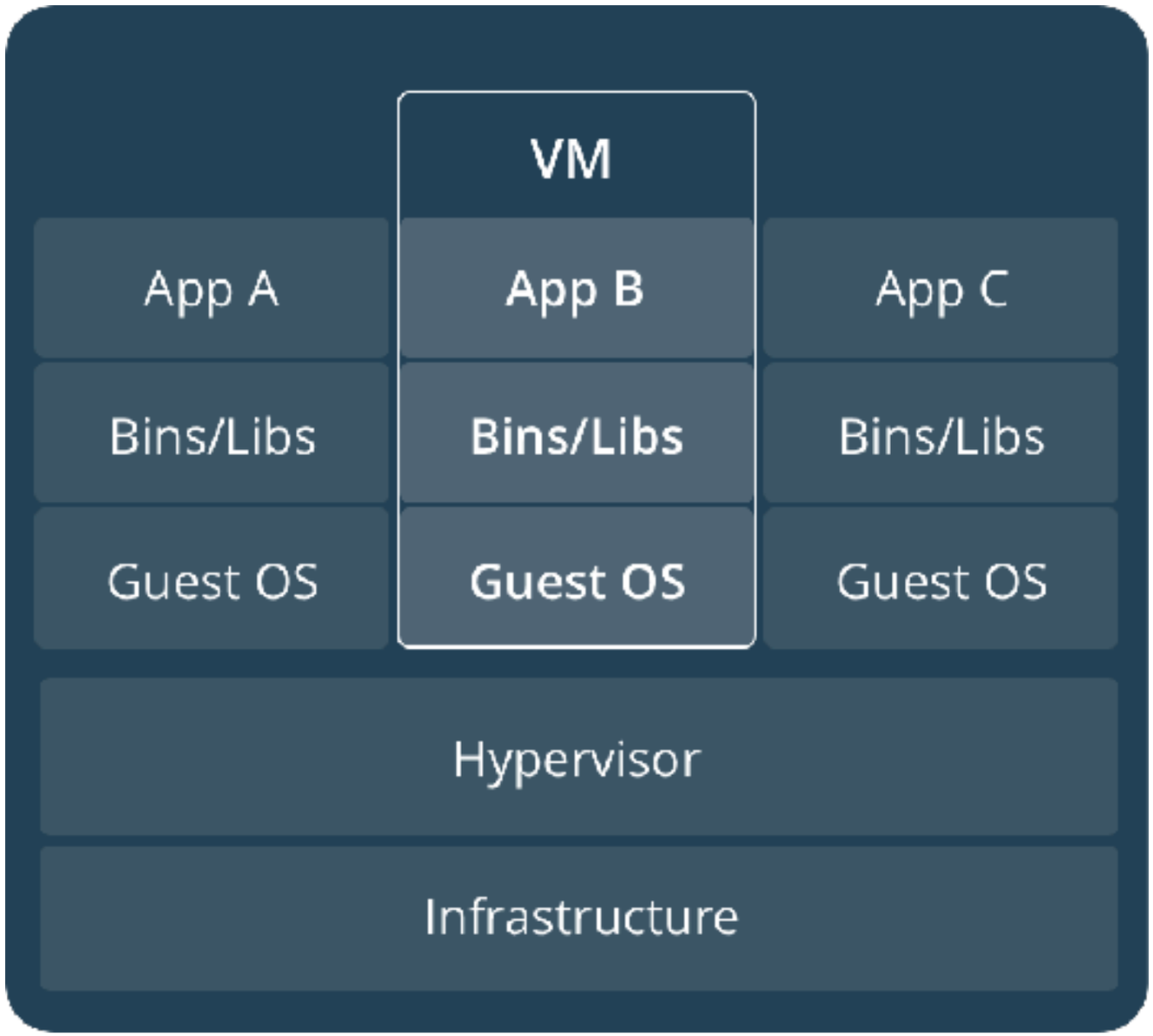
# Solution


Static website
Web frontend
Background workers
User DB
Analytics DB
Queue
Developmen t VM
QA Server
Single Prod Server
Onsite Cluster
Public Cloud
Contributor' s laptop
Customer Servers

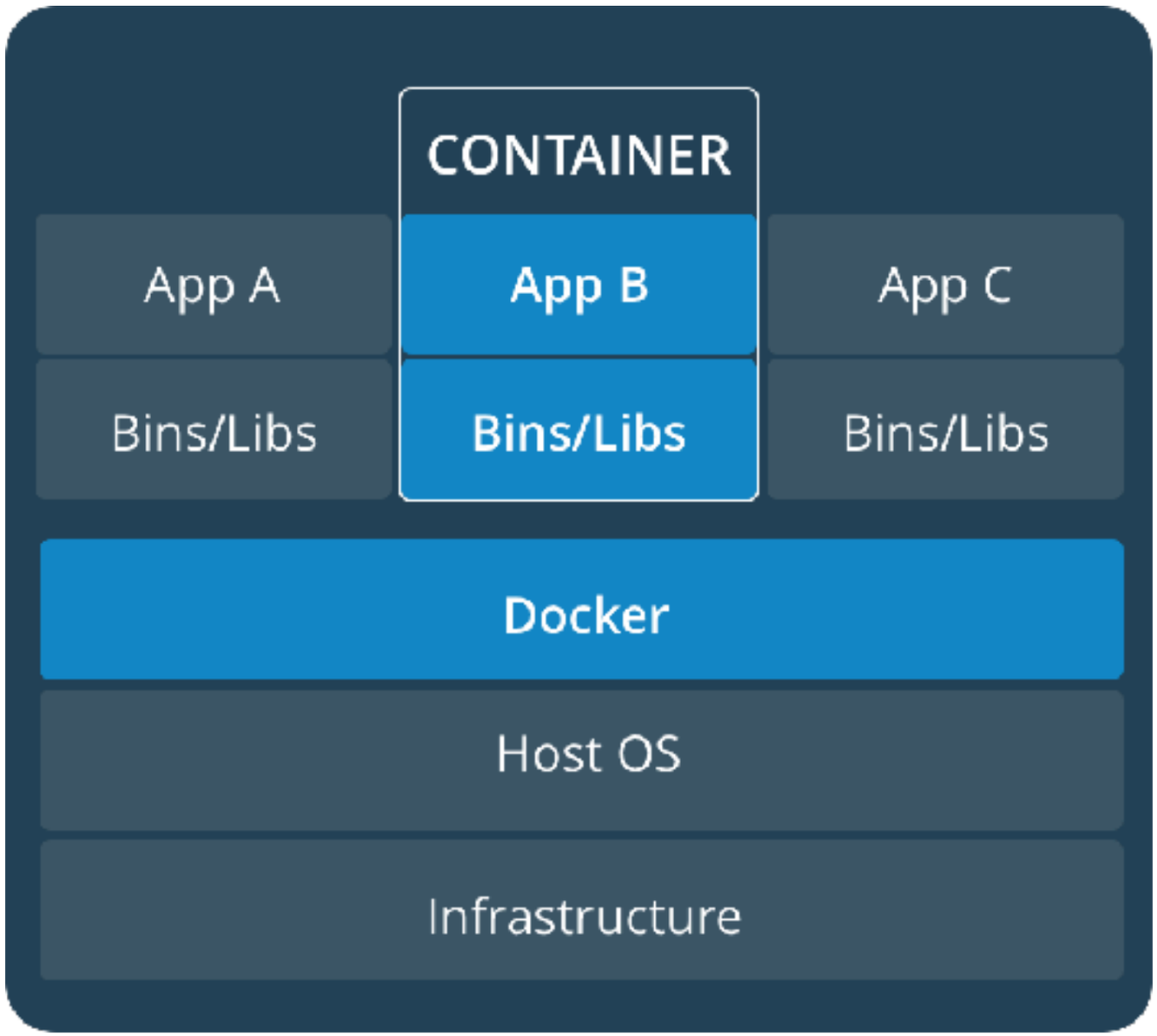
# Container vs. Virtual machine

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

# Virtual Machine



# Container


CONTAINER
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker
Host OS
Infrastructure

# Virtual Machine

Abstraction of physical hardware

Full copy of OS and libraries

Slow to boot

# Container

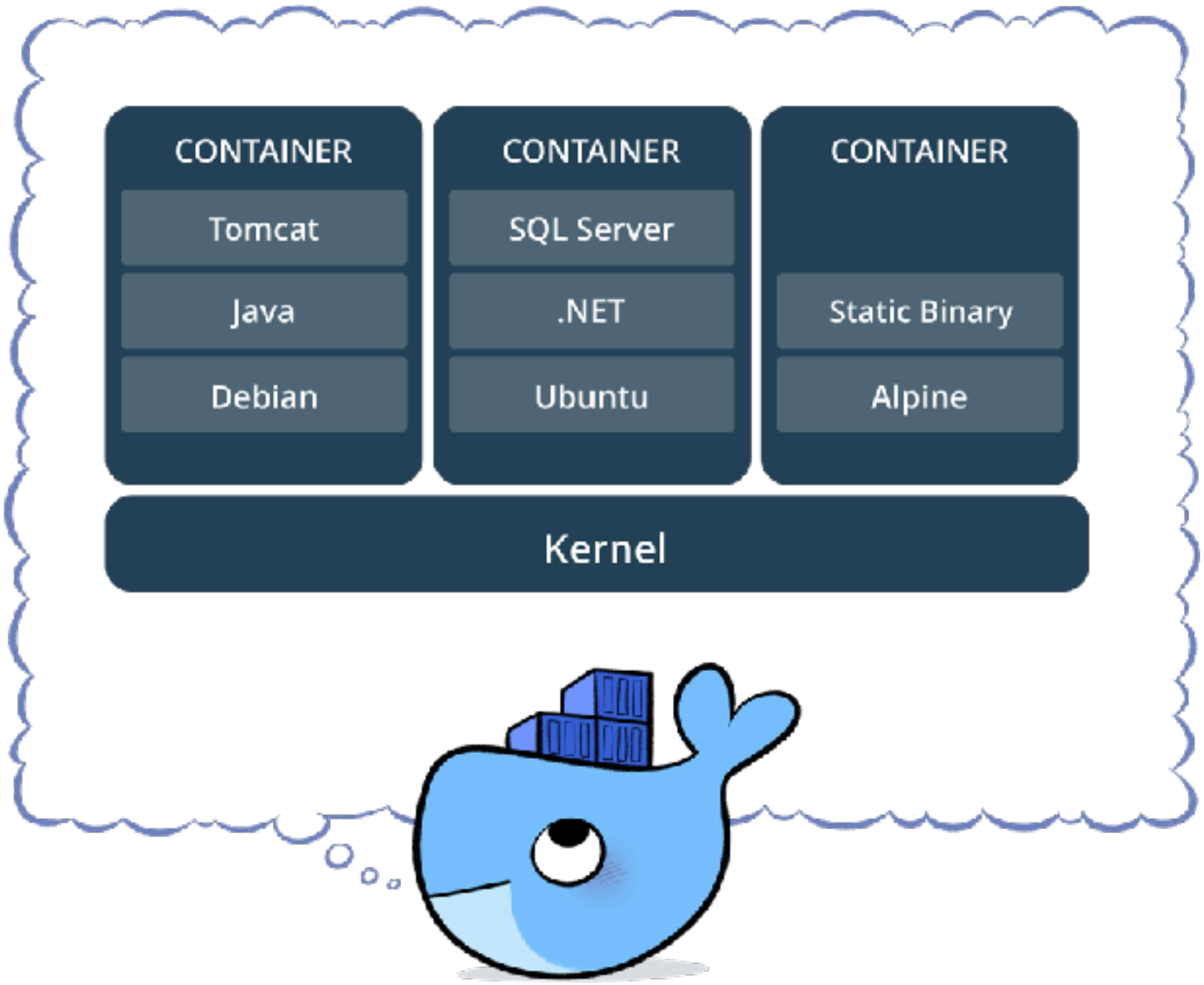
Abstraction at the app level

Share OS kernel

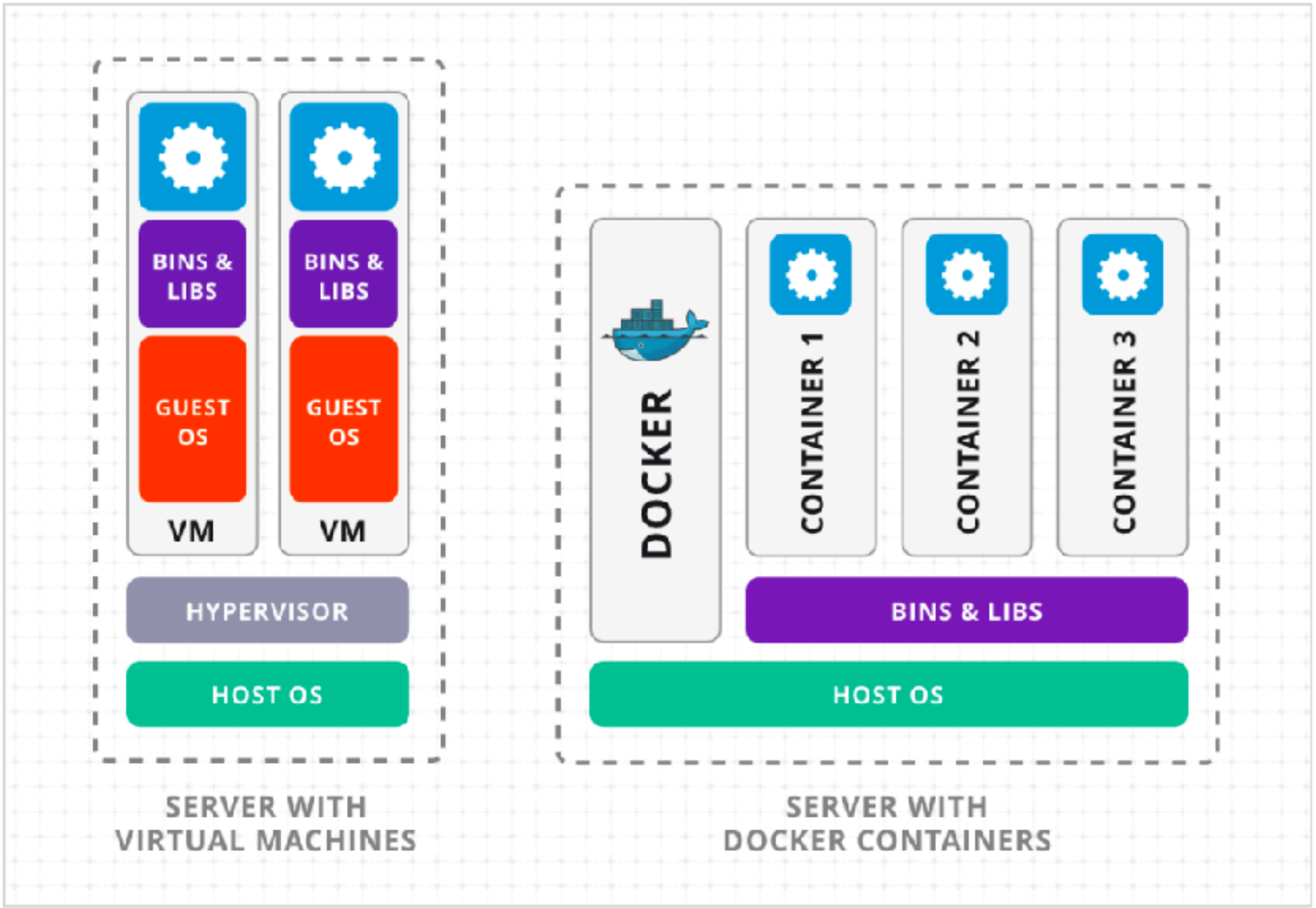
Isolate process

Less space than Virtual Machine

# Docker


CONTAINER
Tomcat
Java
Debian
CONTAINER
SQL Server
.NET
Ubuntu
CONTAINER
Static Binary
Alpine
Kernel

# Container vs Virtual machine

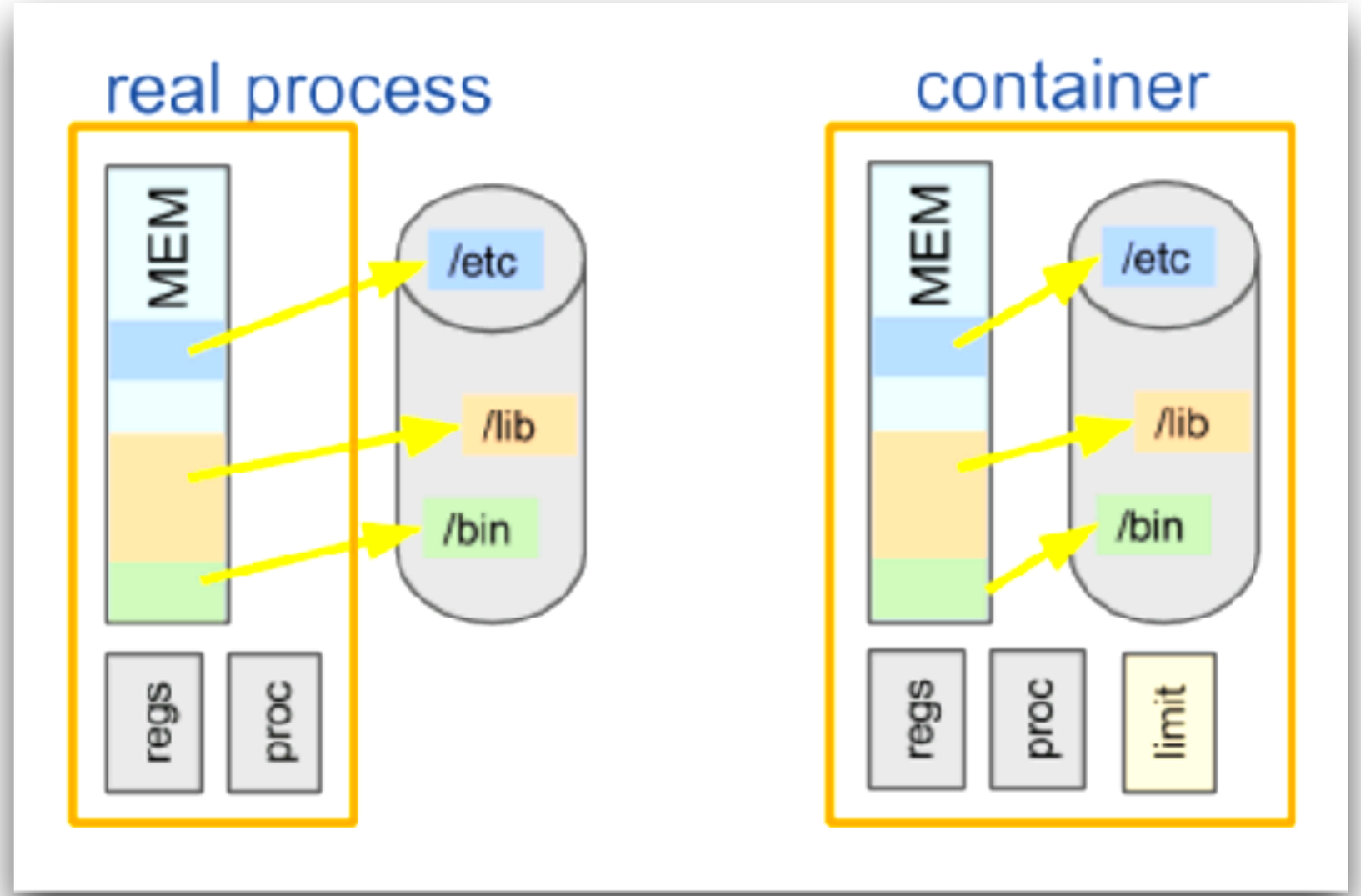


# Container vs Virtual machine

| | 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:
 Version:      17.03.0-ce
 API version:  1.26
 Go version:   go1.7.5
 Git commit:   60ccb22
 Built:        Thu Feb 23 10:40:59 2017
 OS/Arch:      darwin/amd64

Server:
 Version:      17.03.0-ce
 API version:  1.26 (minimum version 1.12)
 Go version:   go1.7.5
 Git commit:   3a232c8
 Built:        Tue Feb 28 07:52:04 2017
 OS/Arch:      linux/amd64
 Experimental: true
```

# Hello docker

## $docker 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/
```

# Echo with docker

$docker run busybox echo "Hello World"

# Container with busybox

Smallest and simplest image

Used in embedded systems

Run single process and echo Hello World

# Hello again

$docker run –it ubuntu bash

# Welcome to container

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

# Check installed packaged

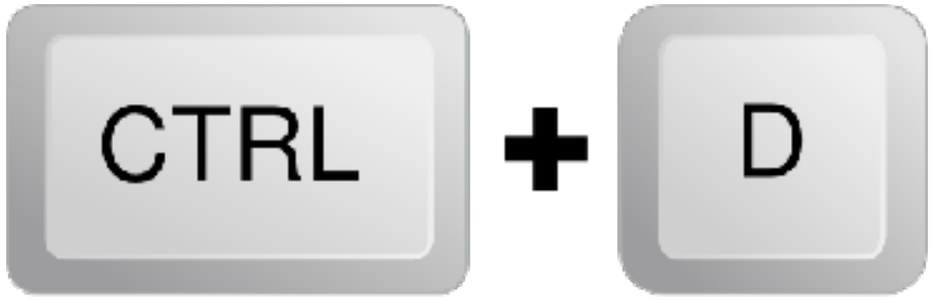
```
root@c20e8c218664:/# dpkg -l | wc -l
103
```

# 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 run -it ubuntu bash

$curl google.com

*bash: curl: command not found*

# This is container

# Basic of Docker

# Basic of Docker

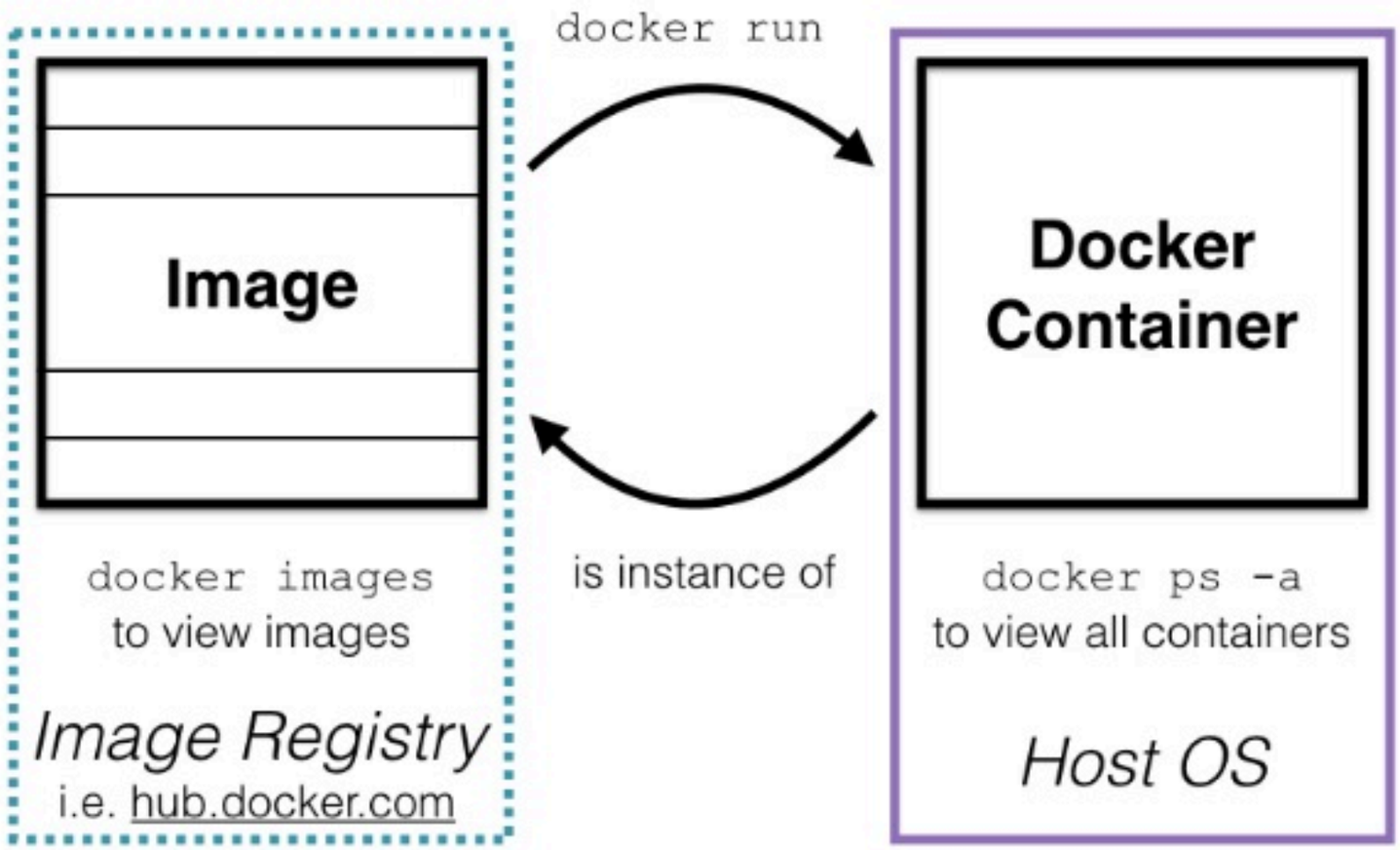
**Image**

**Container**

Dockerfile

Registry

# Image vs Container



# Image vs Container

**Image** like template/blueprint

**Containers** are created from image

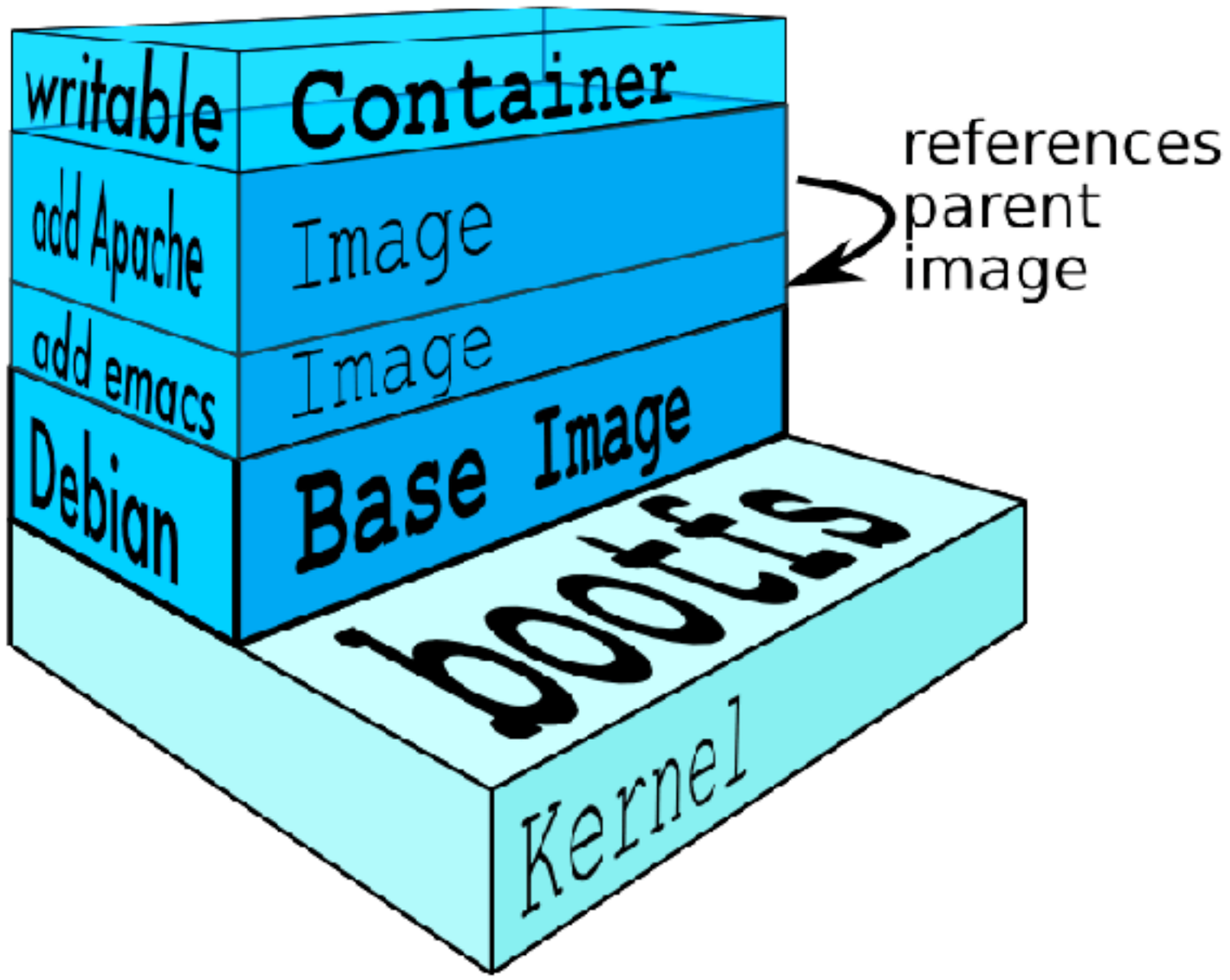
# Image vs Container

**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

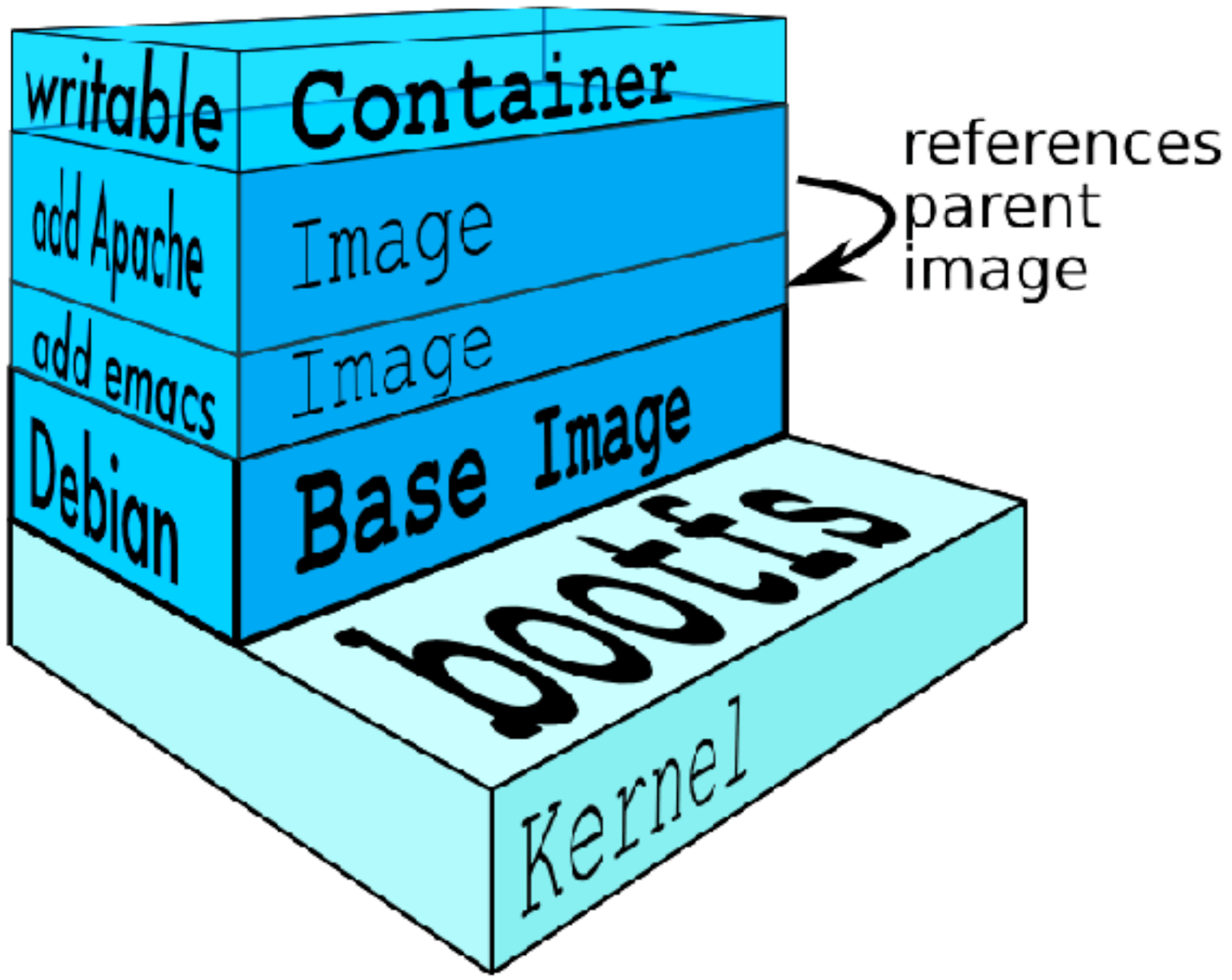
Image can share layers

**Read-only** file system

# How do we change image ?

We do not !!

# How do we change image ?

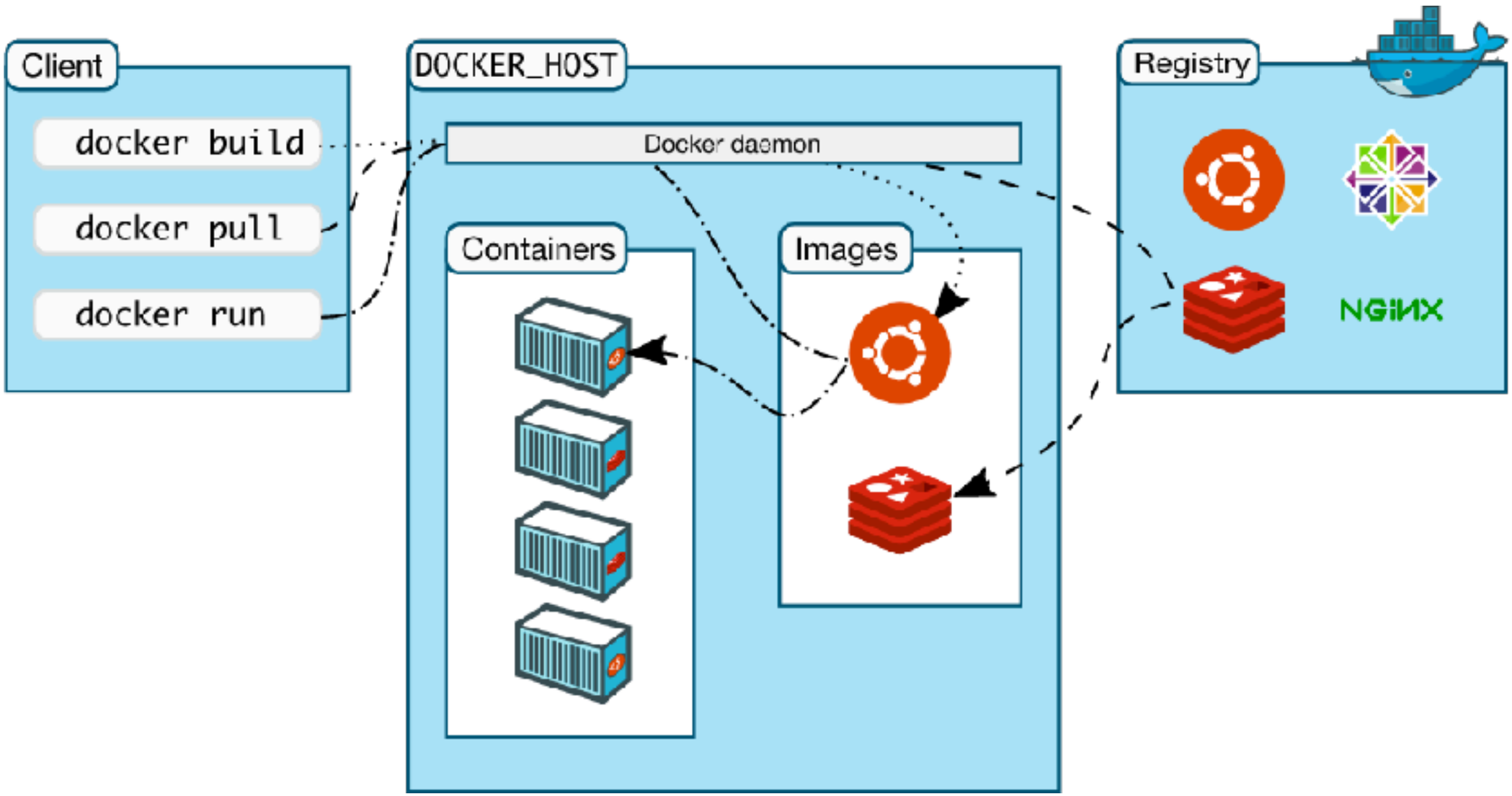


# How do we change image ?

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 ?

# How docker works ?




https://docs.docker.com/engine/understanding-docker/#what-is-dockers-architecture

# 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 |

# 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 |
|---|---|---|---|---|
| divyakumarjain/nginx-proxy | ngix-proxy | 1 | | [OK] |
| juanheredia/ngixkoken | | 0 | | |
| dai1219/ngix | | 0 | | |
| vinithmenon28/admatic-ngix | | 0 | | |
| bowlingx/docker-ngix-php-v8 | | 0 | | |
| almanacproject/ngix-proxy-smqueryapi | | 0 | | |
| fangufl/docker-static-ngix | | 0 | | |
| eferlo/ngix | | 0 | | |
| slashn/ngixiptest2 | | 0 | | |
| gymnae/webserverbase | Web server image with glidenlabs/alpine:ed... | 0 | | [OK] |
| nomiad/ngixsample | test | 0 | | |
| codexpage/ngixx | | 0 | | |
| jerideng/ngixn | | 0 | | |
| yavinenana/ngix | | 0 | | |

https://hub.docker.com/

# Download images (old)

$docker pull <image name>

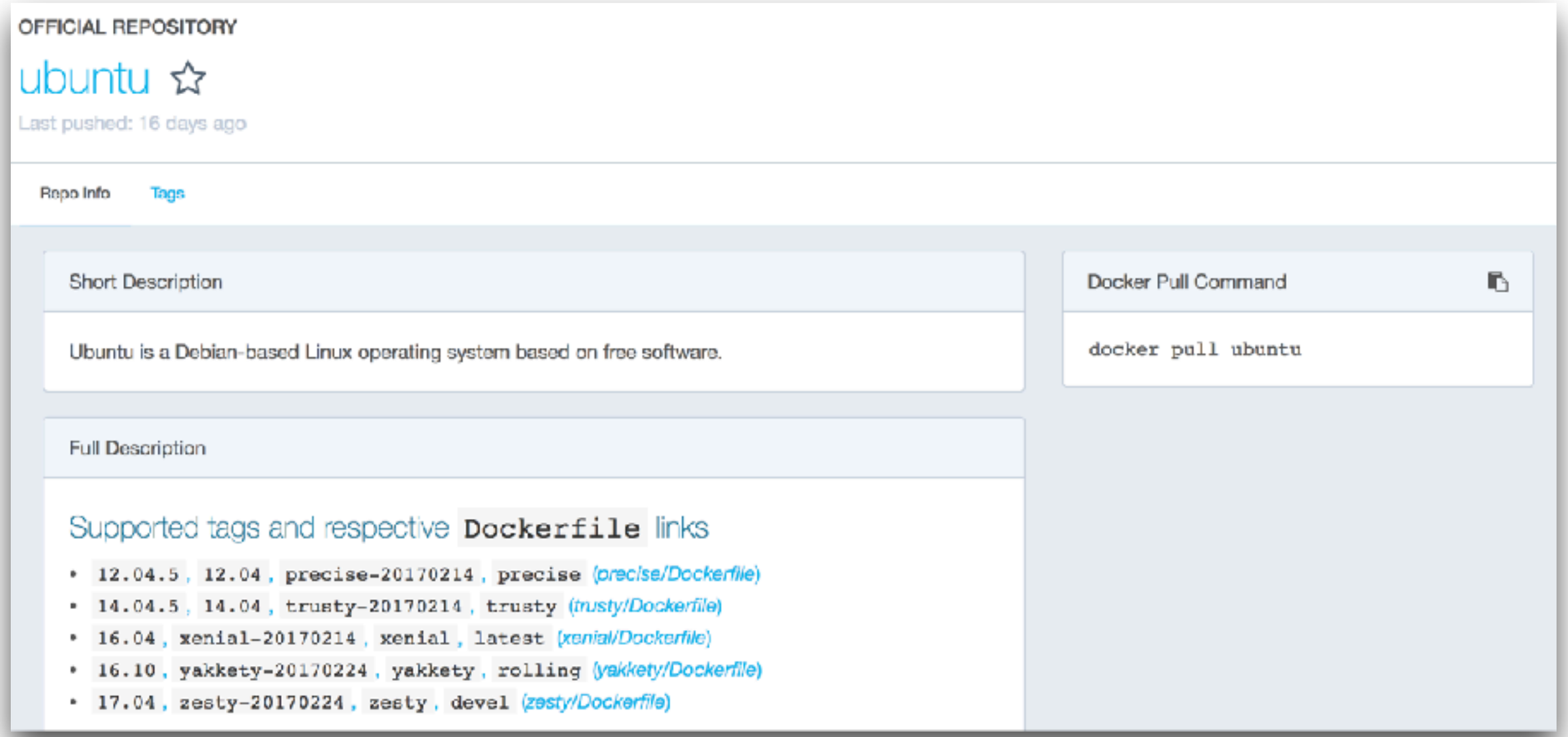
$docker run <image name>

# Download images (new)

$docker image pull <image name>

$docker container run <image name>

# Docker with Ubuntu




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

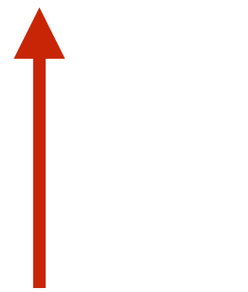
# Pull image

## $docker pull ubuntu:latest

```
latest: Pulling from library/nginx
Digest: sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
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 pull ubuntu:latest



version 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 pull ubuntu:xenial

specified tag

# Remove image

$docker image rm <id>

# Remove all image

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

# Remove all image

$docker image rm ?

# 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 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>

# 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 stop $(docker ps -a -q)
$docker 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

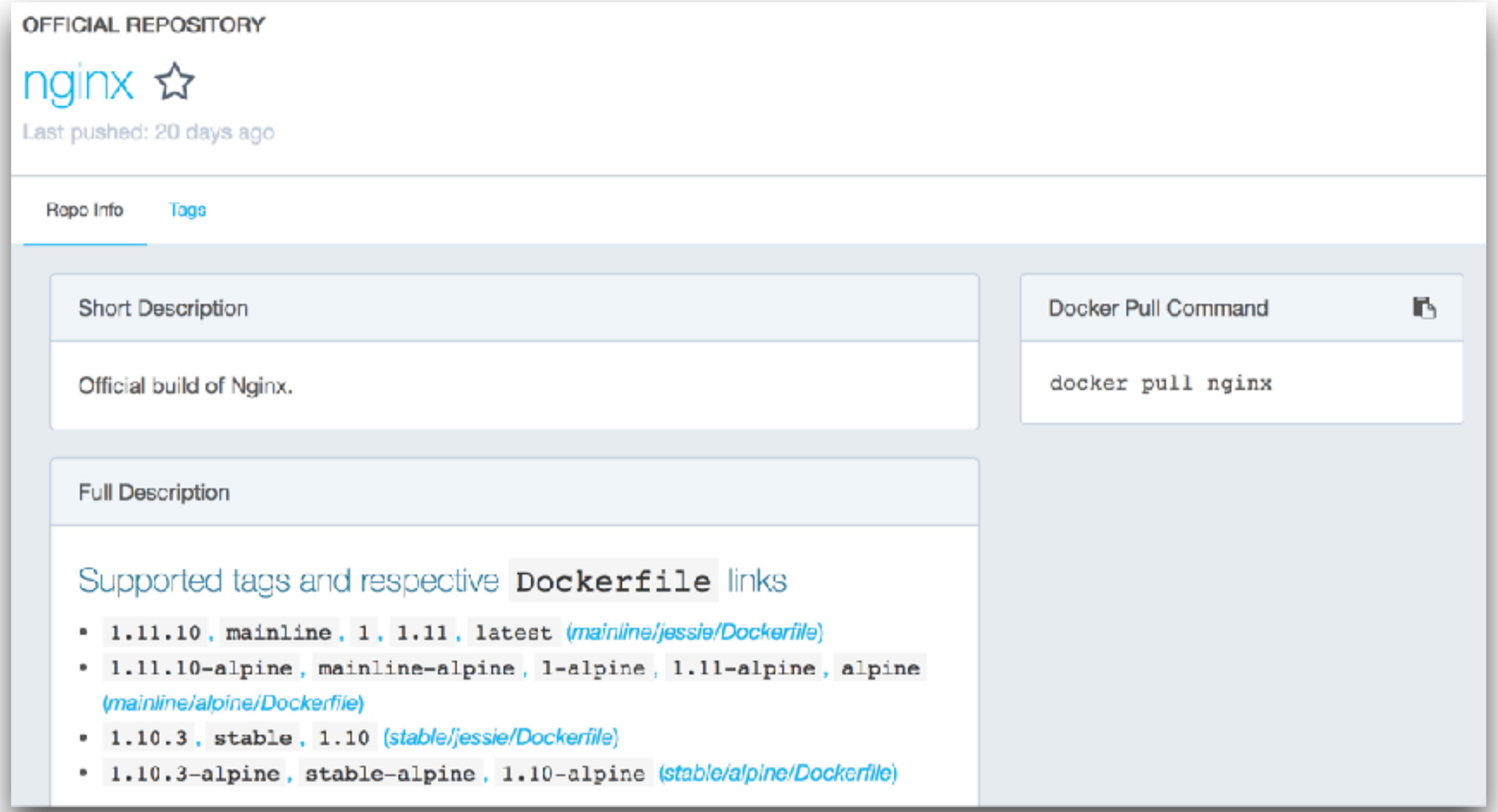
# Let’s start

$docker run

$docker container run

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

# Docker with nginx




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: sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
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 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

# 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 id>

$docker container run it <new id> 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

# 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/mai
Get:5 http://archive.ubuntu.com/ubuntu xenial/res
```

# 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 ENTRYPOINT

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

# Basic of Container networking

# Networking

Run network service in container

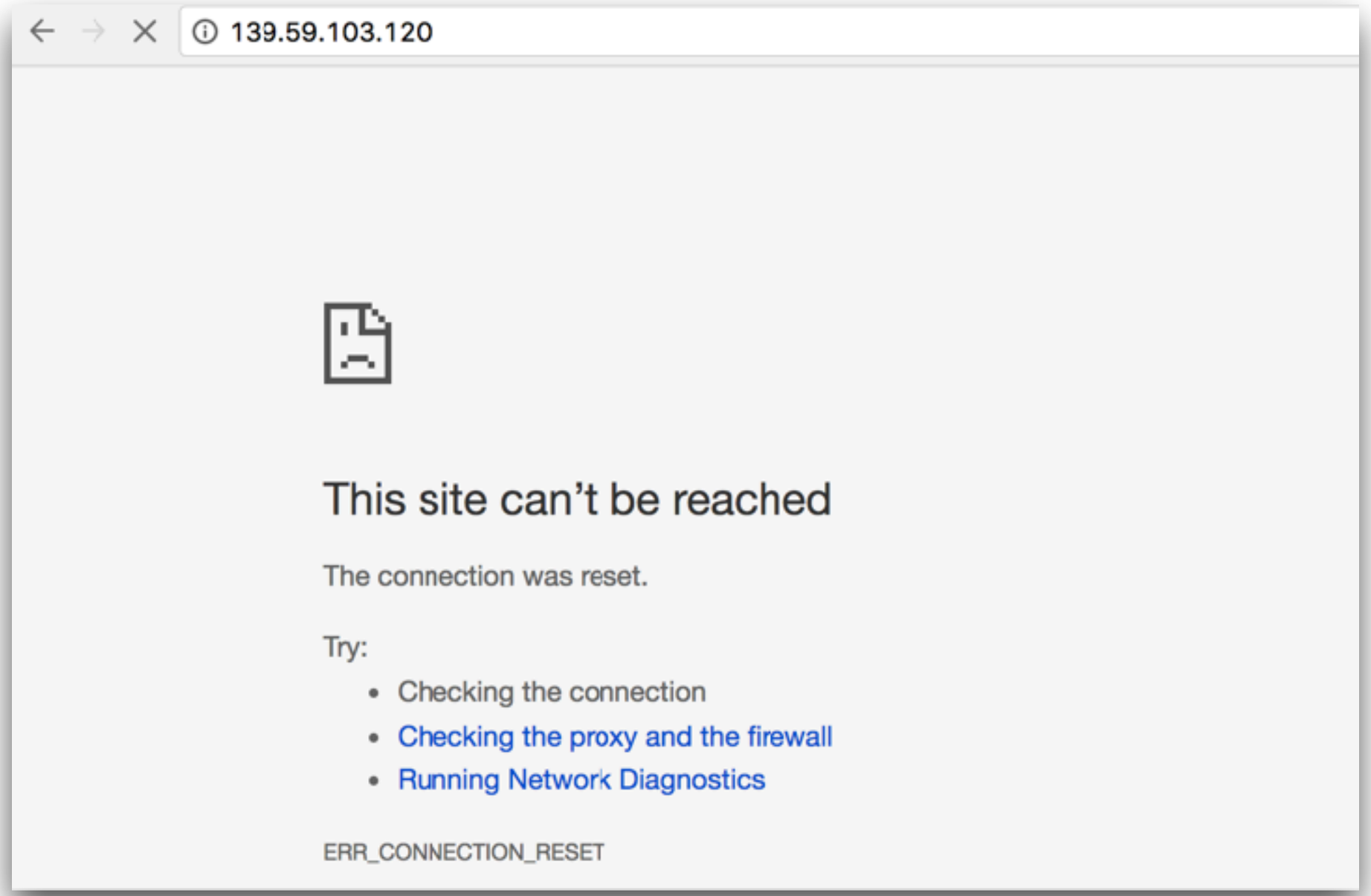
Manipulate container networking

Find IP’s container

# Start nginx with name

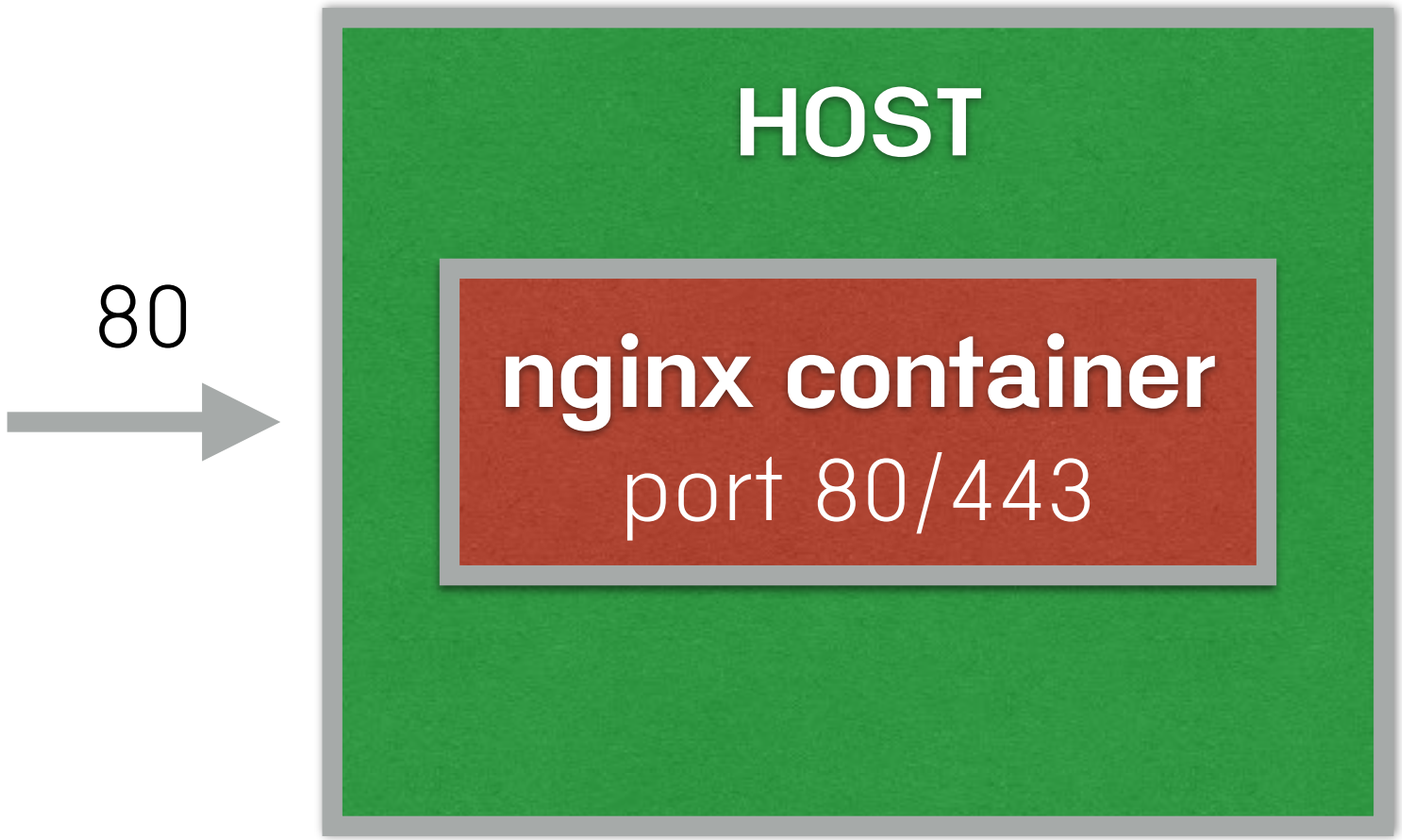
```
$docker container run \
    --name hello-nginx  \
    -d nginx
```

# Try to access with port 80

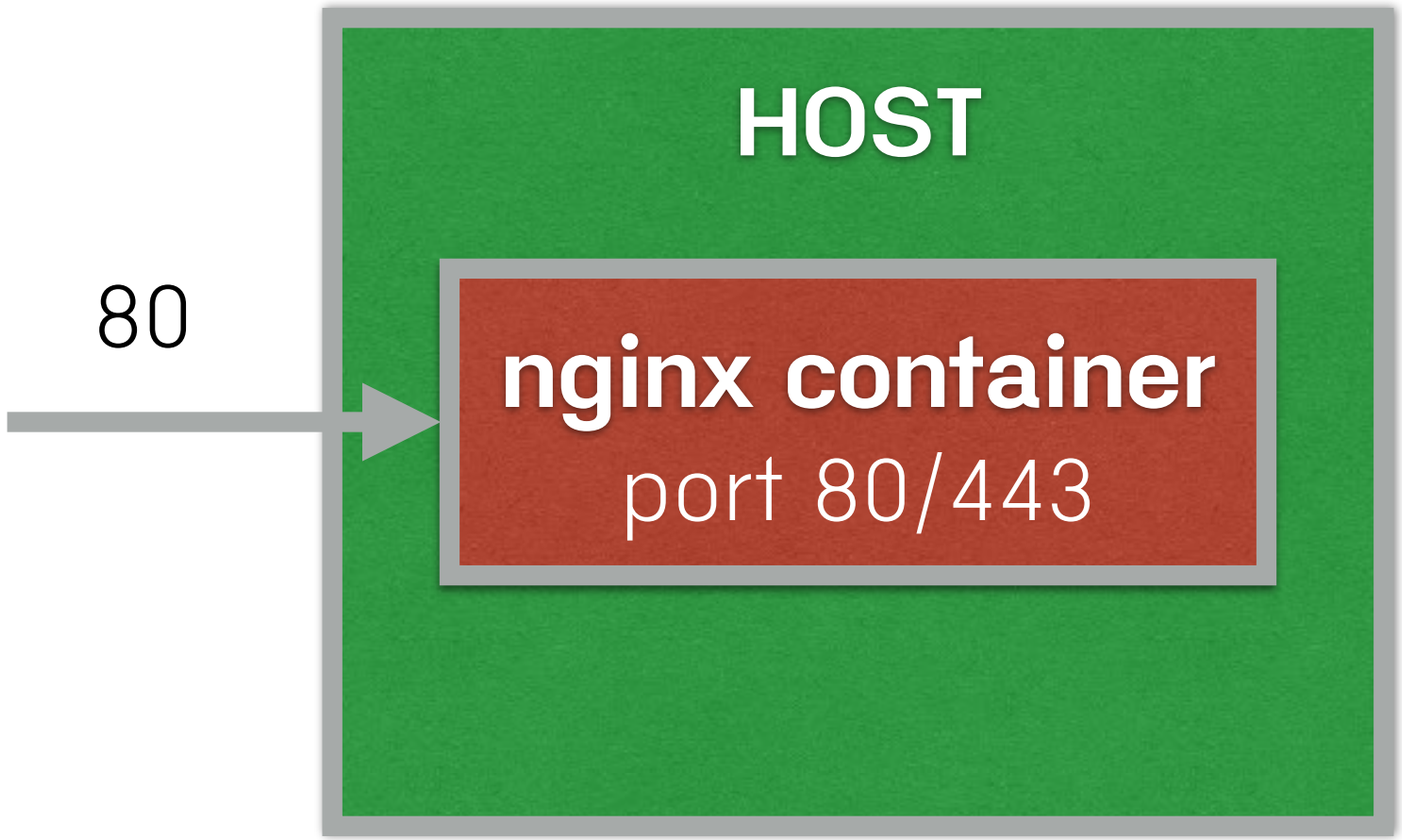


# Why not ?

# 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

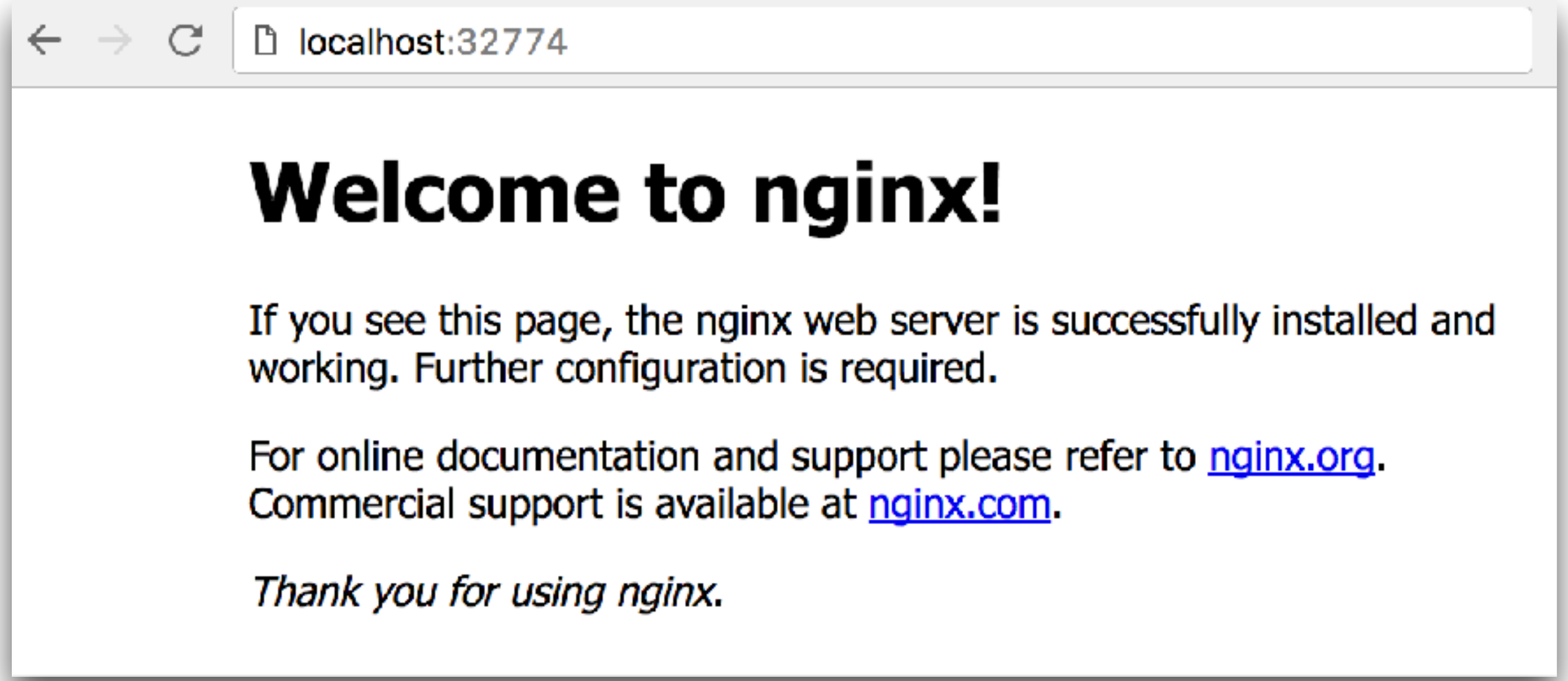
# Custom result format

$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



# Publish by port

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

-p 80:80 nginx

**Host** **Container**

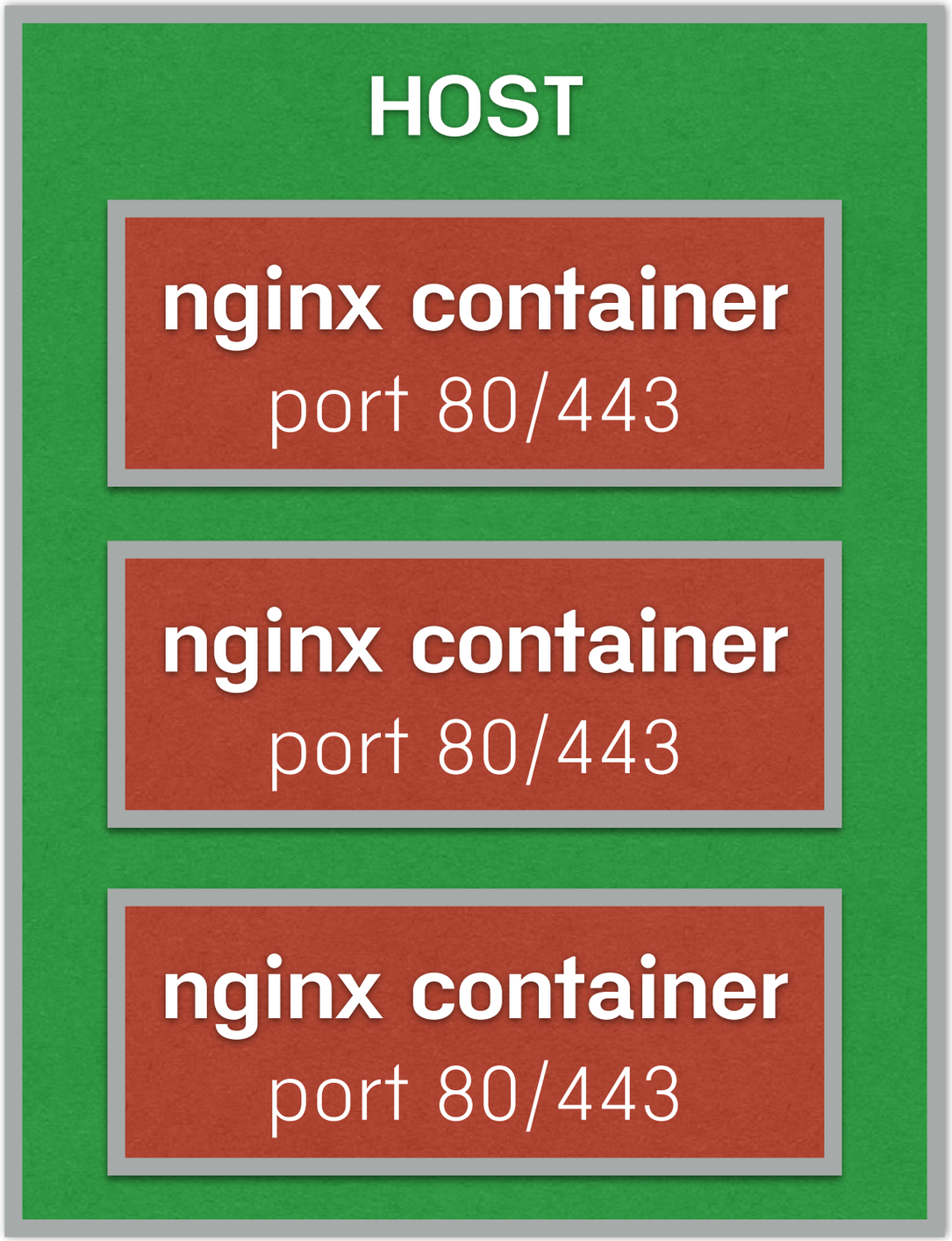
# See result

$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": "cd7aeecb8020cb334f0ccd0601230f9927be15c88511d9ec0340af281e769eb5",
    "HairpinMode": false,
    "LinkLocalIPv6Address": "",
    "LinkLocalIPv6PrefixLen": 0,
    "Ports": {
        "443/tcp": null,
        "80/tcp": [
            {
                "HostIp": "0.0.0.0",
                "HostPort": "80"
            }
        ]
    },
    "SandboxKey": "/var/run/docker/netns/cd7aeecb8020",
    "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,
```

# Find IP’s container

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

# Ping container with IP

$ping <IP address>

# 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

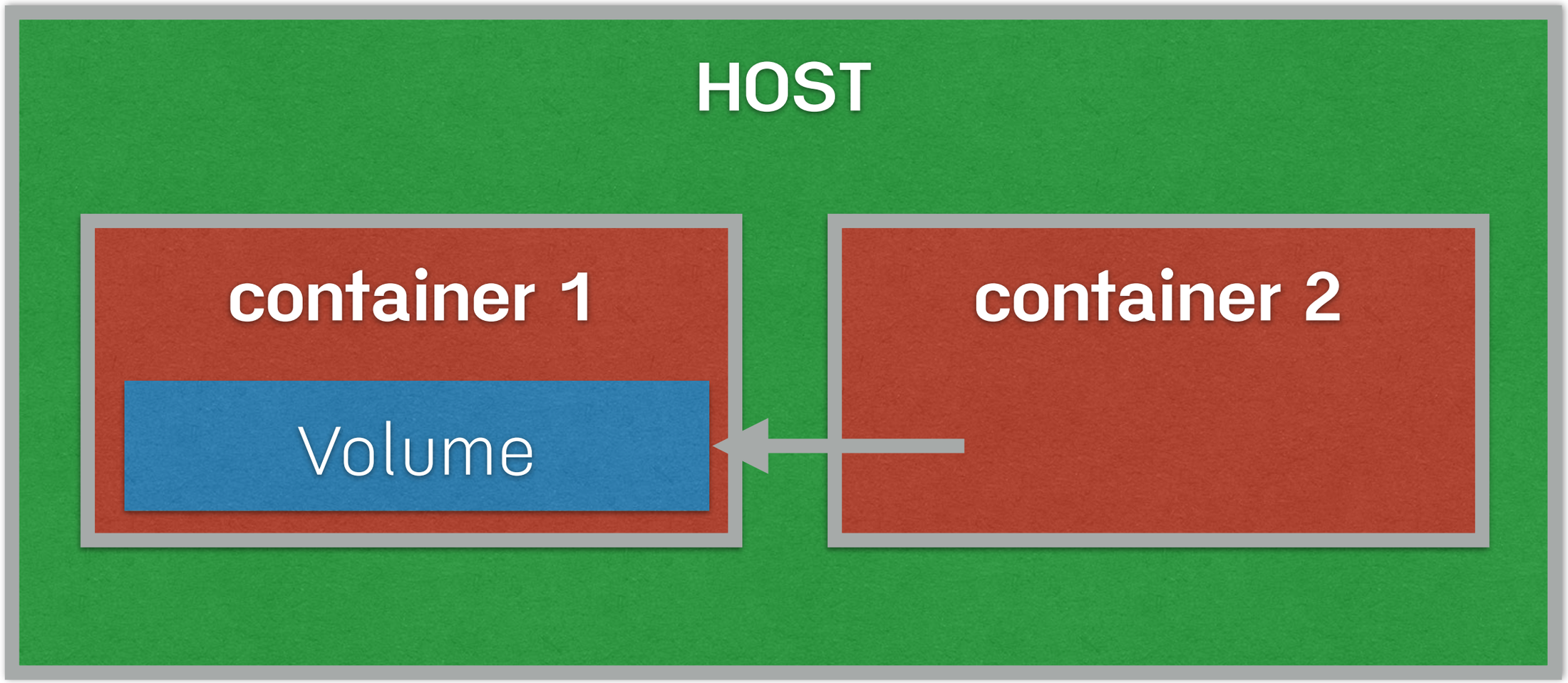
# Dockerfile

**VOLUME** /var/lib/postgresql

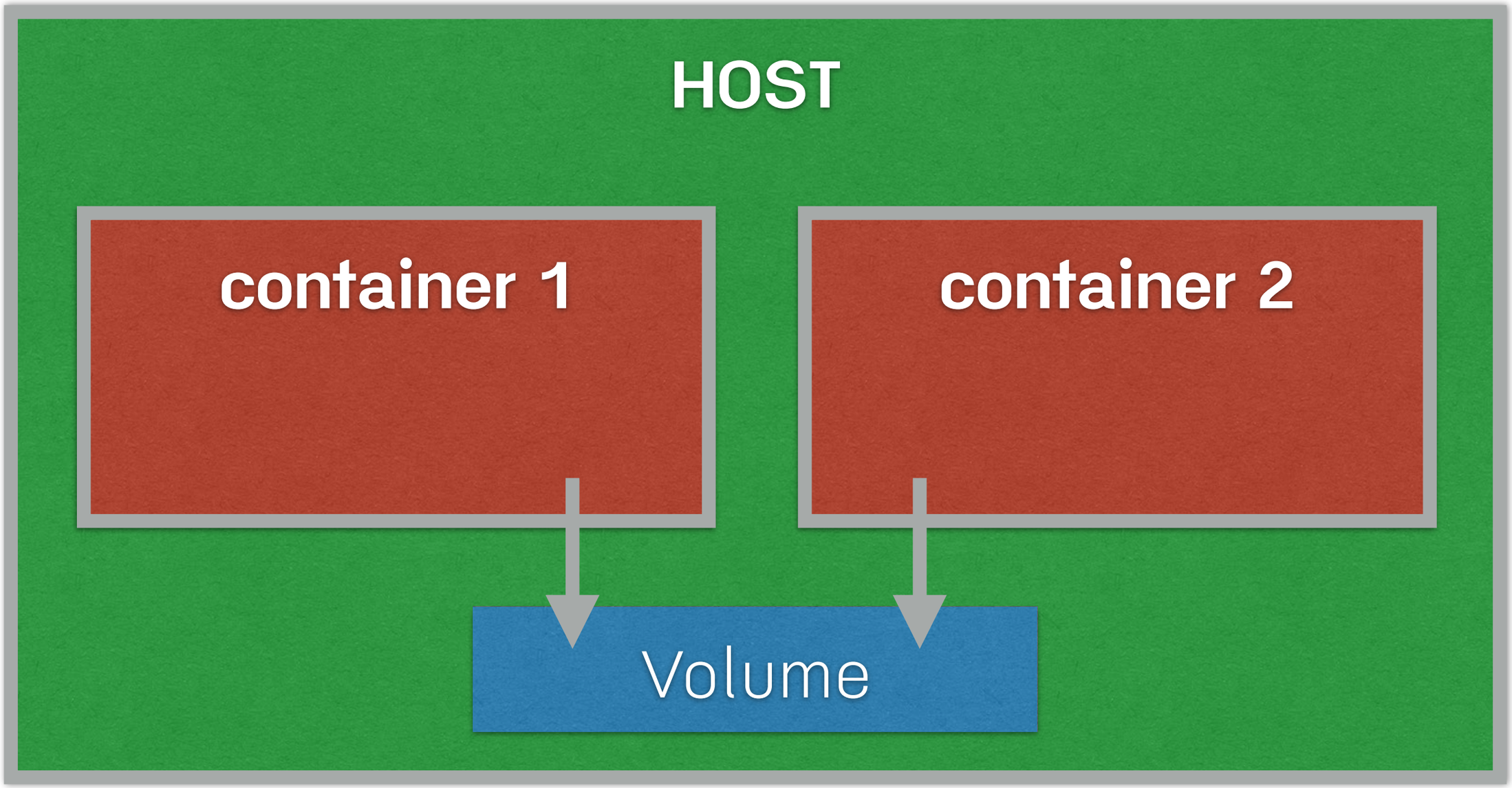
# Command-line

$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

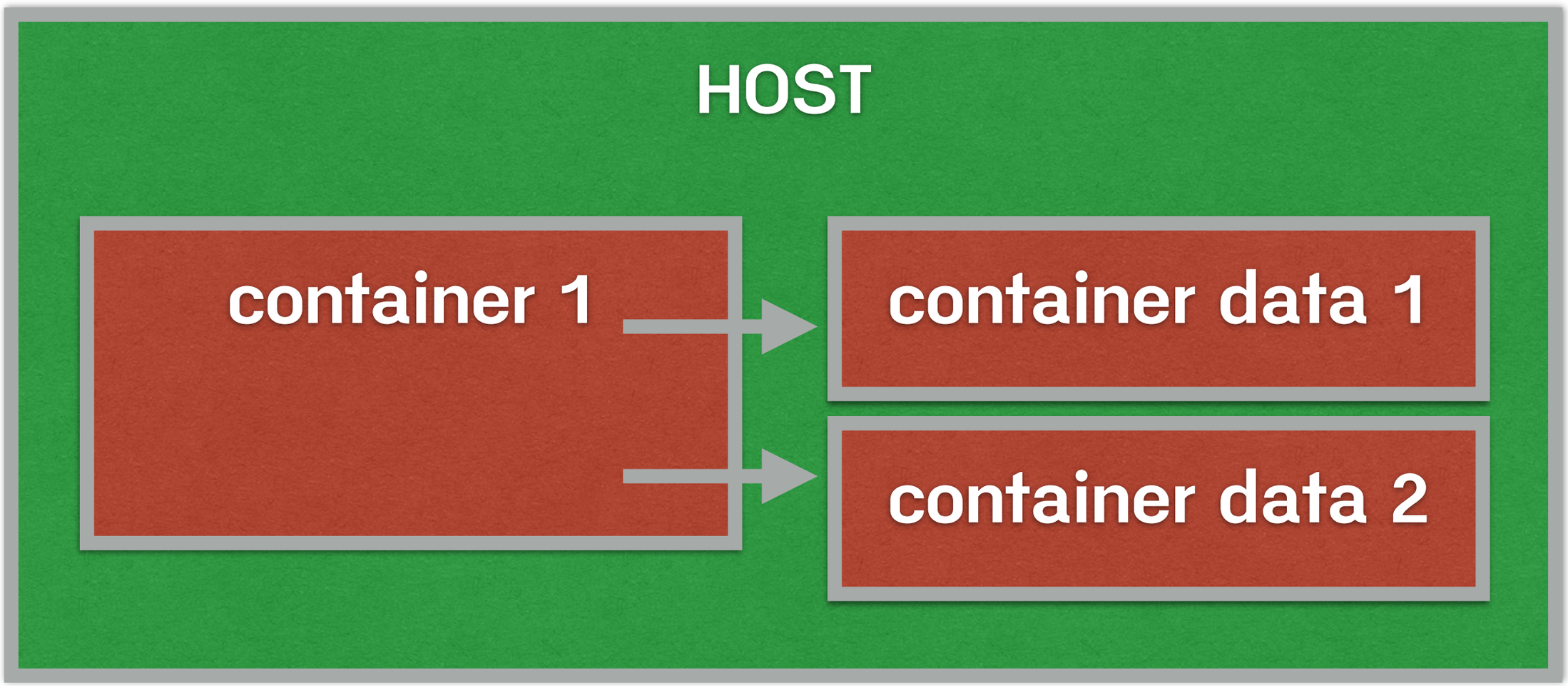
If Container is stopped

Volume still exist and available

Try to stop container 1 !!

# 3. Data container

Container for the purpose of **referencing volumes**



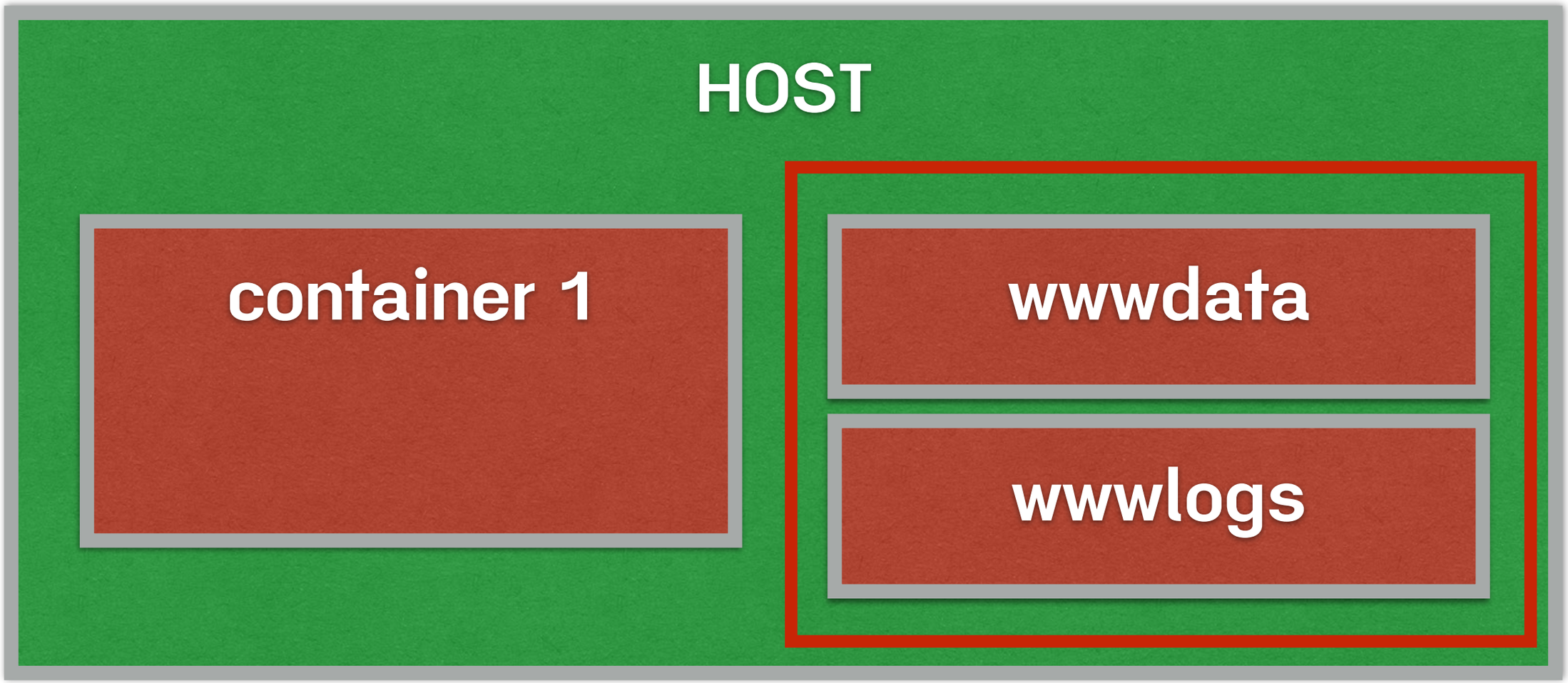
# Create data containers with busybox

```
$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

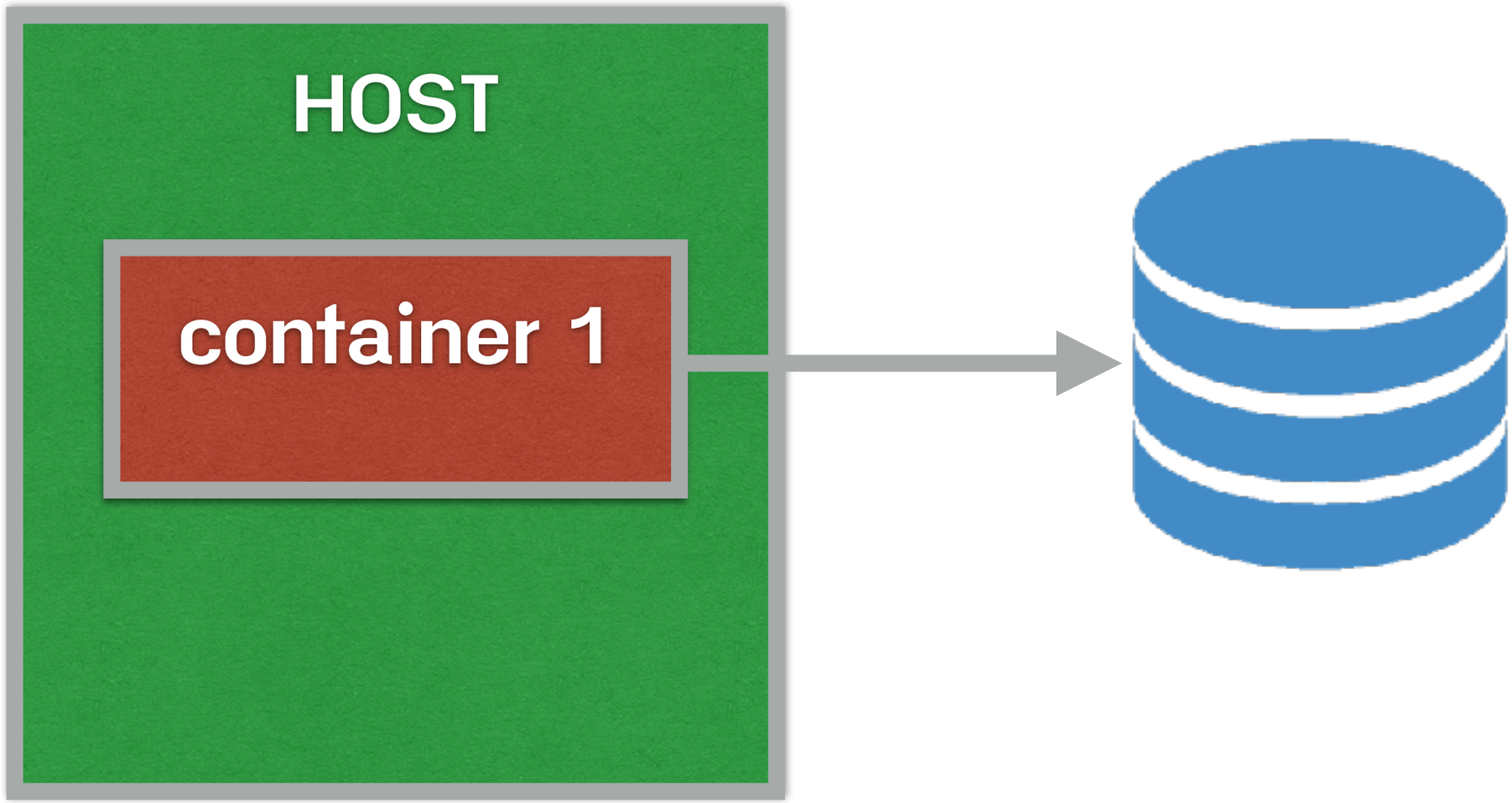
$docker run -d **--volumes-from** wwwdata ftpserver

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

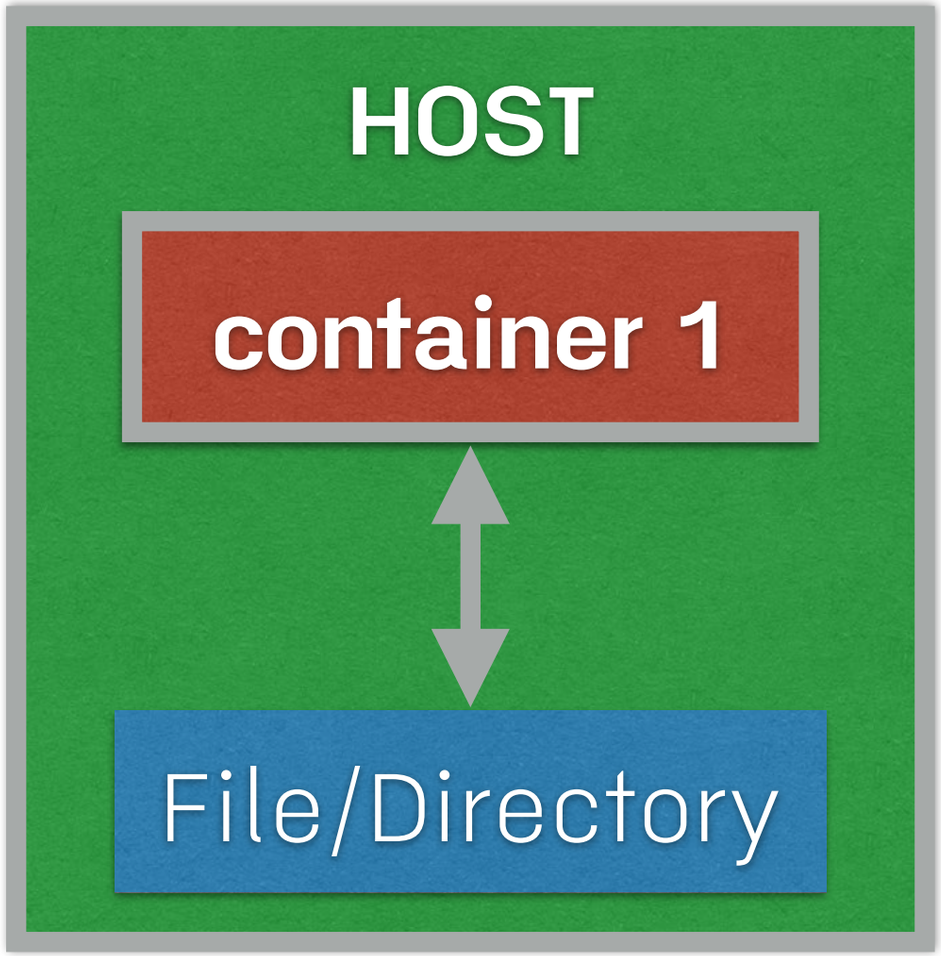
How to test ?

# 4. External storage

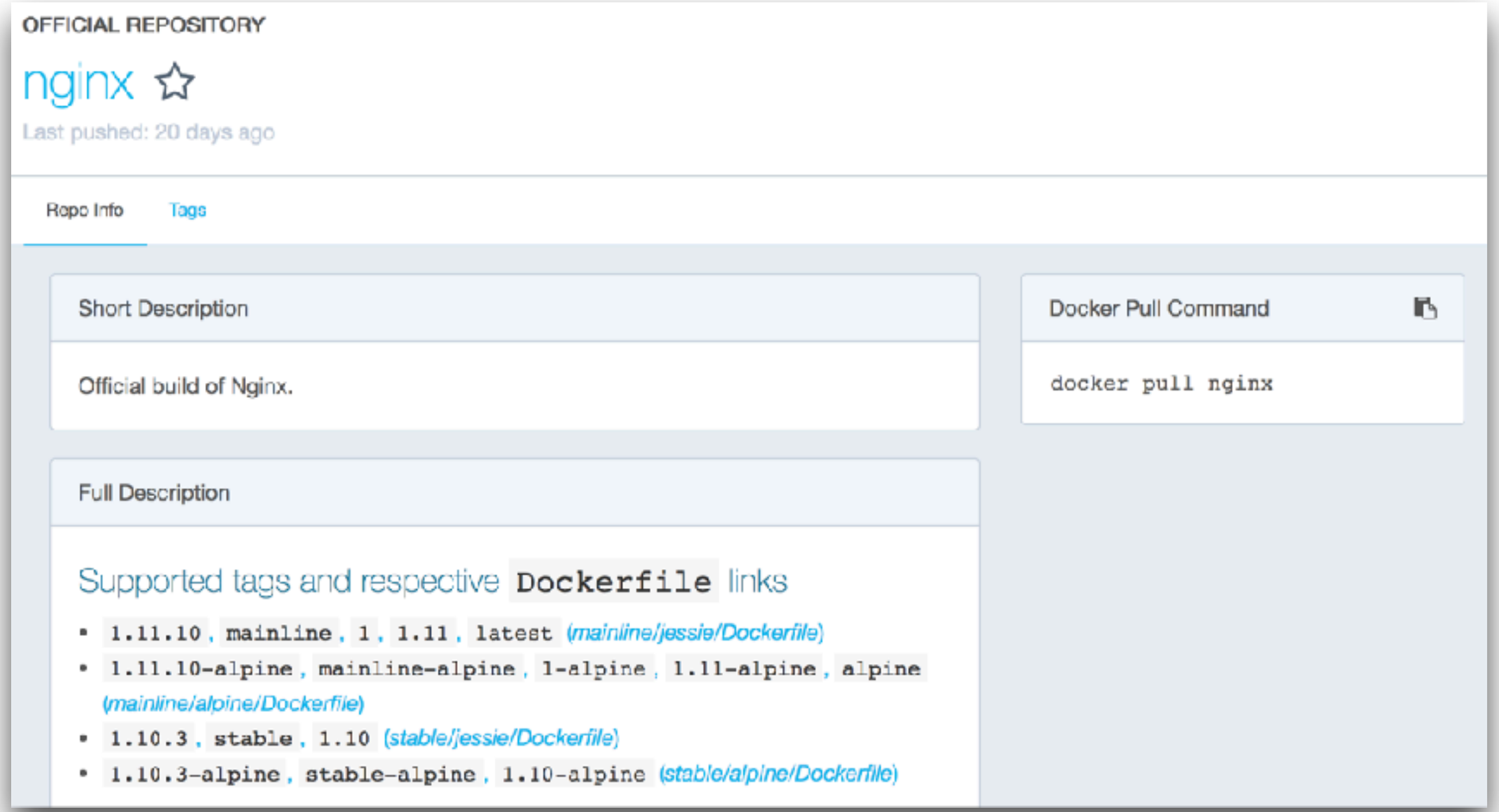
LVM, SAN, NFS, ZFS, Ceph, GlusterFS ... etc.



# 5. Share file/dir between host and container

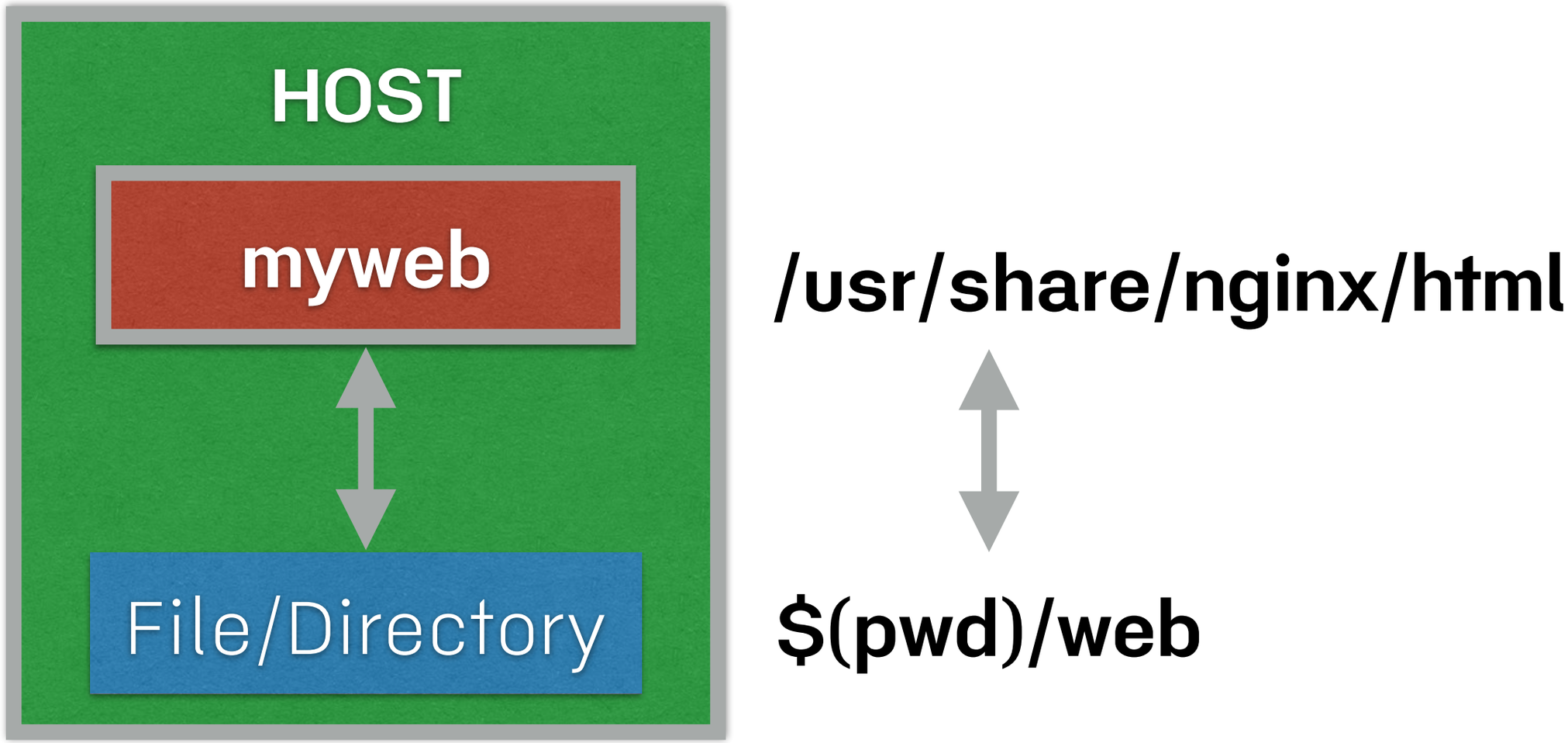


# Working with Nginx




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

# Working with Nginx



# Working with Nginx

```
$mkdir web

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

Host

Container

# See publish port and test

$docker container port my web

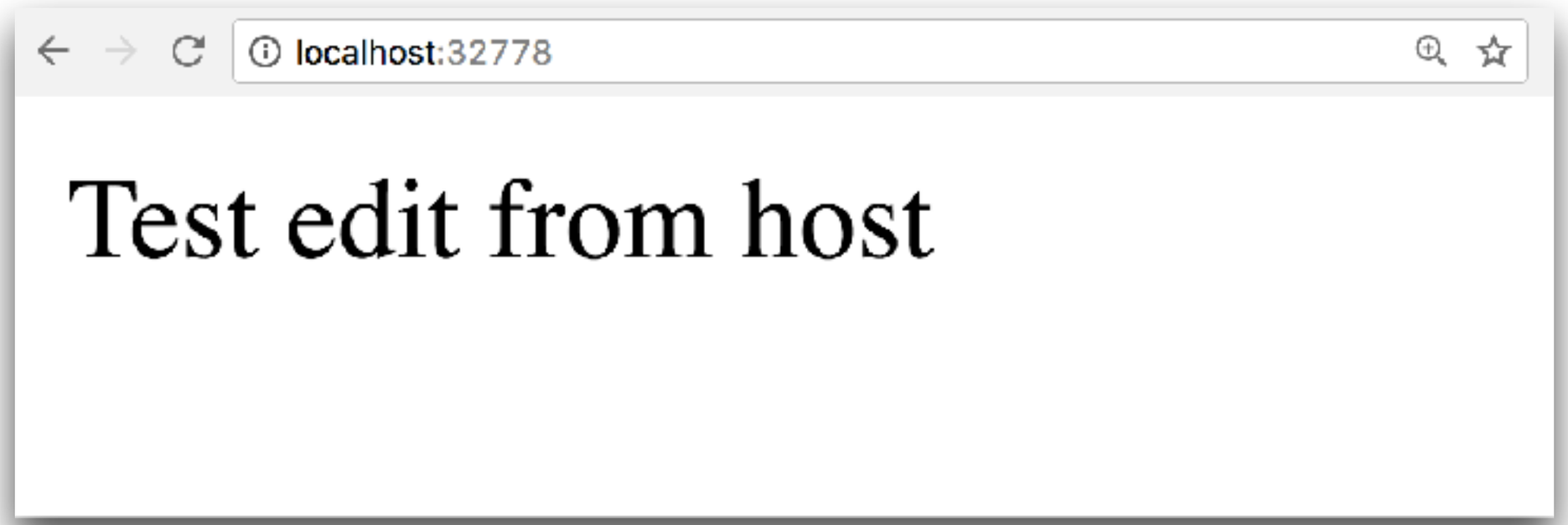
# Create new file in host

```
$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>

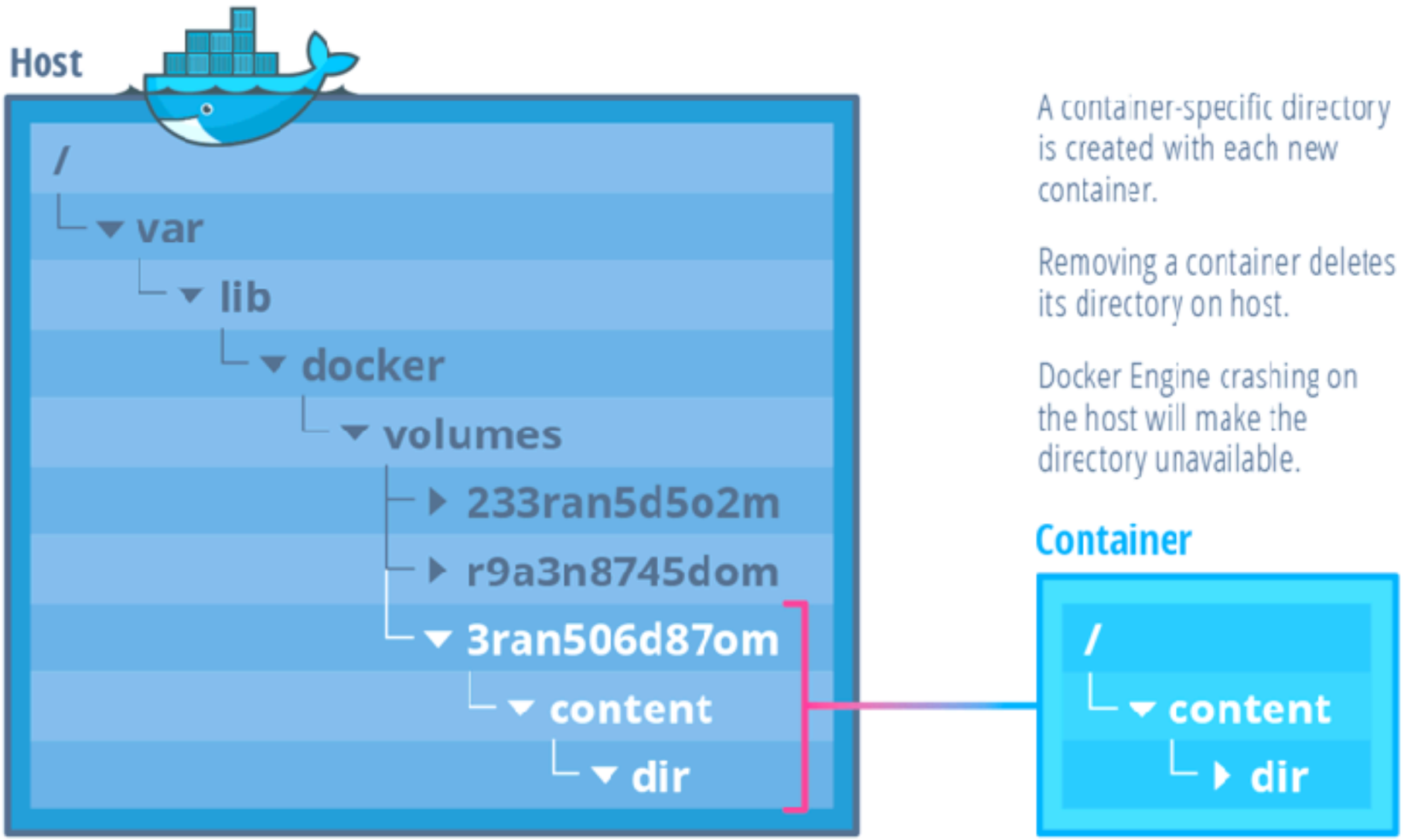
$docker inspect --format='{{json .Mounts}}' <id>

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

# Strategies to Manage Persistent Data

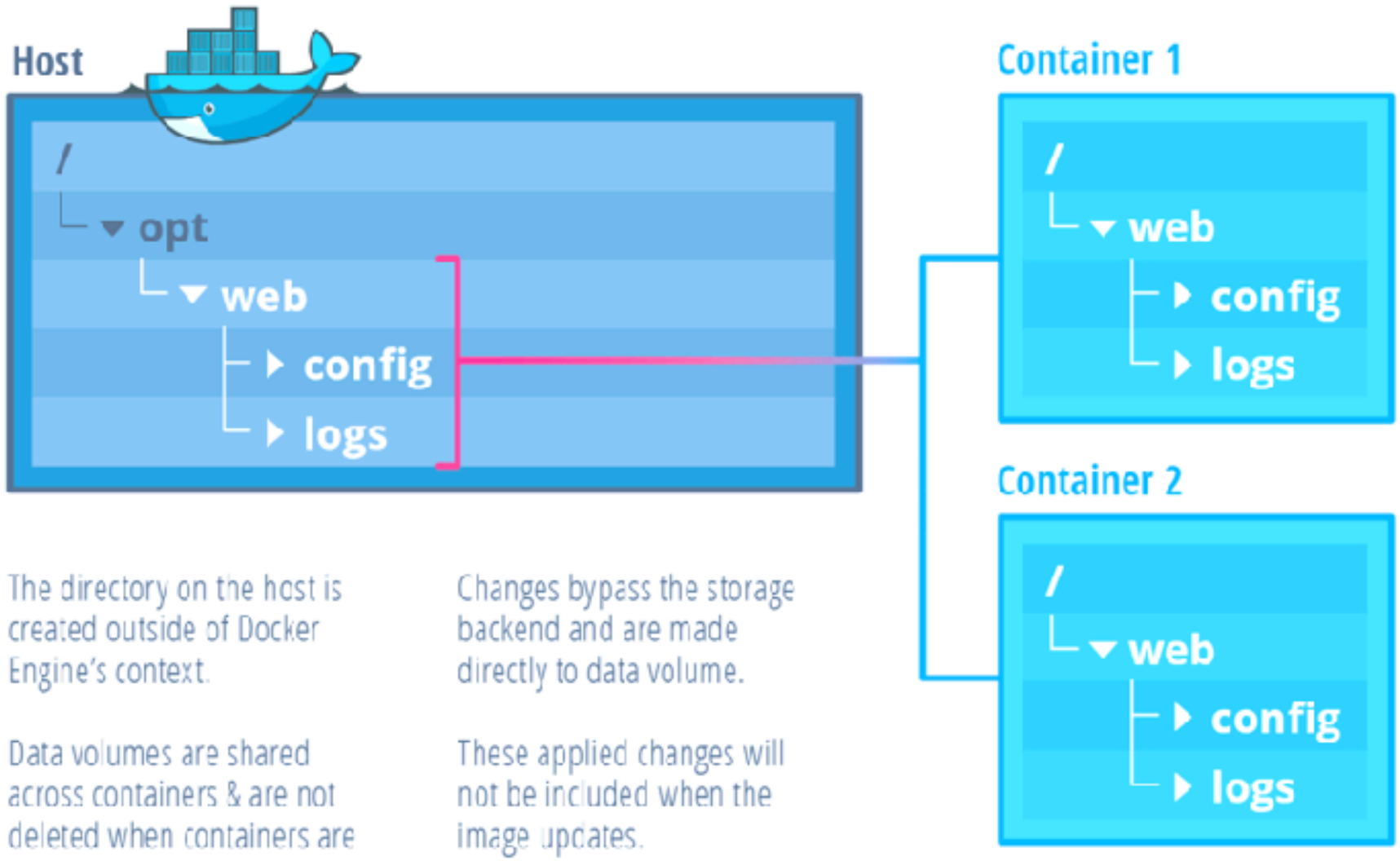
# Host-based persistence

## per container



# Host-based persistence

## shared containers




The directory on the host is created outside of Docker Engine's context.

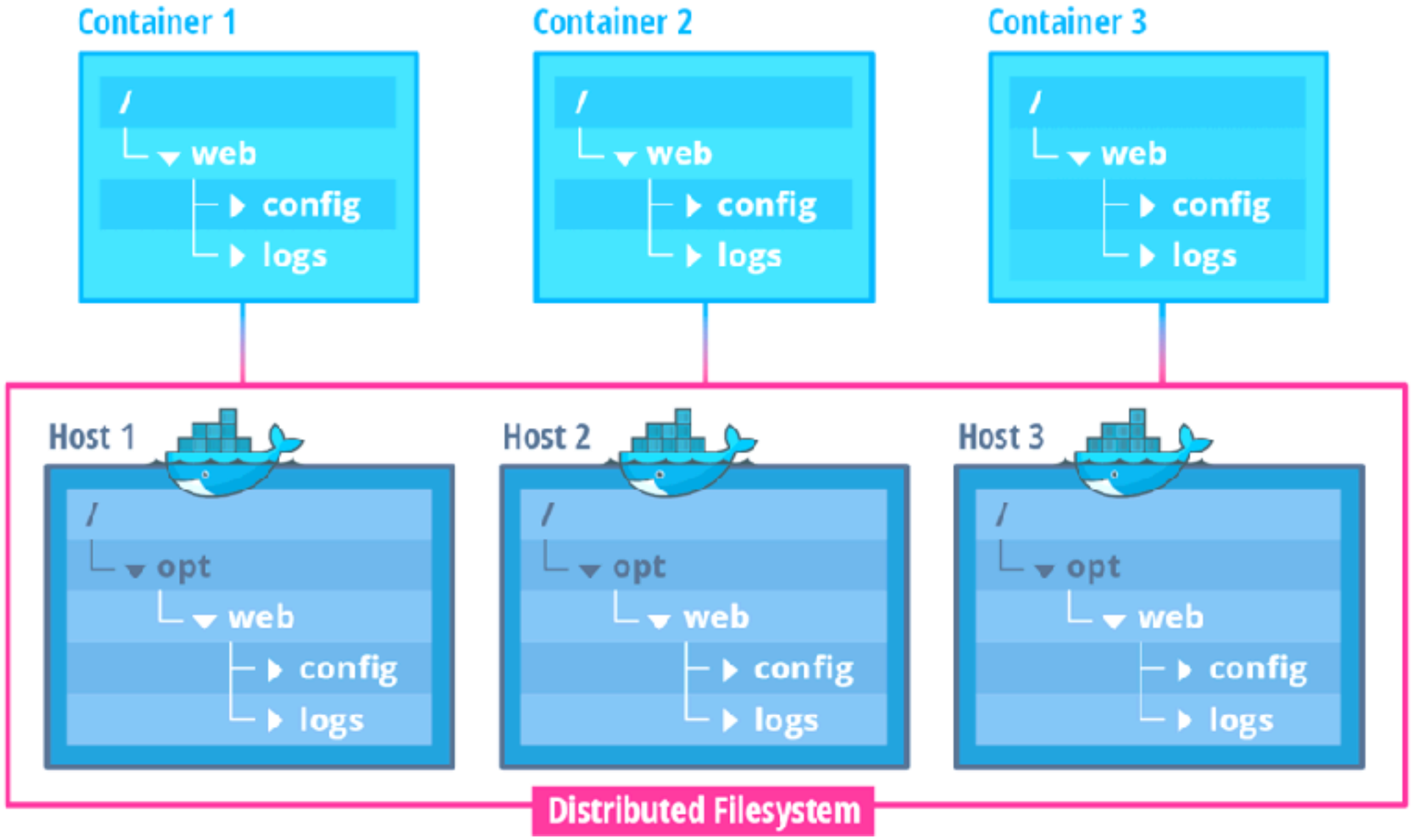
Changes bypass the storage backend and are made directly to data volume.

Data volumes are shared across containers & are not deleted when containers are removed.

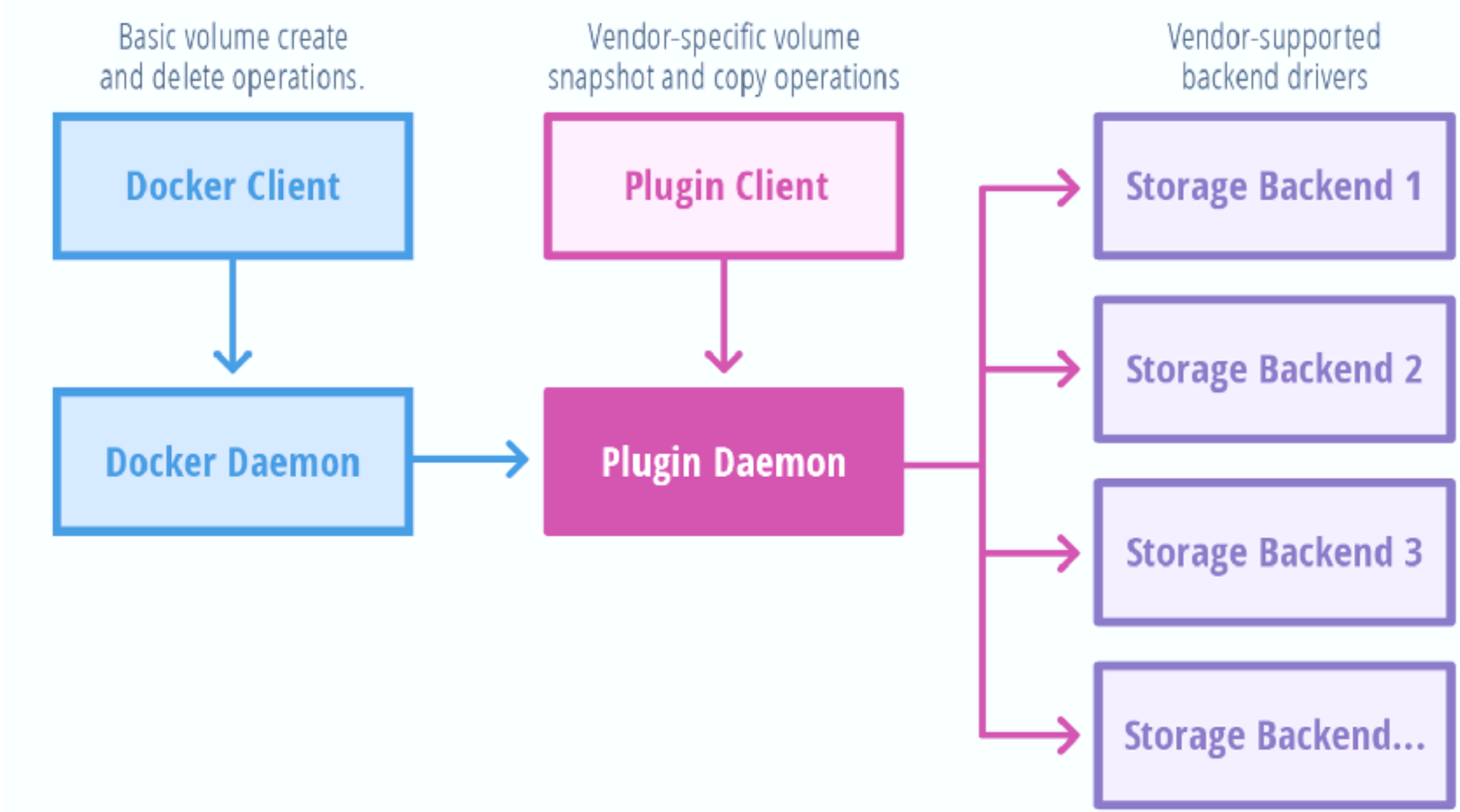
These applied changes will not be included when the image updates.

# Multi-Host persistence

## shared containers



# Docker volume plugin



# Connecting containers

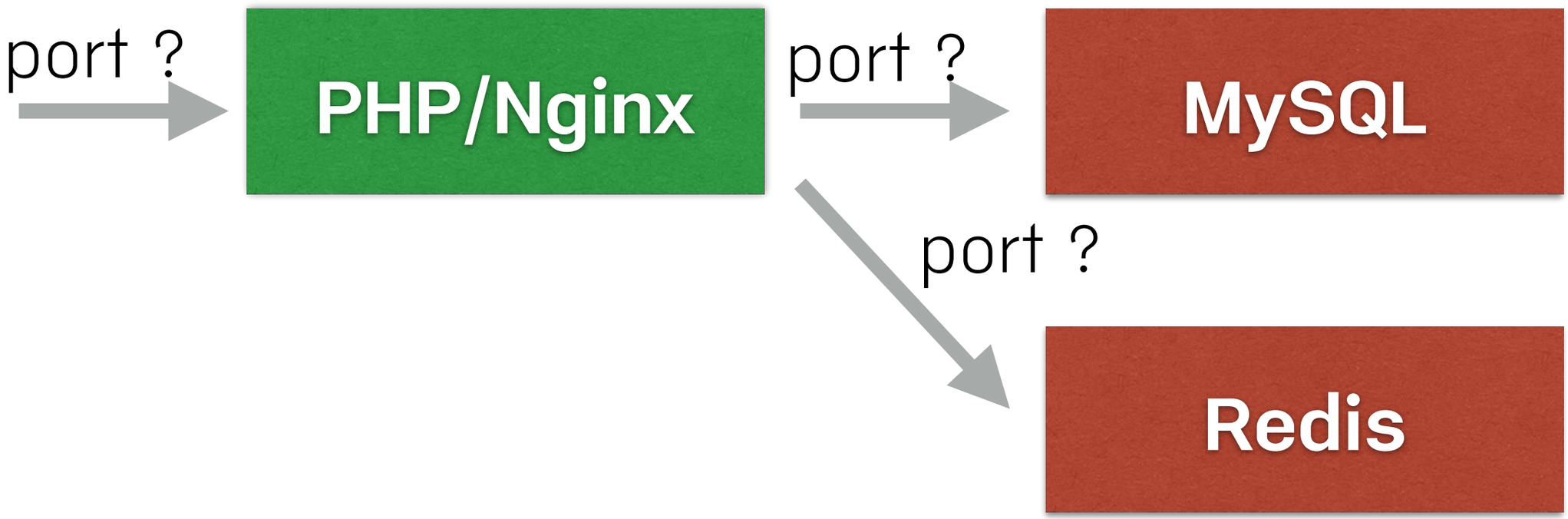
# Connecting containers

PHP/Nginx

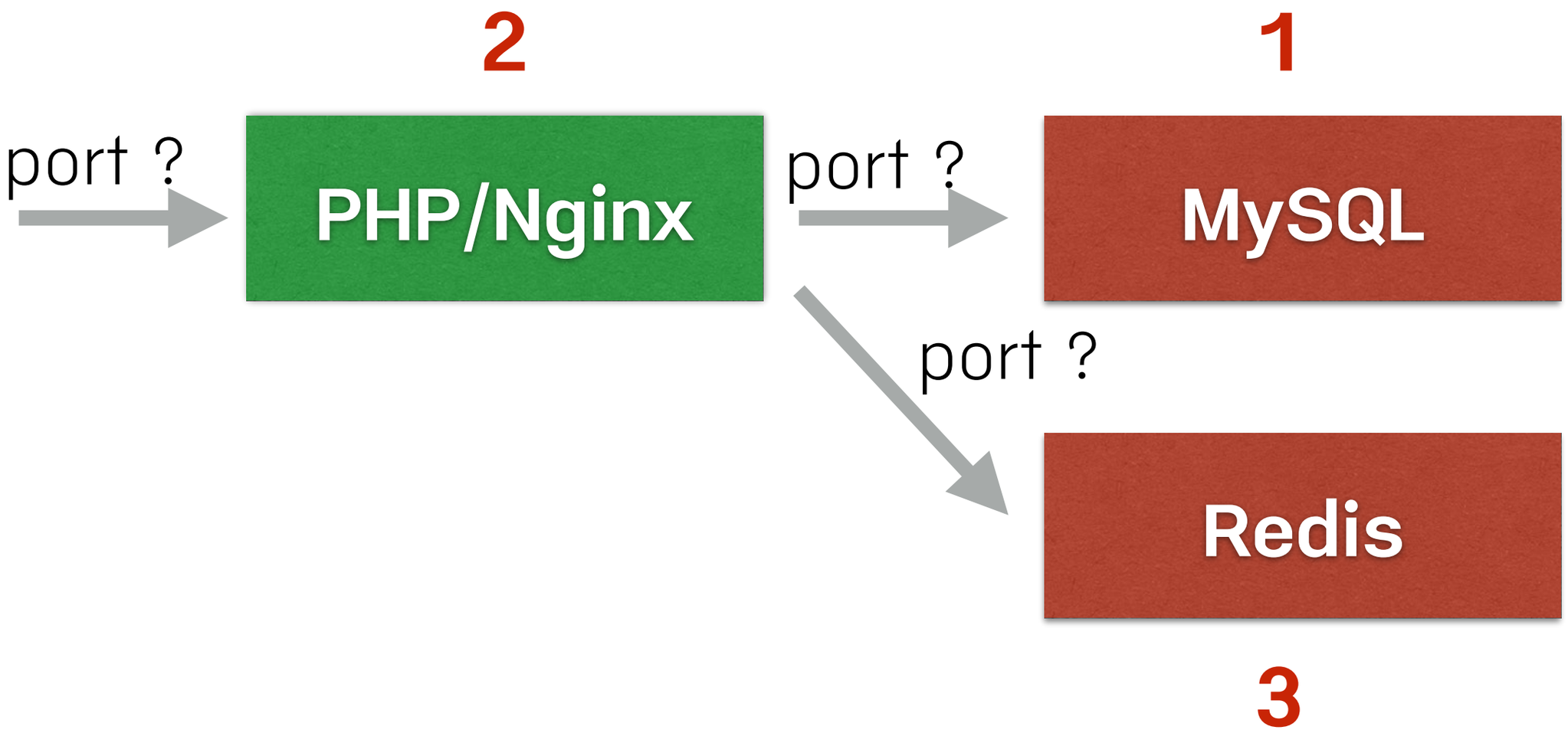
MySQL

Redis

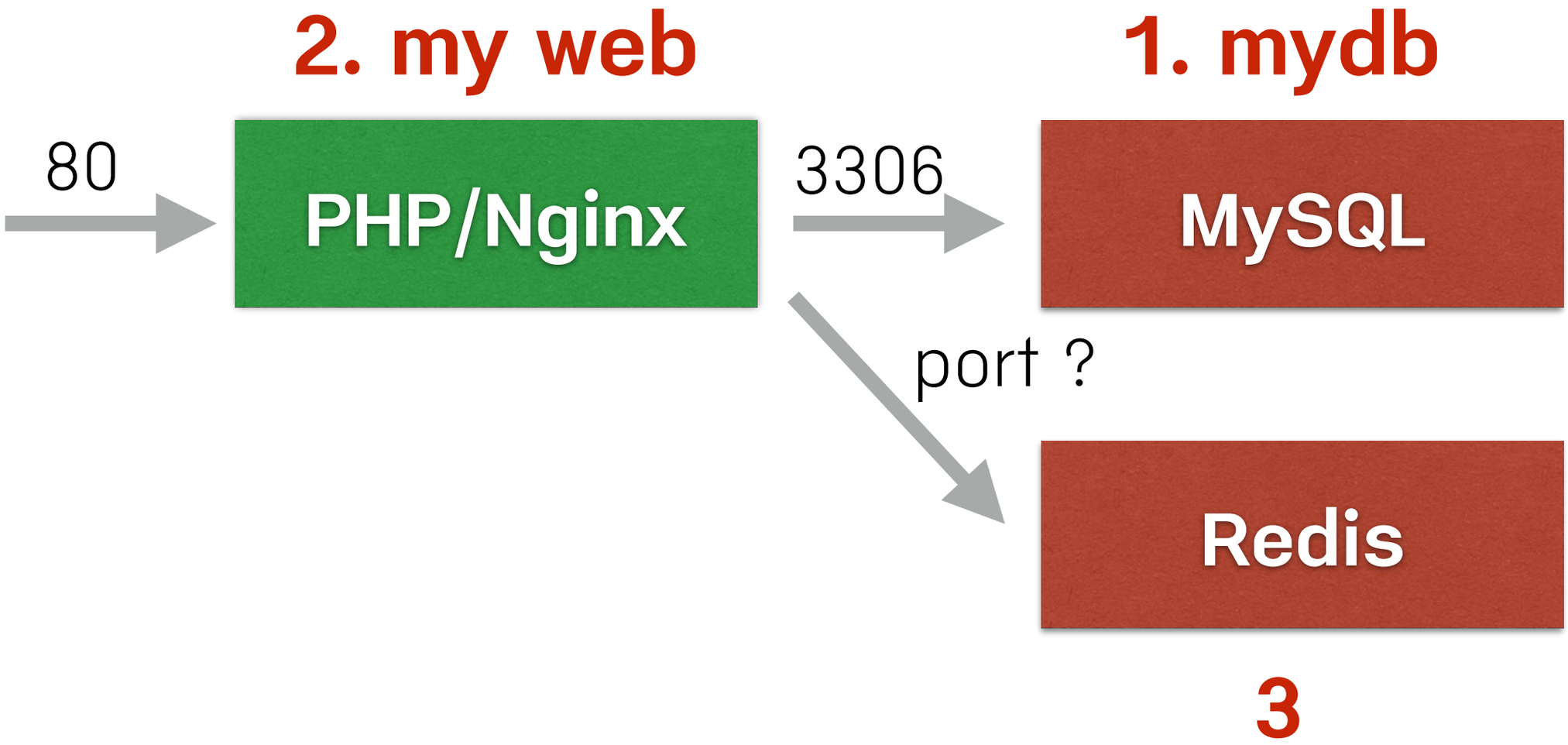
# Connecting containers



# Connecting containers

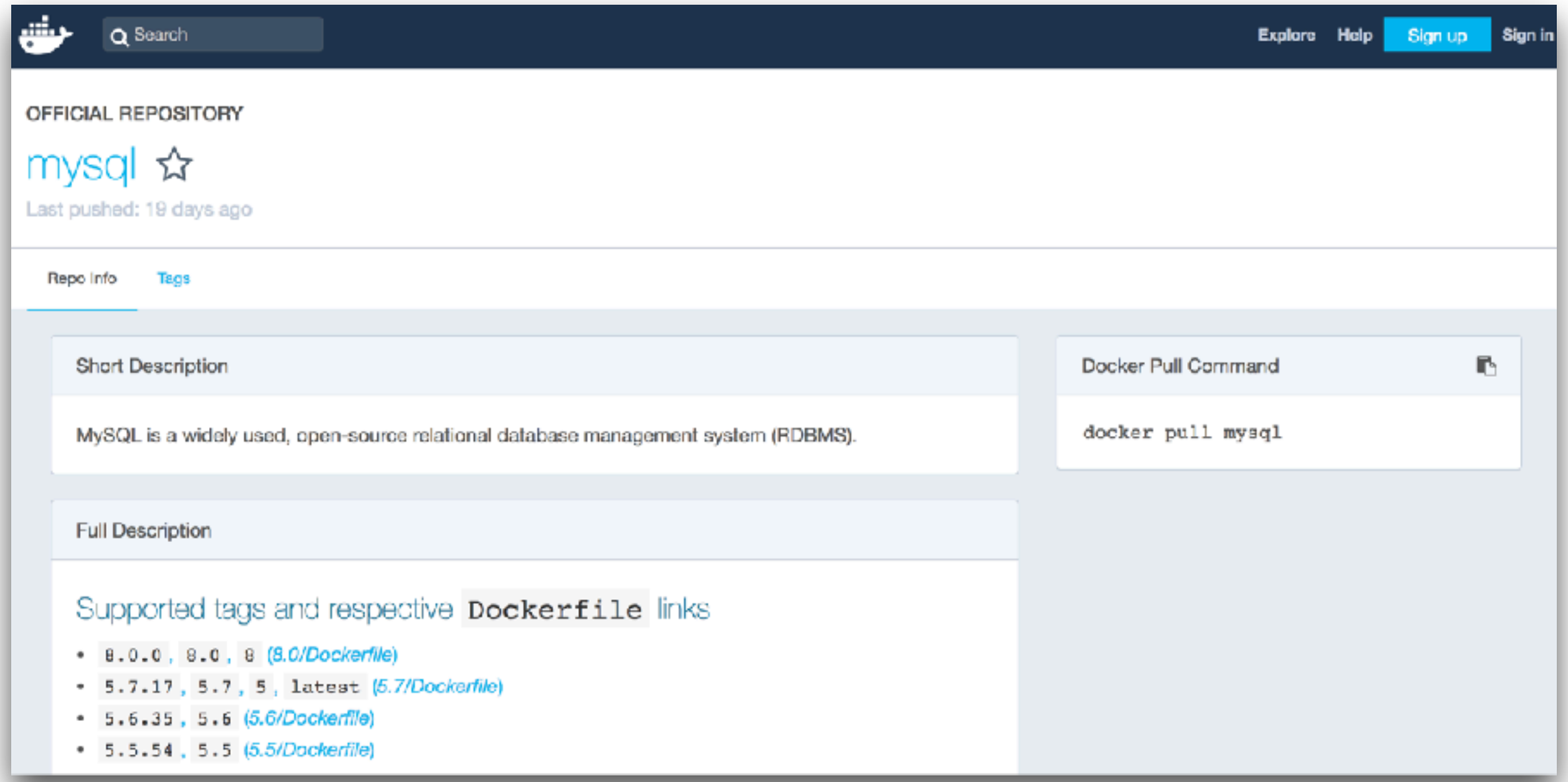


# Connecting containers



# 1. Create MySQL container

# Docker with MySQL




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

# Start MySQL container

```
$docker run  -d --name  mydb  \
  -p 3306:3306 \
  -e MYSQL_ROOT_PASSWORD=123456 \
  -e MYSQL_DATABASE=demo  \
mysql:latest
```

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

# How to test ?

$mysql -uroot -p123456

# How to test ?

$mysql -h<ip> -uroot -p123456

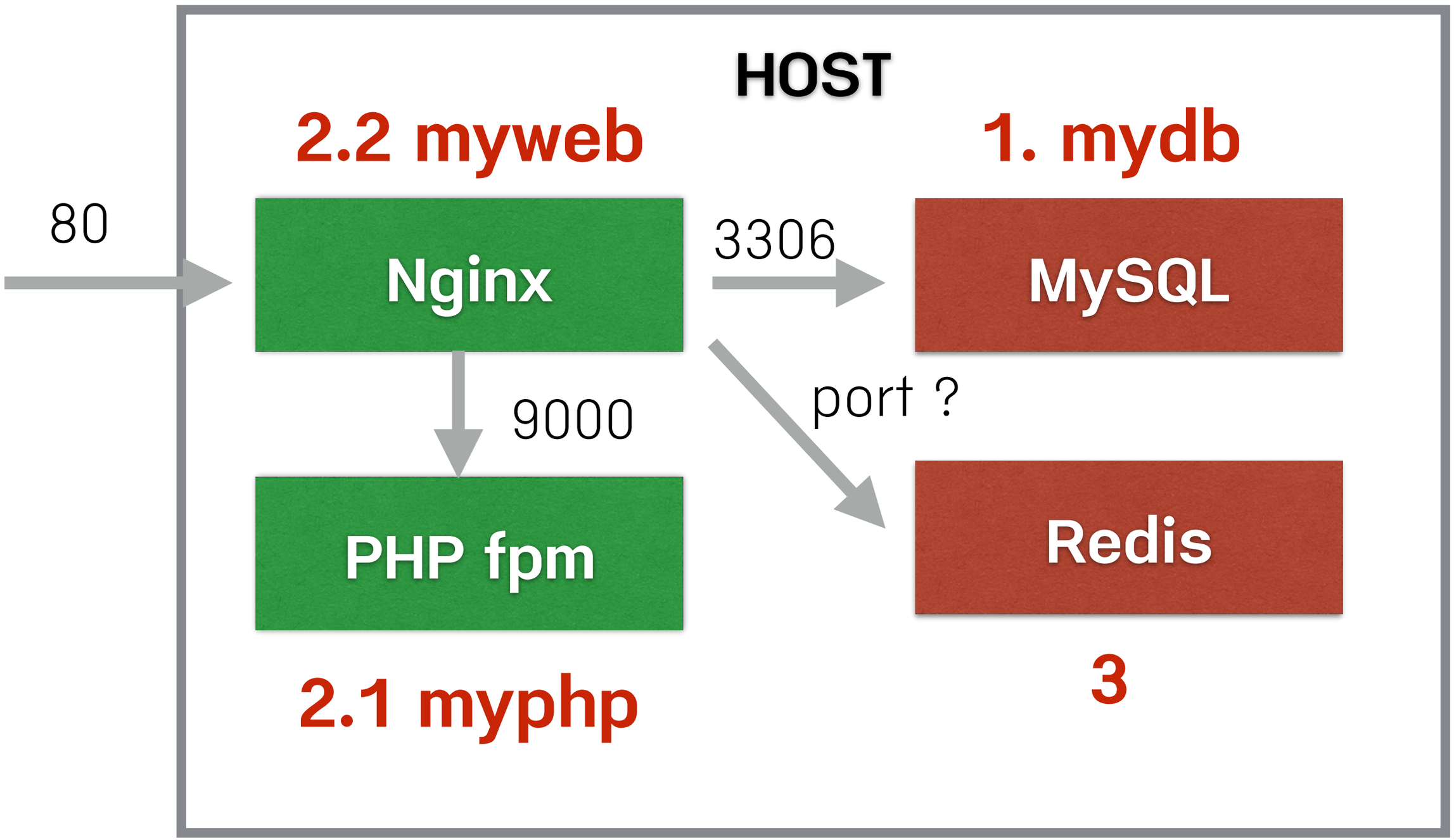
# Container inspect

$docker container inspect <id/name>


```
"SandboxKey": "/var/run/docker/netns/dc2b1c8b3cea",
"SecondaryIPAddresses": null,
"SecondaryIPv6Addresses": null,
"EndpointID": "05c5f7c3d66d9afc106c16f0aa3deee6ad1a561c13da070e3c1650133f7c7f4f",
"Gateway": "172.17.0.1",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0,
"IPAddress": "172.17.0.2",
"IPPrefixLen": 16,
"IPv6Gateway": "",
"MacAddress": "02:42:ac:11:00:02",
```

# 2. Create PHP and Nginx container

# Connecting containers



# Start PHP container

```
$docker container run -d --name myphp \
        -v $(pwd)/web:/web \
        php:fpm
```

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

# Start Nginx container

```
$docker container run --rm -d --name myweb \
    -p 80:80 \
    -v $(pwd)/site.conf:/etc/nginx/conf.d/
default.conf \
        --volumes-from myphp \
        --link myphp nginx:latest
```

# Easy with docker-compose

```
version: "2"

services:
    myweb:
        image: nginx:latest
        ports:
            - "80:80"
        volumes:
            - ./site.conf:/etc/nginx/conf.d/default.conf
        volumes_from:
            - myphp
myphp:
    image: php:fpm
    volumes:
        - ./web:/web
```

# 3. Link php to mysql

# 4. Create Redis container

# 5. Link Web and Redis

# 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

# Workshop

# Resources

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