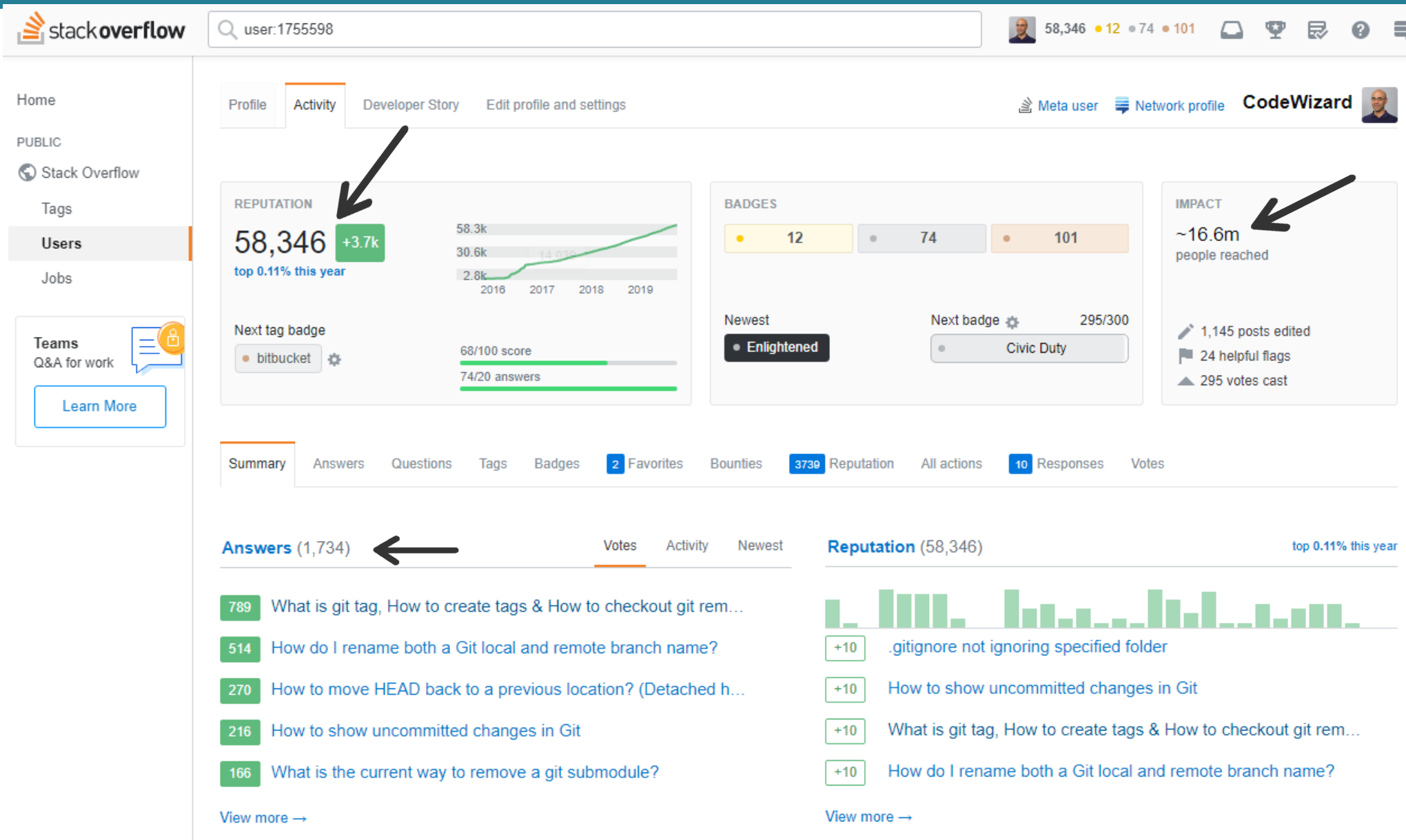
Nir Geier - CodeWizard


WHO AM I?

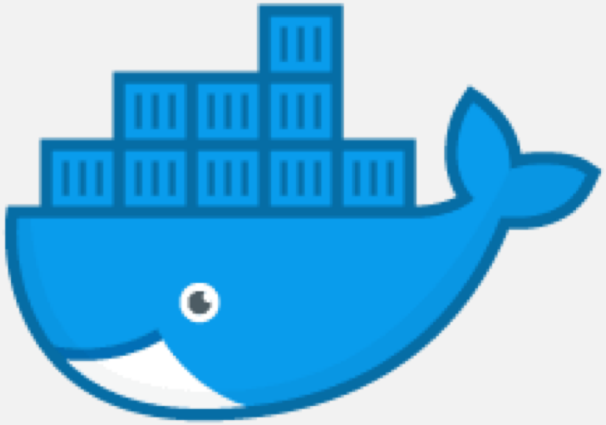

Western Digital
TheMarker
CYBERARK
NICE התעשייה
JOHN BRYCE
תלמד הי-טק. זה עובד!
a matrix company
הטכניון
מכון טכנולוגי לישראל
לאומי
leumi
Texas Instruments
GALA CORAL GROUP
netcraft
TIBA
PARKING SYSTEMS
VERINT
ישראכרט
amdocs
embrace challenge experience success
MARVELL
KRYON
SYSTEMS
אלביט מערכות
יבשה וחקשוב
הראל
משרד הביטחון
MINISTRY OF DEFENSE
Check Point
אוניברסיטת תל-אביב
SKYBOX
SECURITY
ynet
ידיעות אחרונות
רפאל
מערכות לחימה מתקדמות בע"מ
SanDisk
E4D
Mobile | Web | Business Solutions
דיסקונט
הבינתחומי הרצליה
bmcsoftware
בנק הפועלים
intel
Hewlett Packard Enterprise
RADCOM
PEPPER.
אל תקרא לי בנק
ELAD group
moovit
HACKERU
הפריצה שלך להייטק
Medinol
Ingenuity for life
Gilat
HACKERU
PRO UPDATE NEEDED


stackoverflow
user:1755598
58,346 12 74 101
Home
PUBLIC
Stack Overflow
Tags
Users
Jobs
Teams
Q&A for work
Learn More
Profile
Activity
Developer Story
Edit profile and settings
Meta user
Network profile
CodeWizard
REPUTATION
58,346
+3.7k
top 0.11% this year
58.3k
30.6k
2.8k
2016
2017
2018
2019
Next tag badge
bitbucket
68/100 score
74/20 answers
BADGES
12
74
101
Newest
Enlightened
Next badge
295/300
Civic Duty
IMPACT
~16.6m
people reached
1,145 posts edited
24 helpful flags
295 votes cast
Summary
Answers
Questions
Tags
Badges
2 Favorites
Bounties
3739 Reputation
All actions
10 Responses
Votes
Answers (1,734)
Votes
Activity
Newest
789 What is git tag, How to create tags & How to checkout git rem…
514 How do I rename both a Git local and remote branch name?
270 How to move HEAD back to a previous location? (Detached h…
216 How to show uncommitted changes in Git
166 What is the current way to remove a git submodule?
View more →
Reputation (58,346)
top 0.11% this year
+10 .gitignore not ignoring specified folder
+10 How to show uncommitted changes in Git
+10 What is git tag, How to create tags & How to checkout git rem…
+10 How do I rename both a Git local and remote branch name?
View more →

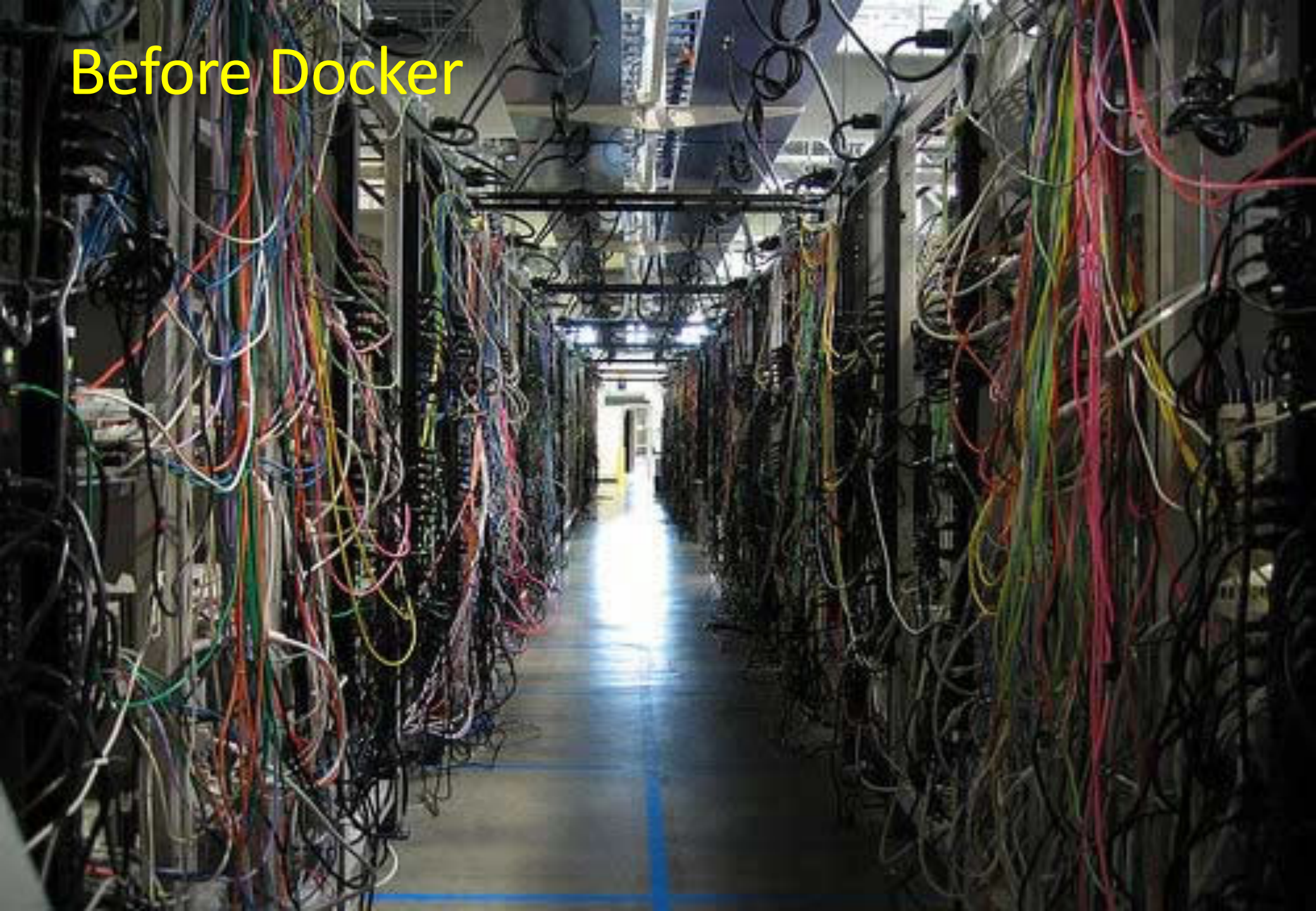
Agenda

- Docker technology overview
- Containers / Images and beyond
- Docker commands
- Docker file
- Demos & Hands on

Why

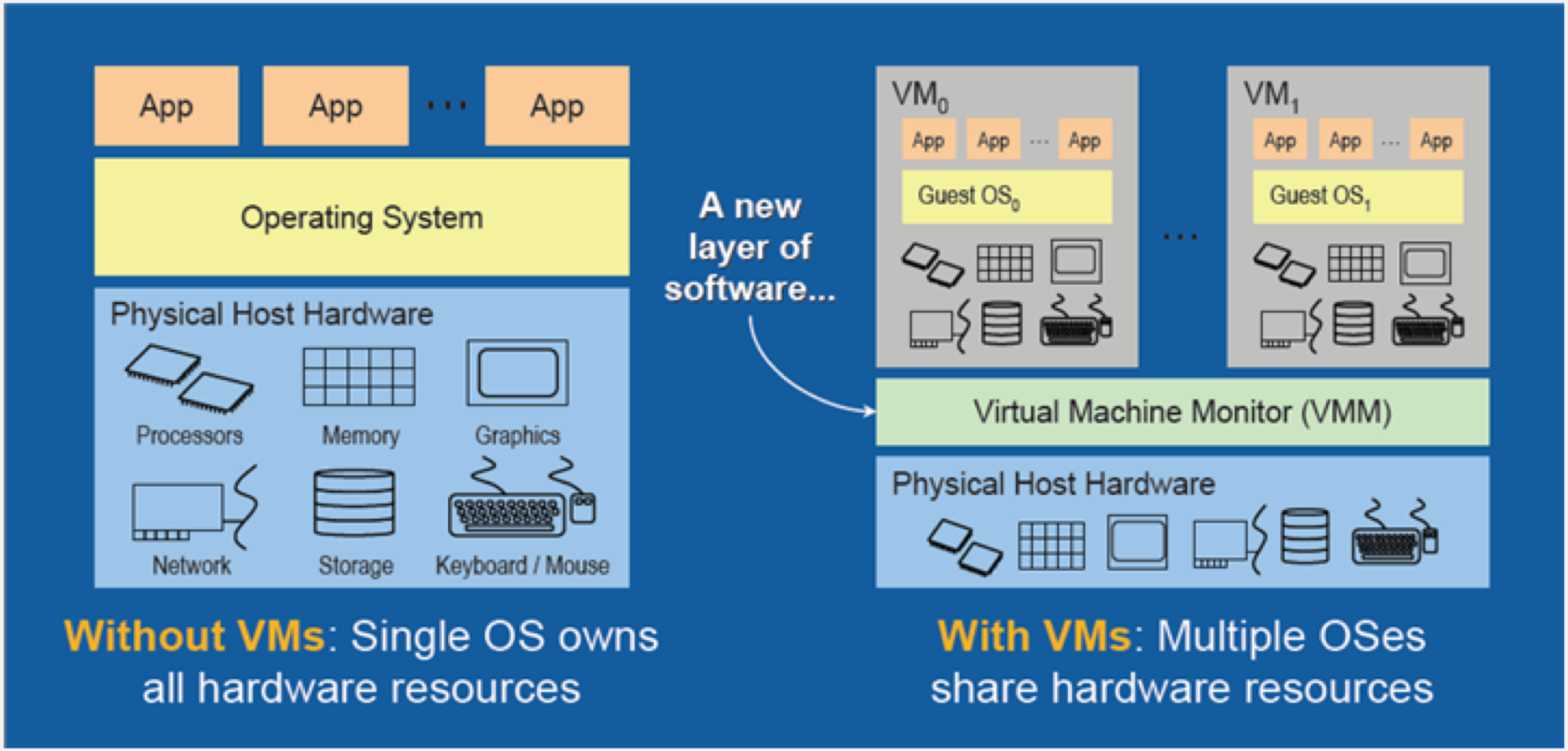


docker

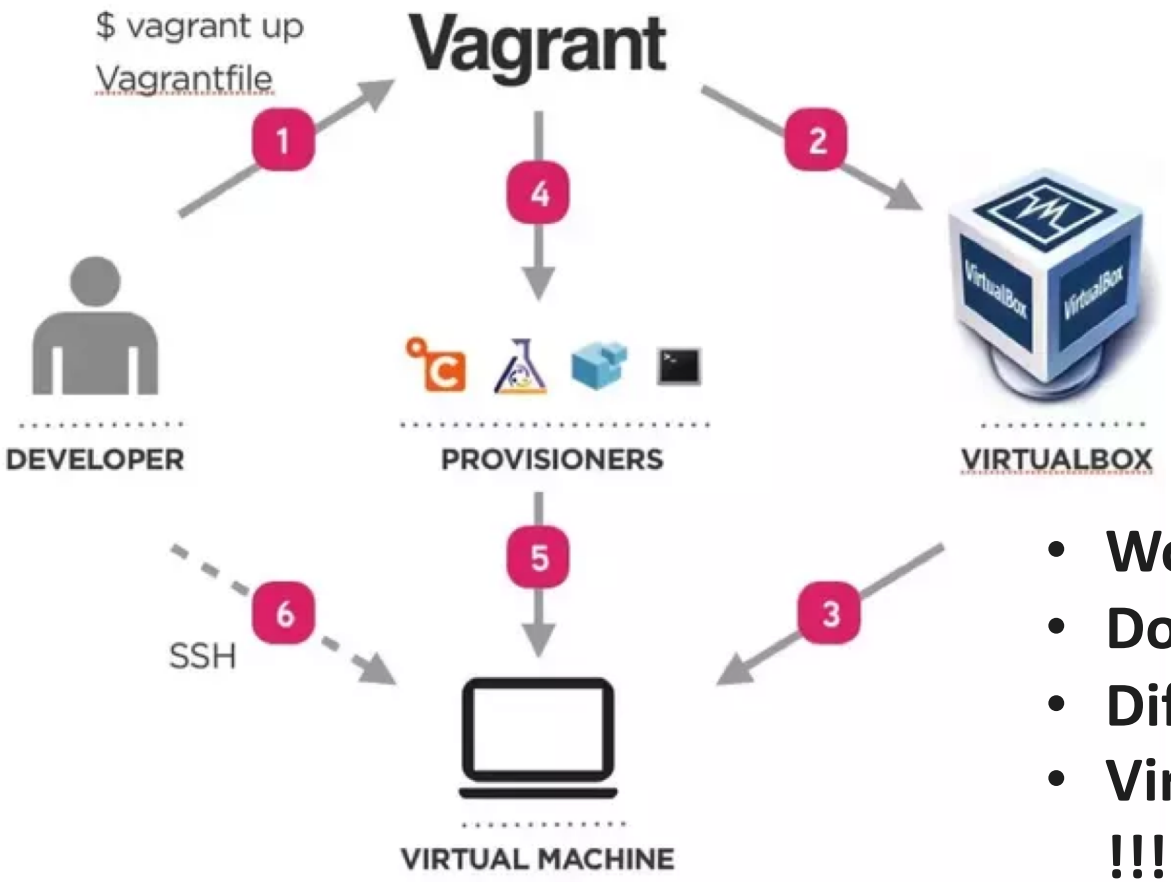

Before Docker


Before Docker
tex

Before Docker (VM)



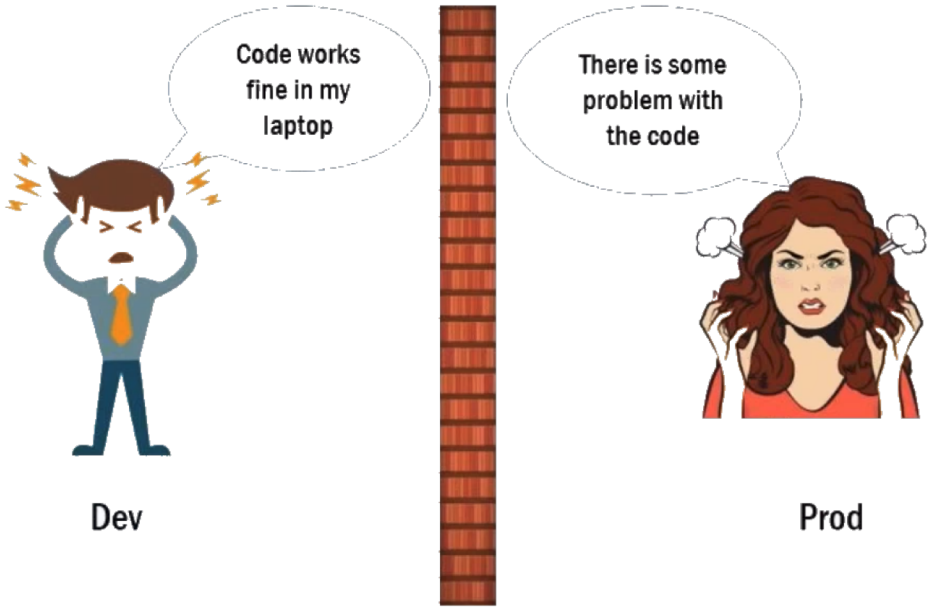
Before Docker (VM - Vagrant)



- Work on my machine (Win/Mac)
- Doesn't work on other machines
- Different images
- Virtual boxes (different boxes !!!)
- Provisioning script
- Code is part of the Virtual machine

Before Docker

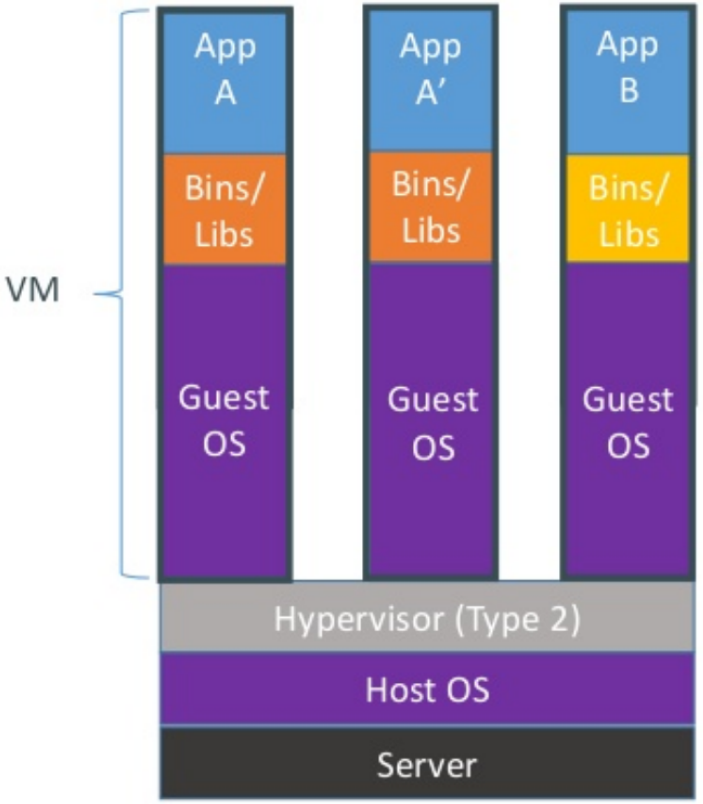
An application works in developer's laptop but not in testing or production. This is due to difference in computing environment between Dev, Test and Prod.



In Dev there can be a software that is upgraded and in Prod the old version of software might be present

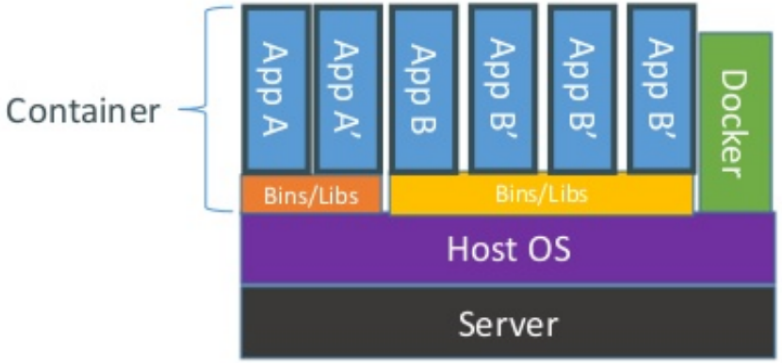
Today (Docker)

Containers vs. VMs



Containers are isolated, but share OS and, where appropriate, bins/libraries

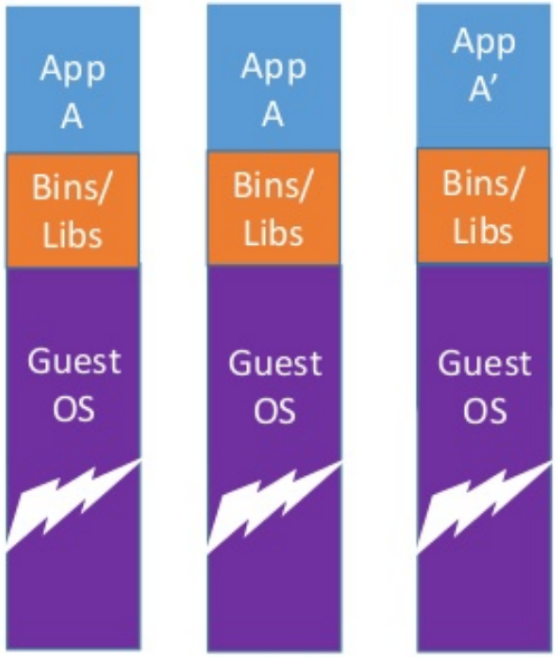
...result is significantly faster deployment, much less overhead, easier migration, faster restart



Today (Docker)

Why are Docker containers lightweight?

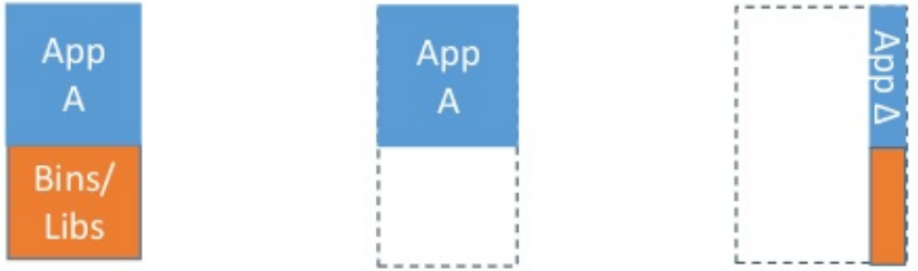
VMs



VMs

Every app, every copy of an app, and every slight modification of the app requires a new virtual server

Containers



Original App
(No OS to take up space, resources, or require restart)

Copy of App
No OS. Can Share bins/libs

Modified App

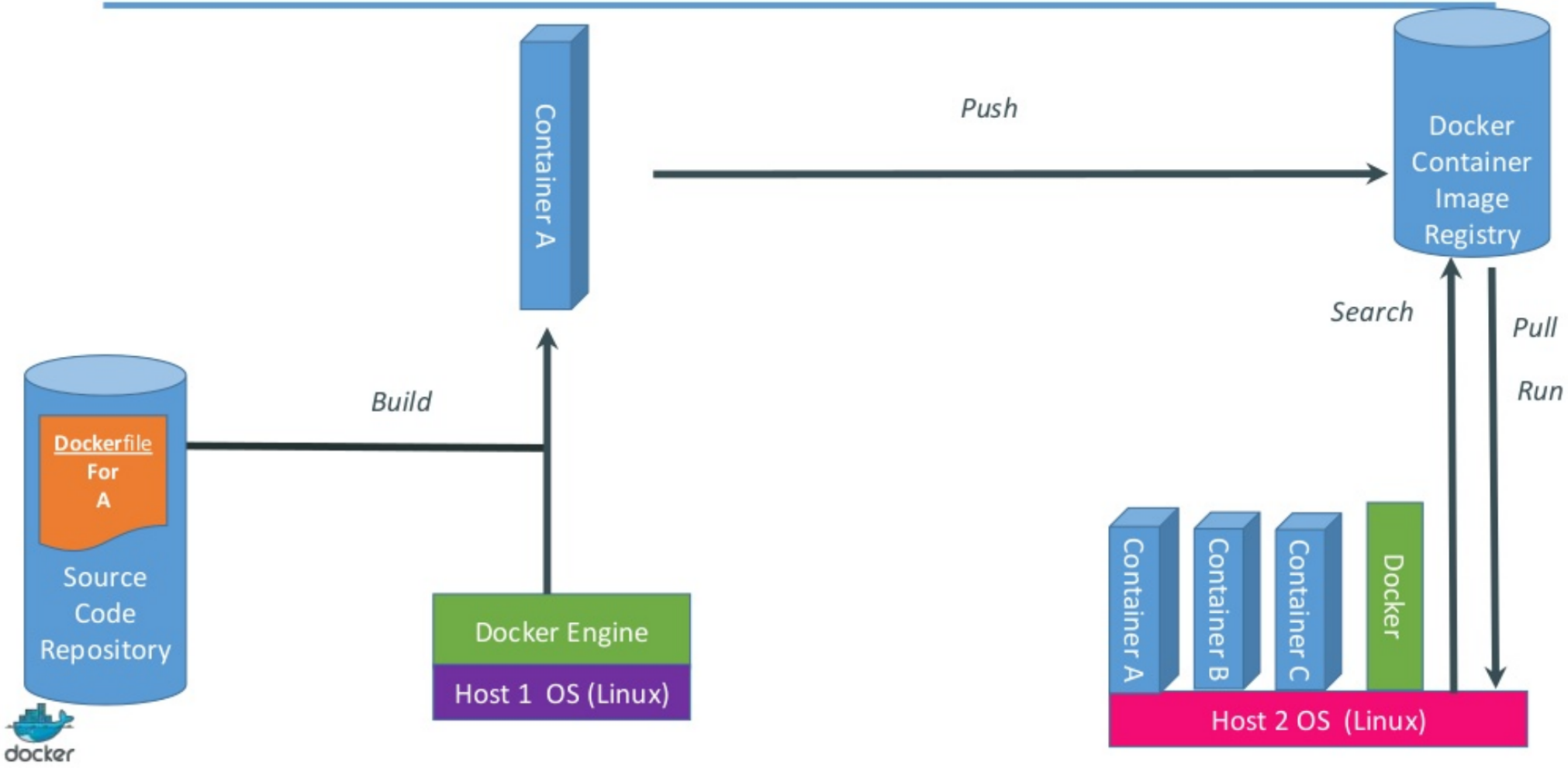
Copy on write capabilities allow us to only save the diffs Between container A and container A'

LXC

- LXC (LinuX Containers) was the **first, most complete implementation of Linux container manager**.
- It was implemented in **2008** using **cgroups** and Linux **namespaces**, and it works on a single Linux kernel without requiring any patches.
- **Namespaces** allow non-root users to pretend to be root

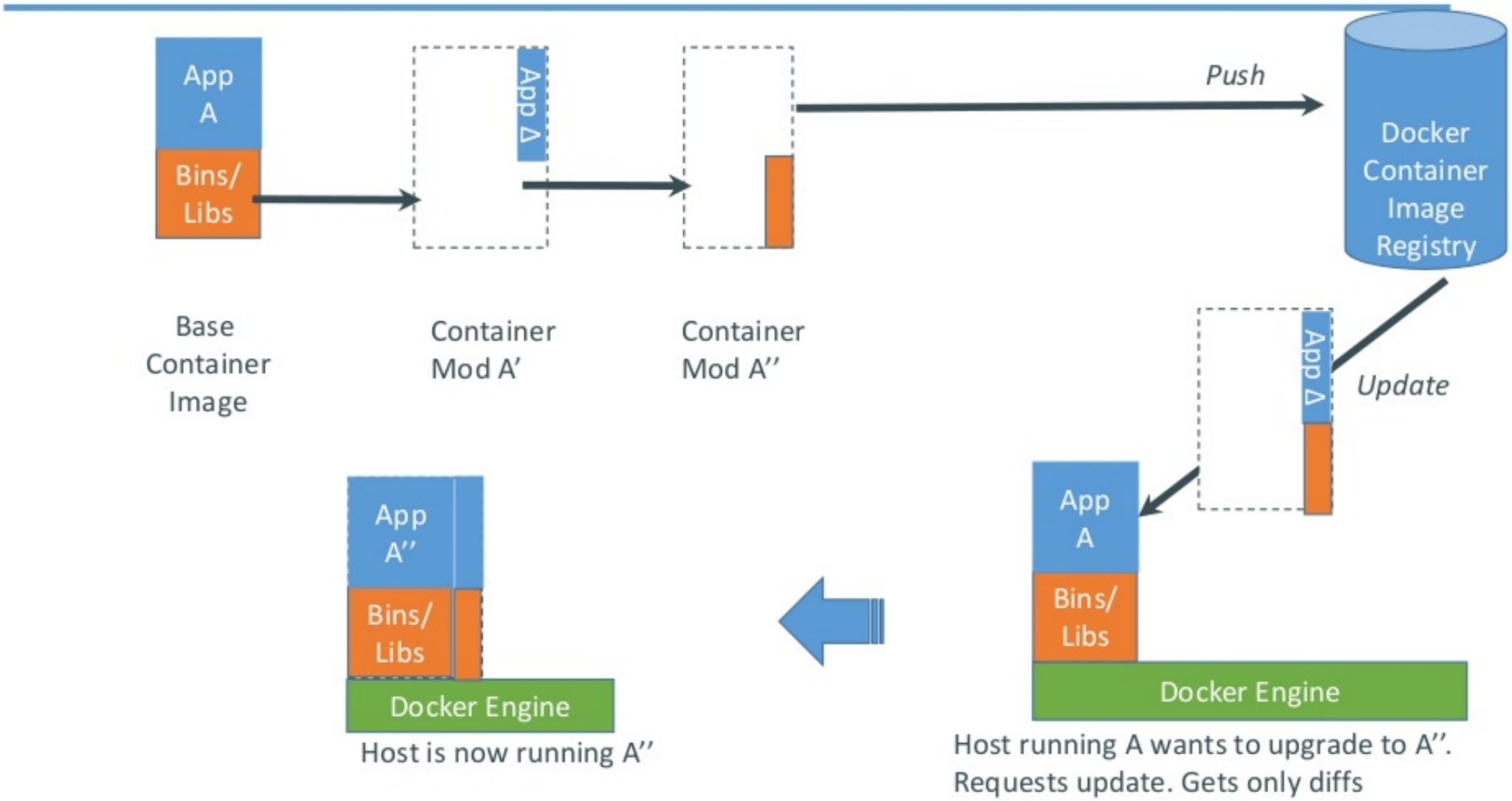
Today (Docker)

What are the basics of the Docker system?



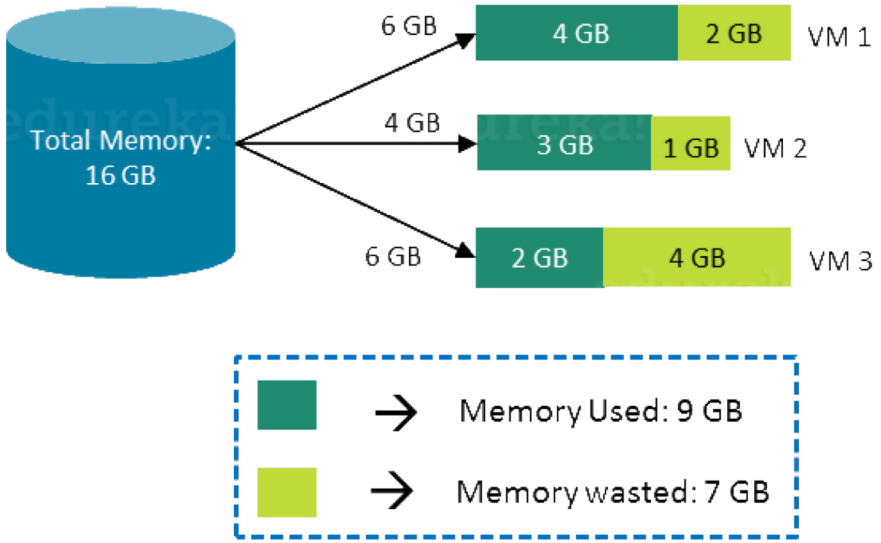
Today (Docker)

Changes and Updates



Today (Docker)

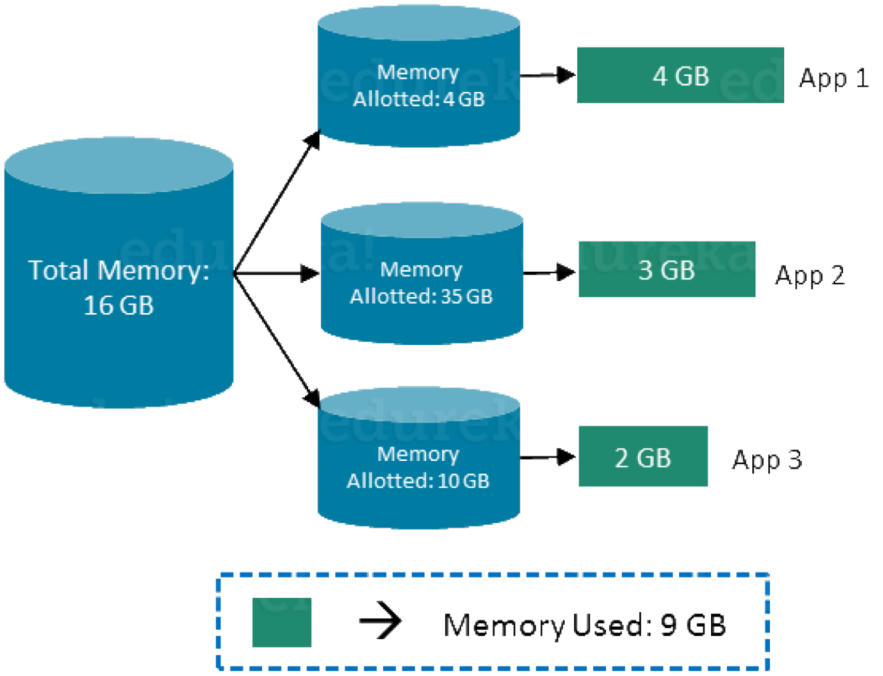
In case of Virtual Machines



7 Gb of Memory is blocked and cannot be allotted to a new VM

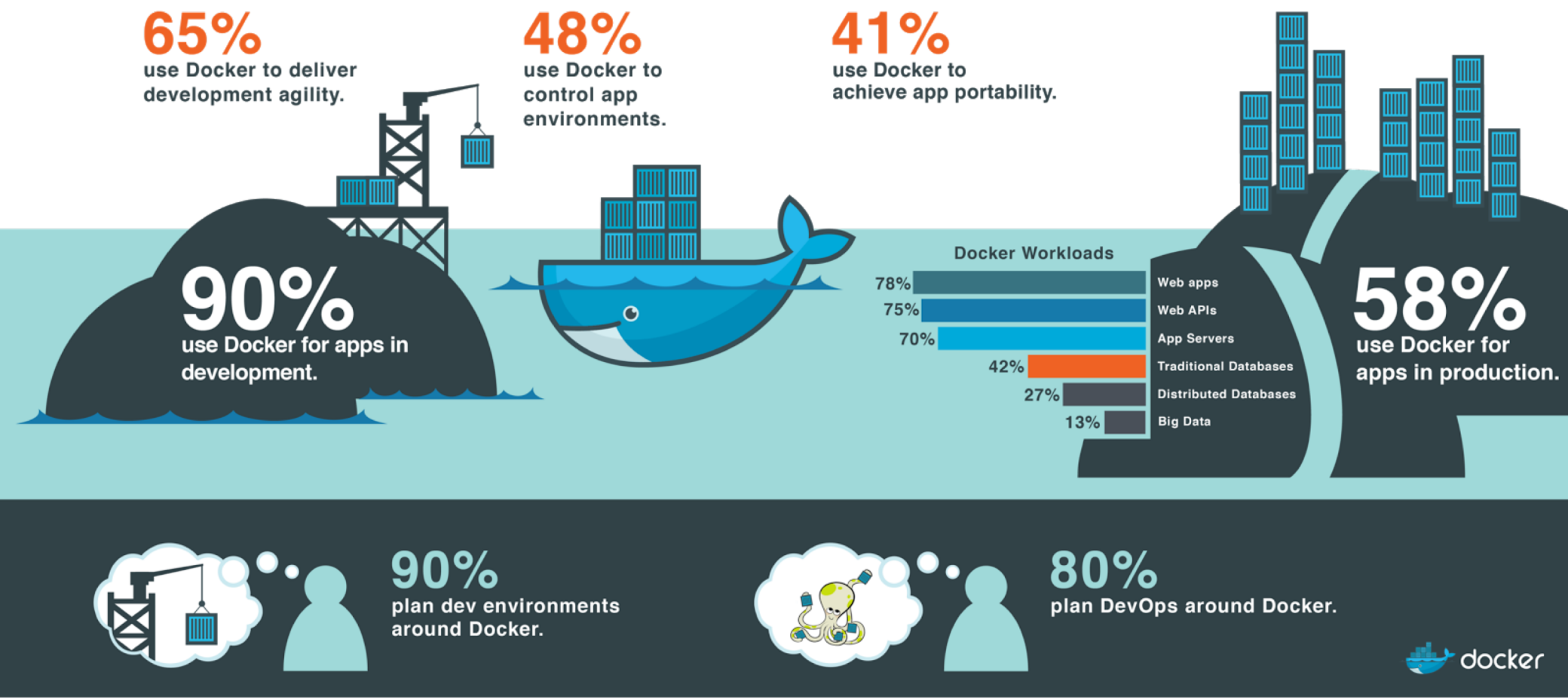
In case of Docker

edureka!



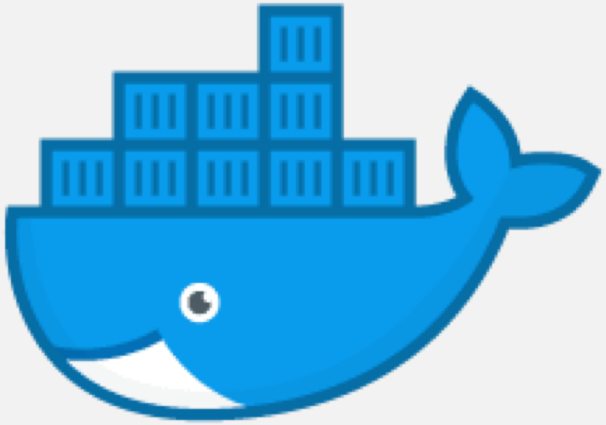
Only 9 GB memory utilized;
7 GB can be allotted to a new Container

Today (Docker)



90%
plan dev environments around Docker.

80%
plan DevOps around Docker.



docker

Intro

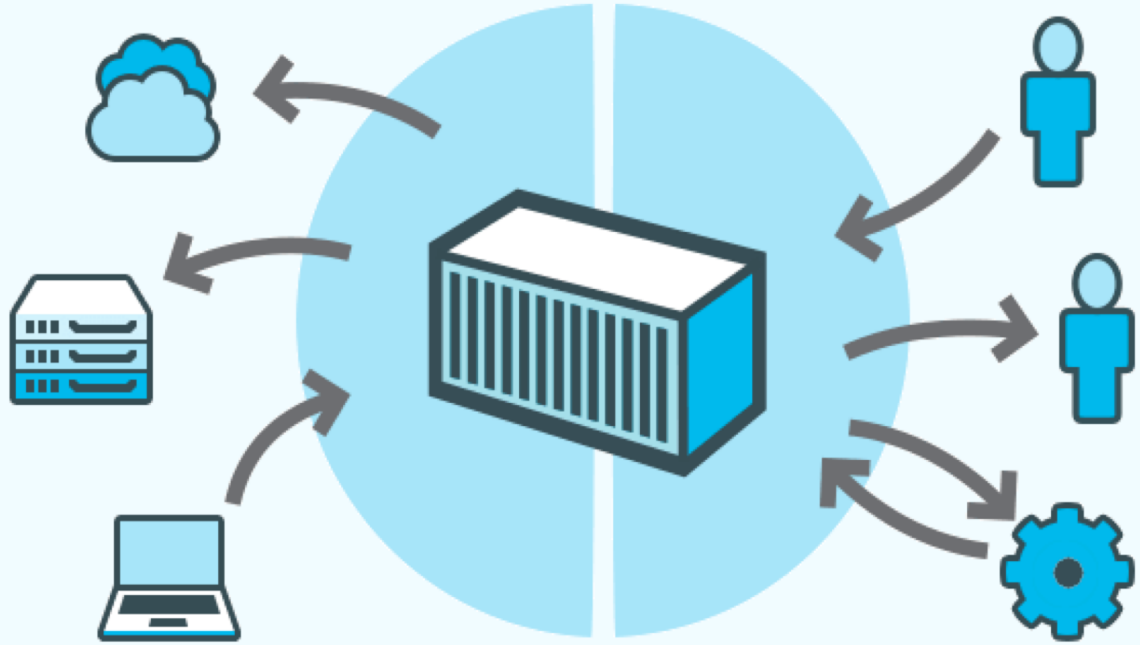
Docker – Info

- Started in March 2013
- Written in GO
- Easy API for managing containers
- Build on cgroup, namespacing, Libcontainers (LXC was used up to v1.8, now based upon opencontainers/runc)

Docker – Info

What Is Docker?

An open platform for distributed applications



Docker Engine

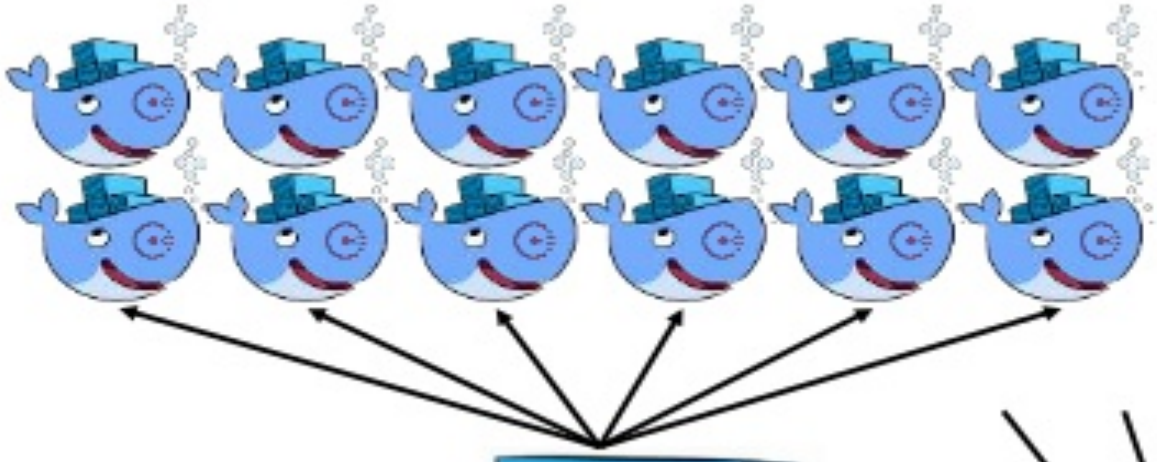
A portable, lightweight application runtime and packaging tool.

Learn More

Docker Hub

A cloud service for sharing applications and automating workflows.

Execution platform



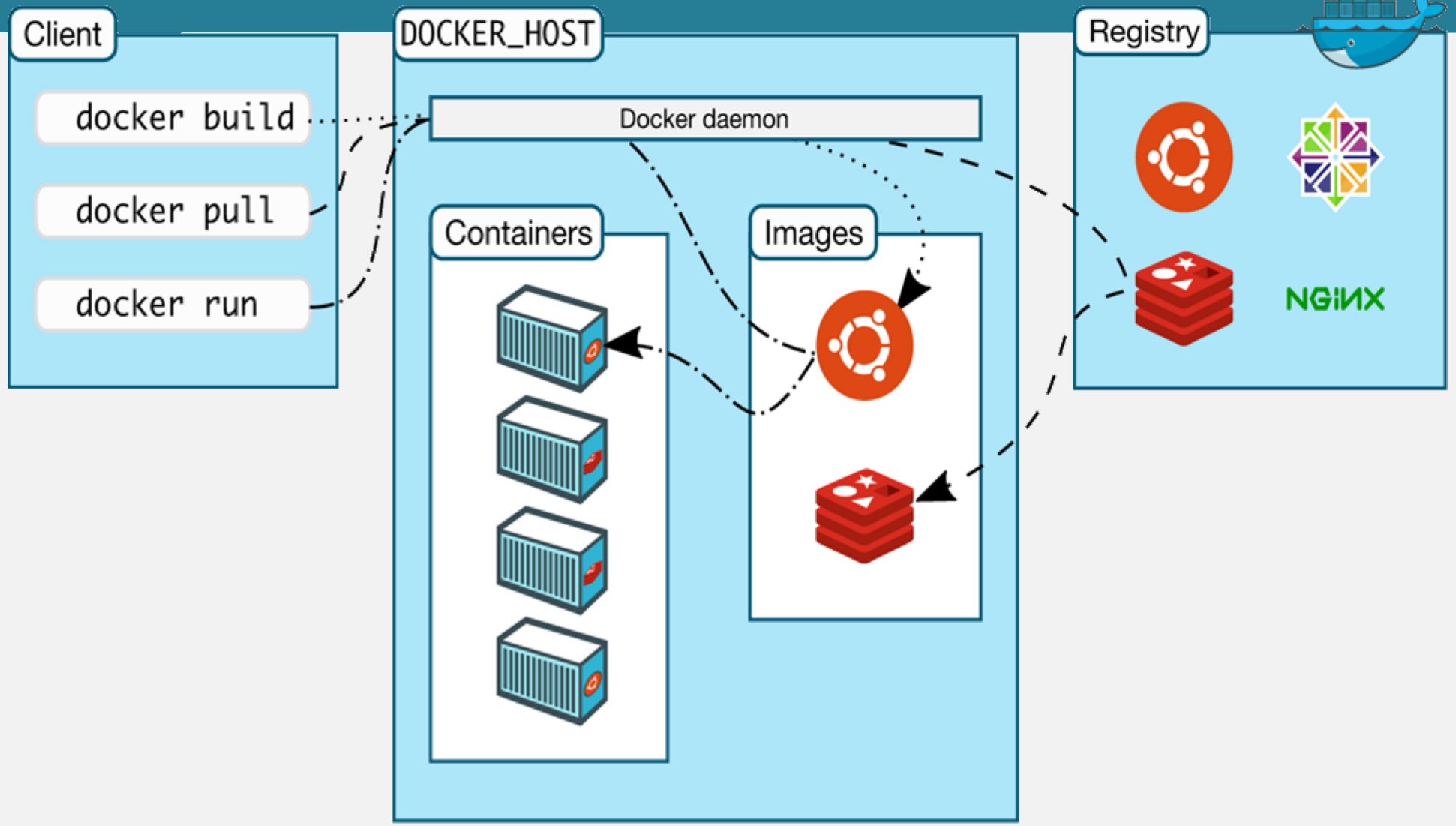
The registrator **discovers** new containers and **feeds** the registry

Docker-machine

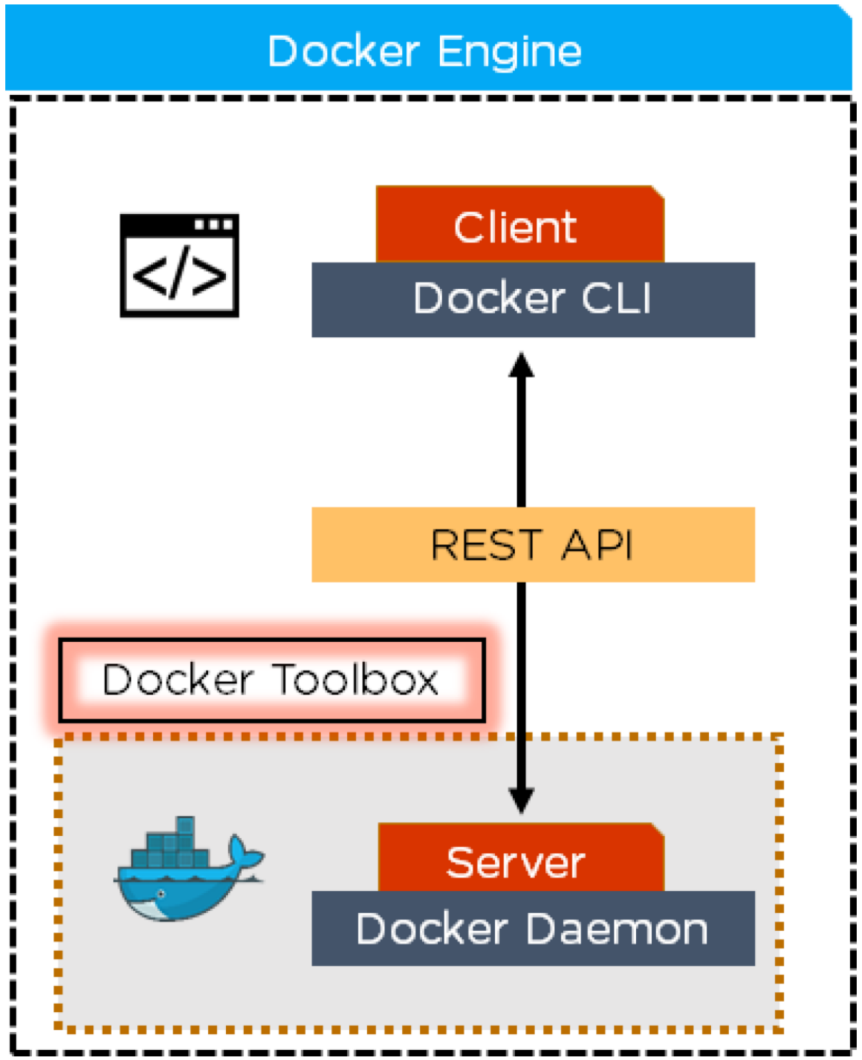
Docker-swarm

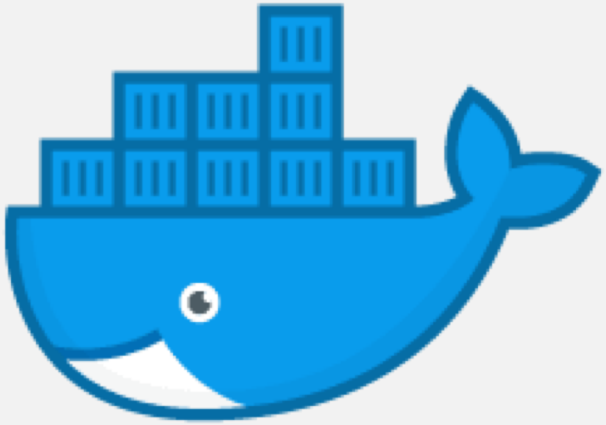
Registry

Docker – Info


Client
docker build
docker pull
docker run
DOCKER_HOST
Docker daemon
Containers
Images
Registry
NGINX

Docker – Info


Docker Engine
Client
Docker CLI
REST API
Docker Toolbox
Server
Docker Daemon



docker

Terminology

Terminology

- **AUFS**

  Union file system

- **Docker engine (aka Docker)**

  The Docker daemon which manage containers and images (using namespaces and cgroup)

- **Docker client**

  The runtime binary code which interact with the Docker engine

- **Docker image**

  The image is read only File System used to assemble up containers

- **Docker container**

  Collection of different images

- **Host**
  - Computer which runs Docker engine

Terminology

- **Docker file**

  Instructions for creating containers

- **Docker hub / registry**

  Registry service to list your containers. Can be private or public and stores docker images

- **Docker swarm / Docker compose**

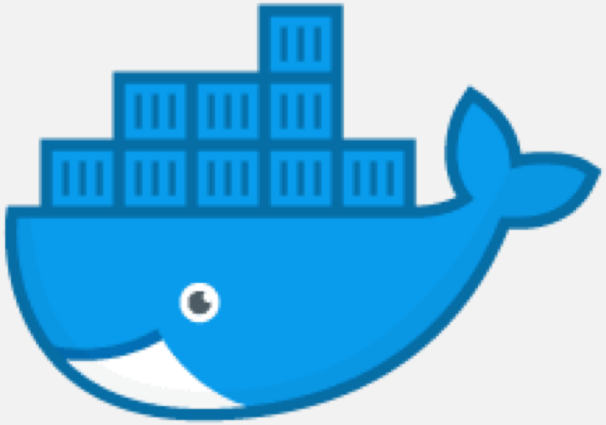
  Mazo / Kubernetes / Docker data center / GKC

- **Docker compose**

  Creates and manage multi-containers architecture

- **Docker swarn**

  Creates and manage multi-containers architecture (ex: clusters)



docker

Images

AUFS

AUFS is **a** **u**nion **f**ile **s**ystem, which means that it layers multiple directories on a single **Linux** host and presents them as a single directory.

These directories are called branches in **AUFS** terminology, and layers in Docker terminology. The unification process is referred to as a union mount.

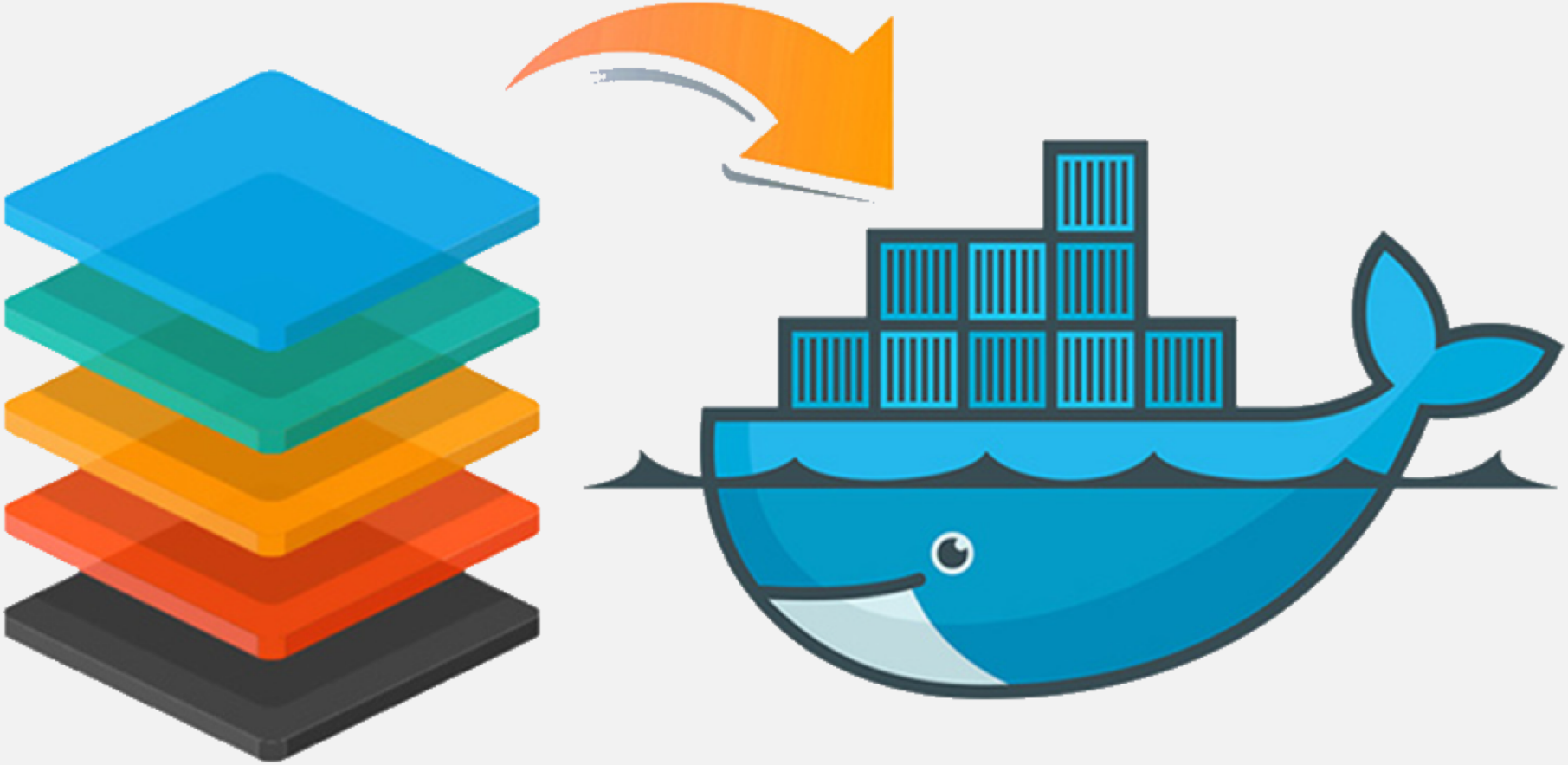
Docker images

- Docker images are **read only templates**
- Images are the application which we run
- Docker images are launched from containers
- Each image contain series of layers using the **AUFS**
- Images are pulled from hub/registry ex (hub.docker.com)

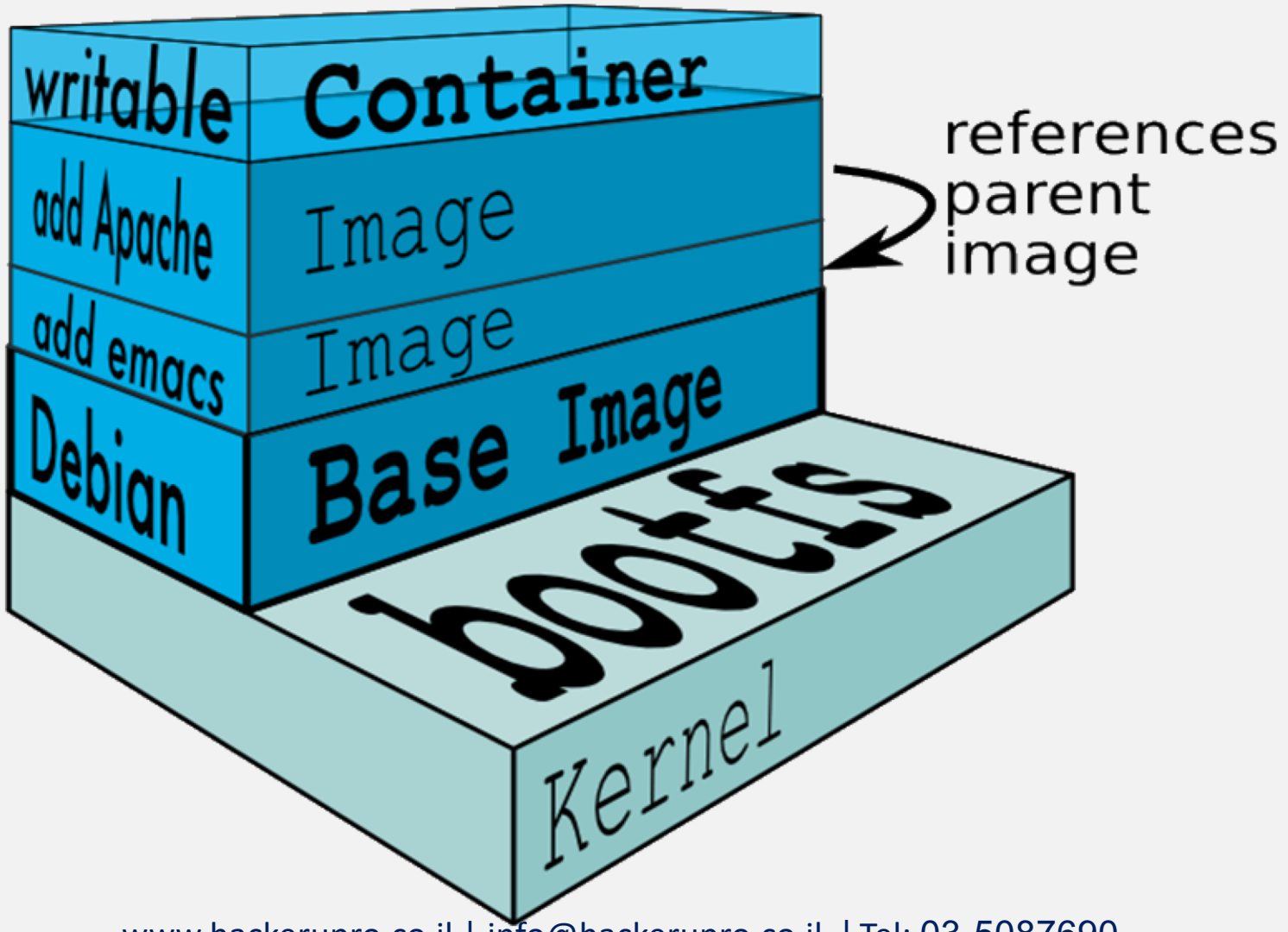
Docker Containers

- An image instance
- Running as process
- We can have multiple container running the same image (each run generate unique id)
- Containers can share information (volumes)

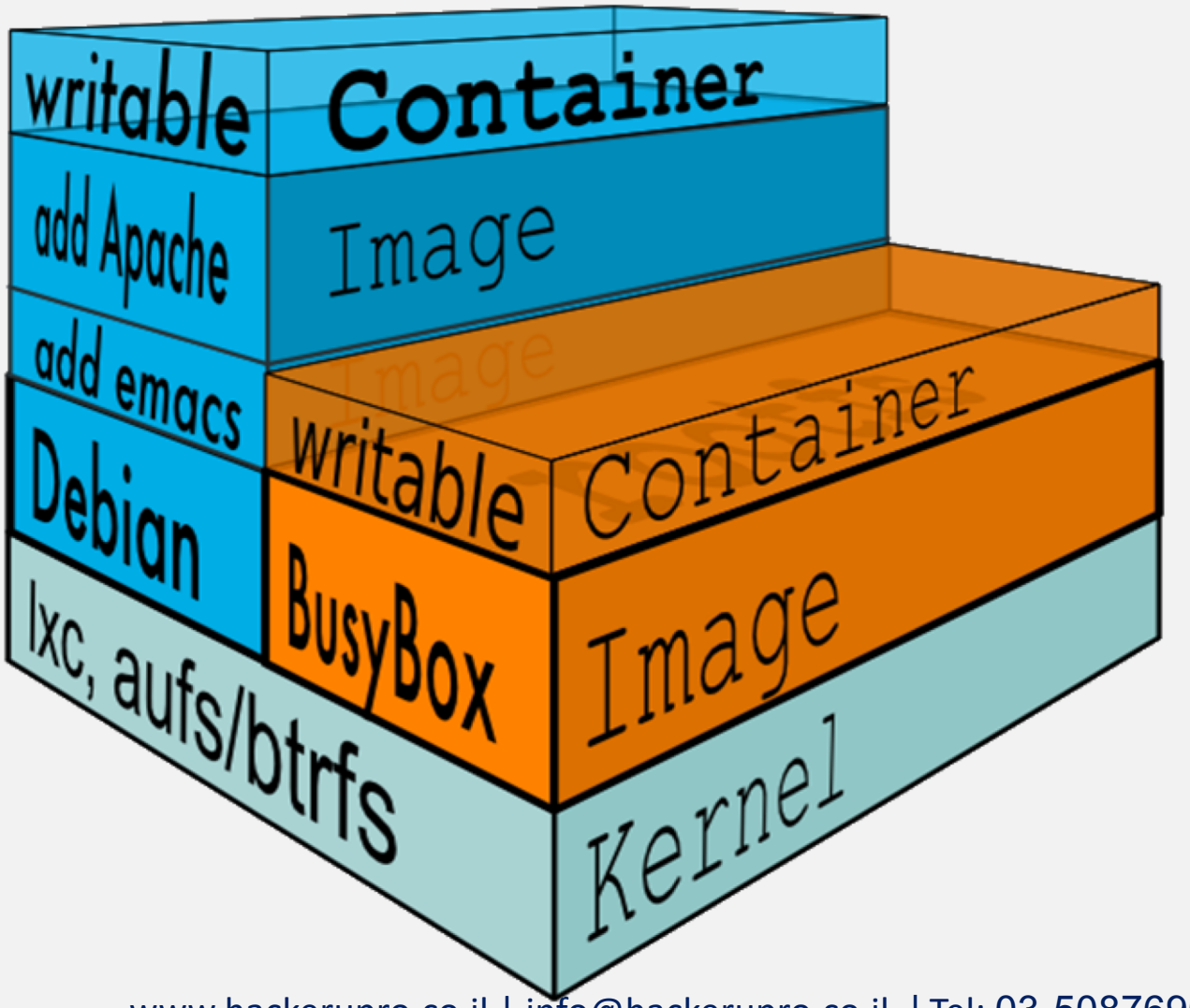
Docker images



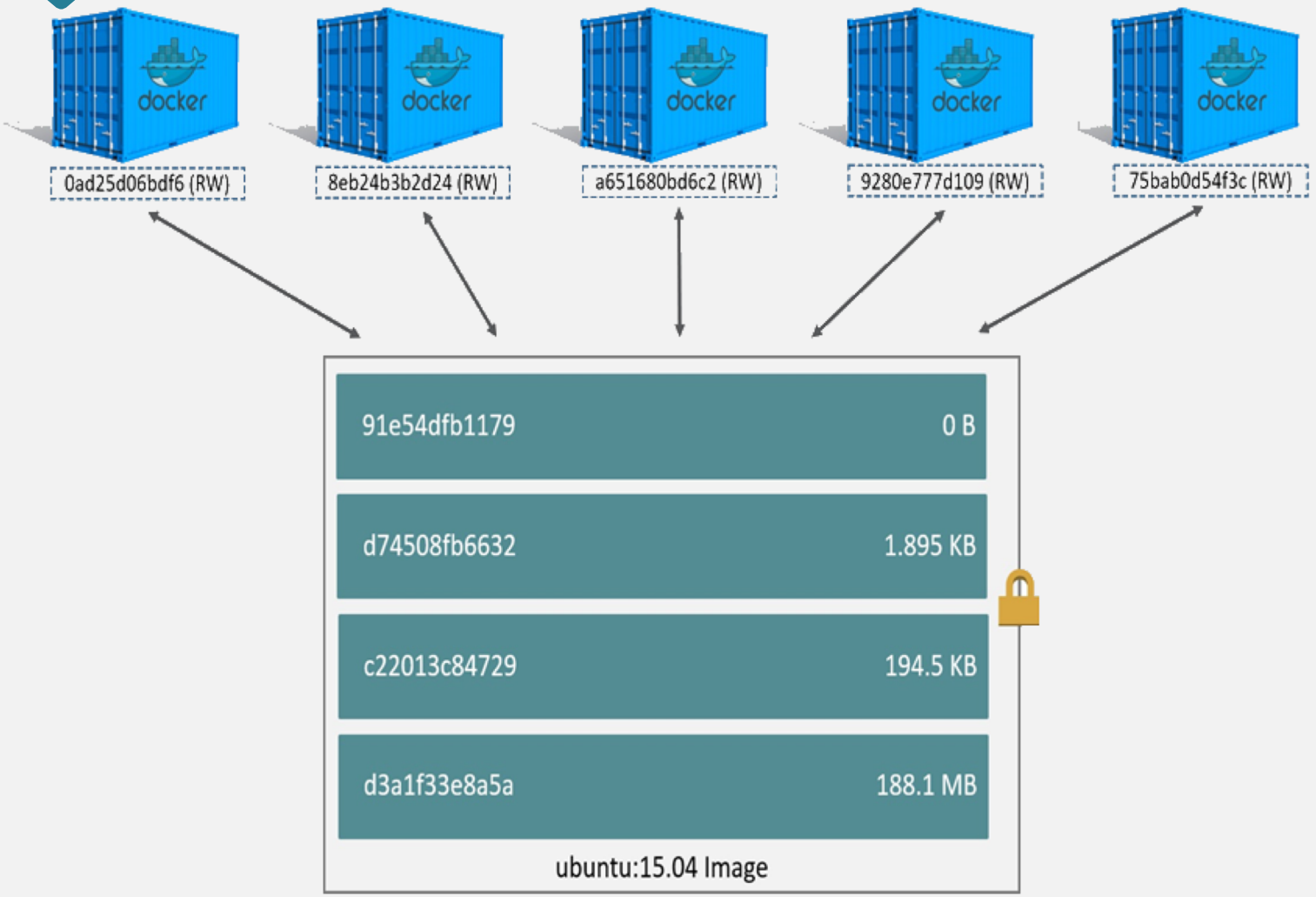
Docker images



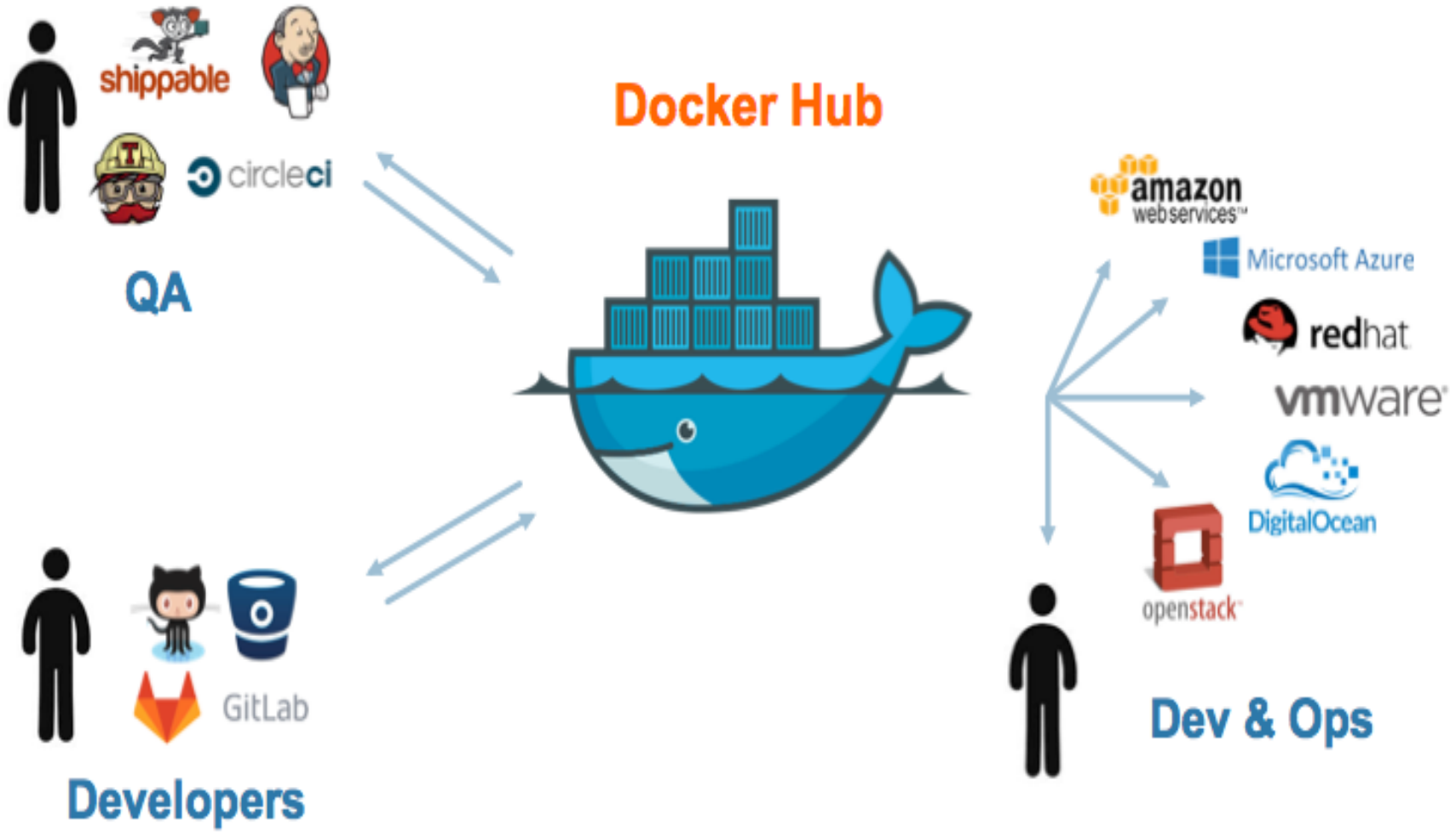
Docker images

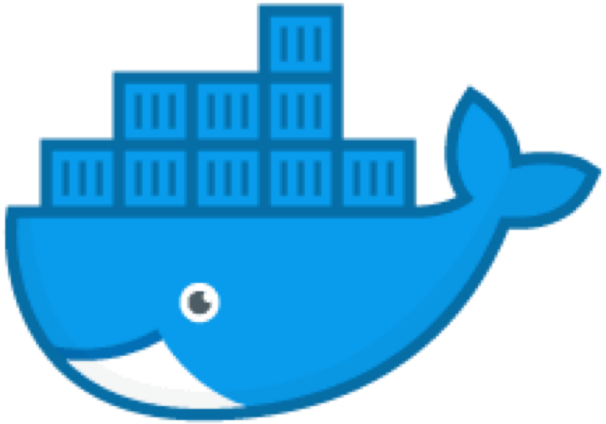


Docker images



Docker Containers





docker

Commands

Docker Commands (partial)

- docker **run / exec**
- docker **ps –a**
- docker **images**
- docker **tag**
- docker **pull**
- docker **push**
- docker **attach**
- docker **rm / rmi**
- docker **commit**
- docker **inspect**
- docker **stats**
- docker **logs**
- docker **cp**
- docker **build**

Docker Commands (partial)

- **attach** Attach to a running container
- **ps** List all running containers
- **ps** –a List all containers (incl. stopped)
- **start** Start a stopped container
- **stop** Stop a running container
- **pause** Pause all processes within a container
- **rm** Delete a container
- **top** Display processes of a container
- **commit** Create an image from a container
- **images** List images
- **build** Build container from Dockerfile

Docker Commands

- **Old way:**
  - **docker <command>**
  - docker run ...
- **New way :**
  - **docker <management> <command>**
  - docker **container** run …..
    - Example: docker container run –p 80:80 nginx

Docker Commands (run)

- **docker run** is the most basic command

  docker run <options> <image>

Ex:

docker run --user 0 -d -p 5901:5901 -e password=1 opencv

Docker Commands (run - flags)

- **foreground (default)**

  Run container in the foreground

  -a=[] Attach to `STDIN`, `STDOUT` and/or `STDERR`

  -t Allocate a pseudo-tty

  -i Keep STDIN open even if not attached

  Ex: docker run -a stdin -a stdout -i -t ubuntu /bin/bash

- **detached (-d/ --detach / -d=true)**

  Run container in the background

  Ex: docker run -i -t -d ubuntu /bin/bash

Docker Commands (run - flags)

--name

If you do not assign a container name with the **--name** option, then the daemon generates a **random** string name for you.

Defining a name can be a handy way to add meaning to a container. If you specify a name, you can use it when referencing the container within a Docker network

Docker generate random names based upon this code:
https://github.com/moby/moby/blob/master/pkg/namesgenerator/names-generator.go

Docker Commands (run - flags)

--add-host

Your container will store in **/etc/hosts** the **hostname** of the container itself as well as localhost.

The ***--add-host*** flag can be used to add **additional** lines to /etc/hosts.

docker run -it --add-host db-static:86.75.30.9 ubuntu cat /etc/hosts

cat /etc/hosts

```
127.0.0.1       localhost
::1             localhost ip6-localhost ip6-loopback
86.75.30.9      db-static
```

Docker Commands

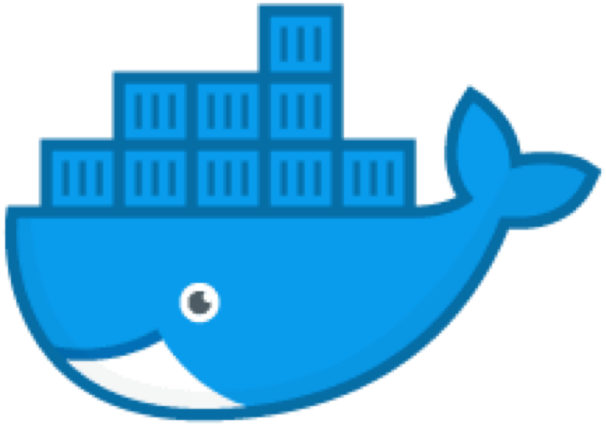
Copy local file to Docker containers:

docker cp <local file> <container id>:<container path>

Docker Commands

stop / remove all of Docker containers:

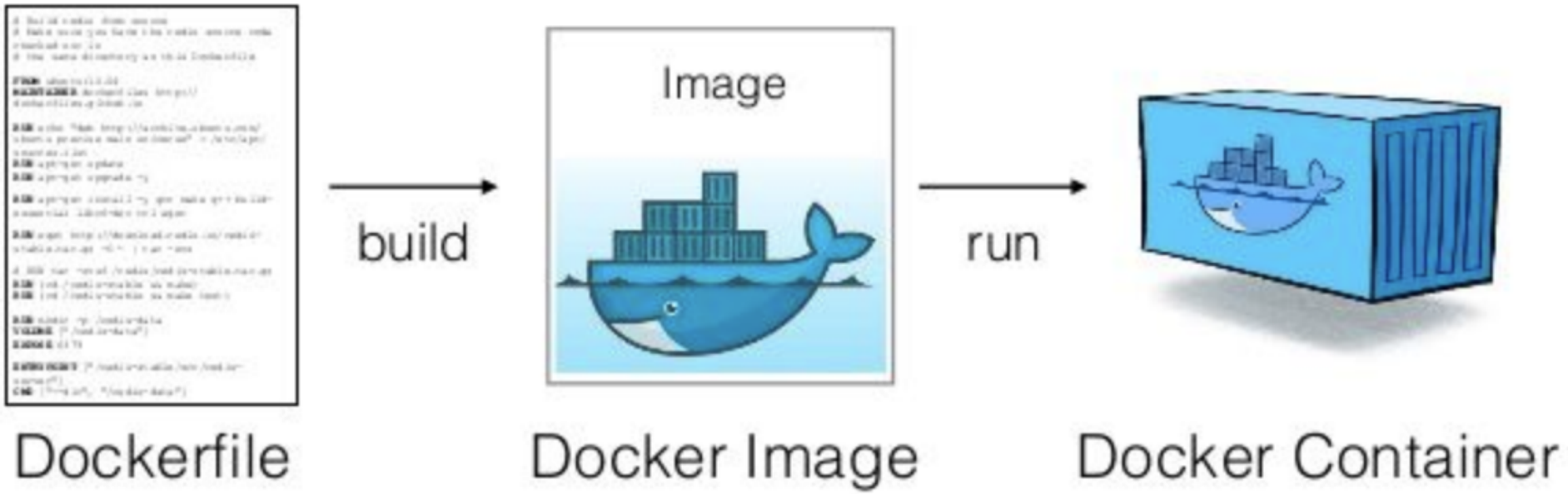
```
docker stop $(docker ps -a -q)

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



docker

File

Docker File



Docker File

- A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image (CLI).

- **docker build** – the build command builds up a the container by reading and executing the content of the Dockerfile

Docker File

```
# Multiple images example
#
# VERSION               0.1

FROM ubuntu
RUN echo foo > bar
# Will output something like ===> 907ad6c2736f

FROM ubuntu
RUN echo moo > oink
# Will output something like ===> 695d7793cbe4

# You'll now have two images, 907ad6c2736f with /bar, and 695d7793cbe4 with
# /oink.
```

FROM

The **FROM** instruction sets the Base Image for subsequent instructions

Docker File

RUN

The **RUN** instruction will execute any commands in a **new layer** on top of the current image and commit the results.

The resulting committed image will be used for the next step in the Dockerfile.

CMD

The main purpose of a CMD is to provide defaults for an executing container.

These defaults can include an executable, or they can omit the executable, in which case you must specify an **ENTRYPOINT** instruction as well.

EXPOSE

The EXPOSE instruction informs Docker that the container **listens** on the specified network **ports** at runtime.

ENV

The ENV instruction sets the environment variable **<key>** to the value **<value>**

ADD / COPY

The Add/ COPY instruction **copies** new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.

ENTRYPOINT

An ENTRYPOINT allows you to configure a container that will run as an executable.

Only the last ENTRYPOINT instruction in the Dockerfile will have an effect.

VOLUME

The VOLUME instruction **creates a mount point** with the specified name and marks it as holding externally mounted volumes **from native host or other** containers.

WORKDIR

The **WORKDIR** instruction sets the working directory for any **RUN**, **CMD**, **ENTRYPOINT**, **COPY** and **ADD** instructions that follow it in the Dockerfile.

If the WORKDIR doesn't exist, it will be created even if it's not used in any subsequent Dockerfile instruction.

ARG

The ARG instruction defines a variable that users can pass at build-time to the builder with the docker build command using the

--build-arg <varname>=<value> flag

SHELL

instruction allows the default shell used for the shell form of commands to be overridden.

The default shell on Linux is *["/bin/sh", "-c"]*,
and on Windows is *["cmd", "/S", "/C"]*.

The SHELL instruction must be written in JSON form in a Dockerfile.

The SHELL instruction is particularly useful on Windows where there are two commonly used and quite different native shells: **cmd** and **powershell**, as well as alternate shells available including **sh**.

Docker File

LABEL

```
"Labels": {
    "com.example.vendor": "ACME Incorporated"
    "com.example.label-with-value": "foo",
    "version": "1.0",
    "description": "This text illustrates that label-values can span multiple lines.",
    "multi.label1": "value1",
    "multi.label2": "value2",
    "other": "value3"
},
```

The LABEL instruction adds metadata to an image.

A LABEL is a key-value pair.

To include spaces within a LABEL value, use quotes and backslashes as you would in command-line parsing

To **view** an image's labels, use the **docker inspect** command.

ONBUILD

The ONBUILD instruction **adds to the image a trigger instruction to be executed at a later time**, when the image is used as the base for another build.

The trigger will be executed in the context of the downstream build, as if it had been inserted immediately

STOPSIGNAL

The STOPSIGNAL instruction sets the system call signal that **will be sent to the container to exit**.

This signal can be a valid unsigned number that matches a position in the **kernel's syscall** table, for instance 9, or a signal name in the format SIGNAME

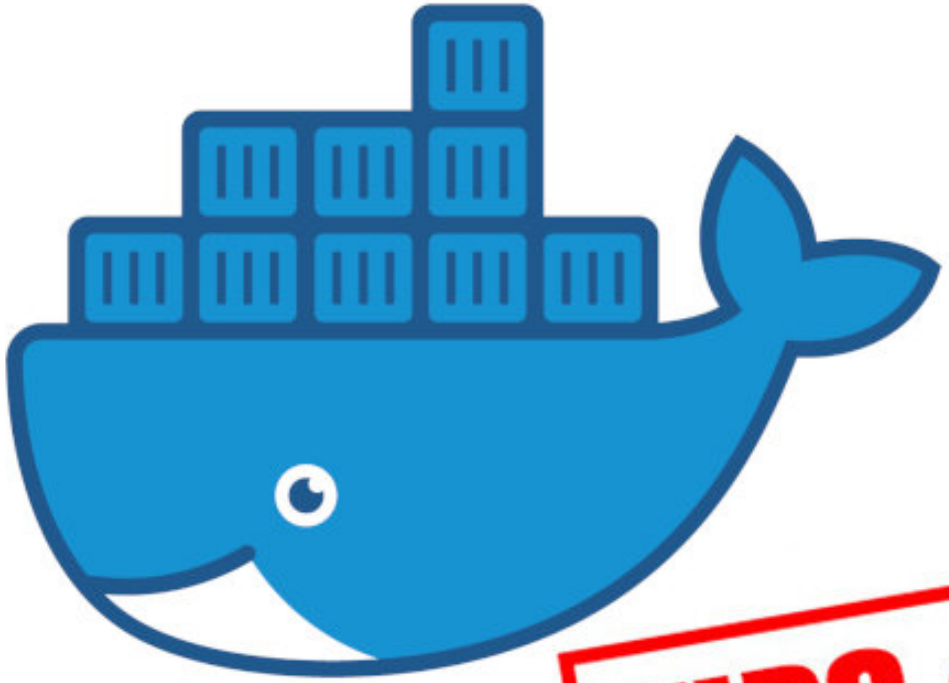
HEALTHCHECK

HEALTHCHECK instruction tells Docker how to **test a container** to check that it is still working.

This can detect cases such as a web server that is stuck in an infinite loop and unable to handle new connections, even though the server process is still running.

The HEALTHCHECK instruction has two forms:

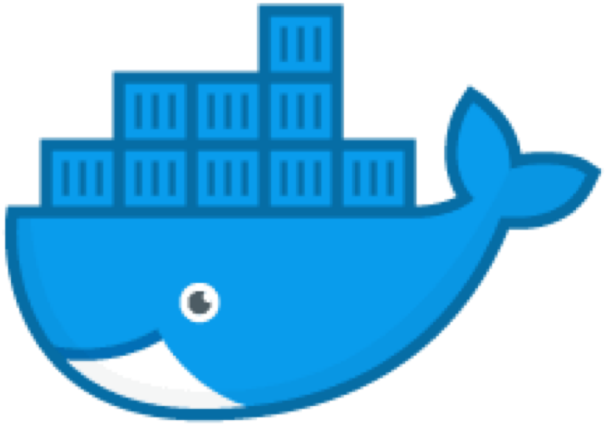
- **HEALTHCHECK [OPTIONS] CMD command**
  (check container health by running a command inside the container)
- **HEALTHCHECK NONE**
  (disable any healthcheck inherited from the base image



TIPS & TRICKS

Docker File Tips

- Order matter for caching
- Avoid COPY if possible
- Reduce the number of RUN commands (bundle whenever you can)
- Reduce image size (faster deploy, reuse)
- Remove unnecessary dependencies
- Delete package manager cache (apt, pip etc)
- Use official images where possible and look for minimal flavors
- Re build on the same env
- Fetch dependencies in its own Run command



docker

Tools overview

Kitematics

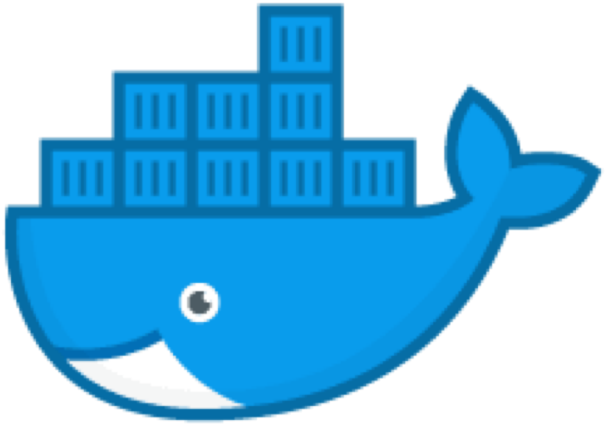
- Kitematic is an **open source project** built to simplify and streamline using Docker on a Mac or Windows PC.
- Kitematic automates the Docker installation and setup process and provides an intuitive graphical user interface (GUI) for running Docker containers.
- Kitematic integrates with Docker Machine to provision a VirtualBox VM and install the Docker Engine locally on your machine.

Minikube

- Minikube is a tool that makes it easy to run Kubernetes locally.
- Minikube runs a single-node Kubernetes cluster inside a Virtual Machine (VM) on your laptop for users looking to try out Kubernetes or develop with it day-to-day.

Build Kit

- Docker Engine 18.09 also includes the option to leverage BuildKit. This is a new Build architecture that improves performance, storage management, and extensibility while also adding some great new features:
  - **Performance improvements:** BuildKit includes a re-designed concurrency and caching model that makes it much faster, more precise and portable.
  - 2x to 10x faster builds.
  - This new implementation also supports these new operational models:
    - Parallel build stages
    - Skip unused stages and unused context files
    - Incremental context transfer between builds



docker

Practice


THE END