Container e virtualizzazione

Alessandro Costantini

Installazione e amministrazione Kubernetes

9, 10, 11 Dicembre 2024

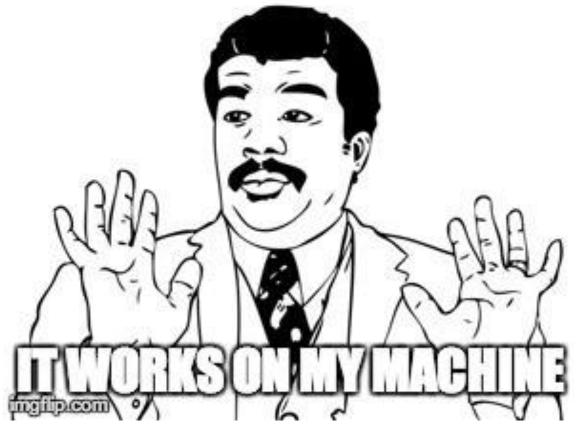
Overview

- Containers
  - Containers vs VMs
  - Working with containers
  - Management
  - Best practices and security

Containers

Background

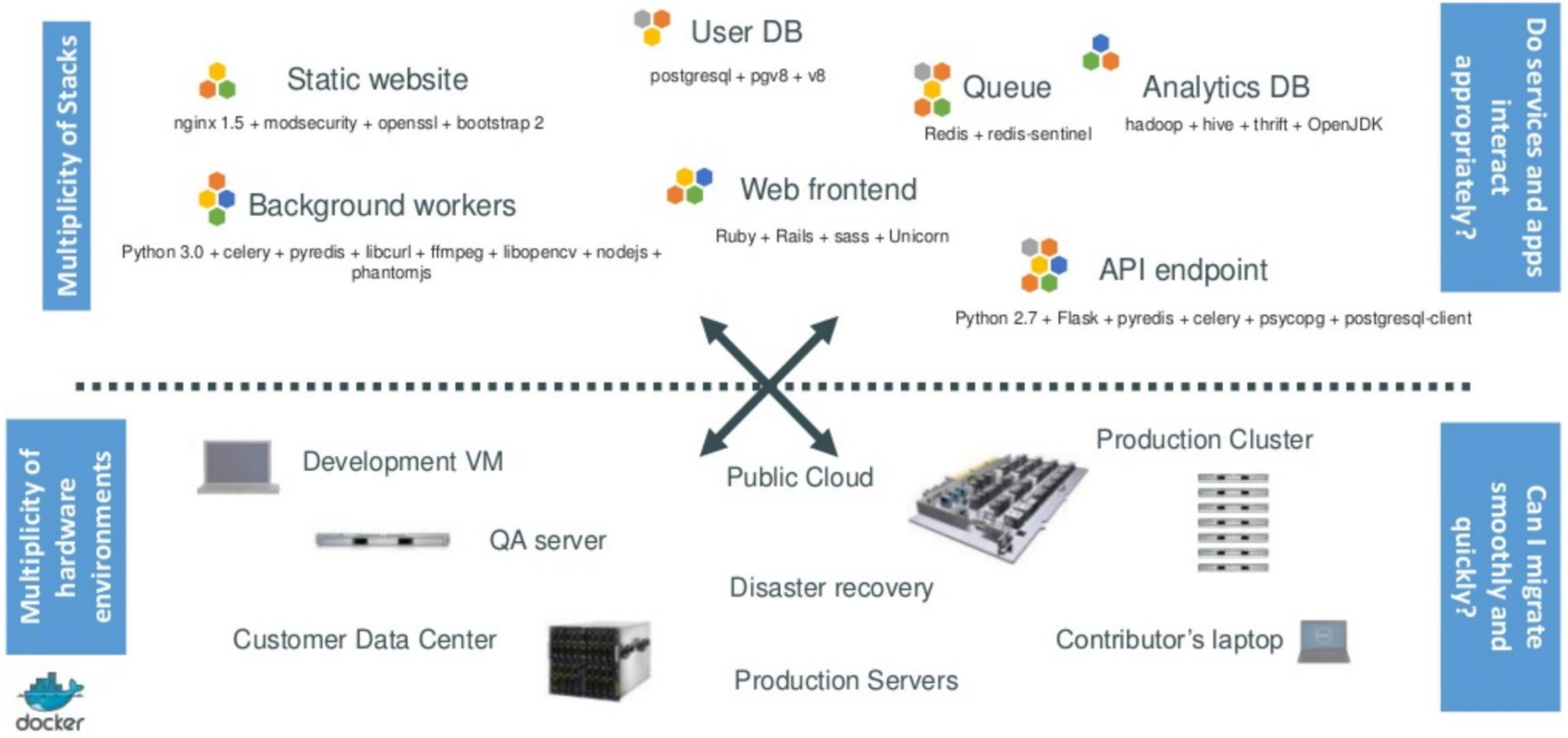
- Building a web service on a Ubuntu machine
- Code works fine on local machine
- Moved to a remote server …. does not work



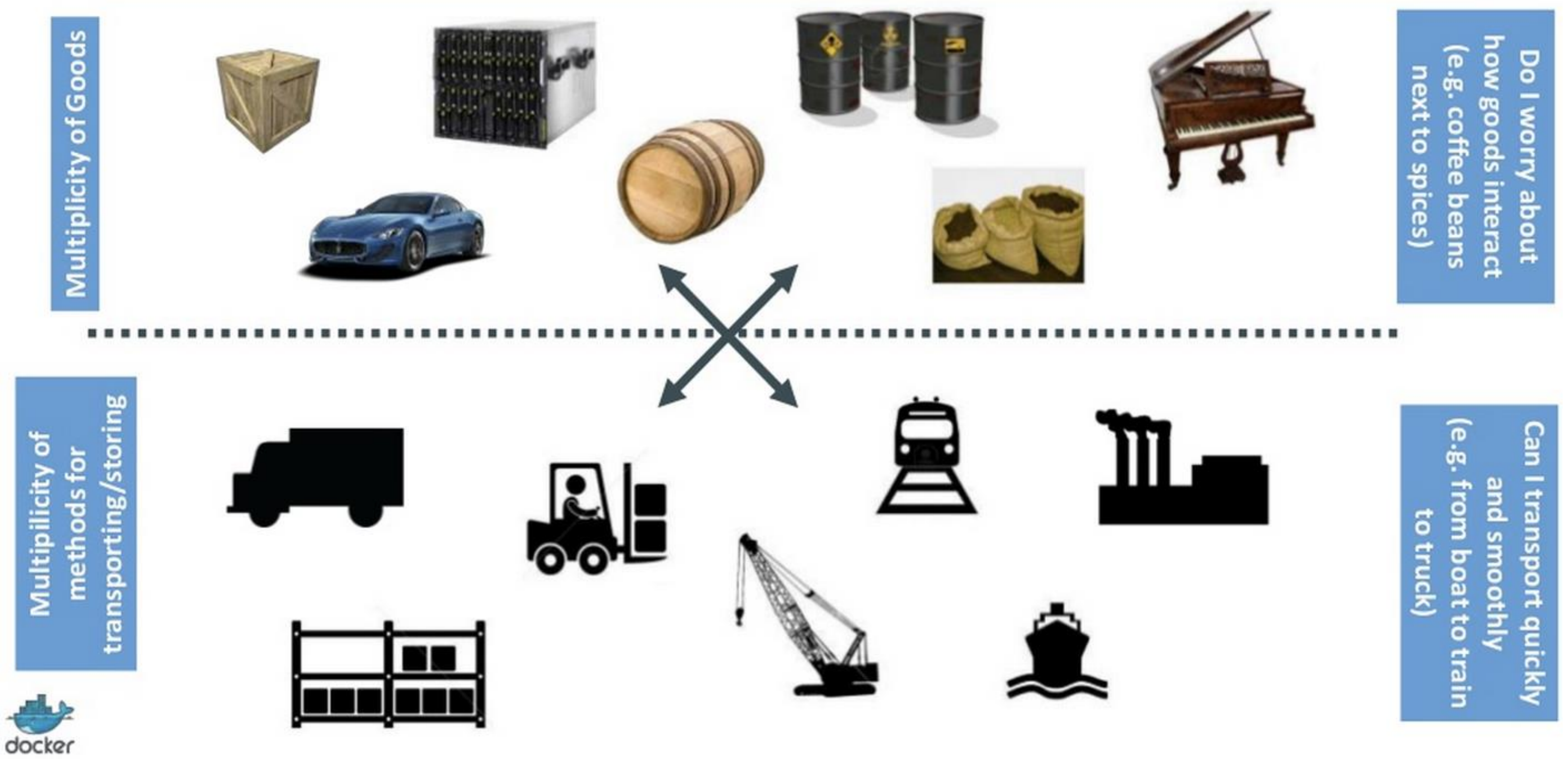
- Reasons:
  - Different OS => missing libraries or files for the runtime
  - Incompatible version of software (python, java)

It is essential to find a solution to these problems

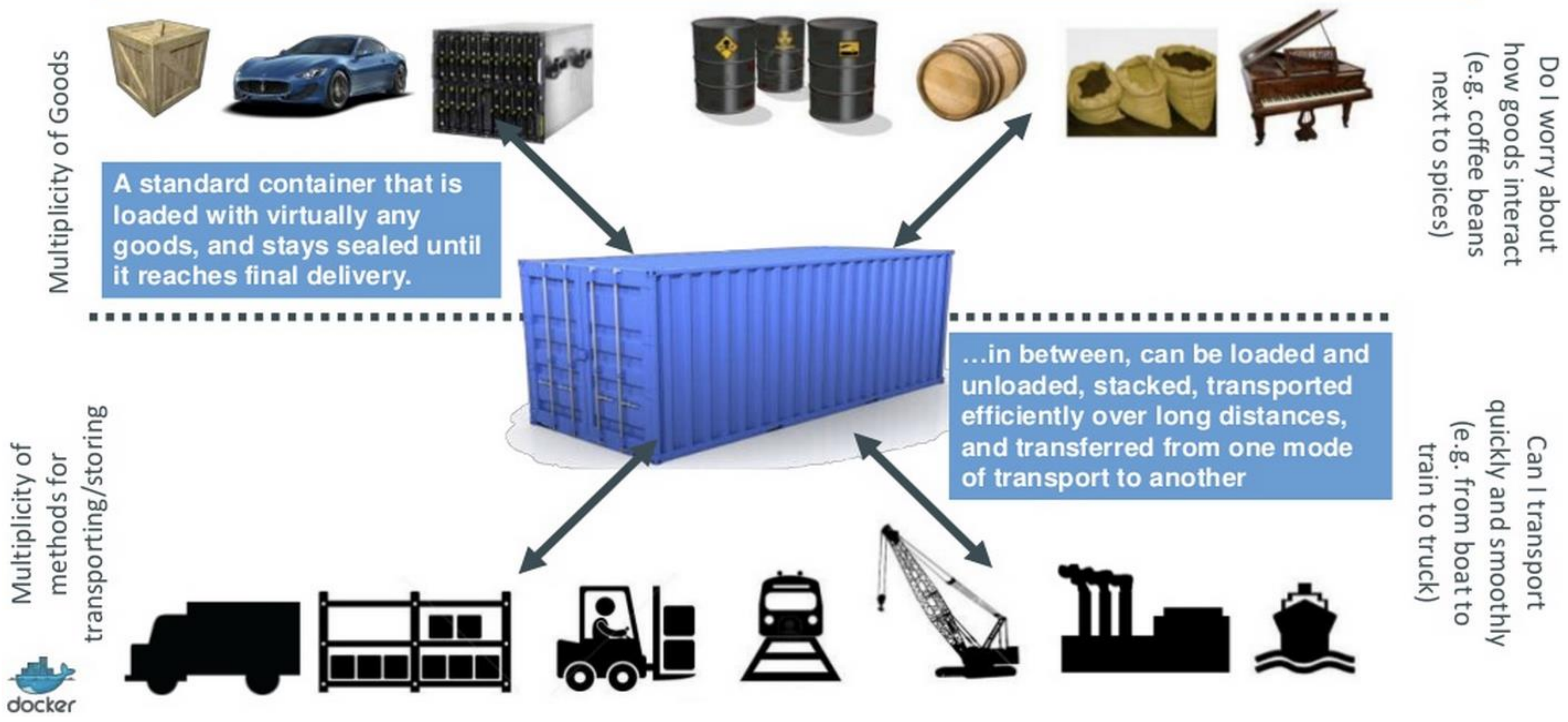
The Challenge


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
Public Cloud
Production Cluster
Disaster recovery
Customer Data Center
Production Servers
Contributor's laptop
Can I migrate smoothly and quickly?
docker

Cargo Transport Pre-1960


Multiplicity of Goods
Do I worry about how goods interact (e.g. coffee beans next to spices)
Multiplicity of methods for transporting/storing
Can I transport quickly and smoothly (e.g. from boat to train to truck)
docker

Solution: Intermodal Shipping Container


Multiplicity of Goods
A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.
Do I worry about how goods interact (e.g. coffee beans next to spices)
Multiplicity of methods for transporting/storing
...in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another
Can I transport quickly and smoothly (e.g. from boat to train to truck)
docker

Intermodal Shipping Container Ecosystem

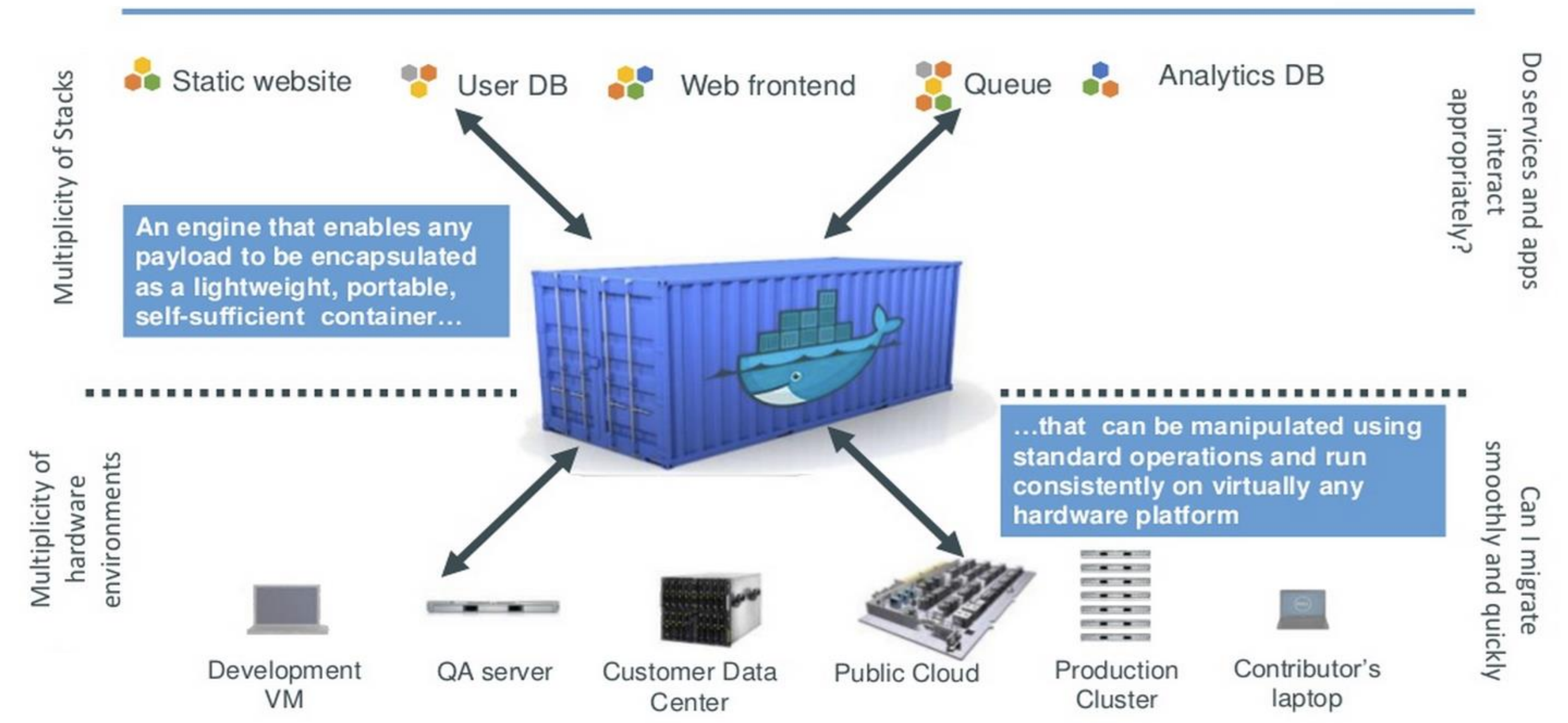
- 90% of all cargo now shipped in a standard container
- Order of magnitude reduction in cost and time to load and unload ships
- Massive reduction in losses due to theft or damage
- Huge reduction in freight cost as percent of final goods (from >25% to <3%)

→ massive globalizations

- 5000 ships deliver 200M containers per year

OK, not everything always goes as planned…

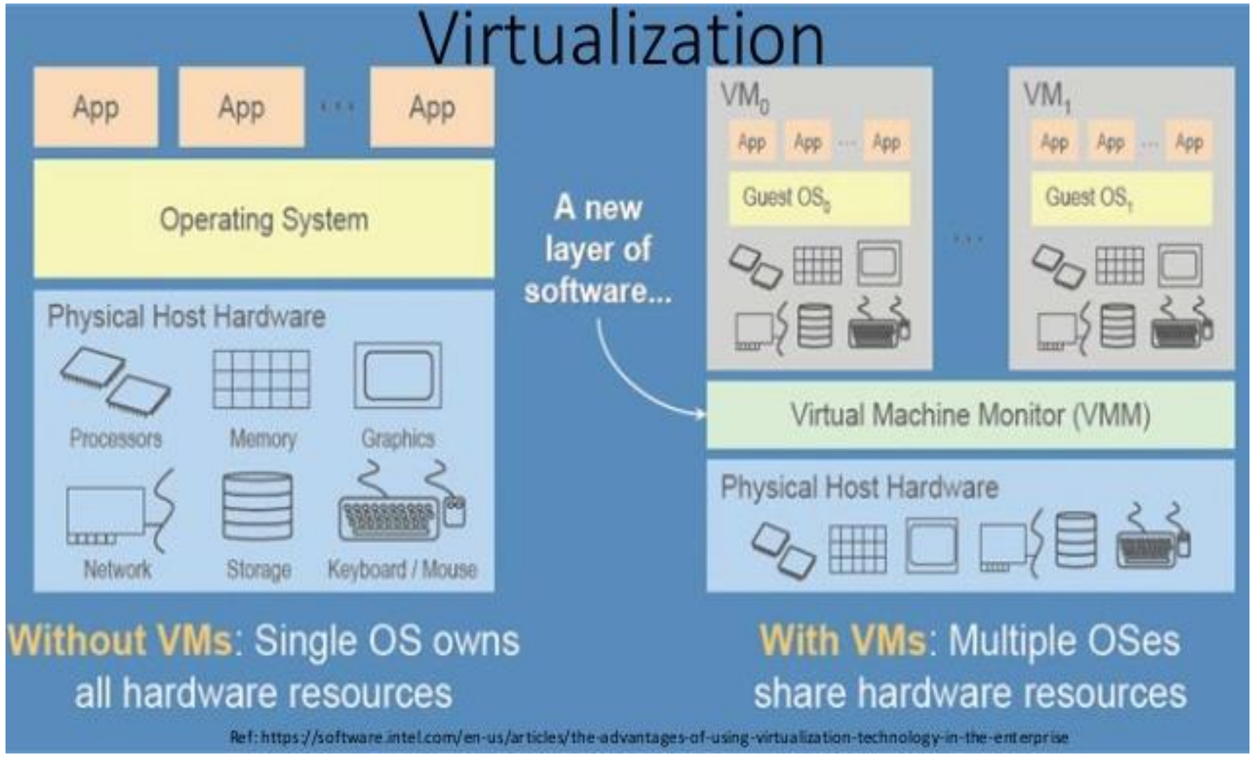
Analogue solution: virtual containers


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
Development VM
QA server
Customer Data Center
Public Cloud
Production Cluster
Contributor's laptop
Can I migrate smoothly and quickly

Virtualization

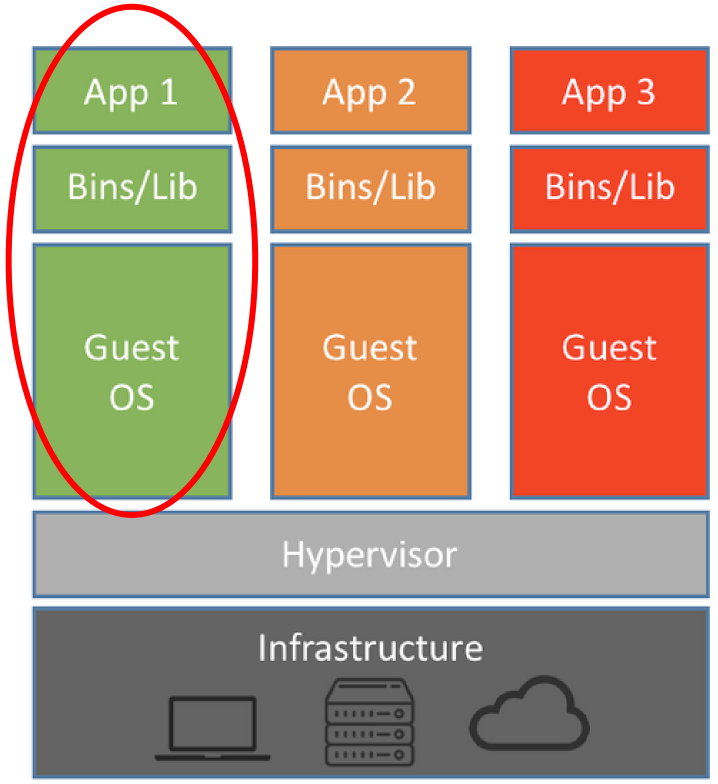
- What is “Virtualization” in general?

- It is **the creation of a virtual version of *something***: an Operating System, a storage device, a network resource: pretty much almost anything can be made virtual.
    - This is done through an abstraction, that hides and simplifies the details underneath.


Virtualization
App
App
App
Operating System
Physical Host Hardware
Processors
Memory
Graphics
Network
Storage
Keyboard / Mouse
A new layer of software...
VM$_0$
VM$_1$
Guest OS$_0$
Guest OS$_1$
Virtual Machine Monitor (VMM)
Physical Host Hardware
Without VMs: Single OS owns all hardware resources
With VMs: Multiple OSes share hardware resources
Ref: https://software.intel.com/en-us/articles/the-advantages-of-using-virtualization-technology-in-the-enterprise

What are VMs?

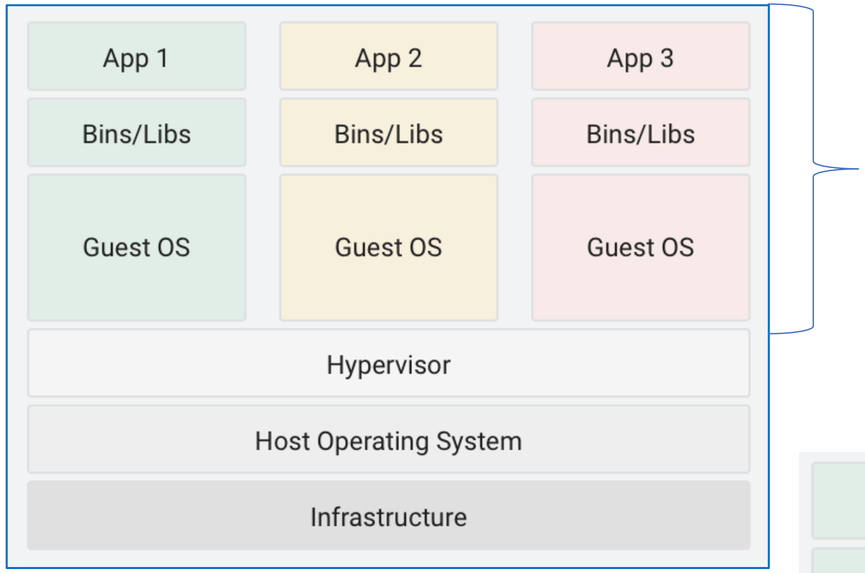
- As server processing **power and capacity increased**, bare metal applications **weren't able to exploit** the new abundance in resources.
  - Thus, **VMs were born**, designed by **running software on top of physical servers** to emulate a particular hardware system.
  - A **hypervisor (VMM)** - > is software, firmware, or hardware that creates and runs VMs.
    - sits between the hardware and the virtual machine and is necessary to virtualize the server.
- Within each VM runs a **unique guest** OS.
  - VMs with **different operating systems** can run on the **same physical server**



Machine Virtualization

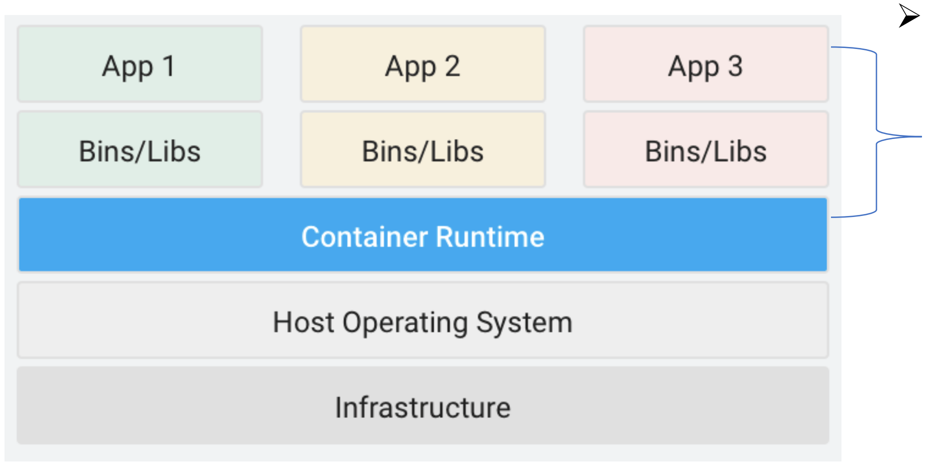
Going beyond .... Virtual Machines

Virtual Machines (VMs) carry quite some overhead with them



Virtual Machine

- Each virtualized application includes not only the **application** — which may be only 10s of MB — and the necessary **binaries** and **libraries**, but also an **entire guest operating system** — which may weigh 10s of GB.

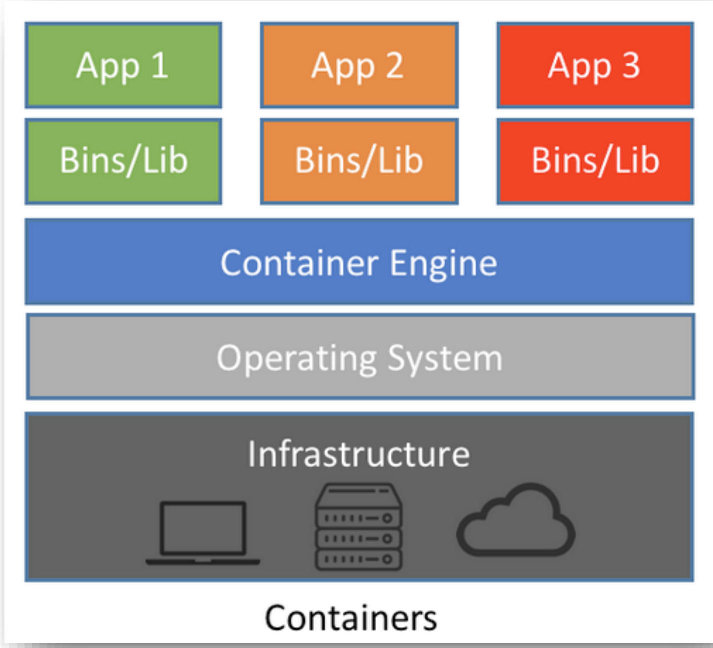


Container

- comprises just the **application** and its **dependencies**. It runs as an **isolated process** in userspace on the host operating system, **sharing the kernel** with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is **much more portable and efficient**.

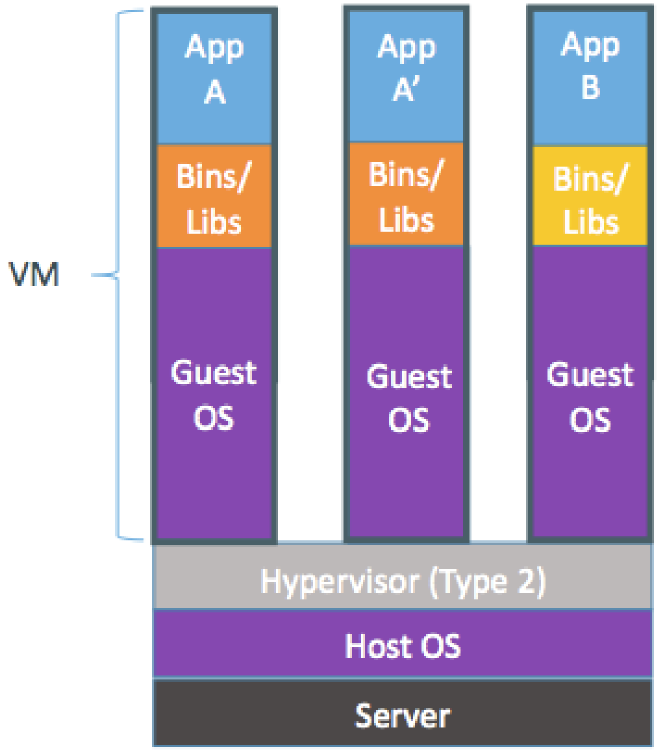
What are containers?

- **Operating system (OS) virtualization** has grown in popularity over the last decade to enable software to run predictably and well when **moved from one server environment to another**.
  - containers provide a way to run these isolated systems on a single server or host OS.
  - containers sit on top of a physical server and its host OS
    - shares the host OS kernel, the binaries and libraries
    - Shared components are read-only =>"light"
    - reduce management overhead as they share a common OS operating system
- Differences
  - **Containers** provide a way to **virtualize an OS** so that multiple workloads can run on a single OS instance
  - VMs, the hardware is being virtualized to run multiple OS instances



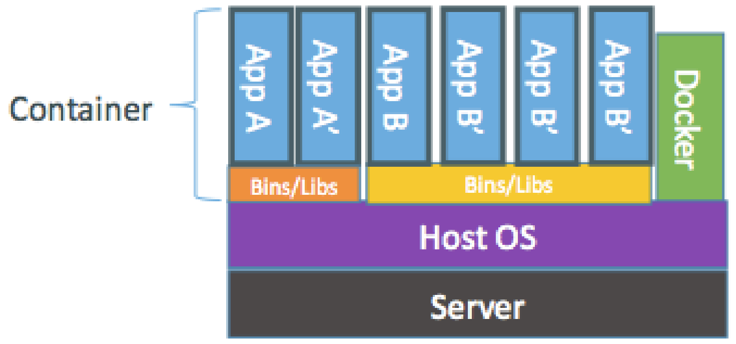
Containers as «lightweight VMs»

A container is a ***standard unit of software*** *that packages up* ***code and all its dependencies,*** *so the application runs quickly and reliably from one computing environment to another*



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

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



Source: http://goo.gl/4jh8cX

Docker

“Lightweight”, in practice

- **Containers require less resources**: they start faster and run faster than VMs, and you can fit many more containers in a given hardware than VMs.
- **Very important**: they provide enormous simplifications to software development and deployment processes, because they allow to simply encapsulate applications in a controlled and extensible way.
- Provide a uniformed wrapper around a software package:
    - ➢ *«Build, Ship and Run Any App, Anywhere»*

“Similar to shipping containers: The container is always the same, regardless of the contents and thus fits on all trucks, cranes, ships, …”

Build

Develop an app using Docker containers with any language and any toolchain.

Ship

Ship the “Dockerized” app and dependencies anywhere - to QA, teammates, or the cloud - without breaking anything.

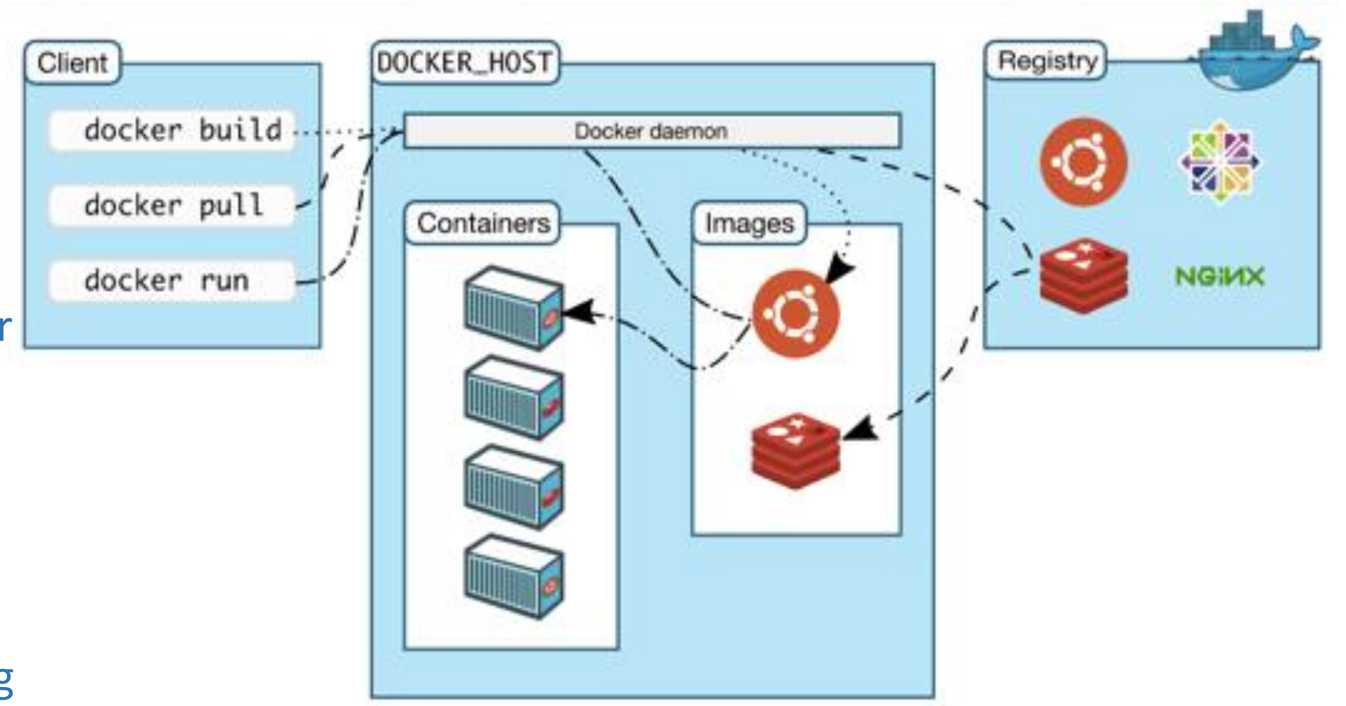
Run

Scale to 1000s of nodes, move between data centers and clouds, update with zero downtime and more.

Docker

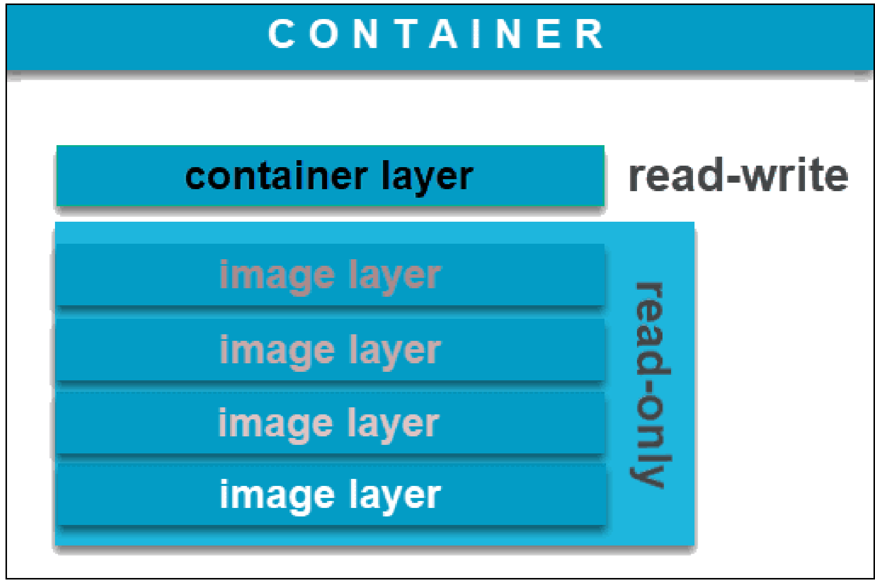
- Docker is an open-source platform that automates the development and deployment of applications inside portable and self-sufficient software “containers”.
  - Like virtualenv for Python

- Main components:
  - ***Docker Engine***
    - portable **runtime** and packaging system that gives **standardized environments** for the development and flexibility for workload deployment so that it is not restricted by infrastructure technology.
  - ***Docker Hub***
    - Docker Hub is a cloud solution for sharing apps and automating workflows.



Containers vs. Images

- "A ***container image*** *is a lightweight*, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings."
  - ➢A **Docker image** is an **immutable (unchangeable)** file that contains the source code, libraries, dependencies, tools, and other files needed for an application to run.
    - ➢ They are **templates, read-only, cannot run**
    - ➢ **Container is a running image**



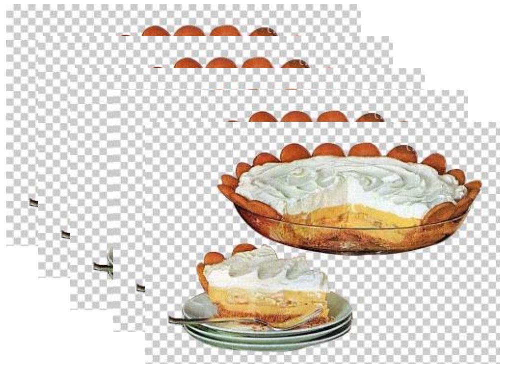
Docker container

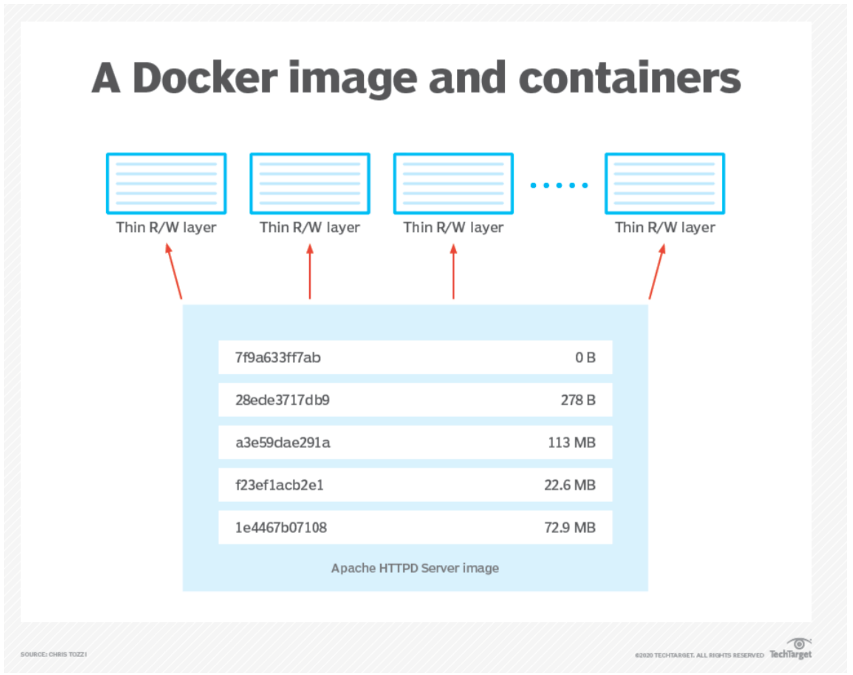
- **From** a container **image**, you can **start a container** based on it. Docker containers are the way to execute that package of instructions in a runtime environment
- Containers run until they fail and crash, stopped.
    - does not change the image on which it is based
- Docker image = **recipe** for a cake
- and a container = **cake** you baked from it.

Simple Cake Re:ipe

225g (8 oz) self-raising flour.
225g (8 oz) soft butter (i.e. room temperature).
225g (8 oz) caster sugar.
4 eggs.
1 teaspoon baking powder.

Mix the ingredients well in a large bowl using an electric whisk.
Halve the mixture and pour into 2 non-stick 18cm (7 inch) cake tins.
Cook till golden brown (15-25 minutes) in a preheated oven at 180 degrees C (gas mark 4).
Cool on a wire rack before serving, add jam between the two halves and optionally top with butter cream.

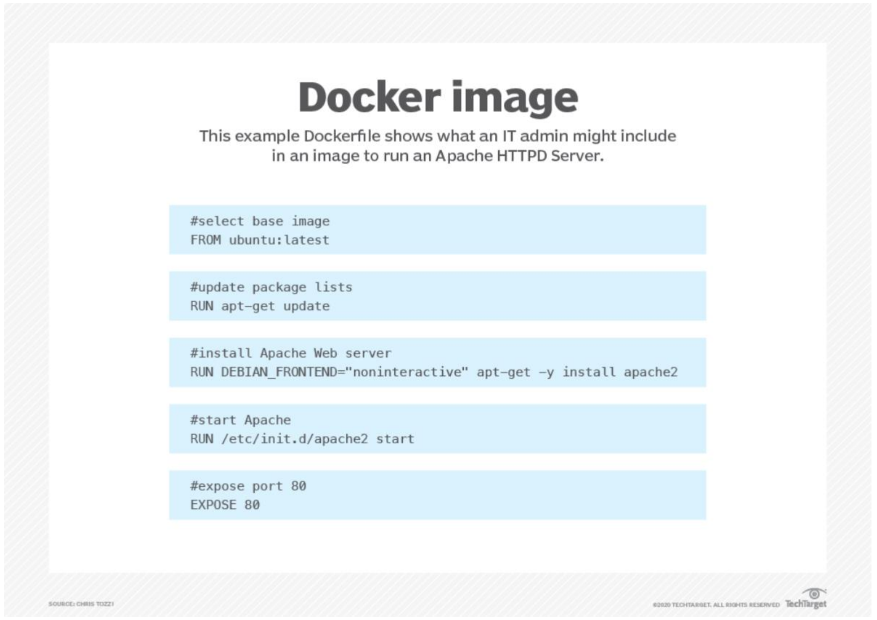




Docker Image

- It is a **set of instructions** that defines what should run inside a container.

- A Docker image typically specifies:
  - Which **external image to use as the basis** for the container, unless the container image is written from scratch;
  - **Commands** to run when the container starts;
  - How to **set up the file system** within the container; and
  - **Additional instructions**, such as which ports to open on the container, and how to import data from the host system.



Docker image

This example Dockerfile shows what an IT admin might include in an image to run an Apache HTTPD Server.

```
#select base image
FROM ubuntu:latest

#update package lists
RUN apt-get update

#install Apache Web server
RUN DEBIAN_FRONTEND="noninteractive" apt-get -y install apache2

#start Apache
RUN /etc/init.d/apache2 start

#expose port 80
EXPOSE 80
```

SOURCE: CHRIS TOZZI

©2020 TECHTARGET. ALL RIGHTS RESERVED TechTarget

Container terminology

- **Container:**
  - In Linux, containers are an ***operating system virtualization technology*** used to package applications and their dependencies and run them in isolated environments.
- **Container Image:**
  - Container images are ***static files*** that define the filesystem and behavior of specific container configurations. Container images are used as a template to create containers.
- **Docker:**
  - Docker was the first technology to successfully popularize the idea Linux containers.
  - Among others, Docker's ecosystem of tools includes ***docker***, a container runtime with extensive container and image management features, ***docker-compose***, a system for defining and running multi-container applications, and ***Docker Hub***, a container image registry.
- **Linux cgroups:**
  - or control groups, are a kernel feature that **bundles processes together** and **determines their access to resources**. Containers in Linux are implemented using cgroups in order to manage resources and separate processes.
- **Linux namespaces:**
  - a kernel feature designed to **limit the visibility** for a process or cgroup to the rest of the system. Containers in Linux use namespaces to help isolate the workloads and their resources from other processes running on the system.
- **LXC:**
  - LXC is a form of Linux containerization that predates Docker and many other technologies while relying on many of the same kernel technologies. Compared to Docker, LXC usually virtualizes an entire operating system rather than just the processes required to run an application, which can seem more similar to a virtual machine.
- **Virtual Machines:**
  - Virtual machines, or VMs, are a hardware virtualization technology that emulates a full computer. A full operating system is installed within the virtual machine to manage the internal components and access the computing resources of the virtual machine.

Docker for different OS

Supported OS:

- https://docs.docker.com/engine/install/
  - **Windows**: https://docs.docker.com/desktop/install/windows-install/
  - **Linux**:
    - for RedHat see https://docs.docker.com/engine/install/centos/
  - **MacOS**: https://docs.docker.com/desktop/install/mac-install/

Working with images

Check hands-on environment

- To avoid specifying `sudo` before each docker command, the users has been added to the docker Unix group. Check it:

```
$ id
(where do you see it?)

$ docker info
[…]
Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
[…]
```

Search, pull, run

- **Try these commands on your environment:**
  - **Search** for a container image at Docker Hub:
    - `$ docker search ubuntu` (or e.g. `docker search rhel` – what would this do?)
  - Fetch (**pull**) a Docker image (in this case, an Ubuntu container image):
    - `$ docker pull ubuntu`
  - List images
    - `$ docker images`
  - Execute (**run**) a docker container:
    - Run the "echo" command inside a container and then exit:

      -
        ```
        $ docker run ubuntu echo "hello from the container"
        hello from the container
        ```
    - Run a container in interactive mode:
      - `$ docker run -it ubuntu /bin/bash`

How efficient is docker?

```
$ docker images

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              7698f282e524        2 weeks ago         72.9MB
```

=> the latest Ubuntu image takes about 70MB of disk space *as a container*. If you had just to download a full Ubuntu (server) distribution, it would be more in the range of 900MB.

```
$ time docker run ubuntu echo “hello from the container”
hello from the container

real    0m1.384s
user    0m0.069s
sys     0m0.106s
```

=> The total time it takes on this system (not a really powerful one) to start a container, execute a command inside it and exit from the container is about half a second.

How to extend a docker container (1)

- Suppose you need a command inside a container, but it is not installed in the image you pulled from Docker Hub. For example, you would like to use the `ping` command but by default it's not available:

  -
    ```
    $ docker run ubuntu ping www.google.com
    docker: Error response from daemon: OCI runtime create failed:
    container_linux.go:345: starting container process caused "exec: \"ping\":
    executable file not found in $PATH": unknown.
    ```
- We can install it ourselves; it is in the package `iputils-ping`:

  -
    ```
    $ docker run ubuntu /bin/bash -c "apt update; apt -y install inetutils-ping"
    ```
- But it still doesn't work!?

  -
    ```
    $ docker run ubuntu ping www.google.com
    docker: Error response from daemon: OCI runtime create failed:
    container_linux.go:345: starting container process caused "exec: \"ping\":
    executable file not found in $PATH": unknown.
    ```

How to extend a docker container (2)

- Whenever you issue a `docker run <container>` command, a **new container** is started, based on **the original container image**.
  - Check it yourself with `$ docker ps -a` command.
- If you modify a container and then want to reuse it (which is often the case!), **you need to save the container, creating a new image**.
- So, install what you need to install (e.g. the `iputils-ping` package) , and then issue a commit command like

```
$ docker commit xxxx ubuntu_with_ping
```

- This **locally commits** a container, creating an image with a proper name (`ubuntu_with_ping`). Take `xxxx` from the container ID shown by the `docker ps -a` output.

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu_with_ping    latest              3e7a8818665f        11 minutes ago      97.2MB
ubuntu              latest              7698f282e524        7 days ago          69.9MB
```

Dockerfile

Container Layers

- Dockerfile
  - A series of instruction for building images
  - Each Dockerfile command creates a Layer
  - Only ADD, RUN and COPY influence the size of the image
- Container layers
  - From image to container

```
•$ cat Dockerfile
FROM ubuntu
ENV DEBIAN_FRONTEND=noninteractive
RUN apt update
RUN apt install -y inetutils-ping
```

This Dockerfile:

- Starts from the Ubuntu container
- Updates all installed packages
- Installs `inetutils-ping`

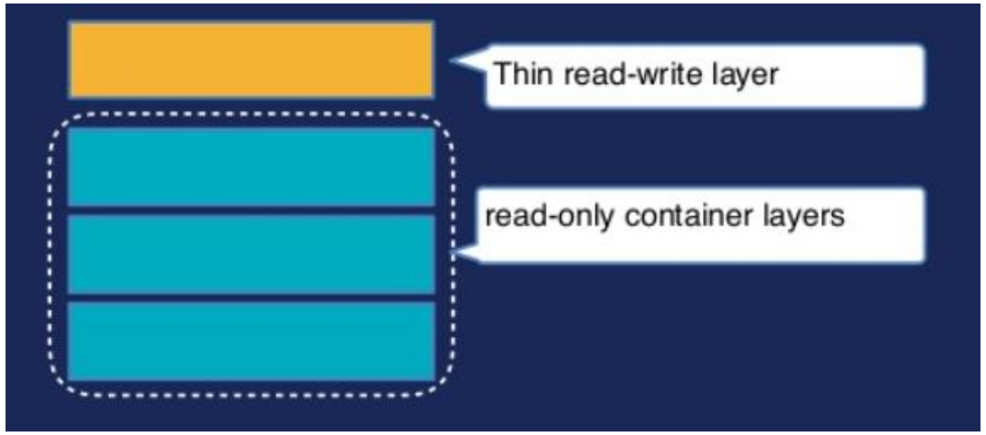


Image building process

```
$ docker build -t ubuntu_ping .
[+] Building 13.6s (7/7) FINISHED                                                               docker:default
 => [internal] load build definition from Dockerfile                                                      0.6s
 => => transferring dockerfile: 133B                                                                      0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest                                          0.0s
 => [internal] load .dockerignore                                                                         0.8s
 => => transferring context: 2B                                                                           0.0s
 => [1/3] FROM docker.io/library/ubuntu:latest                                                            0.0s
 => [2/3] RUN apt update                                                                                  5.6s
 => [3/3] RUN apt install -y inetutils-ping                                                               4.9s
 => exporting to image                                                                                    0.7s
 => => exporting layers                                                                                   0.6s
 => => writing image sha256:58aef102eafa16e65541bf7446aa8ac24c8edf61079f7999bf549ad9caf13d51              0.0s
 => => naming to docker.io/library/ubuntu_ping                                                            0.0s
```

Inspect image building

```
$ docker images
REPOSITORY                    TAG        IMAGE ID       CREATED          SIZE
ubuntu_ping                           latest     58aef102eafa   5 minutes ago   121MB
ubuntu                        latest     26b77e58432b   2 weeks ago      78.1MB
hello-world                   latest     d1165f221234   6 weeks ago      13.3kB

$ docker history 58aef102eafa
IMAGE          CREATED         CREATED BY                                      SIZE       COMMENT
58aef102eafa   3 minutes ago   RUN /bin/sh -c apt install -y inetutils-ping…   1.02MB     buildkit.dockerfile.v0
<missing>      3 minutes ago   RUN /bin/sh -c apt update # buildkit            42.1MB     buildkit.dockerfile.v0
<missing>      3 minutes ago   ENV DEBIAN_FRONTEND=noninteractive              0B         buildkit.dockerfile.v0
<missing>      2 weeks ago     /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      2 weeks ago     /bin/sh -c #(nop) ADD file:34dc4f3ab7a694ecd…   78.1MB
<missing>      2 weeks ago     /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      2 weeks ago     /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      2 weeks ago     /bin/sh -c #(nop)  ARG LAUNCHPAD_BUILD_ARCH     0B
<missing>      2 weeks ago     /bin/sh -c #(nop)  ARG RELEASE                  0B
```

Reduce Layers

- More layers mean a larger image
  - The larger the image, the longer that it takes to build, push and pull
- Smaller images mean faster builds and deploys

- How reduce layers
  - Use shared base images (where possible)
  - Limit the data written on the container layers
  - Chain RUN statemets

- Some links
  - https://dzone.com/articles/docker-layers-explained
  - https://stackoverflow.com/questions/32738262/whats-the-differences-between-layer-and-image-in-docker

Best practices and security

Some best practices for building containers

1. Put a **single application per container**. For example, do not run an application *and* a database used by the application in the same container.

2. **Do not confuse RUN with CMD.**
   - RUN runs a command and commits the result;
   - CMD does not execute anything at build time, it specifies the intended command for the image.

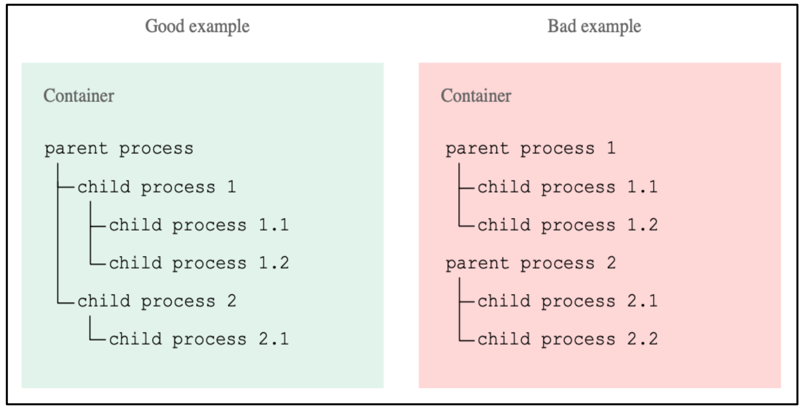
3. If in a Dockerfile you have **layers that change often, put them at the bottom of the Dockerfile**. This way, you speed up the process of building the image.

4. **Keep it small:** use the smallest base image possible, remove unnecessary tools, install only what is needed.

5. Properly **tag your images**, so that it is clear which version of a software it refers to.

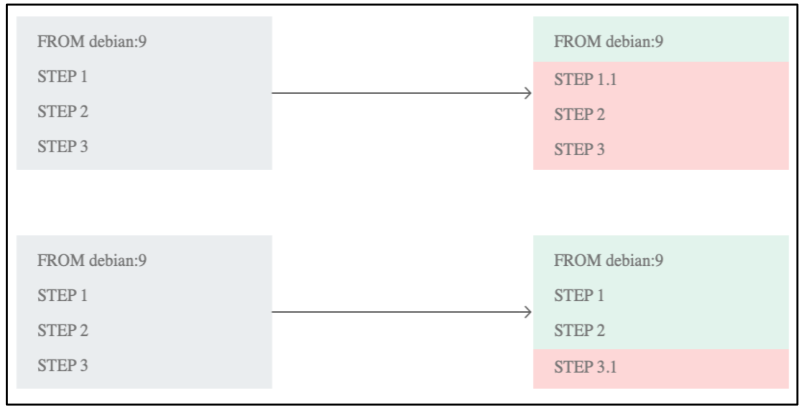
6. **Do you really want / can you use a public image?** Think about possible vulnerabilities, but also about potential license issues.

7. Passwords, certificates, encryption keys, etc. **Do not** embed them into the containers, and **do not** store them e.g. in GitHub repositories!



```
parent process
├child process 1
│ ├child process 1.1
│ └child process 1.2
└child process 2
  └child process 2.1
```

```
parent process 1
├child process 1.1
└child process 1.2
parent process 2
├child process 2.1
└child process 2.2
```



More (and more detailed) information available at https://bit.ly/2Zr6Hyq

Recap of Containers

- We covered the basic concepts about **Containers**, comparing them to Virtual Machines.
- We see how to execute some basic command like list docker images and extend them to create new containers.
- We then saw how to build an image via Dockerfiles.
- We then discussed about some Docker limitations, in particular with regard to security