CSCI 381/780
Cloud Computing

Virtualization 2: Container

Jun Li
Queens College

Are VMs Fit for (All) Today's (Cloud) Usages?

- Performance overhead of indirections (guest OS and hypervisor)
- Large memory footprint
- Slow startup time
- License and maintenance cost of guest OS
- Do we really need to virtualize hardware and a full OS?
- What about DevOps?

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

Customer Data Center

Public Cloud

Disaster recovery

Production Cluster

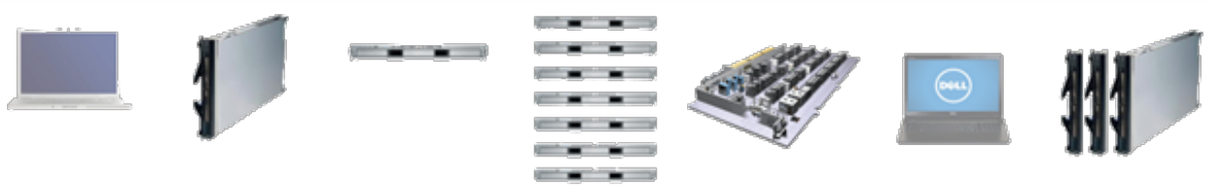
Production Servers

Contributor's laptop

Can I migrate smoothly and quickly?

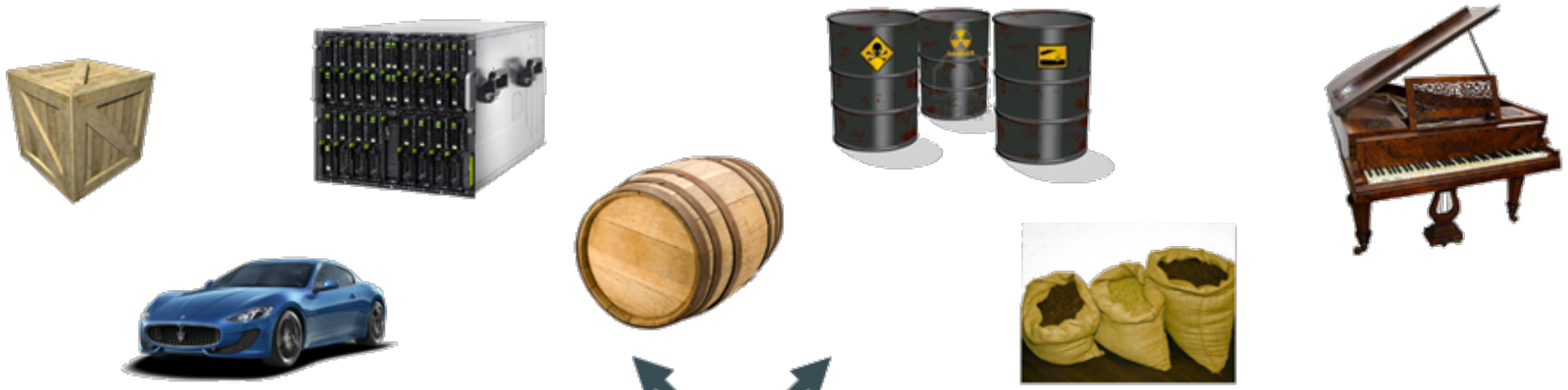
The Matrix from Hell

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



Cargo Transportation Pre-1960

Multiplicity of Goods



Do I worry about how goods interact (e.g. coffee beans next to spices)

Multipilicity of methods for transporting/storing

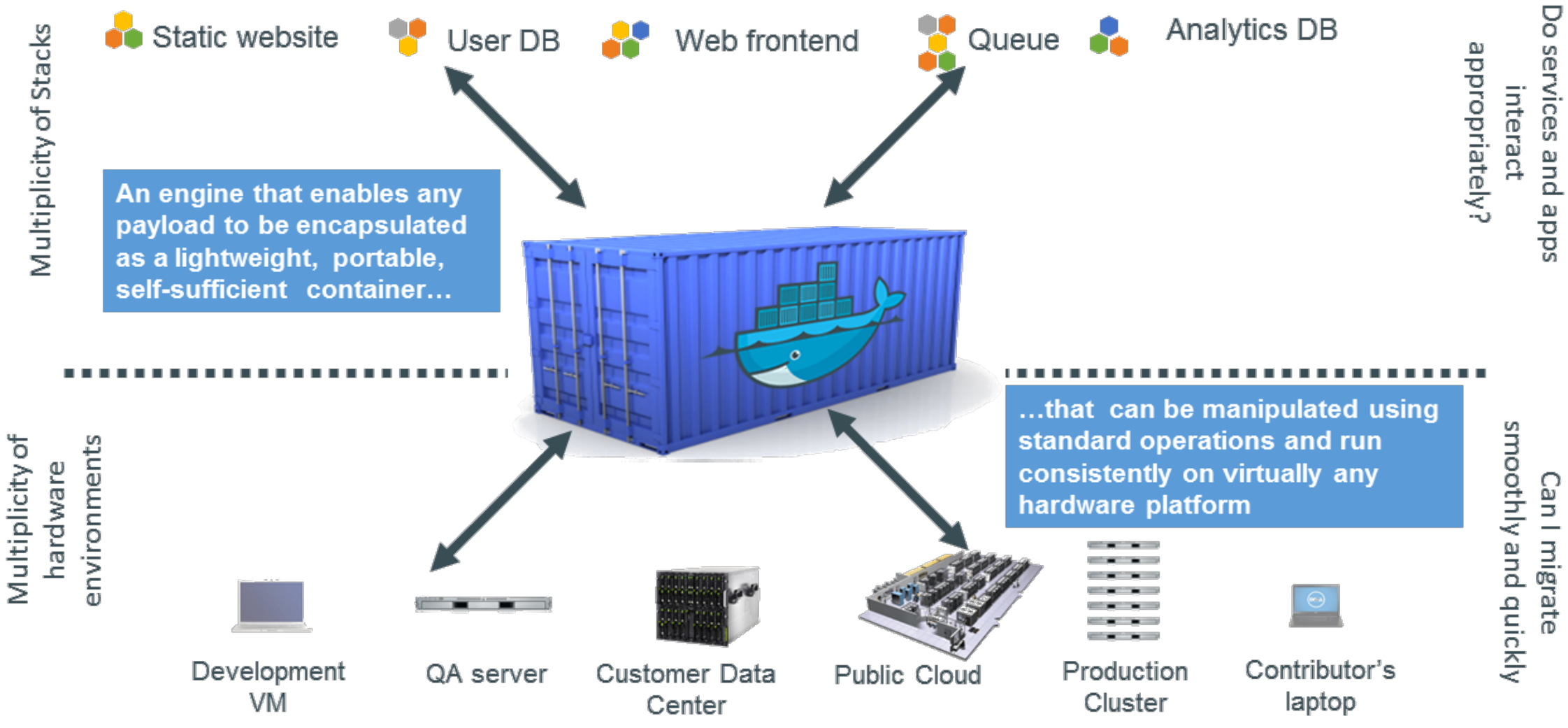
Can I transport quickly and smoothly (e.g. from boat to train to truck)

Also a Matrix from Hell

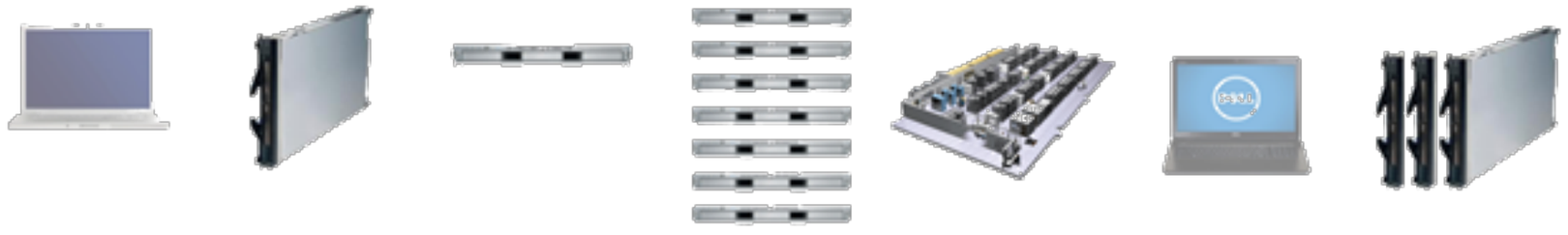
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution: Shipping Container

Docker: Container for Code

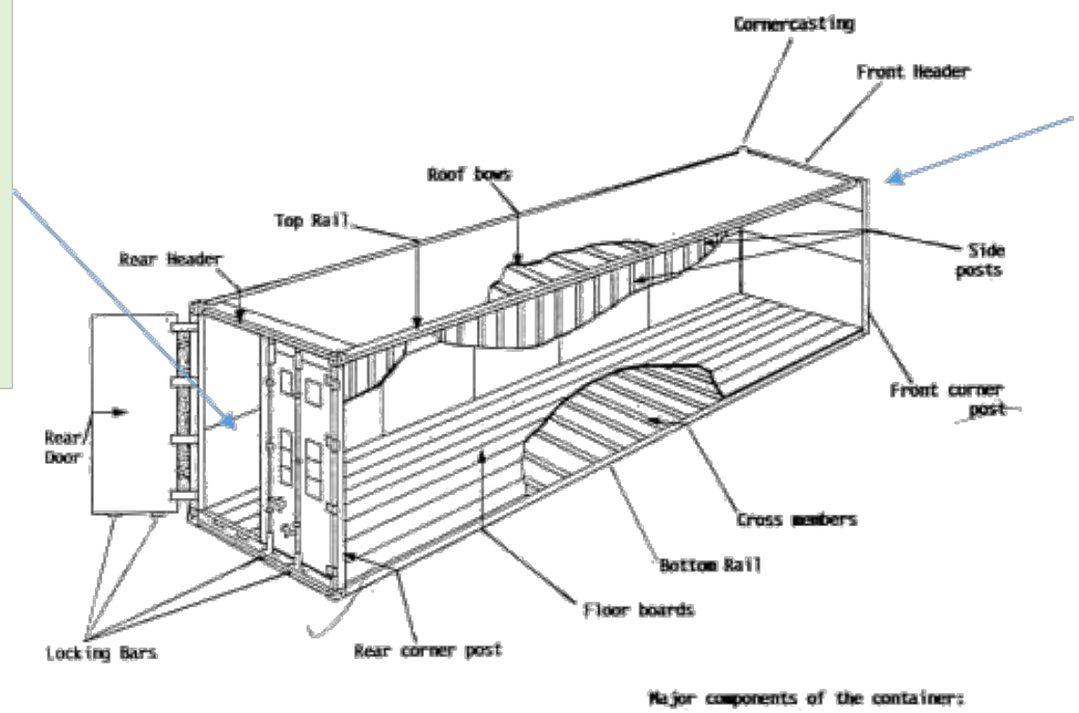


Static website							
Web frontend							
Background workers							
User DB							
Analytics DB							
Queue							
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers



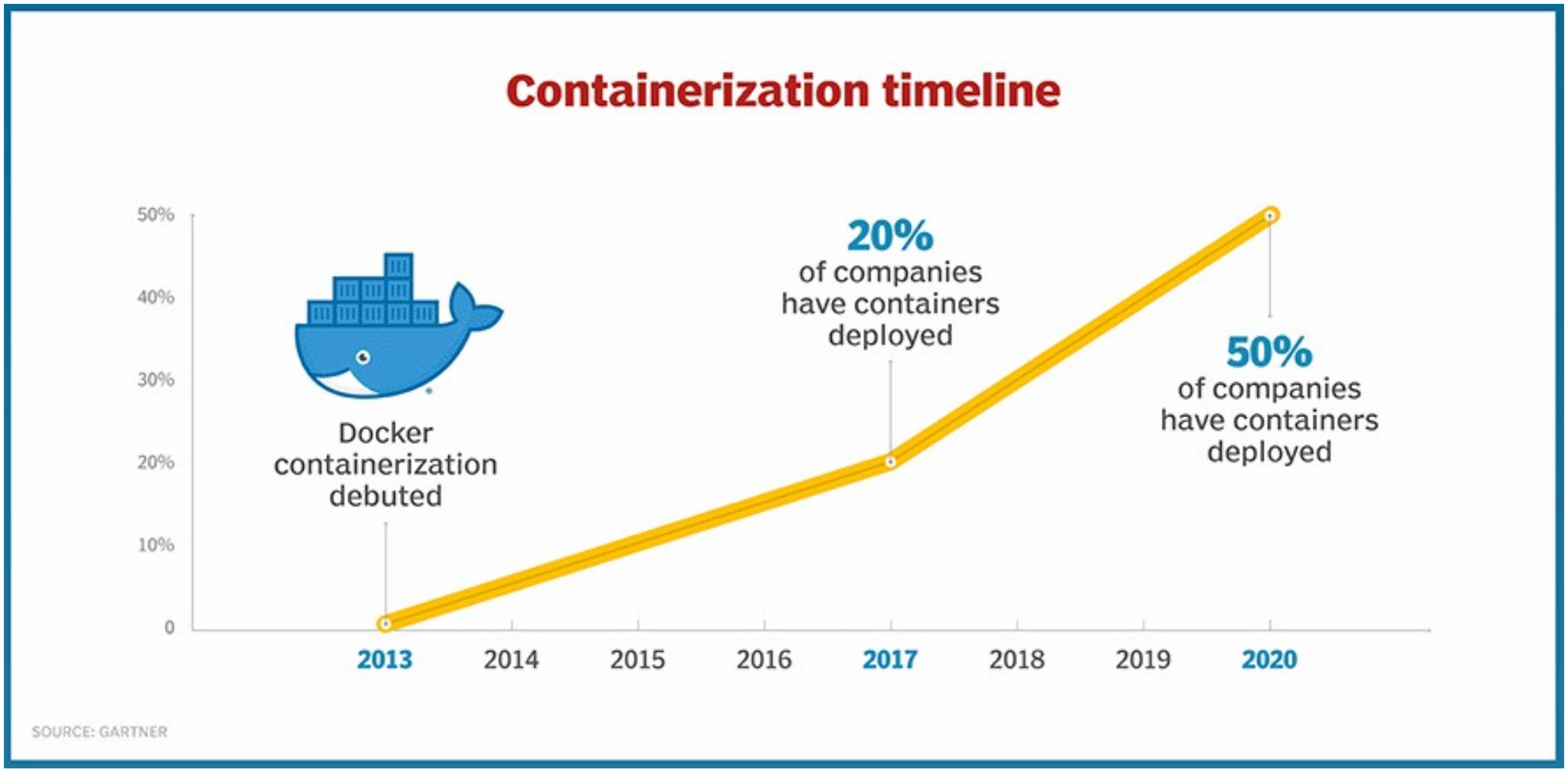
Why Does It Work? Separation of Concerns

- **Dan the Developer**
  - **Worries about what's "inside" the container**
    - **His code**
    - **His Libraries**
    - **His Package Manager**
    - **His Apps**
    - **His Data**
- **All Linux servers look the same**



Major components of the container:

- **Oscar the Ops Guy**
  - **Worries about what's "outside" the container**
    - **Logging**
    - **Remote access**
    - **Monitoring**
    - **Network config**
- **All containers start, stop, copy, attach, migrate, etc. the same way**


Containerization timeline
50%
40%
30%
20%
10%
0
Docker containerization debuted
20% of companies have containers deployed
50% of companies have containers deployed
2013
2014
2015
2016
2017
2018
2019
2020
SOURCE: GARTNER

Linux Containers

- Run everywhere
  - Regardless of kernel version
  - Regardless of host distro
    - Physical or virtual, cloud or not
    - Container and host architecture must match...
- Run anything
  - If it can run on the host, it can run in the container
  - If it can run on a Linux kernel, it can run, it can run in the container

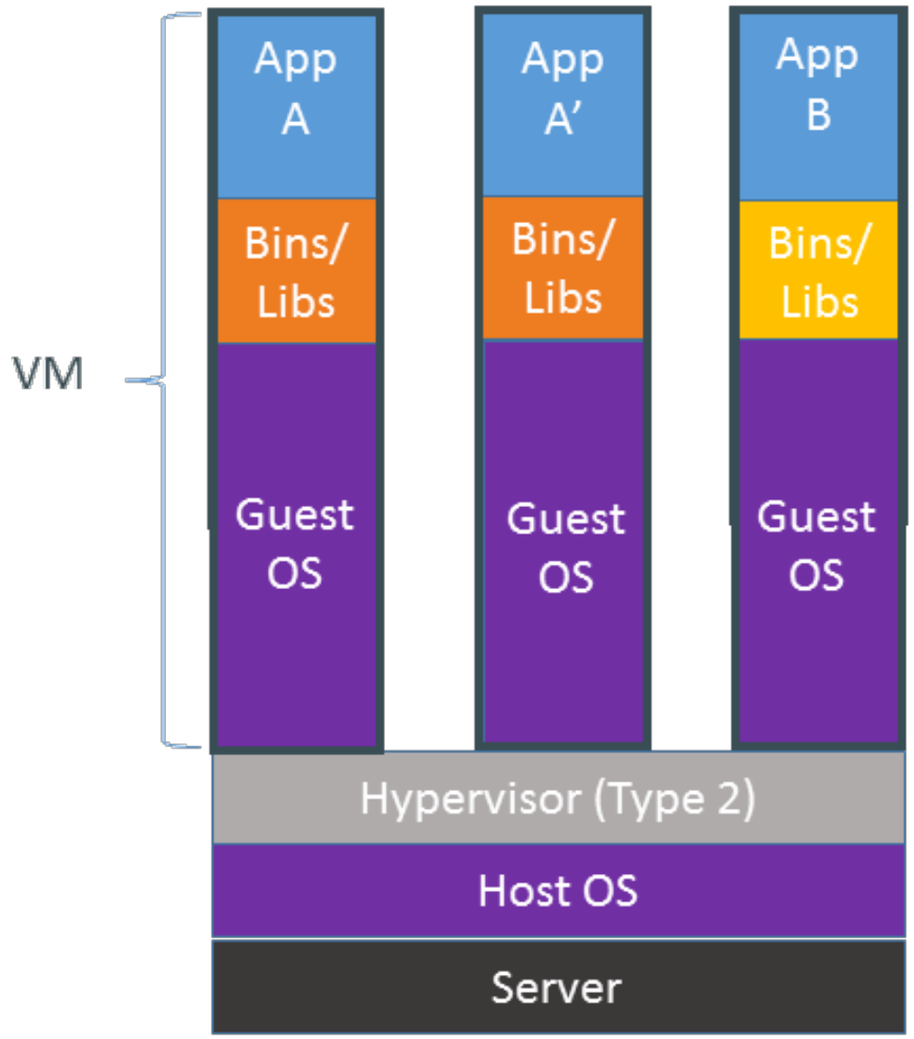
At High-Level: It Looks Like a VM

- Own process space
- Own network interface
- Can run stuff as root
- Can have its own /sbin/init (different from the host)

At Low-Level: OS-Level Virtualization

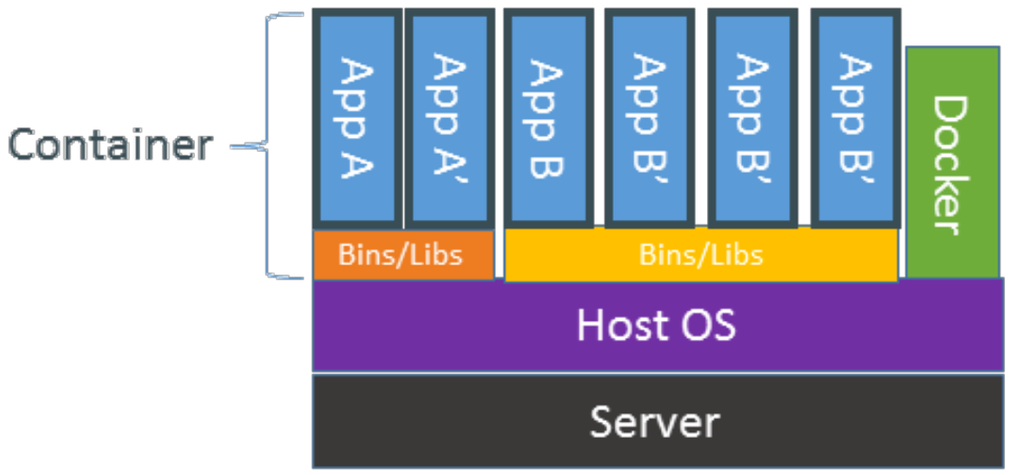
- Containers run on a host OS directly (and share the OS)
- Run as processes
- OS provides resource isolation and namespace isolation

VM vs Container



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

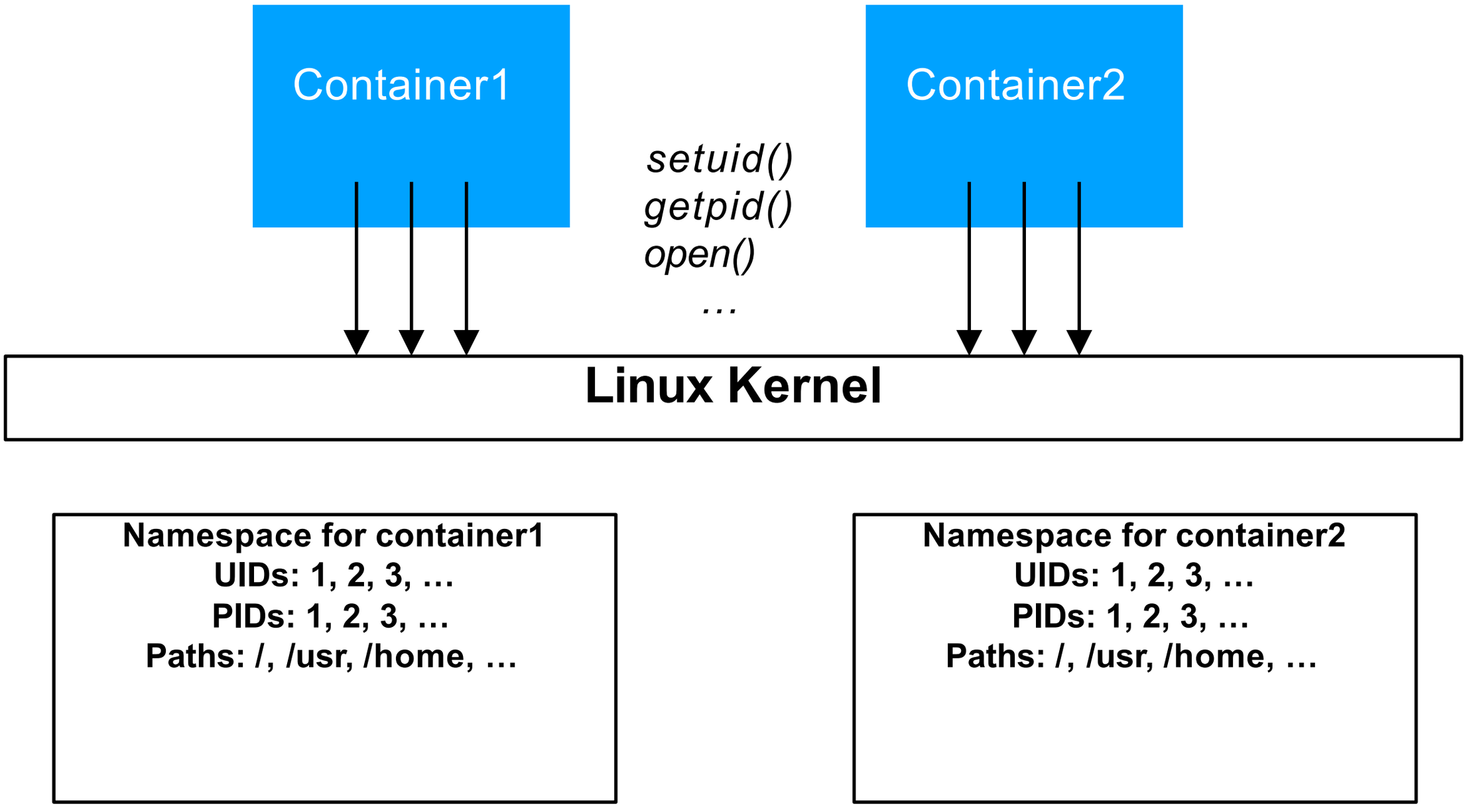
...result is significantly faster deployment, much less overhead, easier migration, faster restart



Using Namespaces to Separate “Views” of Users

- Namespace: naming domain for various resources
  - User IDs (UIDs)
  - Process IDs (PIDs)
  - File paths (mnt)
  - Network sockets
  - Pipe names

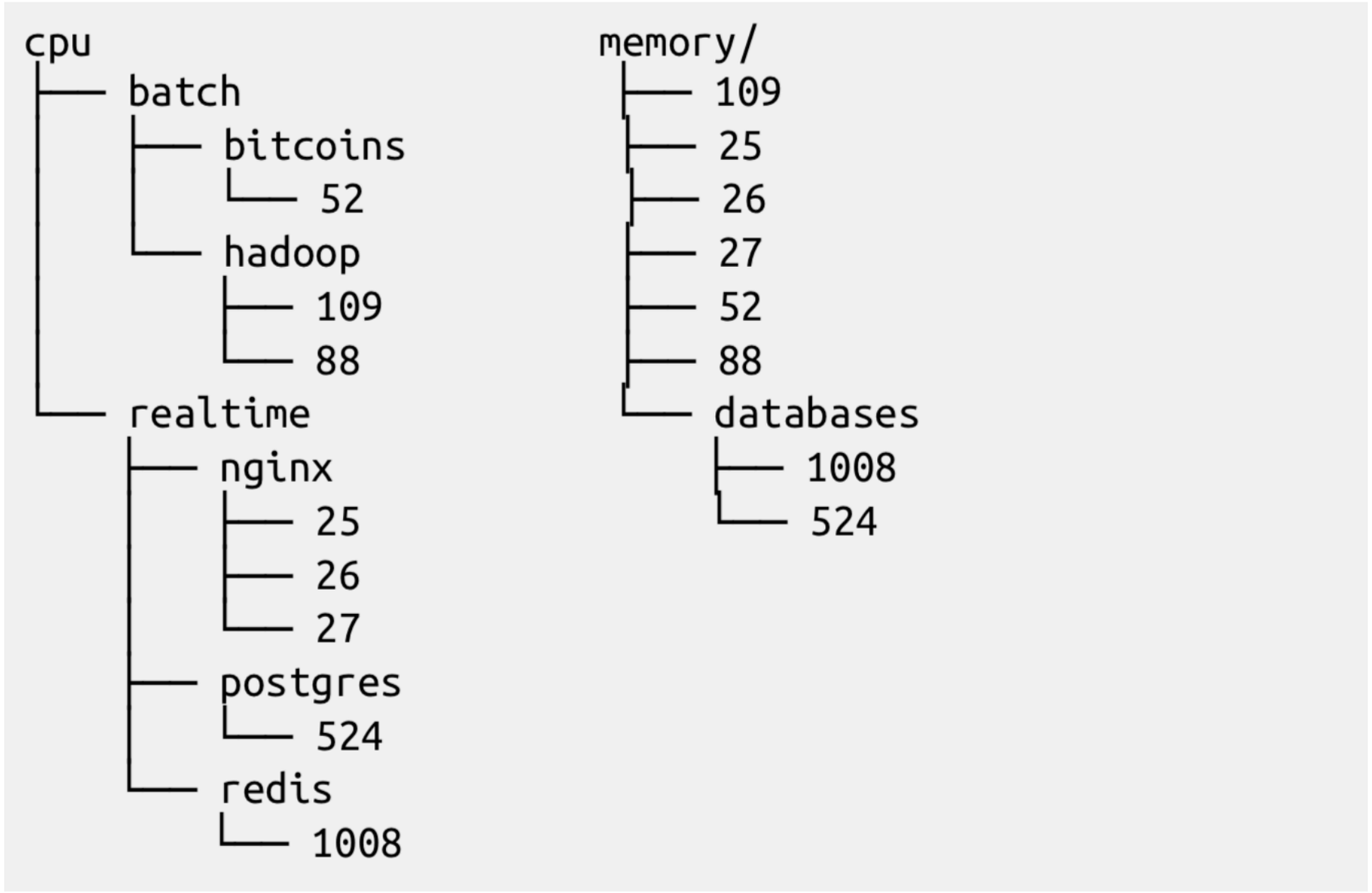
Namespaces Isolated by Kernel



Isolating Resources with cgroups

- Linux Control Groups (cgroups): collection of Linux processes
  - Limits resource usages at group level (e.g., memory, CPU, device)
  - Fair sharing of resources
  - Track resource utilization (e.g., could be used for billing/management)
  - Control processes (e.g., pause/resume, checkpoint/restore)

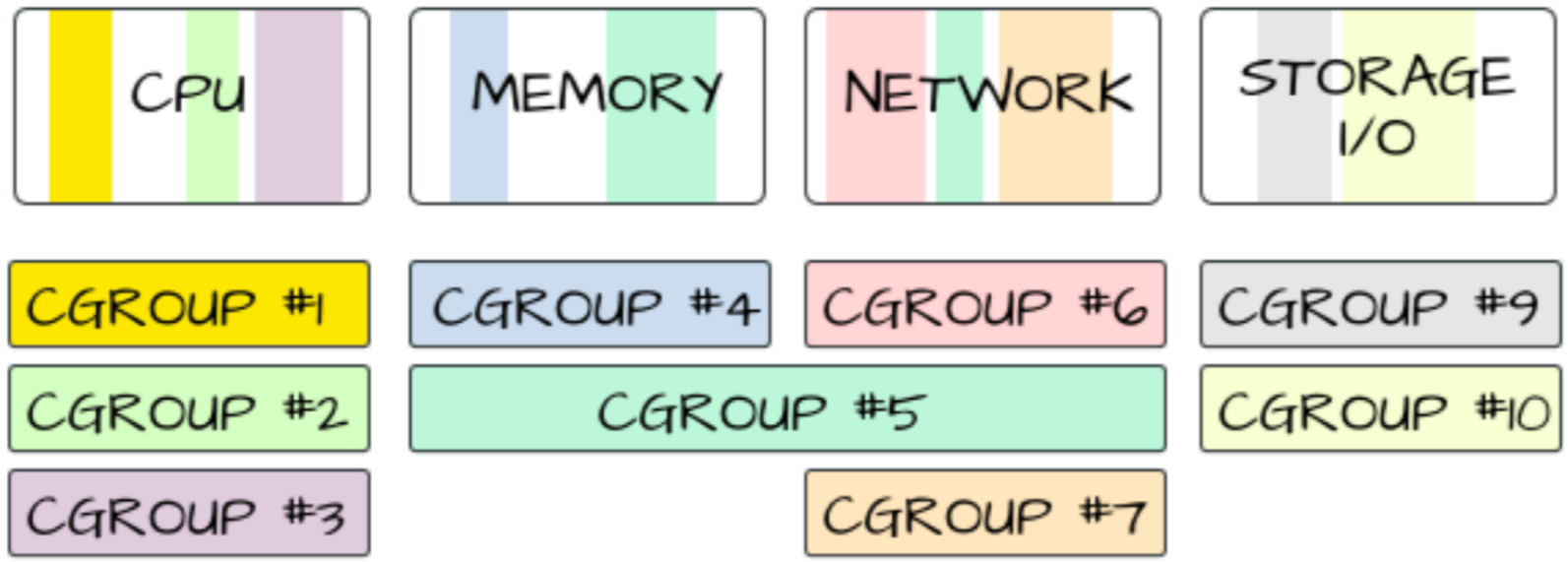
Example


cpu
batch
bitcoins
52
hadoop
109
88
realtime
nginx
25
26
27
postgres
524
redis
1008
memory/
109
25
26
27
52
88
databases
1008
524

Resource Isolation

Cgroups : Isolation and accounting

- cpu
- memory
- block i/o
- devices
- network
- numa
- freezer

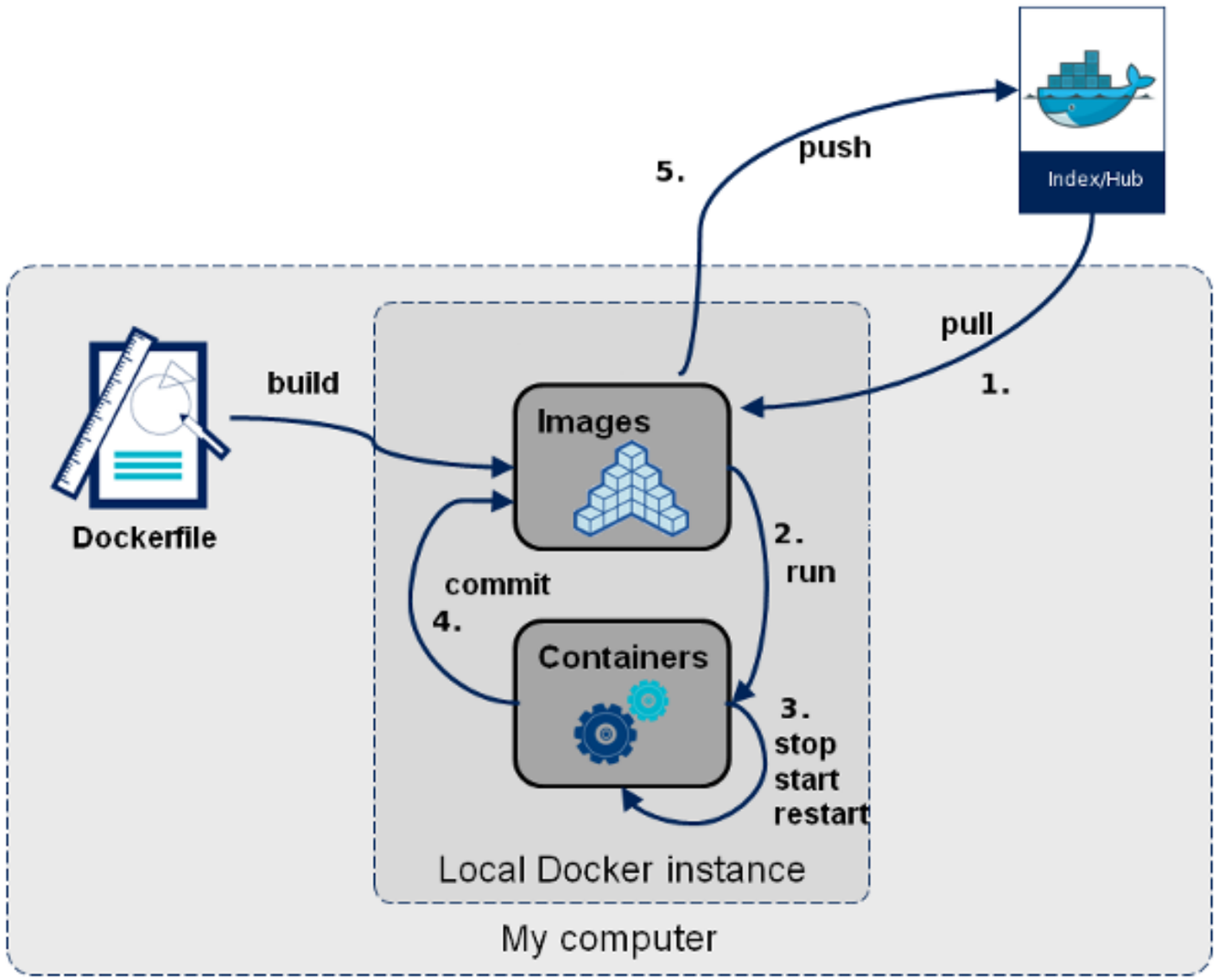


Efficiency: *almost* no overhead

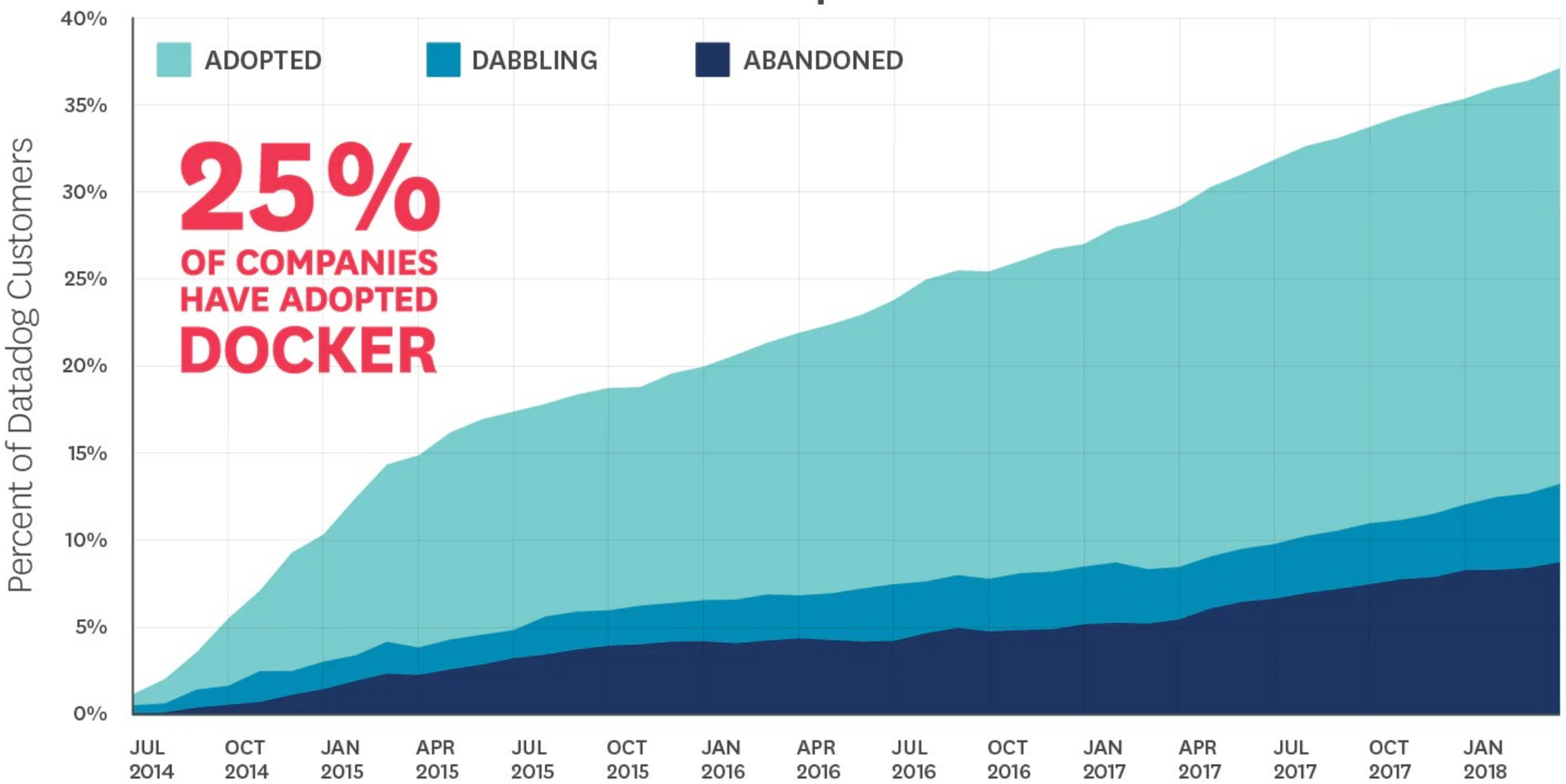
- Processes are isolated, but run straight on the host
- CPU performance = native performance
- Memory performance = a few % shaved off for (optional) accounting
- Network performance = small overhead; can be optimized to zero overhead

Docker

- Docker is a platform for developing, shipping & running application using container based virtualization technology.
  - founded as dotCloud in 2010
  - Docker: A container engine written in Go
  - popularized containerization
  - Provides a straightforward way to build and run containers

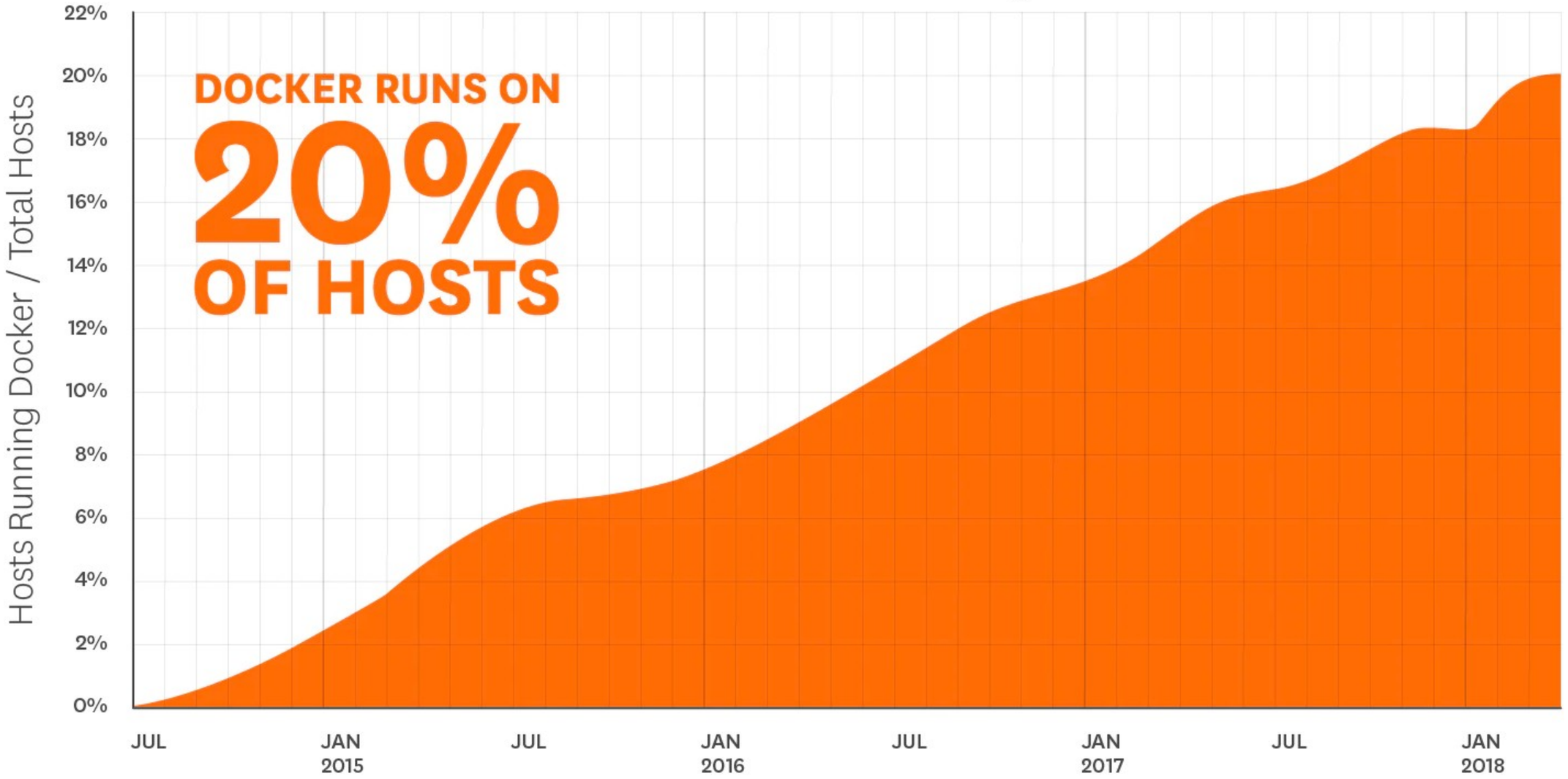


Docker Adoption Behavior



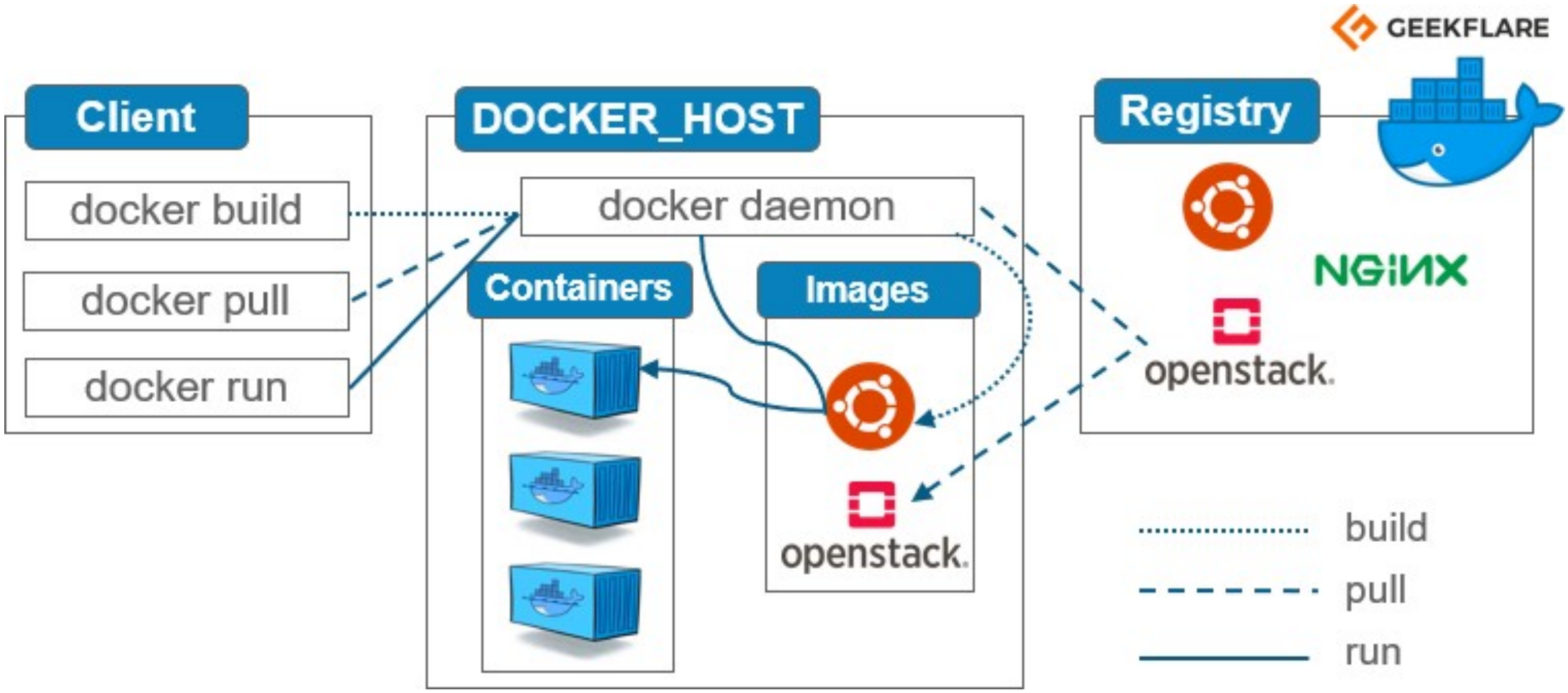
Source: Datadog

Portion of Hosts Running Docker



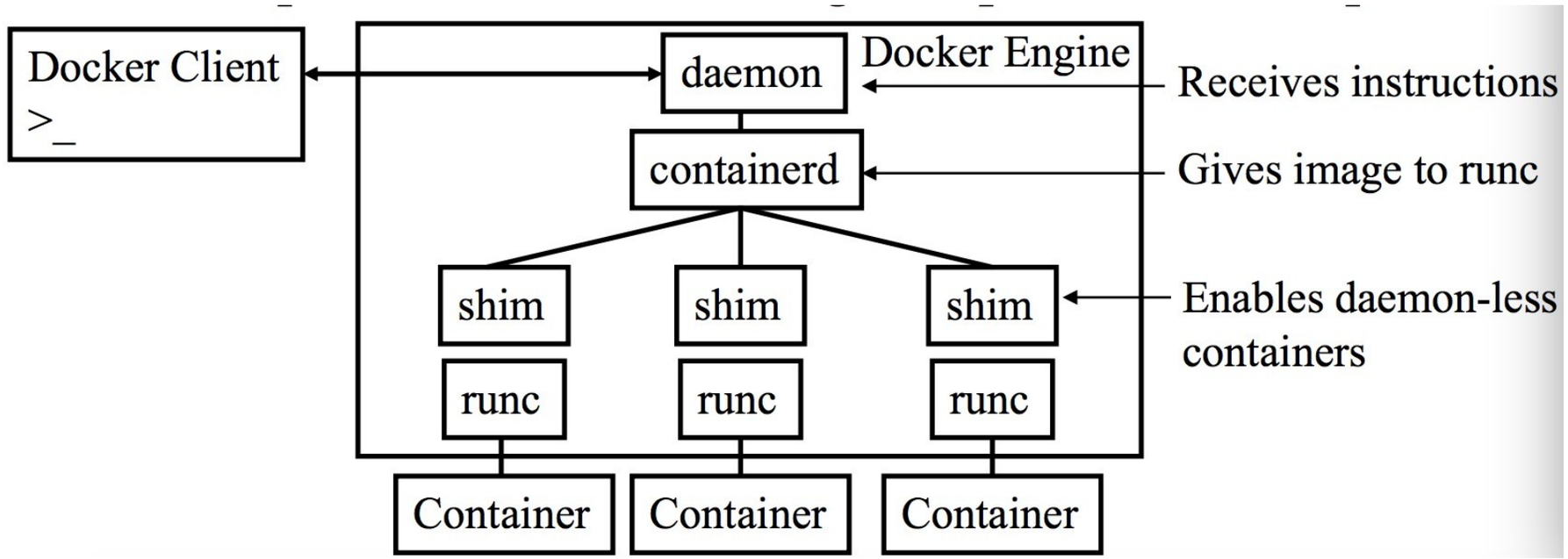
Source: Datadog

Docker Architecture



Docker Engine

- daemon: Rest API (receiving instructions) and other features
- containerd: Execution logic (e.g., start, stop, pause, unpause, delete containers)
- runc: A lightweight runtime CLI



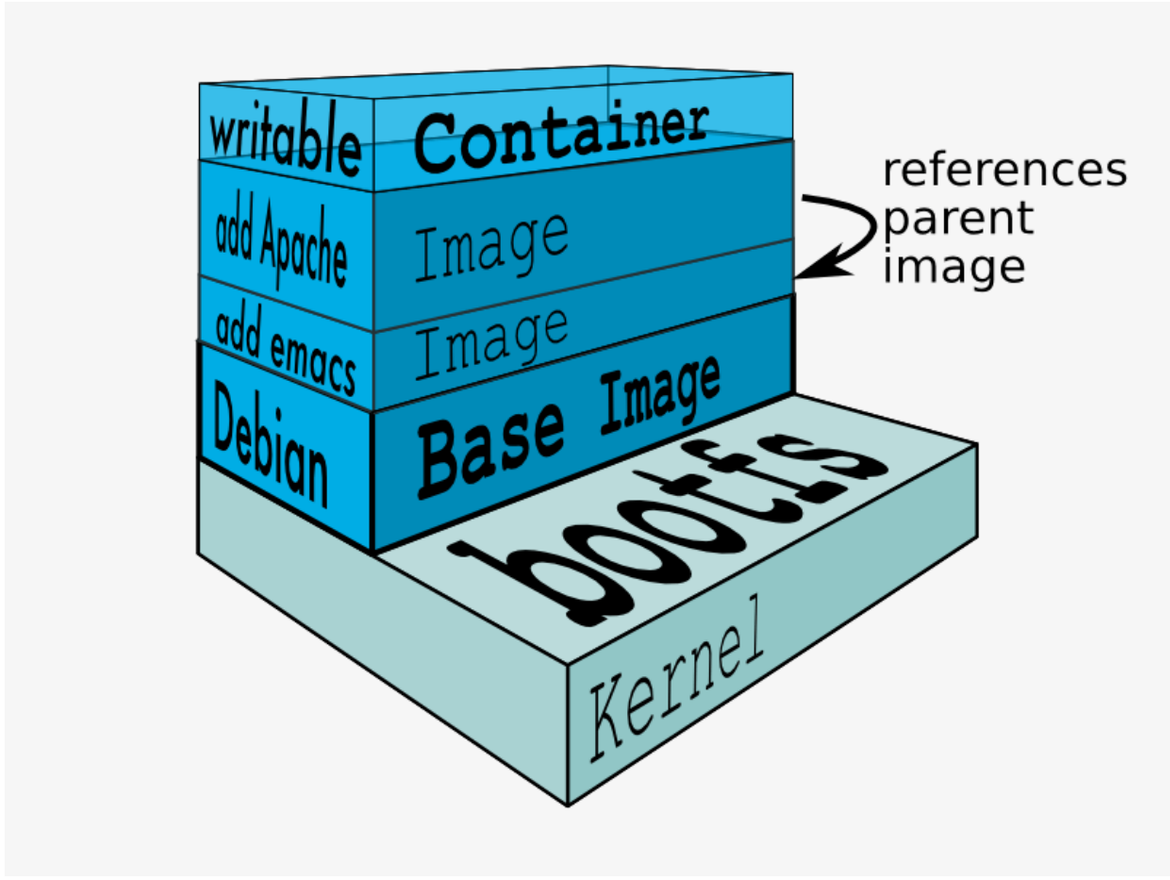
Source: N. Poulton, "Docker Deep Dive," Oct 2017, ISBN: 9781521822807

Docker Image

- A Docker image is a binary that includes all of the requirements for running a single Docker container, as well as metadata describing its needs and capabilities.

- Each image consists of a series of layers using the union file system

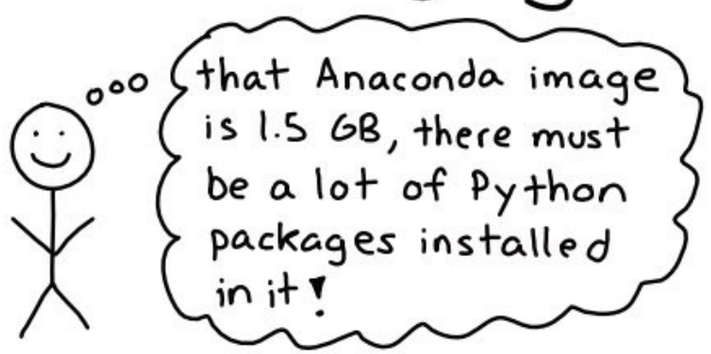
- When you change an image a new layer is created.



JULIA EVANS
@b0rk

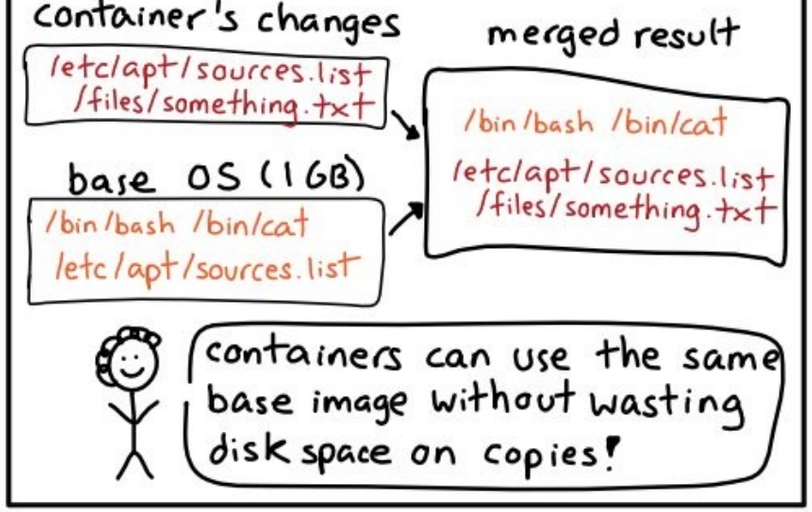
overlay filesystems

Docker images can be really big



every container needs a "copy" of its image

solution: ★ overlays ★



how to overlay

Linux has an 'overlayfs' driver that you can use to overlay directories like this:

```
$ mount -t overlay overlay
  -o lowerdir=/lower,upperdir=/upper,workdir=/work
  /merged
```

↑ base directory, will be read only

↑ where changes will go

↑ must be empty

Try it out! It's really easy.

how Docker runs containers

1. unpack the base image into a directory
2. make an empty directory for changes
3. overlay the new directory on top of the base
4. start the container!

Building an Image

- A Dockerfile that contains a set of instructions that tell Docker how to build our image. The Dockerfile can be used to generate an image stored on your local system.

- Docker daemon does actual build process. Contents of current context (folder) is sent to the daemon.

- Each Instruction creates a new layer in the image. The cache from previous builds is used as much as possible.

Dockerfile

```
FROM tomcat:7.0.62-jre8
MAINTAINER Jeff Ellin jeff.elli
ENV CORE_SQL_URL "jdbc:postgres
ENV CORE_SQL_USERNAME "tamr"
ENV CORE_SQL_PASSWORD "12345"
#Enable use of gui admin tool
add tomcat-users.xml $CATALINA_
#add the tamr war

add tamr.war /tamr/tamr.war
add catalina.sh $CATALINA_HOME/
RUN mv /tamr/*.war $CATALINA_HO
```

base image

environment variables

local files

execute command