

Container

Yiying Zhang

Cloud Computing - Obstacles and Opportunities

	Obstacle	Opportunity
1	Availability of Service	Use Multiple Cloud Providers; Use Elasticity to Prevent DDOS
2	Data Lock-In	Standardize APIs; Compatible SW to enable Surge Computing
3	Data Confidentiality and Auditability	Deploy Encryption, VLANs, Firewalls; Geographical Data Storage
4	Data Transfer Bottlenecks	FedExing Disks; Data Backup/Archival; Higher BW Switches
5	Performance Unpredictability	Improved VM Support; Flash Memory; Gang Schedule VMs
6	Scalable Storage	Invent Scalable Store
7	Bugs in Large Distributed Systems	Invent Debugger that relies on Distributed VMs
8	Scaling Quickly	Invent Auto-Scaler that relies on ML; Snapshots for Conservation
9	Reputation Fate Sharing	Offer reputation-guarding services like those for email
10	Software Licensing	Pay-for-use licenses; Bulk use sales

10 years later

- What's your view on Cloud Computing?

Outline

- What is cloud computing?
- Why cloud computing?
- How clouds are built?

Virtualization

- Traditional: applications run on physical servers
 - Manual mapping of apps to servers
 - Apps can be distributed
 - Storage may be on a SAN or NAS
 - IT admins deal with “change”
- Modern: virtualized data centers
 - App run inside virtual servers; VM mapped onto physical servers
 - Provides flexibility in mapping from virtual to physical resources

Virtualization Benefit

- Resource management is simplified
 - Application can be started from preconfigured VM/container images
 - Virtualization layer / hypervisor permits resource allocations to be varied dynamically
 - VMs can be migrated without application down-time

Virtual Datacenter

- A cluster of machines, each running a set of VMs
 - drive up utilization by packing many VMs onto each cluster node
 - fault recovery is simplified
 - if hardware fails, copy VM image elsewhere
 - if software fails, restart VM from snapshot
- can safely allow third parties to inject VM images into your data center
 - hosted VMs in the cloud, commercial computing grids

Recent Trend: Container

- Light-weight virtualization
 - Running multiple isolated user-space applications on one OS
 - Virtualization layer runs as an application within the OS
 - Focusing on resource and namespace isolation
- Example: Docker, LXC, Kubernetes, Xen Unikernel

Software-Defined Data Center

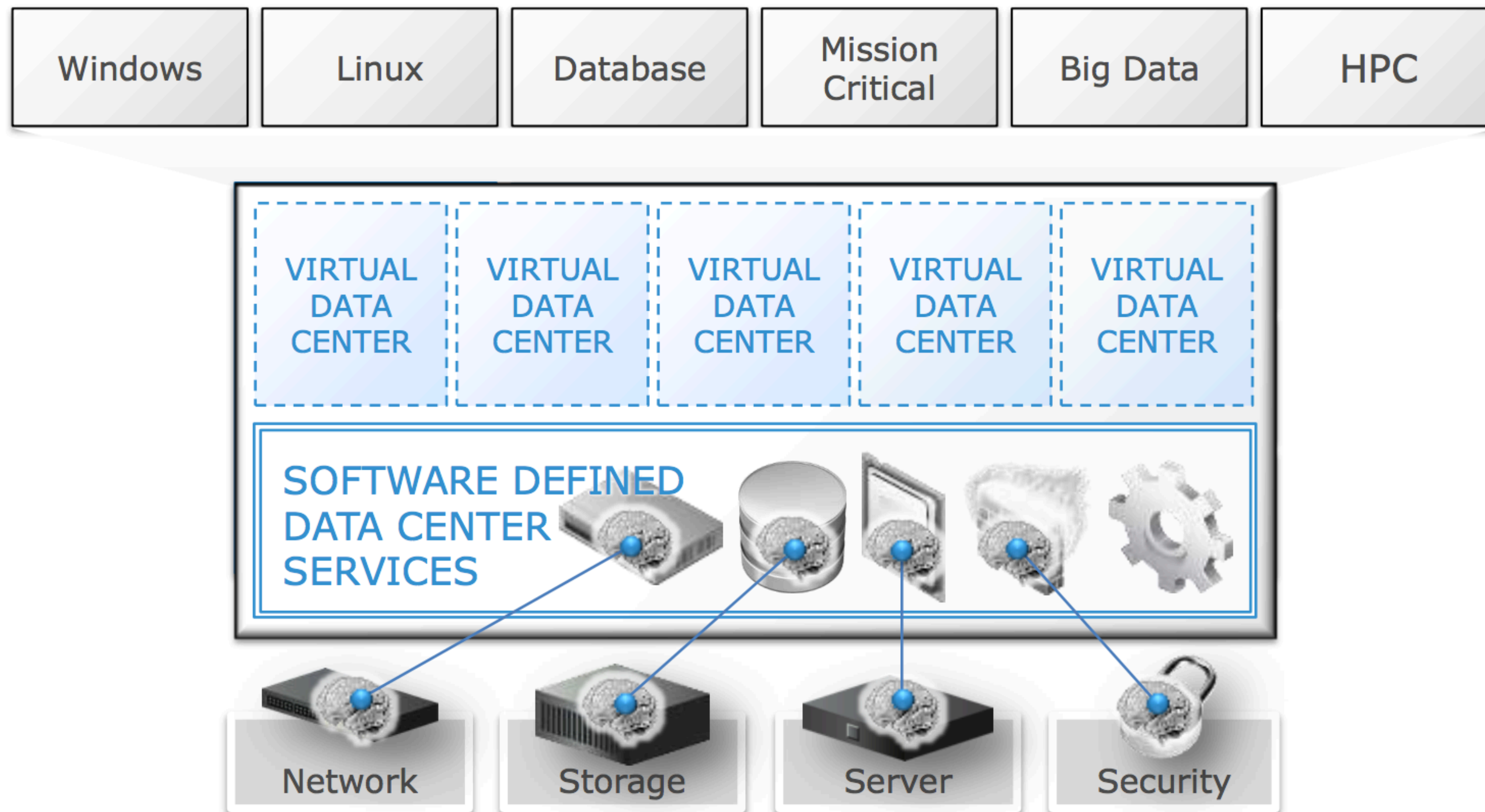
- All infrastructure is virtualized and delivered as a service & the control of this datacenter is entirely automated by software

Traditional Data Center



Source: EMC

Software-Defined Data Center



Source: EMC

Outline

- Motivation and overview
- Linux container techniques
- Docker
- Security of containers

Acknowledgment: some slides from “Introduction to Docker” (Docker Inc.)

Lec1: Summary of Virtualization History

- Invented by IBM in 1960s for sharing expensive mainframes
- Popular research ideas in 1960s and 1970s
- Interest died as the adoption of cheap PCs and multi-user OSes surged in 1980s
- A (somewhat accidental) research idea got transferred to VMware
- Real adoption happened with the growth of cloud computing
- New forms of virtualization: container and serverless, in the modern cloud era

Do VMs Fit All Today's Cloud User Needs?

- 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 + psycopg2 + postgresql-client

Do services and apps
interact
appropriately?

Multiplicity of
hardware
environments

 Development VM

 QA server

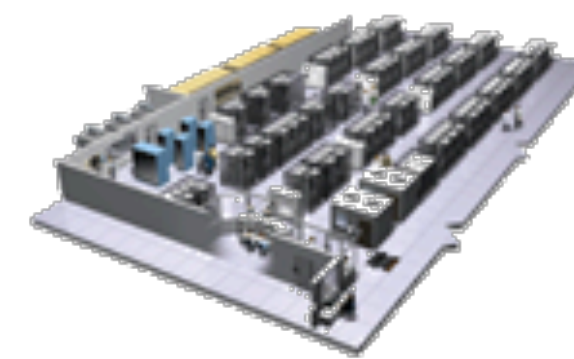
Customer Data Center



Public Cloud

Disaster recovery

Production Servers



Production Cluster

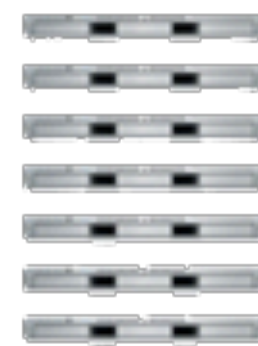


Contributor's laptop



Can I migrate
smoothly and
quickly?

The Matrix from Hell

	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
								

Cargo Transportation 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)

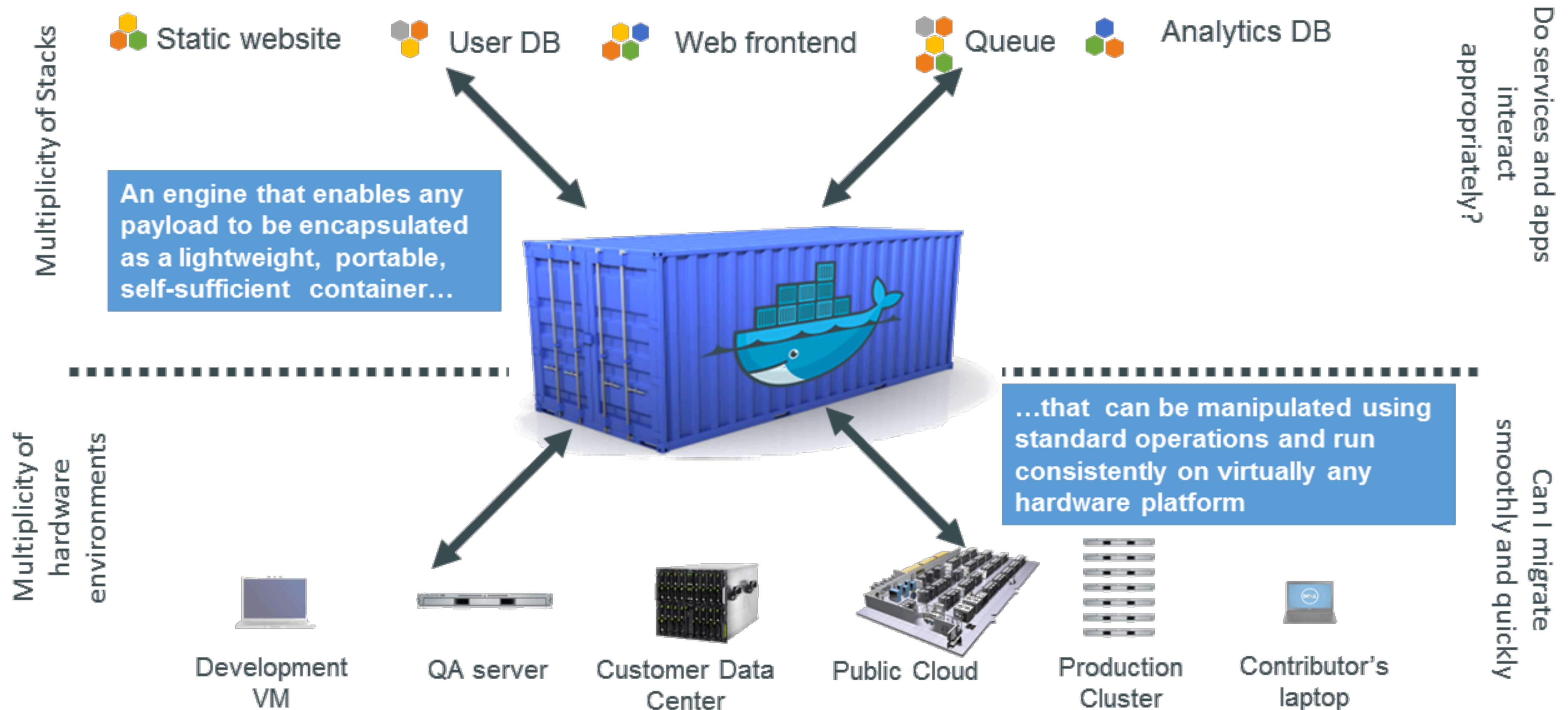
Also a Matrix from Hell

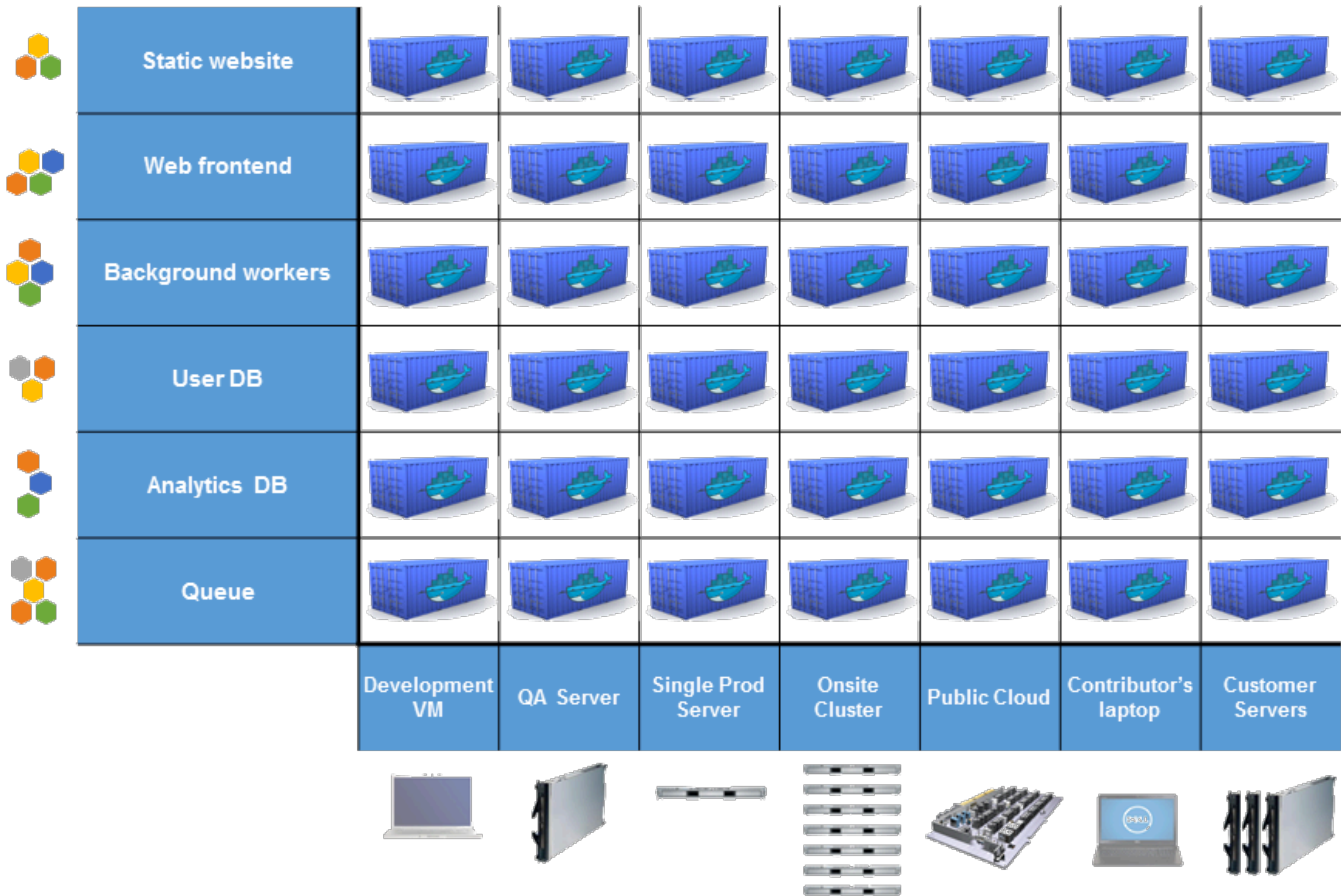
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution: Shipping Container



Docker: Container for Code





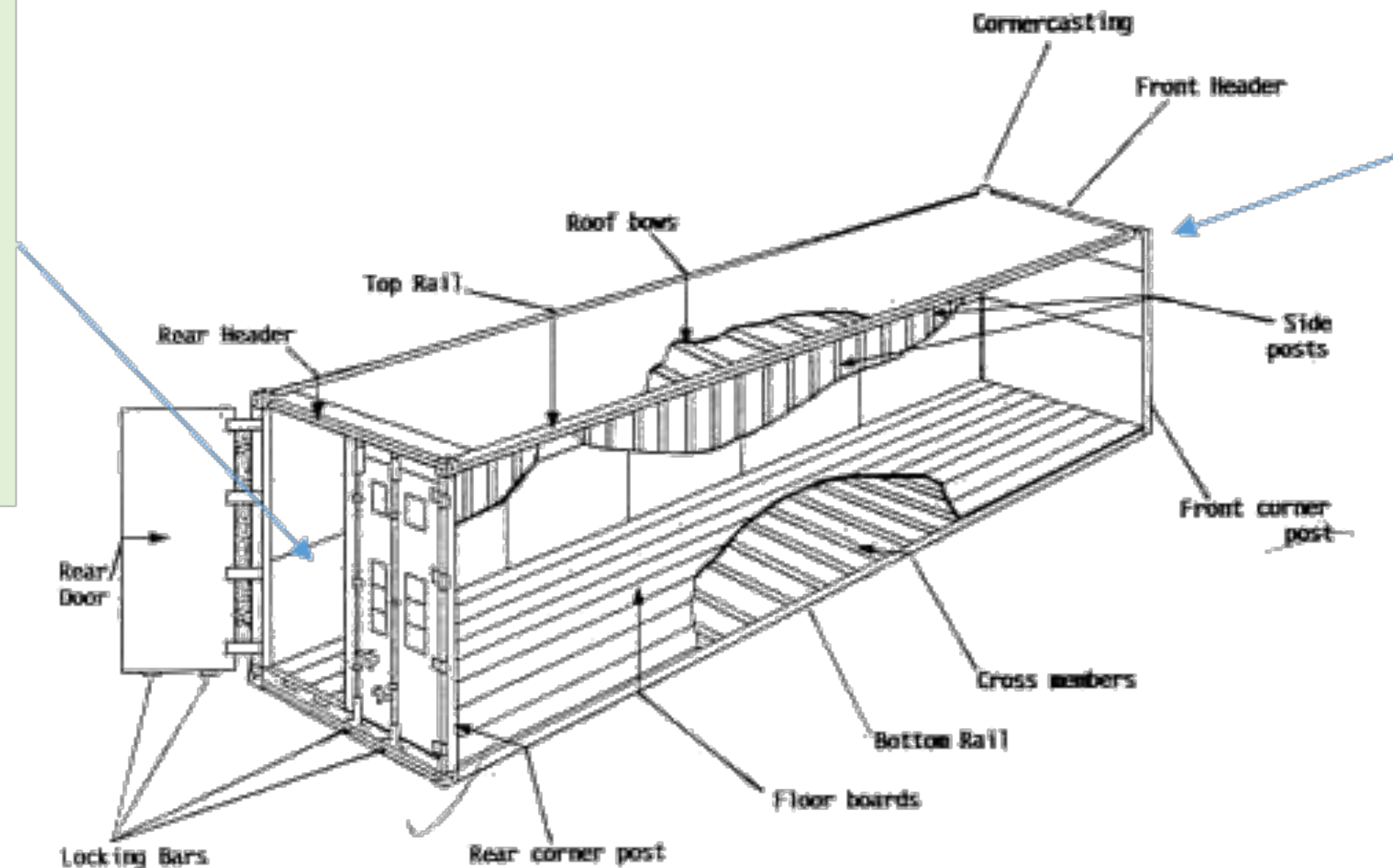
Why Does It Work? Separation of Concerns

- **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

- **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



Major components of the containers:

Why Developers Care?

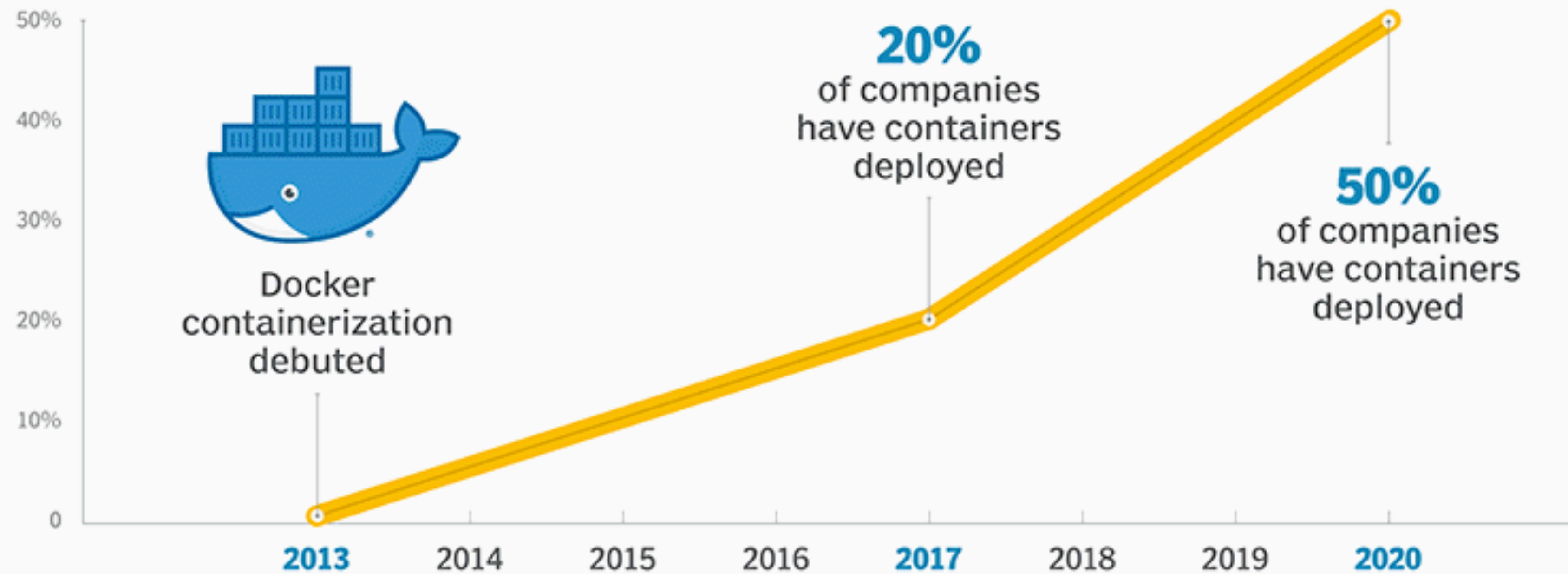
- Build once... (finally) run anywhere*
 - A clean, portable runtime environment for your app
 - No worries about missing dependencies, packages, etc. during deployments
 - Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying
 - Automate testing, integration, packaging...anything you can script
 - Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers
 - Deploy services like VM, but without the overhead of a VM

* "anywhere" means an x86 server running a modern Linux kernel (3.2+ generally or 2.6.32+ for RHEL 6.5+, Fedora, & related)

Why Administrators Care?

- Configure once... run anything
 - Make the entire lifecycle more efficient, consistent, and repeatable
 - Eliminate inconsistencies between development, test, production, and customer environments
 - Support segregation of duties
 - Significantly improves the speed and reliability of continuous deployment and continuous integration systems
 - Address significant performance, costs, deployment, and portability issues normally associated with VMs

Containerization timeline



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

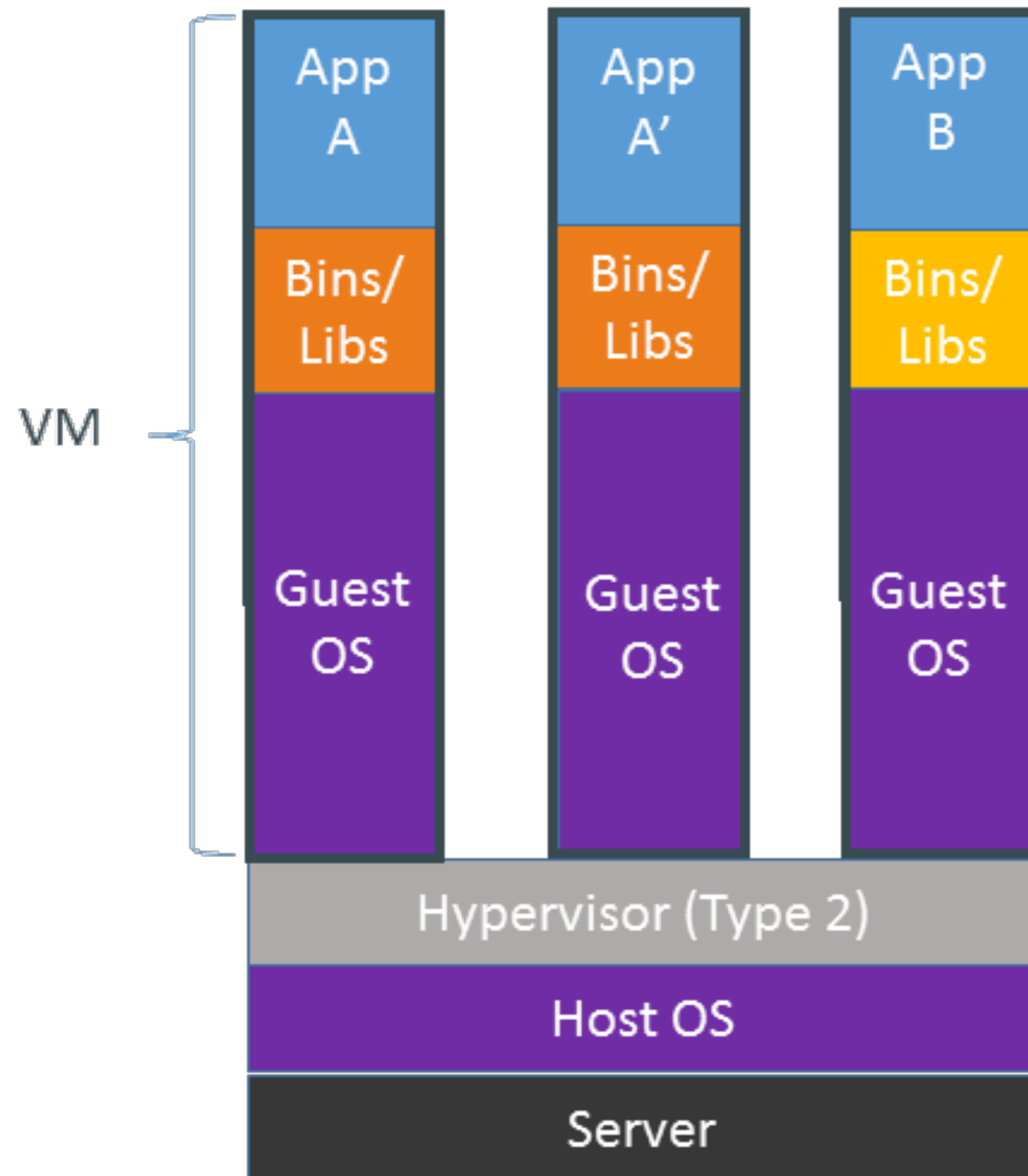
At High-Level: It Looks Like a VM

- 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

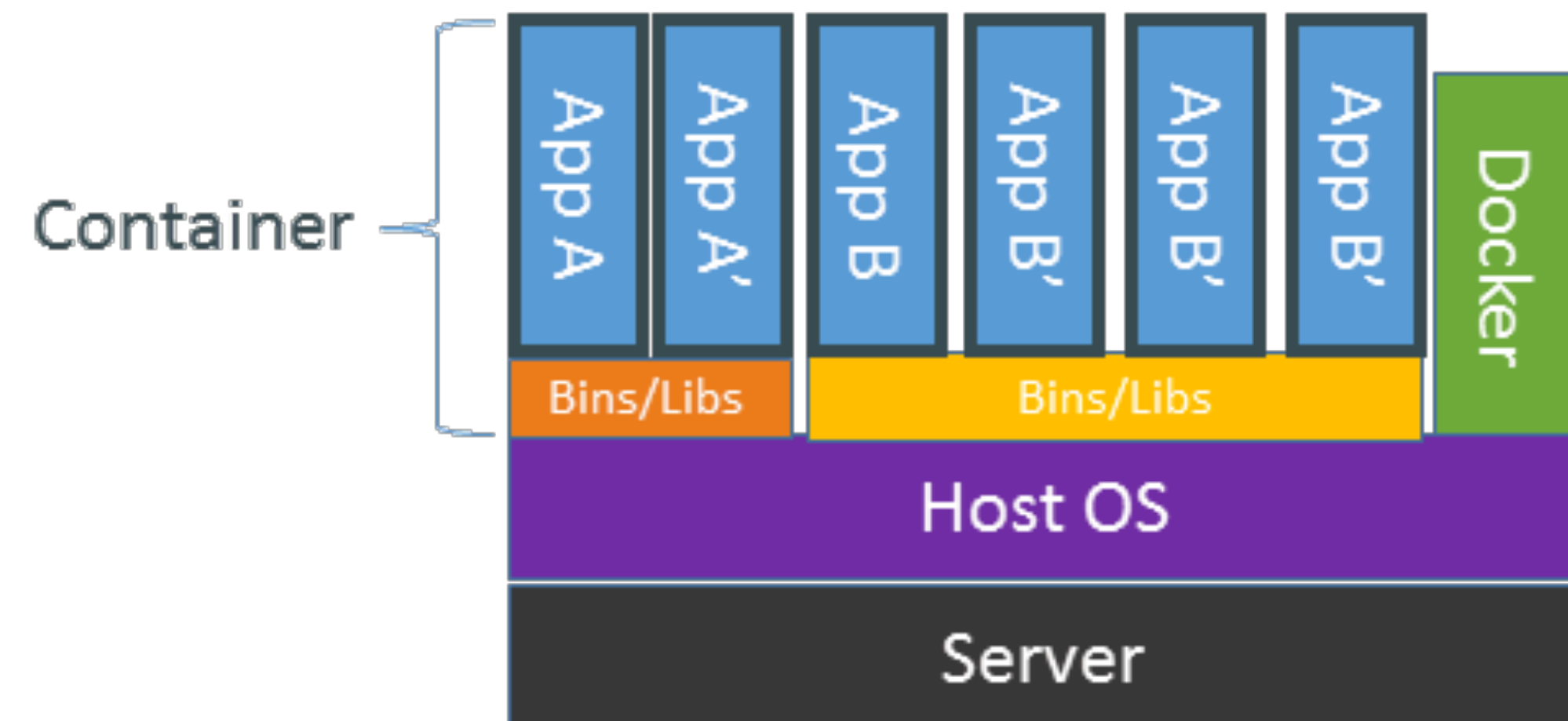
- Containers run on a host OS directly (and share the OS)
- Run as processes
- OS provides resource isolation and namespace isolation

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



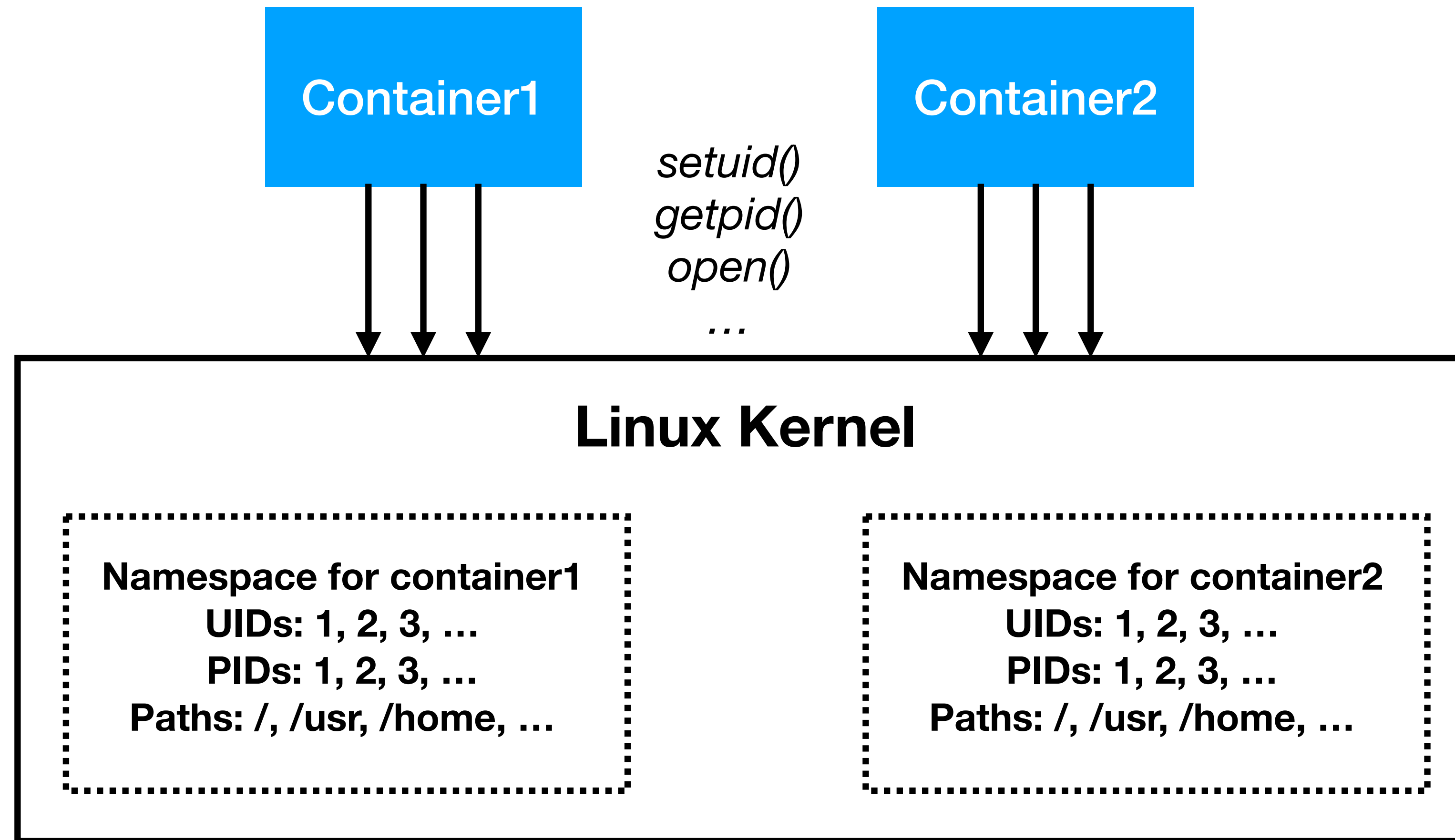
Isolating Resources with cgroups

- 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)

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

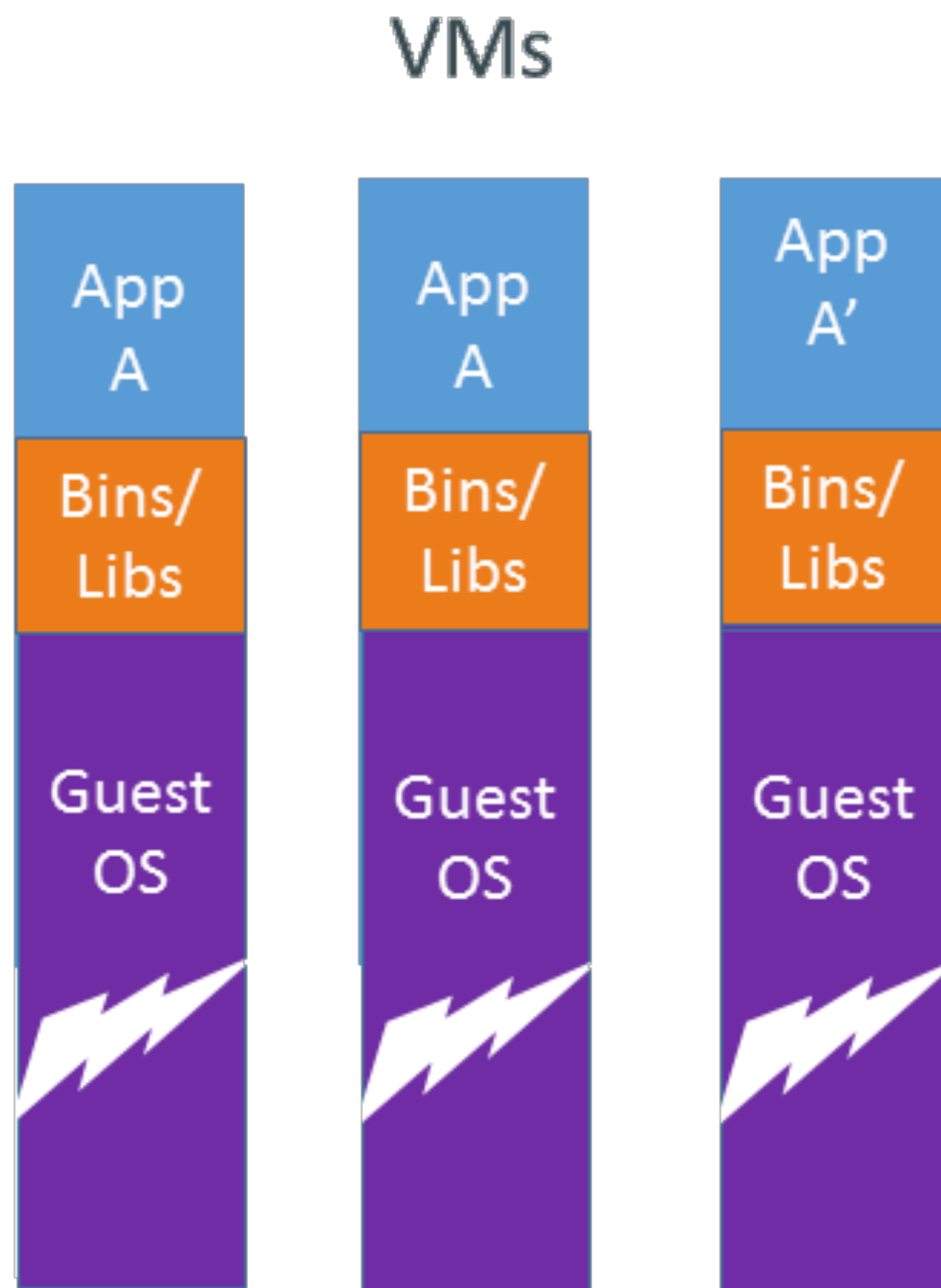
Namespaces Isolated by Kernel



Efficiency: *almost* no overhead

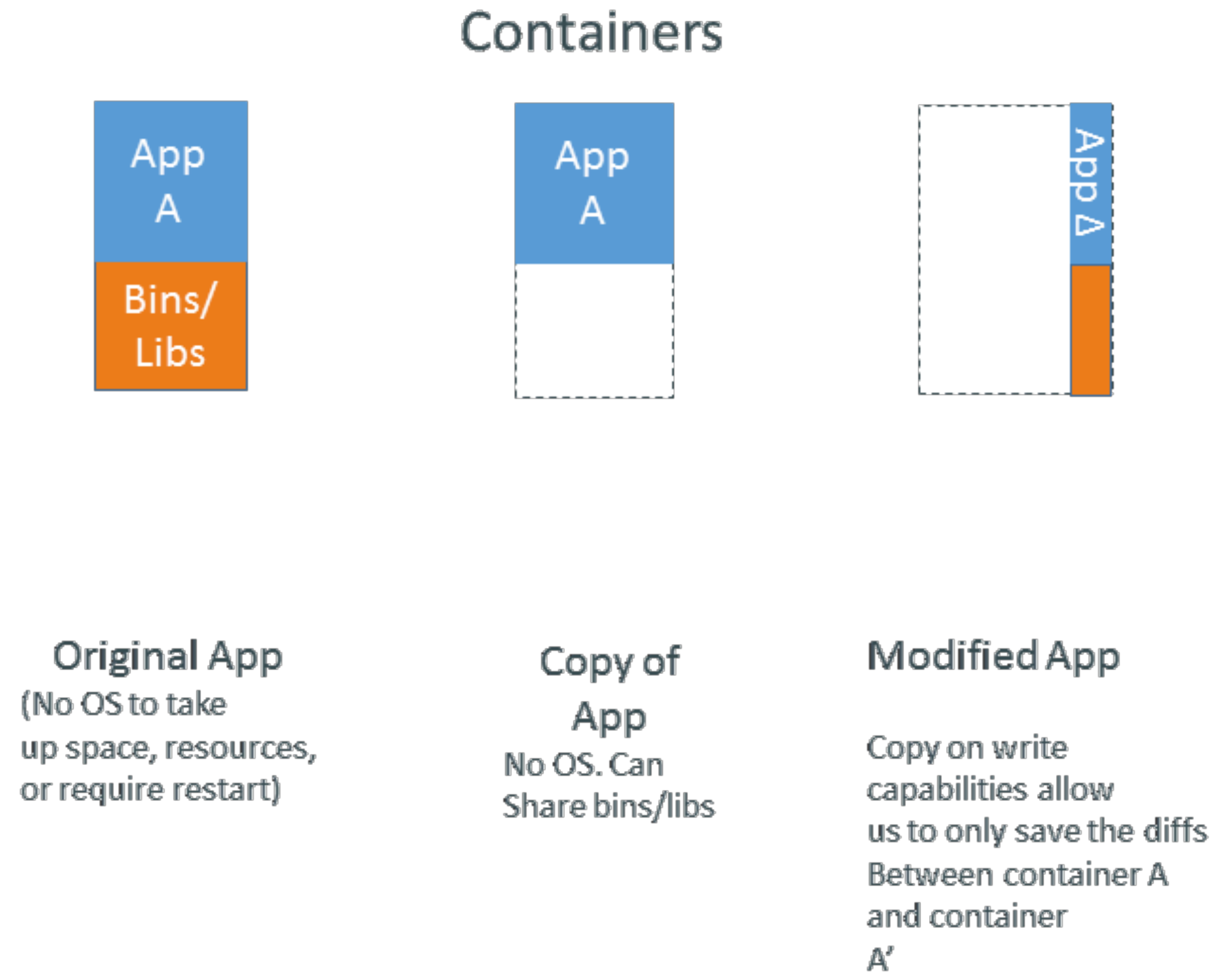
- Processes are isolated, but run straight on the host
- CPU performance = native performance
- Memory performance = a few % shaved off for (optional) accounting
- Network performance = small overhead; can be optimized to zero overhead

Why are Containers Lightweight?



VMs

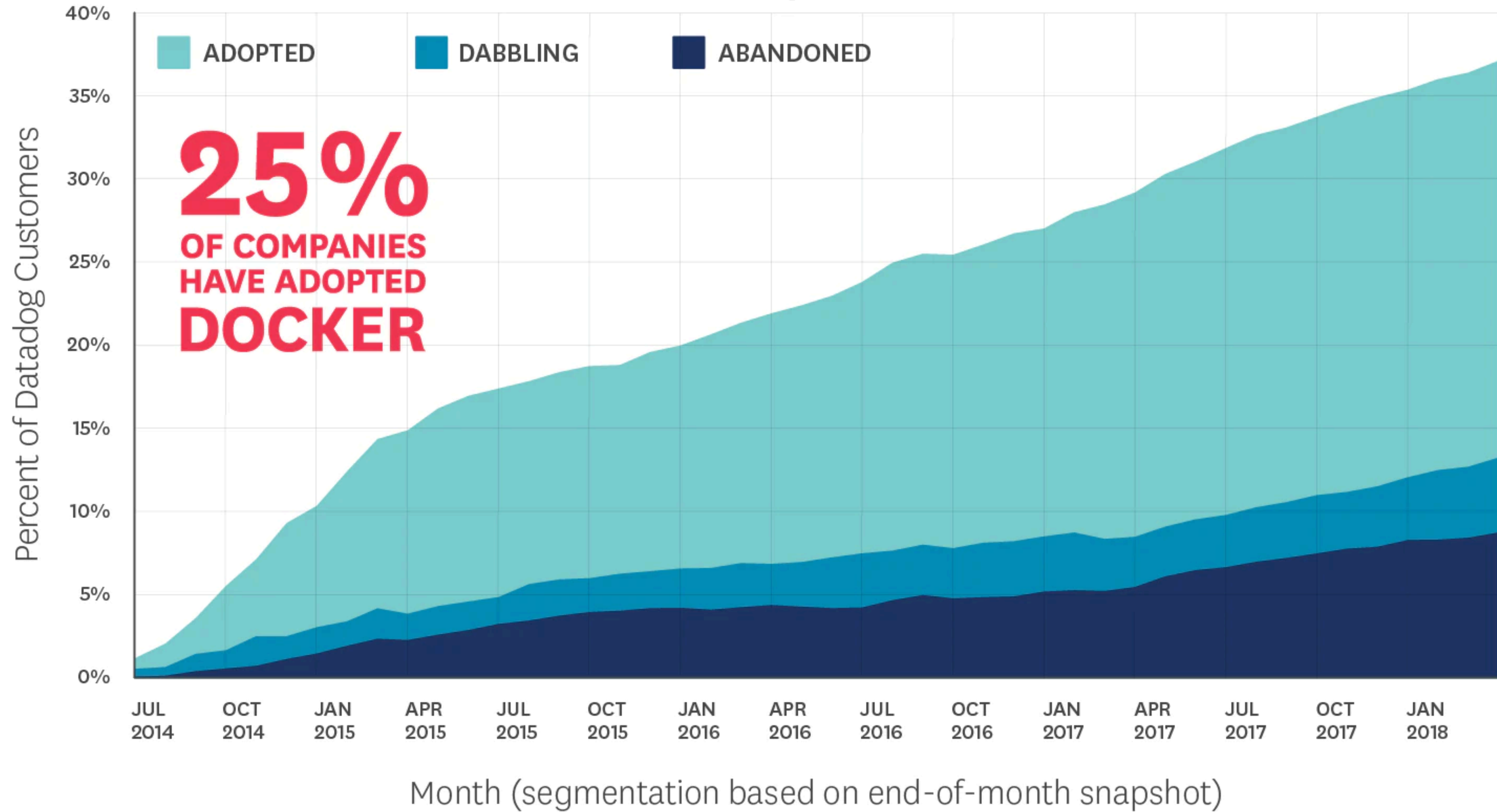
Every app, every copy of an app, and every slight modification of the app requires a new virtual server



Docker

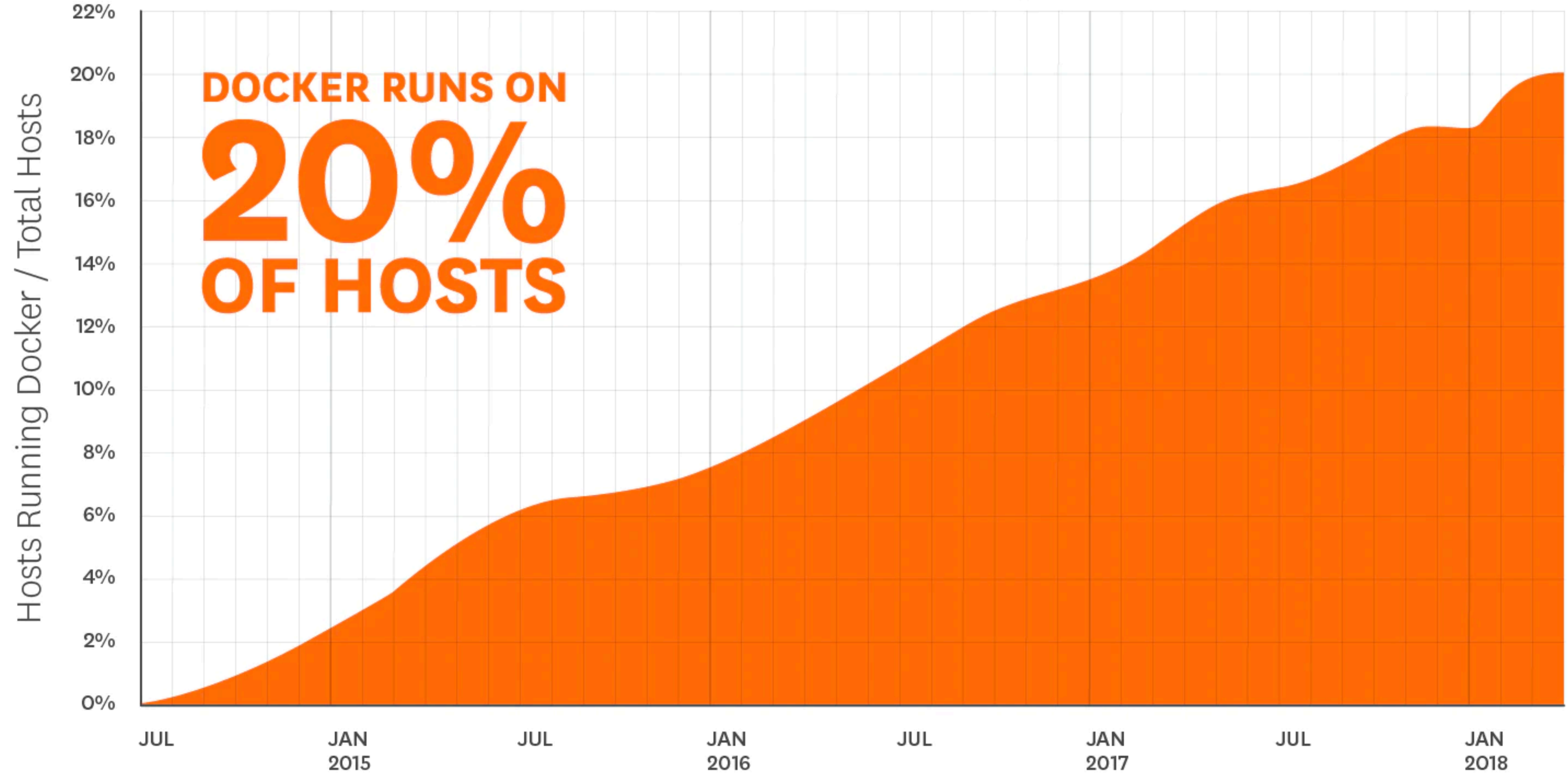
- Docker Inc
 - Founded as dotCloud, Inc. in 2010 by Solomon Hykes (renamed to Docker Inc. in 2013)
 - Estimated to be valued at over \$1 billion (101-250 employees)
- Docker the software
 - A container engine written in Go (based on Linux container)
- Docker community
 - Now 2243 contributors, 19K forks of docker engine on GitHub (called Moby)

Docker Adoption Behavior



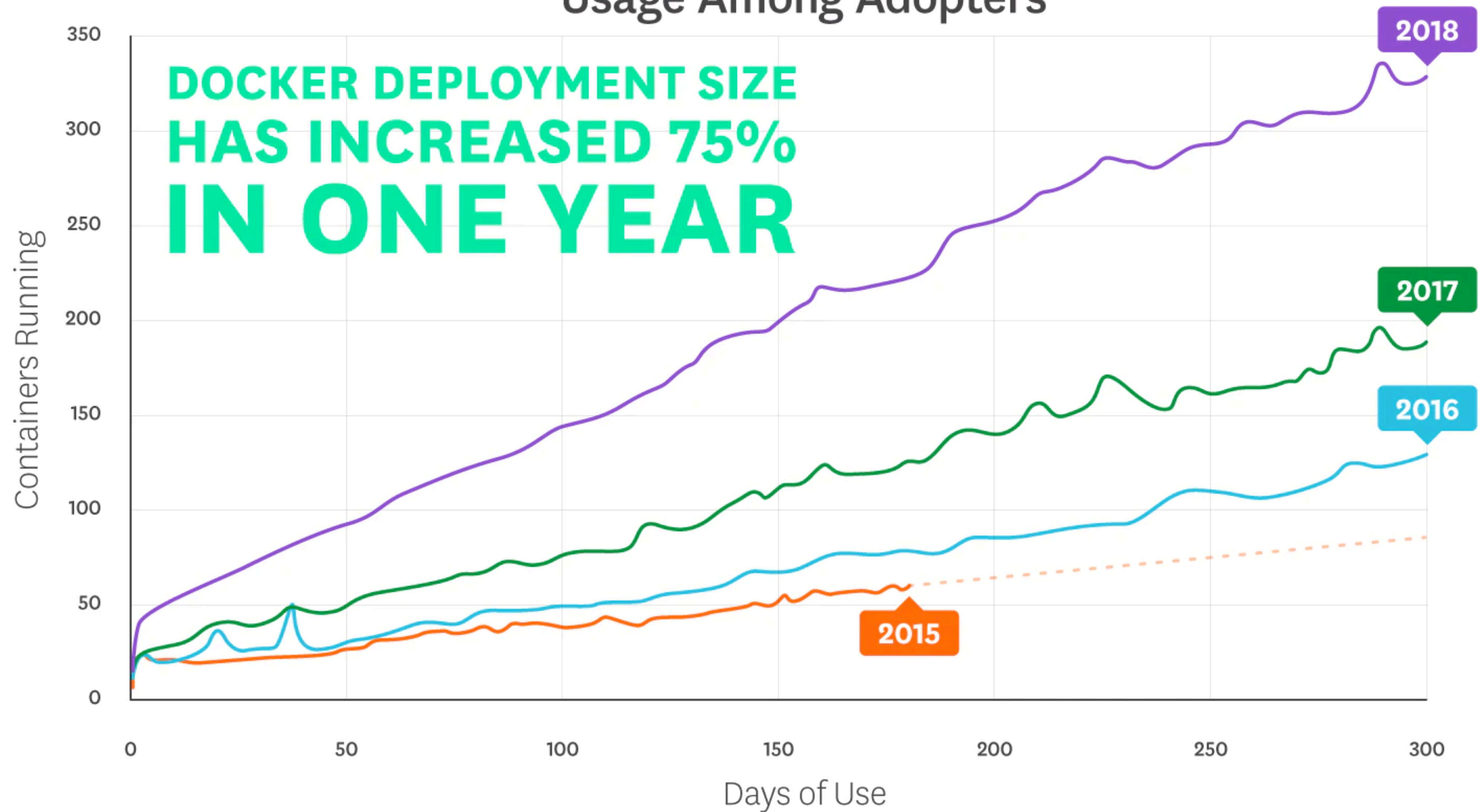
Source: Datadog

Portion of Hosts Running Docker



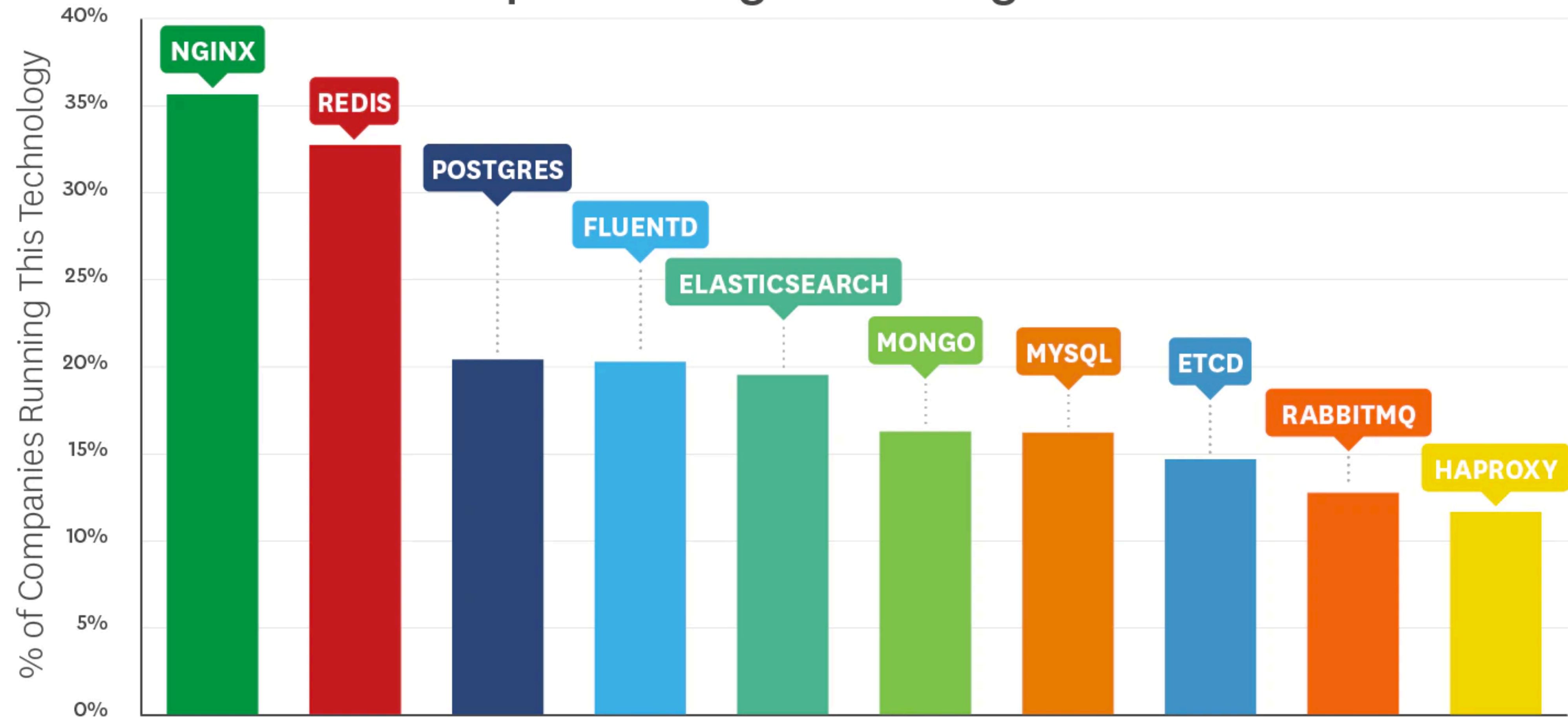
Source: Datadog

Usage Among Adopters



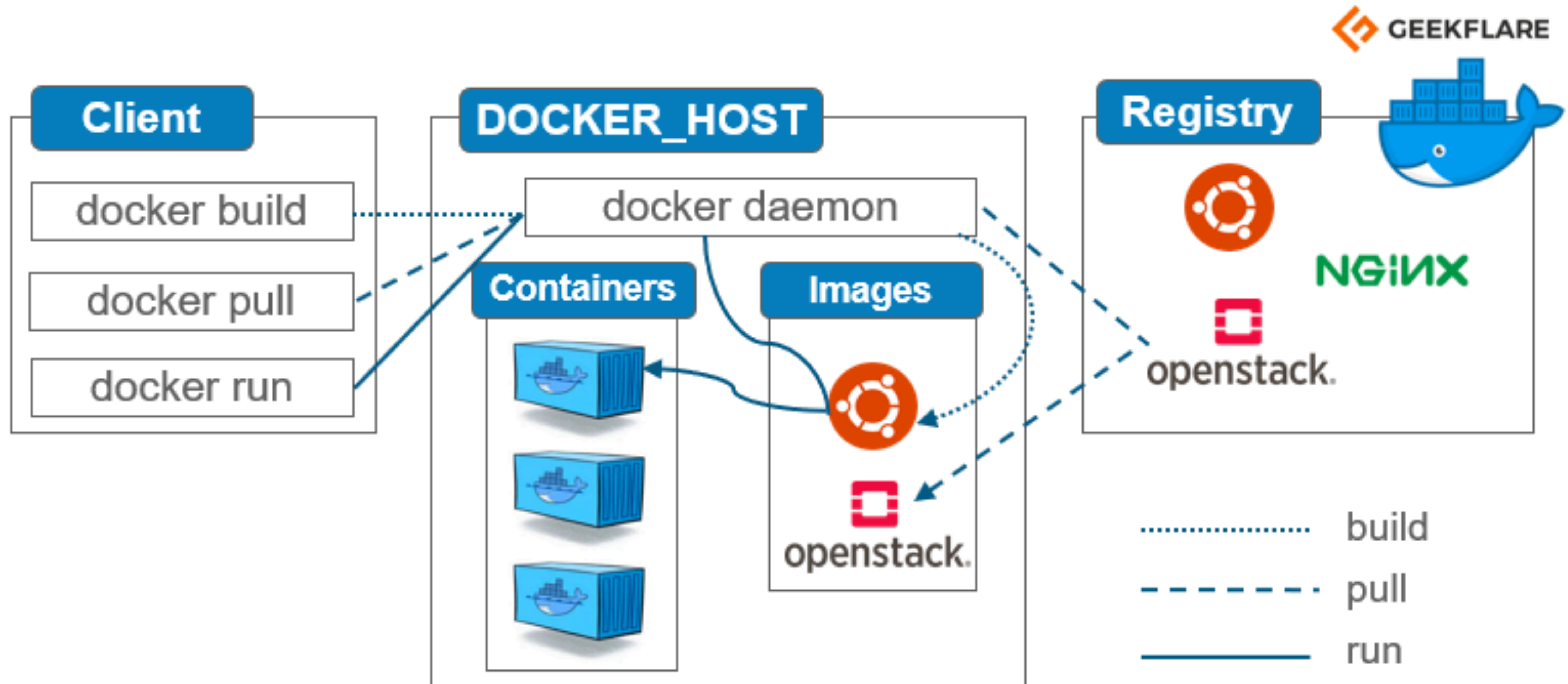
Source: Datadog

Top Technologies Running on Docker

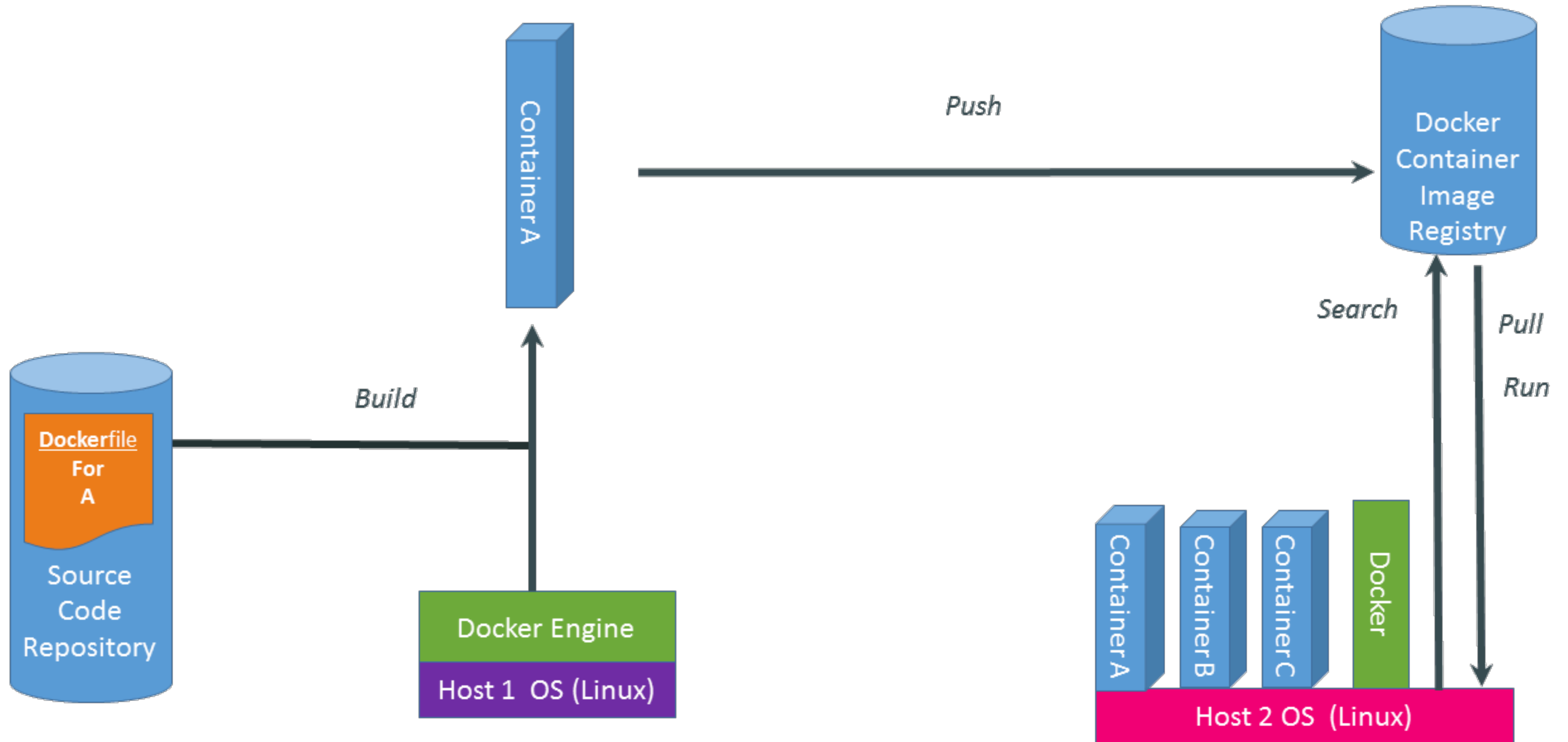


Source: Datadog

Docker Architecture

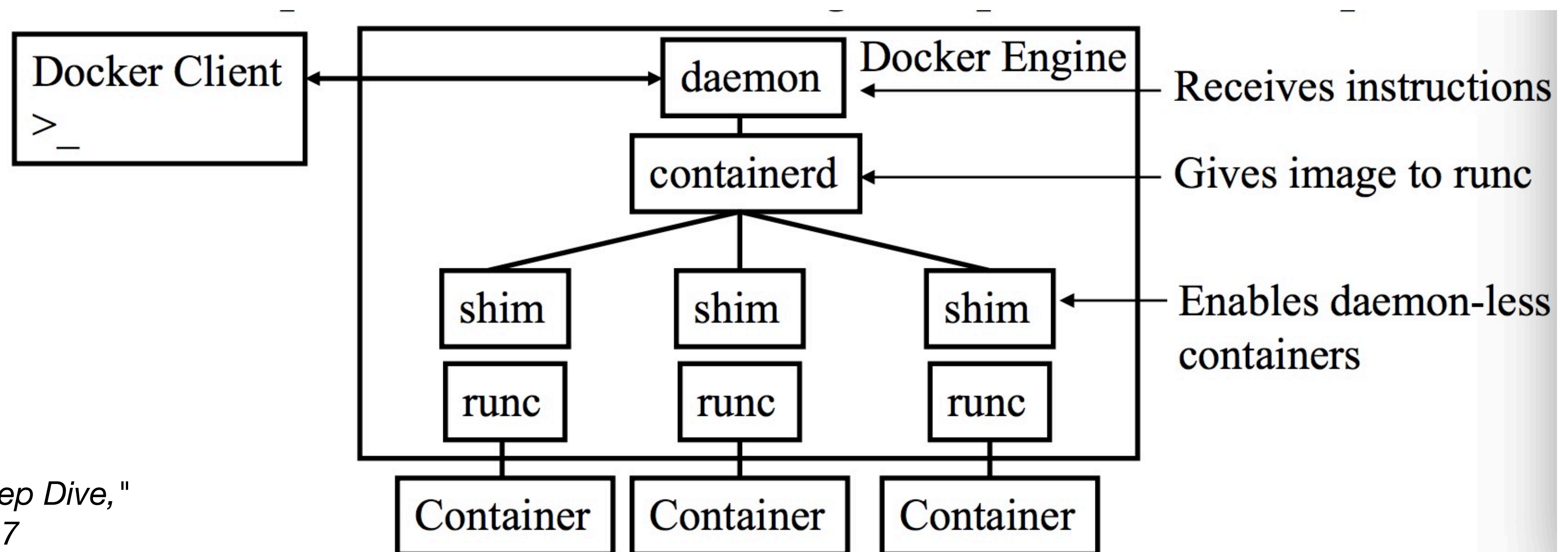


What are the Basics of a Docker System?



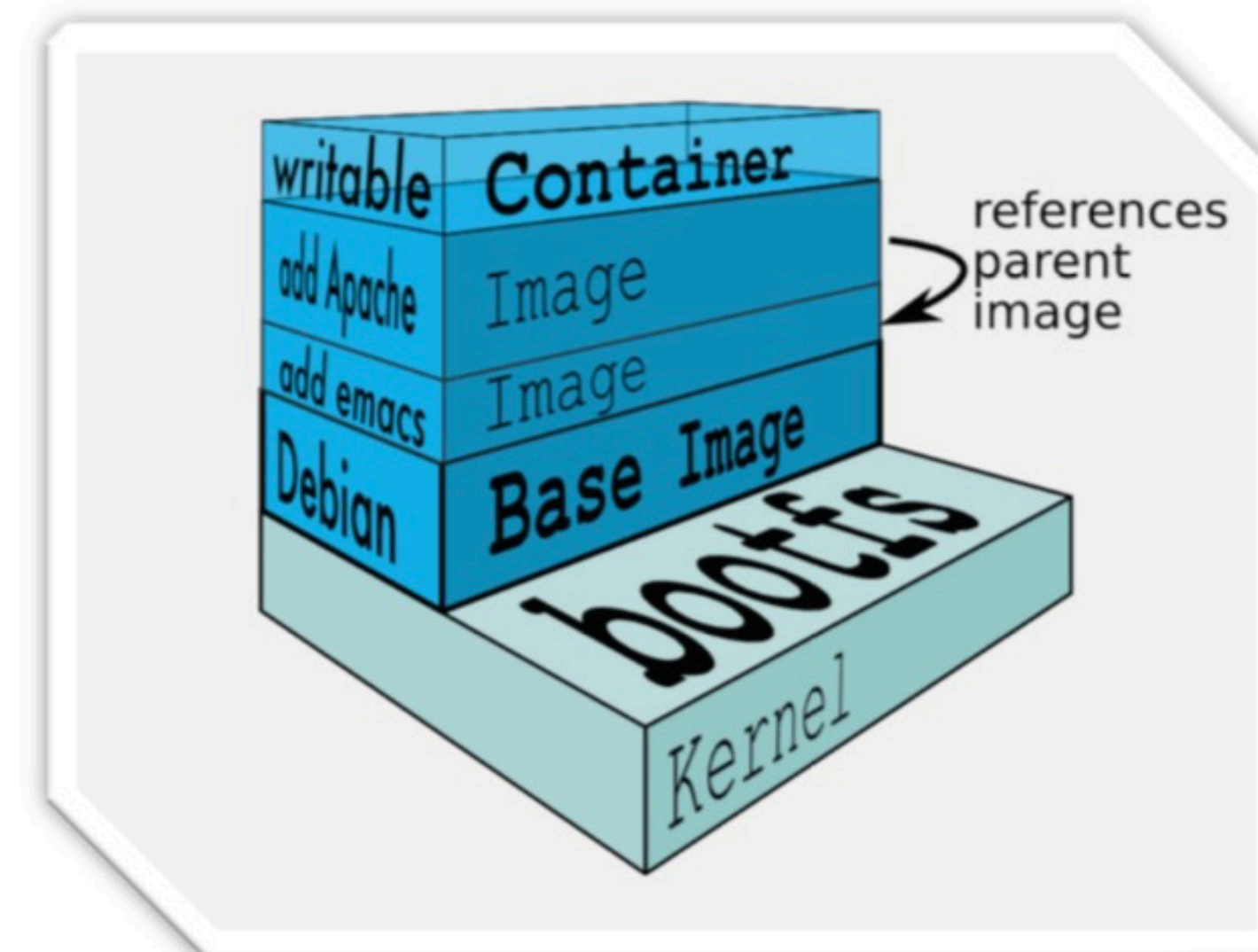
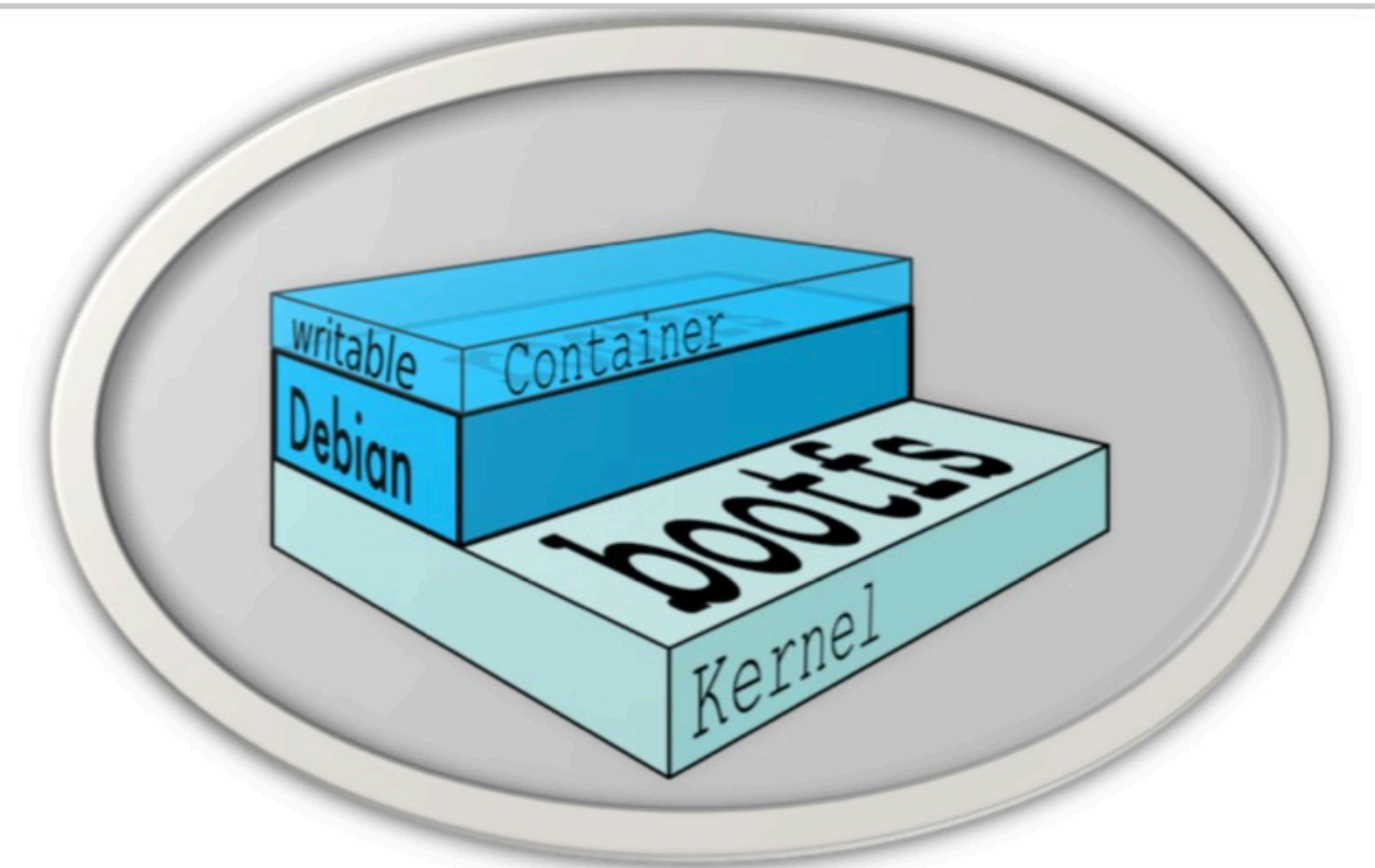
Docker Engine

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



Docker Images

- not a VHD, not a file system
- uses a Union File System to “overlay”
- a read-only Layer
- do not have state
- Basically a tar file
- Has hierarchy (arbitrary depth)



overlay filesystems

Docker images can be really big



that Anaconda image is 1.5 GB, there must be a lot of Python packages installed in it!

every container needs a "copy" of its image



container

I'm installing new programs on top of this Ubuntu 16.04 image

me too!



container

solution: ★ overlays ★

container's changes

/etc/apt/sources.list
/files/something.txt

merged result

/bin/bash /bin/cat
/etc/apt/sources.list
/files/something.txt

base OS (1GB)

/bin/bash /bin/cat
/etc/apt/sources.list



containers can use the same base image without wasting disk space on copies!

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.

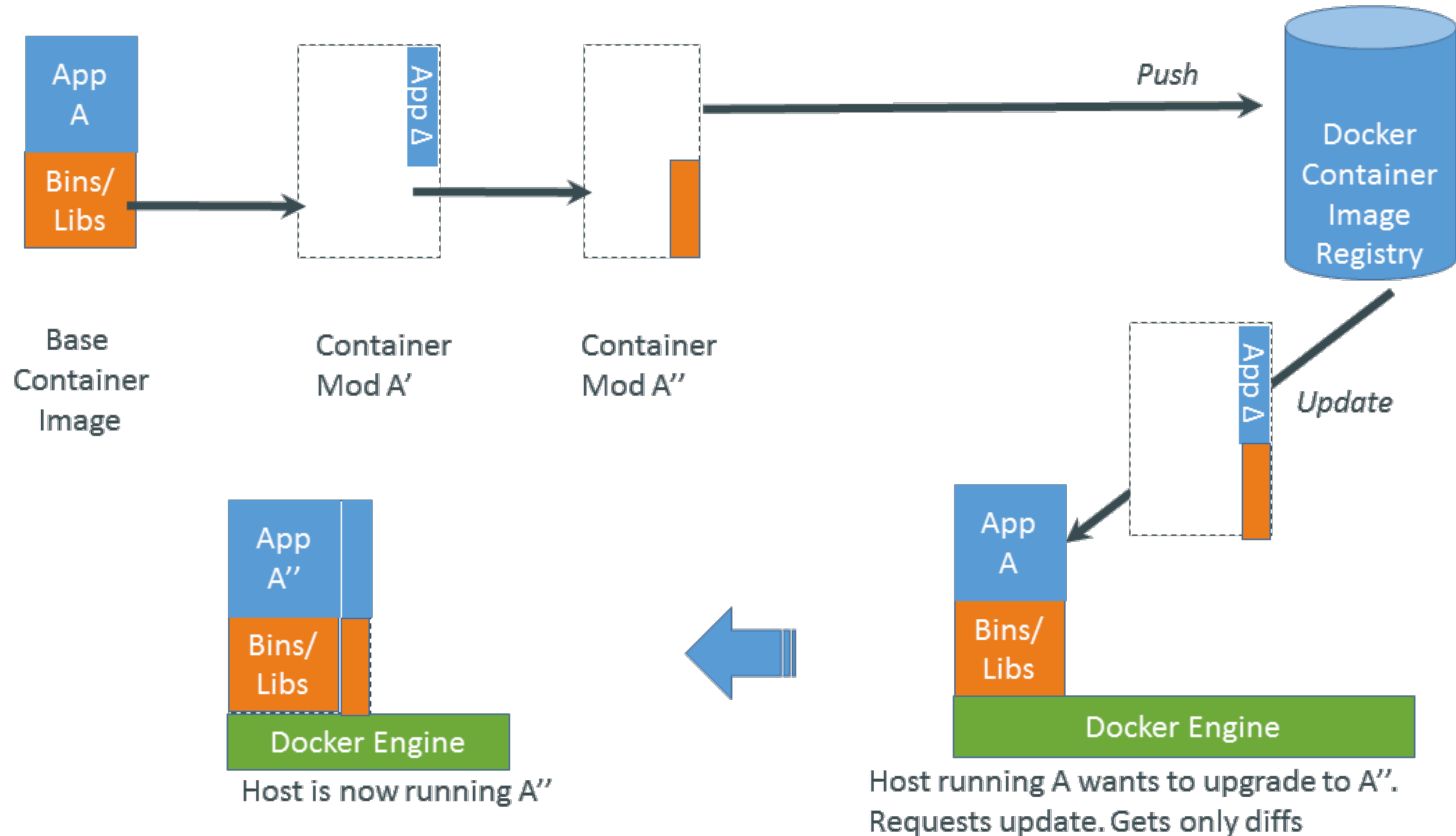
how Docker runs containers

- 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!

Docker Image Registry

- Registry containing docker images
 - Local registry on the same host
 - Docker Hub Registry: Globally shared
 - Private registry on [docker.com](https://hub.docker.com)

Changes and Updates



Threats of Containers

- Unlike VMs whose interface is hardware instructions, containers' interface is OS system calls
- More difficult to protect syscalls
 - Involve large amount of code in the OS
 - And there are many syscalls

Security Implications of Containers

TABLE 1. Threat model specifications for apps, containers, and host for the studied use cases. ‘Semi’ refers to semi-honest. Apps in semi-honest/malicious containers can be semi-honest or malicious too.

Use Case	Apps can be			Containers can be			Host can be		
	honest	semi	malicious	honest	semi	malicious	honest	semi	malicious
(I) Protect container from applications	✓	✓	✓	✓	-	-	✓	-	-
(II) Inter-container protection	✓	-	-	✓	✓	✓	✓	-	-
(III) Protect host from containers	✓	✓	✓	✓	✓	✓	✓	-	-
(IV) Protect containers from host	✓	-	-	✓	-	-	✓	✓	✓

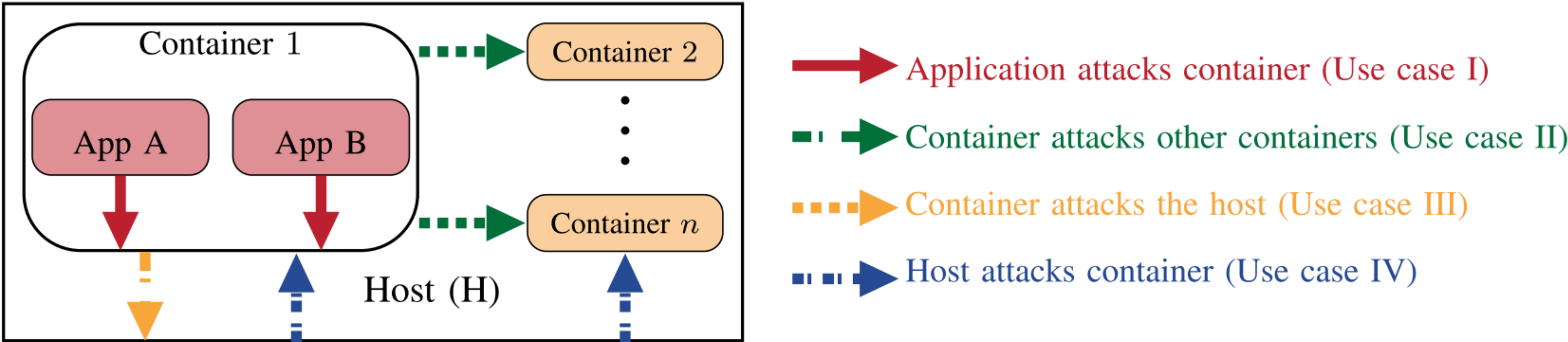


FIGURE 3. Overview of security protection requirements in containers.

Source: S. Sultan et al.: Container Security: Issues, Challenges, and the Road Ahead

Threats of Container Images

- Difficult to understand the source/provenance of images

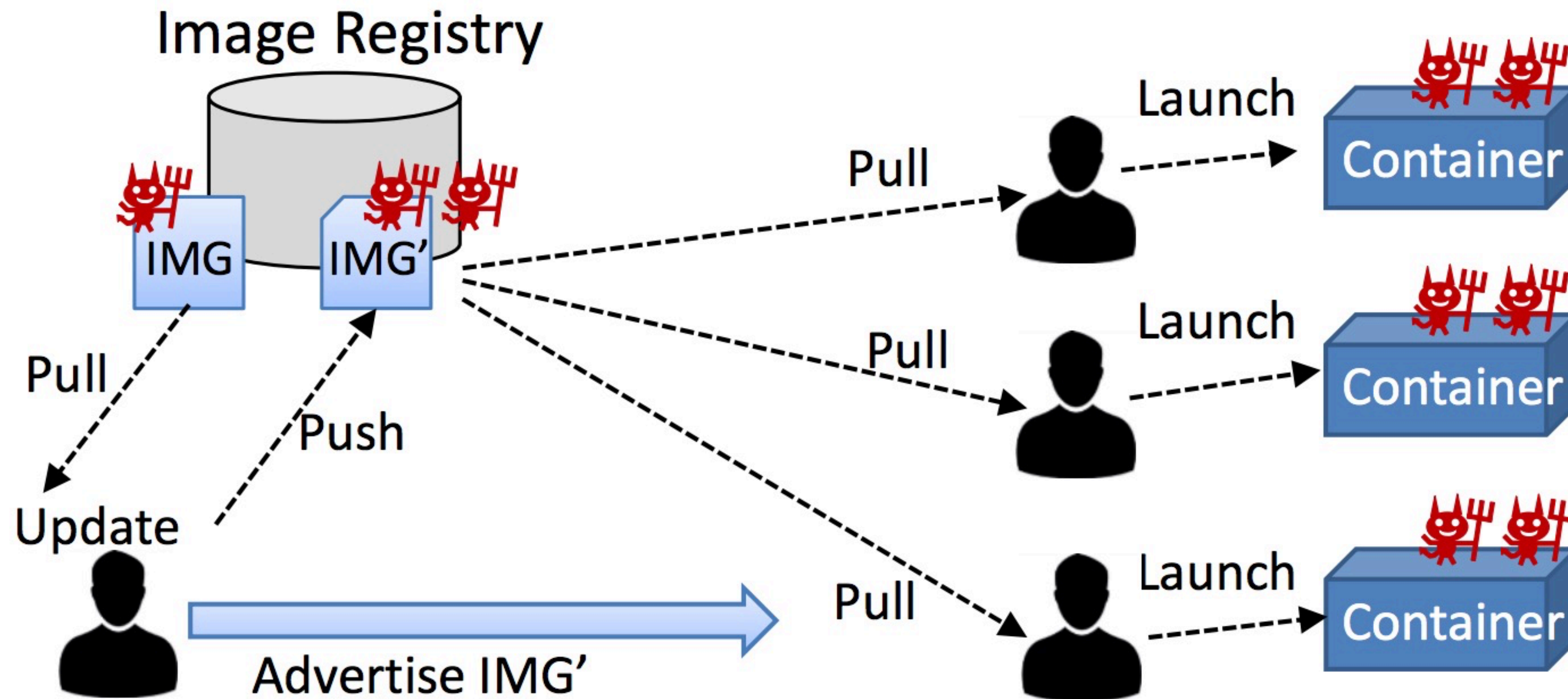


Figure 1: Scenario of vulnerability spread