Are VM's Passé?

Ben Golub
CEO Docker, Inc.

docker

docker

They told me that I needed a provocative title for this talk.

So…

docker

Q: Are VMs Passé?

docker

A: No

docker

Thank you for attending

docker

A different question…

- Q: Is there a better alternative for many use cases & environments?
  - application management & creation?
  - application deployment across clusters & clouds?
  - CI & CD?
  - scale out?
  - high performance?
  - collaborative development?

A: Yes

docker

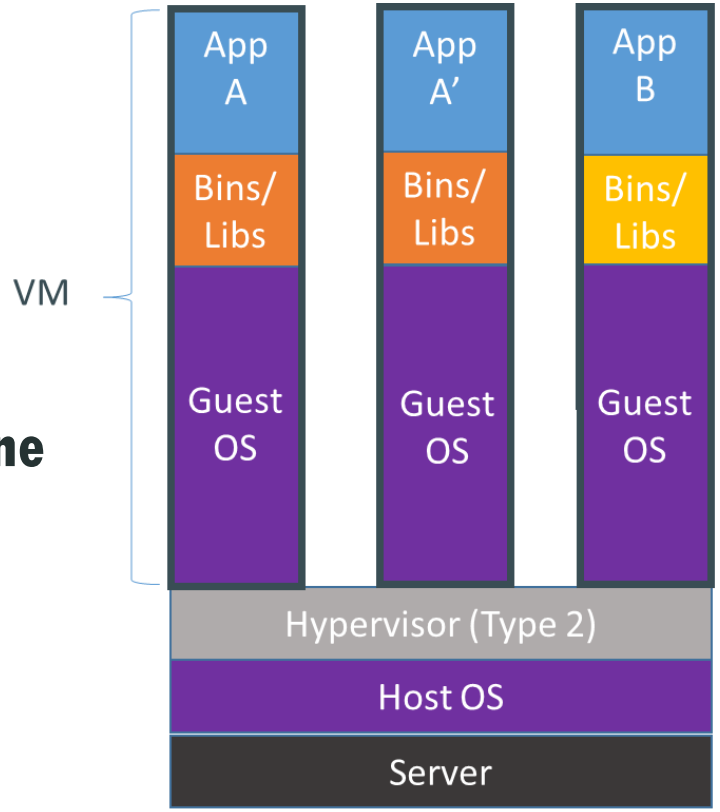
Agenda

- Server Proliferation and the rise of the VM
- The Matrix from Hell and the need for lightweight, interoperable containers
- Step 1: Making lightweight containers:
  - Containers vs. VMs: how they work
- Step 2: Making containers really interoperable:
- Step 3: Making containers *really* lightweight
- Step 4: Creating a container-based system for app mgt & deployment
- Step 5: Creating an ecosystem around containers
- Where to Use VMs vs. Containers/Docker
- Learn more

docker

Some ancient history: Where did VM's come from?

- **PROBLEM IN 2000: Server Proliferation/Consolidation**
  - I've got a single purpose, physical Microsoft Exchange Server, Mac print server, and Custom Unix inventory server
  - Machines are getting more powerful
  - I want to consolidate all those single purpose, physical servers onto a single server
  - BTW--It takes too damn long to provision a physical machine

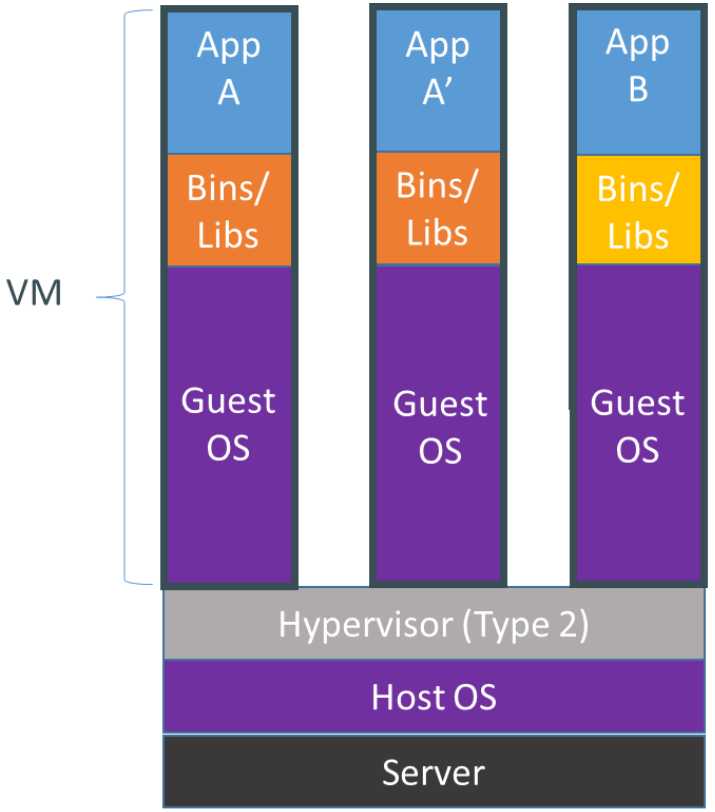


- **ANSWER: Create a Virtual Machine**

docker

Results

- Single purpose physical application servers become single purpose virtual servers
- Provisioning a “server” goes from days/weeks to minutes
- Huge cost savings
- An awesome solution to the server consolidation problem
- An awesome solution for creating flexible infrastructures
- Mature ecosystem and tool set for isolation, security, management



docker

What has changed since the VM was developed?

2000	2014
Apps are long lived	Development is iterative and constant
Apps are monolithic and developed on a single stack	Apps are created from loosely coupled components, themselves created from a multitude of “stacks”
Deployment is to a single server	Deployment is to a variety of servers: VM, physical, cluster, open stack, public cloud, +++

- **Result: An application isn’t easily represented or managed as a single purpose server (whether physical or virtual)**

docker

The Problem in 2014

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

Contributor's laptop

Production Servers

Can I migrate smoothly and quickly?

docker

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	?	?	?	?	?	?	?

docker

An Inspiration…and some really ancient history: Cargo Transport Pre-1960

Multiplicity of Goods



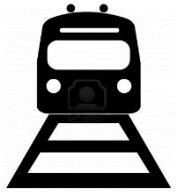




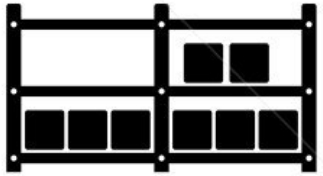
Do I worry about how goods interact (e.g. coffee beans next to spices)

Multipilicity of methods for transporting/storing









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

docker

Solution: Intermodal Shipping Container

Multiplicity of Goods





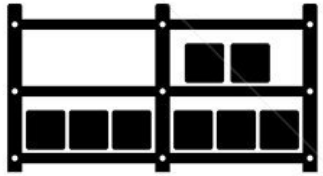


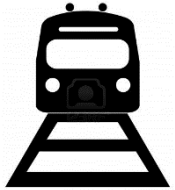
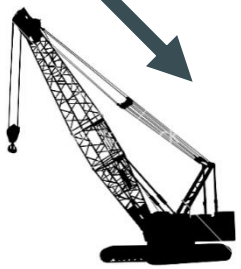
Do I worry about how goods interact (e.g. coffee beans next to spices)

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

Multiplicity of methods for transporting/storing

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

docker

This spawned an 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 globalization

- 5000 ships deliver 200M containers per year

docker

Let's create a shipping container system for code

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

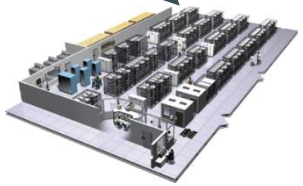
Can I migrate smoothly and quickly

Multiplicity of hardware environments

Development VM

QA server

Customer Data Center



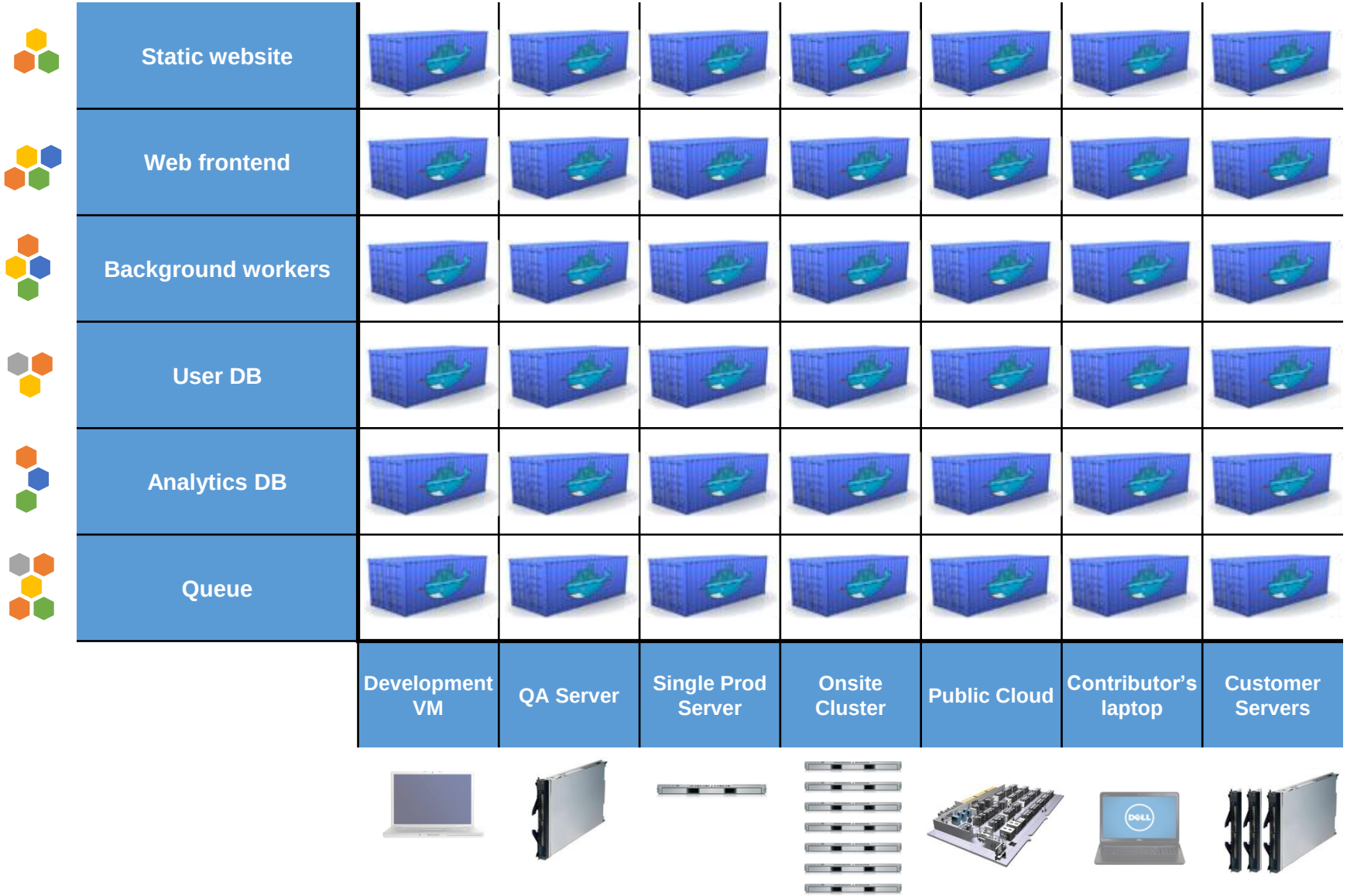
Public Cloud

Production Cluster

Contributor's laptop

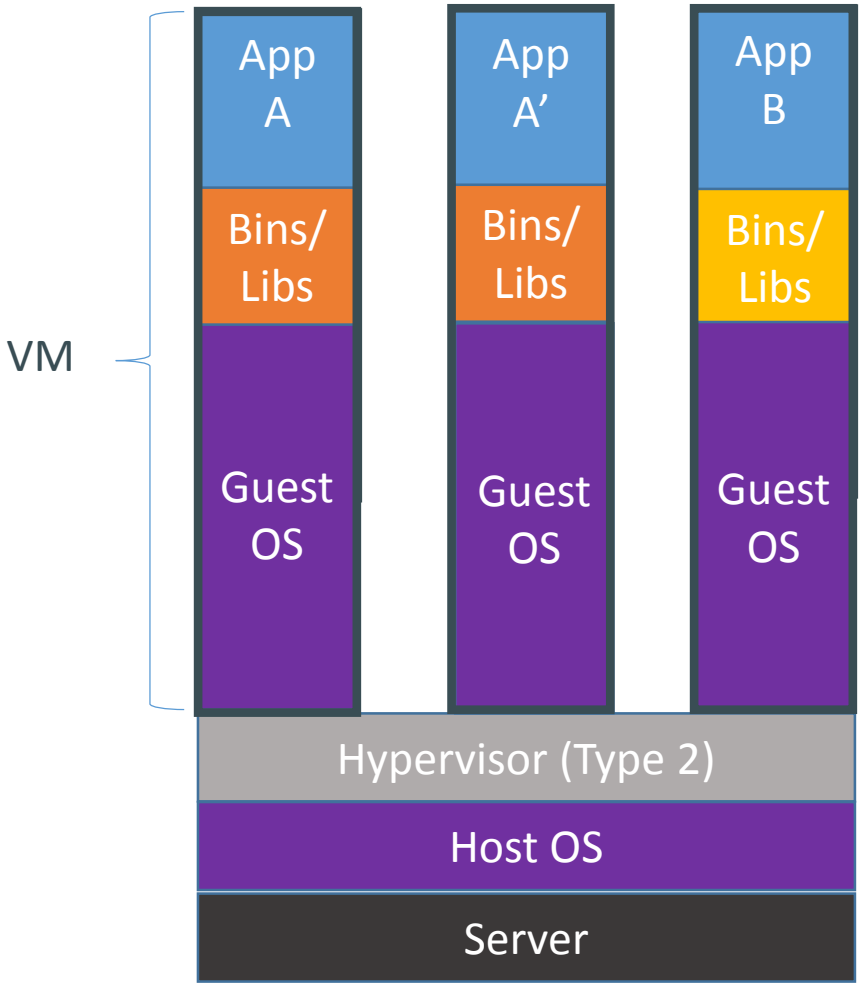
docker

Eliminate the matrix from Hell



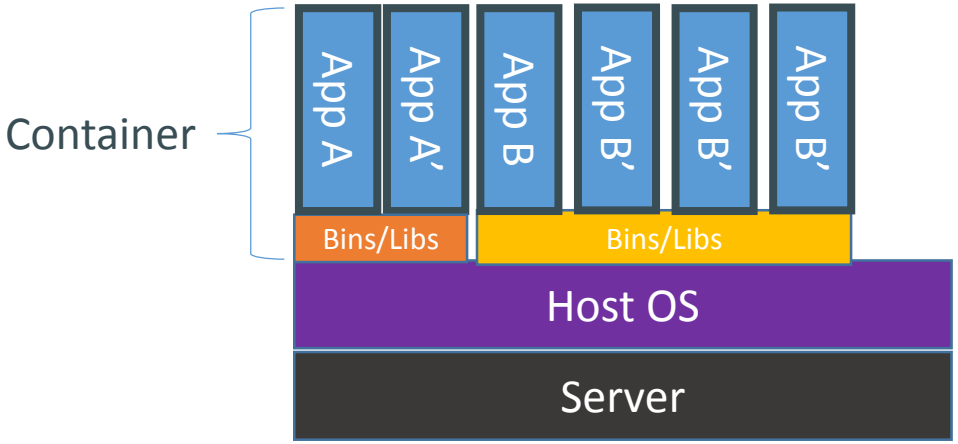
docker

Step One: Create a lightweight container (vs. VMs)



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

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

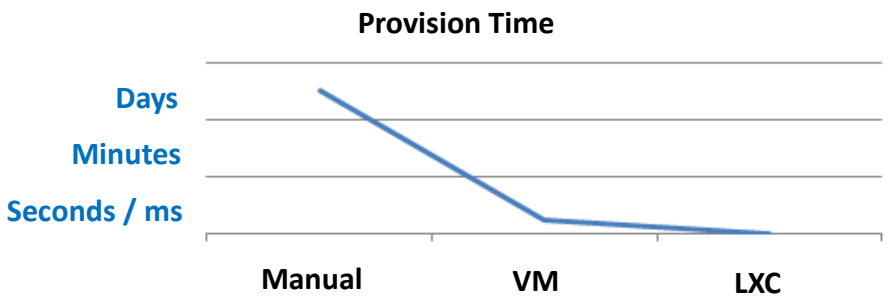


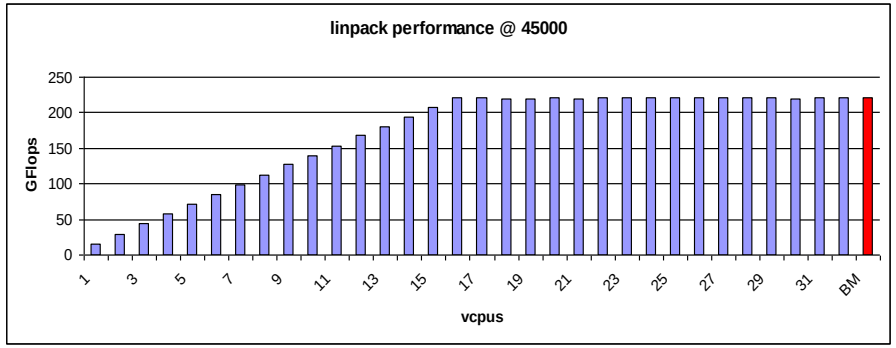
docker

A great slide stolen from IBM: Why Containers?

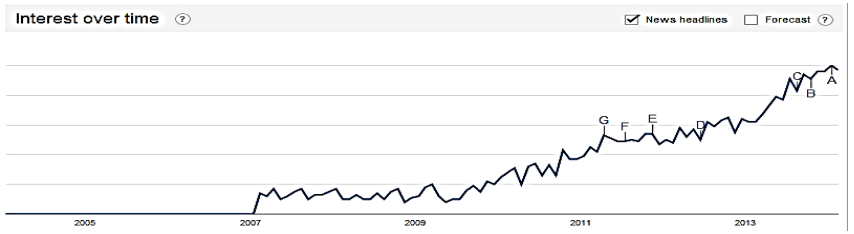
"Containers as poised as the next VM in our modern Cloud era…"

- **Provision in seconds / milliseconds**
- **Near bare metal runtime performance**
- **10 x greater density**
- **VM-like agility – it's still "virtualization"**
- **Flexibility**
  - Containerize a "system"
  - Containerize "application(s)"
- **Lightweight**
  - Just enough Operating System (JeOS)
  - Minimal per container penalty
- **Open source – free – lower TCO**
- **Supported with OOTB modern Linux kernel**
- **Growing in popularity**

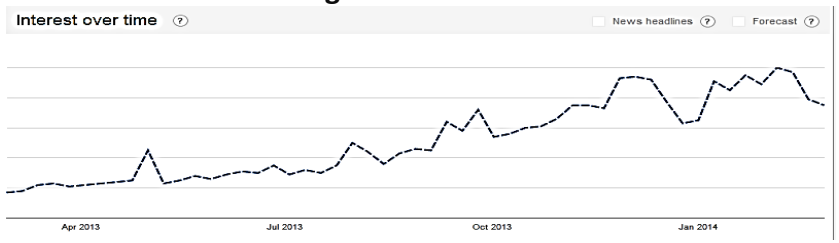




Google trends - *LXC*

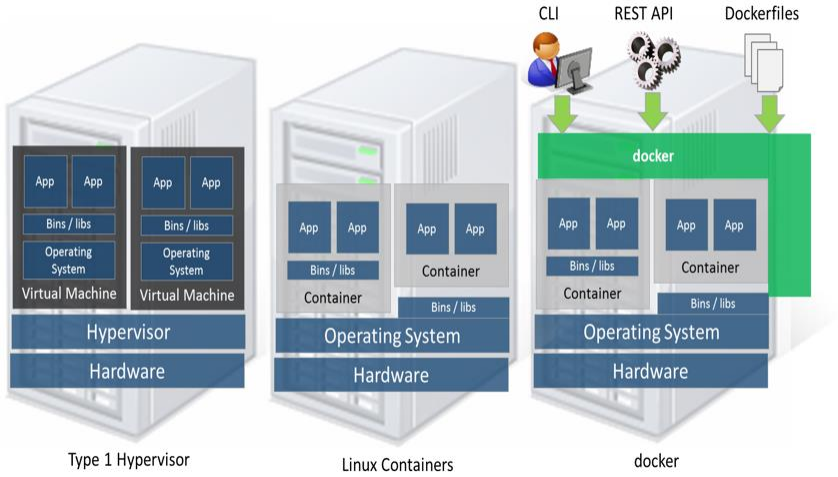


Google trends - *docker*



Step 2: Make the containers easy to use, standardized, interoperable, automatable

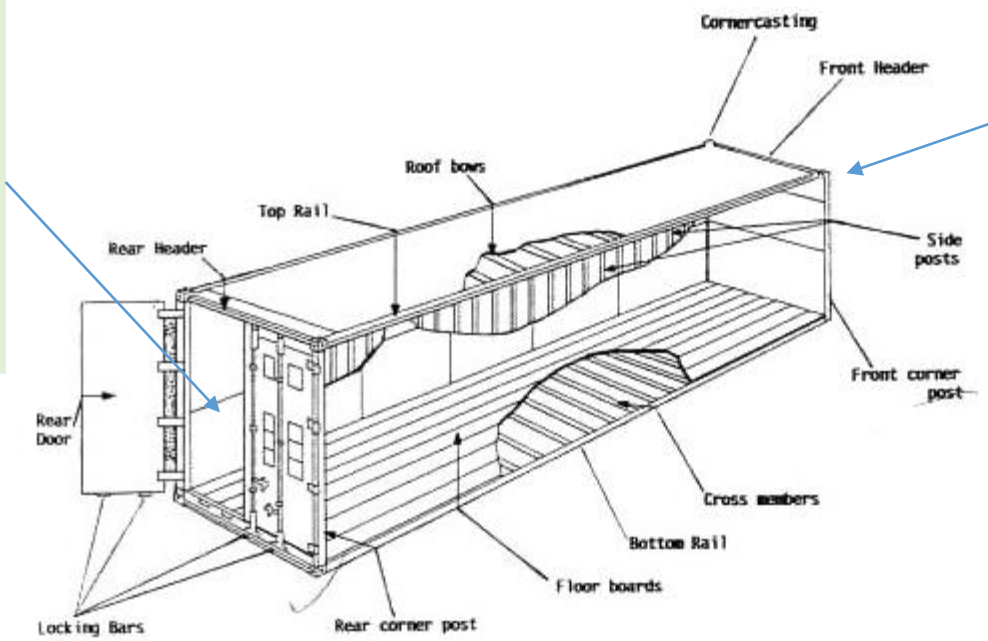
- Shipping containers are a standard size, and have hooks and holes in all the same places
- With Docker, Containers get the following:
  - Ease of use, tooling
  - Re-usable components
  - Ability to run on any Linux server today: physical, virtual, VM, cloud, OpenStack, +++
    - (Stay tuned for other O/S's)
  - Ability to move between any of the above in a matter of seconds-no modification or delay
  - Ability to share containerized components
  - Interoperability with all existing devops tools
  - Self contained environment—no dependency hell
  - Tools for how containers work together: linking, nesting, discovery, orchestration, ++
- You get ability to separate app management from infrastructure management



docker

Technical & cultural revolution: 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



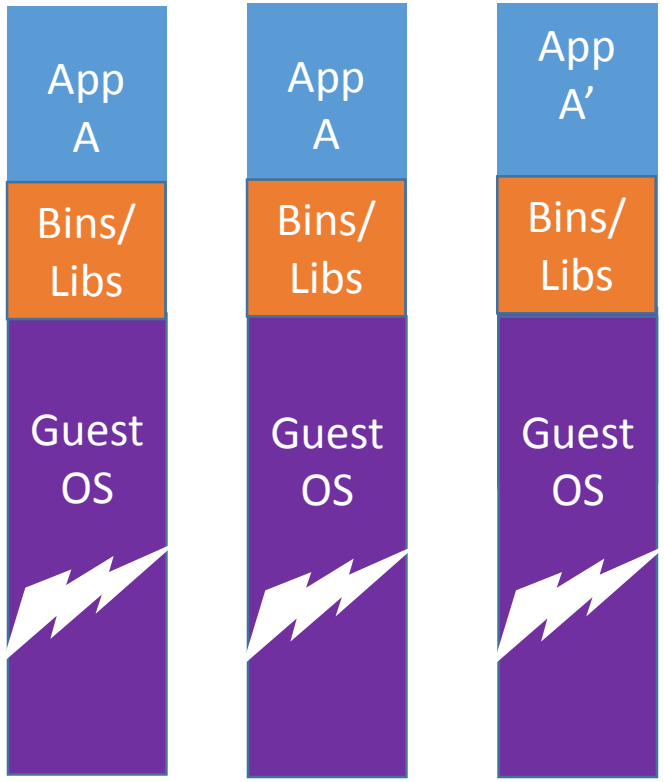
Major components of the container:

- **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

docker

Step 3: Make containers super lightweight

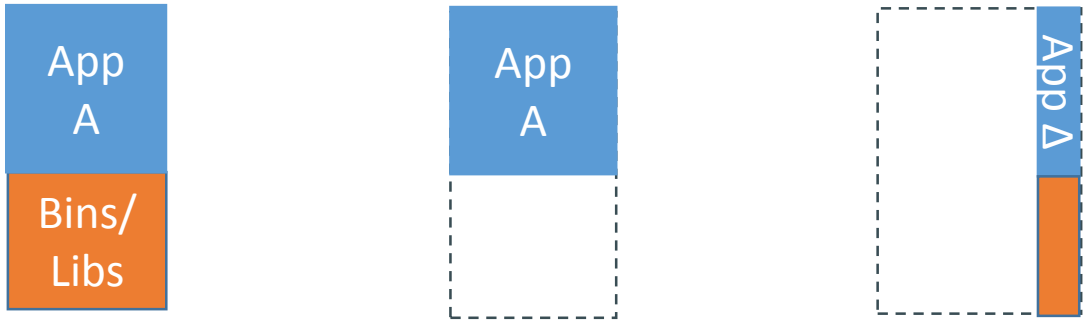
VMs



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

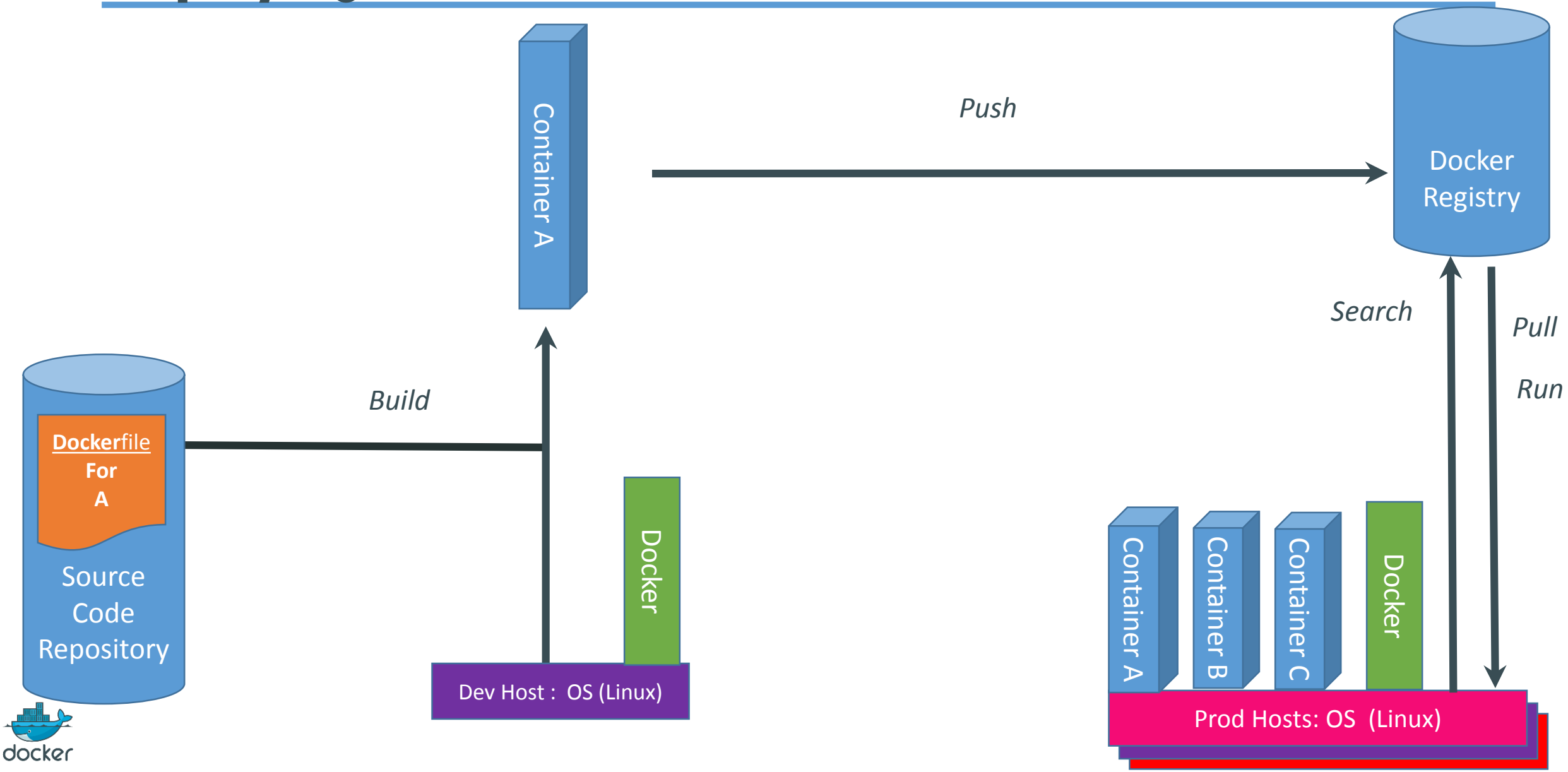
No OS. Can Share bins/libs

Modified App

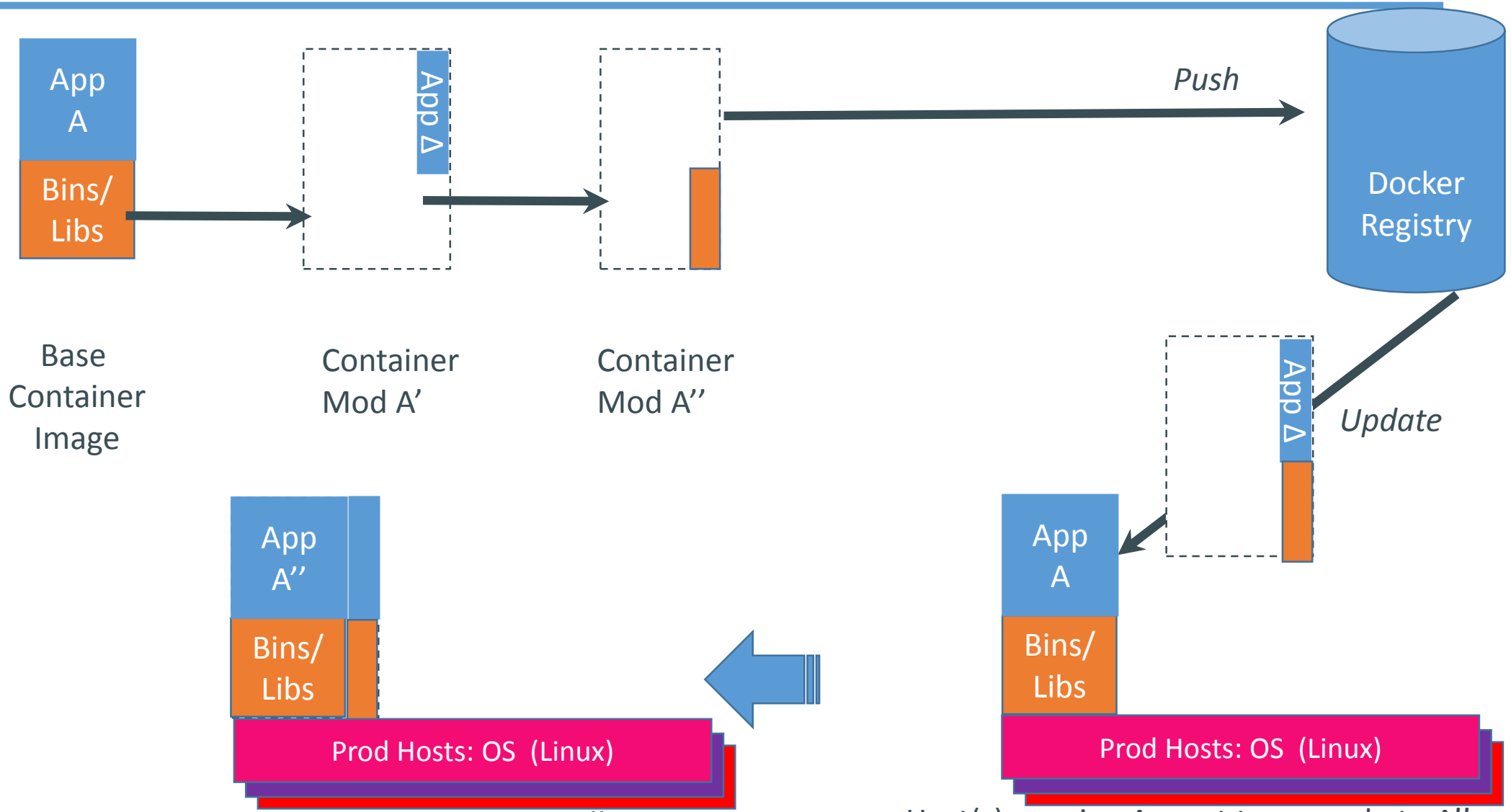
Copy on write allows us to only save the diffs Between container A and container A'

docker

Step 4: Build a System for creating, managing, deploying code



Including a System for Changes and Updates

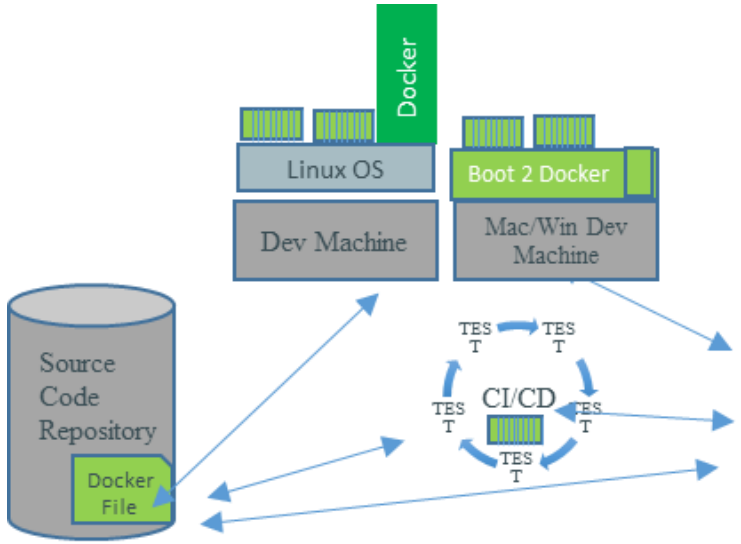


Hosts are now running A''

Host(s) running A want to upgrade to A''. Requests update. Gets only diffs

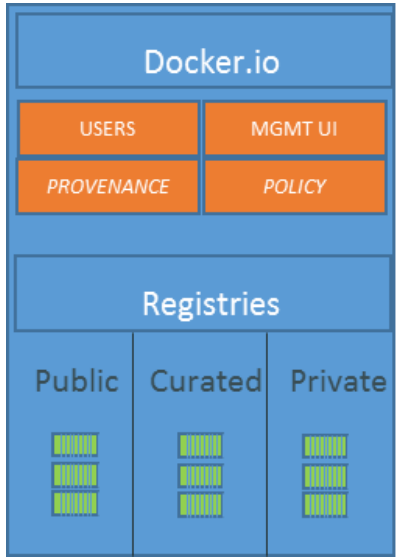
docker

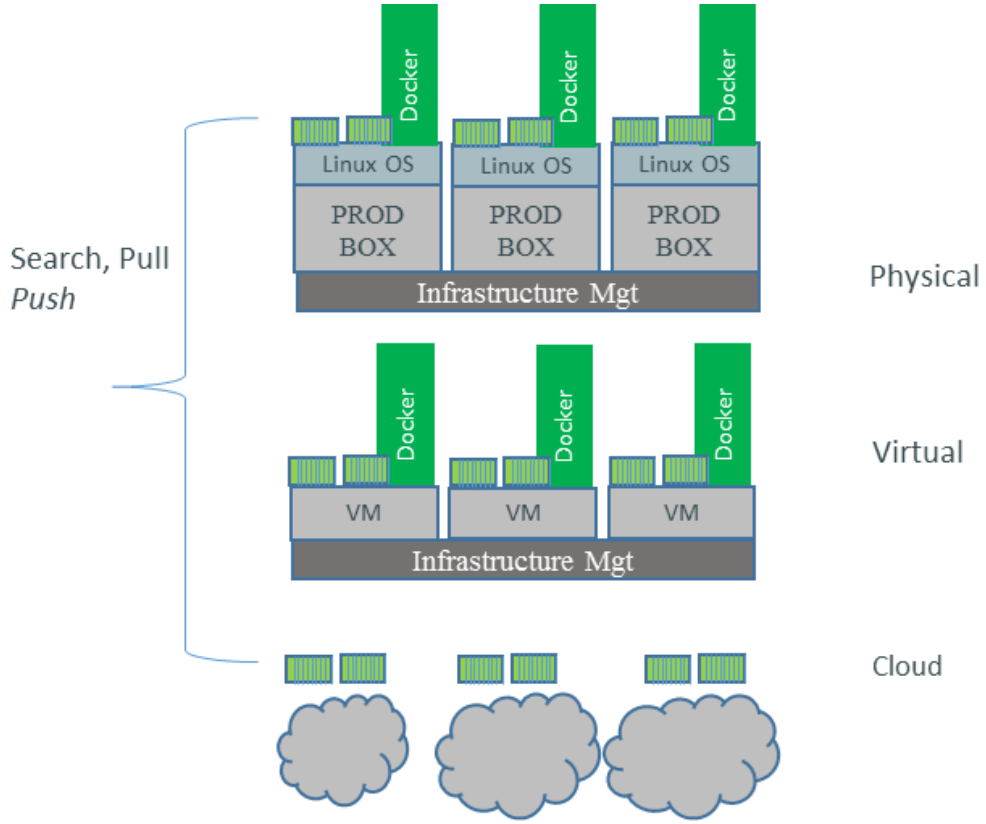
Including a System for the Full Lifecycle



APP CREATION

APP MANAGEMENT





APP DEPLOYMENT

docker

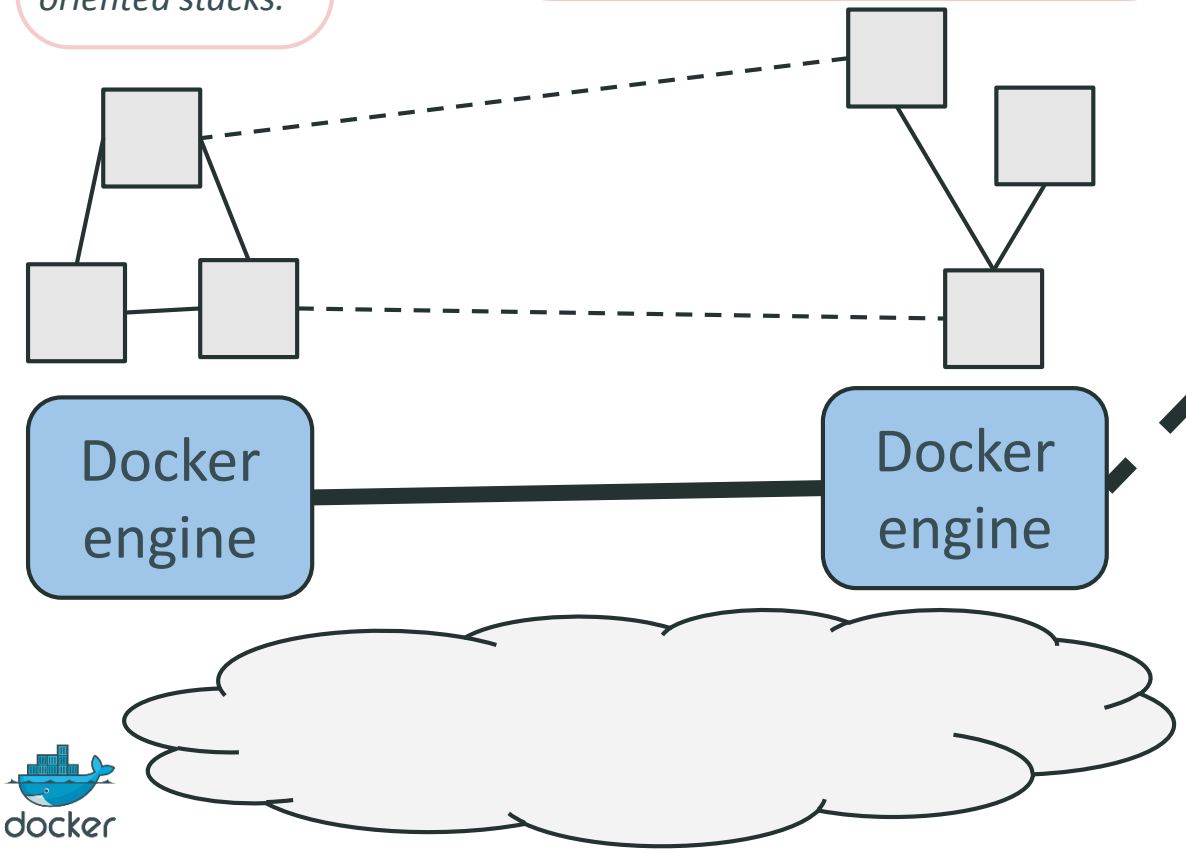
Including a System for Complex Apps

v0.7

Containers can be linked and assembled into complex service-oriented stacks.

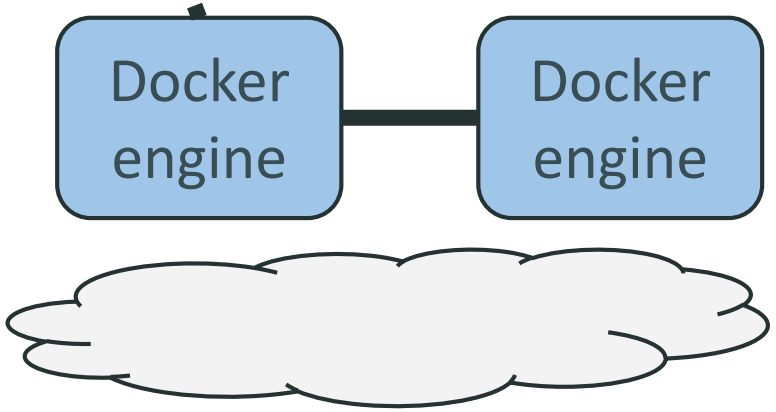
v0.9

Stacks can span multiple machines, using encrypted and authenticated tunnels.



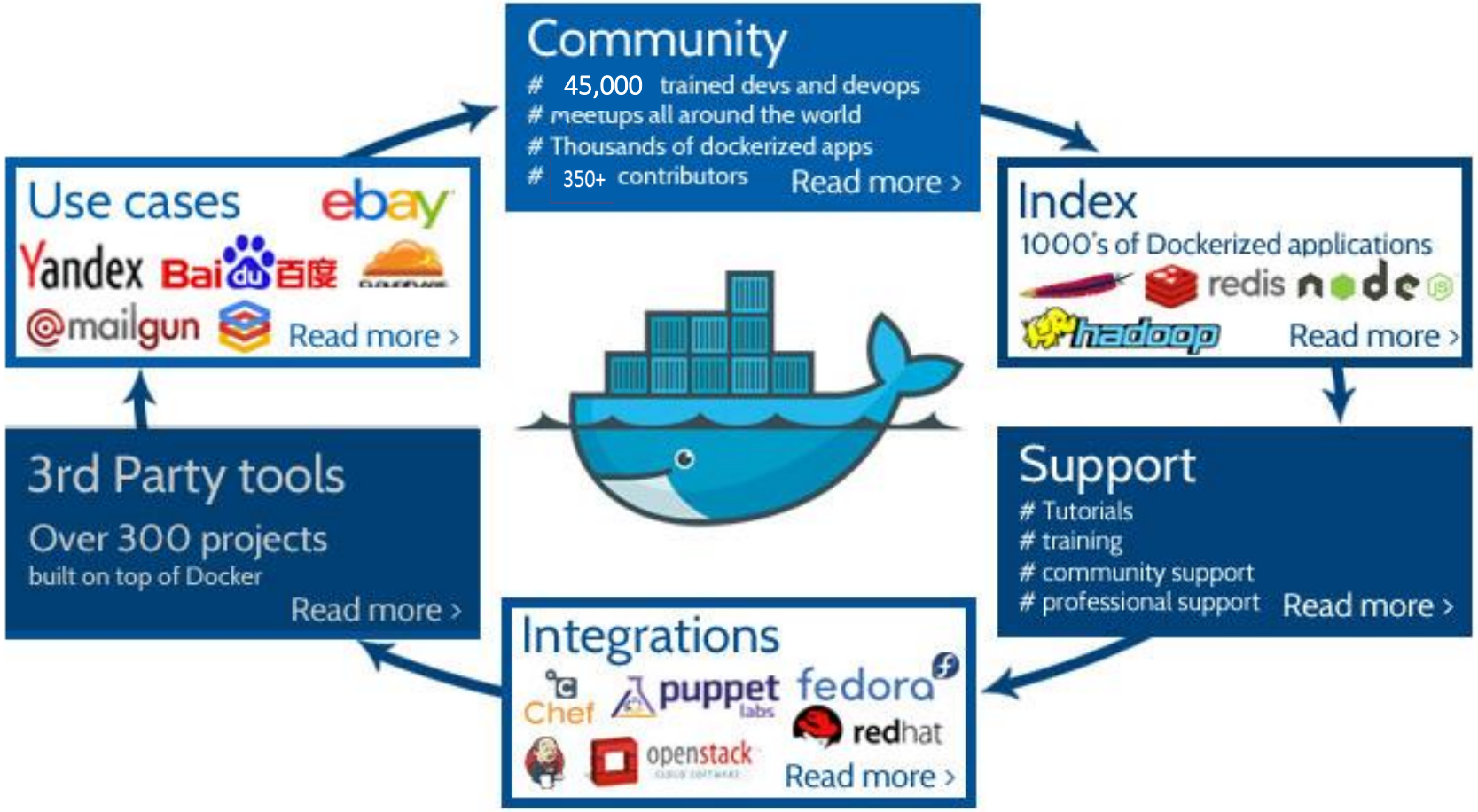
v0.really soon

*Docker deployments can span multiple datacenters and cloud providers by using the **docker.io** service. Docker.io acts as a hub and federates authentication, service discovery and orchestration across all docker engines across an organization*



docker

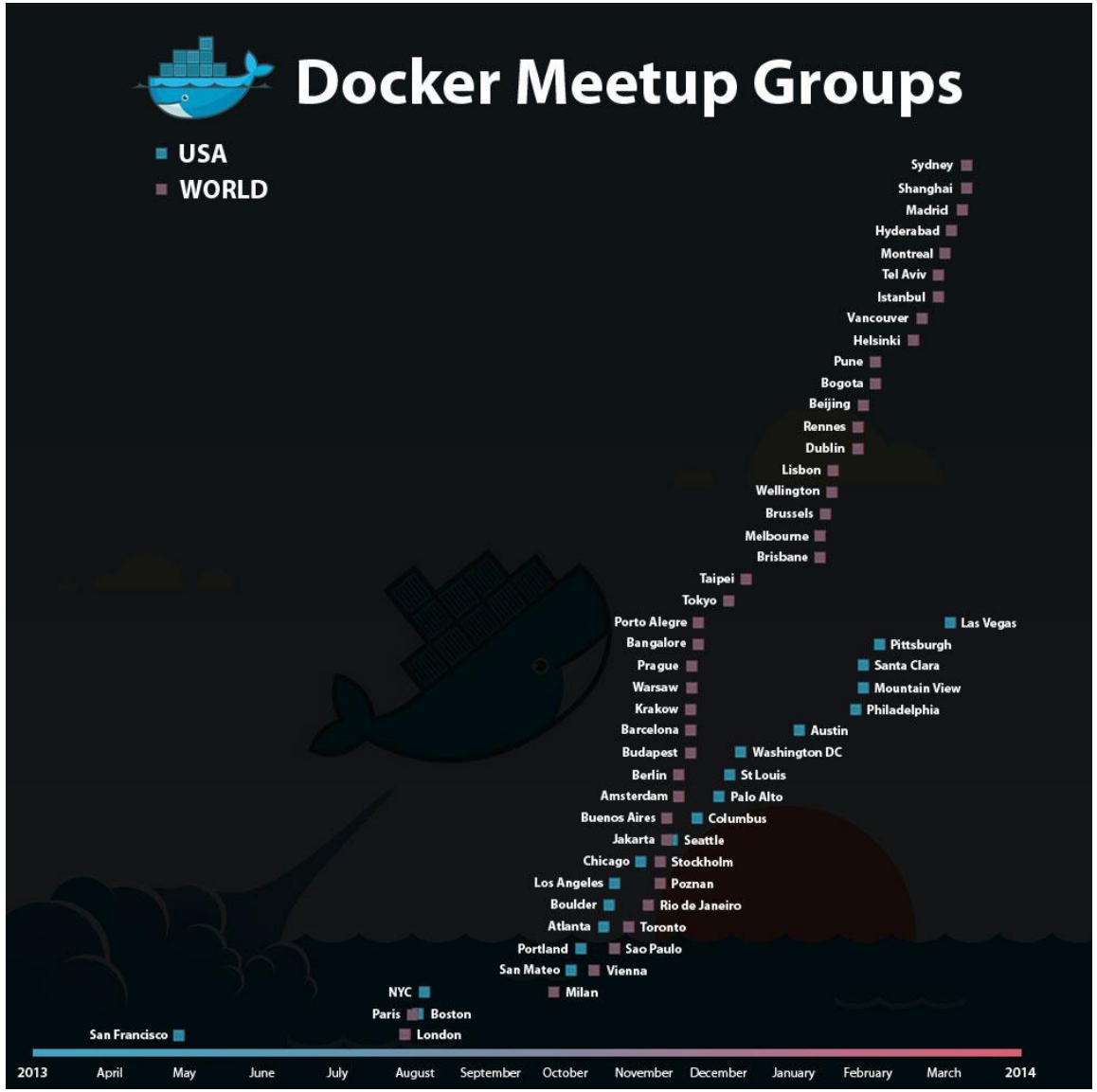
Step 5: Create an Ecosystem



docker

Open Ecosystem Momentum

- Truly open: Apache license, open design, open tooling, non-Docker maintainers
- Downloads: Over 1.2 m container downloads
- Users: Over 45,000 trained developers
- Content: Over 8000 repositories now publishing containers to Docker Index
- Contributors: 380 contributors, 95% of whom don't work for Docker, In.c
- Meetups: Over 80 cities in 30 countries have Docker meetups
- Integrations: OpenStack, RHEL, Ubuntu, Salt, Chef, Puppet, Salt +++
- Github
  - Over 10,000 stars
  - Over 1.7 K forks
  - Over 350 derivative projects



Who is using Docker?

… and hundreds of other small and big companies

Four major use cases

- Continuous Integration/Continuous Delivery:
  - Go from developer's laptop, through automated test, to production, and through scaling without modification
- Alternative form of virtualization for multi-tenant services
- Scale-out:
  - Rapidly scale same application across hundreds or thousands of servers…and scale down as rapidly
- Cross Cloud Deployment
  - Move the same application across multiple clouds (public, private, or hybrid) without modification or noticeable delay

docker

Where should I use VMs?

- VMs are definitely the way to go to solve many problems
  - Heterogeneous O/S families: Run Windows app on a Mac Server
  - Using O/S or kernel that doesn't support containers
  - Your real problem is infrastructure management
  - You want the maturity of the VM toolset
  - VM requires unique kernel setup which is not applicable to other VMs on the host (i.e. per VM kernel config)
  - Need to freeze state and live migrate
- But… you can pursue a hybrid strategy: containers on VMs
- Stay tuned for better Docker/Container answers for many of the above

docker

Conclusion

- Multiple forces are driving a reconsideration of how applications should be created, built, deployed, scaled, and managed
- We believe that the right approach is to decouple application management from infrastructure management
- Container based approach (vs. VM approach) provides right level of abstraction
- Enables infrastructure to be managed consistently and stably
- Enables applications to be built flexibly and deployed flexibly
- Provides greater degree of visibility, control, and management of what runs where and what components are allowed
- Massive cost, speed, efficiency savings
- Docker is becoming the standard for containerization

docker

Learn More

- LXC Technical discussion: slideshare.net/BodenRussell/realizing-linux-containerslxc
- Docker project: www.docker.io/
- Follow Docker on Twitter: twitter.com/docker
- Take the Docker interactive tutorial: www.docker.io/gettingstarted/
- Join Docker on IRC: botbot.me/freenode/docker/
- Go to the Docker repository on GitHub: github.com/dotcloud/docker/
- Go to a meetup: www.docker.io/community/#Docker-Meetups
- See what others are doing: www.docker.io/community/
- Come to DockerCon, Jun 9-10, San Francisco: www.dockercon.com

docker

Are VM's Passé?

Ben Golub
CEO Docker, Inc.

docker