

Z6110X0035: Introduction to Cloud Computing - Container-based Technology

Lecturer: Prof. Zichen Xu

Recap from Previous Class

Q&A

HW4 will be out after May Vacation!

Agenda

1.Intro to Containers
and how they enable
DevOps & CI-CD

2.What is Docker?

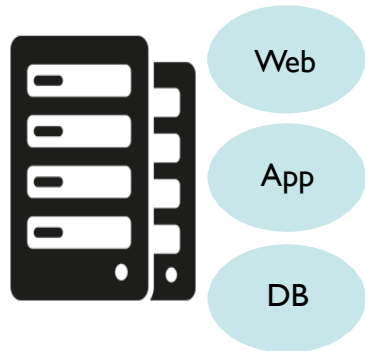
3.Persistent Storage
for Containers

4.Docker Data Center

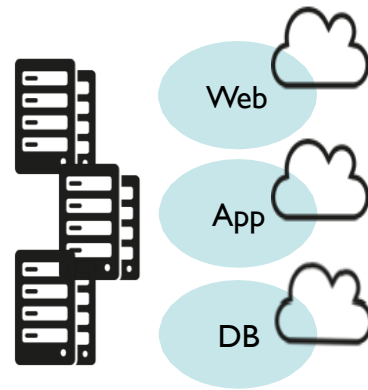
5.Docker Case
Studies

Application Deployment History

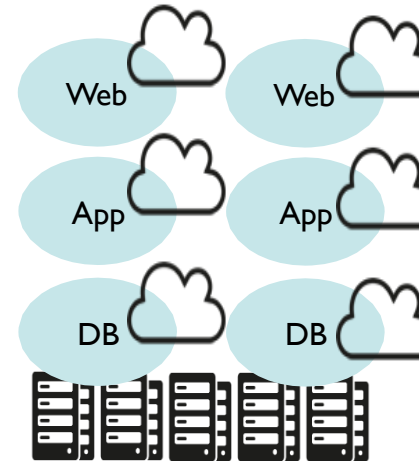
Monolithic Apps on Physical



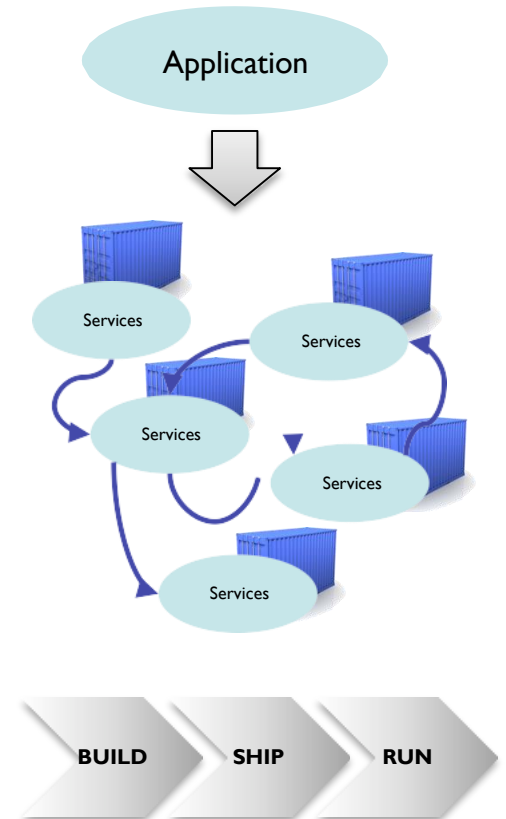
Virtual Machine Abstraction



Stateless & Horizontal Scalable Apps



Micro-services & Containers



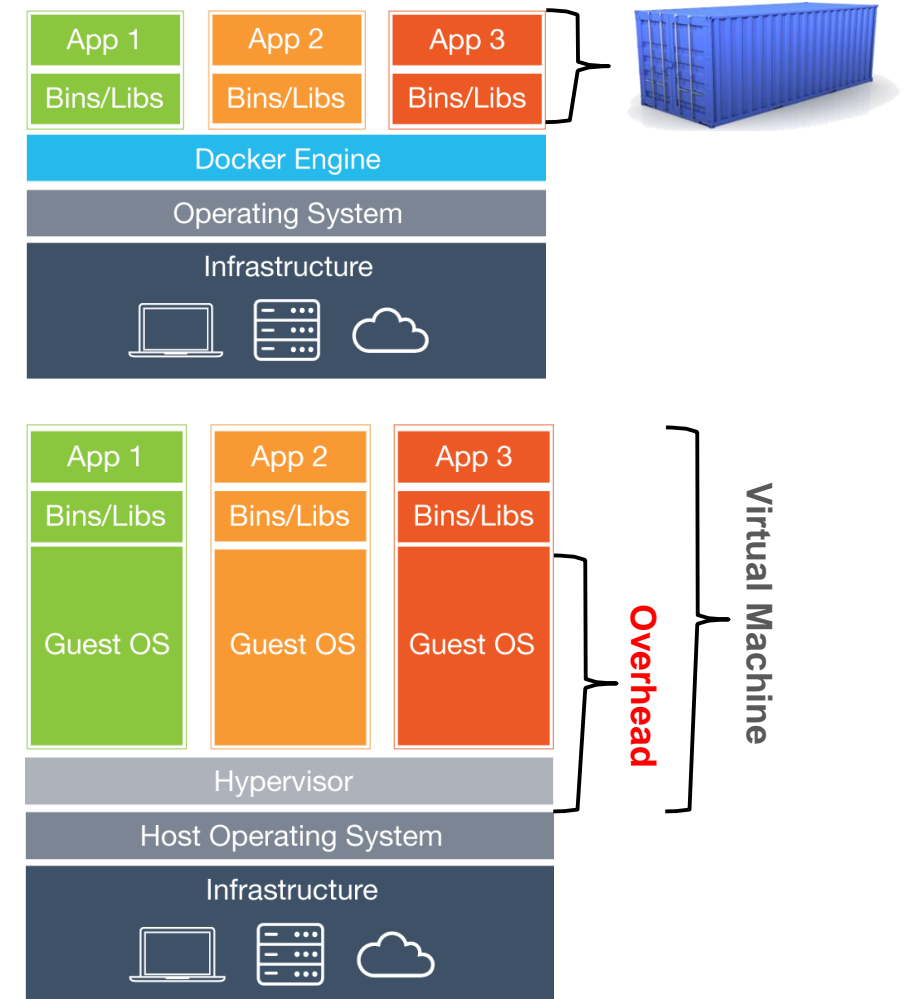
Containers 101

Containers

- Virtualization of application instead of hardware
- Runs on top of the core OS (Linux or Windows)
- Doesn't require dedicated CPU, Memory, Network—managed by core OS
- Optimizes Infrastructure—speed and density

“Containerization seems poised to offer both a complement and a viable alternative to server virtualization”

(1) IDC



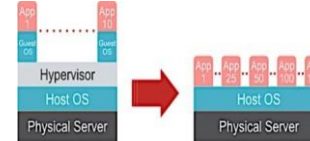
Containers vs VMs vs Bare-metal Servers

	Container	Virtual Machine	Bare-Metal x86 Server
Underlying Platform	OS on Virtual Machine or Bare-Metal x86 Server	Hypervisor on Bare-Metal x86 Server	N/A
Performance: Speed and Consistency	Average	Average	Fastest
Provisioning Time	Seconds	Minutes	Hours
Tenant Isolation Enforcement	OS Kernel	Hypervisor	Physical
Ideal Application Types	Mode 2	Mode 1 or Mode 2	Mode 1 or Mode 2
Configuration and Reconfiguration Flexibility	Highest	Medium	Lowest
Host Consolidation Density	Maximum	Average	None
Application Portability	Application Packaging/ Manifest*	VM Image, VM Migration Tools	Backup and Restore, ISO Images
Granularity	Extremely Small	Average	Largest
*While application portability is somewhat easier in container environments that are leveraging a container management and orchestration solution, portability should not assumed to be universal — differences in the underlying host OS below the containers could still present some interoperability challenges.			

Source: Gartner (September 2015)

Driving Factors for Containers

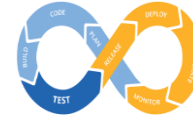
Density & Performance



Licensing Costs



Shift to DevOps



Cloud-native Applications
(Scale-out)

NETFLIX

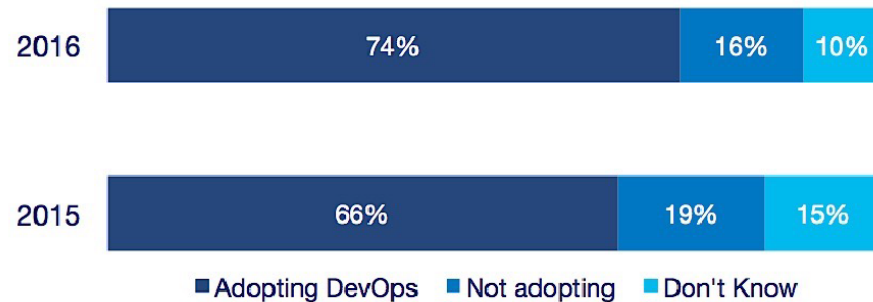
Faster Exploration & Deployment
(CI/CD)



“Containerization seems poised to offer both a **complement** and a **viable alternative** to server virtualization” - IDC

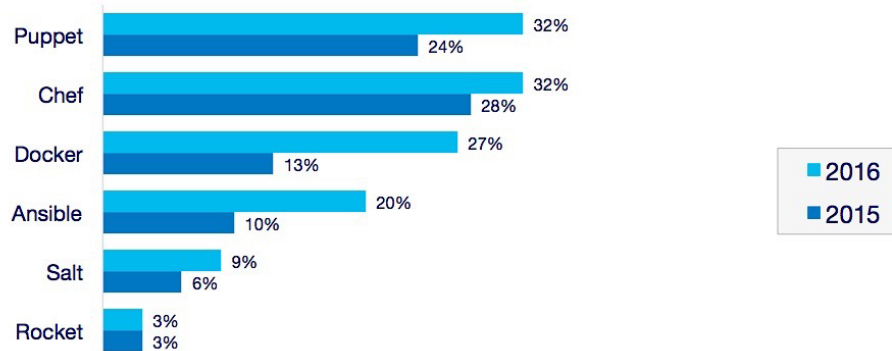
From Waterfall to DevOps

DevOps Adoption Up in 2016



Source: RightScale 2016 State of the Cloud Report

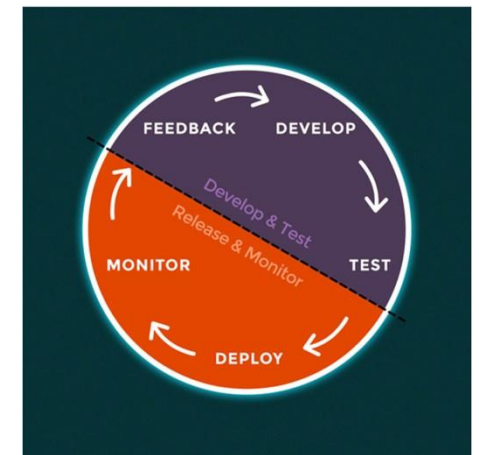
Respondents Using DevOps Tools



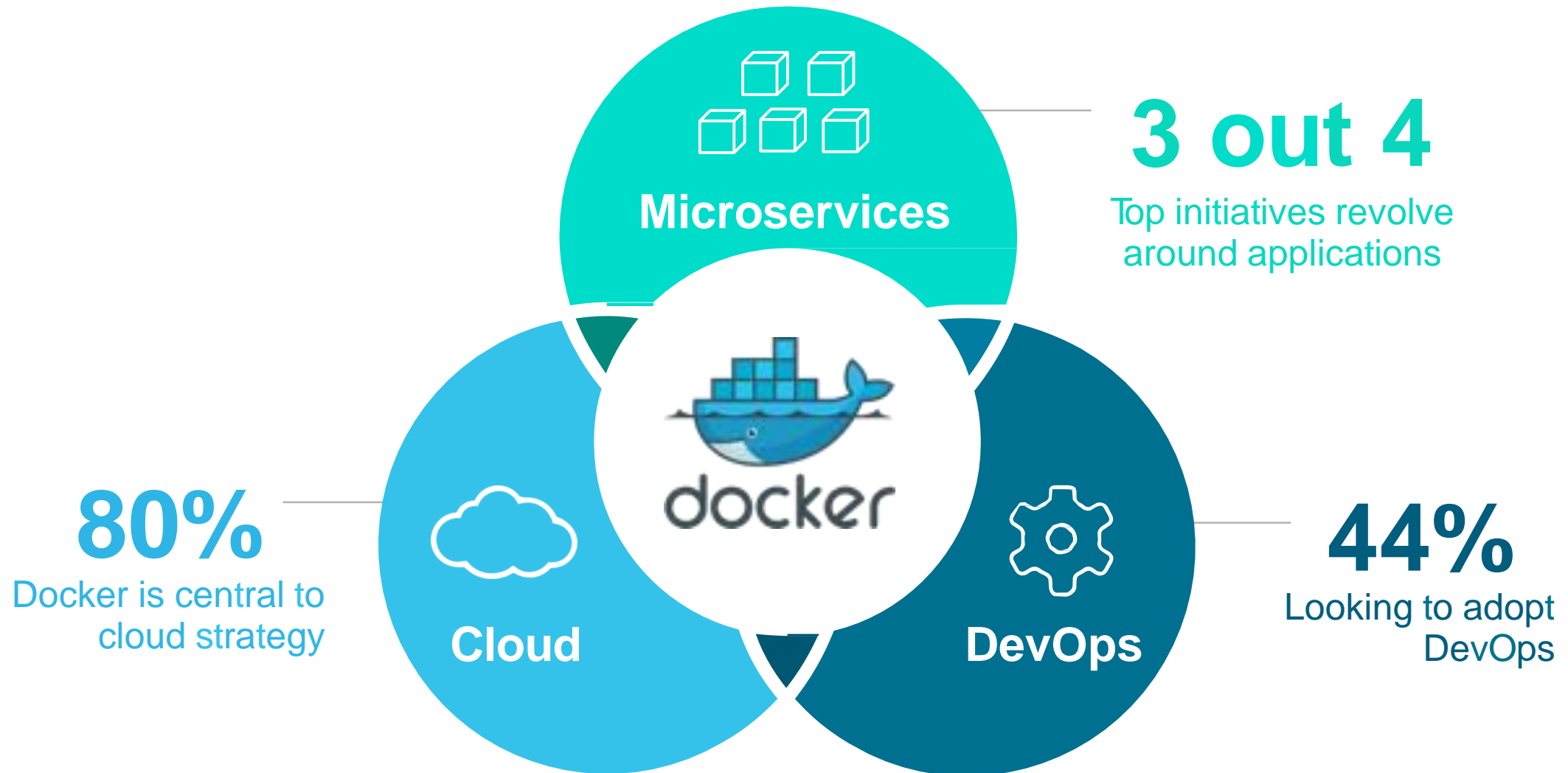
Source: RightScale 2016 State of the Cloud Report

“ In 2018 more than 50% of new Workloads will be deployed into containers in at least one stage of the life application cycle⁽¹⁾ ”

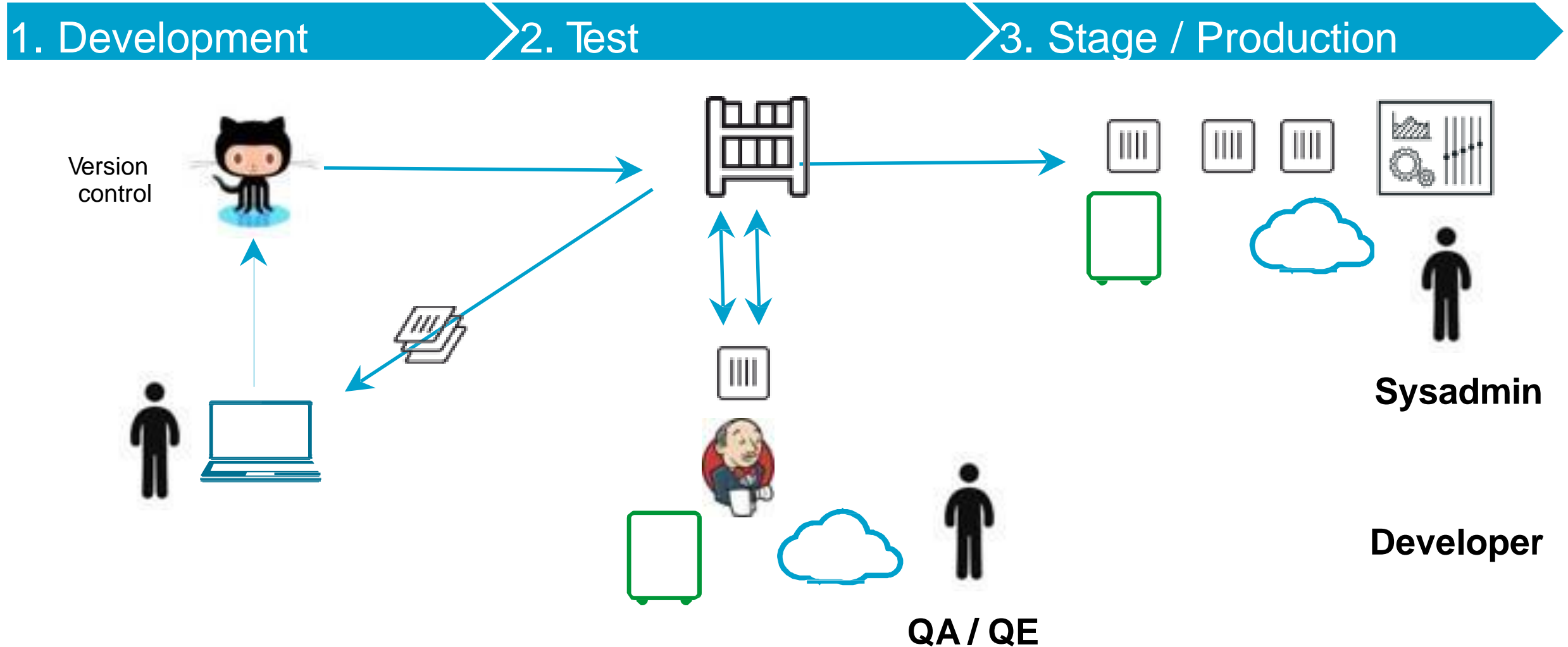
(1) Gartner



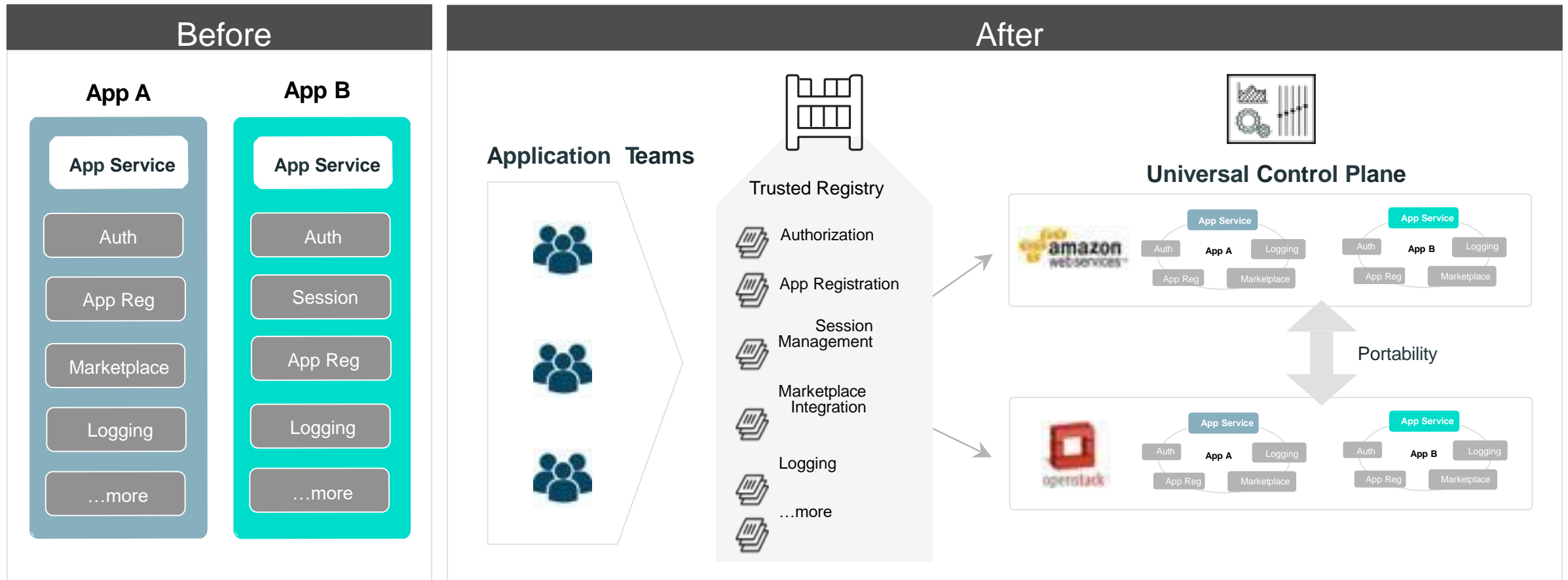
Driving force behind modern app initiatives



Scenario: Continuous Integration and Delivery



Scenario: Enabling Transformation to Microservices



Common services in monoliths are turned into base applications stored in the Trusted Registry available to all app teams

Teams request into central IT maintained portal/registry to provision infrastructure and pull base images

Monoliths are now micro services applications. Each app has its own containers based on the same base image



What is Docker?

What is Docker?

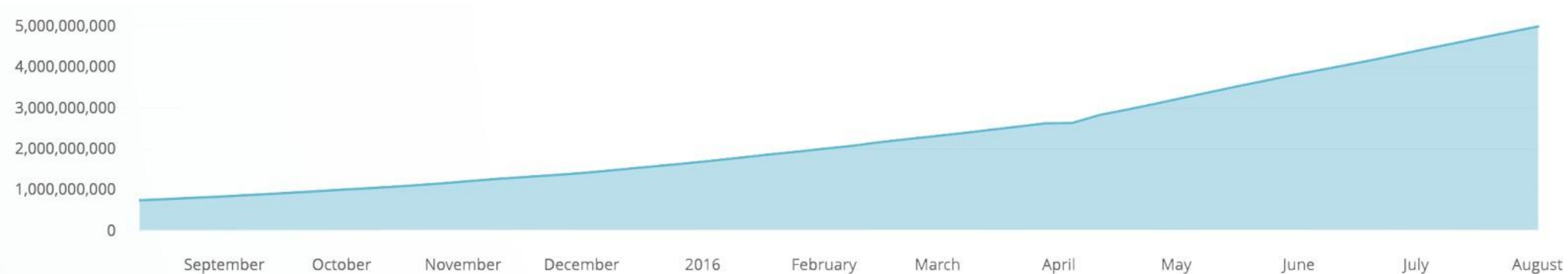
Docker is a platform for developing, shipping and running applications using container technology

The Docker Platform consists of multiple products/tools

- Docker Engine
- Docker Hub
- Docker Trusted Registry
- Docker Machine
- Docker Compose
- Docker for Windows/Mac
- Docker Datacenter

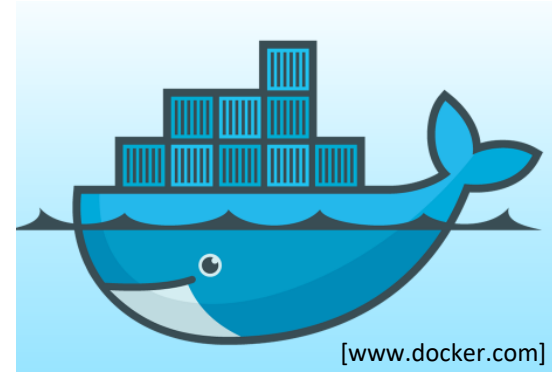
Docker Adoption

- Docker Hub - “Pull” is one download of a container image
- 650,000 registered users
- 5 Billion pulls since 2013
- Growing by 150% per month



Docker in Details

Docker: Name



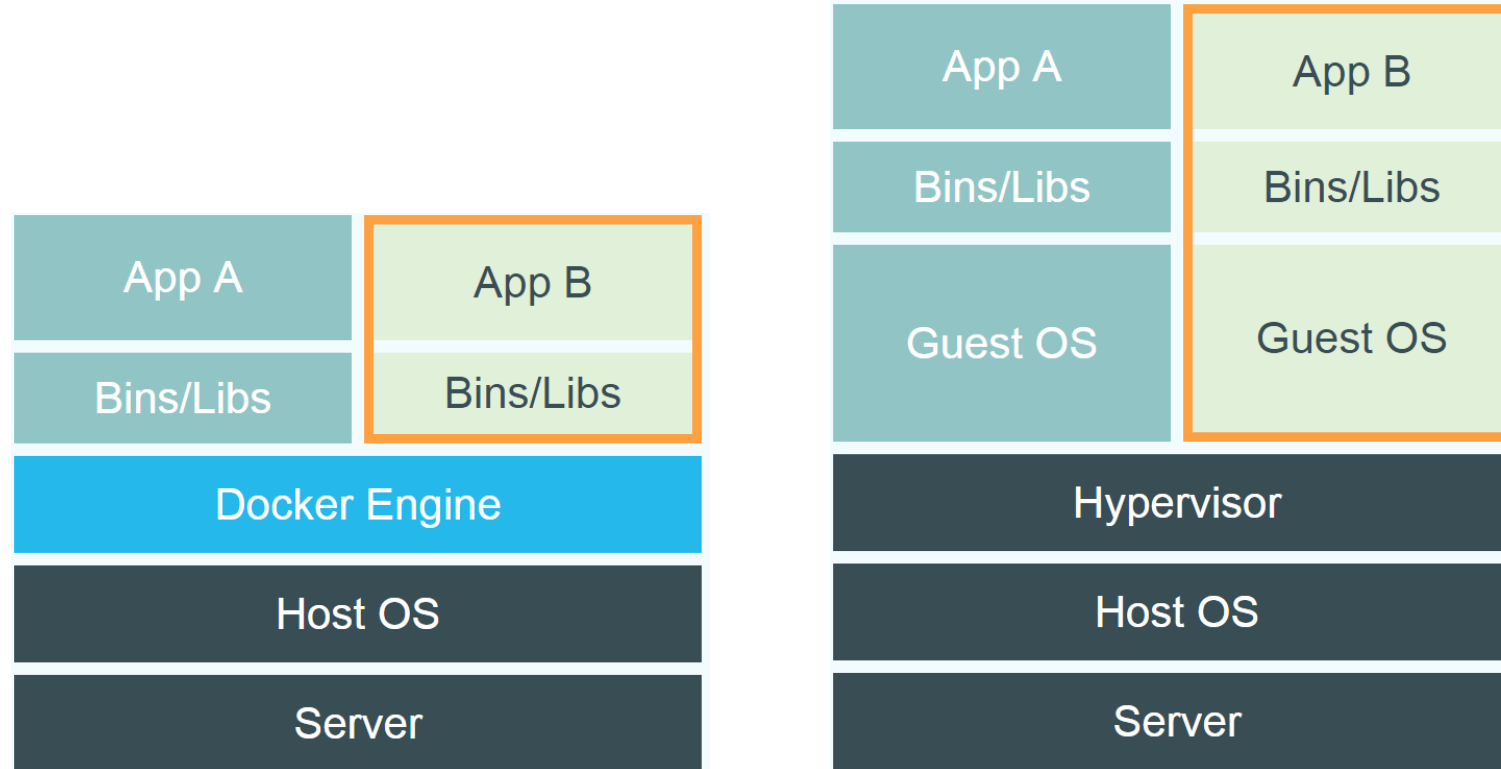
docker [naut.]: der Dockarbeiter, der
Hafenarbeiter

Source: leo.org

Provide a uniformed wrapper around a
software package: «*Build, Ship and Run Any
App, Anywhere*» [www.docker.com]

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

Docker vs. Virtual Machine



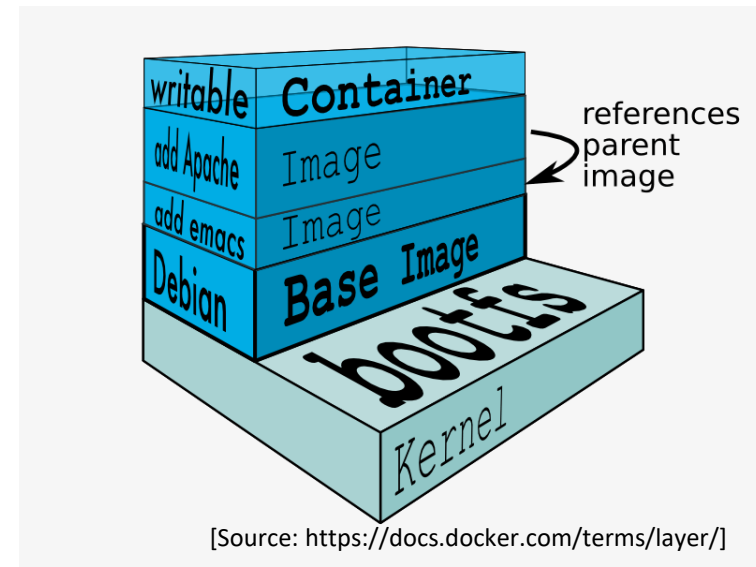
Source: <https://www.docker.com/whatisdocker/>

Docker Technology

libvirt: Platform Virtualization

LXC (Linux Containers): Multiple isolated Linux systems (containers) on a single host

Layered File System



Docker History

2013-03: Releases as Open Source

2013-09: Red Hat collaboration (Fedora, RHEL, OpenShift)

2014-03: 34th most starred GitHub project

2014-05: JAX Innovation Award (most innovative open technology)

Technology Radar

2014-01: Assess

2014-07: Trial

Source:

<http://www.thoughtworks.com/radar/tools/docker>

Run Platforms

Various Linux distributions (Ubuntu, Fedora, RHEL, Centos, openSUSE, ...)

Cloud (Amazon EC2, Google Compute Engine, Rackspace)

2014-10: Microsoft announces plans to integrate Docker with next release of Windows Server

Hello World

Simple Command - Ad-Hoc Container

```
docker run ubuntu echo Hello  
World
```

```
docker images [-a]
```

```
docker ps -a
```

Terminology - Image

Persisted snapshot that can be run

images: List all local images

run: Create a container from an image and execute a command in it

tag: Tag an image

pull: Download image from repository

rmi: Delete a local image

This will also remove intermediate images if no longer used

Terminology - Container

Runnable instance of an image

ps: List all running containers

ps -a: List all containers (incl. stopped)

top: Display processes of a container

start: Start a stopped container

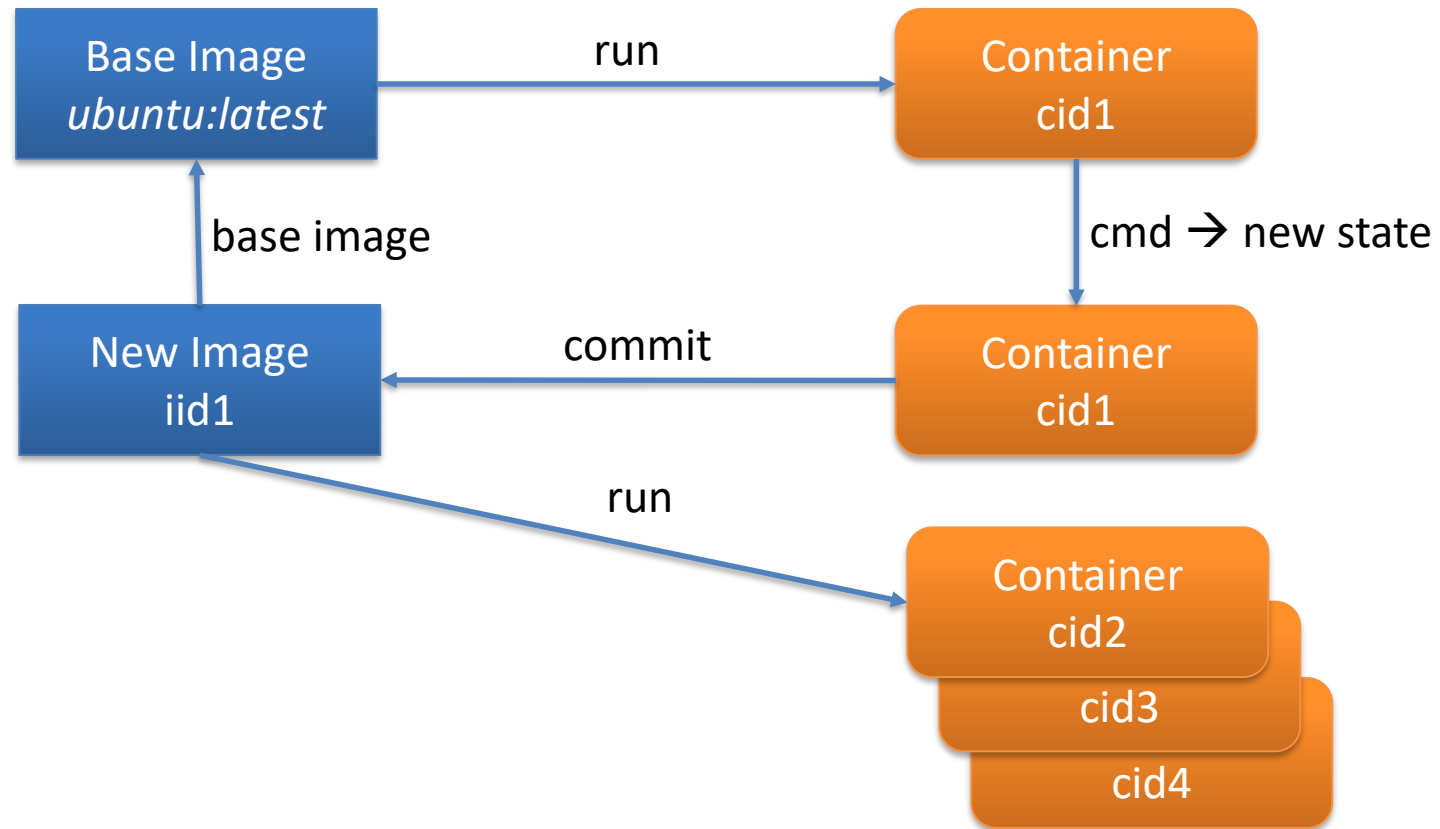
stop: Stop a running container

pause: Pause all processes within a container

rm: Delete a container

commit: Create an image from a container

Image vs. Container



Dockerfile

Create images automatically using a build script:
«Dockerfile»

Can be versioned in a version control system like
Git or SVN, along with all dependencies

Docker Hub can automatically build images based
on dockerfiles on Github

Dockerfile Example

Dockerfile:

```
FROM ubuntu
ENV DOCK_MESSAGE Hello My World
ADD dir /files
CMD ["bash", "someScript"]
docker build [DockerFileDir]
docker inspect [imageId]
```

Mount Volumes

```
docker run -ti -v  
/hostLog:/log ubuntu
```

Run second container: Volume can be shared

```
docker run -ti --volumes-from  
firstContainerName ubuntu
```

Publish Port

```
docker run -t -p 8080:80  
ubuntu nc -l 80
```

Map container port 80 to host port 8080

Check on host: nc localhost 8080

Link with other docker container

```
docker run -ti --link  
containerName:alias ubuntu  
See link info with set
```

Around Docker

Docker Images: Docker Hub

Vagrant: «Docker for VMs»

Automated Setup

Puppet, Chef, Ansible, ...

Docker Ecosystem

skydock / skydns

fig

Docker Hub

Public repository of Docker images

<https://hub.docker.com/>

`docker search [term]`

Automated: Has been automatically built from
Dockerfile

Source for build is available on GitHub



Persistent Storage for Containers

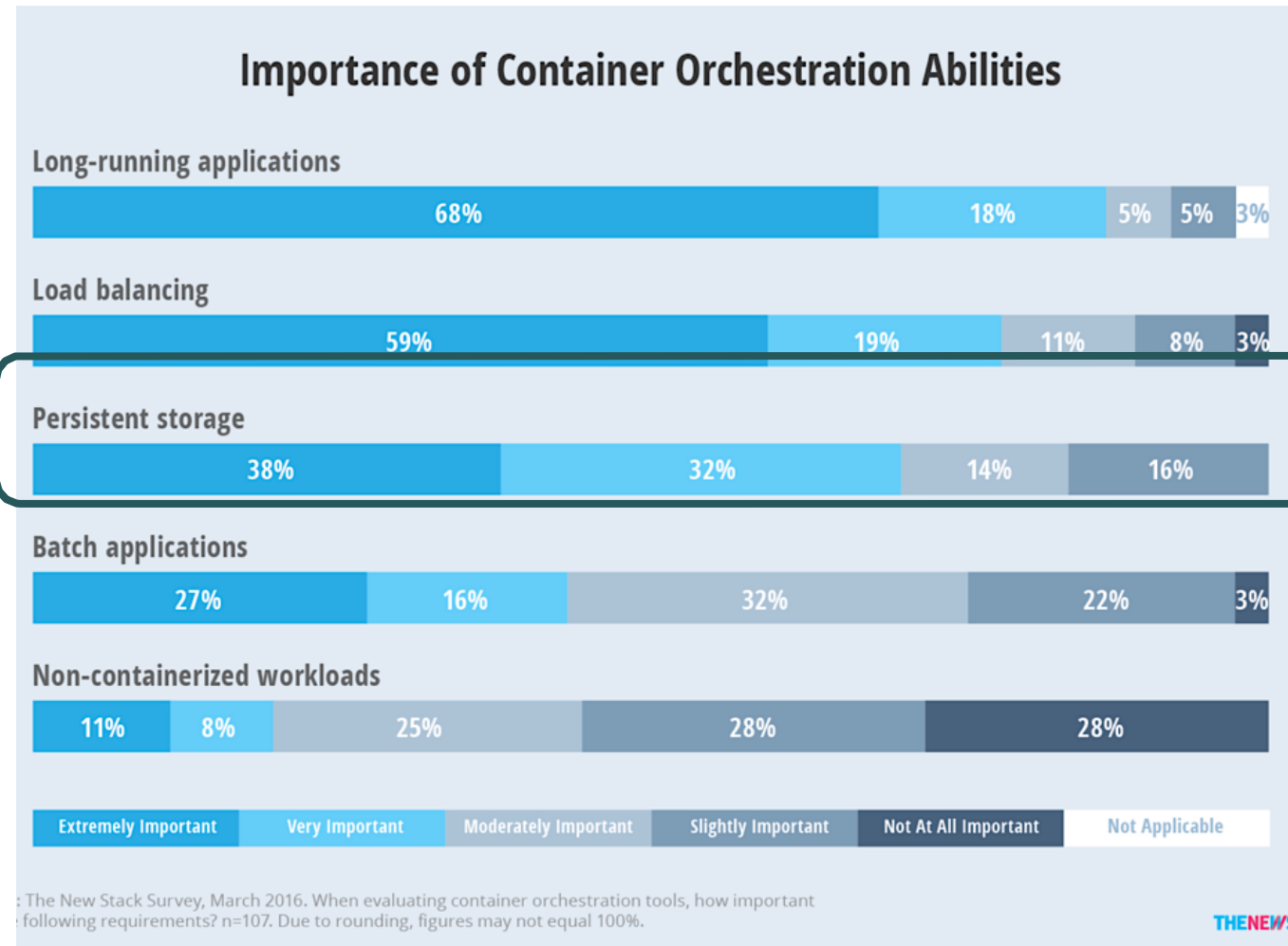
Stateful vs Stateless

“ **Stateful container** apps represent the next big **IT challenge**⁽¹⁾ ”

“ **Persistent storage** among top issues for container enterprise-readiness in **production**⁽²⁾ ”

“ **Stateful Database** applications such as Redis, MySQL, MongoDB among **most pulled** images on Docker Hub⁽²⁾ ”

- (1) Container Journal
(2) Gartner



Persistent Storage—Why

Data Accessibility

Run containers anywhere
without worries about where
data is located

Deployment Cycles

Traditional storage
approaches
slow-down innovation

Data Availability

Data needs to be
always on no matter
what happens



Storage Costs

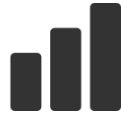
Storage defined by
Software using any
commodity HW or Cloud

Storage Services for Containers

Persistent Storage Management



Scale-Out



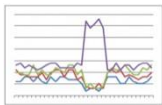
Snapshots



I/O Acceleration



Quality of Service



Encryption



Disaster Recovery



Heterogeneous Support



Poll Question

Do you have storage challenges in running containerized applications in production?

- A. Persistent Storage
- B. High Availability
- C. Quality-of-Service
- D. All of these

Docker Storage Types

Registry

Cold storage of container images

Graph

Active storage of running container images

Volume

Persistent block storage for data

Docker Registry Storage

- Config data stored via standard Docker volumes
- Images stored via driver
- Native filesystem (We don't care what's beneath - NFS or iSCSI is best!)
- Drivers available for cloud object storage for images (S3, Swift, GCS)

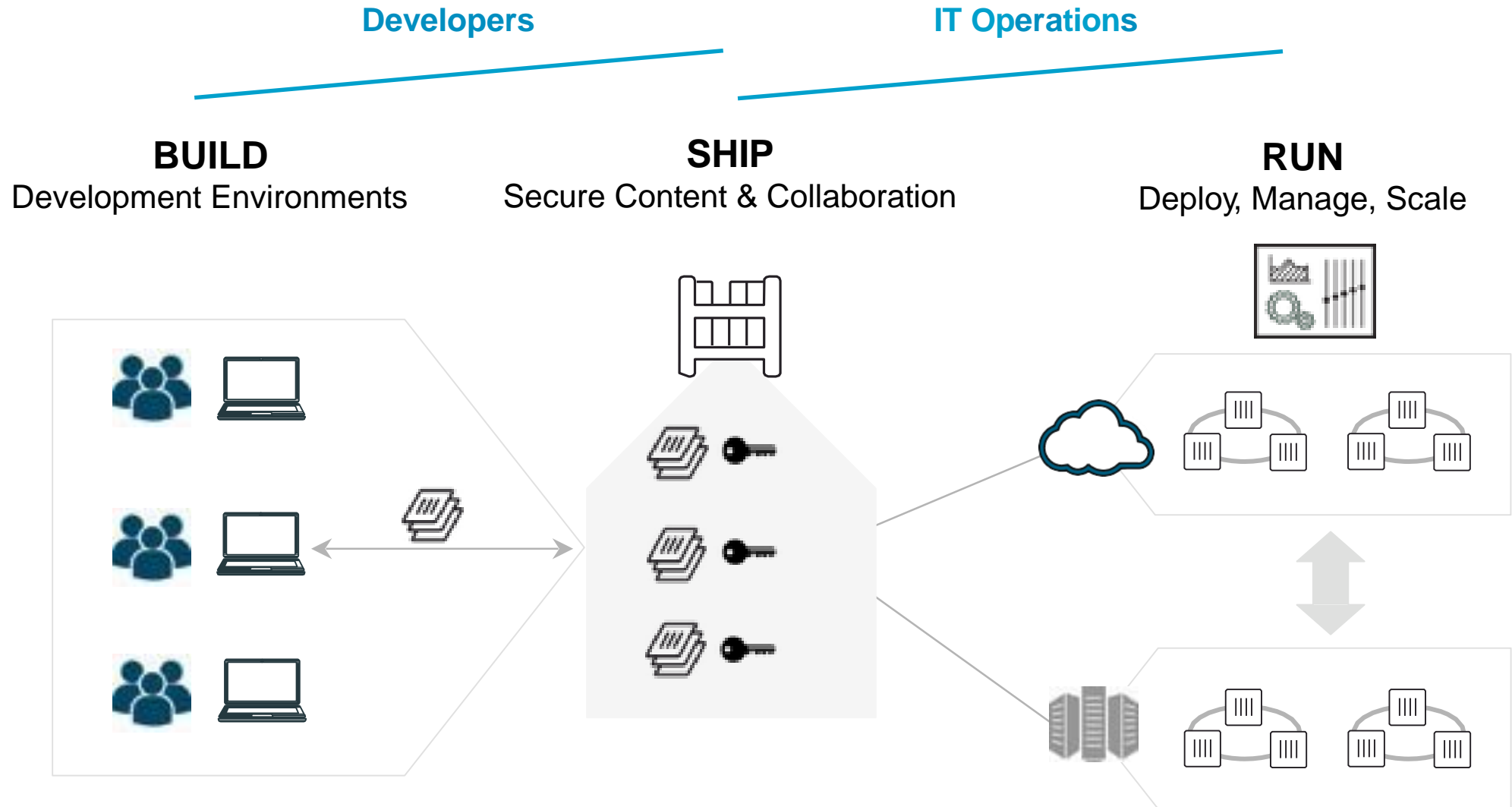
Docker Volume Storage

- This is where persistent data lives
- Extremely pluggable
- Network attached storage is extremely useful here

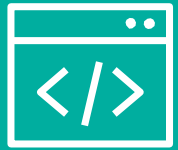


Docker Data Center (DDC)

Docker Datacenter CaaS workflow



DDC Delivers Agility, Portability and Control



Developers

- Freedom to create and deploy apps fast
- Define app needs



IT Operations

- Quickly and flexibly respond to changing needs
- Standardize, secure, and manage

Frictionless portability across teams, environments, infrastructure

Docker Datacenter delivers agility



- **Easy to setup and use**
 - Fastest time to value for ops
 - Developer self service from library of images
- **Native Docker solution**
 - Full support for Docker API
 - Integrated Docker Engine, Swarm, Compose
- **Extend the existing Docker developer experience to deployment**
 - Deploy Compose apps directly in UCP

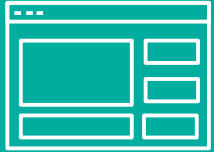
Docker Datacenter delivers portability



- **Compute, network and storage**
 - Network and volume plugins ensure apps work without recoding new environments
 - Move across public, private and hybrid cloudility
- **Seamless dev to prod workflow**
 - Same code in dev runs in prod w/o changes
 - Eliminate the “works on my machine” issues
 - Full support for Docker API

Docker Datacenter delivers control

Control

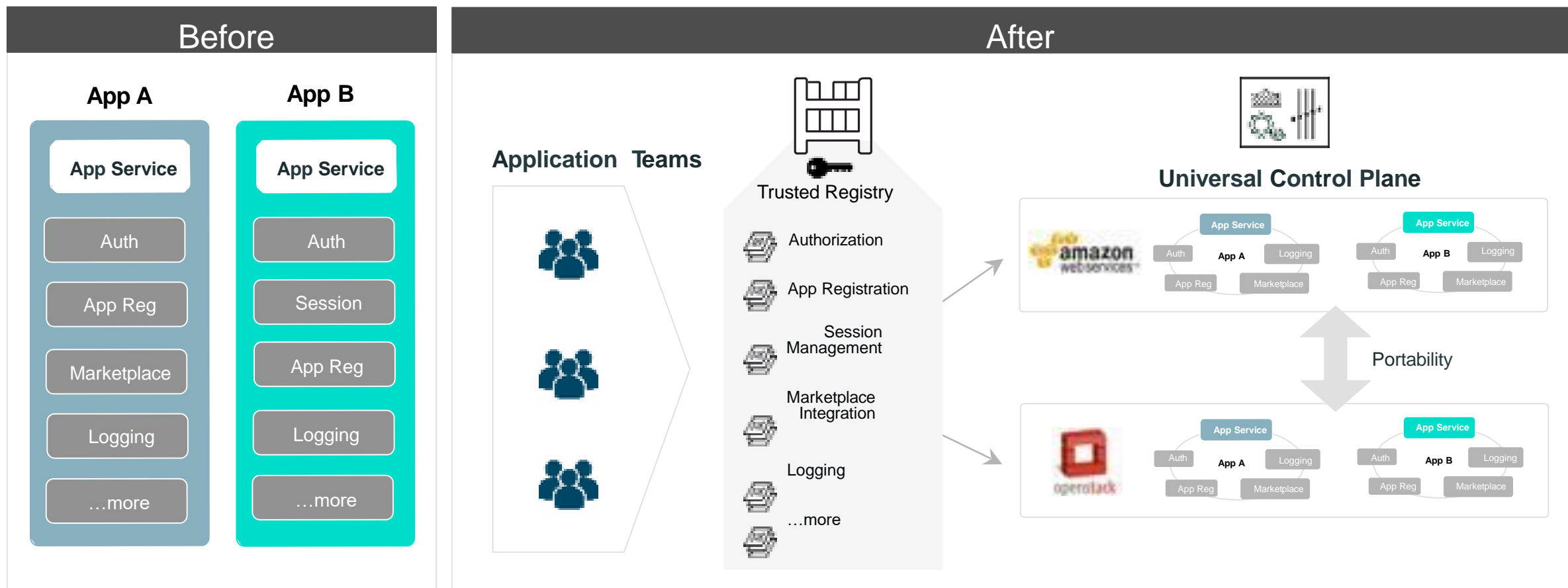


- **Management ease at scale**
 - View and manage apps, containers, nodes, volumes, networks, images, users and groups
 - Built in HA, backup/restore
 - Point and click GUI or CLI support
- **Integrated security and enterprise controls**
 - Content Trust for image signing and verification
 - Secure access with RBAC and LDAP/ADP
 - Out of the box TLS
- **Extend and integrate DDC to your systems and processes**



Docker Case Studies:
Brief overviews of recent
success stories for Docker customers

Case Study: ADP



Common services in monoliths are turned into base applications stored in the Trusted Registry available to all app teams

Teams request into central IT maintained portal/registry to provision infrastructure and pull base images

Monoliths are now micro services applications. Each app has its own containers based on the same base image

Case Study: SA Home Loans



SA Home Loans uses Docker Datacenter to convert Monoliths to Microservices

Goal

- Convert monolithic .Net applications (built in Mono) into microservices

Result

- Evaluated Docker running small-scale postgres services across 2 nodes
- Docker's enterprise-class networking and security capabilities were key but impressed with the ease-of-use of Docker Native orchestration - Swarm
- Currently running Docker Datacenter across 4 nodes as they are working to Dockerize all enterprise-class applications in the next few months

Case Study: GSA (Booz Allen)



Challenge

- Migrate away from monolithic application
- Long and cumbersome application development cycles

Solution

- Build a new developer platform (IAE Common Service Platform) with Docker Trusted Registry and commercially supported Docker Engine on AWS

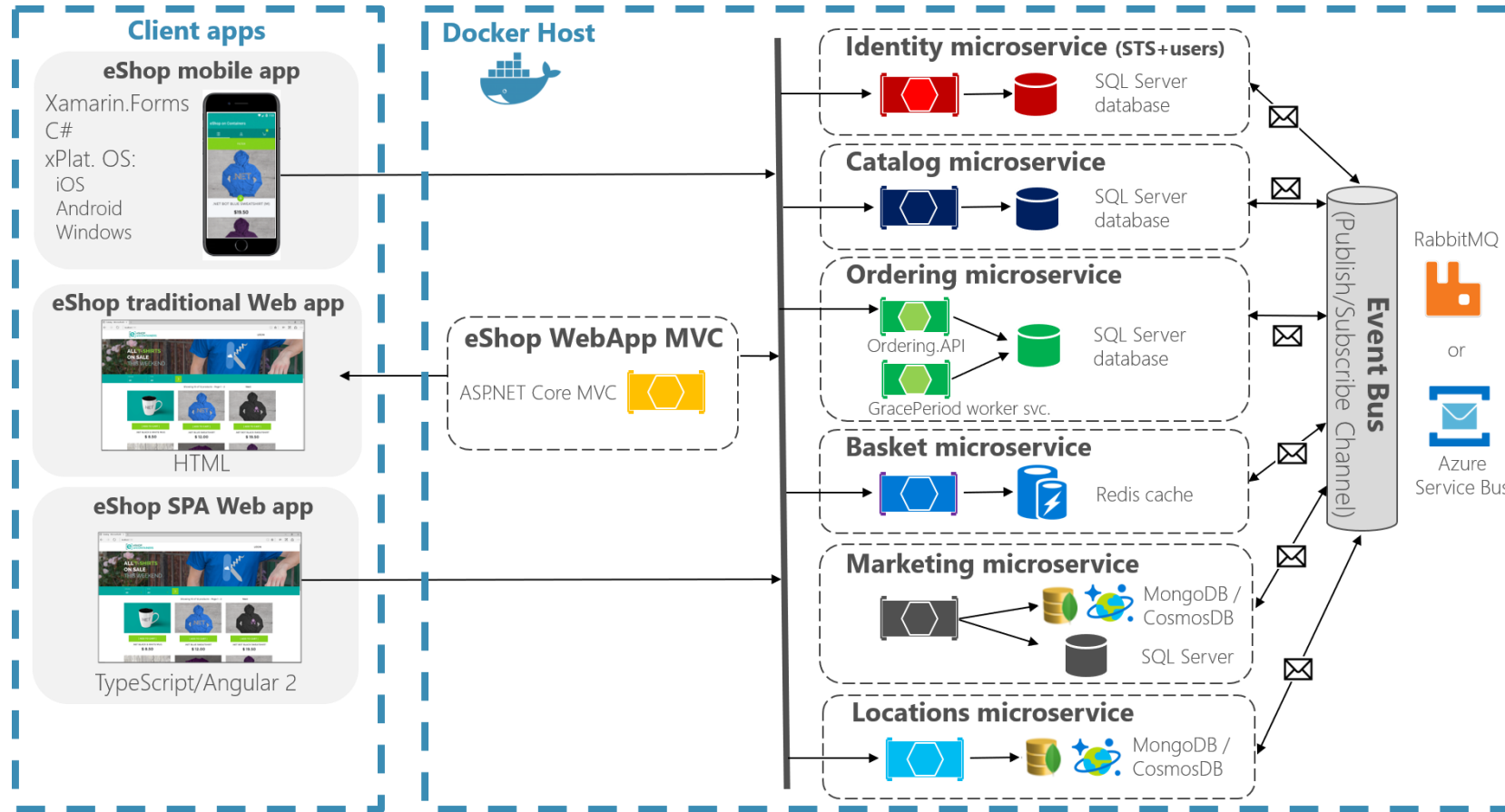
Benefit

- Improved customer centric services Reduced time-to-market
- Improve security and reduce attack surface area

An example of microservice-based system

eShopOnContainers reference application

(Development environment architecture)



Drillinginfo's case study: Why Docker?

A Drillinginfo (Robert Bastian's company) case:

- 5+ individual teams building "micro services" in Java and Scala

- Frictionless deployment of "micro-services" using Chef & AWS

- 25+ separate "micro-services" deployed in the previous 18 months

- Each service is typically deployed to a single AWS virtual machine

- Each service is deployed 6x - dev, test, staging (2x) and production (2x)

- 25+ "micro-services" became nearly 150 AWS virtual machines

Why Docker? COST!

The AWS bill is too damn high!

Decline in the global price of oil causing churn in our business
6 AWS virtual machines per service isn't sustainable with our budget

AWS monthly bill started to gain visibility from sr. management and *the board*

Why Docker? WASTE!

We weren't using the compute and memory resources purchased from AMZN!

- Nearly all "micro-services" were at 1% CPU utilization

- Nearly all "micro-services" were only using 40% of memory (JVM)

- 150+ virtual machines essentially sitting idle

Why Docker? LOCK IN!

How would we leave AMZN if we wanted to?

Could we use Drillinginfo IT's Openstack platform?

What about alternate IaaS providers like Rackspace or Azure?

What about Container as a Service (CaaS) providers like Joyent, Tutum or Profitbricks?














What about using Amazon's Container Service?

My World Needs To Change - Problem Statement

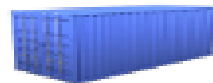
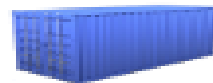
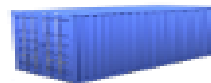
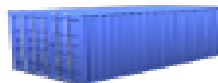
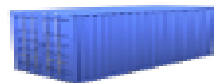
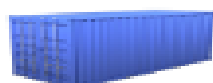
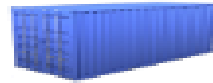
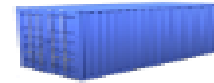
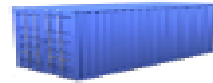
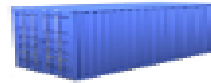
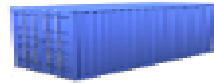
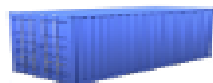
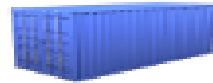
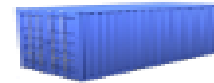
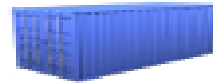
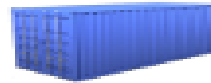
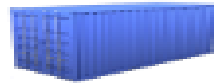
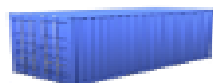
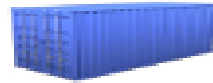
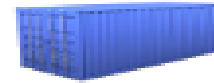
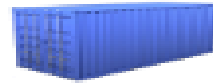
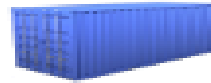
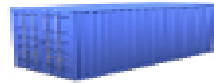
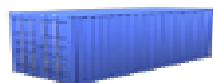
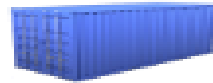
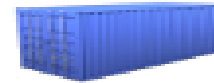
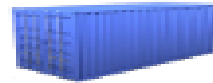
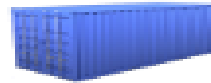
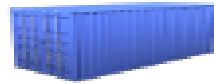
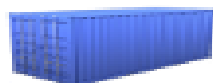
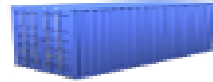
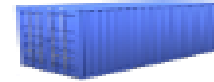
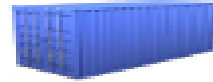
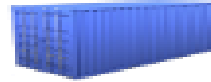
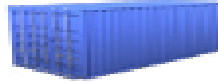
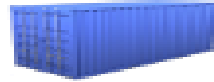
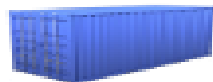
"How can we *deploy fewer* virtual machines while *increasing the density and utilization* of services per machine *without locking* us into a specific IaaS provider?"

How Docker Solves All The Problems

Docker Containers - Shipping Matrix From Hell

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

Docker Containers - Standard Shipping Container

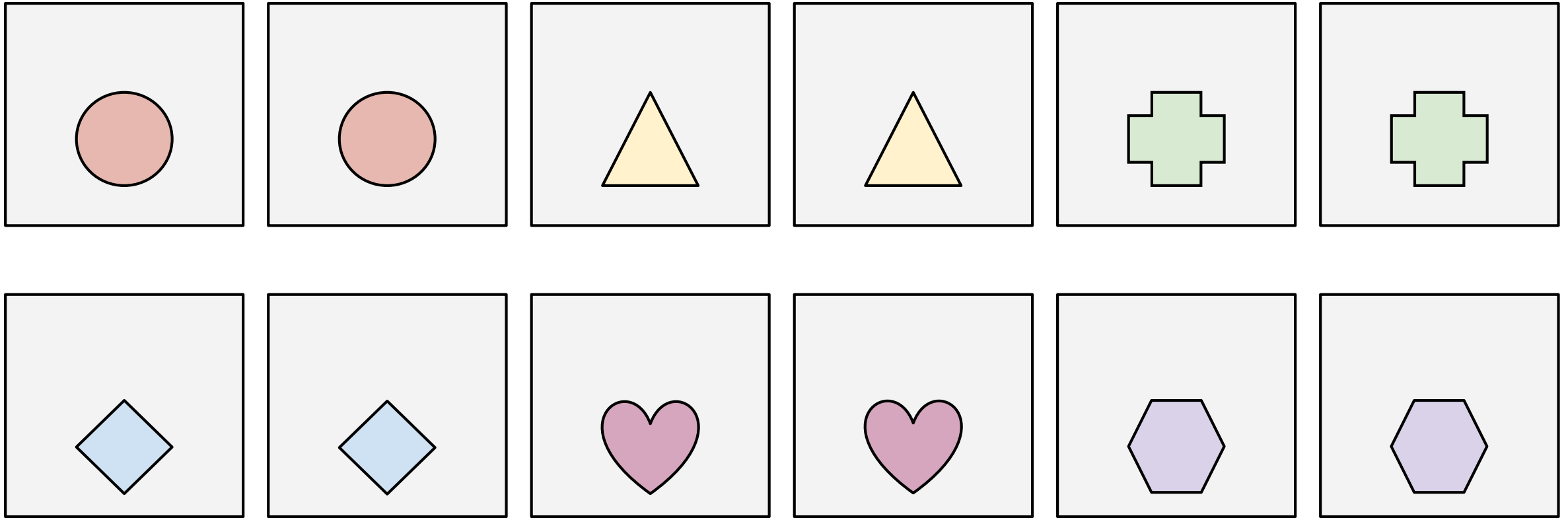


What's Inside Doesn't Matter



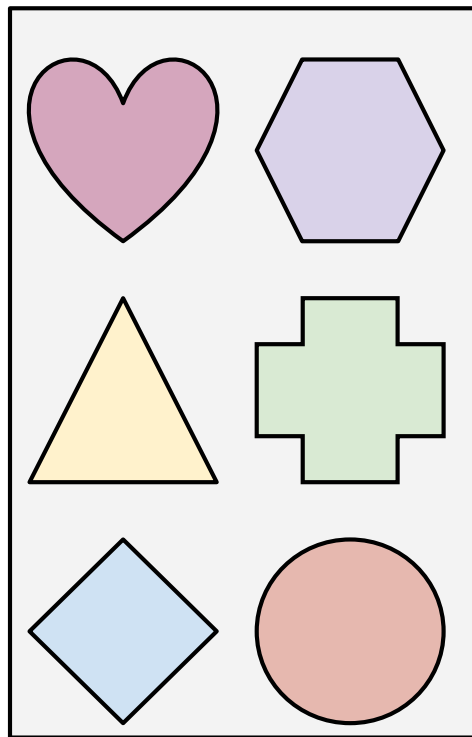
Why Docker Is Important - Before Containers

Very inefficient use of memory and CPU resources

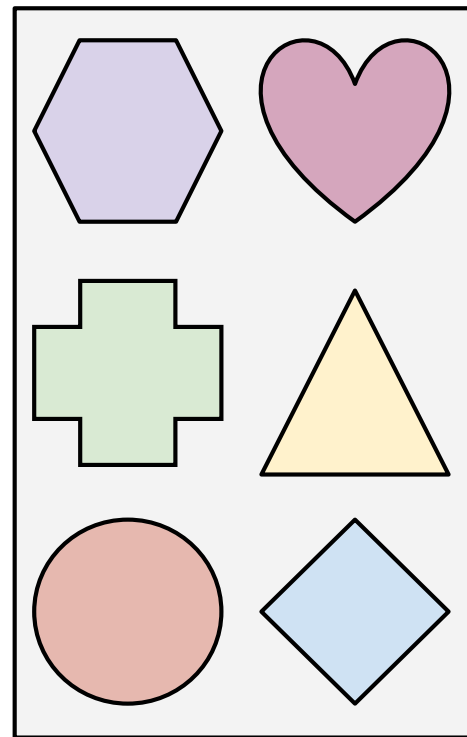


Why Docker Is Important - After Containers

Isolated
services in
fewer VMs...



... and use
VMs more
efficiently.



Why Is Docker Important?

Docker container technology provides our "micro-services" platform:

- Increased *density* of *isolated* "micro-services" per virtual machine (9:1!)

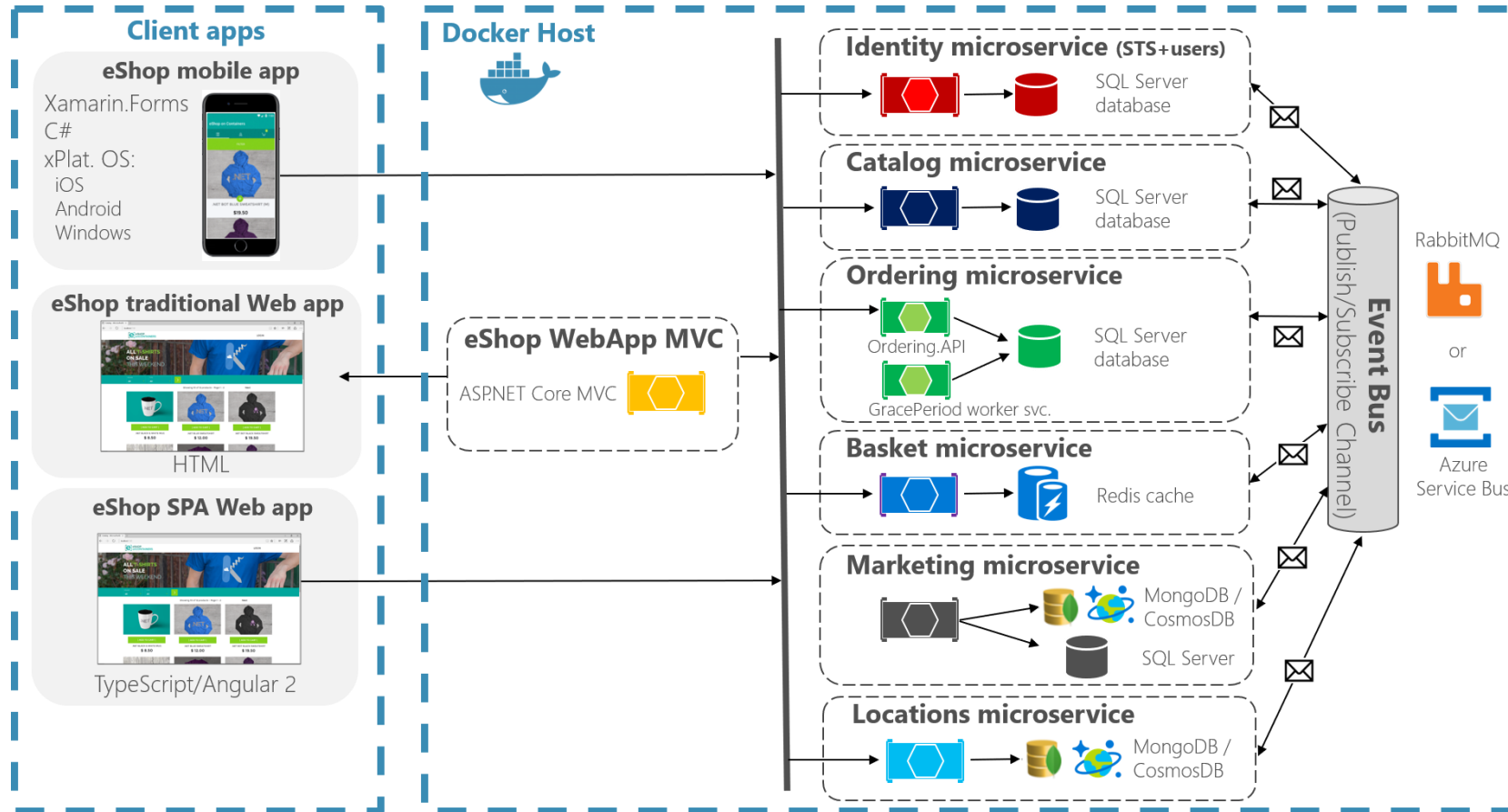
- Containerized "micro-services" are *portable* across machines and providers

- Containerized "micro-services" are much *faster* than virtual machines

Containers Alone Aren't Enough

eShopOnContainers reference application

(Development environment architecture)



Problem: Managing microservices deployed in containers

But Containers Aren't Enough!

Running containerized "micro-services" in production requires *much more* than just Docker.

It requires a "Platform" that can do the following:

- Building and pushing Docker images to an image repository

- Pulling images, provisioning and scheduling containers

- Discovering and binding to services running as containers

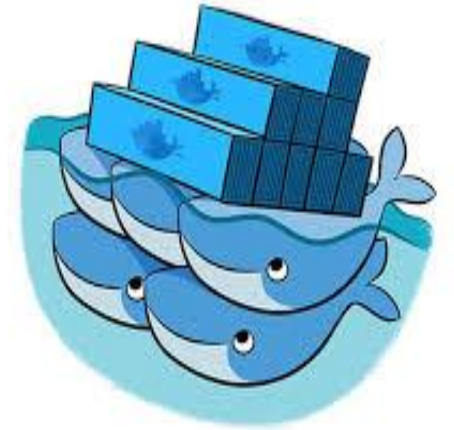
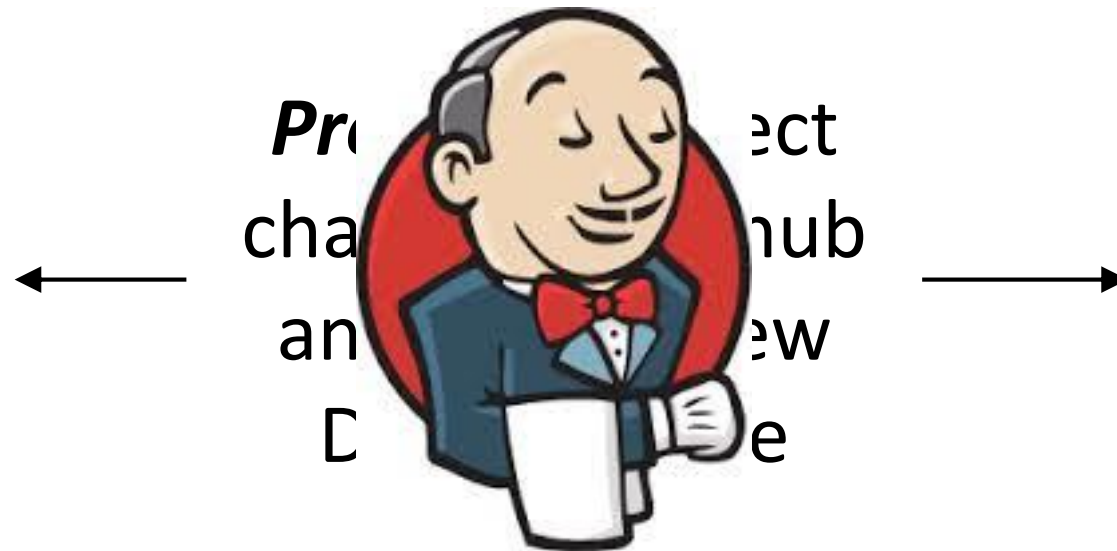
- Containers discovering and binding to other containers

- Operating and managing services in containers

Note

Some tools mentioned in the following slides may have better (or open source) replacements now, as shown in the last few slides

Drillinginfo Docker Platform: Build & Store Images



Drillinginfo Docker Platform: Jenkins & Dockerhub

Problem: How do we *build* images? Jenkins automates the image builds.

We started building our images with Ubuntu 14.04 (1GB)

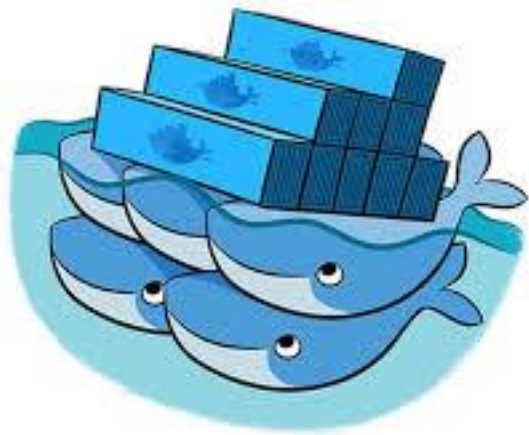
We settled on *Alpine*, a minimal linux distribution (5MB)

Typical "micro-services" now ~ 390MB

Problem: Where do we *put* them? Dockerhub.

- Tried Docker Trusted Registry and Core OS Enterprise Registry
- Settled on using *Dockerhub*
- Use *latest and sem-ver* tags on our images

Drillinginfo Docker Platform: Provisioning, Scheduling



Dockerhub



Drillinginfo Docker Platform - Chef

Problem: How do we determine *which host* to run a container on and how do we *configure and start* the container?

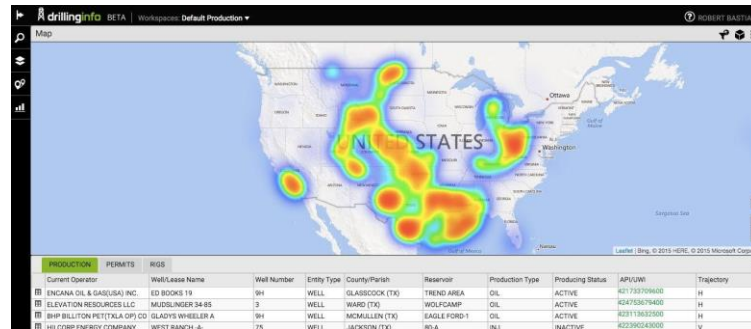
We solve *scheduling and provisioning* with Chef.

Chef *schedules* containers on specific hosts using *Chef roles*

Chef *provisions* and *configures* containers using *Chef recipes and environments*

Each "micro-service" has an associated Chef *recipe* that converts Chef attributes into *container environment variables*

Drillinginfo Docker Platform: Service Directory

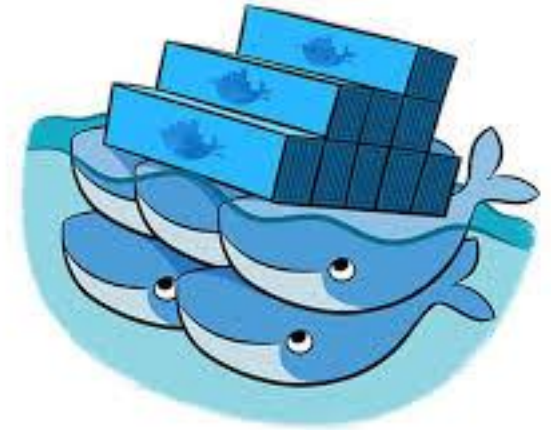


DI Web
Applications

Problem: How
can web



bind to
containers?



DI Docker Containers

Drillinginfo Docker Platform - Consul

Problem: How do our browser applications *locate* service containers?

We use Hashicorp's Consul as our *service directory*.

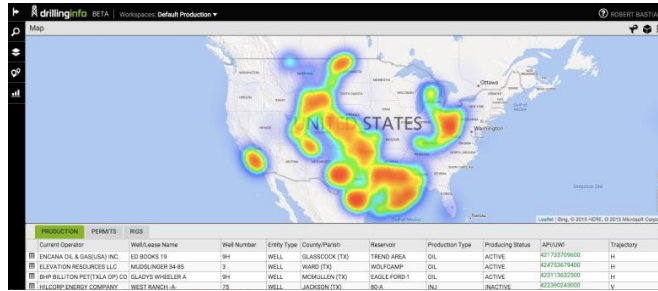
Containers *automatically register themselves* with Consul when started.

The Docker daemon *emits real-time lifecycle events* for container start

We use a utility container called *Registrator* to automate the registration of "micro-service" containers with Consul

Containers are registered with a *health check* that Consul polls to determine the health of the container

Drillinginfo Docker Platform: Service Discovery



Problem: How can web applications *discover and bind* to containers?



CONSUL
TEMPLATE



Drillinginfo Docker Platform - Consul Template

Problem: How do our browser applications use services deployed in containers?

We use Hashicorp's Consul Template for service discovery and Varnish for load balancing.

Consul Template detects containers in Consul and updates Varnish configuration

Consul Template participates in the Consul cluster using Consul Client

Consul Template automatically **adds healthy** containers and **removes sick** containers from the Varnish load balancer by updating Varnish configuration

Browser applications use **Varnish routes** to reach services running in containers

Drillinginfo Docker Platform: Container Dependencies



Drillinginfo Docker Platform - Service Proxy

Problem: How can containers find their containerized dependencies on the same host and different hosts?

We use Consul, Nginx and Consul Template to implement a "Service Proxy" for inter and intra-host container communication.

- We built a utility container called "Service Proxy" that uses Consul's service directory to locate a container's ip address and port
- "Service Proxy" then uses Consul Template to create an nginx.conf with load balanced routes for each service container
- ***Docker Links*** work for intra-host dependencies but with a ***gotcha***

Drillinginfo Docker Platform: Operations & Monitoring



Uptime

Events

Checks

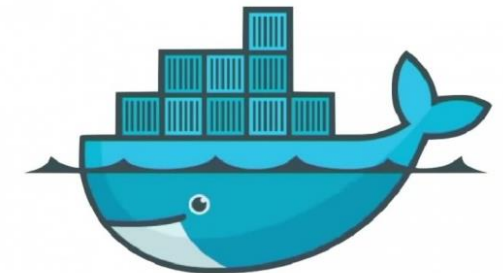
Tags

Up 38

Tags

last 24 hours

Name	Availability	Uptime	Response Time
arcgis-pep-service	100%	<div></div>	18ms
atlas-service	100%	<div></div>	17ms
authz-service	100%	<div></div>	13ms
completions-tile-service	100%	<div></div>	14ms
entitlements-service	100%	<div></div>	15ms
exports-service	100%	<div></div>	14ms
feature-gates-service	100%	<div></div>	17ms
identity-service	100%	<div></div>	19ms
image:drillinginfo/entitlements-service:2.21.4	100%	<div></div>	15ms
intl-file-download-service	100%	<div></div>	15ms
license-generator-service	100%	<div></div>	22ms
mobile-subscriptions-service	100%	<div></div>	56ms
papi-service	100%	<div></div>	56ms



Drillinginfo Docker Platform - Operations & Monitoring

Problem: How do we *monitor containers* and *notify and escalate* when containerized services aren't healthy?

We use Uptime and VictorOps monitor our containerized services.

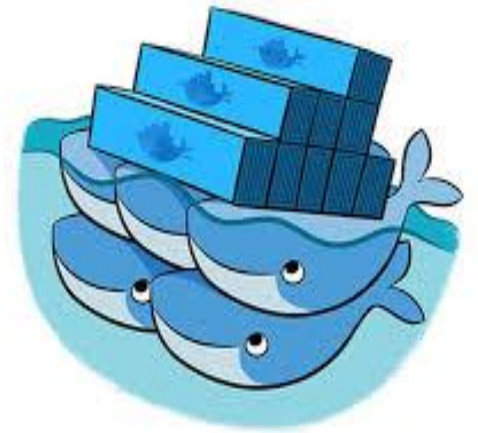
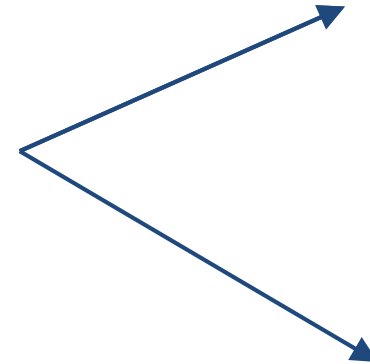
- A utility container monitors Docker container lifecycle events and *automatically registers a service check with Uptime* when a container starts
- Uptime *service interruptions* to VictorOps for on-call scheduling, paging and escalation

Drillinginfo Docker Platform: Operations & Monitoring

Prob
moni
usa



o we
ource
and



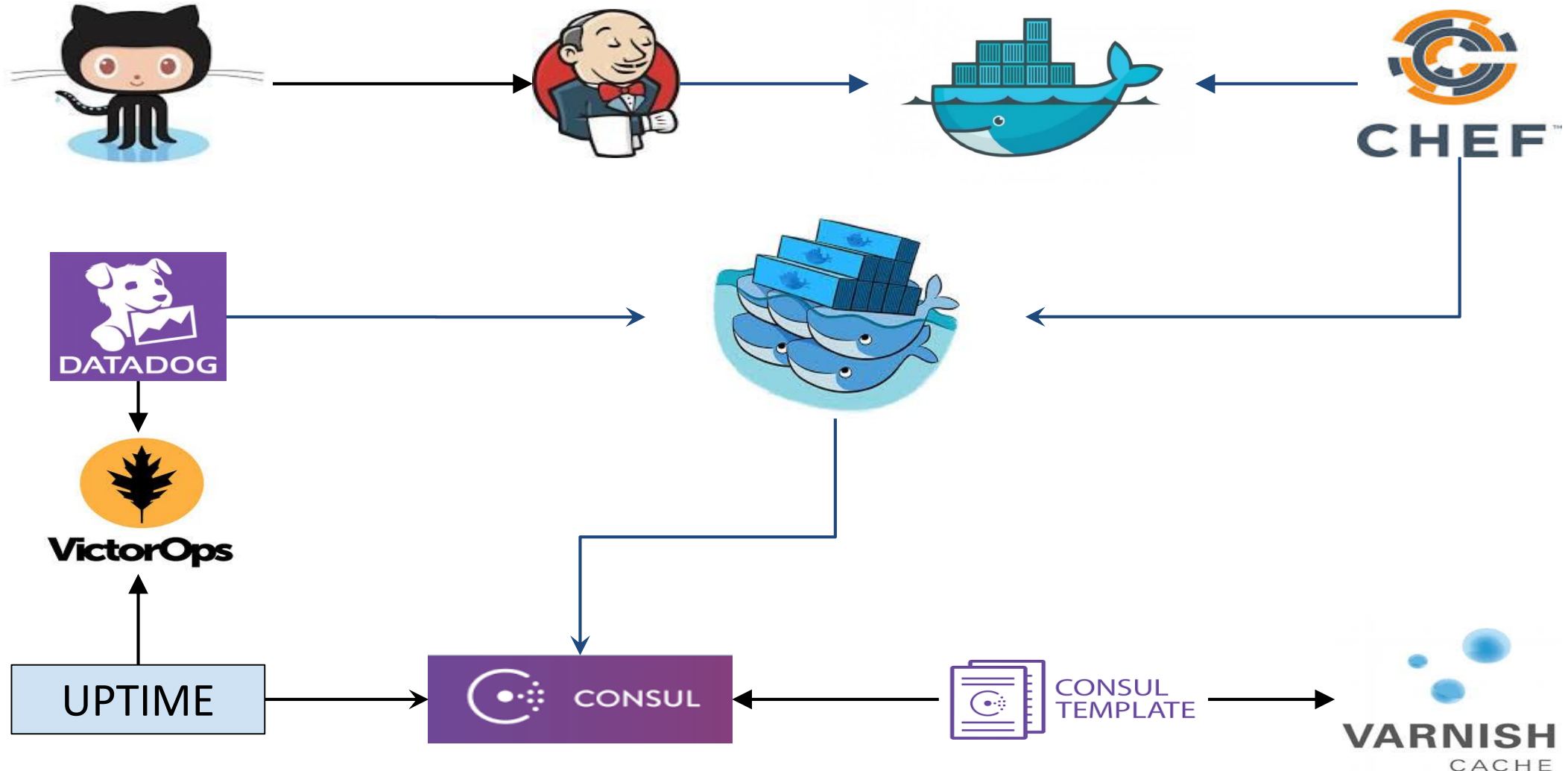
Drillinginfo Docker Platform - Operations & Monitoring

Problem: How do we *monitor* our Docker host's *resource usage*?

We use Datadog to monitor the Docker host utilization and the service's metrics.

- Datadog helps us visualize the resource usage on a host
- Datadog helps us understand how our services are performing
- Datadog helps us understand how to “pack” containers onto hosts by exposing the current utilization of CPU and memory resources on the host

Drillinginfo Docker Platform - Overview



Drillinginfo Docker Platform - Wrap Up

The Docker container technology and the Drillinginfo Docker Platform provide our "micro-services" infrastructure the following benefits:

- Reduced cost for IaaS hosting

- Reduced waste of virtual machine resources

- Standardized deployment mechanism for "micro-services"

- Standardized service directory, service discovery

- Standardized metrics dashboards, monitoring and alerting

Drillinginfo Docker Platform - Future

Chef has gotten us where we are today but *not where we want to be.*

- Container orchestration

- Host provisioning and pooling

Drillinginfo Docker Platform - Orchestration

Docker Compose will **replace Chef roles** defining the "micro-services" deployed on our platform and which Docker host they run on.

The Docker Compose YAML file:

- Defines which containerized "micro-services" run on which host

- Define the environment variables for each container

I believe that IaaS providers will standardize on Docker Compose for container orchestration.

Drillinginfo Docker Platform - Provisioning & Pooling

Docker Machine will *replace Chef for provisioning virtual machines* with Docker.

Docker Machine automates the provisioning of Docker hosts
Docker Swarm/Kubernetes will *replace Chef for scheduling containers* on a host.

Swarm combines Docker Machines into a single pool of compute and memory resources

Swarm provides container scheduling and supports plug-in schedulers

Docker Compose will define all the containers that run on the Swarm

Updates: current tools

- Docker machine
- Docker compose
- Docker swarm/Kubernetes
- Docker ecosystem: the moby project

Docker machine

What is it?

- Since containers are "linux" containers, you need to have the linux host
- For running in any non-linux environment, you will need to create a linux machine
- Docker machine is used for this purpose

Docker machine

How it works?

- It uses VirtualBox to get linux VM running

* Virtualbox runs on Windows, OS X, linux, Solaris Unix

- Automatically create a linux host with VirtualBox

Docker compose

- For automatically provisioning docker containers on a single host
- Specify what you need with a YAML configuration file: docker-compose.yml

```
web:
  build: .
  ports:
    - "5000:5000"
  volumes:
    - .:/code
  links:
    - redis
redis:
  image: redis
```

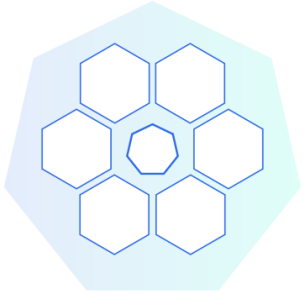
Two containers: “web” and “redis”
“web” is built from the Dockerfile in the current directory, runs on port mapping container 5000->host 5000, mount a volume “.” to “/code” in the container and link to another container “redis”

“redis” uses the docker image redis from Docker hub

Docker swarm

- Used to manage a cluster of container hosts
- Deploy services to the container hosts
- Inspect a service on the swarm
- Scale a service (more containers running the same service)
- Update services

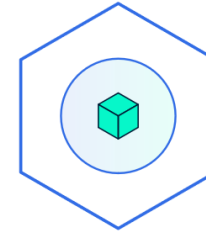
Kubernetes Basics Modules



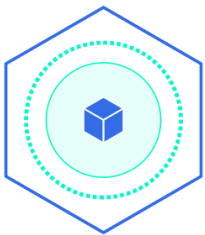
[1. Create a Kubernetes cluster](#)



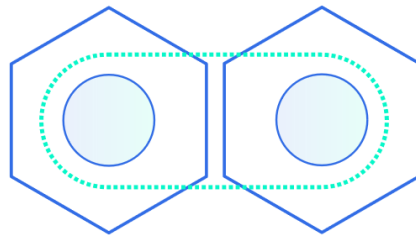
[2. Deploy an app](#)



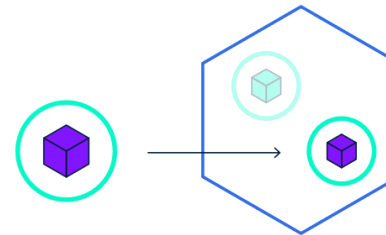
[3. Explore your app](#)



[4. Expose your app publicly](#)



[5. Scale up your app](#)



[6. Update your app](#)