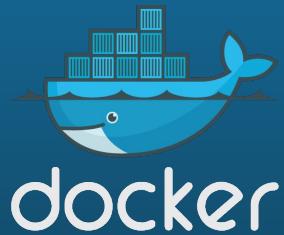


Docker Accredited Consultant MTA Enablement Workshop

October 2017
v17.06



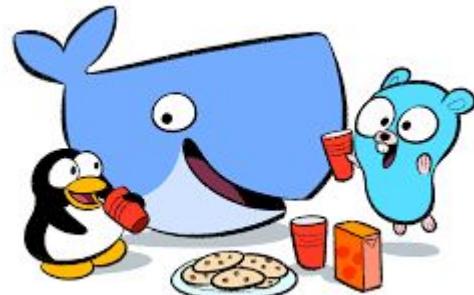
DAC Enablement Workshop

Format:

- Interactive
- Very hands-on, Tech heavy
- Focus on Design Best Practices and Architecture
- 4 Labs (Team)
- Discussion and questions encouraged!

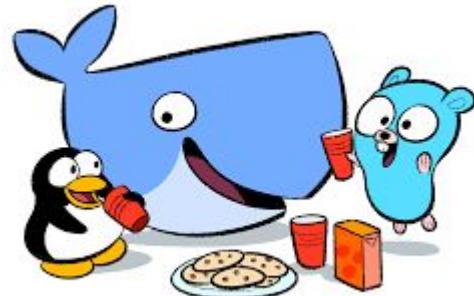
Goals!

- Understand the MTA program
- Become enabled to deliver MTA POCs
- Go deeper into Docker Enterprise Edition(EE)
- Understand EE design and architecture best practices
- Have fun doing all the above :)



Additional Resources

- training.play-with-docker.com
- github.com/docker/labs
- success.docker.com



Agenda

Day 1

- About this Workshop
- MTA Introduction
- MTA POC
 - Foundations
 - Roles & Responsibilities
 - Methodology & Tooling
- Hands-on Lab
- Wrap-Up

Agenda

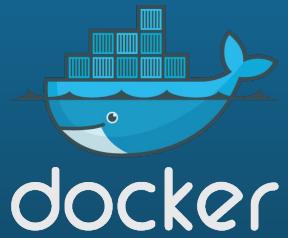
Day 2

- Introductions and Setup (9:00 - 9:15)
- Docker EE Basic Deep Dive(9:15-11)
 - Orchestration (30 mins)
 - Networking (30 mins)
 - Break (5 mins)
 - Security (30 mins)
 - Logging (10 mins)
- Swarm Lab (11:00- 12:00)
- Lunch (12:00-1:00)
- UCP Deep Dive (UCP) (1:00-2:00)
- UCP Lab (2:00-3:00)
- Break (3:00-3:15)
- DTR Deep Dive (3:15-4:15)
- DTR Lab (4:15-5:00)

Day 3

- Application Deployment Deep Dive (9:00-9:45)
- Application Deployment Lab (9:45-11:00)

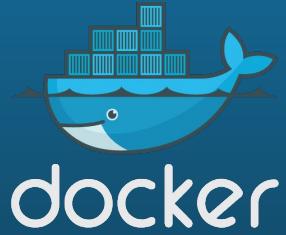
Docker Enterprise Edition Basics



Docker EE Basic Deep Dives

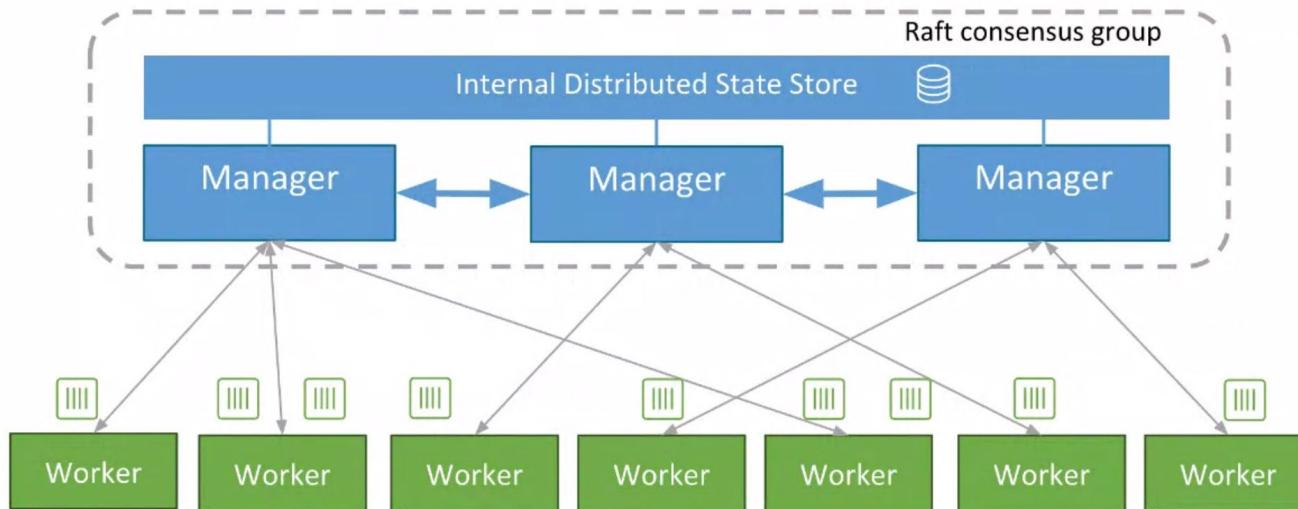
- **Orchestration (30 mins)**
- **Networking (30 mins)**
- **Security(30 mins)**
- **Logging (15 mins)**

Orchestration



Swarm Architecture

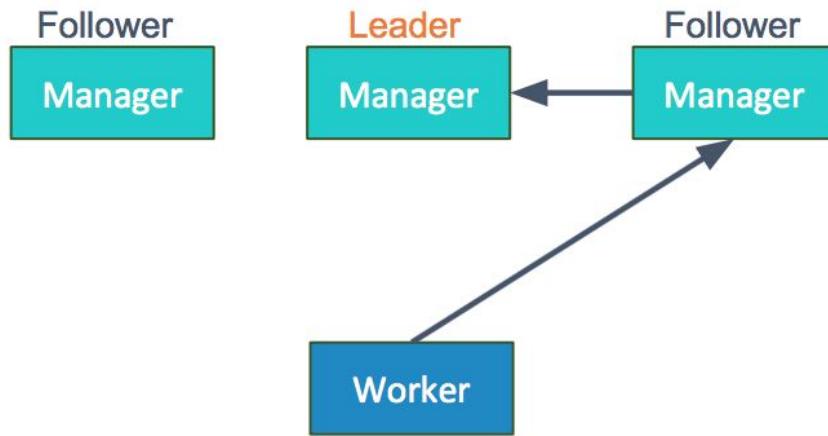
Swarm Architecture



Swarm High Availability

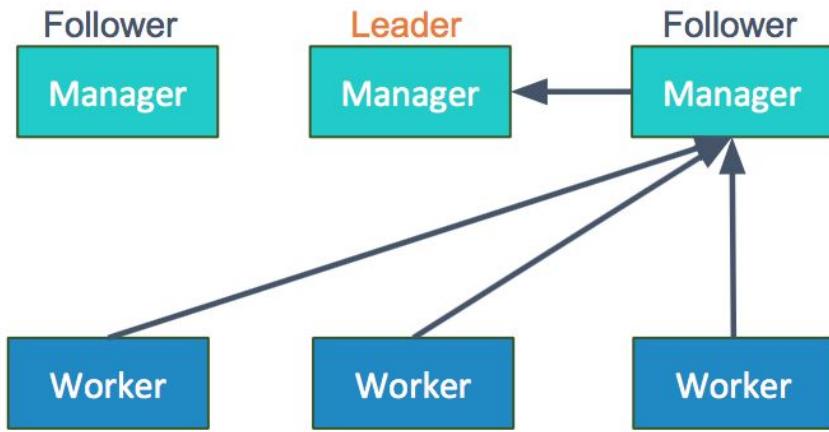
Controller and replicas	Failures tolerated
1	0
3	1
5	2
7	3

Swarm Workers and Managers



- **Worker** can connect to any **Manager**
- **Followers** will forward traffic to the **Leader**

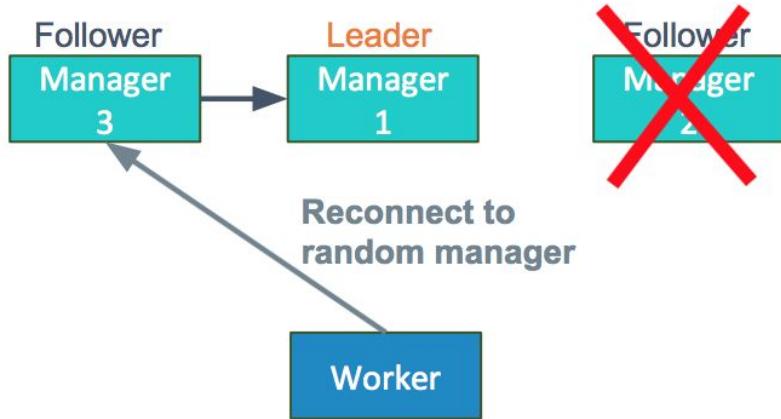
Swarm Workers and Managers



- **Followers** multiplex all **workers** to the **Leader** using a single connection
- Backed by **gRPC** channels (HTTP/2 streams)
- Reduces **Leader** networking load by spreading the connections evenly

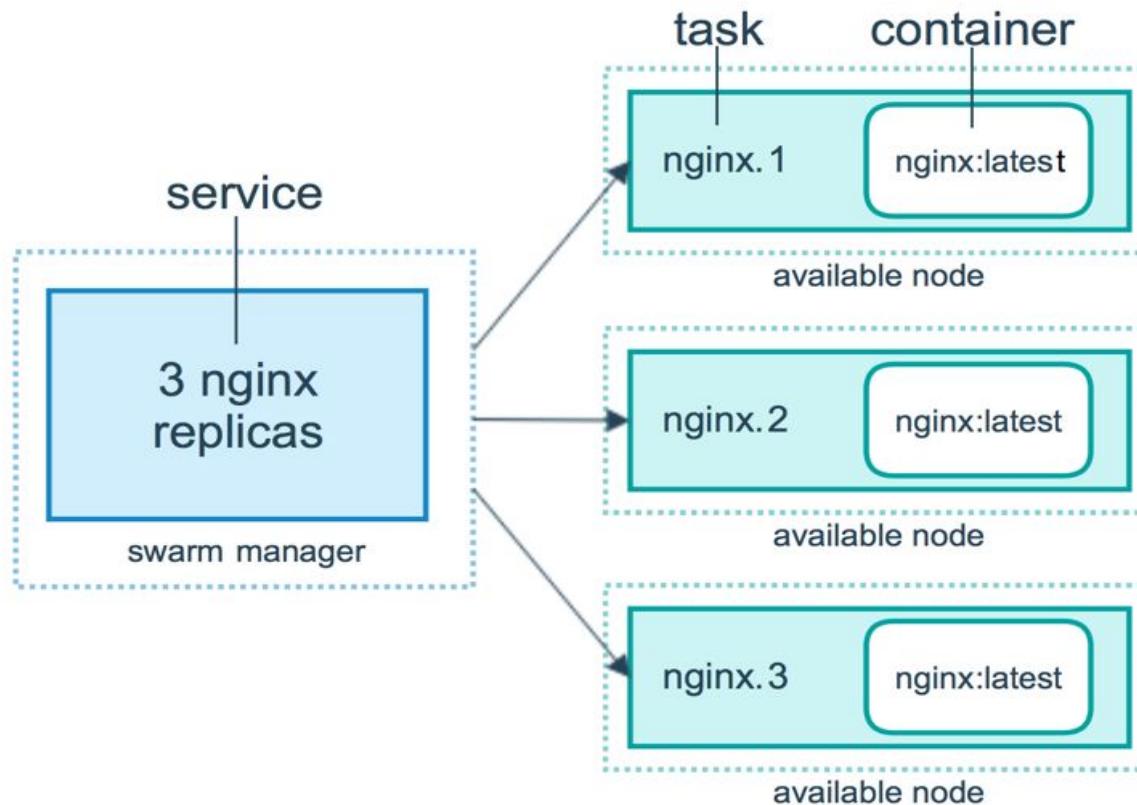
Example: On a cluster with 10,000 workers and 5 managers, each will only have to handle about 2,000 connections. Each follower will forward its 2,000 workers using a single socket to the leader.

Swarm Workers and Managers

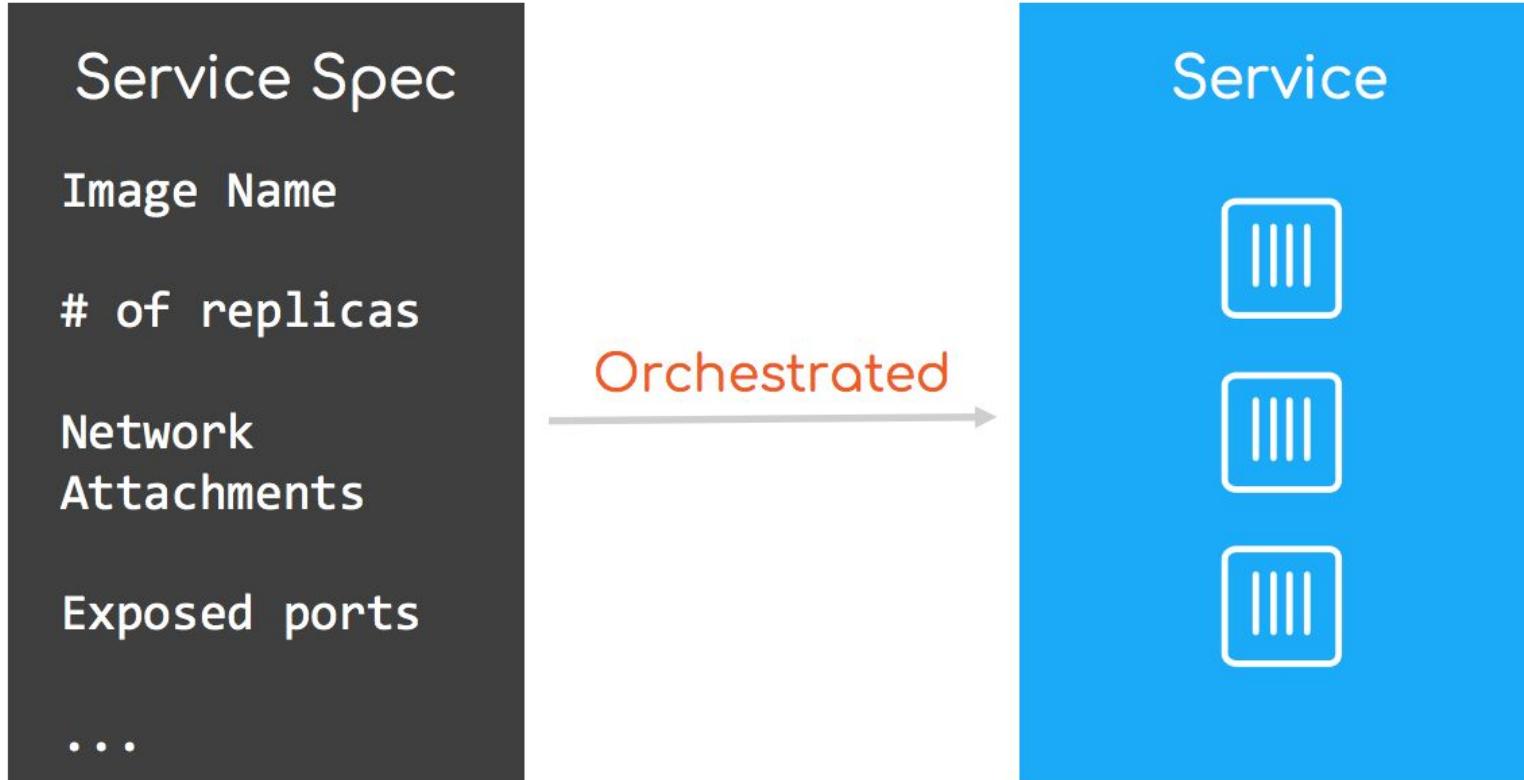


- Manager sends list of all managers' addresses to Workers
- When a new manager joins, all workers are notified
- Upon manager failure, workers will reconnect to a different manager

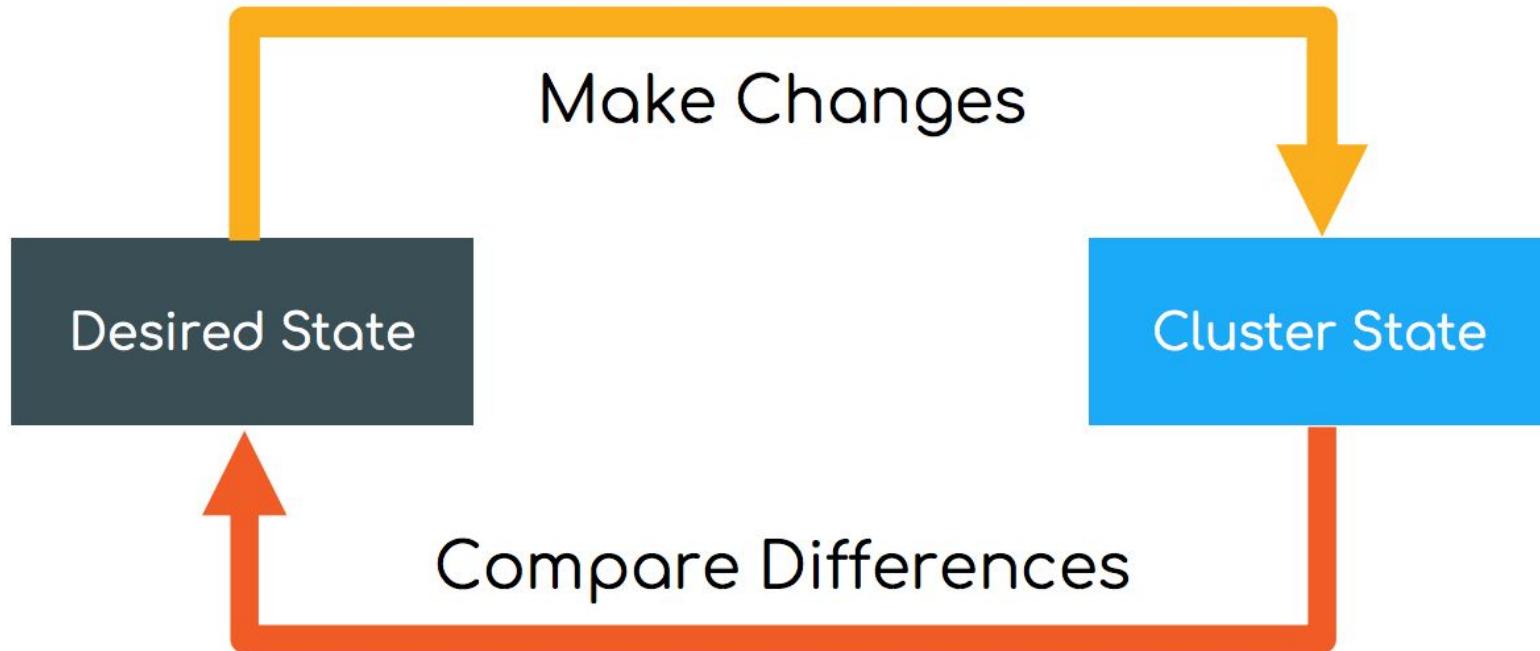
Docker Tasks: Atomic Unit of Orchestration



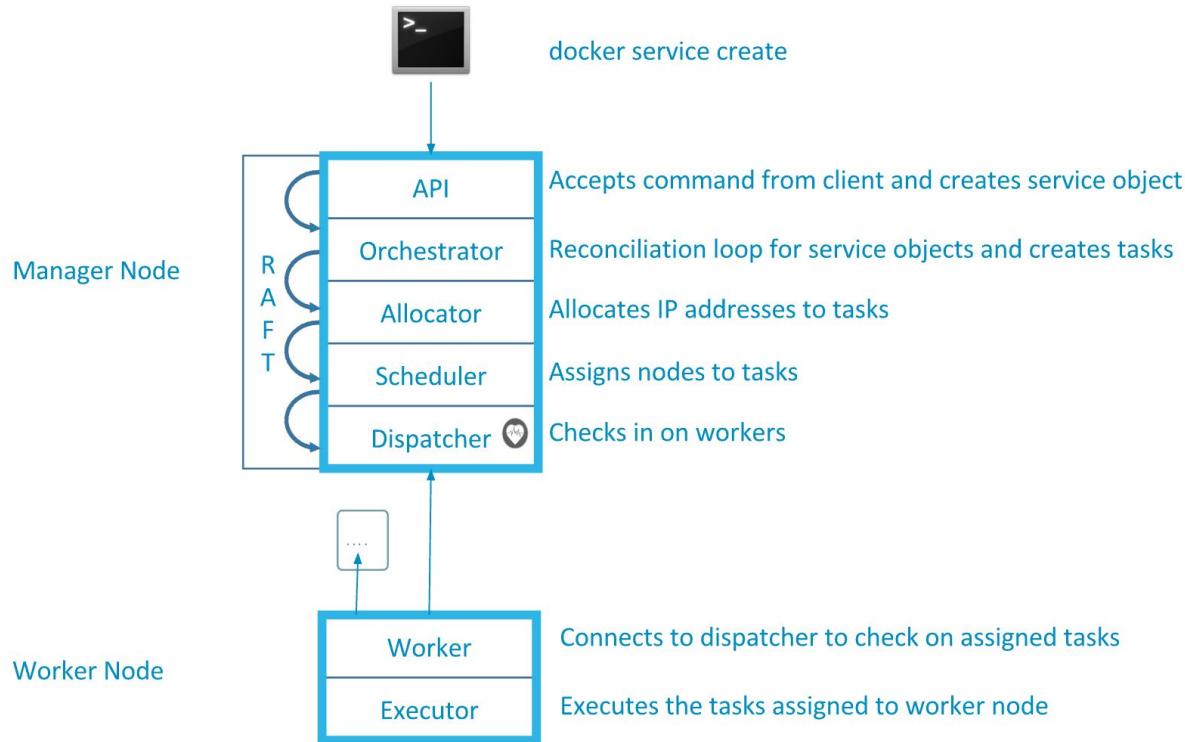
Docker Services



Task Lifecycle Management



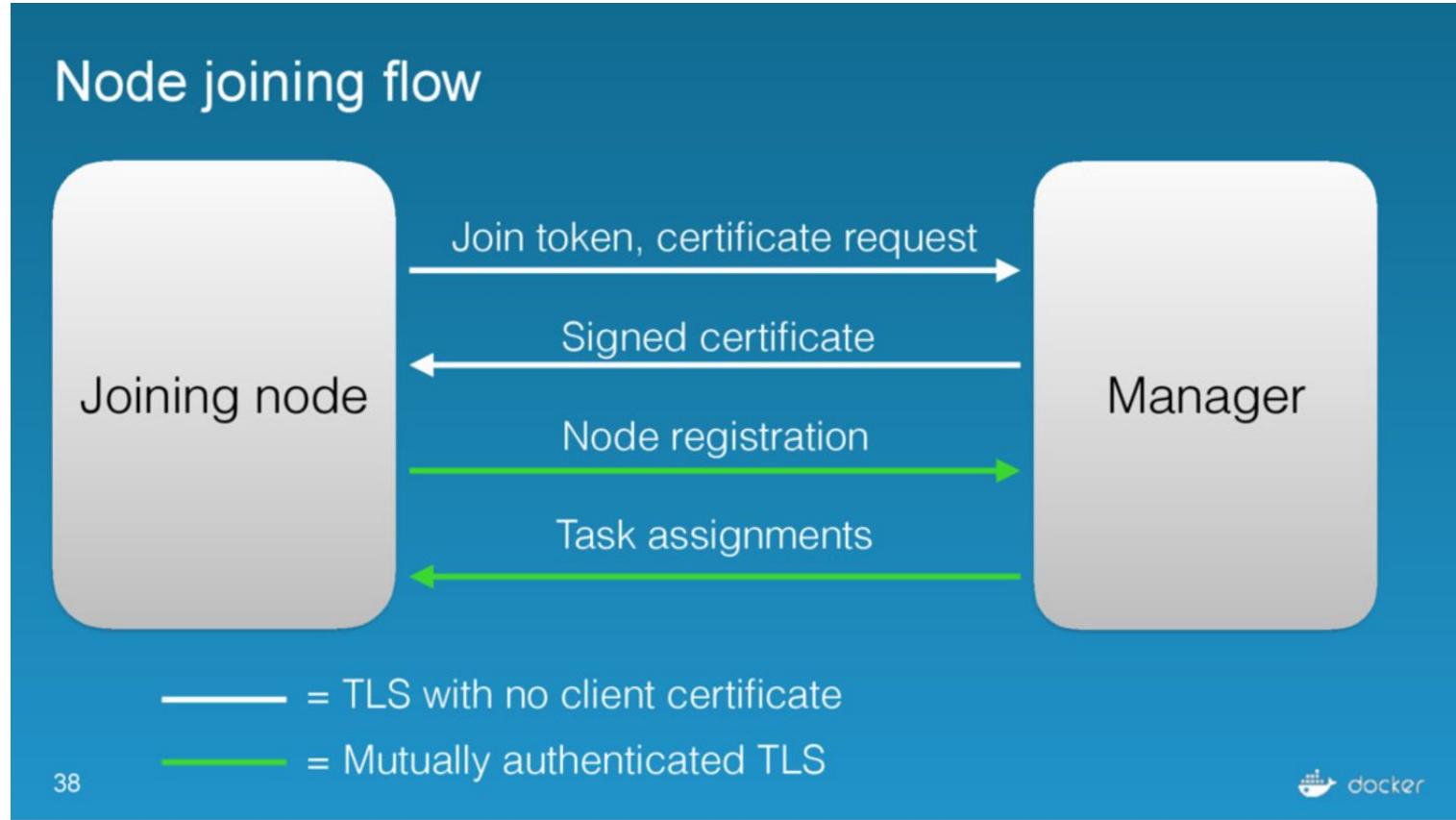
Task State - (docker service ps <service>)



Security Model

- All Swarm connections are encrypted with TLS by default (no way to disable it!)
- Each node is identified by its certificate (CN = node ID)
- By default, each manager acts as a CA with same key
- Join token used to authenticate to obtain cert
- Join token includes digest of root cert to secure against MITM
- Can rotate join tokens to revoke older ones
- Automatically renew certificates (every 90 days)

Node Join Operation



Swarm Certificates

```
## Swarm Manager Node
```

```
$ openssl x509 -noout -text -in swarm-node.crt | grep 'Subject\|Issuer'  
Issuer: CN=swarm-ca  
Subject: O=jbwjyw1e8p2g97g4szuoi5q1j, OU=swarm-manager, CN=unctml1t6jsrnd7v2b9cnohsm  
Subject Public Key Info:  
    X509v3 Subject Key Identifier:  
    X509v3 Subject Alternative Name:
```

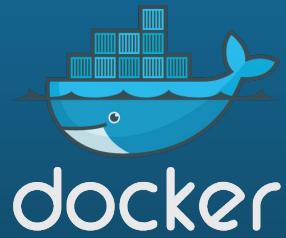
```
## Swarm Worker Node
```

```
$ openssl x509 -noout -text -in swarm-node.crt | grep 'Subject\|Issuer'  
Issuer: CN=swarm-ca  
Subject: O=jbwjyw1e8p2g97g4szuoi5q1j, OU=swarm-worker, CN=ttj80hvbbjolny5g1kye3g5le  
Subject Public Key Info:  
    X509v3 Subject Key Identifier:  
    X509v3 Subject Alternative Name:
```

Managing Swarm Nodes

```
$ docker swarm init  
$ docker swarm join-token worker|manager  
$ docker swarm join  
$ docker node ls  
$ docker node inspect  
$ docker swarm leave  
$ docker node remove
```

Docker Networking



Docker Networking Design Philosophy

Portability

.....

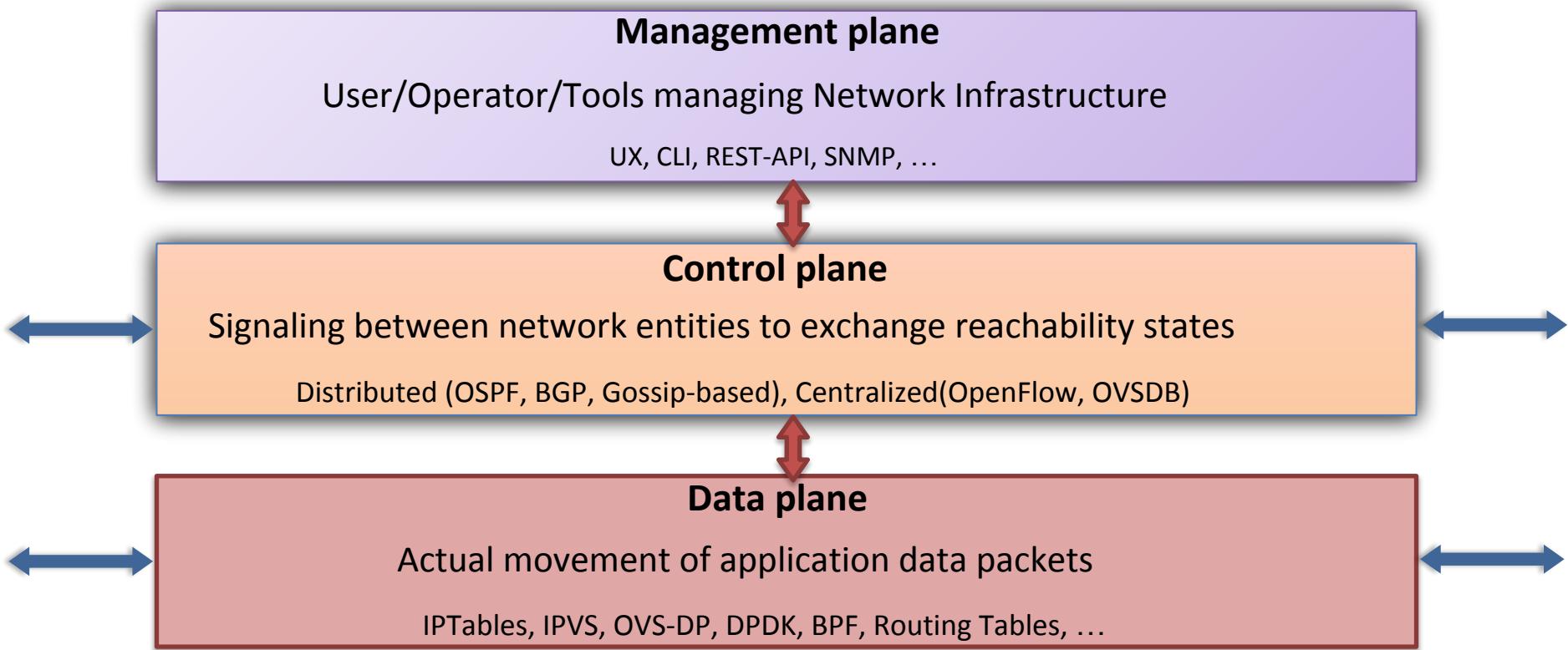
**Consistent behavior
of applications** in
different networking
environments.

Extensibility

.....

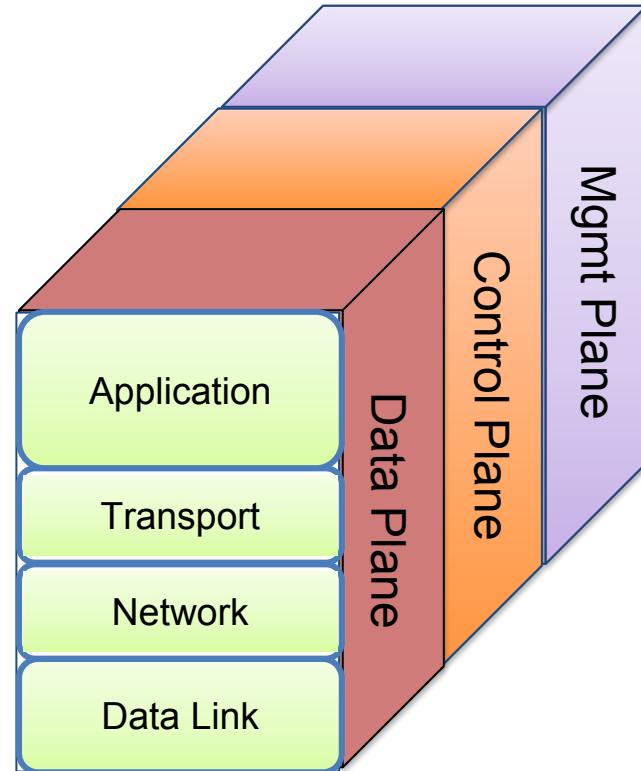
**Batteries included
but swappable** with
Plugin API Design

Networking Planes

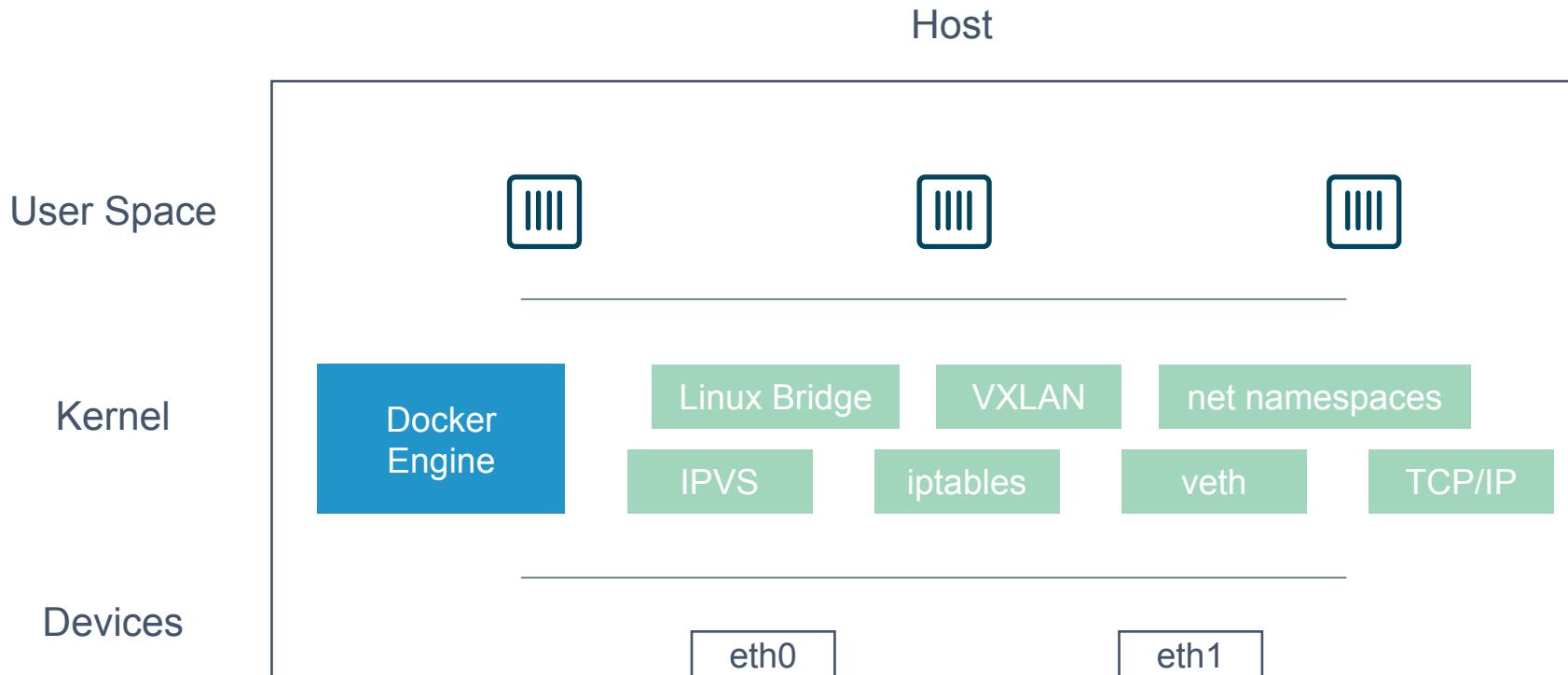


Docker networking

- Provides portable application services
 - Service-Discovery
 - Load-Balancing
- Built-in and pluggable network drivers
 - Overlay, macvlan, bridge
 - Remote Drivers / Plugins
- Built-in Management plane
 - API, CLI
 - Docker Stack / Compose
- Built-in distributed control plane
 - Gossip based
- Encrypted Control & Data plane



Docker Networking *is* Linux (and Windows) Networking



Docker Networking on Linux and Windows

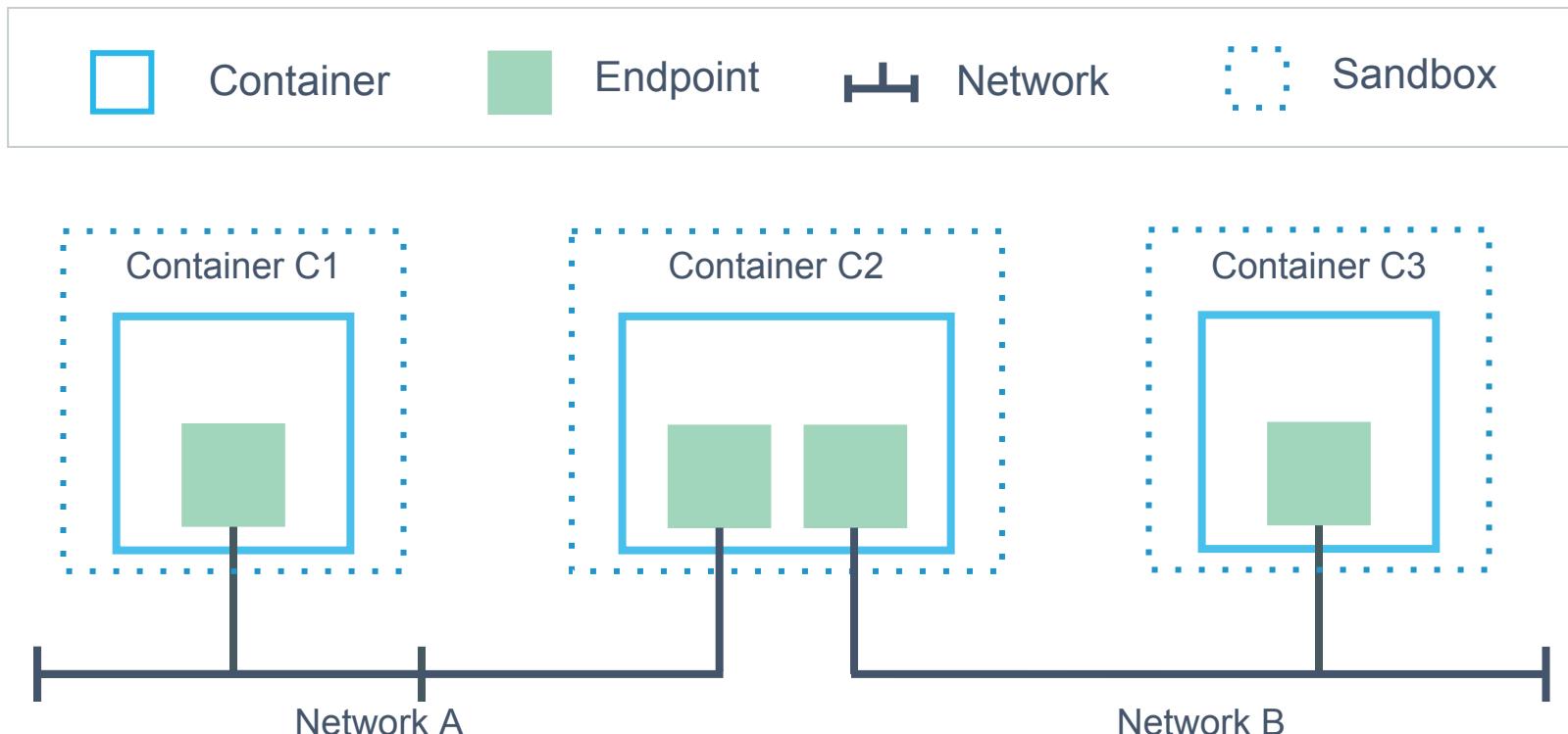
Linux

- Network Namespace
- Linux Bridge
- Virtual Ethernet Devices
- IP Tables

Windows

- Network Compartments
- VSwitch
- Virtual nics
- Firewall & VFP Rules

Containers and the CNM



Native Docker Networking Drivers

```
$ docker info
```

...

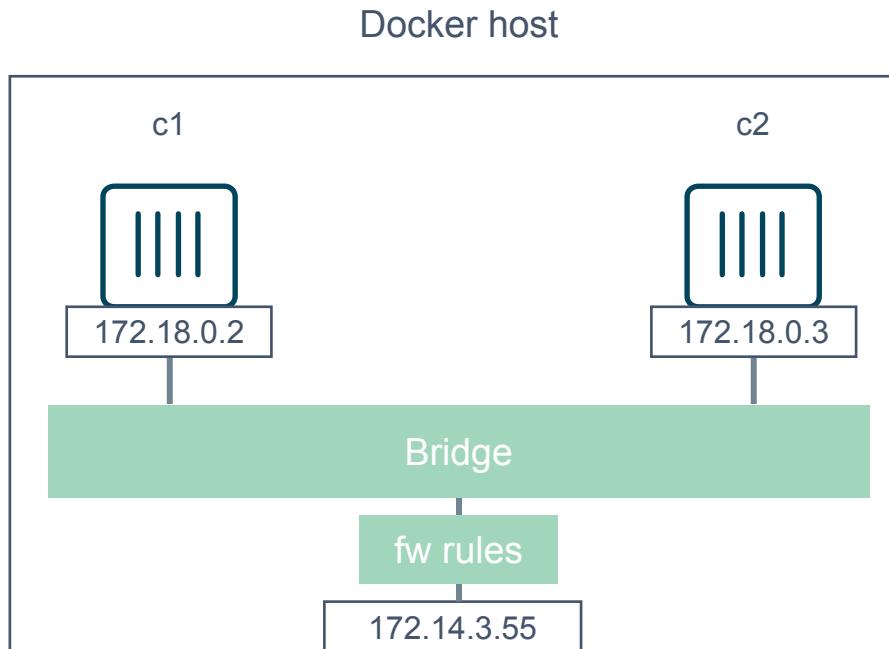
Plugins:

Volume: local

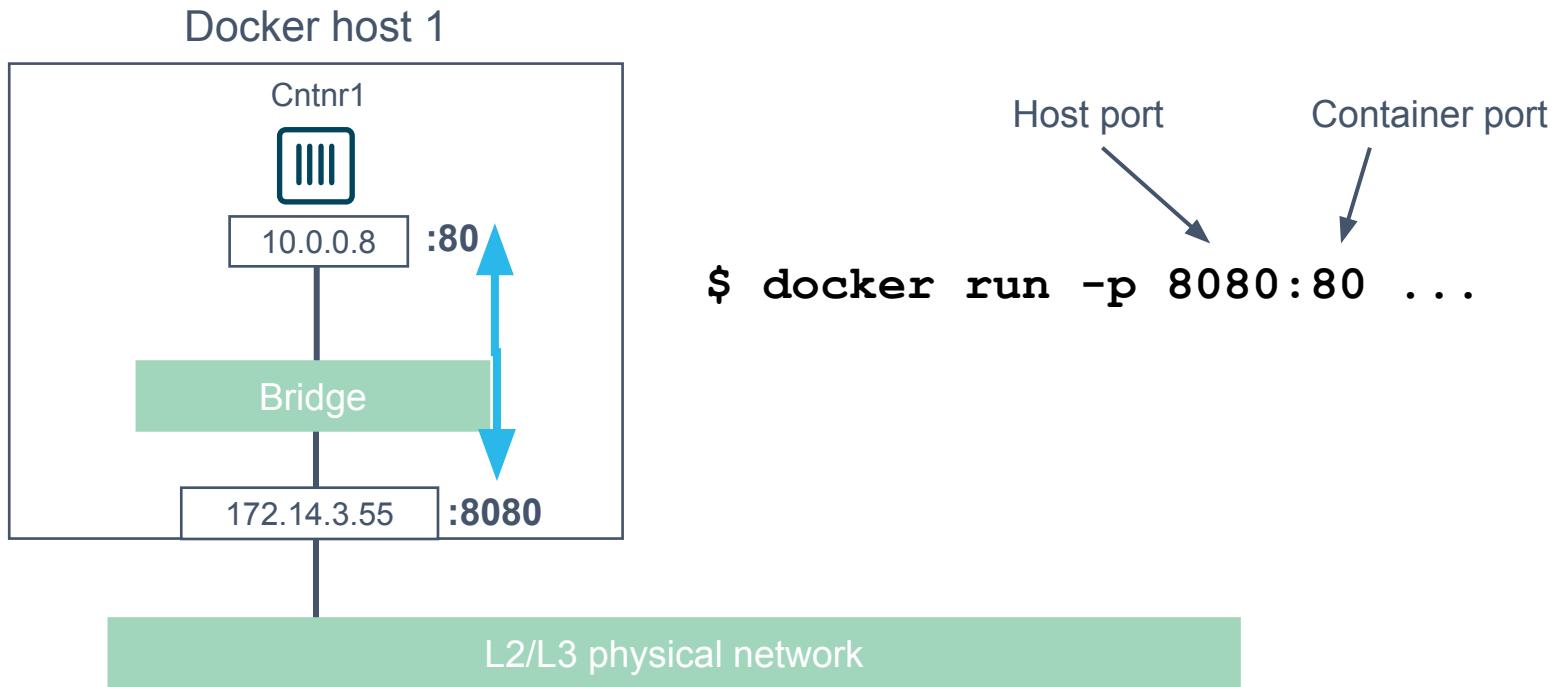
Network: bridge host ipvlan macvlan null overlay

...

Using Docker Networks

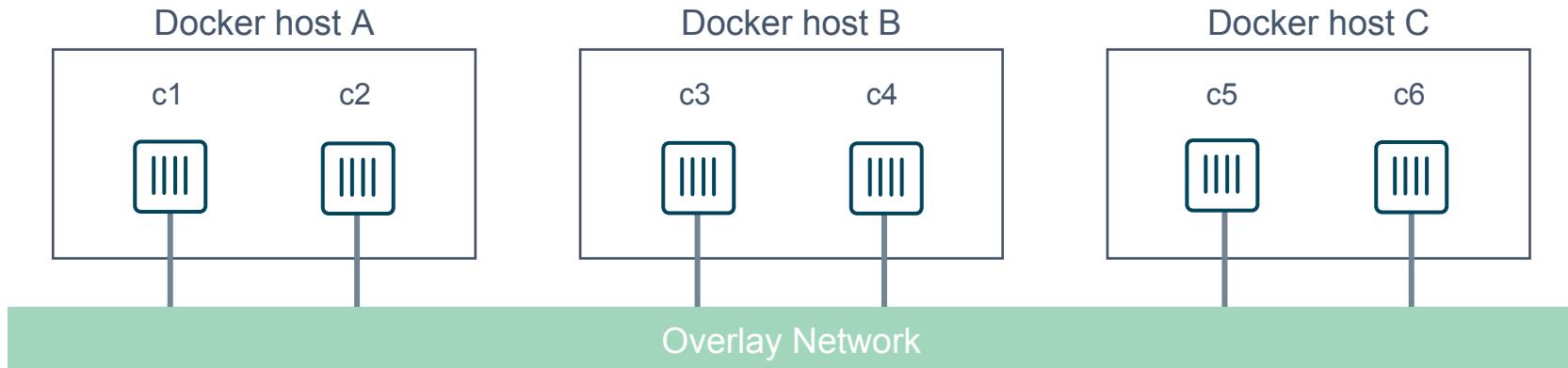


Docker Bridge Networking and Port Mapping



What is Docker Overlay Networking?

The **overlay** driver enables simple and secure **multi-host** networking

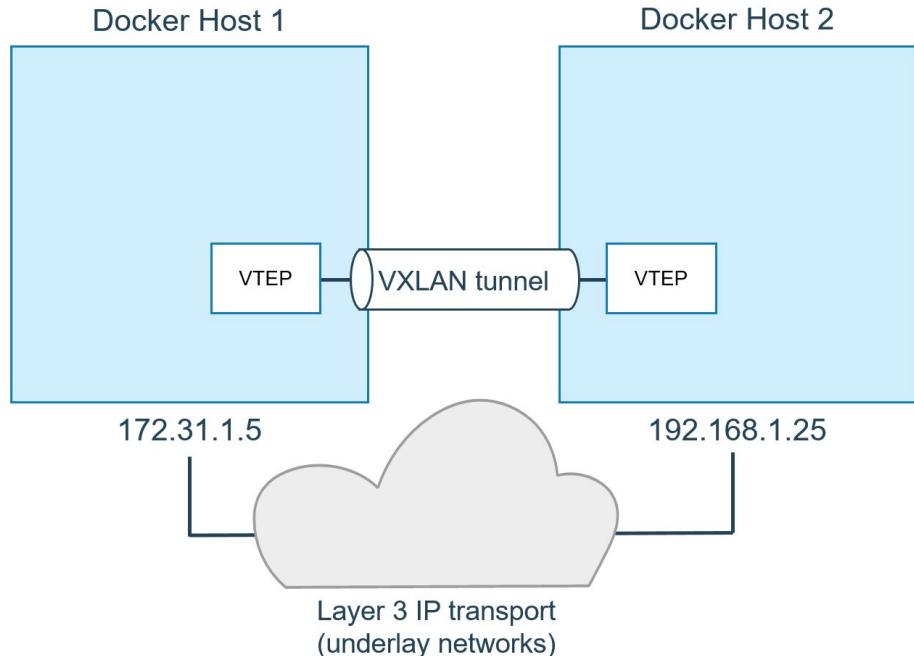


All containers on the **overlay** network can communicate!

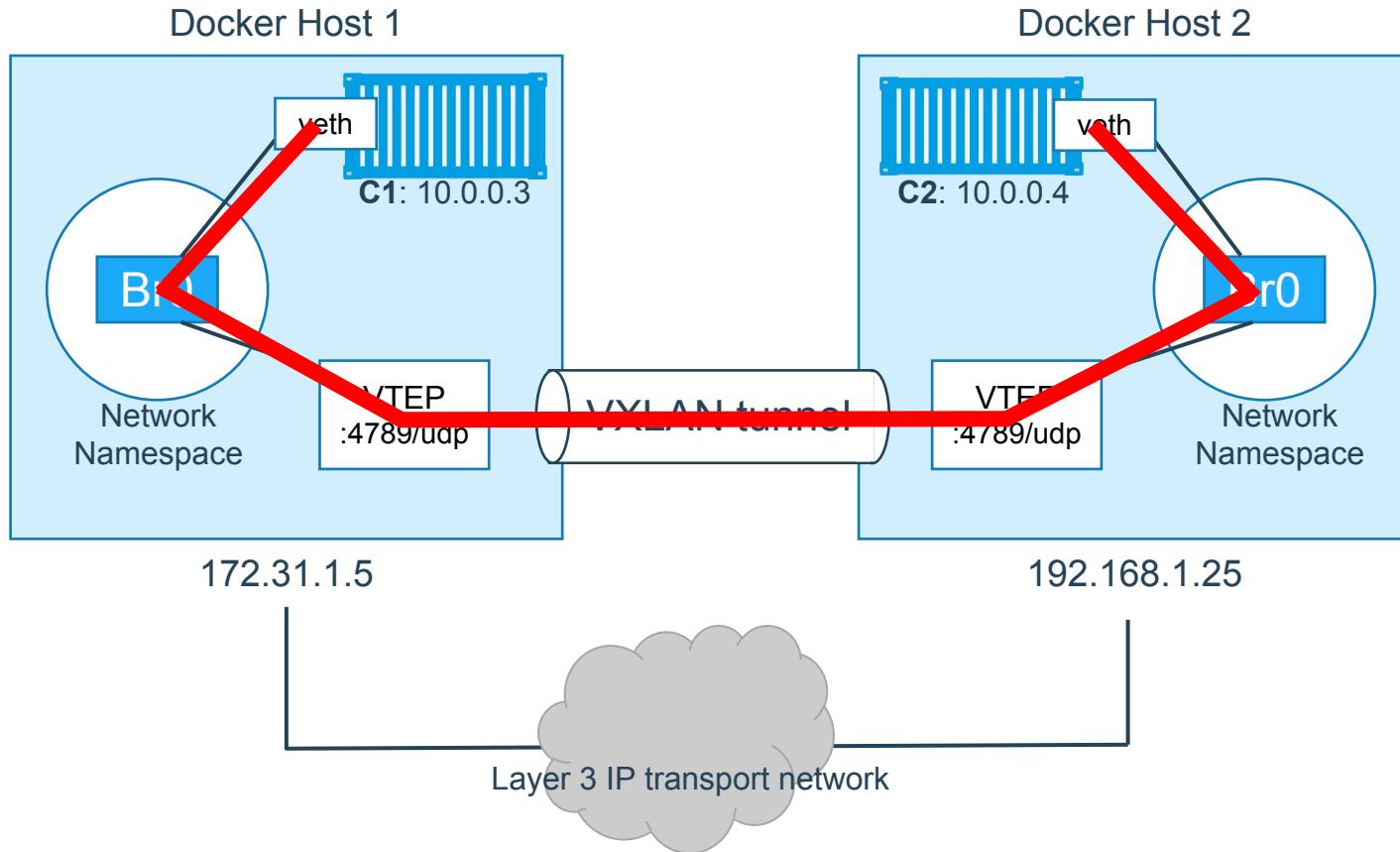
VXLAN

- The **overlay** driver uses VXLAN technology
- A **VXLAN tunnel** is created on top of **underlay network(s)**
- At each end of the tunnel is a VXLAN tunnel end point (**VTEP**)
- The **VTEP** performs encapsulation and de-encapsulation
- The **VTEP** exists in the Docker Host's network namespace

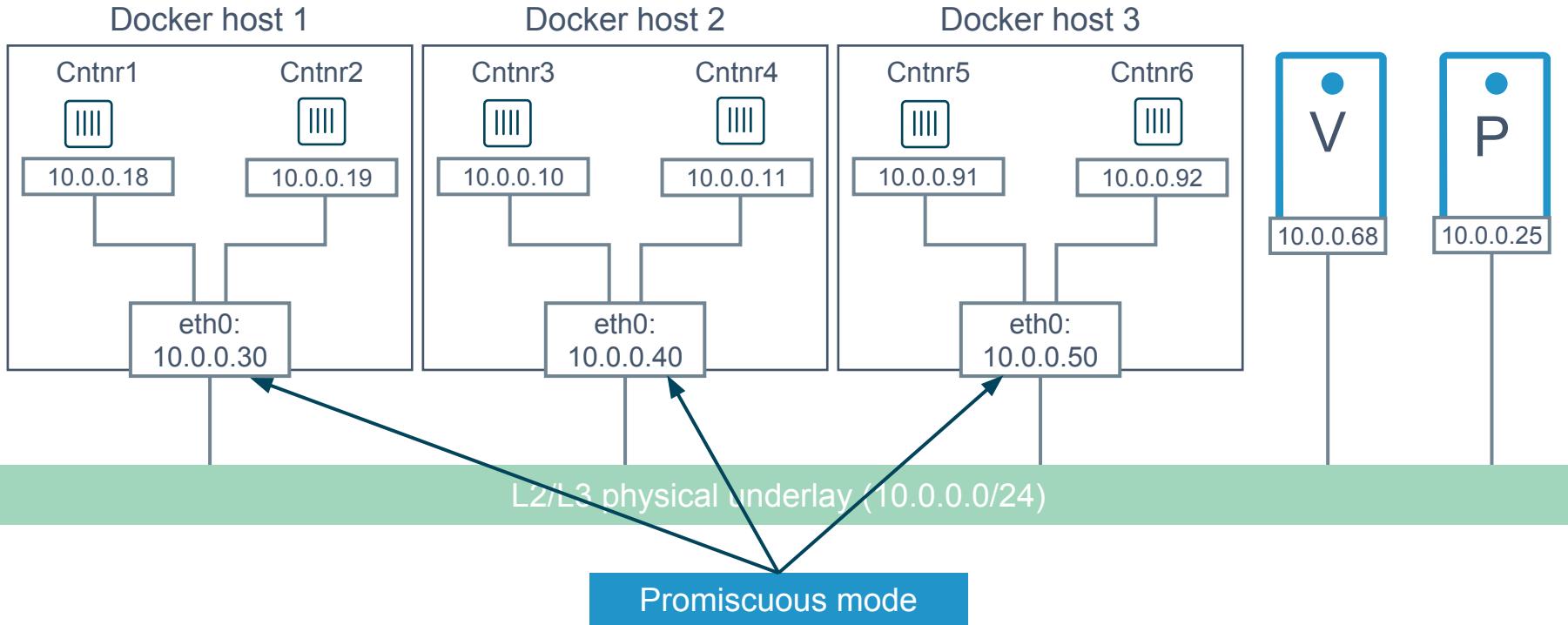
Docker Overlay Driver



Docker Overlay in Detail

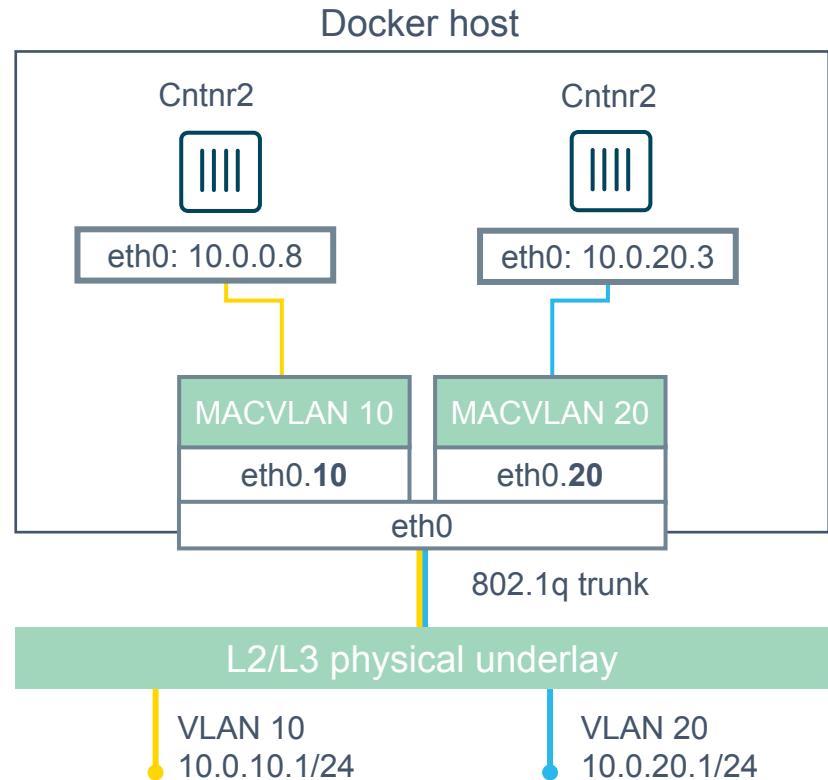


MACVLAN Driver (single VLAN, no subinterface)



MACVLAN Driver (sub-interfaces, VLAN Trunks)

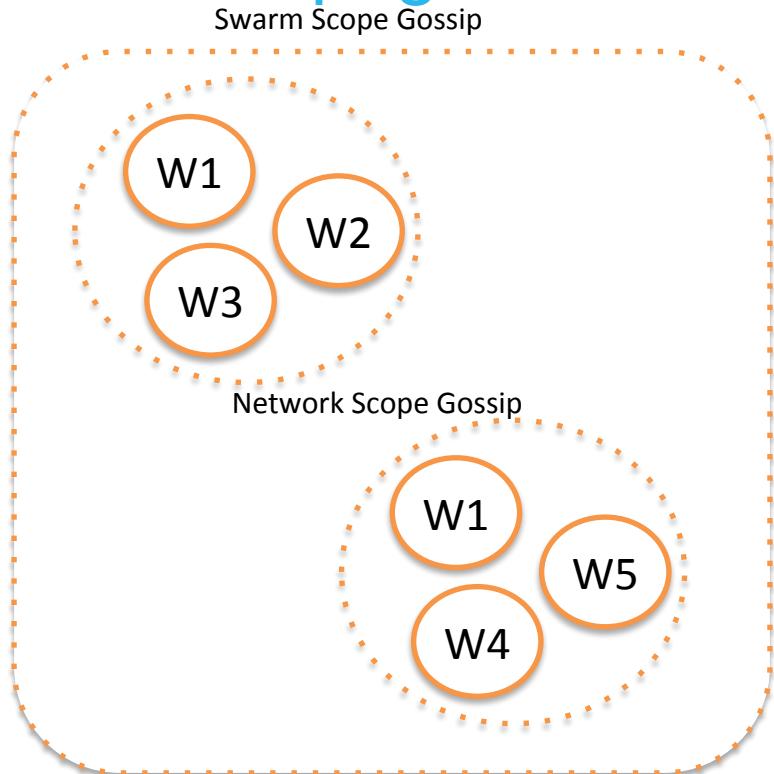
- MACVLAN uses **sub-interfaces** to process 802.1Q VLAN tags.
- In this example, two sub-interfaces are used to enable two separate VLANs
- Yellow lines represent VLAN 10
- Blue lines represent VLAN 20



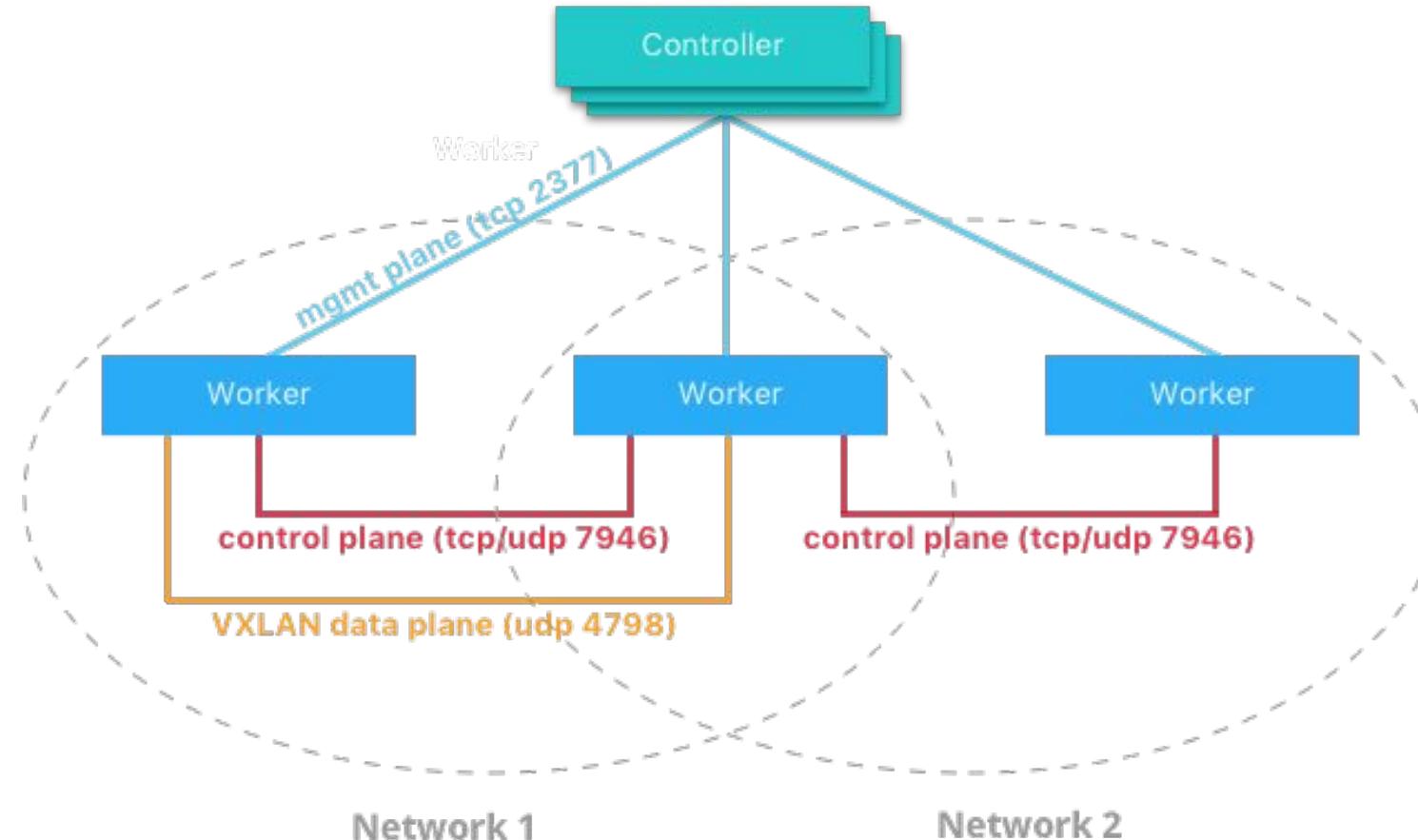
Gossip

- Eventually consistent
- State dissemination through de-centralized events
 - Service Registration
 - Load-Balancer configs
 - Routing states
- Fast convergence
 - $\sim O(\log n)$
- Highly scalable
- Continues to function even if all managers are Down

Decentralized Event Propogation



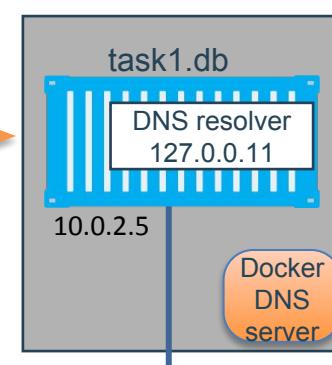
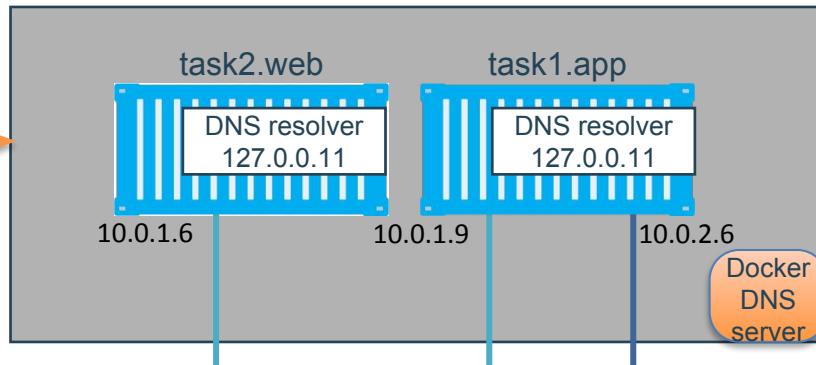
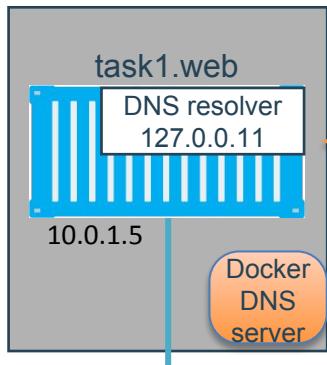
Mgmt, Control, and Data Plane in Practice



Worker1

Worker2

Worker3



Gossip

Gossip

demo_frontend overlay network (vxlan-id 4097)

demo_backend overlay network (vxlan-id 4098)

Service Discovery states

web 10.0.1.4 (vip)
app 10.0.1.8 (vip)
task1.web 10.0.1.5
task2.web 10.0.1.6
task1.app 10.0.1.9

Service Discovery states

web 10.0.1.4 (vip)
app 10.0.1.8 (vip)
task1.web 10.0.1.5
task2.web 10.0.1.6
task1.app 10.0.1.9

Service Discovery states

db 10.0.2.4 (vip)
app 10.0.2.8 (vip)
task1.db 10.0.2.5
task1.app 10.0.2.6

Service Discovery states

db 10.0.2.4 (vip)
app 10.0.2.8 (vip)
task1.db 10.0.2.5
task1.app 10.0.2.6

Routing states

10.0.1.6 :{Worker2,4097}
10.0.1.9 :{Worker2,4097}

Routing states

10.0.1.5 :{Worker1,4097}

Routing states

10.0.2.5 :{Worker3,4098}

Routing states

10.0.2.6 :{Worker2,4098}

Docker Stack Deploy

```
$ docker stack deploy -c d.yml demo
```

Creating network demo_frontend

Creating network demo_backend

Creating service demo_web

Creating service demo_app

Creating service demo_db

- Swarm scope - network resources that are owned and managed centrally by the controllers
- Local scope - network resources that are owned and managed by the worker node

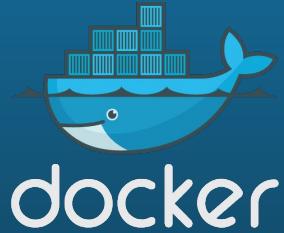
Local scope - bridge, macvlan, host

Swarm scope - Overlay, Contiv

- Manager only operation
- Reserves network resources at mgmt plane such as subnet and vxlan-id. No impact to the data-plane yet.

- Manager reserves service and task resources : Service VIP and Task IPs
- Tasks Scheduled to swarm workers
- Network scoped Service Registration on Docker DNS server
 - Service name -> VIP
 - Task name -> Task IP
 - task.Service-Name -> All Task IPs
- Exchange SD & LB states via Gossip
- **Prepare Data-plane***
- Call Driver APIs and exchange driver states via Gossip

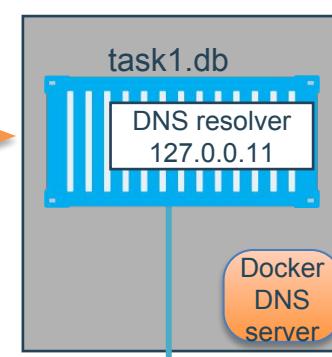
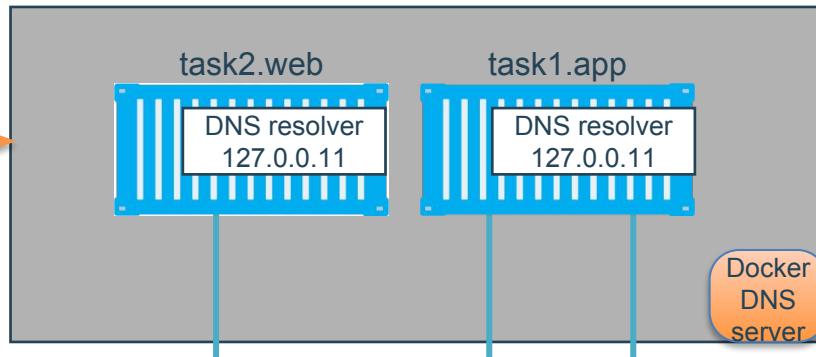
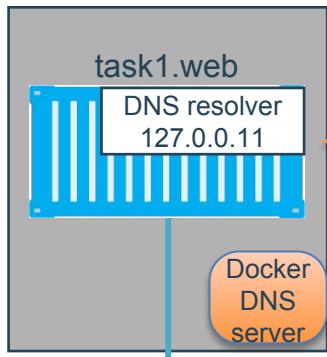
Docker Networking Service Discovery



Worker1

Worker2

Worker3



Gossip

Gossip

demo_frontend overlay network (vxlan-id 4097)

demo_backend overlay network (vxlan-id 4098)

Service Discovery states

web 10.0.1.4 (vip)
app 10.0.1.8 (vip)
task1.web 10.0.1.5
task2.web 10.0.1.6
task1.app 10.0.1.9

Service Discovery states

web 10.0.1.4 (vip)
app 10.0.1.8 (vip)
task1.web 10.0.1.5
task2.web 10.0.1.6
task1.app 10.0.1.9

Service Discovery states

db 10.0.2.4 (vip)
app 10.0.2.8 (vip)
task1.db 10.0.2.5
task1.app 10.0.2.6

Service Discovery states

db 10.0.2.4 (vip)
app 10.0.2.8 (vip)
task1.db 10.0.2.5
task1.app 10.0.2.6

Routing states

10.0.1.6 :{Worker2,4097}
10.0.1.9 :{Worker2,4097}

Routing states

10.0.1.5 :{Worker1,4097}

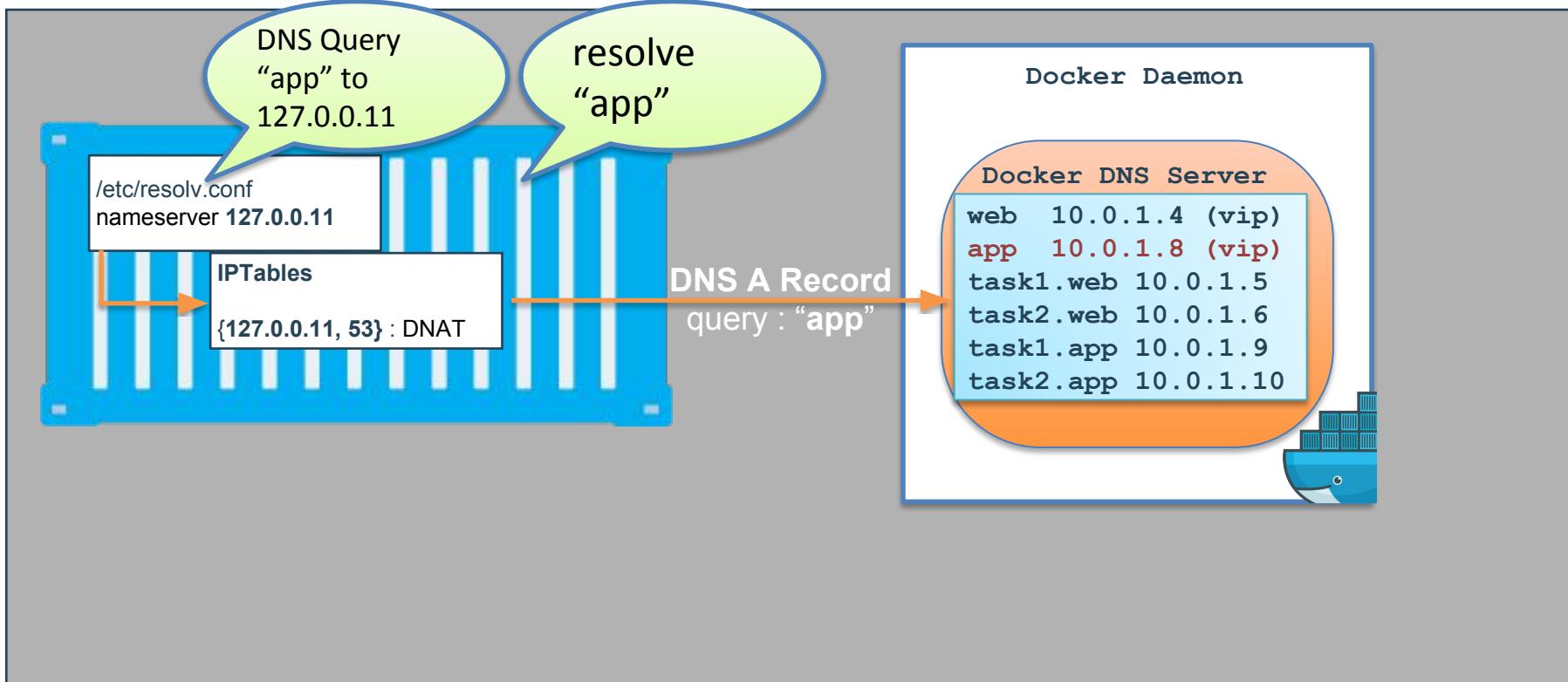
Routing states

10.0.2.5 :{Worker3,4098}

Routing states

10.0.2.6 :{Worker2,4098}

Dissecting DNS Lookups



Dissecting DNS Lookups

task1.web

/etc/resolv.conf
nameserver 127.0.0.11

IPTables

{127.0.0.11, 53} : DNAT

DNS A Record
response : "app"
10.0.1.8

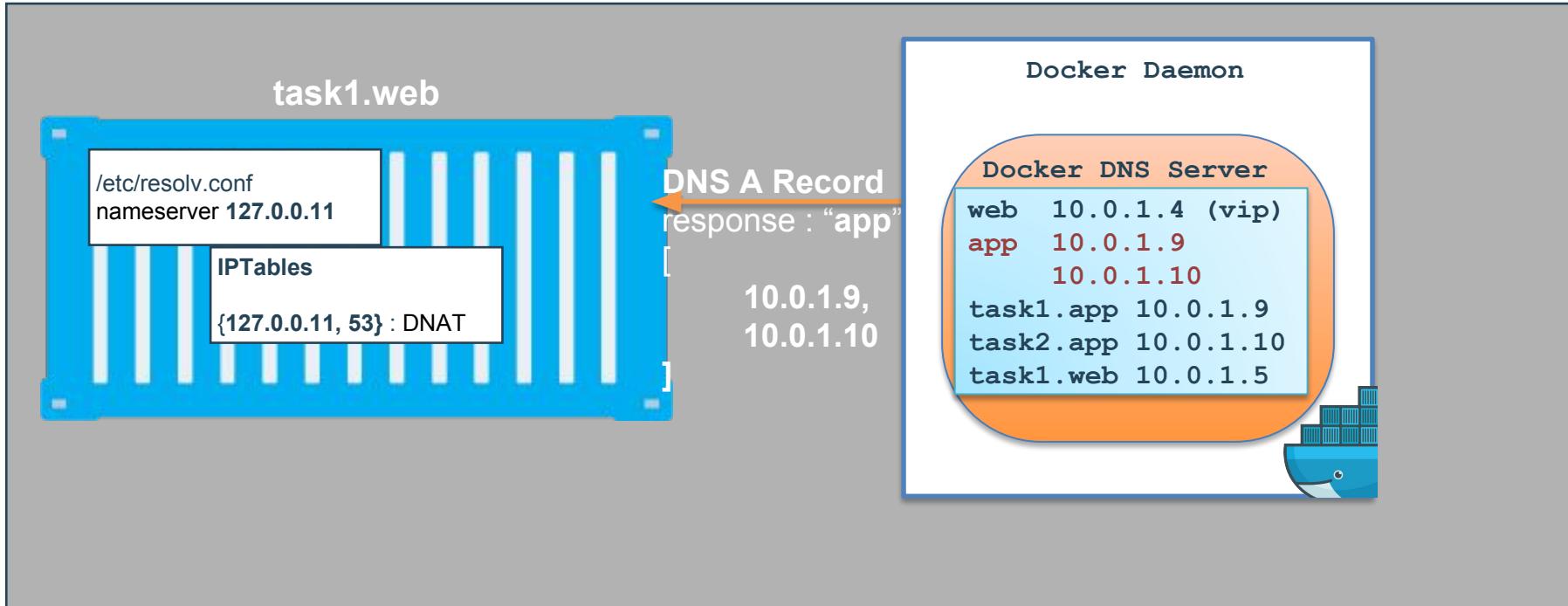
Docker Daemon

Docker DNS Server

web 10.0.1.4 (vip)
app 10.0.1.8 (vip)
task1.web 10.0.1.5
task2.web 10.0.1.6
task1.app 10.0.1.9
task2.app 10.0.1.10

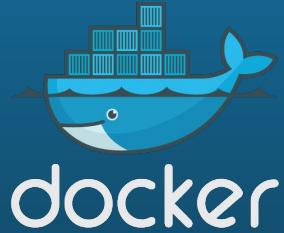
DNS Round Robin Mode

```
docker service create --name=app --endpoint-mode=dns-rr demo/my-app
```



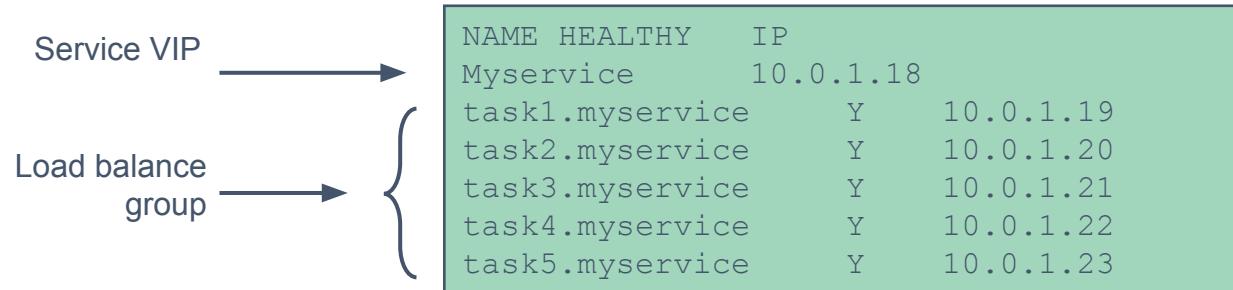
Docker Networking

Load Balancing



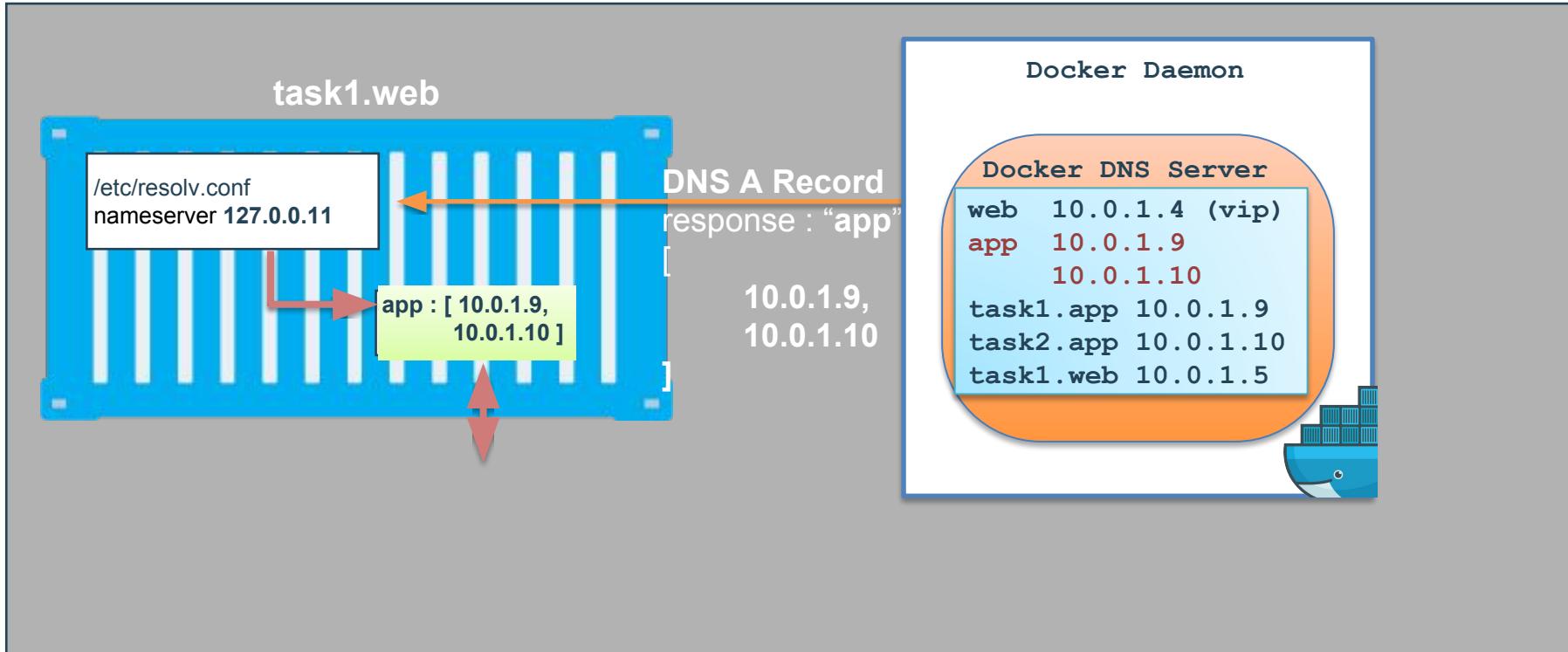
Internal LB: Service Virtual IP (VIP) Load Balancing

- Every **service** gets a **VIP** when it's created
 - This stays with the service for its entire life
- Lookups against the VIP get load-balanced across all **healthy tasks** in the service
- Behind the scenes it uses Linux kernel **IPVS** to perform transport layer load balancing
- `docker service create --name=app --endpoint-mode=vip demo/my-app`
- `docker inspect <service>` (shows the service VIP)

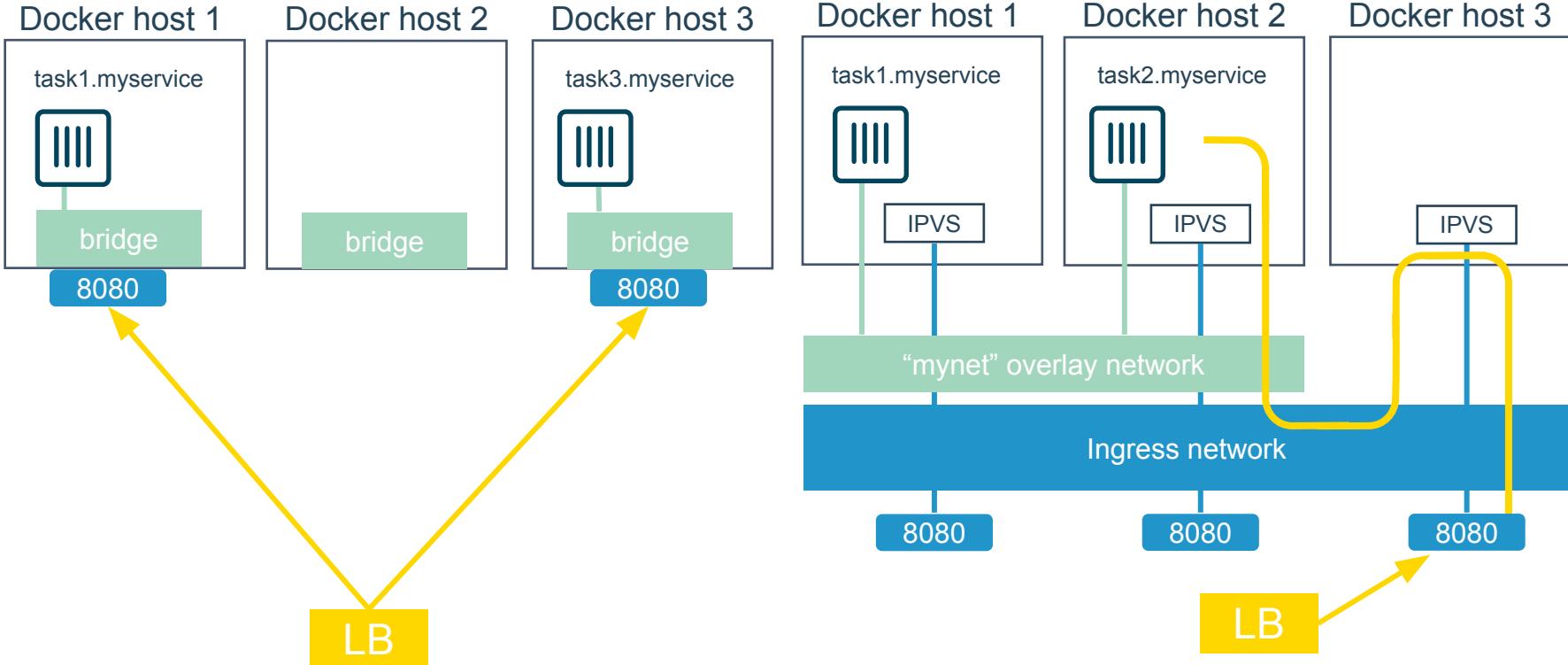


Internal LB: DNS RR Load Balancing

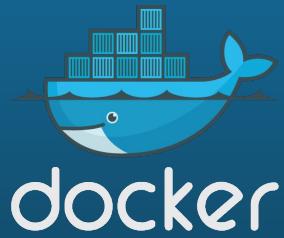
```
docker service create --name=app --endpoint-mode=dns-rr demo/my-app
```



Publish Mode: Host vs Ingress



Docker Security



Docker secures your software supply chain



For Developers

- Does not hinder speed or creativity
- Accelerate secure development

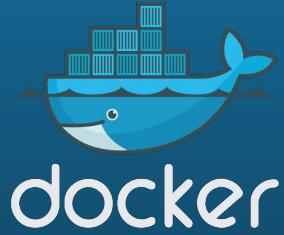
For IT ops

- Flexible and granular controls
- Proactive risk management



Docker Security

Secure Platform



“Gartner asserts that applications deployed in containers are more secure than applications deployed on the bare OS.”

<http://blogs.gartner.com/joerg-fritsch/can-you-operationalize-docker-containers/>



Secure platform

All Linux
isolation
capabilities

- pid namespace
- mnt namespace
- net namespace
- uts namespace
- user namespace
- pivot_root
- uid/gid drop
- cap drop
- all cgroups
- selinux
- apparmor
- seccomp



1. Out of the box
default settings
and profiles

2. Granular
controls to
customize settings

**Secure by
default**



Available Container Security Features, Requirements and Defaults			
Security Feature	LXC 2.0	Docker 1.11	CoreOS Rkt 1.3
User Namespaces	Default	Optional	Experimental
Root Capability Dropping	Weak Defaults	Strong Defaults	Weak Defaults
Procfs and Sysfs Limits	Default	Default	Weak Defaults
Cgroup Defaults	Default	Default	Weak Defaults
Seccomp Filtering	Weak Defaults	Strong Defaults	Optional
Custom Seccomp Filters	Optional	Optional	Optional
Bridge Networking	Default	Default	Default
Hypervisor Isolation	Coming Soon	Coming Soon	Optional
MAC: AppArmor	Strong Defaults	Strong Defaults	Not Possible
MAC: SELinux	Optional	Optional	Optional
No New Privileges	Not Possible	Optional	Not Possible
Container Image Signing	Default	Strong Defaults	Default
Root Interation Optional	True	False	Mostly False

Source: source: “Understanding and Hardening Linux Containers”, page 96



Docker Bench

```
1. diogo@visby: ~ (zsh)

[WARN] * Privileged Port in use: 80 in 005543c21aeb
[PASS] 5.10 - Do not use host network mode on container
[WARN] 5.11 - Limit memory usage for container
[WARN] * Container running without memory restrictions: 57b4f86898d9
[WARN] * Container running without memory restrictions: fa03oe189d81
[WARN] * Container running without memory restrictions: 95292abe9108
[WARN] * Container running without memory restrictions: 005543c21aeb
[WARN] * Container running without memory restrictions: 444c2c5773ba
[WARN] * Container running without memory restrictions: 33873424dd94
[WARN] 5.12 - Set container CPU priority appropriately
[WARN] * Container running without CPU restrictions: 57b4f86898d9
[WARN] * Container running without CPU restrictions: fa03oe189d81
[WARN] * Container running without CPU restrictions: 95292abe9108
[WARN] * Container running without CPU restrictions: 005543c21aeb
[WARN] * Container running without CPU restrictions: 444c2c5773ba
[WARN] * Container running without CPU restrictions: 33873424dd94
[WARN] 5.13 - Mount container's root filesystem as read only
[WARN] * Container running with root FS mounted R/W: 57b4f86898d9
[WARN] * Container running with root FS mounted R/W: fa03oe189d81
[WARN] * Container running with root FS mounted R/W: 95292abe9108
[WARN] * Container running with root FS mounted R/W: 005543c21aeb
[WARN] * Container running with root FS mounted R/W: 444c2c5773ba
[WARN] * Container running with root FS mounted R/W: 33873424dd94
[WARN] 5.14 - Bind incoming container traffic to a specific host interface
[WARN] * Port being bound to wildcard IP: 0.0.0.0 in 005543c21aeb
[PASS] 5.15 - Do not set the 'on-failure' container restart policy to always
[PASS] 5.16 - Do not share the host's process namespace
[PASS] 5.17 - Do not share the host's IPC namespace
[PASS] 5.18 - Do not directly expose host devices to containers
[INFO] 5.19 - Override default ulimit at runtime only if needed
```

Ensure secure host configurations

- Aligned to recommendations in Center for Internet Security's Benchmark for Docker Engine 1.11
- Automates checking your host configs against the benchmark recommendations

Easy to use

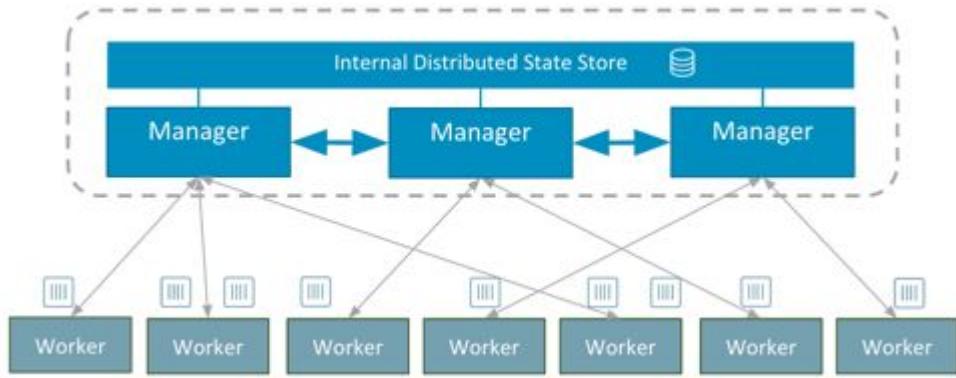
- Available to run as a container or using a Compose file

www.dockerbench.com



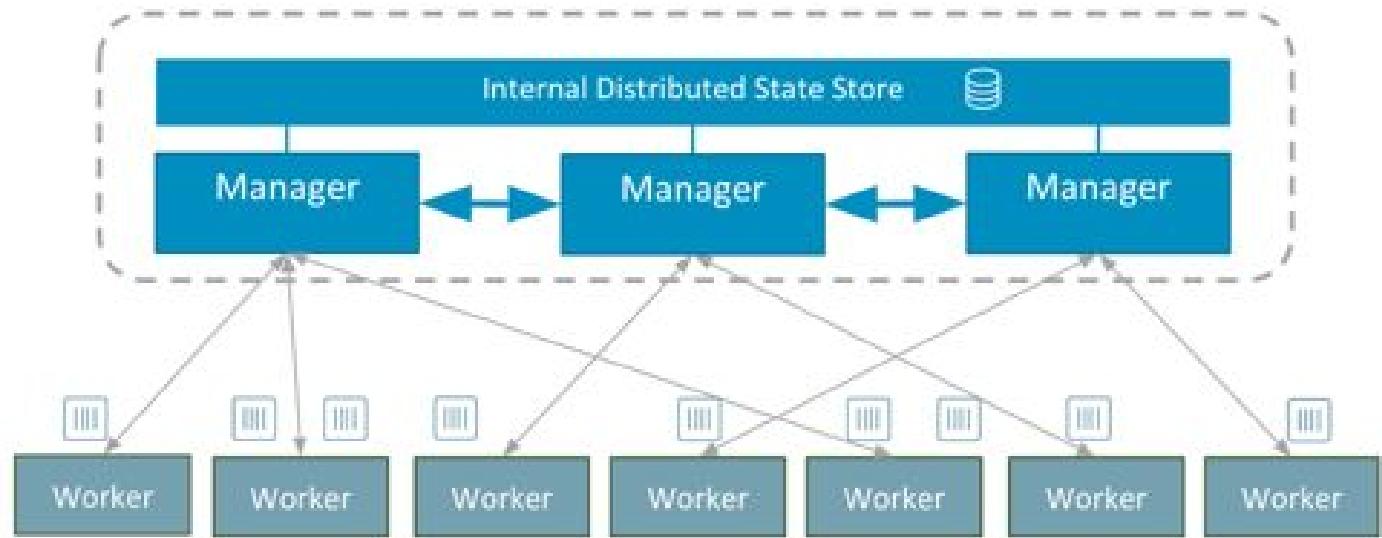
Secure Cluster Management

- Docker 1.12 integrates swarm.
- Strong default swarm security.
- Strict privilege separation between managers and workers.



Orchestration Privilege Separation - What's the goal?

Privileged ->



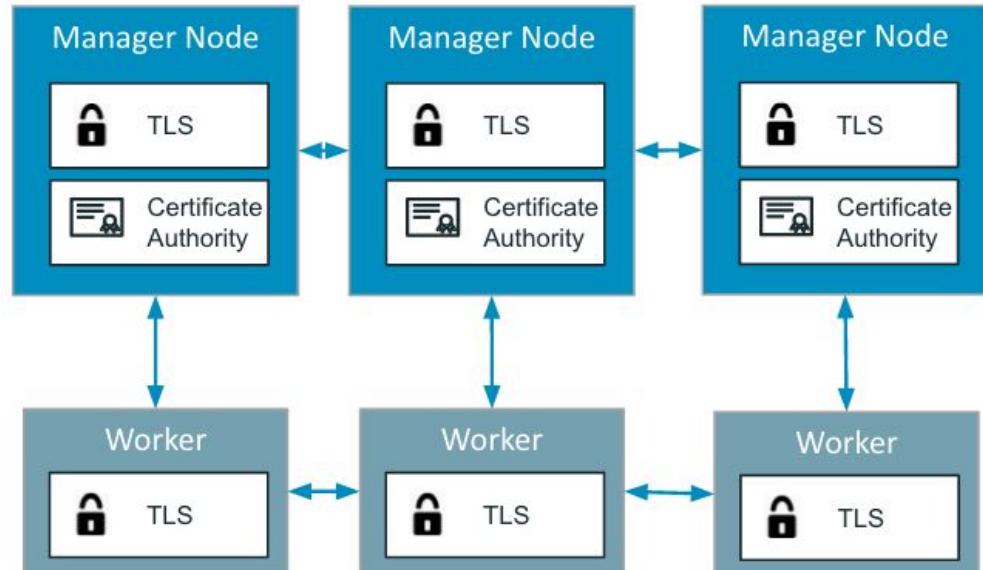
Unprivileged ->

- **Worker nodes must only have access to resources (e.g. networks, secrets, volumes) that are explicitly needed for their function**



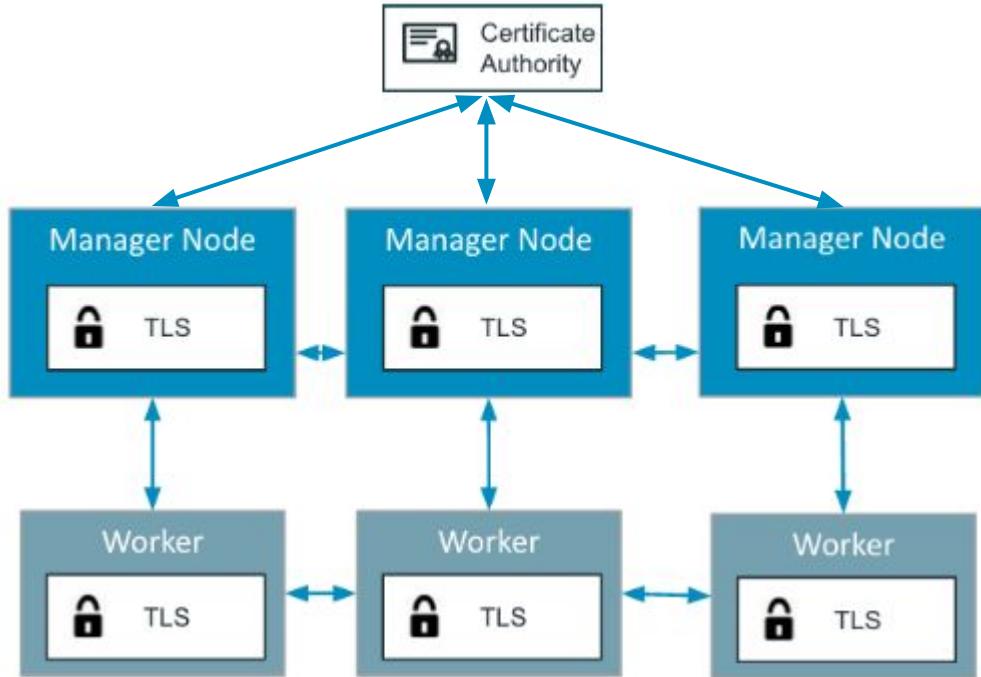
Mutual TLS by default

- Leader acts as CA.
- Any Manager can be promoted to leader.
- Workers and managers identified by their certificate.
- Communications secured with Mutual TLS.



Support for External CAs

- Managers support BYO CA.
- Forwards CSRs to external CA.



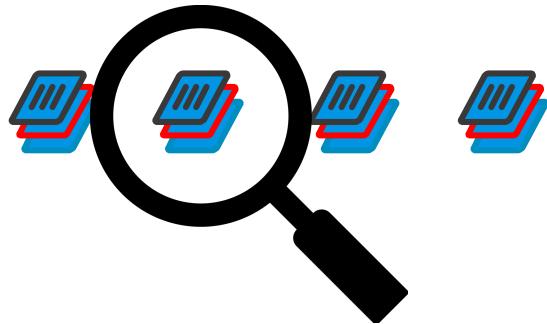
Automatic Certificate Rotation

- Customizable certificate rotation periods.
 - Minimum window of 30 minutes.
- Occurs automatically.
- Ensures potentially compromised or leaked certificates are rotated out of use.
- Whitelist of currently valid certificates.



Image Scanning for Windows and Linux Images

CONTINUOUSLY SCAN FOR VULNERABILITIES IN ALL IMAGES



KEY FEATURES

- Binary-level scans of both Windows and Linux images against a CVE database of known vulnerabilities
- Continuous monitoring after initial scan to alert when new vulnerabilities are found

BENEFITS

- Improve application security by scanning for known vulnerabilities prior to deploying and continuous checks thereafter
- Ensure compliance with alerts to new vulnerabilities



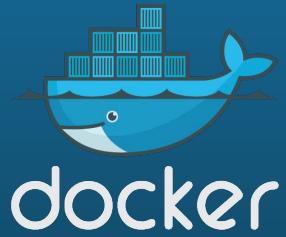
Secure Addition of Nodes

Only one way to approve addition of a node to the cluster:

Secure token



Docker Secrets Management





Safer Apps with Docker Secrets Management

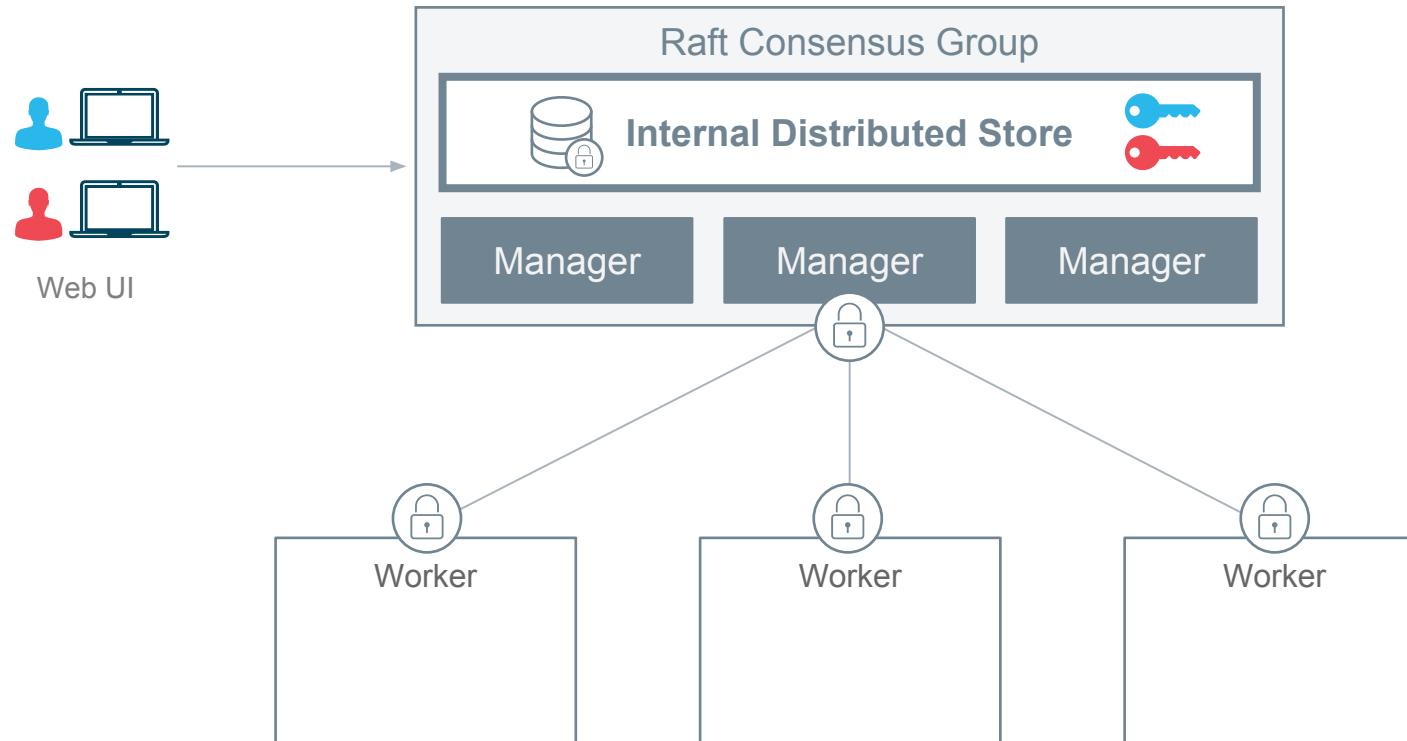
- Apps are safer when there is a standardized interface for accessing secrets
 - Works for legacy/microservices, dev/ops and Linux/Windows
- Apps are safer when secrets are not stored in the app itself



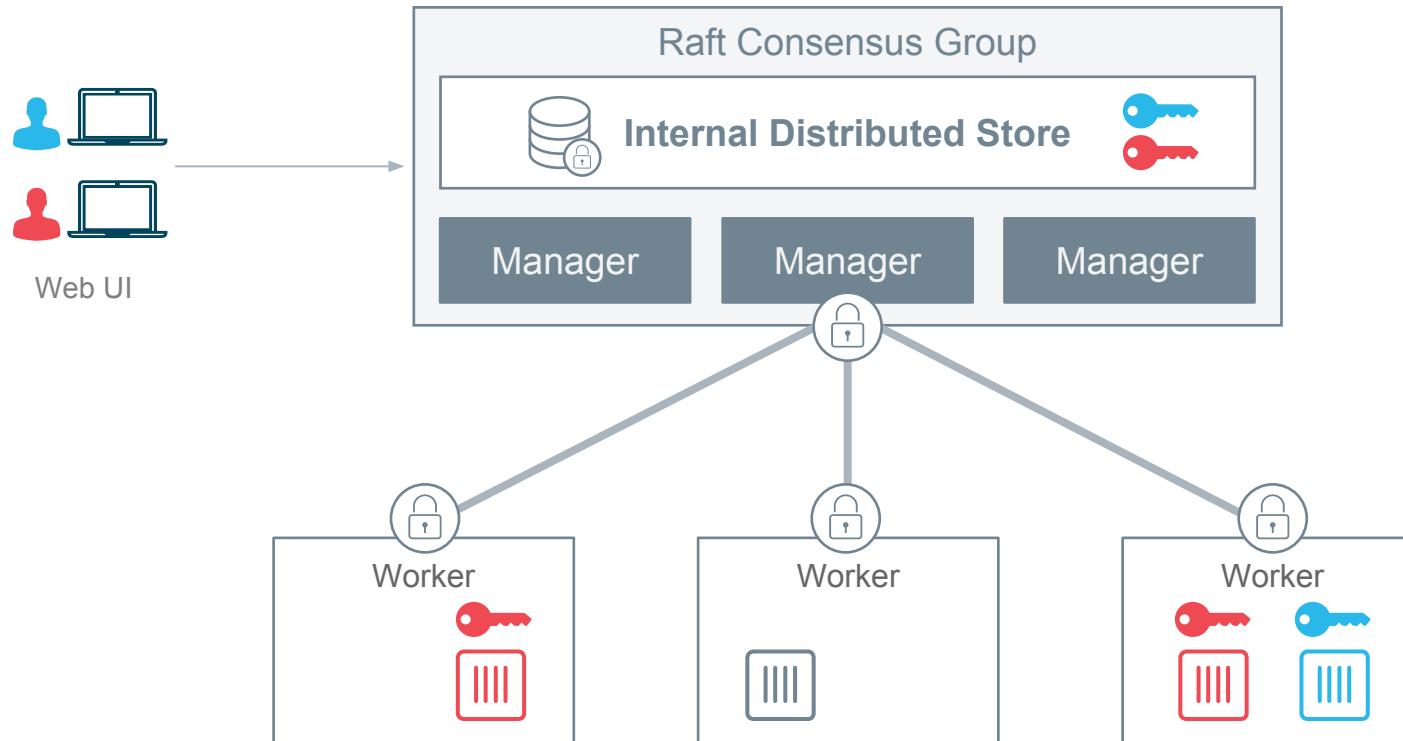
Trusted Delivery with Docker Secrets Management

- Encrypted at rest in the cluster store
- Encrypted while in motion on the network
- Delivered only to the specific authorized app
- Available to containers only in memory, never saved to disk

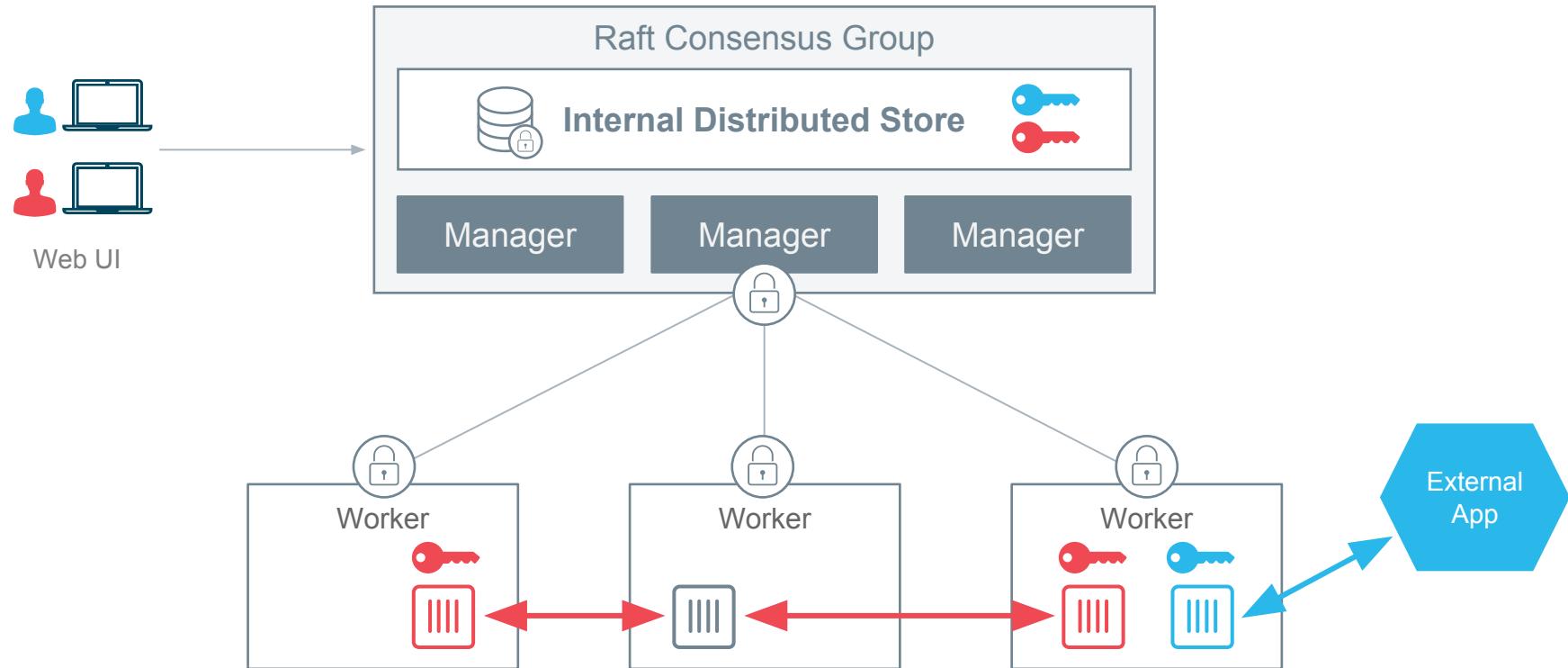
Docker Datacenter Secrets Architecture



Docker Datacenter Secrets Architecture



Docker Datacenter Secrets Architecture

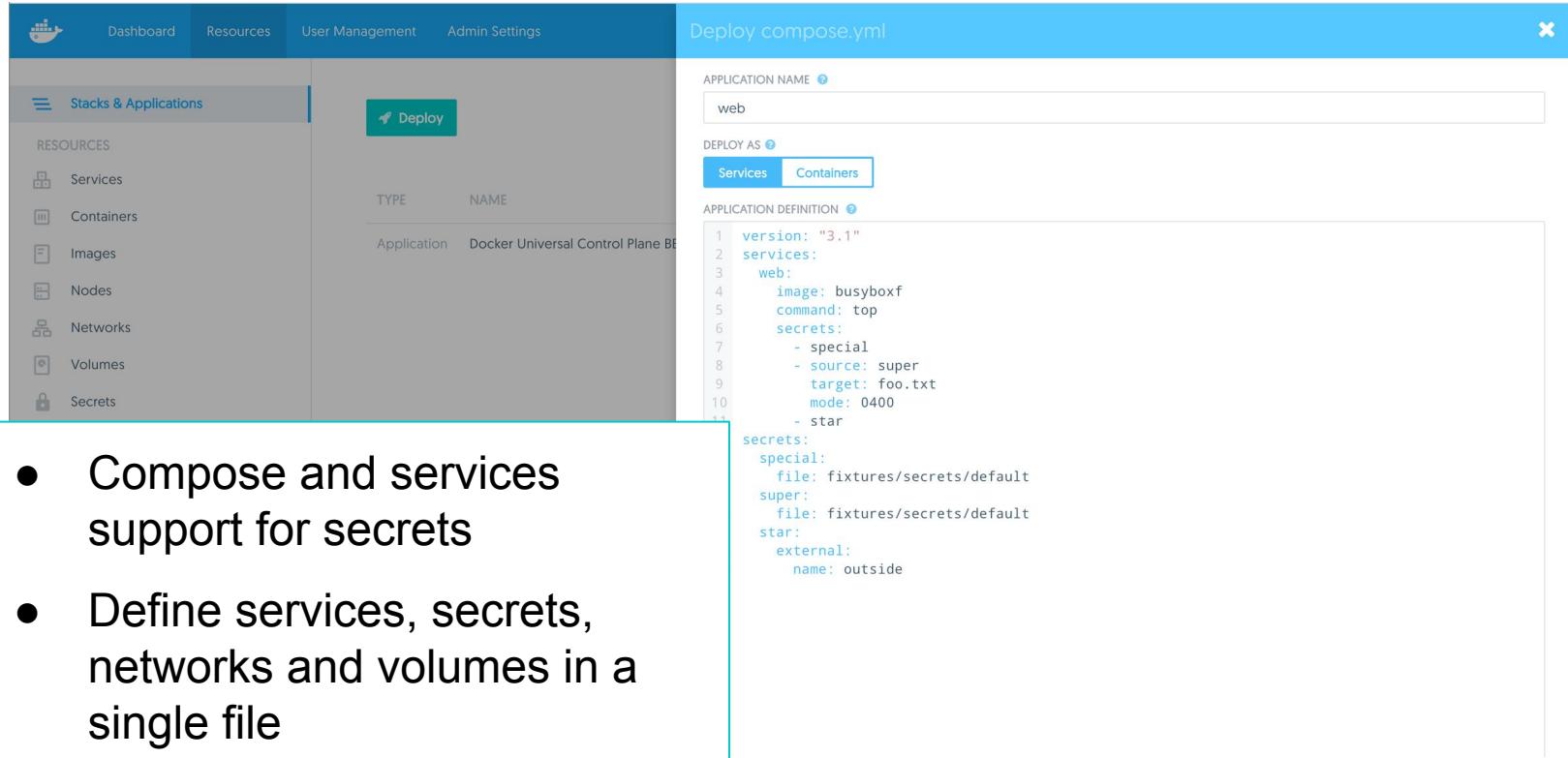




Usable Security with Secrets Management

- Standardized interface for developers and IT ops
- Developers have a simple interface
- IT ops gains secure policy management
- Same security guarantees travel with the app across different infrastructures

For Dev: Simple Developer Workflow

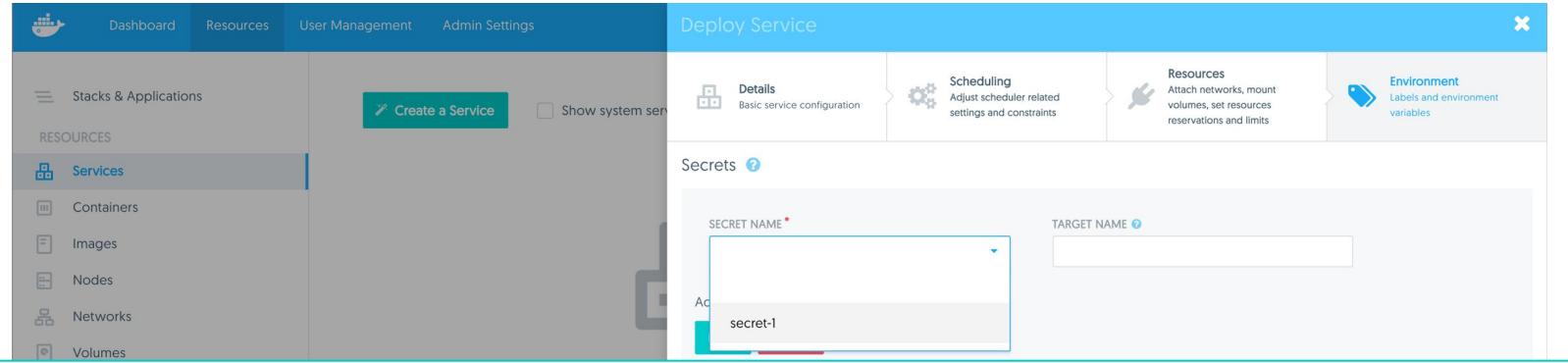


The screenshot shows the Docker Universal Control Plane interface. On the left, there's a sidebar with 'Stacks & Applications' selected, showing 'RESOURCES' like Services, Containers, Images, Nodes, Networks, Volumes, and Secrets. A 'Deploy' button is visible. The main area shows a table with columns 'TYPE' and 'NAME', containing one entry: 'Application' and 'Docker Universal Control Plane BE'. To the right, a modal window titled 'Deploy compose.yml' is open. It has fields for 'APPLICATION NAME' (set to 'web'), 'DEPLOY AS' (with 'Services' selected), and 'APPLICATION DEFINITION' which contains the following YAML code:

```
version: "3.1"
services:
  web:
    image: busybox
    command: top
    secrets:
      - special
      - source: super
        target: foo.txt
        mode: 0400
      - star
    secrets:
      special:
        file: fixtures/secrets/default
      super:
        file: fixtures/secrets/default
    star:
      external:
        name: outside
```

- Compose and services support for secrets
- Define services, secrets, networks and volumes in a single file

For Ops: Docker Datacenter Delivers Integrated Management and Controls

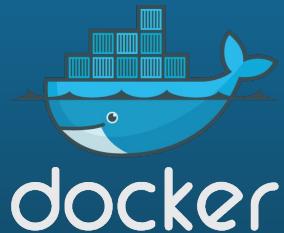


- Integrated secrets and app management in Docker Datacenter
- Deploy Compose file directly with no code changes
- Add granular access control to secrets and services

Docker Security

Secure Content

Securing the Image Pipeline



Common questions on content security

- What is inside my container?
- How do I know where this code came from?
- How do I keep our team safe from bad components?
- How do I stay on top of patches for compliance and governance?
- How do I NOT make this a giant pain for everyone? (including myself)



Securing the software supply chain with Docker

Before Docker

- Cumbersome tools with high failure rates of patches
- Reactive and slow process
- Software and dependency matrix with patches create more dependencies and clashes
- Security is a silo from dev and app ops

After Docker

- Faster, more successful software updates
- Simplify software compliance process
- Trusted delivery with self contained, secure and signed containers
- Unified workflow enabling both dev and ops



Secure Content: Image scanning and vulnerability detection

Repositories / Details

GENERAL TAGS TIMELINE

Tags (4)

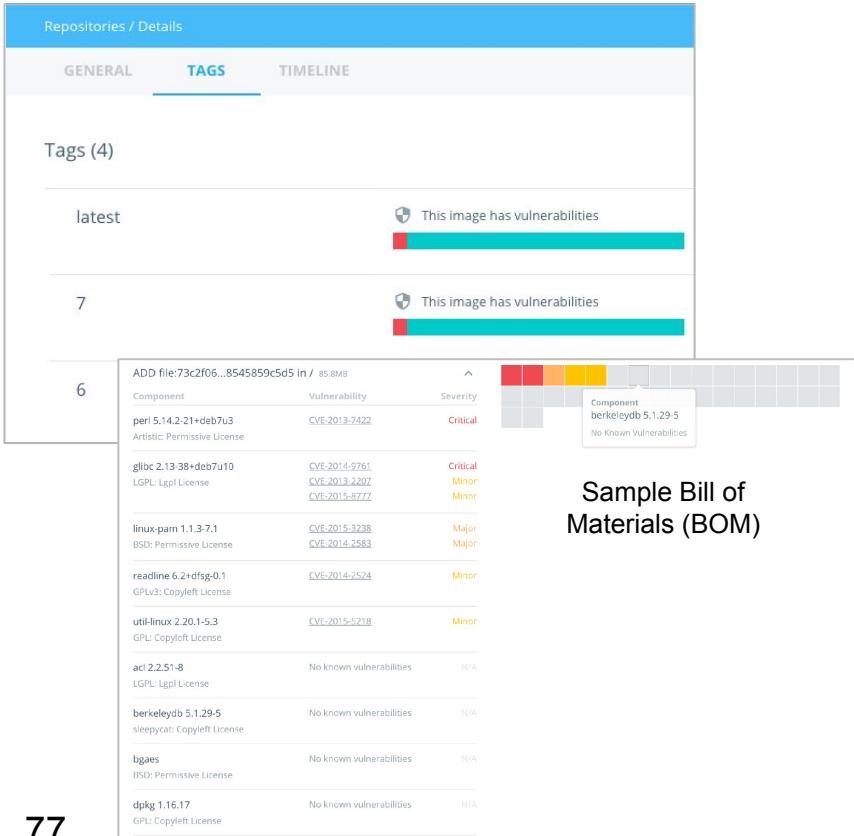
latest This image has vulnerabilities

7 This image has vulnerabilities

6 ADD file:73c2f06...8545859c5d5 in / 85.8MB

Component	Vulnerability	Severity
perl 5.14.2-21+deb7u3 Artistic: Permissive License	CVE-2013-7422	Critical
glibc 2.13-38+deb7u10 GPLv3: Copyleft License	CVE-2014-9761, CVE-2014-2207, CVE-2015-8777	Critical Minor Minor
linux-pam 1.1.3-7.1 BSD: Permissive License	CVE-2015-3238 CVE-2014-2583	Major Major
readline 6.2+dfsg-0.1 GPLv3: Copyleft License	CVE-2014-2524	Minor
util-linux 2.20.1-5.3 GPL: Copyleft License	CVE-2015-5218	Minor
acl 2.2.51-8 LGPL: Lgpl License	No known vulnerabilities	N/A
berkeleydb 5.1.29-5 Sleepycat: Copyleft License	No known vulnerabilities	N/A
bgaes BSD: Permissive License	No known vulnerabilities	N/A
dpkg 1.16.17 GPL: Copyleft License	No known vulnerabilities	N/A

Sample Bill of Materials (BOM)



Deep visibility with binary level scanning

- Detailed BOM of included components and vulnerability profile
- Checks packages against CVE database AND the code inside to protect against tampering
- Covers wide range of languages, binaries, OS

Proactive risk management

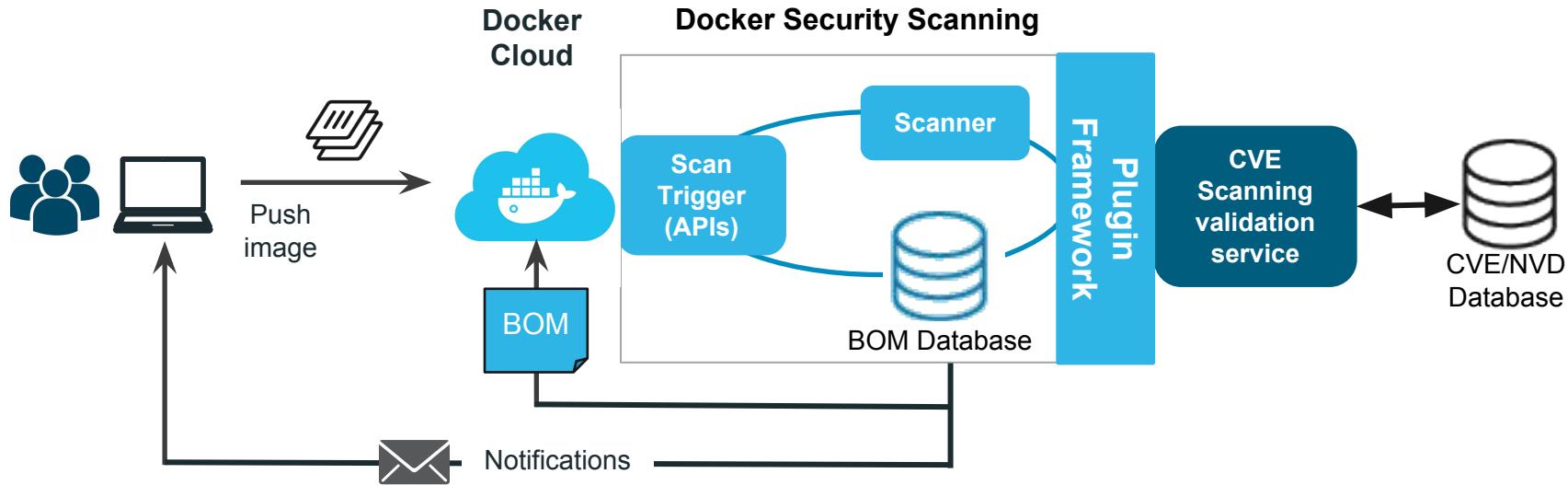
- Continuous monitoring of CVE/NVD databases with notifications pointing to repos and tags that contain new vulnerabilities

Secure the software supply chain

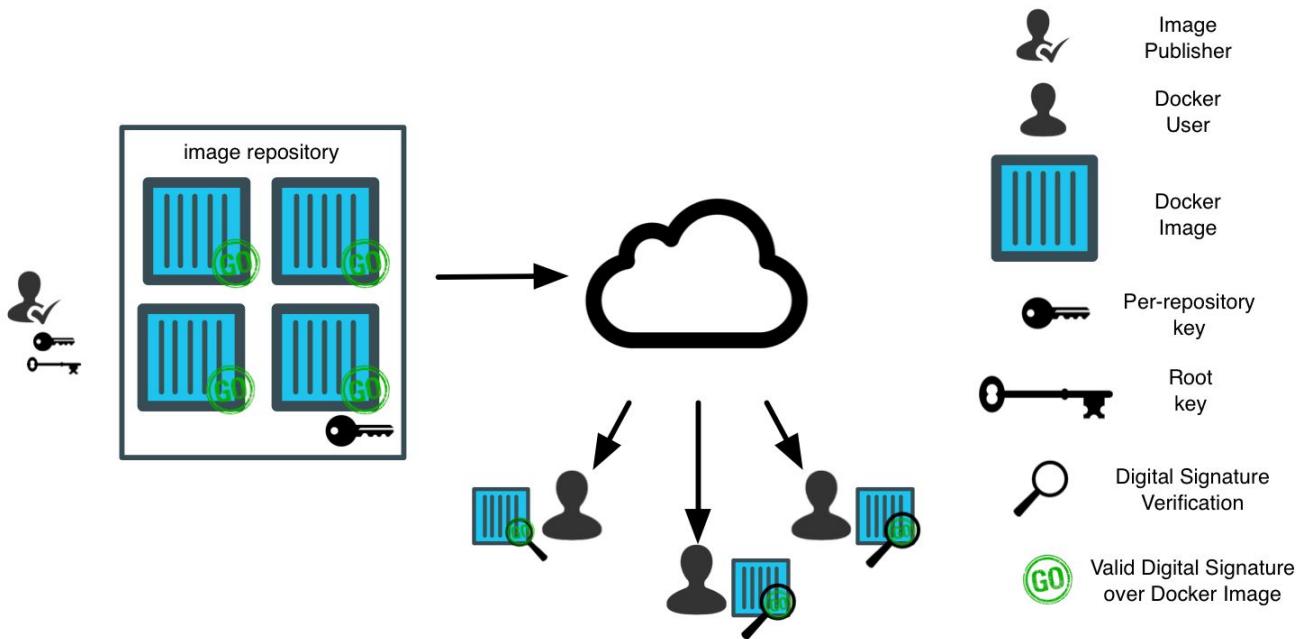
- Integrated workflow with Docker Content Trust
- Available for Official Repos since Nov 2015



Secure Content: Image scanning and vulnerability detection



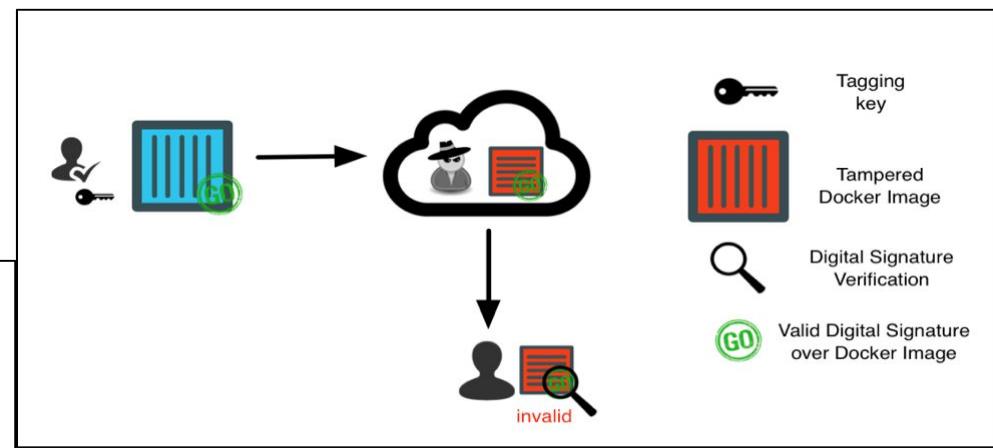
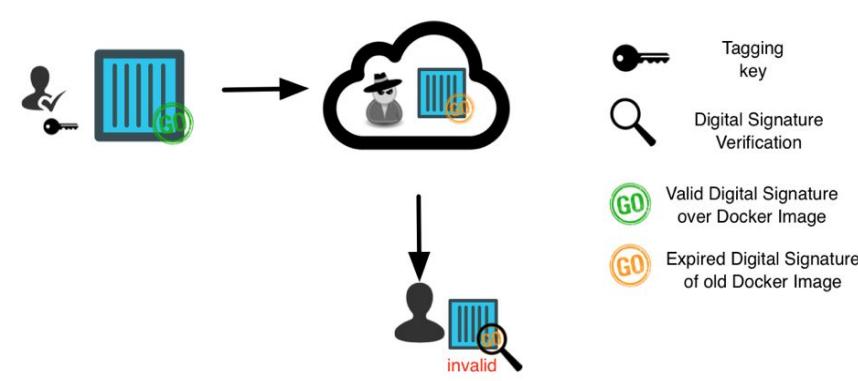
Content: Docker Content Trust



- Publisher digitally signs the Docker images before sending them to the cloud
- Any user can independently verify the digital signatures on the images

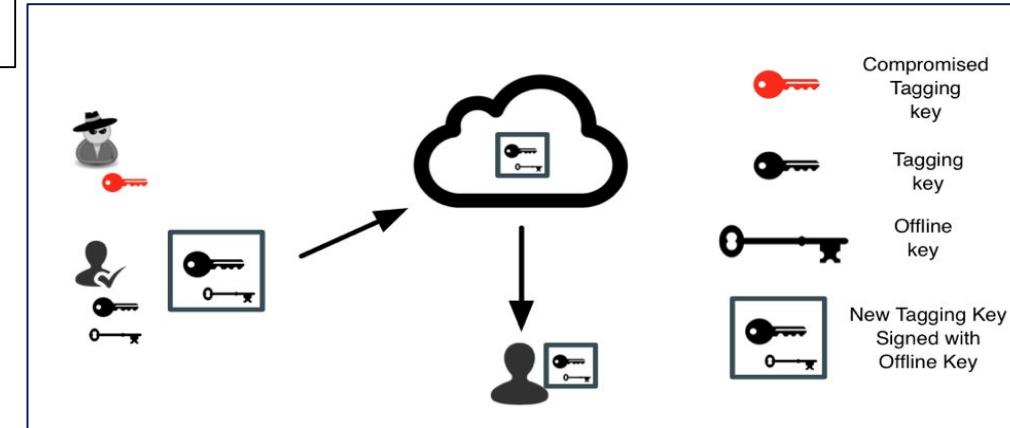


Use Case: Image Forgery —>



<— Use Case: Replay attacks

Use Case: Compromised keys —>



CI



Image passes
CI test

Security Scanning



Scan
image

Staging



Successful
deployment to staging

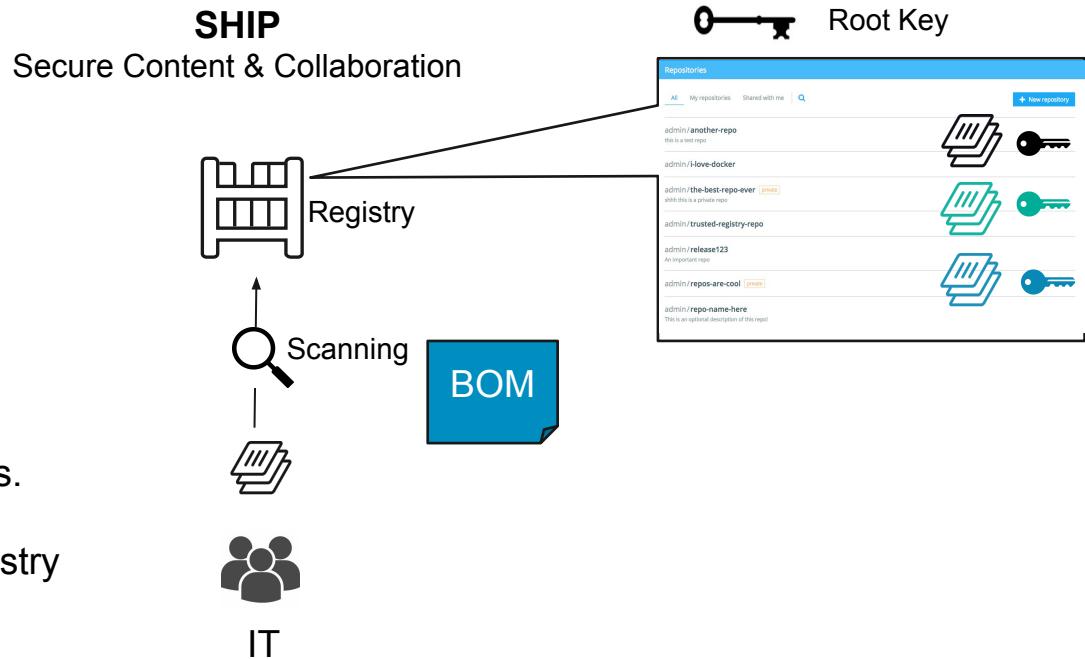
Production



Verify all three
signatures before
deploying to production



1. Start with a secure base

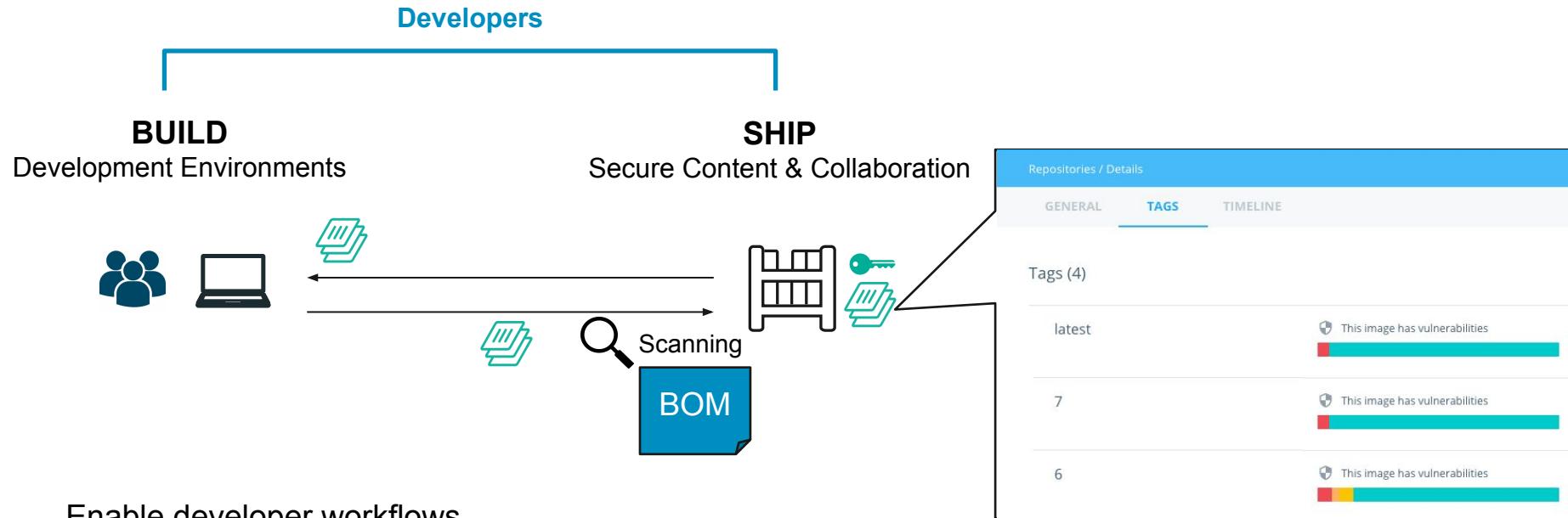


Set up a central repository

- IT creates and scans base images.
- Images are digitally signed
- Images are pushed to central registry



2. Build secure apps



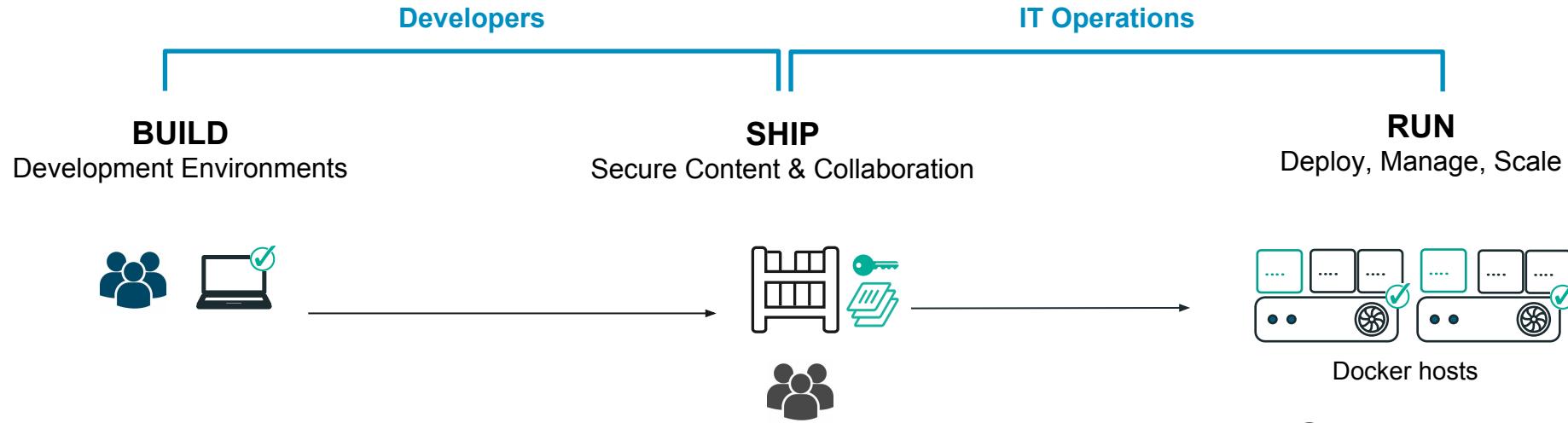
Enable developer workflows

- Pull from library of secure images
- Local Docker host establishes trust with repo and registry
- Build apps, add image layers and CI test
- Scan updated image and generate new BOM
- Remediate issues or deploy

View history of BOMs for all scanned tags per repo



3. Deploy apps



Deploy your apps

- Deploy apps to next environment
- Secure host configurations with Docker Bench
- Hosts establish trust with repo

Proactively manage new vulnerabilities

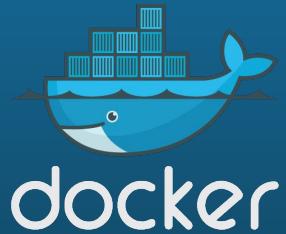
- New VULN added to database regarding a package
- Docker checks all BOMs that have this package
- Notification is sent re: affected repos and tags



Docker Security

Secure Access

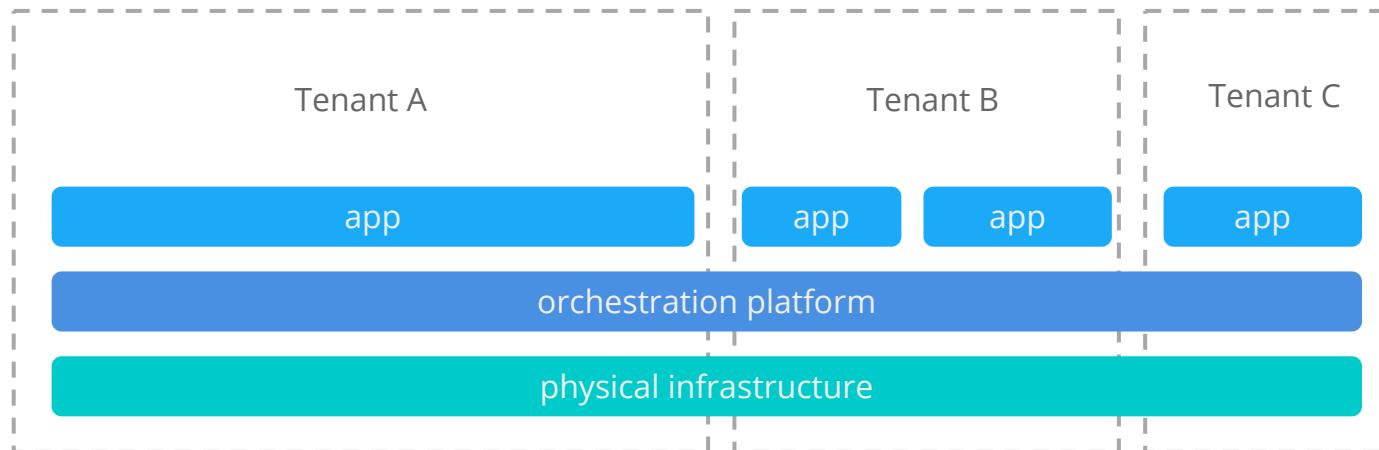
Docker Enterprise Access Control



Resource Sharing

Why we don't give every team their own cluster

- Higher resource utilization
- Specialization of skillsets for discrete layers of IT stack
- Separation of responsibilities for more resilient and manageable architectures



users



**virtual
resources**

secrets

images

networks

volumes

services

**physical
resources**

host

host

host

CPU

memory

I/O

storage



docker

Highly Customizable Roles

CUSTOMIZE ROLES AND IMPROVE ACCESS GRANULARITY AND CONTROL

Create Grant			
admin	Subject	Role	Collection
User Management	Team	Restricted Control	9be4ce3f-edf6-4bfa-82e7-c7cdcaa841d4
Organizations	Team	Restricted Control	d9d9ae73-b1d3-49bc-956a-5c76c4db33dd
Users	Team	Scheduler	d9d9ae73-b1d3-49bc-956a-5c76c4db33dd
Roles	Team	Scheduler	6d55d3ba-33c2-4daf-8874-e4e7b92aaa99
Manage Grants	Team	Scheduler	swarm
Dashboard	sheila	Restricted Control	e6d42fba-5762-4f92-8049-6e221efbc7e0
Collections	vivek	Restricted Control	11870bf8-c6f2-4928-a783-aeafb28ef6e7
Stacks	vivek	Volume Maker	202fe844-ec64-4f5c-9695-98f8809580e7
Services	bob	test	6d55d3ba-33c2-4daf-8874-e4e7b92aaa99
Containers	admin	Restricted Control	d3c6e113-8578-437c-adbd-c0dfaee143ce

KEY FEATURES

- Create custom roles with granular action permissions or leverage pre-defined default roles
- Define resource collections to more easily visualize and assign users to specific cluster resources
- Define Organizations of one or more Teams

BENEFITS

- Easily manage complex organizations by defining permissions across user groups and resource collections
- Improve security by setting permissions that align to your organization's requirements and practices
- Meet compliance and regulatory requirements through tight access control and separation of roles and responsibilities



collections

/production

/payments

UCP
Workers

containers

volumes

configs

networks

secrets

/mobile

UCP
Workers

containers

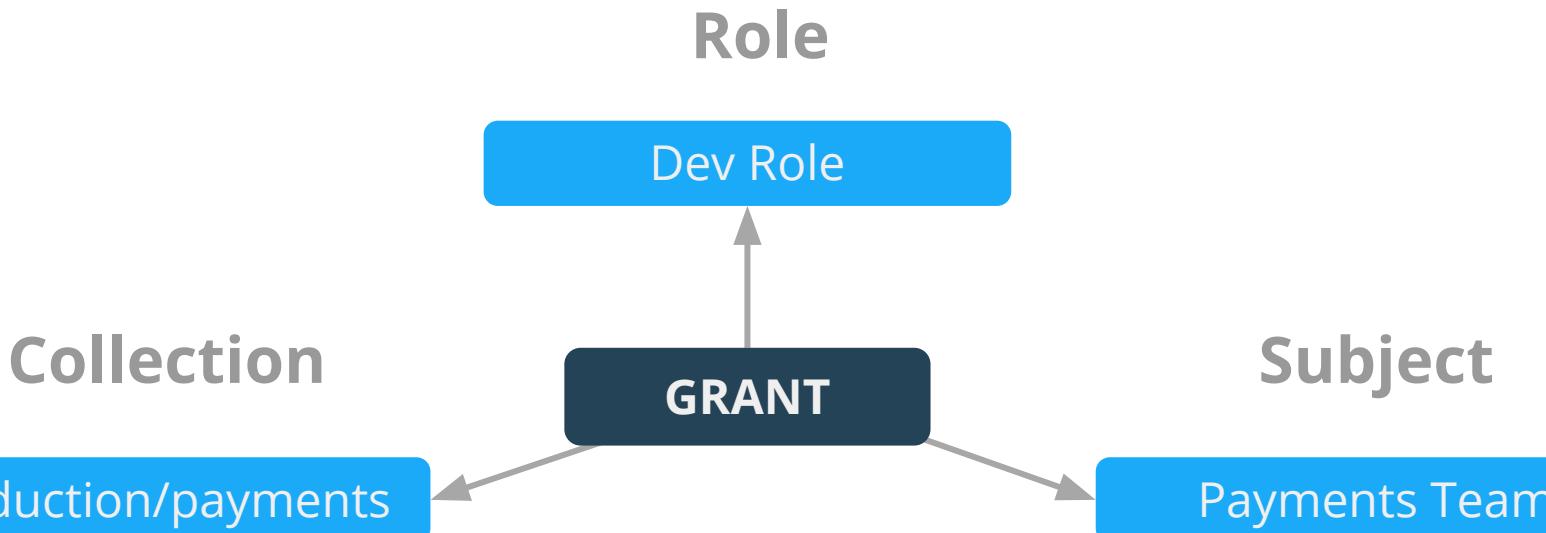
volumes

configs

networks

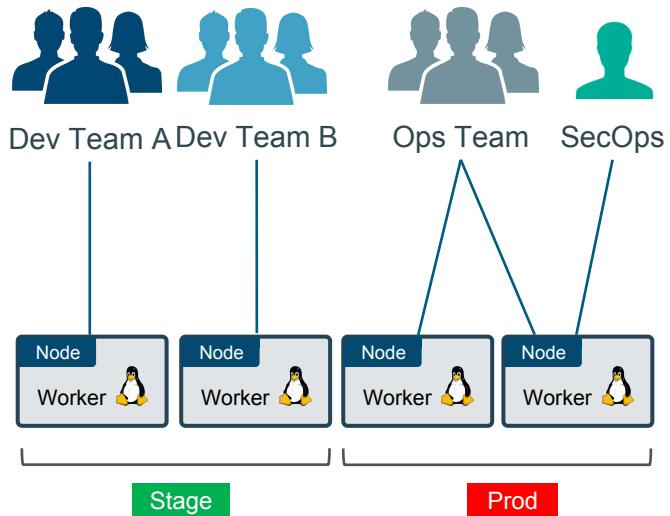
secrets

grants



Node Access Control

SUPPORT SECURE MULTI-TENANCY ACROSS MULTIPLE TEAMS THROUGH NODE-BASED ISOLATION AND SEGREGATION



KEY FEATURES

- Enforce node affinity and node anti-affinity rules to allow certain users/teams/orgs to deploy within a subset of nodes or outside certain nodes (eg. Production nodes)
- Set up different security zones within the same cluster to isolate and segregate access of protected information

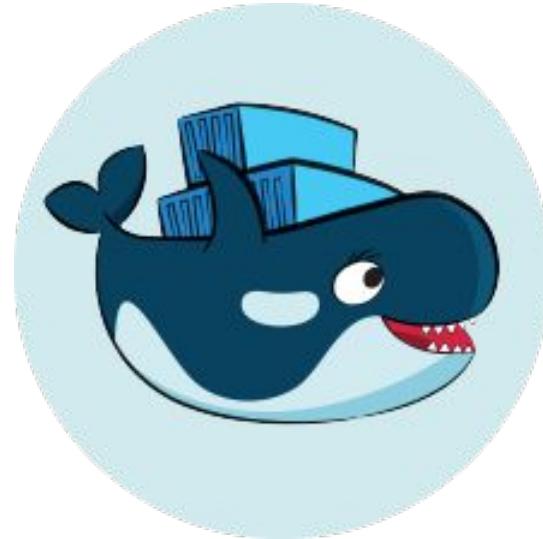
BENEFITS

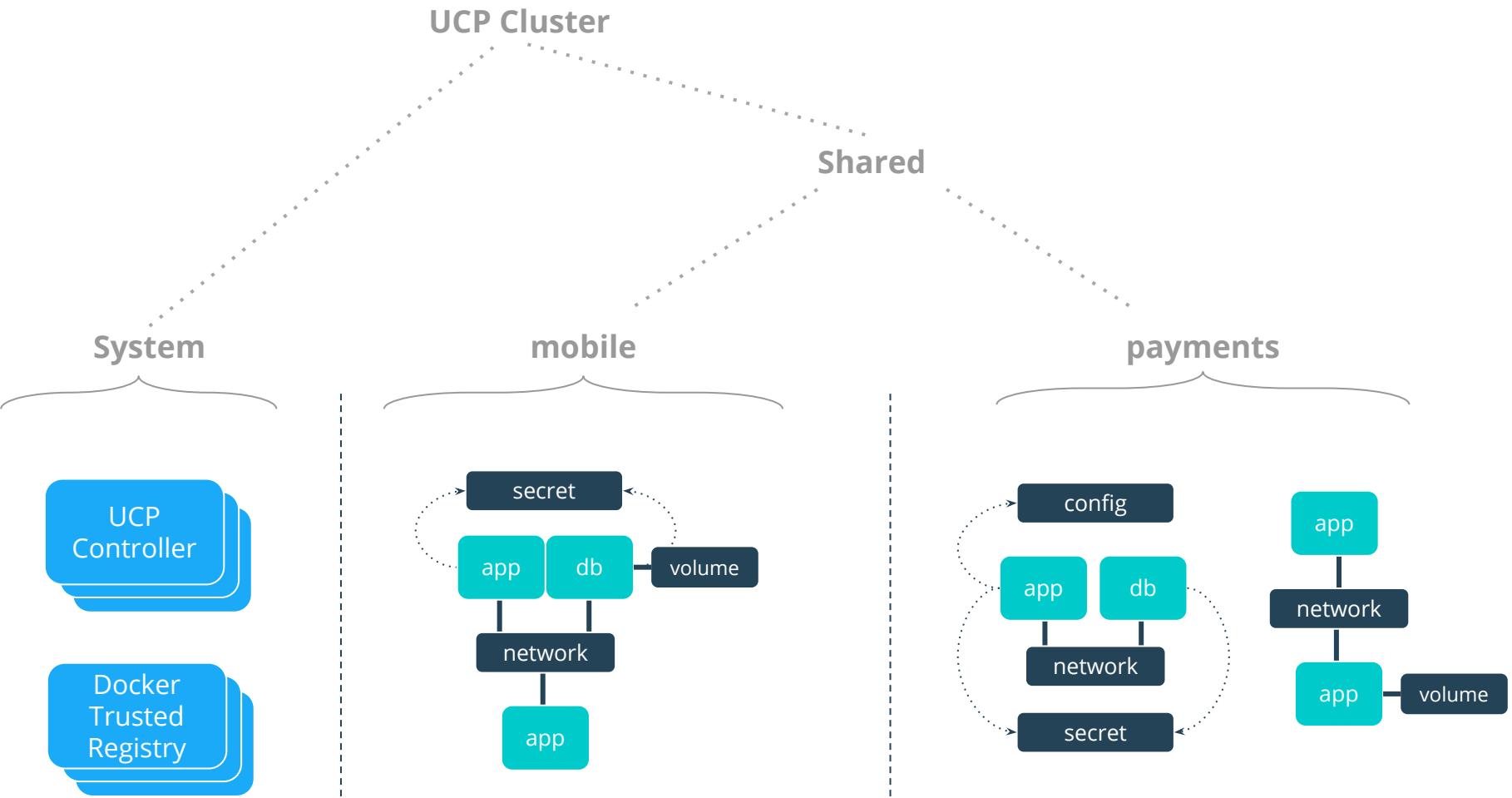
- Support multiple teams within the same cluster while providing physical separation and isolation
- Prevent “noisy neighbors” by limiting a team’s resources to approved nodes
- Meet compliance and regulatory requirements by isolating sensitive workloads to certain nodes and limiting access to those nodes



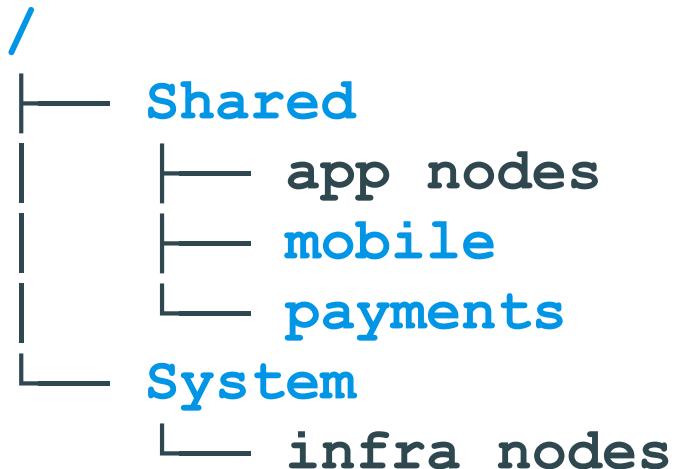
Access Control at OrcaBank

- **Admins** - Full capabilities, full stack
- **Ops team** Full capabilities against application nodes
- **Payments team** can view their containers only
- **Mobile team** can view their containers only





OrcaBank collection architecture



grant composition

Roles

Full Control

Ops

Dev

Collections

Grants

Teams

Users



/System

/Shared

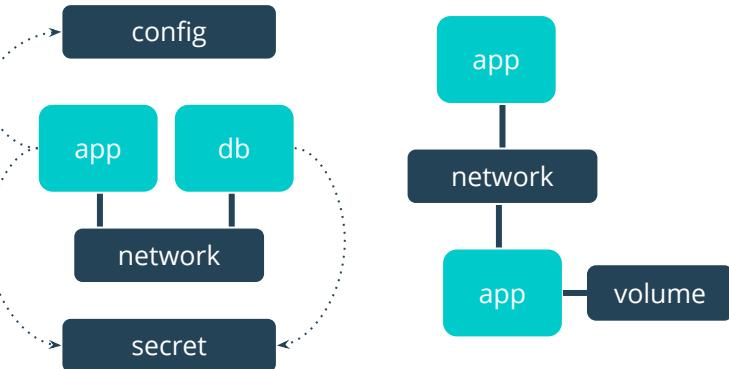
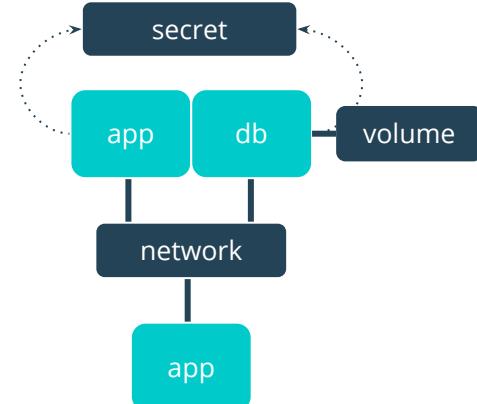
UCP
Controller

Docker
Trusted
Registry

/mobile

Worker
Nodes

/payments



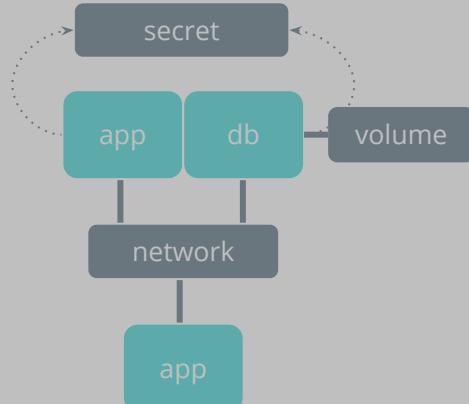
/System

UCP
Controller

No
Access
Docker
Trusted
Registry

Mobile Team

- View containers
- View services
- View logs
- Container Inspect

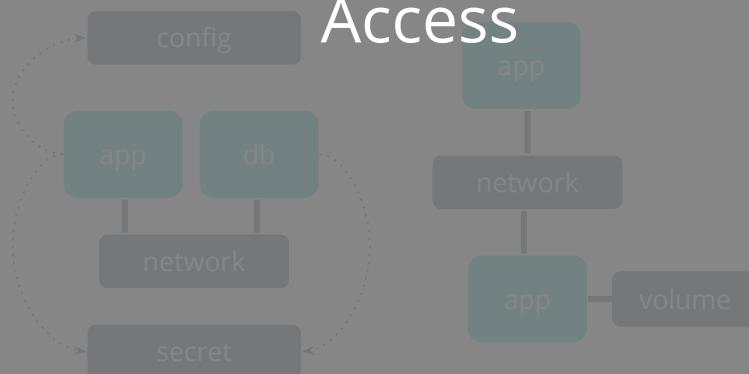


/Shared

UCP
Workers

/payments

No
Access
config



/System

UCP
Controller

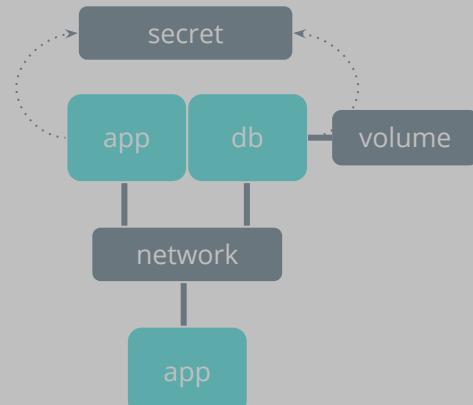
No
Access
Docker
Trusted
Registry

/Shared

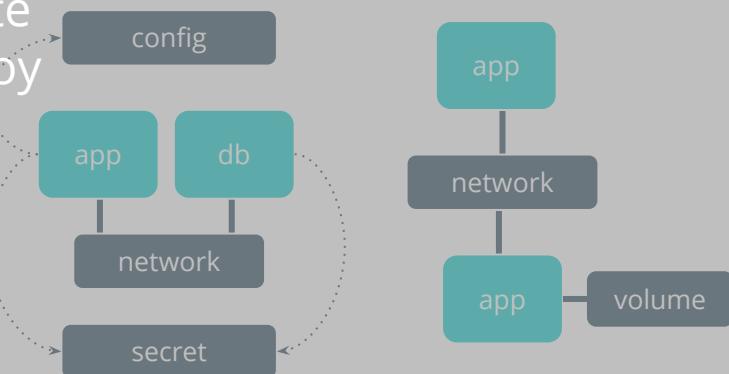
UCP
Workers
Ops Team

/mobile

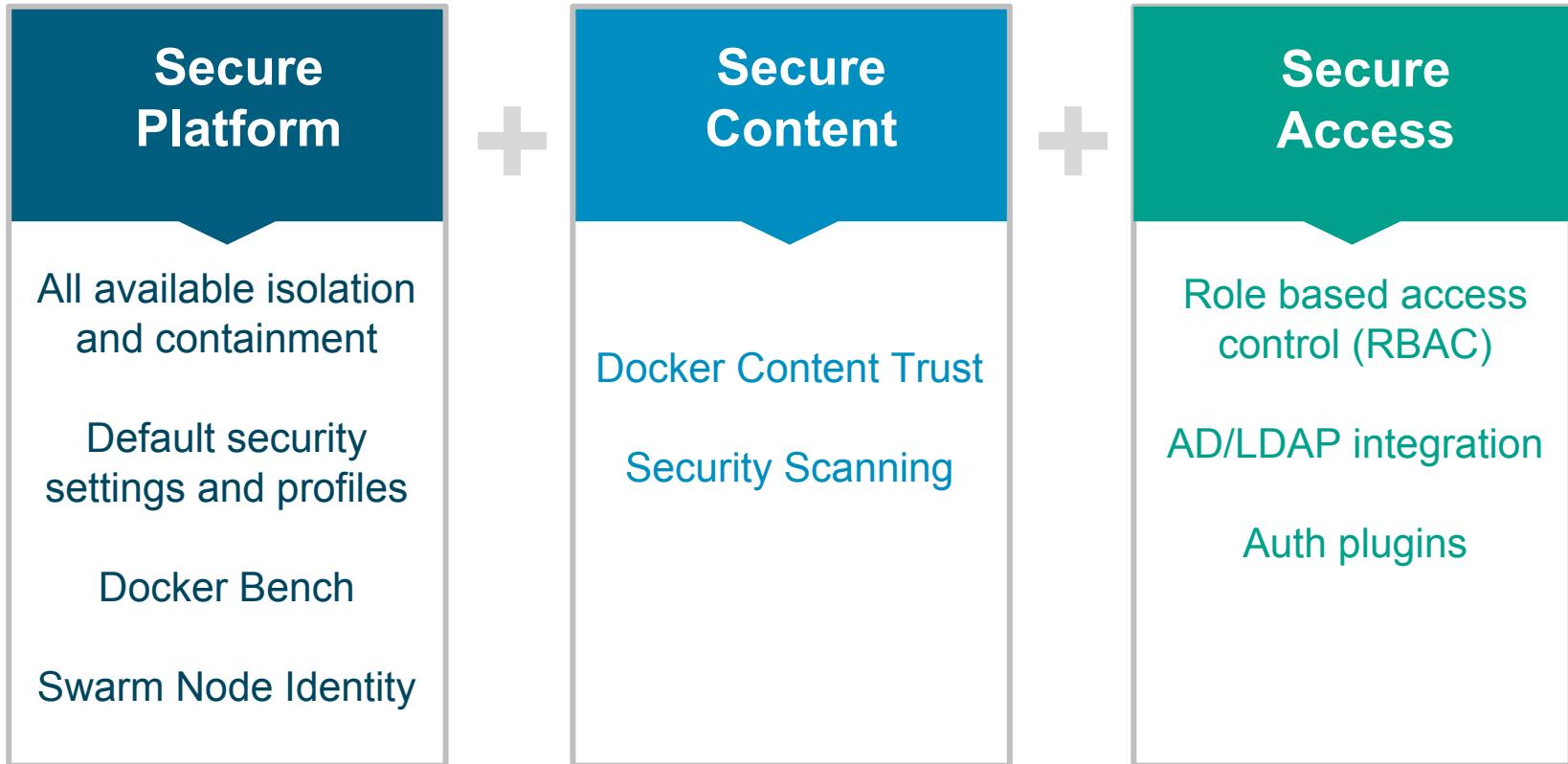
- All Resources
 - Deploy
 - View
 - Update
 - Destroy



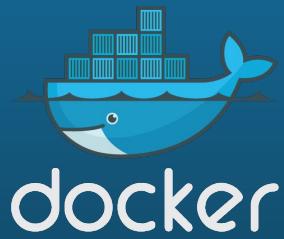
/payments



Docker secures your software supply chain



Logging



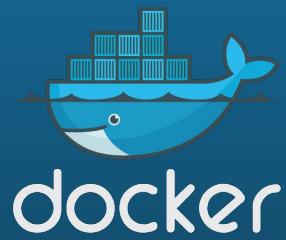
Docker Logging

- Two Types of Logs
- Container vs Engine Logs
- Engine
 - System Logging (e.g journald)
 - Most modern OSs use Journald
- Container
 - Configured at Daemon Level (/etc/docker/daemon.json) applies to ALL containers
 - Configured Container/Service Level (--log-opt) at container runtime.
Overrides ^
- Supported Drivers: json, splunk, syslog, gelf, journald, awslogs, fluentd



OR

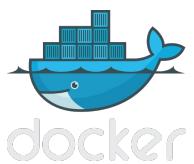
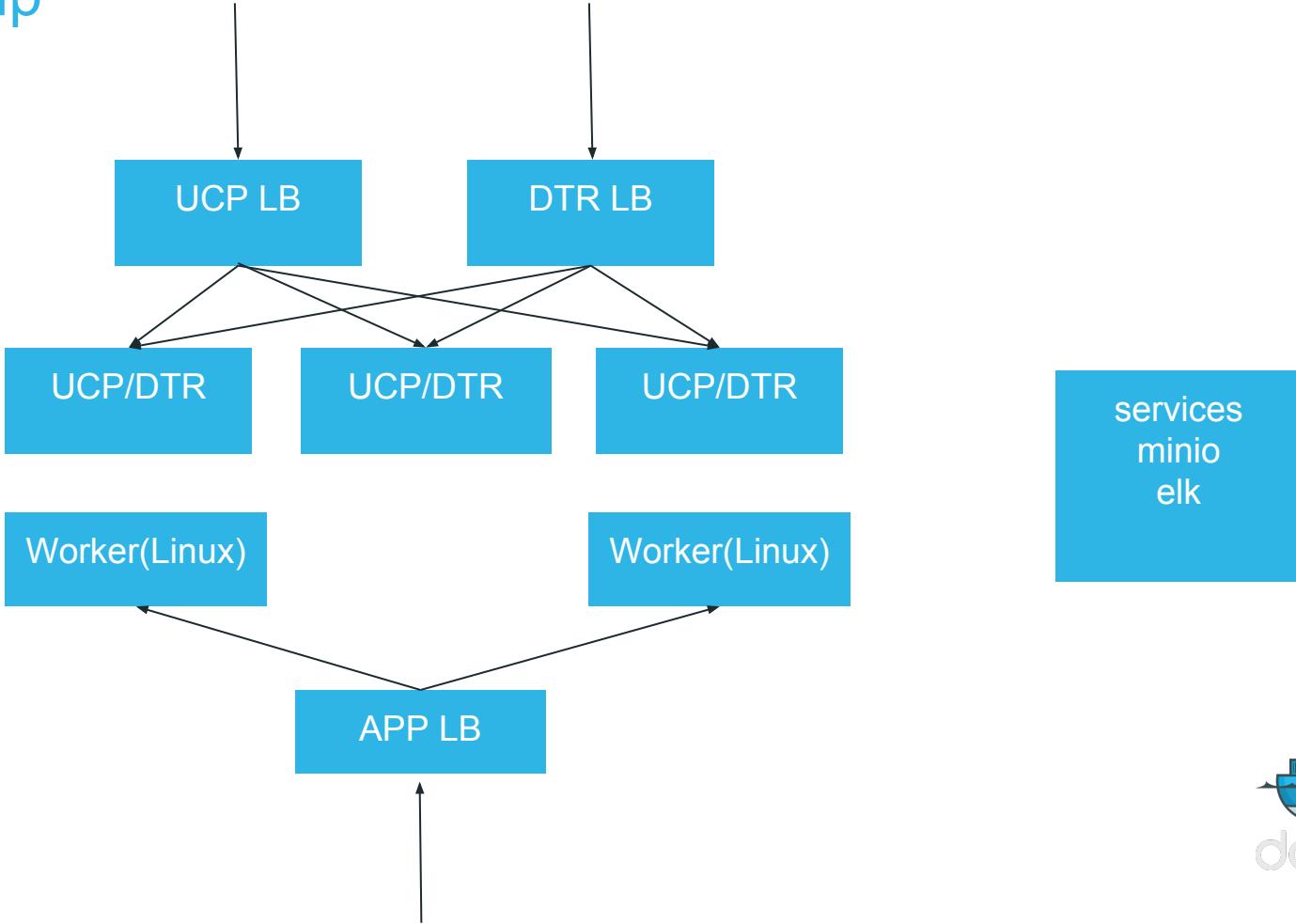
Swarm Lab



Lab Setup

- 1 Setup per Team to include:
 - 3x lb (UCP, DTR, APP)
 - 3x Manager Nodes (UCP and DTR)
 - 1x Infrastructure Services
 - Minio
 - ELK Stack
 - 2x UCP worker nodes
 - CNAMEs DNS entries for UCP, DTR, and application

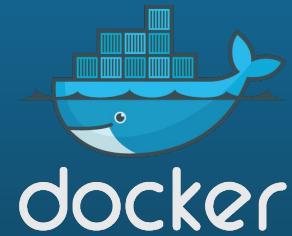
Lab Setup



Swarm Lab Summary

- ❑ Installed Docker EE on the two worker Nodes
- ❑ Created a Swarm (3 managers, 2 worker nodes)
- ❑ Launched a logging service to stream all logs to ELK

Docker Enterprise Edition Standard Universal Control Plane

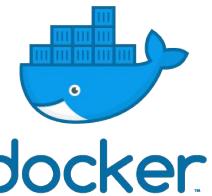


UCP Overview

- What's new in 17.06
- Architecture
- Deep Dives
 - Client Bundle
 - Secure Deployment
 - HRM
 - Backup and Recovery
 - Healthchecks
- UCP In Practice: Installation and Ongoing Operations

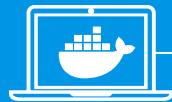
Docker Enterprise Edition 17.06

What's New Technical Overview
August 2017



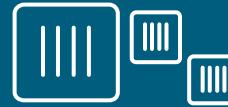
Docker is the only Containers-as-a-Service platform for IT that manages and secures diverse applications across disparate infrastructure, both on-premises and in the cloud

ENGINE FOR INNOVATION



Secure Software Supply Chain

COST SAVINGS



Multi-Architecture Operations

DOCKER ENTERPRISE EDITION



Linux



Windows



Mainframe



AWS



Azure



Other Public Clouds



Infrastructure Independence



Highlights of Docker EE 17.06

Multi-Architecture Operations

- Mixed Windows and Linux cluster support
- Linux on IBM z Systems support
- RBAC for nodes

Secure Software Supply Chain

- Automated image promotions
- Immutable repositories
- Multi-stage builds
- Cluster configuration and security APIs

Infrastructure Independence

- Resource collections
- Custom roles
- Enhanced UI
- Docker service logs



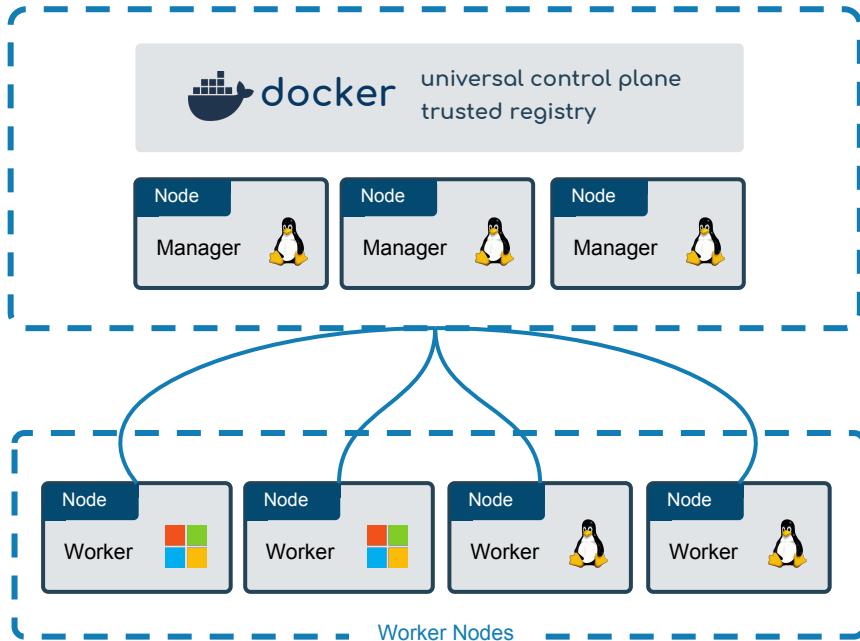
New Feature Details

What's New in Docker EE 17.06



Mixed Windows and Linux Clusters

UNIFORMLY OPERATE, MANAGE, AND SECURE WINDOWS AND LINUX CONTAINERS



KEY FEATURES

- Extend enterprise security features like image signing, image scanning, and secrets management to both Windows and Linux worker nodes
- Leverage the same LDAP/AD integration and RBAC rules across Windows and Linux nodes
- Visualize all apps in the same environment

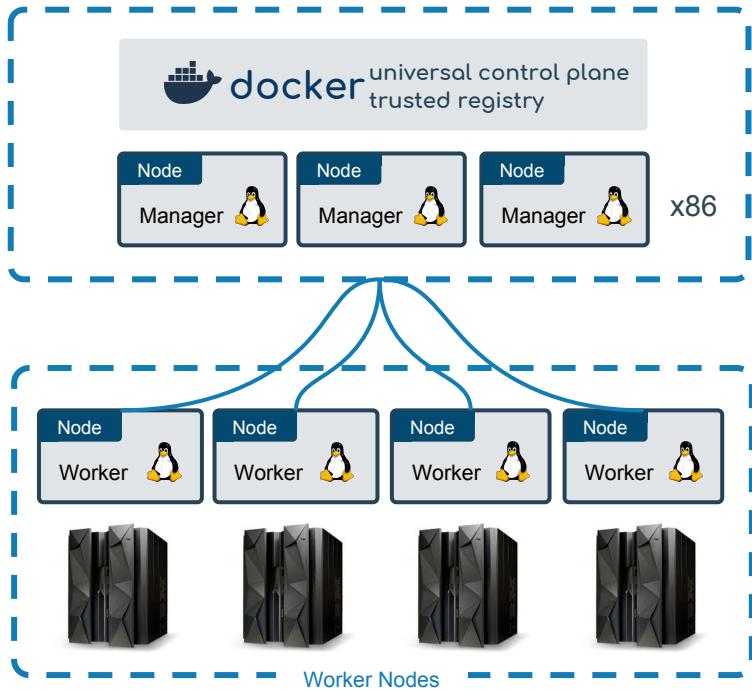
BENEFITS

- Improve resource utilization and incur less management overhead with centralized management across Windows and Linux apps
- Reduce risk with enforced processes and policies that are consistent across Windows & Linux apps



Run Docker on IBM Z

GET THE BENEFITS OF CONTAINERS FOR APPS ON MAINFRAME



KEY FEATURES

- Support Docker on SLES, Ubuntu, and RHEL hosted on IBM Z mainframes
- Extend enterprise security features like image signing, image scanning, and secrets management to mainframe Linux apps
- Leverage the same LDAP/AD integration and RBAC rules across all nodes
- Visualize all apps in the same environment

BENEFITS

- Incur less management overhead with centralized management across x86 and IBM Z apps
- Improve maintenance and patching and reduce downtime with containerized mainframe applications
- Improve resource utilization on IBM Z while standardizing the software supply chain



Preparing Windows Nodes

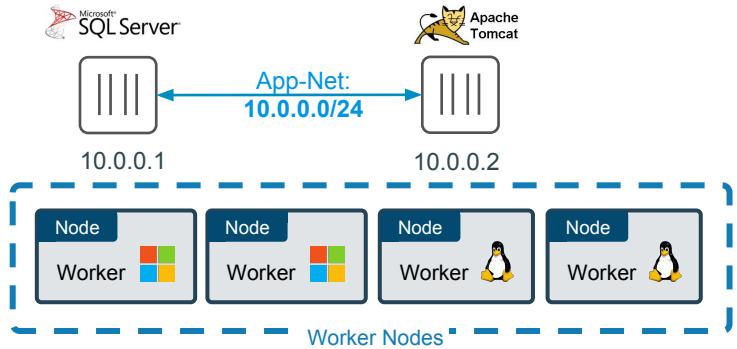
Windows nodes need to complete prerequisite steps before joining a cluster:

- Ensure latest patches have been applied to Windows Server 2016
- Pre-pull UCP Windows agent image
- The actual `swarm join` command is the same for Linux and Windows



Hybrid Applications

CONNECT WINDOWS AND LINUX CONTAINERS



```
version: "3.2"
services:
  database:
    image: sixeyed/atsea-db:mssql
    networks:
      - atsea
    deploy:
      endpoint_mode: dnsrr
    placement:
      constraints:
        - 'node.labels.os == windows'
  appserver:
    image: sixeyed/atsea-app:mssql
    networks:
      - atsea
    deploy:
      endpoint_mode: dnsrr
    placement:
      constraints:
        - 'node.labels.os == linux'
```

- Leverage best-in-class technologies across Windows and Linux
- Connect Windows and Linux containers in the same cluster through a common overlay network
- Build Compose files for hybrid applications
- Leverage labels and constraints for intelligent placement and scheduling



Closer Look: Hybrid OS App Docker Compose File

```
services:  
  database:  
    image: sixeyed/atsea-db:mssql  
    ports:  
      - mode: host  
        target: 1433  
  networks:  
    - atsea  
  deploy:  
    endpoint_mode: dnsrr  
    placement:  
      constraints:  
        - 'node.platform.os == windows'  
  appserver:  
    image: sixeyed/atsea-app:mssql  
    ports:  
      - target: 8080  
        published: 8080  
    networks:  
      - atsea  
    deploy:  
      placement:  
        constraints:  
          - 'node.platform.os == linux'  
networks:  
  atsea:
```

Windows currently only supports host-mode for publishing ports

Windows nodes should be set to use DNS round robin service discovery

Windows and Linux nodes can share a common overlay network

`node.platform.os` is a built in label that can be used for workload placement



Windows Limitations and Alternate Approaches

- Networking
 - No ingress routing mesh
 - Use host mode and the HTTP routing mesh
 - No encrypted networks
 - Encrypt communications at the app level using HTTPS, TLS, etc.
- Secrets
 - Temporary secret files are stored on disk.
 - Use BitLocker to encrypt the secret data at rest.
 - Specify UID, GID, and mode are not supported for secrets on Windows services.
 - However secrets are currently only accessible by administrators and users with system access within the container.
- Mounts
 - On Windows Docker can't listen on a Unix socket.
 - Use TCP or a named pipe instead.



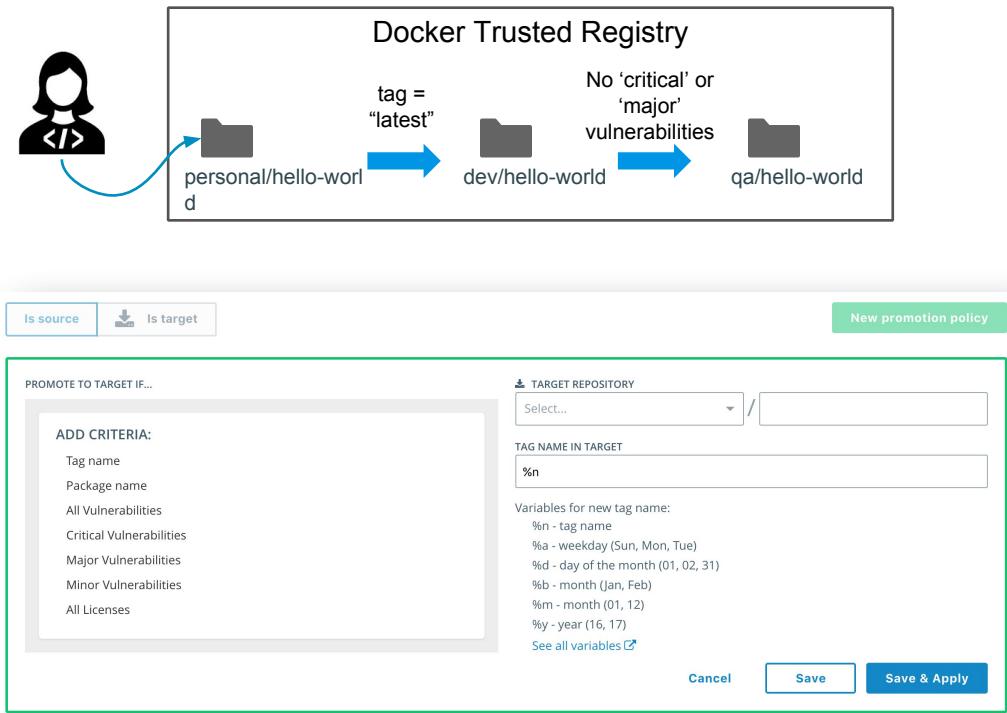
Scanning Windows Images: Additional Steps

- By default image layers based on official Microsoft images are not pushed to DTR
 - Only the manifest and layers not based on MSFT images are sent
 - DTR cannot scan these foreign layers because they are not present on the system
 - Microsoft Image layers are actually pulled from a Microsoft registry
- Need to modify C:\ProgramData\docker\config\daemon.json
 - Add: "allow-nondistributable-artifacts": ["<dtr-domain>:<dtr-port>"]



Automatic Image Promotion

SCALE WITH AUTOMATED POLICIES FOR IMAGE MANAGEMENT



KEY FEATURES

- Define policies to automatically promote images from one repository to another repository within Docker Trusted Registry
 - Each repository can have multiple policies
 - Criteria can include tags, package names, vulnerabilities, or license review
 - Images can be automatically re-tagged when promoted
- Promote images manually as well

BENEFITS

- Automate image management for improved agility, removing manual bottlenecks
- Ensure images are not promoted unless they meet certain criteria for improved consistency



Immutable Repositories

PREVENT INADVERTENT CHANGES TO TAGS

The screenshot shows a form for creating or configuring a Docker repository. It includes fields for ACCOUNT (my-org), REPOSITORY NAME, DESCRIPTION (OPTIONAL), and VISIBILITY (Public vs. Private). A 'SCAN ON PUSH' toggle is also present. Below these, there's a 'Hide advanced settings' link, followed by an 'IMMUTABILITY' section. This section contains two options: 'On' (Tags are immutable) and 'Off' (Tags can be overwritten). The 'On' option is highlighted with a red box. At the bottom right are 'Cancel' and 'Save' buttons.

KEY FEATURES

- Prevent users from altering existing tags on repositories that are marked for immutability
 - New tags can be added but pushing tags to existing tags is blocked

BENEFITS

- Improve transparency in repositories by preventing tags from being overwritten – either accidentally or maliciously



New APIs for Configuration and Management

AUTOMATE AND ORCHESTRATE DAY-TO-DAY TASKS

UCP : API endpoints which are specific to UCP

Show/Hide | List Operations | Expand Operations

GET /api/banner

Return a list of warning and informational banners for the cluster.

GET /collections

List all visible collections.

POST /collections

Create a new collection of resources that share mutual authorization settings.

Implementation Notes

Create a new collection of resources that share mutual authorization settings.

Response Class (Status 201)

Success

Model | Example Value

```
authz.CollectionCreateResponse {  
    id (string)  
}
```

Response Content Type | application/json +

Try it out!

DELETE /collections/{id}

Delete a single collection by ID.

GET /collections/{id}

Retrieve a single collection by ID.

PATCH /collections/{id}

Updates an existing collection

GET /collections/{id}/children

Retrieve all children collection to a specific collection.

DELETE /defaultcollection/{userID}

Delete the default collection setting for a user

GET /defaultcollection/{userID}

Retrieve a user's default collection.

PUT /defaultcollection/{userID}/{collectionID}

Set a user's default collection.

KEY FEATURES

- Automate and orchestrate the management of your CaaS platform with new APIs:
 - Access Control permissions
 - User / Team / Org management
 - Cluster configuration
- Interact with the API directly from the UI

BENEFITS

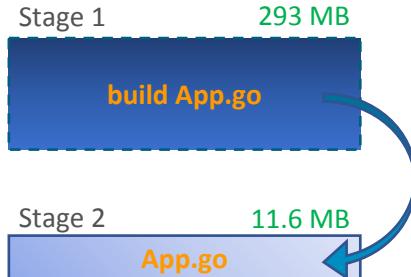
- Improve administrative efficiency with automated scripts and policies
- Improve security through programmatic handling of user roles and permissions
- Integrate with 3rd party tools with open APIs



Multi-Stage Builds

OPTIMIZE IMAGES FOR MORE EFFICIENT RESOURCE UTILIZATION

```
1 # First Stage
2 FROM golang:1.6-alpine
3
4 RUN mkdir /app
5 ADD . /app/
6 WORKDIR /app
7 RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o main .
8
9 # Second Stage
10 FROM alpine
11 EXPOSE 80
12 CMD ["/app"]
13
14 # Copy from first stage
15 COPY --from=0 /app/main /app
16
```



KEY FEATURES

- Use multiple FROM statements to define stages
- Selectively copy artifacts from one stage to another

BENEFITS

- Create thinner images, optimized for production without manual tuning
- Accelerate image pulls and deployment time while also improving resource efficiency in both the registry and resulting containers



Multi-Stage Builds in Action

```
docker build -t mikegcoleman/multi:1 .
```

```
FROM node:latest AS storefront
```

```
WORKDIR /usr/src/atsea/app/react-app
```

```
COPY react-app .
```

```
RUN npm install
```

```
RUN npm run build
```

```
FROM maven:latest AS appserver
```

```
WORKDIR /usr/src/atsea
```

```
COPY pom.xml .
```

```
RUN mvn -B -f pom.xml -s /usr/share/maven/ref/settings-docker.xml dependency:resolve
```

```
COPY ..
```

```
RUN mvn -B -s /usr/share/maven/ref/settings-docker.xml package -DskipTests
```

```
FROM java:8-jdk-alpine
```

```
RUN adduser -Dh /home/gordon gordon
```

```
WORKDIR /static
```

```
COPY --from=storefront /usr/src/atsea/app/react-app/build/ .
```

```
WORKDIR /app
```

```
COPY --from=appserver /usr/src/atsea/target/AtSea-0.0.1-SNAPSHOT.jar .
```

```
ENTRYPOINT ["java", "-jar", "/app/AtSea-0.0.1-SNAPSHOT.jar"]
```

```
CMD ["--spring.profiles.active=postgres"]
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mikegcoleman/multi	1	dcb067be6d33	17 seconds ago	210MB
<none>	<none>	5a3dbc1cc11b	24 seconds ago	889MB
<none>	<none>	203fc1a9e91a	2 minutes ago	808MB
node	latest	5e553613f1d8	6 hours ago	669MB
maven	latest	88714384d642	2 weeks ago	749MB
java	8-jdk-alpine	3fd9dd82815c	5 months ago	145MB



Expanded Access Control

CUSTOMIZE ROLES AND IMPROVE ACCESS GRANULARITY AND CONTROL

Create Grant			
	Subject	Role	Collection
User Management	Team	Restricted Control	9be4ce3f-edf6-4bfa-82e7-c7cdffaa841d4
Organizations	Team	Restricted Control	d9d9ae73-b1d3-49bc-956a-5c76c4db33dd
Users	Team	Scheduler	d9d9ae73-b1d3-49bc-956a-5c76c4db33dd
Roles	Team	Scheduler	6d55d3ba-33c2-4daf-8874-e4e7b92aaa99
Manage Grants	Team	Scheduler	swarm
Dashboard	sheila	Restricted Control	e6d42fba-5762-4f92-8049-6e221efbc7e0
Collections	vivek	Restricted Control	11870bf8-c6f2-4928-a783-aeafb28ef6e7
Stacks	vivek	Volume Maker	202fe844-ec64-4f5c-9695-98f8809580e7
Services	bob	test	6d55d3ba-33c2-4daf-8874-e4e7b92aaa99
Containers	admin	Restricted Control	d3c6e113-8578-437c-adbd-c0dfaee143ce

KEY FEATURES

- Create custom roles with granular action permissions or leverage pre-defined default roles
- Define resource collections to more easily visualize and assign users to specific cluster resources
- Apply roles to orgs, teams, or users

BENEFITS

- Easily manage complex organizations by defining permissions across user groups and resource collections
- Improve security by setting permissions that align to your organization's requirements and practices
- Meet compliance and regulatory requirements through tight access control and separation of roles and responsibilities



Collections

IDENTIFY AND LOCATE CONTAINERS AND RESOURCES MORE QUICKLY AND APPLY PERMISSIONS THROUGH RESOURCE COLLECTIONS

admin

User Management

UCP Dashboard

Stacks

Services

Containers

Images

Nodes

Networks

Volumes

Secrets

Docker

Filter Containers...					
	ID	Node	Service	Name	Image
<input type="checkbox"/>	3eac2d963ddc	qa-ddc-nightly		nginx.1.vnr6fo89mip61bf57q6j246uk	nginx:latest@sha256:12d30ce4...
<input type="checkbox"/>	48b8b9ceec04	qa-ddc-nightly	dtr-scanningstore-4c440	dockerhubenterprise/dtr-postg...	dtr-scanningstore-4c440
<input type="checkbox"/>	7c52bd87e1c4	qa-ddc-nightly	dtr-notary-signer-4c440l	dockerhubenterprise/dtr-notar...	dtr-notary-signer-4c440l
<input type="checkbox"/>	858113f5b30d	qa-ddc-nightly	dtr-jobrunner-4c440b84	dockerhubenterprise/dtr-jobru...	dtr-jobrunner-4c440b84
<input type="checkbox"/>	80e5b359b892	qa-ddc-nightly	dtr-nginx-4c440b842c1a	dockerhubenterprise/dtr-nginx...	dtr-nginx-4c440b842c1a
<input type="checkbox"/>	4b1bef8af6de	qa-ddc-nightly	dtr-notary-server-4c440l	dockerhubenterprise/dtr-notar...	dtr-notary-server-4c440l
<input type="checkbox"/>	a3e0df262f15	qa-ddc-nightly	dtr-api-4c440b842c1a	dockerhubenterprise/dtr-api:2...	dtr-api-4c440b842c1a
<input type="checkbox"/>	3732e2a4693c	qa-ddc-nightly	dtr-garant-4c440b842c1.	dockerhubenterprise/dtr-garan...	dtr-garant-4c440b842c1.
<input type="checkbox"/>	8b2b3005cfa5	qa-ddc-nightly	dtr-registry-4c440b842c	dockerhubenterprise/dtr-regis...	dtr-registry-4c440b842c
<input type="checkbox"/>	12b83e0bfad3	qa-ddc-nightly	dtr-rethinkdb-4c440b842	dockerhubenterprise/dtr-rethi...	dtr-rethinkdb-4c440b842
<input type="checkbox"/>	4878533659f7	qa-ddc-nightly	ucp-port-check-12376	dockerorcadef/ucp-agent:2.2.0...	ucp-port-check-12376
<input type="checkbox"/>	92f7feb24e5	qa-ddc-nightly	ucp-port-check-12385	dockerorcadef/ucp-agent:2.2.0...	ucp-port-check-12385
<input type="checkbox"/>	11b554917641	qa-ddc-nightly	ucp-port-check-12380	dockerorcadef/ucp-agent:2.2.0...	ucp-port-check-12380

DETAILS

Id
3eac2d963ddc0873cc873ef73afa35b22835bc4
cf3b53ee16431531f4876a

Name
/nginx.1.vnr6fo89mip61bf57q6j246uk

Image
sha256:3448f27c273f3122fc554d7acf33796efb4d
f2ad9886efc092c3bf7e10e897b7

Restart Policy

Entrypoint

Command
nginx -g daemon off;

Created
2017-05-15T16:33:08.322416013Z

STATUS (RUNNING)

NODE (QA-DDC-NIGHTLY-051404-220-LATEST-WORKER-1)

MOUNTS ()

ENVIRONMENT (3)

LABELS (6)

NODE LABELS (5)

NETWORKS (INGRESS)

KEY FEATURES

- Create and filter by resource collections
- Inspect resource details
- Faster response time
- Used to apply permissions
- Defined via Docker label



Roles

IMAGE OPERATIONS

All Image operations ▾

NETWORK OPERATIONS

All Network operations ^

Network Connect

Network Create

Network Create parameters ▾

Network Disconnect

Network Remove

Network View

NODE OPERATIONS

All Node operations ^

Node Schedule

Node Update

Node View

Default roles

These can be selected to create arbitrary custom roles

Very granular permissions

Control a single operation (network create or node schedule)



Enhanced RBAC: Additional Details

Subject:



Role: What they can do

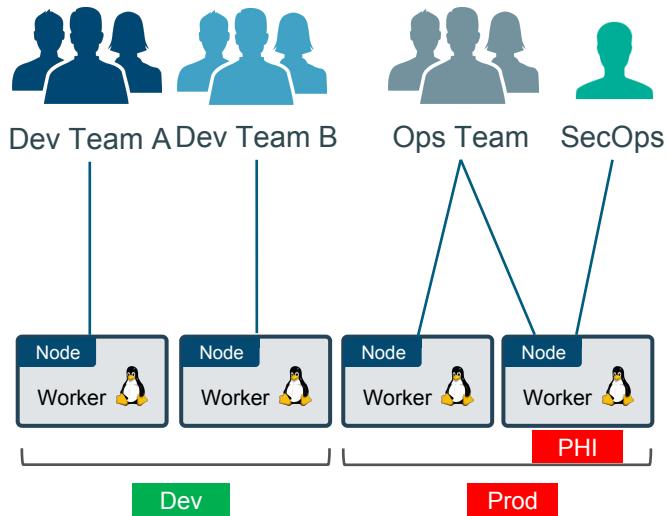
Collection: Where work can be done

Grant: 1:1:1 mapping of subject to role to collection.



RBAC for Nodes

SUPPORT SECURE MULTI-TENANCY ACROSS MULTIPLE TEAMS THROUGH NODE-BASED ISOLATION AND SEGREGATION



KEY FEATURES

- Enforce node affinity and node anti-affinity rules to allow certain users/teams/orgs to deploy within a subset of nodes or outside certain nodes (eg. Production nodes)
- Set up different security zones within the same cluster to isolate and segregate access of protected information

BENEFITS

- Support multiple teams within the same cluster while providing physical separation and isolation
- Prevent “noisy neighbors” by limiting a team’s resources to approved nodes
- Meet compliance and regulatory requirements by isolating sensitive workloads to certain nodes and limiting access to those nodes



Additional Enhancements

- “docker service logs”
 - Batch-retrieve logs for the entire service
 - Previously experimental, now GA
- “docker prune --filter label”
 - Prune on an individual node
 - Target or avoid specific labels (e.g. label critical system or app resources)
- Healthcheck grace periods
 - `--start-period` provides delay before health check activates
 - Useful for long startup times
- Update/rollback order
 - Define whether old task is stopped before starting new one, or if old and new tasks should overlap
- Webhooks UI
 - Set up notifications for other 3rd party and monitoring tools
- Chinese language support

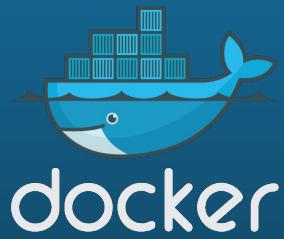


Docker Subscription

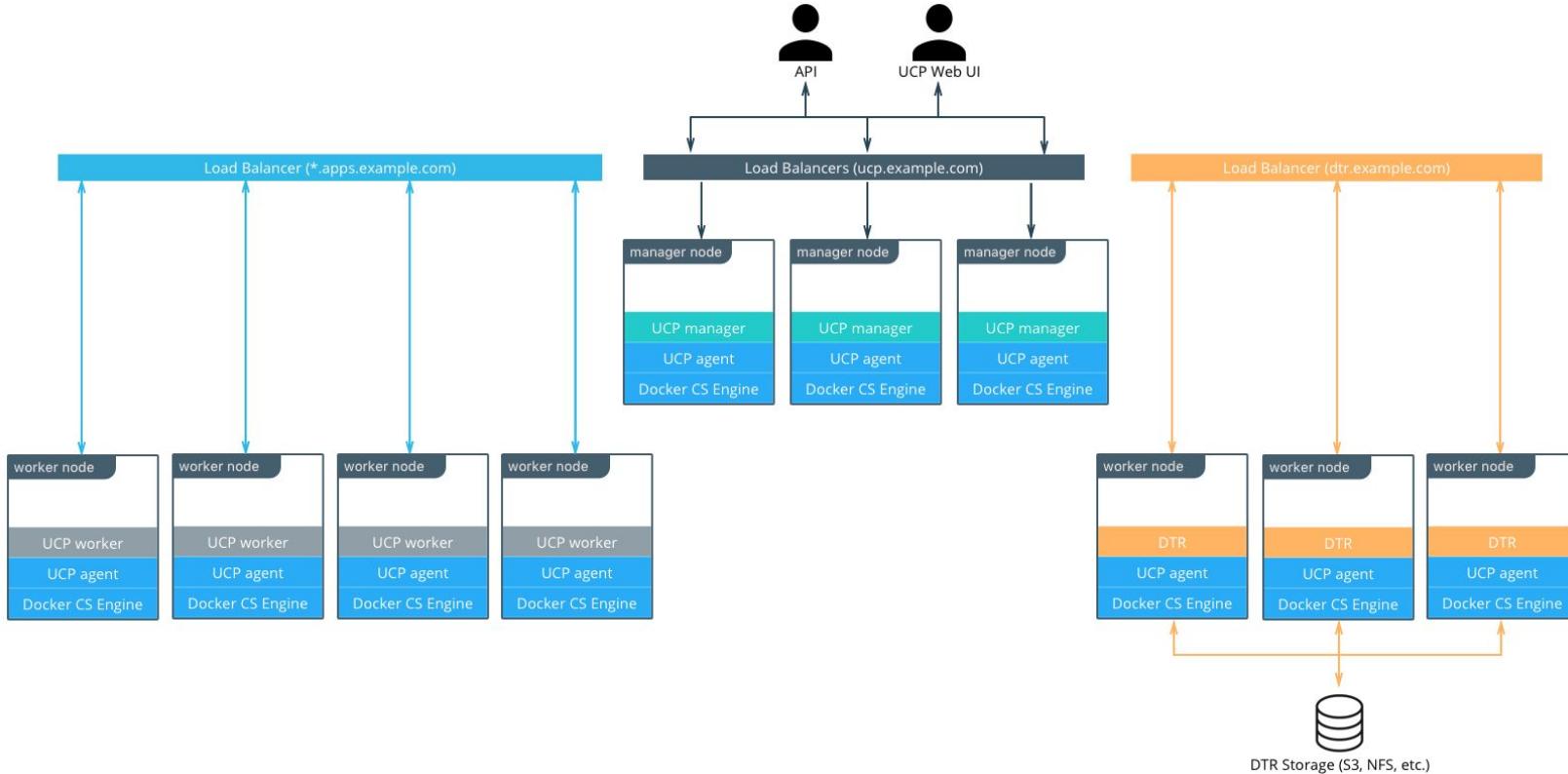
	CE	EE Basic	EE Standard	EE Advanced
		Supported container runtime and orchestration	Integrated security and management	Multi-tenancy and specialized compliance
Support	Community Support	Business Day or Business Critical	Business Day or Business Critical	Business Day or Business Critical
Container engine and built in orchestration, networking, security, plugins	x	x	x	x
CaaS enabled platform supporting both Windows and Linux		x	x	x
Docker Certified Infra, Plugins and ISV Containers		x	x	x
Image management with private registry, caching (Windows and Linux)			x	x
Integrated container app management			x	x
Multi-tenancy with RBAC, LDAP/AD			x	x
Integrated secrets mgmt, image signing, policy (Windows and Linux)			x	x
Secure multi-tenancy				x
Automated image promotion				x
Image security scanning and continuous vulnerability monitoring (Windows and Linux)				x



UCP Architecture



Docker Enterprise Advanced Architecture



UCP Building Blocks

UCP component	Description
ucp-agent	Monitors the node and ensures the right UCP services are running
ucp-reconcile	When ucp-agent detects that the node is not running the right UCP components, it starts the ucp-reconcile container to converge the node to its desired state. It is expected for the ucp-reconcile container to remain in an exited state when the node is healthy.
ucp-auth-api	The centralized service for identity and authentication used by UCP and DTR
ucp-auth-store	Stores authentication configurations, and data for users, organizations and teams
ucp-auth-worker	Performs scheduled LDAP synchronizations and cleans authentication and authorization data
ucp-client-root-ca	A certificate authority to sign client bundles
ucp-cluster-root-ca	A certificate authority used for TLS communication between UCP components
ucp-controller	The UCP web server
ucp-kv	Used to store the UCP configurations. Don't use it in your applications, since it's for internal use only
ucp-metrics	Used to collect and process metrics for a node, like the disk space available
ucp-proxy	A TLS proxy. It allows secure access to the local Docker Engine to UCP components
ucp-swarm-manager	Used to provide backwards-compatibility with Docker Swarm

Client Bundle

```
root @ ~/bundle/ucp.dockerenterpris.dckr.org [111] ↵ → tree
.
└── ca.pem
└── cert.pem
└── cert.pub
└── env.cmd
└── env.ps1
└── env.sh
└── key.pem

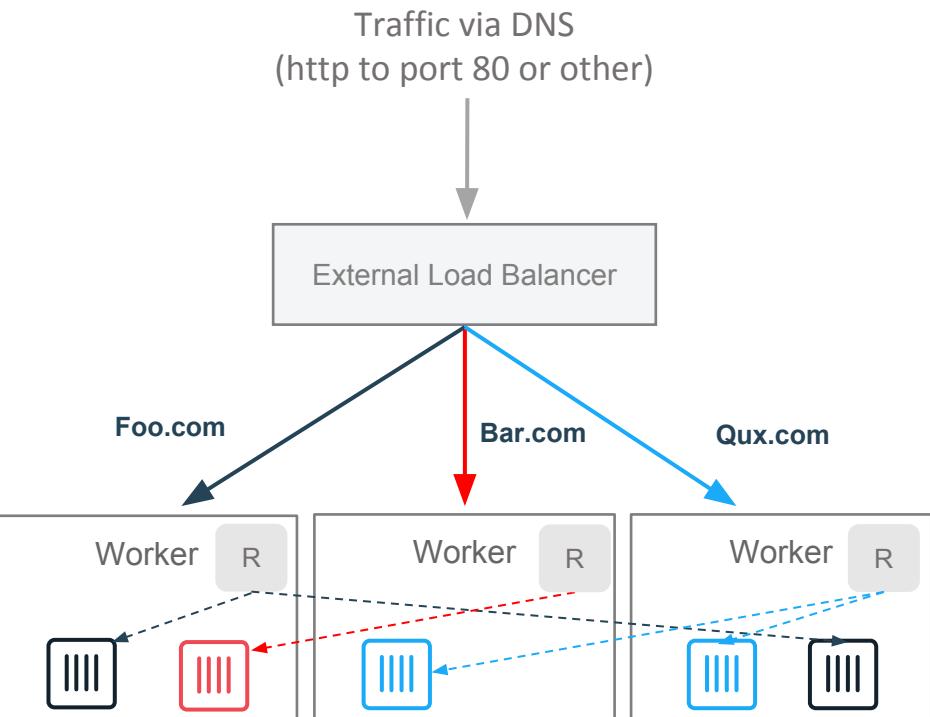
0 directories, 7 files

root @ ~/bundle/ucp.dockerenterpris.dckr.org [112] ↵ → openssl x509 -noout -text -in ca.pem | grep 'Subject\\!Issuer'
Issuer: O=Digital Signature Trust Co., CN=DST Root CA X3
Subject: O=Digital Signature Trust Co., CN=DST Root CA X3
Subject Public Key Info:
    X509v3 Subject Key Identifier:
        X509v3 Subject Key Identifier:
        X509v3 Subject Alternative Name:

root @ ~/bundle/ucp.dockerenterpris.dckr.org [113] ↵ → openssl x509 -noout -text -in cert.pem | grep 'Subject\\!Issuer'
Issuer: CN=swarm-ca
Subject: C=, ST=, L=, O=Orca: NZDQ:ES53:V5PR:POSH:Z2BI:EI3D:CAEA:45SM:LHTP:QA3Z:Q33E:A4U2, OU=Client, CN=moby
Subject Public Key Info:
    X509v3 Subject Key Identifier:
    X509v3 Subject Alternative Name:

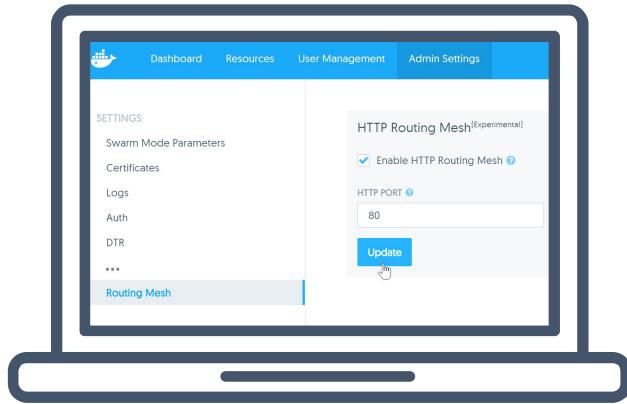
root @ ~/bundle/ucp.dockerenterpris.dckr.org [114] ↵ → cat env.sh
export DOCKER_TLS_VERIFY=1
export DOCKER_CERT_PATH="$(pwd)"
export DOCKER_HOST=tcp://ucp.dockerenterprise.dckr.org:443
#
# Bundle for user moby
# UCP Instance ID NZDQ:ES53:V5PR:POSH:Z2BI:EI3D:CAEA:45SM:LHTP:QA3Z:Q33E:A4U2
#
# This admin cert will also work directly against Swarm and the individual
# engine proxies for troubleshooting. After sourcing this env file, use
# "docker info" to discover the location of Swarm managers and engines.
# and use the --host option to override $DOCKER_HOST
#
# Run this command from within this directory to configure your shell:
# eval $(<env.sh)
```

Built in HTTP Routing Mesh



- Extend TCP routing mesh to HTTP hostname routing for services
- HTTPS support via SNI protocol
- Support for multiple HRM networks for enhanced app isolation
- External LB routes hostnames to nodes
- Can add hostname routing via UI
- Non-service containers continue to use Interlock ref arch

Enabling and Using the HTTP Routing Mesh



Enable HTTP routing mesh in DDC

- 1 a) Creates **ucp-hrm** *network*
b) Creates **ucp-hrm** *service* and exposes it on a port (80 by default)

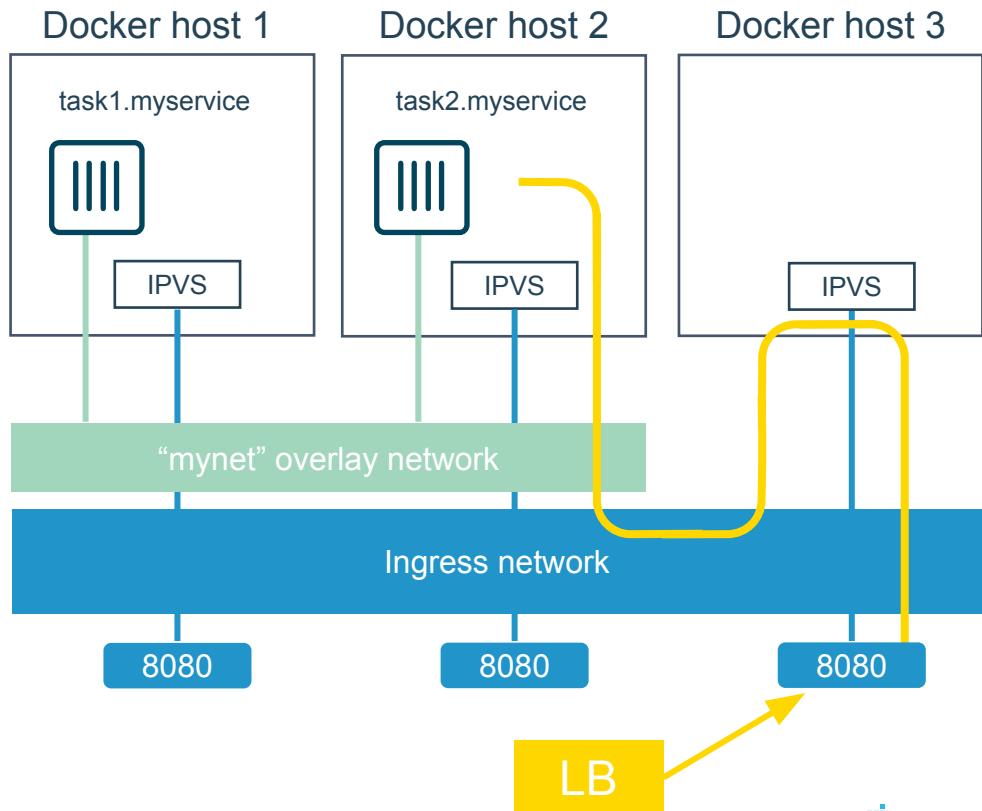
```
docker service create -p 8080 \
\--network ucp-hrm \
\--label
com.docker.ucp.mesh.http=8080=
http://foo.example.com \
...
```

Create new service

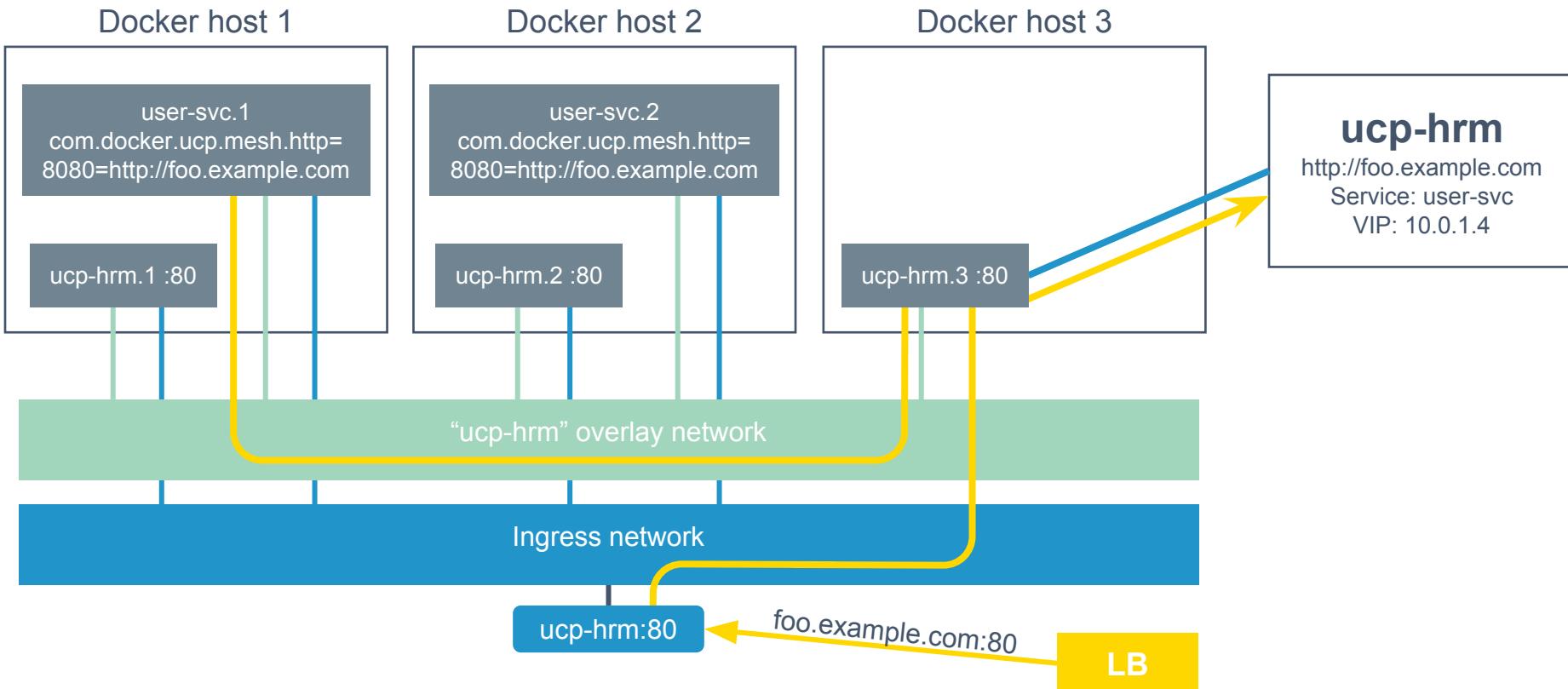
- 2 a) **Add to ucp-hrm** *network*
b) **Assign label** specifying hostname
(links service to <http://foo.example.com>)

Routing Mesh Example

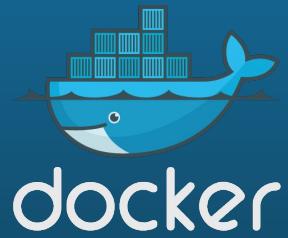
1. Three Docker hosts
2. New service with 2 tasks
3. Connected to the **mynet** overlay network
4. Service published on port 8080 swarm-wide
5. External LB sends request to Docker host 3 on port 8080
6. Routing mesh forwards the request to a healthy task using the ingress network



HTTP Routing Mesh Example



UCP Operations

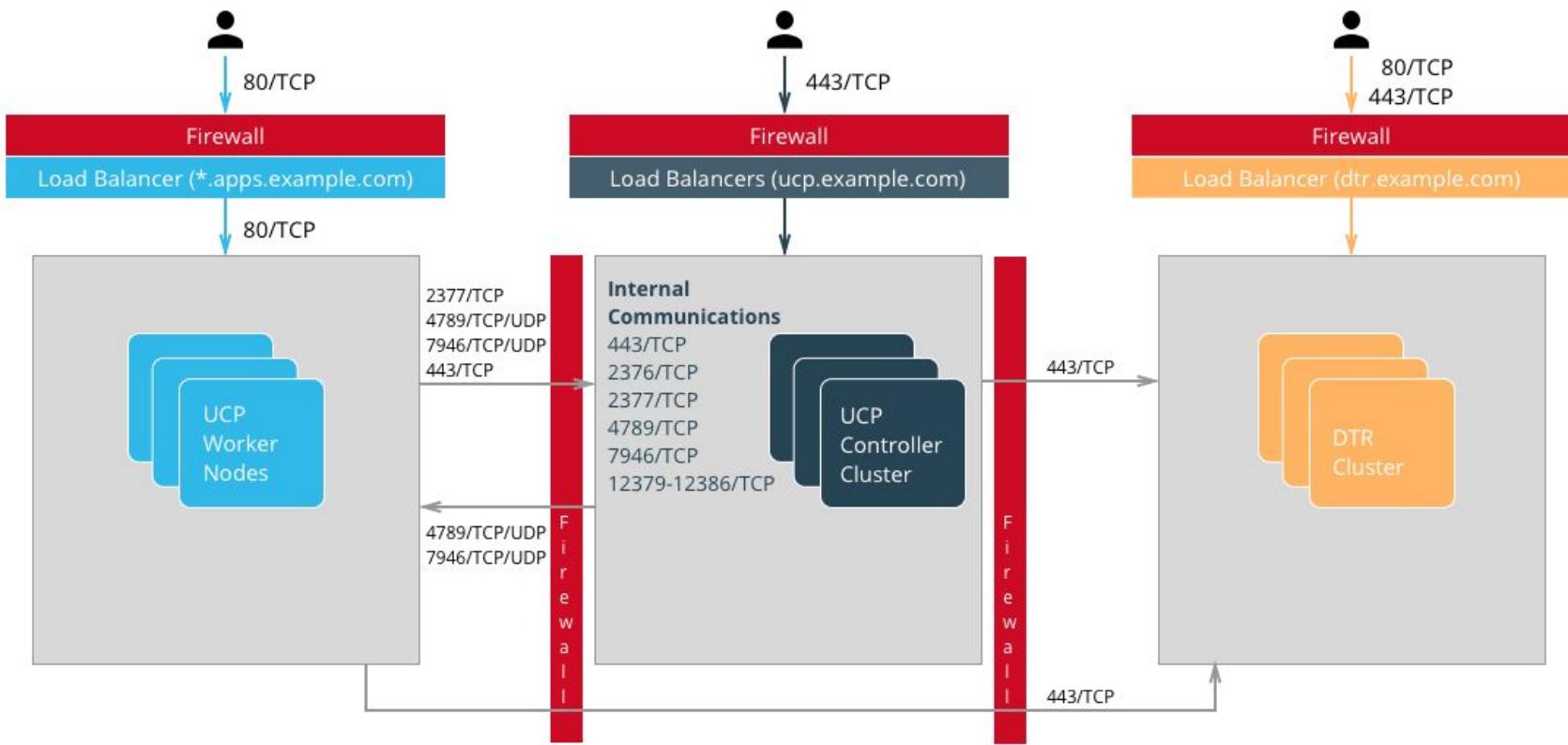


UCP Installation

- Single operation on Swarm manager node
- UCP will bootstrap itself on every node in the cluster

```
# Install UCP
$ docker container run --rm -it --name ucp \
-v /var/run/docker.sock:/var/run/docker.sock \
docker/ucp:2.2.2 install \
--host-address <node-ip-address> \
--controller-port <port> \
--san team1.ucp-1b.dockerpartners.org \
--debug \
--interactive
```

Environment Network Requirements



Network Latency Consideration

Role	Component	Heartbeat (ms)	Timeout (ms)	Impact
Swarm Manager	Raft (SwarmKit)*	1000	3000	swarm ops
Swarm Manager, Swarm Worker	Gossip (libnetwork)	1000	5000	overlay, ingress, routing mesh, dns, lb
UCP Manager	Etcd*	500	5000	UCP ops.
UCP Manager, DTR	Rethink*	2000	10000	UCP and DTR Ops.

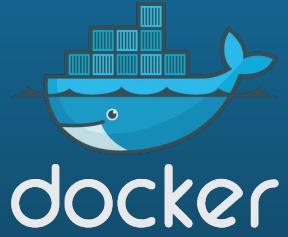
Load Balancer Configuration

- DNS entry for UCP
- TCP / SSL passthrough
- Set idle timeout to 300s
- Health checks
 - `https://<ucp_controller>/_ping`

Backup & Restore

- Set automated backups (separate for UCP and DTR)
- Impact
 - HA: one manager will be down → no impact
 - non-HA: there will be some downtime (no impact to applications)
- What do Backups do:
 - Users, Teams and Permissions.
 - Cluster Configuration, such as the default Controller Port or the KV store timeout.
 - DDC Subscription License.
 - Options on Scheduling, Content Trust, Authentication Methods and Reporting
- ALWAYS backup before upgrades!
- Swarm Backup is separate (need to backup /var/lib/docker/swarm)

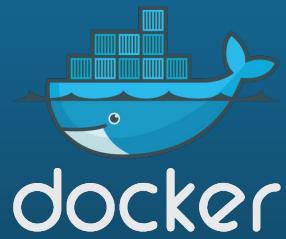
UCP Lab



Lab 2 Summary

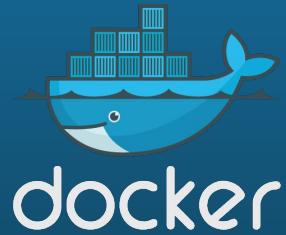
- Installed Docker EE Standard (UCP)
- Generated Let's Encrypt Certs
- Installed generated certs
- Configured a Loadbalancer
- Installed your license
- Set up HRM
- Configured node labels
- Performed a backup

Break



Docker Enterprise Edition Standard

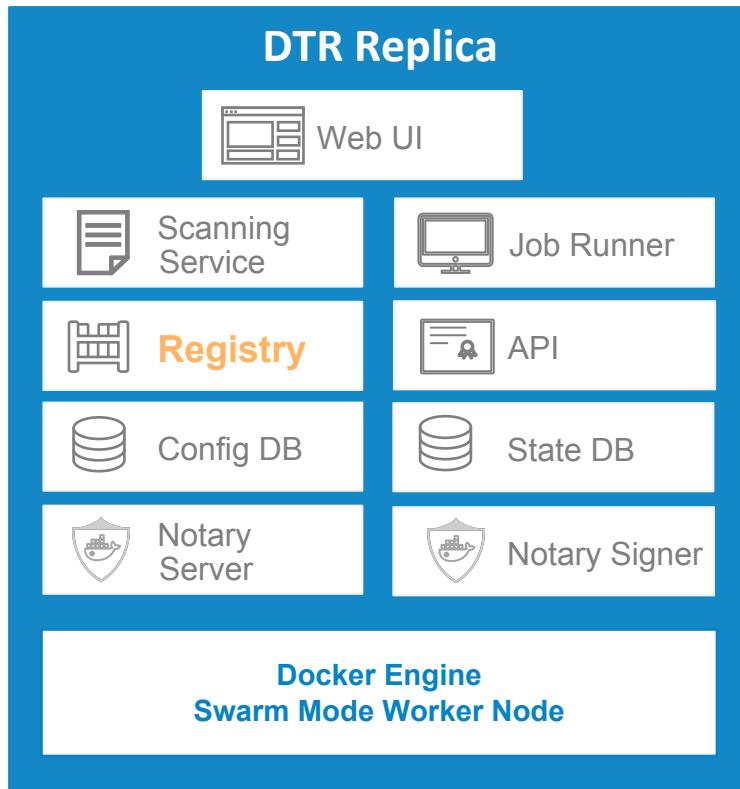
Docker Trusted Registry



DTR Overview

- Overview and Value Prop
- Architecture
- Deep Dives
 - API
 - Storage
 - Garbage Collection
 - Configuration Data
 - Content Caching
 - Webhooks
- Installation Best Practices
- Troubleshooting

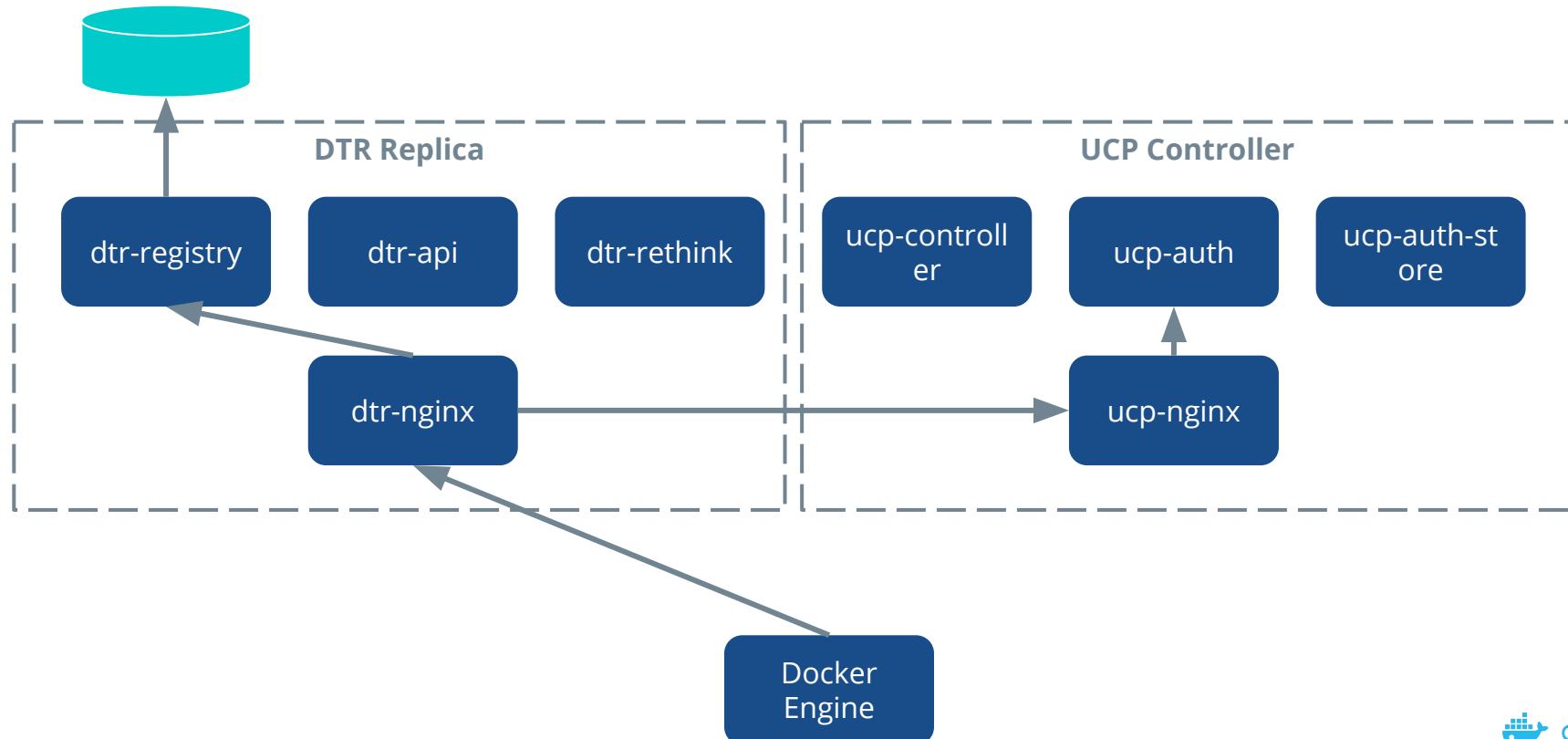
Deep Dive: DTR Replica Worker Nodes



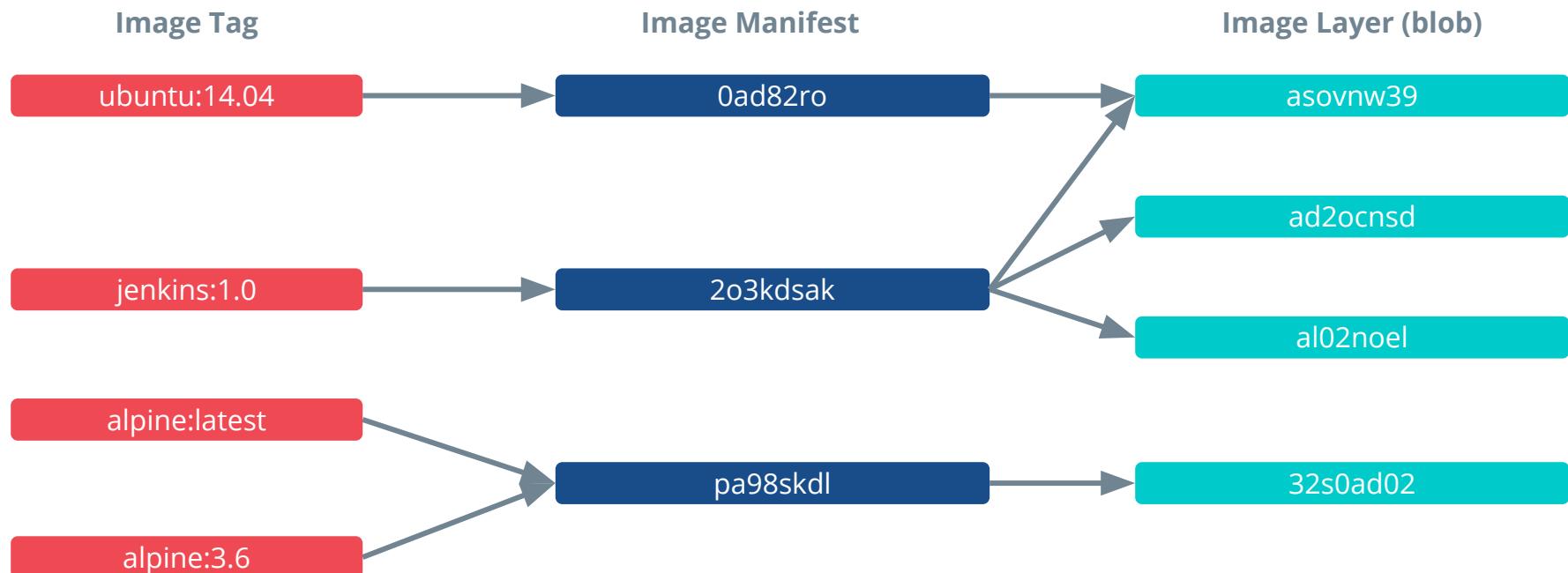
- Point and click UI to manage repos, images and team collaboration
- Image management with labels, tag store and garbage collection
- HA and redundant system
- Content security with built in image signing and verification
- Wide variety of storage driver support for image store

DTR Architecture

NFS/Object Storage



DTR Image Objects



DTR State

NFS/Object Storage



- Image Blobs
- Image Manifests

dtr-rethink

- Repository Structure/Metadata
- DTR Configurations
- Repo-Team Membership

AD/LDAP

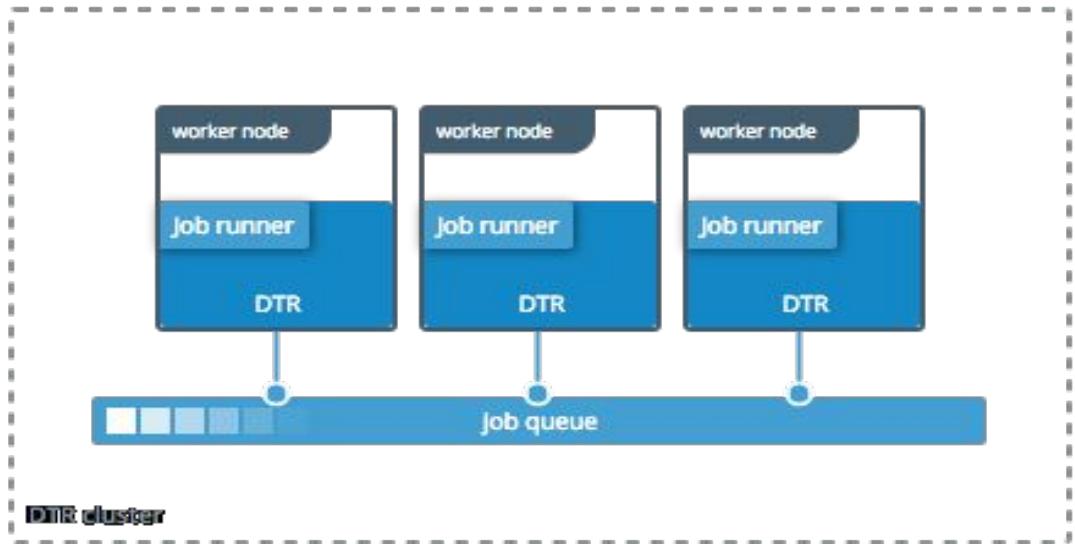
ucp-auth

DTR Jobs

- Garbage Collection
- Security Vulnerability Scanning
- Webhooks
- Image Promotions

Future

- Image Mirroring
- Image Pruning Policies



Name	Description
dtr-api-<replica_id>	Executes the DTR business logic. It serves the DTR web application, and API
dtr-garant-<replica_id>	Manages DTR authentication
dtr-jobrunner-<replica_id>	Runs cleanup jobs in the background
dtr-nautilusstore-<replica_id>	Stores security scanning data
dtr-nginx-<replica_id>	Receives http and https requests and proxies them to other DTR components. By default it listens to ports 80 and 443 of the host
dtr-notary-server-<replica_id>	Receives, validates, and serves content trust metadata, and is consulted when pushing or pulling to DTR with content trust enabled
dtr-notary-signer-<replica_id>	Performs server-side timestamp and snapshot signing for content trust metadata
dtr-registry-<replica_id>	Implements the functionality for pulling and pushing Docker images. It also handles how images are stored
dtr-rethinkdb-<replica_id>	A database for persisting repository metadata

DTR Building Blocks

DTR API



The header of the DTR API documentation page. It features the Docker Trusted Registry logo, a search bar with a magnifying glass icon, and a user profile icon labeled "moby". Below the header is a blue navigation bar with "Docs" and "API" tabs.

API

DTR 2.2.2 API Documentation

api/v0/accounts : Accounts

Show/Hide | List Operations | Expand Operations

GET	/api/v0/accounts/{orgname}/teams/{teamname}/repositoryAccess	List repository access grants for a team
GET	/api/v0/accounts/{username}/repositoryAccess/{namespace}/{reponame}	Check a user's access to a repository
GET	/api/v0/accounts/{username}/settings	Check a user's settings
PATCH	/api/v0/accounts/{username}/settings	Update a user's settings
DELETE	/api/v0/accounts/{namespace}	Removes a user or organization along with all repositories
DELETE	/api/v0/accounts/{namespace}/repositories	Removes all of a user or organization's repositories
GET	/api/v0/accounts/{namespace}/webhooks	List the webhook subscriptions for a namespace

api/v0/action_configs : ActionConfigs

Show/Hide | List Operations | Expand Operations

GET	/api/v0/action_configs	List all action config
POST	/api/v0/action_configs	Configure actions
DELETE	/api/v0/action_configs/{action}	Delete the action config. The defaults will be used.
GET	/api/v0/action_configs/{action}	Get info about the actionConfig with the given action

api/v0/content_caches : Content Caches

Show/Hide | List Operations | Expand Operations



** This is not
the registry API

Garbage Collection

- Garbage collection is the process of removing blobs from the filesystem which are no longer referenced by a manifest. Blobs can include both layers and manifests.
- Configured in DTR
- Runs as a Cron job

The screenshot shows the 'GARBAGE COLLECTION' tab selected in the top navigation bar. Below it, there's a section titled 'Remove Untagged Images' with a broom icon. A button labeled 'Run garbage collection on your storage backend to remove deleted tags and images. Learn more' is present. Underneath, there's a 'Delete images' section with three options: 'Until done' (selected), 'For 1 minutes', and 'Never'. The 'Never' option is highlighted with a blue border. At the bottom is a 'Save' button.

Until done This may take a while	For 1 minutes	Never Disable garbage collection
-------------------------------------	---------------	-------------------------------------

Save

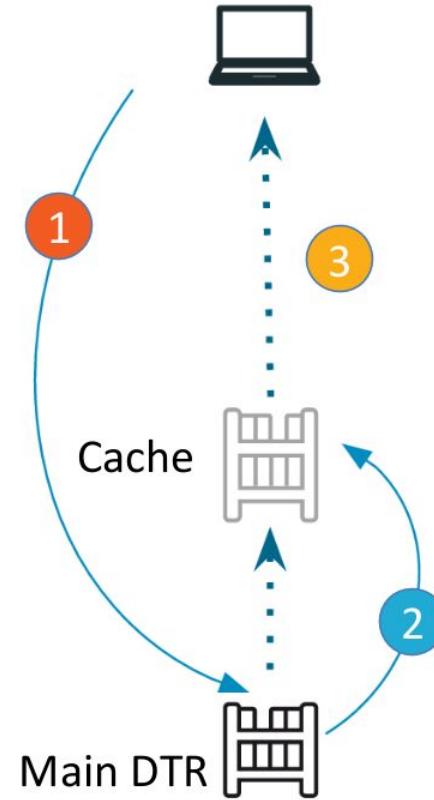
Configuration Data

- YAML configuration of the registry component
- Can be configured in UI or API
- Stored in distributed DB and presented to each replica as a YAML config file

The screenshot shows a user interface for configuring storage settings. At the top, there are four tabs: GENERAL, STORAGE (which is selected), SECURITY, and GARBAGE COLLECTION. Below the tabs, a message reads "Set up your storage with a". There are two radio button options: "Manual form" (unchecked) and "YAML file" (checked). A large input field labeled "YAML file" contains the placeholder text "Select a file (.yml, .yaml, .txt)". Below this is a blue "Upload" button. At the bottom of the section is a blue "Download YAML" button.

Content Caching

- Faster Pulls
- Reduces WAN Traffic
- Per User Configuration
- Chaining OK
- Configurable TTL
- Supports Same Storage backends as DTR
- Advanced Caching (e.g Secure Image Caching)



Security Scanning: Get a full BOM for a Docker Image

The screenshot shows the Docker Trusted Registry interface. At the top, there's a search bar and a user profile icon. Below the header, the URL indicates the repository: `Repositories > enterprise/voting-app > latest, ver6.3.2`. The main card displays the image details: `enterprise/voting-app`, `latest`, `ver6.3.2` (private), pushed 11 hours ago by `admin`, and `SIGNED`. It also shows `6 critical`, `11 major`, and `17 minor` vulnerabilities, with a note that all layers are already scanned. A yellow circle with the number `1` is visible in the top right corner.

Below the card, there are two tabs: `Layers` (selected) and `Components`. The `Layers` tab shows the Dockerfile content:

```
1 ADD file:cd937b840fff16e04e1f59d56f4424d08544b0bb8a...  
2 RUN set -xe && echo '#!/bin/sh' > /usr/sbin/policy-rc.d && echo 'exit 101' >> /usr/sbin/policy-rc.d && chmod +x /usr/sbin/policy-rc.d &&...  
3 RUN rm -rf /var/lib/apt/lists/*  
4 RUN sed -i 's/^#\s*\(\deb.*universe\)\s*/\1/g' /etc/apt/sources.list  
5 RUN mkdir -p /run/systemd  
6 CMD ["/bin/bash"]  
7 ENV AEROSPIKE_VERSION=3.9.1.1
```

The `Components` tab shows the following data:

Component	Critical	Major	Minor
apt 1.11.1	1	1	2
closedssl 2.0.2	3	3	4
cron 0.1		1	major
dash 4.0.12			1 minor
adduser 2.3.1			

A red banner at the bottom of the page reads: `Scanning your Dockerfiles for security issues is now available in the Docker Hub registry. Learn more`.

Security Scanning: Vulnerabilities and Licensing for Each Component

The screenshot shows a Docker security scanning interface for a repository named "enterprise/voting-app". The repository details are as follows:

- Latest tag: ver6.3.2
- Pushed 11 hours ago by admin
- Signed status: SIGNED
- Vulnerabilities: 6 critical, 11 major, 17 minor
- Scanned 22 minutes ago

The interface includes tabs for "Layers" and "Components". The "Components" tab is selected, displaying a list of dependencies and their security status.

Component	Version	Licenses	Vulnerabilities
closedssl	2.0.2	Apache 2.0 (PERMISSIVE)	3 critical, 3 major, 8 minor
closedssl	3.1.2	Apache 2.0 (PERMISSIVE)	1 critical, 1 major, 2 minor
closedssl	5.0.0	Apache 2.0 (PERMISSIVE)	1 critical, 2 minor
semi-openssl	1.0.5-0ubuntu15.3	Apache 2.0 (PERMISSIVE)	1 critical
bhb5-fpm	1.0.3	Apache 2.0 (PERMISSIVE)	2 major, 2 minor

The detailed view for the "closedssl" component (version 2.0.2) shows the following vulnerabilities:

CVE ID	Severity	Description
CVE-2014-0160	critical	The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, related to d1_both.c and t1_lib.c, aka the Heartbleed bug.
CVE-2015-0163	critical	The WUT implementation in CloseSSL will somehow implode, causing packets to sprinkle all around your application. This bug (that does not exist since it's not 2017 yet) will be installed in every one of your repositories without a check or test.
CVE-2012-0002	critical	Overflow in text-util/colcrt.c in util-finux 3.1 allows users to cause a denial of service (crash).
CVE-2013-0002	major	Overflow in text-util/colcrt.c in util-finux 3.1 allows users to cause a denial of service (crash).

Security Scanning: Set Automated Policy for Scanning

The screenshot shows the 'SETTINGS' tab of a GitHub repository named 'enterprise/voting-app'. The repository is private and has a description: 'The epic battle between two arbitrary choices continues'.

General

VISIBILITY

- Public: Visible to everyone
- Private: Hide this repository (selected)

DESCRIPTION

The epic battle between two arbitrary choices continues

Image scanning

Check for vulnerabilities in your images.

[Learn more ↗](#)

Scan on push & Scan manually

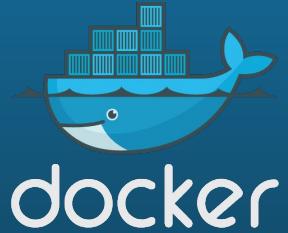
Every image gets automatically scanned on push.

Delete repository

This cannot be undone!

Delete

DTR Installation & Operations



DTR Installation

- Install first replica with the “install” bootstrapper operation
- Join other replicas with the “join” bootstrapper operation

```
# Installing First DTR Replica

$ docker run -it --rm \
  docker/dtr install \
  --ucp-node <ucp-node-name> \
  --ucp-insecure-tls

# Joining other Replicas to existing DTR

$ docker run -it --rm \
  docker/dtr join \
  --ucp-node <ucp-node-name> \
  --existing-replica-id <id> \
  --ucp-insecure-tls
```

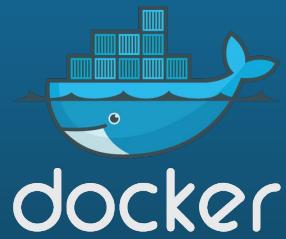
Backup and Restore

- Set automated backups (separate for UCP and DTR)
- What do Backups do:
 - Configurations
 - Repository metadata
 - Certificates and keys used by DTR.
- What they do NOT do:
 - Backup image blobs
- ALWAYS backup before upgrades!

Load Balancer Configuration

- DNS entry for DTR
- TCP / SSL passthrough
- Health checks
 - `https://<dtr_replica>/health` OR
 - `https://<dtr_replica>/_ping` (similar to UCP)

DTR Lab

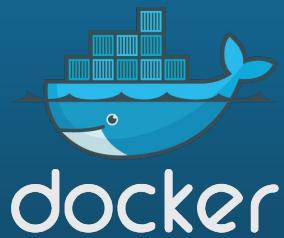


Lab 4 Summary

- Generated valid certs
- Installed DTR LB
- Installed DTR
- Configured Certs
- Configured storage
- Configured security scanning
- Created repos
- Backup DTR
- Upgraded DTR

Day 3

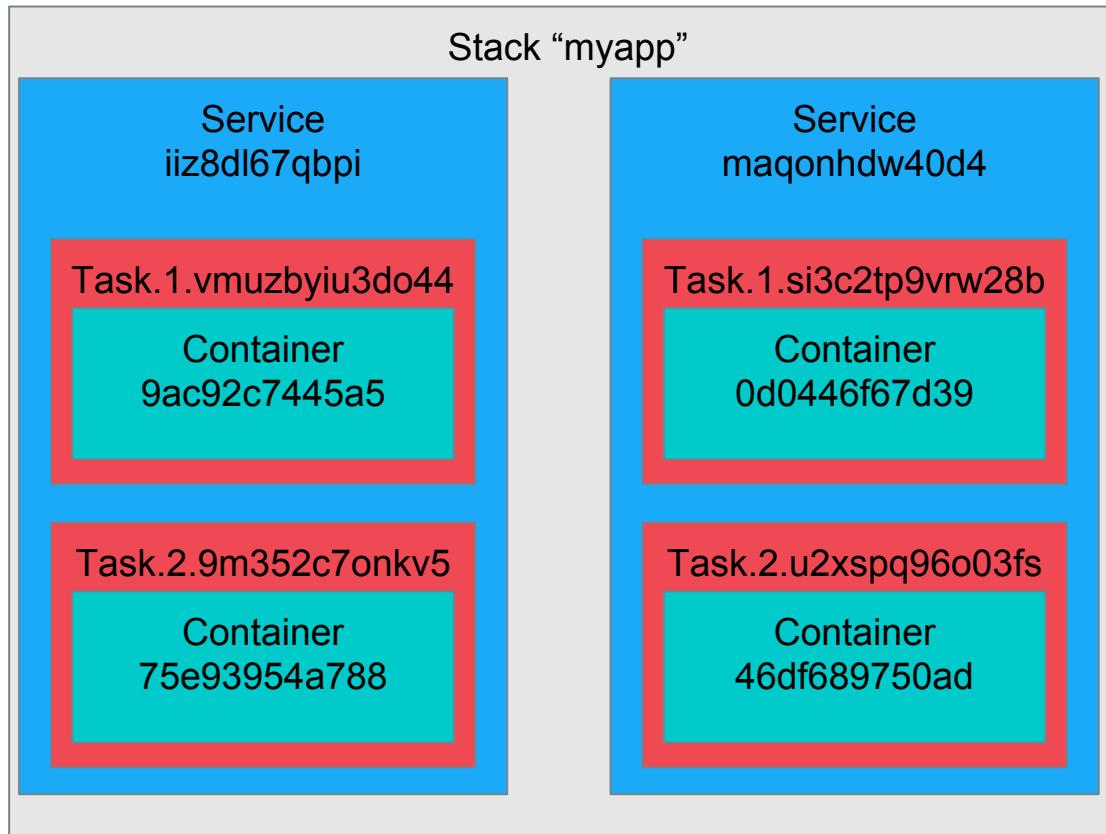
MTA Application Deployment on EE



Application Deployment on Docker EE

- Stacks, Services, Tasks and Containers
- Scheduling Modes
- Health Checks
- Networks
- Secrets
- Config
- Rolling Updates
- Labels & Constraints

Stacks, Services, Tasks and Containers



Stacks

- A Stack is a collection of services making up an application.
- Define a stack using a compose yaml file.
- Manage stacks using cli or UCP UI.

```
# Deploy or update a stack
$ docker stack deploy -c pets-prod-compose.yml pets

# List all stacks
$ docker stack ls

# List all tasks of a stack
$ docker stack ps pets

# List all services of a specific stack
$ docker stack services pets

# Remove a stack
$ docker stack rm pets
```

Services

- A service represents a single unique component of an application.
- Composed of one or more tasks
- Created in docker cli, compose, and UCP

```
# Create a Service
$ docker service create --name proxy nginx

# Inspect an existing Service
$ docker service inspect proxy

# View all service tasks logs.
$ docker service logs proxy

# List all services
$ docker service ls

# List all service tasks along with their status
$ docker service ps proxy

# Remove a service
$ docker service rm proxy

# Scale a service
$ docker service scale proxy=5

# Update a Service
$ docker service update --image nginx:v2 proxy
```

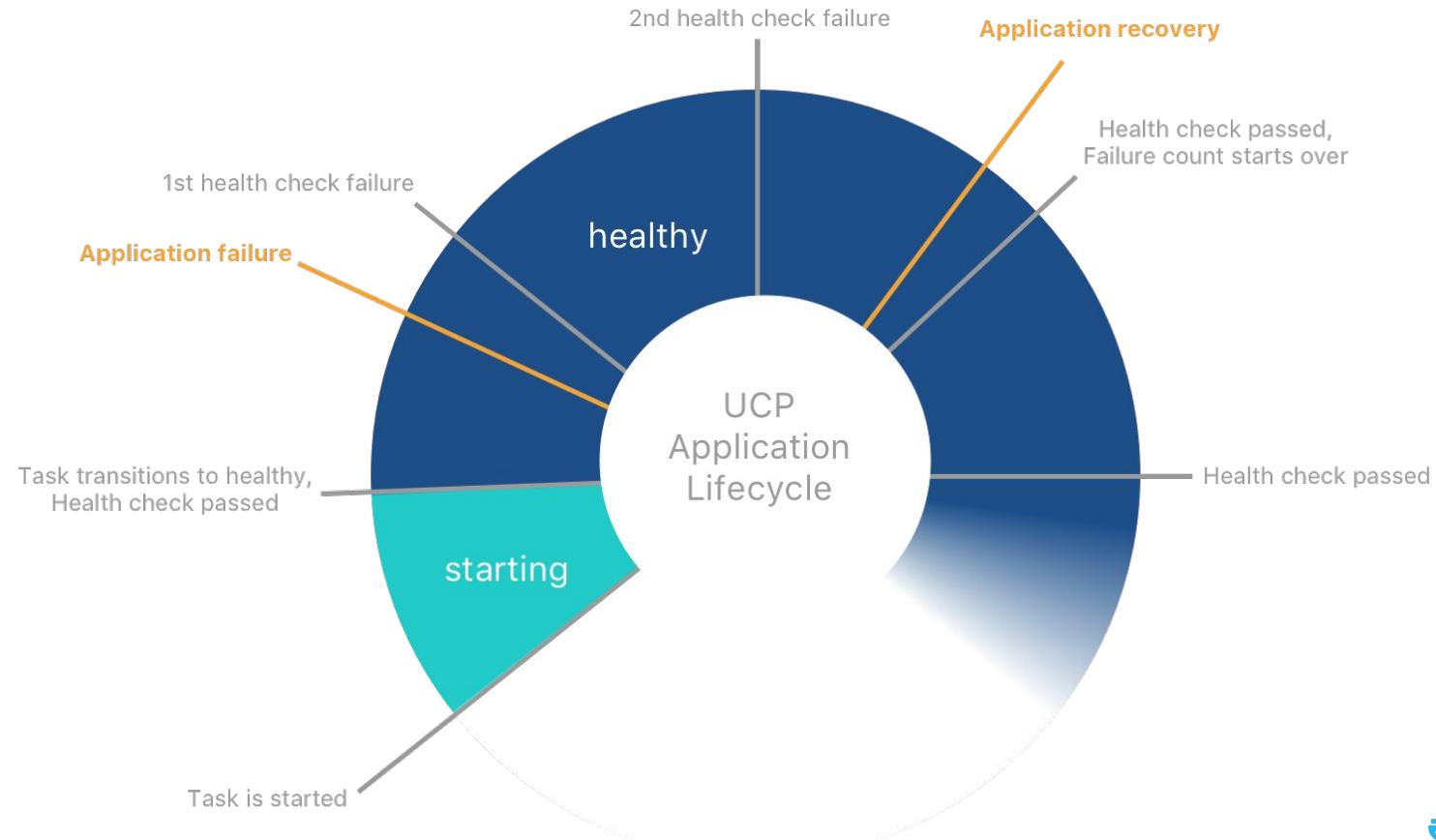
Scheduling Mode

- Global: runs a task on every (worker or manager) node in the cluster
- Replicated: runs N number of tasks across the cluster based on scheduling algorithm + provided constraint

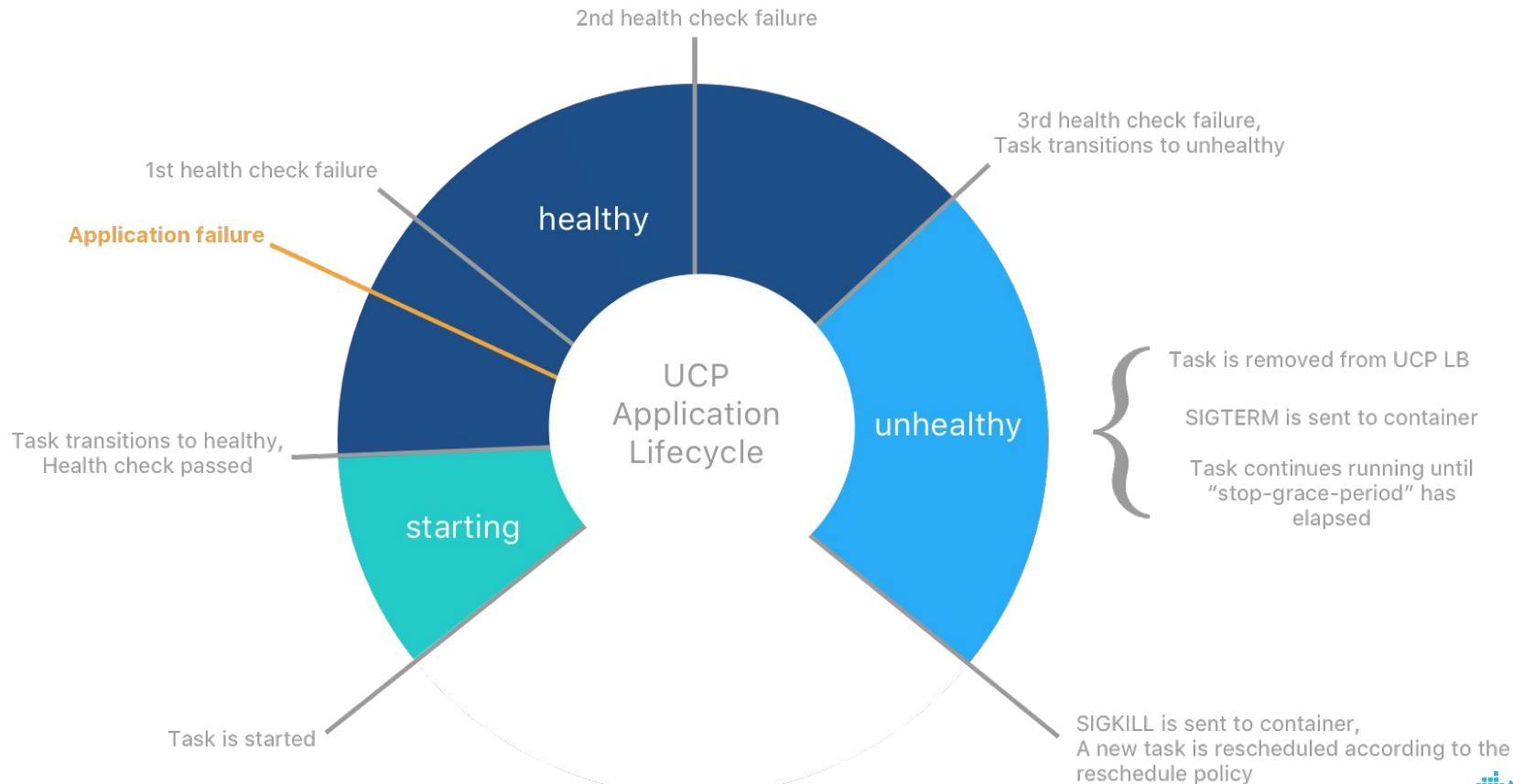
```
# Docker CLI  
  
docker service create --name=proxy --mode=global nginx  
  
# Docker Compose Syntax  
  
services:  
  proxy:  
    image: nginx  
    deploy:  
      mode: global
```

```
# Docker CLI  
  
docker service create --name=proxy --mode=replicated --replicas=2 nginx  
  
# Docker Compose Syntax  
  
services:  
  proxy:  
    image: nginx  
    deploy:  
      mode: replicated  
      replicas: 2
```

Health Checks



Health Checks



Health Checks

```
# Dockerfile

HEALTHCHECK CMD curl --fail http://localhost || exit 1

# CLI

docker service create --health-cmd "curl --fail http://localhost || exit 1" nginx

# Compose File

services:
  proxy:
    image: nginx
    deploy:
      mode: replicated
      replicas: 2
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 10s
      timeout: 2s
      retries: 3
```

Networks and Load Balancing

- Default unpublished services will use “bridge” network of the node they get deployed on
- Published services will use `ingress` network by default and all nodes in cluster will listen on the published port (if any)
- If using `--publish mode=host`, task will be exposed on the local host only and attached to the `docker_gwbridge`
- If attached to an overlay network, loadbalancing mode options are `vip` and `dnsrr`

Networks and Load Balancing

```
# Publishing Modes

# Host Mode
$ docker service create --publish mode=host,target=80,published=8080,protocol=tcp --name proxy nginx

# Ingress Mode (Default)
$ docker service create --publish mode=ingress,target=80,published=8080,protocol=tcp --name proxy nginx

# Loadbalancing Modes (with overlay networks only)

# Virtual IP (VIP)
$ docker service create --endpoint-mode=vip --network=mynet --name proxy nginx

# DNS Round Robin
$ docker service create --endpoint-mode=dnsrr --network=mynet --name proxy nginx
```

Networks and Load Balancing with Compose

- When using Compose v3, you need to define networks under the networks section.
- You can refer to pre-created external networks
- If no network is provided, a new overlay one will be created for the stack and all services will be attached to it.

```
version: '3.1'
services:
  web:
    image: webimage
    deploy:
      mode: replicated
      replicas: 2
    networks:
      - mynet
      - ucp-hrm

  db:
    image: dbimage
    deploy:
      mode: replicated
      replicas: 2
    networks:
      - mynet
networks:
  mynet:
    driver: overlay
    ipam:
      driver: default
      config:
        -
          subnet: 10.200.10.0/24
  ucp-hrm:
    external: true
```

Secrets

- Need to pre-create the secret from STDIN or file before you can use it
- In Compose, you can create secrets from local files and assign them to services
- In UCP, secret and service labels must match to attach a secret to a service

```
# Creating a Secret

# STDIN
$ echo "dockerdocker" | docker secret create mysecret -

# File
$ docker secret create mysecret cert.pem

# Attaching a secret to a service

$ docker service create --secret mysecret redis

# With Compose
services:
  redis:
    image: redis:latest
    deploy:
      replicas: 1
    secrets:
      - mysecret
      - myothersecret
  secrets:
    mysecret:
      file: ./cert.pem
    myothersecret:
      external: true
```

Configs

- Store non-sensitive data outside the image or running container
- Operate similar to secrets, except they're not encrypted and don't use a temp fs.
- Supported on Linux and Windows Swarm Services (not standalone containers)

```
# Creating a Config
$ docker config create app-nginx-v1 app-nginx-v1.conf
kwasrrry30ccn67ln7y5a4lti

# Creating a Service with the config
$ docker service create --name nginx \
--config source=app-nginx-v1,target=/etc/nginx/conf.d/nginx.conf \
nginx:latest

# Creating a new version of the config
$ docker config create app-nginx-v2 app-nginx-v2.conf

# Updating the service with the new config

$ docker service update \
--config-rm app-nginx-v1 \
--config-add source=app-nginx-v2,target=/etc/nginx/conf.d/nginx.conf \
nginx
```

Rolling Updates and Automated Rollbacks

- Rolling updates rules are applied when updating one or more service attribute (image, network, published port..etc)
- Rollback rules are applied when a rolling update fails

```
# Rolling Update

$ docker service create \
  --replicas 6 \
  --name pets \
  --update-delay 10s \
  --update-parallelism 2 \
  --update-max-failure-ratio 0.25 \
  --update-failure-action pause \
  --update-monitor 10s \
  --update-order start-first \
  chrch/paas:v1

# Automated Rollbacks

$ docker service update \
  --rollback-delay 10s \
  --rollback-failure-action pause \
  --rollback-max-failure-ratio 0.25 \
  --rollback-monitor 10s \
  --rollback-order stop-first \
  --rollback-parallelism 0 \
  --image chrch/paas:v2 \
  pets
```

Labels and Deployment Constraints

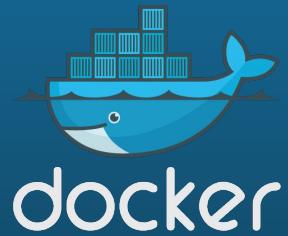
- Labels are key/value identifiers for cluster resources (services, containers, networks, volumes, nodes, secrets)
- Use labels to apply RBAC in UCP
- Use labels as scheduling constraints

```
# Service Labels
$ docker service create \
--label foo=bar \
--label bar=foo \
--label com.docker.ucp.access.label=production \
--name proxy \
nginx

# Node Labels
$ docker node update --label-add com.docker.ucp.access.label=production worker-01

# Service Constraints
$ docker service create \
--constraint 'node.labels.com.docker.ucp.access.label == production' \
--constraint 'node.role == manager' \
--name proxy \
nginx
```

MTA Application Deployment Lab



Lab 3 Summary

- ❑ Configured an application loadbalancer
- ❑ Deployed your MTA application using Stacks and Compose v3
- ❑ Exposed the Application with HRM
- ❑ Configured Rolling Updates
- ❑ Scaled the application

Thank you!

