

# DOCKER FOR ENTERPRISE OPERATIONS



# HOW WE TEACH

- Docker believes in learning by doing.
- The course is lab driven:
  - Demo Exercises
  - Signature Assignments
- Work together!
- Ask questions at any time



# SESSION LOGISTICS

- 2 days duration
- mostly exercises
- regular breaks



# ASSUMED KNOWLEDGE AND REQUIREMENTS

- Familiarity with using the Linux command line
- Linux Cheat Sheet: <http://bit.ly/2mTQr8I>
- A UCP License (download one at <https://store.docker.com/bundles/docker-datacenter/purchase?plan=free-trial>)
- You should know the basics of Docker and Kubernetes
  - Run a Docker container
  - Set up a service on a Swarm
  - Deploy an app on Kubernetes
  - Search for and pull official images from Docker Store, community images from Docker Hub



# YOUR LAB ENVIRONMENT

- You have been given several instances for use in exercises.
- Ask instructor for access credentials if you don't have them already.



# COURSE LEARNING OBJECTIVES

By the end of this course, learners will be able to:

- Identify the key features of UCP and DTR
- Build a complete, basic software supply chain using UCP and DTR that includes CI/CD, content trust, and image scanning
- Describe the methodological management differences between containers and virtual machines





# INTRODUCTION TO DOCKER ENTERPRISE EDITION



# A CONTAINERIZED MINDSET

- Expect churn!
- Talk to your developers
- Abstract your datacenter
- Automate everything.



VMs



Containers

Dog photo [jeffreyw](#); Cattle photo [Donald Judge](#); images CC-BY 2.0



# THE SOFTWARE SUPPLY CHAIN

- Image Creation
- Image Distribution
- Container Execution

Docker EE enables security, ease of use, and enterprise-grade control at each of these steps.



# KEY EE FEATURES

- Build:
  - Security Scanning
  - Repository Automation
- Ship:
  - Content Trust
  - Content Cache
- Run:
  - Role based access control
  - Universal Control Plane (GUI)

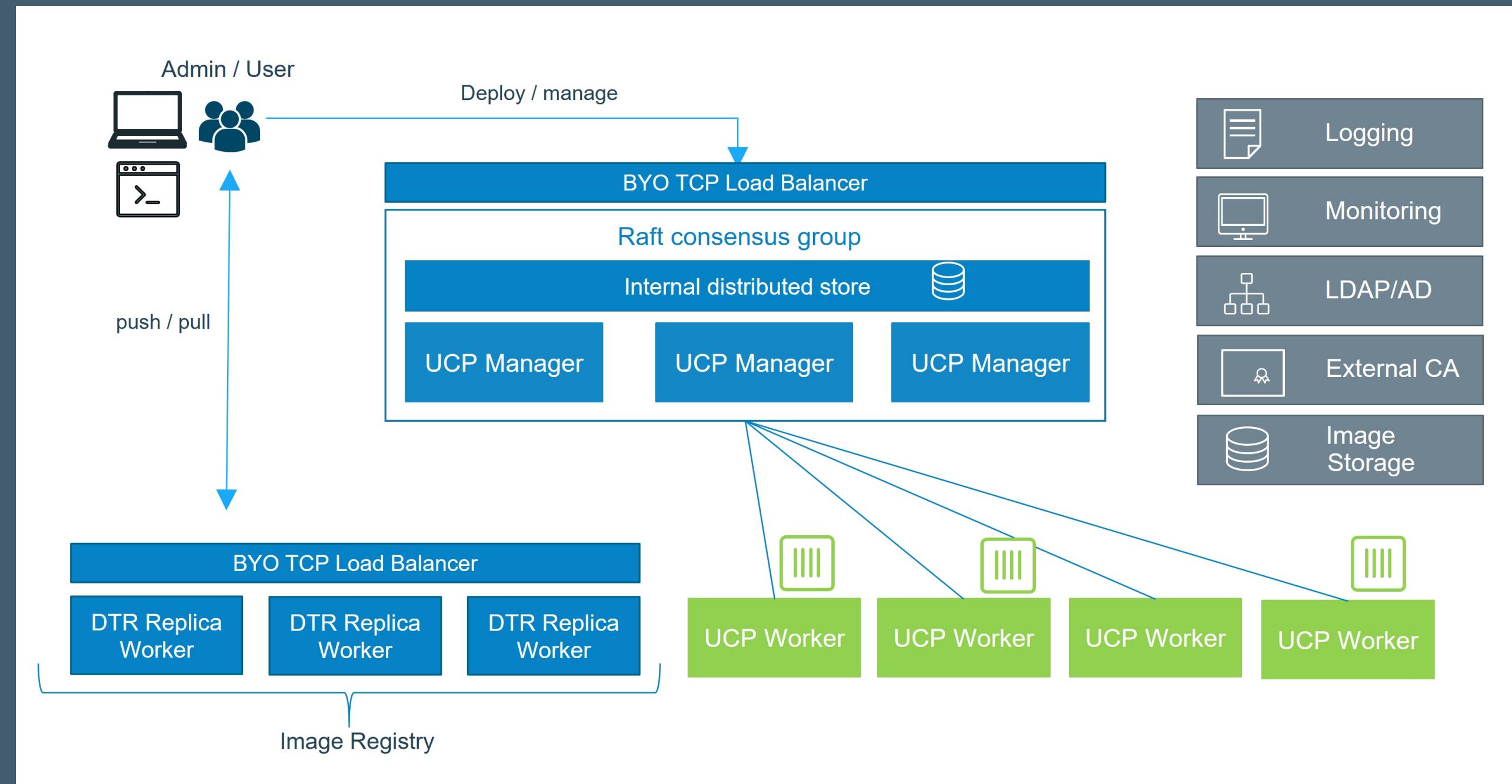


# INTEGRATION: BATTERIES INCLUDED BUT SWAPPABLE

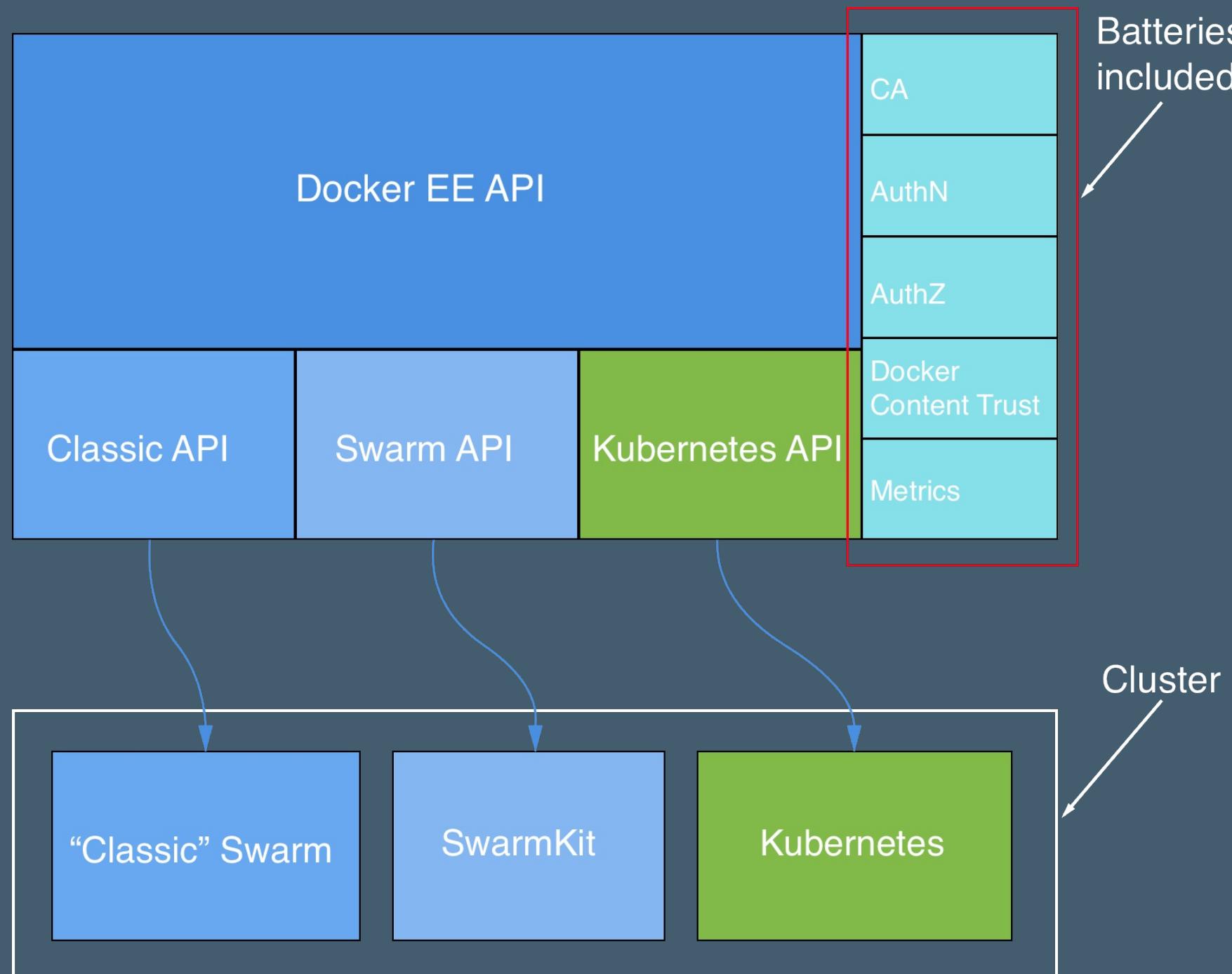
- Certificate authorities
- Network drivers
- Storage backends
- User management
- Monitoring
- ...



# DOCKER EE ARCHITECTURE



# CHOOSE YOUR ORCHESTRATOR





## EXERCISE: CONFIGURE UCP

Work through the 'Install UCP' exercise in the Docker for Enterprise Operations Exercises book.



# FURTHER READING

- About Docker EE <http://dockr.ly/2oq6bPY>





# UNIVERSAL CONTROL PLANE



# LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Describe the interactions and responsibilities of the containers that serve UCP
- Identify the necessary firewall configurations to support inter-node communication
- Create Docker resources by issuing API calls and configuring client bundles

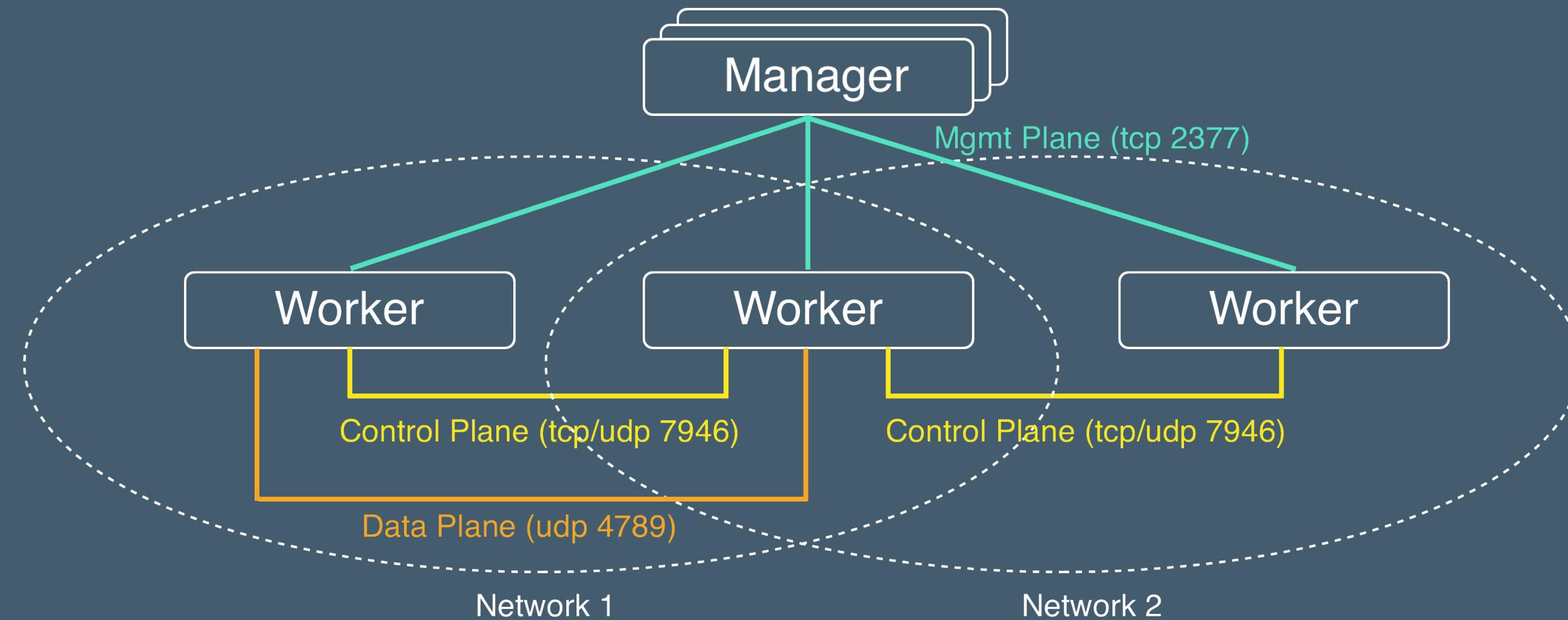


# WHAT IS UCP?

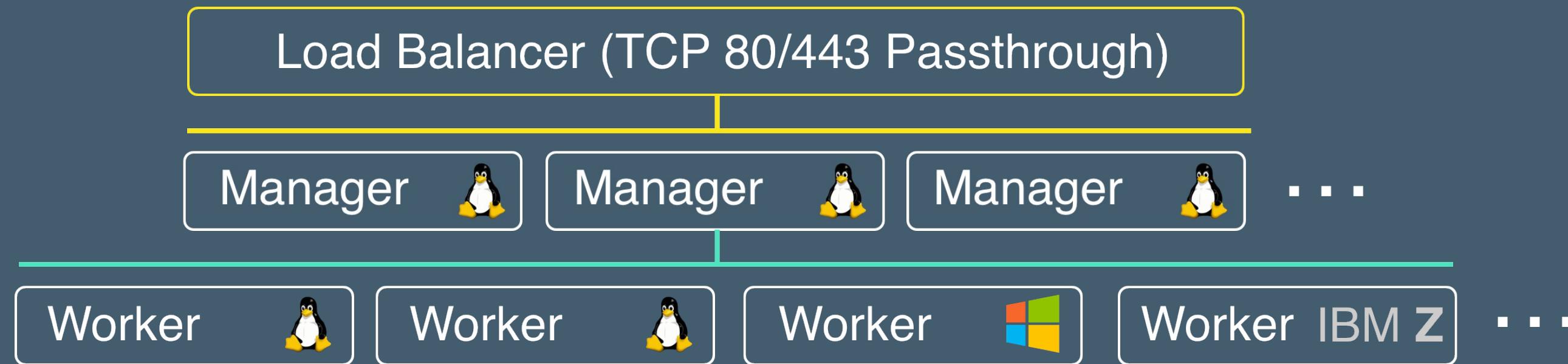
- Containerized app running on a Swarm
- Supports Swarm and Kubernetes orchestration
- Adds:
  - Role based access control
  - Secure remote API and CLI access
  - L7 routing
  - Web frontend



# SWARM MODE ARCHITECTURE



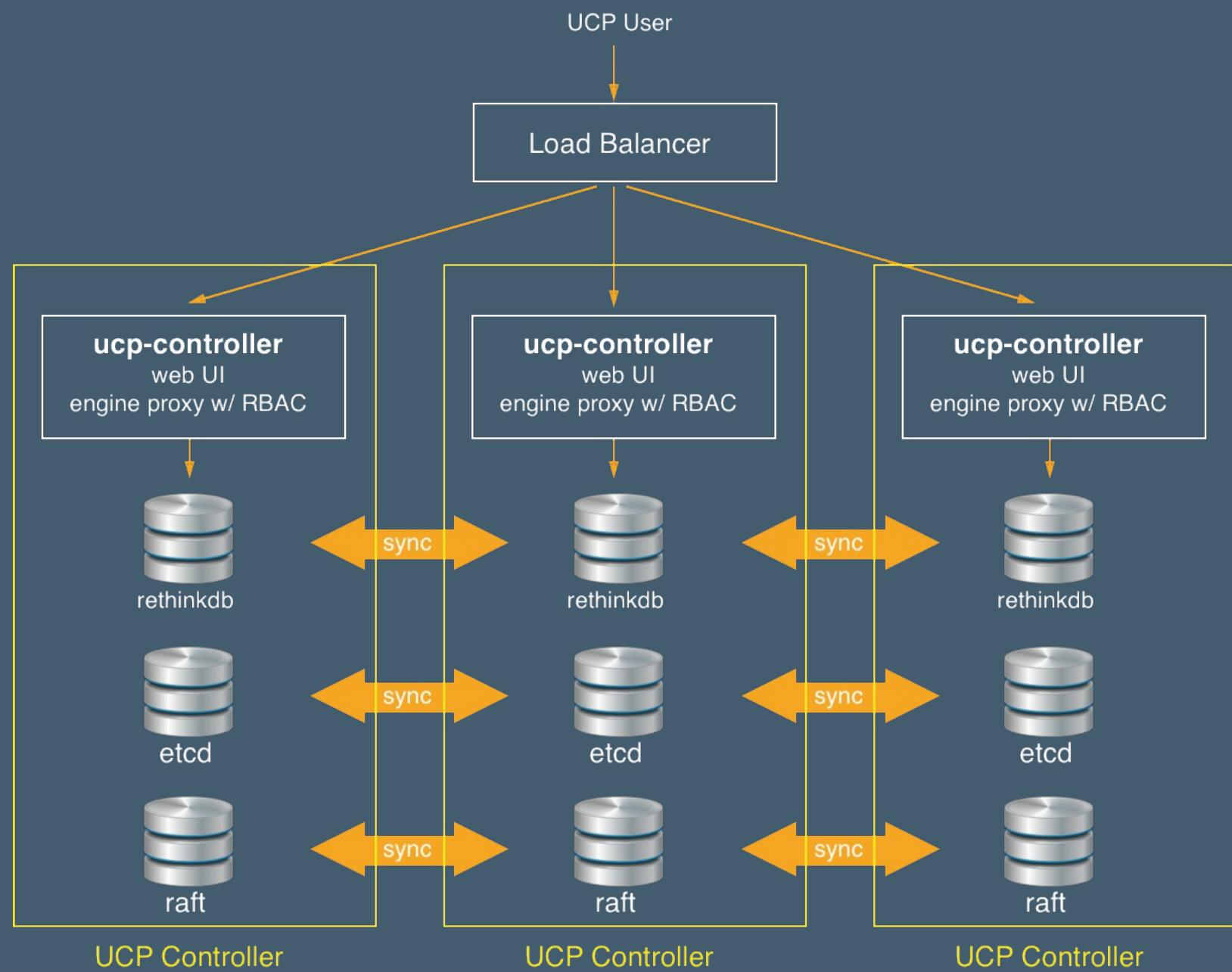
# CLUSTER CONFIGURATION



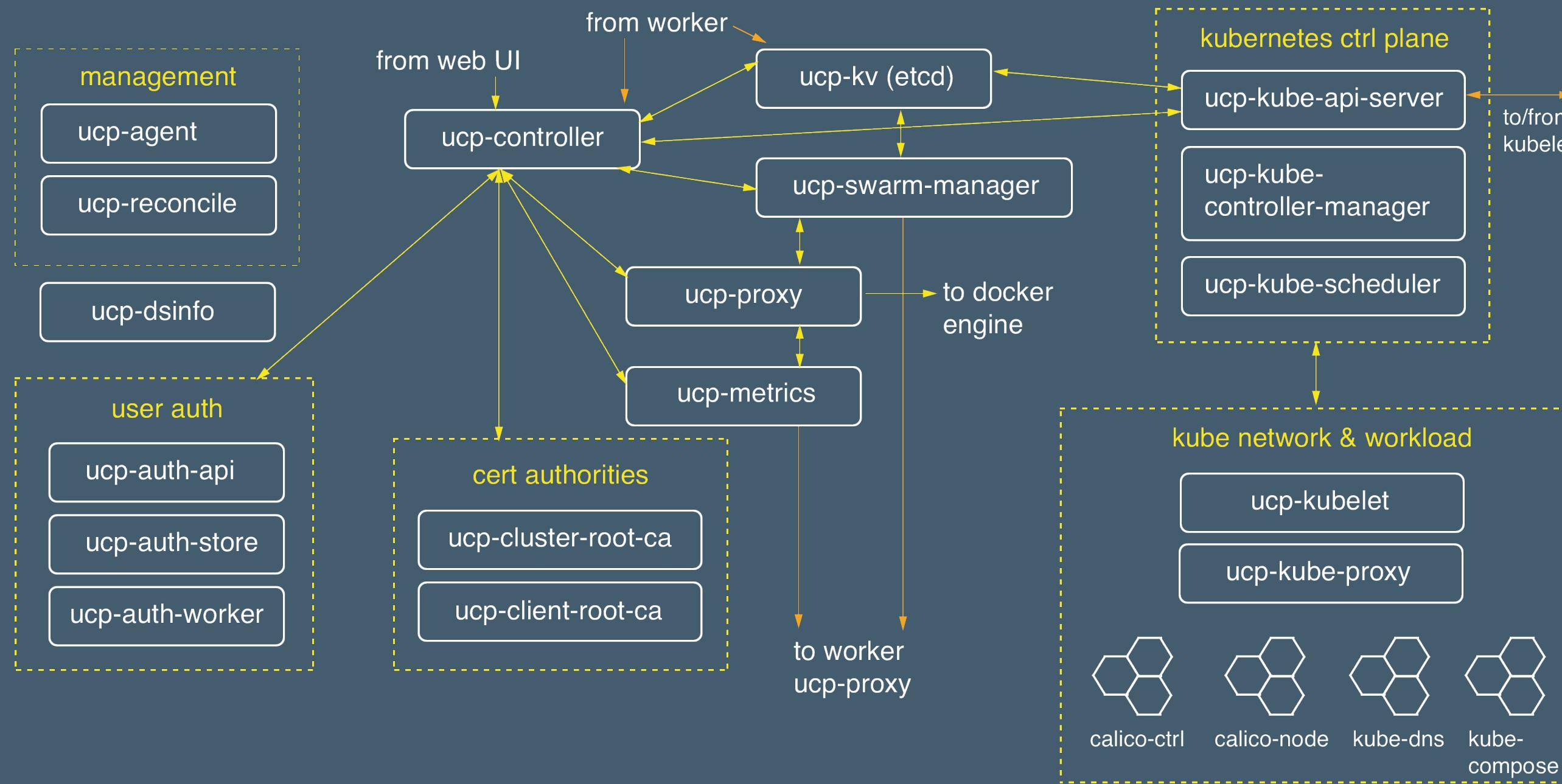
- Odd no. of managers
- Don't run workload on managers
- Don't terminate HTTPS in manager LB
- See `/_ping` for manager health
- See system requirements at <http://dockr.ly/2DnYFPa>



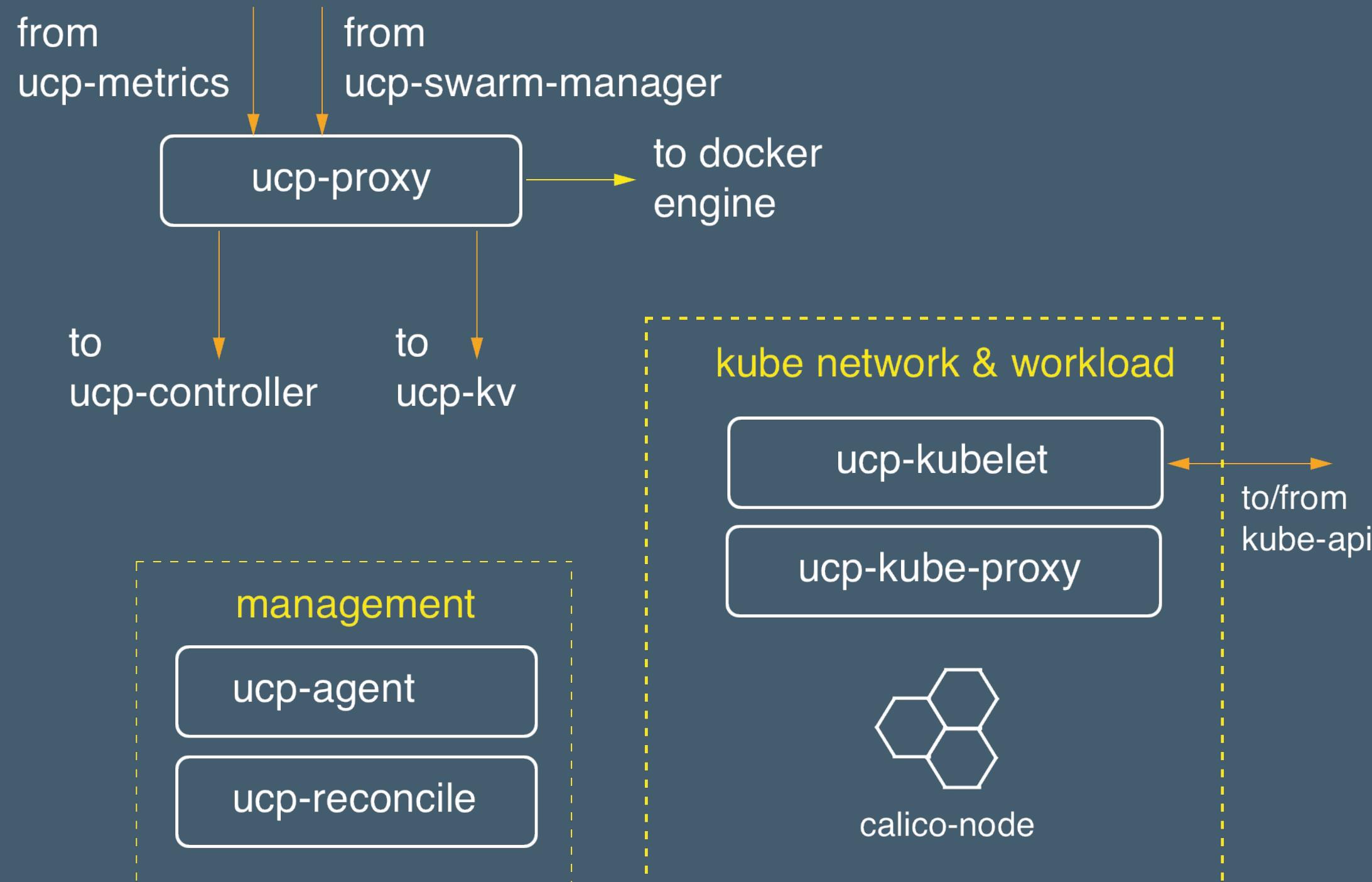
# CLUSTER SYNCHRONIZATION



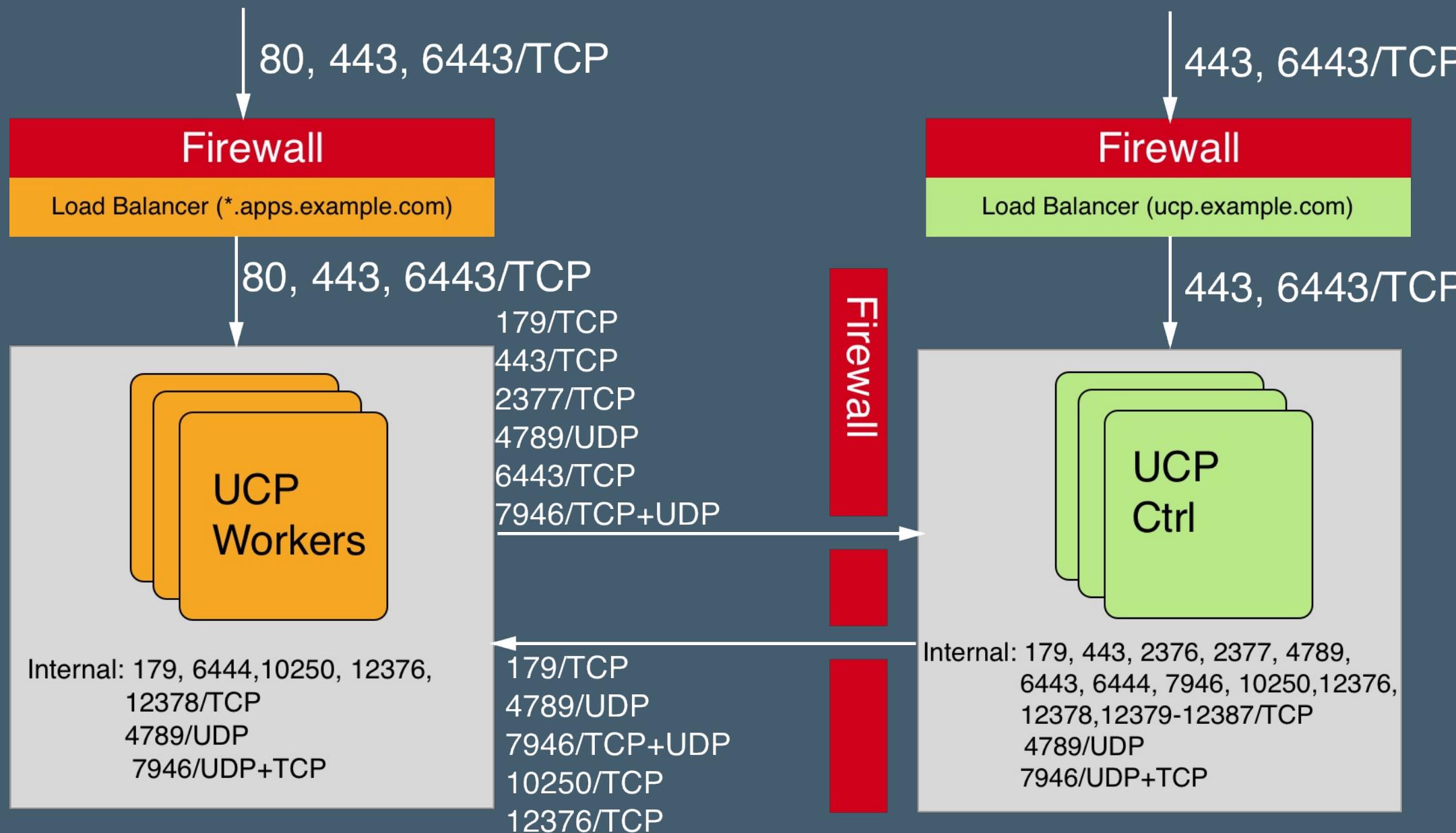
# UCP MANAGER ARCHITECTURE OVERVIEW



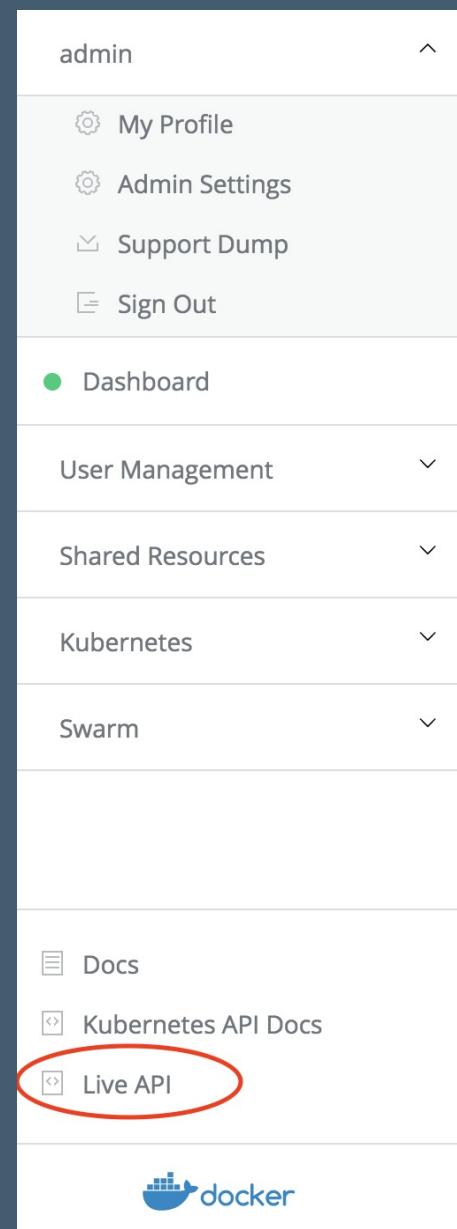
# UCP WORKER ARCHITECTURE OVERVIEW



# NETWORK TOPOLOGY SUMMARY



# UCP API



- Automate UCP via API
- Also used by Web UI
- Docs: <https://<UCP FQDN>/apidocs> and <http://dockr.ly/2E0Hrp1>



# UCP CLIENT BUNDLES

- Control remote UCP through local Docker and Kubernetes CLI
- Secured with TLS and RBAC
- Certs available in UCP:
  - Web: username -> My Profile -> Client Bundles
  - API endpoint:

`/api/clientbundle`





## EXERCISE: UCP HIGH AVAILABILITY

Work through:

- Adding UCP Manager Nodes
- UCP API & Client Bundles

exercises in the Docker for Enterprise Operations Exercises book.



# DISCUSSION

- How many managers can you lose and still be able to schedule services, and how many can you lose and still be able to recover your cluster?
- Questions?



# FURTHER READING

- Intro to UCP: <https://dockr.ly/2K4aXNi>
- Docker Reference Architecture: Service Discovery and Load Balancing with UCP: <http://dockr.ly/2gGTP6c>
- Docker Reference Architecture: UCP 2.0 Service Discovery and Load Balancing <http://dockr.ly/2rbxDDX>
- High Availability Architecture and Apps with DDC: <http://dockr.ly/1sqPrIH>
- Docker Reference Architecture: Running Docker Enterprise Edition at Scale: <http://dockr.ly/2DW9R3n>





# USER MANAGEMENT & ACCESS CONTROL



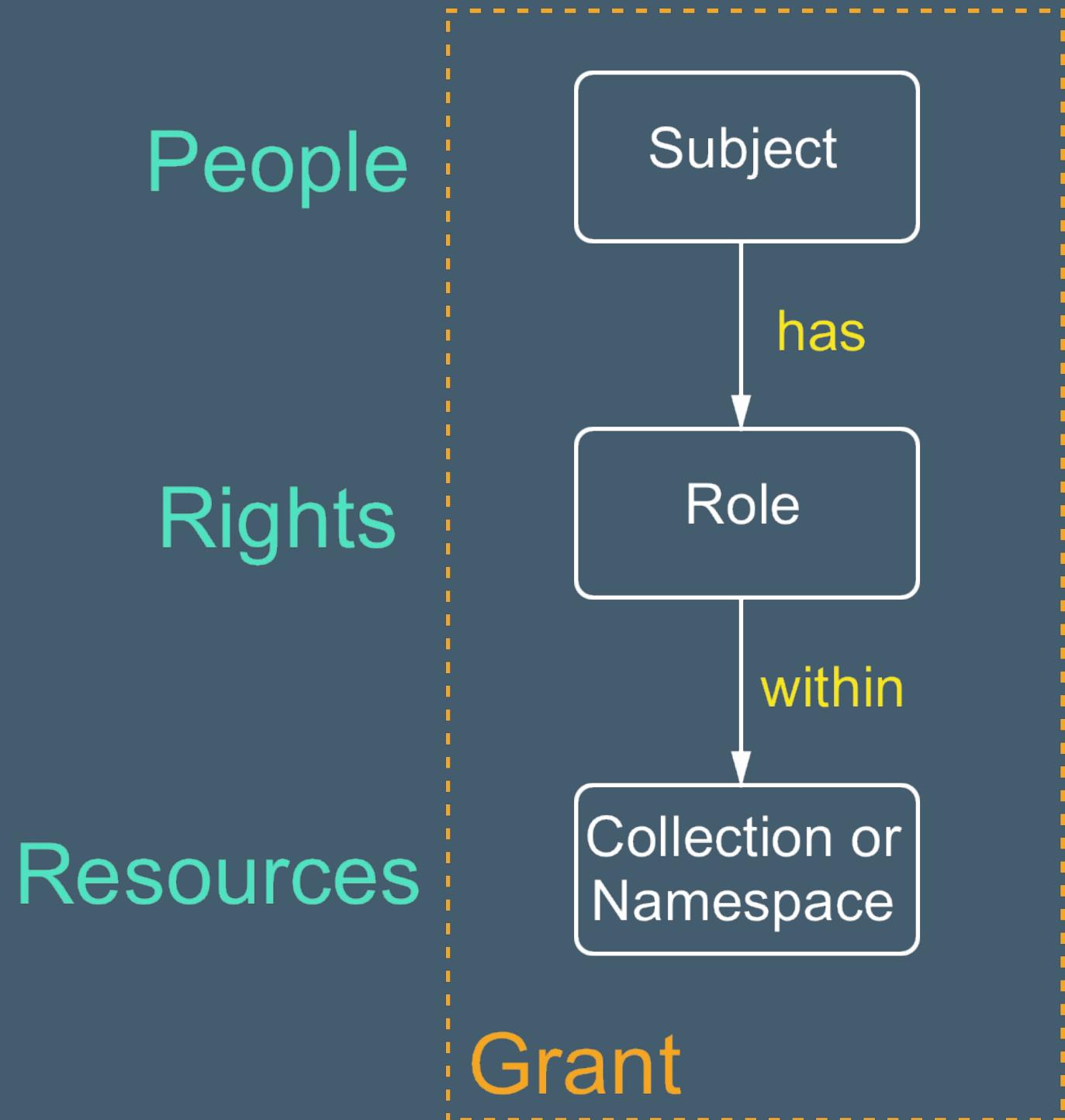
# LEARNING OBJECTIVES

By the end of this module, learners will be able to:

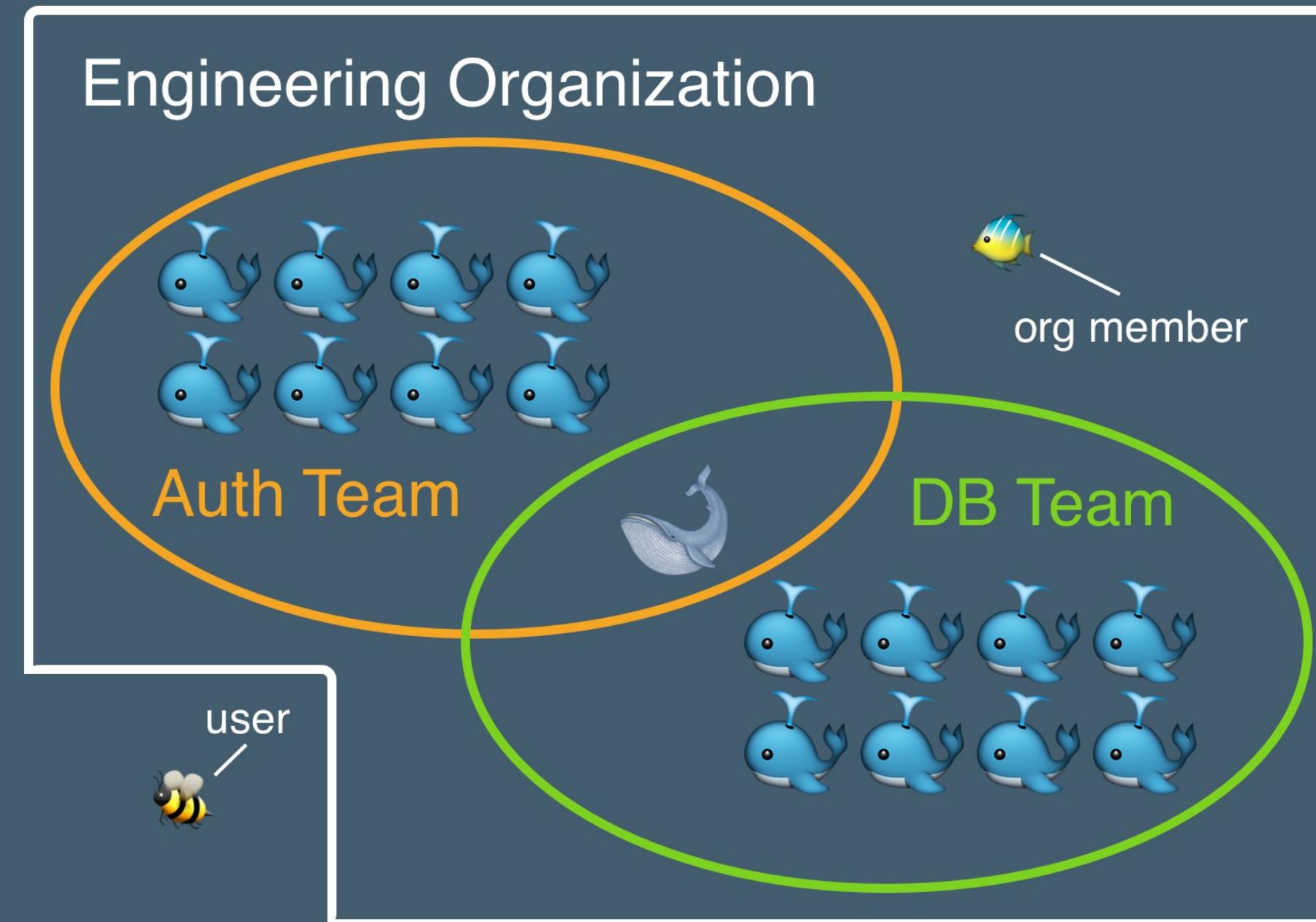
- Define resource collections, users, team, organizations, roles and grants to confer controlled access to Docker resources for specific users
- Manage Kubernetes namespaces and administer access control rights to the same



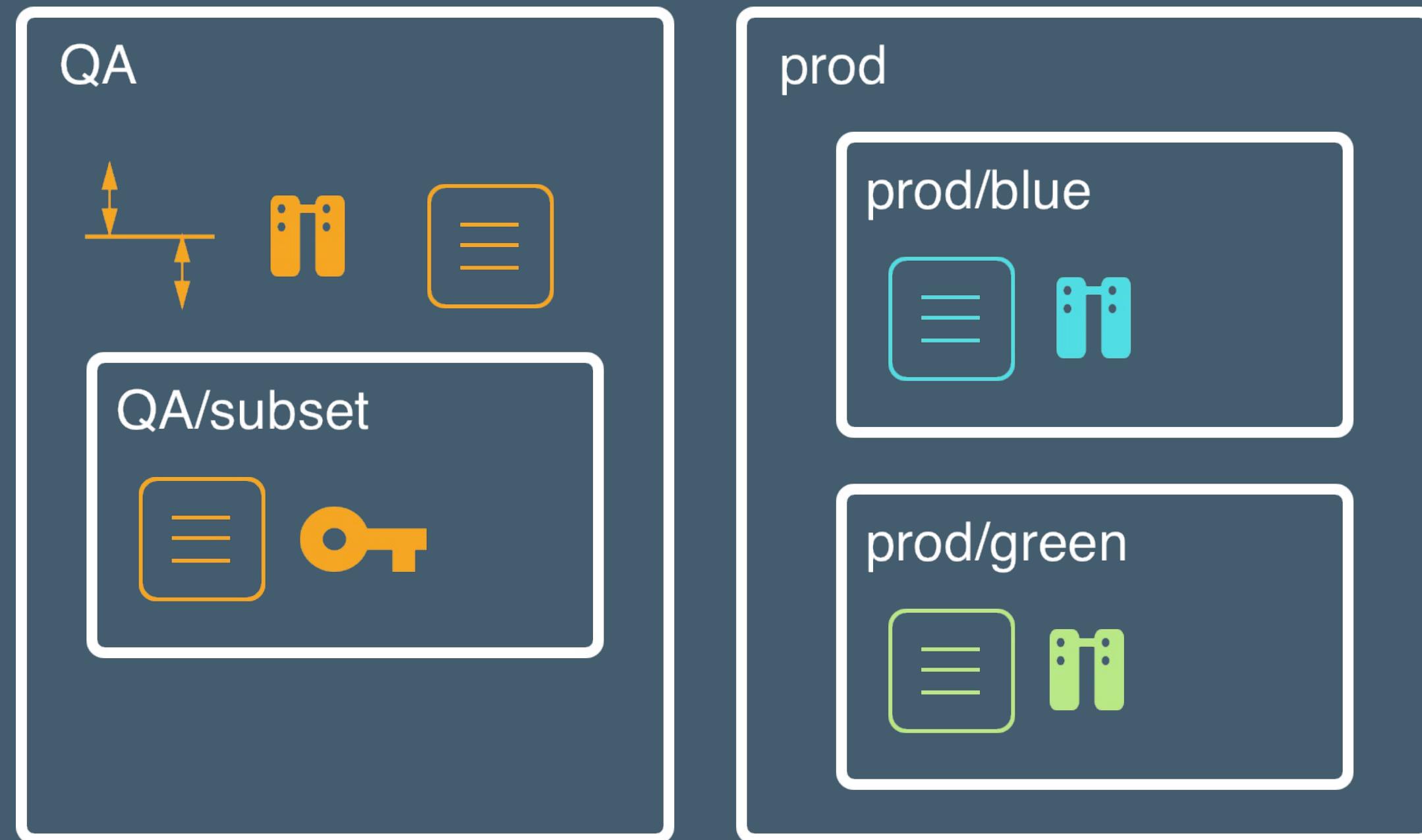
# UCP ROLE BASED ACCESS CONTROL



# DDC ORG CHARTS



# RESOURCE COLLECTIONS



# PRE-MADE COLLECTIONS

- **/System**: UCP managers, DTR nodes, system services
- **/Shared**: worker nodes
- **/Shared/Private/<username>**: user-specific collections (JIT-provisioned on first login)



# DEFAULT COLLECTIONS

- User can have a Default Collection defined
- Default location for new objects created by this user
- Initialized as **Collections/Shared/Private/<username>**

The screenshot shows the Docker EE UI interface. On the left, there's a sidebar with navigation links: Dashboard, User Management (with sub-links for Organizations & Teams, Users, Roles, and Grants), and Actions (with Create User button). The main area displays a table of users:

Status	User Name	Full Name
Active	admin	
Active	alovelace	alovelace
Active	ghopper	ghopper

To the right, a modal window is open for the user 'ghopper'. It shows the user's details: Name (ghopper), Full Name (ghopper), and Admin status (No). Under 'Client Bundle', the 'Default Collection' option is highlighted with a red oval. Other options include Details and Security.



# KUBERNETES NAMESPACES

Auto-generated

default

- default namespace for user content

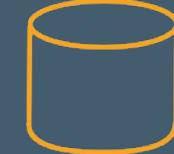
kube-system

- calico
- kube-dns
- compose adapter

kube-public

User-defined

development



prod-east



prod-west



# ROLES

- Bundle of permissions
- Connect subjects to resource collections
- Conflicts resolved by most-privilege



# DEFAULT ROLES

## FULL CONTROL:

- Exec
- Namespaces
- Custom Kernel Capabilities
- Host Bind Mounts
- Privileged Mode

## SCHEDULER

- Node Schedule
- Node View

 **Node permissions**

 **No Node Permissions**

## RESTRICTED CONTROL:

- Create
- Run
- Restart
- Stop
- Delete

 **No Node Permissions**

## VIEW ONLY:

- Inspect
- View

 **No Node Permissions**

**NONE**

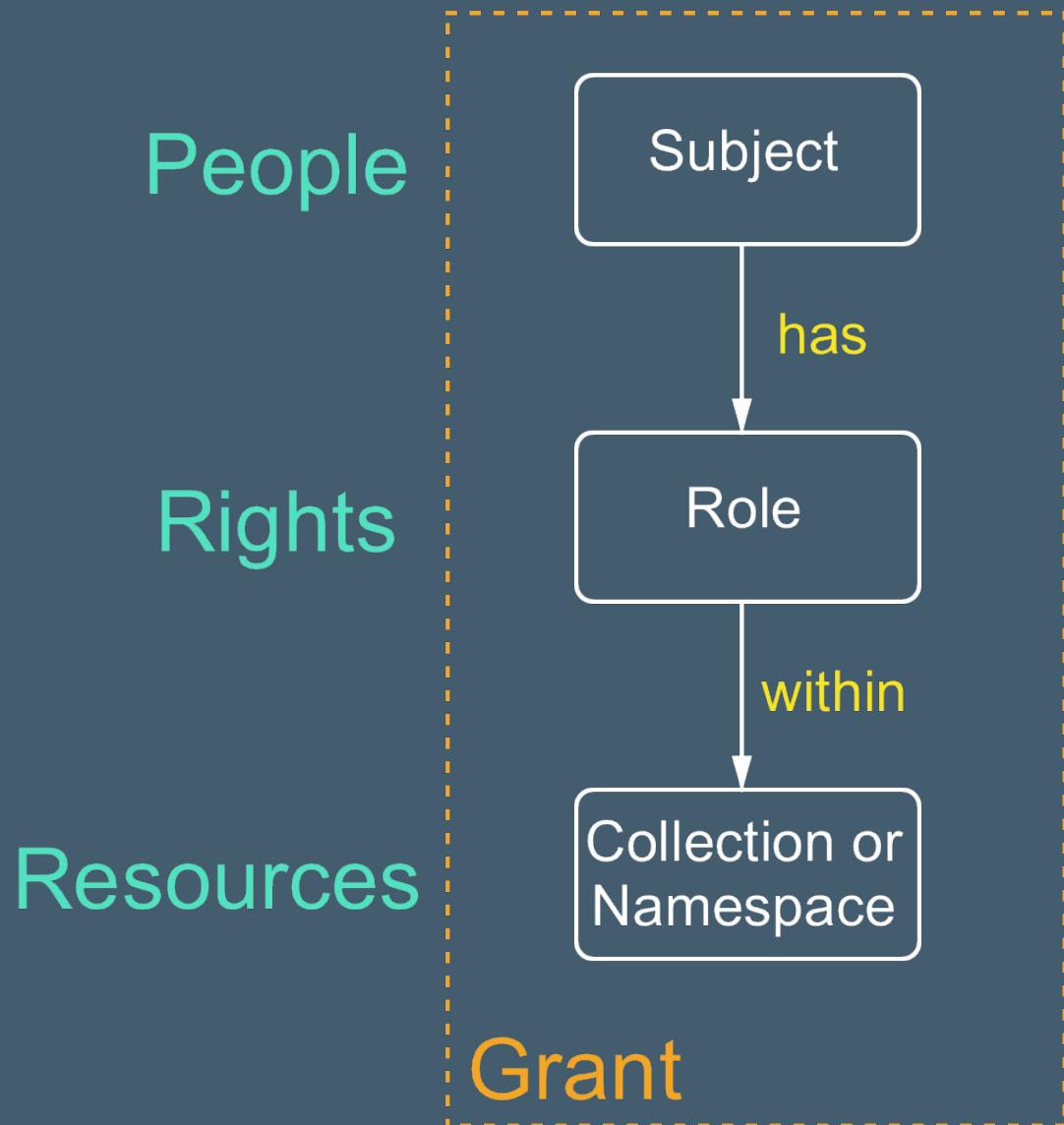


# CUSTOM ROLES

- Admins can create new custom roles
- Once created, roles are **immutable**



# PERMISSION GRANTS



A screenshot of a web-based interface for managing permission grants. The sidebar shows navigation links: Dashboard, User Management (expanded), Organizations & Teams, Users, Roles, Grants (selected), Shared Resources, Kubernetes, and Swarm. The main area displays a table of grants:

Subject	Role	Set of Resources
Org - docker-datacenter	Scheduler	/
admin	Restricted Control	/Shared/Private/admin
admin	Full Control	kubernetesnamespaces
admin	Scheduler	/Shared
admin	Full Control	/
ghopper	Restricted Control	/Shared/Production
alovelace	Full Control	/Shared/Development

Only admins can define new grants.



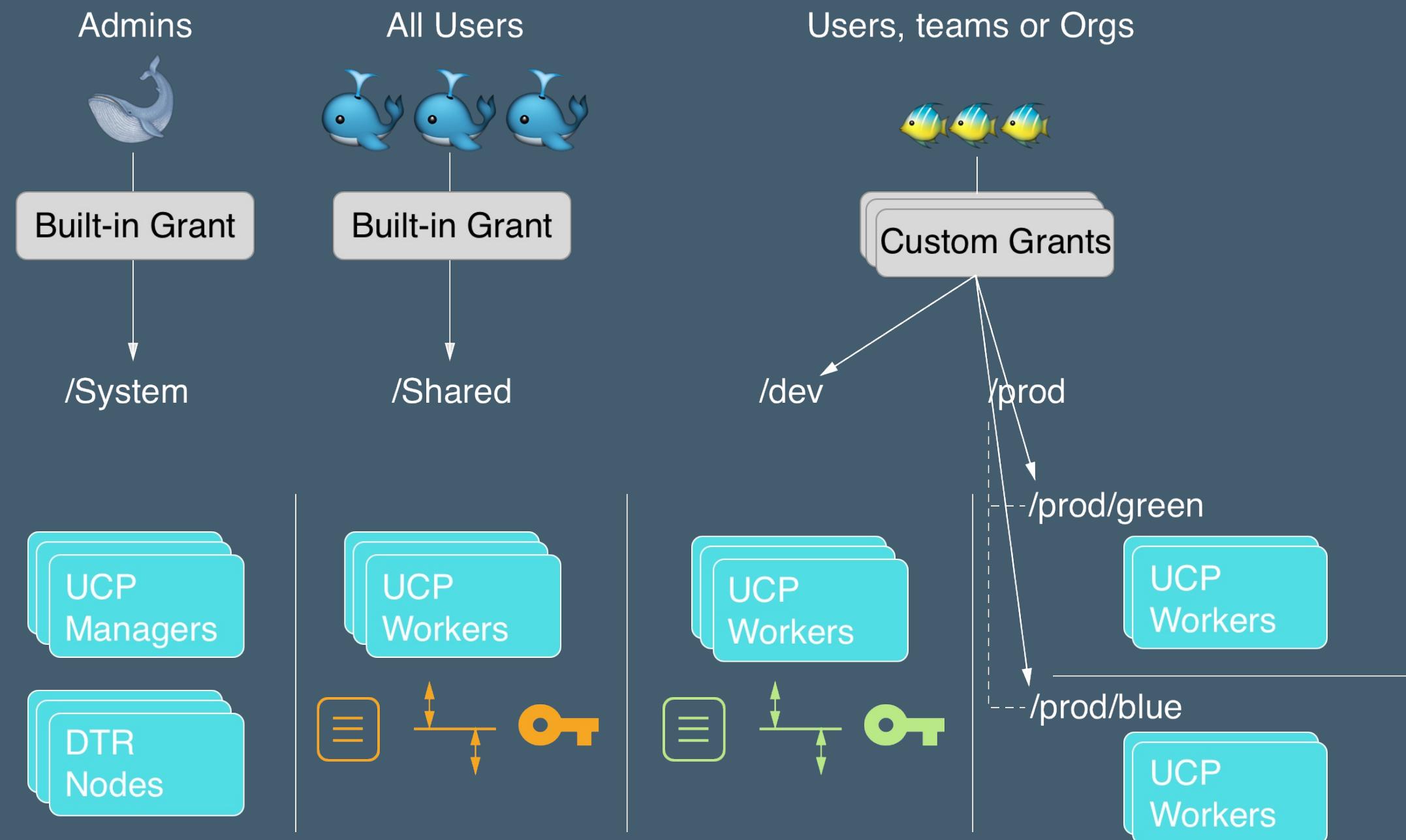
# NODE RBAC: STANDARD TIER

- All worker nodes part of `/Shared` collection
- All users have `Scheduler` grant to `/Shared` collection

`/shared`



# NODE RBAC: ADVANCED TIER



# NODE RBAC: KUBERNETES

The screenshot shows the Rancher UI interface. On the left, there's a sidebar with navigation items like Dashboard, User Management, Shared Resources, Kubernetes, Create, Namespaces (which is selected), and Controllers. The main area shows '4 Namespaces' and a 'Nodes Collection' table with columns for Name and Nodes Collection. A context menu is open over a row named 'development', with options like Configure, Actions (Set Context, Link Nodes in Collection, Remove), and METADATA (Name: development, Creation Timestamp: 2018-01-04T12:51:12Z).

This is a modal dialog titled 'Add Nodes From A Collection'. It has a 'Choose Collection' section with three options: Private, Production, and Development. The 'Development' option is selected, indicated by a 'Selected' button. Below this, there's a 'View Nodes in selected Collection' table with columns for Name, Role, Address, and OS/Arch. One row is visible: ucp-node-0, worker, 10.10.14.221, and linux/x86\_64.

Name	Role	Address	OS/Arch
ucp-node-0	worker	10.10.14.221	linux/x86_64



# LDAP INTEGRATION

- Delegate authentication to an LDAP server
- User accounts will be synced from LDAP based on an LDAP search configuration
- Default just-in-time provisioning
- Must define mapping between LDAP groups & UCP teams





## EXERCISE: USER MANAGEMENT

Work through:

- Access Control in UCP
- User Management with LDAP
- Password Recovery

in the Docker for Enterprise Operations Exercises book.



# DISCUSSION

- What are some pros and cons of RBAC for worker nodes, compared to having separate clusters?
- Questions?



## FURTHER READING

- Create users and team manually: <https://dockr.ly/2HgZdJo>
- Create teams with LDAP: <https://dockr.ly/2HTAtnX>
- Integrate with an LDAP directory: <https://dockr.ly/2K3jC2z>
- Access control model: <https://dockr.ly/2HhTiUN>





# UCP ORCHESTRATION



ee2.0-v2.2 © 2018 Docker, Inc.

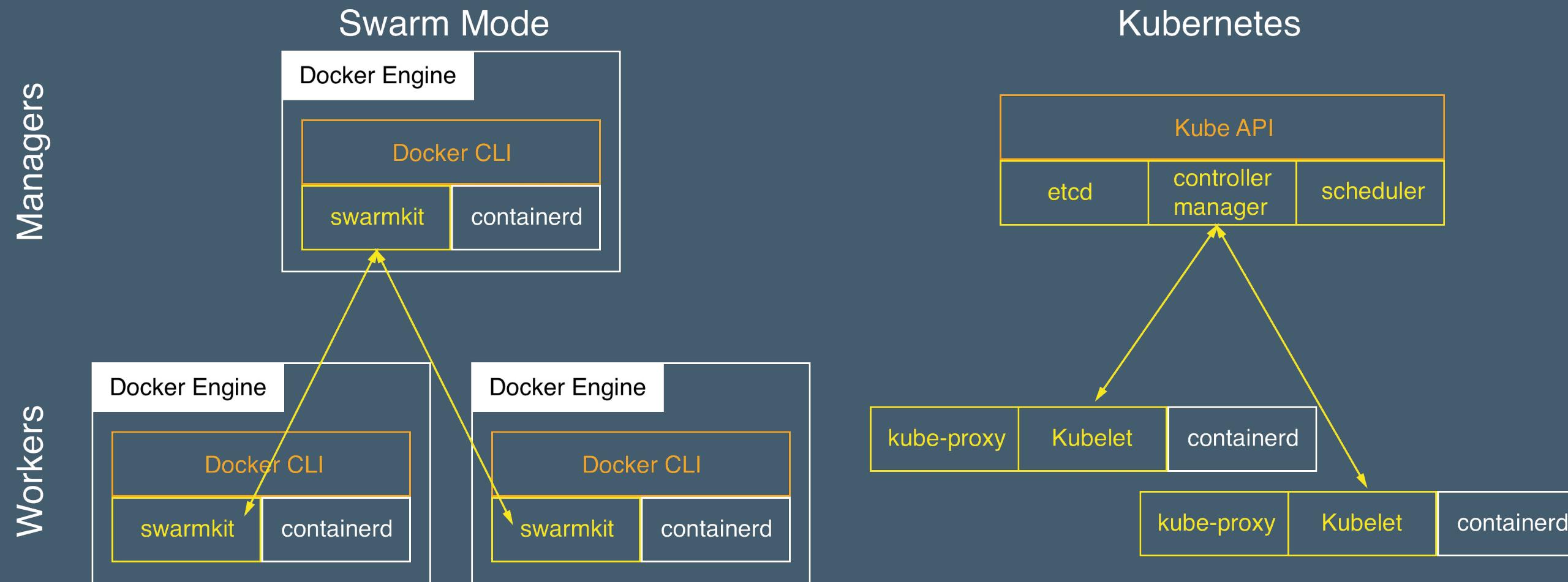
# LEARNING OBJECTIVES

By the end of this module, learners will be able to:

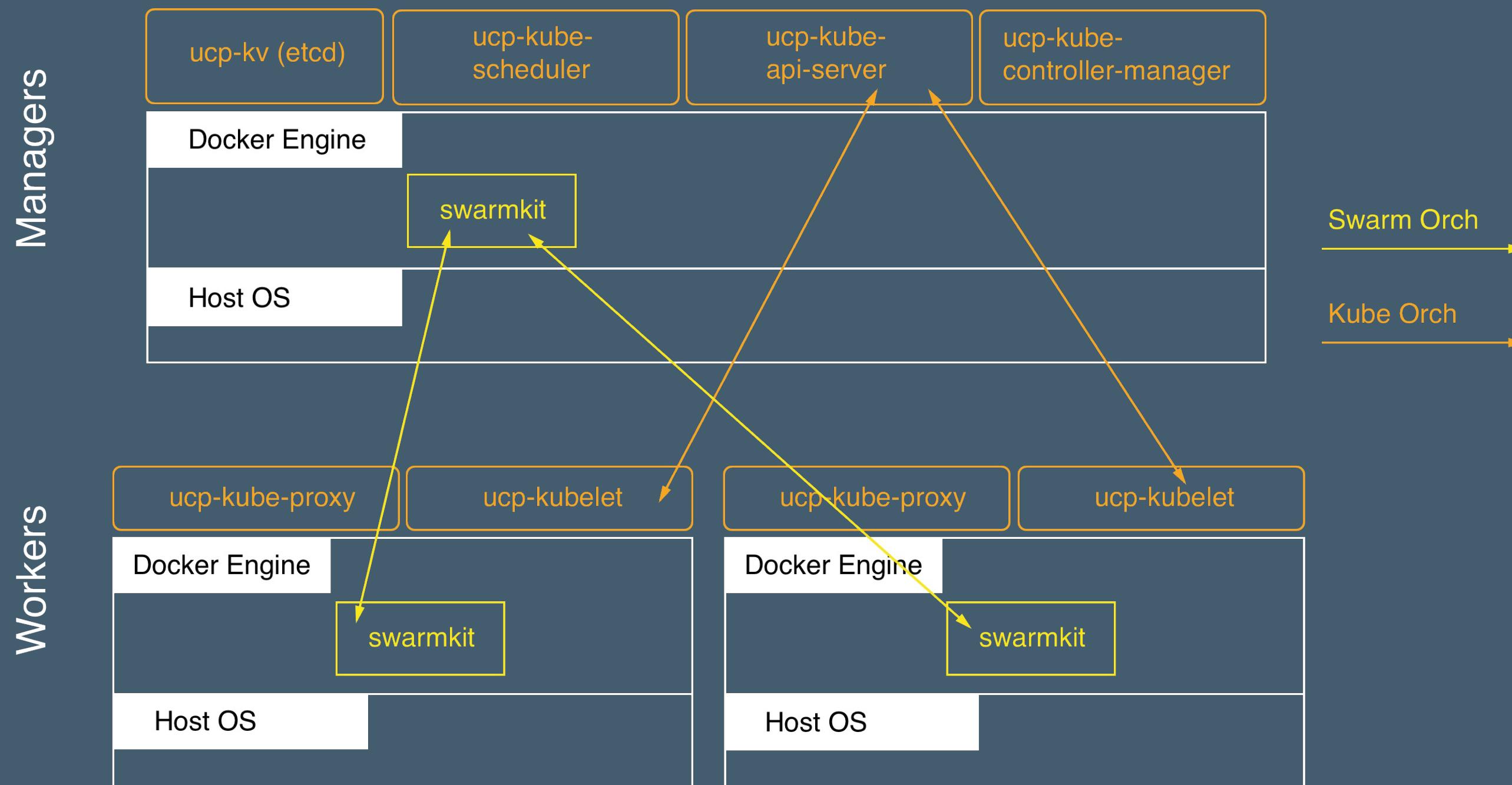
- Compare and contrast Swarm and Kubernetes orchestration components and networking models
- Use UCP to deploy apps on both orchestrators



# ORCHESTRATOR ARCHITECTURE



# ORCHESTRATORS IN UCP



# ADDITIONAL KUBERNETES COMPONENTS

- Kube DNS
- Calico
- Compose Adapter

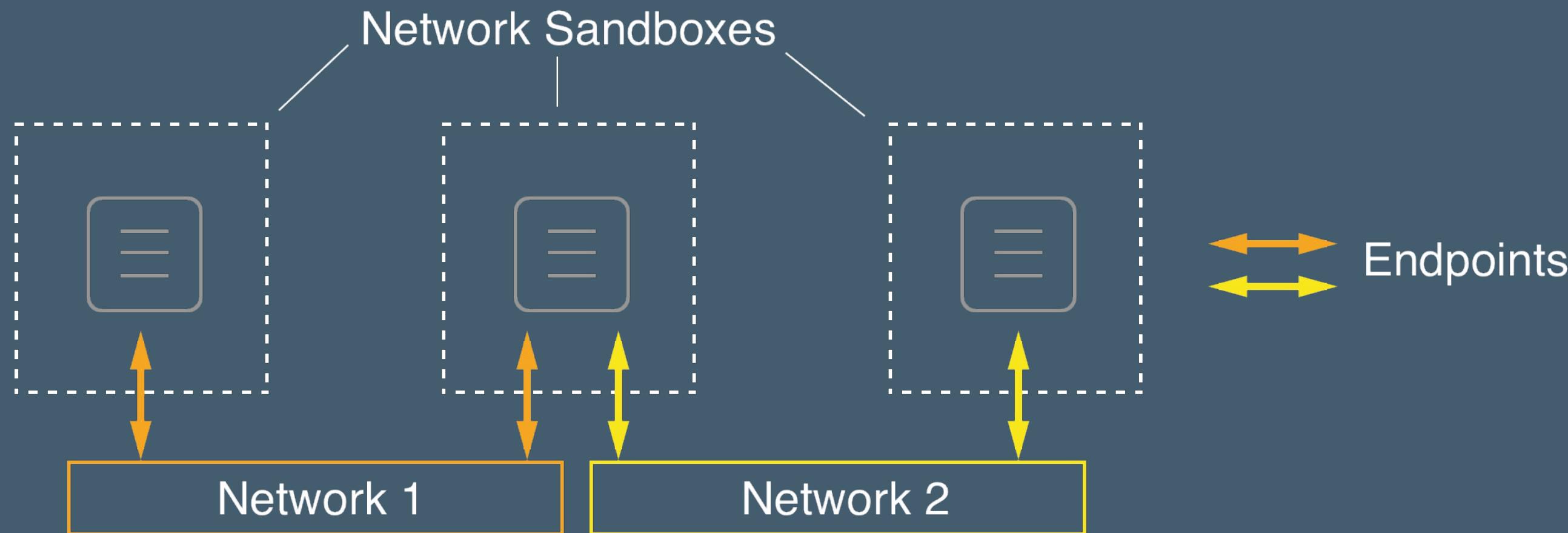


# NETWORKING MODELS

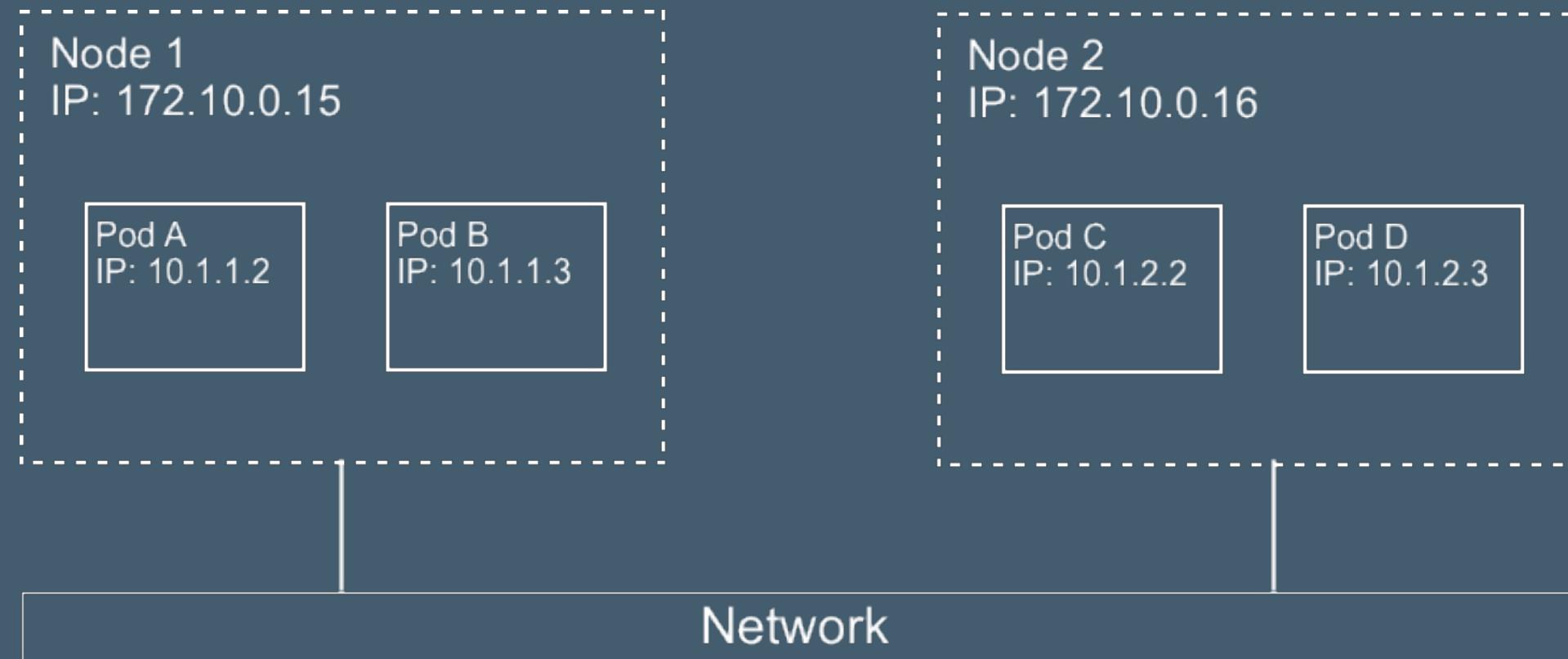
- Fundamental spec for how containers communicate
- Flexible and high level
- Standardization for how networks are built
- Distinct for Docker native vs. Kubernetes



# DOCKER'S CONTAINER NETWORK MODEL



# KUBERNETES NETWORK MODEL

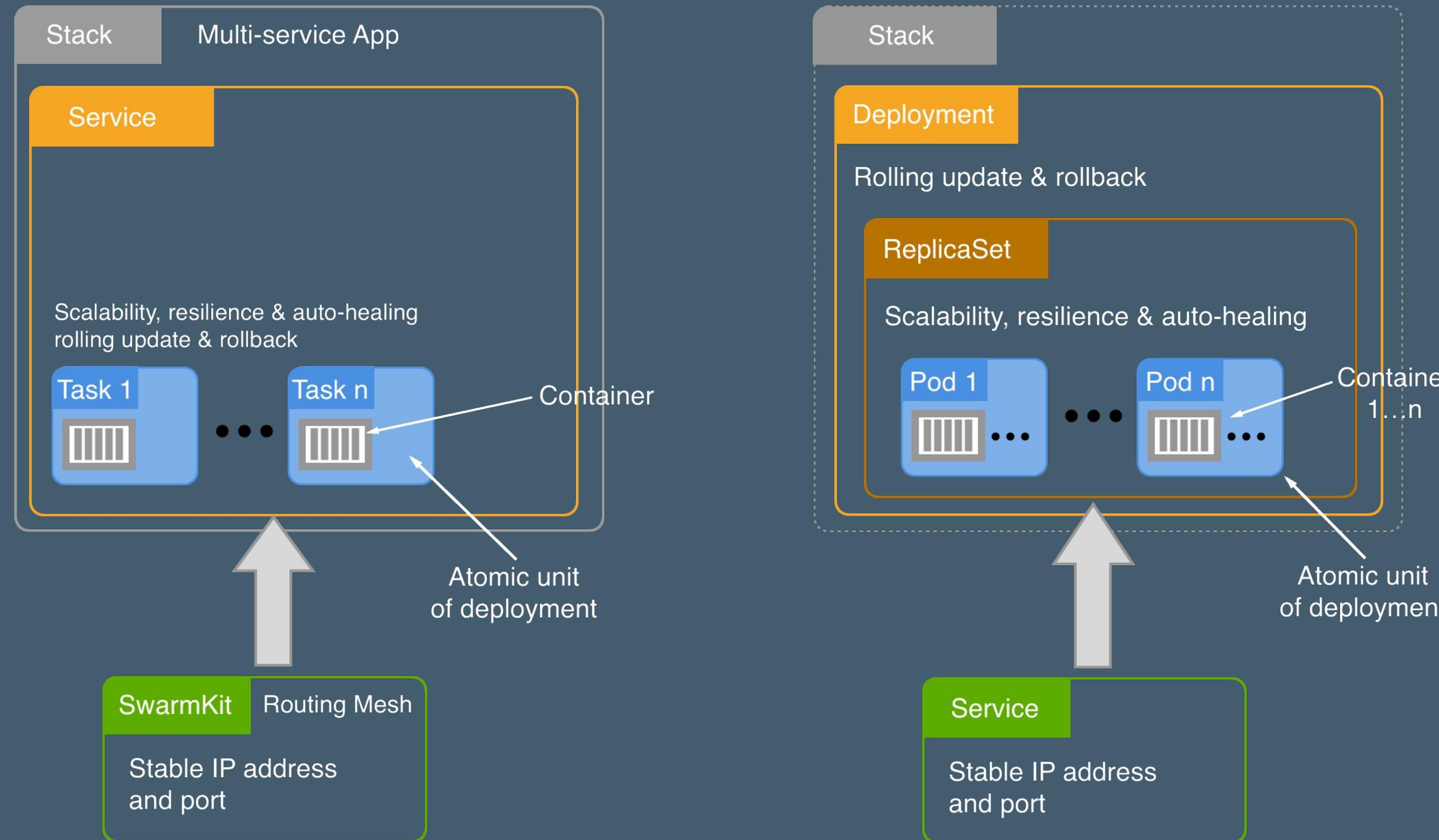


## Requirements

- Pod <--> Pod without NAT
- Node <--> Pod without NAT
- Pod's peers find it at the same IP it finds itself
- Creates a **flat network**, like VMs



# ORCHESTRATION COMPONENTS



# ORCHESTRATOR UI

The screenshot displays a user interface for managing containerized applications across multiple orchestrators. On the left, a sidebar lists categories: **Kubernetes**, **Namespaces** (selected), **default**, **Controllers**, **Load Balancers**, **Pods**, **Configurations**, and **Storage**. A central panel shows two sections: **Kubernetes** and **Swarm**.

- Kubernetes:** Contains a **Create** button and a **Namespaces** section with a **default** namespace selected.
- Swarm:** Contains sections for **Services** (0), **Volumes** (1), **Networks** (5), and **Secrets** (0).

- Create stacks, Kube yaml, individual
- List all Objects of the given Type





# EXERCISE: ORCHESTRATING APPLICATIONS IN UCP

Work through:

- Orchestrating Applications
- Combining Collections and Kubernetes Namespaces

in the Docker for Enterprise Operations Exercises book.



# ORCHESTRATION TAKEAWAYS

- UCP provides Kubernetes and Swarm for orchestration
- UCP Manager = Swarm Manager = Kubernetes Master
- If UCP is HA => Kubernetes is HA
- Workers: either Kubernetes or Swarm workload
- RBAC applies to both orchestrators
- Stacks implemented for Kubernetes



# FURTHER READING

- Docker & Kubernetes: <https://www.docker.com/kubernetes>
- Official Kubernetes Docs: <https://kubernetes.io/docs>
- Kubernetes tutorials: <http://bit.ly/K8-tutorials>
- Introduction to Kubernetes: <http://bit.ly/K8-intro>
- Understanding Kubernetes Networking: <http://bit.ly/2kdl1qQ>
- Kubernetes the Hard Way: <http://bit.ly/29Dq4wC>





# SERVICE MESH



# LEARNING OBJECTIVES

By the end of this module, learners will be able to:

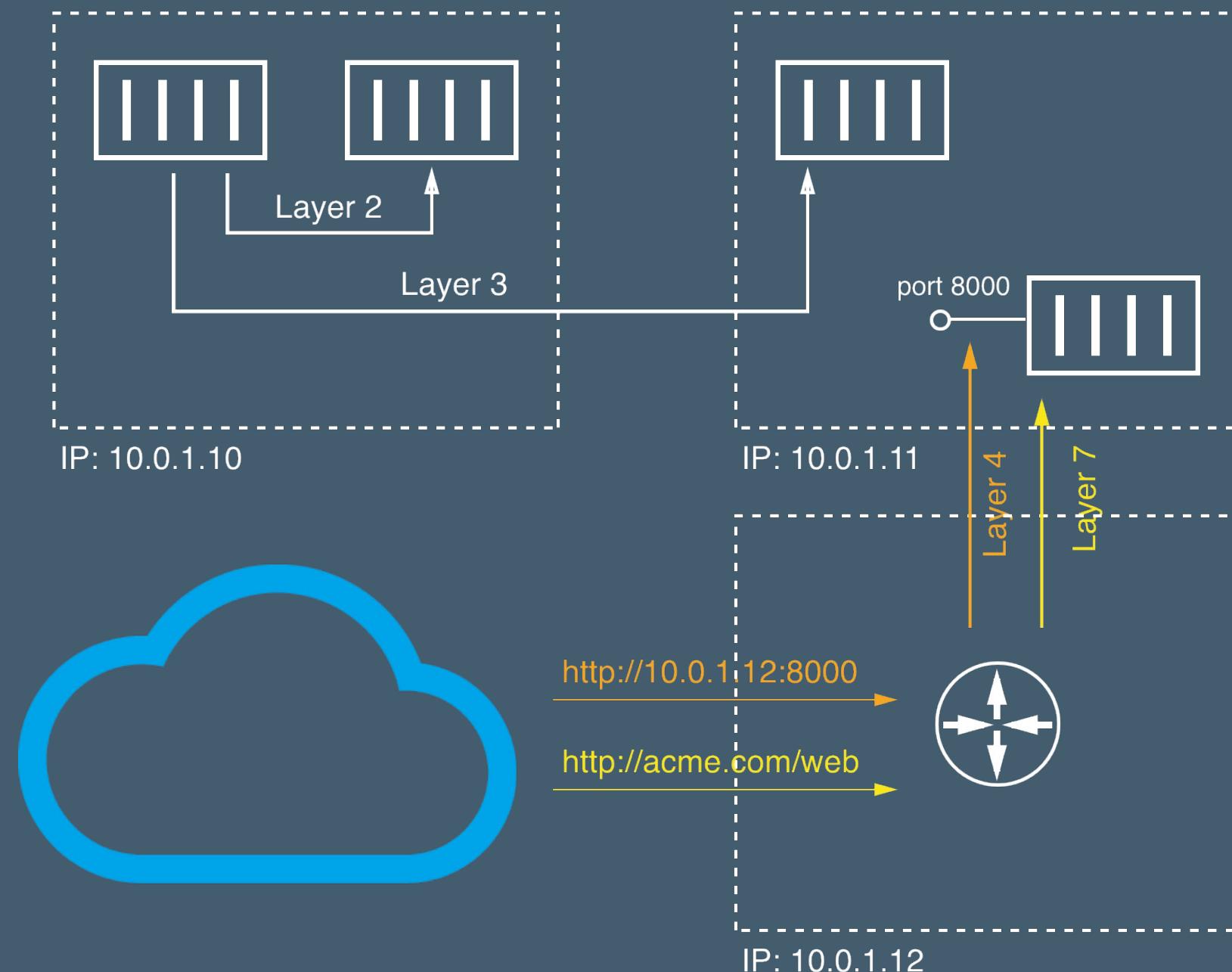
- Describe the role of routing, load balancing, and service discovery in containerized application management
- Route network traffic between containers at L2, 3, 4, and 7 with Swarm or Kubernetes



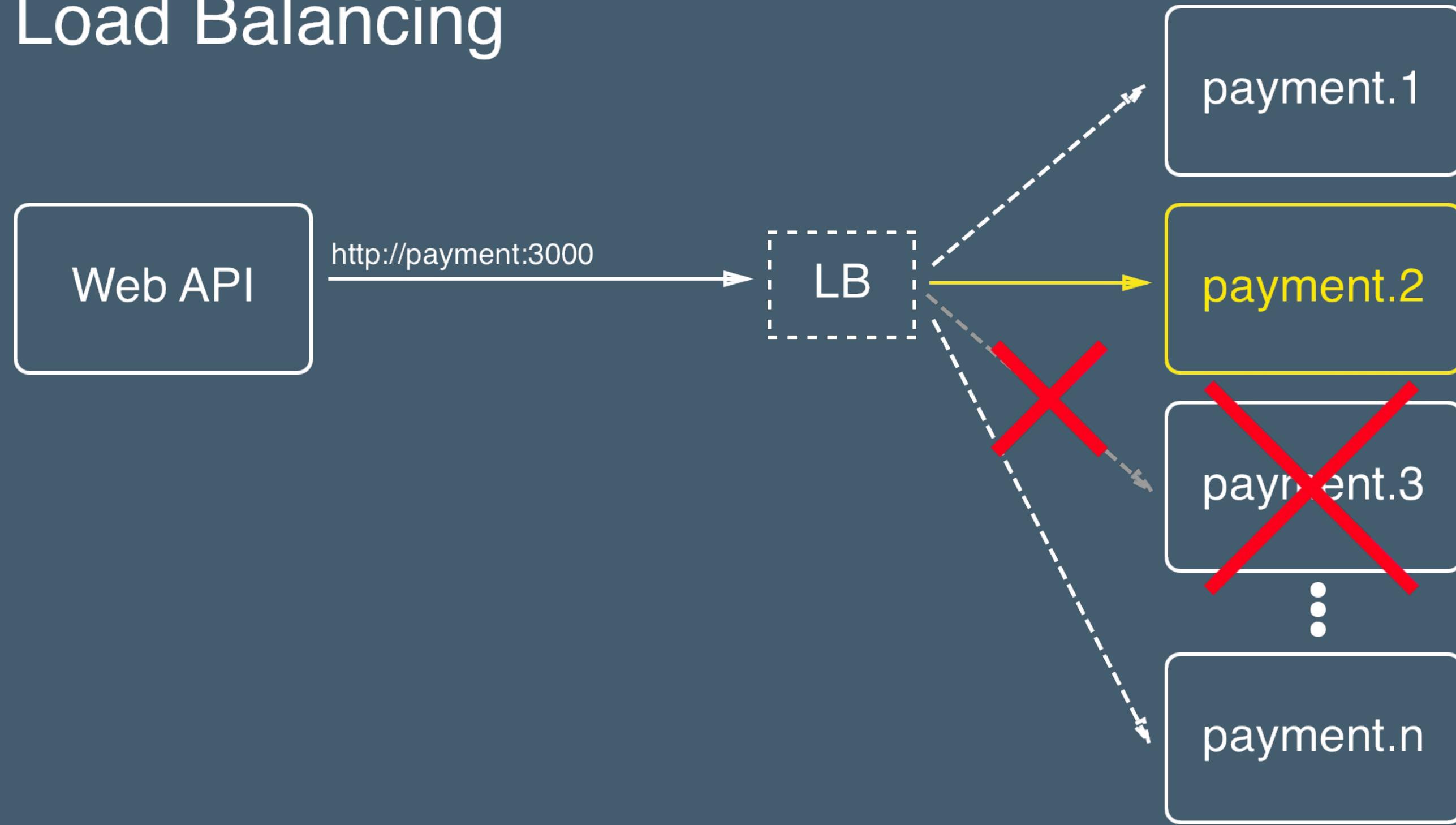
# Service Discovery



# Routing



# Load Balancing



# HIGHER LEVEL PATTERNS

- Introspection (distributed tracing and logging)
- Self healing systems
- Blue/green deployments
- Canary releases
- Seamless rollbacks



# SERVICE DISCOVERY - IMPLEMENTATION

- Swarm uses DNS service included in engine
- Kubernetes:
  - Environment Variables
  - Kubernetes DNS service





## EXERCISE: SERVICE MESH (1/3): SERVICE DISCOVERY

Work through exercise 'Service Mesh (1/3): Service Discovery' in the Docker for Enterprise Operations Exercises book.

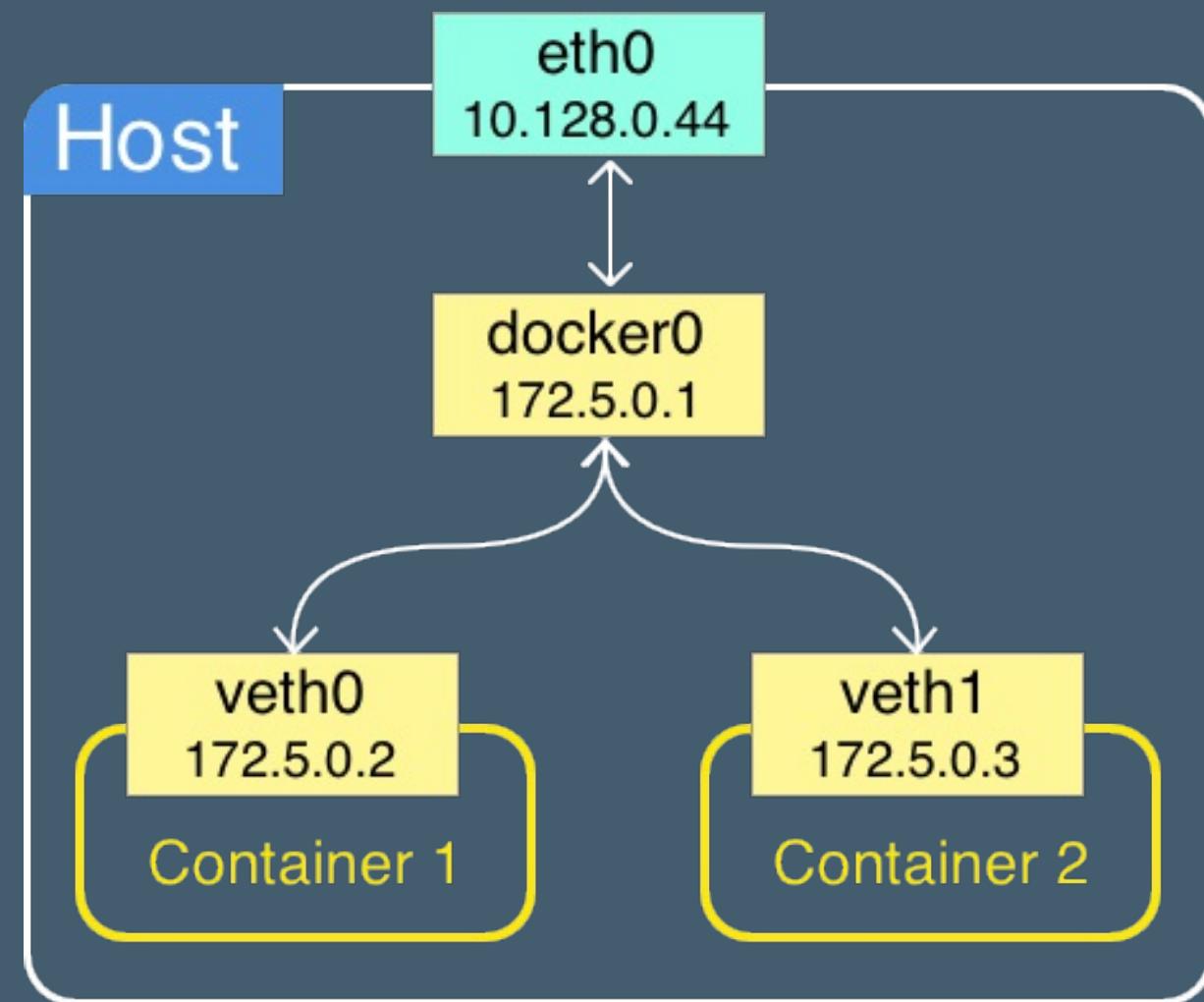


# ROUTING - IMPLEMENTATION

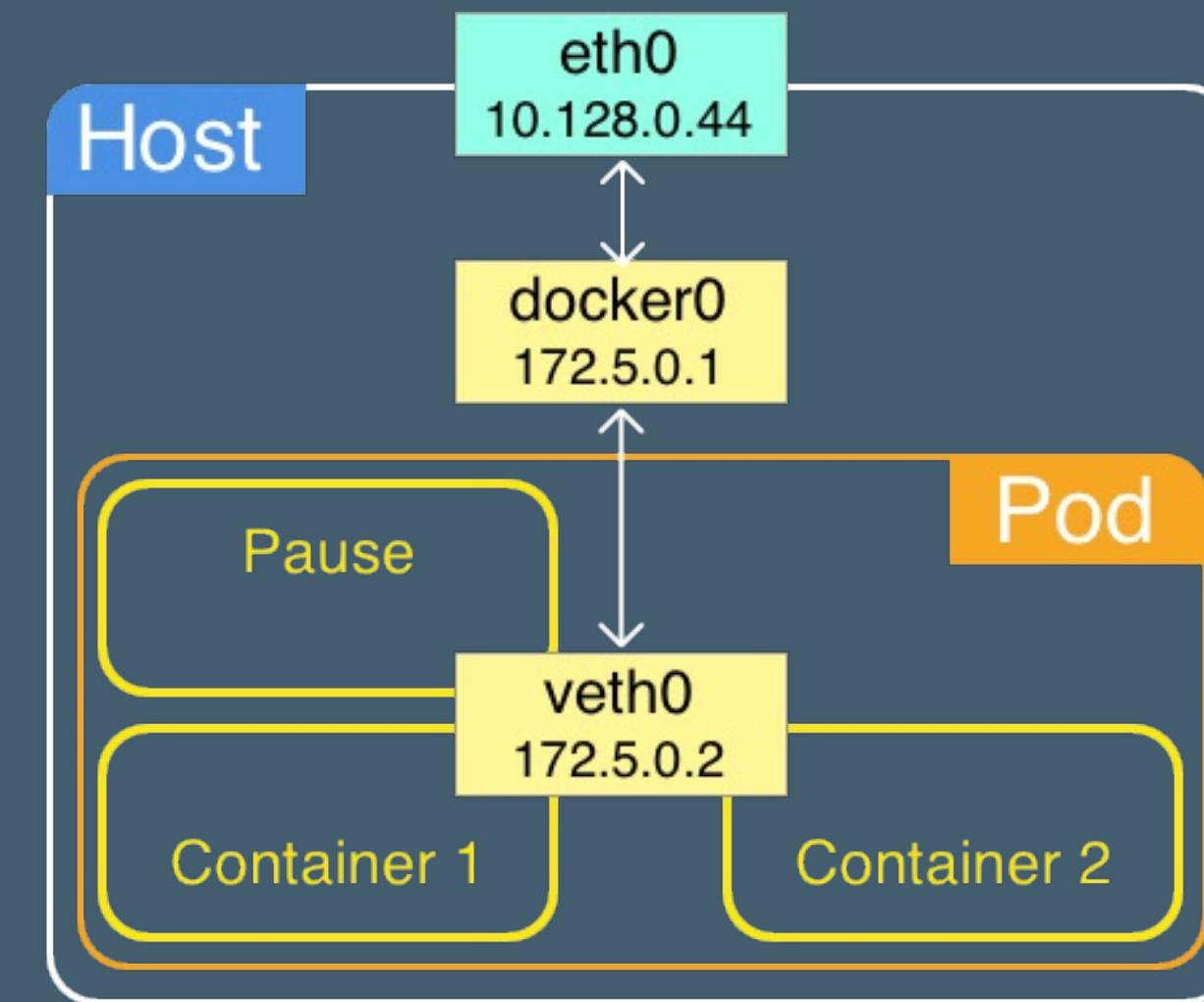
- Layer 2: Linux bridges
- Layer 3:
  - Swarm: VXLAN overlay
  - Kubernetes: plugin-dependent; **clusterIP** service
- Layer 4:
  - Swarm: Routing Mesh
  - Kubernetes: **nodePort** service
- Layer 7:
  - Swarm: Interlock 2
  - Kubernetes: Kubernetes Ingress resource



# L2 ROUTING



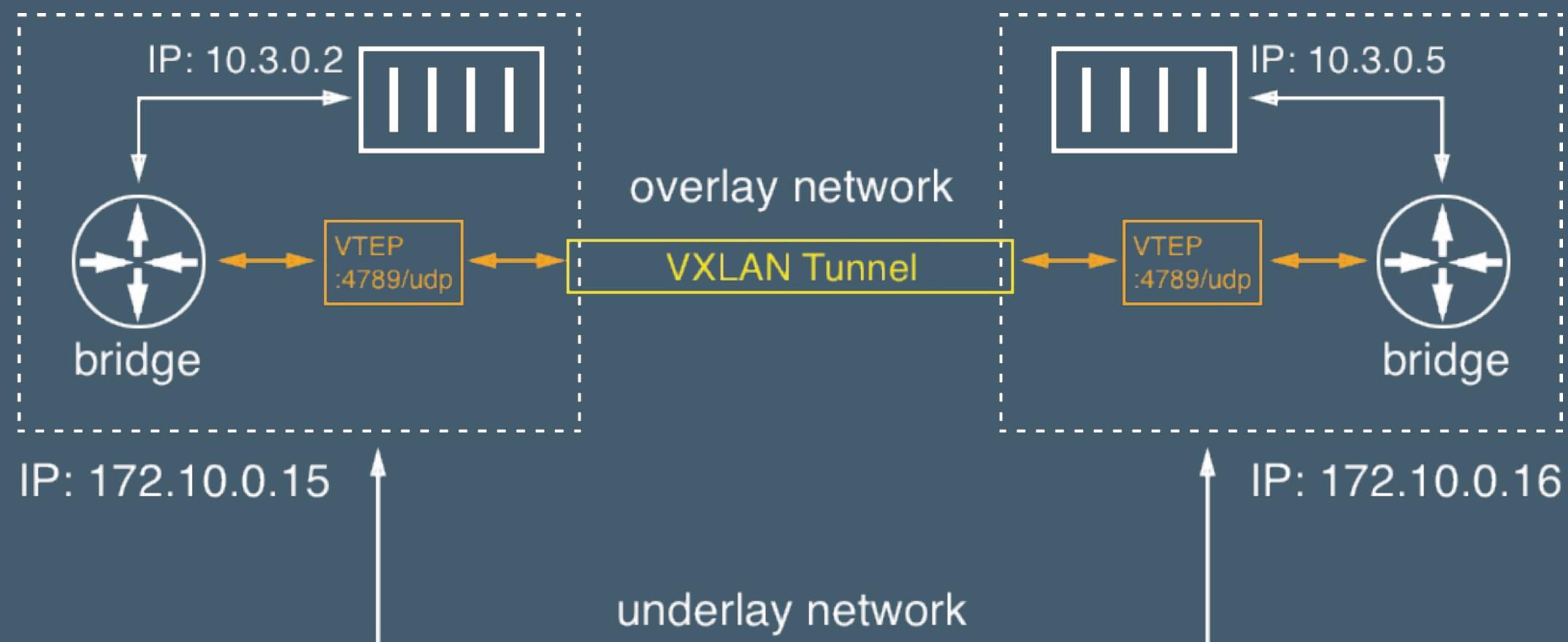
Docker Bridge Network



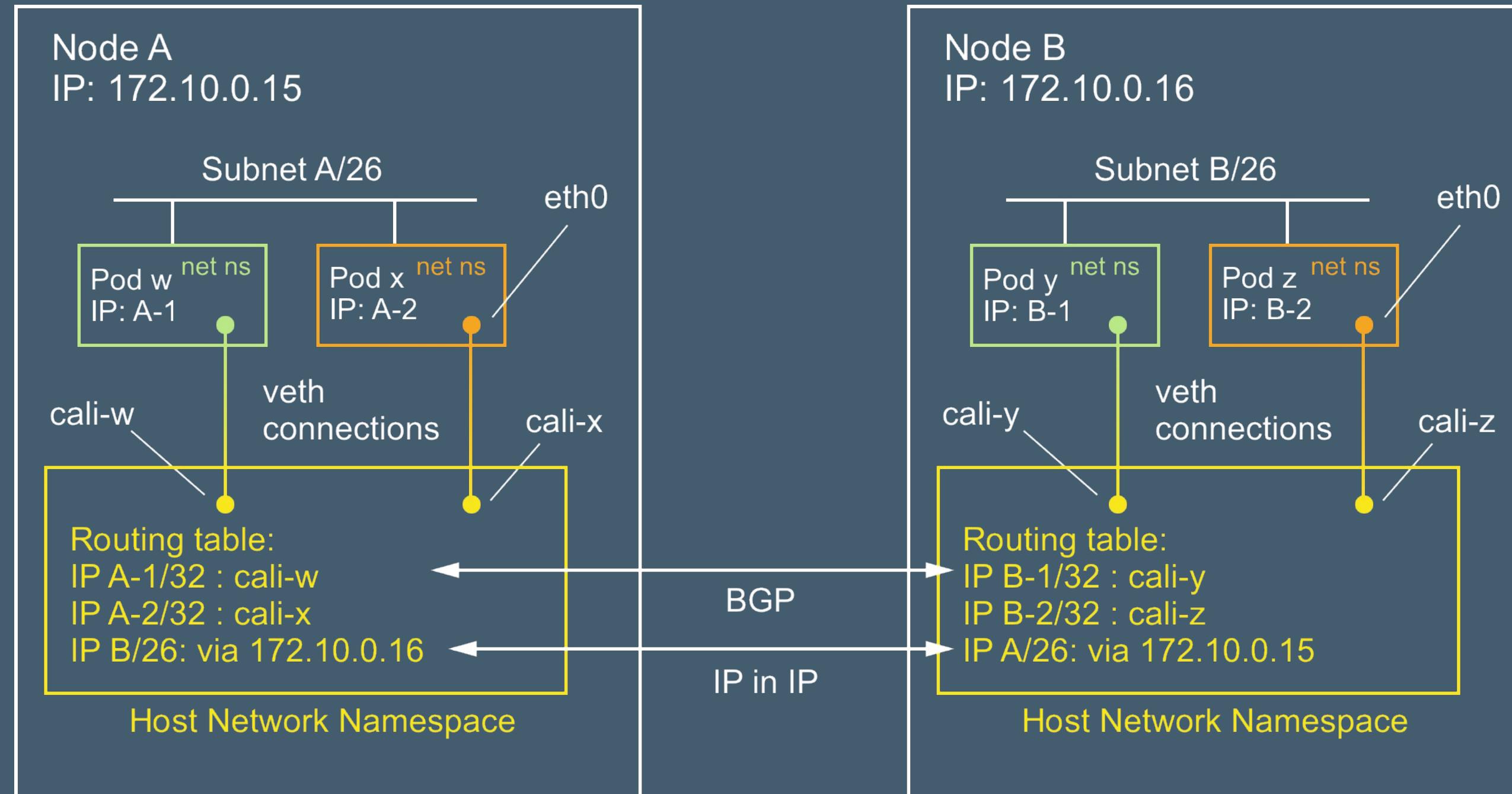
Kubernetes Network



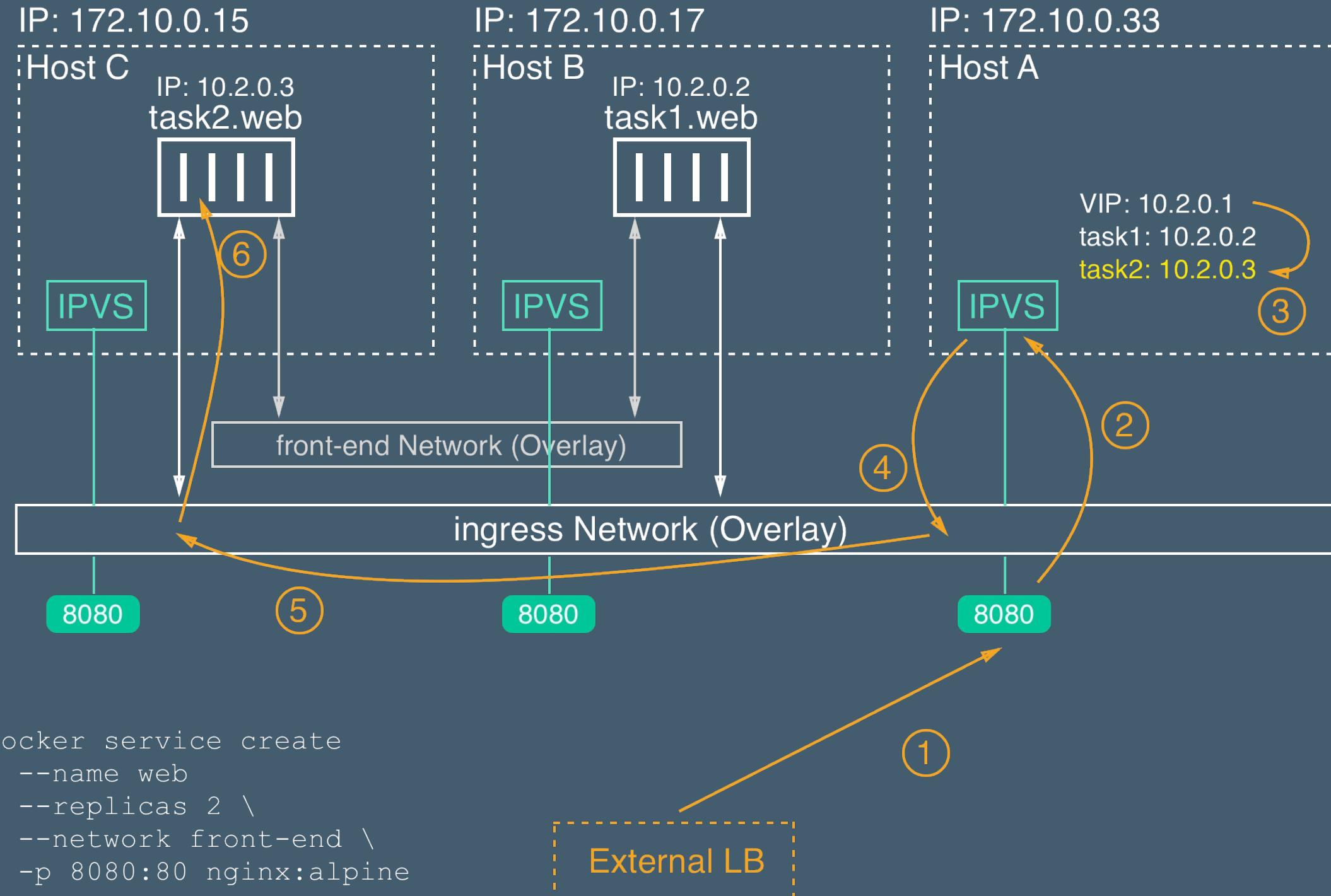
# L3 ROUTING - SWARM: VXLAN



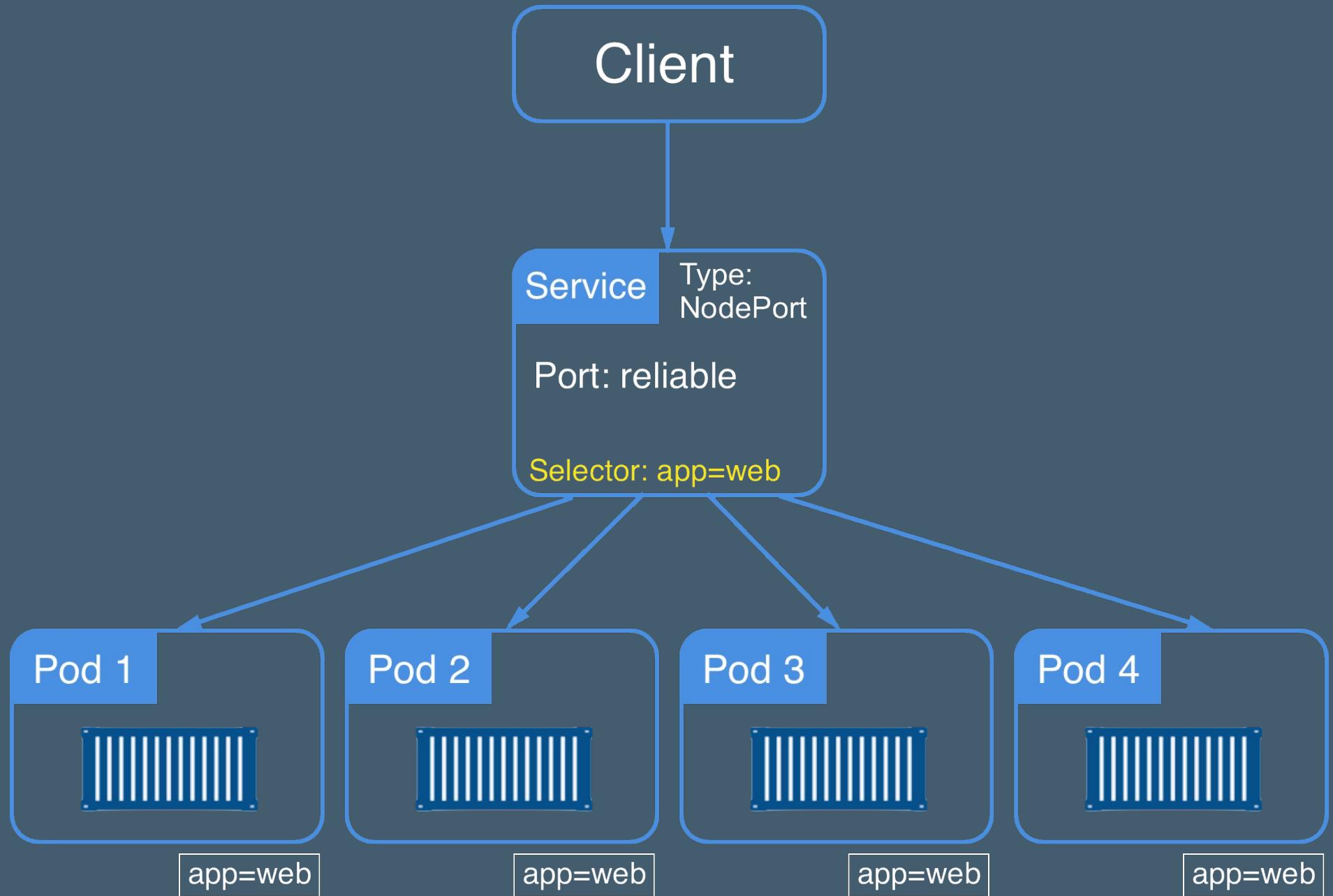
# L3 ROUTING - KUBE / CALICO



# L4 ROUTING - SWARM: ROUTING MESH



# L4 ROUTING - KUBERNETES: SERVICE OBJECT



# L7 ROUTING - SWARM: INTERLOCK 2

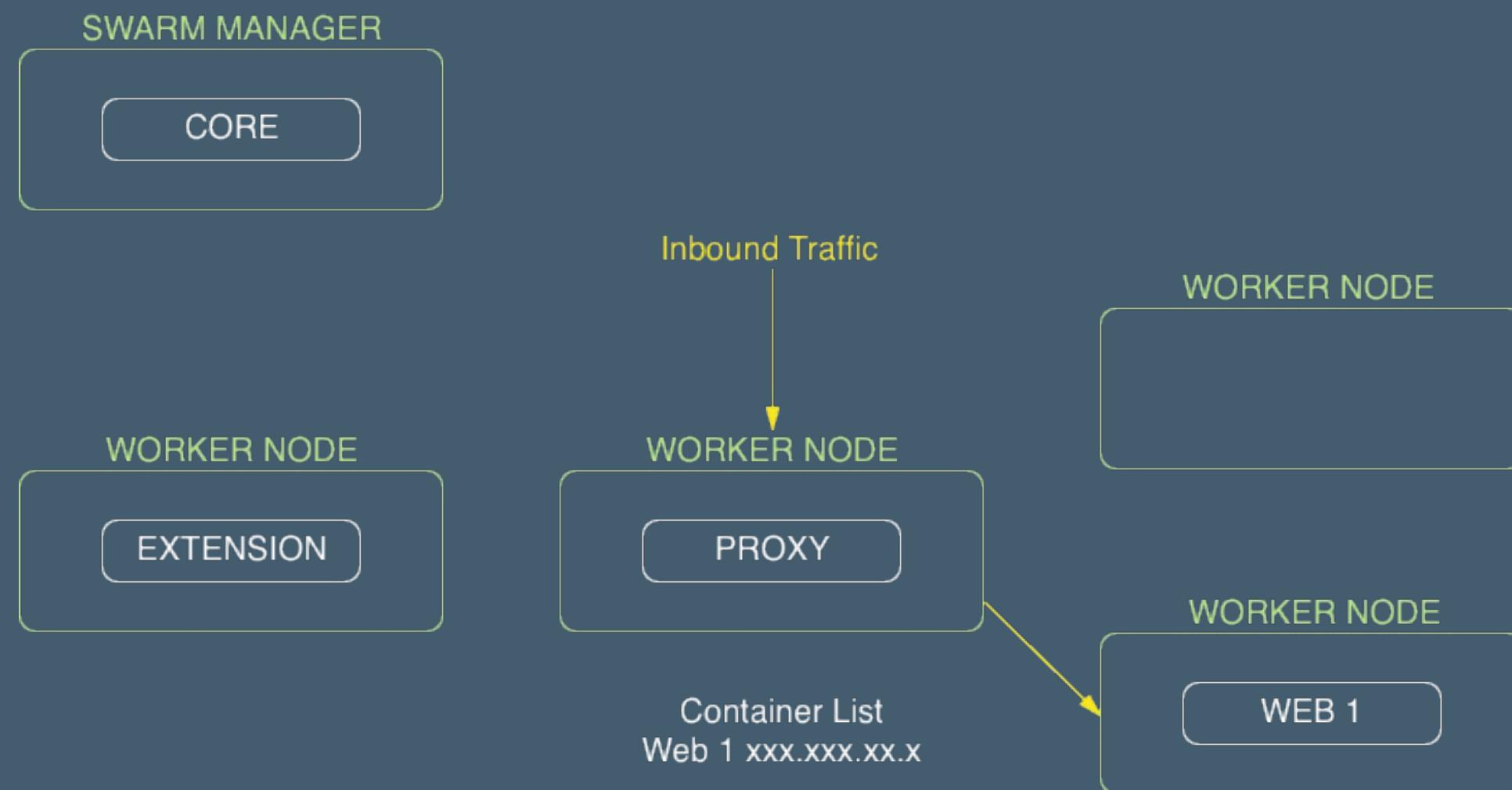
- Extends Routing Capabilities of Routing Mesh
- Routes Traffic based on Extensions

Architecture: 3 Docker Swarm services:

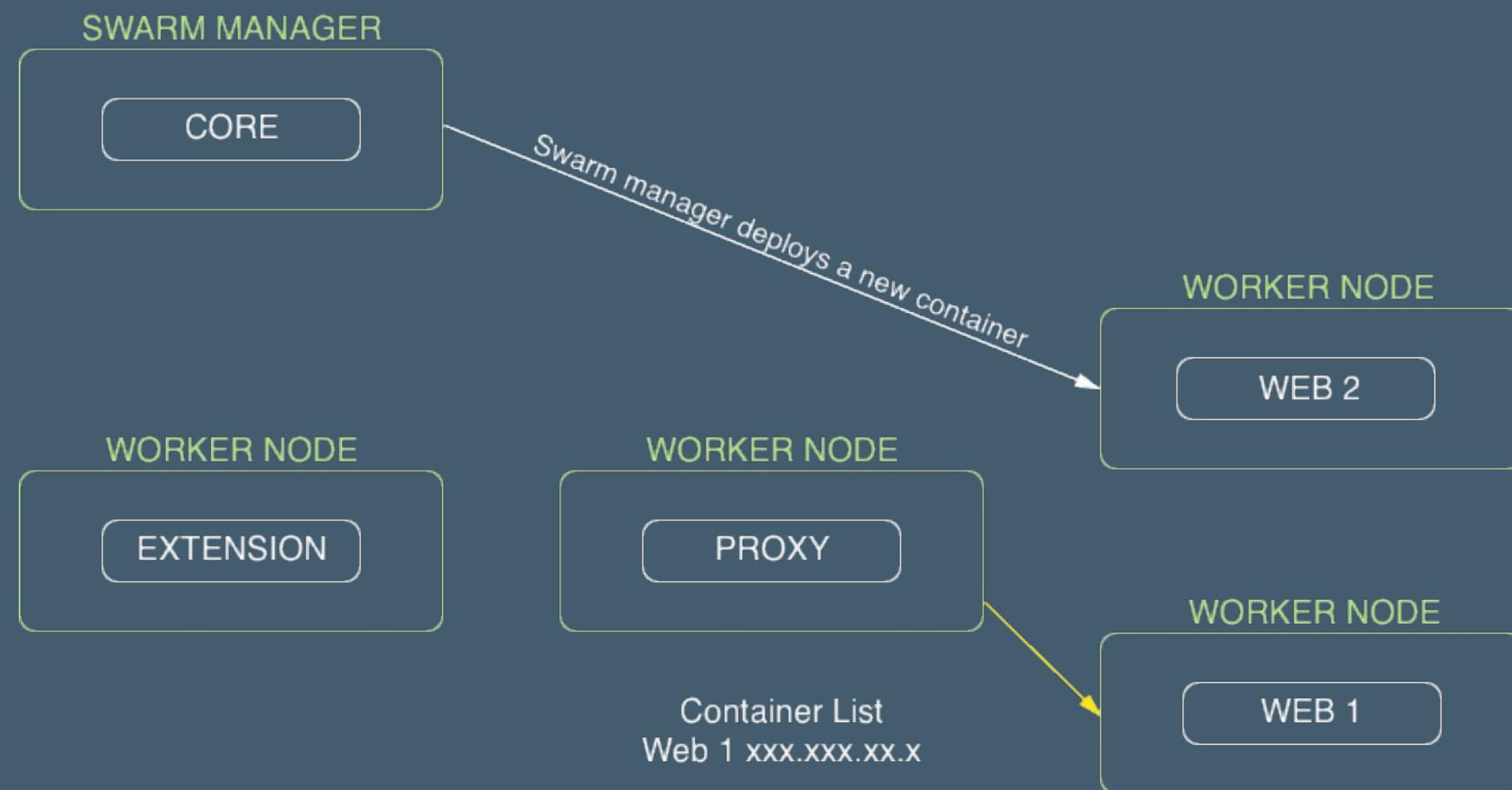
- Core
- Extension
- Proxy



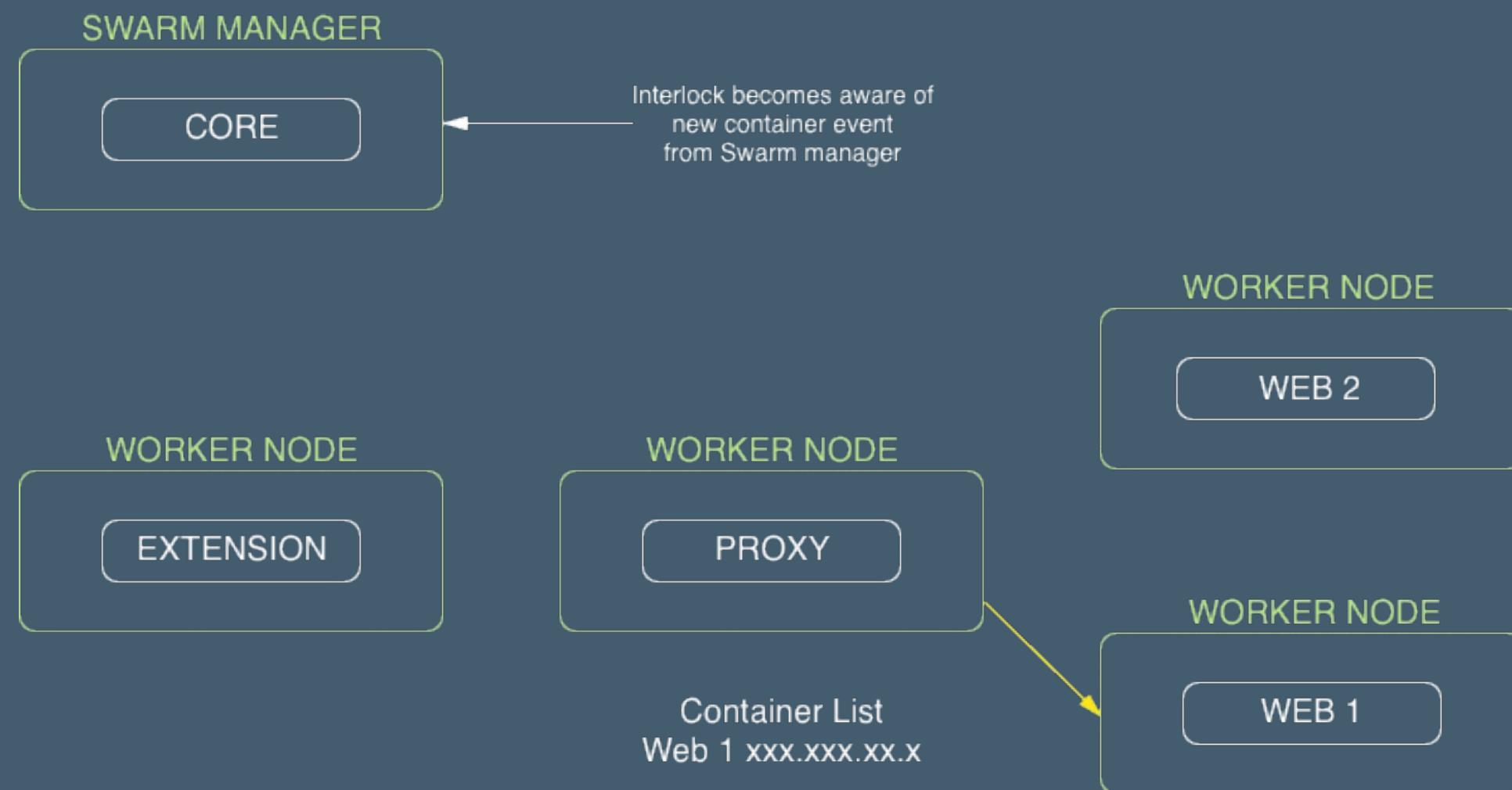
# INTERLOCK 2 TRAFFIC



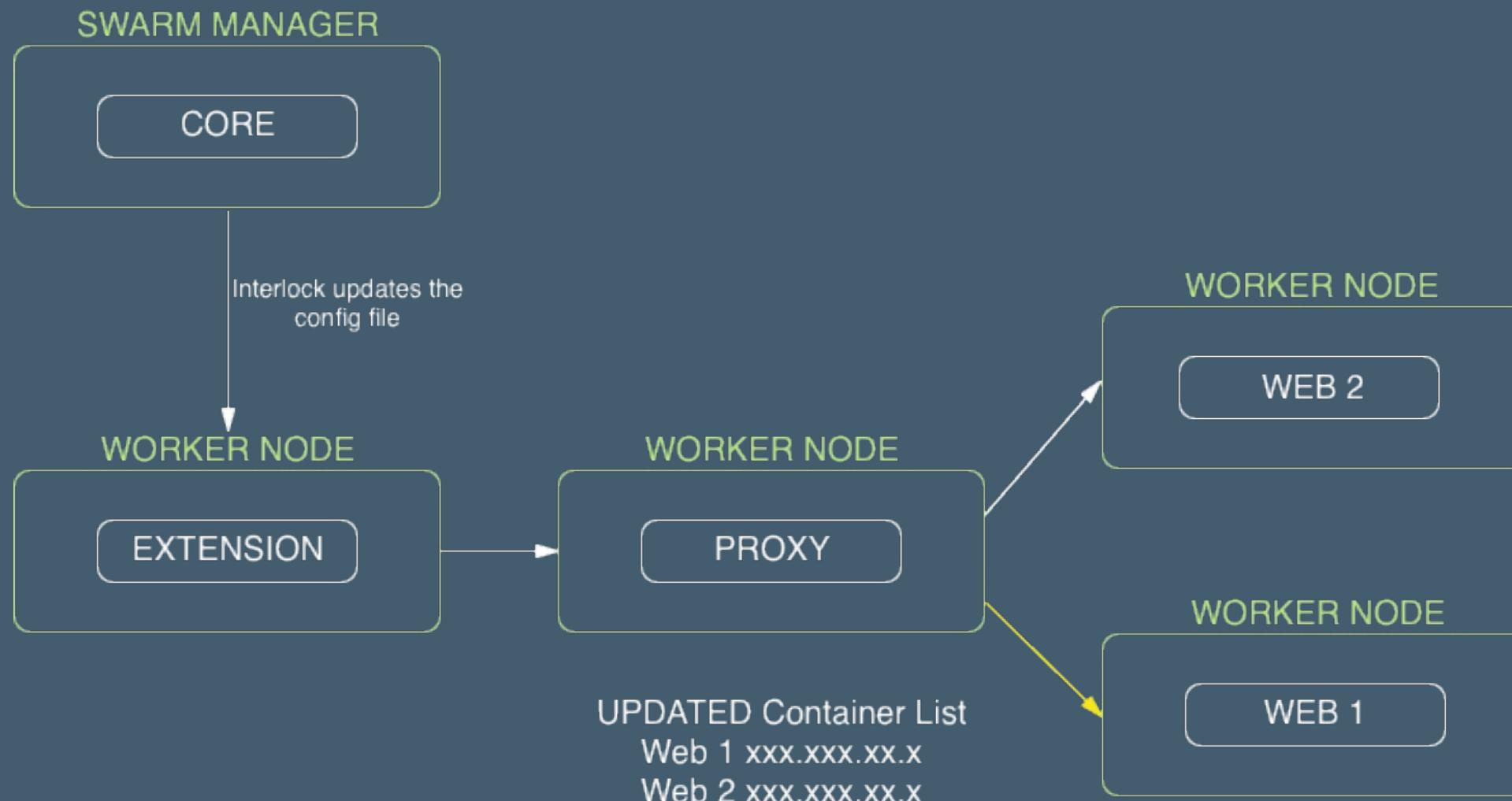
# INTERLOCK 2 TRAFFIC



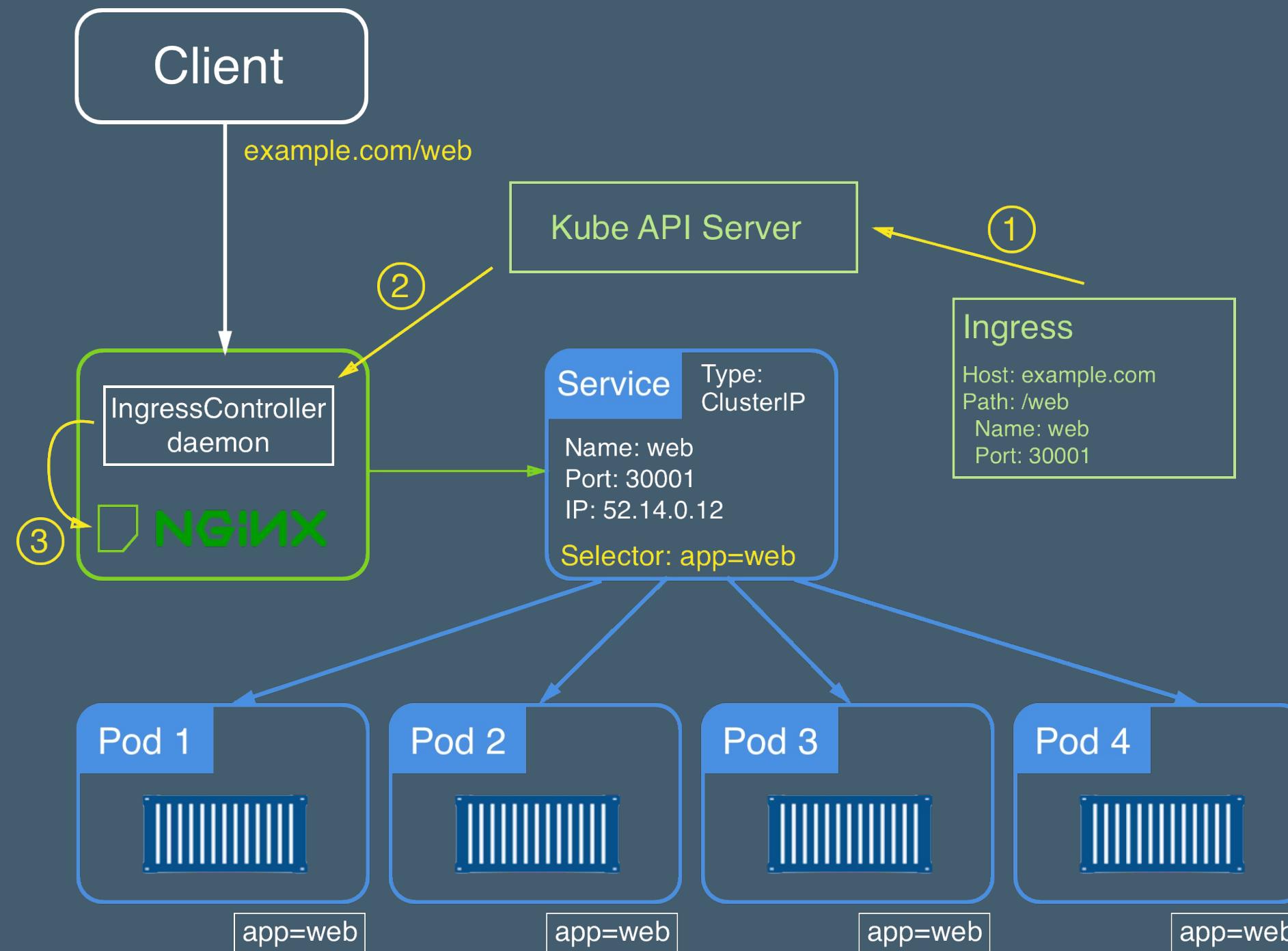
# INTERLOCK 2 TRAFFIC



# INTERLOCK 2 TRAFFIC



# L7 ROUTING - KUBE: INGRESS CONTROLLER





## EXERCISE: SERVICE MESH (2/3): ROUTING

Work through exercise 'Service Mesh (2/3): Routing' in the Docker for Enterprise Operations Exercises book.

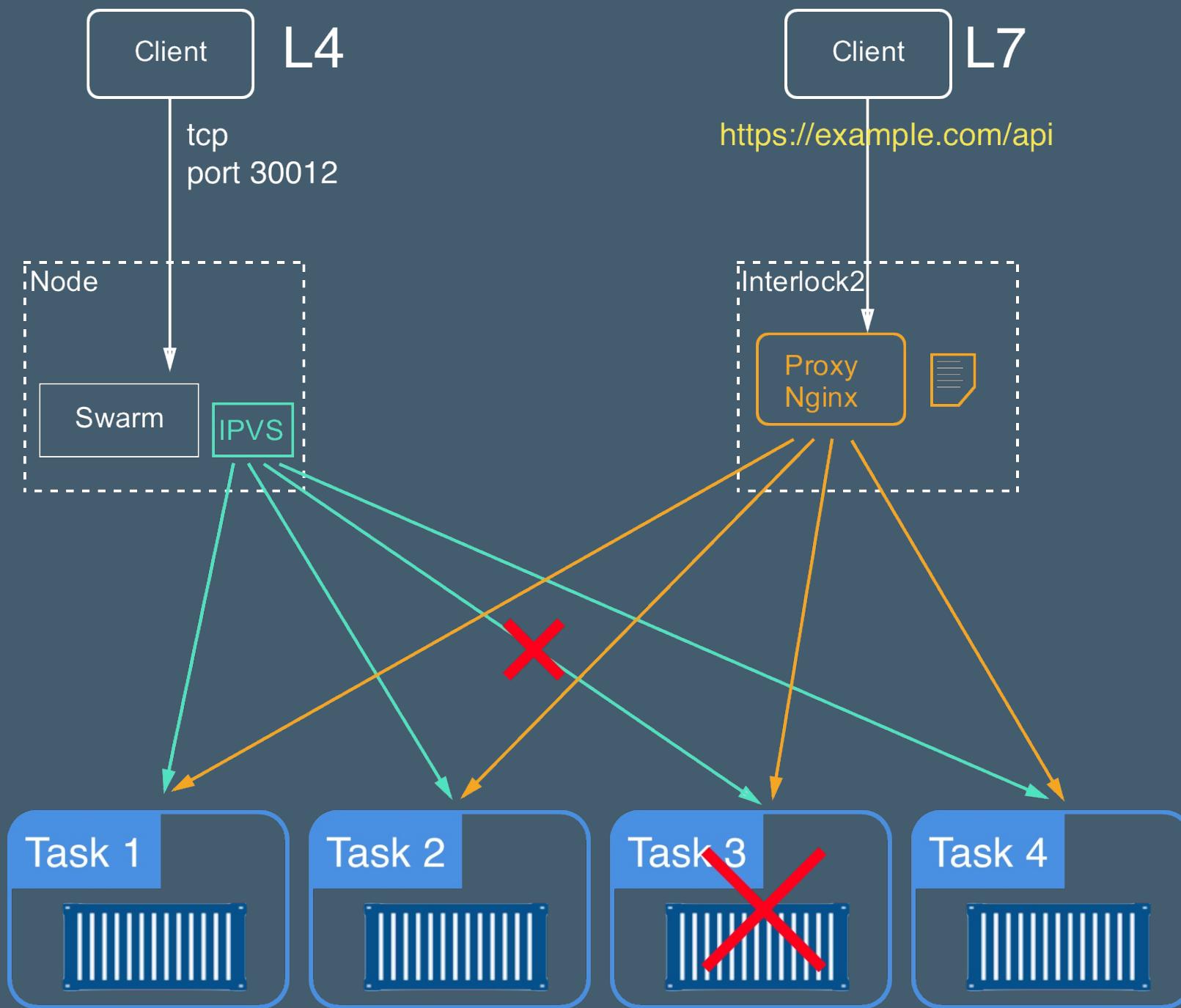


# LOAD BALANCING - IMPLEMENTATION

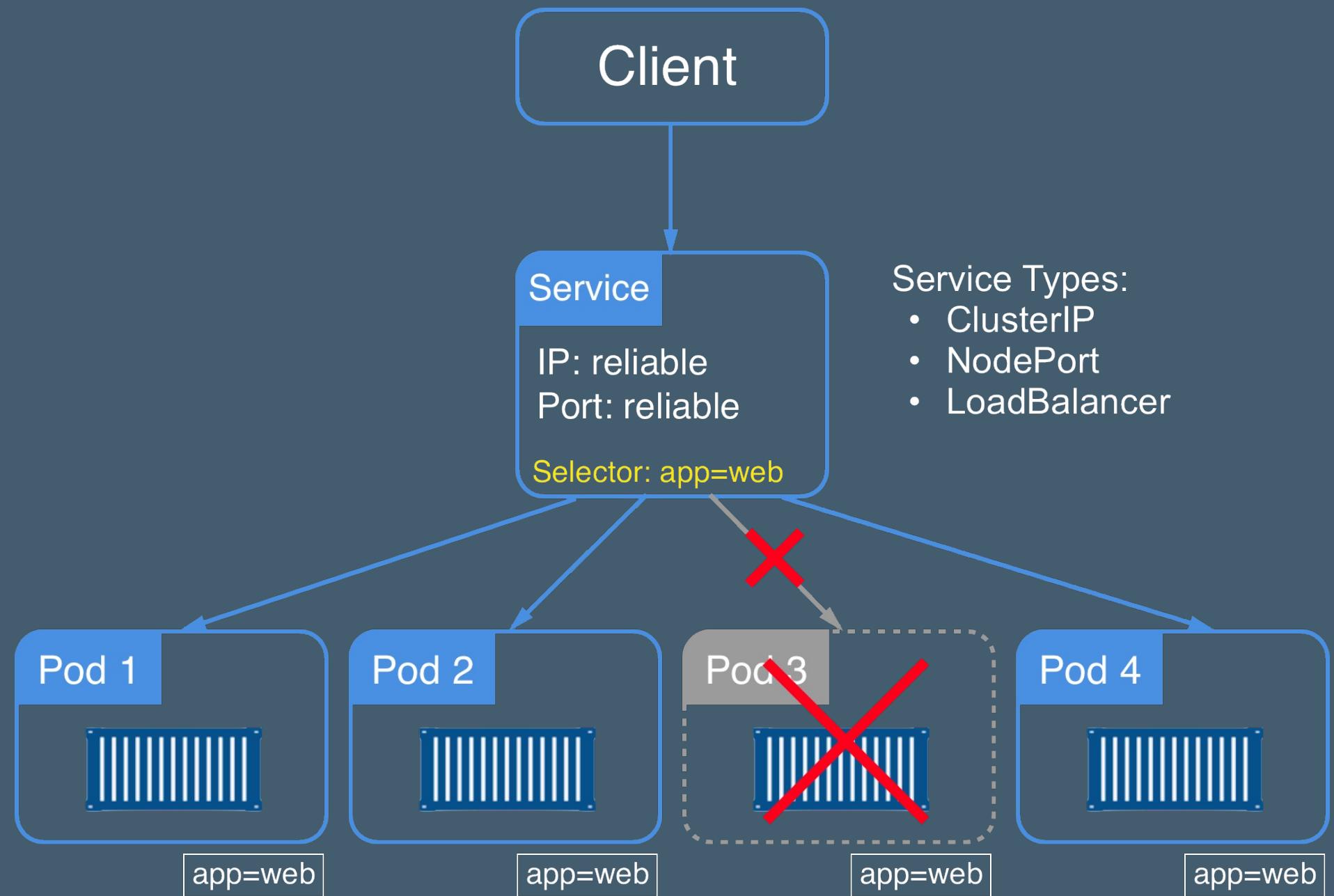
- Swarm: RoutingMesh & Interlock2
- Kubernetes: Kubernetes Services & Ingress Controller



# SWARM - LOAD BALANCING



# KUBERNETES - LOAD BALANCING





## EXERCISE: SERVICE MESH (3/3): LOAD BALANCING

Work through exercise 'Service Mesh (3/3): Internal Load Balancing' in the Docker for Enterprise Operations Exercises book.



# DISCUSSION

- In what situations would L7 routing be preferable to L4 routing?
- Questions?



## FURTHER READING

- Use a local node network in a cluster: <https://dockr.ly/2HmWxdD>
- Virtual Extensible LAN (VXLAN): <http://bit.ly/2EjUhii>
- Kubernetes Networking: <http://bit.ly/2rPHAcI>
- Kubernetes Services: <https://bit.ly/2GSXwyB>
- Accessing Kubernetes Pods from Outside of the Cluster: <http://bit.ly/2DR6Jg>





# LOGGING



# LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Identify the sources of logging information in the Docker platform
- Configure an ELK stack to ingest and summarize logs from the Docker platform



# WHY IS CENTRAL LOGGING CRUCIAL?

- Containerized deployments are distributed but connected
- large number of components = high rate of breakage
- Must correlate events...
- ... without granting access to production nodes!



# DOCKER LOGGING

- Logs come from two sources:
  - dockerd: system-level
  - containers: STDOUT & STDERR
- Globally configured by **/etc/docker/daemon.json**:

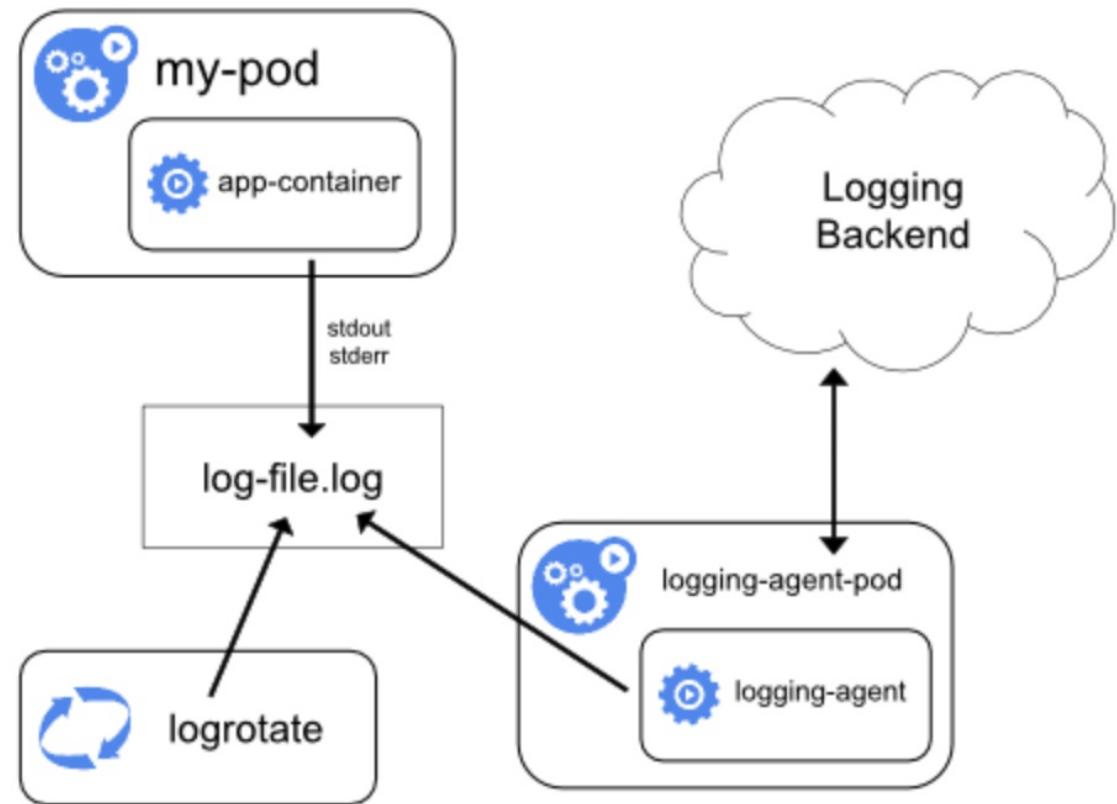
```
{  
  "log-driver": "journald",  
  "log-level": "debug",  
  "log-opt": {  
    "tag": "{ .ImageName }/{ .Name }/{ .ID }",  
    "max-size": "100m",  
    "max-file": 3  
  }  
}
```

- Override for containers with **--log-opt**

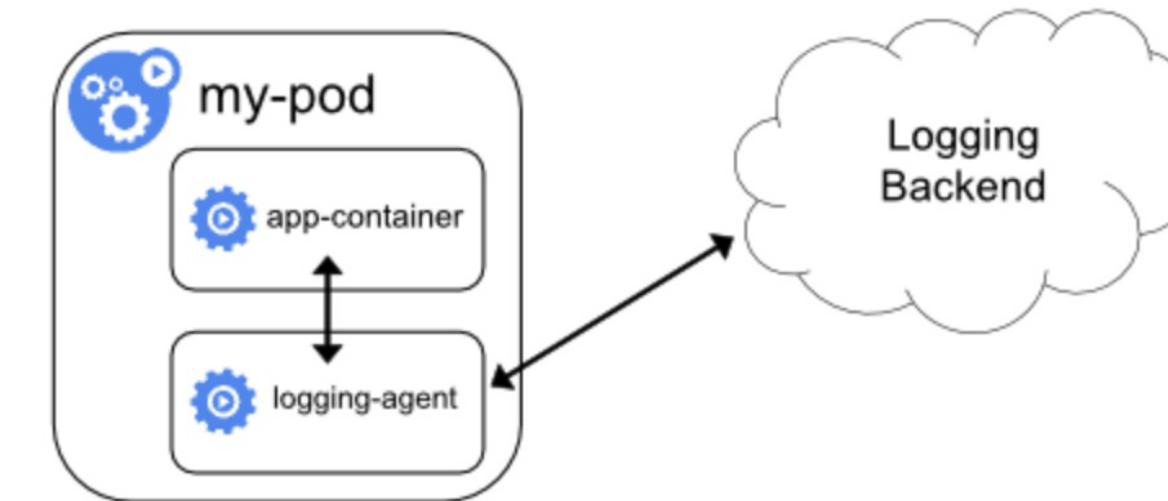


# KUBERNETES LOGGING

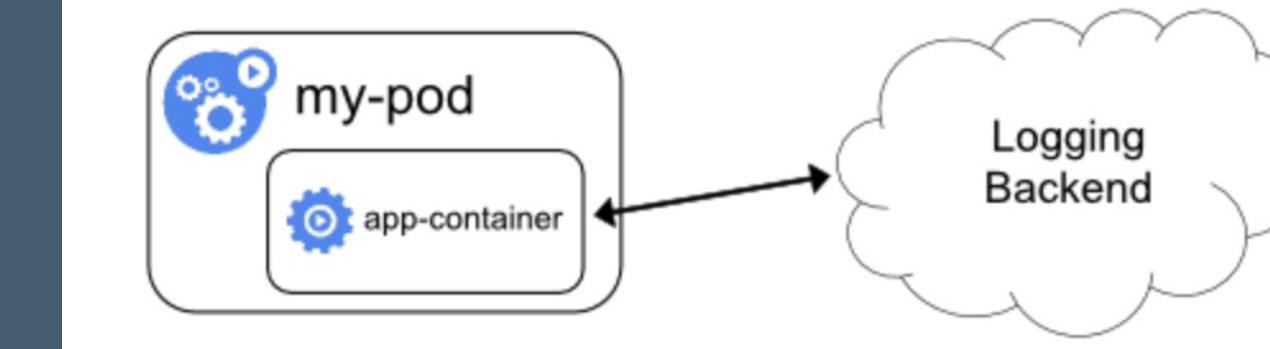
Using a node logging agent



**Sidecar container with a logging agent**



Exposing logs directly from the application





## EXERCISE: LOGGING

Work through the 'Logging' exercise in the Docker for Enterprise Operations Exercises book.



# DISCUSSION

- What logging info would you like your containers to provide?
- Questions?



## FURTHER READING

- View logs for a container or service: <http://dockr.ly/2ezdZdl>
- Docker Reference Architecture: Docker Logging Design and Best Practices:  
<http://dockr.ly/2gG6ZjG>



 APPLICATION HEALTH & READINESS CHECKS

# LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Describe the steps of a generic health check protocol
- Configure Swarm and Kubernetes to kill unhealthy containers, and configure Kubernetes to remove unready pods from load balancing



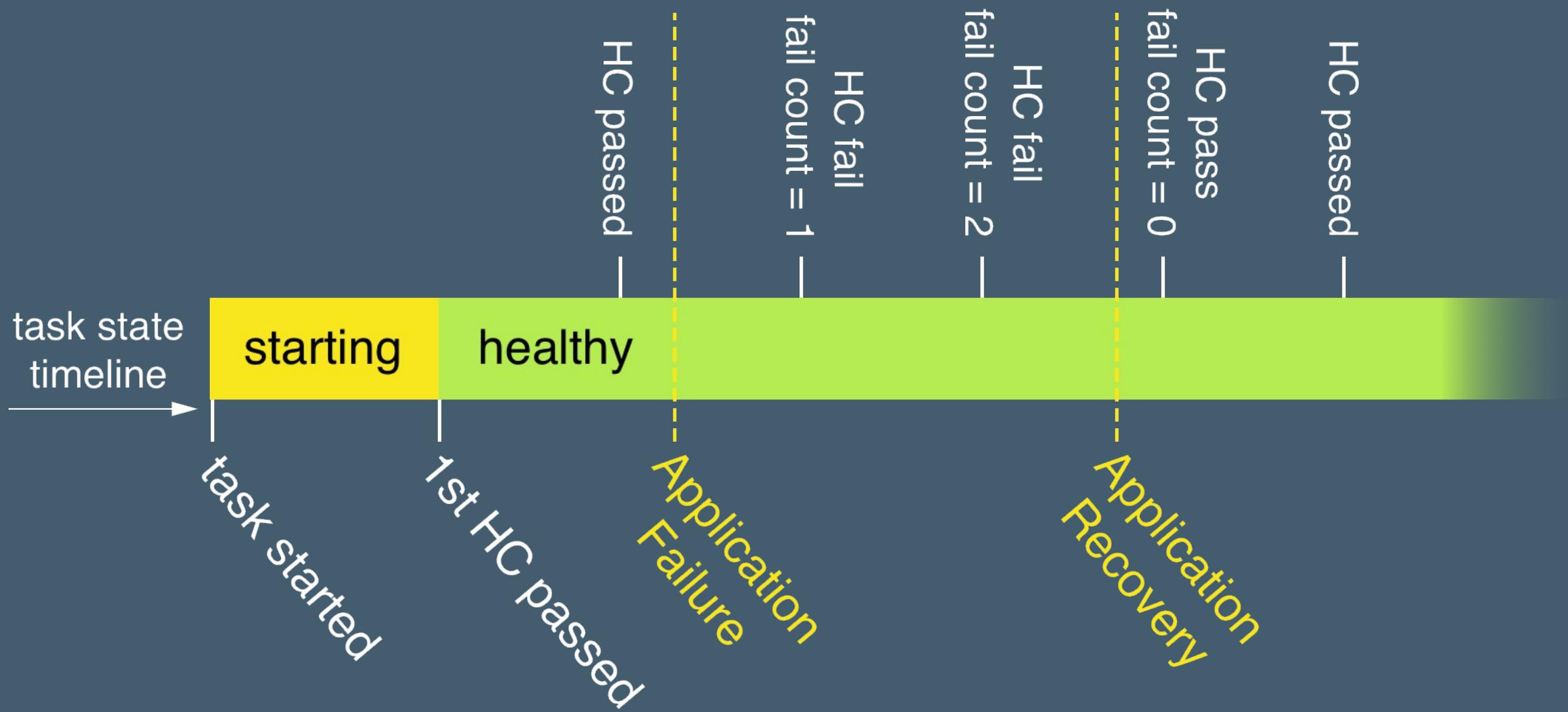
# HEALTH CHECK PROTOCOLS

Monitoring application health requires:

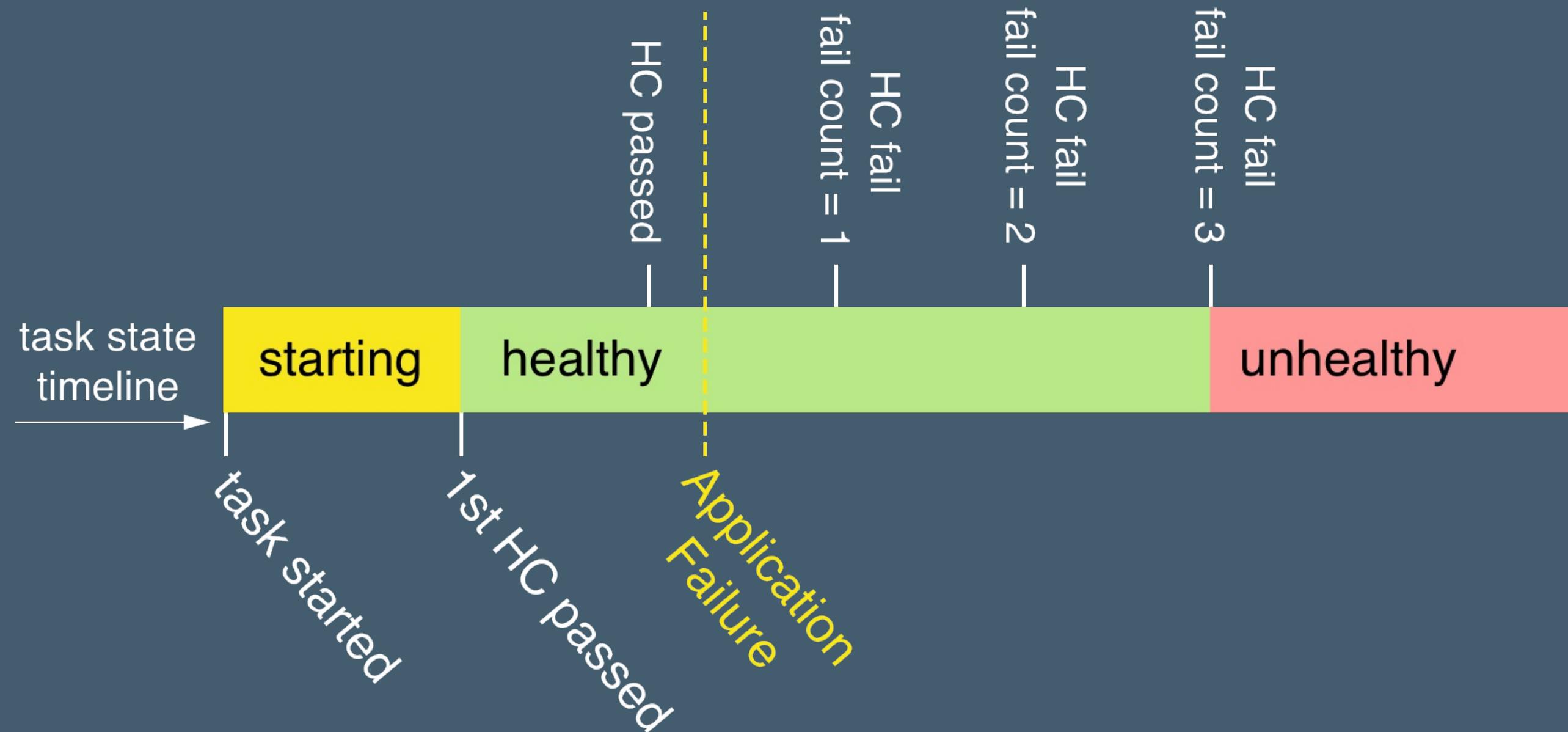
- Action to check health
- Frequency of checks
- Timeout per check
- Number of checks



# HEALTH CHECK PROTOCOL - HEALTHY



# HEALTH CHECK PROTOCOL - UNHEALTHY



# HEALTHCHECK: SWARM SERVICE CONTAINER

- Dockerfile

```
HEALTHCHECK CMD curl --fail http://localhost:5000/health || exit 1
```

- Docker Compose File

```
healthcheck:  
  interval: 10s  
  timeout: 2s  
  retries: 3  
  start-time: 30s
```



# HEALTHCHECK: KUBE LIVENESS PROBE

Kube yaml:

```
kind: Pod
...
spec:
  containers:
    - name: demo
      image: ...
      livenessProbe:
        periodSeconds: 10
        timeoutSeconds: 2
        failureThreshold: 3
        initialDelaySeconds: 30
        successThreshold: 1
        exec:
          command:
            - cat
            - /tmp/healthy
...

```



# ALTERNATE LIVENESS PROBES

- HTTP Request: success if  $200 \leq \text{response} < 400$
- TCP socket: success if connection succeeds on specified port



# KUBERNETES READINESS PROBE

- Defined under **readinessProbe**
- Same syntax as **livenessProbe**
- No service traffic to unready pods





## EXERCISE: HEALTH CHECKS

Work through the 'Health Checks' exercise in the Docker for Enterprise Operations Exercises book.



# DISCUSSION

- Suppose a misconfiguration guarantees application health failures in containers after an update to a service's image. How can you prevent all replicas of the service from being 'upgraded' to the new, unhealthy containers?
- Questions?



## FURTHER READING

- How to implement service upgrades without impacting your application:  
<http://dockr.ly/2iXctqY>





# DOCKER TRUSTED REGISTRY



ee2.0-v2.2 © 2018 Docker, Inc.

# LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Install DTR, configure its storage backend, and establish push and pull rights with a remote machine
- Overview a complete software supply chain supported by DTR
- Identify and troubleshoot common DTR installation problems



# THE SOFTWARE SUPPLY CHAIN

- Image Creation
- Image Distribution
- Container Execution

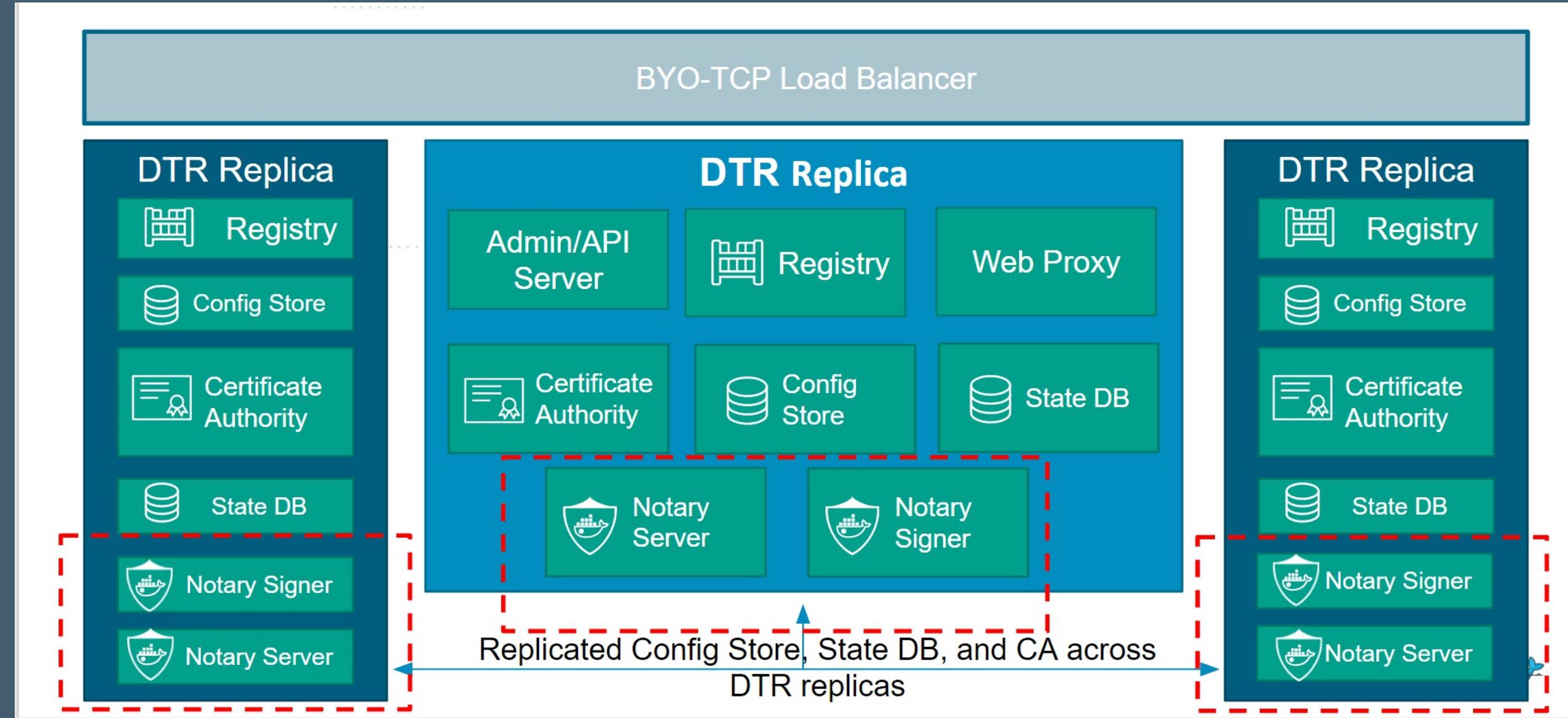


# DTR KEY FEATURES

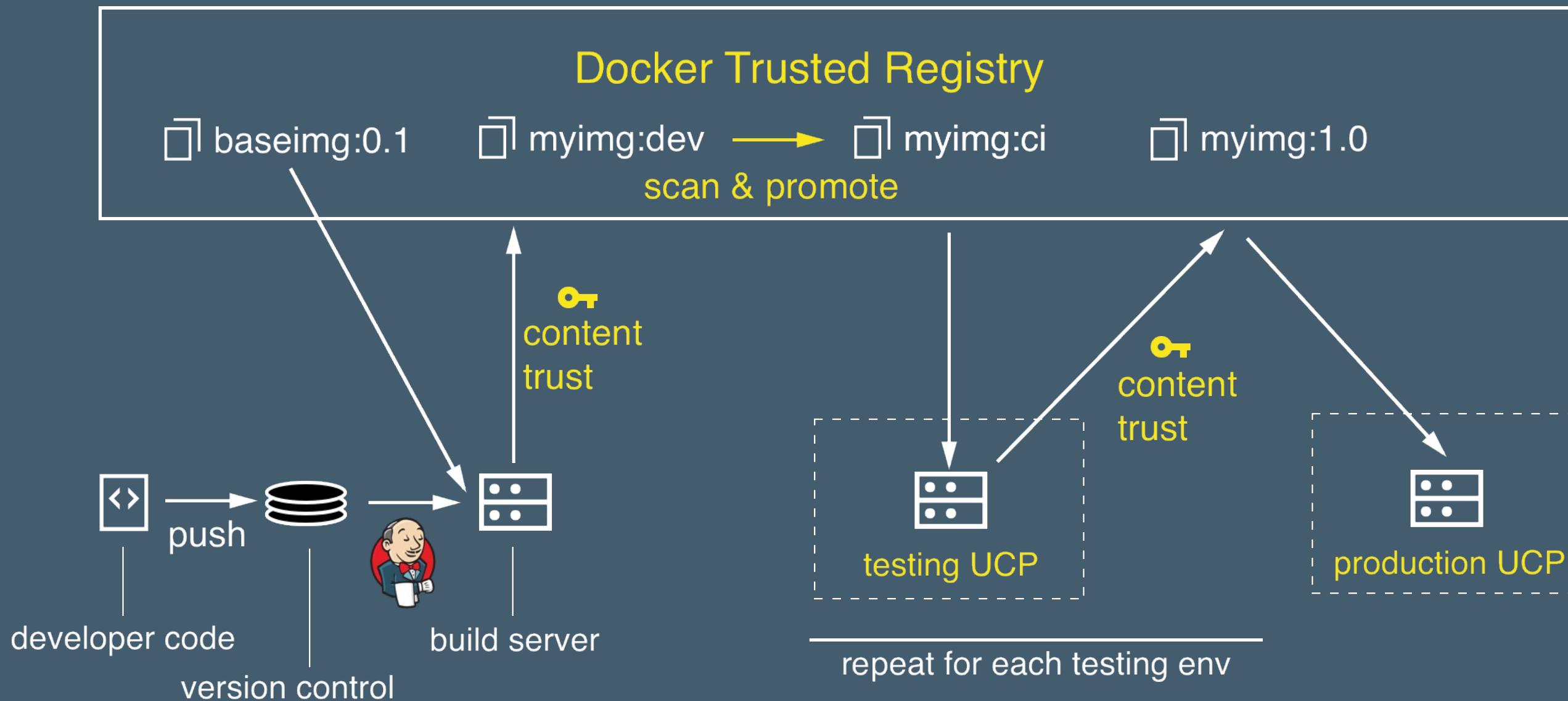
- Image Creation:
  - Image Security Scanning
  - Repository Automation
- Image Distribution:
  - Content Trust / Notary
  - Content Cache
- Image Storage:
  - Pluggable Storage Drivers



# DTR ARCHITECTURE



# A DOCKER PIPELINE





# DTR INSTALLATION

Work through:

- Install DTR
- Optional: Install DTR Replicas

in the Docker for Enterprise Operations Exercises book.



# DTR INSTALLATION PROBLEMS

- "Failed to execute phase2" -> allow containers on managers
- x509 errors -> establish cert trust
- JWT expiration -> impose ntp clock sync



# EMERGENCY REPAIR

- Allows an admin to recover a DTR cluster after a loss of quorum
- Works from a single `dtr-rethink` volume
- Use as a tool of last resort



## FURTHER READING

- DTR architecture: <https://dockr.ly/2JiEVvG>





# DTR ORGANIZATIONS AND TEAMS



# LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Understand and manipulate Role Based Access Control in DTR



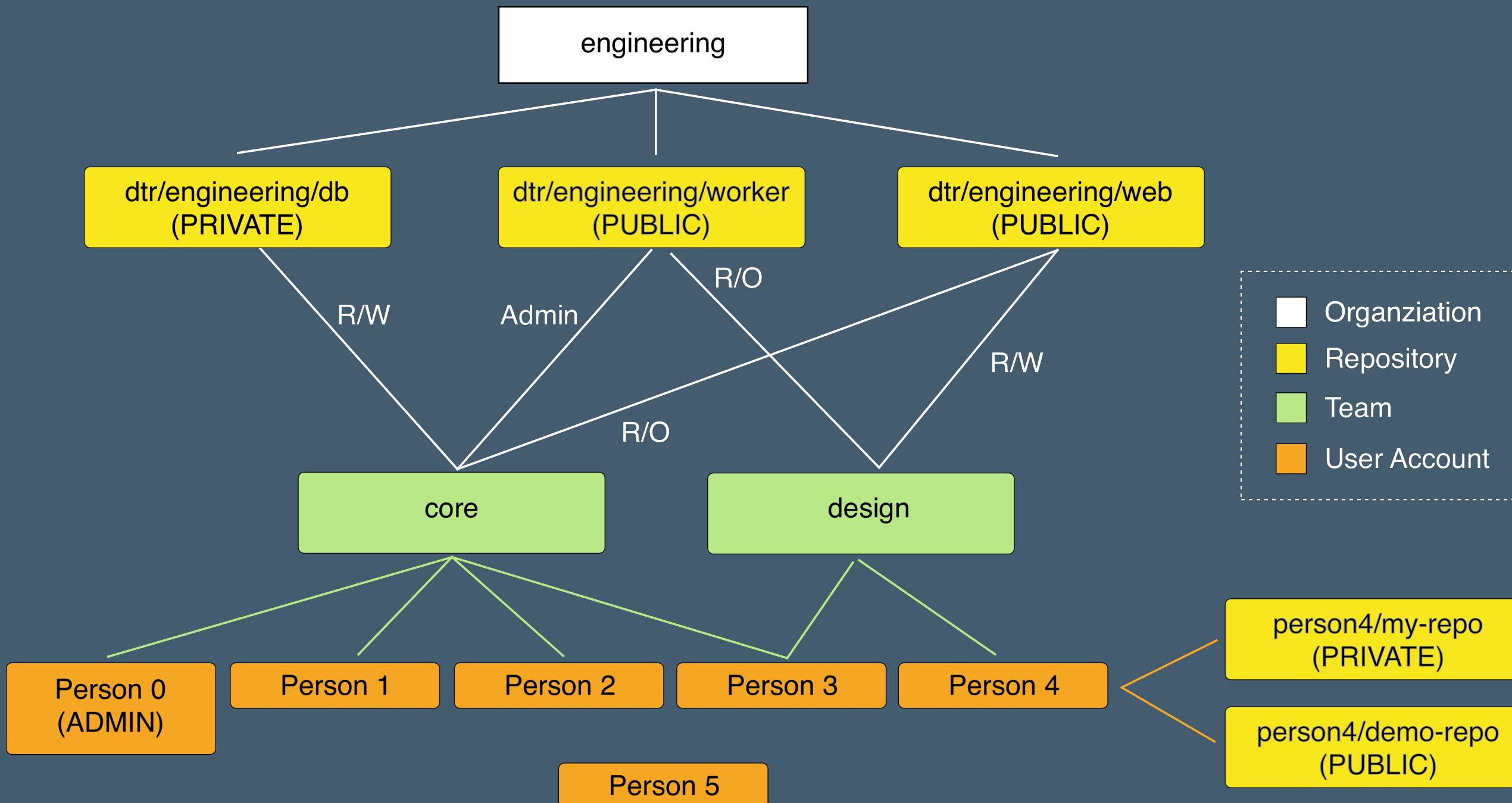
# DTR ORG CHARTS

Four key entities:

- Organizations namespace all other assets.
- Repositories hold images.
- Teams define access control to repositories.
- Users are grouped by teams and orgs.



# A 'SIMPLE' CASE



# REPOSITORY PERMISSIONS

Repo access is controlled by two concerns:

- Public vs. Private, and Org vs User owned
- Team Permissions



# PUBLIC/PRIVATE/OWNERSHIP MATRIX

	Public	Private
User-Owned	<ul style="list-style-type: none"><li>• Anyone can pull</li><li>• Visible to all</li><li>• Push by owner</li></ul>	<ul style="list-style-type: none"><li>• Only visible to owner &amp; admins</li></ul>
Org-Owned	<ul style="list-style-type: none"><li>• Anyone can pull</li><li>• Visible to all</li><li>• Push by R/W team</li></ul>	<ul style="list-style-type: none"><li>• Must be R/W or R/O team to see repo</li></ul>



# REPOSITORY PERMISSIONS

- Read only
  - Browse and pull
- Read write
  - Read only plus:
  - Push images
  - Delete tags
- Admin
  - Read write plus:
  - Edit repository description
  - Set public or private
  - Change team access level



# ORGANIZATION MEMBERS

- Org member without team membership
- Team members are automatically organization members
- Organization members can:
  - View other members
  - View org teams and members
  - View and pull images from public repositories in the org
- No way to grant org member write access to org-owned repo.



# ORGANIZATION OWNERS

- Individual org members can be made into an organization owner
- Organization owners:
  - Have admin access to all org-owned repos
  - Can manage org and team membership



# USER TOKEN MANAGEMENT

- Allows the management of 'docker login' tokens
- Avoids use of passwords



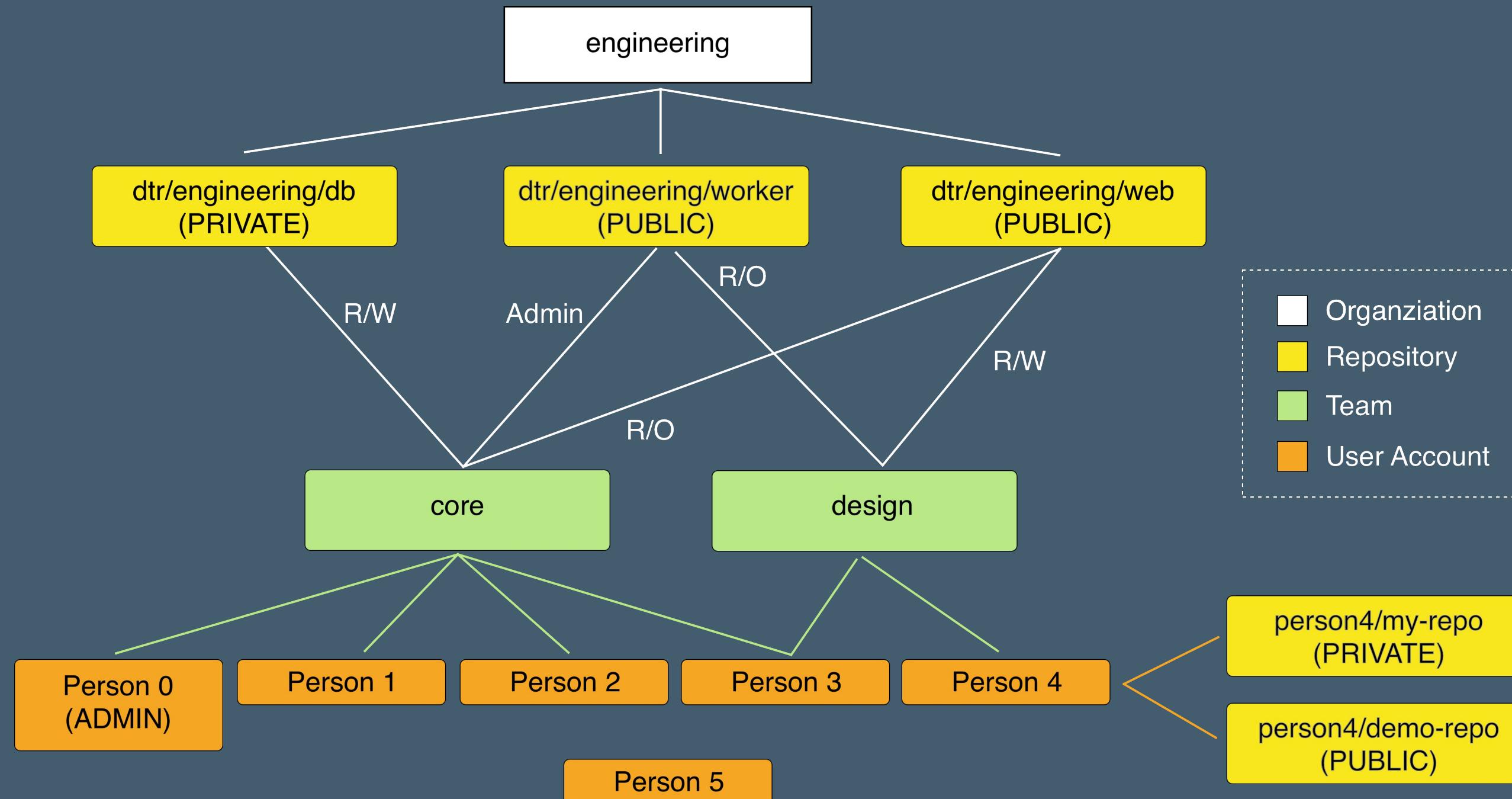


## EXERCISE: DTR TEAMS

Work through the 'Working with Organizations and Teams' exercise in the Docker for Enterprise Operations exercise book.



# QUIZ



## FURTHER READING

- Create and Manage Organizations: <https://dockr.ly/2qUoYVb>





# CONTENT TRUST



# LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Identify the kinds of attacks and countermeasures considered by The Update Framework
- Describe the differences in Docker's behavior with content trust enabled versus disabled
- Establish content trust for a repository in DTR
- Demand UCP only runs images signed by a set of teams

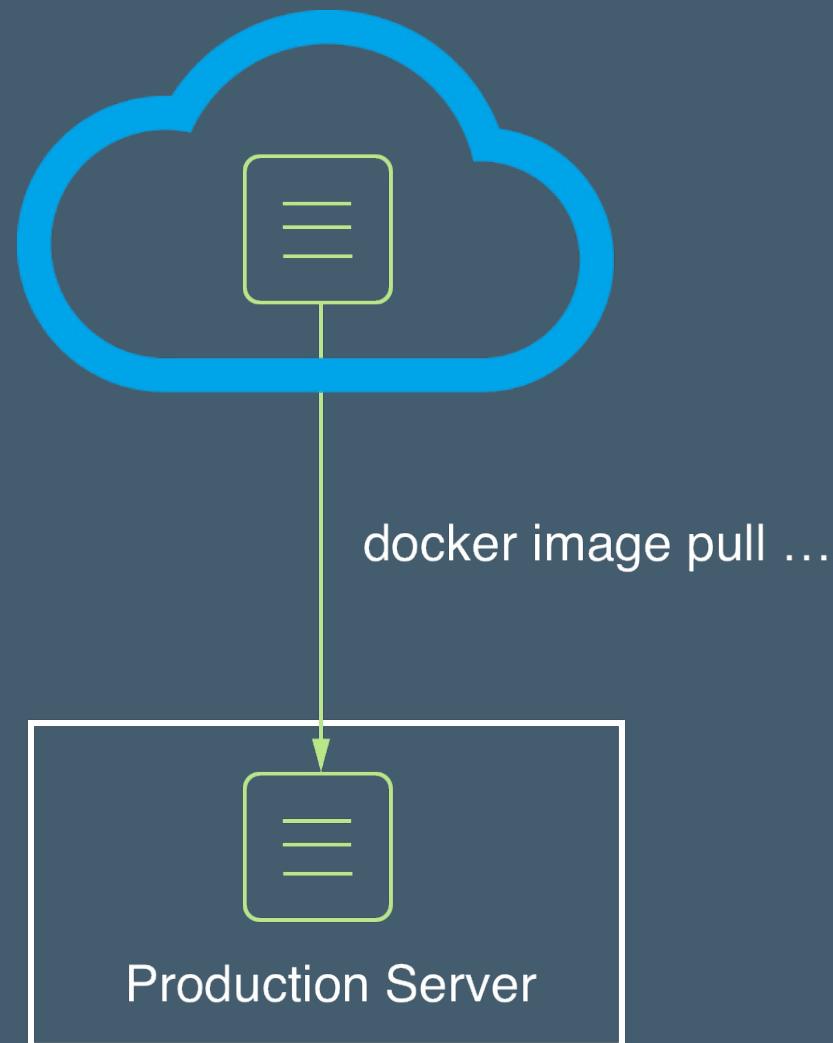


# WHAT IMAGE DO YOU MEAN?

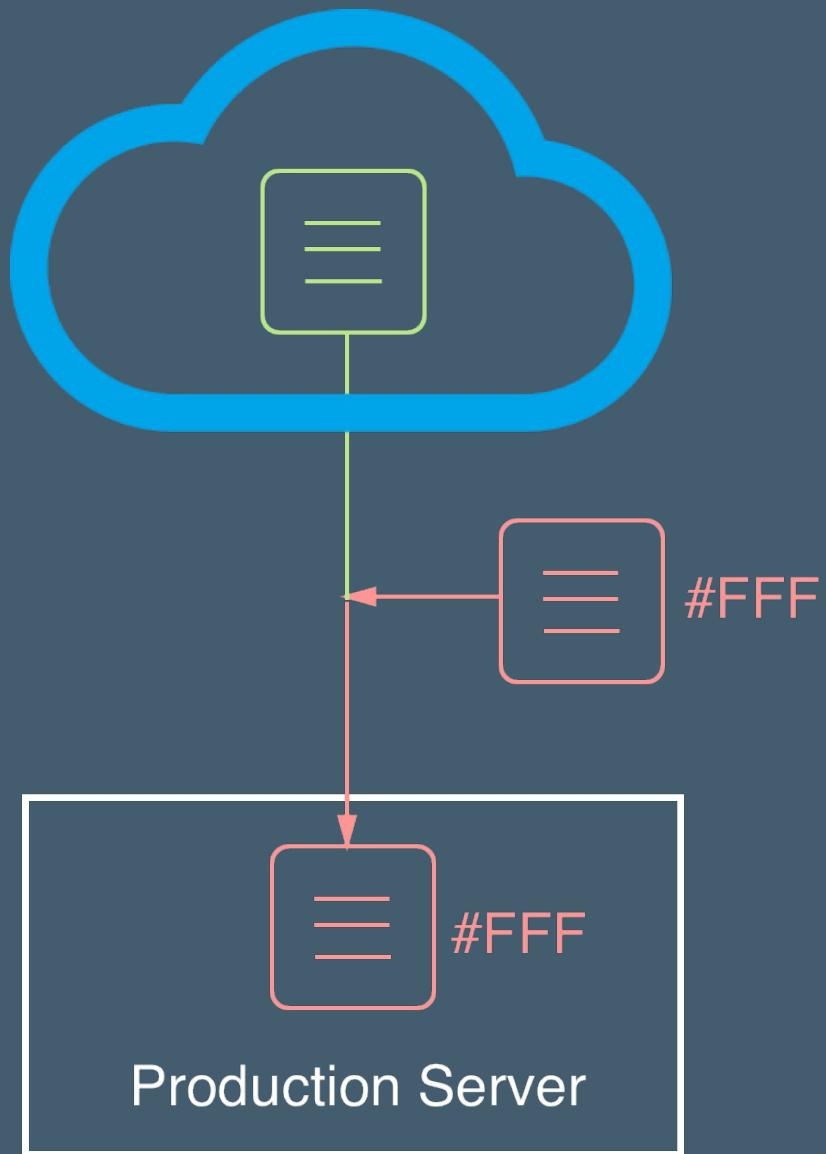
- Mutable: ref by tag: **alpine:latest**
- Immutable: ref by sha: **alpine@sha256:FFFF....**



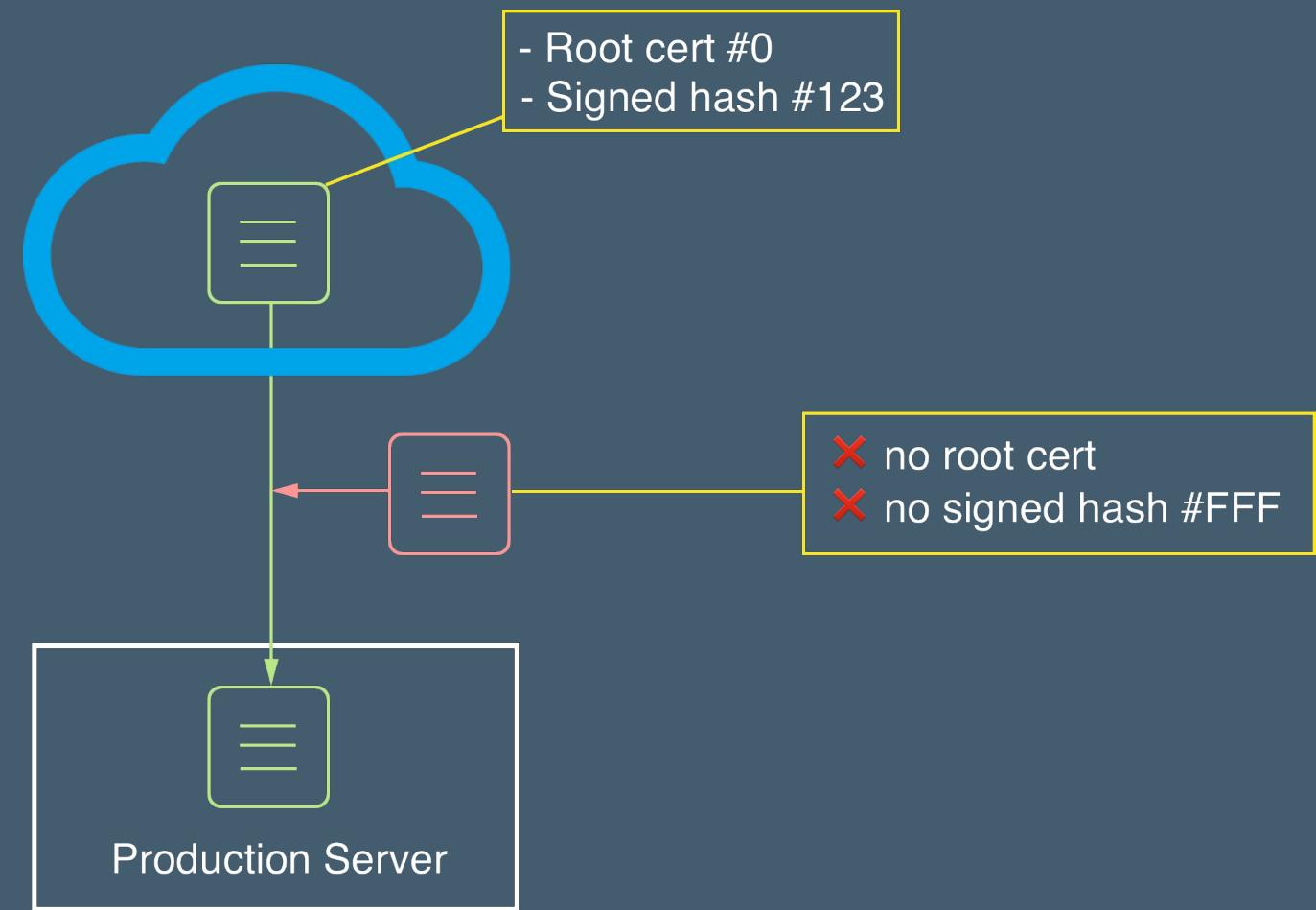
# DOWNLOADING SOFTWARE



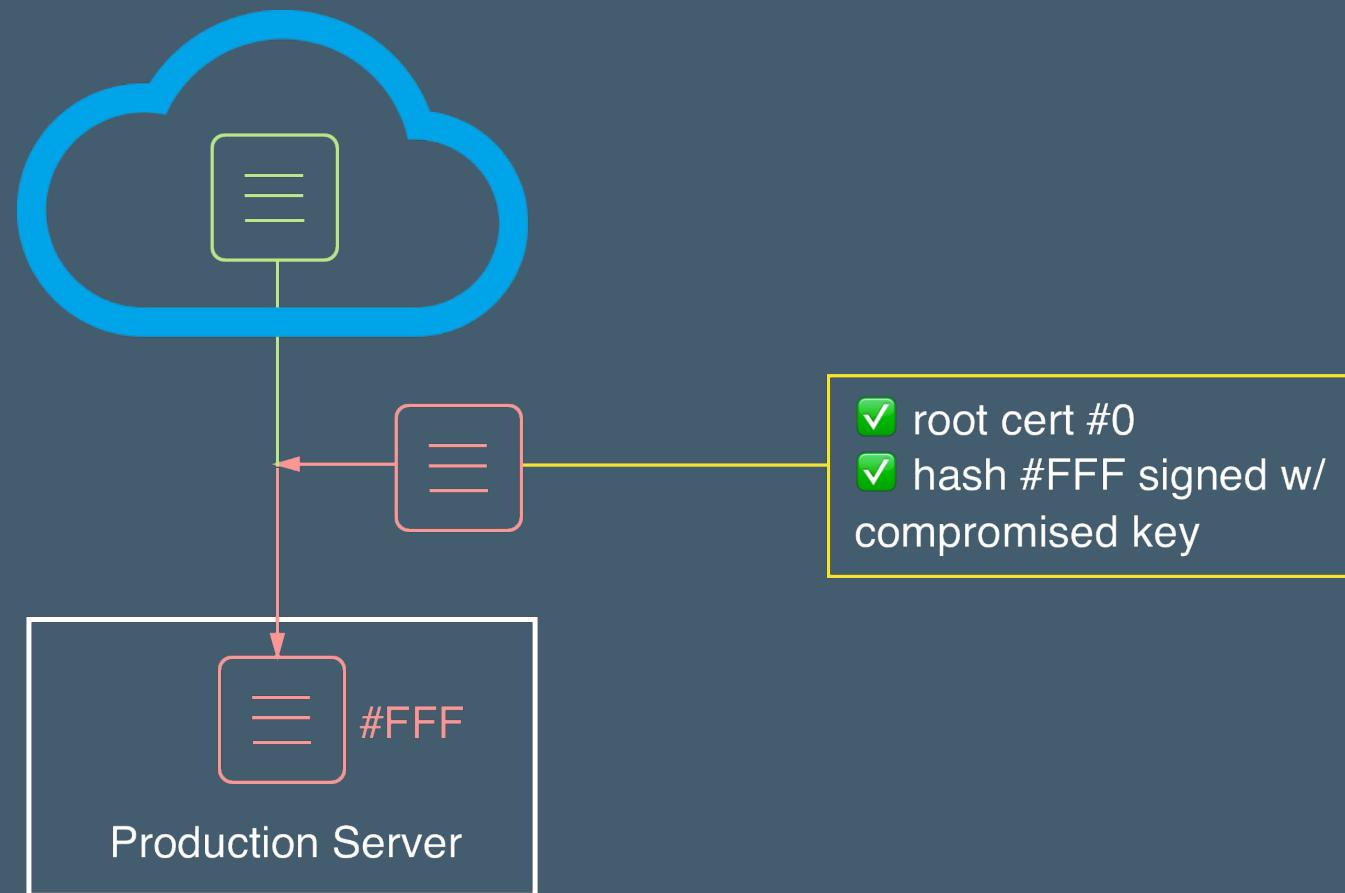
# SIMPLE INJECTION ATTACK



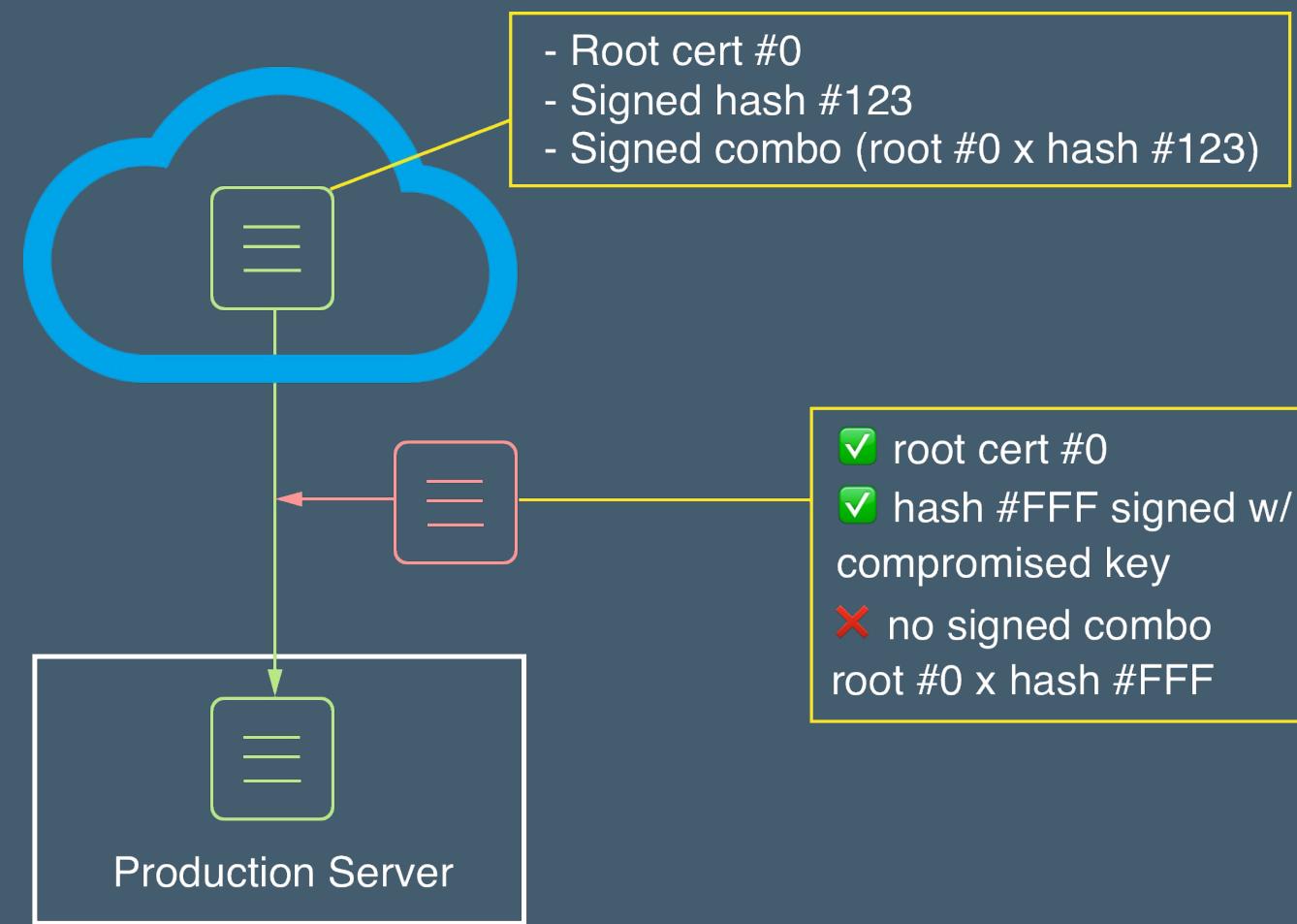
# SIGNED CONTENT



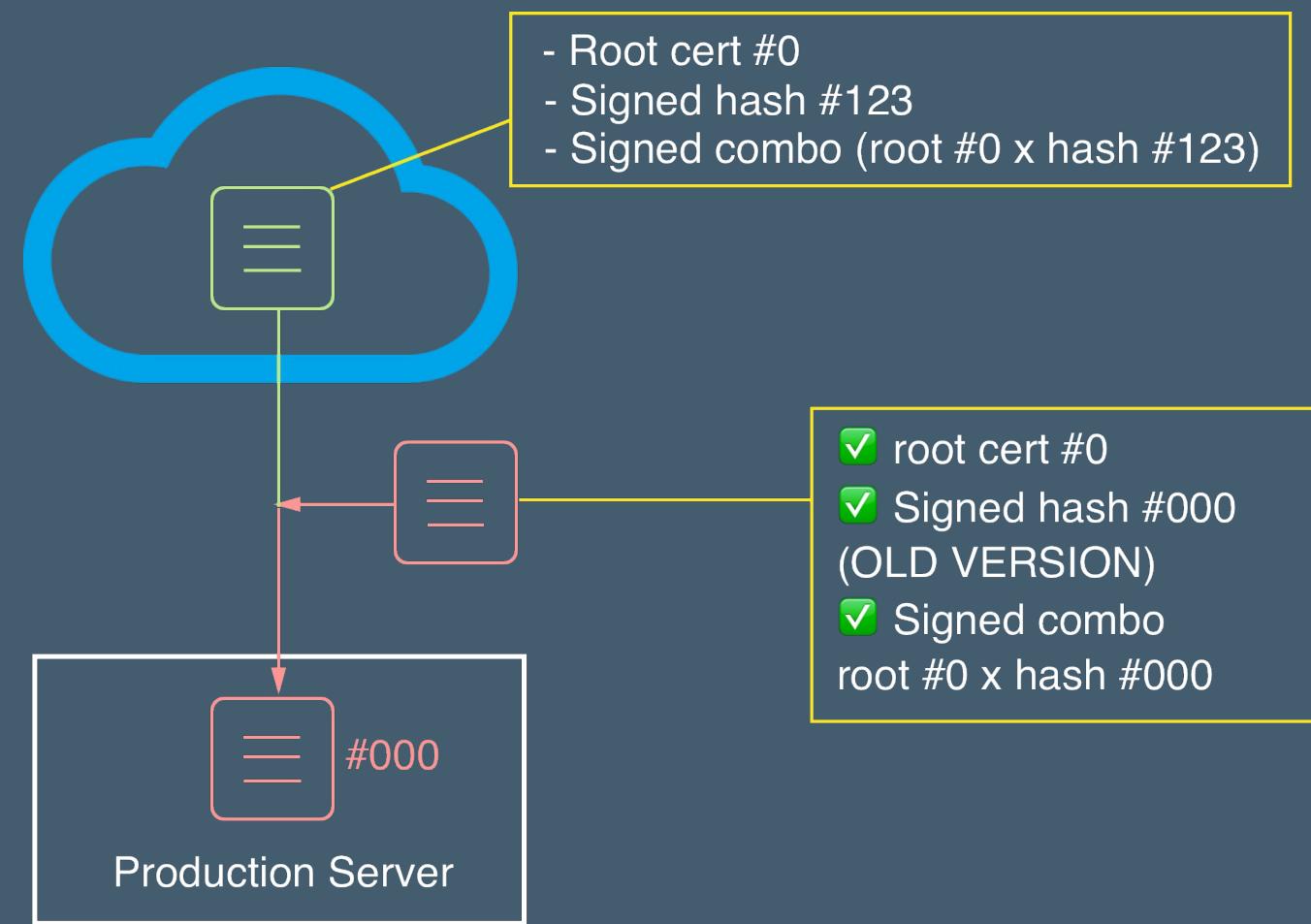
# MIX AND MATCH ATTACK



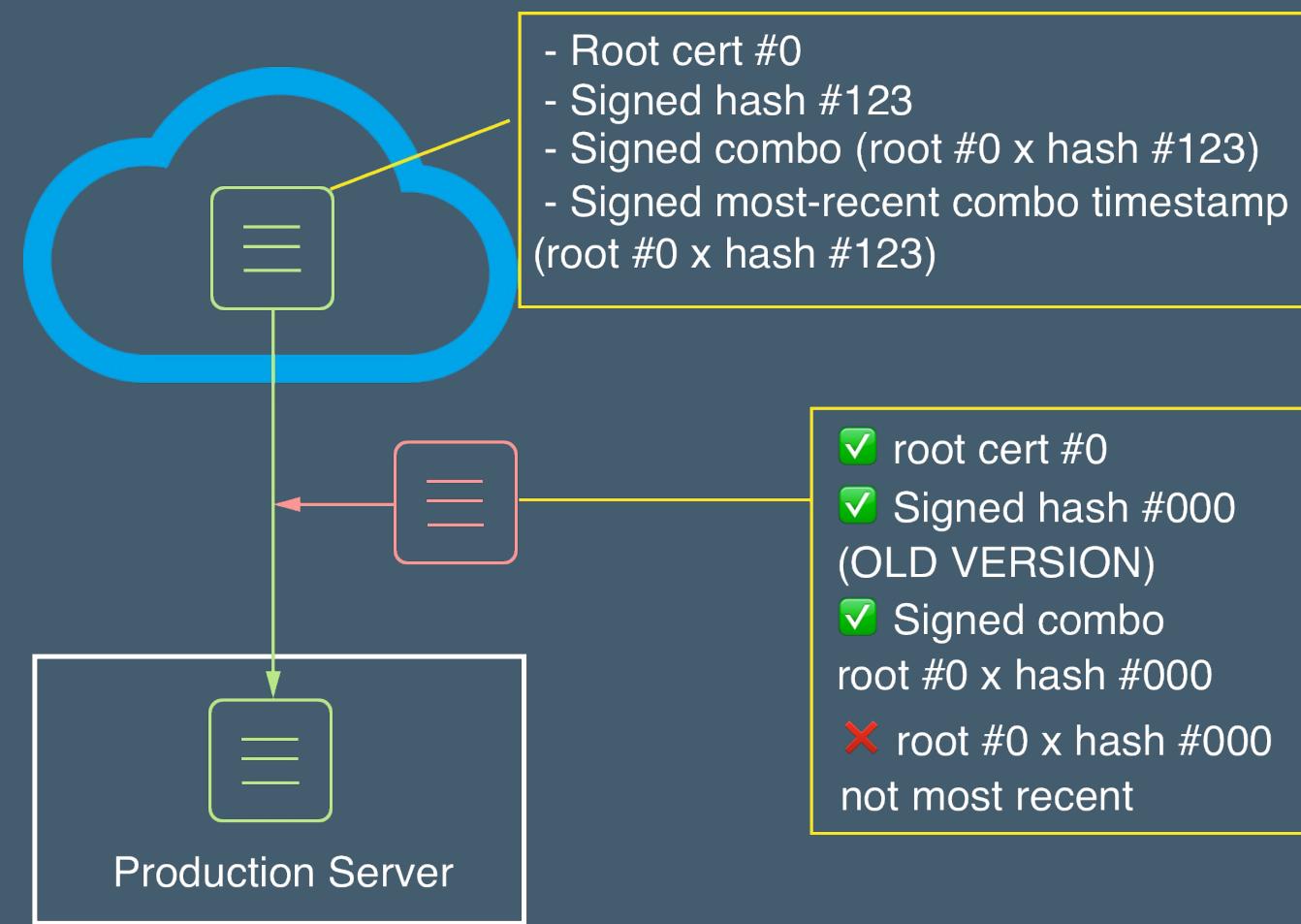
# METADATA VALIDATION



# DOWNGRADE ATTACK



# FRESHNESS GUARANTEED



# DOCKER CONTENT TRUST

- Image publishers sign their images
- Image consumers can ensure their images are signed
- Implements The Update Framework (TUF) into Docker as Notary
  - <http://theupdateframework.com/>
  - <https://github.com/docker/notary>



# CONTENT TRUST FOR IMAGE PUBLISHERS

- Metadata generated and signed on image push certifying authenticity
- Four (+1) different keys are used:
  - Root key: fundamental signing authority
  - Target key: repository content integrity
  - Snapshot key: metadata integrity
  - Timestamp key: repository freshness
  - Delegation key: multiple signing entities (optional)

(Jargon aside: the target key and snapshot key are sometimes collectively called the repository key, and the root key is sometimes called the offline key).



# CONTENT TRUST FOR IMAGE CONSUMERS

If content trust is enabled, only signed images are available for use with:

- docker image push
- docker image pull
- docker image build
- docker container create
- docker container run
- docker service create



# CONTENT TRUST IN DTR

- DTR comes with a built in Notary server
- Can enable trust directly from Docker Engine
- Optional: install Notary client side for finer-grained control



# INTEGRATION WITH UCP

- UCP can be configured to only run signed images
- Checked during application deployment
- Any unsigned image will be rejected
- Multiple signatures possible

Admin Settings

Swarm  
Certificates  
Routing Mesh  
Cluster Configuration  
Authentication & Authorization  
Logs  
License  
Docker Trusted Registry  
• Docker Content Trust

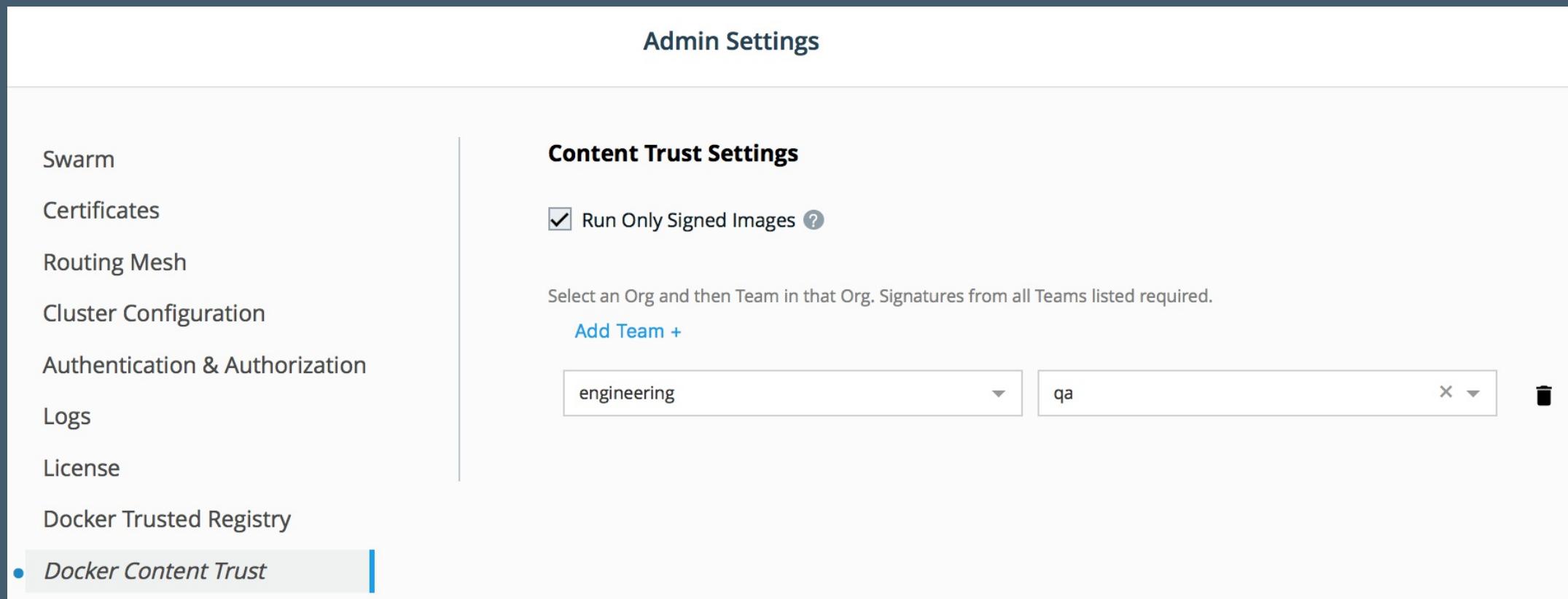
**Content Trust Settings**

Run Only Signed Images ?

Select an Org and then Team in that Org. Signatures from all Teams listed required.

Add Team +

engineering qa x ▼ trash





## EXERCISE: CONTENT TRUST

Work through the 'Content Trust' exercise in your exercise book.



# DISCUSSION

- What is one simple alternative to content trust?
- Questions?



# FURTHER READING

- Intro to content trust: <http://dockr.ly/2gAglo3>
- Features of Docker content trust: <http://dockr.ly/1EyCxrR>
- Automation with content trust: <http://dockr.ly/2x5lLaC>





# IMAGE SECURITY SCANNING



# LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Describe the process DTR takes to scan images for vulnerabilities
- Set up security scanning on DTR, and configure it for an individual repository
- Use DTR's UI to find out an image's vulnerabilities, and more information about that vulnerability



# WHAT'S IN YOUR IMAGES?

- Components installed in base layer
- Components installed in custom layers
- Known vulnerabilities?
- (Currently) unknown vulnerabilities?



# DTR SECURITY SCANNING FLOW

1. New layer pushed to DTR
2. Scan triggered (auto or manual)
3. Bill-of-materials generated and saved (**slow, resource intensive**)
4. BoM re-checked against CVE database every time the db is updated (**fast**)
5. Regular database updates from <https://dss-cve-updates.docker.com/>



# SCAN RESULTS PER LAYER

engineering/engineering-app private : latest

linux/amd64 1eb35305b7 47.15 MB Pushed 10 minutes ago by pixel 6 critical 10 major 14 minor All layers already scanned

[Layers](#) [Components](#) [Delete](#) [Promote](#) [Scan](#)

Layer	Content	Actions
1 ADD	file:96db69a1ba6c80f604d07b14bcbf84445624ad3eb5b0471eddabf09cb7925366 in /	<a href="#">show</a>
2 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 && dpkg-divert --local --rename --add /sbin/initctl && cp -a /usr/sbin/policy-rc.d	47.15 MB	<a href="#">show</a>
3 rm -rf /var/lib/apt/lists/*		

**ADD**  
file:96db69a1ba6c80f604d07b14bcbf84445624ad3eb5b0471eddabf09cb7925366 in /

**47.15 MB** [show](#)

**COMPONENTS (46)** **VULNERABILITIES (30) ▾**

**zlib 1.2.8.dfsg-2ubuntu4.1** **2 critical 2 major**

**ncurses 6.0+20160213-1ubuntu1** **2 critical 2 major**



# SCAN RESULTS PER COMPONENT

 engineering/engineer/app private : latest

 linux/amd64 1eb35305b7 47.15 MB ⏲ Pushed 11 minutes ago by pixel ! 6 critical 10 major 14 minor All layers already scanned

[Layers](#) [Components](#) [Delete](#) [Promote](#) [Scan](#)

Component	Version	License	Vulnerabilities
<b>zlib</b> 1.2.8.dfsg-2ubuntu4.1 <span style="color: red;">2 critical 2 major</span>	<b>zlib</b> 1.2.8.dfsg-2ubuntu4.1	 <span style="background-color: #5b9bd5; color: white; border: 1px solid #5b9bd5; padding: 2px;">PERMISSIVE</span>	<a href="#">VULNERABILITIES</a> <a href="#">SEVERITY ▾</a> <a href="#">DESCRIPTION</a>
<b>ncurses</b> 6.0+20160213-1ubuntu1 <span style="color: red;">2 critical 2 major</span>			
<b>systemd</b> 229-4ubuntu17 <span style="color: red;">1 critical 2 major</span>			

**zlib**

VERSION: 1.2.8.dfsg-2ubuntu4.1 | LICENSE: zlib  PERMISSIVE

VULNERABILITIES | SEVERITY ▾ | DESCRIPTION

[CVE-2016-9841](#) critical  inffast.c in zlib 1.2.8 might allow context-dependent attackers to have unspecified impact by leveraging improper pointer arithmetic.

[CVE-2016-9843](#) critical  The crc32\_big function in crc32.c in zlib 1.2.8 might allow context-



# VULNERABILITY OVERRIDES

pdevine/ubuntu: latest private

linux / amd64 56e96244fd 47.61 MB Pushed 20 hours ago by admin 12 critical 37 major 4 minor 1 hidden All layers already scanned

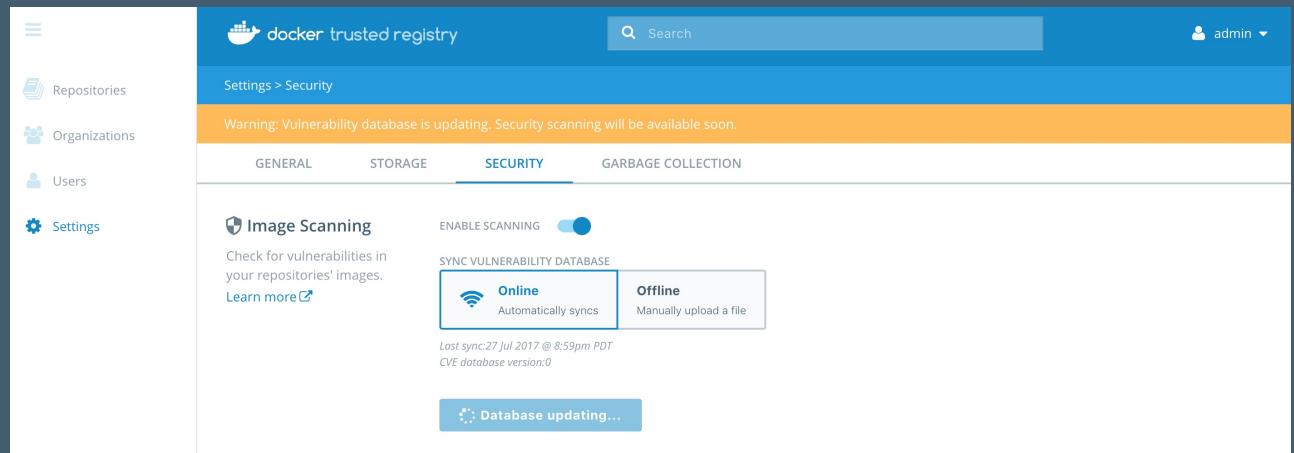
Layers Components Delete Promote Scan

Component	Vulnerability ID	Severity	Description	Action
glibc 2.23-0ubuntu3	CVE-2017-0323	major	The <code>makecontext</code> function in the GNU C Library (aka glibc or libc6) before 2.25 creates execution contexts incompatible with the...	hide
systemd 229-4ubuntu6	CVE-2015-5180	major	<code>res_query</code> in <code>libresolv</code> in glibc before 2.25 allows remote attackers to cause a denial of service (NULL pointer dereference and proce...	hide
ncurses 6.0+20160213-1ubuntu1	CVE-2017-15671	major	The <code>glob</code> function in <code>glob.c</code> in the GNU C Library (aka glibc or libc6) before 2.27, when invoked with <code>GLOB_TILDE</code> , could skip freeing...	hide
zlib 1.2.8.dfsg-2ubuntu4	CVE-2017-12133	major	The DNS stub resolver in the GNU C Library (glibc) before version 2.26, when EDNS support is enabled, will solicit large UDP...	hide
	CVE-2016-10228	major	The <code>iconv</code> program in the GNU C Library (aka glibc or libc6) 2.25 and earlier, when invoked with the <code>-c</code> option, enters an infinite lo...	show
	CVE-2017-12132	major	The DNS stub resolver in the GNU C Library (aka glibc or libc6) before version 2.26, when EDNS support is enabled, will solicit...	hide

- Allows vulnerabilities to be hidden
- Matches based on different components



# CVE DATABASE UPDATES



- DTR -> System -> Security
- Automatic updates: daily 3 AM UTC; must be able to reach <https://dss-cve-updates.docker.com/> on port 443.
- Manual updates: uploaded through DTR, downloaded through store.docker.com.



# SCANNING AUTOMATION

Repositories > engineering/enterprise-app > Settings

**General**

VISIBILITY

**Public**  
Visible to everyone

**Private**  
Hide this repository

IMMUTABILITY

**On**  
Tags are immutable

**Off**  
Tags can be overwritten

DESCRIPTION

**Image scanning**

Check for vulnerabilities in your images.

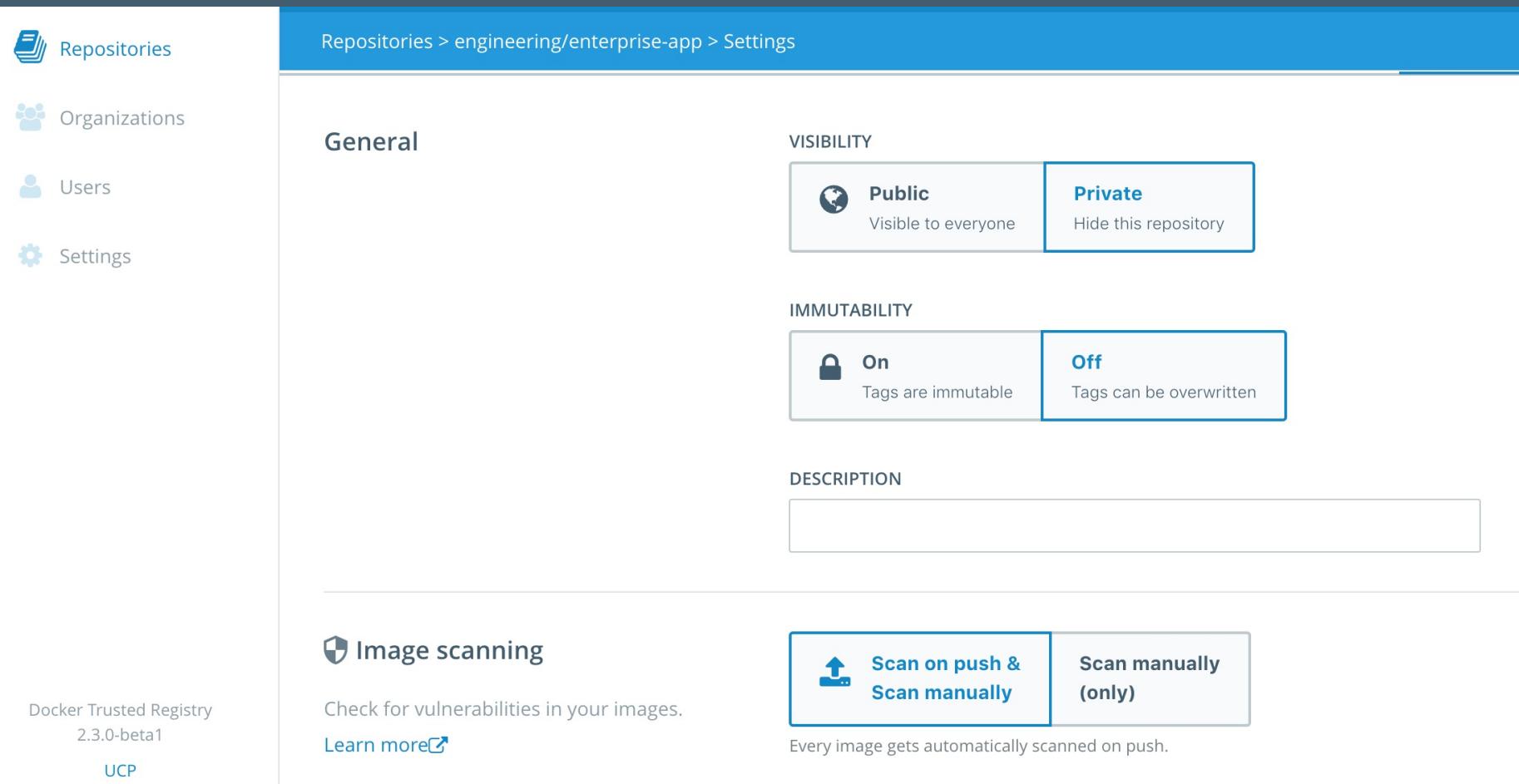
[Learn more](#)

**Scan on push & Scan manually**

Every image gets automatically scanned on push.

**Scan manually (only)**

Docker Trusted Registry  
2.3.0-beta1  
UCP



# COMMON SCANNING MISTAKES

- Make sure initial CVE database is downloaded before starting scans
- Make sure all DTR replicas can reach storage backend
- Consider manual-only scans as part of pipeline





## EXERCISE: IMAGE SCANNING

Work through the 'Image Scanning in DTR' exercise in your exercise book.



# DISCUSSION

- Scanning identifies a vulnerability. What are some generic steps to mitigate?
- Questions?



## FURTHER READING

- Set up security scanning in DTR: <https://dockr.ly/2HMgp9d>
- Scan images for vulnerabilities: <https://dockr.ly/2HNOdCK>





# REPOSITORY AUTOMATION



# LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Automatically retag an image from one DTR repo to another
- Define webhooks triggered by DTR events
- Integrate DTR into a CI/CD chain using the above

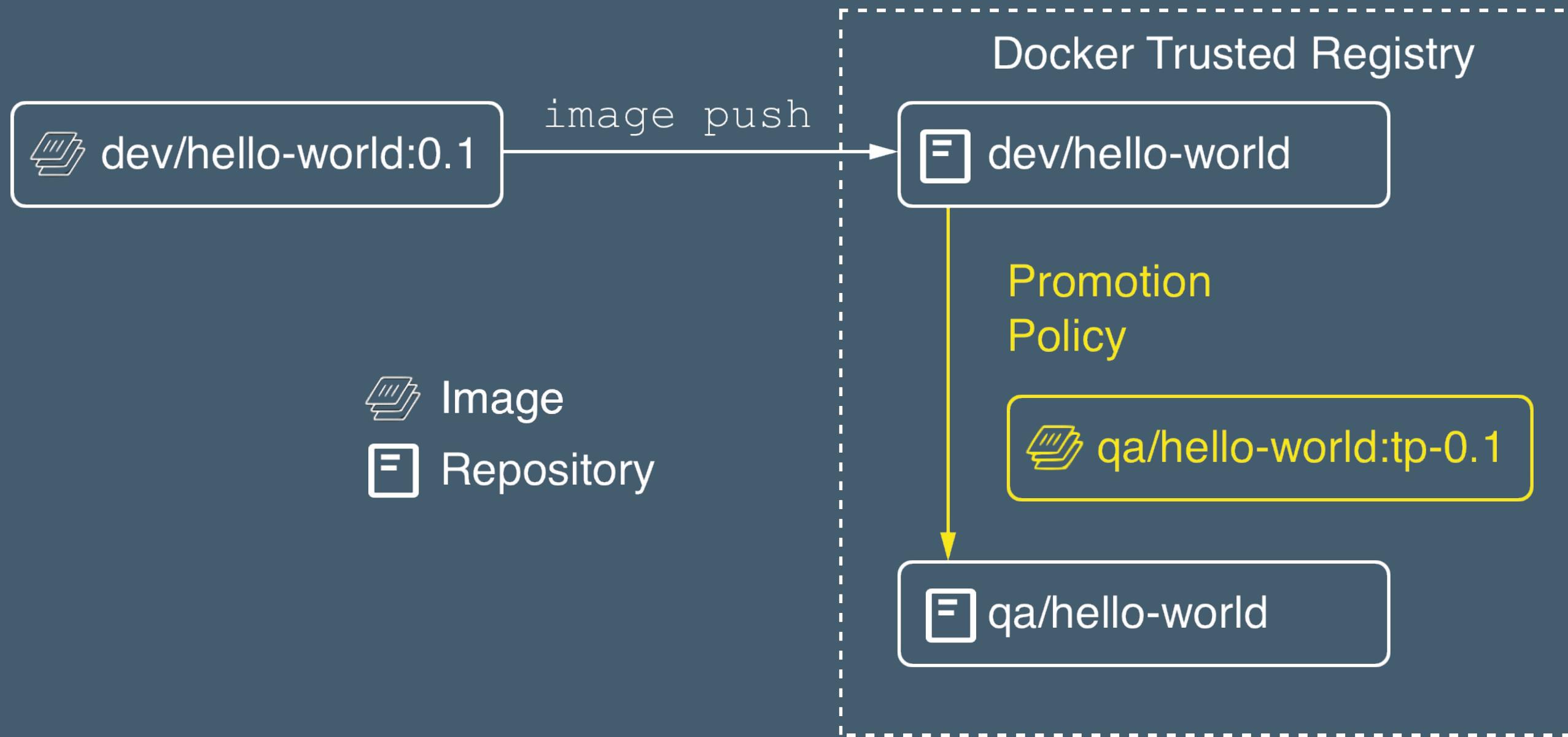


# AUTOMATION TOOLS

- Image Promotion & Mirroring
- Webhooks



# IMAGE PROMOTION

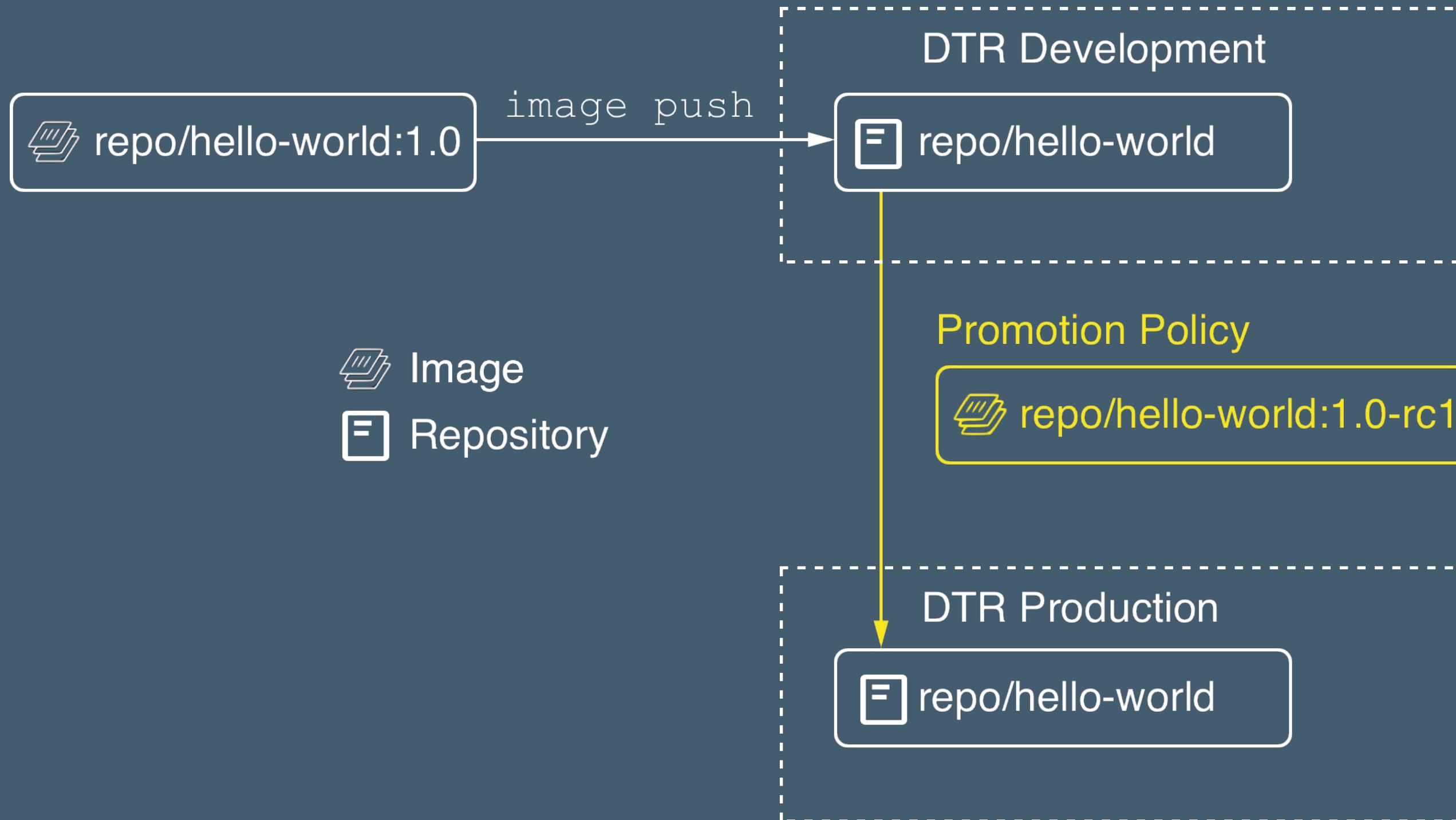


# IMAGE PROMOTION POLICIES

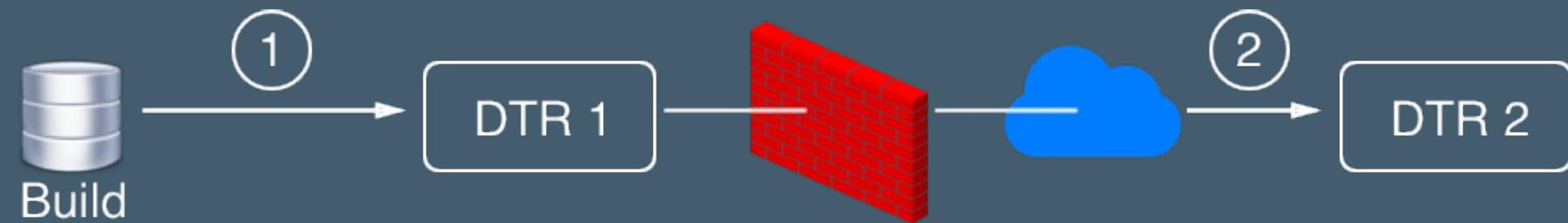
- Manual or automatic
- Automatic promotion can be triggered on:
  - Tag name
  - Package name
  - Minor / Major / Critical / All vulnerabilities
  - Component licenses
- No limit to number of policies that can be defined



# IMAGE MIRRORING



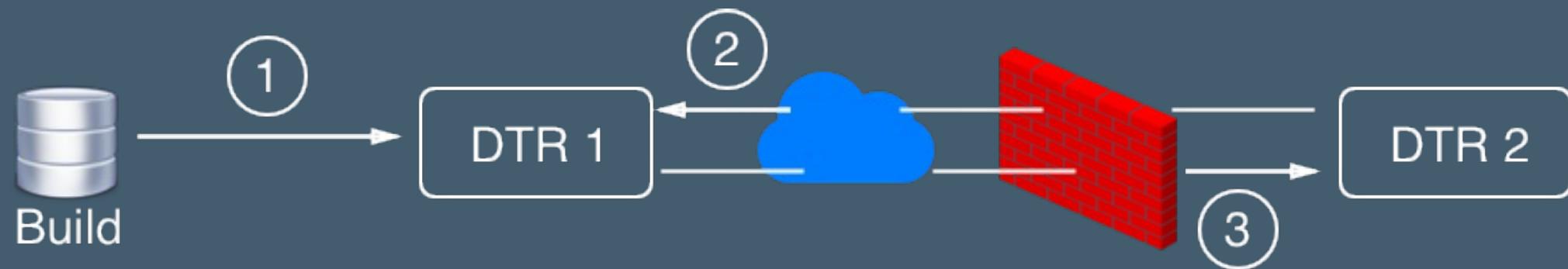
# IMAGE MIRRORING - PUSH BASED



- Image pushed to DTR 1
- If policies met => push to DTR 2
- AuthN & AuthZ managed by each DTR
- Signing & scan data not (yet) preserved



# IMAGE MIRRORING - PULL BASED



- Image pushed to DTR 1
- DTR 2 polls DTR 1 for updates
- New image found => pull to DTR 2
- Combine with Promotion Policies



# WEBHOOKS

POST message with JSON payload, triggered on:

- Tag push or delete
- Manifest push or delete
- Security scan failed
- Security scan complete

Defined per repository.



# WEBHOOK PAYLOAD

- Webhook payloads always come in a wrapper:

```
{  
  "type": "...",  
  "createdAt": "2012-04-23T18:25:43.511Z",  
  "contents": {...}  
}
```

- The **contents** key depends on the event type; see <https://dockr.ly/2JjcAW7> for the full spec.





## EXERCISE: REPOSITORY AUTOMATION

Work through the 'Image Promotion & Webhooks' exercise in your exercise book.



# DISCUSSION

- What are the pros and cons of promoting images in a single DTR, versus mirroring them across multiple DTRs?
- Questions?



## FURTHER READING

- Managing webhooks: <https://dockr.ly/2JjcAW7>
- Promotion policies overview: <https://dockr.ly/2KarnDN>
- Image Promotions and Immutable Repos: <http://bit.ly/2eEz7TH>





# IMAGE CACHING



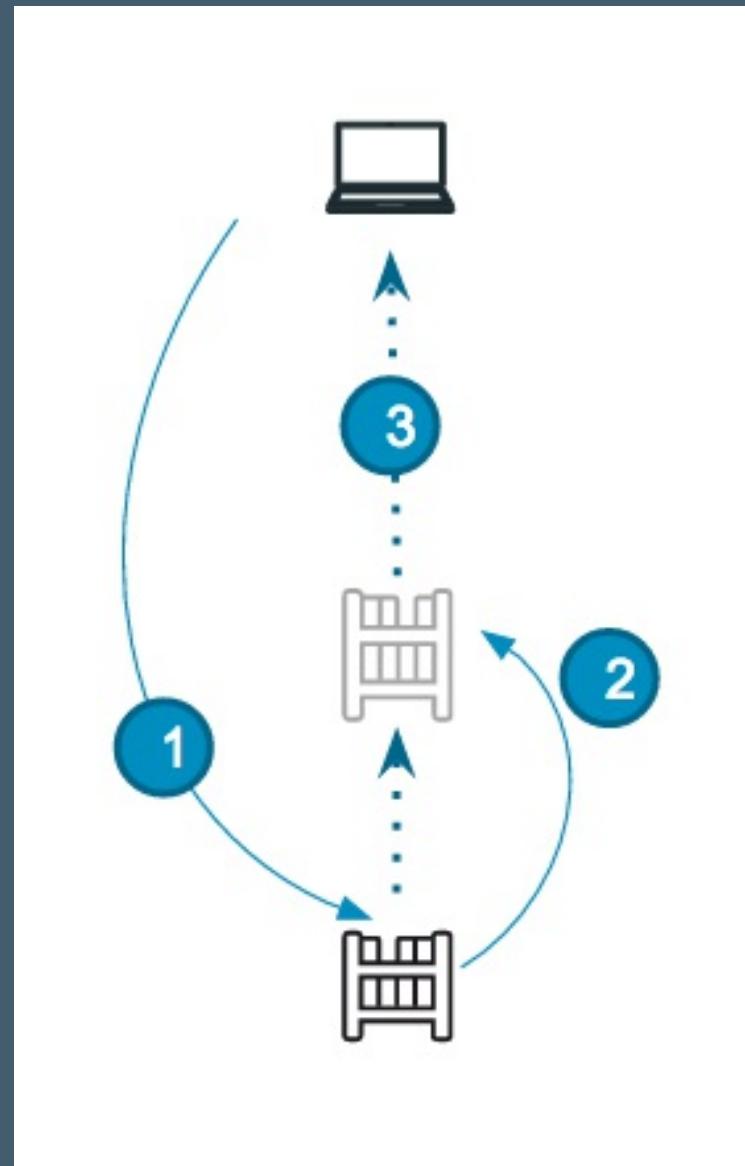
# LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Configure a remote image cache for DTR



# IMAGE CACHE WORKFLOW



- Client requests an image from DTR
- DTR authenticates the request
- Request redirected to user-selected cache
- Pulls from cache if available
- Cache missing -> pull from DTR



# SETTING UP A CACHE

1. Write config file
2. Deploy cache as container or service
3. Register cache in DTR



# CACHE CONFIG

- Configured via **config.yml** file:

```
version: 0.1
storage:
  delete:
    enabled: true
  filesystem:
    rootdirectory: /var/lib/registry
http:
  addr: :5000
middleware:
  registry:
    - name: downstream
      options:
        blobttl: 24h
        upstreams:
          - https://<dtr-url>
      cas:
        - /certs/dtr-ca.pem
```

(See <http://dockr.ly/2FWzKRA> for more details and config examples)



# DEPLOY CACHE

1. Download the CA certificate used by DTR:

```
curl -k https://<DTR_FQDN>/ca > dtr-ca.pem
```

2. Run Docker content cache container

```
docker container run --detach --restart always \  
--name content-cache \  
--publish 5000:5000 \  
--volume $(pwd)/dtr-ca.pem:/certs/dtr-ca.pem \  
--volume $(pwd)/config.yml:/config.yml \  
docker/dtr-content-cache:<version> /config.yml
```



# REGISTER CACHE IN DTR

- In DTR, click API
- Expand `/api/v0/content_caches` POST form
- Click Try Me Out!
- Specify the name and host for your cache:

```
{  
  "name": "region-us",  
  "host": "http://<cache-public-ip>:5000"  
}
```

- Click Execute to fire the API call and register the cache.



# USER-SELECTED CACHES

DTR user settings to select content cache

The screenshot shows the Docker Trusted Registry (DTR) user settings interface. On the left sidebar, there are links for Repositories, Organizations, Users, System, API, Docs, and Support. At the bottom of the sidebar, it says "Docker Trusted Registry 2.5.0-beta3" and "Universal Control Plane". The main content area shows the "User Password" section with a "NEW PASSWORD" input field and a "Save" button. Below this is the "Content cache" section, which contains the text "For faster pull times, choose a content cache close to you." and a "REGION" dropdown menu. The dropdown menu has three options: "local" and "region-us", with "region-us" currently selected. A red box highlights the "Content cache" section. At the bottom of the page, there is a "Delete" link with a "Delete" button.

docker trusted registry

Search

admin

Users > cheeto > Settings

Save

User Password

NEW PASSWORD

Save

Content cache

For faster pull times, choose a content cache close to you.

REGION

local

region-us

Save

Delete

Delete user (this cannot be undone)



# CACHE TOPOLOGY

- Cache chaining possible
- Don't chain caches more than two deep
- Do run the cache as an HA service



## FURTHER READING

- DTR cache overview: <https://dockr.ly/2HkS8Uh>
- Use a cache: <https://dockr.ly/2vGzHli>

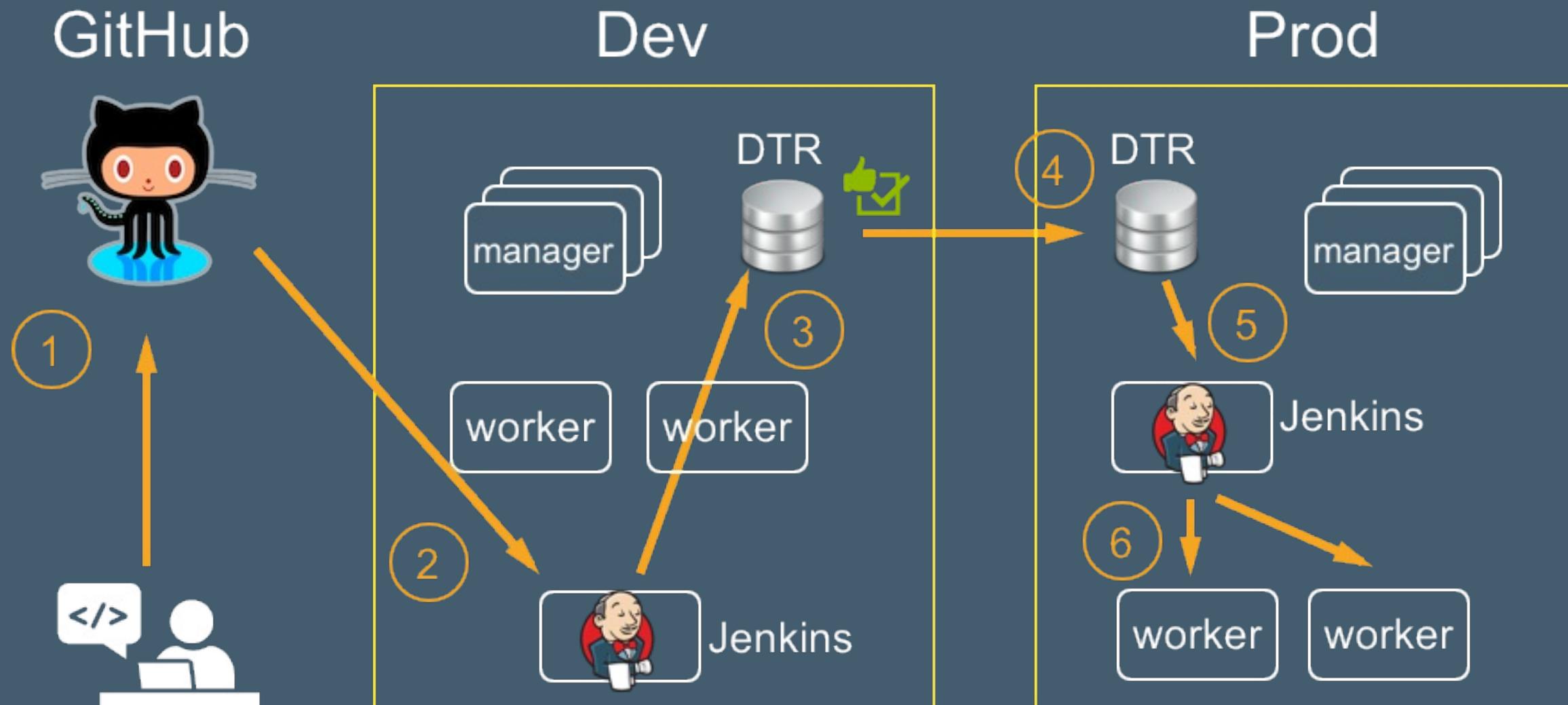




# OPERATIONS SIGNATURE ASSIGNMENT



# THE USE CASE





## SIGNATURE ASSIGNMENT: THE SOFTWARE SUPPLY CHAIN

Work through the 'The Software Supply Chain' exercise in your exercise book.



# DOCKER FOR ENTERPRISE OPERATIONS

Thanks for coming! Please take one of our feedback surveys:

- Docker for Enterprise Operations (standalone): <http://bit.ly/2FiYZ0b>
- Docker Fundamentals + Enterprise Ops (combined class): <https://bit.ly/2J1ryiT>

Get in touch: [training@docker.com](mailto:training@docker.com)

[training.docker.com](https://training.docker.com)



# YOU'RE ON YOUR WAY TO BECOMING DOCKER CERTIFIED!

- Study up with our Study Guides at <http://bit.ly/2yPzAdb>
- Take it online 24 hours a day
- Results delivered immediately
- Benefits include:
  - Digital certificate
  - Online verification
  - Private LinkedIn group
  - Exclusive events

**SUCCESS.DOCKER.COM/CERTIFICATION**

