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/2mTQr8l
- 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.

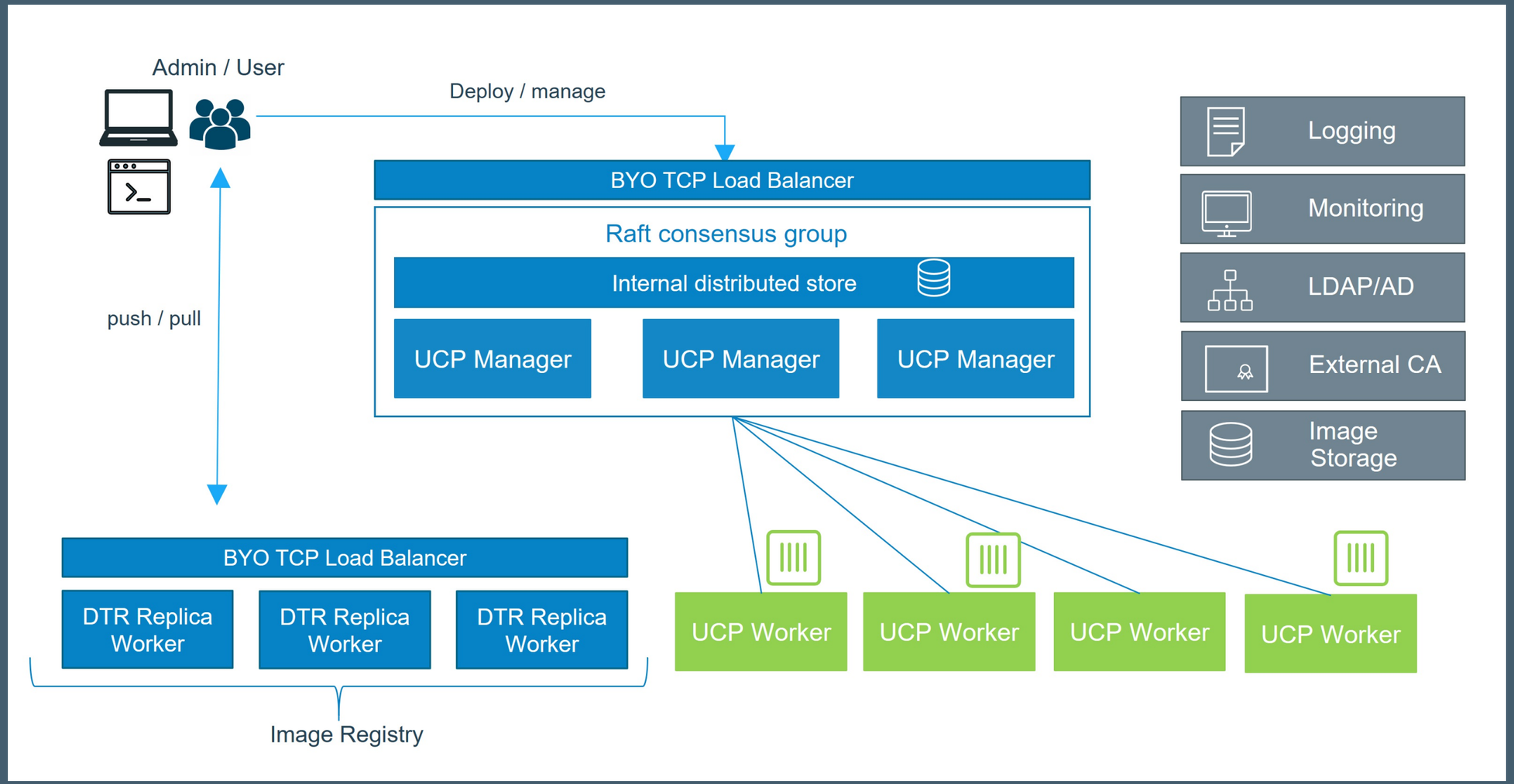
KEY EE FEATURES

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

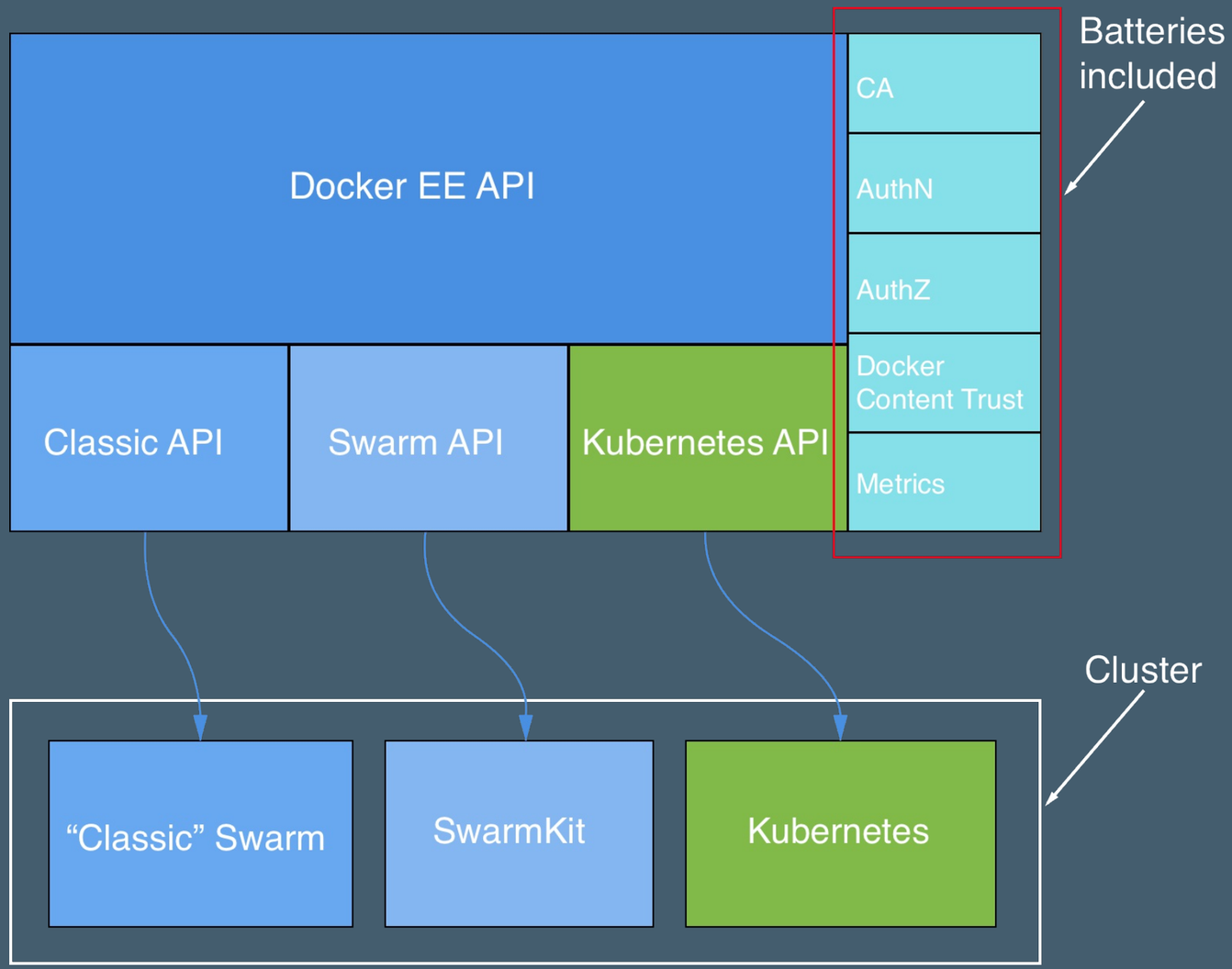
INTEGRATION: BATTERIES INCLUDED BUT SWAPPABLE

- 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

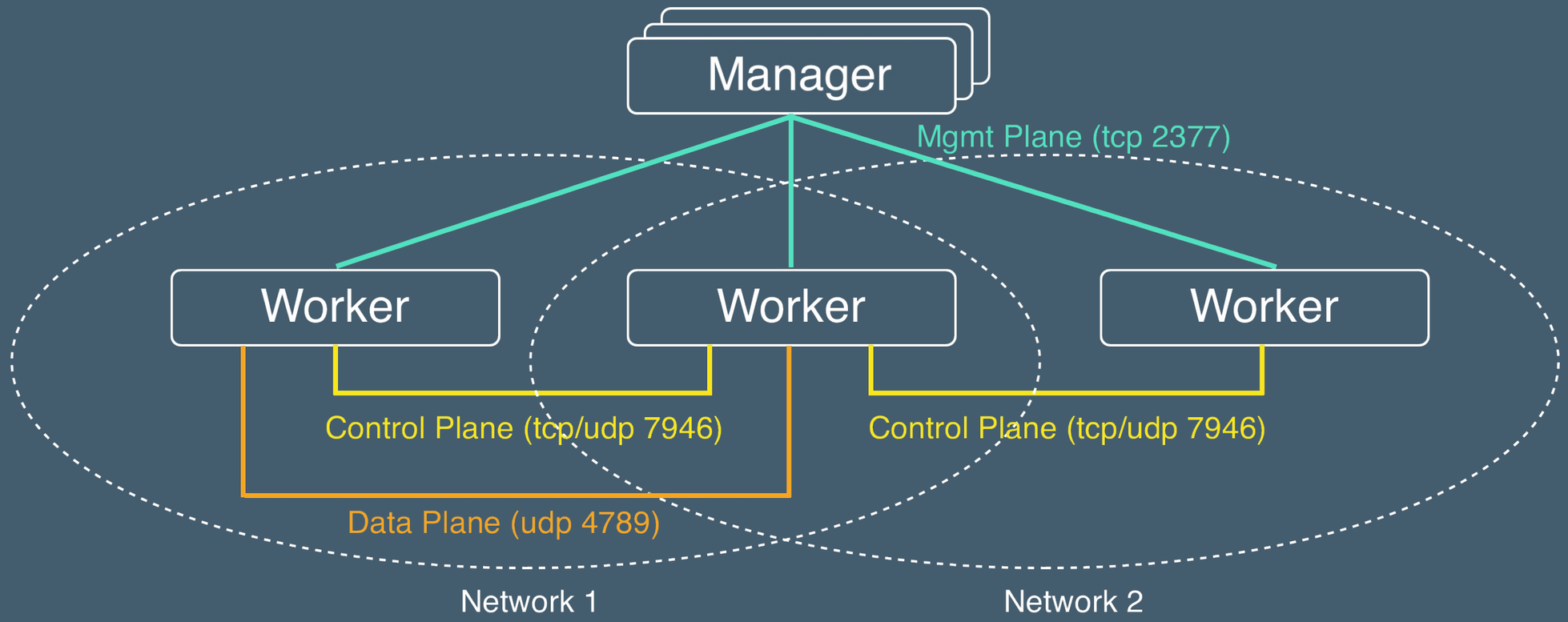
By the end of this module, learners will be able to:

- 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

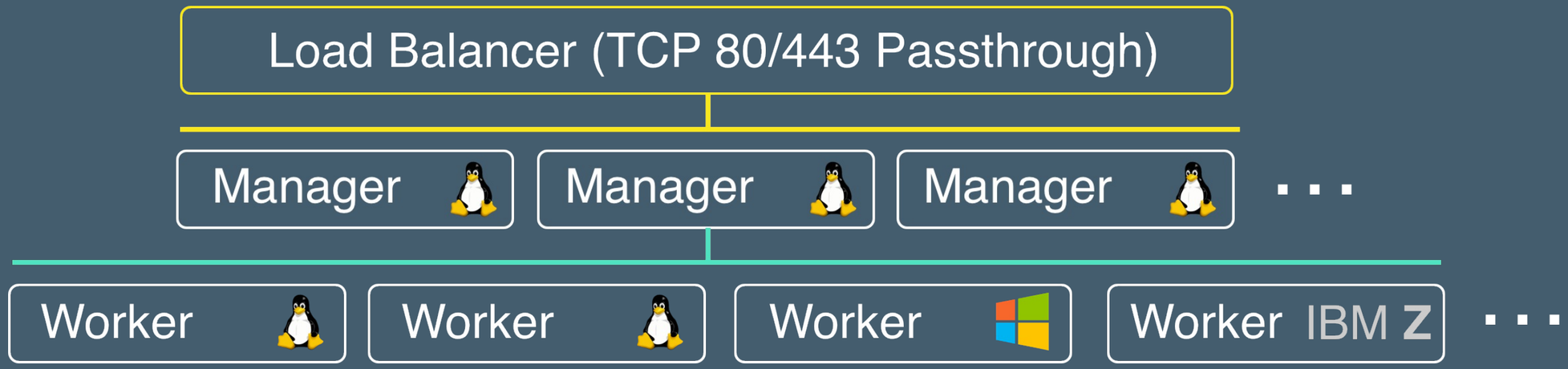
WHAT IS UCP?

- 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

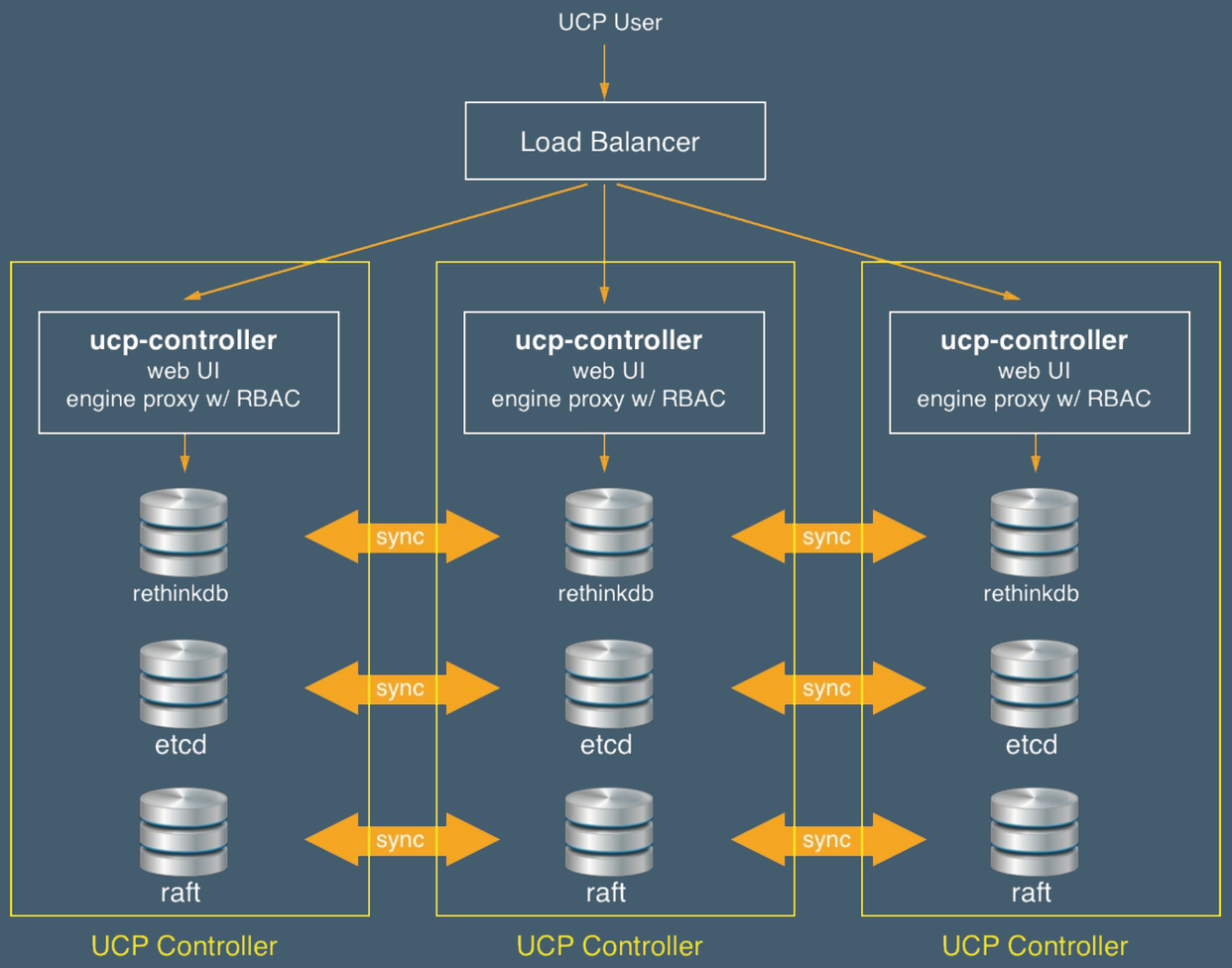


CLUSTER CONFIGURATION

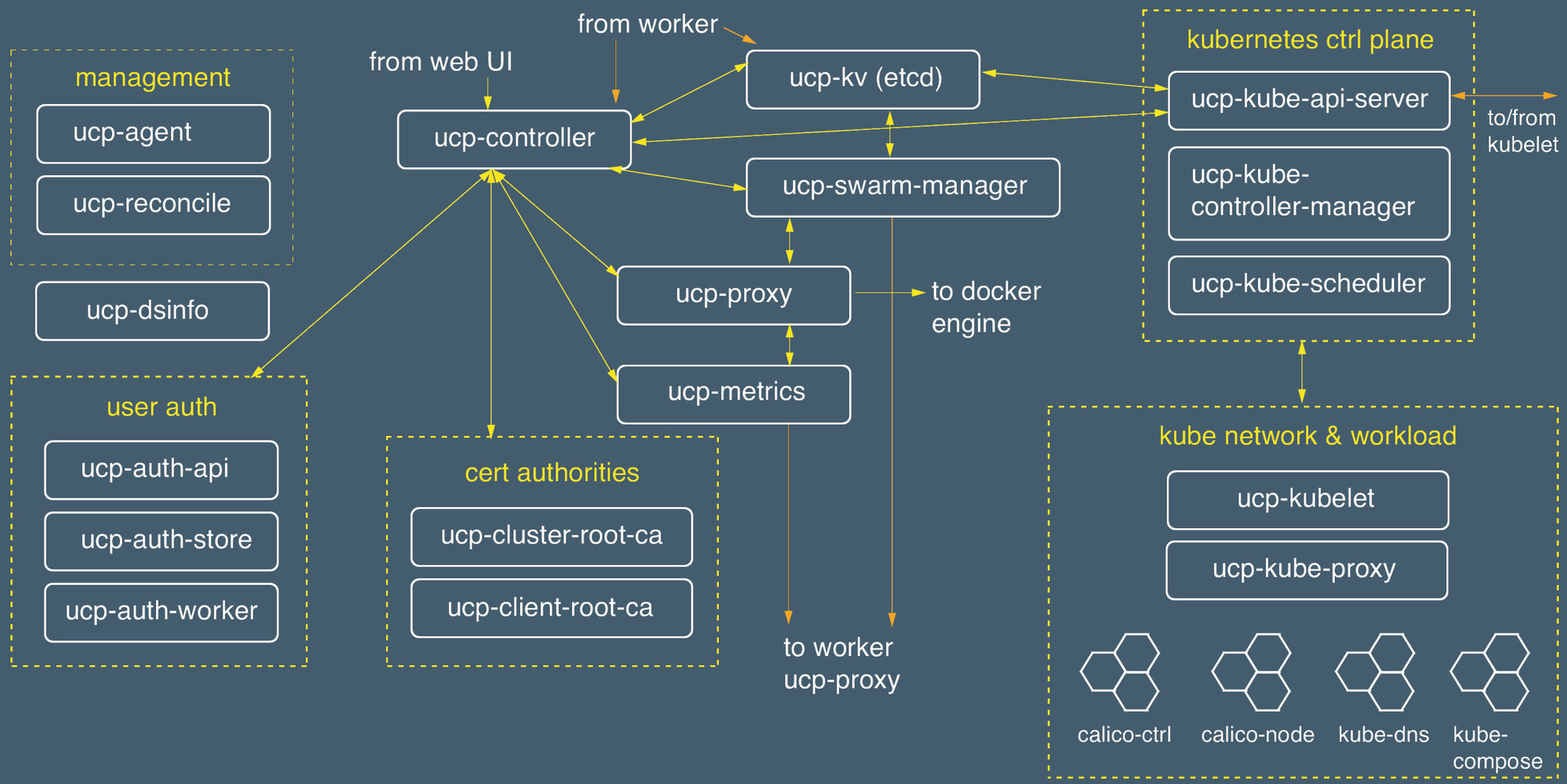


- Odd no. of managers
- Don't run workload on managers
- Don't terminate HTTPS in manager LB
- See /_ping for manager health
- See system requirements at http://dockr.ly/2DnYFPa

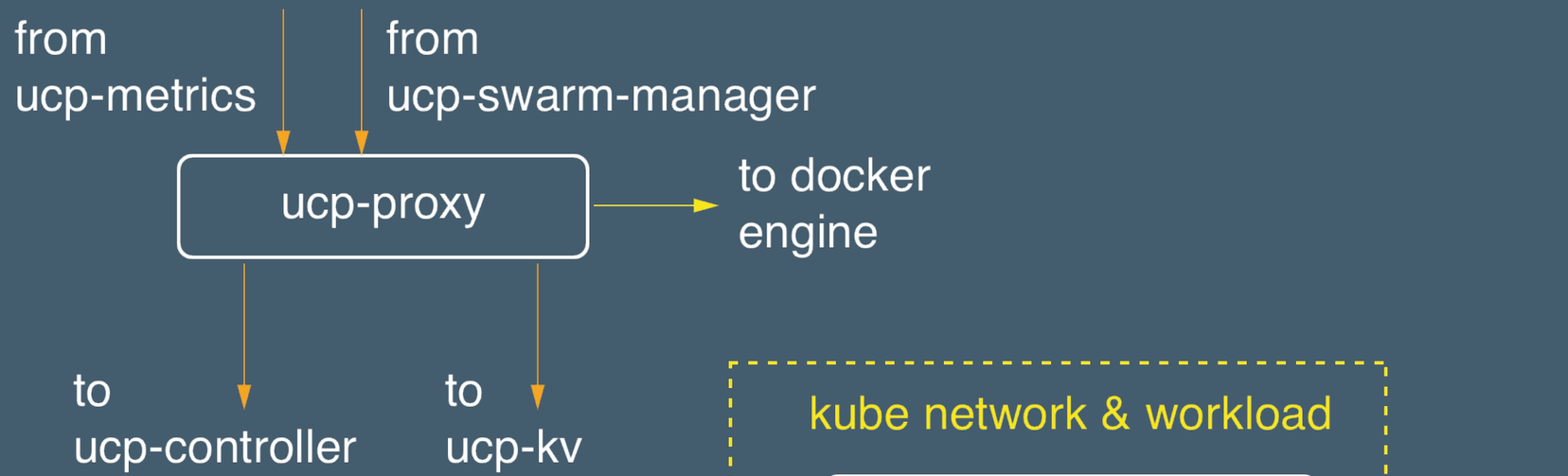
CLUSTER SYNCHRONIZATION

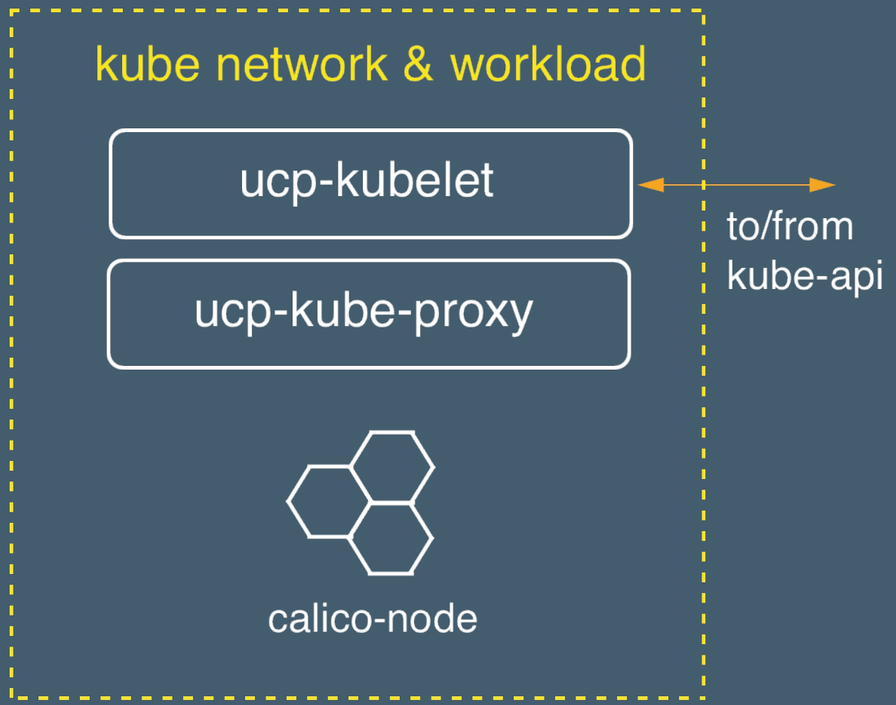


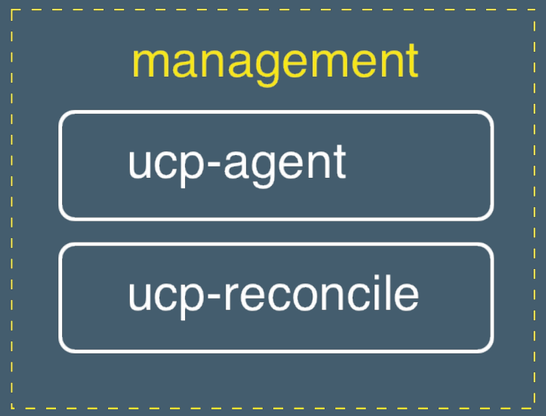
UCP MANAGER ARCHITECTURE OVERVIEW



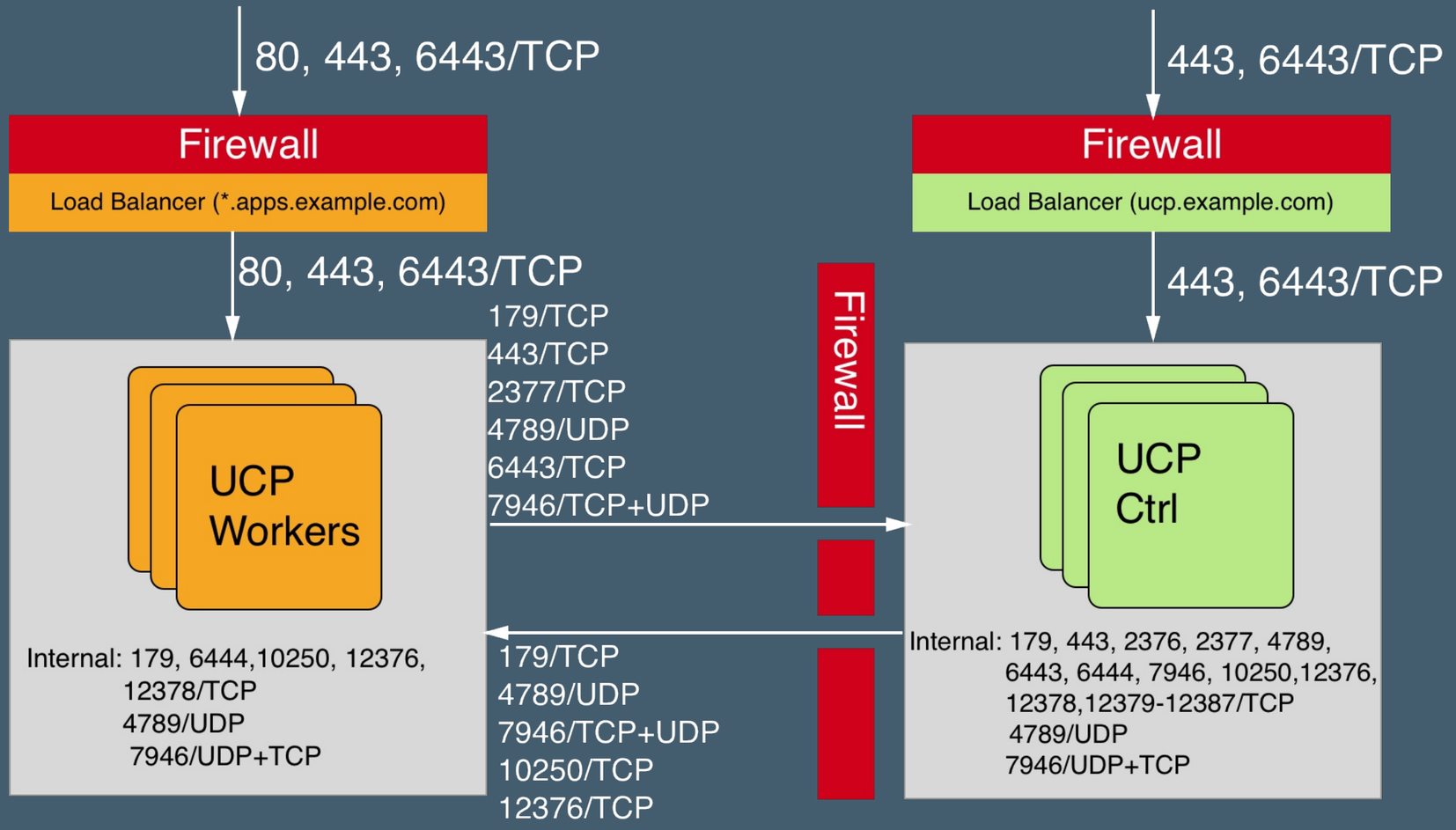
UCP WORKER ARCHITECTURE OVERVIEW



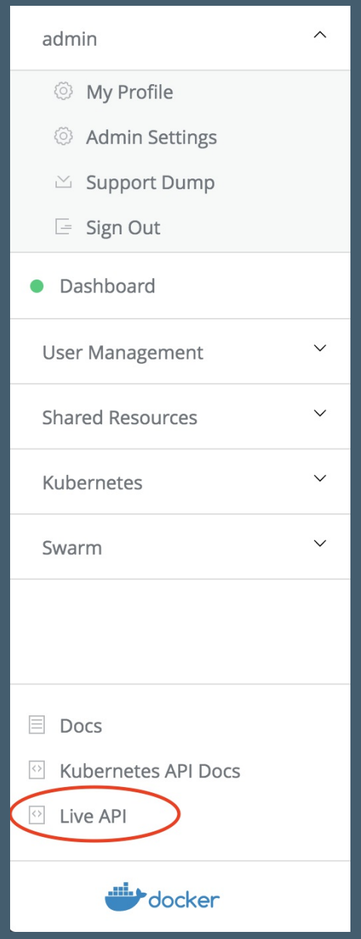




NETWORK TOPOLOGY SUMMARY



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

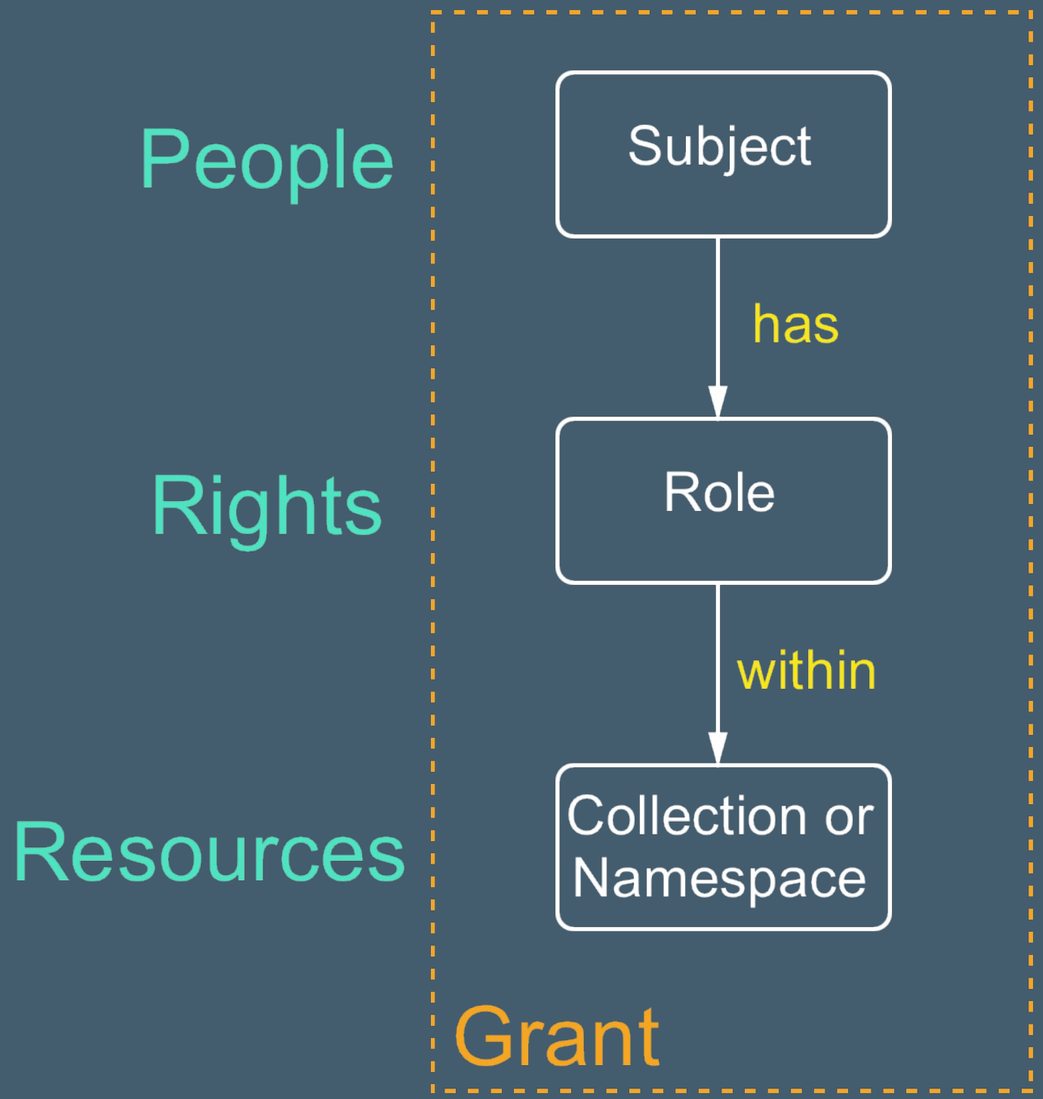
USER MANAGEMENT & ACCESS CONTROL

LEARNING OBJECTIVES

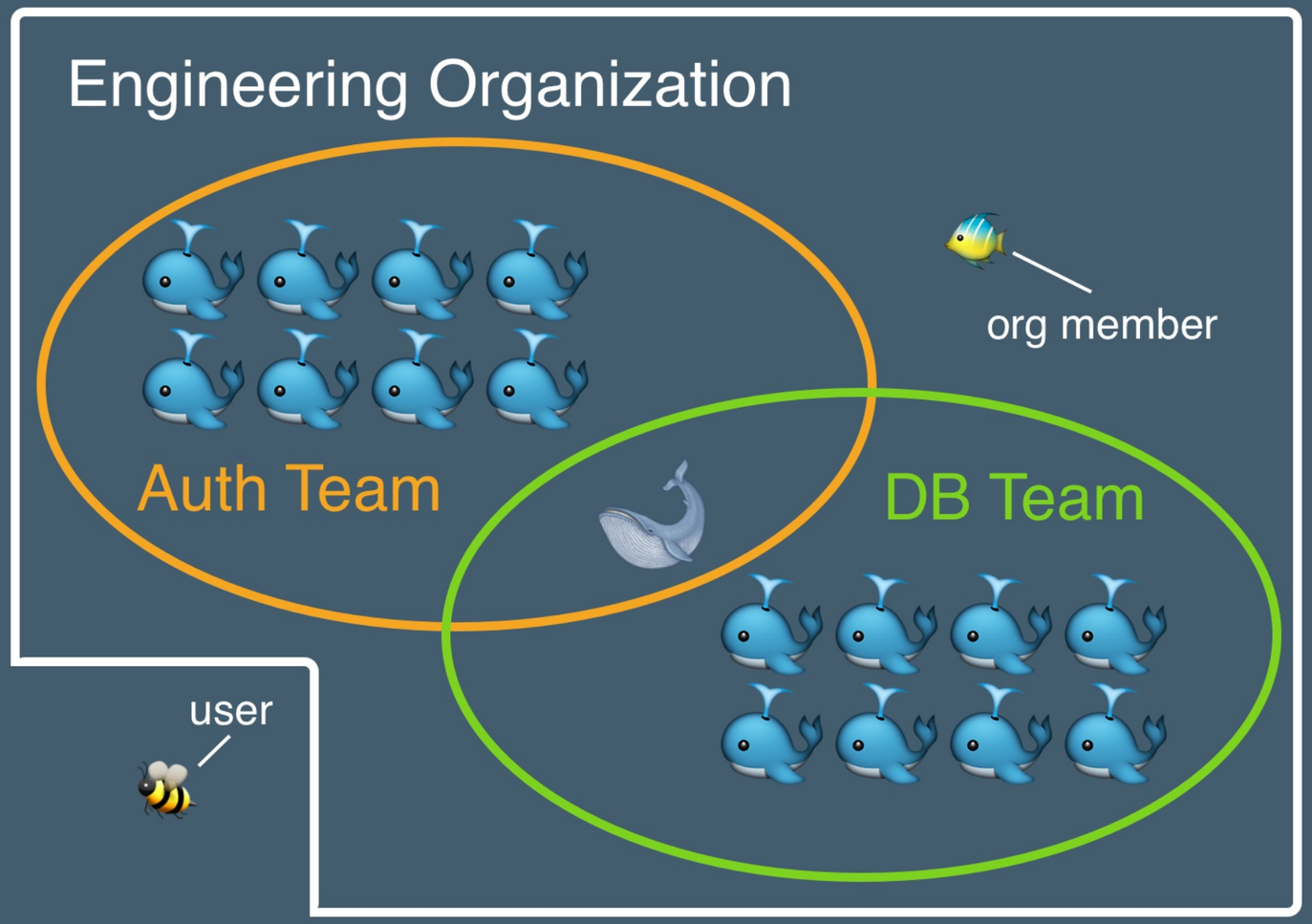
By the end of this module, learners will be able to:

- 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

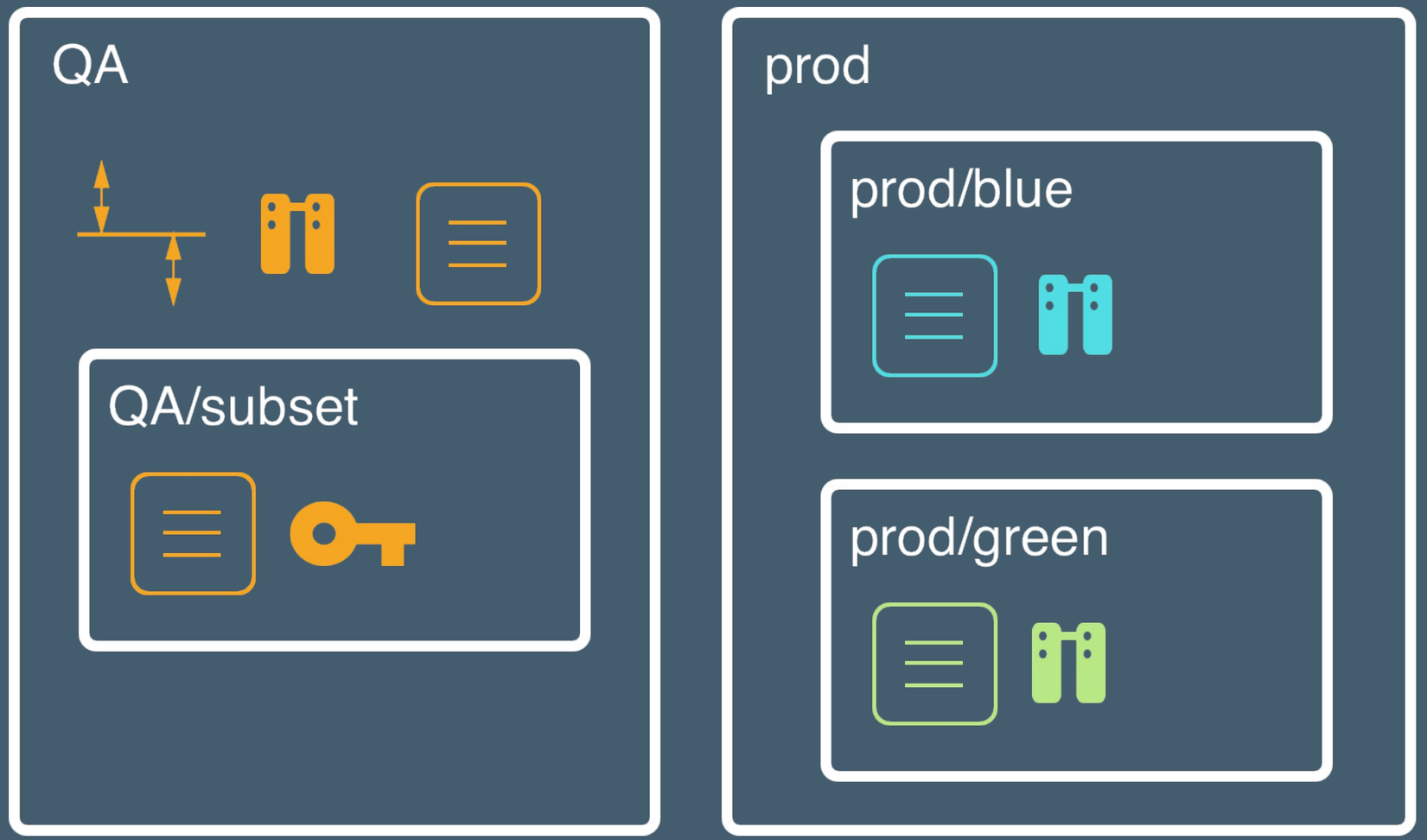
UCP ROLE BASED ACCESS CONTROL



DDC ORG CHARTS



RESOURCE COLLECTIONS

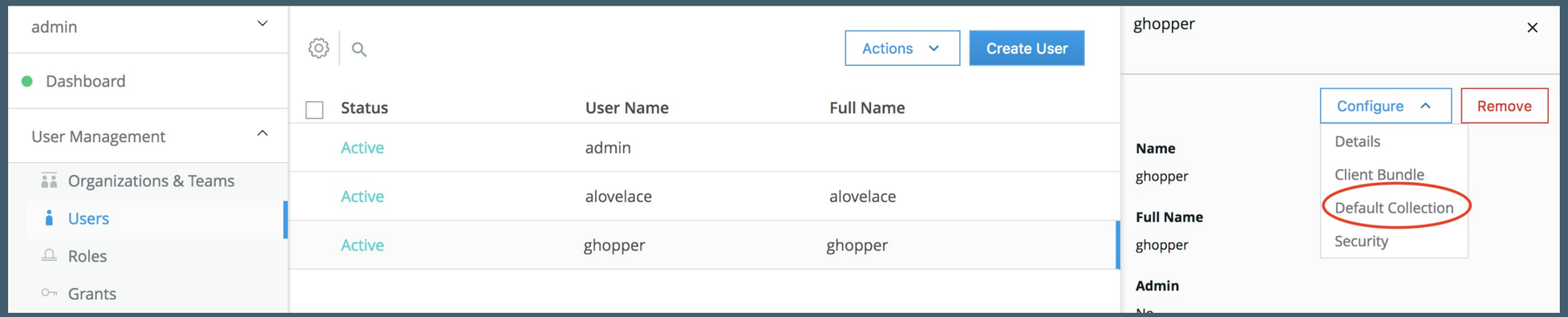


PRE-MADE COLLECTIONS

- /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>



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

No Node Permissions

RESTRICTED CONTROL:

- Create
- Run
- Restart
- Stop
- Delete

No Node Permissions

VIEW ONLY:

- Inspect
- View

No Node Permissions

NONE

SCHEDULER

- Node Schedule
- Node View

Node permissions

CUSTOM ROLES

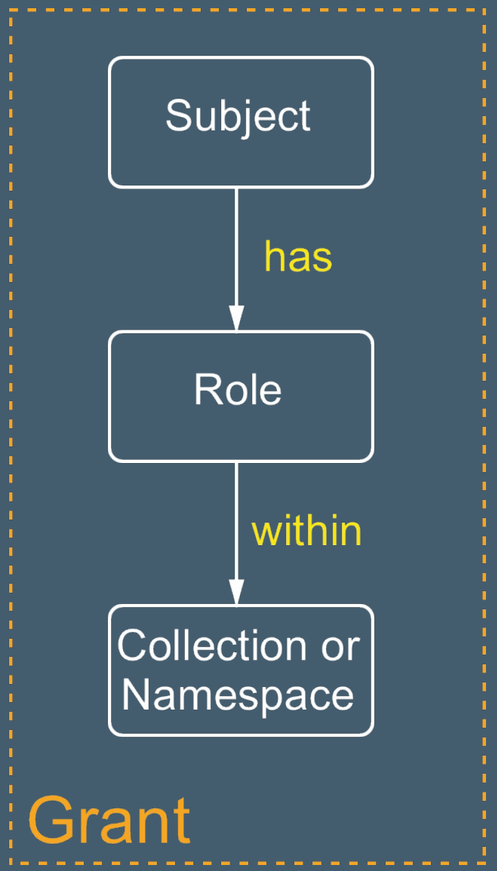
- Admins can create new custom roles
- Once created, roles are immutable

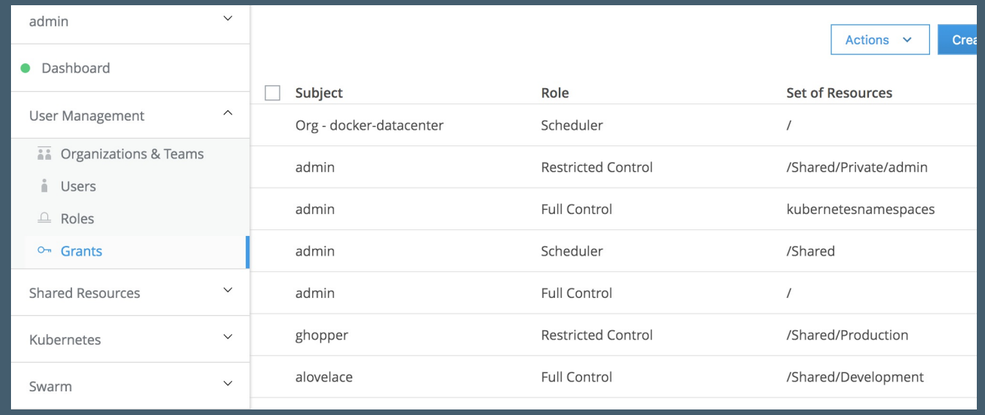
PERMISSION GRANTS

People

Rights

Resources



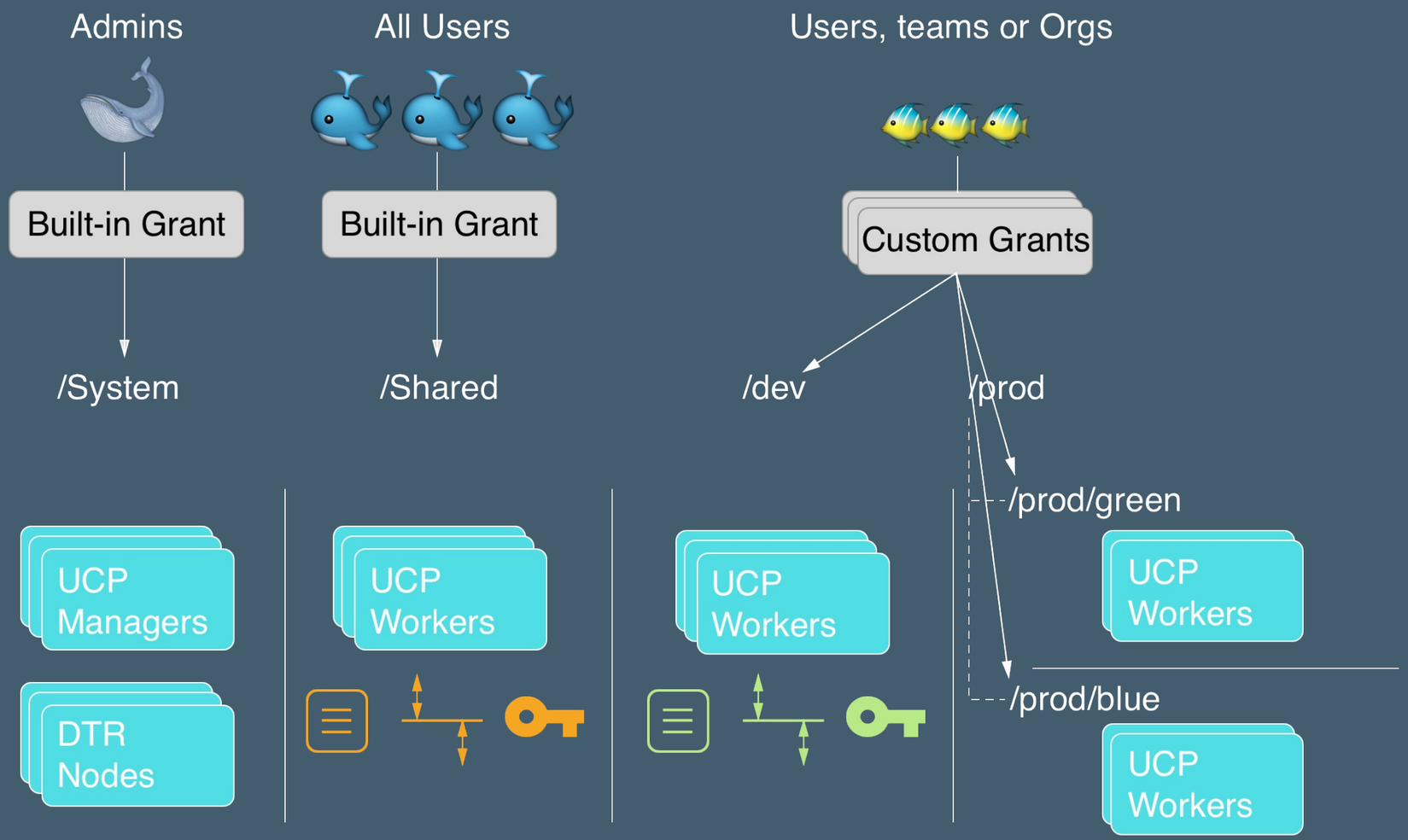


Only admins can define new grants.

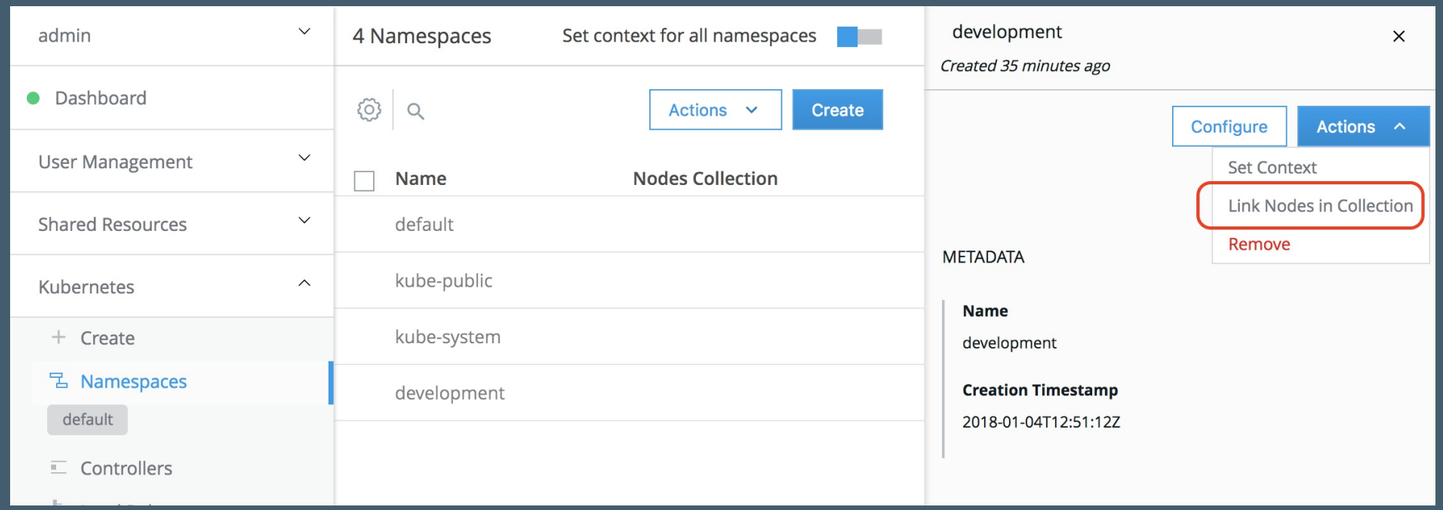
NODE RBAC: STANDARD TIER

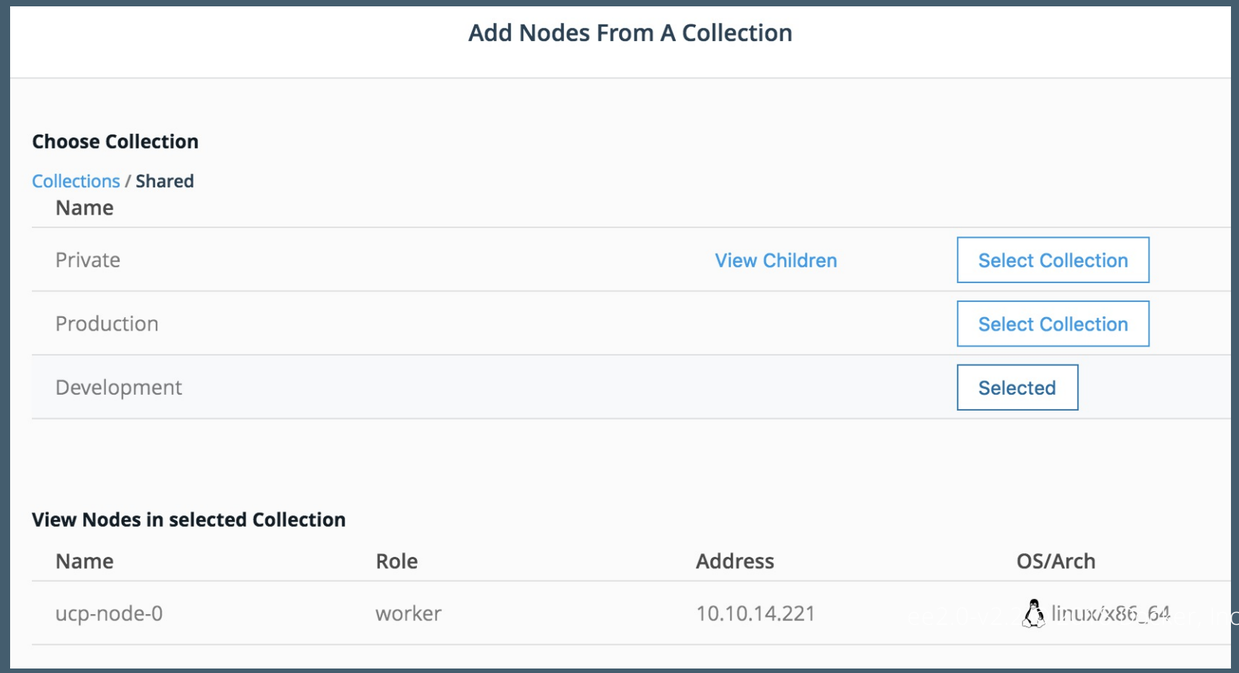
- All worker nodes part of /Shared collection
- All users have Scheduler grant to /Shared collection

NODE RBAC: ADVANCED TIER



NODE RBAC: KUBERNETES





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

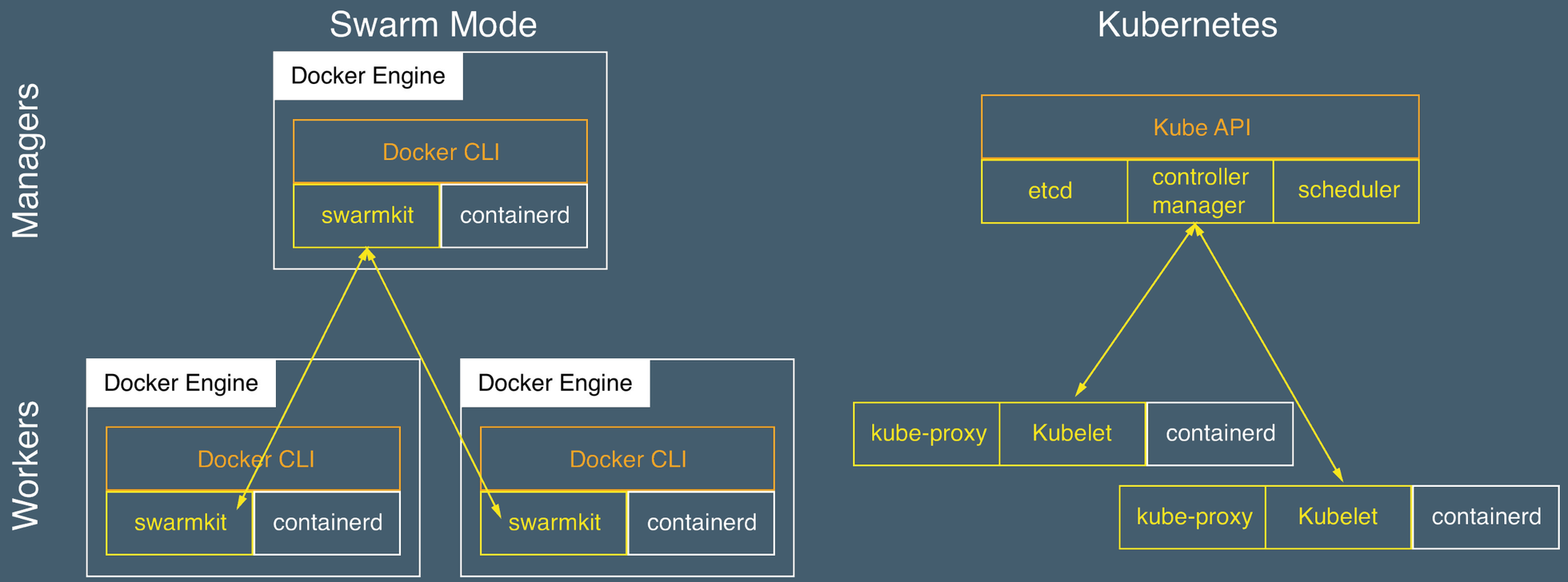
UCP ORCHESTRATION

LEARNING OBJECTIVES

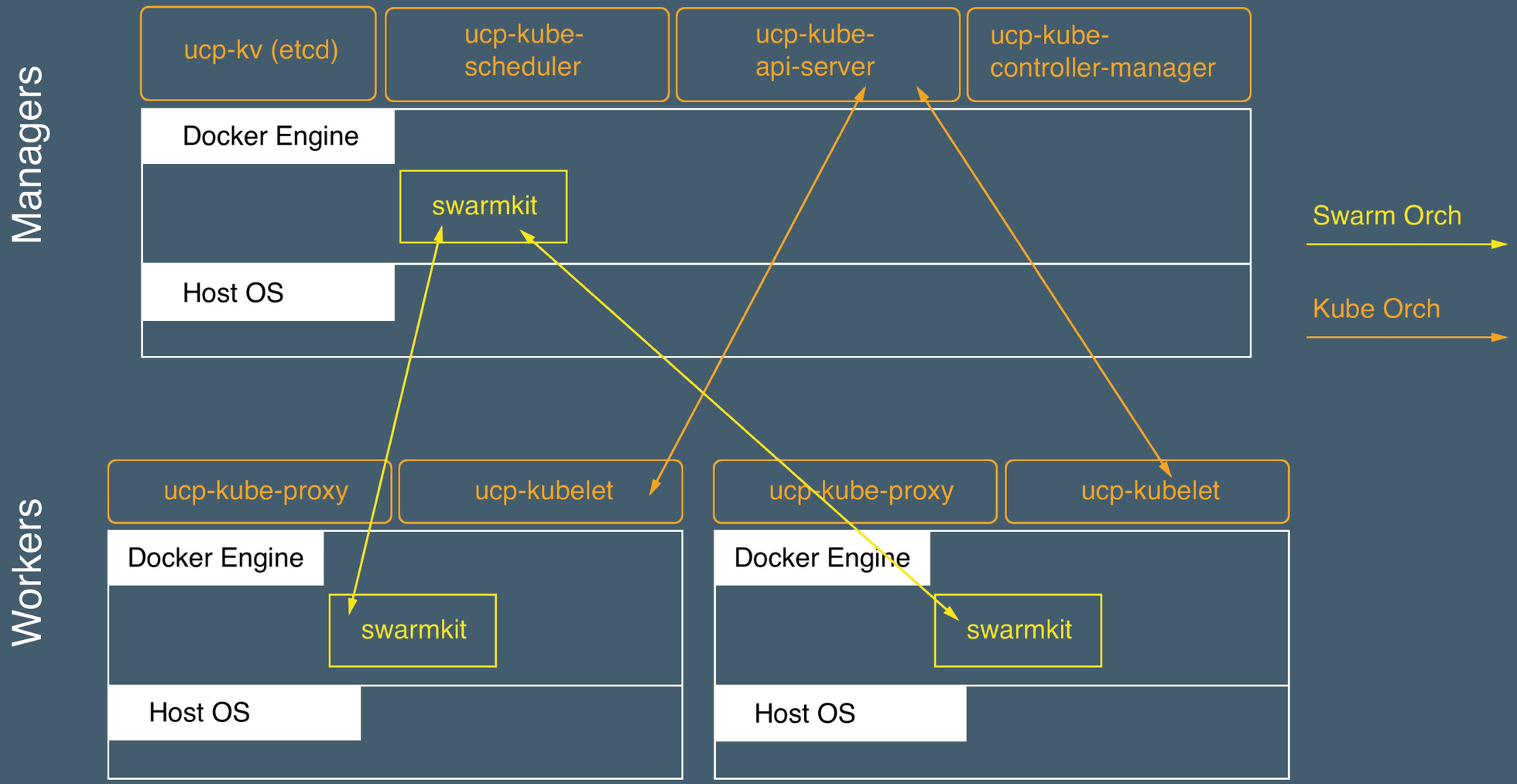
By the end of this module, learners will be able to:

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

ORCHESTRATOR ARCHITECTURE



ORCHESTRATORS IN UCP



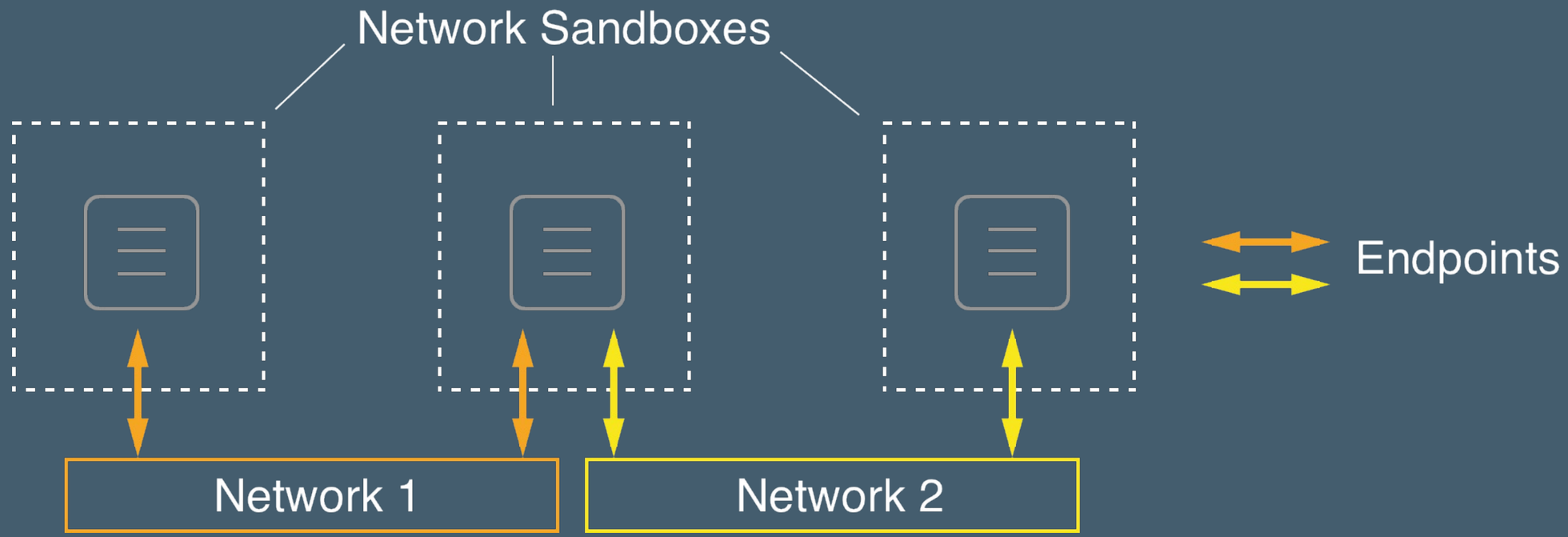
ADDITIONAL KUBERNETES COMPONENTS

- Kube DNS
- Calico
- Compose Adapter

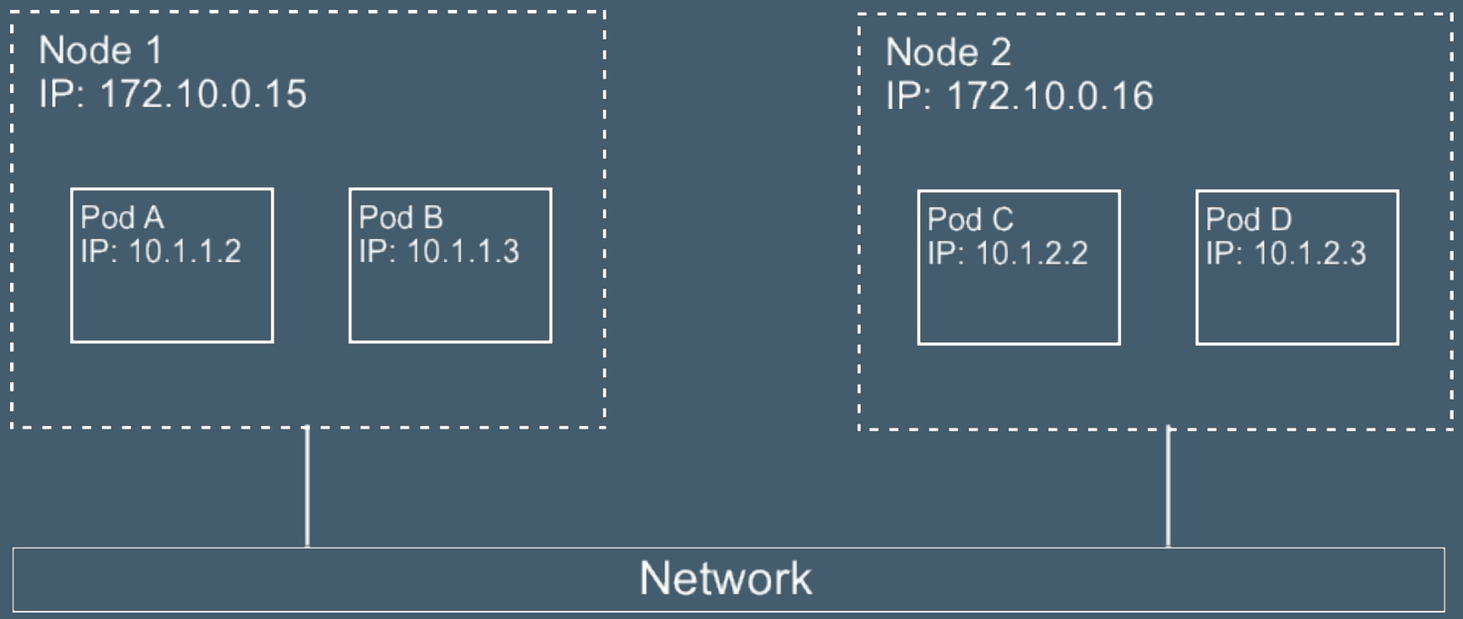
NETWORKING MODELS

- 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



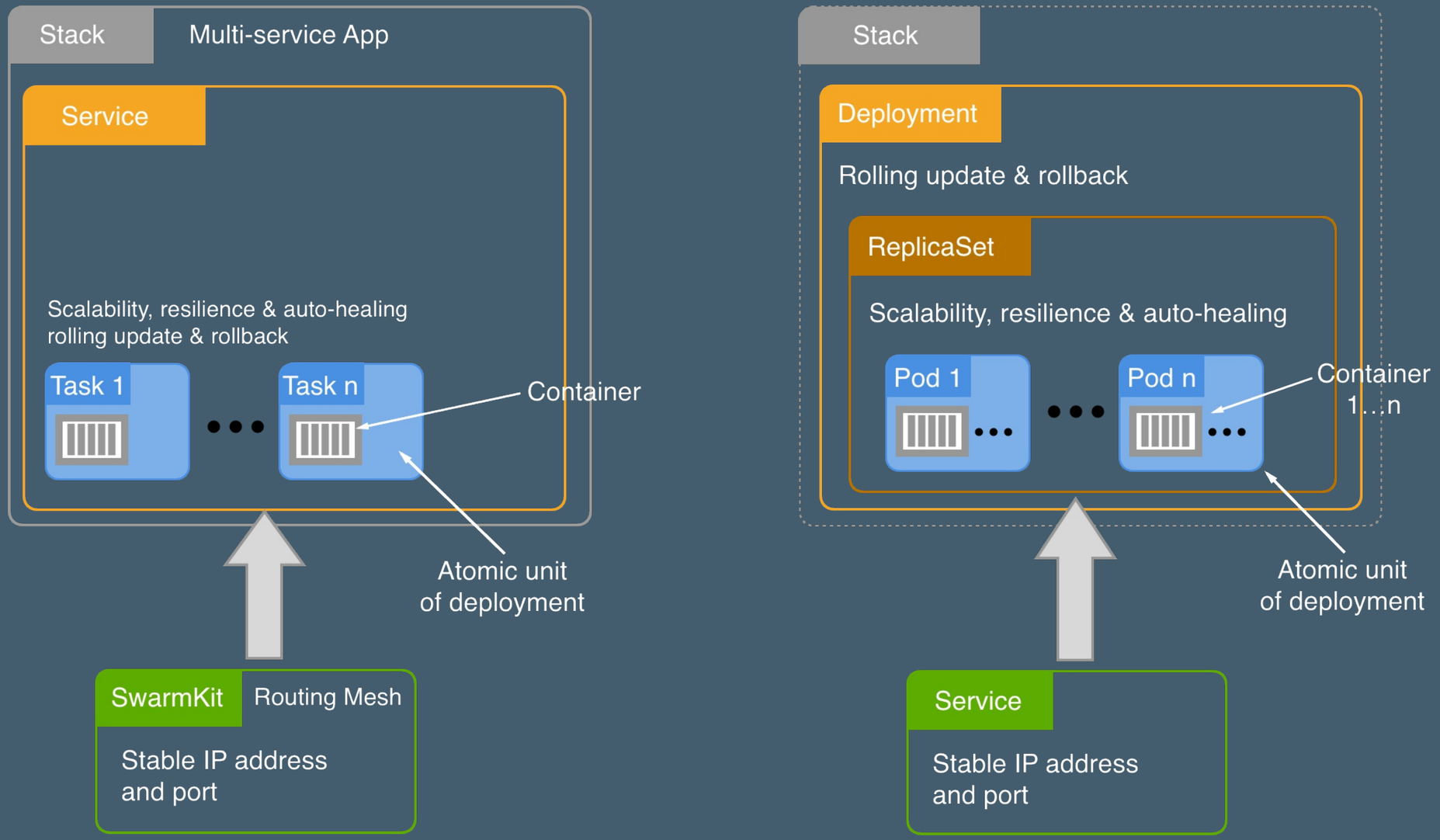
KUBERNETES NETWORK MODEL



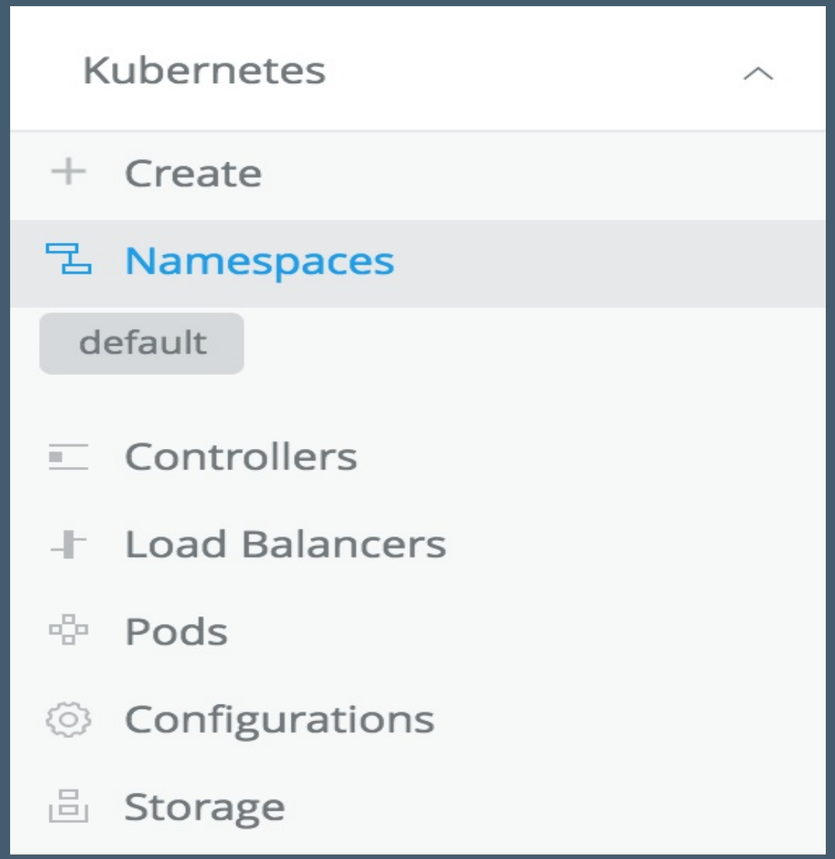
Requirements

- 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

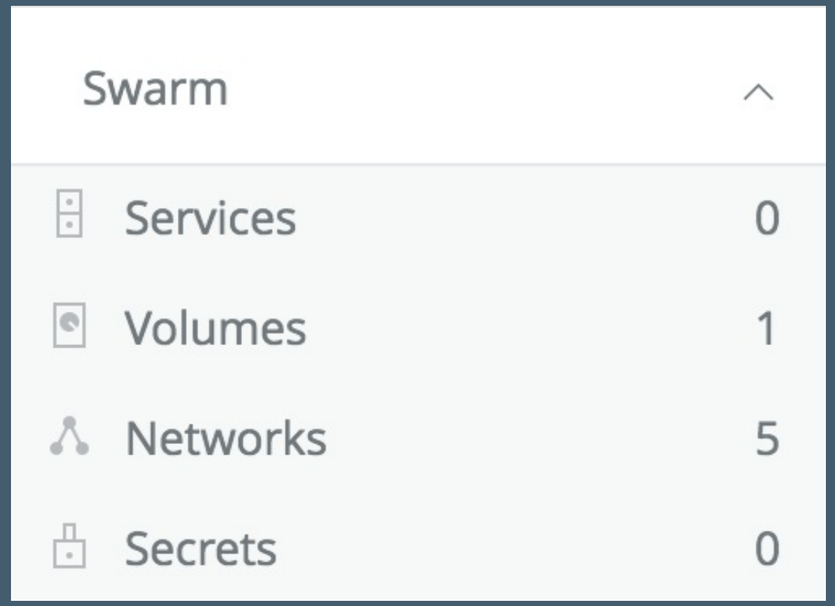
ORCHESTRATION COMPONENTS



ORCHESTRATOR UI



- 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/2kdI1qQ
- Kubernetes the Hard Way: http://bit.ly/29Dq4wC

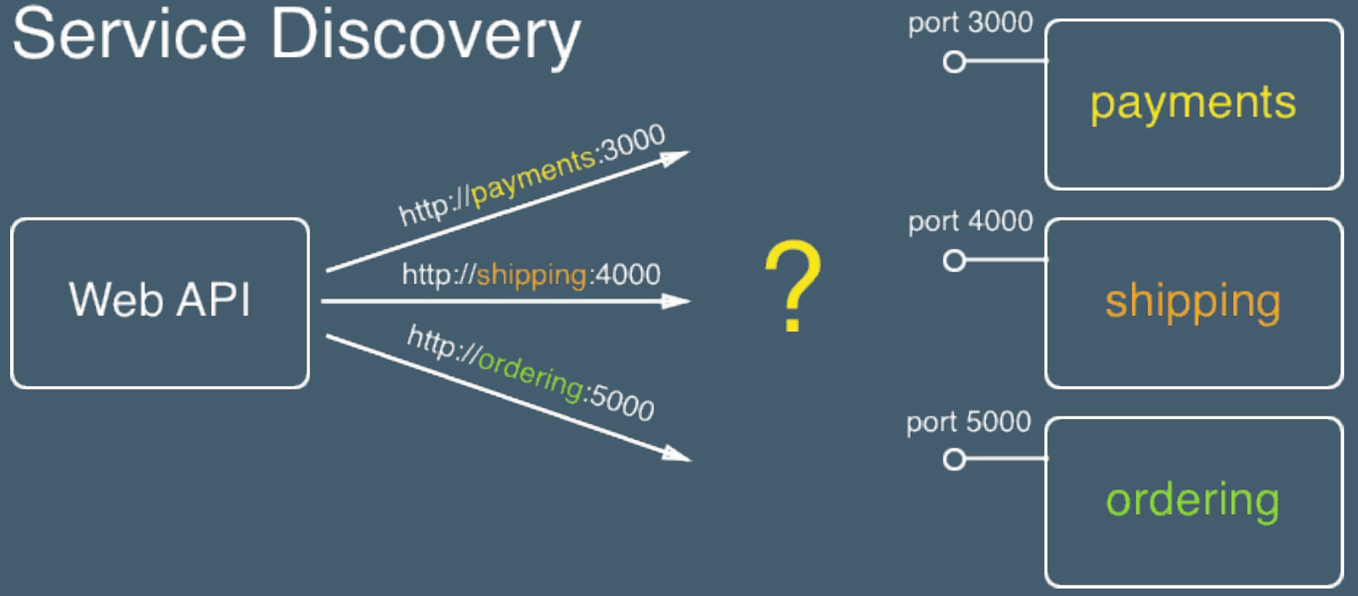
SERVICE MESH

LEARNING OBJECTIVES

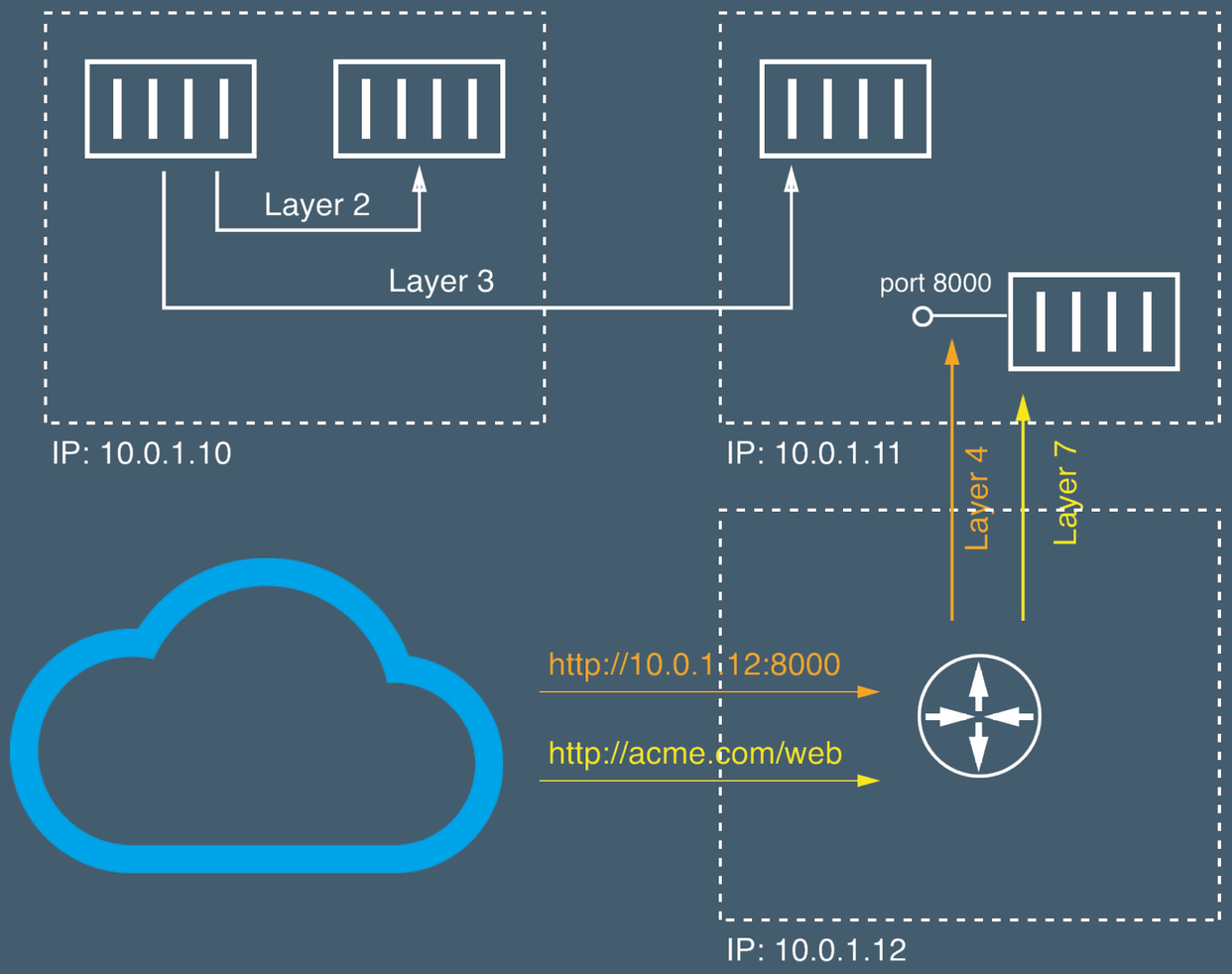
By the end of this module, learners will be able to:

- 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

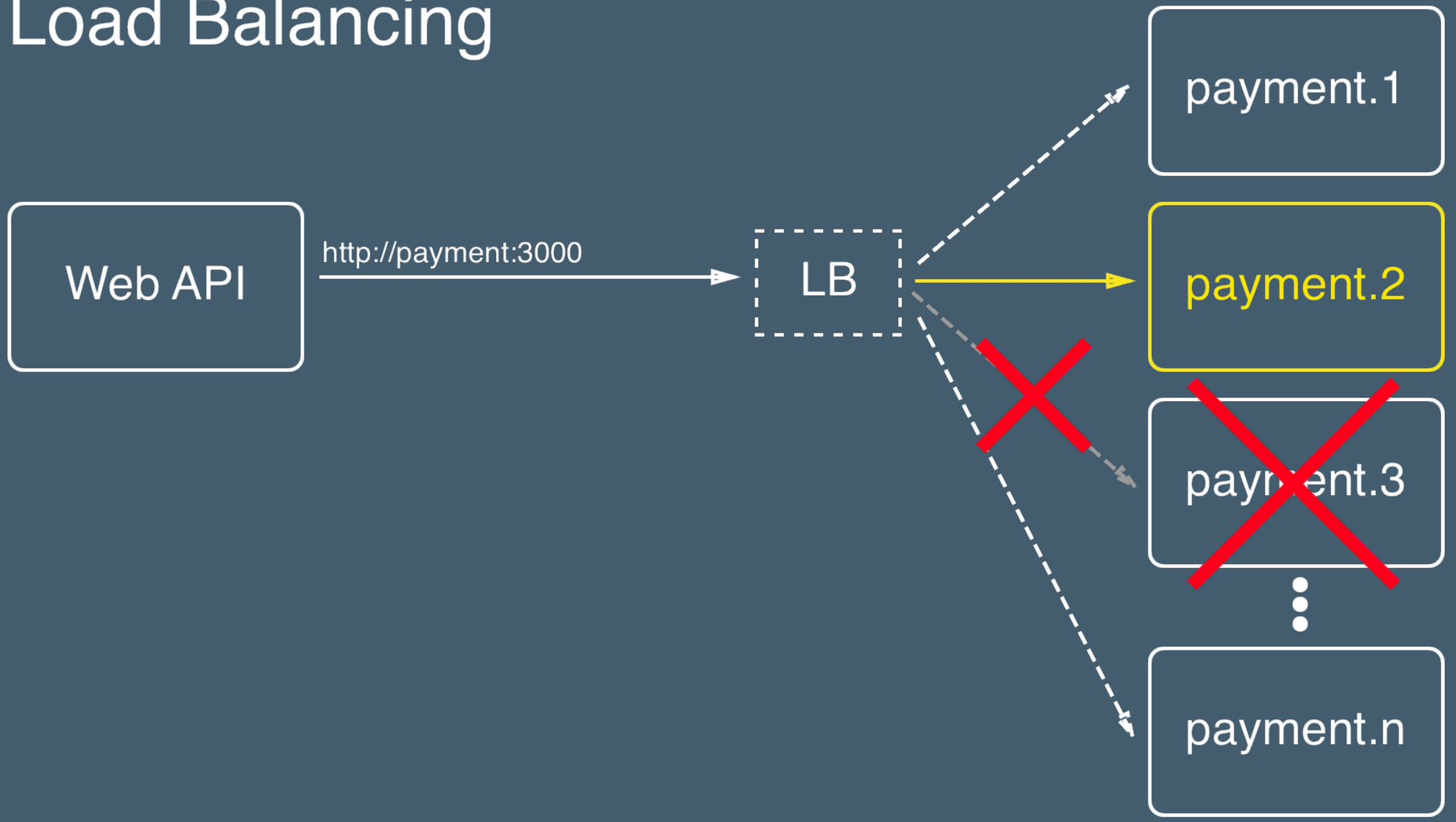
Service Discovery



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

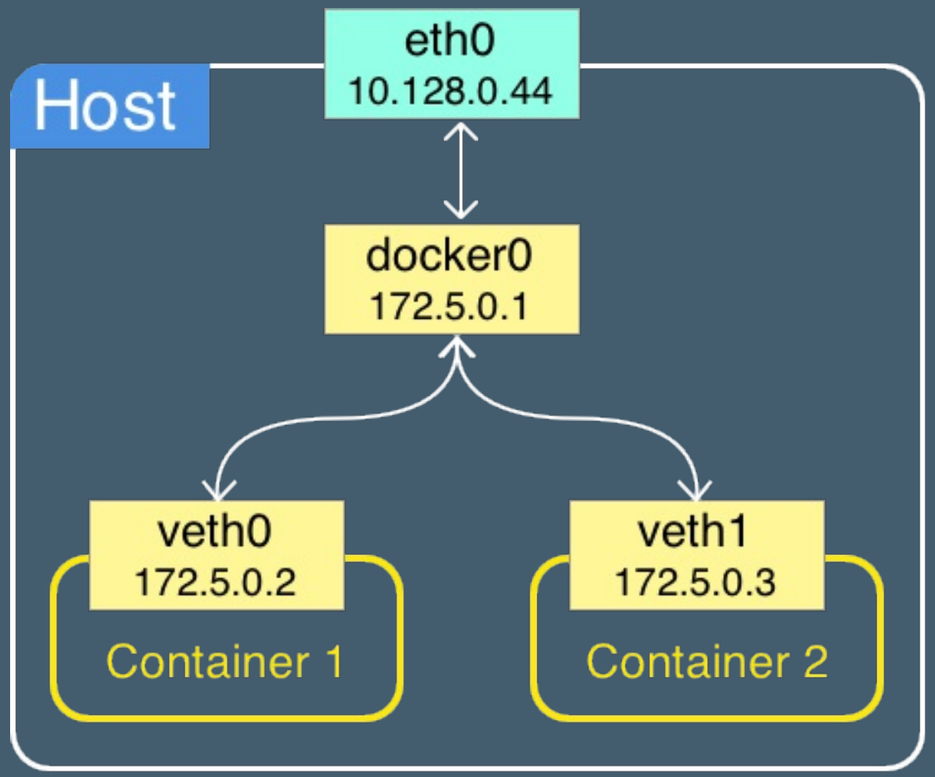
EXERCISE: SERVICE MESH (1/3): SERVICE DISCOVERY

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

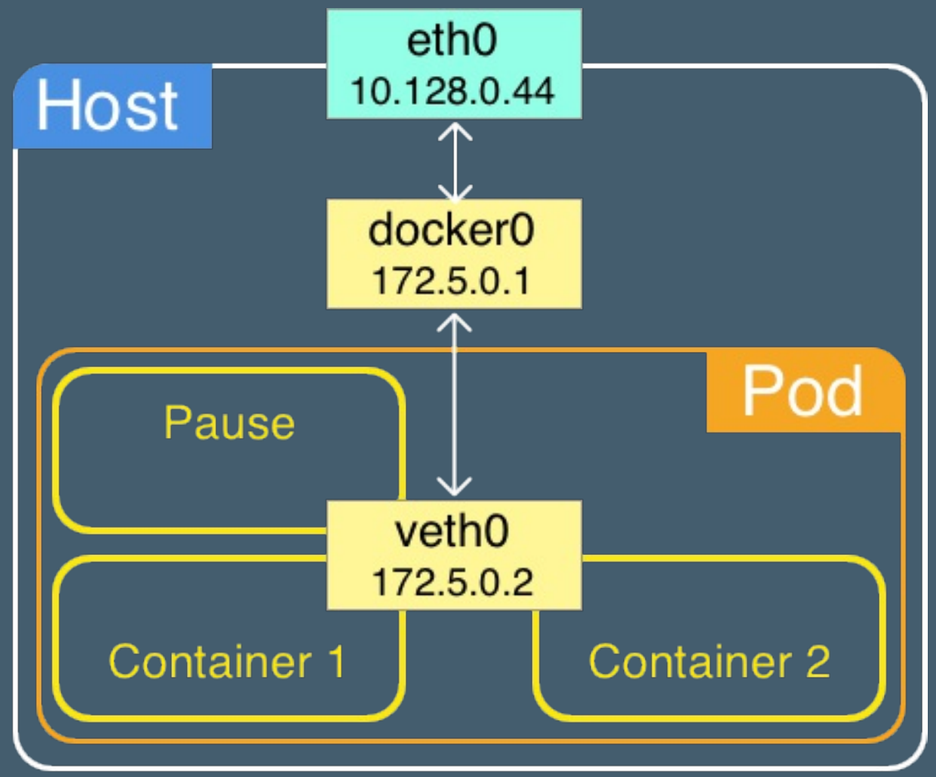
ROUTING - IMPLEMENTATION

- 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

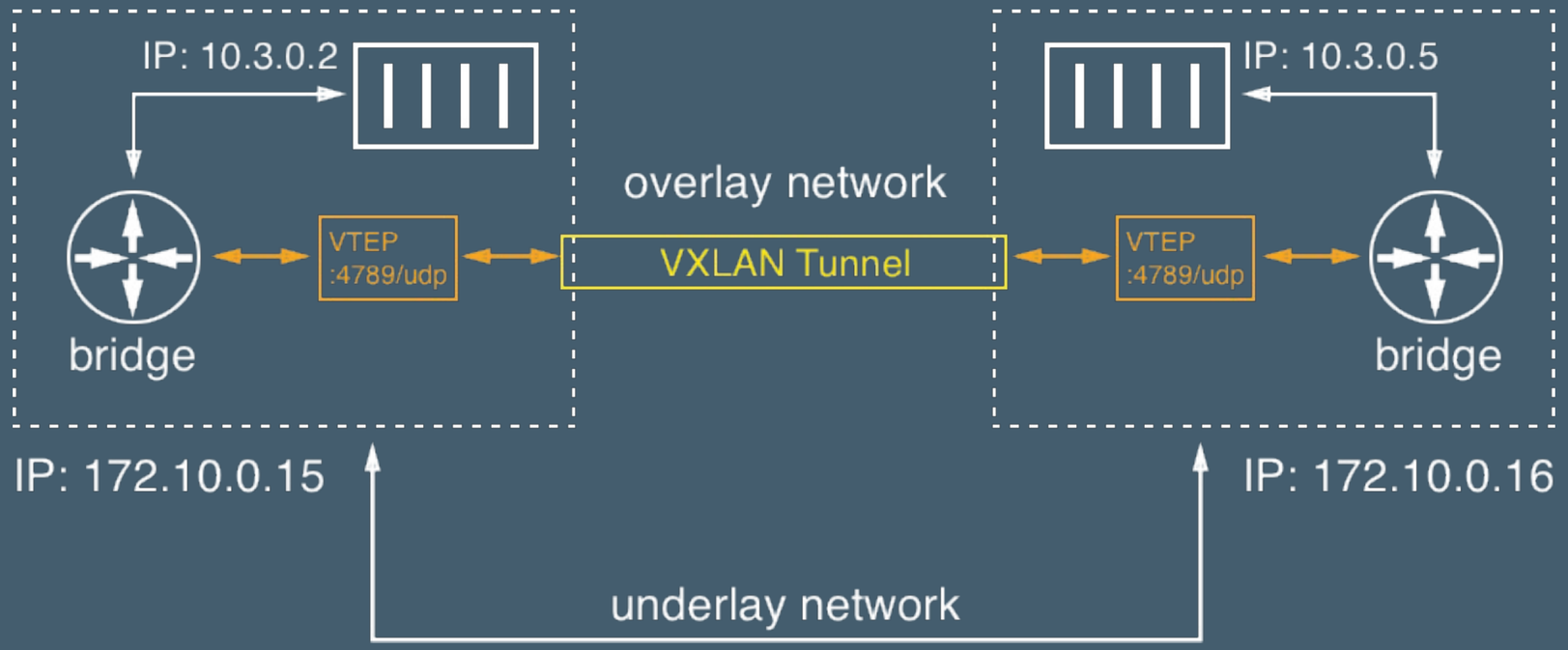


Docker Bridge Network

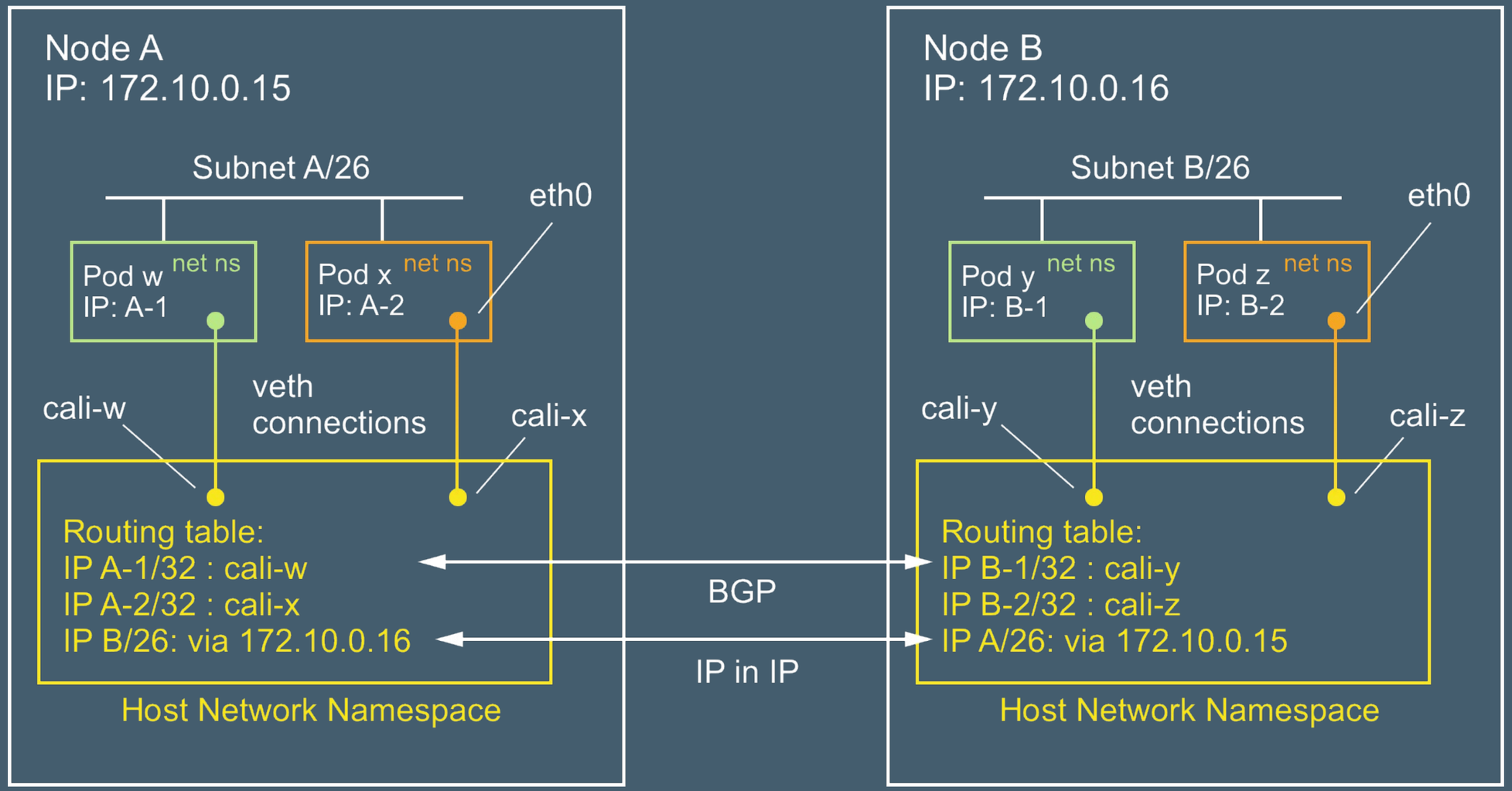


Kubernetes Network

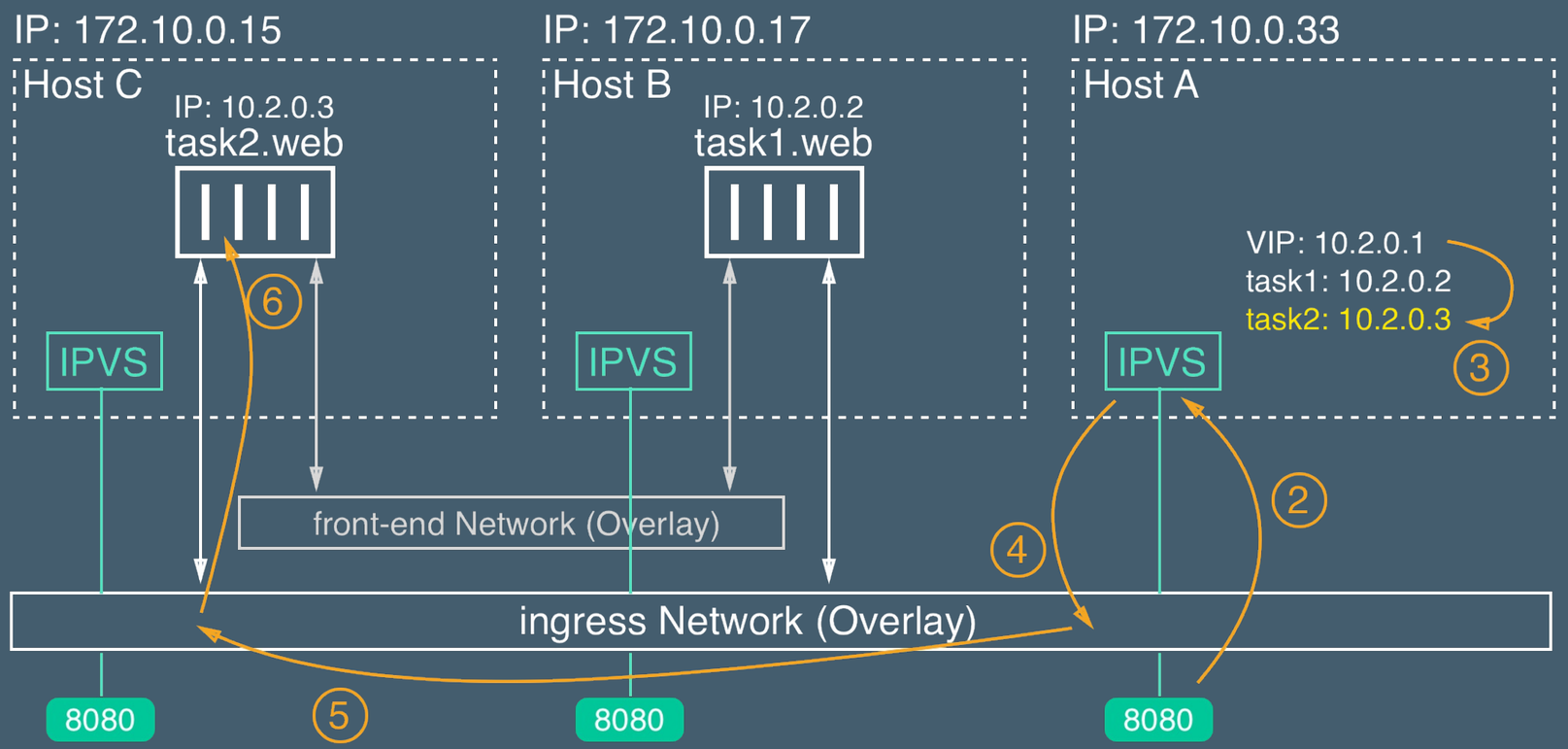
L3 ROUTING - SWARM: VXLAN



L3 ROUTING - KUBE / CALICO



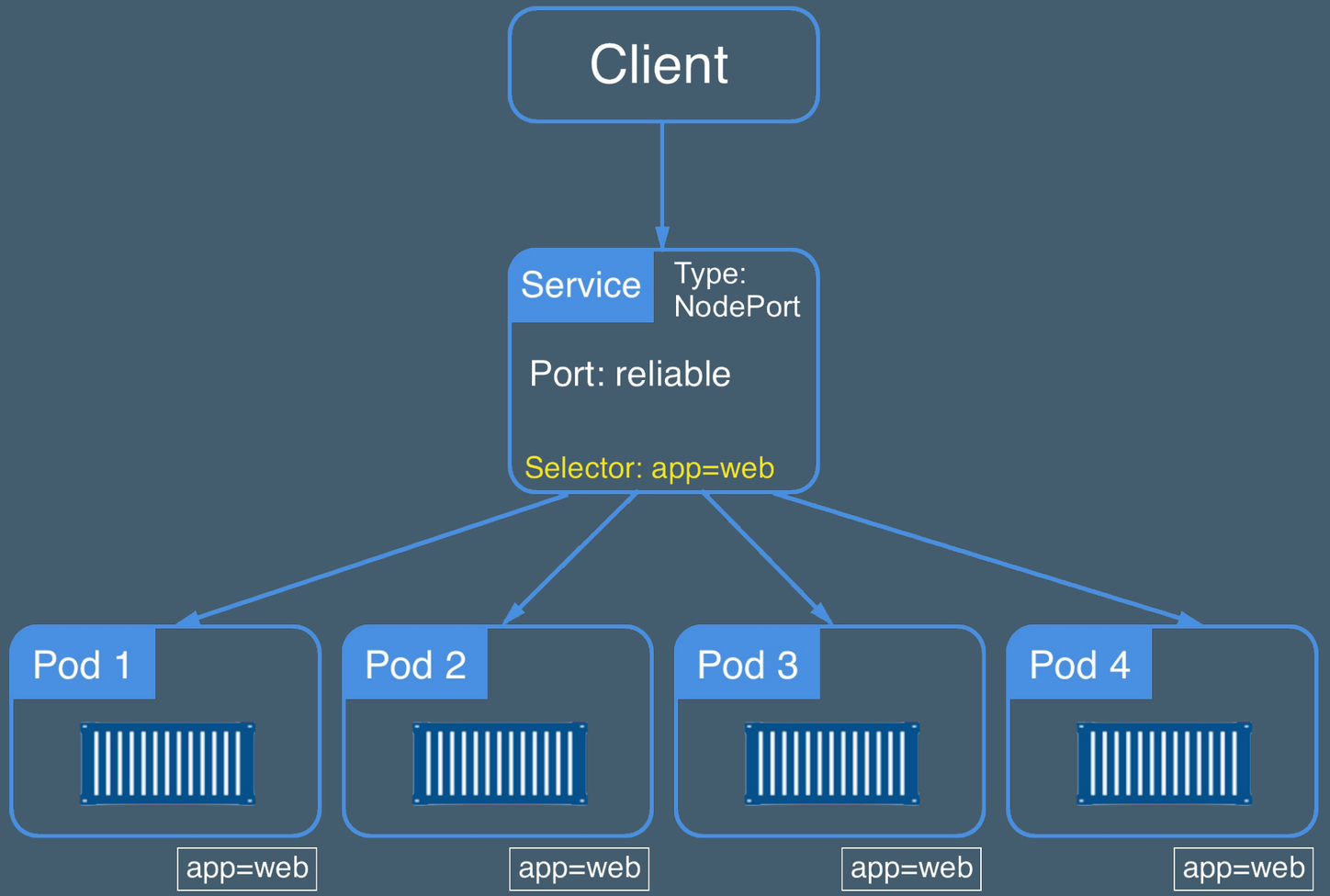
L4 ROUTING - SWARM: ROUTING MESH




```
docker service create
  --name web
  --replicas 2 \
  --network front-end \
  -p 8080:80 nginx:alpine
```

External LB

L4 ROUTING - KUBERNETES: SERVICE OBJECT



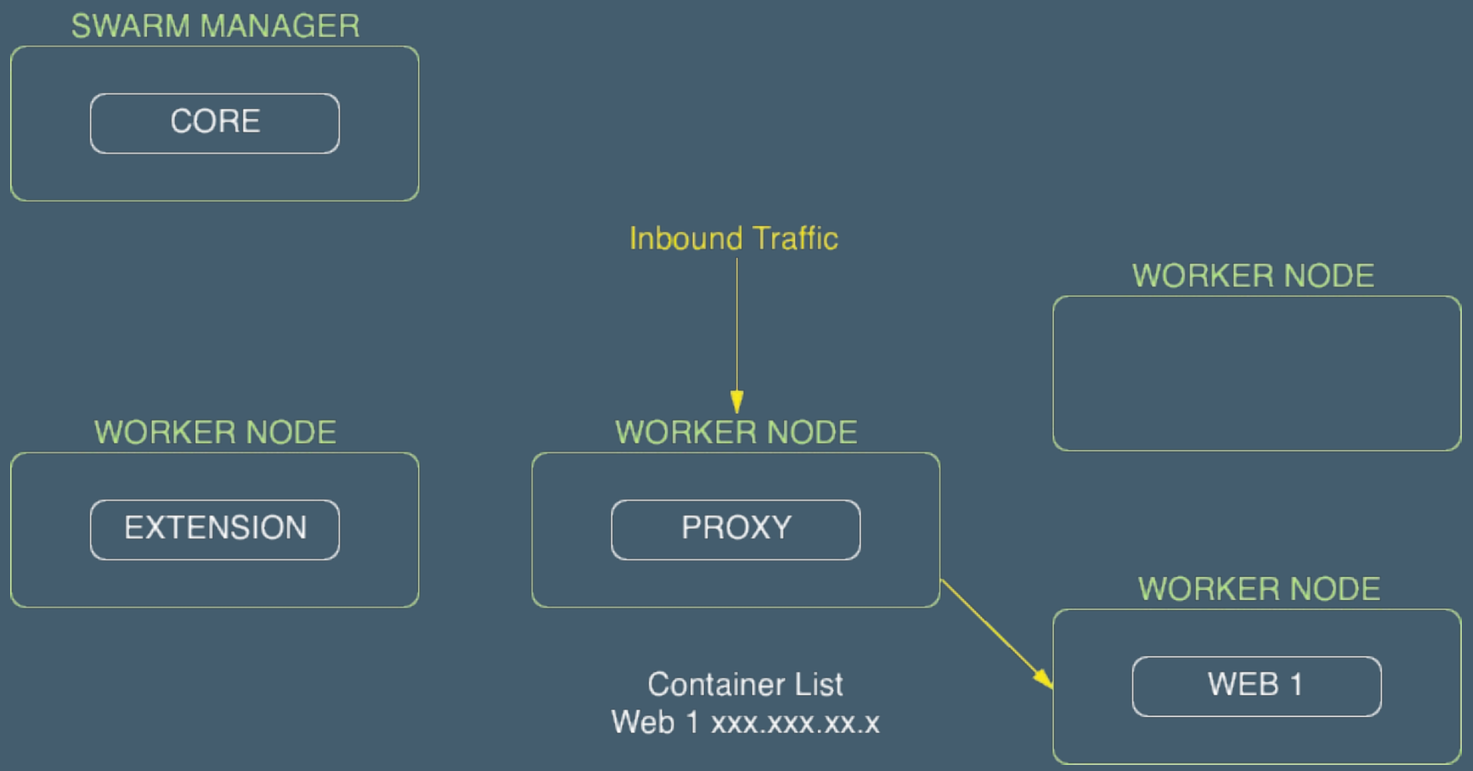
L7 ROUTING - SWARM: INTERLOCK 2

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

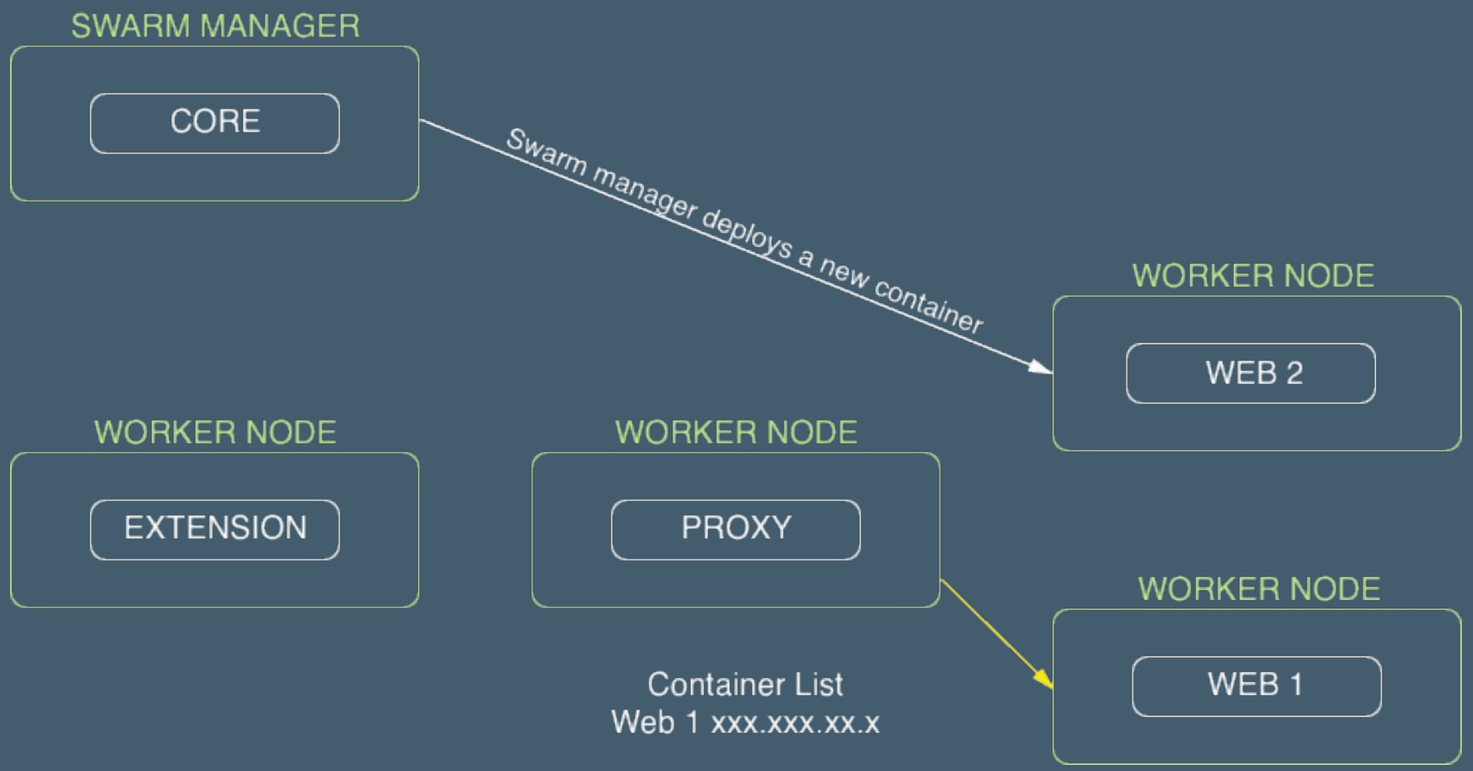
Architecture: 3 Docker Swarm services:

- Core
- Extension
- Proxy

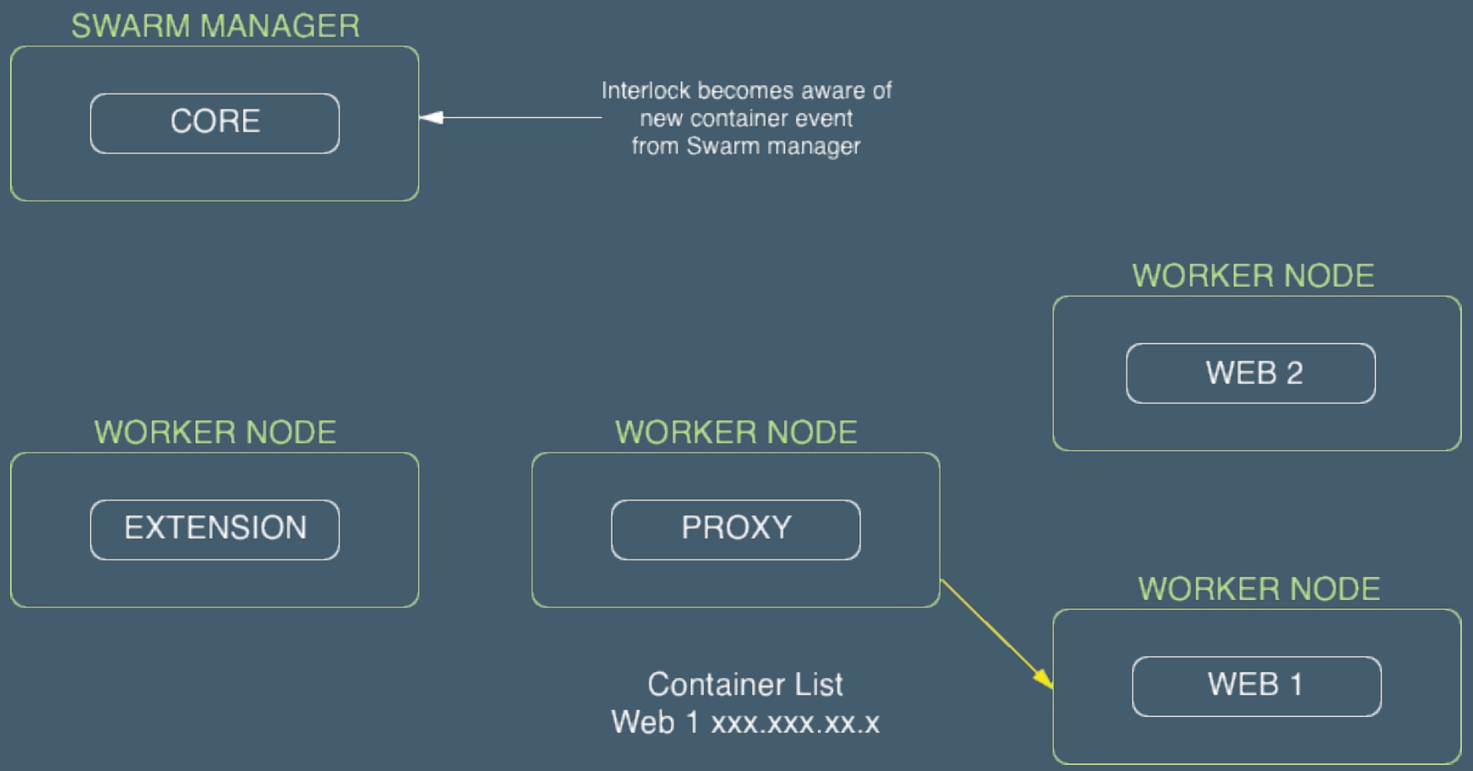
INTERLOCK 2 TRAFFIC



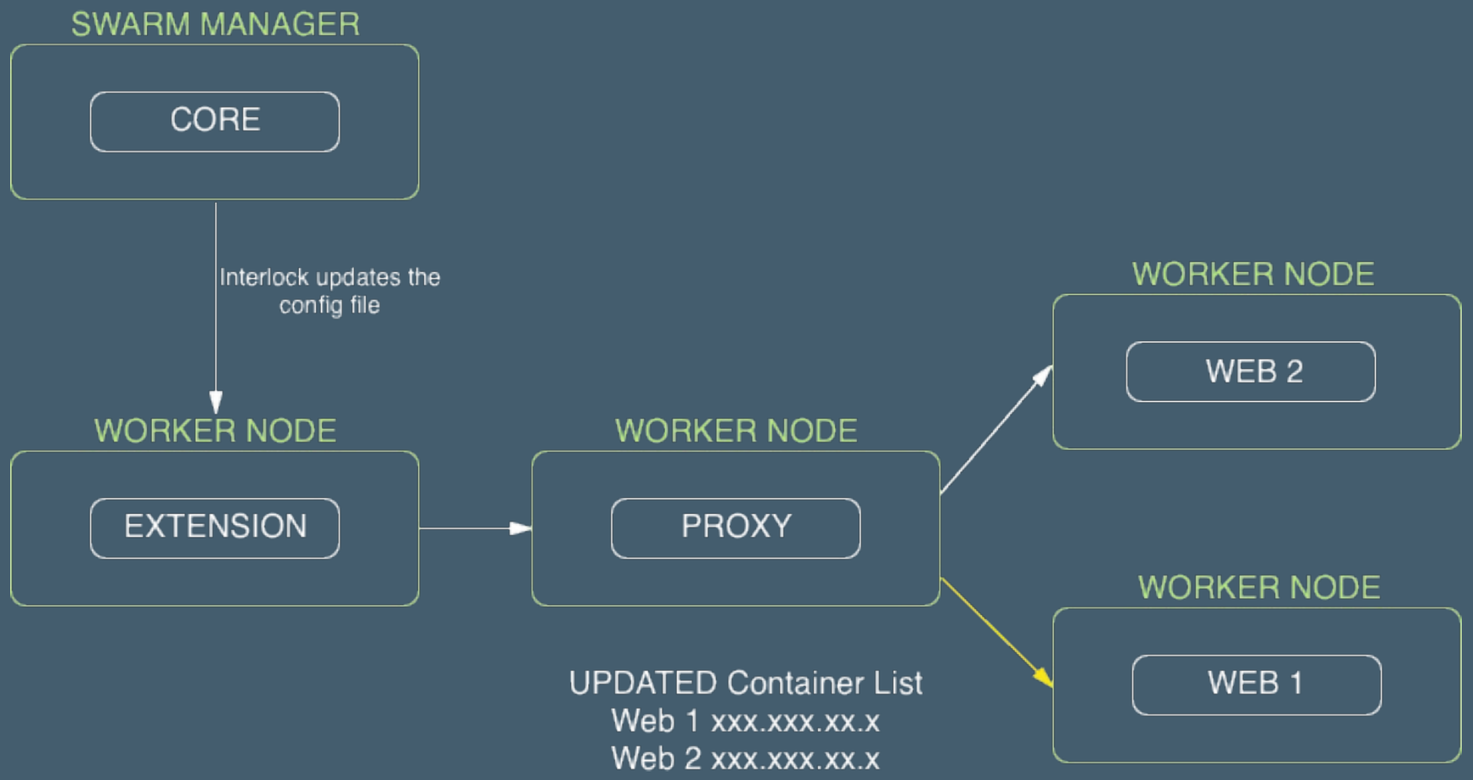
INTERLOCK 2 TRAFFIC



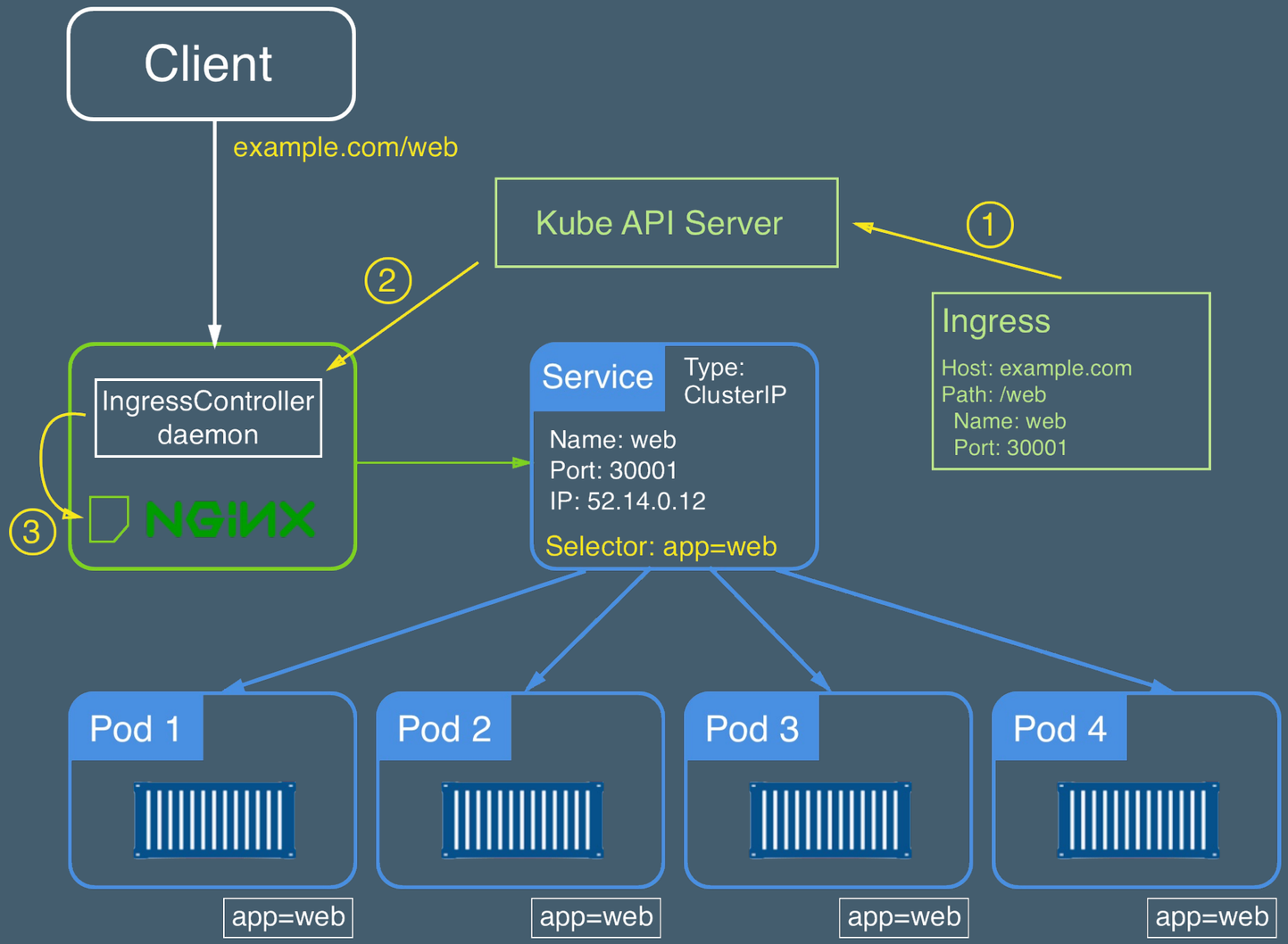
INTERLOCK 2 TRAFFIC



INTERLOCK 2 TRAFFIC



L7 ROUTING - KUBE: INGRESS CONTROLLER



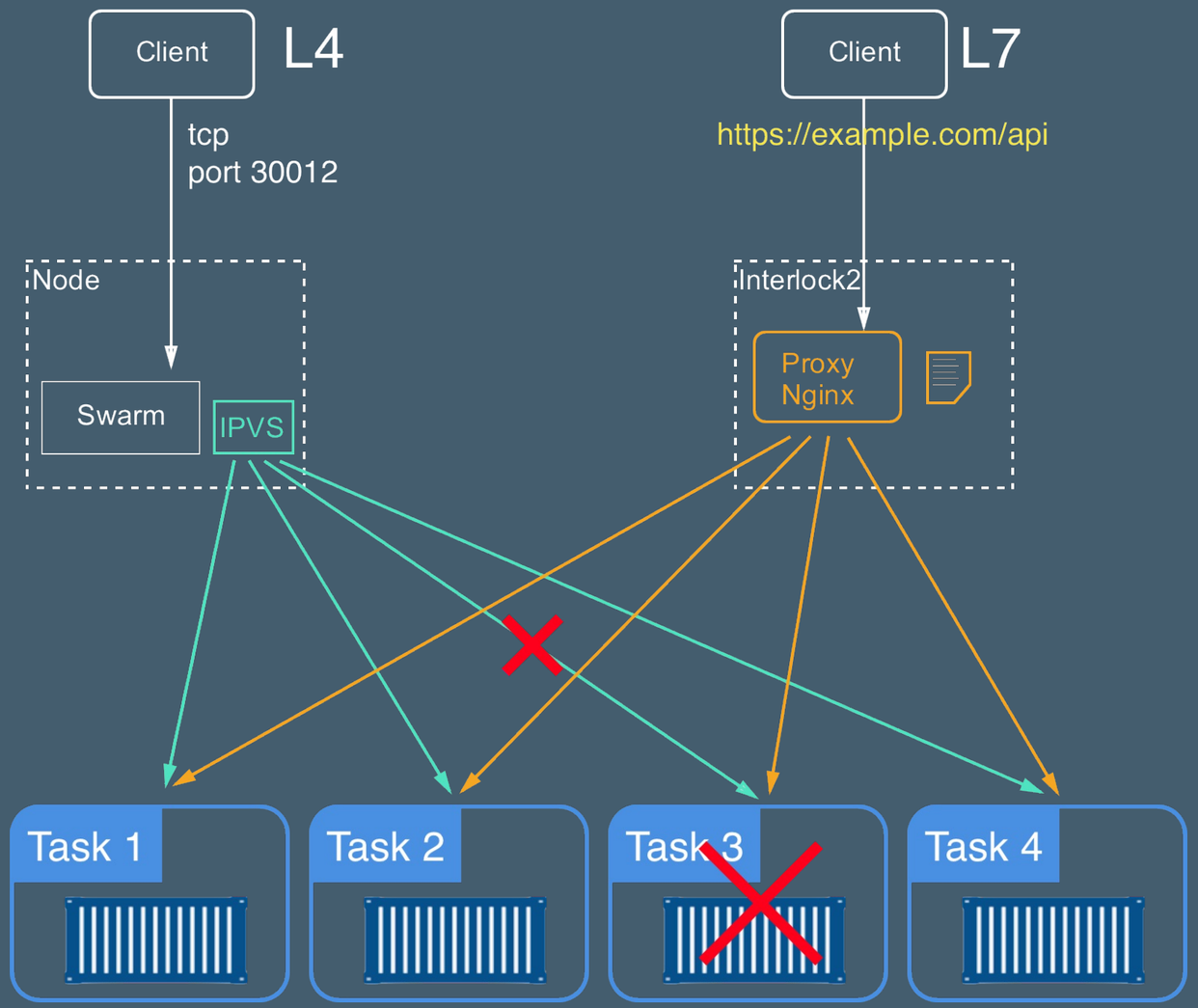
EXERCISE: SERVICE MESH (2/3): ROUTING

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

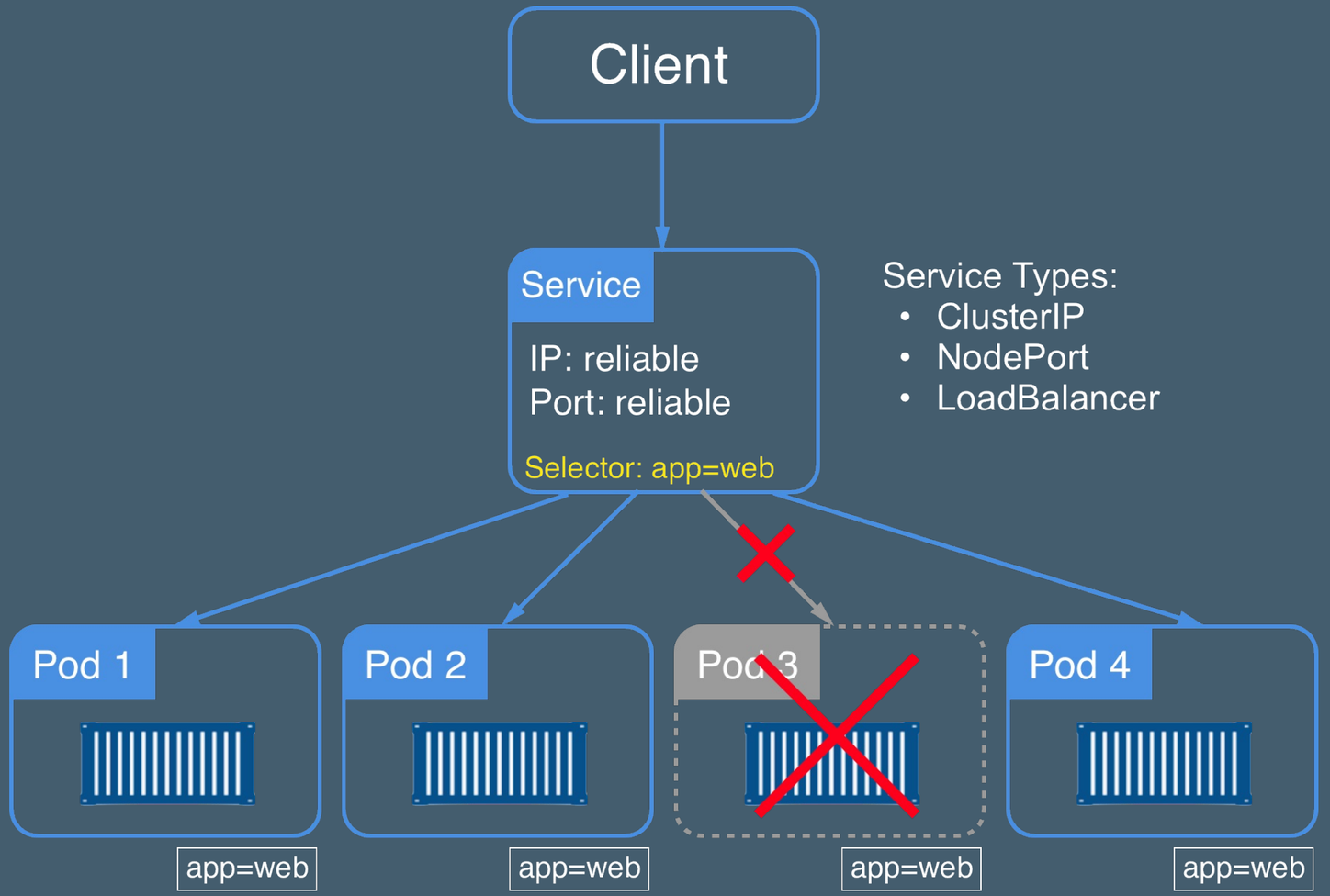
LOAD BALANCING - IMPLEMENTATION

- 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/2rPHAcl
- Kubernetes Services: https://bit.ly/2GSXwyB
- Accessing Kubernetes Pods from Outside of the Cluster: http://bit.ly/2DR6JJg

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

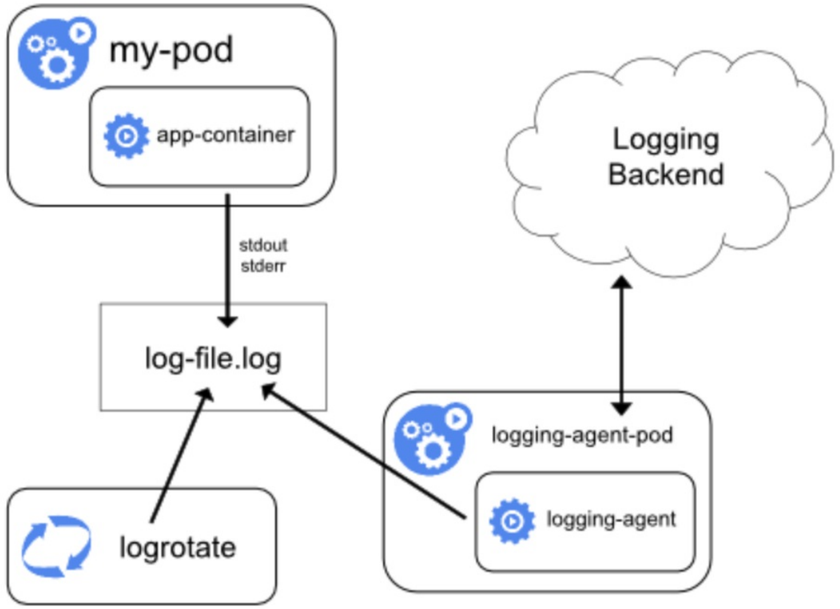
- Logs come from two sources:
  - dockerd: system-level
  - containers: STDOUT & STDERR
- Globally configured by /etc/docker/daemon.json:

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

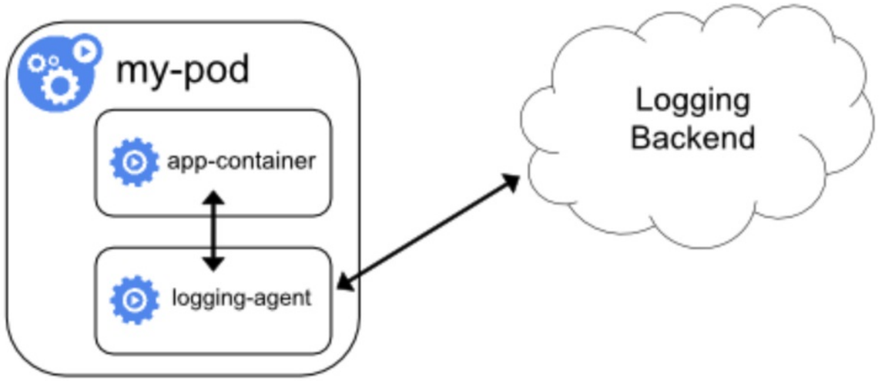
- Override for containers with --log-opt

KUBERNETES LOGGING

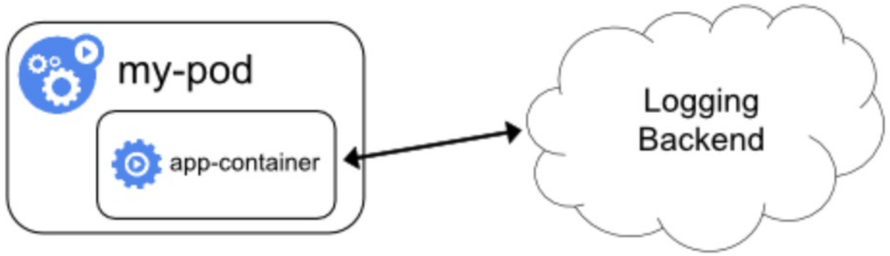
Using a node logging agent



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/2ezdZdI
- 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:

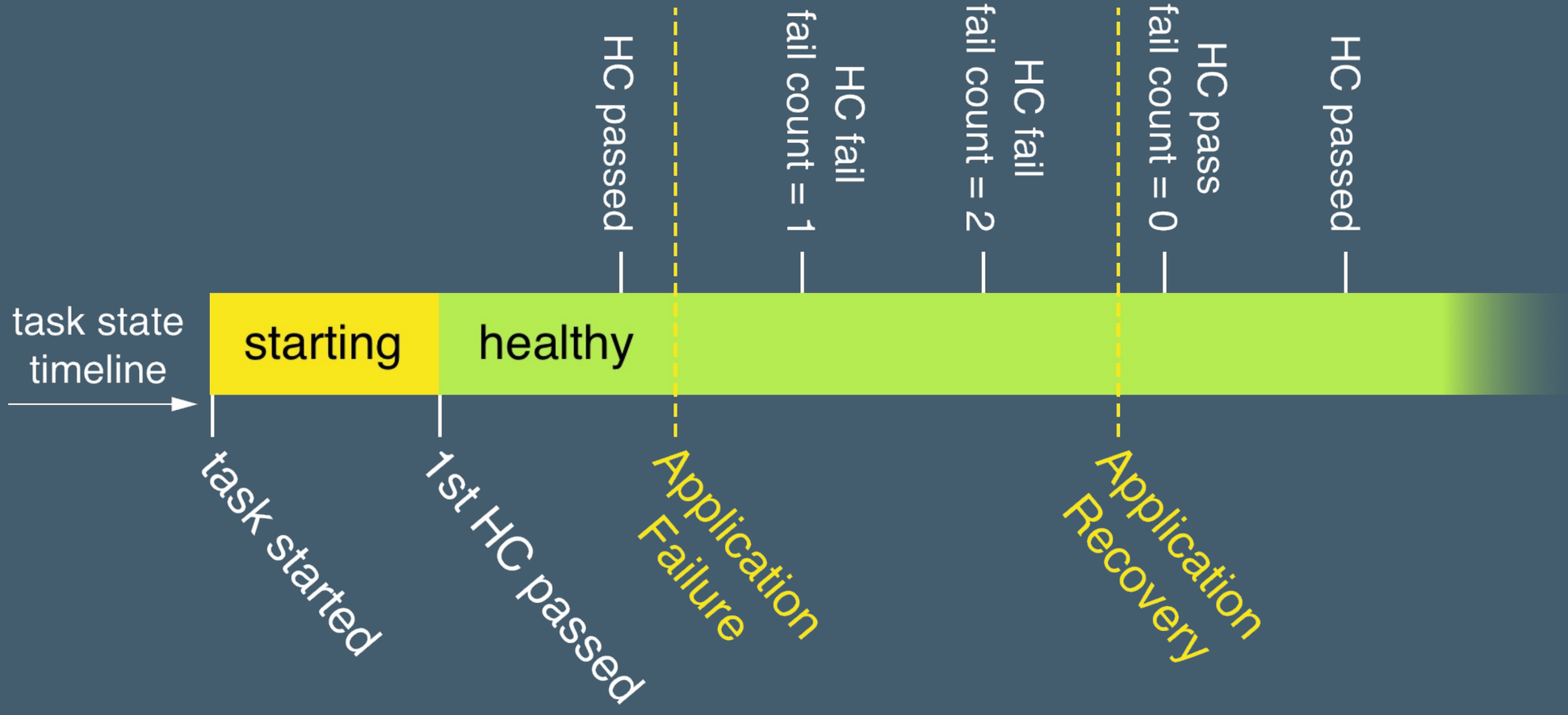
- Describe the steps of a generic health check protocol
- Configure Swarm and Kubernetes to kill unhealthy containers, and configure Kubernetes to remove unready pods from load balancing

HEALTH CHECK PROTOCOLS

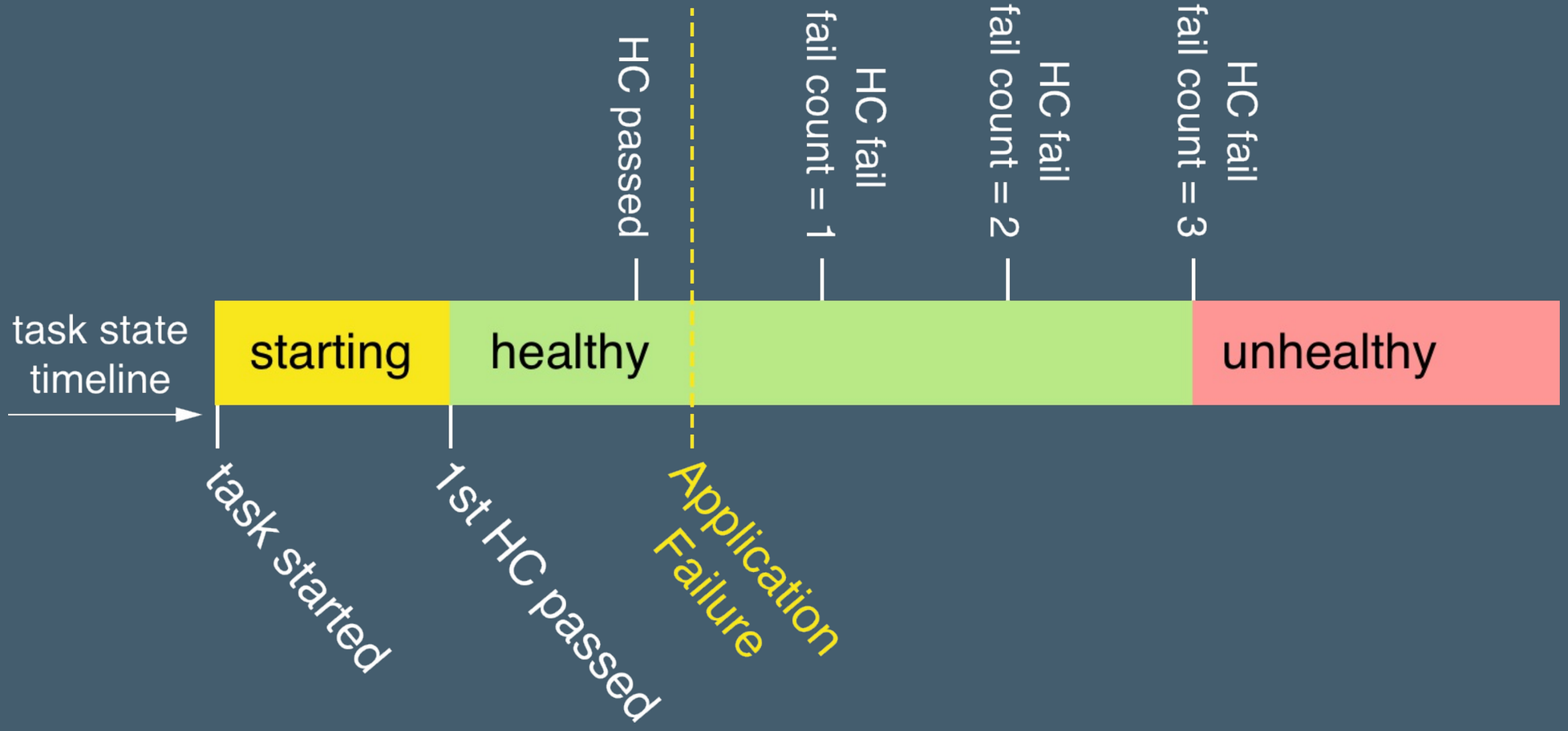
Monitoring application health requires:

- 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 <= 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

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

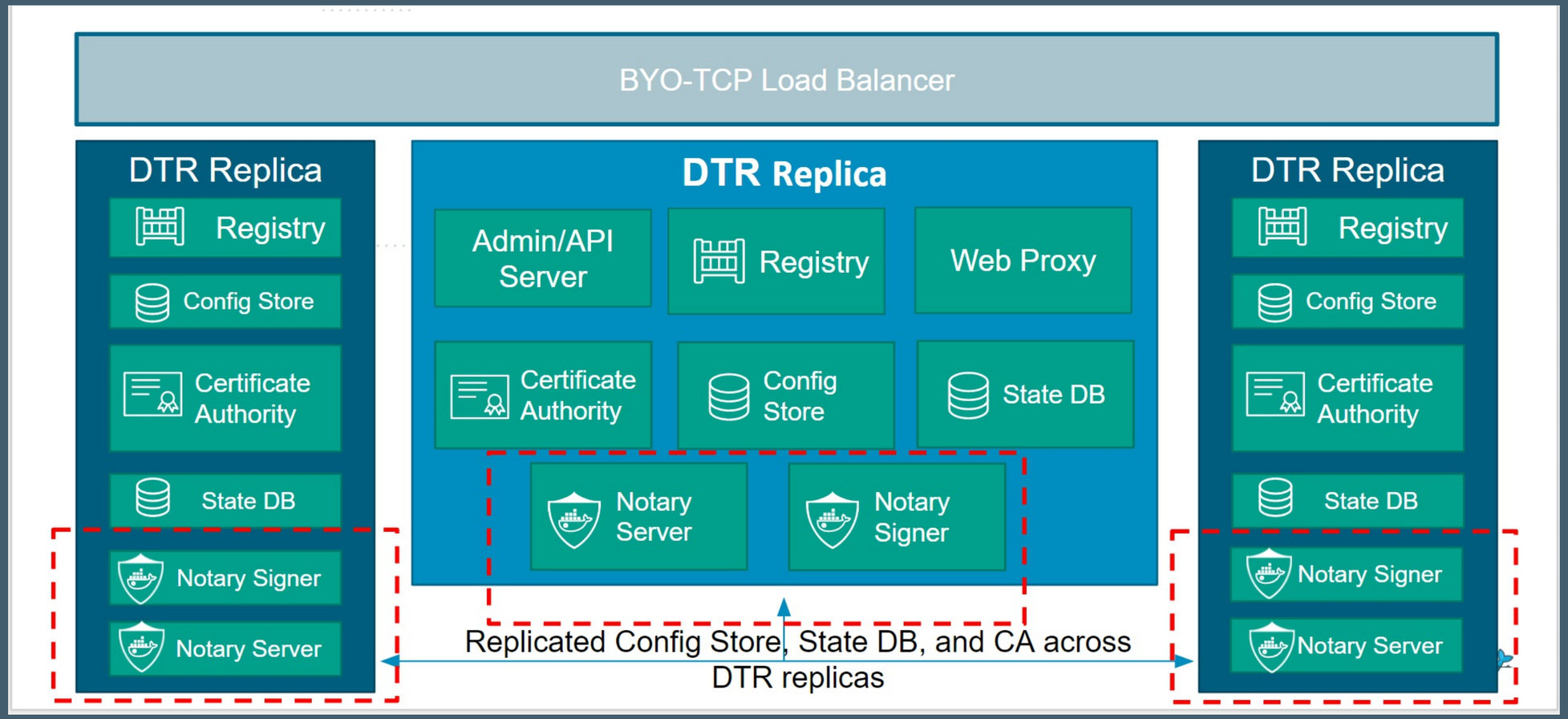
THE SOFTWARE SUPPLY CHAIN

- Image Creation
- Image Distribution
- Container Execution

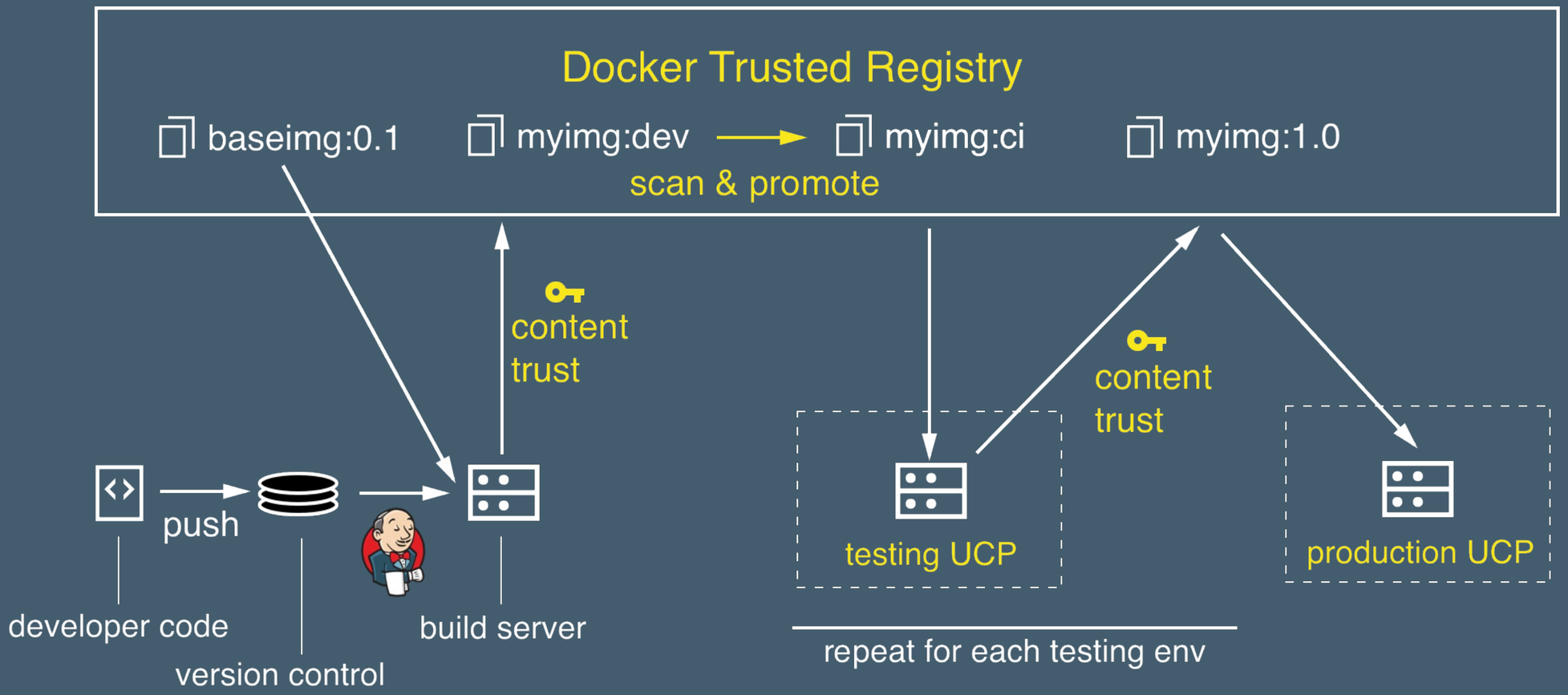
DTR KEY FEATURES

- 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:

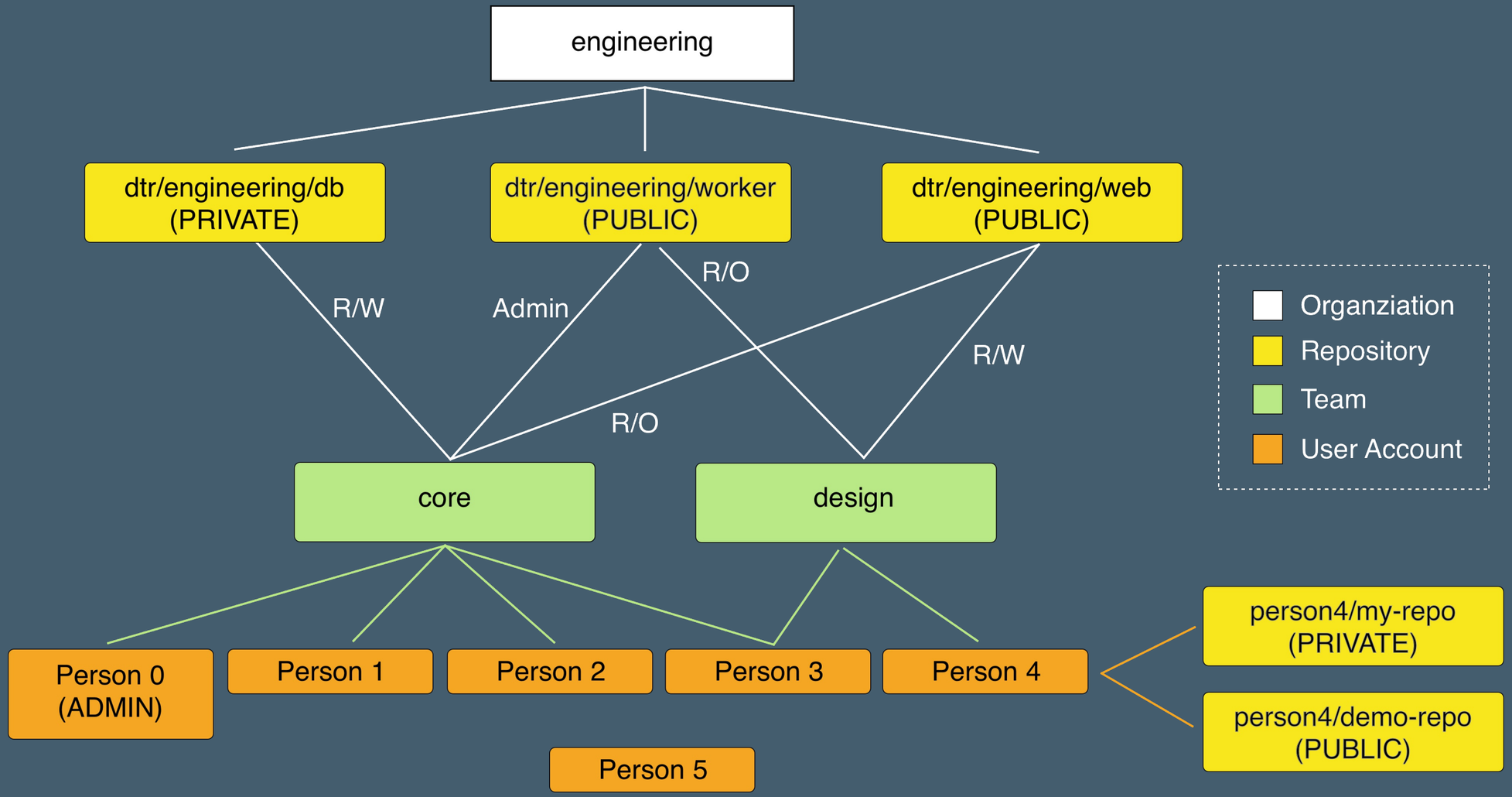
- Understand and manipulate Role Based Access Control in DTR

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	• Anyone can pull • Visible to all • Push by owner	• Only visible to owner & admins
Org-Owned	• Anyone can pull • Visible to all • Push by R/W team	• Must be R/W or R/O team to see repo

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

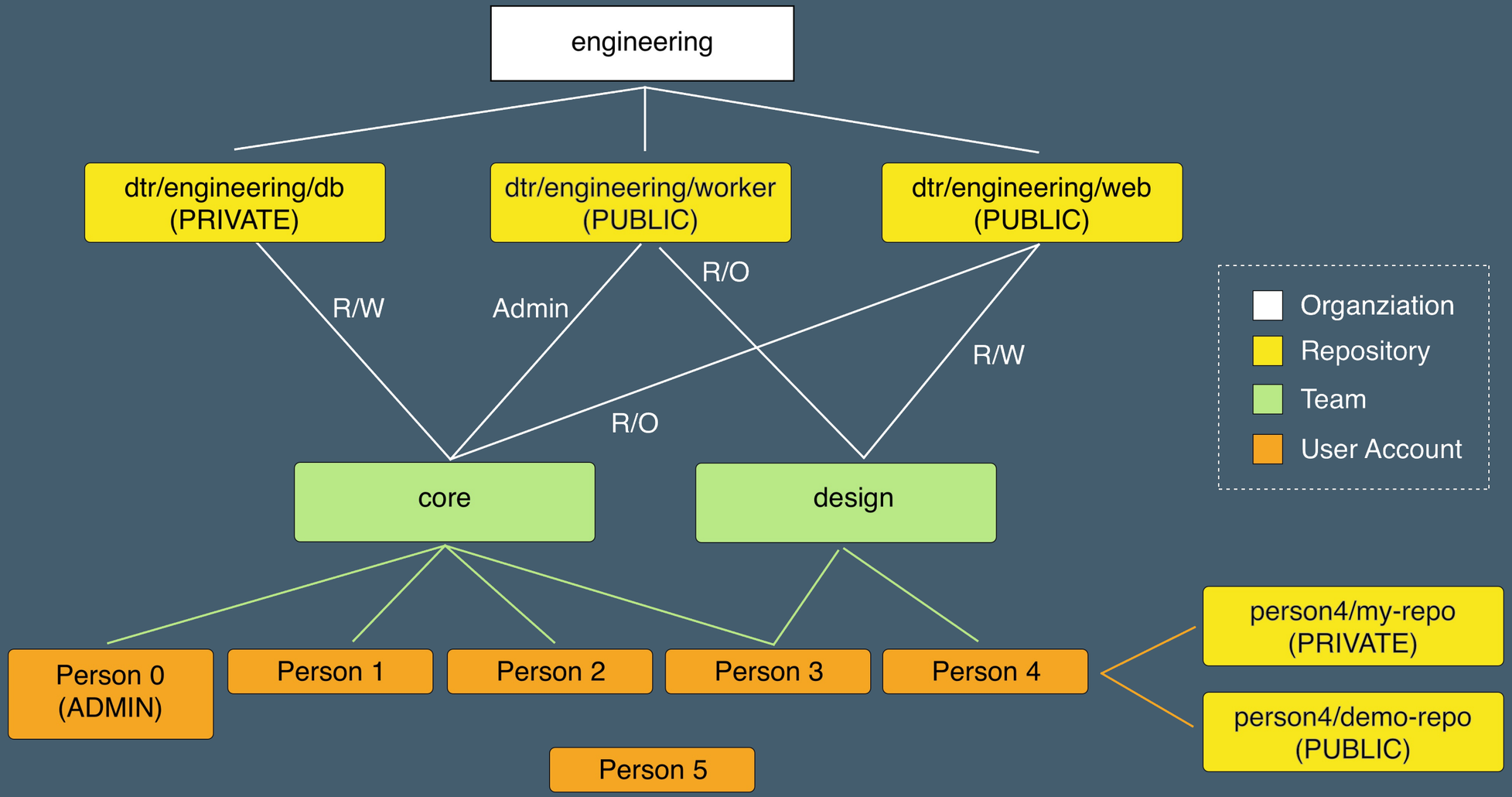
USER TOKEN MANAGEMENT

- 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

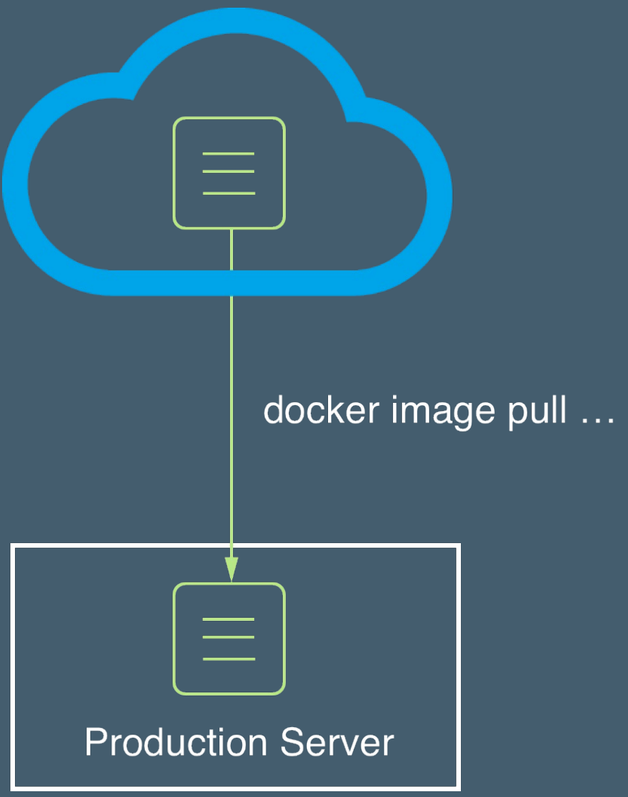
By the end of this module, learners will be able to:

- 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

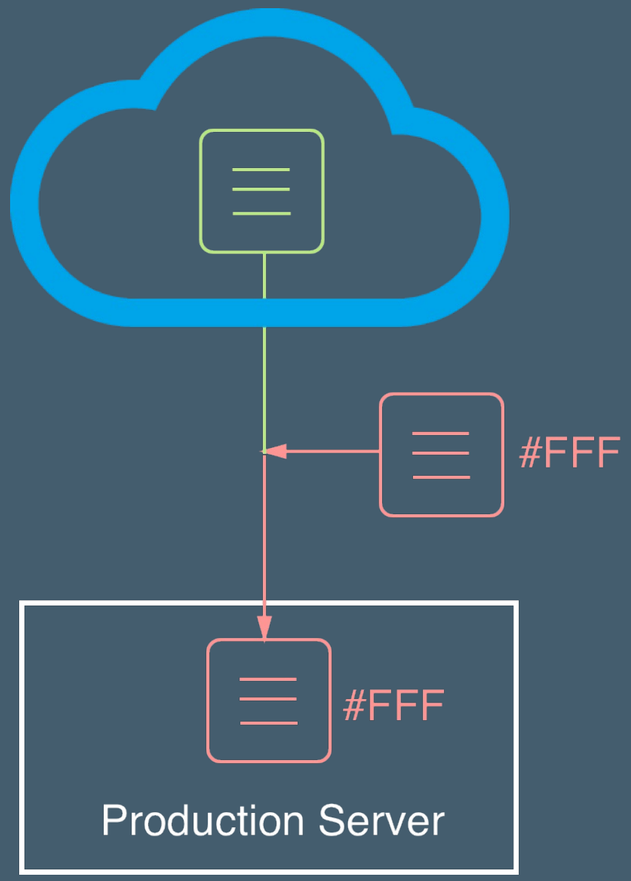
WHAT IMAGE DO YOU MEAN?

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

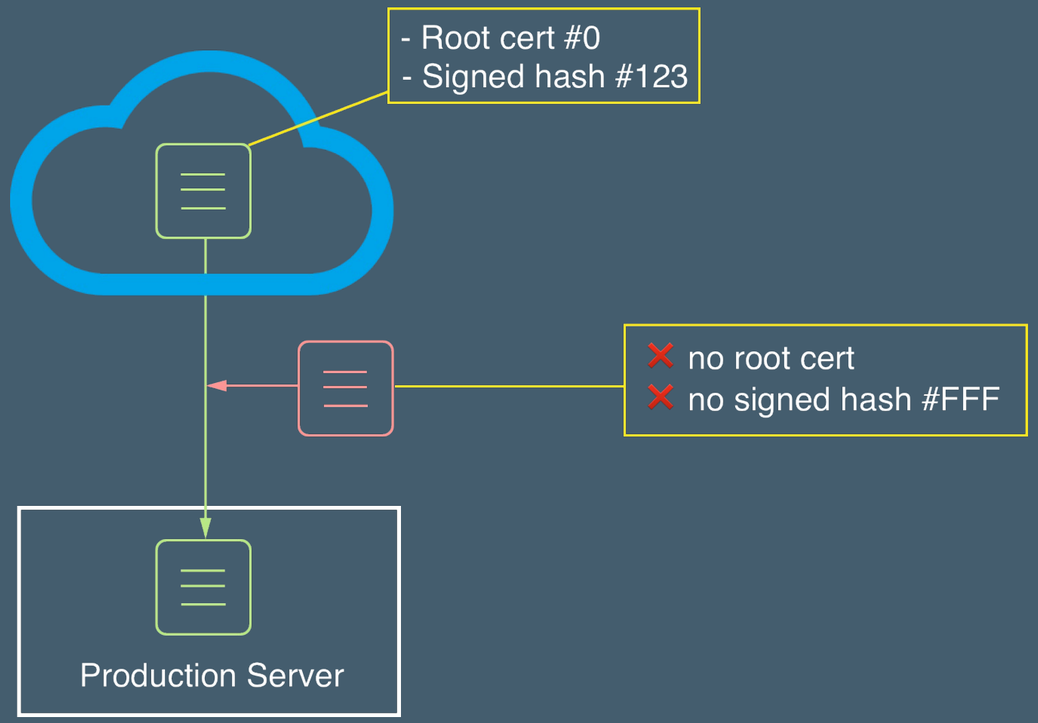
DOWNLOADING SOFTWARE



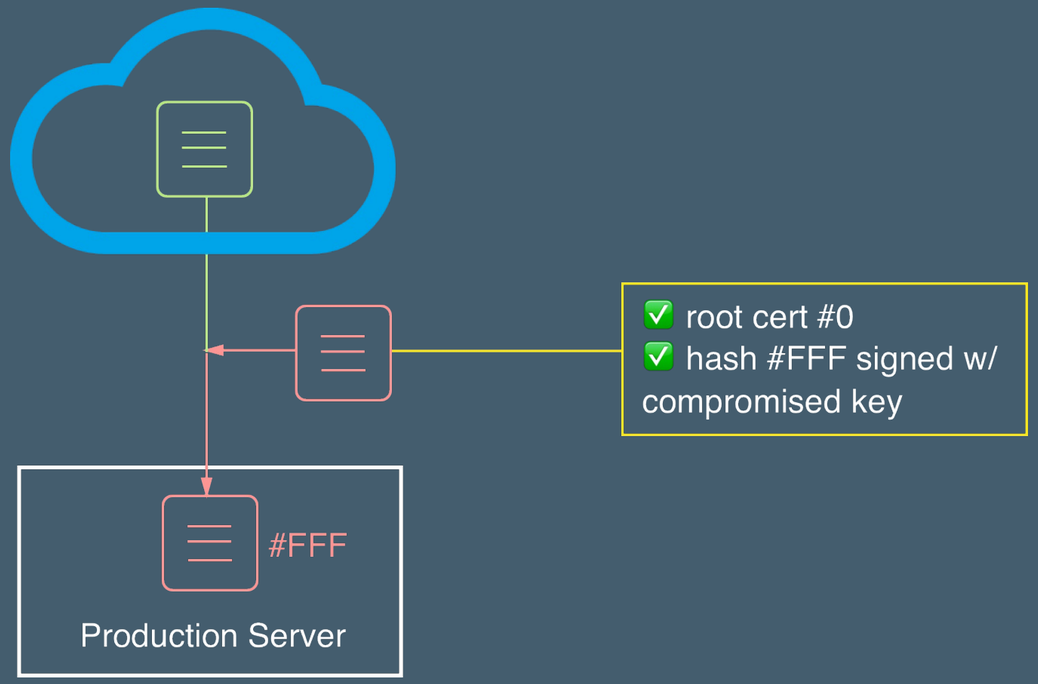
SIMPLE INJECTION ATTACK



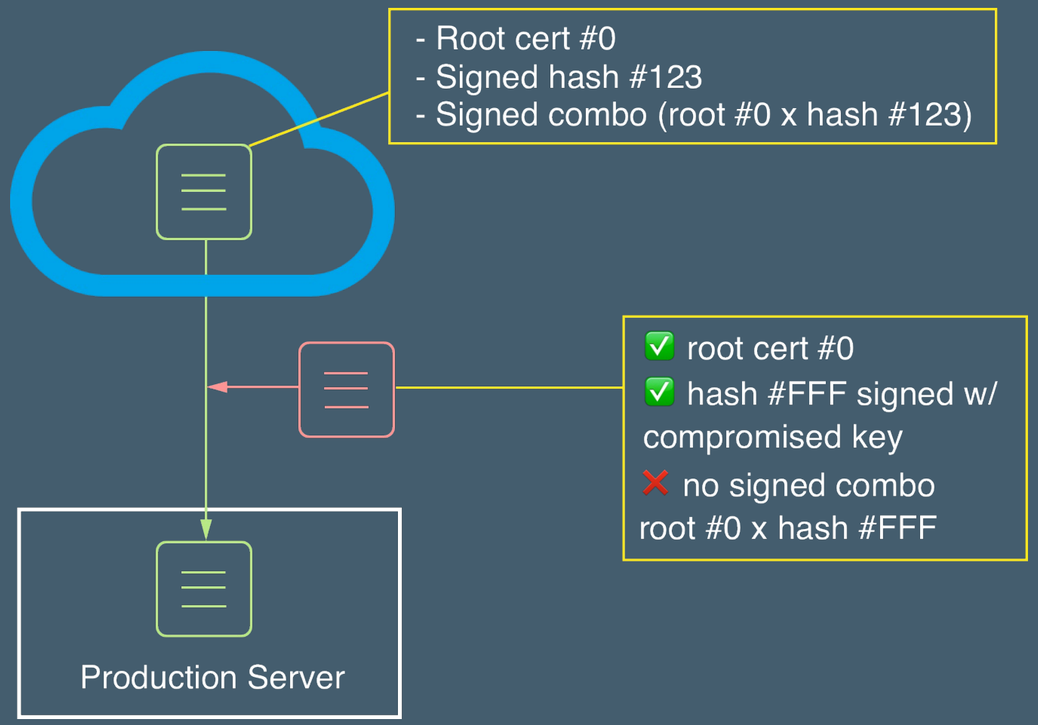
SIGNED CONTENT



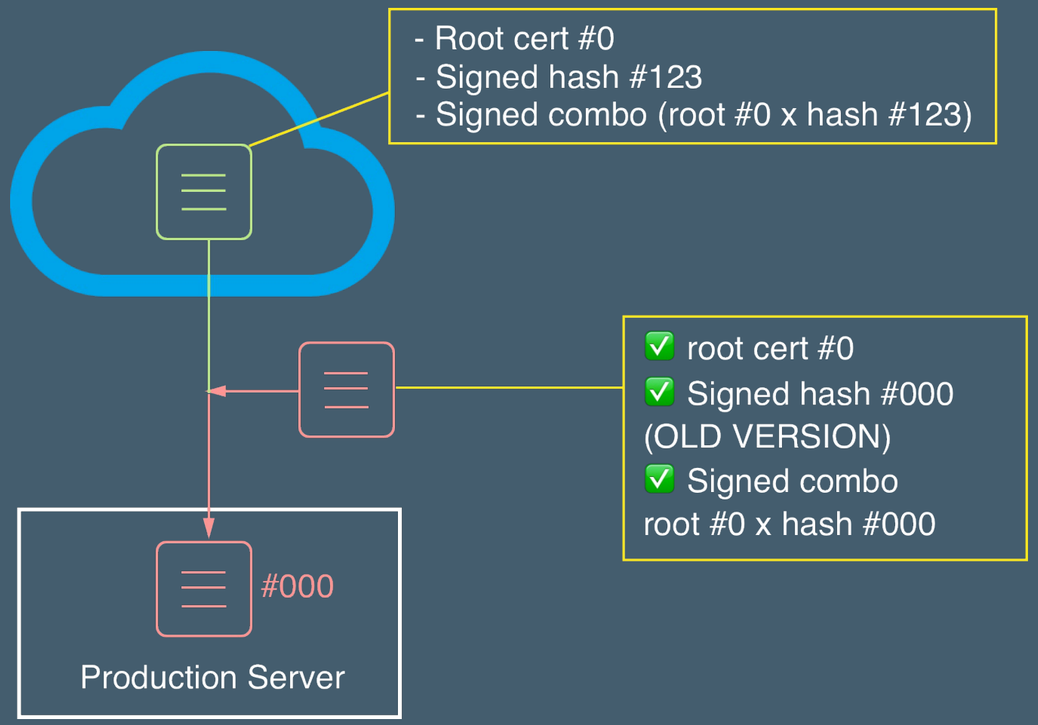
MIX AND MATCH ATTACK



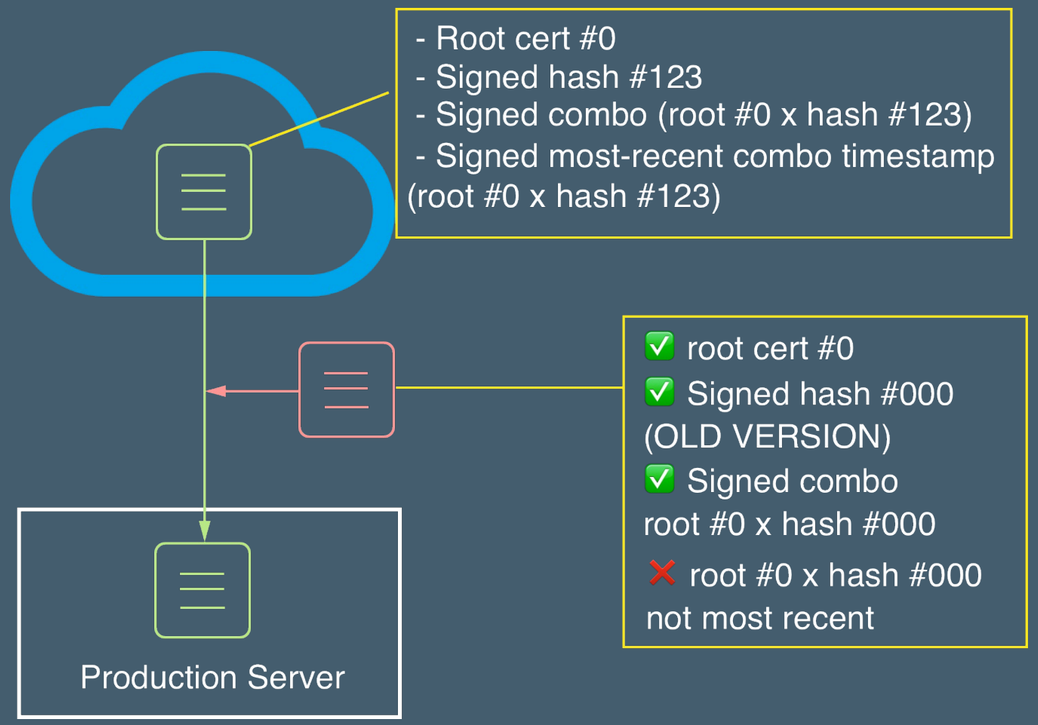
METADATA VALIDATION



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:

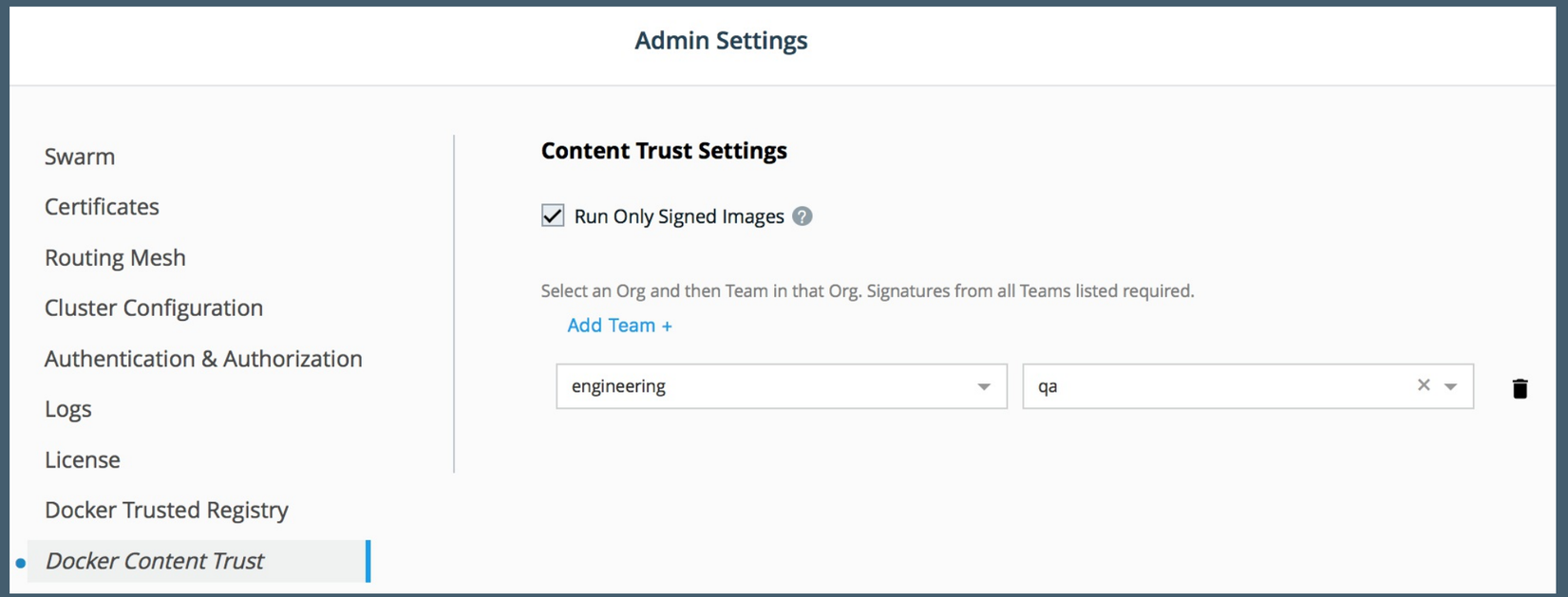
- docker image push
- docker image pull
- docker image build
- docker container create
- docker container run
- docker service create

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



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/2gAgIo3
- 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

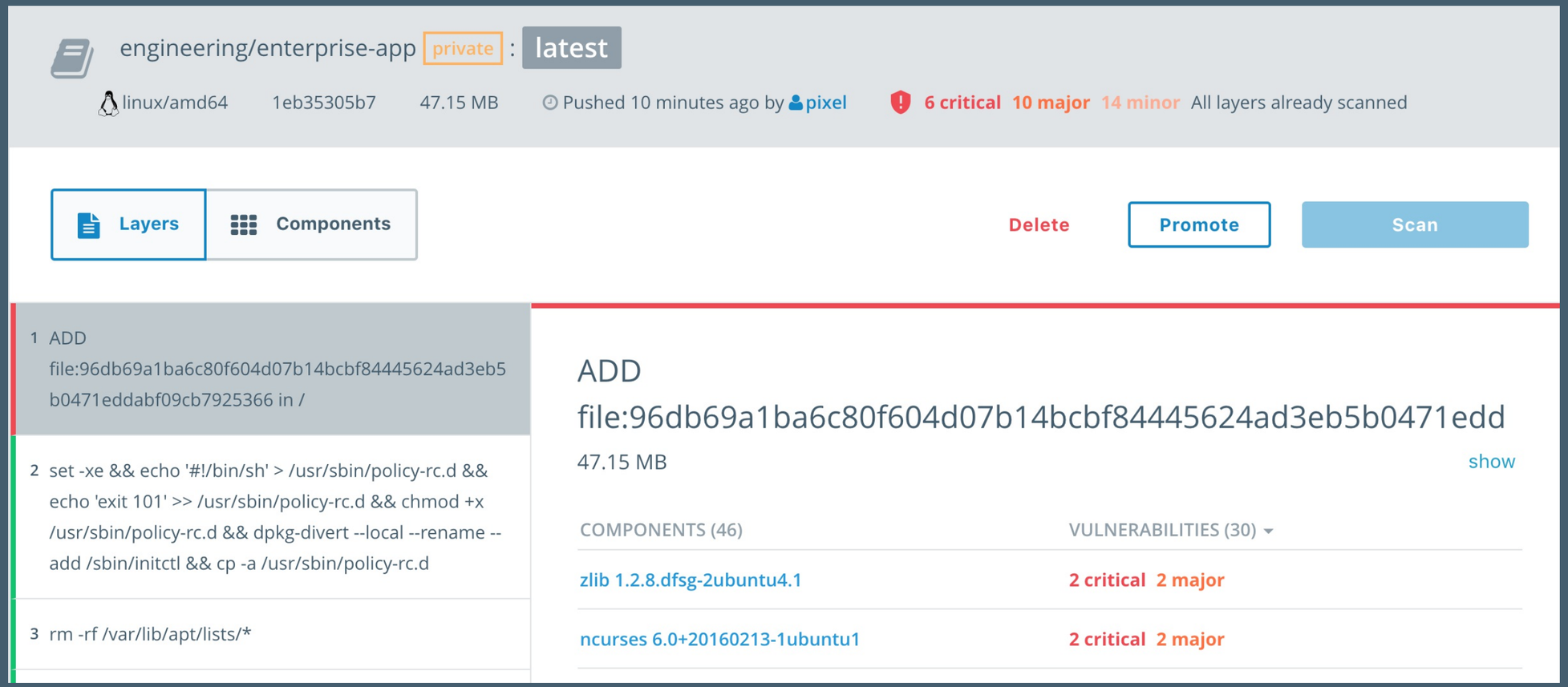
WHAT'S IN YOUR IMAGES?

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

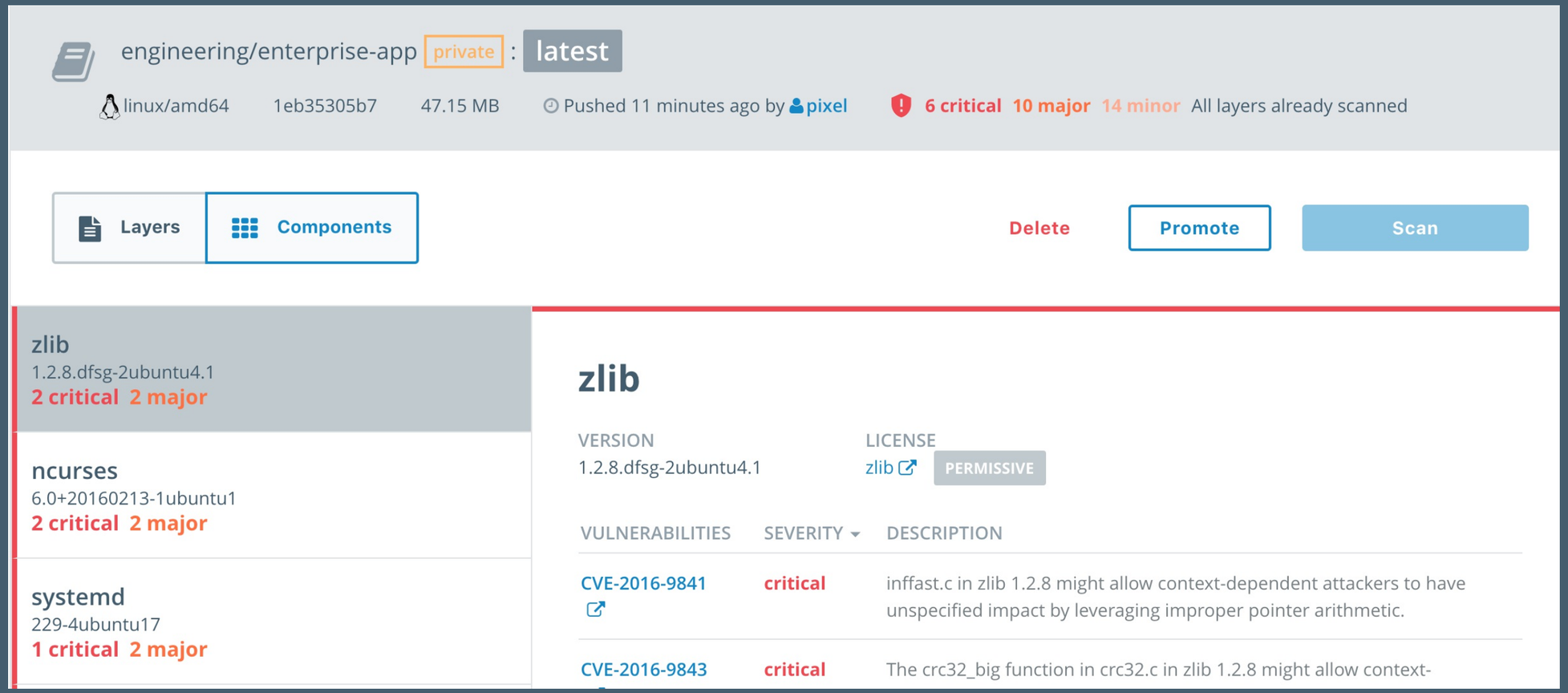
DTR SECURITY SCANNING FLOW

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/

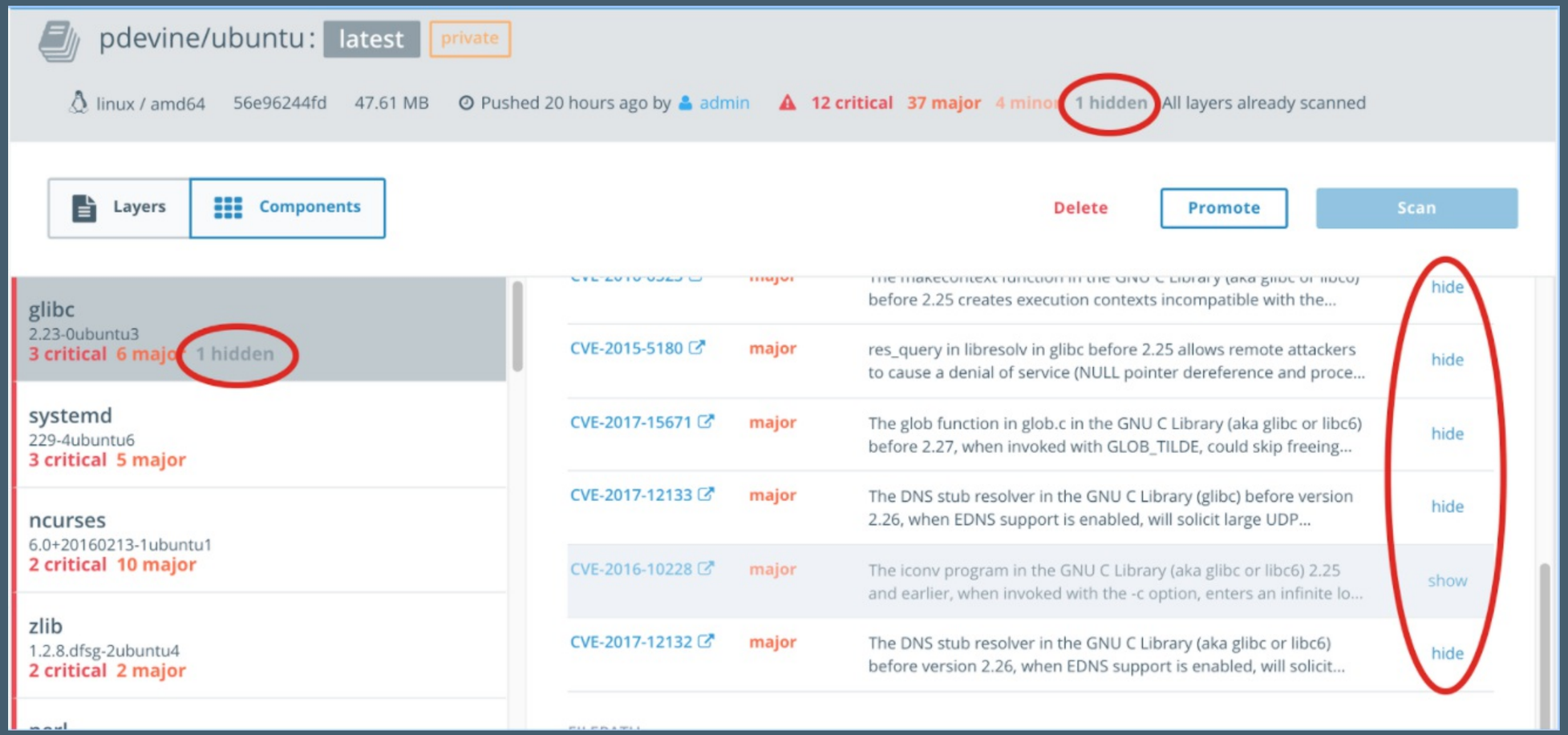
SCAN RESULTS PER LAYER



SCAN RESULTS PER COMPONENT

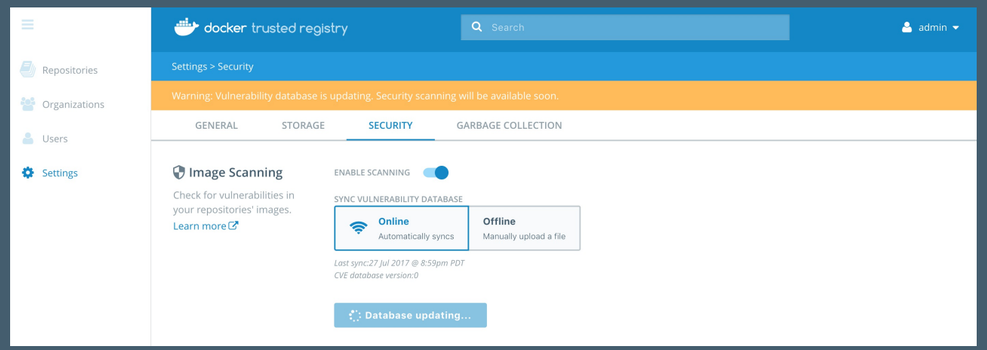


VULNERABILITY OVERRIDES



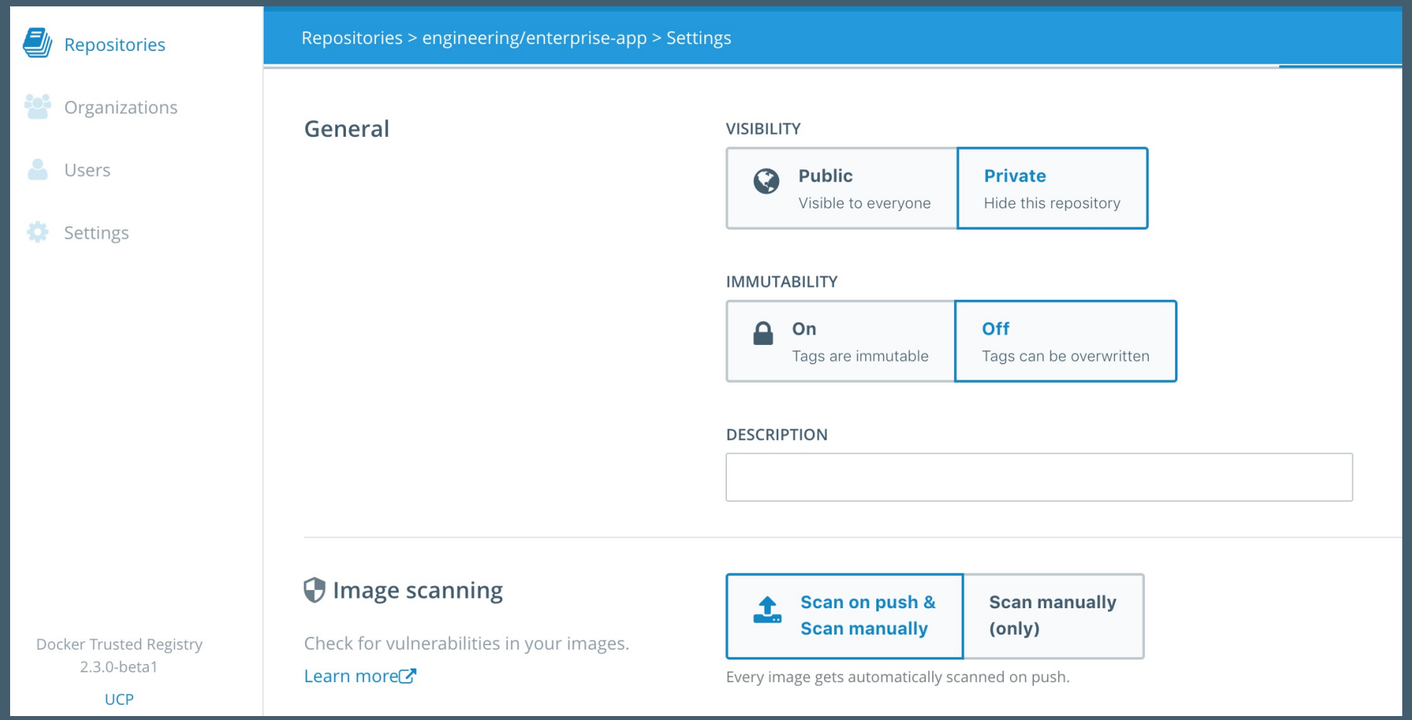
- Allows vulnerabilities to be hidden
- Matches based on different components

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



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

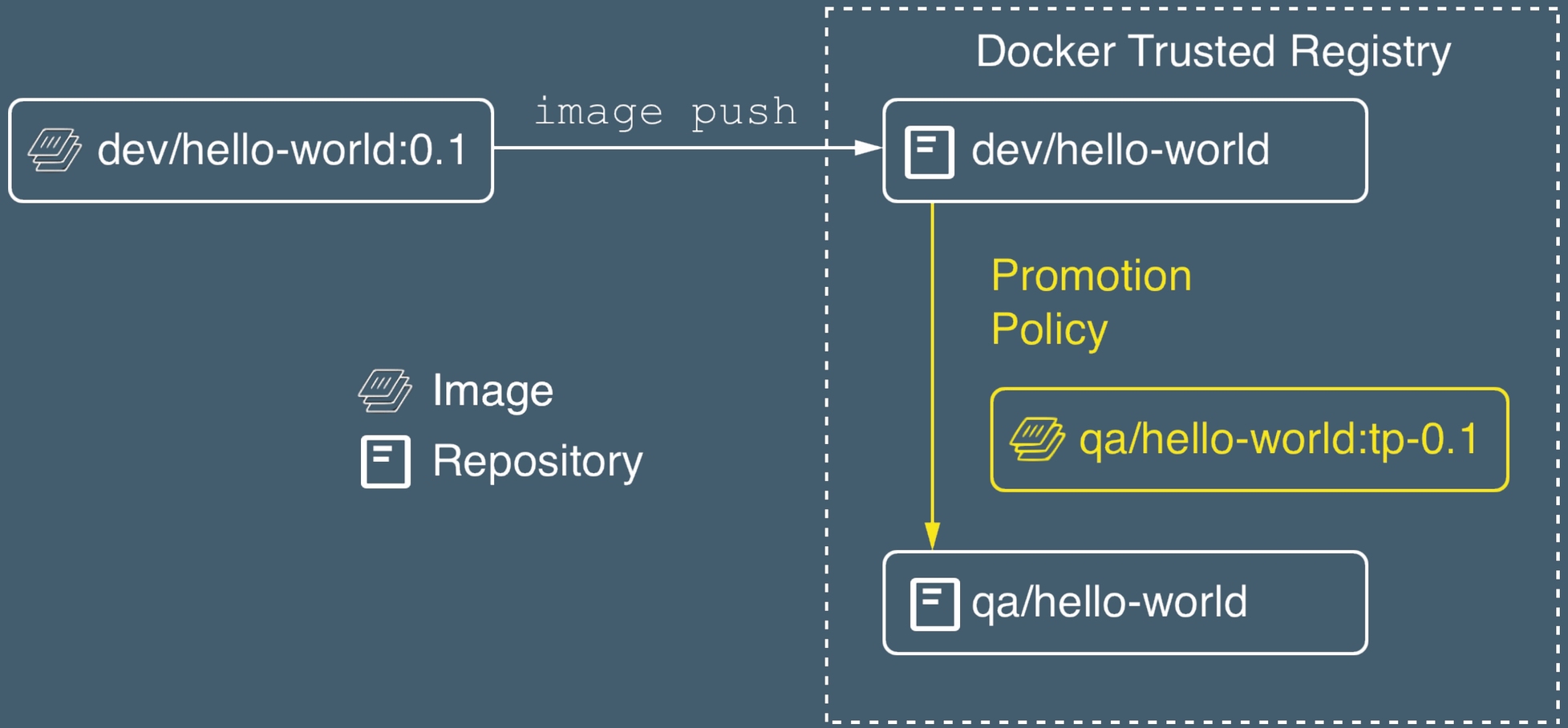


IMAGE PROMOTION POLICIES

- 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

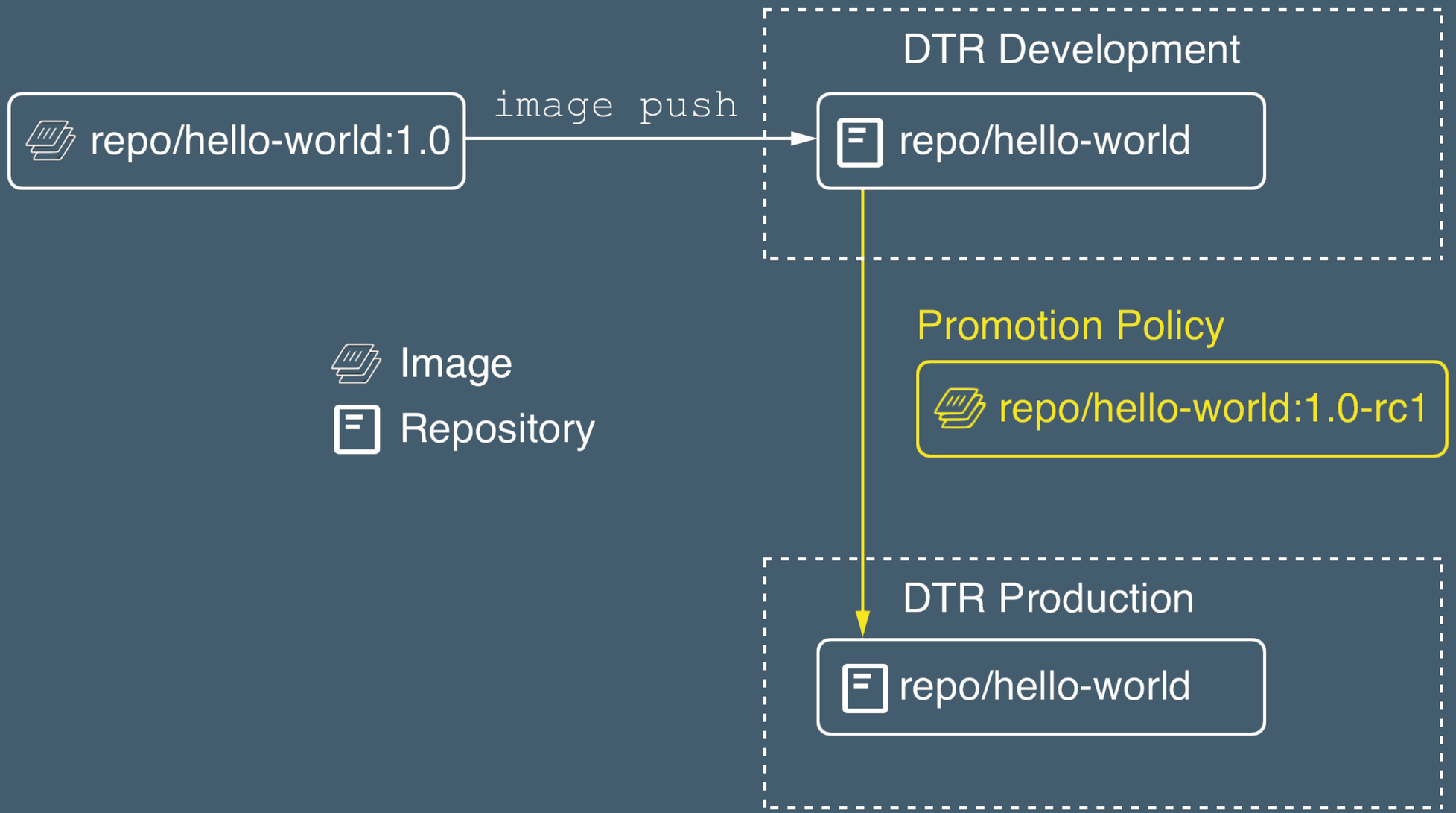
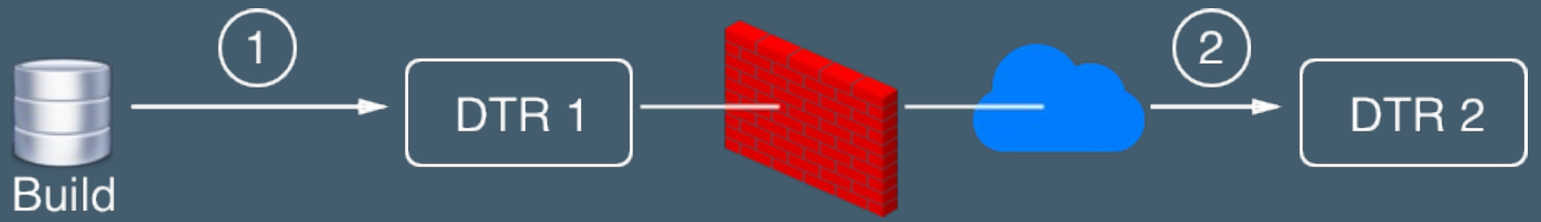


IMAGE MIRRORING - PUSH BASED



- 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

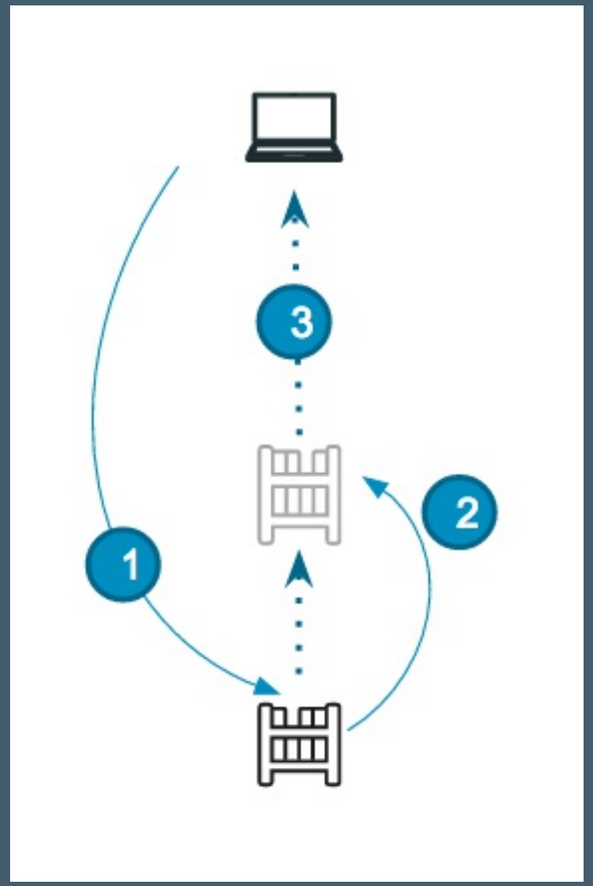
IMAGE CACHING

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

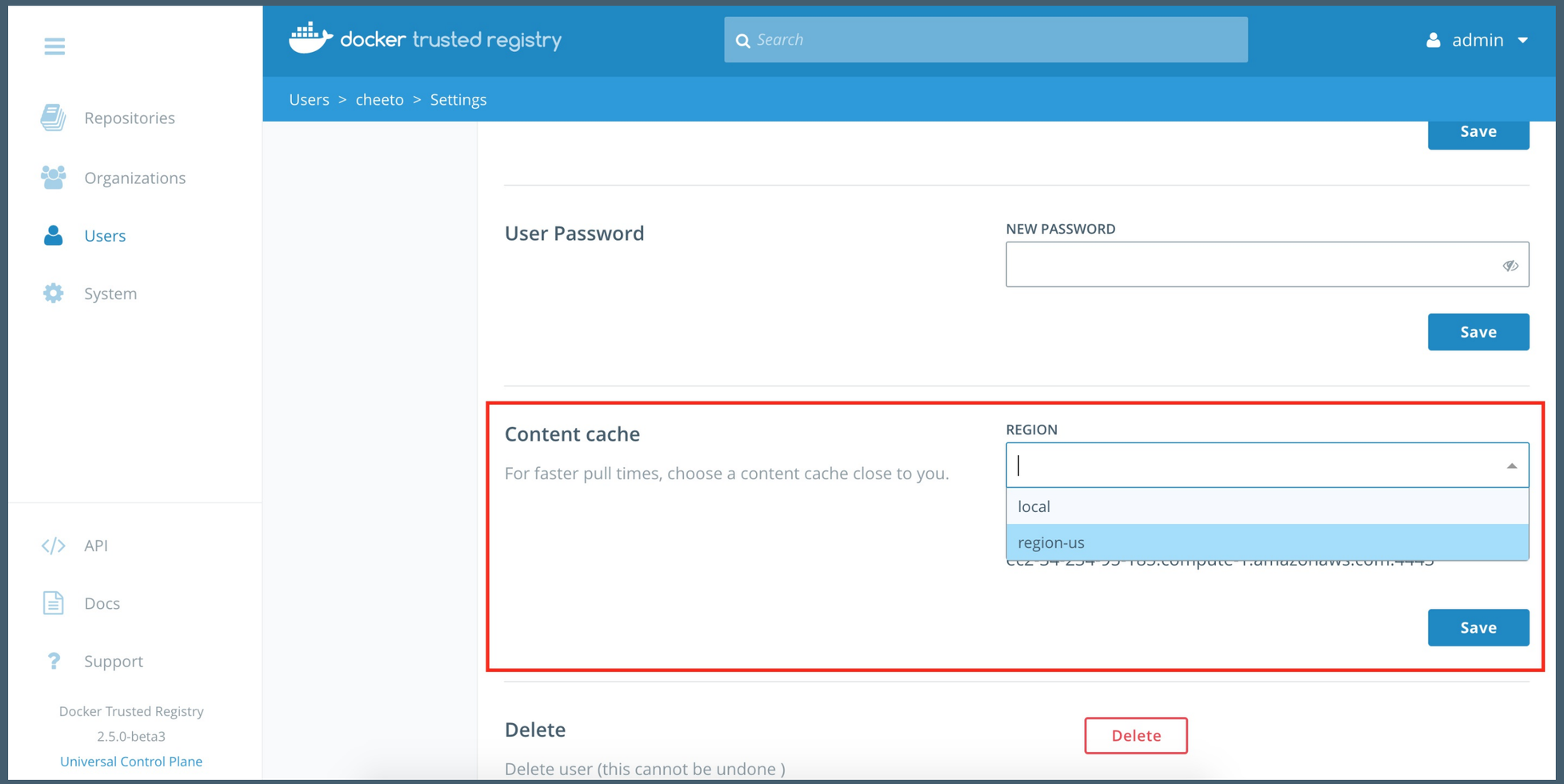
- In DTR, click API
- Expand /api/v0/content_caches POST form
- Click Try Me Out!
- Specify the name and host for your cache:

```
{
 "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



CACHE TOPOLOGY

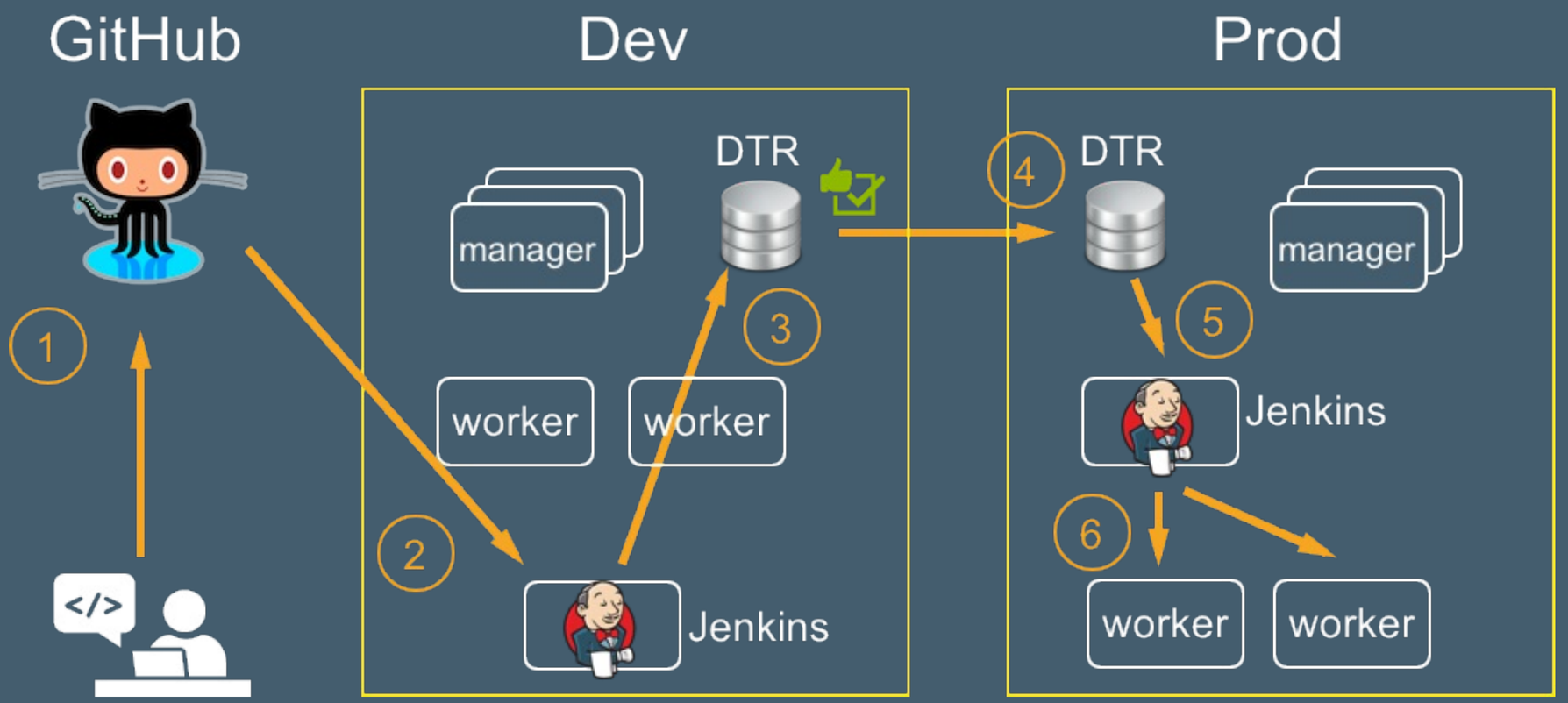
- Cache chaining possible
- Don't chain caches more than two deep
- Do run the cache as an HA service

FURTHER READING

- DTR cache overview: https://dockr.ly/2HkS8Uh
- Use a cache: https://dockr.ly/2vGzHIi

OPERATIONS SIGNATURE ASSIGNMENT

THE USE CASE


GitHub
Dev
Prod
DTR
manager
worker
worker
Jenkins
1
2
3
4
DTR
manager
5
Jenkins
6
worker
worker
</>

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

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