A NOTE ON PEDAGOGY

- Docker believes in learning by doing, with support.
- The course is lab driven with lecture.
- Work with your colleagues to solve problems, and don't hesitate to interrupt the lecture for clarification.

SESSION LOGISTICS

- 2 days duration
- mostly exercises
- regular breaks

ASSUMED KNOWLEDGE AND REQUIREMENTS

- Familiarity with using the Linux command line

YOUR LAB ENVIRONMENT

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

AGENDA

- Fundamental Containerization
  - Container Basics
  - Creating & Managing Images
  - Docker System Management
  - Data Volumes
  - Docker Plugins
- Fundamental Orchestration
  - Basic Networking
  - Docker Compose
  - Docker Swarm
  - Docker Secrets
- ...plus other odds and ends.

docker

INTRODUCING DOCKER

WHAT IS COMPLEXITY COSTING US?

Dependency conflicts, infrastructure mismatches, and lack of scalability are all examples of problems with

STANDARDIZATION AND ENCAPSULATION.

Luckily, this problem is not new.

Devops circa 1912

Photo Roel Hemkes, CC-BY 2.0

Photo Jay Phagan, CC-BY 2.0

Photo Roy Luck, CC-BY 2.0

Encapsulation eliminates friction across infrastructure, and standardization facilitates scale.

DEPLOYMENT NIGHTMARE

Static Website	?	?	?	?	?	?	?
Web Frontend	?	?	?	?	?	?	?
Background Workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's Laptop	Customer's Servers

ANY APP, ANYWHERE

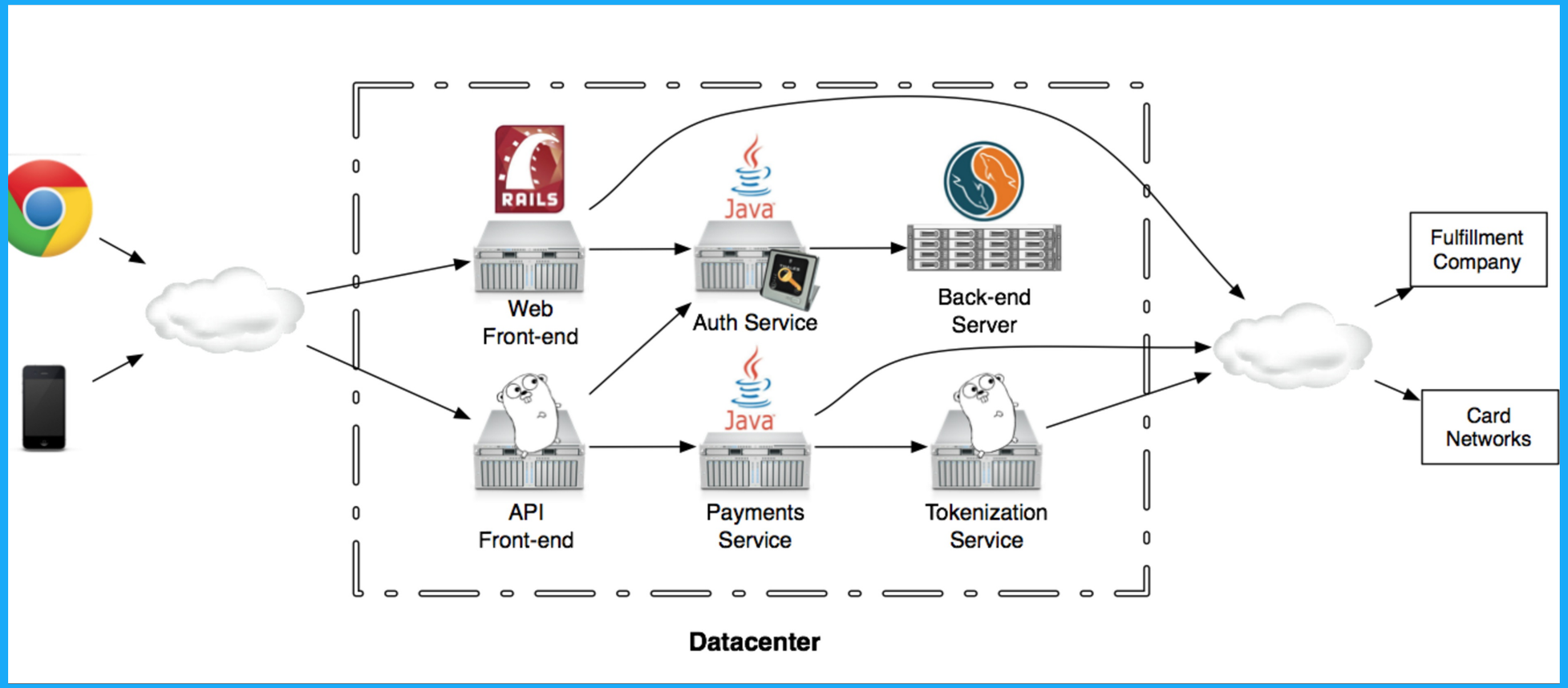
Static Website							
Web Frontend							
Background Workers							
User DB							
Analytics DB							
Queue							
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's Laptop	Customer's Servers

SECURITY

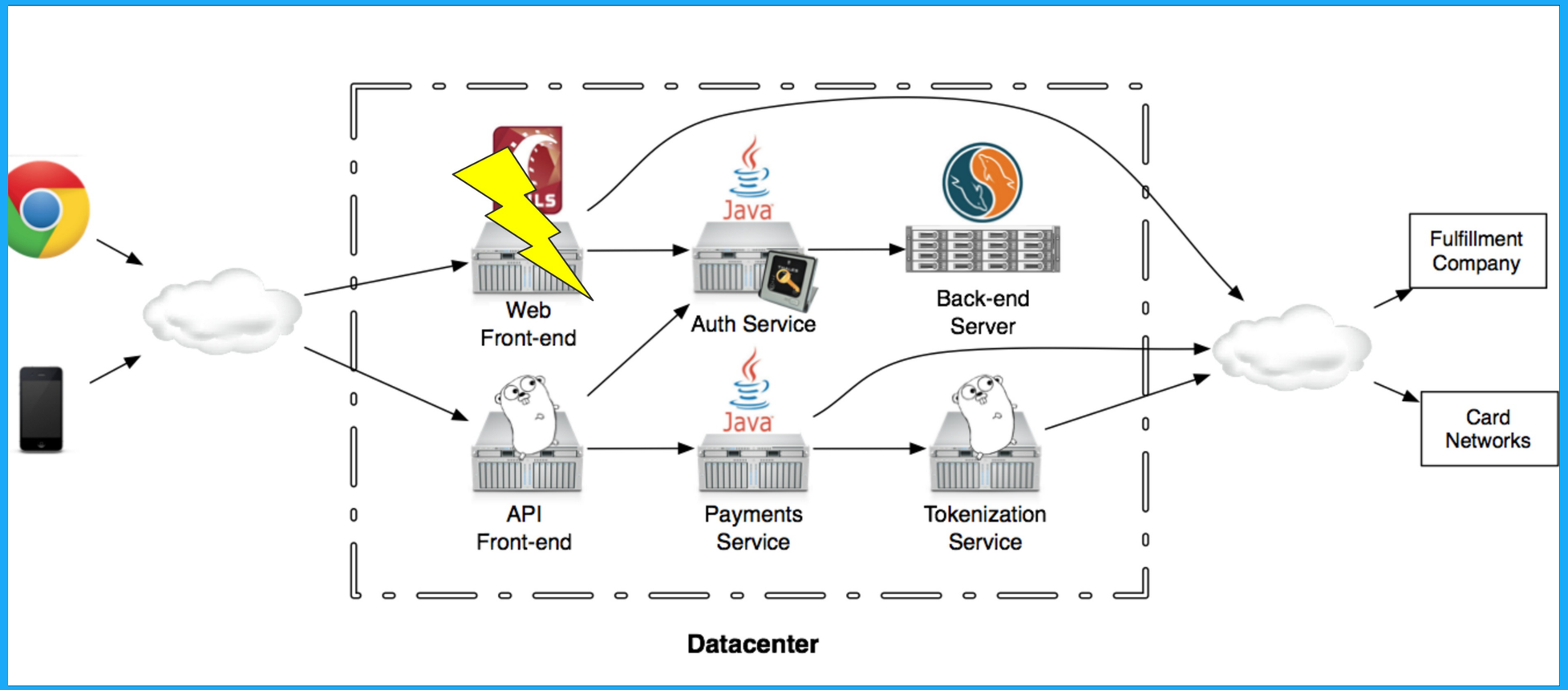
*"Gartner asserts that applications deployed in containers **are more** secure than applications deployed on the bare OS."*

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

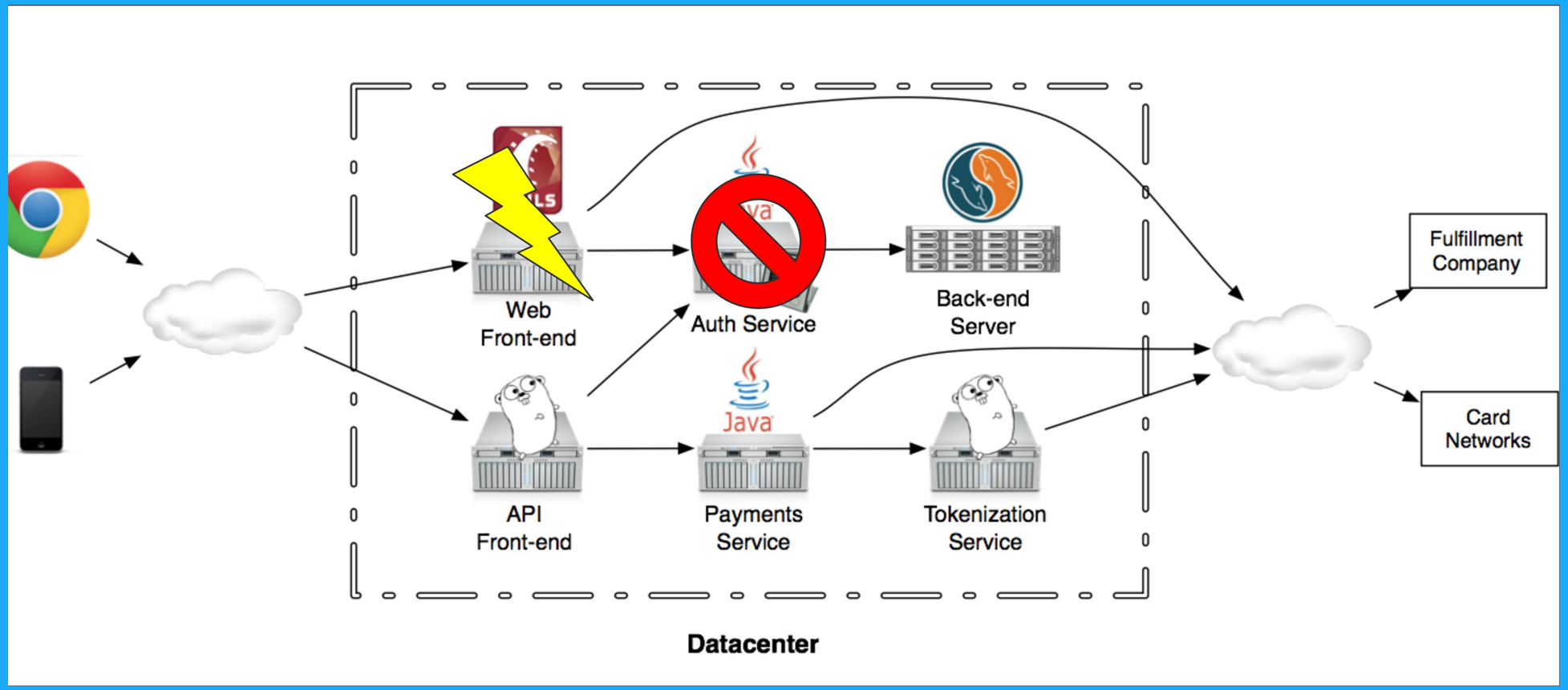
WHAT DOES THIS HAVE TO DO WITH DOCKER...?



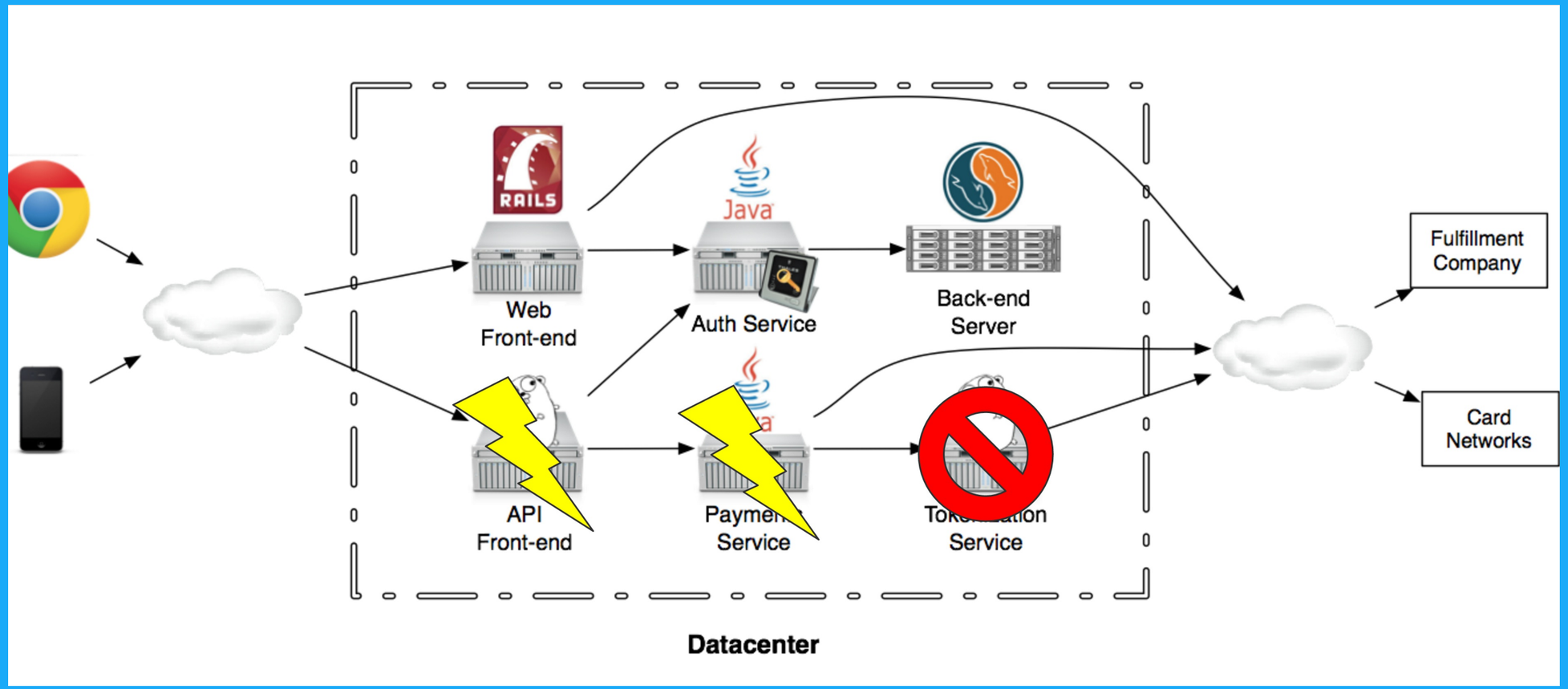
WHAT DOES THIS HAVE TO DO WITH DOCKER...?



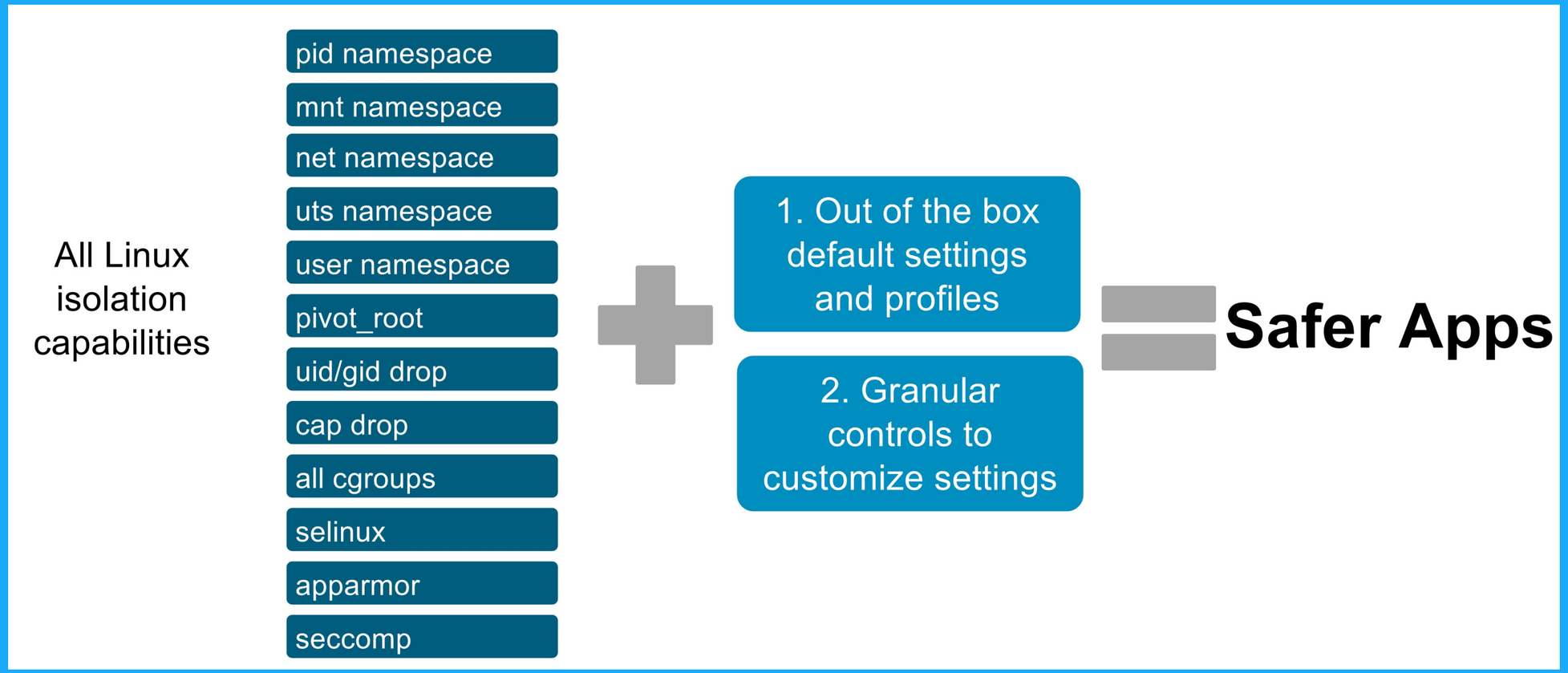
WHAT DOES THIS HAVE TO DO WITH DOCKER...?



WHAT DOES THIS HAVE TO DO WITH DOCKER...?

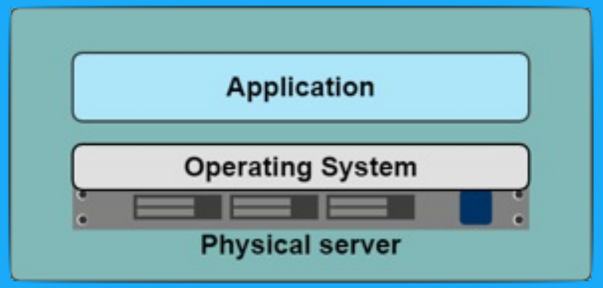


SAFER APPLICATIONS



ENCAPSULATION I: PHYSICAL SERVERS

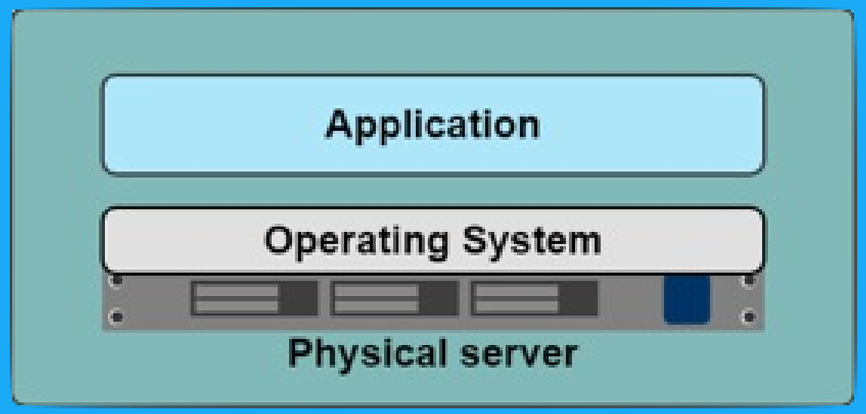
IN THE DARK AGES



One application, one physical server

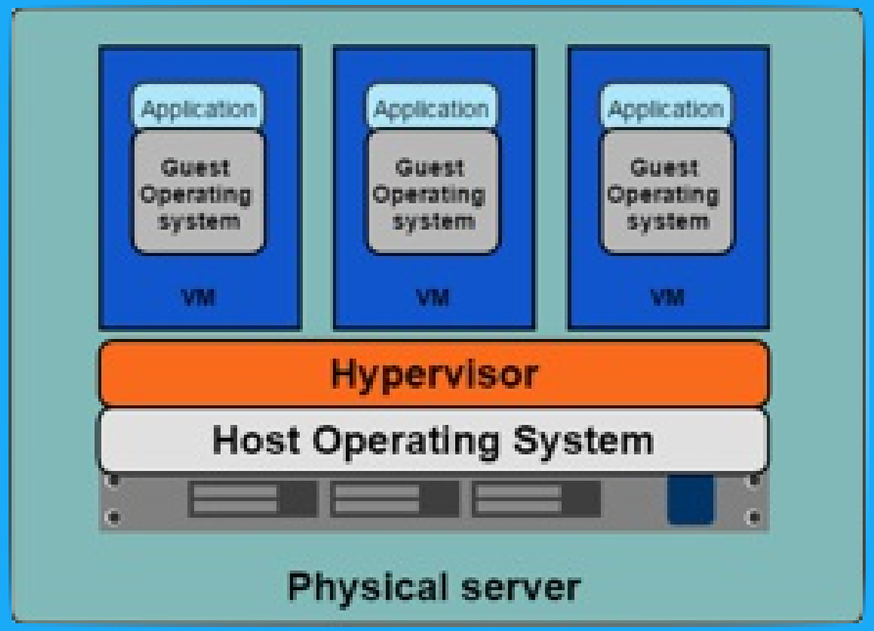
ENCAPSULATION I: PHYSICAL SERVERS

LIMITS OF PHYSICAL ENCAPSULATION



- Slow deployment
- Huge costs
- Provisioning speed limited by physical logistics
- Difficult to scale
- Difficult to migrate
- Vendor lock-in

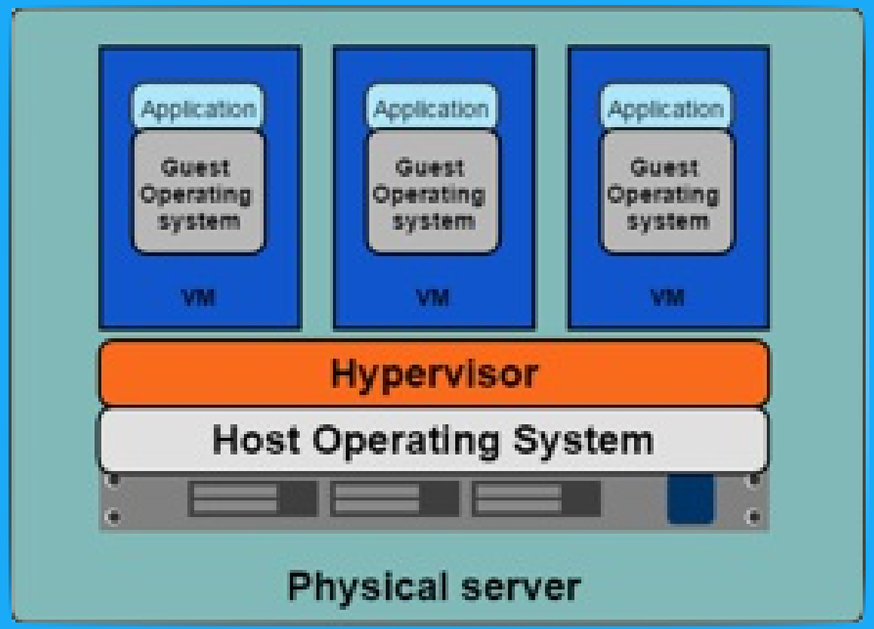
ENCAPSULATION II: VM



- Multiple apps on one server
- Elastic, real time provisioning
- Scalable pay-per-use cloud models viable

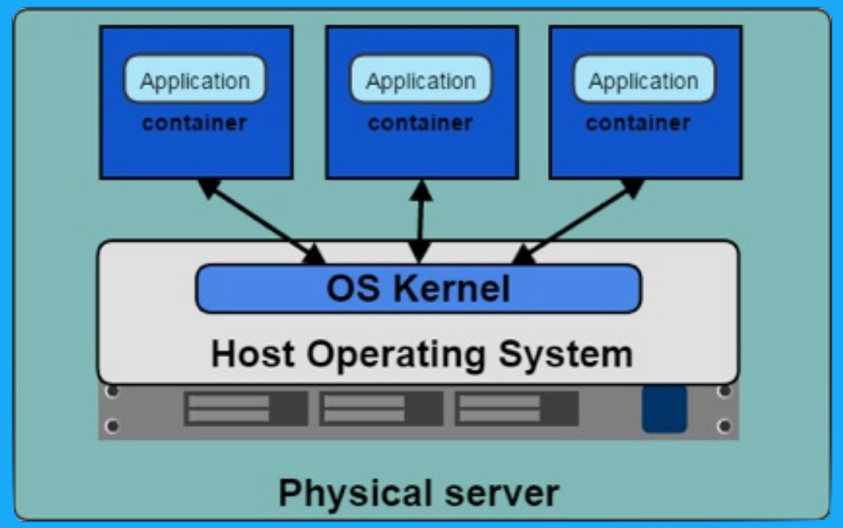
ENCAPSULATION II: VM

VM LIMITATIONS



- VMs require CPU & memory allocation
- Significant overhead from guest OS

ENCAPSULATION III: CONTAINERS



Containers leverage kernel features to create extremely light-weight encapsulation:

- Kernel namespaces
- Network namespaces
- Linux containers
- cgroups & security tools
- Results in faster spool-up and denser servers

The most basic thing Docker provides is a

FRAMEWORK FOR SERVICE ENCAPSULATION

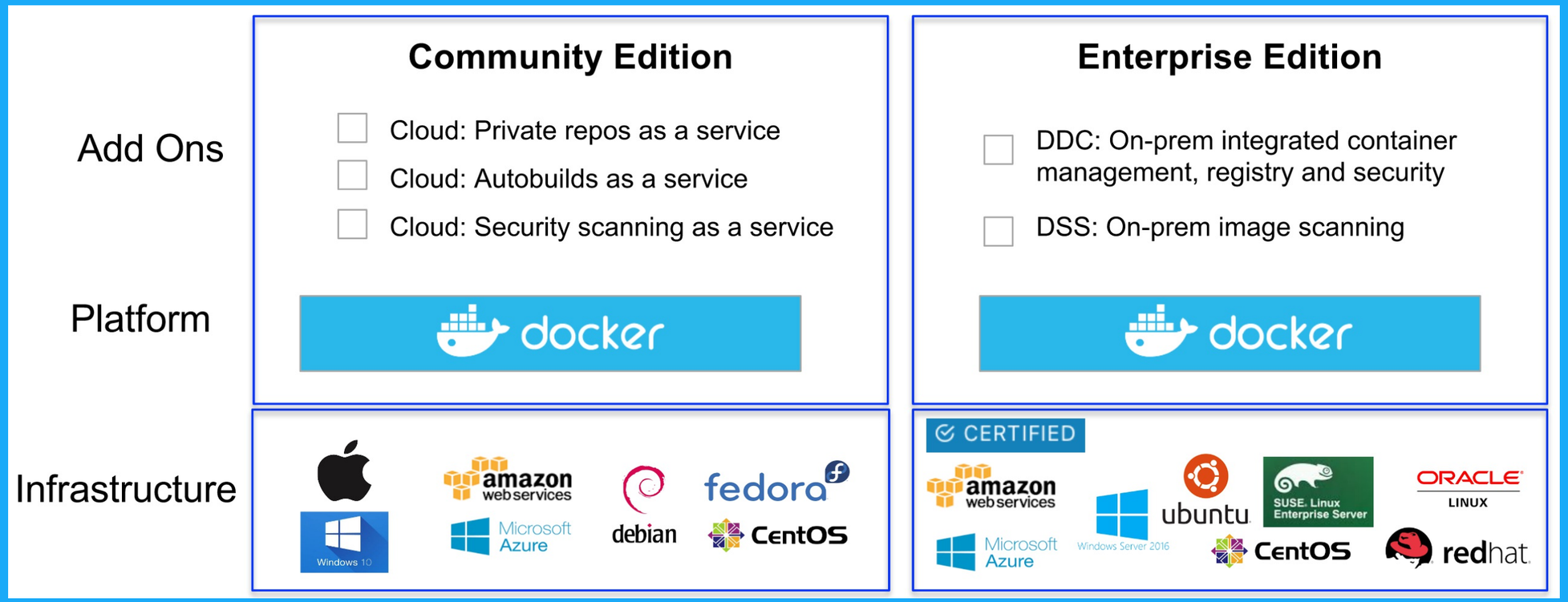
But what are the implications of this for developers, ops, and orgs?

DISTRIBUTED APPLICATION ARCHITECTURE

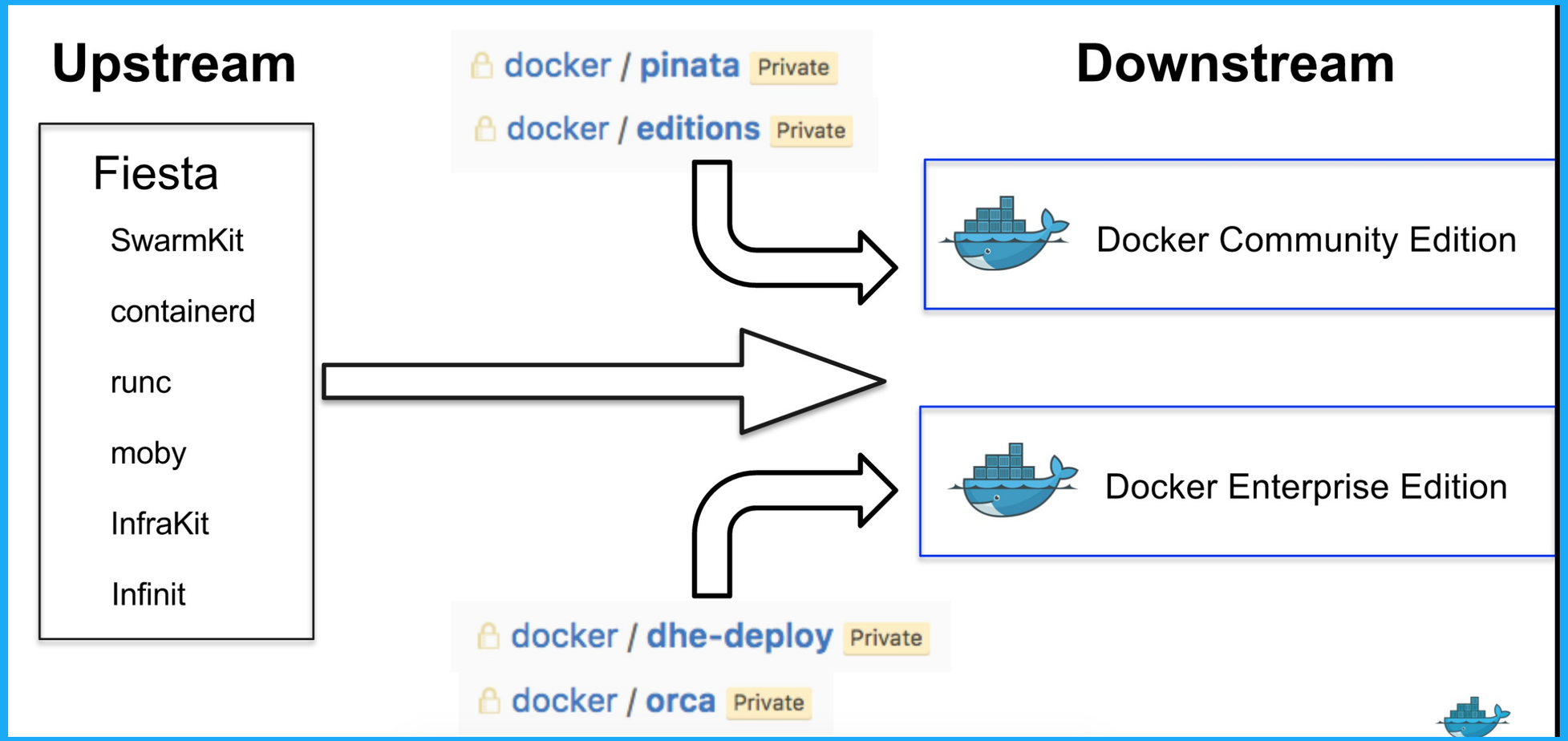
Encapsulation supercharges:

- Monolith Densification
- Service-Based Architecture
- Devops

DOCKER PRODUCT OFFERINGS



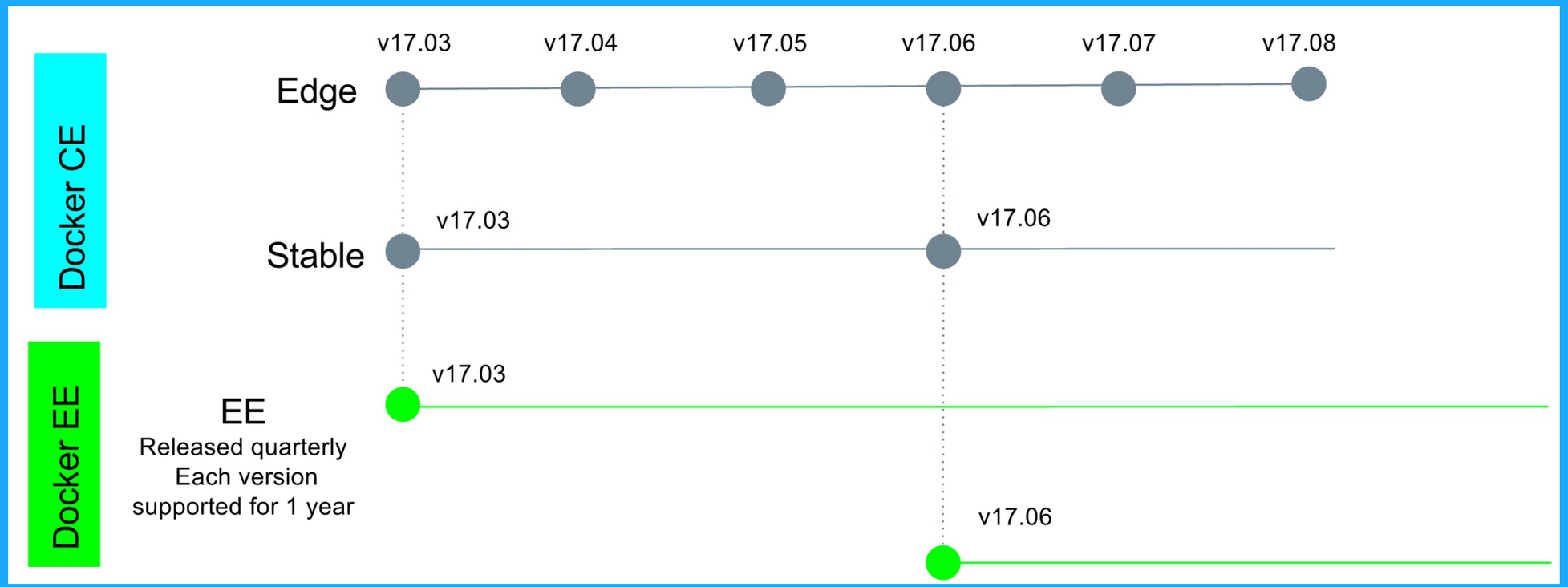
COMPONENTS OF CE/EE



EE SUBSCRIPTION TIERS

	Docker Enterprise Edition		
	Basic	**Standard**	**Advanced**
Container engine and built in orchestration	X	X	X
Image management (private registry, caching)		X	X
Integrated container app management		X	X
Multi-tenancy, RBAC, LDAP/AD		X	X
Integrated secrets mgmt, image signing, policy		X	X
Security Scanning			X

CE/EE RELEASE CADENCE



DOCKER CERTIFIED PRODUCTS

- Infrastructure
  - Integrated to and optimized for infrastructure running Docker
  - Available for Linux and Windows OS, Cloud providers
- Plugins
  - Containerized plugins
  - Assurances for interoperability
  - Cooperative support
- Containers
  - ISV software packages
  - High quality images built with best practices
  - Strong security profile
  - Cooperative support

Certified Plugins & Containers available at Docker Store, store.docker.com

DOCKER TRAINING

- Enablement for all these products
- Deep dives on special topics like networking, security and architecture
- Docker Distributed Application Engineer & Ops Certifications (coming soon!)

e: training@docker.com

w: training.docker.com

PART 1: CONTAINERS

CONTAINERIZATION FUNDAMENTALS

TOPICS

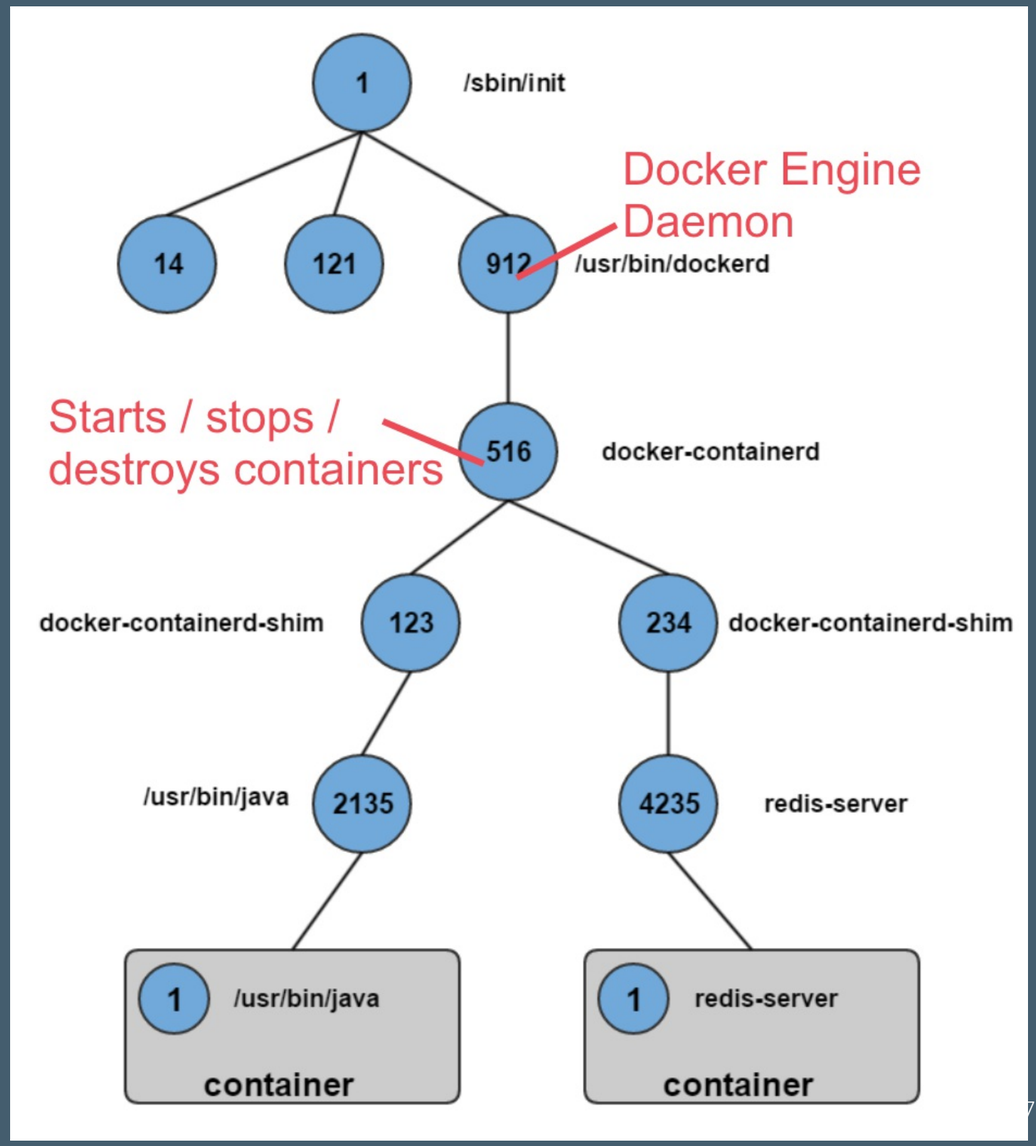
- Containers under the hood
- Starting, stopping and deleting containers
- Inspecting containers
- Executing processes inside running containers

CONTAINERS ARE PROCESSES

Containers are processes sandboxed by:

- Kernel namespaces
- Root privilege management
- System call restrictions
- Private network stacks
- etc

CONTAINER ISOLATION



CONTAINER LOGS

- STDOUT and STDERR for a containerized process
- docker container logs <container name>

EXERCISE: CONTAINER BASICS

Work through:

- Running and Inspecting a Container
- Interactive Containers
- Detached Containers and Logging
- Starting, Stopping, Inspecting and Deleting Containers

In the Docker Fundamentals Exercises book.

CONTAINER BASICS TAKEAWAYS

- Single process with PID 1
- Private & ephemeral filesystem and data

New Syntax:

- docker container run
- docker container rm
- docker container start
- docker container stop & docker container kill
- docker container exec
- docker container attach
- docker container logs
- docker container inspect
- docker container ls

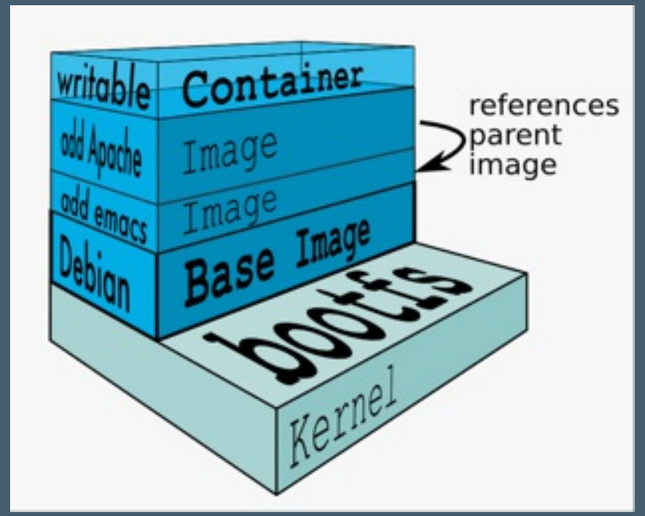
CREATING IMAGES

TOPICS

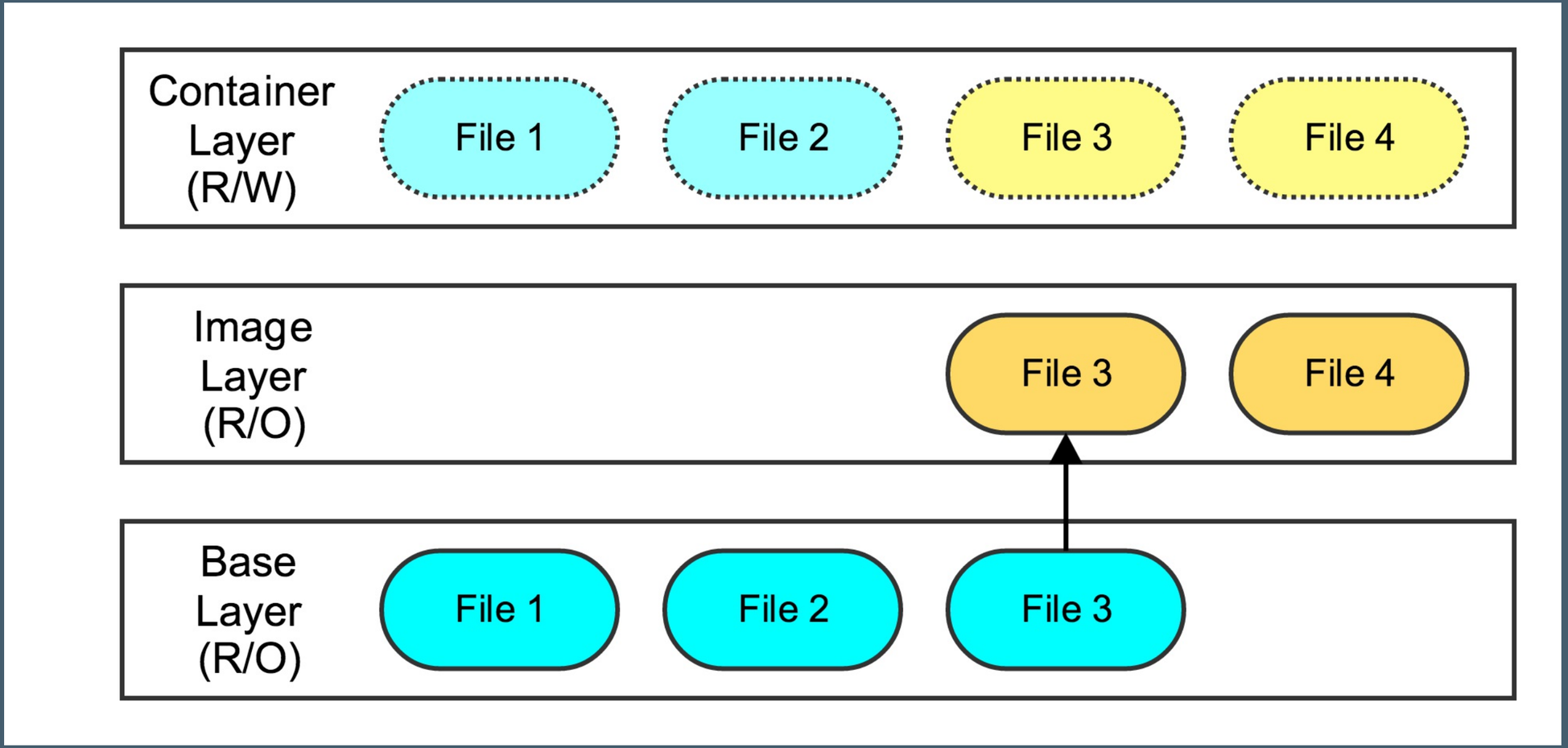
- Layered filesystems
- Creating images
- Dockerfiles & best practice
- Tagging, Namespacing & Sharing images

IMAGES ARE LAYERED FILESYSTEMS

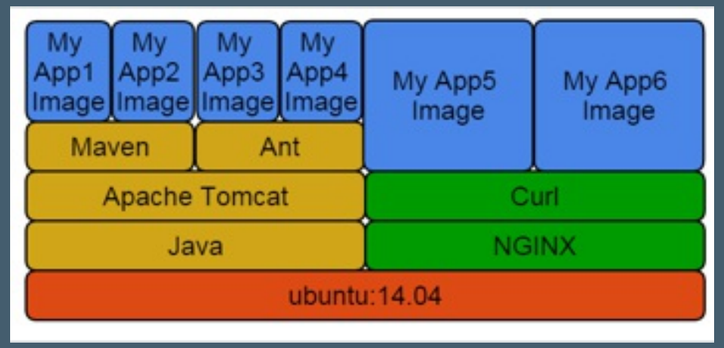
- Provide filesystem for container process
- Image = stack of immutable layers
- Start with a base image
- Add layer for each change



COPY ON WRITE



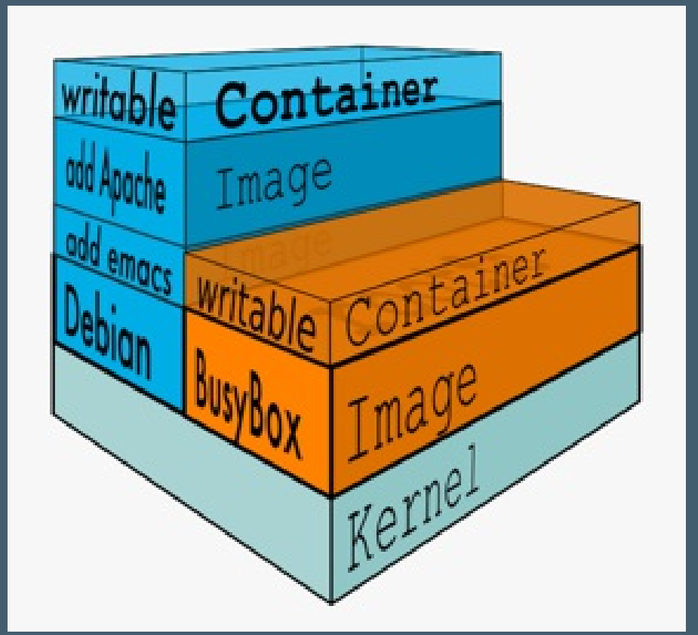
SHARING LAYERS



- Faster download
- Optimize disk and memory usage
- Leverage local cache

THE WRITABLE CONTAINER LAYER

- docker container run creates writable container layer.
- Parent images are read only.
- When changing a file from a read only layer, the copy on write system copies the file into the writable layer.



CREATING IMAGES

Three methods:

- Commit the R/W container layer as a new R/O image layer.
- Define new layers to add to a starting image in a Dockerfile.
- Import a tarball into Docker as a standalone base layer.

COMMITTING CONTAINER CHANGES

- docker container commit saves the container layer as a new R/O image layer
- Pro: build images interactively
- Con: hard to reproduce or audit

EXERCISE: INTERACTIVE IMAGE CREATION

Work through the 'Interactive Image Creation' exercise in the Docker Fundamentals Exercises book.

DOCKERFILES

- Content manifest
- Provides image layer documentation
- Enables automation (CI/CD)

DOCKERFILES

- FROM command defines base image.
- Each subsequent command adds a layer
- docker image build ... builds image from Dockerfile

```
# Comments begin with the pound sign
FROM ubuntu:16.04
RUN apt-get update
ADD /data /myapp/data
...
```

EXERCISE: DOCKERFILES (1/2)

Work through the 'Creating Images with Dockerfiles (1/2)' exercise in the Docker Fundamentals Exercises book.

BUILD OUTPUT

```
Bills-MBP:demo billmills$ docker image build -t demo .
Sending build context to Docker daemon 2.048 kB
Step 1/3 : FROM ubuntu:16.04
 ---> 104bec311bcd
Step 2/3 : RUN apt-get update
 ---> Running in ea184fe46743
...
Reading package lists...
 ---> 33455cf3dab6
Removing intermediate container 4e7332b92863
Step 3/3 : RUN apt-get install -y iputils-ping
 ---> Running in f025fb4515c2
Reading package lists...
...
 ---> 3e47103498fa
Removing intermediate container f025fb4515c2
Successfully built 3e47103498fa
```

BUILD CONTEXT

```
Bills-MBP:demo billmills$ docker image build -t demo .
Sending build context to Docker daemon 2.048 kB
```

- Directory archive
- Must contain all local files necessary for image
- Will omit anything listed in .dockerignore

EXAMINING THE BUILD PROCESS

For each command:

- Launch a new container based on the image thus far, and executes in that container:

```
Step 2/3 : RUN apt-get update
 ---> Running in ea184fe46743
```

- Commit R/W layer to image and delete intermediate container:

```
 ---> 33455cf3dab6
Removing intermediate container 4e7332b92863
```

EXAMINING THE BUILD PROCESS

This:

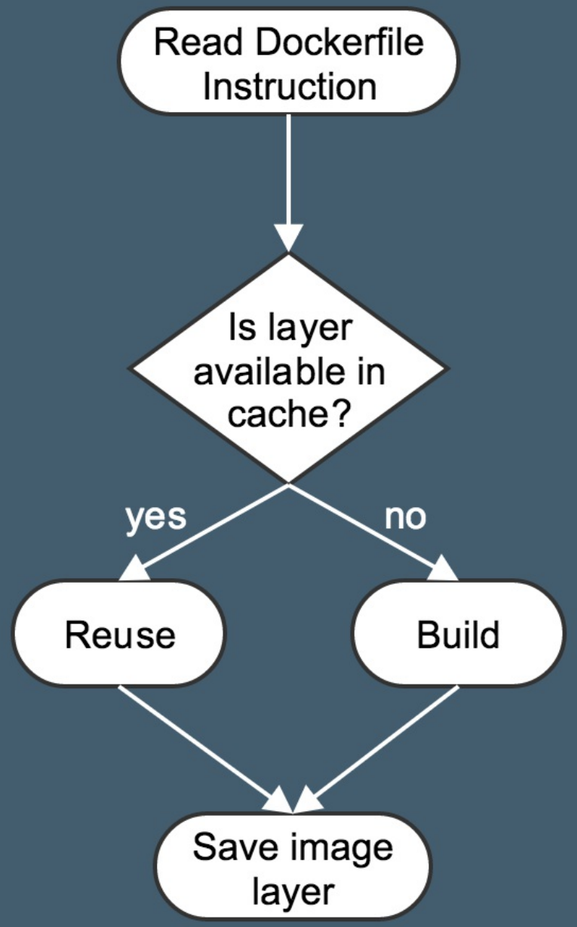
```
RUN cd /src
RUN bash setup.sh
```

is different than this:

```
RUN cd /src && bash setup.sh
```

because every Dockerfile command runs in a different container, and only the filesystem, not the in-memory state, is persisted from layer to layer.

BUILD CACHE



CMD AND ENTRYPOINT

- Recall all containers run a process as their PID 1
- CMD and ENTRYPOINT allow us to specify default processes.

EXERCISE: DOCKERFILES (2/2)

Work through the 'Creating Images with Dockerfiles (2/2)' exercise in the Docker Fundamentals Exercises book.

CMD AND ENTRYPOINT

- CMD alone: default command and list of parameters.
- CMD + ENTRYPOINT: ENTRYPOINT provides command, CMD provides default parameters.
- CMD overridden by command arguments to docker container run
- ENTRYPOINT overridden via --entrypoint flag to docker container run.

SHELL VS. EXEC FORMAT

```
# Shell form
ENTRYPOINT sudo -u ${USER} java ...

# Exec form
ENTRYPOINT ["sudo", "-u", "jdoe", "java", ...]
```

COPY AND ADD COMMANDS

COPY copies files from build context to image:

```
COPY <src> <dest>
```

ADD nearly identical, can also untar and fetch URLs.

- In both cases, a checksum for the files added is logged in the build cache
- Cache invalidated if added files change.

ENV COMMANDS

- ENV sets environment variables inside container: `ENV APP_PORT 8080`
- Many more Dockerfile commands are available; see the docs at https://docs.docker.com/engine/reference/builder/

"IT WORKS ON MY LAPTOP, I SWEAR!"

A perfectly healthy Java app will break when migrated to a new environment if:

- Missing JDK / JVM
- Wrong versions of JDK or JVM
- Missing libraries
- Wrong versions of libraries

Capturing all dependencies and environment setup steps in a Dockerfile is the easiest way to make your application reliably portable.

EXERCISE: DOCKERIZING AN APPLICATION

Work through the 'Dockerizing an Application' exercise in the Docker Fundamentals Exercises book.

DOCKERFILE BEST PRACTICES: LAYERS

- Start with official images
- Combine commands
- Single ENTRYPOINT

DOCKERFILE BEST PRACTICES: CACHING

```
FROM ubuntu:14.04
MAINTAINER Docker Education Team <education@docker.com>
RUN apt-get update
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q \
    python-all python-pip
ADD ./webapp /opt/webapp/
WORKDIR /opt/webapp
RUN pip install -qr requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

DOCKERFILE BEST PRACTICES: CACHING

```
FROM ubuntu:14.04
MAINTAINER Docker Education Team <education@docker.com>
RUN apt-get update
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q \
    python-all python-pip
ADD ./webapp/requirements.txt /tmp/requirements.txt
RUN pip install -qr /tmp/requirements.txt
ADD ./webapp /opt/webapp/
WORKDIR /opt/webapp
EXPOSE 5000
CMD ["python", "app.py"]
```

IMAGE TAGS

- Optional string after image name, separated by :
- :latest by default
- Same image with two tags shares same ID, image layers:

```
$ docker image ls centos*
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              7                   8140d0c64310        7 days ago          193 MB
$ docker image tag centos:7 centos:mytag
$ docker image ls centos*
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              7                   8140d0c64310        7 days ago          193 MB
centos              mytag               8140d0c64310        7 days ago          193 MB
```

IMAGE NAMESPACES

Images exist in one of three namespaces:

- Root (ubuntu, nginx, mongo, mysql, ...)
- User / Org (jdoe/myapp:1.1, tutum/mongodb:latest, ...)
- Registry (FQDN/jdoe/myapp:1.1, ...)

Root and User/Org indicate images distributed on store.docker.com; Registry namespacing indicates a Docker Trusted Registry image.

IMAGE TAGGING & NAMESPACING

- Tag on build: `docker image build -t myapp:1.0`
- Retag an existing image: `docker image tag myapp:1.0 me/myapp:2.0`
- Note `docker image tag` can set both tag and namespace.

SHARING IMAGES

- Log in to store.docker.com: docker login
- Share an image: docker image push <image name>
- Public repos available for anyone to docker pull.

EXERCISE: MANAGING IMAGES

Work through the 'Managing Images' exercise in the Docker Fundamentals Exercises book.

IMAGE CREATION TAKEAWAYS

- Images are built out of read-only layers.
- Dockerfiles specify image layer contents.
- Key Dockerfile commands: FROM, RUN, COPY and ENTRYPOINT
- Images must be namespaced according to where you intend on sharing them.

DOCKER SYSTEM COMMANDS

CLEAN-UP COMMANDS

- docker system df

```
TYPE            TOTAL     ACTIVE    SIZE        RECLAIMABLE
Images          39        2         9.01 GB     7.269 GB (80%)
Containers      2         2         69.36 MB    0 B (0%)
Local Volumes   0         0         0 B         0 B
```

- docker system prune

more limited...

- docker image prune
- docker container prune
- docker volume prune
- docker network prune

INSPECT THE SYSTEM

docker system info

```
Containers: 2
 Running: 2
 Paused: 0
 Stopped: 0
Images: 105
Server Version: 17.03.0-ee
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host ipvlan macvlan null overlay
Swarm: active
 NodeID: ybmqksh6fm627armruq0e8id1
 Is Manager: true
 ClusterID: 2rbf1dv6t5ntro2fxbry6ikr3
 Managers: 1
 Nodes: 1
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  ...
```

SYSTEM EVENTS

Start observing with ...

docker system events

Generate events with ...

docker container run --rm alpine echo 'Hello World!'

```
2017-01-25T16:57:48.553596179-06:00 container create 30eb630790d44052f26c1081...
2017-01-25T16:57:48.556718161-06:00 container attach 30eb630790d44052f26c1081...
2017-01-25T16:57:48.698190608-06:00 network connect de1b2b40f522e69318847ada3...
2017-01-25T16:57:49.062631155-06:00 container start 30eb630790d44052f26c1081d...
2017-01-25T16:57:49.164526268-06:00 container die 30eb630790d44052f26c1081dbf...
2017-01-25T16:57:49.613422740-06:00 network disconnect de1b2b40f522e69318847a...
2017-01-25T16:57:49.815845051-06:00 container destroy 30eb630790d44052f26c108...
```

EXERCISE: SYSTEM COMMANDS

Work through:

- Inspection Commands
- Cleanup Commands

in the Docker Fundamentals Exercises book.

DOCKER VOLUMES

TOPICS

- Creating & deleting volumes
- Mounting volumes
- Inspecting volumes
- Sharing volumes

VOLUMES

- Persist when a container is deleted
- Can be shared between containers
- Separate from the union file system

DOCKER VOLUME COMMAND

docker volume sub-commands:

```
docker volume create --name demo
docker volume ls
docker volume inspect demo
docker volume rm demo
docker volume prune
```

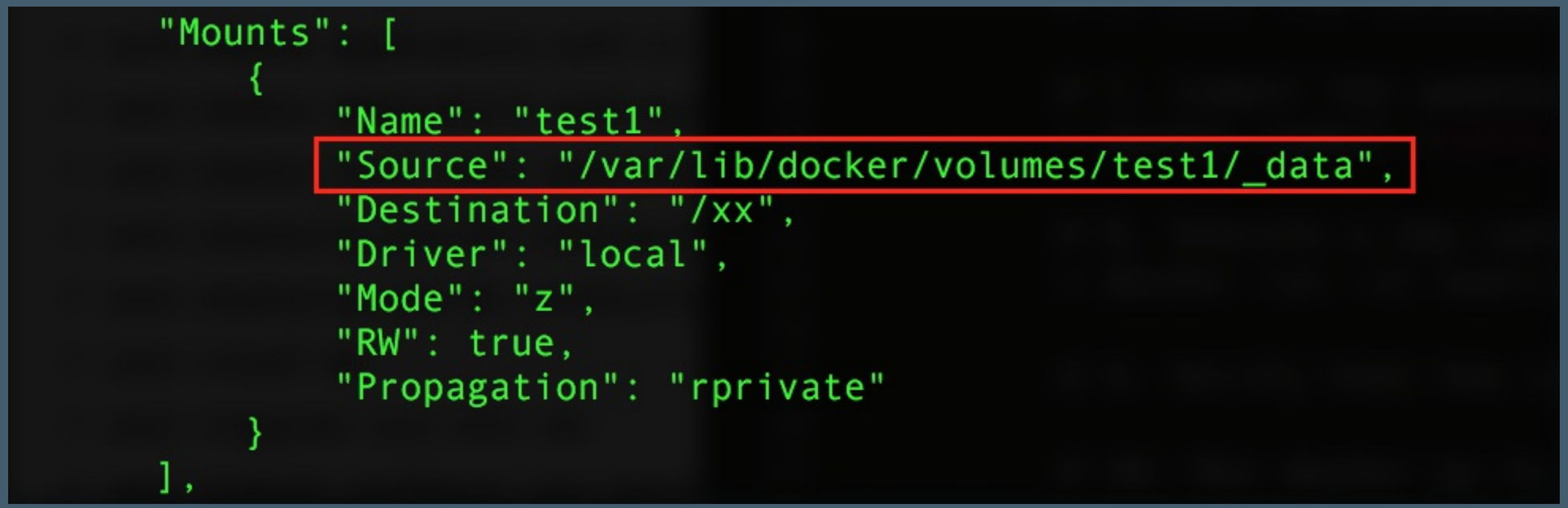
MOUNT A VOLUME

- Mounted at container startup
- docker container run -v [name]:[path in container FS] ...
- Example:

```
# Execute a new container and mount the volume test1 in the folder /www/test1
docker container run -it -v test1:/www/test1 ubuntu:16.04 bash
```

WHERE ARE OUR VOLUMES?

- Volumes exist independently from containers
- If a container is stopped, we can still access our volume
- To find where the volume is use **docker container inspect** on the container and look for the “source” field as shown below:



DOCKER VOLUME INSPECT COMMAND

- The **docker volume inspect** command shows all the information about a specified volume
- Information includes the “Mountpoint” which tells us where the volume is located on the host

```
Bills-MBP:images billmills$ docker volume inspect test1
[
    {
        "Name": "test1",
        "Driver": "local",
        "Mountpoint": "/var/lib/docker/volumes/test1/_data",
        "Labels": null,
        "Scope": "local"
    }
]
```

DELETING A VOLUME

```
# Delete the volume called test1
> docker volume rm test1

# Delete a container and remove its associated volumes
> docker container rm -v <container ID>
```

Note the **-V** option: volumes not automatically deleted when deleting a container.

DELETING VOLUMES

- Cannot delete a volume if it is being used by a container (running or stopped)
- docker container rm -v <container ID> will not delete a volume associated with the container if that volume is mounted in another container

EXERCISE: CREATING AND MOUNTING VOLUMES

Work through the 'Creating and Mounting Volumes' exercise in the Docker Fundamentals Exercises book.

MOUNTING HOST DIRECTORIES

- Can map directories on the host to a container path
- Changes made on the host are reflected inside the container
- Syntax:
  docker container run -v [host path]:[container path]:[rw|ro]
- rw or ro controls the write status of the directory inside the container

```
# Mount the contents of the public_html directory on the
# hosts to the container volume at /data/www
> docker container run -d \
        -v /home/user/public_html:/data/www ubuntu
```

INSPECTING THE MAPPED DIRECTORY

- docker container inspect
  - Source: host path to mounted directory
  - Destination: container path

```
"Mounts": [
    {
        "Source": "/home/ubuntu/public_html",
        "Destination": "/data/www",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
],
```

USE CASES FOR MOUNTING HOST DIRECTORIES

- Storage management
- Rapid updates (ex code development)

SHARING DATA BETWEEN CONTAINERS

- Volumes can be mounted into multiple containers
- Allows data to be shared between containers
- Example use cases
  - One container collects and writes metrics data
  - Another container analyzes the metrics
- Note: Be aware of potential conflicts and security breaches!

EXERCISE: RECORDING LOGS

Work through the Volumes Usecase: 'Recording Logs' exercise in the Docker Fundamentals Exercises book.

VOLUMES IN DOCKERFILE

- VOLUME instruction creates a mount point
- Can specify arguments in a JSON array or string
- Cannot map volumes to host directories
- Volumes are initialized when the container is executed

```
# String example
VOLUME /myvol

# String example with multiple volumes
VOLUME /www/website1.com /www/website2.com

# JSON example
VOLUME [“/myvol”, “/myvol2”]
```

EXAMPLE DOCKERFILE WITH VOLUMES

- Volume initialized along with data on **docker container run ...**

```
FROM ubuntu:16.04

RUN apt-get update
RUN apt-get install -y vim wget

RUN mkdir /data/myvol -p && echo "hello world" > /data/myvol/testfile
VOLUME ["/data/myvol"]
```

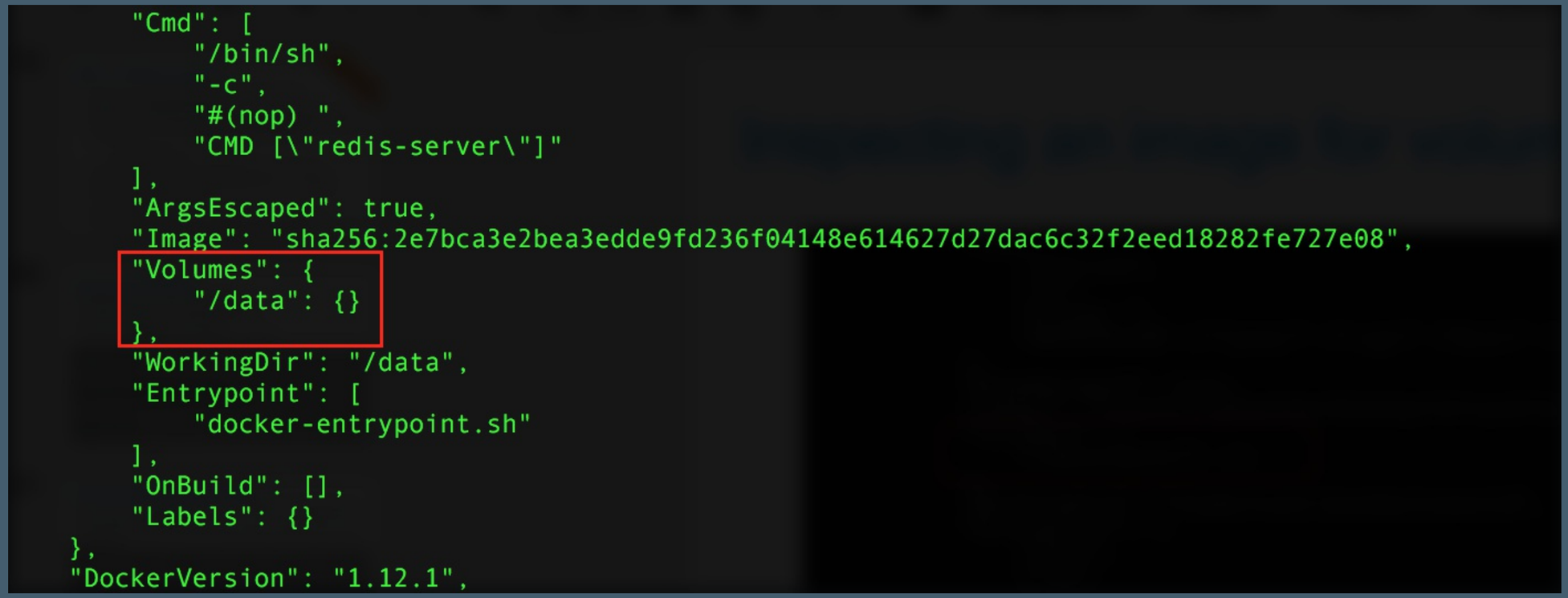
VOLUMES DEFINED IN IMAGES

- docker image inspect lists volumes defined in images

```
# Inspect the properties of the mongo:latest image
docker image pull mongo:latest
docker image inspect mongo:latest

# OR
docker image inspect <image id>
```

INSPECTING AN IMAGE FOR VOLUMES



DOCKER VOLUME TAKEAWAYS

- Volumes are for persistent data
- Volumes bypass the copy on write system
- A volume persists even after its container has been deleted

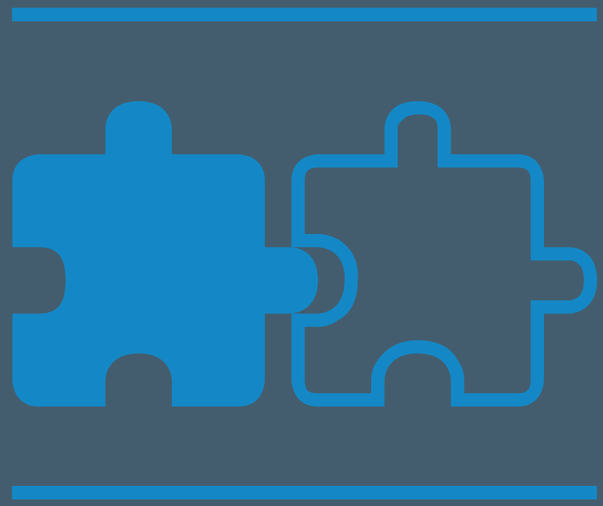
DOCKER PLUGINS

PLUGINS

- Extend the Docker platform
- Distributed as Docker images
- Hosted on store.docker.com

List plugins on system:

```
$ docker plugin ls
ID                  NAME                DESCRIPTION                ENABLED
bee424413706        vieux/sshfs:latest  sshFS plugin for Docker    true
```



INSTALL A PLUGIN

```
$ docker plugin install vieux/sshfs
Plugin "vieux/sshfs" is requesting the following privileges:
 - network: [host]
 - device: [/dev/fuse]
 - capabilities: [CAP_SYS_ADMIN]
Do you grant the above permissions? [y/N] y
latest: Pulling from vieux/sshfs
ca06593c3b54: Download complete
Digest: sha256:660b5302a6951aa1c47c3042ca288d16973f236adbd2f77fc5571f164143adf5
Status: Downloaded newer image for vieux/sshfs:latest
Installed plugin vieux/sshfs
```

USING THE PLUGIN

Create a volume...

```
docker volume create -d vieux/sshfs \
    -o sshcmd=<user@host:path> \
    -o password=<password> \
    sshvolume
```

Use the volume...

```
docker container run --rm -it \
    -v sshvolume:/shared \
    busybox ls -al /shared
```

EXERCISE: PLUGINS

Work through the 'Docker Plugins' exercise in the Docker Fundamentals Exercises book.

CONTAINERIZATION FUNDAMENTALS CONCLUSION: ANY APP, ANYWHERE.

- Containers are isolated processes
- Images provide filesystem for containers
- Volumes persist data

PART 2: ORCHESTRATION

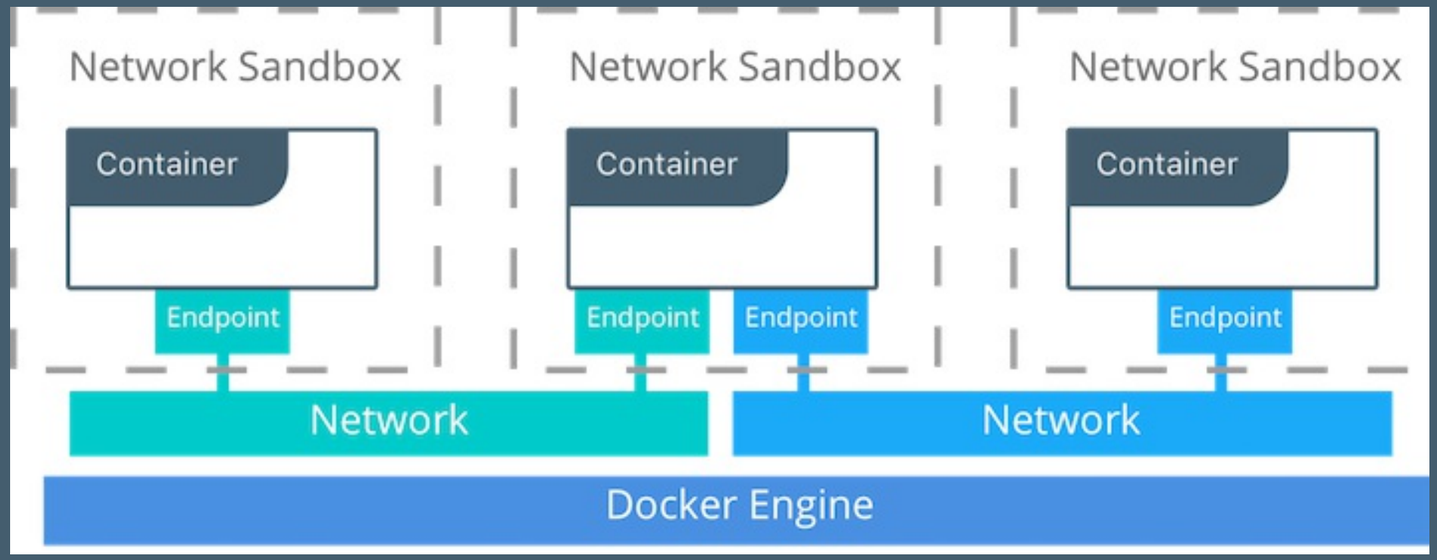
Image CC-BY Phil Roeder

DOCKER NETWORKING BASICS

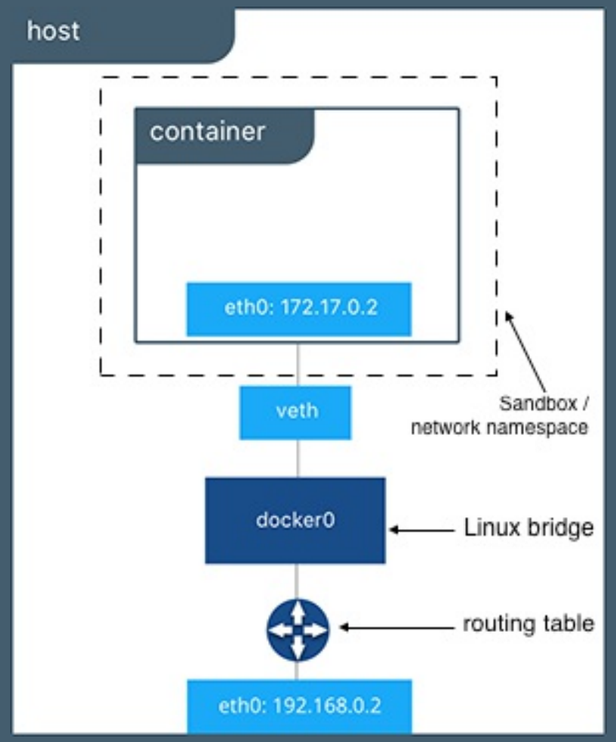
TOPICS

- Bridge networks
- Network firewalling
- Port management

THE CONTAINER NETWORK MODEL



LINUX BRIDGES

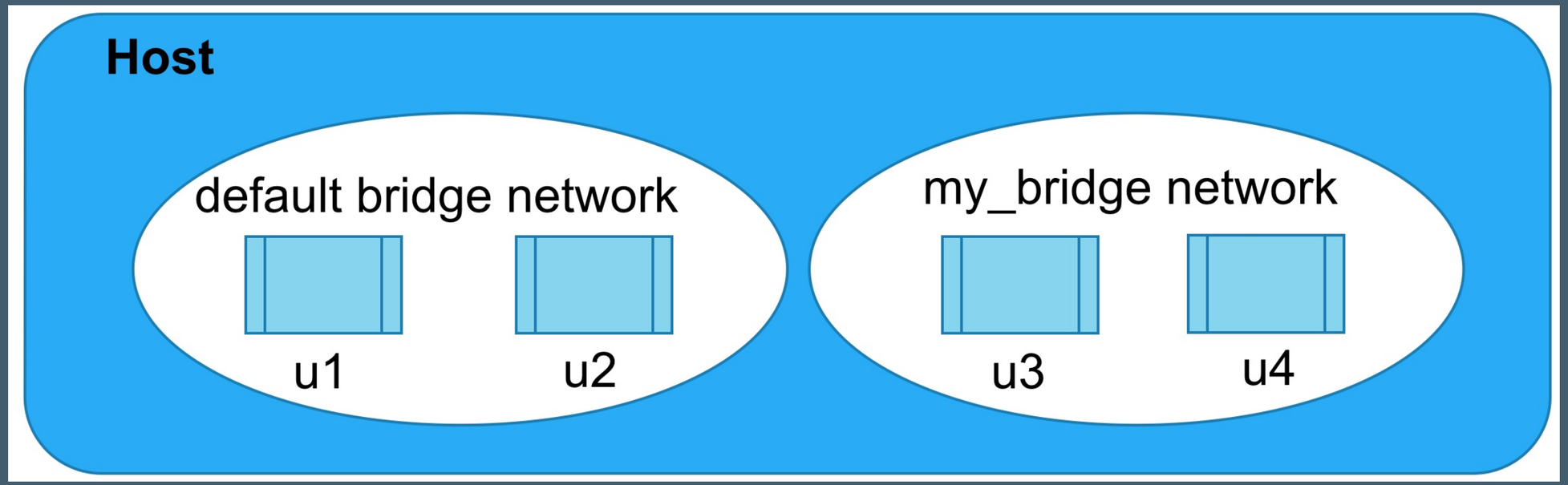


- Level 2 network (ie routing between MAC addresses)
- docker0: default linux bridge
- Firewalls containers from outside traffic by default

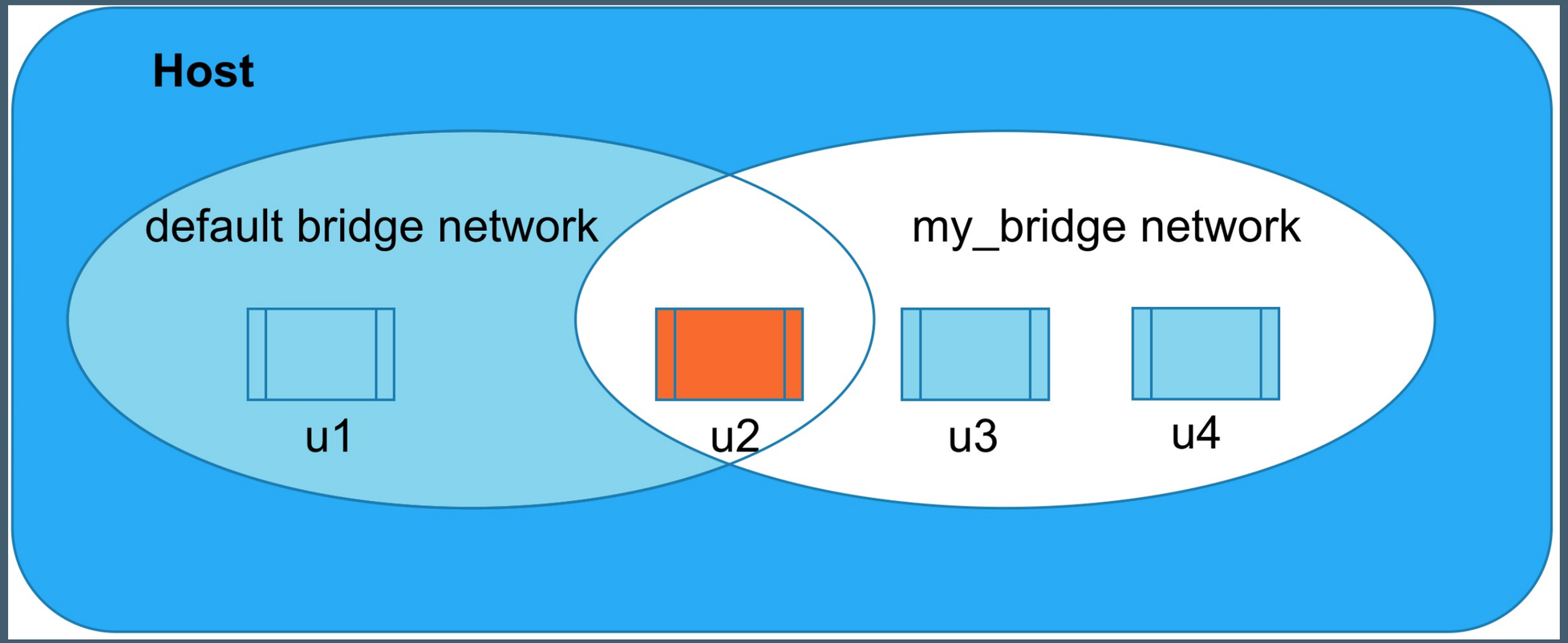
EXERCISE: INTRODUCTION TO CONTAINER NETWORKING

Work through the 'Introduction to Container Networking' exercise in the Docker Fundamentals Exercises book.

NETWORK FIREWALLS



NETWORK FIREWALLS




```
docker network connect my_bridge u2
```

SECURITY WARNING

Do not use the **host** network in production.

EXPOSING CONTAINER PORTS

- Containers have no public IP address by default; reachable only locally via their host's linux bridge.
- Can map a container port to a host port to allow container reachability.
- Ports can be mapped manually or automatically.
- Port mappings visible via
  `docker container ls` or
  `docker container port`

EXERCISE: CONTAINER PORT MAPPING

Work through the 'Container Port Mapping' exercise in the Docker Fundamentals Exercises book.

DOCKER NETWORKING TAKEAWAYS

- Docker networks on a single host use linux bridges to route traffic between containers
- Separate Docker networks are firewalled from each other by default
- Containers are firewalled from the outside world by default, but can expose ports on the host
- Advanced networking: http://dockr.ly/2eTRNdy

INTRODUCTION TO DOCKER COMPOSE

TOPICS

- Services
- Defining Application with Docker Compose
- Scaling Applications

DISTRIBUTED APPLICATION ARCHITECTURE

- Orchestration of one or more scalable services across one or more nodes
- Docker Compose facilitates multi-service design.

DOCKER SERVICES

Containers:

- (meant to be) lightweight
- easy to start and stop
- fundamentally just a process

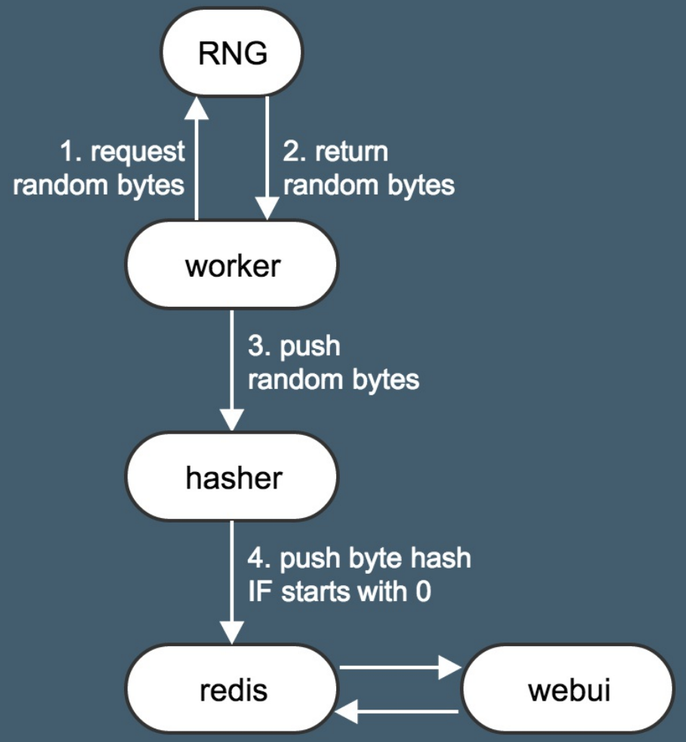
Services:

- defines desired state of a collection of containers
- is a self healing & scalable collection of containers
- is based on single image

OUR APPLICATION: DOCKERCOINS

(DockerCoins 2016 logo courtesy of @XtlCnslt and @ndeloof. Thanks!)

- It is a DockerCoin miner!
- Dockercoins consists of 5 services working together:



OUR SAMPLE APPLICATION

- https://github.com/docker-training/orchestration-workshop/tree/17.3
- The application is in the dockercoins subdirectory
- Each service has its own subdirectory & Dockerfile

DOCKER-COMPOSE.YML

- Applications are built around the concept of services.
- docker-compose.yml: manifest of services and peripherals

SERVICE DISCOVERY

- Containers can have network aliases (resolvable through DNS)
- Our code can connect to services using their short name (instead of e.g. IP address or FQDN)

EXAMPLE IN WORKER/WORKER.PY

```
16
17  redis = Redis("redis")
18
19
20  def get_random_bytes():
21      r = requests.get("http://rng/32")
22      return r.content
23
24
25  def hash_bytes(data):
26      r = requests.post("http://hasher/",
27                        data=data,
28                        headers={"Content-Type":
```

EXERCISE: STARTING A COMPOSE APP

Work through the 'Starting a Compose App' exercise in the Docker Fundamentals Exercises book.

CONNECTING TO THE WEB UI

- webui dashboard: http://[IP]:8000/
- Looks like about 3.33 coins/second.

SCALING UP THE APPLICATION

- Want higher performance
- Need to determine bottlenecks
- Common UNIX tools to the rescue!

LOOKING AT RESOURCE USAGE

- top

  (you should see idle cycles)
- vmstat 3

  (the 4 numbers should be almost zero, except bo for logging)

We have available resources; how can we use them?

EXERCISE: SCALING A COMPOSE APP

Work through the 'Scaling a Compose App' exercise in the Docker Fundamentals Exercises book.

DOCKER COMPOSE TAKEAWAYS

- Docker Compose makes single node orchestration easy
- Docker Compose makes scaling services easy
- Bottleneck identification important
- Syntactically: docker-compose.yml + API

INTRODUCTION TO SWARM MODE

TOPICS

- Swarmkit
- Services and Tasks
- Creating Swarms
- Manager & Worker Coordination
- Routing Mesh
- Swarm Scheduling
- Service Upgrades

DISTRIBUTED APPLICATION ARCHITECTURE

- Orchestration of one or more scalable services across one or more nodes
- Docker Swarm facilitates multi-node design.
- Also supports multiple interacting services (like Compose)

SWARMKIT

- Open source toolkit to build multi-node systems
- SwarmKit comes with two components:
  - swarmctl (a CLI tool to "speak" the SwarmKit API)
  - swarmd (an agent that can federate existing Docker Engines into a Swarm)
- Project repository: https://github.com/docker/swarmkit
- Integrated into the Docker platform.

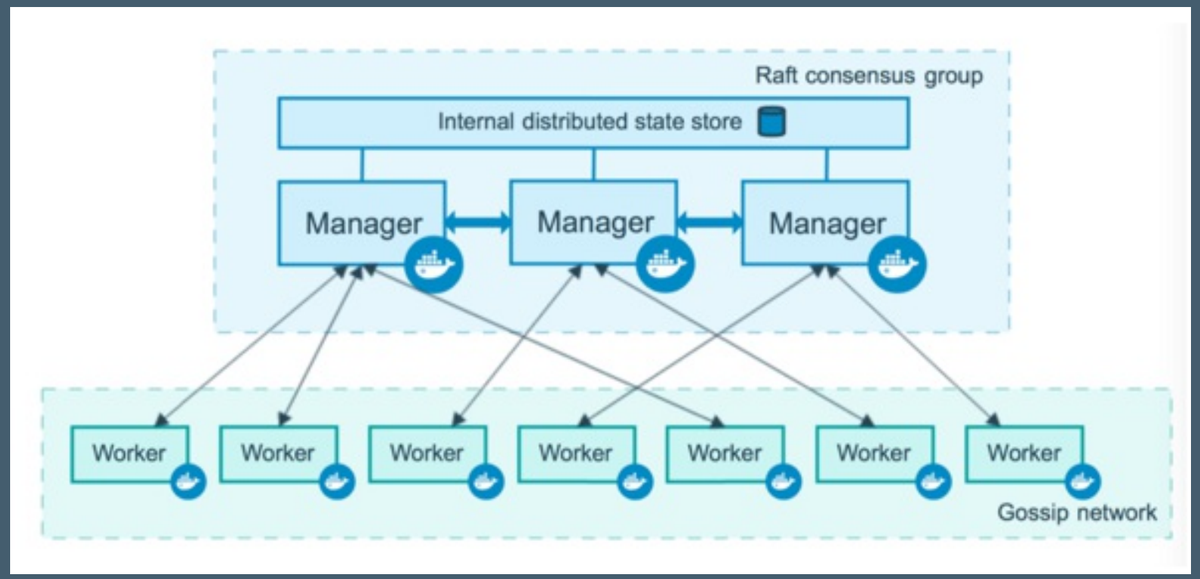
SWARMKIT KEY FEATURES

- Native orchestration
- Maintains desired state of services
- Highly-available raft consensus
- Automatic TLS encryption
- Full features list: https://docs.docker.com/engine/swarm/

KEY CONCEPTS – SWARM AND NODE

- Swarm:
  - A collection of engines on which services can be deployed
- Node:
  - A single Docker host participating in a Swarm
  - Can be a manager node or worker node
  - Managers are also workers

A TYPICAL SWARM



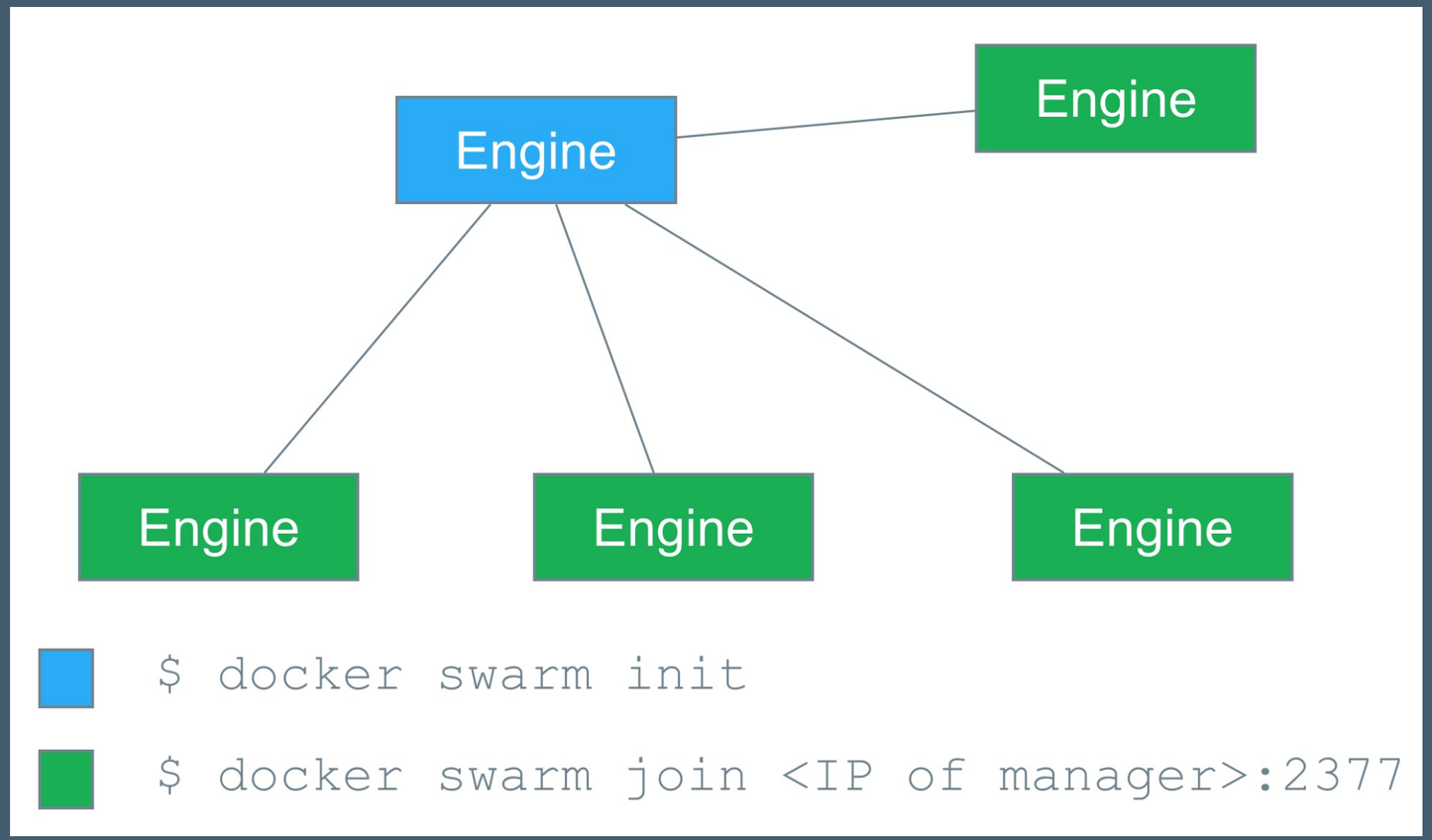
KEY CONCEPTS - SERVICES

- Service: desired state of containers based on a given image
- Primary point of user interaction with the swarm
- Replicated vs global:
  - Replicated: user defined concurrency, scheduler manages details
  - Global: exactly one container of this service is run per node, at all times

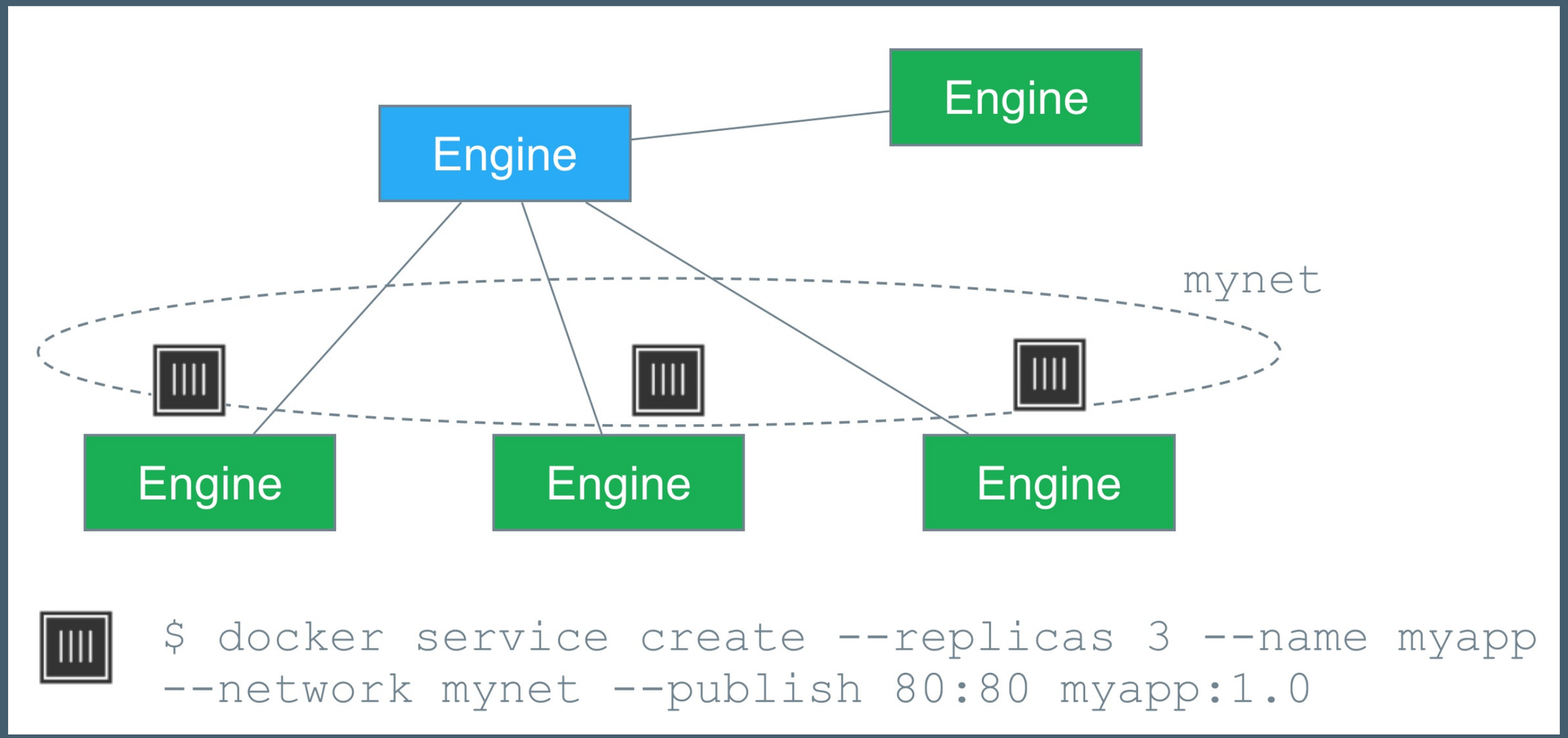
KEY CONCEPTS - TASKS

- A task represents a unit of work assigned to a node
- One task, one container
- Atomic scheduling unit of swarm

SWARMS & SERVICES

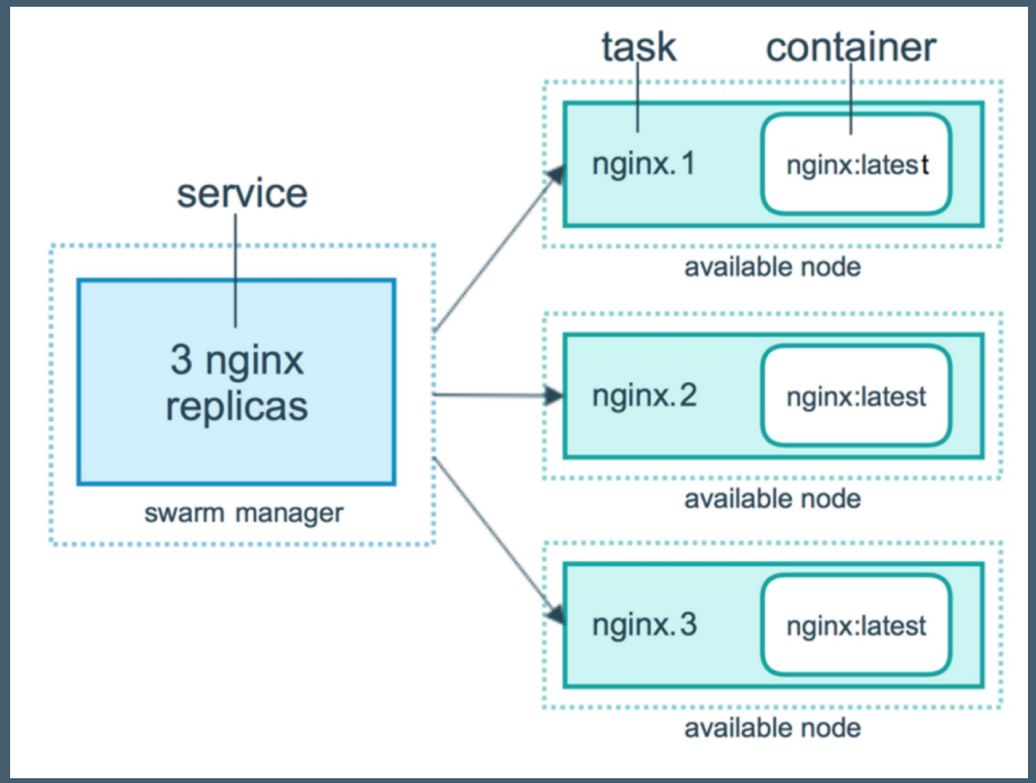


SWARMS & SERVICES

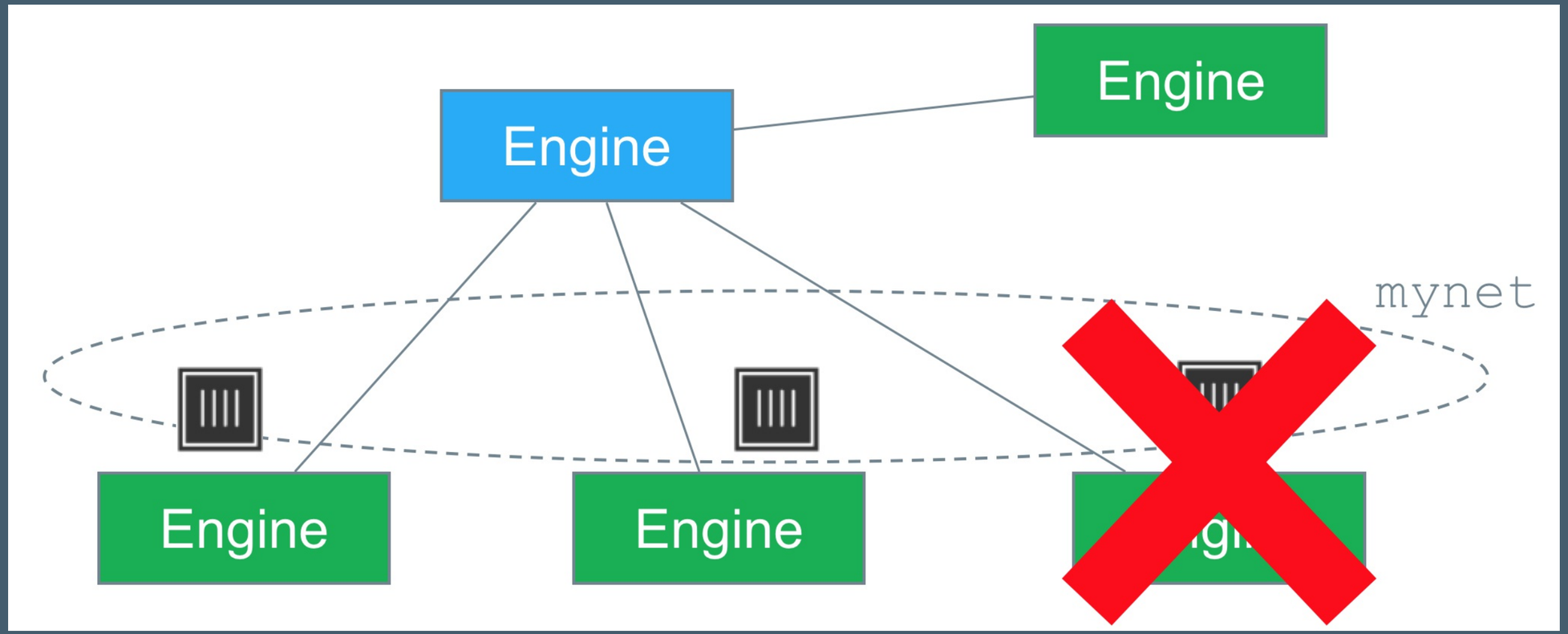



```
$ docker service create --replicas 3 --name myapp
--network mynet --publish 80:80 myapp:1.0
```

SERVICES, TASKS & CONTAINERS

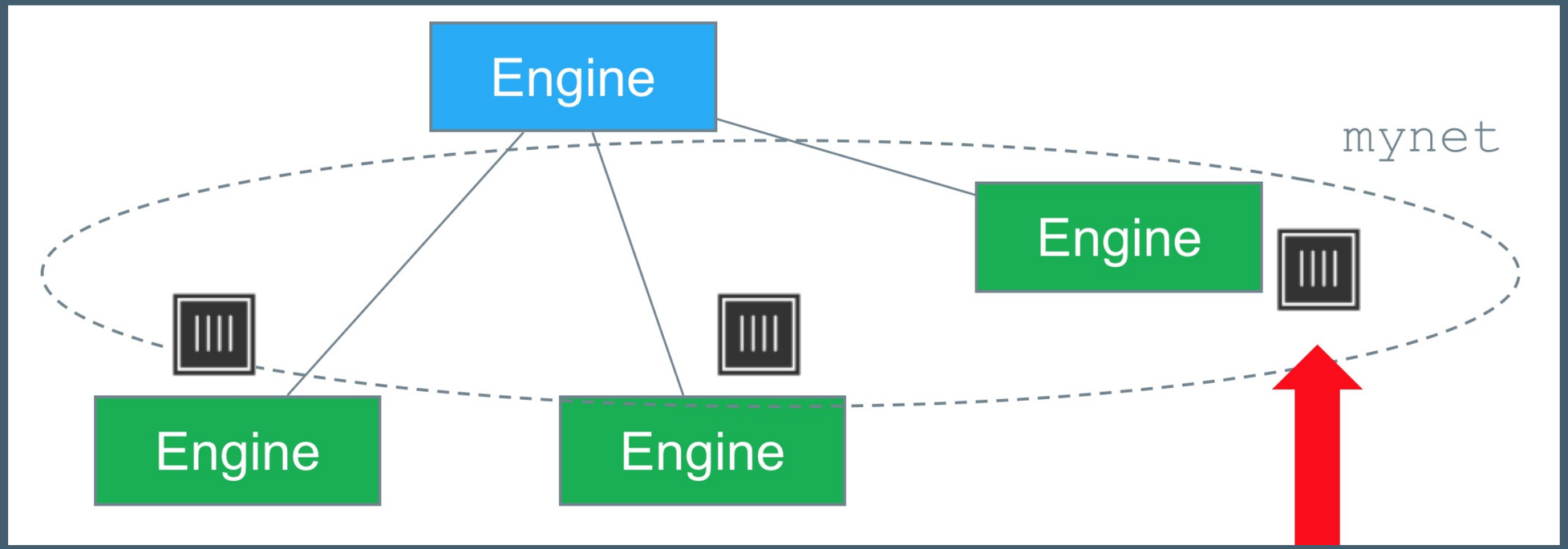


RECOVERING FROM NODE FAILURE




```
docker service create --replicas 3 --name myapp --network mynet --publish 80:80 myapp:1.0
```

RECOVERING FROM NODE FAILURE



Swarm will schedule a new task in order to create the new container so that we once again have 3 replicas.

EXERCISE: SWARMS & SERVICES

Work through:

- Creating a Swarm
- Starting a Service
- Node Failure Recovery

in the Docker Fundamentals Exercises book.

UNDER THE HOOD - SWARM INIT

When we first run **docker swarm init**:

- current node enters Swarm Mode as manager-leader
- listens for workers on :2377
- prepares to accept tasks from the scheduler
- starts an internal distributed data store for network and scheduling state
- generates a self-signed root CA for the swarm
- generates tokens for worker and manager nodes to join the swarm
- creates an overlay network named ingress for external traffic inbound to services
- ... and a whole lot more.

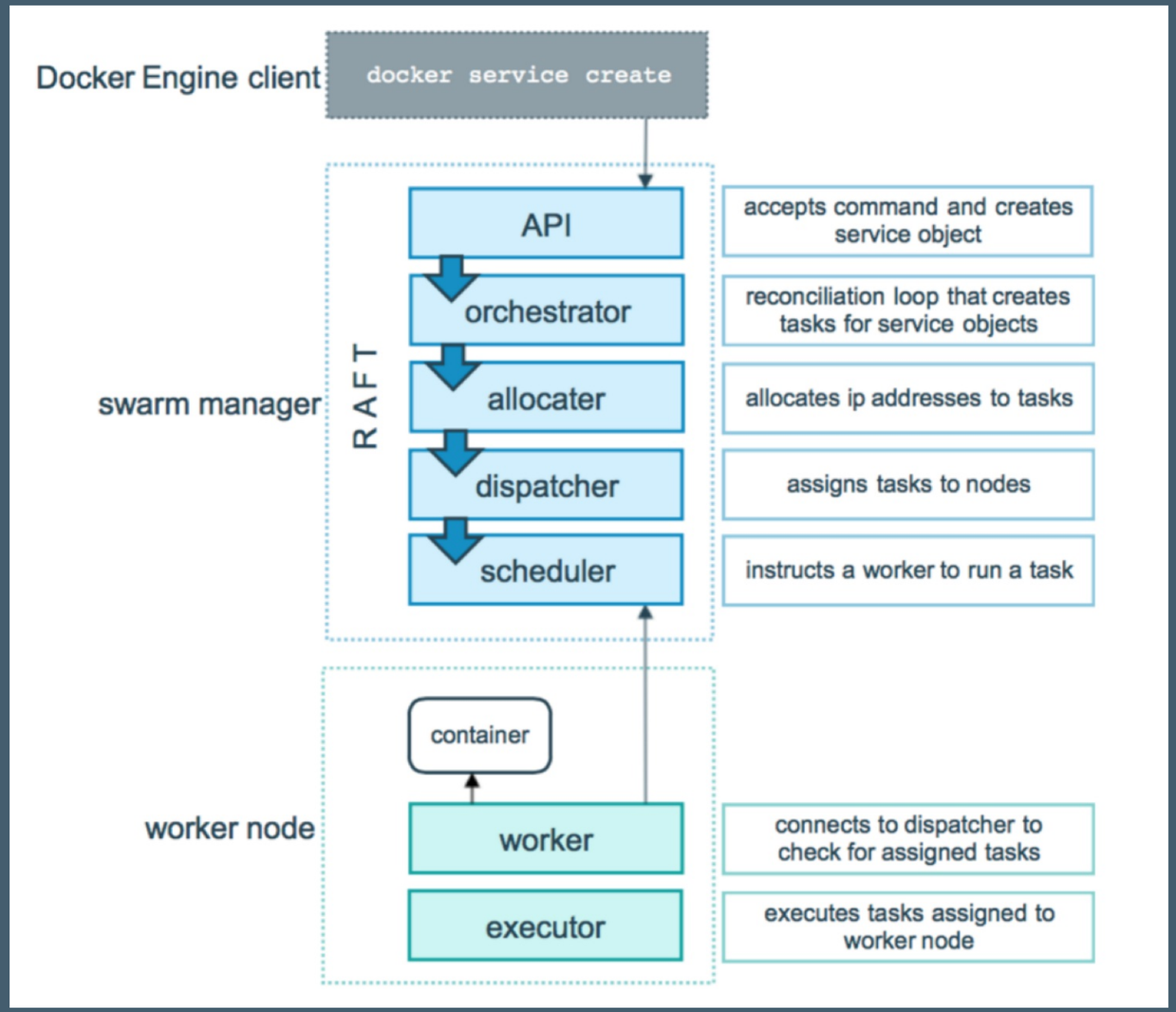
UNDER THE HOOD - CERTIFICATES

- When we do `docker swarm init`, a TLS root CA is created. Then a keypair is issued for the first node, and signed by the root CA.
- When further nodes join the Swarm, they are issued their own keypair, signed by the root CA, and they also receive the root CA public key and certificate.
- All communication is encrypted via TLS.
- The node keys and certificates are automatically renewed on regular intervals (by default, 90 days; this is tunable with `docker swarm update`).

SWARM MODE NETWORK TOPOLOGY

- Fixed IP address for manager nodes
  - Workers can use dynamic IP
- Open ports between your hosts:
  - TCP port 2377 for cluster management communication
  - TCP and UDP port 7946 for communication among nodes
  - TCP and UDP port 4789 for overlay network traffic

BEHIND THE SCENES: SERVICE MANAGEMENT

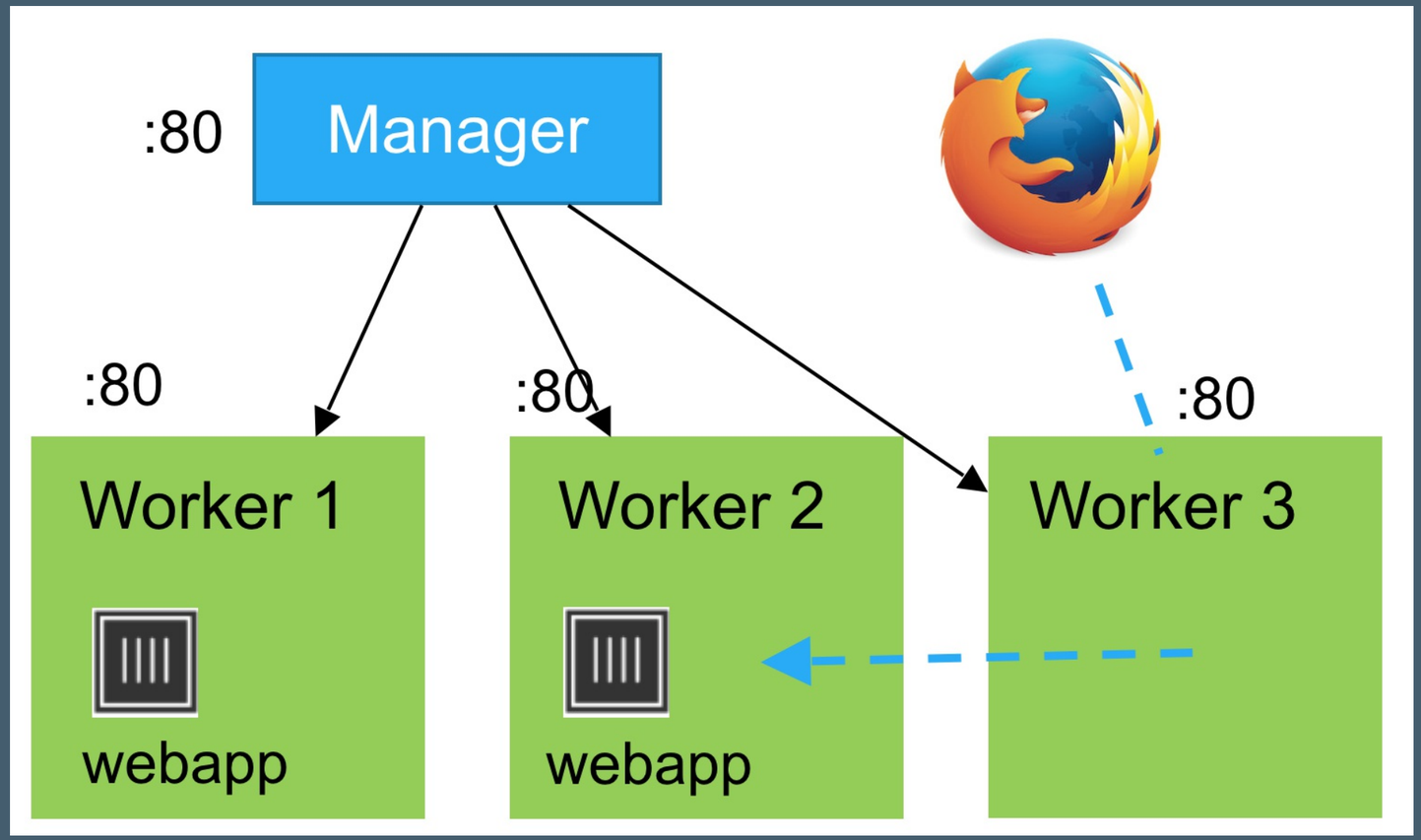


SERVICES & THE OUTSIDE WORLD

Services can be exposed to the web on a host port, with two special properties:

- the public port is available on every node of the Swarm - even ones that aren't running the service
- requests coming on the public port are load balanced across all instances.

THE ROUTING MESH




```
docker service create --replicas 2 --publish 80:80 --name webapp webapp:1.0
```

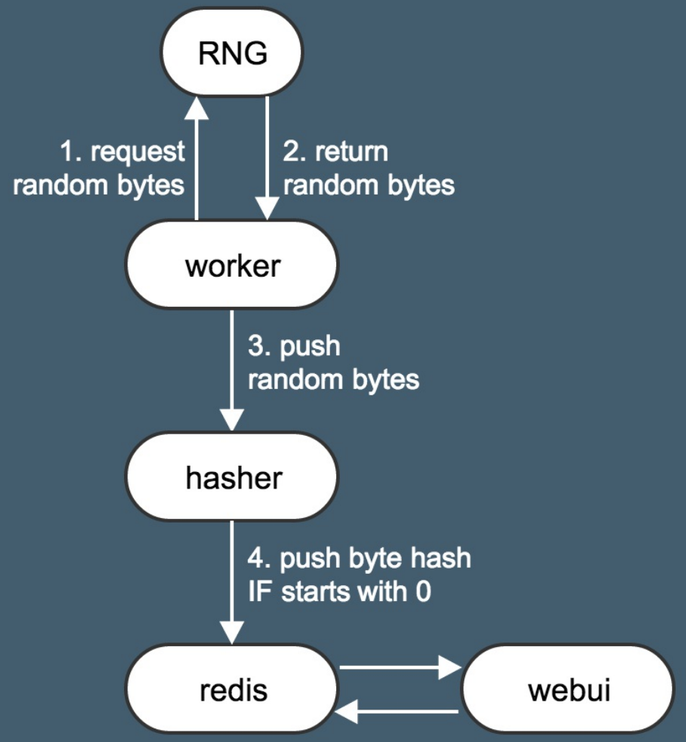
EXERCISE: LOAD BALANCING & THE ROUTING MESH

Work through the 'Load Balancing & the Routing Mesh' exercise in the Docker Fundamentals Exercises book.

OUR APPLICATION: DOCKERCOINS

(DockerCoins 2016 logo courtesy of @XtlCnslt and @ndeloof. Thanks!)

- It is a DockerCoin miner!
- Dockercoins consists of 5 services working together:



SWARMING OUR APP: STACKS

- Collection of services
- Uses v3 docker-compose.yml as manifest
- can specify replicas, networks, volumes...

SCALING & SCHEDULING SERVICES

- Default: each service will run exactly one container
- Improve performance by adding more containers
- VIPs load balance across tasks
- Works best with stateless containers
- Scheduling strategies: replicated or global
  - Replicated is the default we've seen so far
  - Global schedules exactly one task for the service on all available workers in the swarm

UPDATING SERVICES

- Code updates will periodically land for all services
- We'd like to update services without taking our app offline
- Swarm mode provides tooling for rolling updates with user-defined parallelism

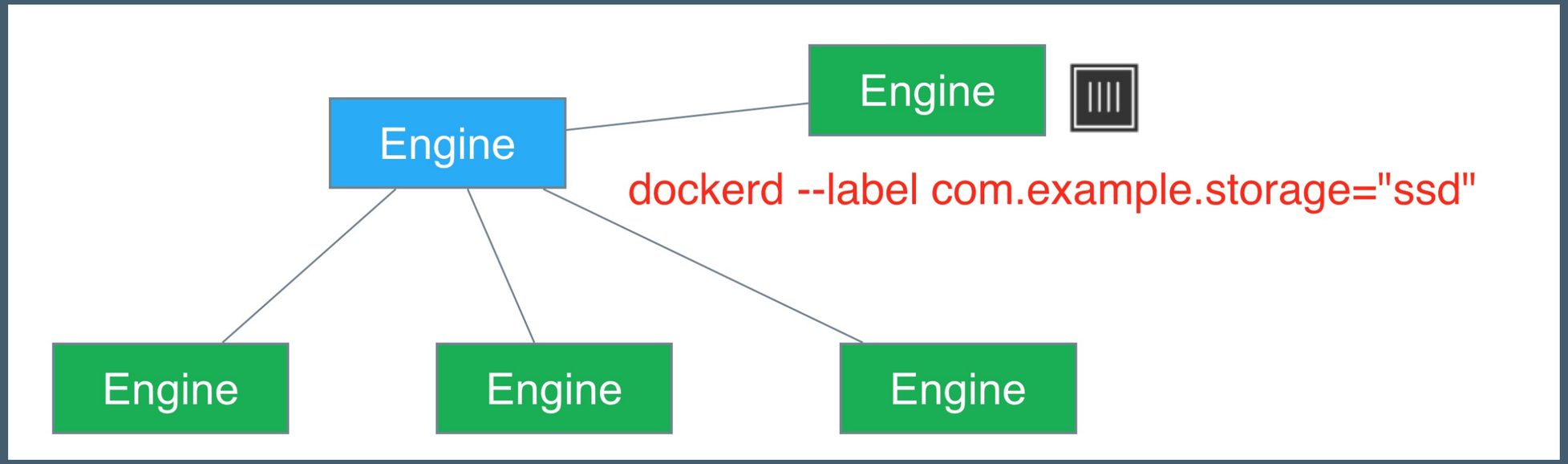
EXERCISE: APPLICATION DEPLOYMENT

Work through:

- Dockercoins On Swarm
- Scaling and Scheduling Services
- Updating a Service

in the Docker Fundamentals Exercises book.

SWARM DETAILS: NODE CONSTRAINTS




```
docker service create --replicas 3 --name myapp \
--network mynet --publish 80:80 \
–-constraint com.example.storage=“ssd” myapp:1.0
```

SWARM DETAILS: SERVICE LOAD BALANCING

- Network requests for service names resolve to a VIP, internally load balanced by IPVS.
- DNSRR also available
- Default is now VIP
- Can specify on service creation: `docker service create --endpoint-mode [VIP|DNSRR]`

SWARM MODE ROBUSTNESS

It doesn't matter:

- which node a container runs on
- if a few nodes die
- if interacting processes are on different nodes
- which nodes are running user-facing containers

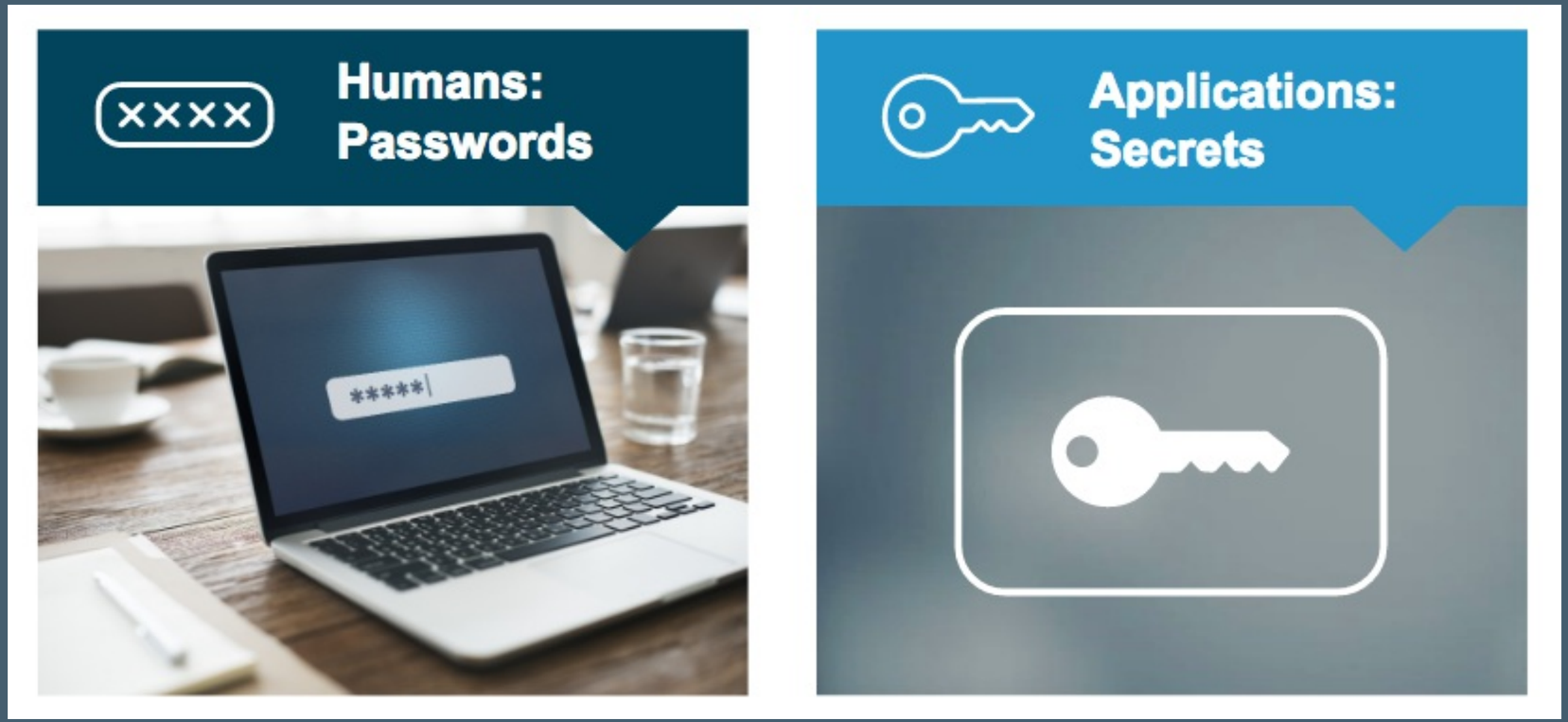
... everything will still work.

SWARM MODE TAKEAWAYS

- Distributed applications across infrastructure
- High availability
- Self healing service definitions
- Default security via mutual TLS encryption & certificate rotation.
- Simple service discovery & load balancing

DOCKER SECRETS

WHAT IS A SECRET?



MOTIVATION FOR SECRETS

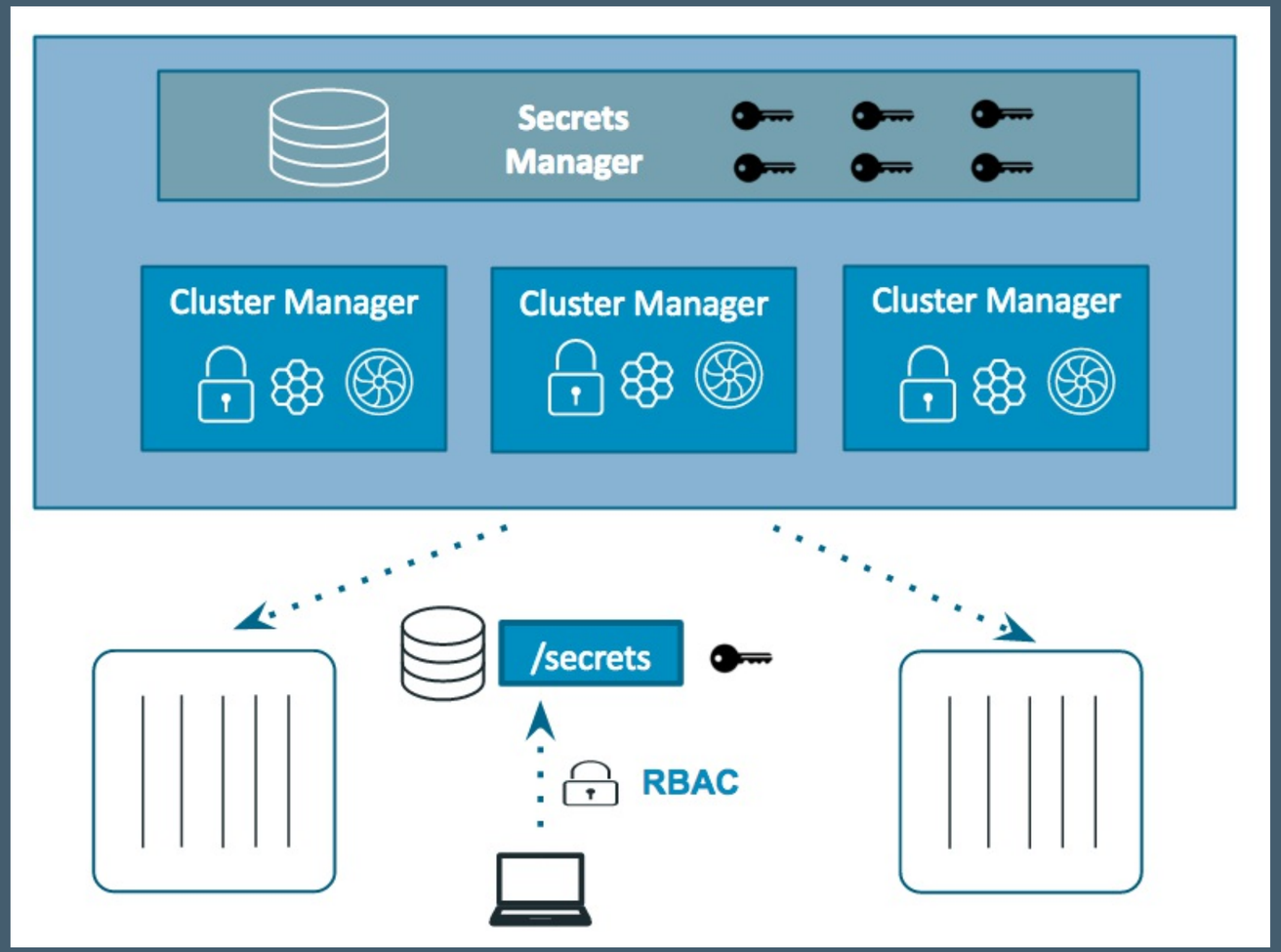
Challenges in distributed systems:

- Secrets can be embed in source code in GitHub
- Secrets can be distributed to other nodes in Container orchestration systems
- Secrets could be tampered with in transit

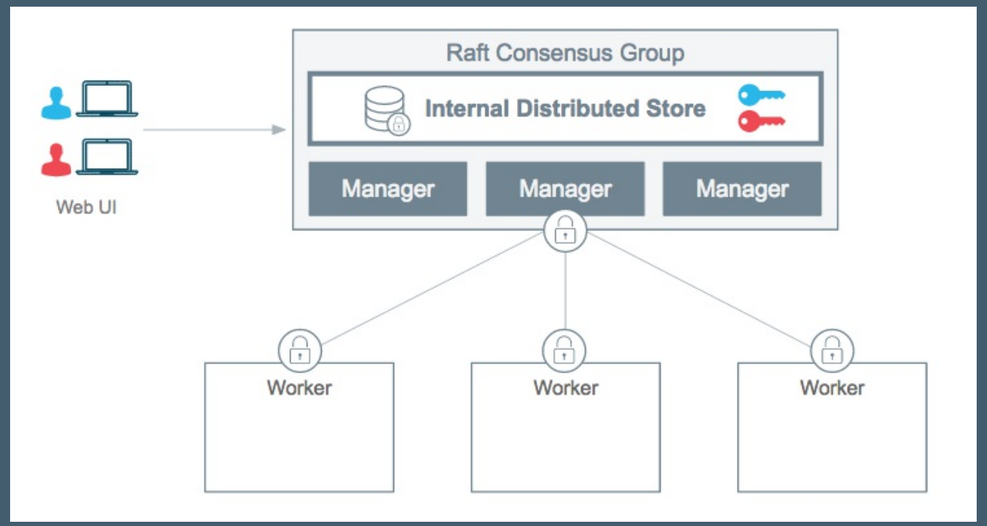
SECRETS USE CASE

Manage services with sensitive information in Docker Swarm such as:

- passwords
- TLS certificates
- private keys
- and more...

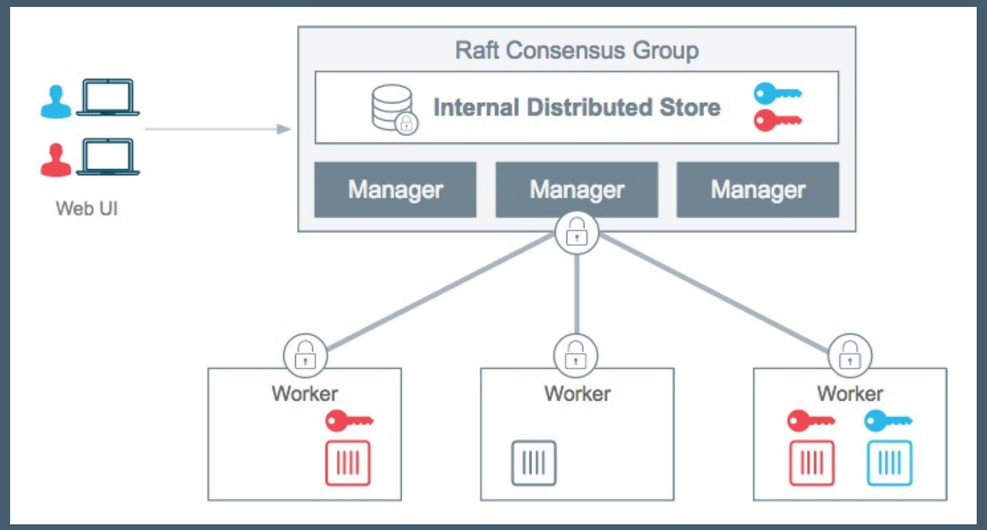


SECRETS WORKFLOW 1: CREATION & STORAGE



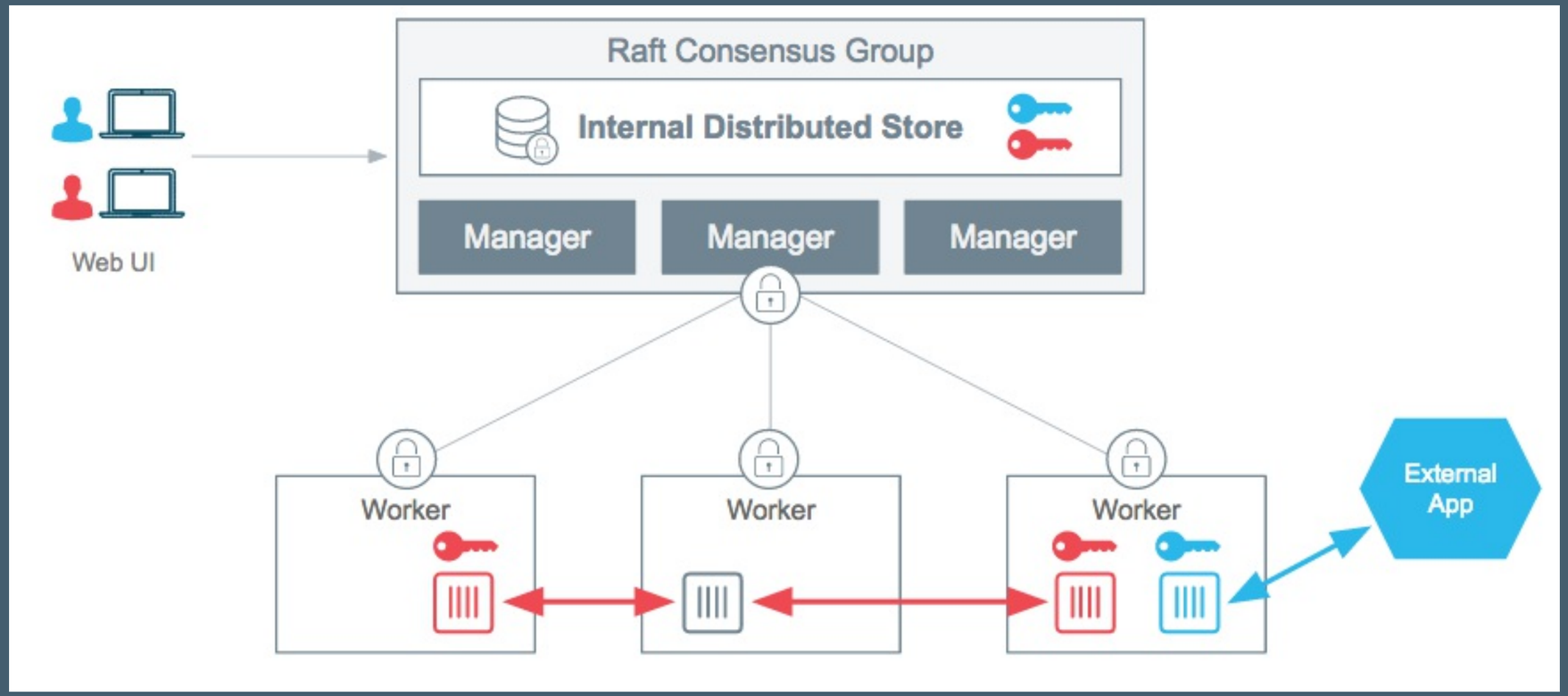
- Transmitted over mutual TLS
- Encrypted at rest
- Part of the Raft datastore (therefore HA)
- Label-based access control

SECRETS WORKFLOW 2: DISTRIBUTION



- Secret access is per service
- Managers propagate secrets (TLS) to only the containers that need them ("Least Privilege")
- tmpfs-mounted unencrypted in container at /run/secrets/<secret_name>
- Deleted when service loses access to a secret

SECRETS WORKFLOW 3: SECRET USAGE



EXERCISE: SECRETS

Work through the 'Docker Secrets' exercise in the Docker Fundamentals Exercise book.

FUNDAMENTAL ORCHESTRATION TAKEAWAYS

- Distributed Application Architecture orchestrates one or more services across one or more nodes
- Docker Swarm and Docker Compose provide native node and container orchestration support
- Services, Swarms and Stacks enhance scalability and stability

DOCKER FUNDAMENTALS

Please take our feedback survey:

https://dockertraining.typeform.com/to/gjciUl

Get in touch: training@docker.com

training.docker.com