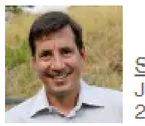
Sources: Microsoft Tried To Buy Docker for $4B

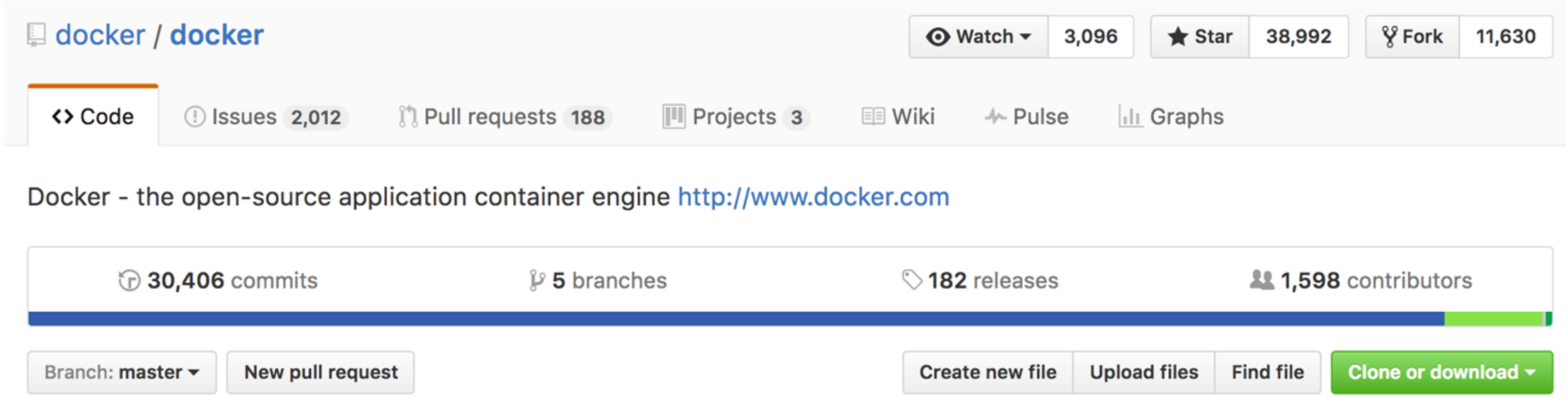


Scott Raynovich
June 27, 2016
2:05 pm MT

Source
https://www.youtube.com/watch?v=ZNdc4-yFTeA
https://github.com/docker/labs/blob/master/slides/docker-java-dockercon-2017.pdf

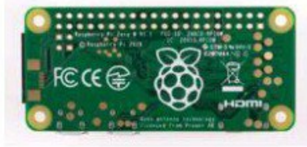
What is Docker?

- Open source project and company



- Used to create containers for software applications

Today **Docker** runs on -



archlinux

alpine Linux

FreeBSD

Windows 10

debian

Mac OS

CentOS

SUSE

ubuntu

openstack.

Windows Server

Google Cloud Platform

Microsoft Azure

DigitalOcean

amazon web services

Docker inside Docker

THE EVOLUTION OF

SOFTWARE ARCHITECTURE

1990's

SPAGHETTI-ORIENTED
ARCHITECTURE
(aka Copy & Paste)

2000's

LASAGNA-ORIENTED
ARCHITECTURE
(aka Layered Monolith)

2010's

RAVIOLI-ORIENTED
ARCHITECTURE
(aka Microservices)

WHAT'S NEXT?

PROBABLY PIZZA-ORIENTED ARCHITECTURE

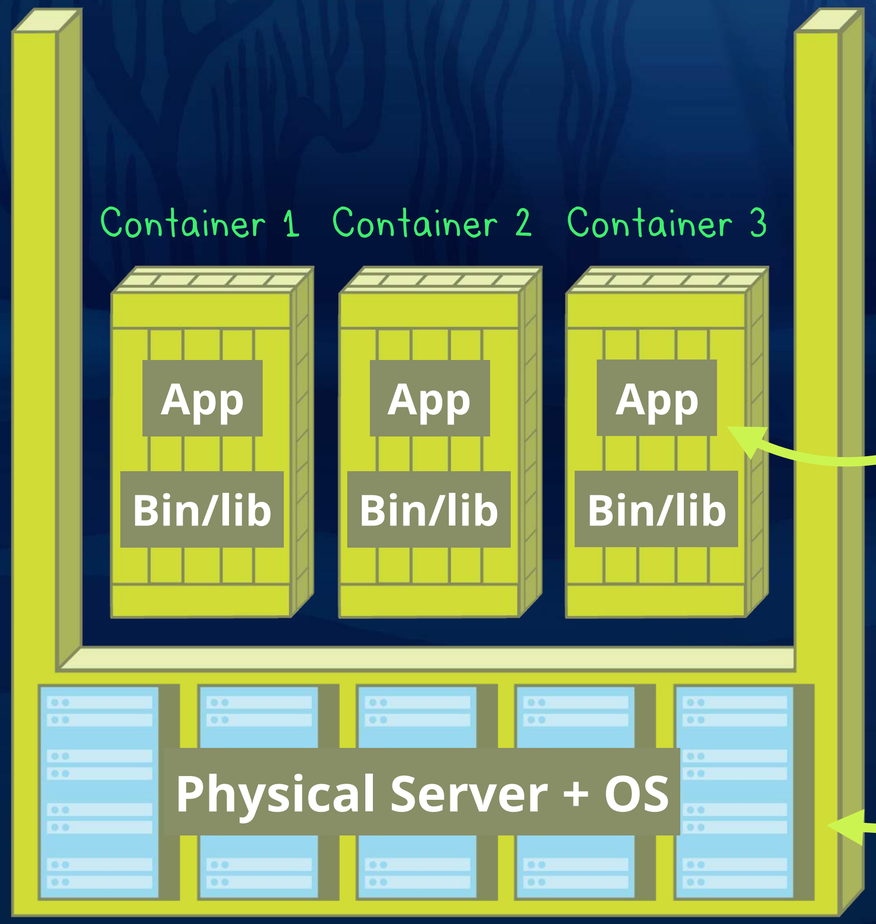
By @benorama

Containers & Images

Running Your First Container

What Are Linux Containers?

Linux containers are a way to create isolated environments that can run code while sharing a single operating system.

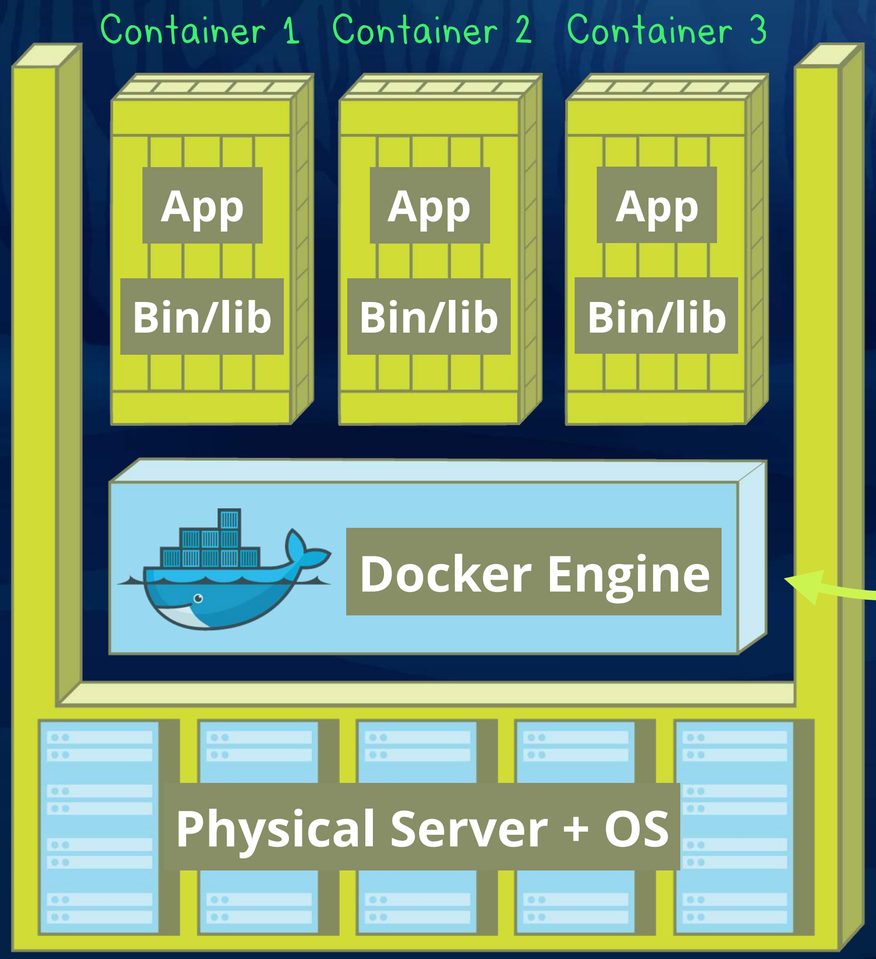


Each container is completely isolated from the others

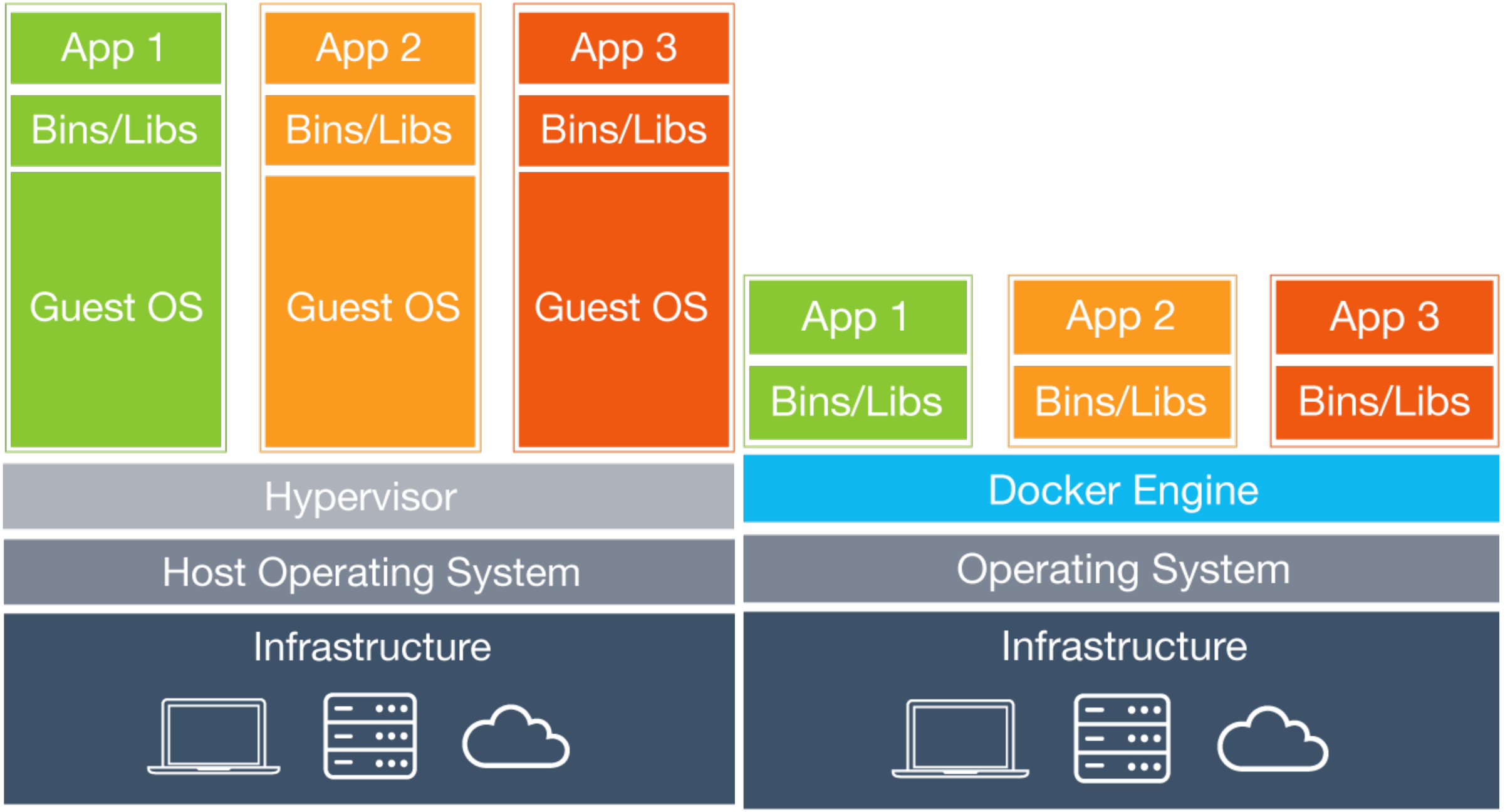
A computer somewhere – could even be the laptop or desktop computer you're using right now!

Why Docker?

Managing Linux containers is hard.
Docker is a tool that makes it much easier to manage Linux containers.



Application that manages containers behind the scenes


App 1
Bins/Libs
Guest OS
App 2
Bins/Libs
Guest OS
App 3
Bins/Libs
Guest OS
Hypervisor
Host Operating System
Infrastructure
App 1
Bins/Libs
App 2
Bins/Libs
App 3
Bins/Libs
Docker Engine
Operating System
Infrastructure

6

How Can Docker Help Me?

There are many different ways people can use Docker.

Developers

- Create contained, controlled dev environment
- Share identical dev environment across team
- Bug reporting

IT Ops

- Testing
- Deployment

Installing Docker

The simplest way to install Docker is to download one of the official Docker applications.

Applications

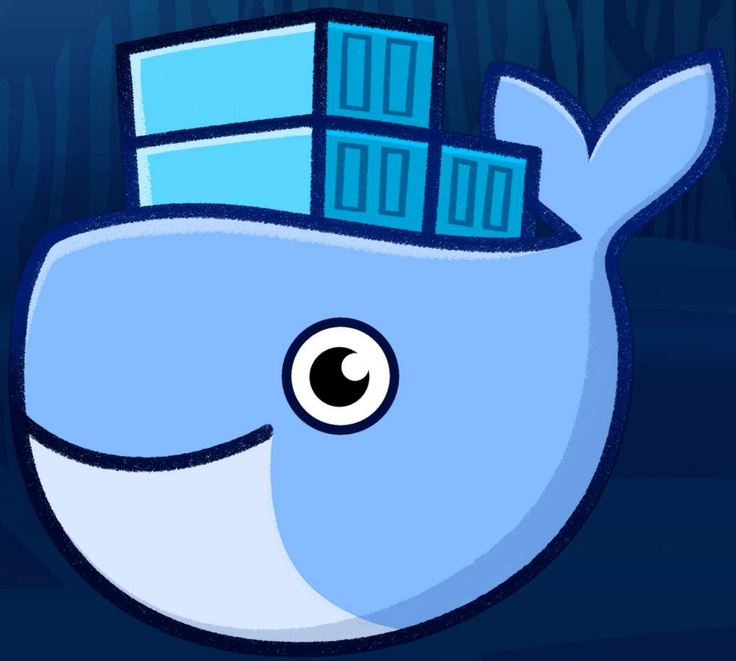
Docker for Mac - Community Edition

Docker for Windows - Community Edition

Installation Instructions

Linux AWS

Azure Windows Server



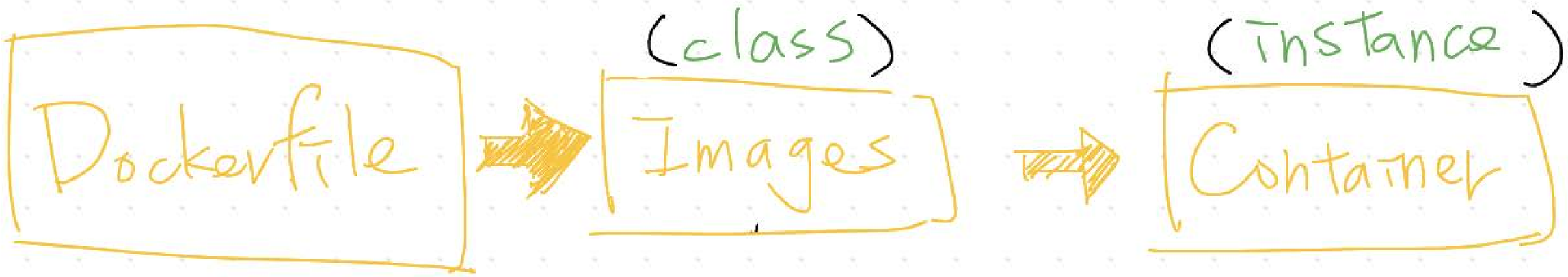
try docker

Containers & Images

An image is a blueprint for creating a container.

Pre-built images available in Docker Store (and Docker Hub)

trydocker

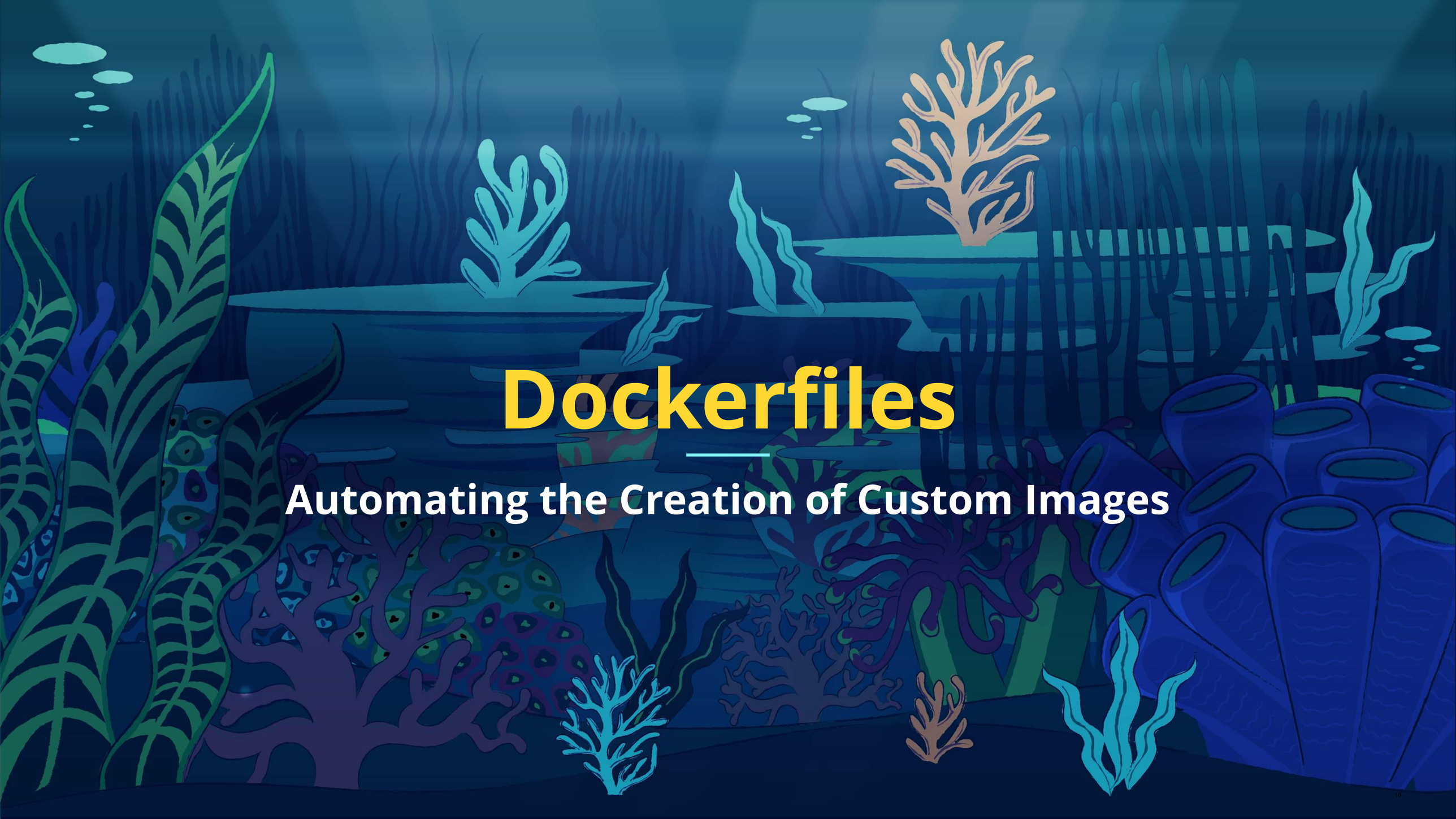


設定應用程式所需要一切所需要的：

- 使用哪一個image當作目前image的基底
- 環境變數
- port的內外對應
  ex: 外面的80 port 對應 container內的 80 port

一切應用程式所需要的：
程式碼 \ 環境變數 \ 套件 \ 設定檔

run-time instance

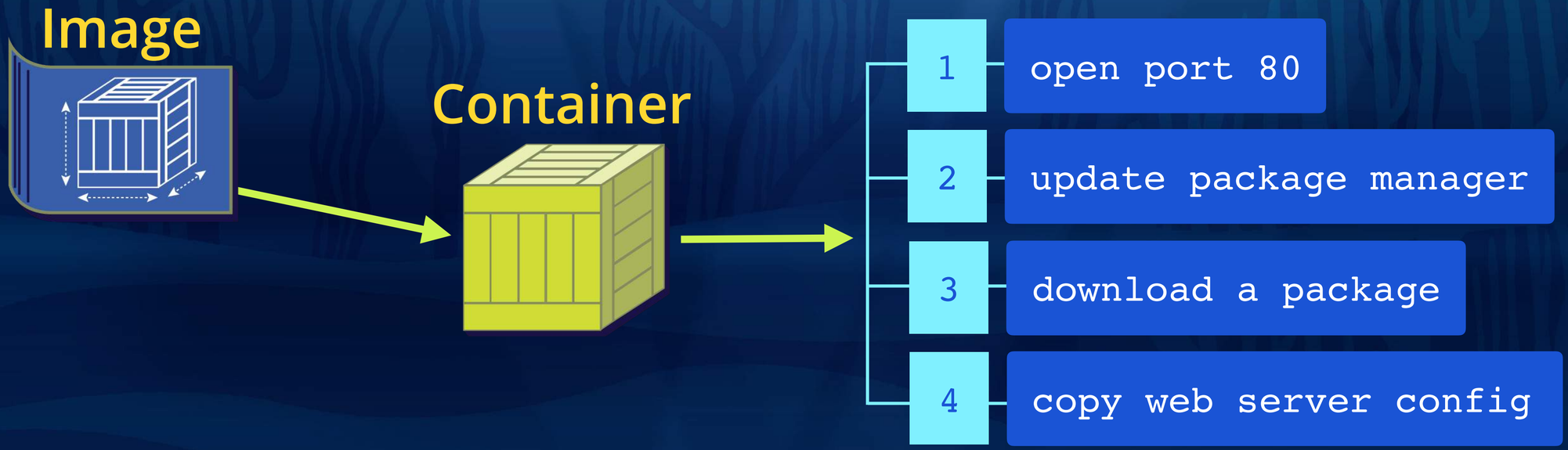


Dockerfiles

Automating the Creation of Custom Images

The Problem: Creating Containers Is Clunky

Creating containers from the command line works, but it quickly gets a little clunky the more customization that you need to do.



Each step modifies the container a little bit

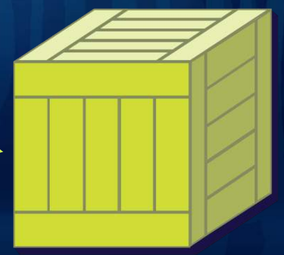
Dockerfiles help make this process slightly less manual

Dockerfiles Help You Create Images

A Dockerfile is a specially formatted text file where you can add a list of instructions that will run and result in a new image that can be used to make a container.

Dockerfile

Image



Container

1	open port 80
2	update package manager
3	download a package
4	copy web server config

The steps in a Dockerfile are run and turned into a single image

```
FROM httpd:2.4
EXPOSE 80
RUN apt-get update
COPY ./my-httpd.conf /usr/local/
apache2/conf/httpd.conf
```

Build

Develop an app using Docker containers with any language and any toolchain.

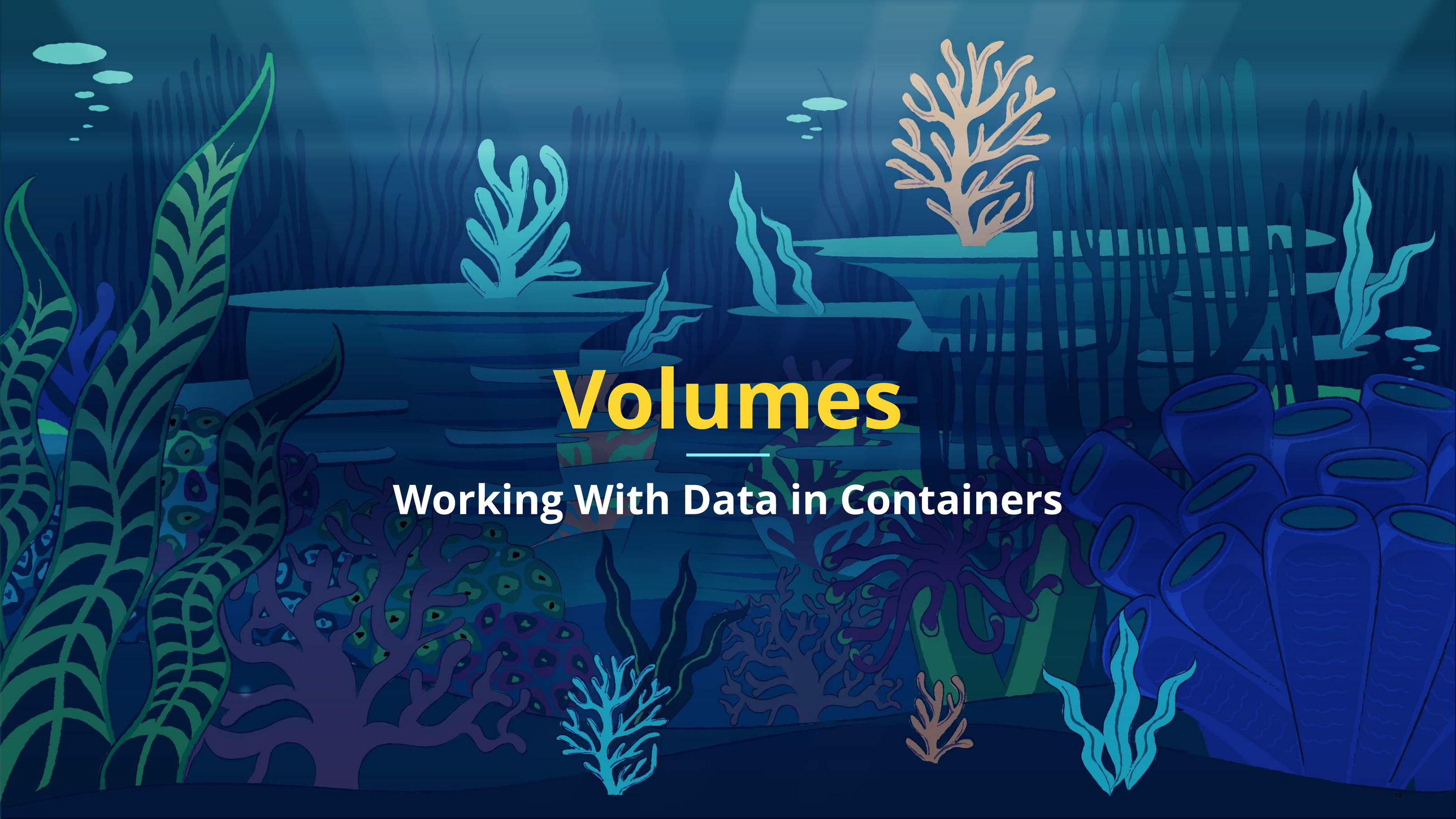
```
FROM ubuntu

CMD echo “Hello world”
```

```
FROM openjdk

COPY target/hello.jar /usr/src/hello.jar

CMD java -cp /usr/src/hello.jar org.example.App
```



Volumes

Working With Data in Containers

Getting Data Into Containers

If the image you're building a container with doesn't already contain application files, you'll need an extra step to get them into your container.

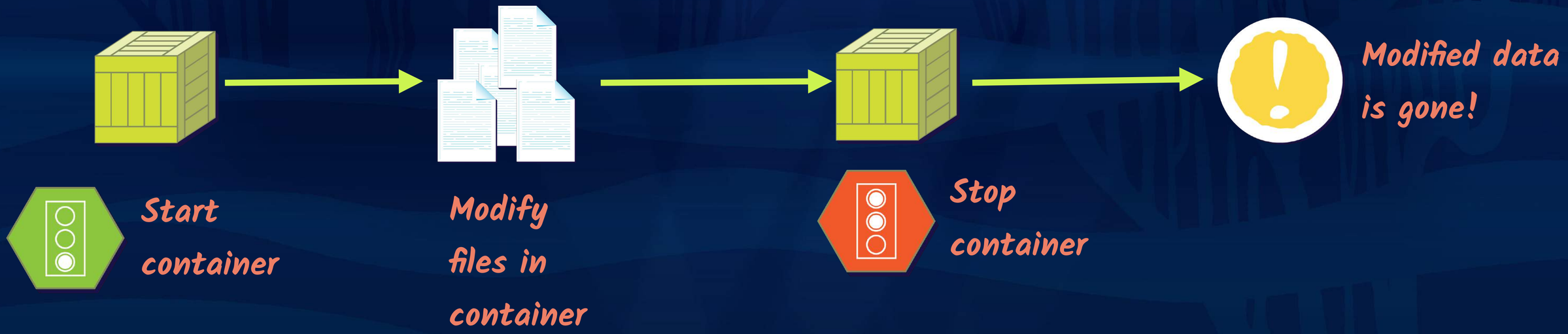
Copy a file into a container from the command line

Copy a file into an image with instructions in a Dockerfile

trydocker

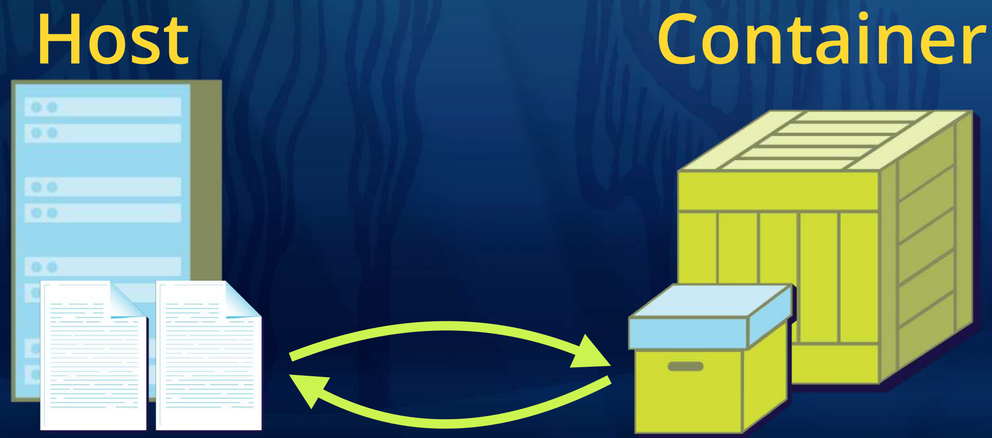
The Problem: Containers Don't Persist Data

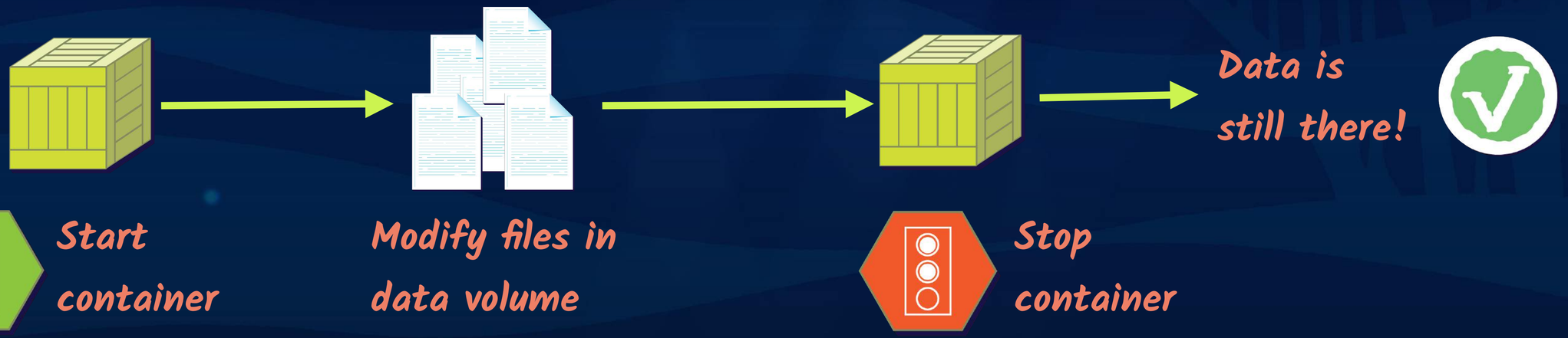
Our containers aren't really doing much right now because we don't have a way to get data in them.

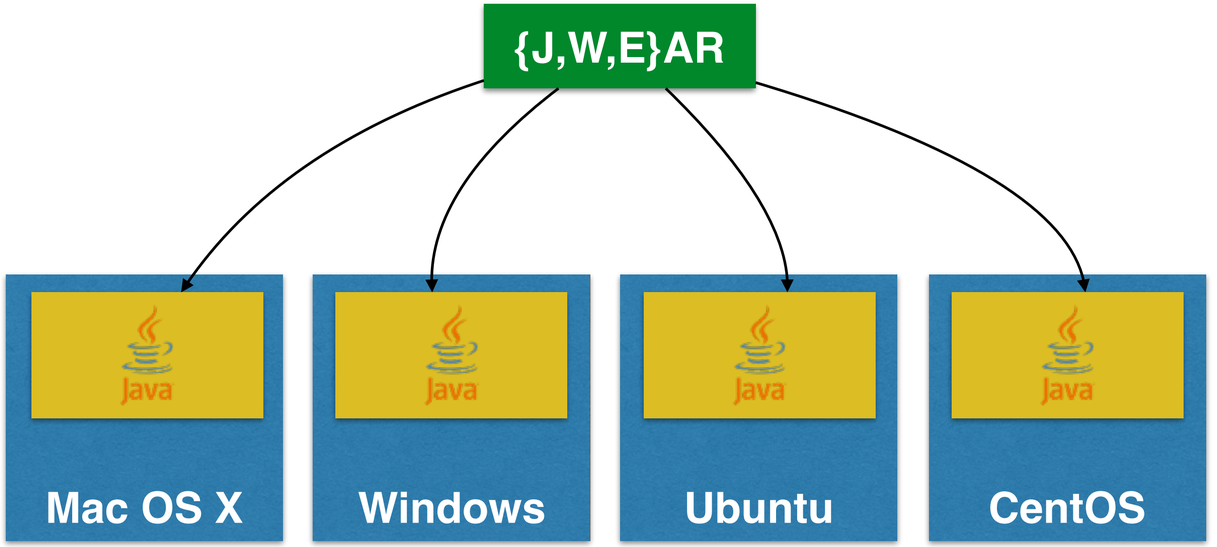


The Solution: Data Volumes

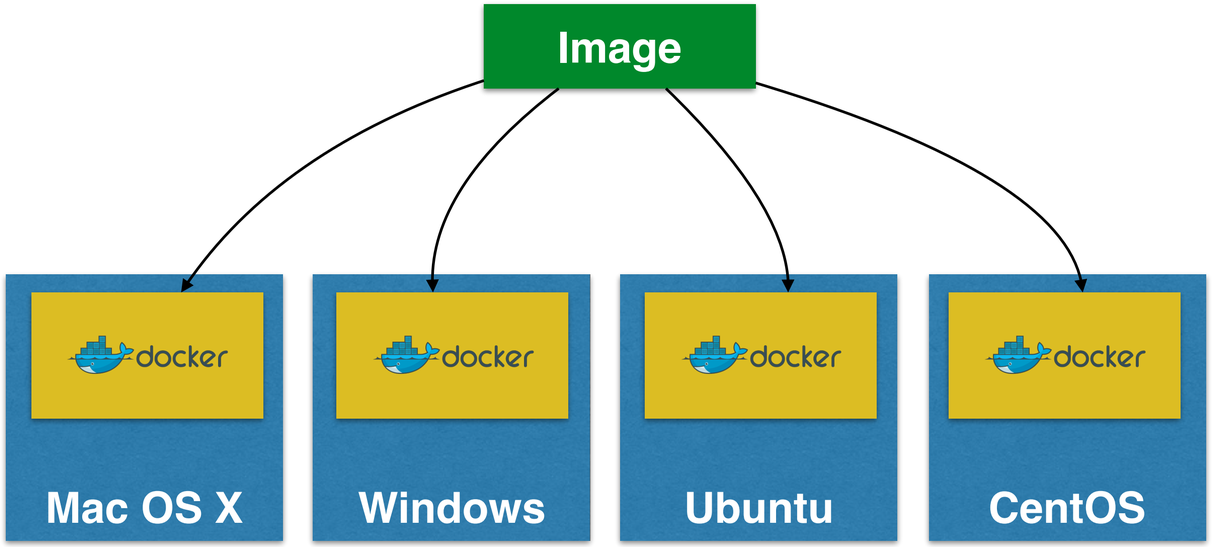
Data volumes expose files on your host machine to the container.





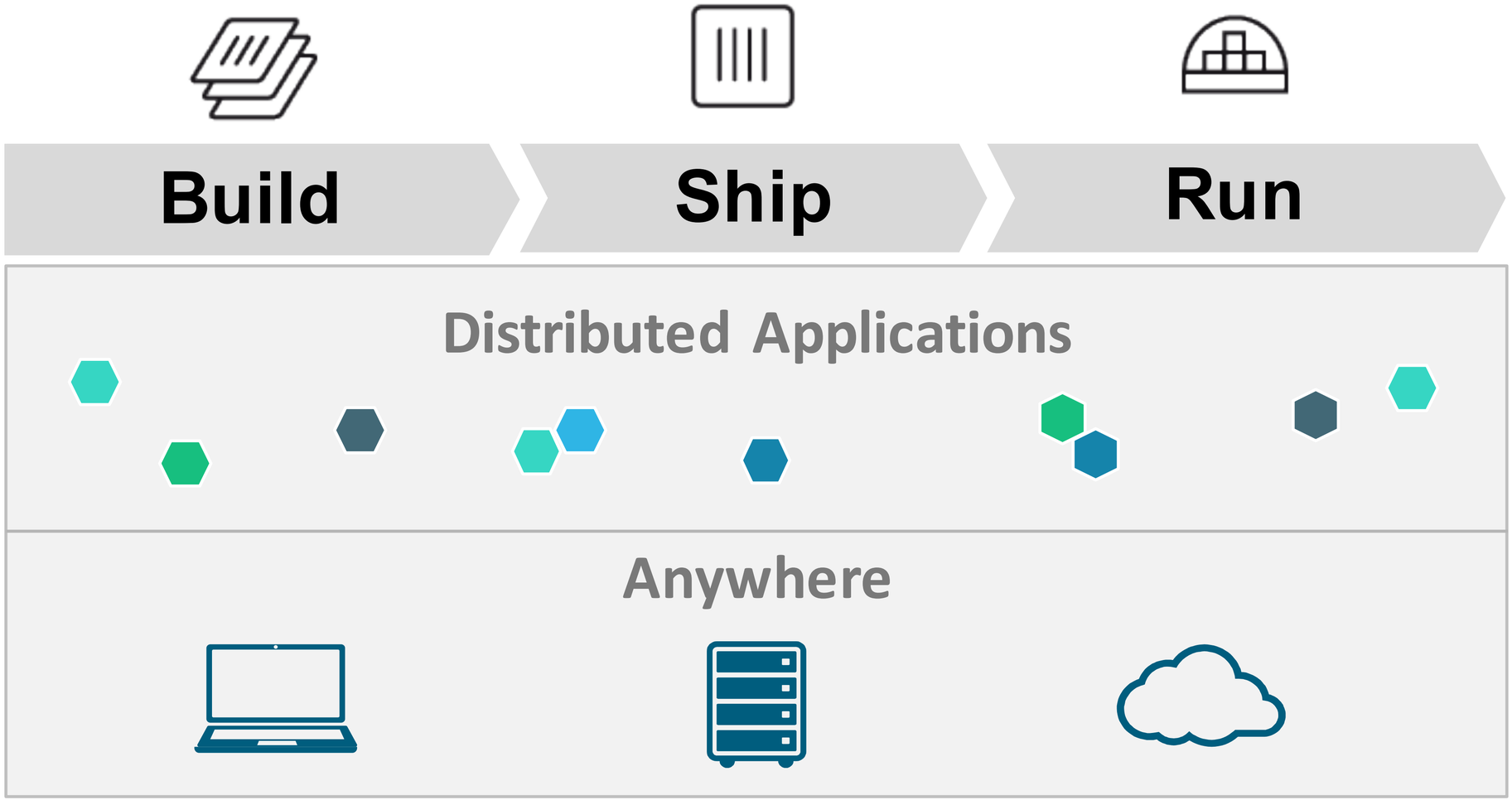


WORA = Write Once Run Anywhere



PODA = Package Once Deploy Anywhere

Docker Mission



Dockerfile reference

Usage

Format

Parser directives

escape

Environment replacement

.dockerignore file

FROM

RUN

Known issues (RUN)

CMD

LABEL

MAINTAINER (deprecated)

EXPOSE

ENV

ADD

COPY

ENTRYPOINT

Exec form ENTRYPOINT example

Shell form ENTRYPOINT example

Understand how CMD and ENTRYPOINT interact

VOLUME

USER

WORKDIR

ARG

Impact on build caching

ONBUILD

STOPSIGNAL

HEALTHCHECK

SHELL

Dockerfile examples

https://docs.docker.com/engine/reference/builder/

Docker Workflow

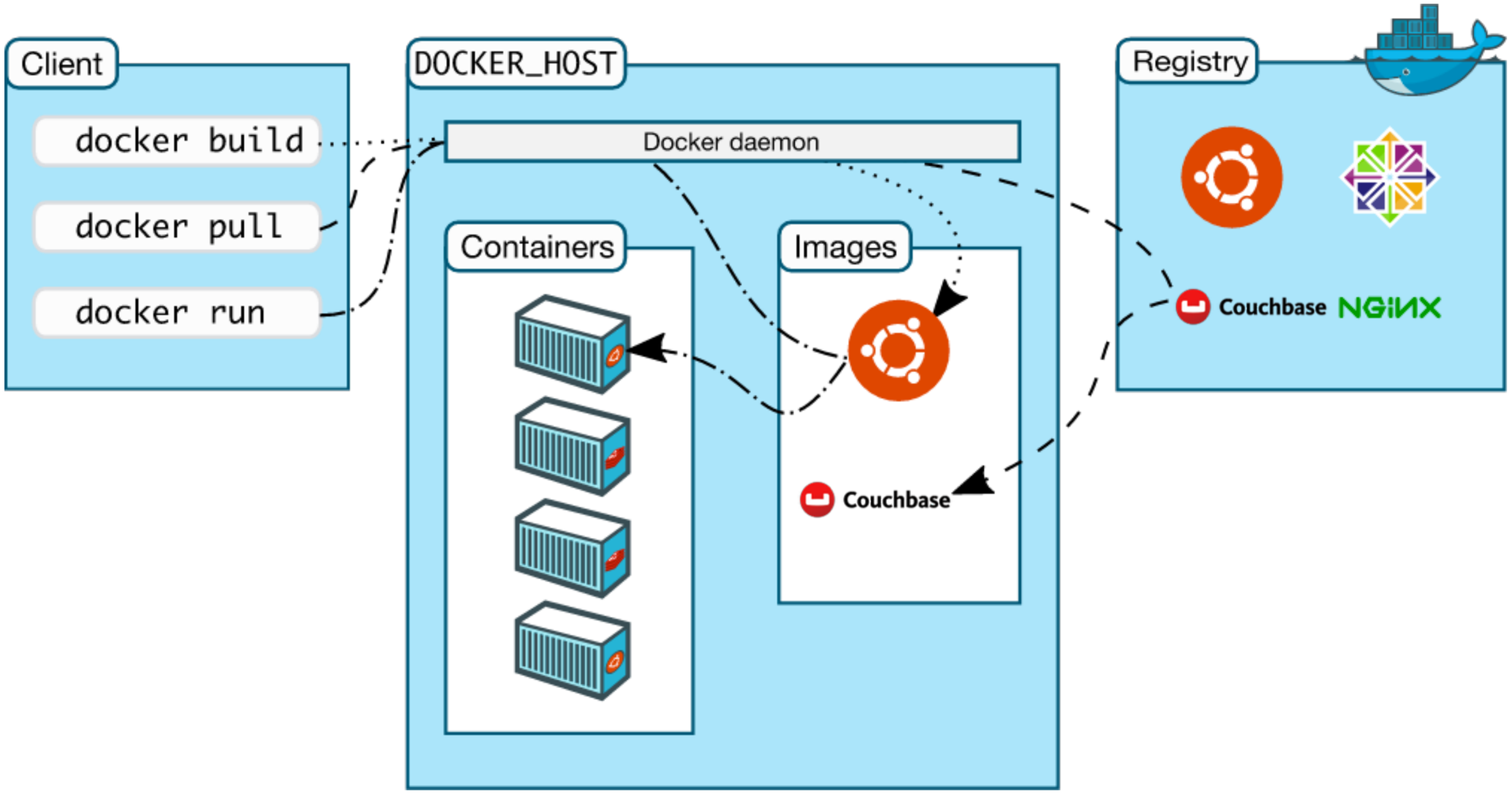


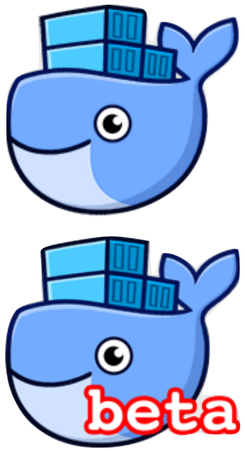
Image Layers - OpenJDK

```
~ > docker image ls openjdk
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
openjdk             latest              d23bdf5b1b1b        5 days ago          643 MB
~ > docker image history openjdk
IMAGE               CREATED             CREATED BY                                      SIZE
COMMENT
d23bdf5b1b1b        5 days ago          /bin/sh -c /var/lib/dpkg/info/ca-certifica...   419 kB
<missing>           5 days ago          /bin/sh -c set -x  && apt-get update  && a...   352 MB
<missing>           5 days ago          /bin/sh -c #(nop)  ENV CA_CERTIFICATES_JAV...   0 B
<missing>           5 days ago          /bin/sh -c #(nop)  ENV JAVA_DEBIAN_VERSION...   0 B
<missing>           5 days ago          /bin/sh -c #(nop)  ENV JAVA_VERSION=8u111       0 B
<missing>           5 days ago          /bin/sh -c #(nop)  ENV JAVA_HOME=/usr/lib/...   0 B
<missing>           5 days ago          /bin/sh -c {   echo '#!/bin/sh';   echo 's...   87 B
<missing>           5 days ago          /bin/sh -c #(nop)  ENV LANG=C.UTF-8             0 B
<missing>           5 days ago          /bin/sh -c echo 'deb http://deb.debian.org...   55 B
<missing>           5 days ago          /bin/sh -c apt-get update && apt-get insta...   1.29 MB
<missing>           5 days ago          /bin/sh -c apt-get update && apt-get insta...   123 MB
<missing>           5 days ago          /bin/sh -c apt-get update && apt-get insta...   44.3 MB
<missing>           6 days ago          /bin/sh -c #(nop)  CMD ["/bin/bash"]            0 B
<missing>           6 days ago          /bin/sh -c #(nop) ADD file:89ecb642d662ee7...   123 MB
```

Docker for AWS/Azure

- Amazon Web Services
  - **EKS**
  - Integrated with Auto**S**caling, ELB, and **EBS.**
- Azure
  - Integrated with VM Scale Sets for autoscaling, Azure Load Balancer, Azure Storage
- docker.com

Docker for Mac/Windows



- Native application and UI
- Auto update capability
- No additional software required, e.g. VirtualBox
  - OSX: xhyve VM using **`Hypervisor.framework`**
  - Windows: Hyper-V VM
- Download: docker.com/getdocker
- Requires Yosemite 10.10+ or Windows 10 64-bit

Container management commands

command	description
`docker create` *image* [*command*] `docker run` *image* [*command*]	create the container = `create` + `start`
`docker start` *container...* `docker stop` *container...* `docker kill` *container...* `docker restart` *container...*	start the container graceful[2] stop kill (SIGKILL) the container = `stop` + `start`
`docker pause` *container...* `docker unpause` *container...*	suspend the container resume the container
`docker rm` [`-f`[3]] *container...*	destroy the container

[2] send SIGTERM to the main process + SIGKILL 10 seconds later

[3] `-f` allows removing running containers (= `docker kill` + `docker rm`)

Container management commands

command	description
`docker create` *image* [*command*] `docker run` *image* [*command*]	create the container = `create` + `start`
`docker start` *container...* `docker stop` *container...* `docker kill` *container...* `docker restart` *container...*	start the container graceful[2] stop kill (SIGKILL) the container = `stop` + `start`
`docker pause` *container...* `docker unpause` *container...*	suspend the container resume the container
`docker rm` [`-f`[3]] *container...*	destroy the container

[2] send SIGTERM to the main process + SIGKILL 10 seconds later

[3] `-f` allows removing running containers (= `docker kill` + `docker rm`)

Interacting with the container

command	description
`docker attach` *container*	attach to a running container (stdin/stdout/stderr)
`docker cp` *container:path hostpath*|- `docker cp` *hostpath*|- *container:path*	copy files from the container copy files into the container
`docker export` *container*	export the content of the container (tar archive)
`docker exec` *container args...*	run a command in an existing container (**useful** for debugging)
`docker wait` *container*	wait until the container terminates and return the exit code
`docker commit` *container image*	commit a new docker image (snapshot of the container)

Image management commands

command	description
`docker images`	list all local images
`docker history` *image*	show the image history (list of ancestors)
`docker inspect` *image...*	show low-level infos (in json format)
`docker tag` *image tag*	tag an image
`docker commit` *container image*	create an image (from a container)
`docker import` *url*|- [*tag*]	create an image (from a tarball)
`docker rmi` *image...*	delete images

Image management commands

command	description
`docker images`	list all local images
`docker history` *image*	show the image history (list of ancestors)
`docker inspect` *image...*	show low-level infos (in json format)
`docker tag` *image tag*	tag an image
`docker commit` *container image*	create an image (from a container)
`docker import` *url*|- [*tag*]	create an image (from a tarball)
`docker rmi` *image...*	delete images

Cheatsheet for Docker CLI

Run a new Container

Start a new Container from an Image
```
docker run IMAGE
docker run nginx
```

...and assign it a name
```
docker run --name CONTAINER IMAGE
docker run --name web nginx
```

...and map a port
```
docker run -p HOSTPORT:CONTAINERPORT IMAGE
docker run -p 8080:80 nginx
```

...and map all ports
```
docker run -P IMAGE
docker run -P nginx
```

...and start container in background
```
docker run -d IMAGE
docker run -d nginx
```

...and assign it a hostname
```
docker run --hostname HOSTNAME IMAGE
docker run --hostname srv nginx
```

...and add a dns entry
```
docker run --add-host HOSTNAME:IP IMAGE
```

...and map a local directory into the container
```
docker run -v HOSTDIR:TARGETDIR IMAGE
docker run -v ~/:/usr/share/nginx/html nginx
```

...but change the entrypoint
```
docker run -it --entrypoint EXECUTABLE IMAGE
docker run -it --entrypoint bash nginx
```

Manage Containers

Show a list of running containers
```
docker ps
```

Show a list of all containers
```
docker ps -a
```

Delete a container
```
docker rm CONTAINER
docker rm web
```

Delete a running container
```
docker rm -f CONTAINER
docker rm -f web
```

Delete stopped containers
```
docker container prune
```

Stop a running container
```
docker stop CONTAINER
docker stop web
```

Start a stopped container
```
docker start CONTAINER
docker start web
```

Copy a file from a container to the host
```
docker cp CONTAINER:SOURCE TARGET
docker cp web:/index.html index.html
```

Copy a file from the host to a container
```
docker cp TARGET CONTAINER:SOURCE
docker cp index.html web:/index.html
```

Start a shell inside a running container
```
docker exec -it CONTAINER EXECUTABLE
docker exec -it web bash
```

Rename a container
```
docker rename OLD_NAME NEW_NAME
docker rename 096 web
```

Create an image out of container
```
docker commit CONTAINER
docker commit web
```

Manage Images

Download an image
```
docker pull IMAGE[:TAG]
docker pull nginx
```

Upload an image to a repository
```
docker push IMAGE
docker push myimage:1.0
```

Delete an image
```
docker rmi IMAGE
```

Show a list of all Images
```
docker images
```

Delete dangling images
```
docker image prune
```

Delete all unused images
```
docker image prune -a
```

Build an image from a Dockerfile
```
docker build DIRECTORY
docker build .
```

Tag an image
```
docker tag IMAGE NEWIMAGE
docker tag ubuntu ubuntu:18.04
```

Build and tag an image from a Dockerfile
```
docker build -t IMAGE DIRECTORY
docker build -t myimage .
```

Save an image to .tar file
```
docker save IMAGE > FILE
docker save nginx > nginx.tar
```

Load an image from a .tar file
```
docker load -i TARFILE
docker load -i nginx.tar
```

Info & Stats

Show the logs of a container
```
docker logs CONTAINER
docker logs web
```

Show stats of running containers
```
docker stats
```

Show processes of container
```
docker top CONTAINER
docker top web
```

Show installed docker version
```
docker version
```

Get detailed info about an object
```
docker inspect NAME
docker inspect nginx
```

Show all modified files in container
```
docker diff CONTAINER
docker diff web
```

Show mapped ports of a container
```
docker port CONTAINER
docker port web
```

```
FROM python:3.6-alpine
RUN adduser -D microblog
WORKDIR /home/microblog
COPY requirements.txt requirements.txt
RUN apk add --no-cache --update gcc musl-dev libffi-dev openssl-dev
RUN python3 -m venv venv
RUN venv/bin/pip3 install --upgrade pip
RUN venv/bin/pip3 install -r requirements.txt
RUN venv/bin/pip3 install gunicorn
COPY app app
COPY migrations migrations
COPY microblog.py config.py run.py boot.sh ./
RUN chmod +x boot.sh
ENV FLASK_APP run.py
RUN chown -R microblog:microblog ./
USER microblog
EXPOSE 5000
ENTRYPOINT ["./boot.sh"]
```

Docker default runs as root!
You should alway create user for your container!

Docker Compose

- Defining and running **multi-container** applications
- Configuration defined in one or more files
  - `docker-compose.yml` (default)
  - `docker-compose.override.yml` (default)
  - Multiple files specified using `-f`
- Deployed as Docker Stack
- Great for dev, staging, and CI

Docker Compose - One Service


```
version: “3”
services:
  db:
   image: couchbase
   volumes:
     - ~/couchbase:/opt/couchbase/var
   ports:
     - 8091:8091
     - 8092:8092
     - 8093:8093
     - 11210:11210
```

```
docker-compose up -d
```

```
docker stack deploy \
--compose-file=docker-compose.yml \
couchbase
```

Docker Compose - Two Services

GET
POST
PUT
DELETE

CRUD
using
N1QL

Couchbase

Docker Compose - Two Services

```
version: "3"
services:
  db:
    image: arungupta/couchbase:travel
    ports:
      - 8091:8091
      - 8092:8092
      - 8093:8093
      - 11210:11210
  web:
    image: arungupta/couchbase-wildfly-javaee:travel
    environment:
      - COUCHBASE_URI=db
    ports:
      - 8080:8080
      - 9990:9990
```

```
docker stack deploy \
--compose-file=docker-compose.yml \
couchbase
```

Overriding Services in Docker Compose

```
web:
  image: jboss/wildfly
  ports:
    - 8080:8080
```

docker-compose.yml

```
web:
  ports:
    - 9080:8080
```

docker-compose.override.yml

Dev/Prod with Compose

```
db-dev:
  image: arungupta/couchbase
  ports:
    - . . .
web:
  image: arungupta/wildfly
  environment:
    - COUCHBASE_URI=db-dev:8093
  ports:
    - 8080:8080
```

docker-compose.yml

```
docker-compose up -d
```

```
web:
 environment:
    - COUCHBASE_URI=db-prod:8093
  ports:
    - 80:8080
db-prod:
  image: . . .
```

production.yml

```
docker-compose up
-f docker-compose.yml
-f production.yml
-d
```

Docker Compose Common Use Cases

Use Case	Command
Dev Setup	`docker-compose up`
Local/remote host	`DOCKER_HOST, DOCKER_TLS_VERIFY, DOCKER_CERT_PATH`
Single/multiple hosts	Integrated with Swarm
Multiple isolated environments	`docker-compose up -p <project>`

Automated test setup	`docker-compose up`
	`mvn test`
	`docker-compose down`

Dev/Prod Impedance mismatch	`docker-compose up -f docker-compose.yml -f production.yml`

Docker 1.13

- Deploy Compose services to Swarm
- CLI restructured
- Clean-up commands
- Monitoring commands
- Build improvements
- Improved CLI backwards compatibility
- Docker for AWS/Azure for Production

Docker 1.13 - Compose v3

- **`docker stack deploy`** now supports Compose file
  - Number of desired instances of each service
  - Rolling update
  - Server constraints

Docker 1.13 - CLI Restructured

```
Management Commands:
  checkpoint  Manage checkpoints
  container   Manage containers
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  volume      Manage volumes
```

Docker 1.13 - Cleanup Commands

- `docker system df` and `docker system cleanup`

```
docker system df
TYPE                TOTAL               ACTIVE              SIZE                RECLAIMABLE
Images              15                  1                   5.081 GB            4.498 GB (88%)
Containers          1                   0                   130.1 kB            130.1 kB (100%)
Local Volumes       7                   0                   110.1 MB            110.1 MB (100%)
```

Docker 1.13 - Monitoring Commands

- `docker service logs` and Prometheus endpoint

```
couchbase_db.1.rchu2uykeuuj@moby    | ++ set -m
couchbase_db.1.rchu2uykeuuj@moby    | ++ sleep 15
couchbase_db.1.rchu2uykeuuj@moby    | ++ /entrypoint.sh couchbase-server
couchbase_db.2.kjy7l14weao8@moby    | ++ set -m
couchbase_db.2.kjy7l14weao8@moby    | ++ sleep 15
couchbase_db.1.rchu2uykeuuj@moby    | Starting Couchbase Server -- Web UI av
couchbase_db.1.rchu2uykeuuj@moby    | ++ curl -v -X POST http://127.0.0.1:80
couchbase_db.2.kjy7l14weao8@moby    | ++ /entrypoint.sh couchbase-server
couchbase_db.2.kjy7l14weao8@moby    | Starting Couchbase Server -- Web UI av
```

Docker 1.13 - Build Improvements

- **`docker build --squash`**: Squash newly built layers into a single layer
- **`docker build --compress`**: Compress the build context using gzip

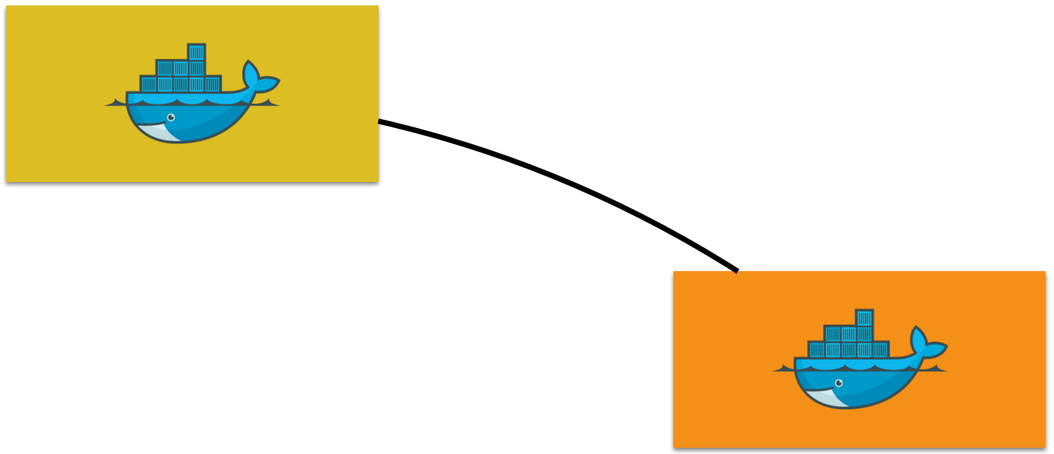
Swarm Mode

- New in 1.12
- Natively managing a cluster of Docker Engines called a Swarm
- Docker CLI to create a swarm, deploy apps, and manage swarm
  - Optional feature, need to be explicitly enabled
- No Single Point of Failure (SPOF)
- Declarative state model
- Self-organizing, self-healing
- Service discovery, load balancing and scaling
- Rolling updates

Swarm Mode: Initialize

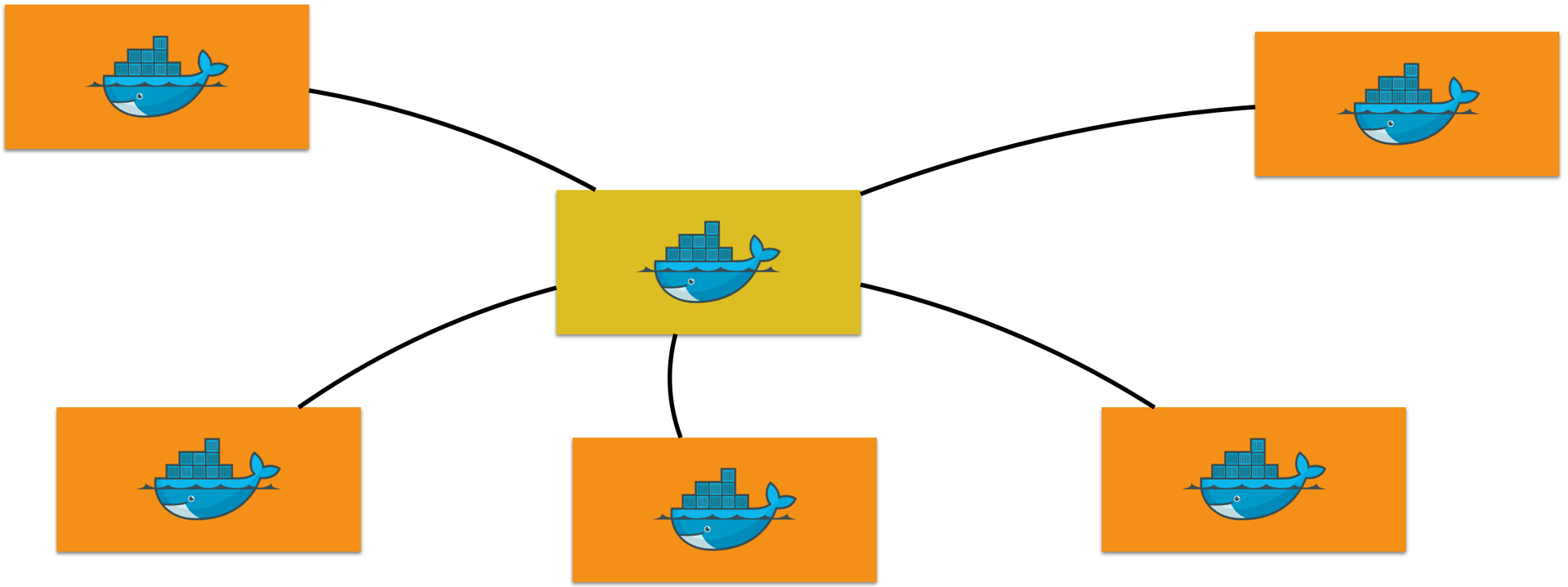
```
docker swarm init --listen-addr <ip>:2377
```

Swarm Mode: Add Worker



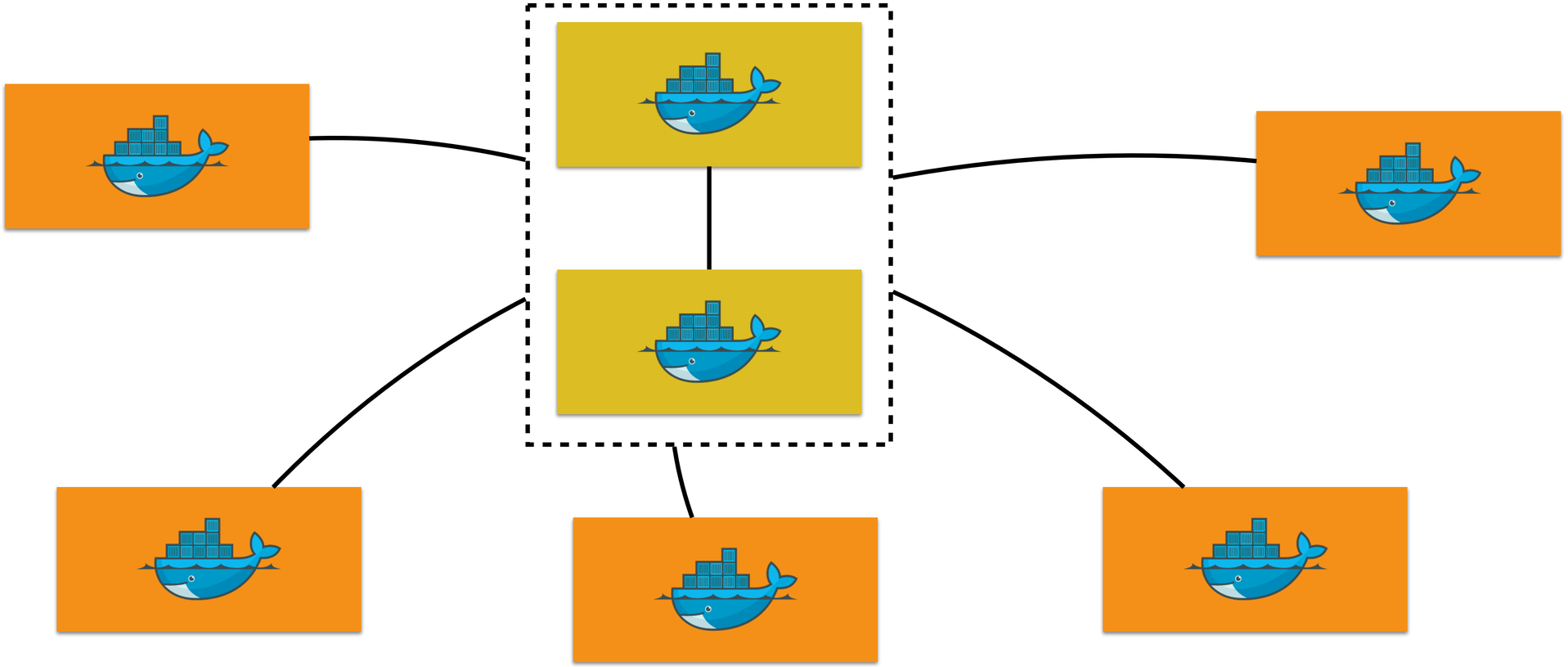
```
docker swarm join --token <worker_token> <manager>:2377
```

Swarm Mode: Add More Workers



```
docker swarm join --token <worker_token> <manager>:2377
```

Swarm Mode: Primary/Secondary Master



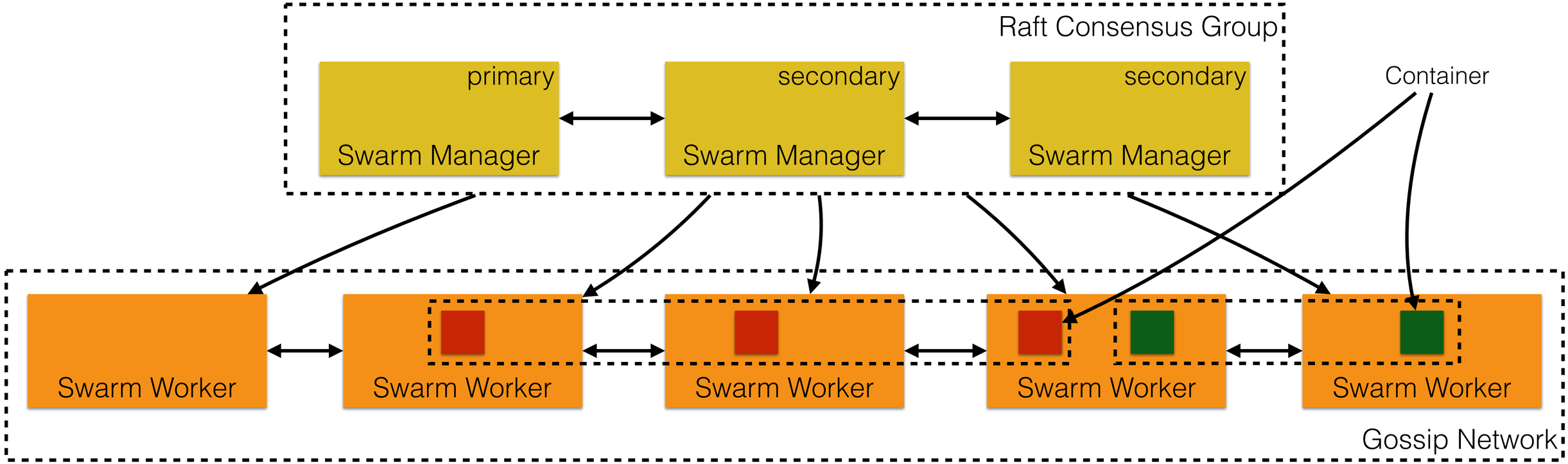
```
docker swarm join --manager --token <manager_token> --listen-
addr <master2>:2377 <master1>:2377
```

Swarm Mode using Docker Machine

Task	Command
Create manger	`docker-machine create -d virtualbox managerX`
Create worker	`docker-machine create -d virtualbox workerX`
Initialize Swarm mode	`docker swarm init --listen-addr <ip1> --advertise-addr <ip1>`
Manager token	`docker swarm join-token manager -q`
Worker token	`docker swarm join-token worker -q`
Manager X join	`docker swarm join --token manager_token --listen-addr <ipX> --advertise-addr <ipX> <ip1>`
Worker X join	`docker swarm join --token worker_token --listen-addr <ipX> --advertise-add <ipX> <ip1>`

https://github.com/docker/labs/blob/master/swarm-mode/quickstart/buildswarm-node-vbox-setup.sh

Swarm Mode: Protocols

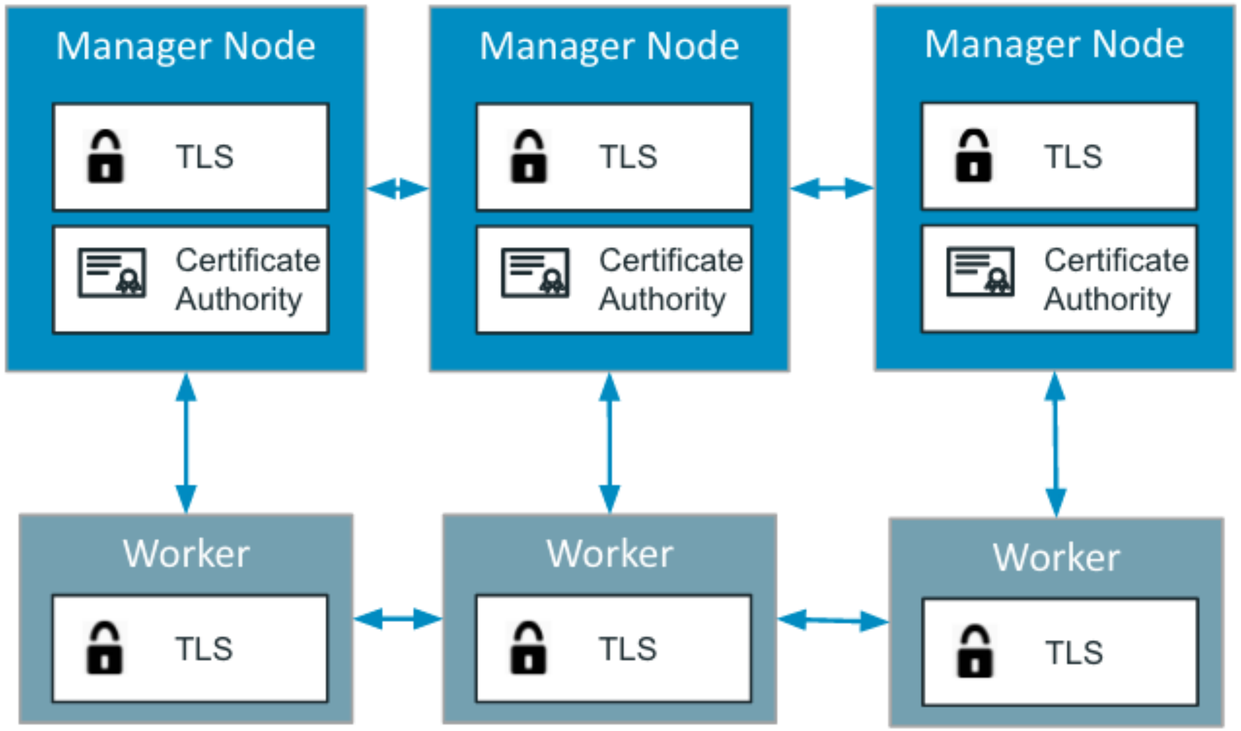


Strongly consistent
Replicated (Raft based)
Extremely fast (in-memory reads)

Swarm Mode in Production

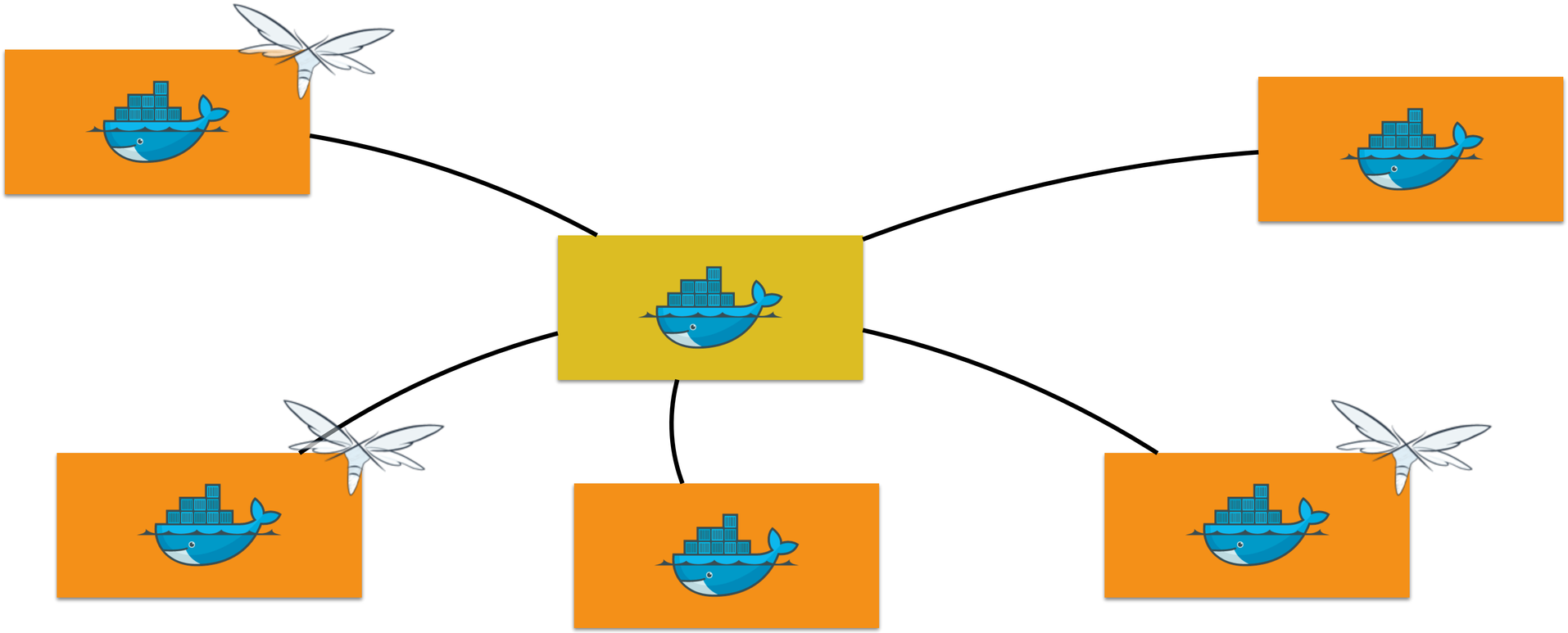
https://blog.online.net/2016/07/29/docker-swarm-an-analysis-of-a-very-large-scale-container-system/

Secure by Default



- Cryptographic node identity
- Automatic encryption and mutual authentication (TLS)
- Automatic cert rotation (90 days, can be up to 30 mins)
- External CA integration

Swarm Mode: Replicated Service

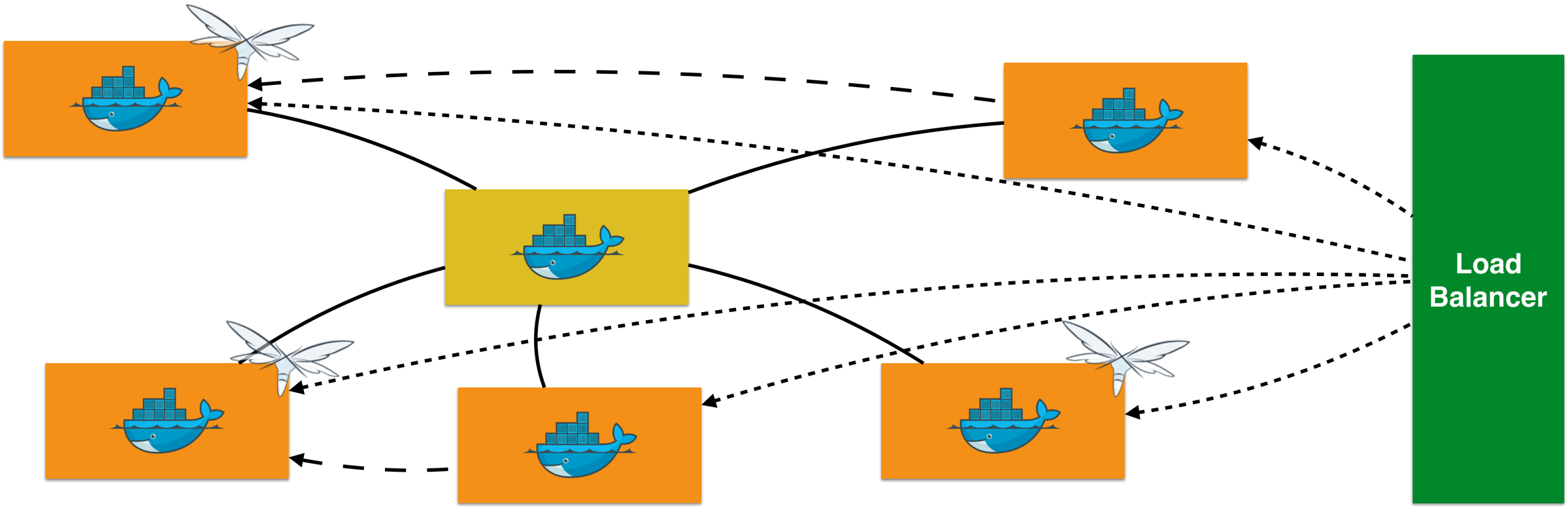


```
docker service create --replicas 3 --name web jboss/wildfly
```

Swarm Mode - Routing Mesh

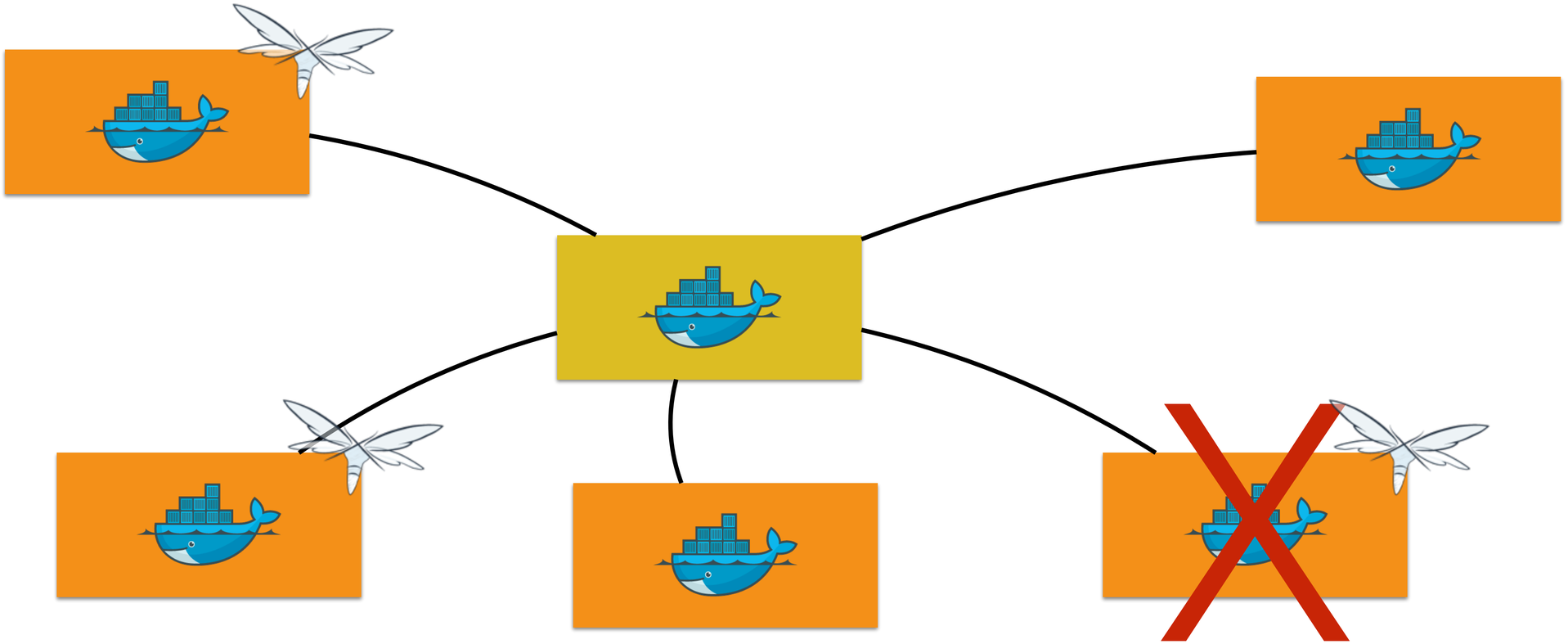
- Load balancers are host-aware, not container-aware
- Swarm mode introduces container-aware routing mesh
- Reroutes traffic from any host to a container
  - Reserves a Swarm-wide ingress port
  - Uses DNS-based service discovery

Swarm Mode: Routing Mesh

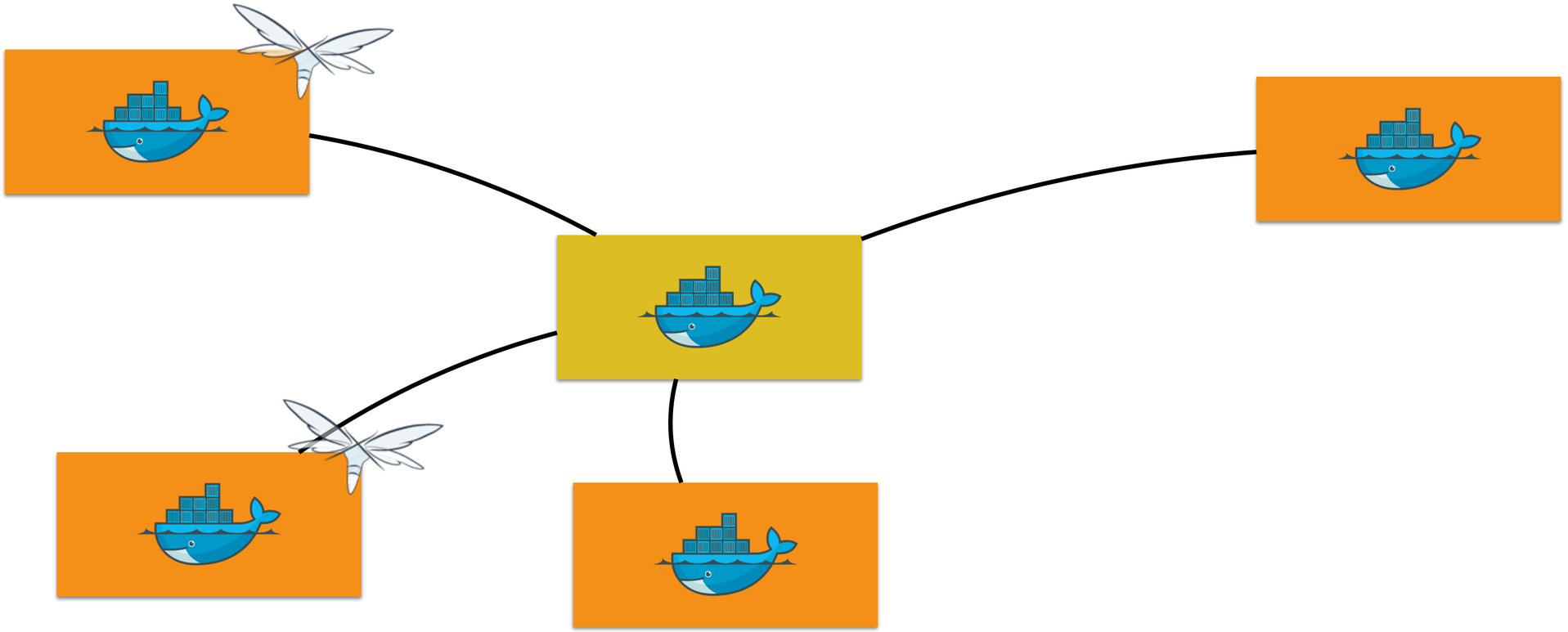



```
docker service create --replicas 3 --name web -p 8080:8080 jboss/wildfly
```

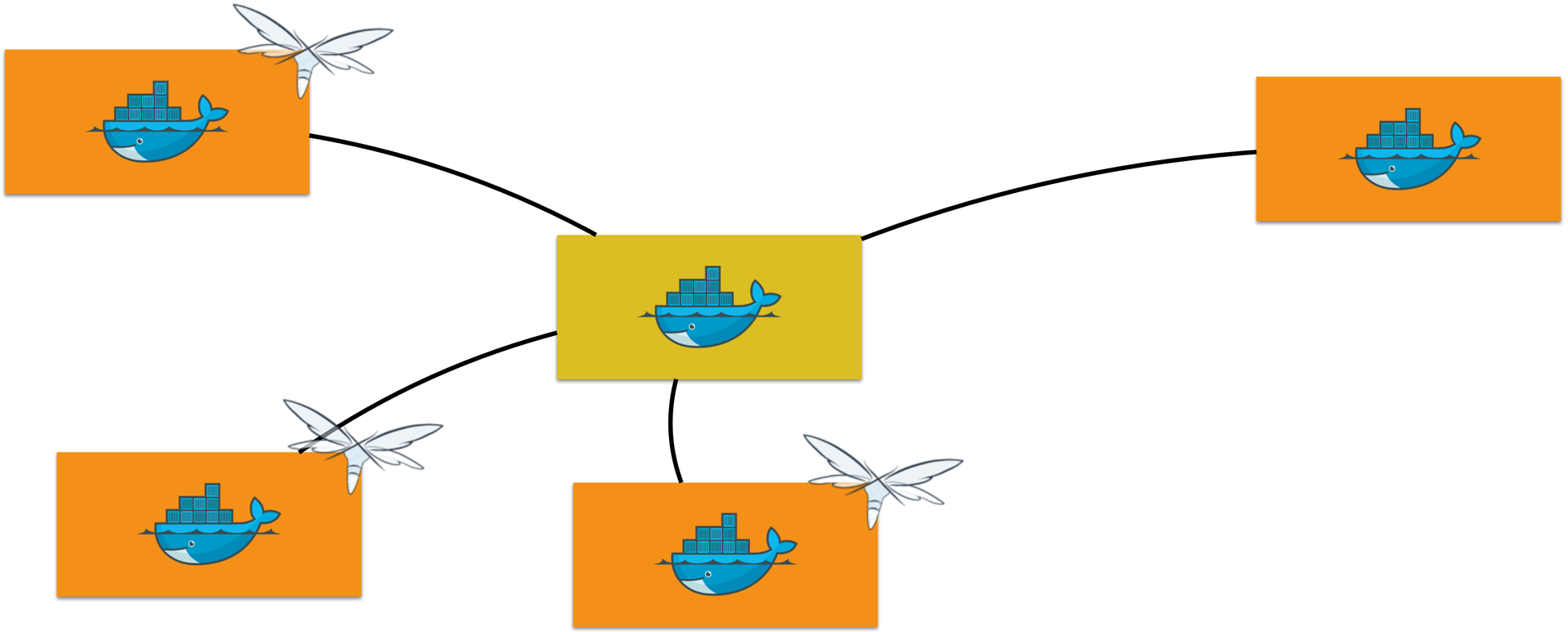
Swarm Mode: Node Failure



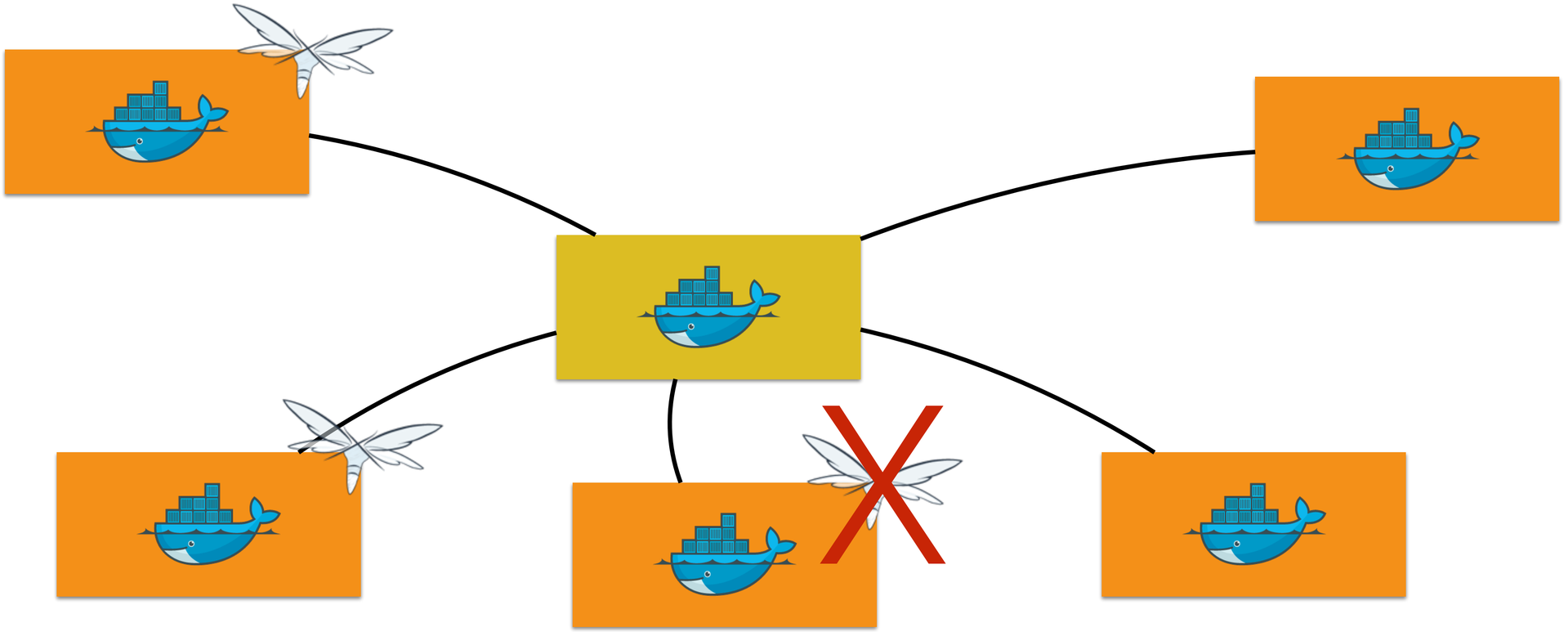
Swarm Mode: Desired != Actual



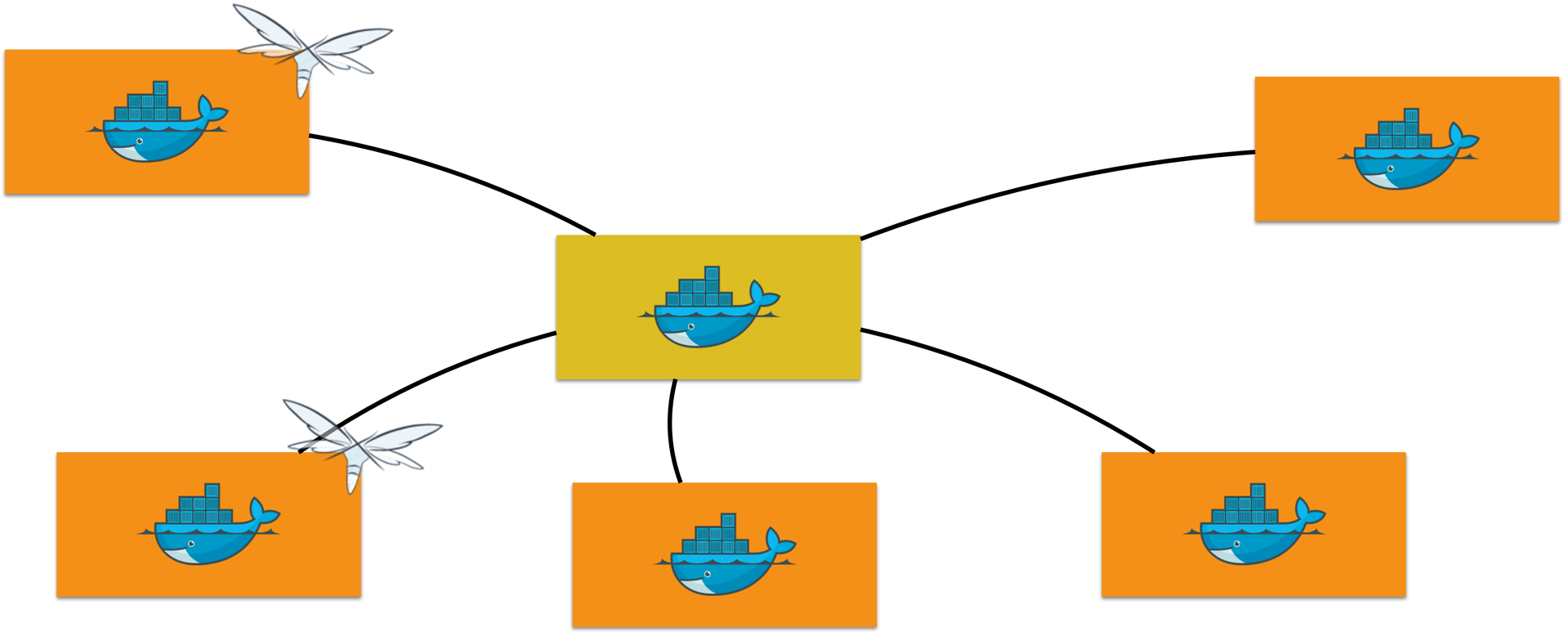
Swarm Mode: Reconcile



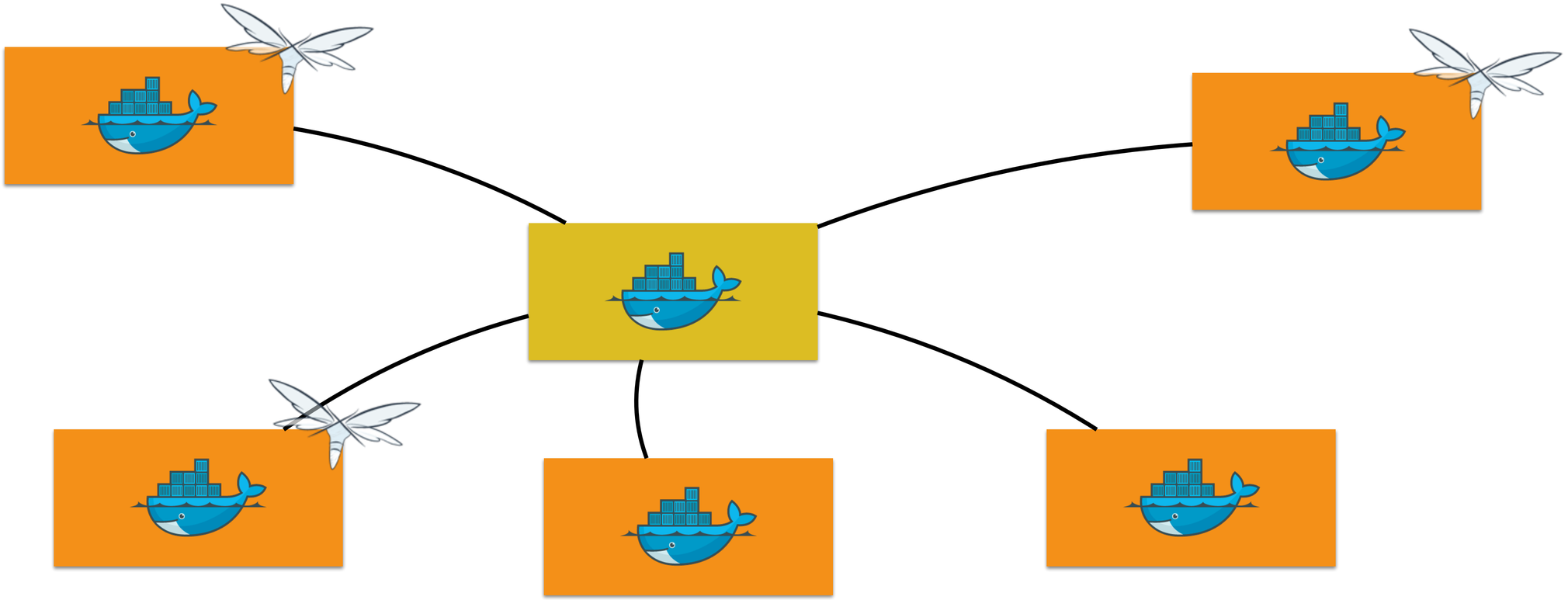
Swarm Mode: Container Failure



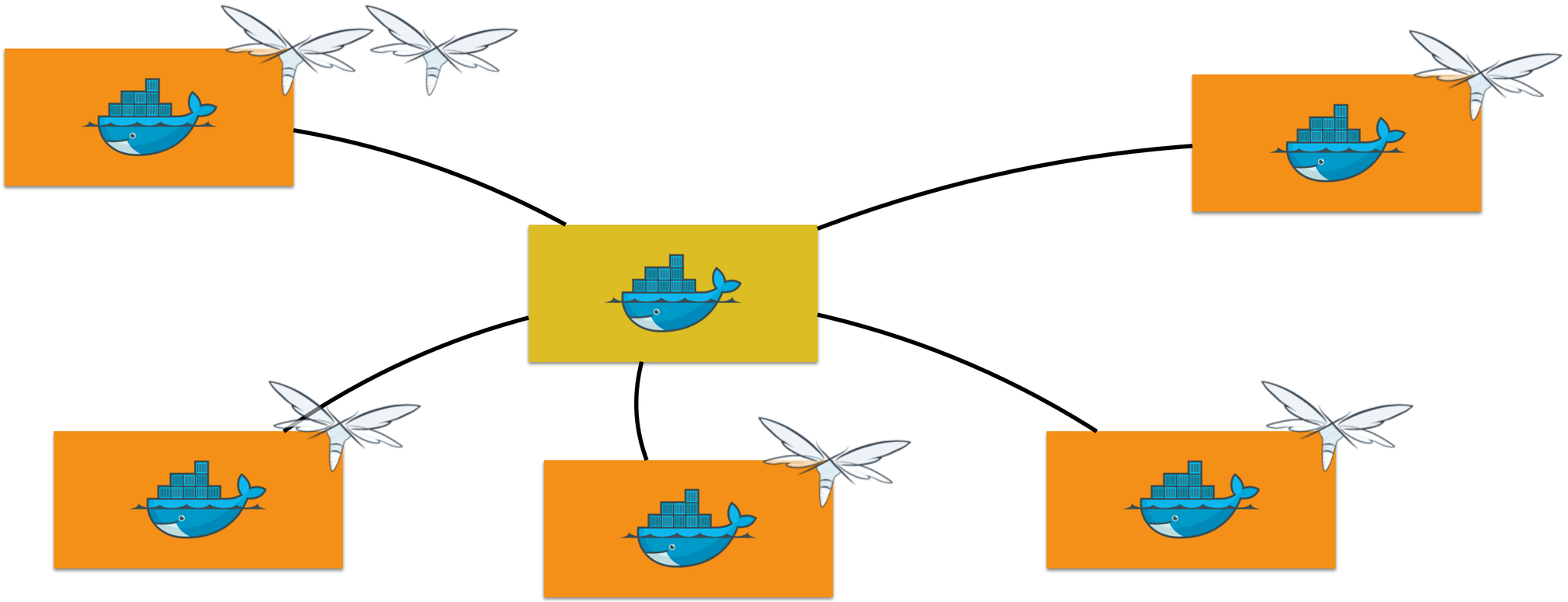
Swarm Mode: Desired != Actual



Swarm Mode: Reconcile

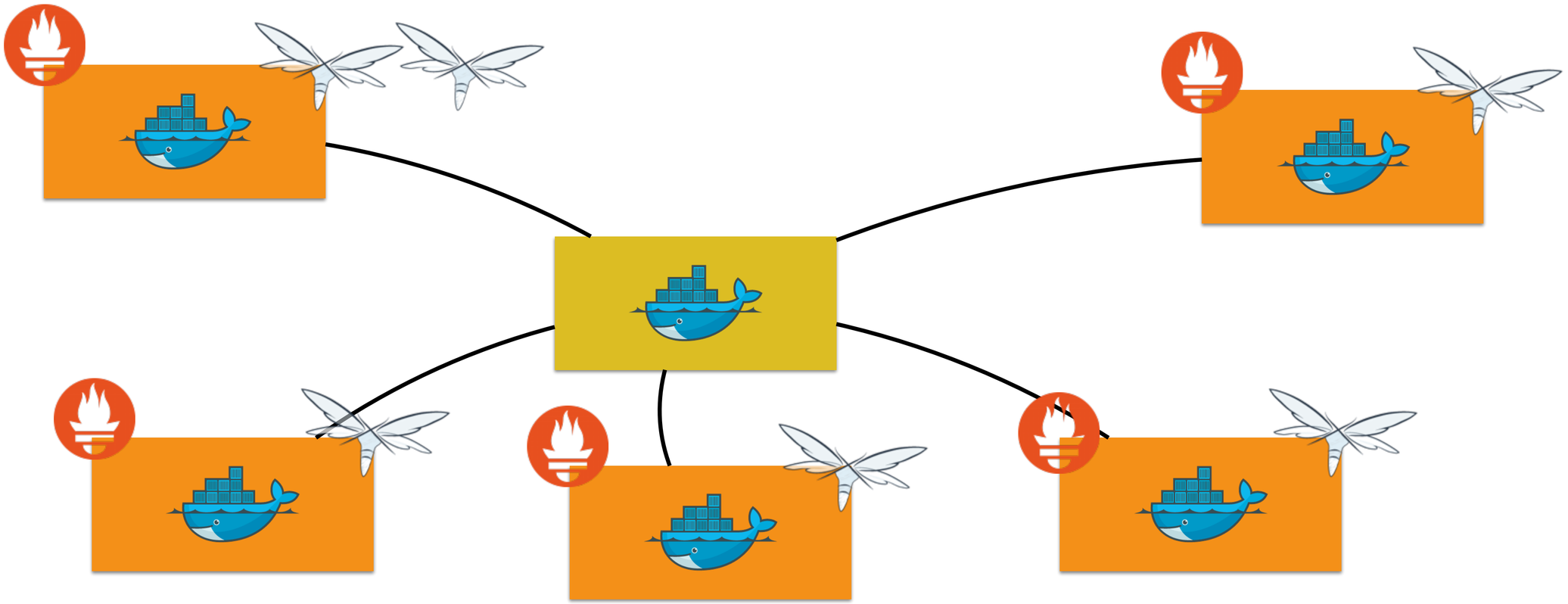


Swarm Mode: Scale



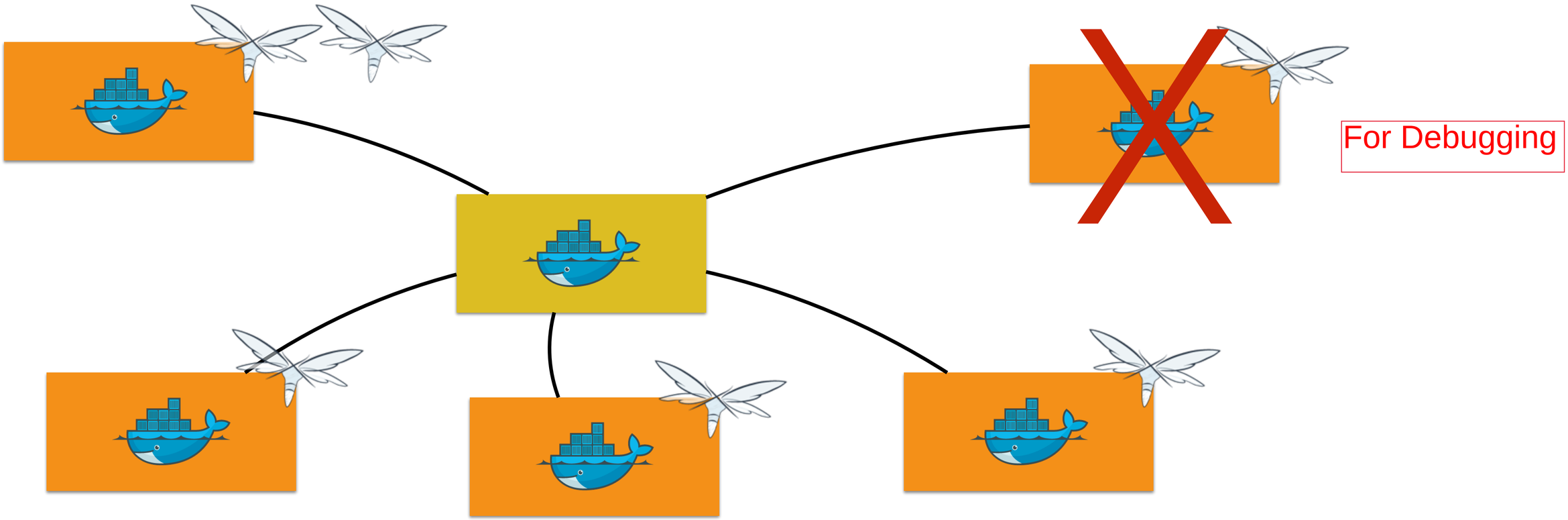
```
docker service scale web=6
```

Swarm Mode: Global Service



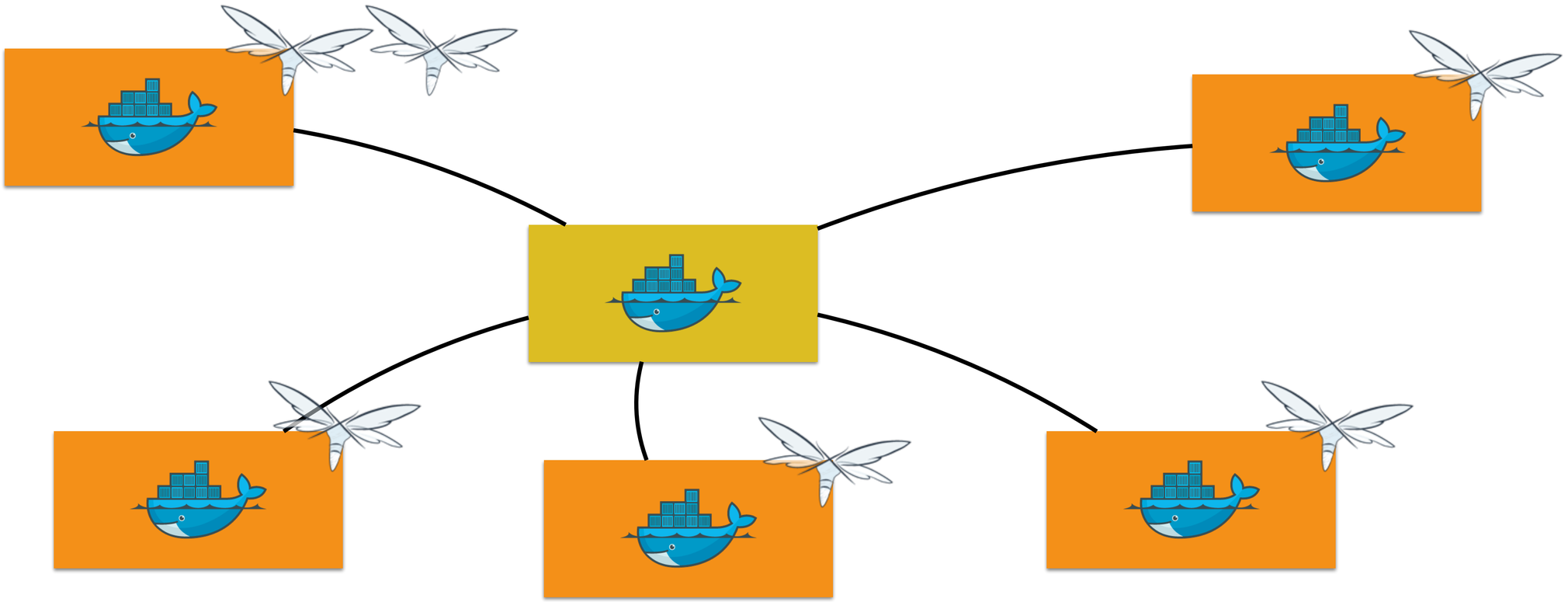
```
docker service create --mode=global --name=prom prom/prometheus
```

Swarm Mode: Pause Node



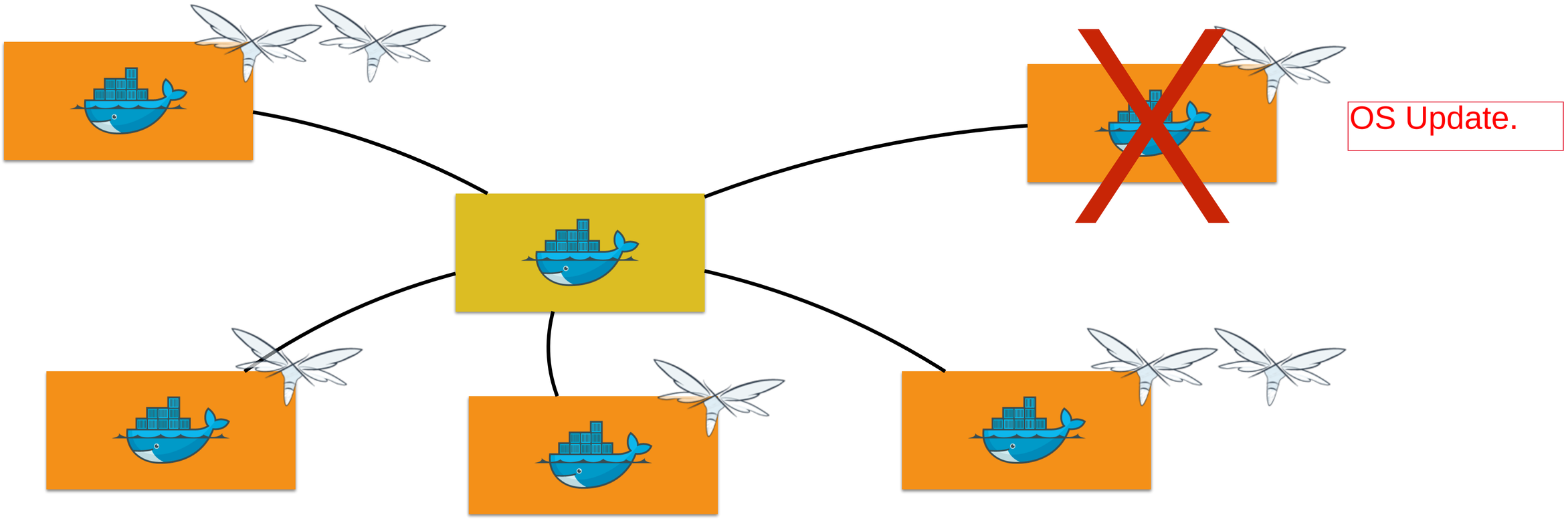

```
docker node update --availability pause <nodename>
```

Swarm Mode: Active Node



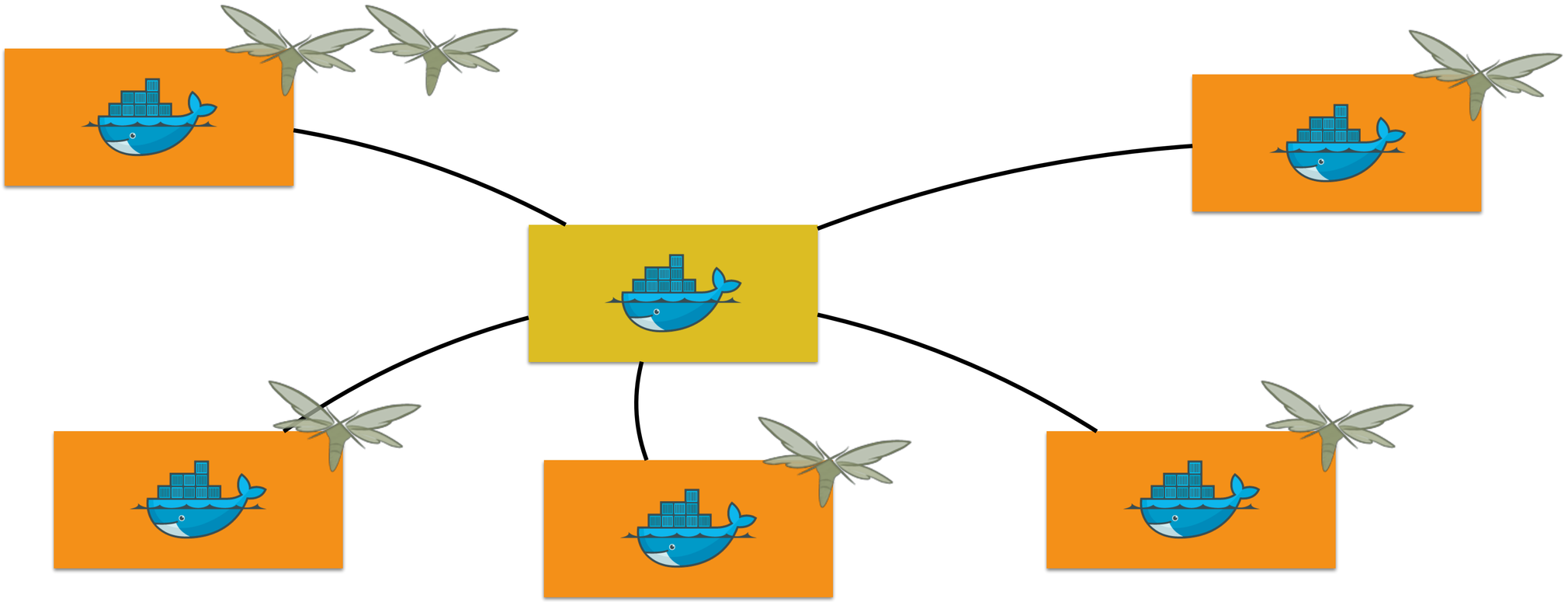
```
docker node update --availability active <nodename>
```

Swarm Mode: Drain Node



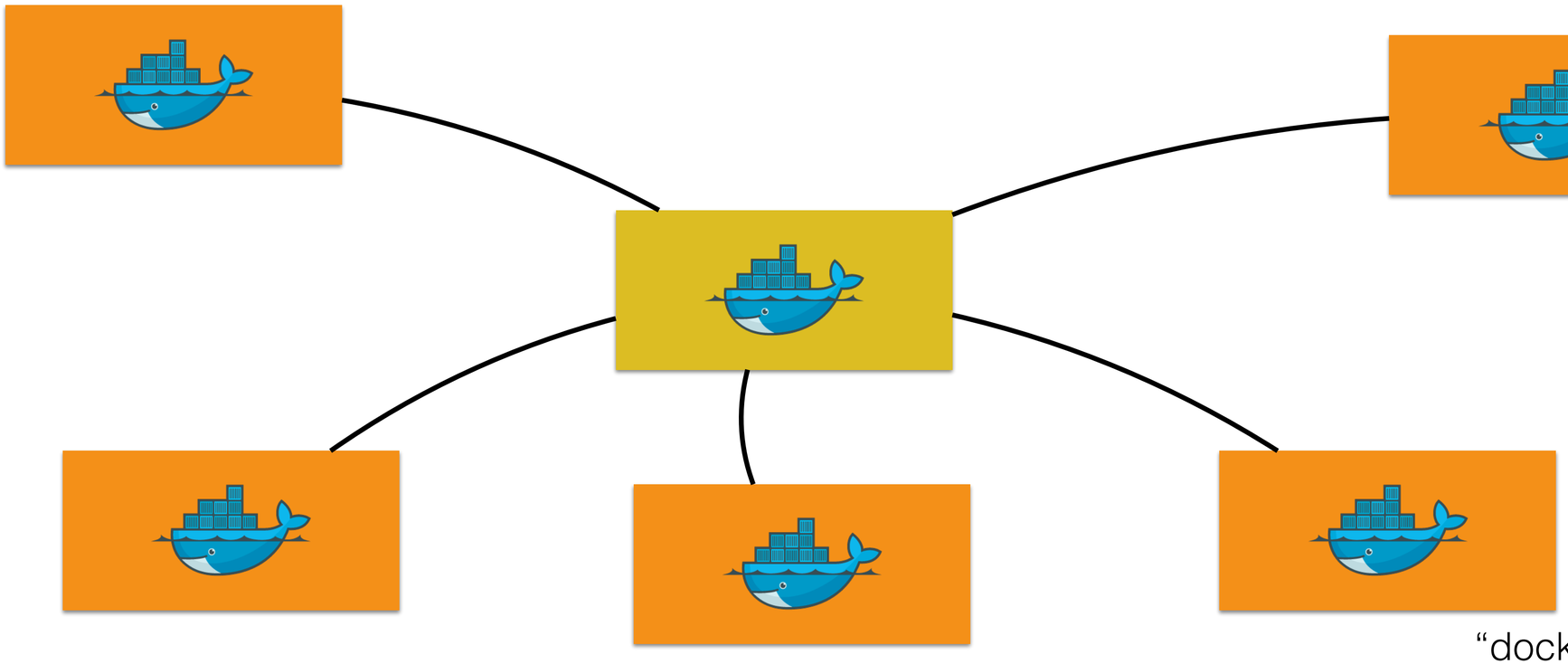

```
docker node update --availability drain <nodename>
```

Swarm Mode: Rolling Updates



```
docker service update web --image wildfly:2 --update-parallelism
2 --update-delay 10s
```

Swarm Mode: Label

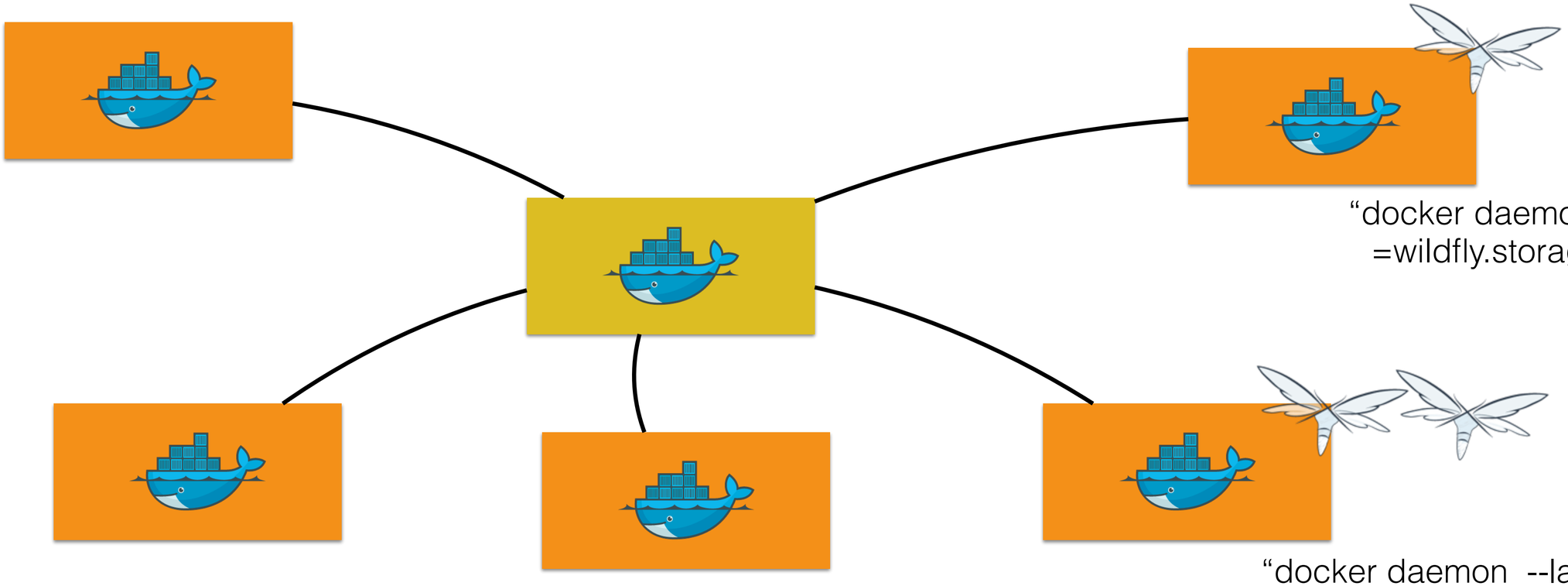


“docker daemon --label =wildfly.storage=ssd”

“docker daemon --label =wildfly.storage=ssd”

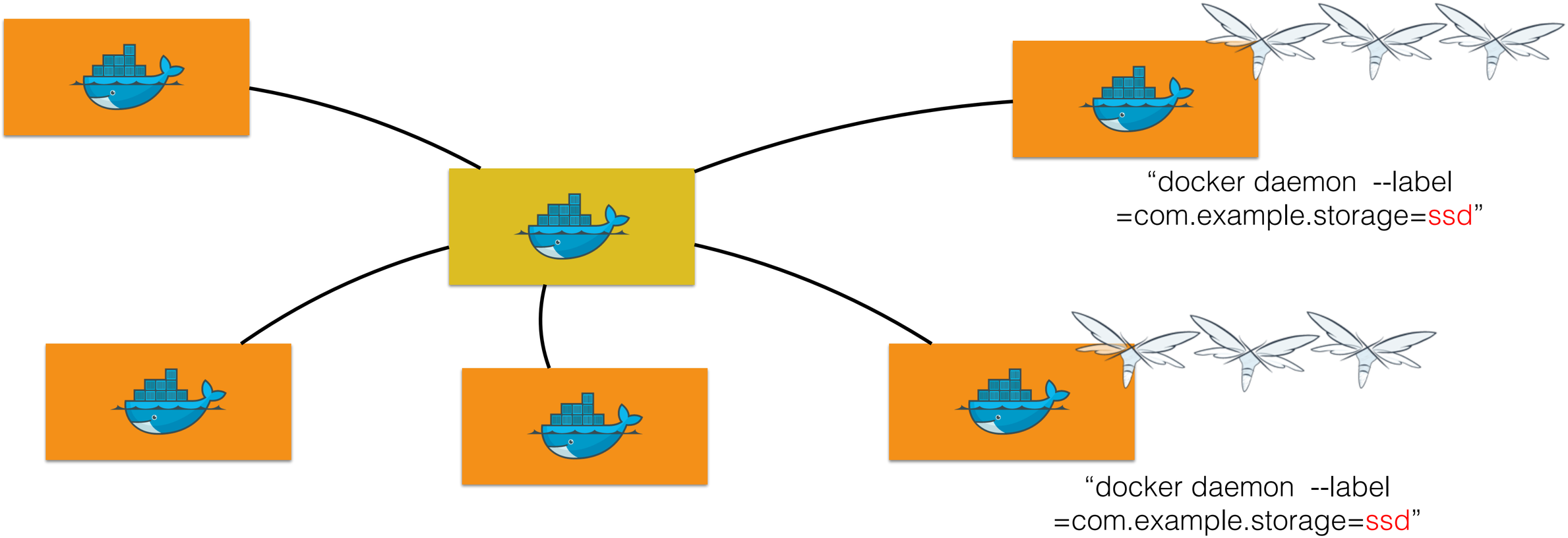
```
DOCKER_OPTS="--label=wildfly.storage=ssd"
```

Swarm Mode: Constraints



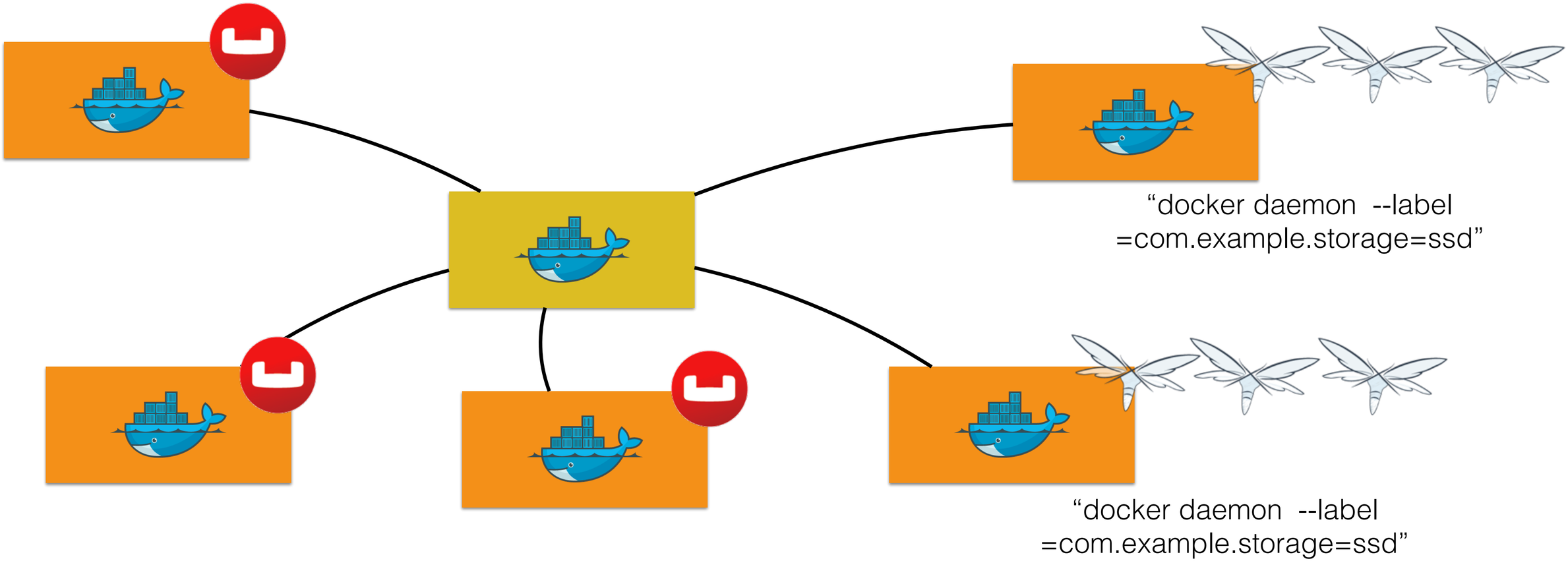

```
docker service create --replicas=3 --name=web --constraint
engine.labels.wildfly.storage==ssd jboss/wildfly
```

Swarm Mode: Constraints




```
docker service scale web=6
```

Swarm Mode: Constraints




```
docker service create --replicas=3 --name=db couchbase
```

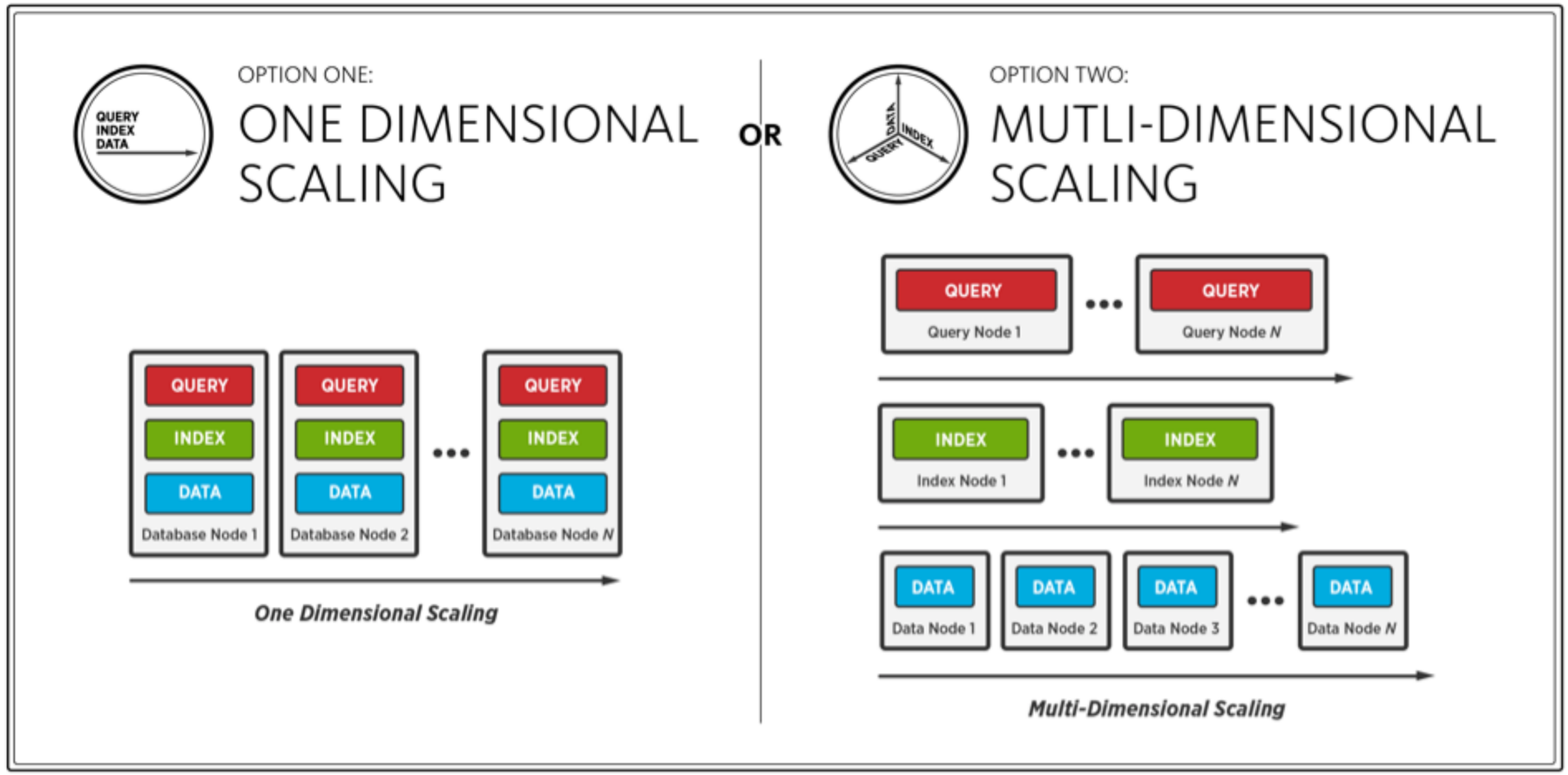
Scheduling Backends using Filters

- **Label**: Metadata attached to Docker Daemon
- **Filters**: Used by Docker Swarm scheduler to create and run container

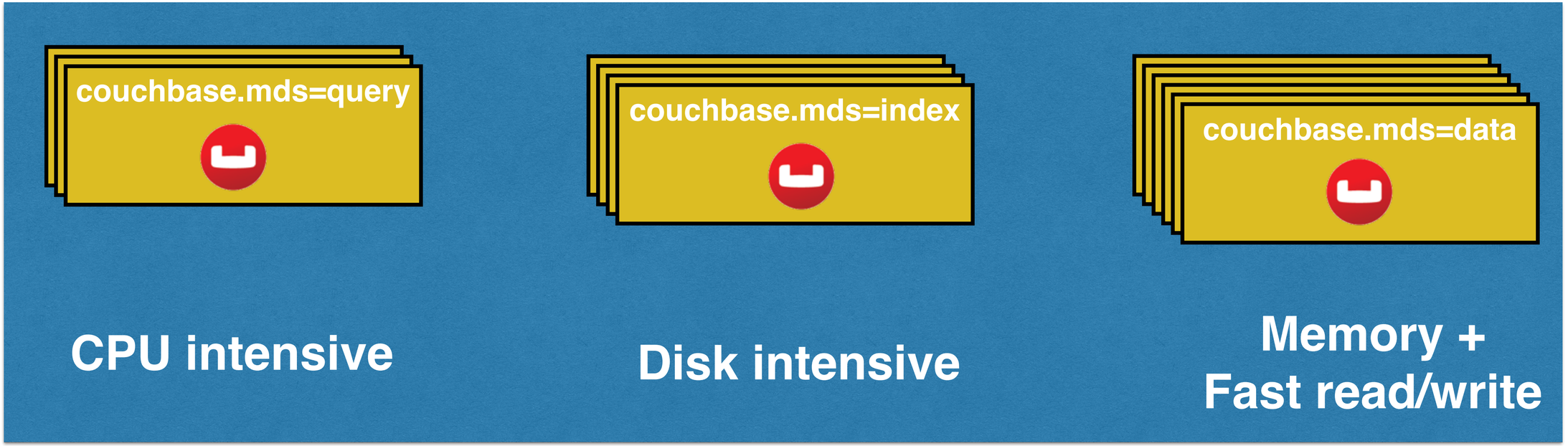
Node		
Constraint	Default or custom tags	`node`, `operatinsystem`, `kernelversion`, …
Health	Schedule containers on healthy nodes only	
Container Slots	Maximum number of containers on a node	`--labels containerslots=3`
Container		
Affinity	“Attraction” between containers	`-e affinity:container=<name>/<id>`, `image`, …
Dependency	Dependent containers on same node	`--volumes-from=<id>`, `--net=container:<id>`, …
Port	Port availability on the host	`-p <host>:<container>`

Couchbase Multi Dimensional Scaling

Only Couchbase gives customers two options for scaling: Standard One-Dimensional Scaling and New Multi-Dimensional Scaling.

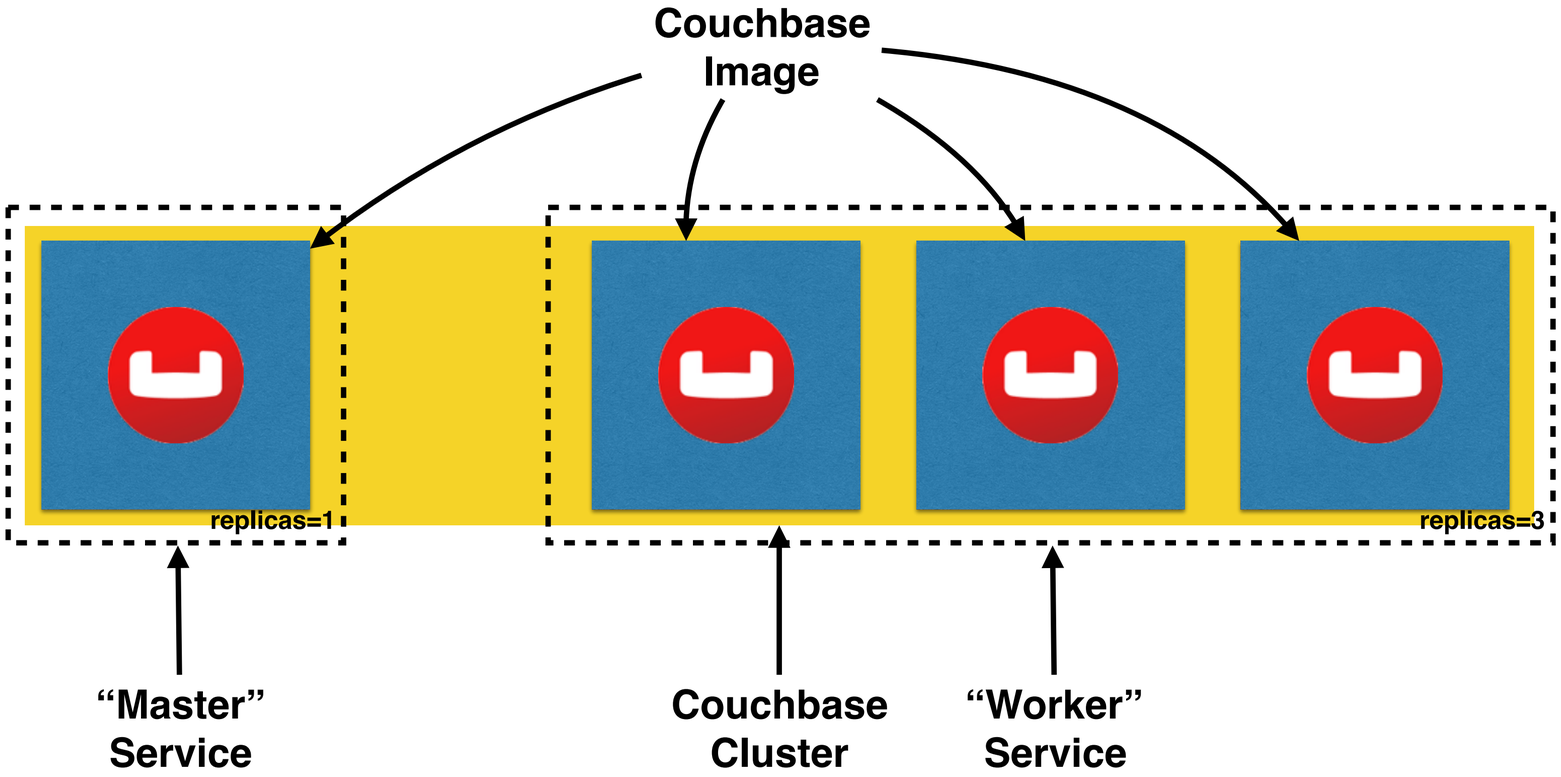


Optimal Utilization of Resources



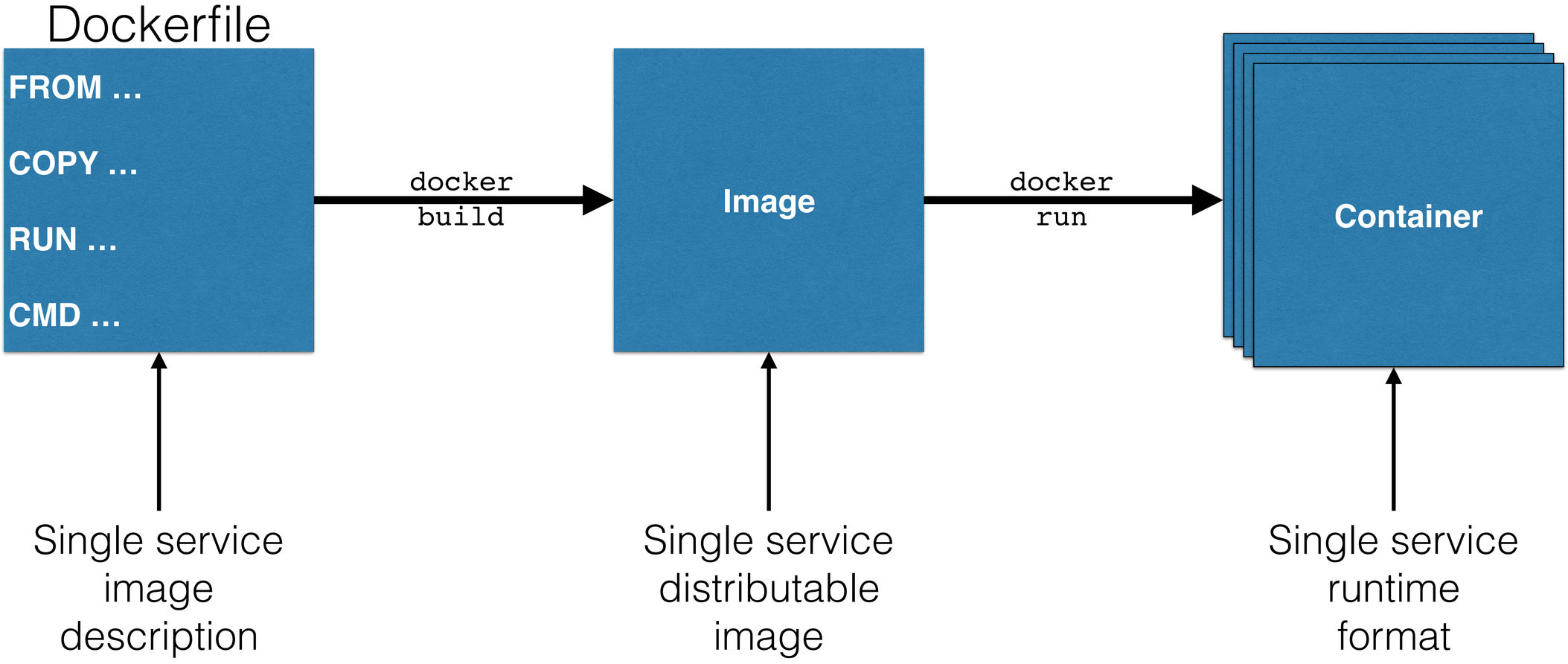
- **Attach labels**: `DOCKER_OPTS="--label.couchbase.mds=data"`
- **Run Containers**: `docker service create --constraint engine.labels.couchbase.mds==index couchbase`

Couchbase Cluster using Docker Services

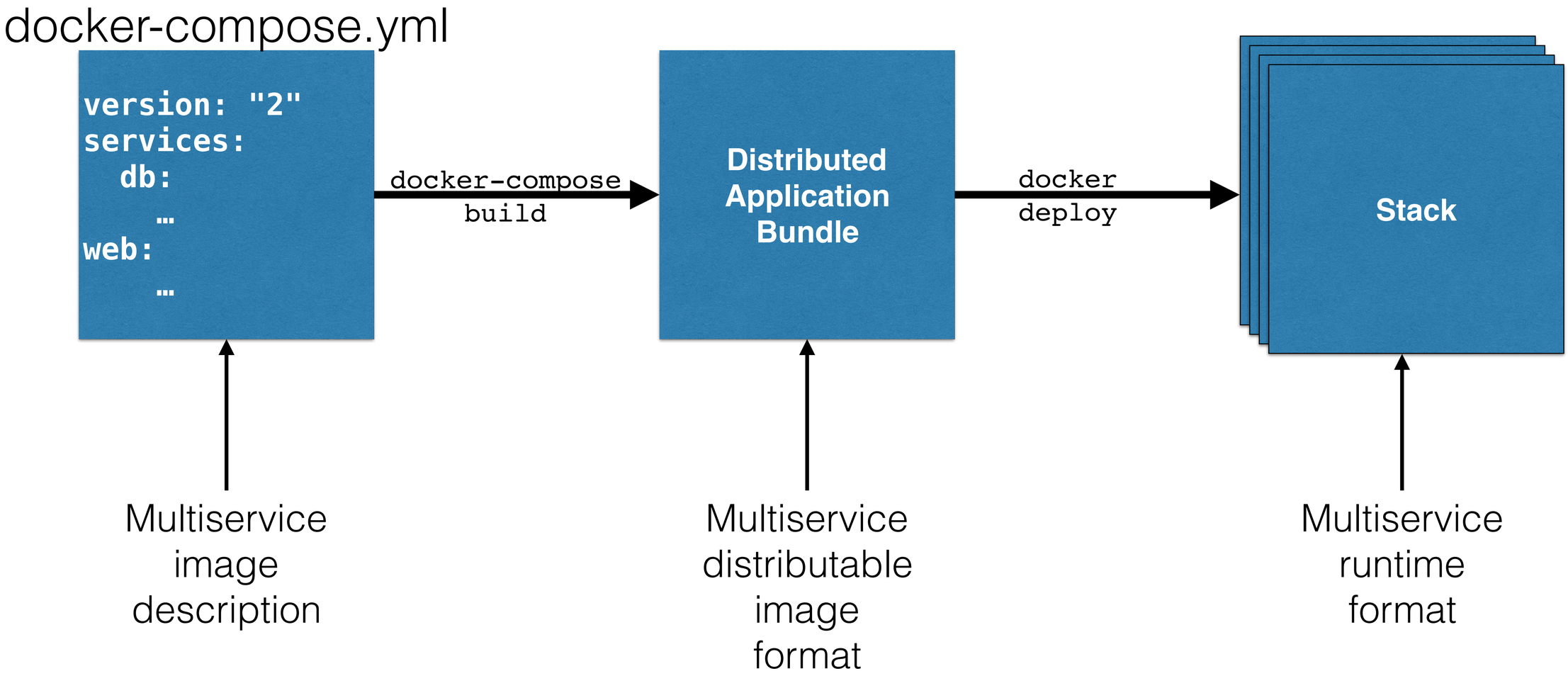


http://blog.couchbase.com/2016/september/docker-service-swarm-mode-couchbase-cluster

Docker Lifecycle



Distributed Application Bundle

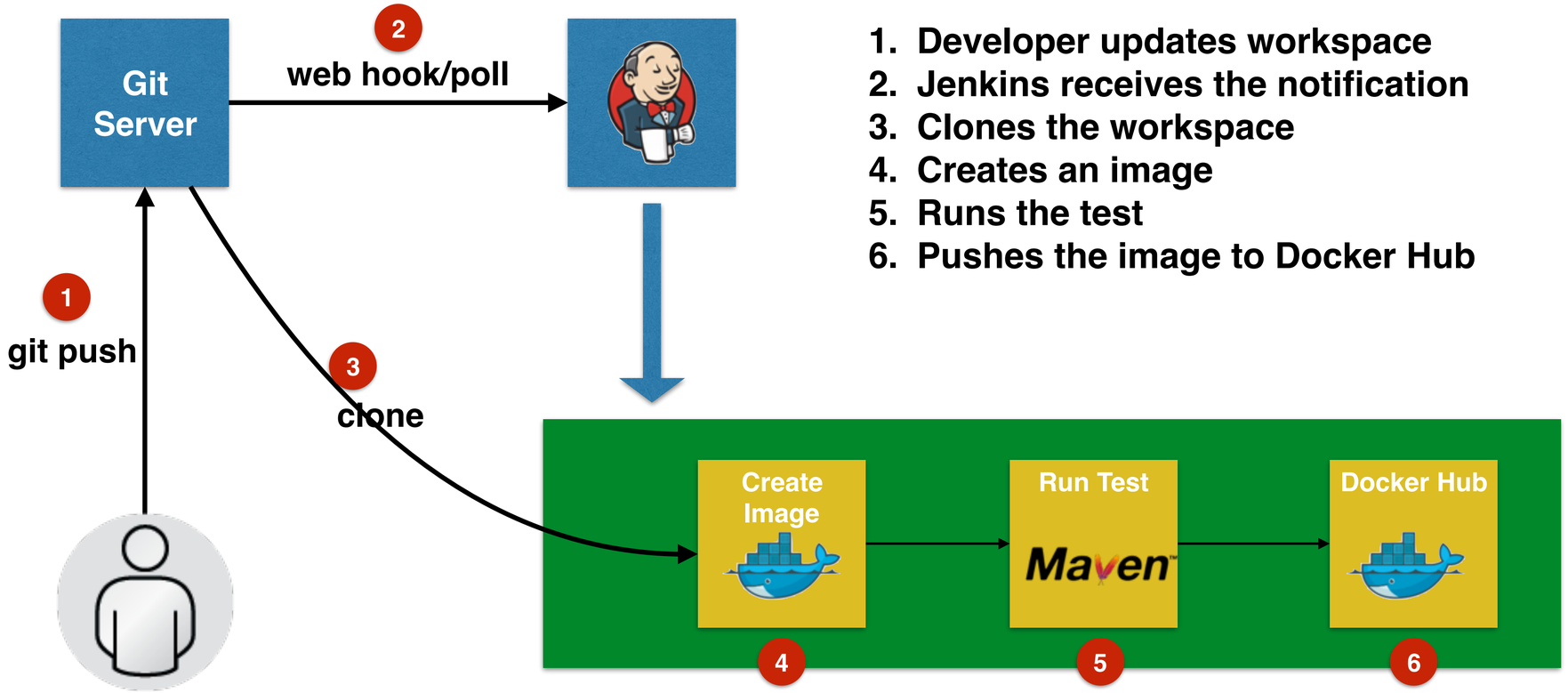


NOTE: Docker Engine and Registry do not support distribution of DABs

Monitoring Docker Containers

- **`docker stats`** command
  - LogEntries
- Service logs: **`docker service logs <service>`**
- Prometheus endpoint - New in Docker 1.13
- Docker Remote API: **`/container/{container-name|cid}/stats`**
- Docker Universal Control Plane
- cAdvisor
  - Prometheus
  - InfluxDB

CI/CD with Docker + Jenkins



1. Developer updates workspace
2. Jenkins receives the notification
3. Clones the workspace
4. Creates an image
5. Runs the test
6. Pushes the image to Docker Hub

http://blog.couchbase.com/2016/september/deployment-pipeline-docker-jenkins-java-couchbase

References

- Slides: github.com/docker/labs/tree/master/slides
- Workshop: github.com/docker/labs/tree/master/java
- Docs: docs.docker.com