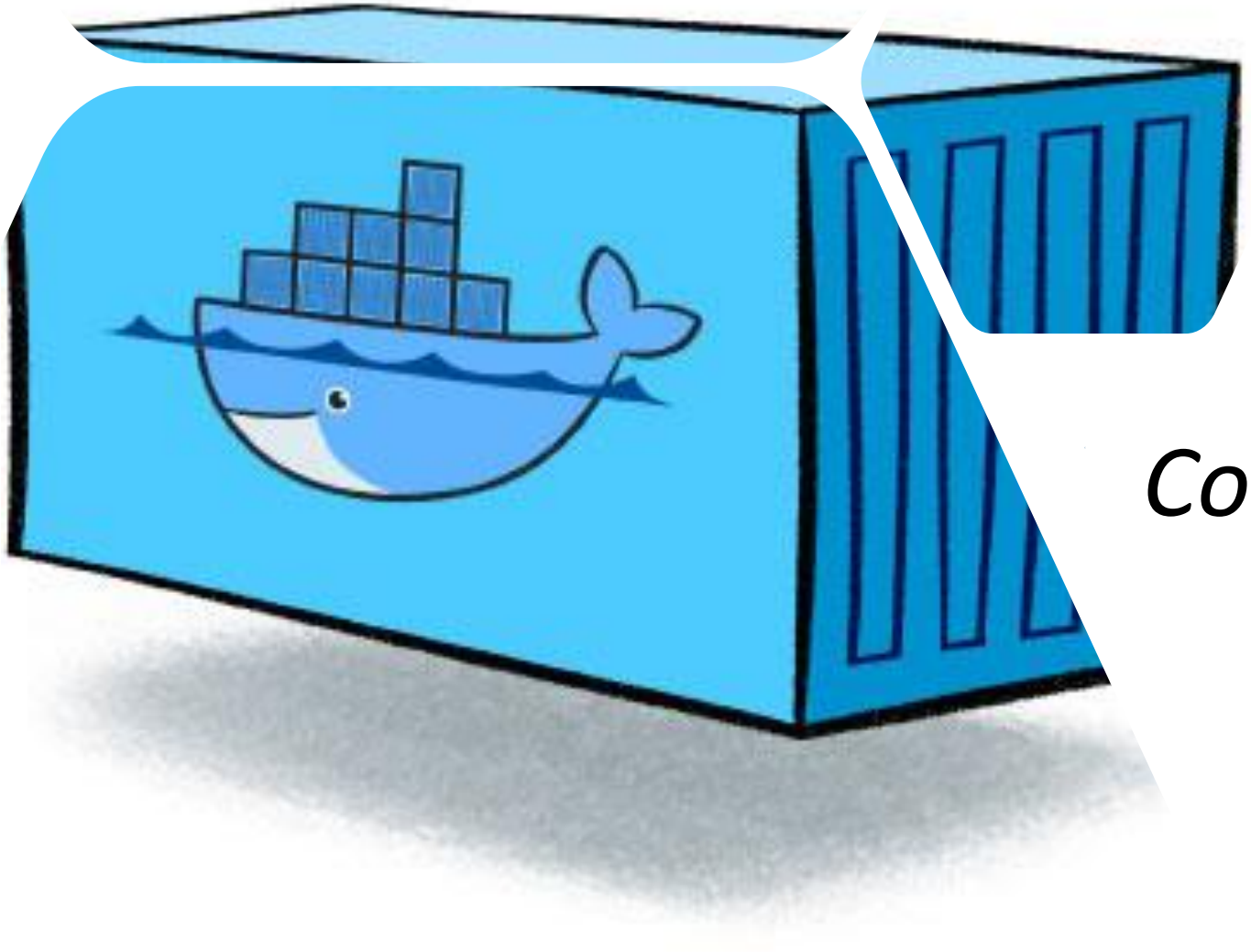


Containerization and Docker

Outlines

Containerization

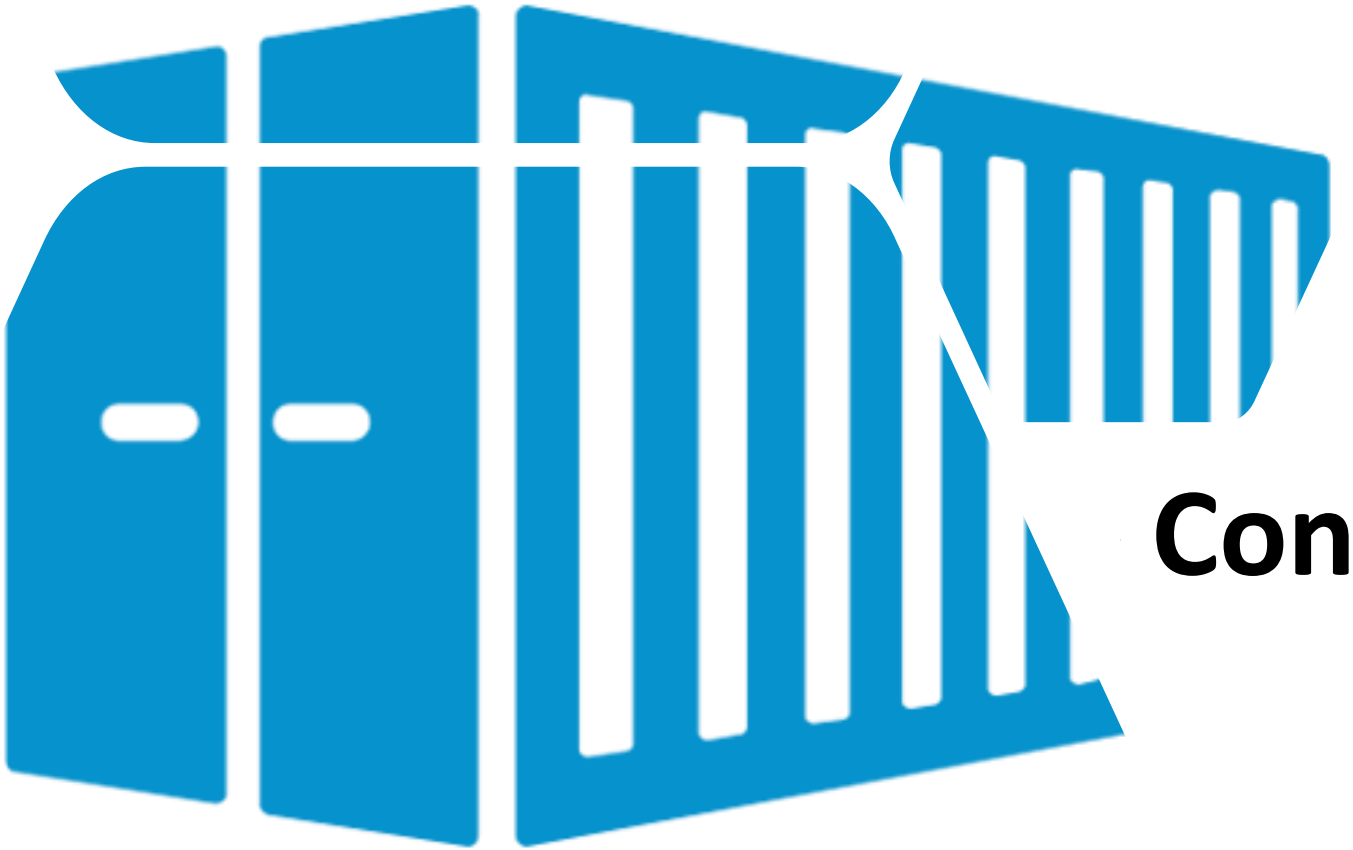
- Limitations of Virtual Machines
- Virtual Machines vs Containers
- Container orchestration tools

Introduction to Docker

- Containers
- Images
- Dockerfile
- Docker Registry
- Networking

Hands-on

- Building an image
- Running a container
- Docker compose and Docker SWARM mode



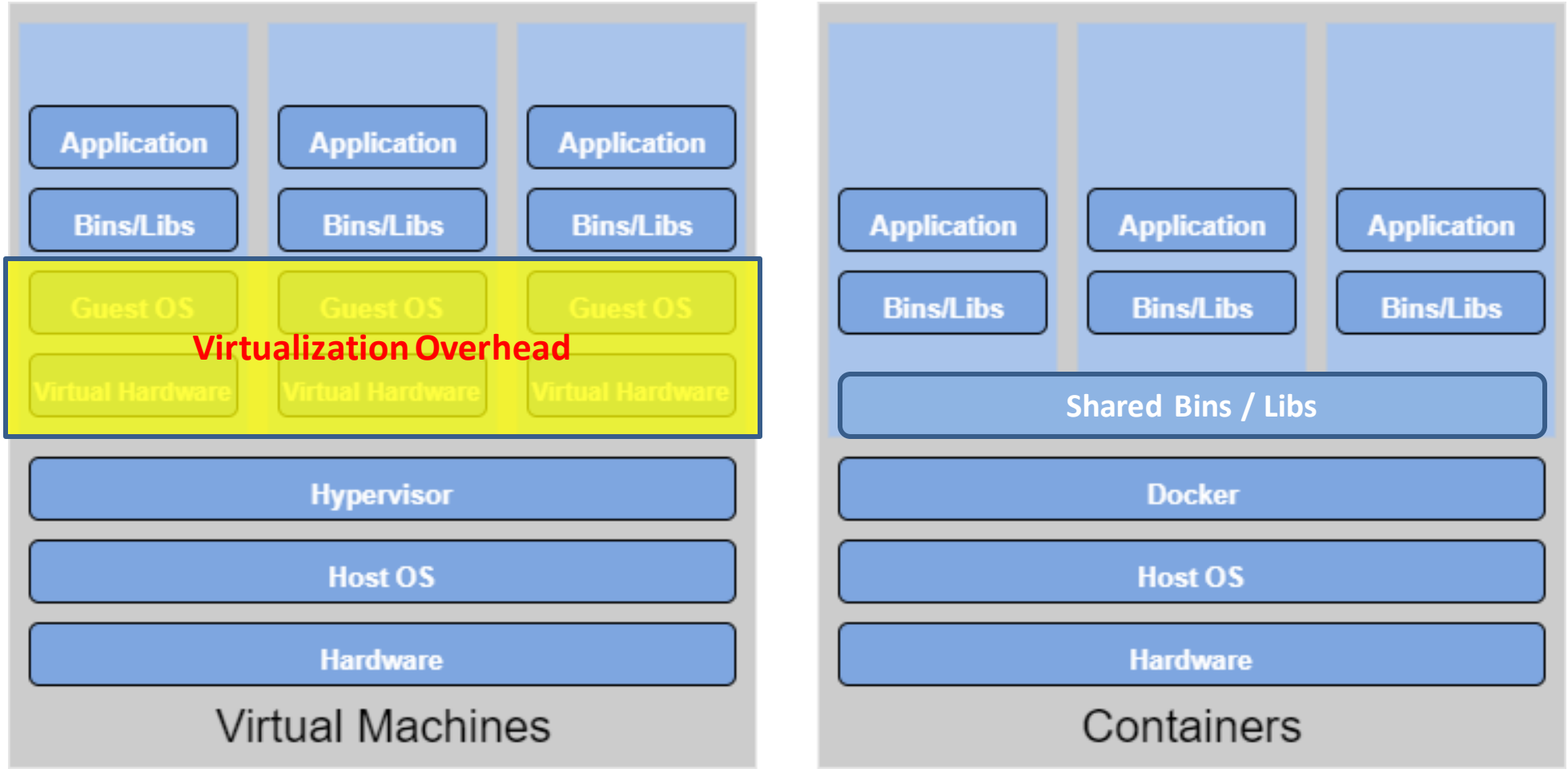
Containerization

Virtualization vs Containerization

- The many advantages of virtualization, such as application containment and horizontal scalability, come at a cost: resources. The guest OS and binaries can give rise to duplications between VMs wasting server processors, memory and disk space and limiting the number of VMs each server can support.

- Containerization allows ***virtual instances*** to share a single host OS (and associated drivers, binaries, libraries) to reduce these wasted resources since each container only holds the application and related binaries. The rest are shared among the containers.

Virtualization vs Containerization

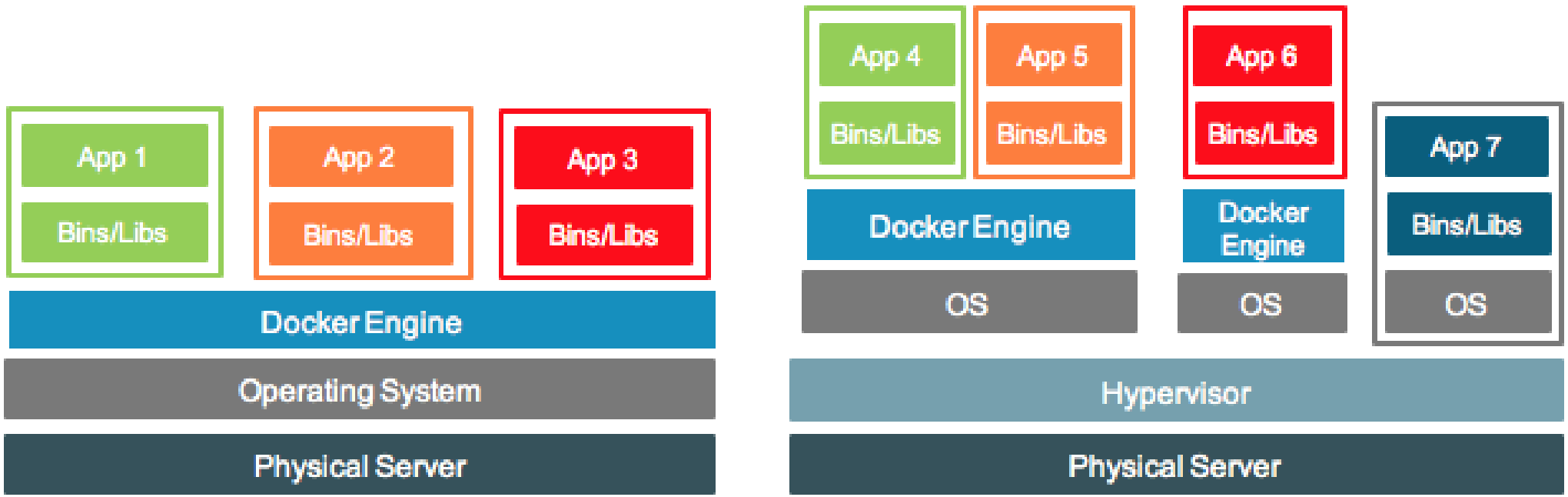


Virtualization vs Containerization

Parameter	Virtual Machines	Containers
Guest OS	Run on virtual HW, have their own OS kernels	Share same OS kernel
Communication	Through Ethernet devices	IPC mechanisms (pipes, sockets)
Security	Depends on the Hypervisor	Requires close scrutiny
Performance	Small overhead incurs when instructions are translated from guest to host OS	Near native performance
Isolation	File systems and libraries are not shared between guest and host OS	File systems can be shared, and libraries are
Startup time	Slow (minutes)	Fast (a few seconds)
Storage	Large size	Small size (most is re-use)

In the Real World, VMs and Containers can/do Co-exist

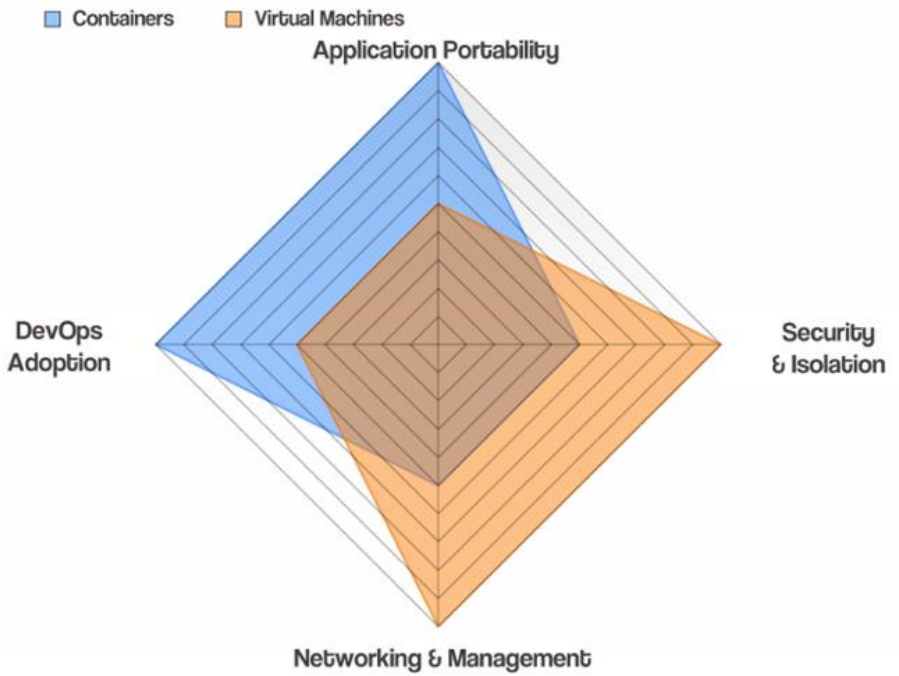
When deploying applications on the cloud, the base computation unit is a Virtual Machine. Usually Docker containers are deployed on top of VMs.



Are Containers Better than VMs?

It depends …

- The size of the task on hand
- The life span of the application
- Security concerns
- Host operation system



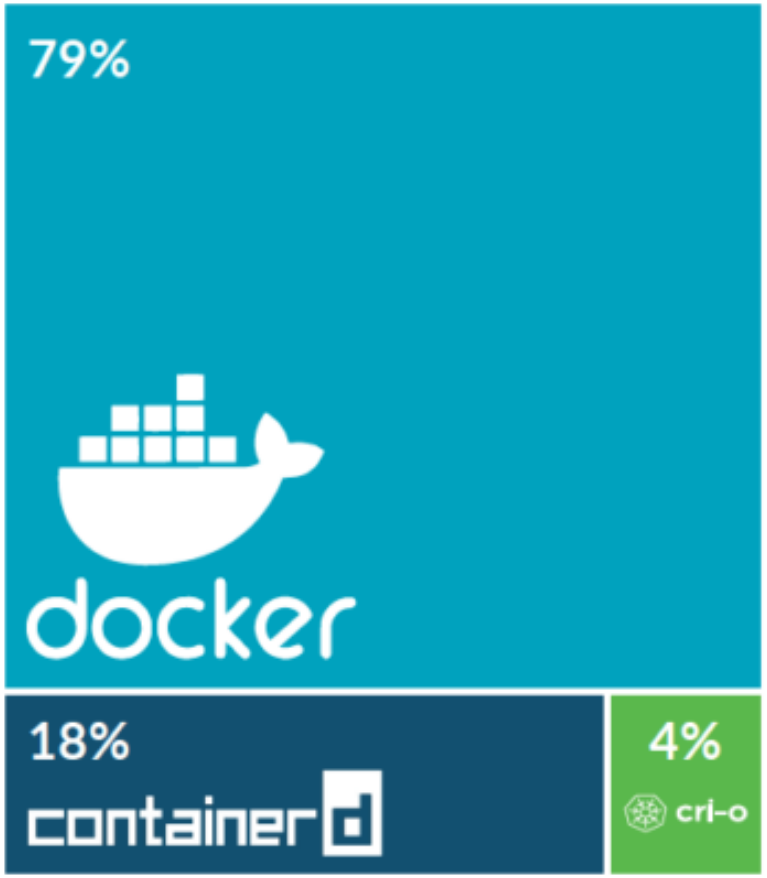
[Containers vs Virtual Machines: Which is a better fit for you?]

[Containers vs Virtual Machines: How to tell which is the right choice for your enterprise?]

What is a Container?

- Similar concept of resource isolation and allocation as a virtual machine.

- Without bundling the entire hardware environment and full OS.



docker

containerd

LXD

rkt

cri-o

What container runtimes are in use?

- Container technologies: Docker, containerd, cri-o, rkt, LxD ...
- **Docker** is currently the leading software container platform

What are Container Orchestration Tools?

Container orchestration technologies provides a framework for integrating and managing containers ***at scale***

Features:

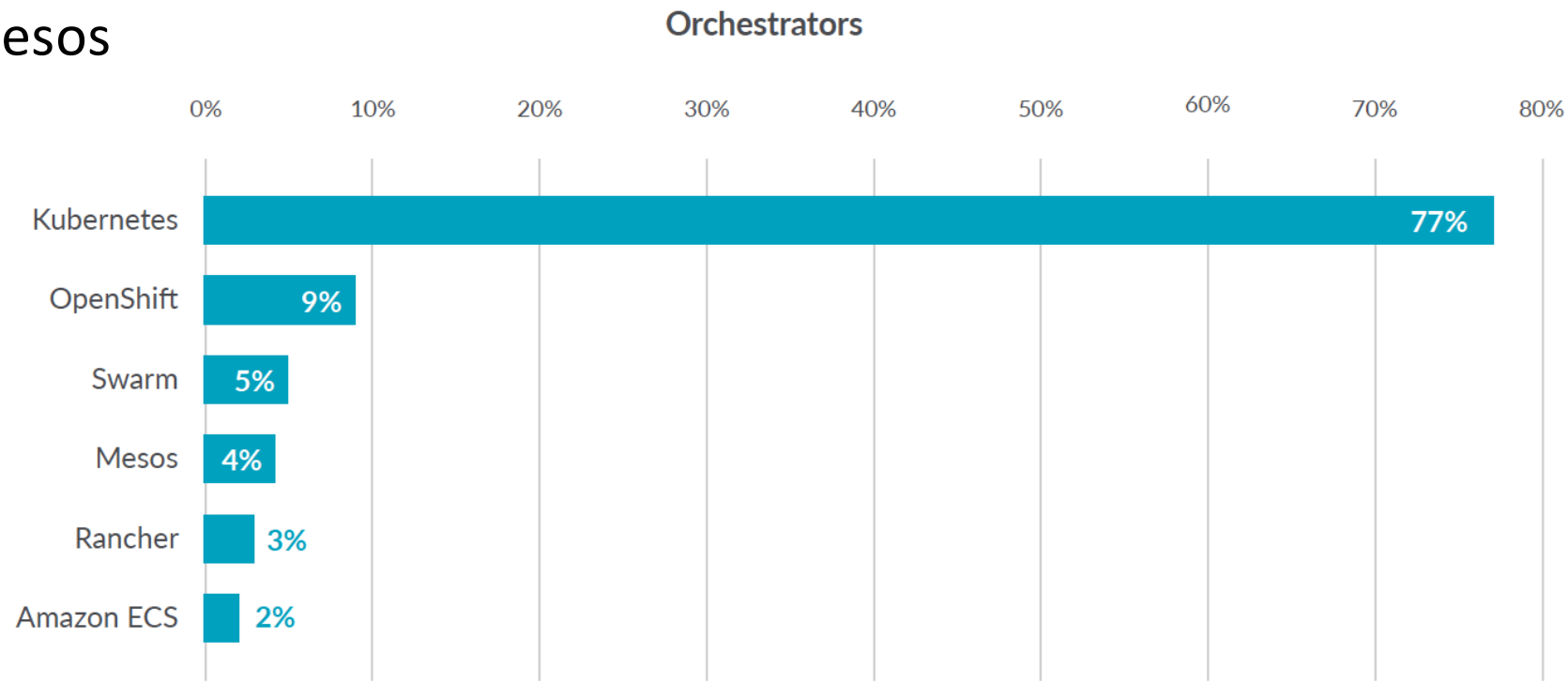
- Networking
- Scaling
- Service discovery and load balancing
- Health check and self-healing
- Security
- Rolling updates

Goals:

- Simplify container management processes
- Help to manage availability and scaling of containers

Container Orchestration Tools

- **Kubernetes and Hosted Kubernetes:**
- **Docker SWARM / Docker Compose**
- **Others:**
  - OpenShift
  - Amazon Elastic Container Service (ECS)
  - Apache Mesos



Introduction to Docker

"Leads the pack with a robust container platform well-suited for the enterprise"

The Forrester New Wave Enterprise Container Platform, Q4 2018 Report

Docker

- Docker is by far the most successful containerization technology.
- Over 3.5 million applications have been placed in Docker containers and over 37 billion containerized applications have been downloaded (March 2018, according to Docker).
- It uses resource isolation features of the Linux kernel to allow independent “containers” to run within a single Linux instance.
- Docker can also be installed on Mac and Windows computers. It it is deeply integrated with the macOS Hypervisor framework and Microsoft Hyper-V virtualization or LCOW.

Docker

- Microsoft have announced native support for Docker.
- Native Windows apps can now be built, shipped and run in Docker containers that work on both Windows Server 2016 and Windows 10 Enterprise.
- Docker versions:
  - Docker CE (Community Edition, free)
  - Docker EE (Enterprise Edition, $$$, USD$750 - USD$3500+ per node per year)

Capabilities	Docker Engine - Community	Docker Engine - Enterprise	Docker Enterprise
Container engine and built in orchestration, networking, security	✓	✓	✓
Certified infrastructure, plugins and ISV containers		✓	✓
Image management			✓
Container app management			✓
Image security scanning			✓

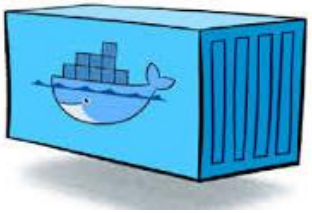
Install Docker

- How to install Docker:
  - Get Docker CE for Linux (Ubuntu) — Share Linux kernel
  - Get Docker CE for Mac
  - Get Docker CE for Windows
  - Get Docker Compose

No more VirtualBox, but LinuxKit VM on macOS, LinuxKit VM / LCOW on Windows

Linux Container on Windows: *https://docs.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/linux-containers*

A Bit of Docker Nomenclature



- **Container**: a process that behaves like an independent machine, it is a runtime instance of a docker image.
- **Image**: a blueprint for a container.

- **Dockerfile**: the recipe to create an image.
- **Registry**: a hosted service containing repositories of images. E.g., the Docker Hub (https://hub.docker.com)
- **Repository:** is a sets of Docker images.

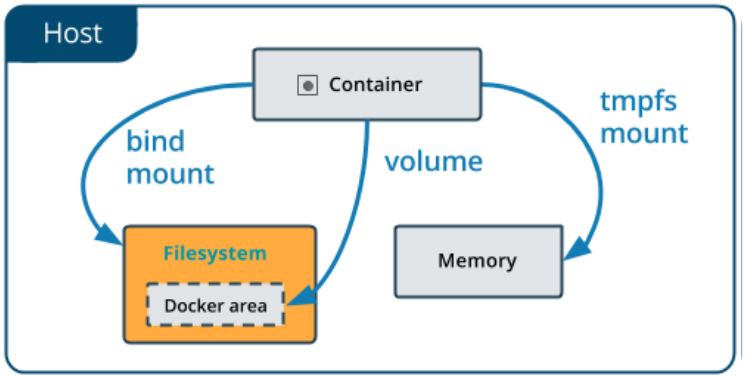
- **Tag:** a label applied to a Docker image in a repository.
- **Docker Compose**: Compose is a tool for defining and running multi-containers Docker applications.
- **Docker SWARM:** a standalone native clustering / orchestration tool for Docker.

Manage Data in Docker

- By default, data inside a Docker container won't be persisted when a container is no longer exist.

- You can copy data in and out of a container.

- Docker has two options for containers to store files on the host machine, so that the files are persisted even after the container stops.
  - Docker volumes (Managed by Docker, /var/lib/docker/volume/)
  - Bind mounts (Managed by user, any where on the file system)



Networking

Docker has different networking options:

- “host”: every container uses the host network stack; which means all containers share the same IP address, hence ports cannot be shared across containers (Linux only, not for macOS or Windows)

```
ubuntu@ubuntu:~$ curl -I localhost:80
curl: (7) Failed to connect to localhost port 80: Connection refused
ubuntu@ubuntu:~$ sudo docker run --name network-test --network host -d nginx
401e9e73d50b11ddaf38dd8745fd854545ab14995322a636dae8a00053a4adab
ubuntu@ubuntu:~$ curl -I localhost:80
HTTP/1.1 200 OK
Server: nginx/1.17.9
```

Nginx container running on “host” network on an Ubuntu Server

Networking

```
alwynpan@ravpn-200-staff-ten-1-4-155 ~ curl -I localhost:80
curl: (7) Failed to connect to localhost port 80: Connection refused
alwynpan@ravpn-200-staff-ten-1-4-155 ~ docker run --name network-test --network host -d nginx
e3ef8ab096af42c01d0555dc36cb598e348466b1ba11ad39740c731a4c8d5ad7
alwynpan@ravpn-200-staff-ten-1-4-155 ~ curl -I localhost:80
curl: (7) Failed to connect to localhost port 80: Connection refused
```

Nginx container running on “host” network on macOS

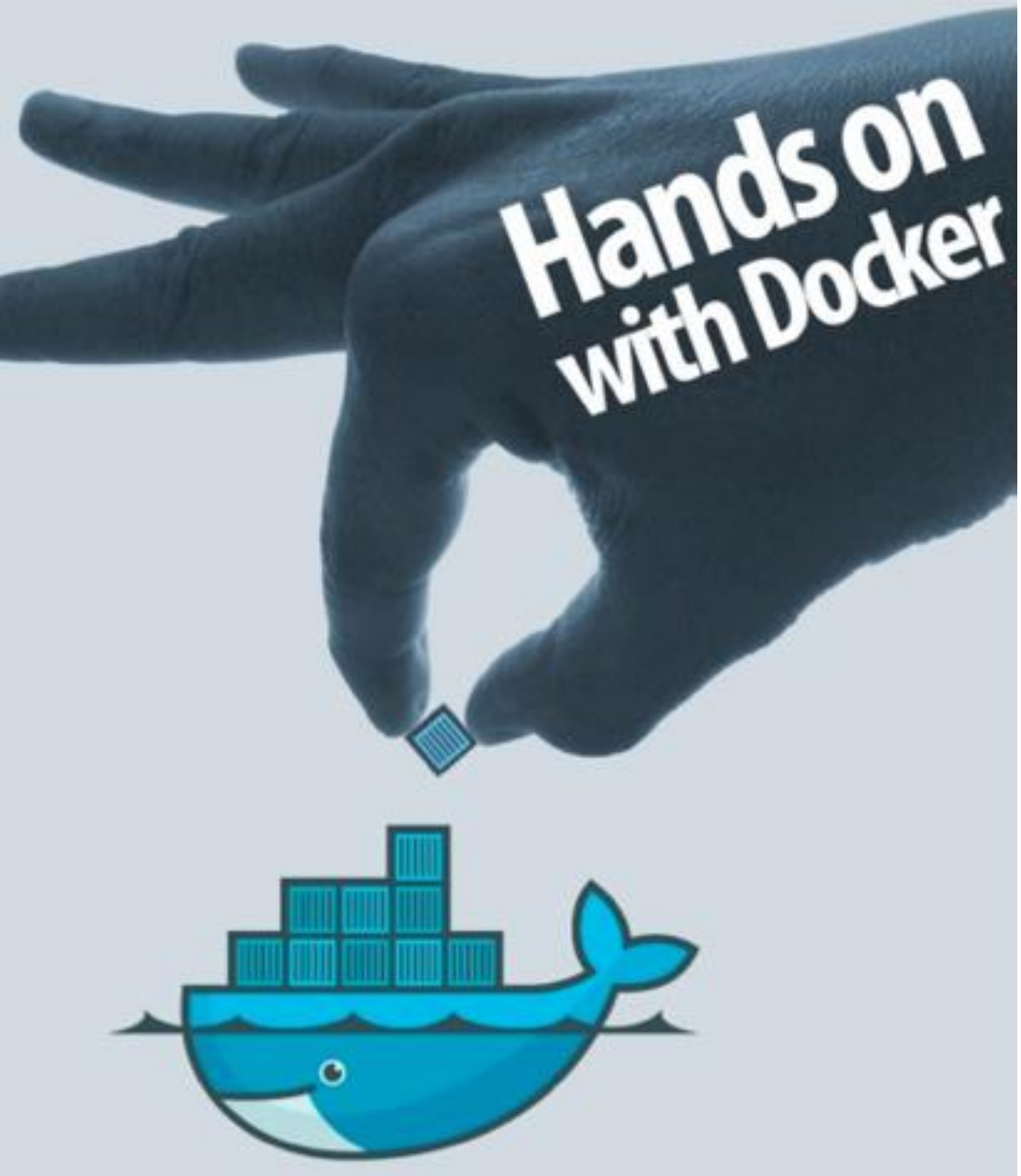
```
C:\WINDOWS\system32> curl -Uri http://localhost:80
curl : Unable to connect to the remote server
At line:1 char:1
+ curl -Uri http://localhost:80
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRe
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Com

C:\WINDOWS\system32> docker run --name network-test --network host -d nginx
40fc5e9d1dd341c4b4fae68475d66439d3a6ac235989b511f25b7c76cf1325e7
C:\WINDOWS\system32> curl -Uri http://localhost:80
curl : Unable to connect to the remote server
At line:1 char:1
+ curl -Uri http://localhost:80
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRe
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Com
```

Nginx container running on “host” network on Windows

Networking

- “bridge”: containers can re-use the same port, as they have different IP addresses, and expose a port of their own that belongs to the hosts, allowing the containers to be somewhat visible from the outside.

Read more: https://docs.docker.com/network/

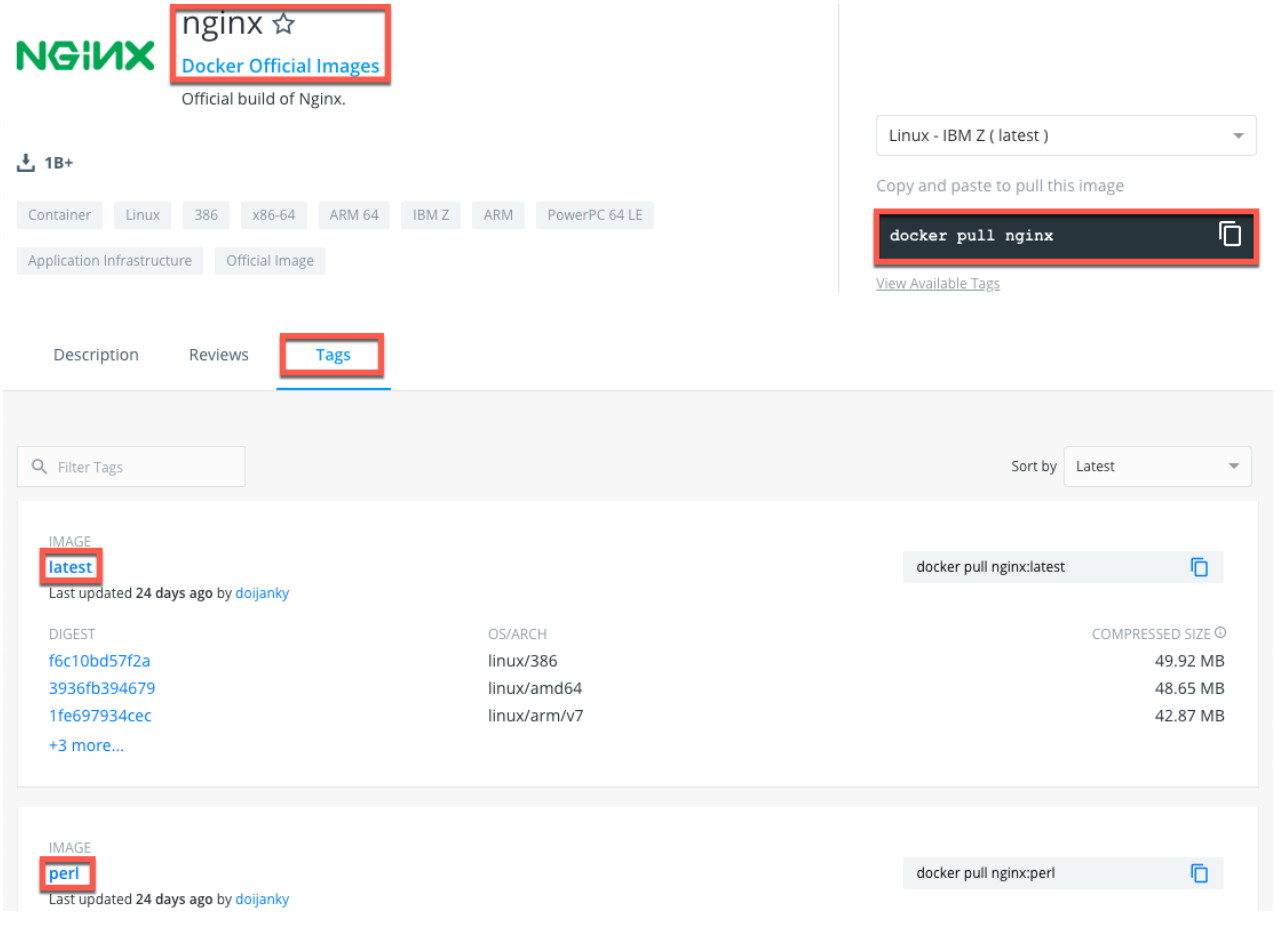


Hands-on

Docker Image

- Public Docker registry (Where you can find the images)
  https://hub.docker.com

- Docker image name
  Syntax: [registry hostname]/repository[:tag]
  E.g.: [docker.io/]nginx:alpine or *nginx[:latest]*



Login to a Docker Registry

- Login to a public Docker Registry, i.e. Docker Hub.

  Syntax: docker login [OPTIONS] [SERVER]

  E.g.: docker login --username=foo

```
alwynpan@Alwyns-MBP ~ docker login -u alwynpan
Password:
Login Succeeded
```

- Login to a private Docker Registry (e.g., AWS ECR, Nexus Server)

  E.g.: docker login -u AWS https://ecr.ap-southeast-2.amazonaws.com

```
alwynpan@Alwyns-MBP ~ docker login -u AWS -p eyJwYXlsb2FkIjoiZC9YQmF5MGNnSXNVekhLejFSUjFpQTJhbnl5Q2Z0LzFUbVVUaXZJN
JzaW9uIjoiMiIsInR5cGUiOiJEQVRBX0tFWSIsImV4cGlyYXRpb24iOjE1NTQ2NjQwMDh9 https://[redacted].dkr.ecr.ap-southeast-2.amaz
onaws.com
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Login Succeeded
```

- Logout

  Syntax: docker logout [SERVER]

  E.g.: docker logout

```
alwynpan@Alwyns-MBP ~ docker logout
Removing login credentials for https://index.docker.io/v1/
```

Pulling a Docker Image

- Pull an image from a public Docker Registry, i.e. Docker Hub.

  Syntax: docker pull NAME[:TAG]

  E.g.: docker pull nginx

```
alwynpan@Alwyns-MBP ~ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
27833a3ba0a5: Pull complete
e83729dd399a: Pull complete
ebc6a67df66d: Pull complete
Digest: sha256:c8a861b8a1eeef6d48955a6c6d5dff8e2580f13ff4d0f549e082e7c82a8617a2
Status: Downloaded newer image for nginx:latest
```

- List all images

  Syntax: docker images [OPTIONS] [REPOSITORY[:TAG]]

  E.g.: docker images

```
alwynpan@Alwyns-MBP ~ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              2bcb04bdb83f        11 days ago         109MB
```

Pushing a Docker Image

- (Optional) Tag an image

  Syntax: docker tag <SOURCE_IMAGE> <TARGET_IMAGE>

  E.g.: docker tag nginx alwynpan/comp90024:nginx

```
alwynpan@Alwyns-MBP ~ docker tag nginx alwynpan/comp90024:nginx
alwynpan@Alwyns-MBP ~ docker images
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
alwynpan/comp90024    nginx               2bcb04bdb83f        11 days ago         109MB
nginx                 latest              2bcb04bdb83f        11 days ago         109MB
```

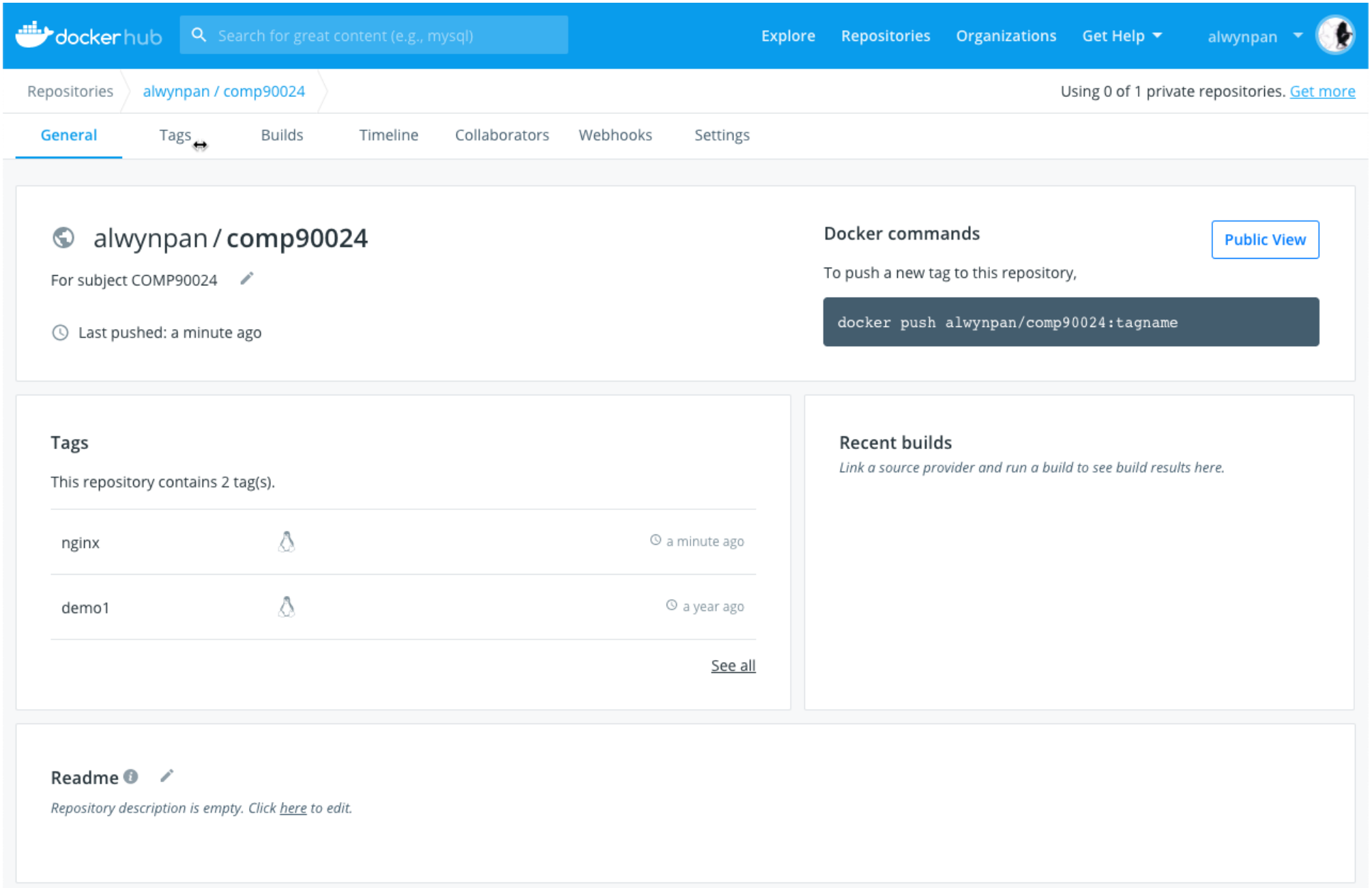
- Push an image

  Syntax: docker push <NAME[:TAG]>

  E.g.: docker push alwynpan/comp90024:nginx

```
alwynpan@Alwyns-MBP ~ docker push alwynpan/comp90024:nginx
The push refers to repository [docker.io/alwynpan/comp90024]
7e274c0effe8: Mounted from library/nginx
dd0338cdfab3: Mounted from library/nginx
5dacd731af1b: Mounted from library/nginx
nginx: digest: sha256:dabecc7dece2fff98fb00add2f0b525b7cd4a2cacddcc27ea4a15a7922ea47ea size: 948
```

Pushing a Docker Image



Run a Docker Container

- (Option 1) Create a container, then start the container
  Syntax: docker create [OPTIONS] IMAGE [COMMAND] [ARG …]
  E.g.: docker create --name nginx -p 8080:80 nginx
  Syntax: docker start [OPTIONS] CONTAINER [CONTAINER …]
  E.g.: docker start nginx

```
alwynpan@Alwyns-MBP ~ docker create --name nginx -p 8080:80 nginx
51eb524eb9b9bc2a2843bf24d396f6ec5a8c07887e3382e75ab75fef0a135bc3
alwynpan@Alwyns-MBP ~ docker start nginx
nginx
```

- (Option 2) Run a container
  Syntax: docker run [OPTIONS] IMAGE [COMMAND] [ARG …]
  E.g.: docker run --name nginx -p 8080:80 -d nginx

```
alwynpan@Alwyns-MBP ~ docker run --name nginx -p 8080:80 -d nginx
54f52f88def92c800cf8194a7557dcbbc6608d1c1365727c3842fe8a688e2d53
```

List Docker Containers

```
alwynpan@Alwyns-MBP ~ docker create --name nginx-created -p 8080:80 nginx
34977c499d0694fc1f9e1de732114cd6845be237c56652f1d5426cb67e1d268e
alwynpan@Alwyns-MBP ~ docker run --name nginx-running -p 8080:80 -d nginx
56aa6c304480dc2674585e27f76a15fdb181518bc027a631d03969c093161543
```

- List running containers

  Syntax: docker ps [OPTIONS]

  E.g.: docker ps

```
alwynpan@Alwyns-MBP ~ docker ps                                    ✓ 10070 17:55:06
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
                    NAMES
56aa6c304480        nginx               "nginx -g 'daemon of…"   3 seconds ago       Up 2 seconds        0.0.0.0:8080-
>80/tcp             nginx-running
```

- List all containers

  docker ps -a

```
alwynpan@Alwyns-MBP ~ docker ps -a                                 ✓ 10071 17:55:08
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
                    NAMES
56aa6c304480        nginx               "nginx -g 'daemon of…"   About a minute ago  Up About a minute   0.0.0.0:8080
->80/tcp            nginx-running
34977c499d06        nginx               "nginx -g 'daemon of…"   About a minute ago  Created
                    nginx-created
```

Stop / Restart / Remove a Docker Container

- Restart a container
  Syntax: docker restart [OPTIONS] CONTAINER [CONTAINER …]
  E.g.: docker restart nginx

- Stop the container
  Syntax: docker stop [OPTIONS] CONTAINER [CONTAINER …]
  E.g.: docker stop nginx

- Remove a non-running container
  Syntax: docker rm [OPTIONS] CONTAINER [CONTAINER …]
  E.g.: docker rm nginx

- Remove a running container
  docker rm -f nginx

Running a Shell within a Container

Containers can be started and stopped, and behave like VMs. Instead of using *SSH* to access a VM, a container can be accessed with *exec* command.

(Not recommended, mainly for debug or testing purpose.)

Syntax: docker exec [OPTIONS] CONTAINER COMMAND [ARG ...]
E.g.: docker exec -ti -w /usr/share/nginx/html/ nginx sh

sed -i 's/nginx!/nginx in Docker!/g' index.html

Demo 1: Manage data in Docker

- Create a volume

  Syntax: docker volume create [OPTIONS] [VOLUME]

  E.g.: docker volume create --name htdocs

- Start a container with a **volume** attached

  *E.g.: docker run --name nginx-volume -p 8080:80 *
  -v htdocs:/usr/share/nginx/html -d nginx

- Start a container with **bind mount** attached

  *E.g.: docker run --name nginx-bind -p 8081:80 *
  -v $(pwd)/htdocs:/usr/share/nginx/html -d nginx

With named volume, the content of the container (index.html etc) was there, but with bind mount the directory was empty. Why?

*A **new** named volume's contents can be pre-populated by a container.*

What's in a Dockerfile

```
FROM nginx:latest

ENV WELCOME_STRING "nginx in Docker"

WORKDIR /usr/share/nginx/html

COPY ["./entrypoint.sh", "/"]

RUN cp index.html index_backup.html; \
    chmod +x /entrypoint.sh; \
    apt-get update && apt-get install -qy vim

ENTRYPOINT ["/entrypoint.sh"]
CMD ["nginx", "-g", "daemon off;"]
```

Run at build time

Run at start up

ENTRYPOINT

- ENTRYPOINT gets executed when the container starts. CMD specifies arguments that will be fed to the ENTRYPOINT.

- Unless it is overridden, ENTRYPOINT will always be executed.

```
#!/usr/bin/env sh

# Replace the substring with the value of the environment variable ${WELCOME_STRING}

sed -i 's/Welcome to nginx!/Welcome to '"${WELCOME_STRING}"'!/g'
/usr/share/nginx/html/index.html

# Make the entrypoint a pass through, then runs the docker command. By default “$@”
variable points to the command line arguments.
exec "$@“
# nginx -g “daemon off”
```

Demo 2: Create an image

- Create an image

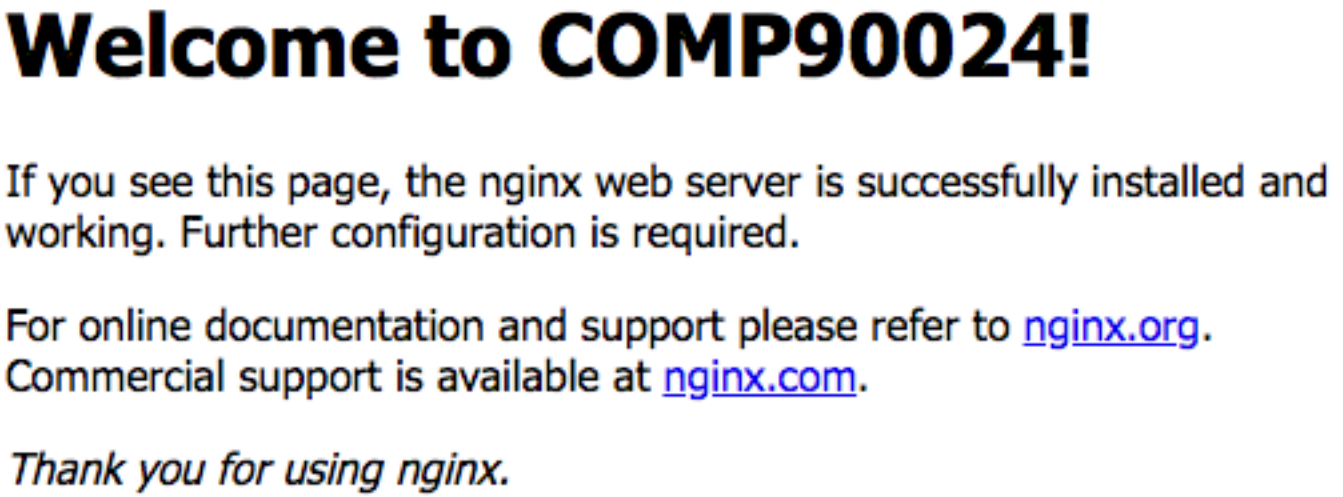
  Syntax: docker build [OPTIONS] PATH

  E.g.: docker build -t demo2 .

- Create a container from the image

  *docker run --name demo2 -e WELCOME_STRING="COMP90024" *

  -p 8081:80 -d demo2



Demo 3: WordPress + MySQL + phpMyAdmin

WordPress is a free and open-source content management system based on PHP and MySQL.

Demo 3: use Compose file to start a WordPress website including WordPress, MySQL and phpMyAdmin.

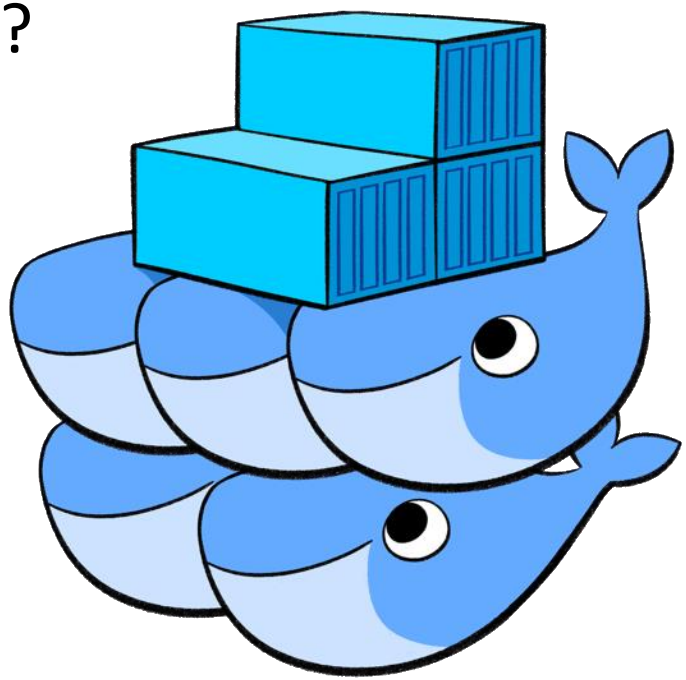
- Start the containers
  Syntax: docker-compose up [OPTIONS]
  E.g.: docker-compose up -d
- Stop the containers
  Syntax: docker-compose stop [OPTIONS] [SERVICE …]
  E.g.: docker-compose stop
- Remove the containers
  Syntax: docker-compose down [OPTIONS]
  E.g.: docker-compose down

Docker SWARM

- What is Docker SWARM (the correct name: Docker in SWARM mode)?

  It is a Docker orchestration tool.

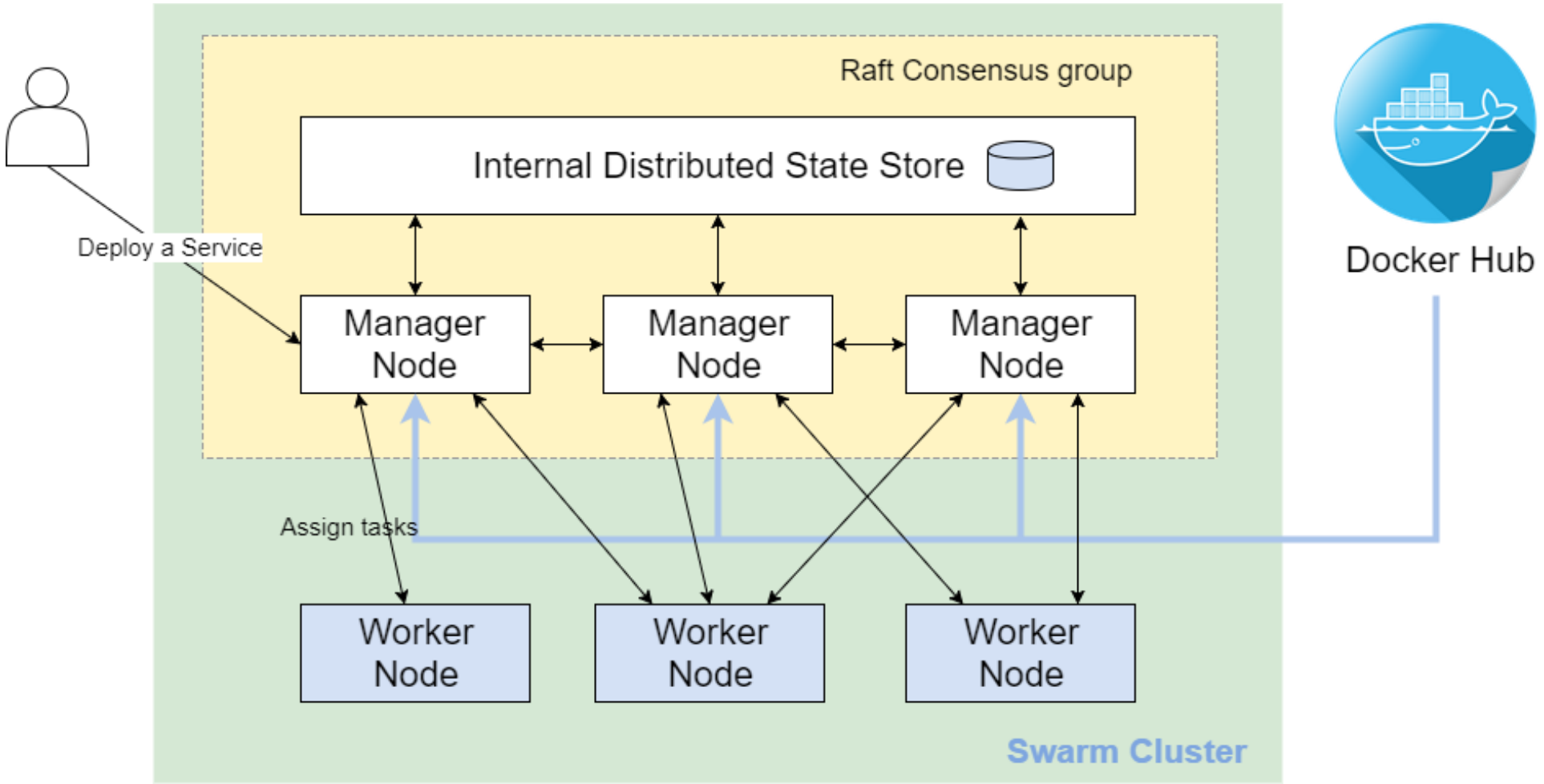
- Why Docker SWARM?
  - Hundreds of containers to manage?
  - Scalability
  - Self-healing
  - Rolling updates
  - And more …



Docker SWARM

- **Raft consensus group** consists of internal distributed state store and all manager nodes.
- **Internal Distributed State Store** is a built-in key-value store of Docker Swarm mode.
- **Manager Node** conducts orchestration and management tasks. Docker Swarm mode allows multiple manager nodes in a cluster. However, only one of the manager nodes can be selected as a leader.
- **Worker Node** receives and executes tasks directly from the manager node
- **Node Availability**: In Docker Swarm mode, all nodes with ACTIVE availability can be assigned new tasks, even the manager node can assign itself new tasks (unless it is in DRAIN mode).
- **Service** consists of one or more replica tasks which are specified by users when first creating the service.
- **Task**: A task in Docker Swarm mode refers to the combination of a single docker container and commands of how it will be run.

Docker SWARM



Demo 4: Docker in SWARM mode

Docker Machine lets you install Docker Engine on virtual hosts, and manage the hosts with docker-machine commands. You can use Docker Machine to create Docker hosts on a local computer, across a network, in a data centre, or on cloud providers.

In this demo we will create one manager and two workers.

Syntax: docker-machine create [OPTIONS] [ARG…]
docker-machine create manager
docker-machine create worker1
docker-machine create worker2

For Windows users:
*docker-machine create --driver hyperv *
--hyperv-virtual-switch <vswitch-name> manager
https://docs.docker.com/machine/drivers/hyper-v/

Demo 4: Create Docker Machines

```
alwynpan@Alwyns-MBP ~/docker-demo/demo3 docker-machine create manager ✓ 10101 21:00:25
Running pre-create checks...
(manager) Default Boot2Docker ISO is out-of-date, downloading the latest release...
(manager) Latest release for github.com/boot2docker/boot2docker is v19.03.5
(manager) Downloading /Users/alwynpan/.docker/machine/cache/boot2docker.iso from https://github.com/boot2docker/boot2d
ocker/releases/download/v19.03.5/boot2docker.iso...
(manager) 0%....10%....20%....30%....40%....50%....60%....70%....80%....90%....100%
Creating machine...
(manager) Copying /Users/alwynpan/.docker/machine/cache/boot2docker.iso to /Users/alwynpan/.docker/machine/machines/ma
nager/boot2docker.iso...
(manager) Creating VirtualBox VM...
(manager) Creating SSH key...
(manager) Starting the VM...
(manager) Check network to re-create if needed...
(manager) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-machine env
 manager
```

Demo 4: Docker Machines

docker-machine ls

```
alwynpan@Alwyns-MBP ~/docker-demo/demo3 docker-machine ls                    ✓ 10105 21:13:41
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER     ERRORS
manager   -        virtualbox   Running   tcp://192.168.99.100:2376           v19.03.5
worker1   -        virtualbox   Running   tcp://192.168.99.101:2376           v19.03.5
worker2   -        virtualbox   Running   tcp://192.168.99.102:2376           v19.03.5
```

docker-machine ssh manager

```
alwynpan@Alwyns-MBP ~/Workspace/Teaching/docker/docker-demo/demo2 docker-machine ssh manager
   ( '>')
  /) TC (\   Core is distributed with ABSOLUTELY NO WARRANTY.
 (/-_--_-\)           www.tinycorelinux.net

docker@manager:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
  NAMES
docker@manager:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
docker@manager:~$
```

Demo 4: Create a Docker SWARM

Syntax: docker swarm init [OPTIONS]

docker swarm init --advertise-addr 192.168.99.100

```
alwynpan@Alwyns-MBP ~/docker-demo/demo3 docker-machine ssh manager docker swarm init --advertise-addr 192.168.99.
100
Swarm initialized: current node (ig4l6zysup56qmxhx0k5olduq) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-62exrqdyql6boahnj8fuq3d2h4psk47ccwt50ihvp9s2rmb9is-7lnuoe6slig9k1vd0smz4mkcb 19
2.168.99.100:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

docker-machine ssh manager docker swarm join-token manager

```
alwynpan@Alwyns-MBP ~/docker-demo/demo3 docker-machine ssh manager docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-62exrqdyql6boahnj8fuq3d2h4psk47ccwt50ihvp9s2rmb9is-6j5zgvpkjroqqfew415xkcpx8 19
2.168.99.100:2377
```

Demo 4: Join a Docker SWARM

Syntax: docker swarm join [OPTIONS]

docker-machine ssh worker1 docker docker swarm join --token SWMTKN-1-2uj8tpltiekyk1e5n4dcugcokjo8a2cfuvgby9s8ru8jwgg6q2-f3xw8888sfw0ayvgrcuv4efam 192.168.99.100:2377

```
alwynpan@Alwyns-MBP ~/docker-demo/demo3 docker-machine ssh worker1 docker swarm join --token SWMTKN-1-0w6v3
fru93tzreuearb1x8hqog4w90htarxxqwxkkuhl-90hr4wi5congw5fca9jm5a4hu 192.168.99.100:2377
This node joined a swarm as a worker.
alwynpan@Alwyns-MBP ~/docker-demo/demo3 docker-machine ssh worker2 docker swarm join --token SWMTKN-1-0w6v3
fru93tzreuearb1x8hqog4w90htarxxqwxkkuhl-90hr4wi5congw5fca9jm5a4hu 192.168.99.100:2377
This node joined a swarm as a worker.
```

docker-machine ssh manager docker node ls

```
alwynpan@Alwyns-MBP ~/docker-demo/demo3 docker-machine ssh manager docker node ls  130 ↵  10131  21:41:35
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE
ERSION
taocdux3z0a7bzpjicwjt3a32 *   manager             Ready               Active              Leader              19.03.5
sbzjarlokr3ibut9oxb9krl29     worker1             Ready               Active                                  19.03.5
sfgmspmanvpormrne1akguq80     worker2             Ready               Active                                  19.03.5
```

Demo 4: Create a Service

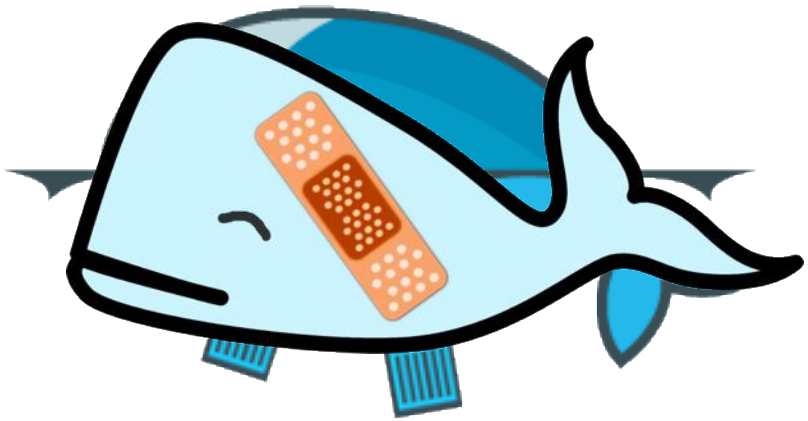
- Create a service
  Syntax: docker service create [OPTIONS] IMAGE [COMMAND]
  docker-machine ssh manager docker service create --replicas 3 -p 8083:80 --name nginx nginx:alpine

- List a service
  Syntax: docker service ls [OPTIONS]
  docker-machine ssh manager docker service ls

- Check a service
  Syntax: docker service ps [OPTIONS] SERVICE [SERVICE …]
  docker-machine ssh manager docker service ps nginx

What if one of the containers stops?

It is self-healing …



Demo 4: Scaling and Rolling Update

- Scale up / down
  Syntax: docker service scale SERVICE=REPLICAS
  docker-machine ssh manager docker service scale nginx=6

  docker-machine ssh manager docker service scale nginx=2

- Rolling update
  Syntax: docker service update [OPTIONS] SERVICE
  docker-machine ssh manager docker service update --image alwynpan/comp90024:demo1 nginx

References

Bigelow, S. J. (n.d.). *How is containerization different from virtualization?* Retrieved from Search Server Virtualization

Docker Inc. (n.d.). Docker. Retrieved from Docker: https://www.docker.com/

Sysdig (29 Oct 2019). 2019 Docker Usage Report: https://sysdig.com/blog/sysdig-2019-container-usage-report/