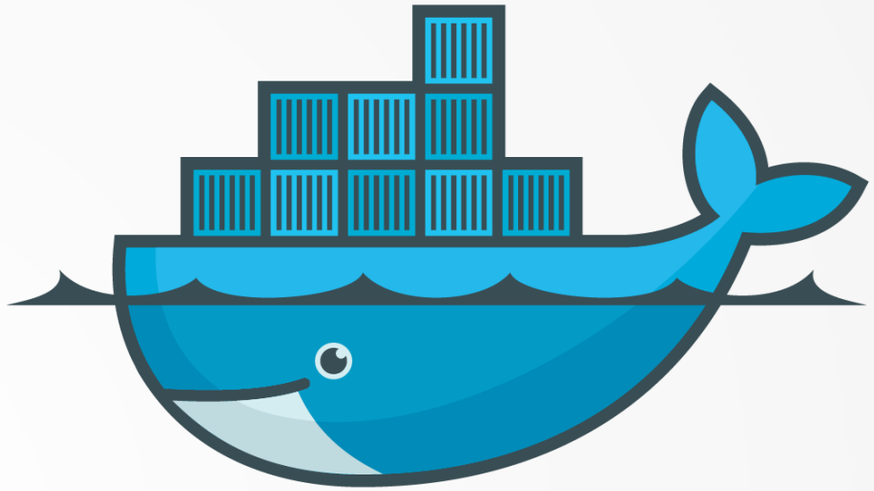


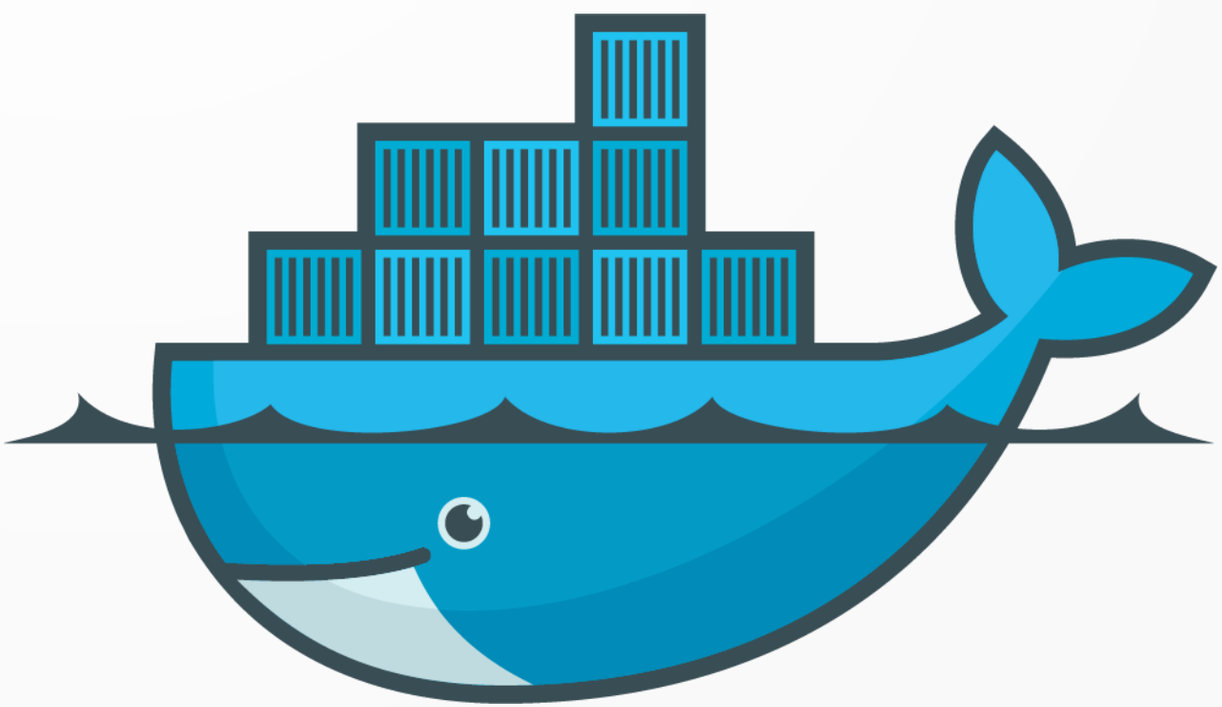
Docker 101: course starter day 1

Porramet Promsang
Cloud Systems Engineer
Digital Transformation Services
Metro Systems Corporation
porrapro@metrosystems.co.th

Course Outline

Day 1

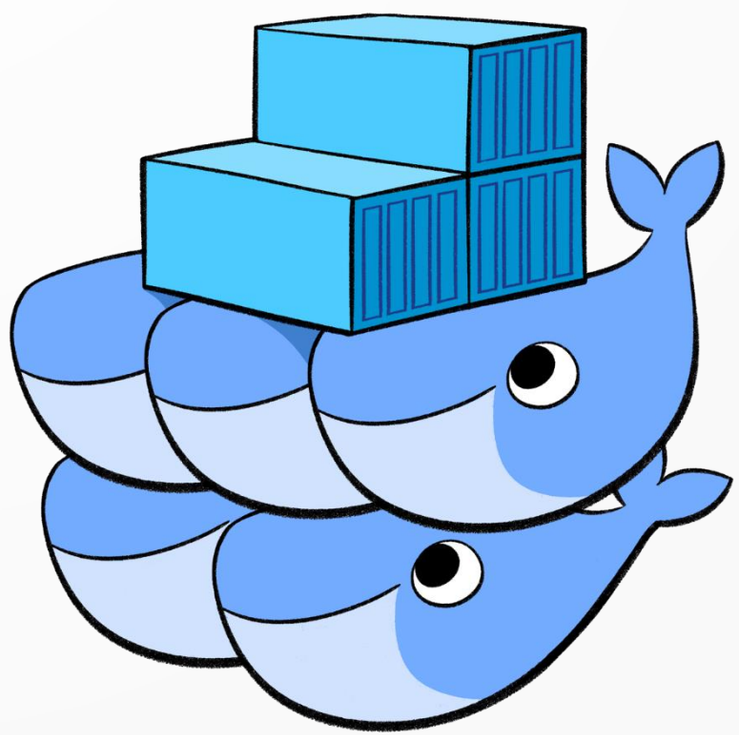
- Technology Overview
- Image
- Registry
- Container
- Volume
- Network
- Log Inspect and Stats



Course Outline

Day 2

- Dockerfile
- Docker Compose
- Docker Swarm
- Portainer



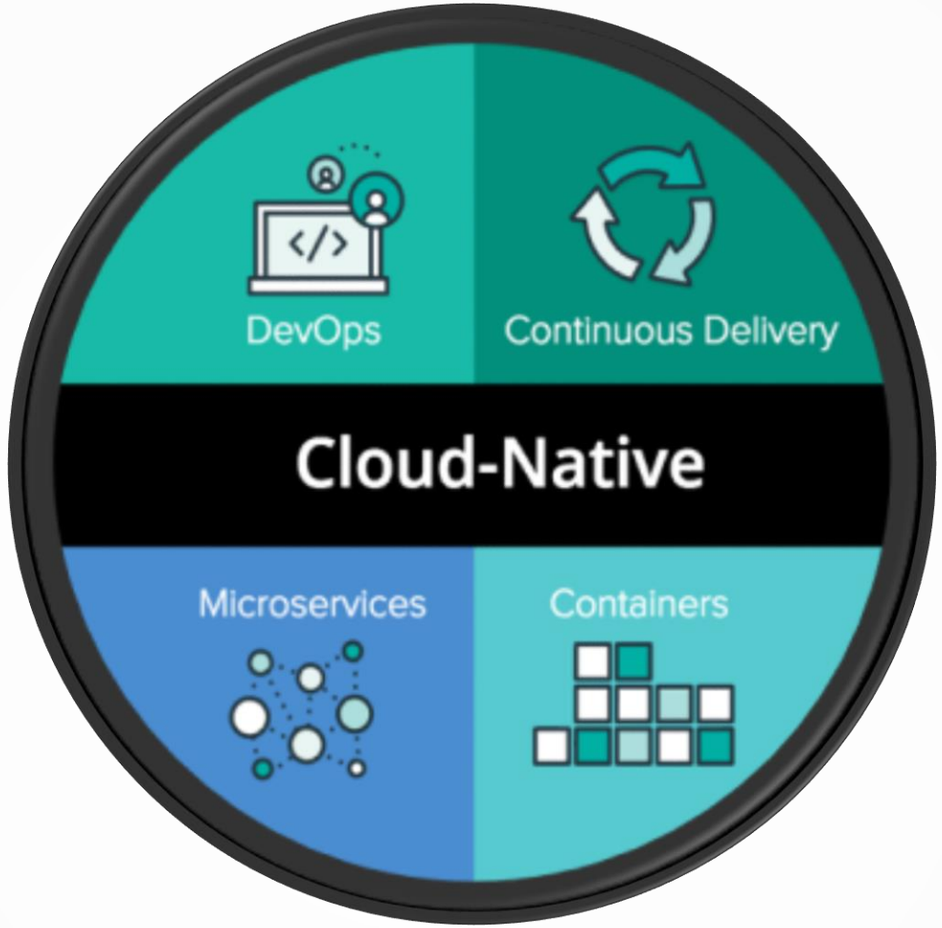
Container Overview

What is Contrainer?

- A standard way to package an application and all its dependencies so that it can be moved between environments and run without change
- Work by hiding the differences between applications inside the container so that everything outside the container can be standardized
- Docker: provides a standard way to create images for Linux Containers

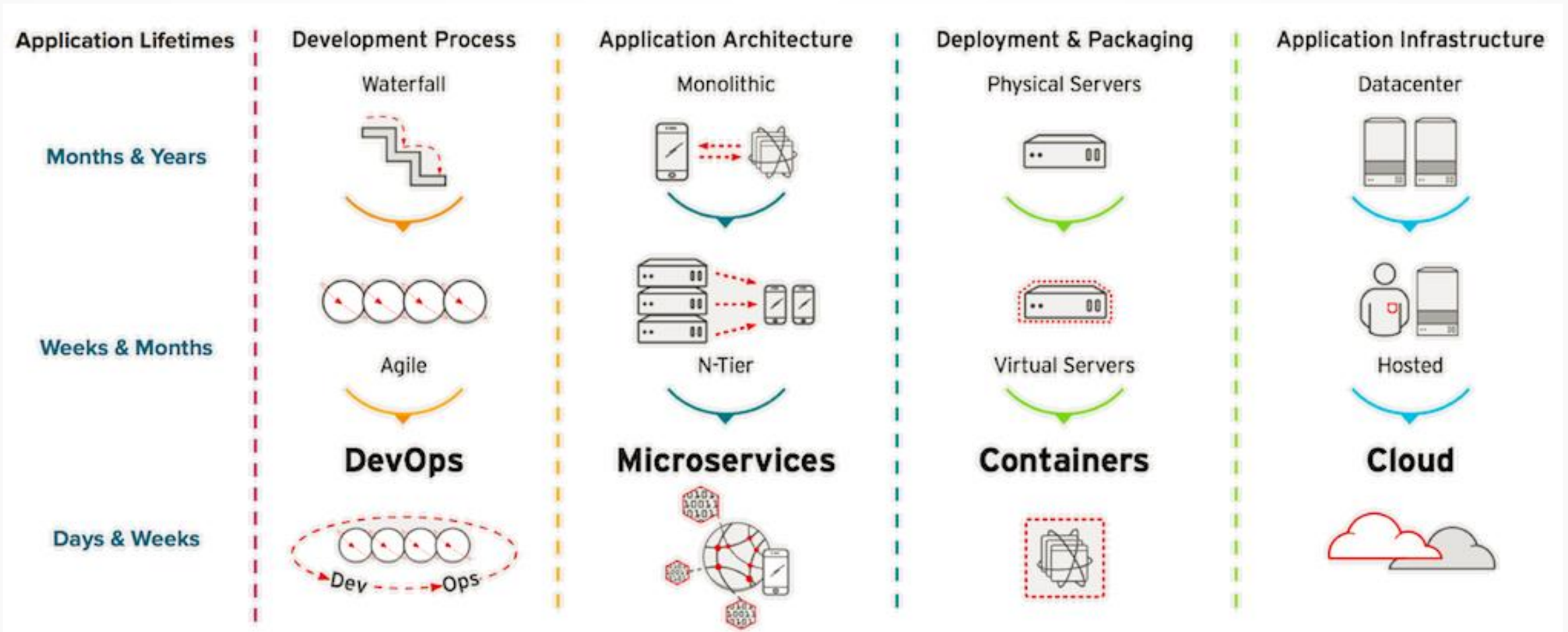
Linux Containers (LXC) details:

- An isolated user space within a running Linux OS
- Shared kernel across containers
- Direct device access
- All packages and data in an isolated runtime, saved as a filesystem
- Resource management implemented with control groups (cgroups)
- Resource isolation through namespaces



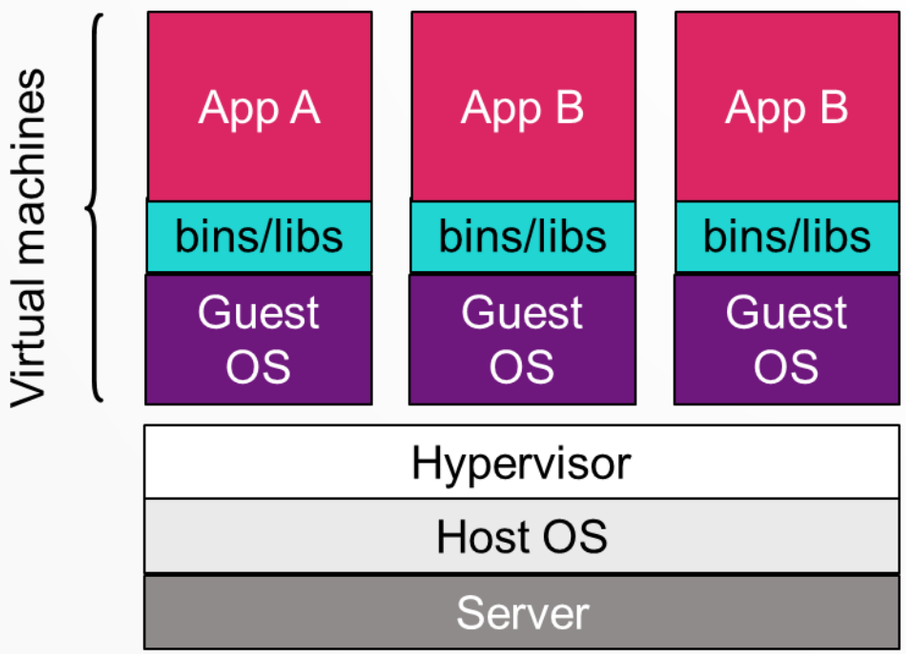
Cloud-native practices comprise of four main tenets

The 4 Elements Technology Required

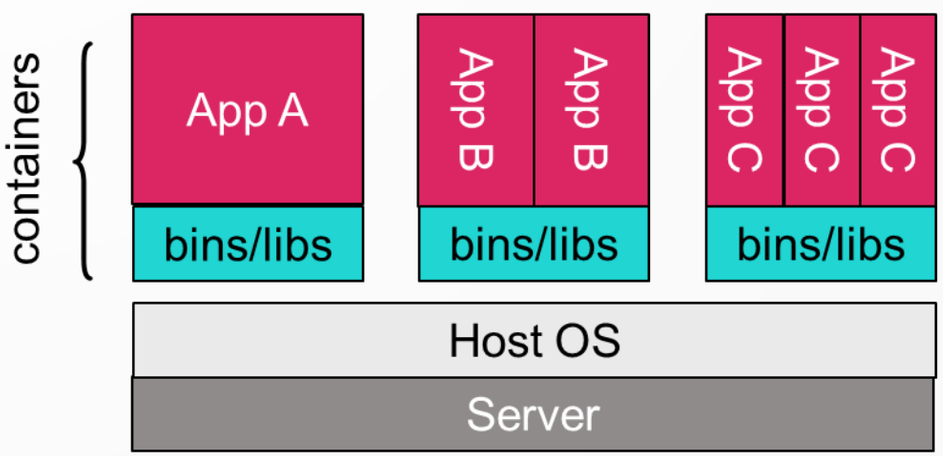


Source: https://www.stackinnovator.com/blog/why-migrate-workloads-to-containers/

Virtual Machine vs Contrainer



Containers are isolated, but share OS and, where appropriate, bins/libraries



Why use Contrainer ?

Containers are a critical foundation for distributed apps in hybrid clouds

Ship more software

Accelerate development, CI and CD pipelines by eliminating headaches of setting up environments and dealing with differences between environments. On average, Docker users ship software more frequently.

Resource efficiency

Lightweight containers run on a single machine and share the same OS kernel while images are layered file systems sharing common files to make efficient use of RAM and disk and start instantly.

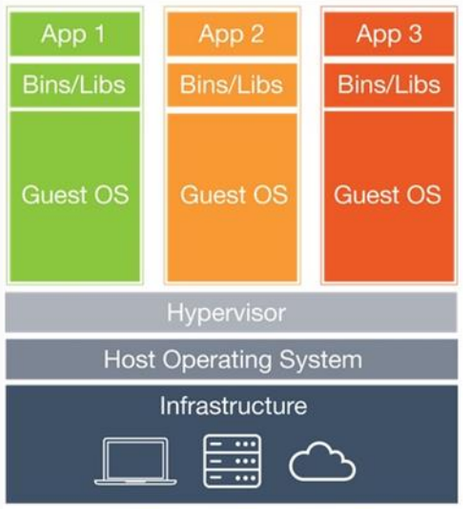
App portability

Isolated containers package the application, dependencies, and configurations together. These containers can then seamlessly move across environments and infrastructures.

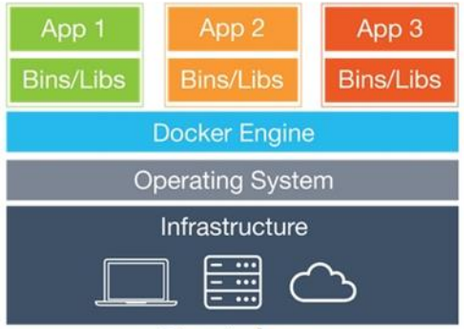
Benefits of using containers

- Can run on many **different platforms**
- Processes **share** OS resources, but remain segregated
- **Isolate** the different requirements between the applications that run inside the container, and the operations that run outside the container
- Quick and easy to create, delete, start, stop, download, and share
- Use hardware resources more efficiently than virtual machines, and are more lightweight
- Can be treated as **unchangeable**

Containers



Virtual Machines



Containers

1/50 Boot time

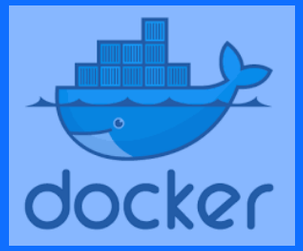
1/90VM cost

1/8CPU

Container ecosystem

Docker

The most common standard, made Linux containers usable by the masses



Rocket (rkt)

An emerging container standard from CoreOS, the company that developed etcd

Garden

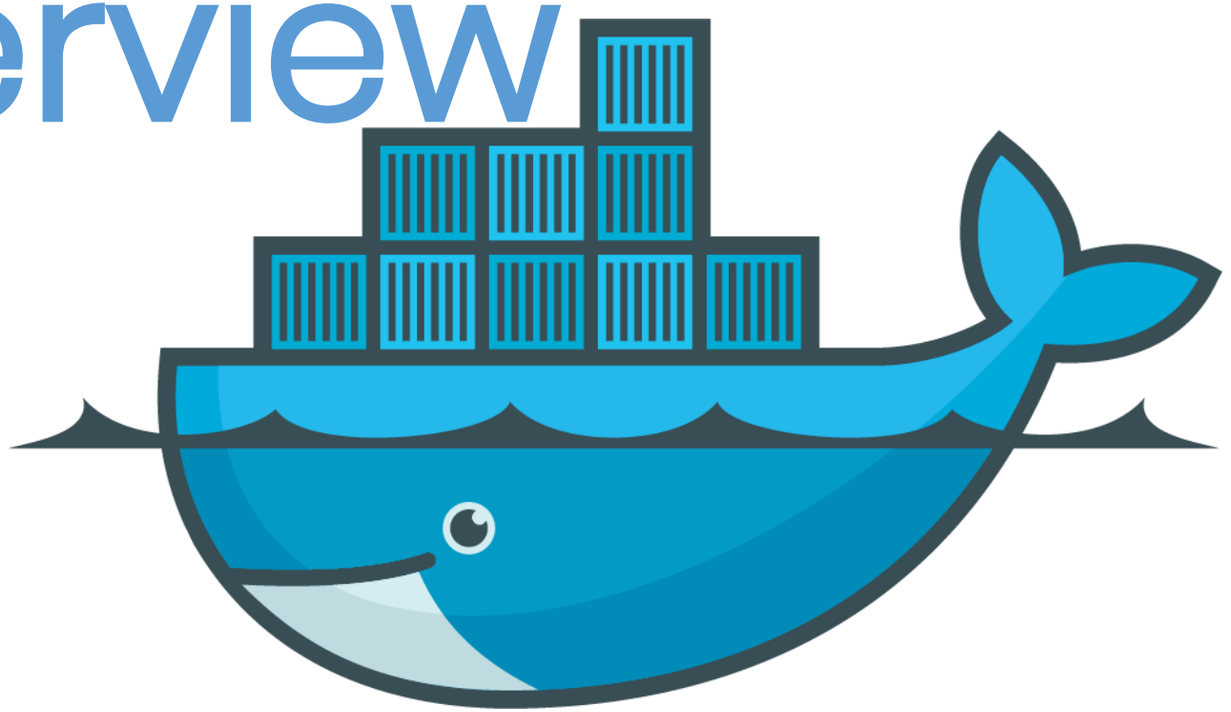
Cloud Foundry component for creating and managing containers

Open Container Initiative (OCI)

A Linux Foundation project that is developing a governed container standard

© 2018 IBM Corporation

Docker Overview



The challenge

Multiplicity of Stacks

Static website:
- Nginx
- OpenSSL
- Bootstrap 2
- ModSecurity

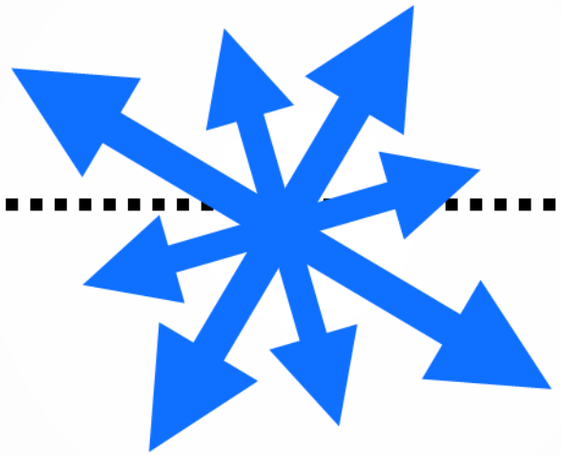
User DB:
- PostgreSQL
- pgv8
- v8

Web front end:
- Ruby
- Rails
- Sass
- Unicom

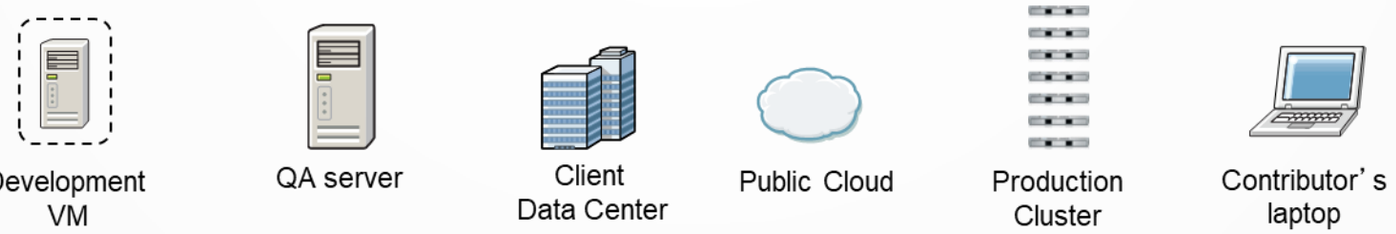
Queue:
- Redis
- Redis-sentinel

Analytics DB:
- Hadoop
- Hive
- Thrift
- OpenJDK

Do services and apps interact appropriately?



Multiplicity of hardware environments



Can I migrate smoothly and quickly?

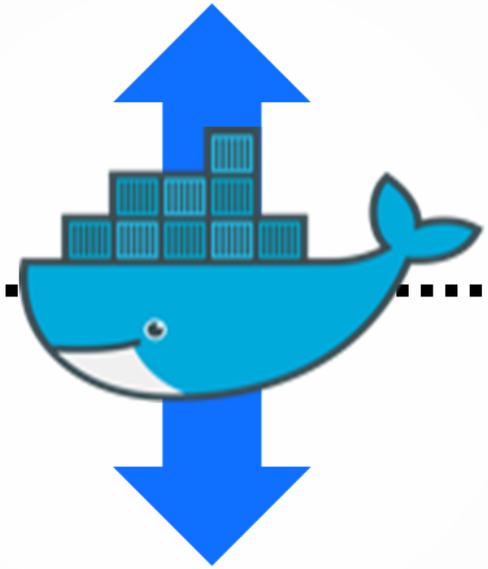
Docker: a shipping container for code

Multiplicity of Stacks

Static website | User DB | Web front end | Queue | Analytics DB

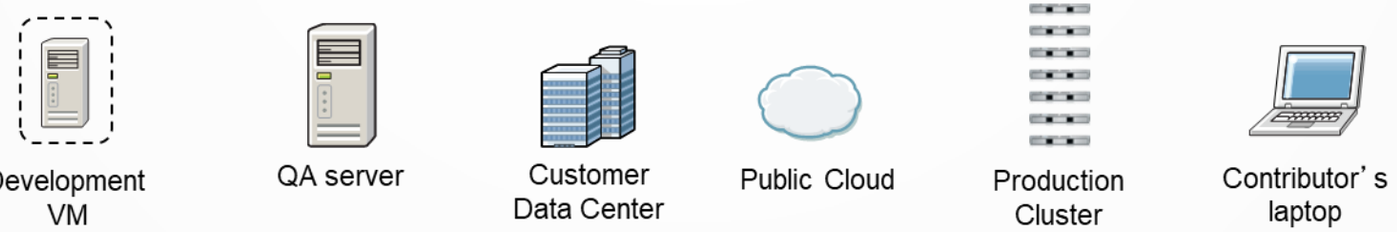
Do services and apps interact appropriately?

An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…



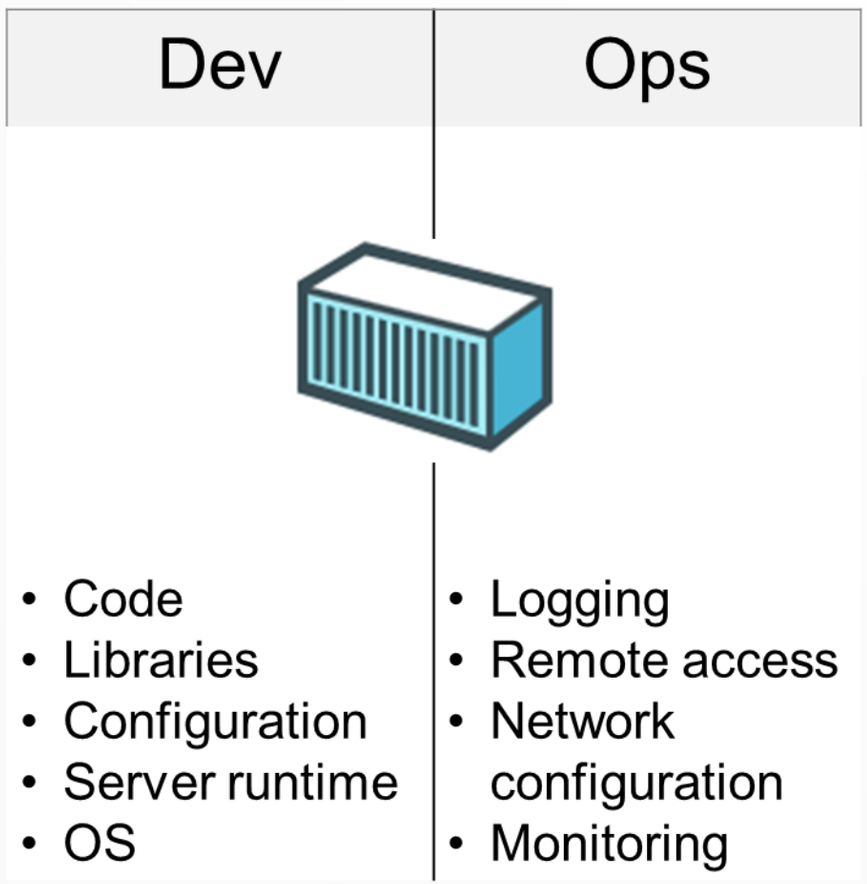
…that can be manipulated by using standard operations, and run consistently on virtually any hardware platform.

Multiplicity of hardware environments



Can I migrate smoothly and quickly?

Dev vs Ops



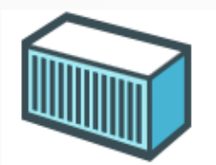
Separation of concerns

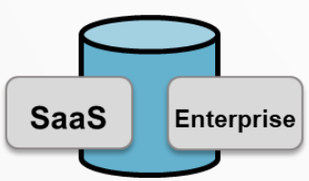
- A container separates and bridges the **Dev** and **Ops** in DevOps
- **Dev** focuses on the application environment
- **Ops** focuses on the deployment environment

Docker Benefit

Return on Investment and cost savings	• Reduce migration time • Reducing Infrastructure (Server, Storage) resources. • Server-independent, OS-independent
Standardization and Productivity	• Reduce implement, speed to market • Docker-based architecture is actually standardization
CI Efficiency	• Separate non-dependent steps and run them in parallel
Compatibility and Maintainability	• Eliminate the “it works on my machine” problem once and for all. • Less time spent setting up environments, debugging environment-specific issues,
Rapid Deployment	• Docker manages to reduce deployment to seconds.
Multi-Cloud Platforms	• Docker containers can be run inside an Amazon EC2 instance, Google Compute Engine instance, Rackspace server, or VirtualBox, provided that the host OS supports Docker.
Security	• No Docker container can look into processes running inside another container.
Loosely coupled, distributed, elastic, liberated micro-services:	• Applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a fat monolithic stack running on one big single-purpose machine.

Docker basic concepts





```
>_ docker
```

Image

- A read-only snapshot of a container that is stored in a Docker registry and used as a template for building containers

Container

- The standard unit in which the application service resides or is transported

Registry

- Available in SaaS or Enterprise to deploy anywhere you choose

Stores, distributes and shares container images

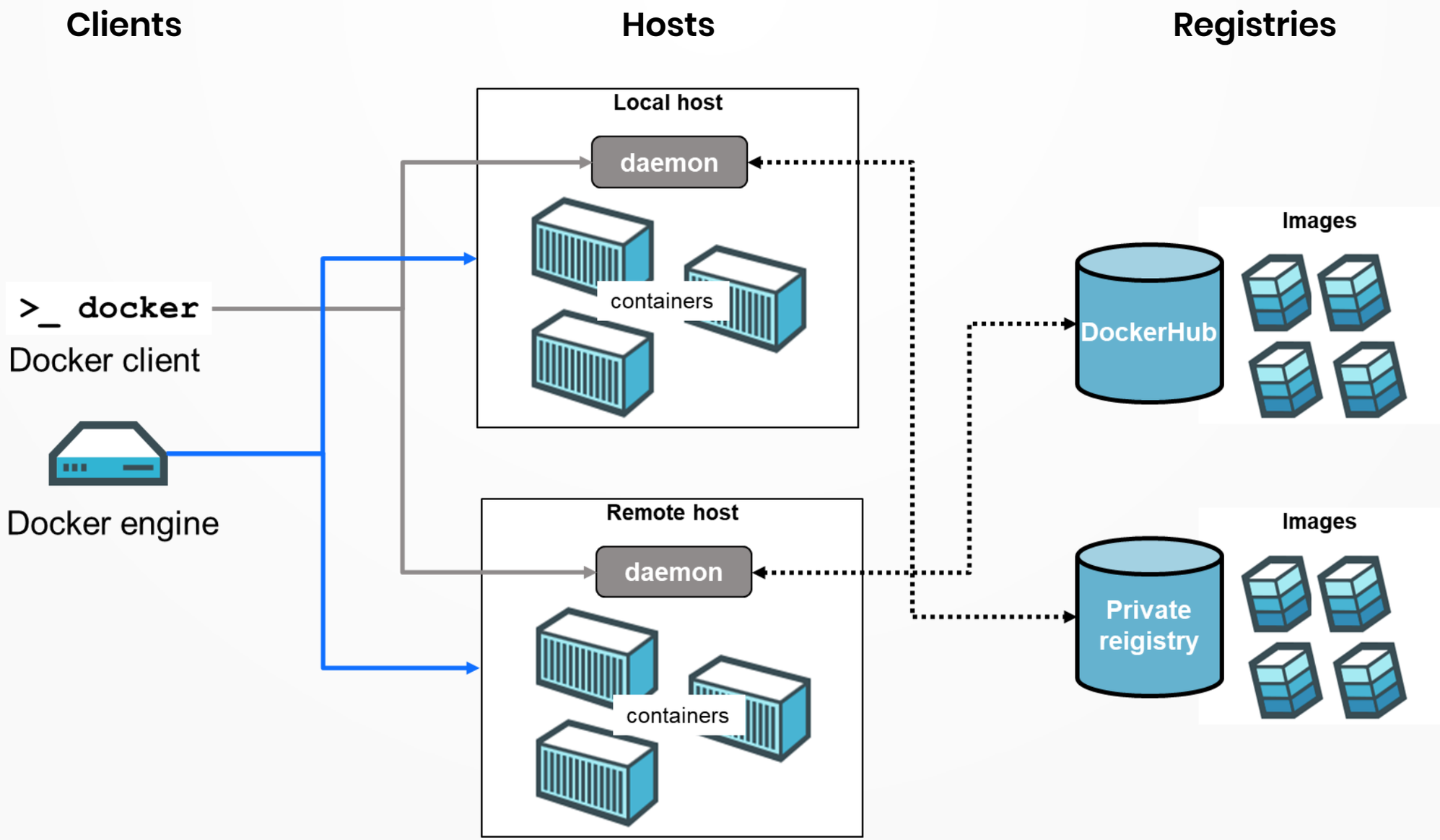
Engine

- A program that creates, ships and runs application containers
- Runs on any physical or virtual machine locally, in private, or public cloud

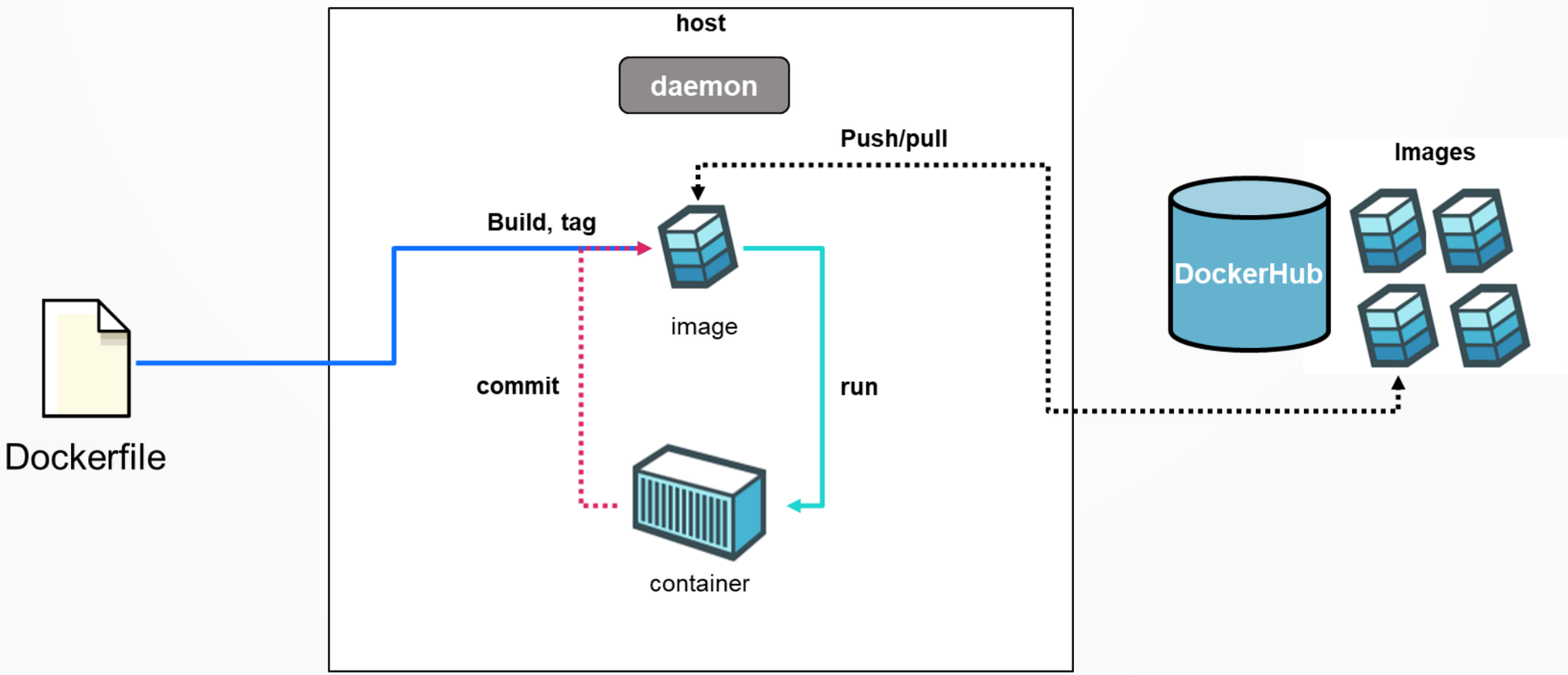
Client

- Communicates with engine to execute commands

Docker Architecture

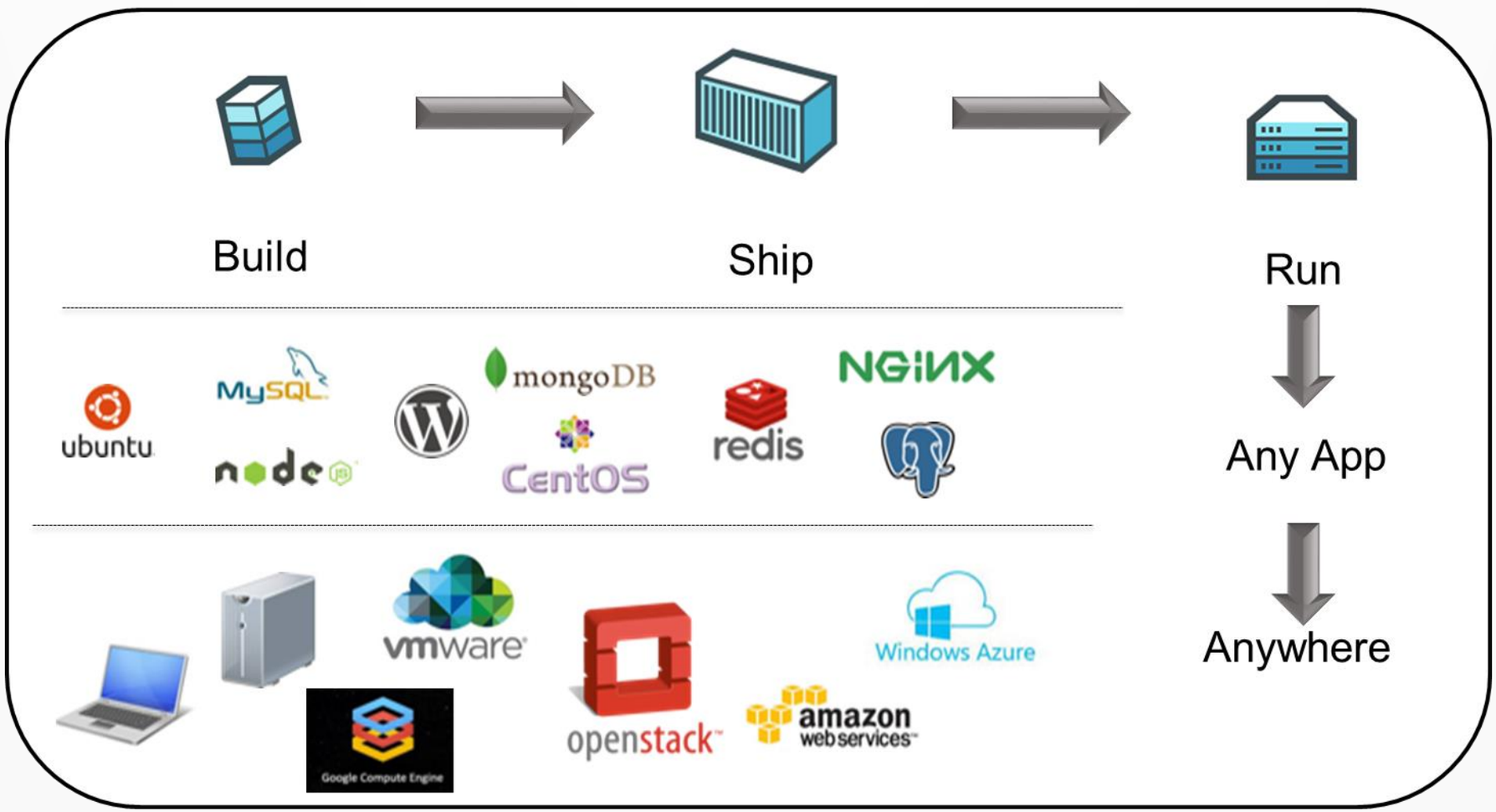


Typical Workflow



Docker Mission

Docker is an open platform for building distributed applications for developers and system administrators.

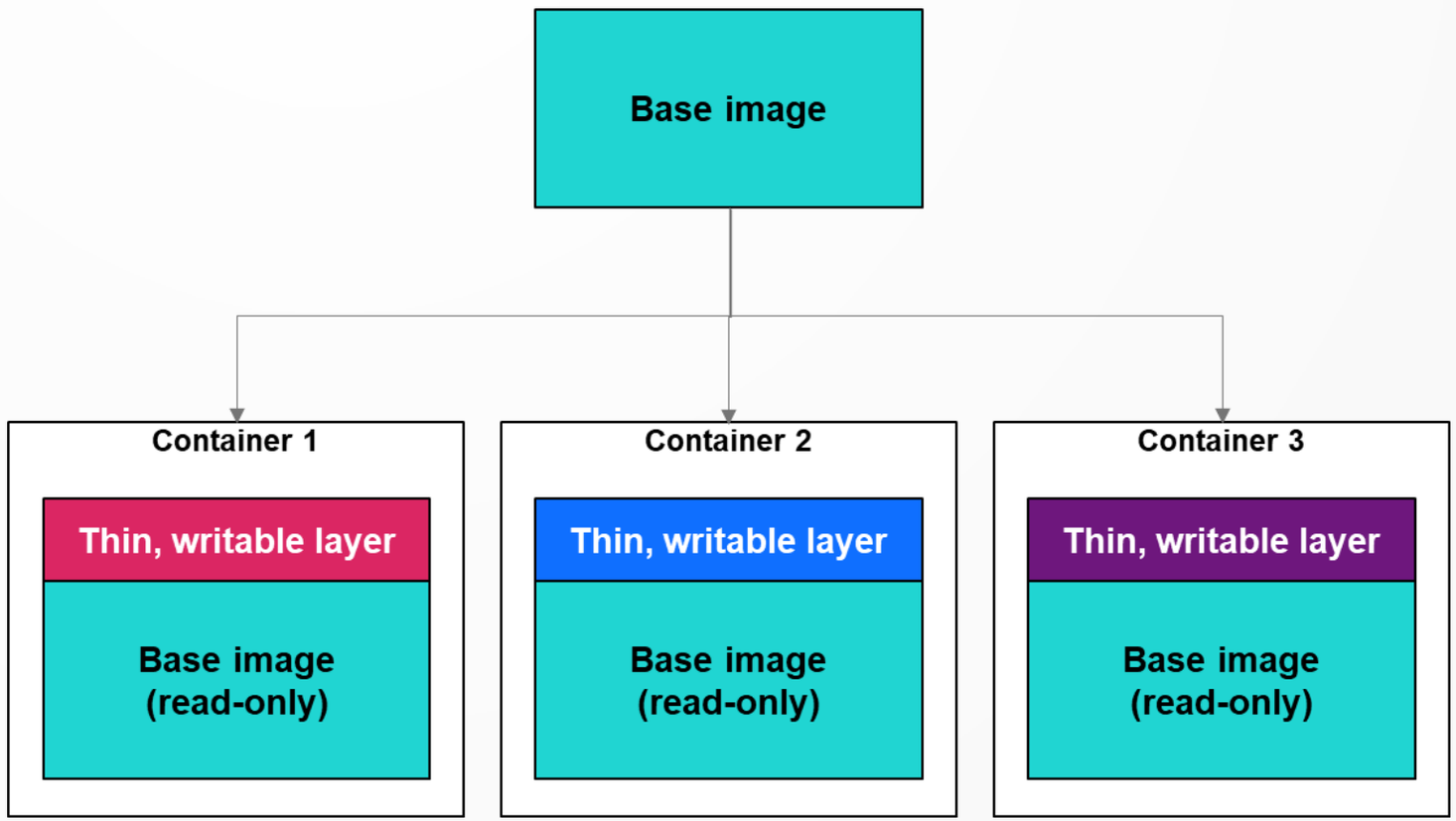


Typical Workflow

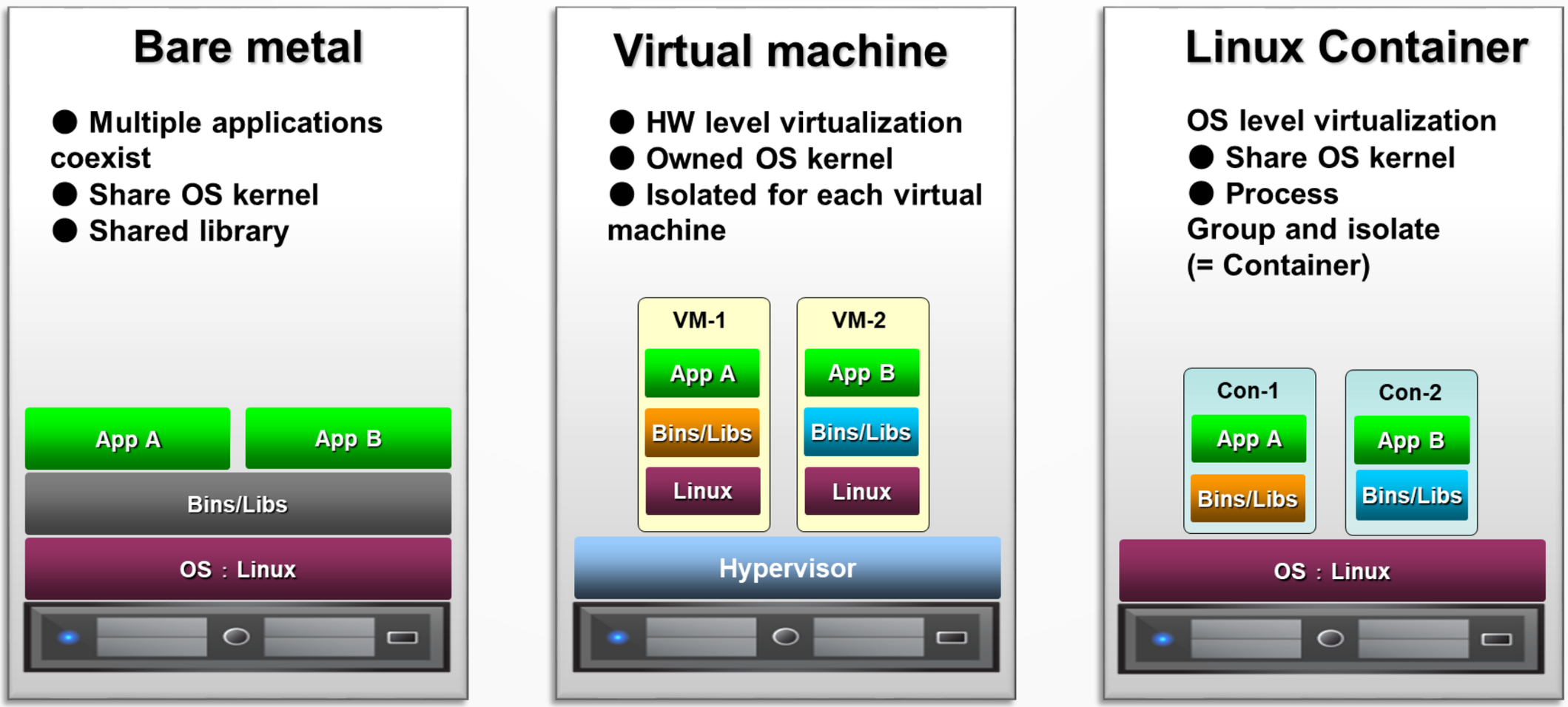
Docker uses a copy-on-write (union) file system. New files and edits are only visible to current and above layers.

Saves disk space and allows images to build faster.

Maintains filesystem integrity by isolating the contents.

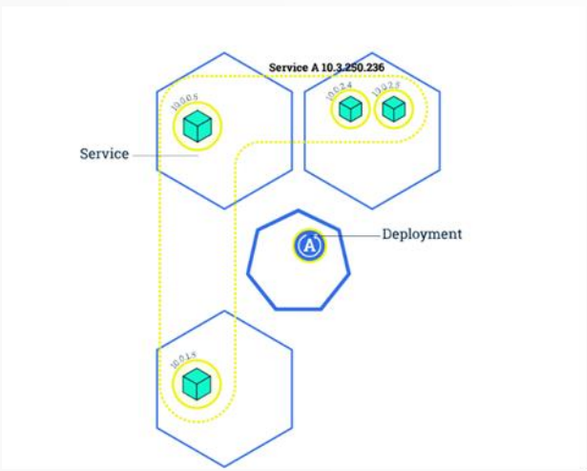


Overview of Container Orchestration.

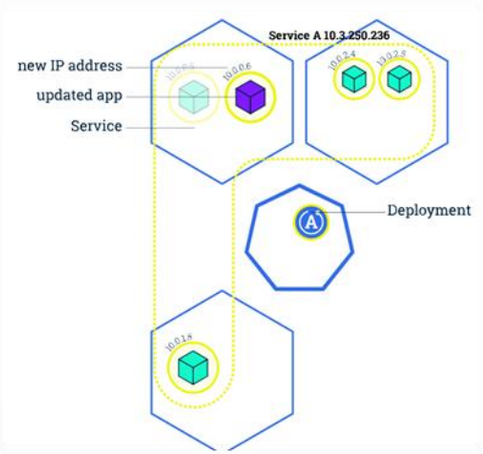


Overview of Container Orchestration.

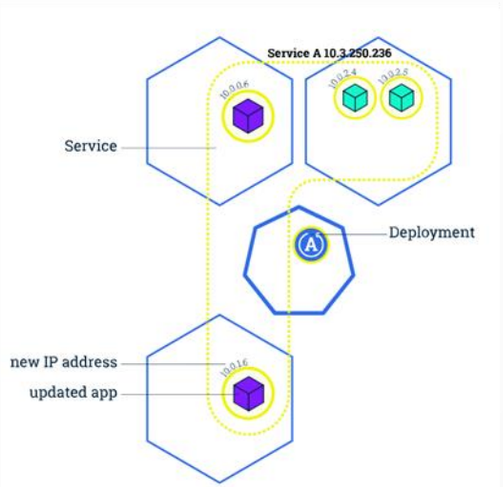
Green version



Step 1

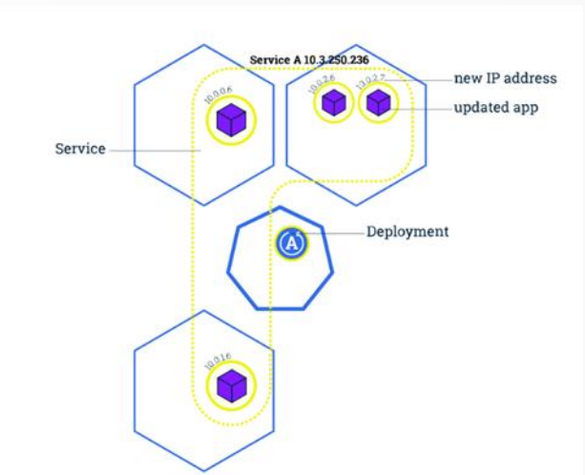


Step 2



Step 3

Purple version



Step 4

Improve Virtual Machine with Container Orchestration

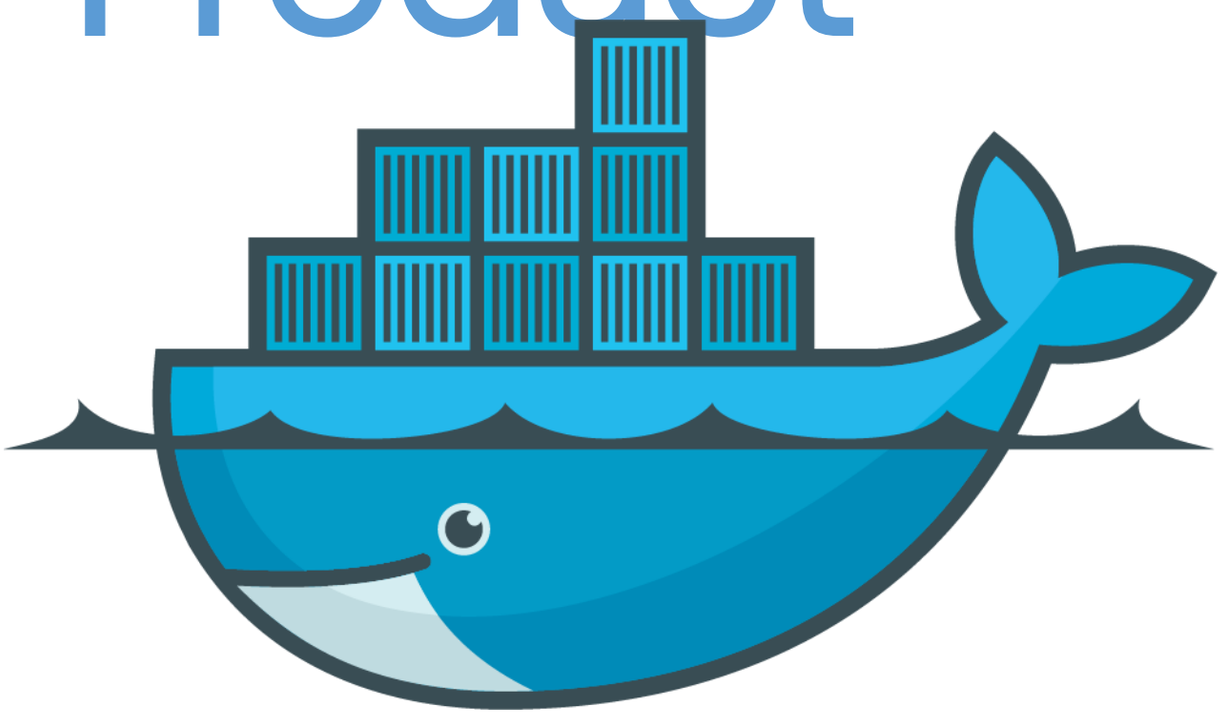
High Availability

- Horizontal Scale
- Auto scaling
- Reduce downtime

Manage applications, not machines

- Rolling updates
- canary deploys
- blue-green deployments

Docker Product



Docker Product

Docker CE

Docker Community Edition (CE) is the new name for the free Docker products. Docker CE runs on Mac and Windows 10, on AWS and Azure, and on CentOS, Debian, Fedora, and Ubuntu and is available from Docker Store. Docker CE includes the full Docker platform and is great for developers and DIY ops teams starting to build container apps.

Docker CE comes in two variants:

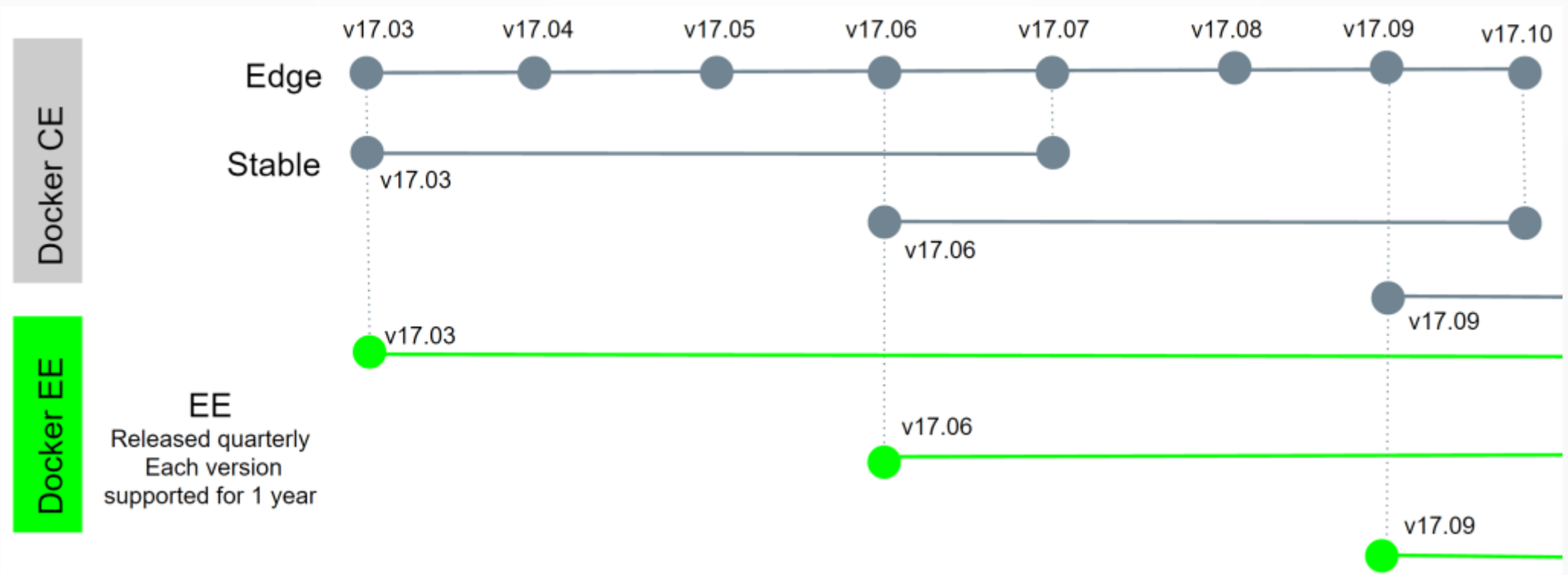
- Edge is for users wanting a drop of the latest and greatest features every month
- Stable is released quarterly and is for users that want an easier-to-maintain release pace

Docker EE

Docker Enterprise Edition (EE) is an integrated, supported and certified container platform for CentOS, Red Hat Enterprise Linux (RHEL), Ubuntu, SUSE Linux Enterprise Server (SLES), Oracle Linux, and Windows Server 2016, as well as for cloud providers AWS and Azure. In addition to certifying Docker EE on the underlying infrastructure, we are introducing the Docker Certification Program which includes technology from our ecosystem partners: ISV containers that run on top of Docker and networking and storage and networking plugins that extend the Docker platform.

Docker Product

Docker CE and EE are released quarterly, and CE also has a monthly “Edge” option. Each Docker EE release is supported and maintained for one year and receives security and critical bugfixes during that period. We are also improving Docker CE maintainability by maintaining each quarterly CE release for 4 months. That gets Docker CE users a new 1-month window to update from one version to the next.



Docker Product

Platform	Docker CE	Docker EE
Desktop		
• Mac	✔	
• Windows	✔	
Cloud		
• Amazon Web Services	✔	✔
• Microsoft Azure	✔	✔
Server		
• CentOS	✔	✔
• Oracle Linux	✔	✔
• Ubuntu	✔	✔
• Microsoft Windows Server 2016	✔	✔
• Red Hat Enterprise Linux		✔
• SUSE Linux Enterprise Server		✔

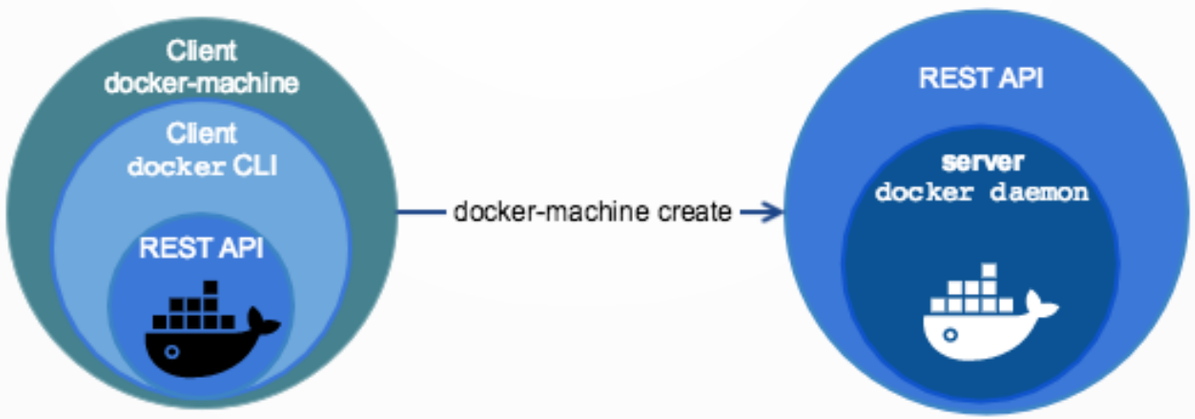
Docker Tools

Docker Engine is a client-server application with these major components:

- A server which is a type of long-running program called a daemon process.
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client.

Docker Machine is a tool that lets you install Docker Engine on virtual hosts, and manage the hosts with docker-machine commands. You can use Machine to create Docker hosts on your local Mac or Windows box, on your company network, in your data center, or on cloud providers like Azure, AWS, or Digital Ocean.

Docker Tools



Docker Preparing

Lab 0 – Preparing

** For Windows

- Enable Virtualization Technology (vt-x/AMD-v) on BIOS.
  ** Normally virtualization technology should be enable by default.*
- Open PowerShell by “select windows powershell” and run as administrator.
- Run command.

```
> Disable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V-All
```

Lab 0 – Preparing

- Download and Install docker toolbox.

Mac:

https://download.docker.com/mac/stable/DockerToolbox.pkg

Windows (Pro edition):

https://download.docker.com/win/stable/DockerToolbox.exe

*** For Windows Please read next slide.*

Lab 0 – Preparing

- Check Docker version.

```
docker version
```

- Check Docker toolbox version.

```
docker-machine version
```

Lab 0 – Preparing

- Create virtual machine for run Docker Server.

```
docker-machine create --driver=virtualbox --virtualbox-memory “1024” <servername>
```

- Check state and IP Docker Servers.

```
docker-machine ls
```

- Connect to Docker Server.

```
docker-machine ssh <servername>
```

Lab 0 – Preparing

** For using PuTTY

- You can use Docker Server IP address with
  - User: ***docker***
  - Password: ***tcuser***

Lab 0 – Preparing

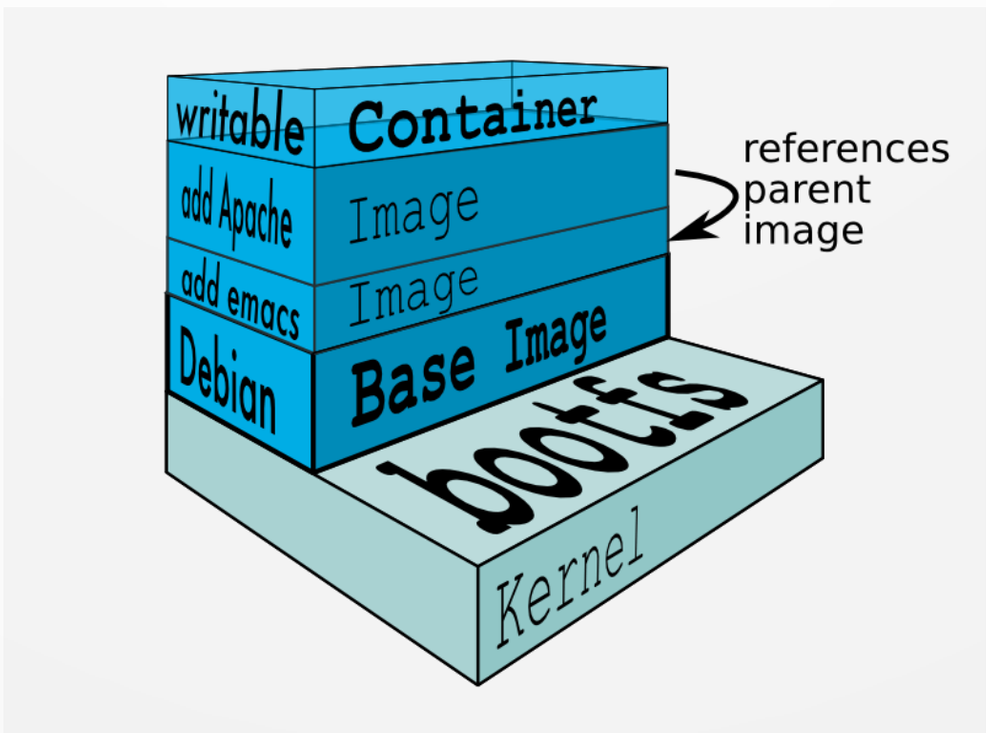
- After ssh Run command test docker.

```
$ docker run hello-world
```

Image

Image

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization.



Image


Image alpine

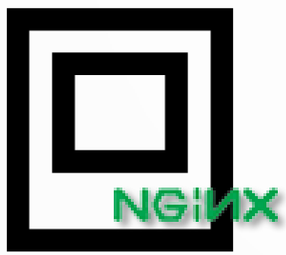


Image nginx

Image php

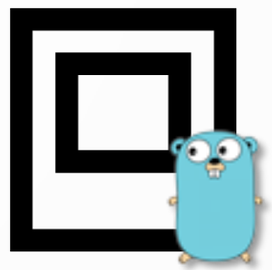


Image golang


Image ubuntu


Image mongo

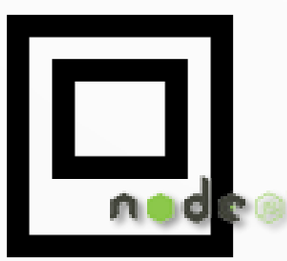


Image node

Image mysql


Image python

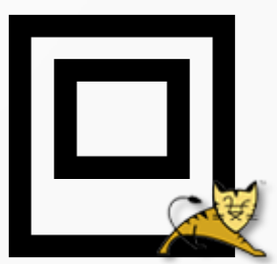


Image tomcat

Image

Docker images commands	
docker image ls	List images.
docker tag <source image:tag> <target image:tag>	Create a tag target image that refers to source image.
docker save --output <output> <image:tag>	Save one or more images to a tar archive (streamed to STDOUT by default).
docker load --input <imagefile>	Load an image from a tar archive or STDIN.
docker pull <image:tag>	Pull an image or a repository from a registry.
docker push <image:tag>	Push an image or a repository to a registry.
docker image rm <images>	Remove one or more images.

Lab 1 – Image

- Download image from registry.

```
$ docker pull nginx
```

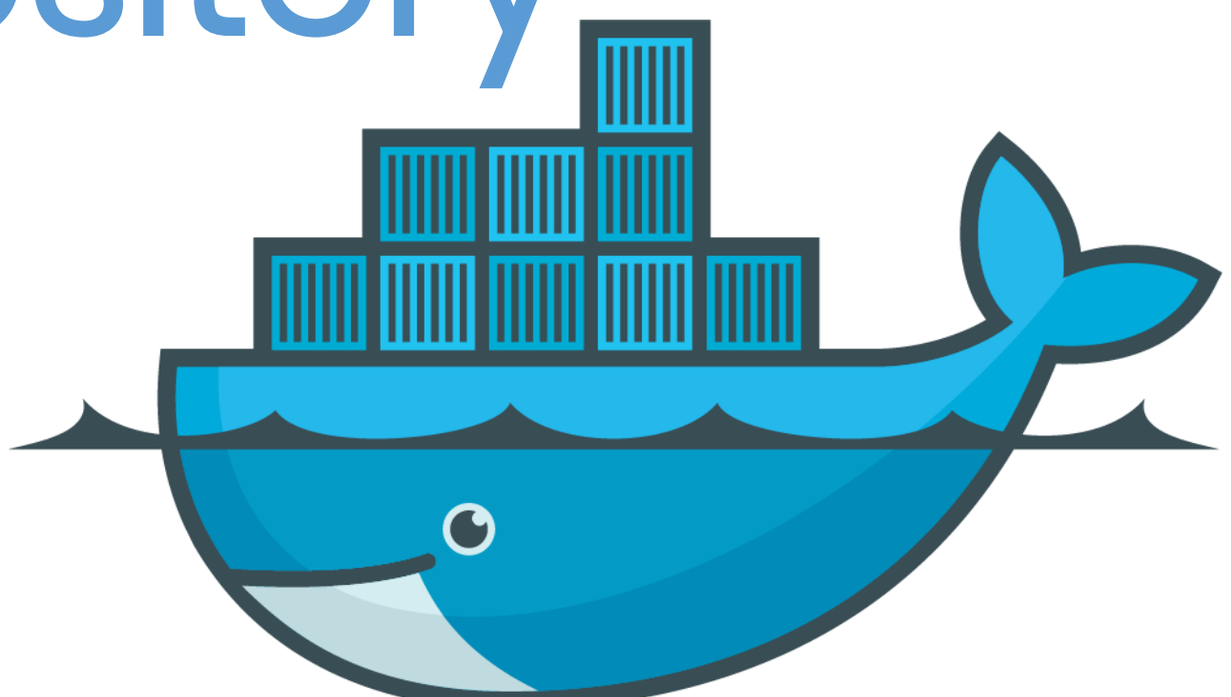
- Show images list.

```
$ docker image ls
```

OpenSSH SSH client

```
docker@labdocker:~$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               proxy               2d7a92d4afc6        12 days ago         156MB
tomcat              latest              8aeff3616a22        12 days ago         510MB
nginx               latest              c818be869fd2        13 days ago         156MB
```

Repository



Repository (Registry)


Image alpine

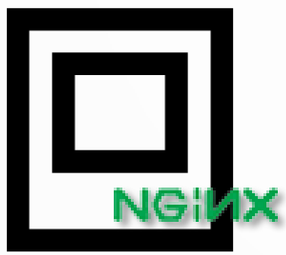


Image nginx

Image php

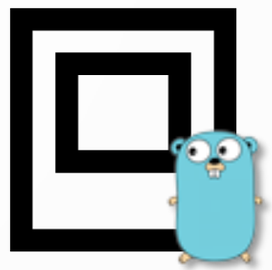


Image golang


Image ubuntu


Image mongo

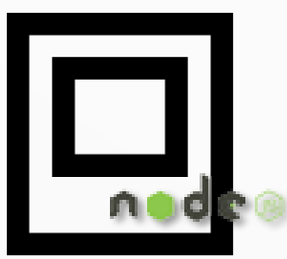


Image node

Image mysql

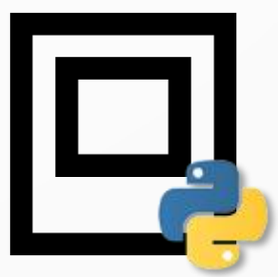


Image python

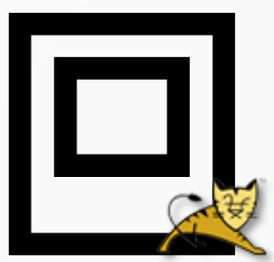
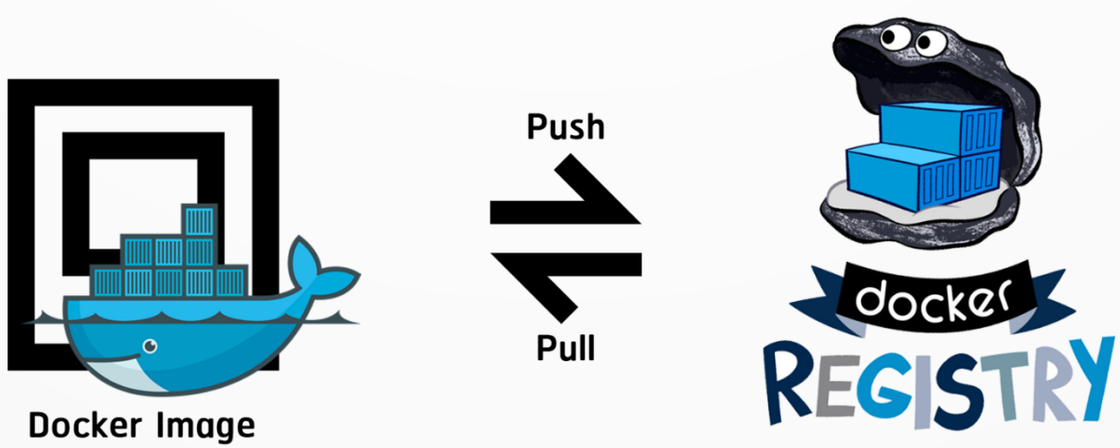


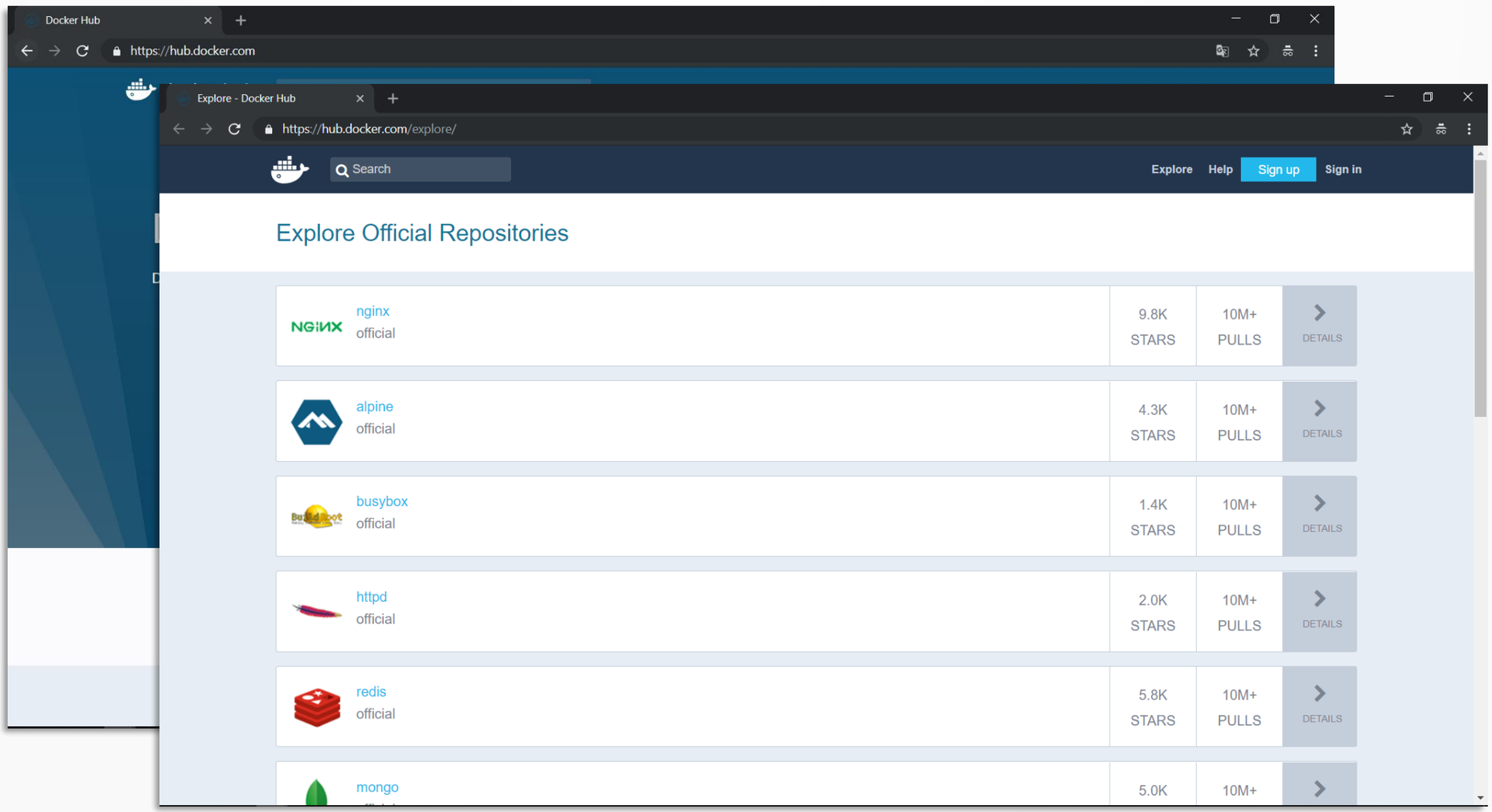
Image tomcat

Repository (Registry)

Docker users store images in private or public repositories, and from there can deploy containers and share them. Docker offers Docker Hub, which is a cloud-based registry service that includes private and public image repositories. It also has Docker Trusted Registry, which adds image management and access control features.



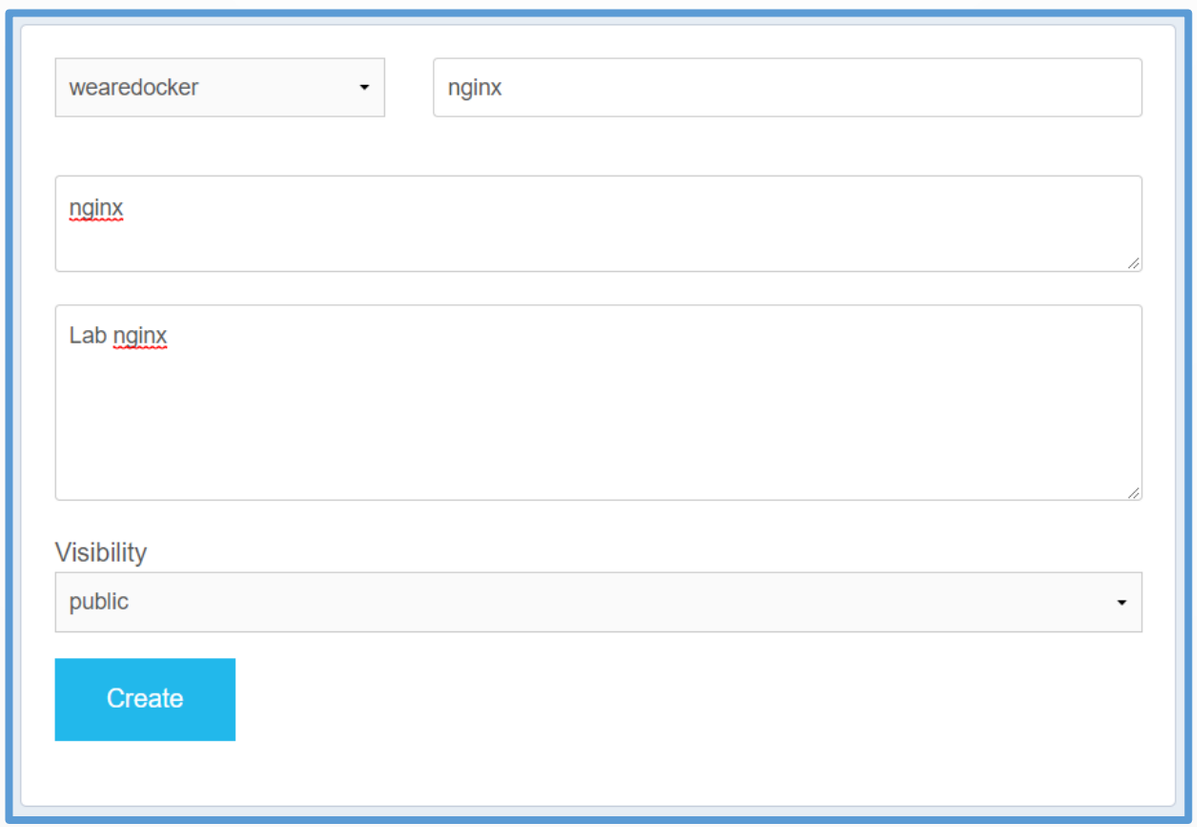
Docker Repository (Registry)



https://hub.docker.com/

Lab 2 – Repository (Registry)

- Go to http://hub.docker.com
- Create repository.



Lab 2 – Repository (Registry)

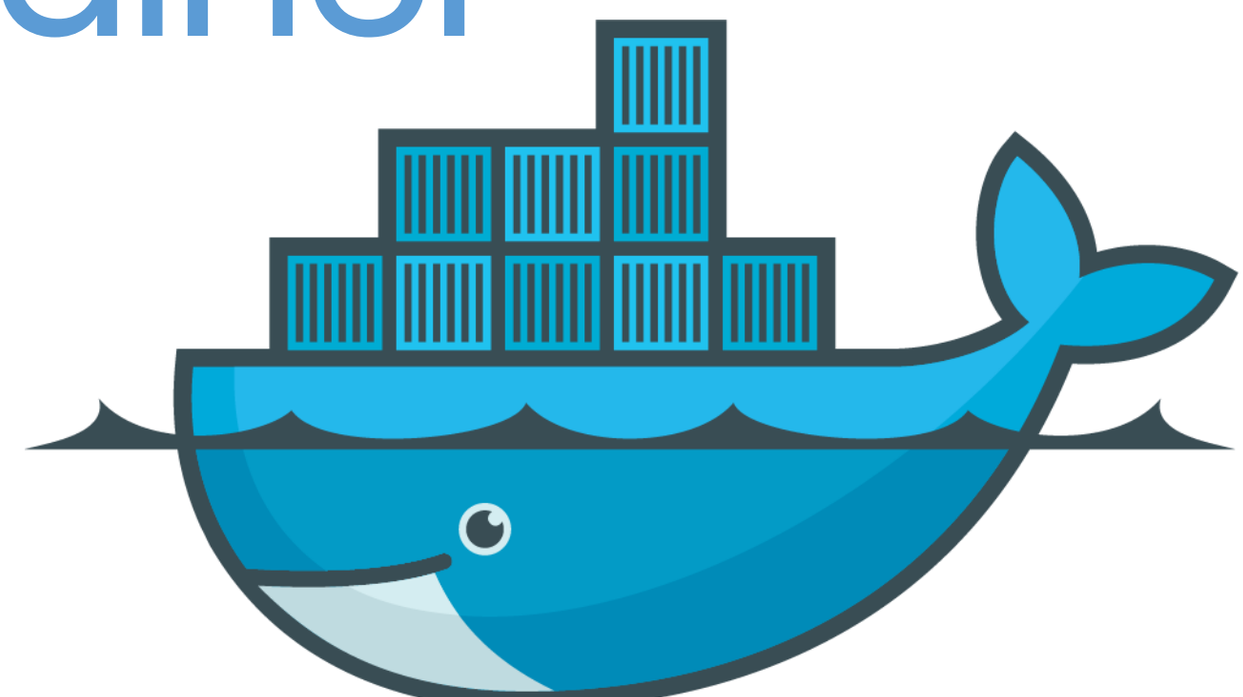
- Create tag image.

```
$ docker tag nginx <username>/nginx:lab2
```

- Push an image to a registry.

```
$ docker login -u <username>
$ docker push <username>/nginx:lab2
$ docker logout
```

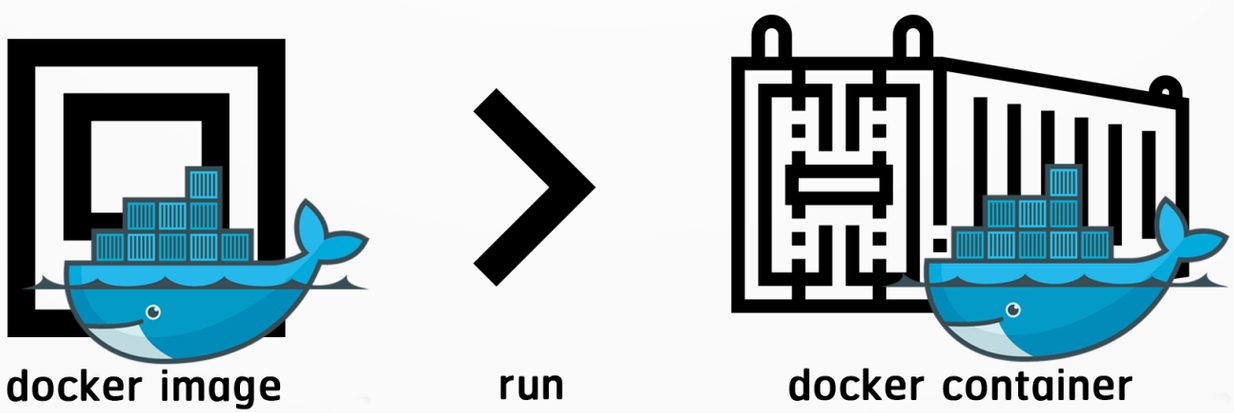
Container



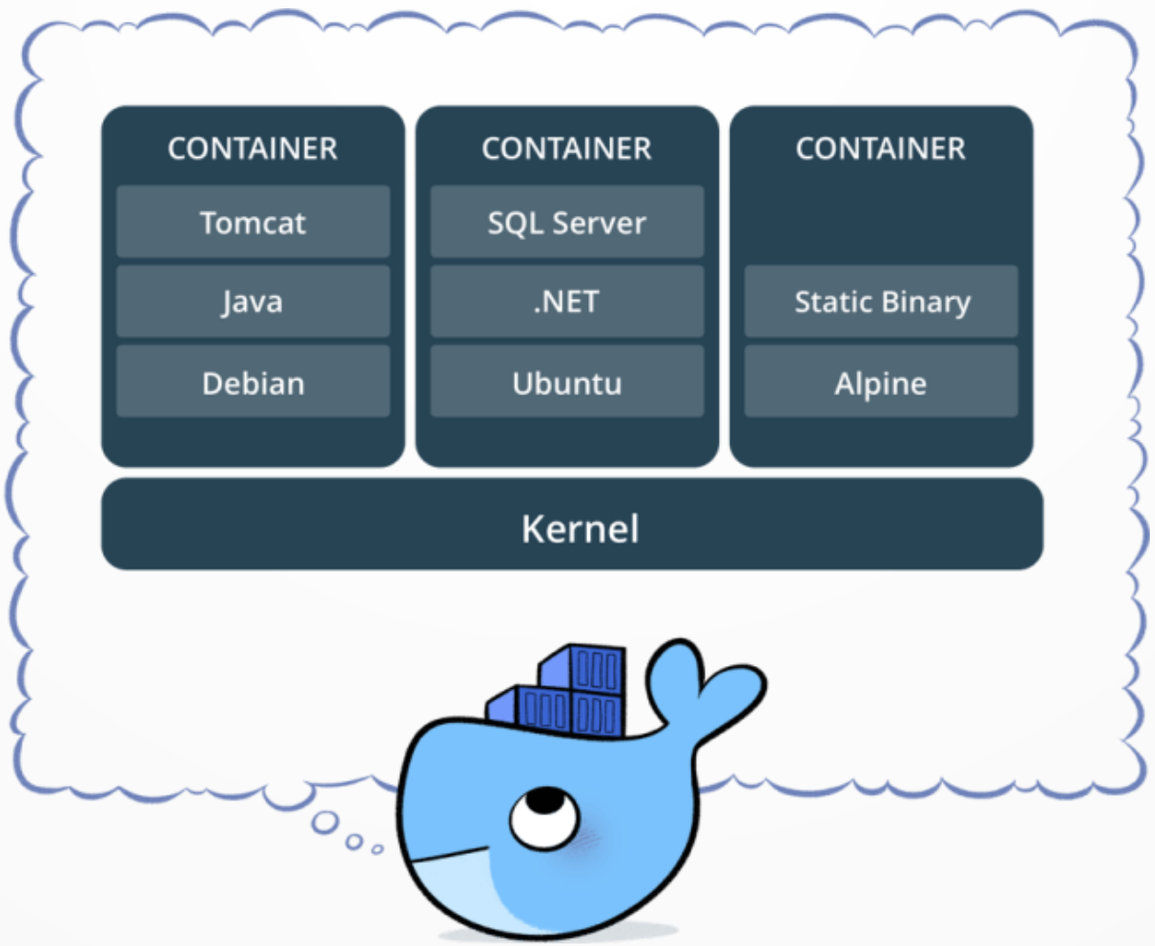
Container

Docker provides the ability to package and run an application in a loosely isolated environment called a ***container***.

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.



Container


CONTAINER
Tomcat
Java
Debian
CONTAINER
SQL Server
.NET
Ubuntu
CONTAINER
Static Binary
Alpine
Kernel

Container

Docker container commands	
docker ps <option>	List containers.
docker run <options> <image:tag> <command>	Run a new container.
docker exec <options> <container> <command>	Run a command in a running container.
docker commit <options> <container> <image:tag>	Create a new image from a container's changes.
docker <start/stop/restart> <container>	Start, Stop and Restart a container.
docker rm <container>	Remove one or more containers.

Lab 3 – Container

- Run a container.

```
$ docker pull nginx:1.15.5-alpine
$ docker run -d -p 80:80 --name <containername>
nginx:1.15.5-alpine
```

- List containers.

```
$ docker ps
```

- Open URL in browser.
  http://192.168.99.100:80/

Lab 3 – Container

- Executes command in a container.

```
$ docker exec -it <contianername> sh
```

- Edit html file.

```
$ vi /usr/share/nginx/html/index.html
```

- Commit a container's changes.

```
$ docker commit <contianername> nginx:html
```

Storage

Storage

By default all files created inside a container are stored on a writable container layer. This means the data doesn't persist when that container is no longer running, and it can be difficult to get the data out of the container if another process needs it.

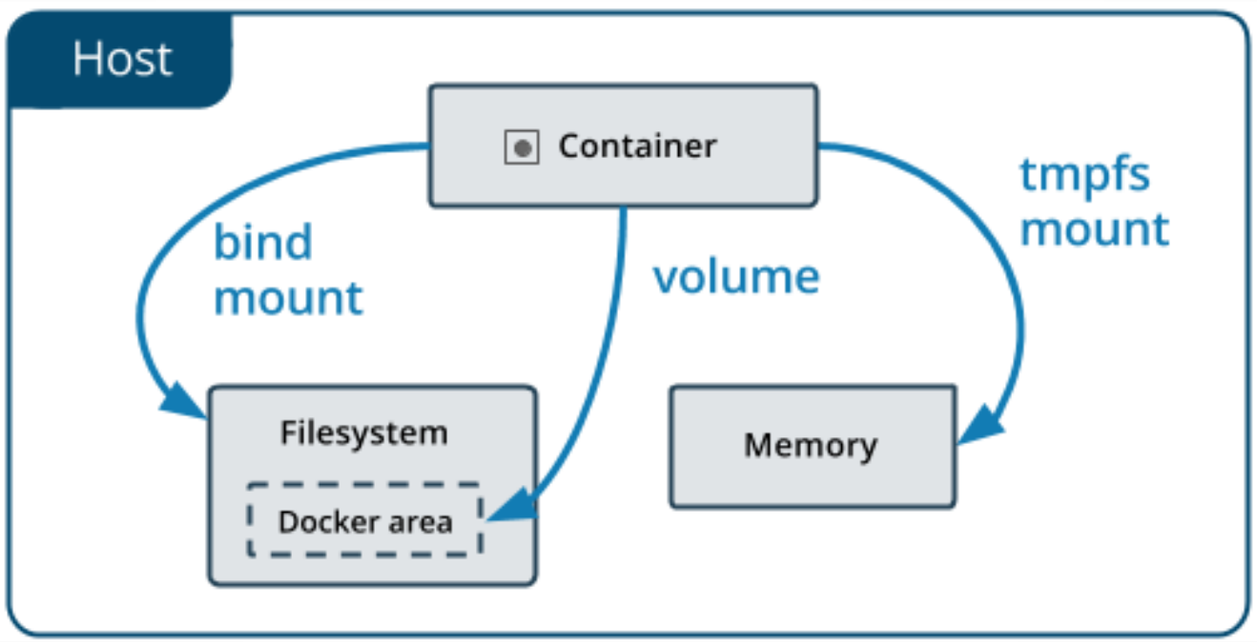
Docker has two options for containers to store files in the host machine, so that the files are persisted even after the container stops: *volumes*, *bind mounts and tmpfs mount*.

Storage

Type of mount

- ***Volumes*** are stored in a part of the host filesystem which is managed by Docker. Non-Docker processes should not modify.
- ***Bind mounts*** may be stored *anywhere* on the host system. They may even be important system files or directories. Non-Docker processes can modify them at any time.
- ***tmpfs mounts*** are stored in the host system's memory only.

Storage



Type of mount

Storage

Docker storage commands	
docker volume ls	List volumes.
docker volume create <options> <volume>	Create a new volume.
docker volume rm <volumes>	Remove one or more volumes.
docker run <options> --volume <volume>:<target> <image:tag>	Mount a volume.
docker run <options> --tmpfs <directory>:rw,size=<size>,mode=<mode> <image:tag>	Mount a tmpfs directory.

Lab 4.1 – Storage(volume)

- Create a volume.

```
$ docker volume create volumelab
```

- Run containers and mount a volume.

```
$ docker run -d -p 80:80 --name vol1 -v
volumelab:/usr/share/nginx/html nginx:html
$ docker run -d -p 88:80 --name vol2 -v
volumelab:/usr/share/nginx/html nginx:html
```

Lab 4.1 – Storage(volume)

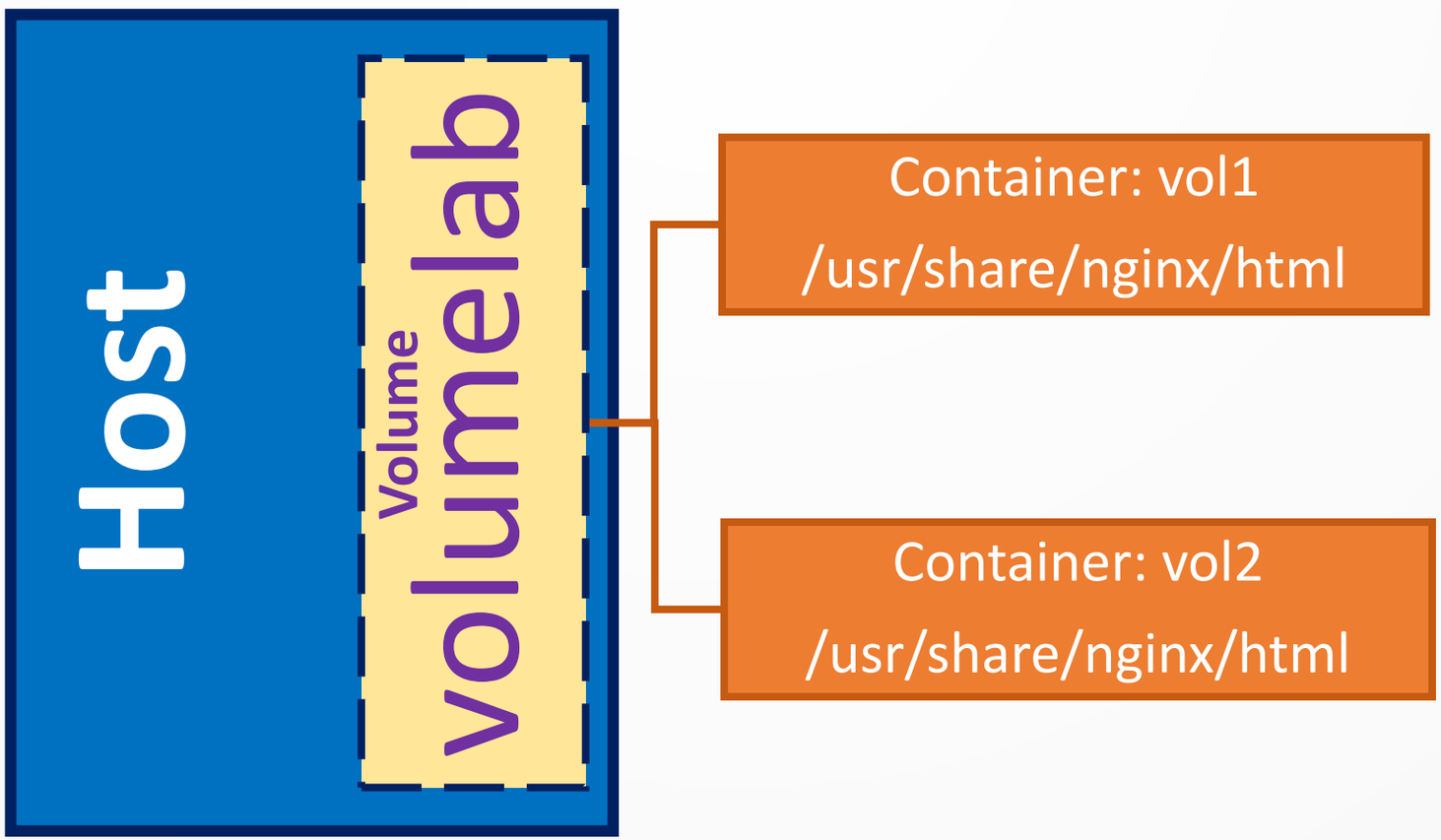
- Open URL in browser.
  http://192.168.99.100:80/
  http://192.168.99.100:88/
- Executes command in a container.

```
$ docker exec -it vol1 sh
```

- Edit html file.

```
$ vi /usr/share/nginx/html/index.html
```

Lab 4.1 – Storage(volume)



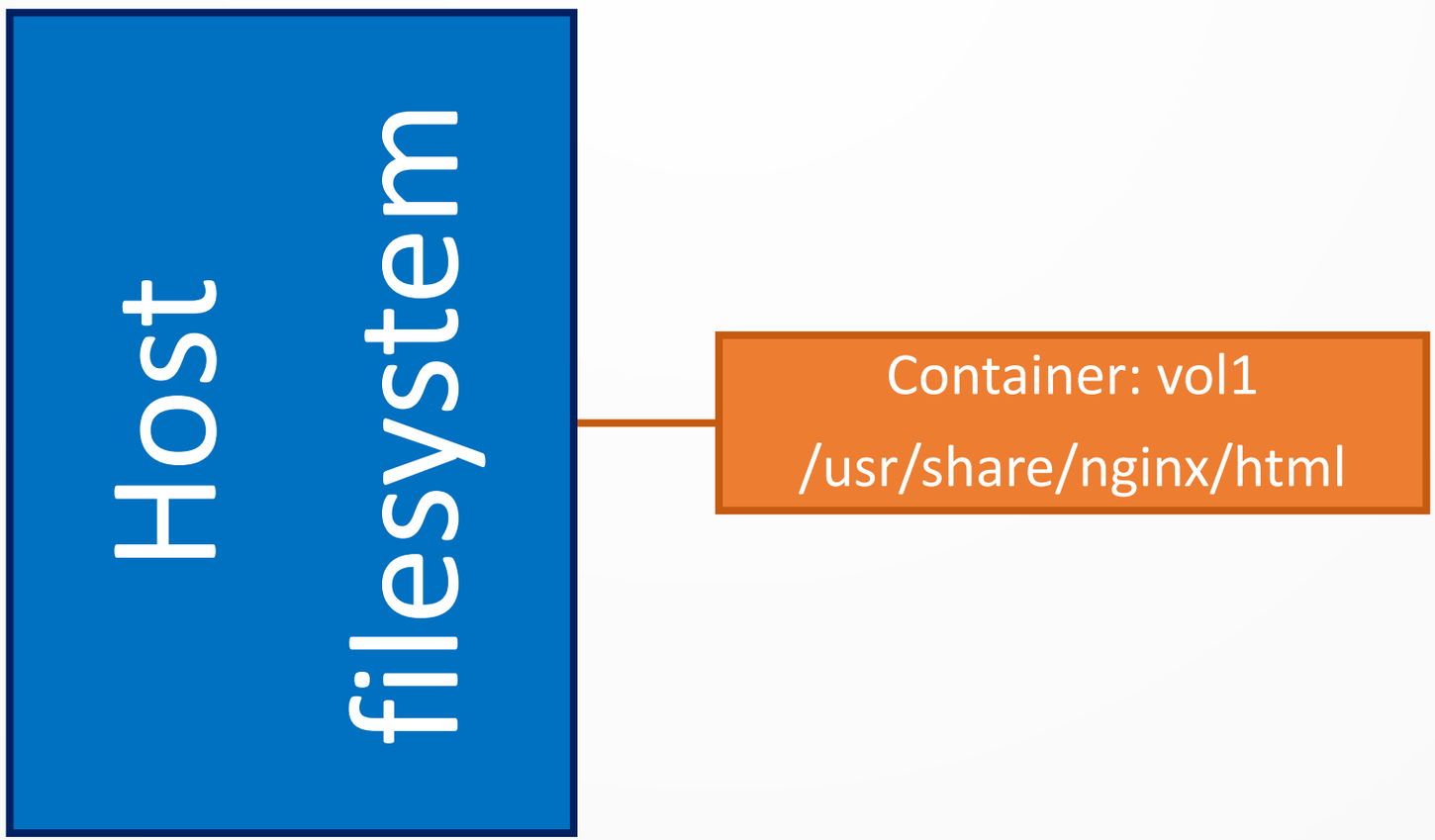
Lab 4.2 – Storage(bind)

- Create share file for Docker Toolbox.
- Run containers and mount a volume type bind.

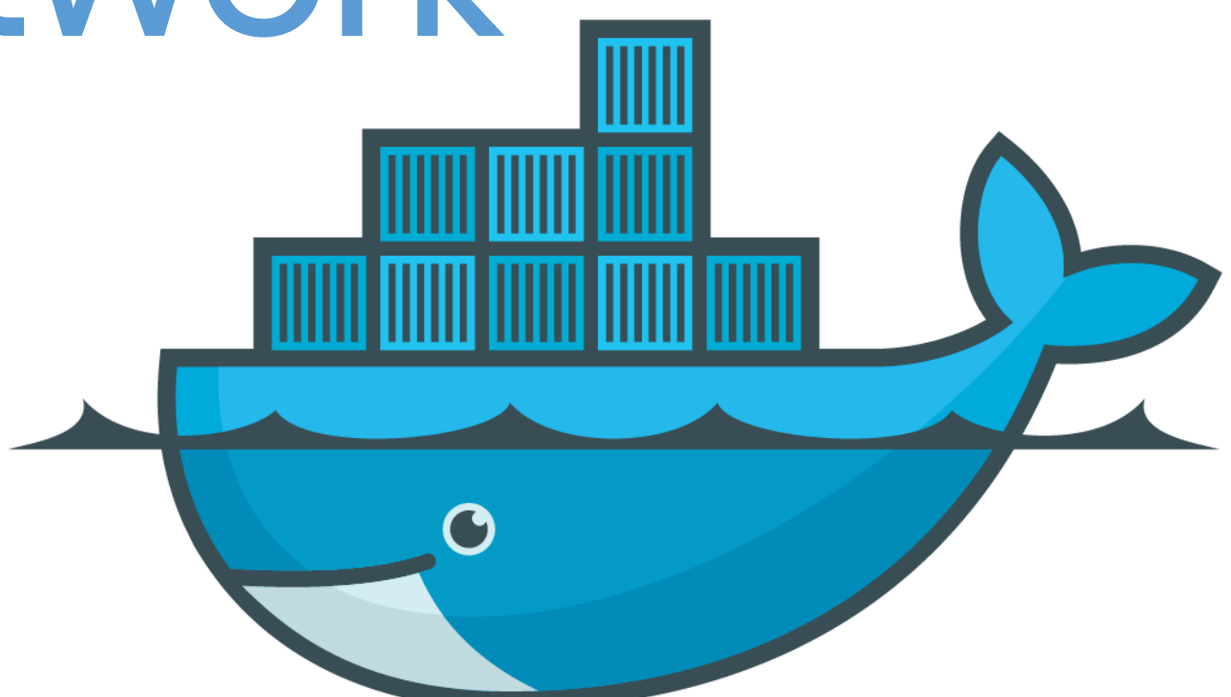
```
$ docker run -d -p 80:80 --name bind1 -v
/share:/usr/share/nginx/html nginx:html
```

- Copy index file to shared folder.

Lab 4.2 – Storage(bind)



Network



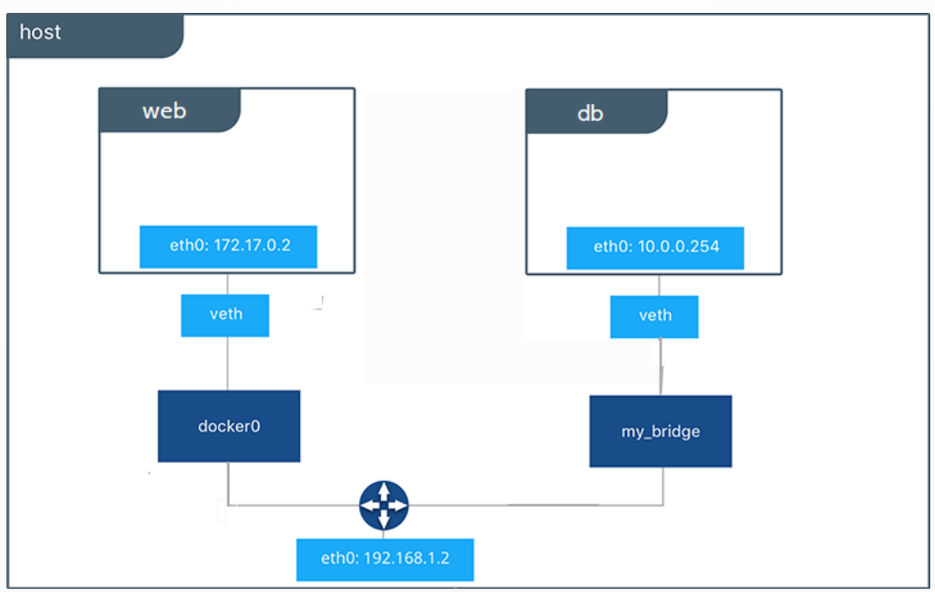
Network

When you initially install Docker, the platform automatically configures three different networks that are named ***none***, ***host*** and ***bridge***. The none and host networks cannot be removed; they're part of the network stack in Docker, but not useful to network administrators since they have no external interfaces to configure. Admins can configure the bridge network, also known as the docker0 network. This network automatically creates an IP subnet and gateway. All containers that belong to this network are part of the same subnet, so communication between containers in this network can occur via IP addressing.

Network

Type of network

- ***Bridge*** are stored in a part of the host filesystem which is managed by Docker. Non-Docker processes should not modify.



Network Bridge

Network

Type of network

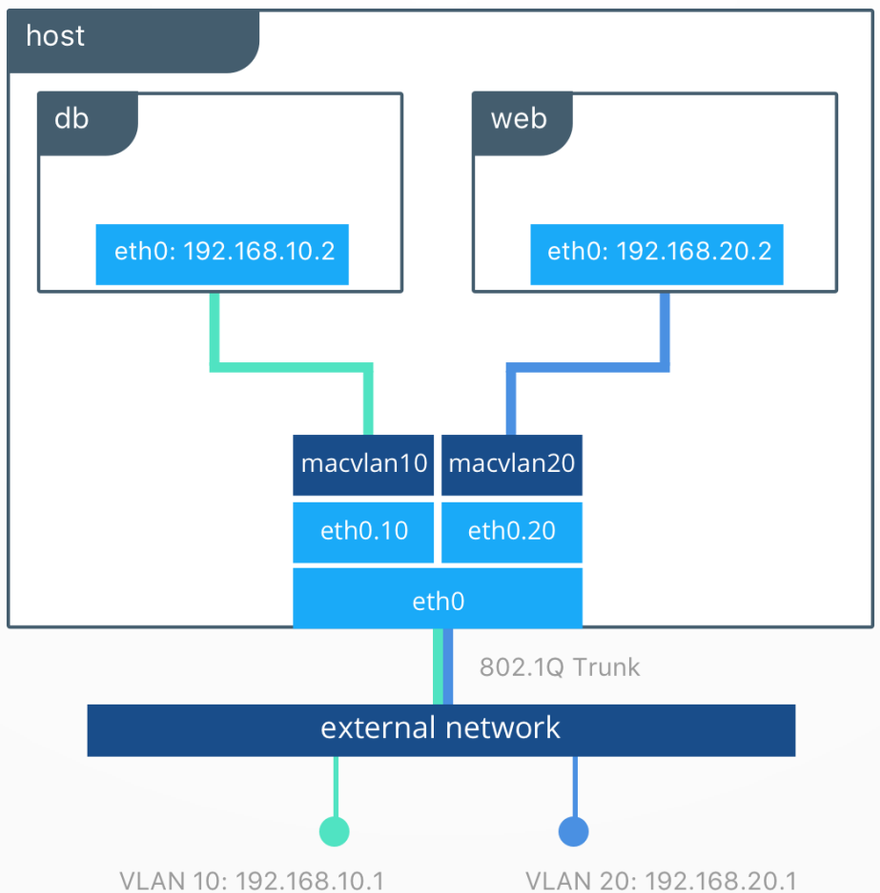
- ***None*** For this container, disable all networking. Usually used in conjunction with a custom network driver.
- ***Host*** For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly.

Network

Type of network

- ***MACVLAN*** Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses. Using the macvlan driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host's network stack.

Network



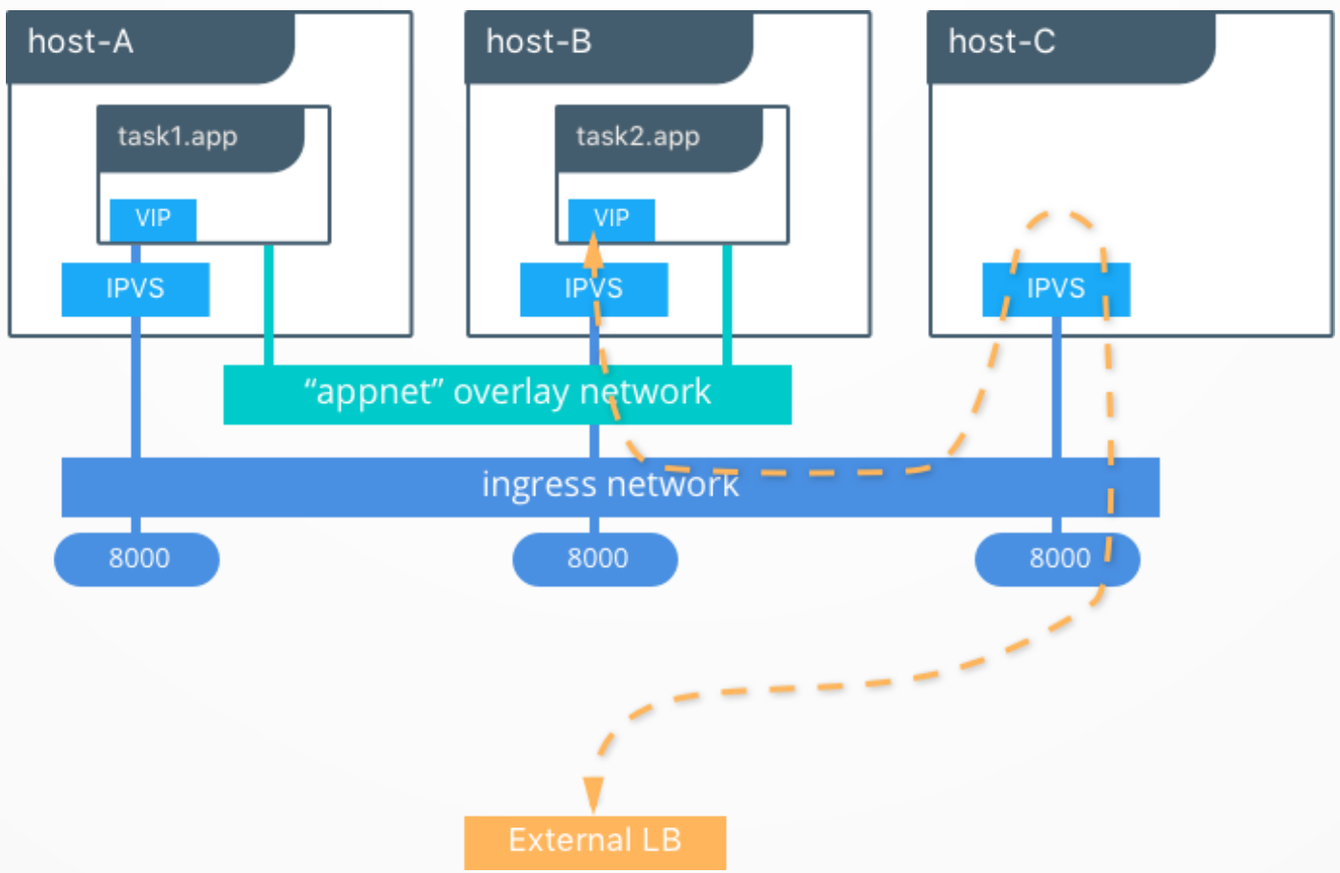
MACVLAN

Network

Type of network

- ***Overlay*** Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers.

Network



Overlay

Network

Docker network commands	
docker network ls	List networks.
docker create create <options> <network>	Create a new network.
docker network rm <networks>	Remove one or more networks.
docker network connect <options> <network> <container>	Connect a container to a network.
docker run <options> --network <network> <image:tag>	Run a container to a network.

Lab 5.1 – Network(bridge)

- Create network private bridge.

```
$ docker network create --driver bridge --
subnet=192.168.101.0/24 --ip-
range=192.168.101.128/25 --
gateway=192.168.101.48 private
```

- Create run reverse proxy.

```
$ docker run -d --net private --ip 192.168.101.80 --
name proxy wearedocker/proxy
```

Lab 5.1 – Network(bridge)

- Create run web 1.

```
$ docker run -d --net private --net-alias site1 --
name web1 wearedocker/nginx:web1
```

- Create run web 2.

```
$ docker run -d --net private --net-alias site2 --
name web2 wearedocker/nginx:web2
```

- Curl URL.

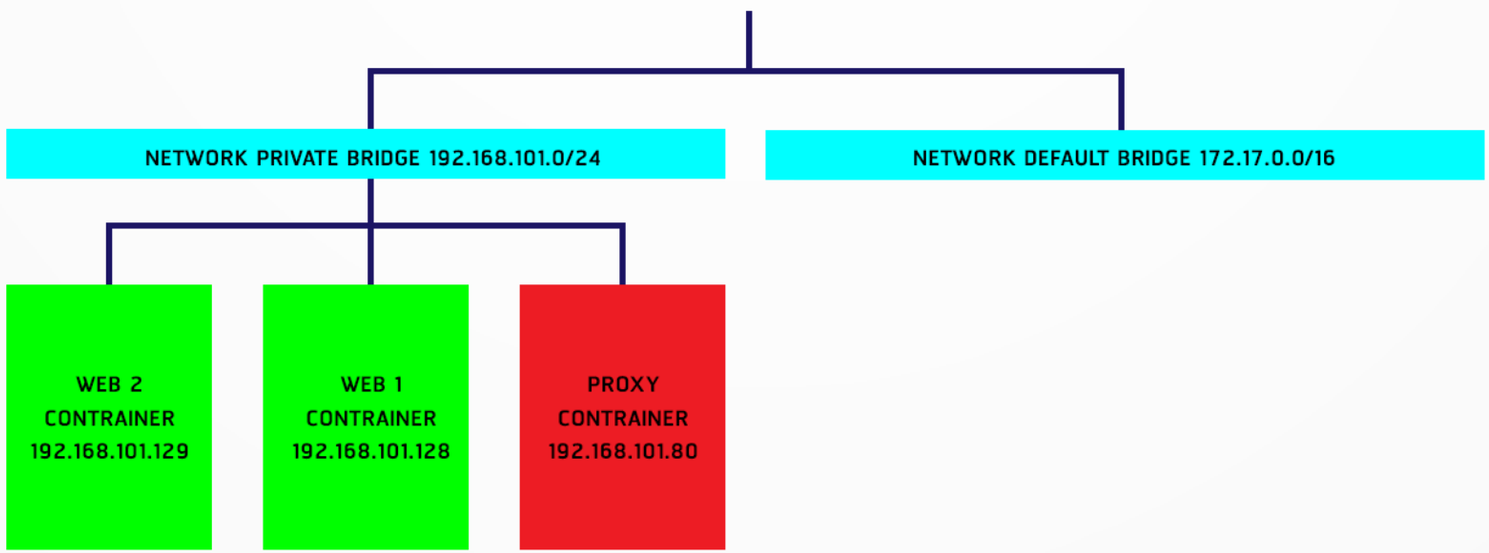
```
$ curl http://192.168.101.80/website
```

Lab 5.1 – Network(bridge)

```
http {
        upstream site {
                server web1;
                server web2;
        }
        server {
                listen 80;
                location /website {
                        proxy_pass http://site/;
                }
        }
}
```

Nginx configuration file

Lab 5.1 – Network(bridge)



Lab 5.2 – Network

- Create network public bridge.

```
$ docker network create --driver bridge --
subnet=192.168.100.0/24 --ip-
range=192.168.100.128/25 --
gateway=192.168.100.84 public
```

- Connect container to network.

```
$ docker network connect --ip 192.168.100.80 /
public proxy
```

Lab 5.2 – Network

- Copy file nginx.conf

```
$ docker cp nginx.conf proxy:/etc/nginx/nginx.conf
```

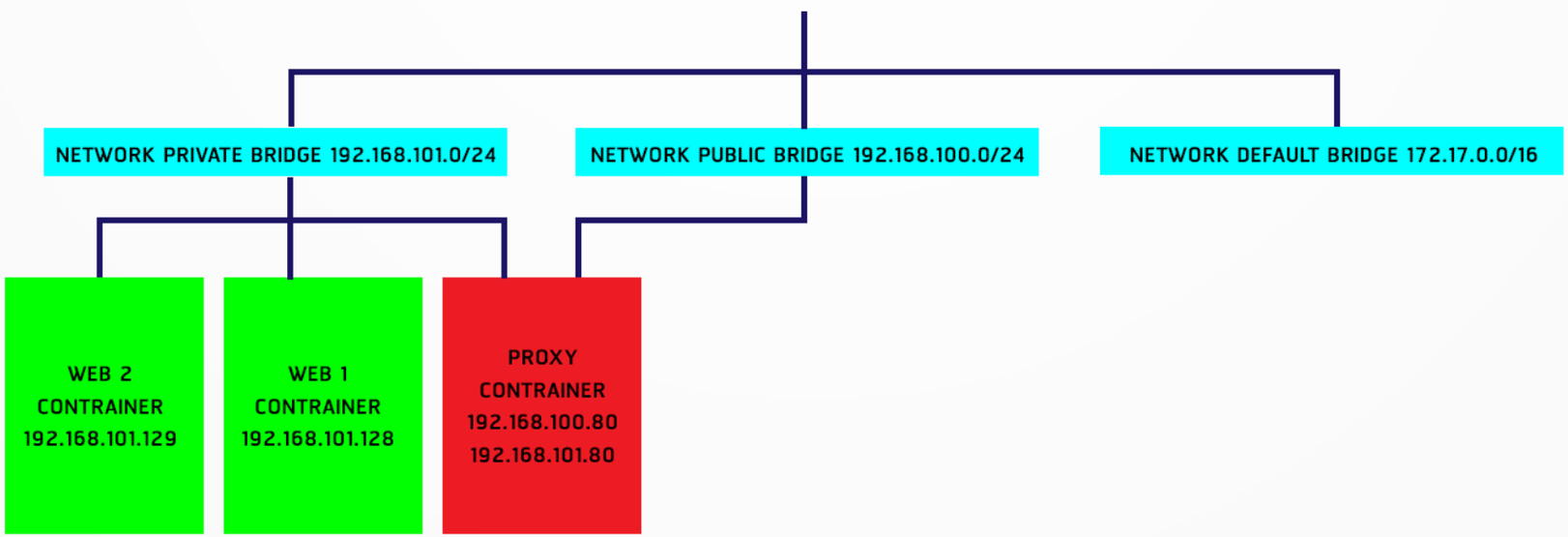
- Restart proxy container

```
$ docker restart proxy
```

- Curl URL.

```
$ curl http://192.168.101.80/website  <  Private IP
$ curl http://192.168.100.80/website  <  Public IP
```

Lab 5.2 – Network



Logs Inspect Stats

Logs Inspect and Stats

- ***Logs***
  Show the logs of a container.
- ***Inspect***
  Display detailed configurations on Containers, Networks and Volumes.
- ***Stats***
  Display resource usage statistics on a live container.

Logs Inspect and Stats

Docker logs and Inspect commands	
docker logs <options> <container>	Fetch the logs of a container.
docker inspect <options> <container>	Return low-level information on Docker objects.
docker network inspect <options> <networks>	Display detailed information on one or more networks.
docker volume inspect <options> <volumes>	Display detailed information on one or more volumes.
docker stats <options> <container>	Display a live stream of container(s) resource usage statistics

Logs Inspect and Stats

Logs

```
docker@labdocker:~$ docker logs proxy
192.168.101.48 - - [20/Oct/2018:07:56:29 +0000] "GET /website HTTP/1.1" 200 17 "-" "curl/7.54.1"
192.168.101.48 - - [20/Oct/2018:07:56:30 +0000] "GET /website HTTP/1.1" 200 17 "-" "curl/7.54.1"
192.168.101.48 - - [20/Oct/2018:07:56:31 +0000] "GET /website HTTP/1.1" 200 17 "-" "curl/7.54.1"
192.168.101.48 - - [20/Oct/2018:07:56:31 +0000] "GET /website HTTP/1.1" 200 17 "-" "curl/7.54.1"
192.168.101.48 - - [20/Oct/2018:07:56:31 +0000] "GET /website HTTP/1.1" 200 17 "-" "curl/7.54.1"
192.168.101.48 - - [20/Oct/2018:07:56:32 +0000] "GET /website HTTP/1.1" 200 17 "-" "curl/7.54.1"
192.168.101.48 - - [20/Oct/2018:07:56:32 +0000] "GET /website HTTP/1.1" 200 17 "-" "curl/7.54.1"
192.168.101.48 - - [20/Oct/2018:07:56:33 +0000] "GET /website HTTP/1.1" 200 17 "-" "curl/7.54.1"
192.168.101.48 - - [20/Oct/2018:07:56:33 +0000] "GET /website HTTP/1.1" 200 17 "-" "curl/7.54.1"
192.168.101.48 - - [20/Oct/2018:07:56:33 +0000] "GET /website HTTP/1.1" 200 17 "-" "curl/7.54.1"
192.168.101.48 - - [20/Oct/2018:07:56:34 +0000] "GET /website HTTP/1.1" 200 17 "-" "curl/7.54.1"
192.168.101.48 - - [20/Oct/2018:07:56:34 +0000] "GET /website HTTP/1.1" 200 17 "-" "curl/7.54.1"
docker@labdocker:~$
```

Logs Inspect and Stats

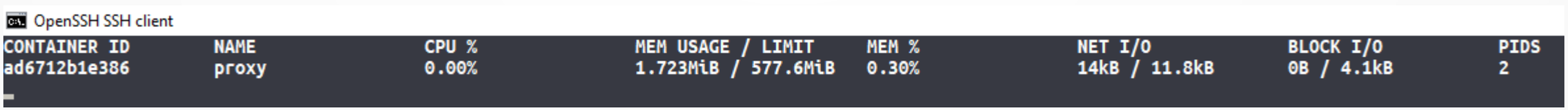
Inspect

OpenSSH SSH client

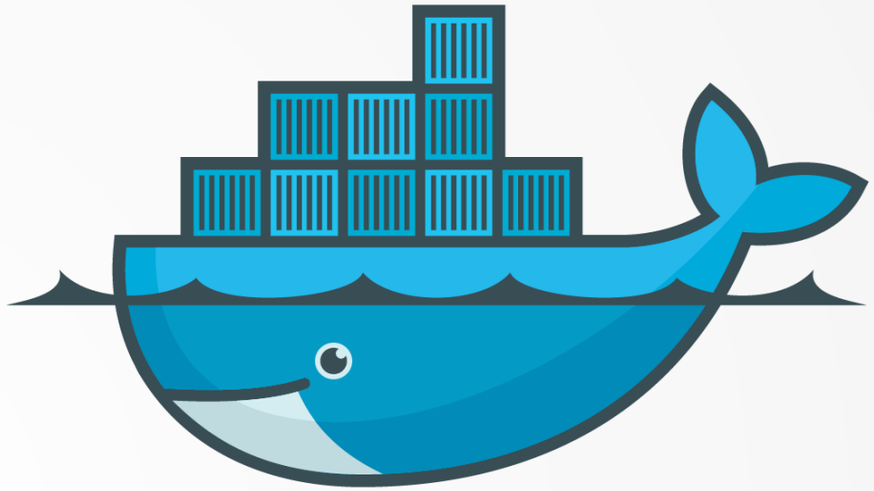
```
docker@labdocker:~$ docker inspect proxy
[
    {
        "Id": "ad6712b1e38658ebc20d4563046c9ac77e4c5aa275c07d6b3b11ec027a0af18f",
        "Created": "2018-10-20T07:55:49.912699774Z",
        "Path": "nginx",
        "Args": [
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 6735,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2018-10-20T07:55:50.158028278Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:b7cd7f81b30715b05352012839e8fa773c9d756dd47f342b1ef7306add35ad55",
        "ResolvConfPath": "/mnt/sda1/var/lib/docker/containers/ad6712b1e38658ebc20d4563046c9ac77e4c5aa275c07d6b3b11ec027a0af18f/resolv.conf",
        "HostnamePath": "/mnt/sda1/var/lib/docker/containers/ad6712b1e38658ebc20d4563046c9ac77e4c5aa275c07d6b3b11ec027a0af18f/hostname",
        "HostsPath": "/mnt/sda1/var/lib/docker/containers/ad6712b1e38658ebc20d4563046c9ac77e4c5aa275c07d6b3b11ec027a0af18f/hosts",
        "LogPath": "/mnt/sda1/var/lib/docker/containers/ad6712b1e38658ebc20d4563046c9ac77e4c5aa275c07d6b3b11ec027a0af18f/ad6712b1e38658ebc20d4563046c9ac77e4c5aa275c07d6b3b11ec027a0af18f-json.log",
        "Name": "/proxy",
        "RestartCount": 0,
        "Driver": "aufs",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
            "NetworkMode": "private",
            "PortBindings": {},
            "RestartPolicy": {
                "Name": "no",
                "MaximumRetryCount": 0
            },
            "AutoRemove": true,
            "VolumeDriver": "",
            "VolumesFrom": null,
            "CapAdd": null,
            "CapDrop": null,
            "Dns": [],
            "DnsOptions": [],
            "DnsSearch": [],
            "ExtraHosts": null,
            "GroupAdd": null,
            "IpcMode": "shareable",
            "Cgroup": "",
            "Links": null,
            "OomScoreAdj": 0,
```

Logs Inspect and Stats

Stats



Questions and Answers

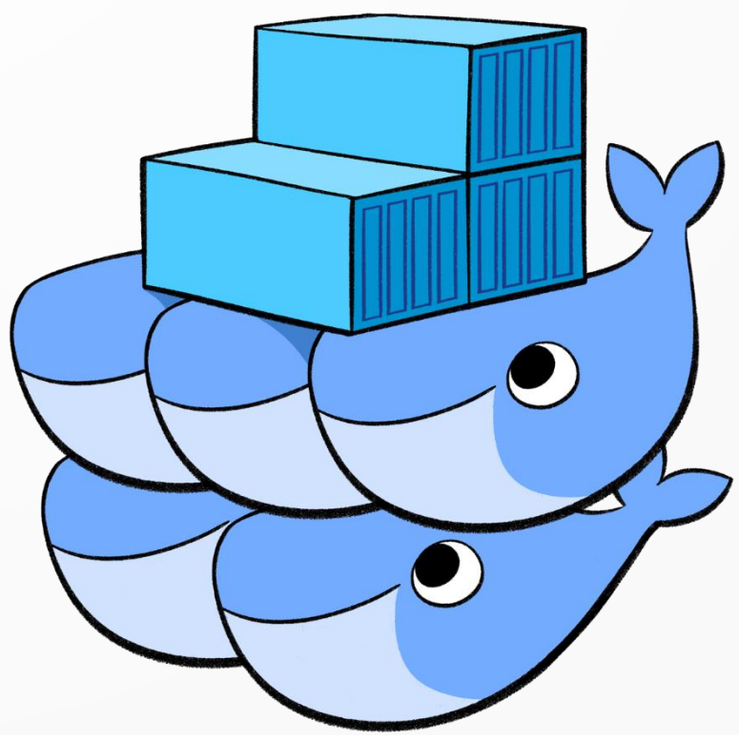


Docker 101: course starter day 2

Course Outline

Day 2

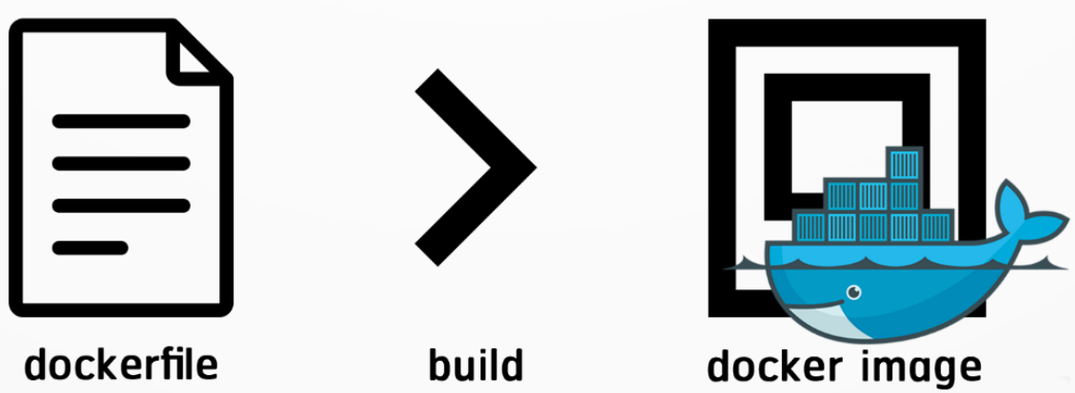
- Dockerfile
- Docker Compose
- Docker Swarm
- Portainer



Dockerfile

Dockerfile

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.



Dockerfile

Dockerfile reference	
FROM <image:tag>	Sets the base image for subsequent instructions.
RUN <command>	Execute any commands in image.
CMD <command> <param1> <param2>	Sets the command to be executed when running the image.
LABEL <key>=<value> ...	Adds metadata to an image.
EXPOSE <port>	Informs Docker that the container listens on the specified network ports at runtime.
ENV <key> <value>	Sets the environment variable.
COPY <src> <dest>	Copies new files from source to the filesystem of the container at the path destinations.
ENTRYPOINT <command> <param1> <param2>	Command line arguments will be appended after all elements in an exec form ENTRYPOINT.
VOLUME ["/data"]	Sets mounted volumes from native host or other containers.
WORKDIR <path>	Sets the working directory for any commands.

Dockerfile

Example Dockerfile.

```
# fetch node v4 LTS codename argon
FROM node:argon

# Request samplename build argument
ARG samplename

# Create app directory
RUN mkdir -p /usr/src/spfx-samples
WORKDIR /usr/src/spfx-samples

#Install app dependencies
RUN git clone https://github.com/SharePoint/sp-dev-fx-webparts.git .
WORKDIR /usr/src/spfx-samples/samples/$samplename

# install gulp on a global scope
RUN npm install gulp -g

# RUN ["npm", "install", "gulp"]
RUN npm install
RUN npm cache clean

# Expose required ports
EXPOSE 4321 35729 5432

# Run sample
CMD ["gulp", "serve"]
```

Lab 6 – Dockerfile

- Dockerfile.

Base Image: alpine
Install Package: node.js
Work Directory: /tmp/run
Copy: copy node.js file to work directory
Command: run script node.js file.

Lab 6 – Dockerfile

- Create python file on docker host.

```
$ vi <filename>.js
```

- Create dockerfile.

```
$ vi dockerfile
```

Lab 6 – Dockerfile

- Build image from docker file.

```
$ docker build -t <imagename:tag> <dockerfile>
```

- Show images list.

```
$ docker image ls
```

- Start container.

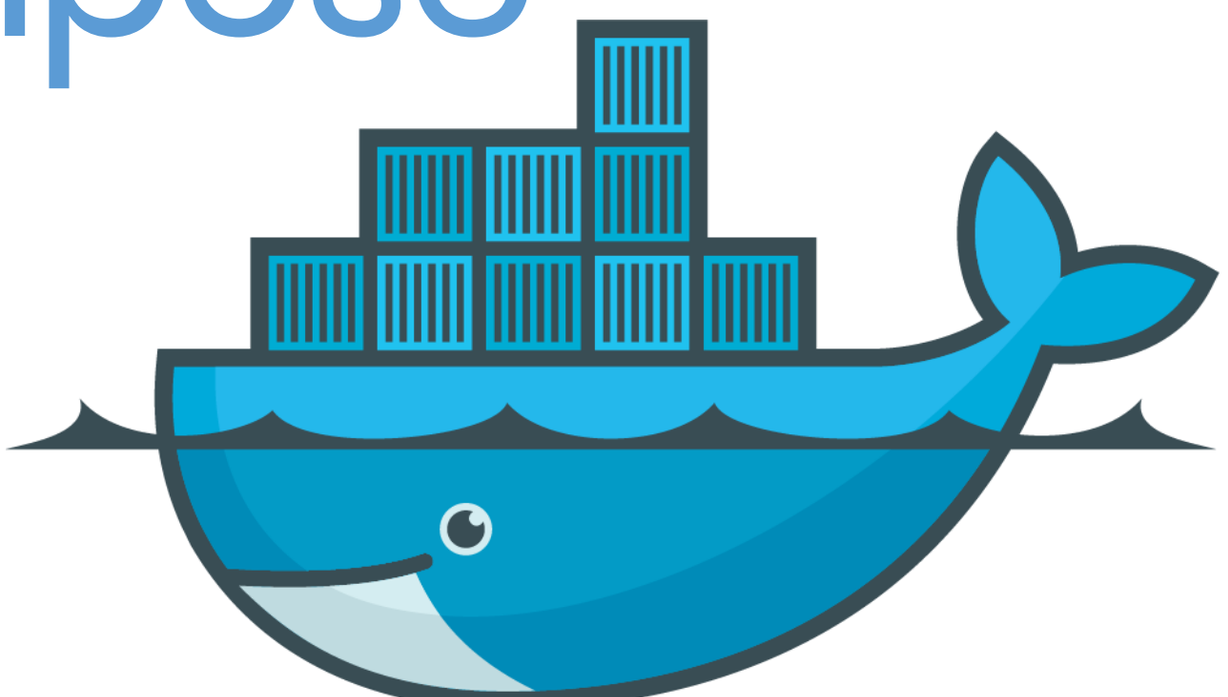
```
$ docker run <imagename:tag>
```

Lab 6 – Dockerfile

- Inside dockerfile

```
FROM alpine
RUN apk update && \
apk add --no-cache nodejs-current
RUN mkdir -p /tmp/run
WORKDIR /tmp/run
COPY <filename>.js .
ENTRYPOINT ["/bin/sh","-c"]
CMD [“node <filename>.js"]
```

Compose

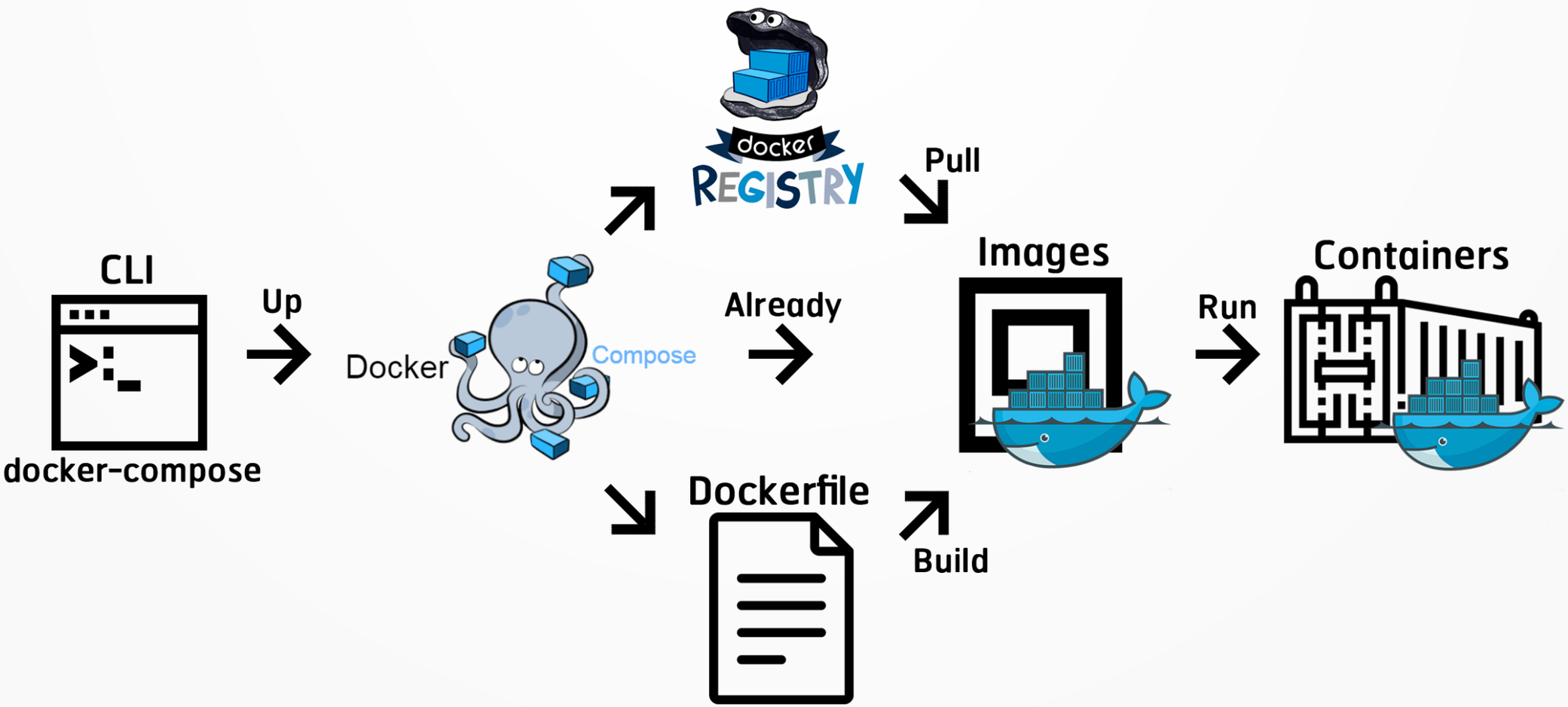


Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

Docker Compose commands	
docker-compose up <options>	Builds, (re)creates, starts, and attaches to containers for a service.
docker-compose down <options>	Stops containers and removes containers, networks, volumes, and images.

Compose


CLI
docker-compose
Up
Docker
Compose
docker
REGISTRY
Pull
Already
Images
Run
Containers
Dockerfile
Build

Compose

Example Docker Compose file.

```yaml
version: '3.7'
services:
  node:
    container_name: node
    build: .
    ports:
      - "8080:8080"
```

Compose

- Download Docker Compose.

```
$ sudo curl -L
"https://github.com/docker/compose/releases/do
wnload/1.22.0/docker-compose-$(uname -s)-
$(uname -m)" -o /usr/local/bin/docker-compose
```

- Apply executable permissions to the binary.

```
$ sudo chmod +x /usr/local/bin/docker-compose
```

Lab 7.1- Compose

- Create docker compose file.

```
$ vi docker-compose.yml
```

- Docker Compose file.

```
version: '3.7'
services:
        node:
                container_name: node
                build: .
                ports:
                  - “8080:8080”
```

Lab 7.1 - Compose

- Start Docker Compose.

```
$ docker-compose up
```

- Open URL in browser.

```
http://192.168.99.100:8080
```

Lab 7.2 - Compose

- Create a docker compose following Lab3.

Services: proxy, web1 and web2

Proxy service has external nginx.conf file.

Web service depend on Proxy service.

IP addr Proxy: 192.168.100.80 and 192.168.101.80

IP addr Web1: 192.168.101.128

IP addr Web2: 192.168.101.129

Lab 7.2 - Compose

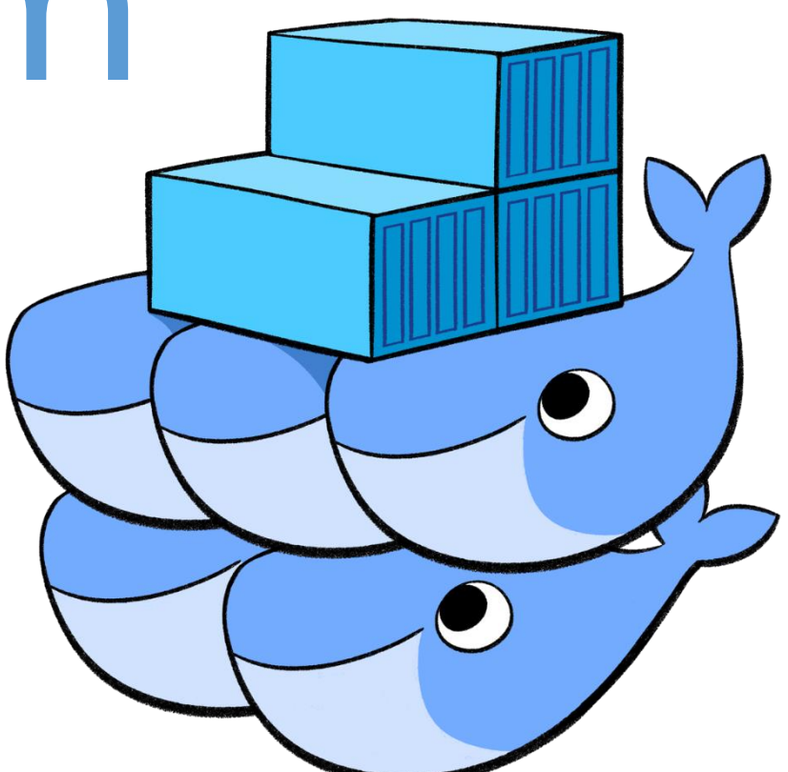
- Start Docker Compose.

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

- Curl Proxy URL.

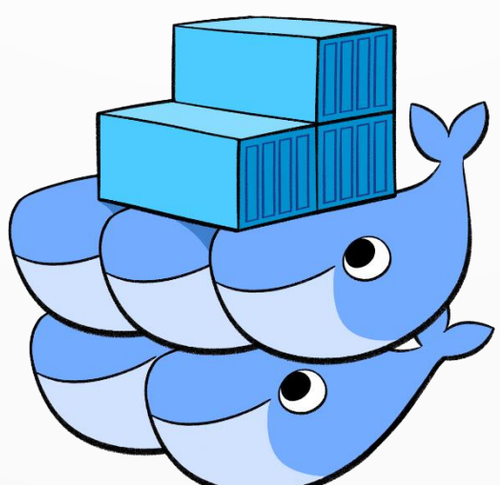
```
$ curl http://192.168.101.80/website
$ curl http://192.168.100.80/website
```

Swarm



What is Swarm?

A swarm consists of multiple Docker hosts which run in swarm mode and act as managers (to manage membership and delegation) and workers (which run swarm services). When you create a service, you define its optimal state (number of replicas, network and storage resources available to it, ports the service exposes to the outside world, and more).



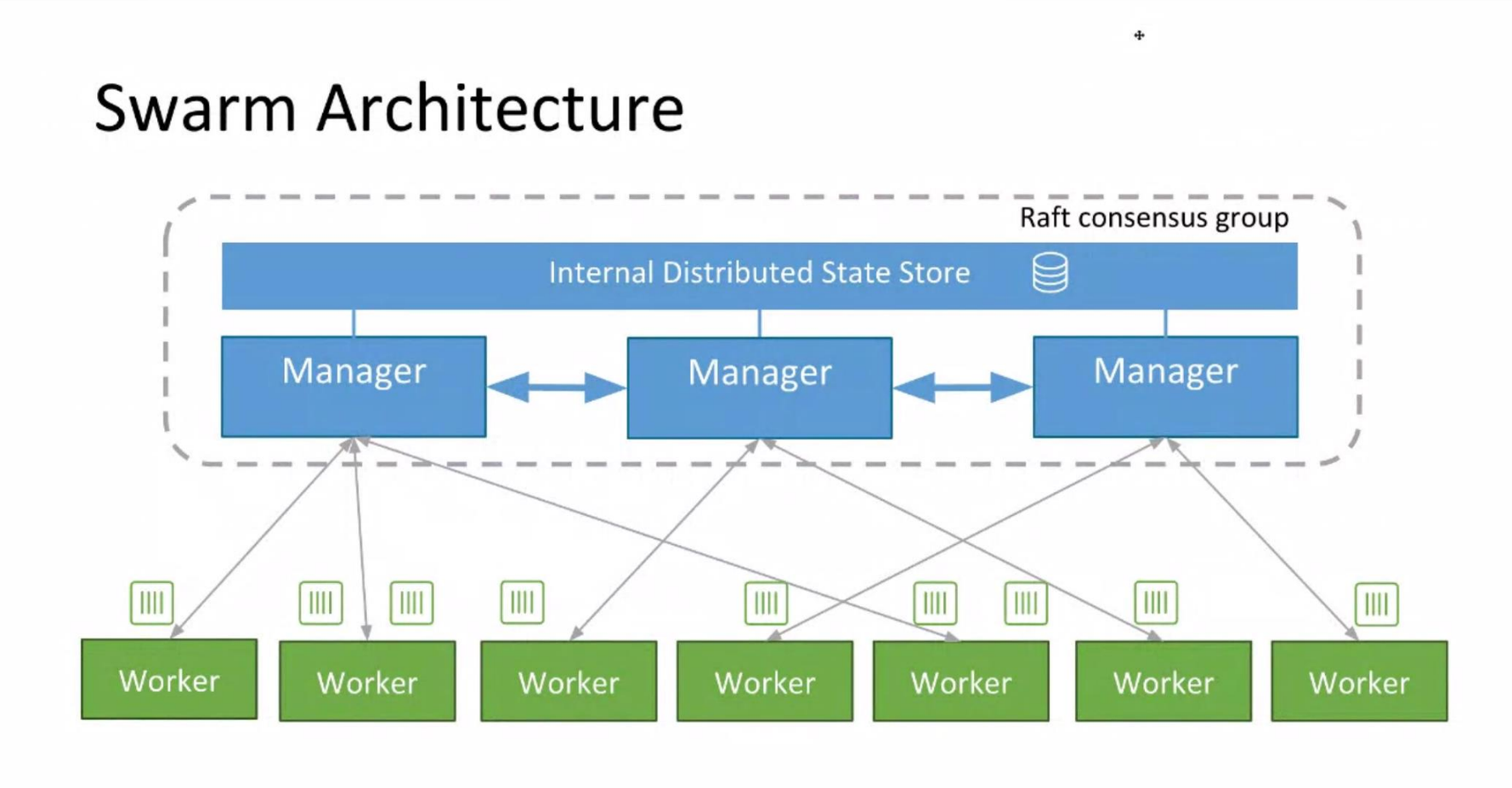
Swarm

Swarm feature

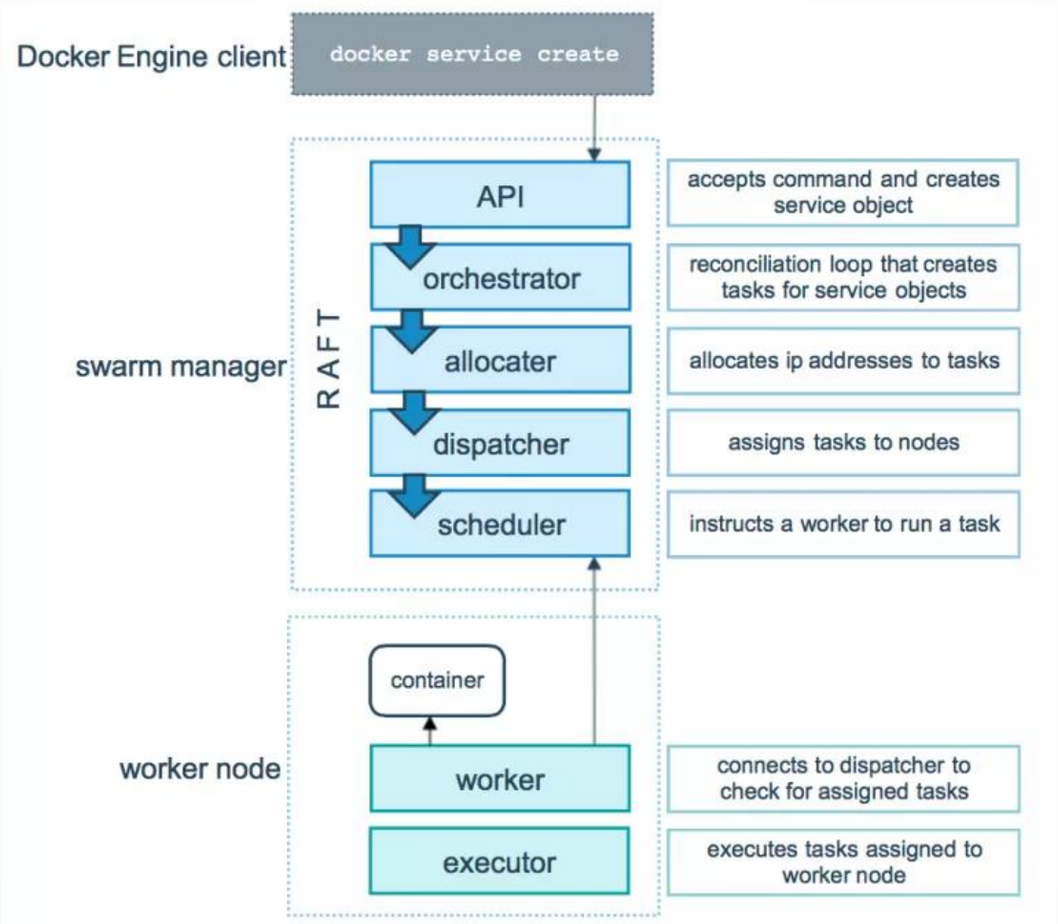
- Cluster management integrated with Docker Engine
- Scaling
- Health Checks on the Containers
- Desired state reconciliation (High Availability)
- Load balancing

Current versions of Docker (version 1.12 and higher) include swarm mode for natively managing a cluster of Docker Engines called a swarm. Use the Docker CLI to create a swarm, deploy application services to a swarm, and manage swarm behavior.

Swarm Architecture



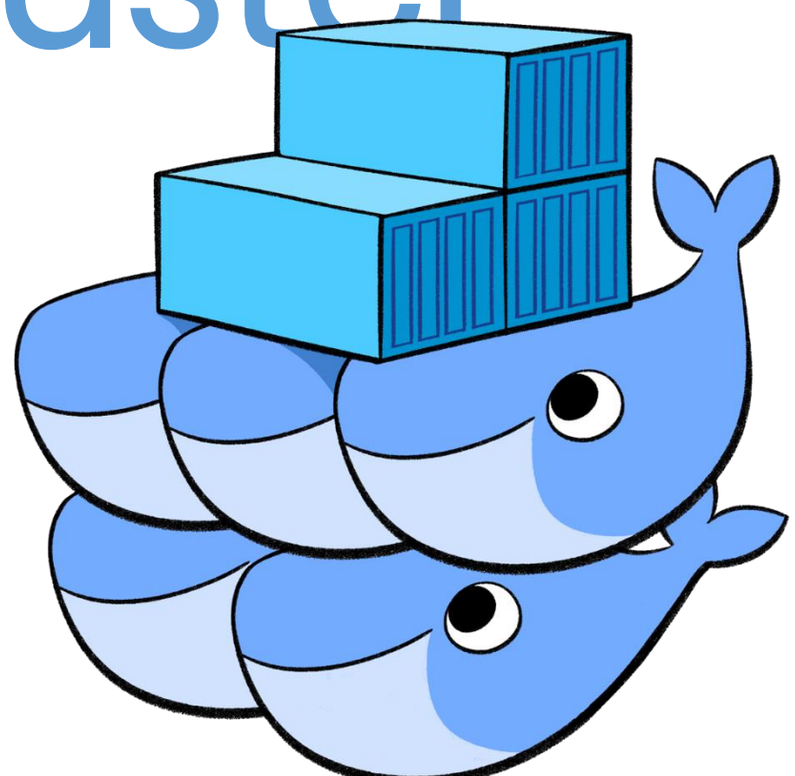
Swarm Architecture



Why do you want a Swarm?

To keep this simple, imagine that you had to run hundreds of containers. You can easily see that if they are running in a distributed mode, there are multiple features that you will need from a management angle to make sure that the cluster is up and running, is healthy and more.

Swarm Cluster



Swarm Cluster

A Swarm based cluster is made up of two main roles: a Swarm manager node and a Swarm worker node.

Create Swarm Cluster

- You can create swarm manager by using command.

```
$ docker swarm init <options>
```

- Output of command has token for add worker node to join swarm and you can run the following command on a host of worker node.

```
$ docker swarm join --token <token key> <IP>
```

Lab - Swarm Cluster

- Create three hosts for a swarm.

```
$ docker-machine create --driver=virtualbox --virtualbox-memory=600 mng
```

```
$ docker-machine create --driver=virtualbox --virtualbox-memory=600 work1
```

```
$ docker-machine create --driver=virtualbox --virtualbox-memory=600 work2
```

Lab - Swarm Cluster

- You can create swarm manager.

```
$ docker swarm init --advertise-addr 192.168.99.100
```

Output

```
docker swarm join --token SWMTKN-1-
32fvxu3gsb8vgzlj5fvdboo4fr96h2y1q74qpgxve5mtj
lel7t-0nk4nuliteye1ma51csr8badm
192.168.99.100:2377
```

Lab - Swarm Cluster

- Add a node workers to the Swarm Cluster.

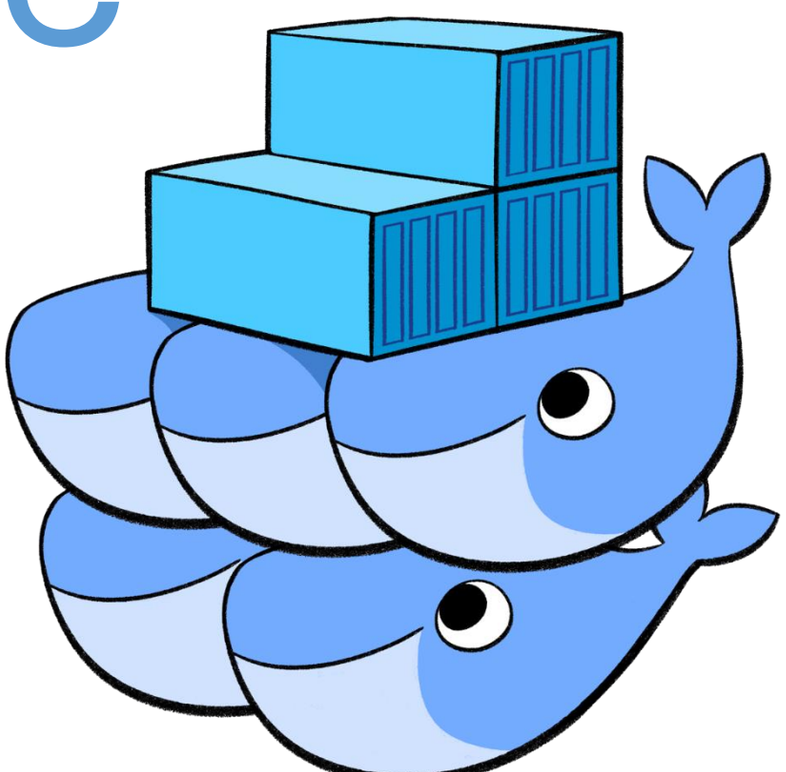
```
$ docker swarm join --token SWMTKN-1-
32fvxu3gsb8vgzlj5fvdboo4fr96h2y1q74qpgxve5mtj
lel7t-0nk4nuliteye1ma51csr8badm
192.168.99.100:2377
```

Lab – Swarm Cluster

- List node of swarm.

```
$ docker node ls
```

Service



Service

With Docker Swarm Mode, a service is a long-running Docker container that can be deployed to any node worker. It's something that either remote systems or other containers within the swarm can connect to and consume.

Service

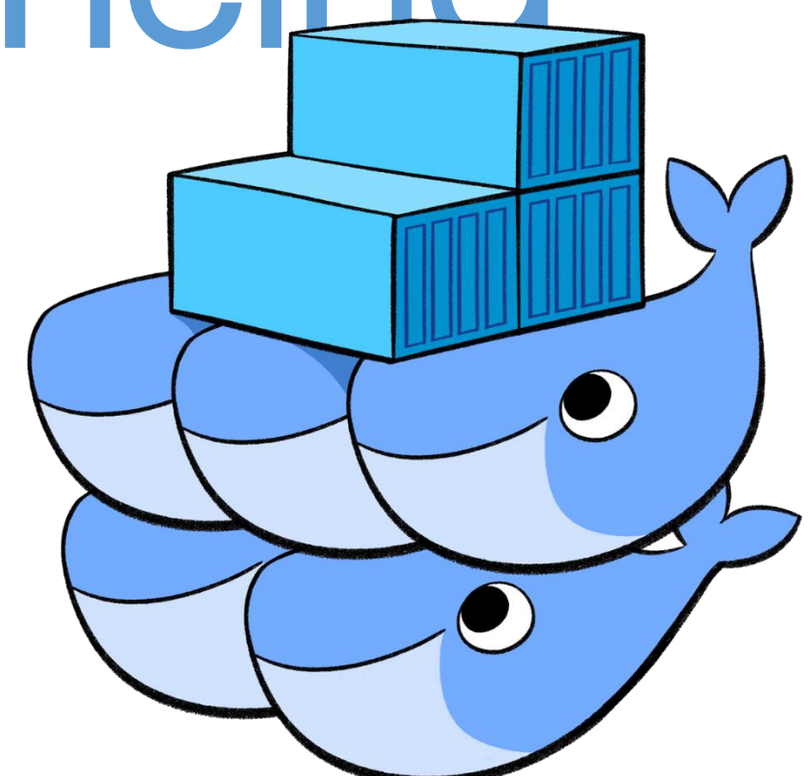
Command service deployment.

```
$ docker service create <options> <image>
```

Command stop service.

```
$ docker service rm <service name>
```

Load Balancing



Load Balancing

Swarm assigns services to underlying nodes and optimizes resources by automatically scheduling container workloads to run on the most appropriate host.

Lab – Load Balance

- (Manager) Create service on cluster.

```
$ docker service create -p 8080:8080 --name hello wearedocker/goapp
```

- (Manager) List of service.

```
$ docker service ps hello
```

- (Manager) Curl test.

```
$ curl localhost:8080
```

Lab – Load Balance

- (Manager) Scale up service.

```
$ docker service scale hello=3
```

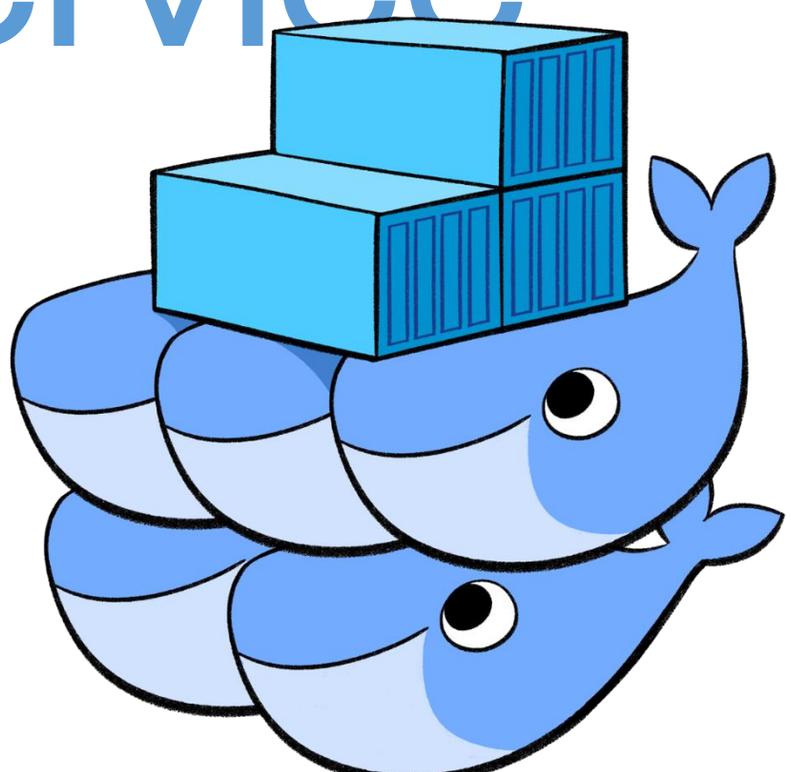
- (Manager) List of service.

```
$ docker service ps hello
```

- (Manager) Curl test.

```
$ curl localhost:8080
```

Control Service



Control Service

Swarm services provide a few different ways for you to control scale and placement of services on different nodes.

- Docker stack deploy (replicated number of container by manual).
- Assign by default node constrain.
- Assign by service placement.

Lab – Node Constraint

- (Manager) Create service by constraint.

```
$ docker service create --constraint 'node.hostname==work1' --name web --replicas 2 -p 80:80 wearedocker/goapp
```

- (Manager) List of service.

```
$ docker service ps web
```

Lab – Node Label

- (Manager) Add label on Workers node.

```
$ docker node update --label-add 'zone=tester' work1
```

```
$ docker node update --label-add 'zone=tester' work2
```

- (Manager) Check label has been added.

```
$ docker node inspect work1
```

Lab – Node Label

- (Manager) Check label has been added.

```
$ docker node inspect work2
```

- (Manager) Create service on label zone tester.

```
$ docker service create --constraint
'node.labels.zone==tester' --name web --replicas 2
-p 8080:8080 wearedocker/goapp
```

Lab – Node Label

- (Manager) List service on cluster.

```
$ docker service ps web
```

Lab – Spread Service

- (Manager) Add label on Manager node.

```
$ docker node update --label-add 'zone=ops' mng
```

- (Manager) Create service and spread by label zone.

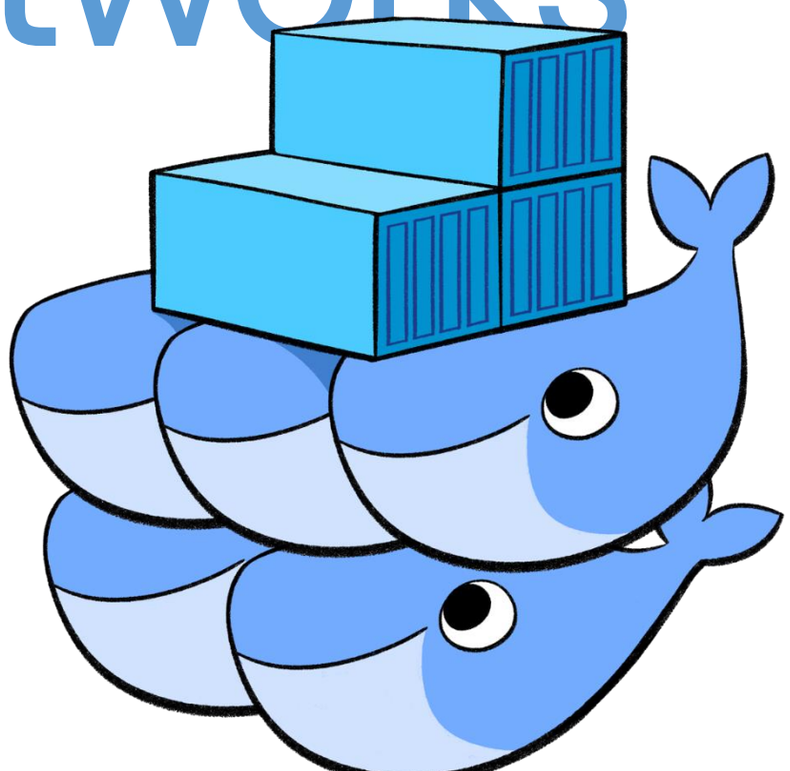
```
$ docker service create --placement-pref
'spread=node.labels.zone' --name web --
replicas=10 wearedocker/goapp
```

Lab – Spread Service

- (Manager) List service on cluster.

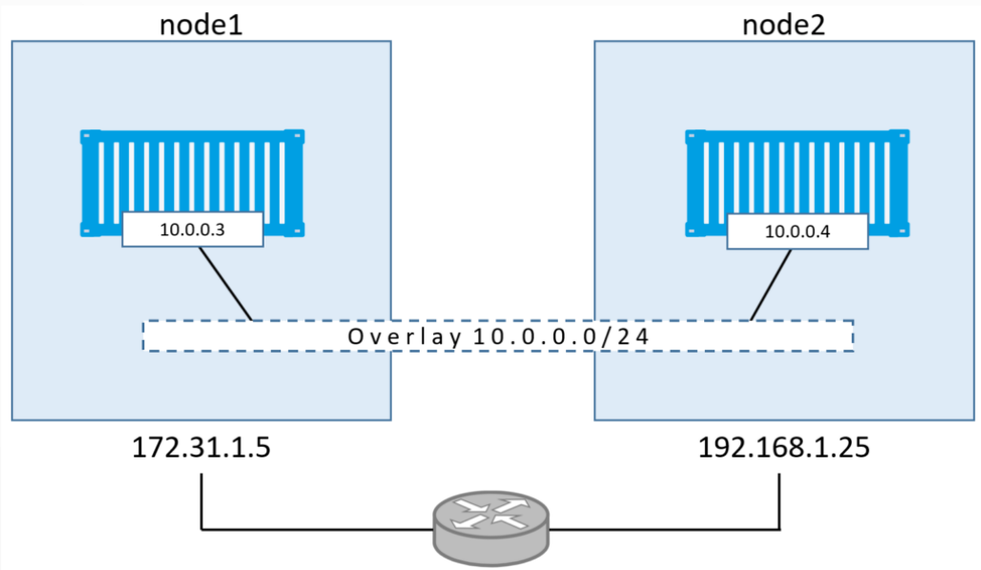
```
$ docker service ps web
```

Overlay Networks



Overlay Networks

The overlay network driver creates a distributed network among multiple Docker daemon hosts. This network sits on top of (overlays) the host-specific networks, allowing swarm service containers connected to it to communicate securely.

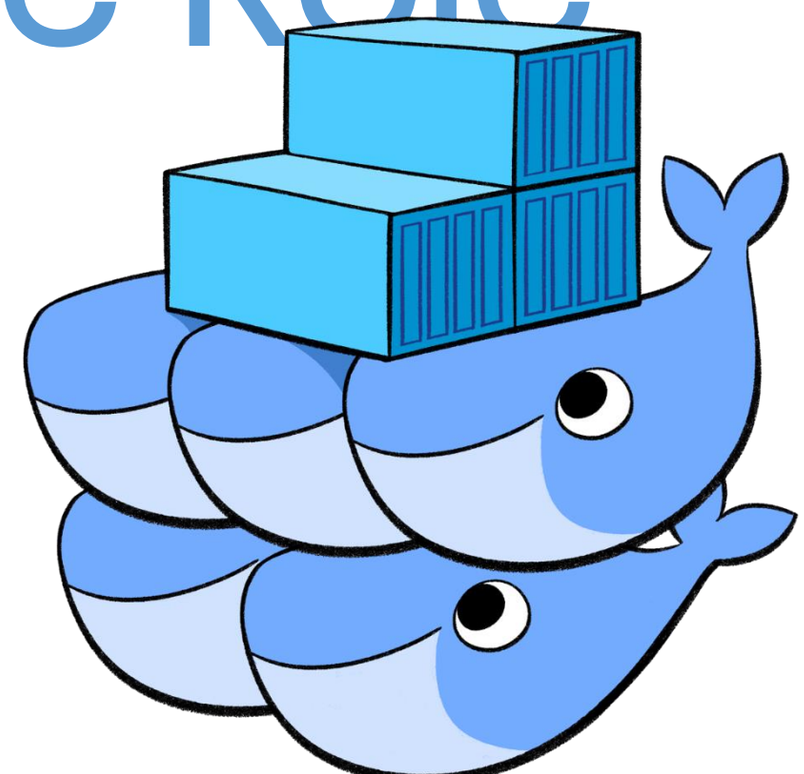


Lab – Overlay Networks

- (Manager) Create Overlay network.

```
$ docker network create --driver overlay --subnet=192.168.100.0/24 nwswarm
```

HA Manage Role



HA Manage role

Add redundancy swarm manage in swarm cluster by command

```
$ docker node update --role manage <node name>
```

Remove role manage in swarm cluster by command

```
$ docker node update --role woker <node name>
```

Lab – HA Manage Role

- (Manager) Add manager role to worker 1.

```
$ docker node update --role manager work1
```

- (Manager) List node on cluster.

```
$ docker node ls
```

- (Manager) Shut down server.

```
$ sudo shutdown –h now
```

Lab – HA Manage Role

- (Worker 1) List service on cluster.

```
$ docker node ls
```

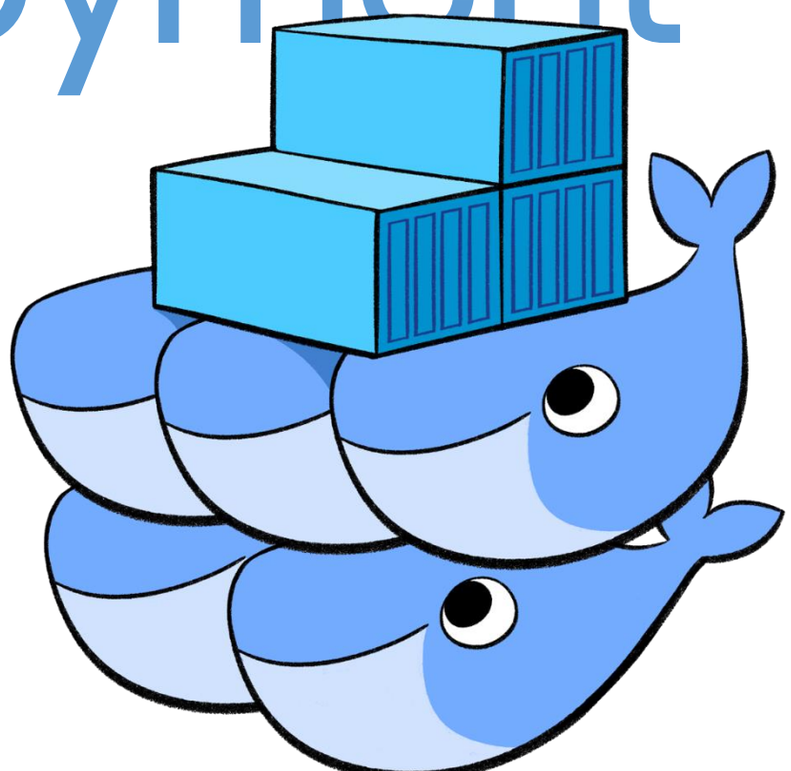
- (Host) Start manager node.

```
$ docker-machine start mng
```

- (Manager) Remove manager role from Worker 1.

```
$ docker node update --role worker work1
```

Stack Deployment



Swarm Stack

Swarm stack command

```
$ docker stack deploy -c <compose> <stack name>
```

```
$ docker stack ls
```

```
$ docker stack ps <stack name>
```

```
$ docker stack service <stack name>
```

```
$ docker stack rm <stack name>
```

Lab – Swarm Stack

- (Manager) Deploy Docker Compose file.

```
$ docker stack deploy -c docker-compose.yml stackgoapp
```

- (Manager) List stack.

```
$ docker node ls
```

- (Manager) List the tasks in the Stack.

```
$ docker stack ps stackgoapp
```

Lab – Swarm Stack

- (Manager) Scale up service in stack.

```
$ docker service scale stackgoapp_balanceweb=3
```

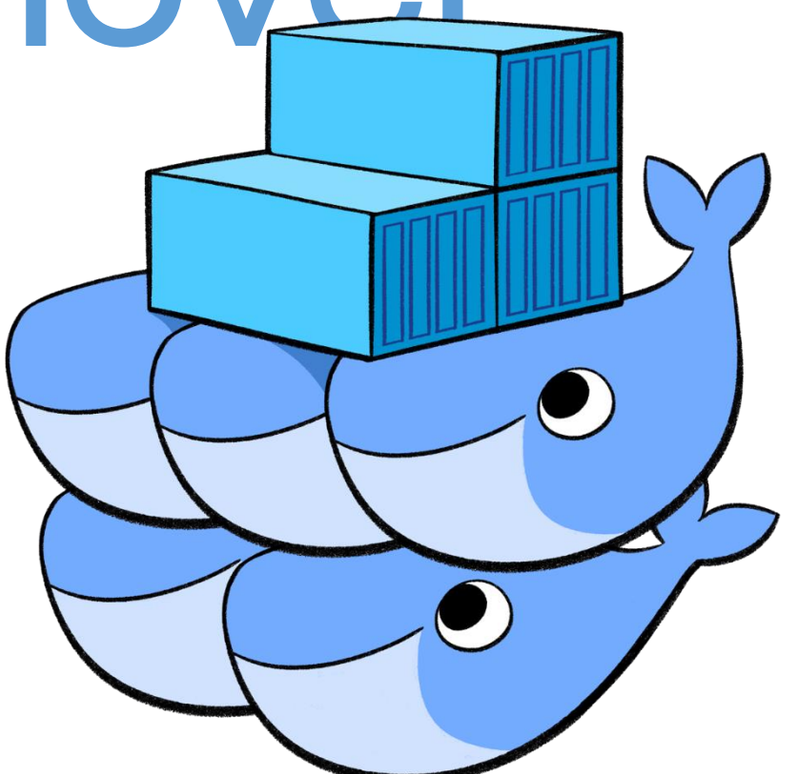
- (Manager) List service.

```
$ docker docker service ps
stackgoapp_balanceweb
```

- (Manager) Remove swarm stack.

```
$ docker rm stackgoapp
```

Node Remover



Lab – Remove Node

- (Manager) Update Availability Worker 2 to drain.

```
$ docker node update --availability drain work2
```

- (Worker 2) Leave Work2 from Swarm Cluster.

```
$ docker swarm leave
```

Lab – Remove Node

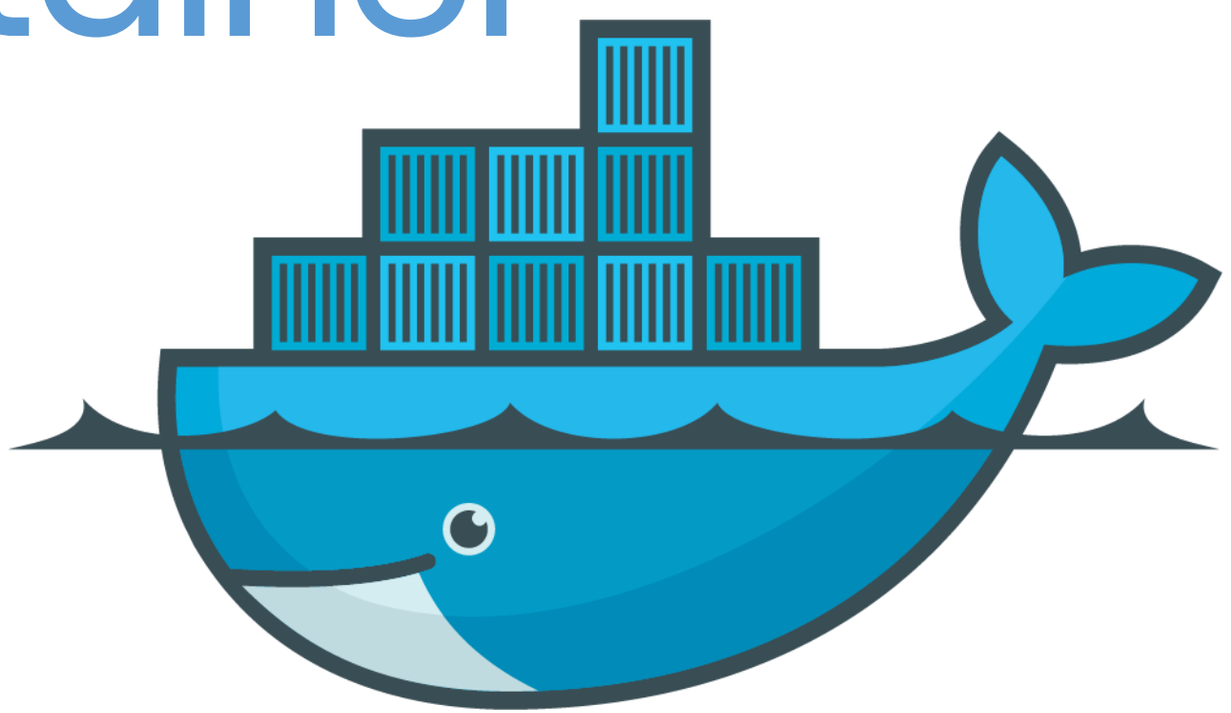
- (Manager) List node of Swarm Cluster.

```
$ docker node ls
```

- (Manager) Remove worker 2 node when worker 2 status is down.

```
$ docker node rm work2
```

Portainer



Portainer

Portainer gives you access to a central overview of your Docker host or Swarm cluster. From the dashboard, you can easily access any manageable entity.

Portainer runs as a lightweight Docker container (the Docker image weights less than 4MB) on a Docker engine or Swarm cluster. Therefore, you are one command away from running container on any machine using Docker.

Portainer

Installation Portainer command

- Create volume for Protainer.

```
$ docker volume create portainer_data
```

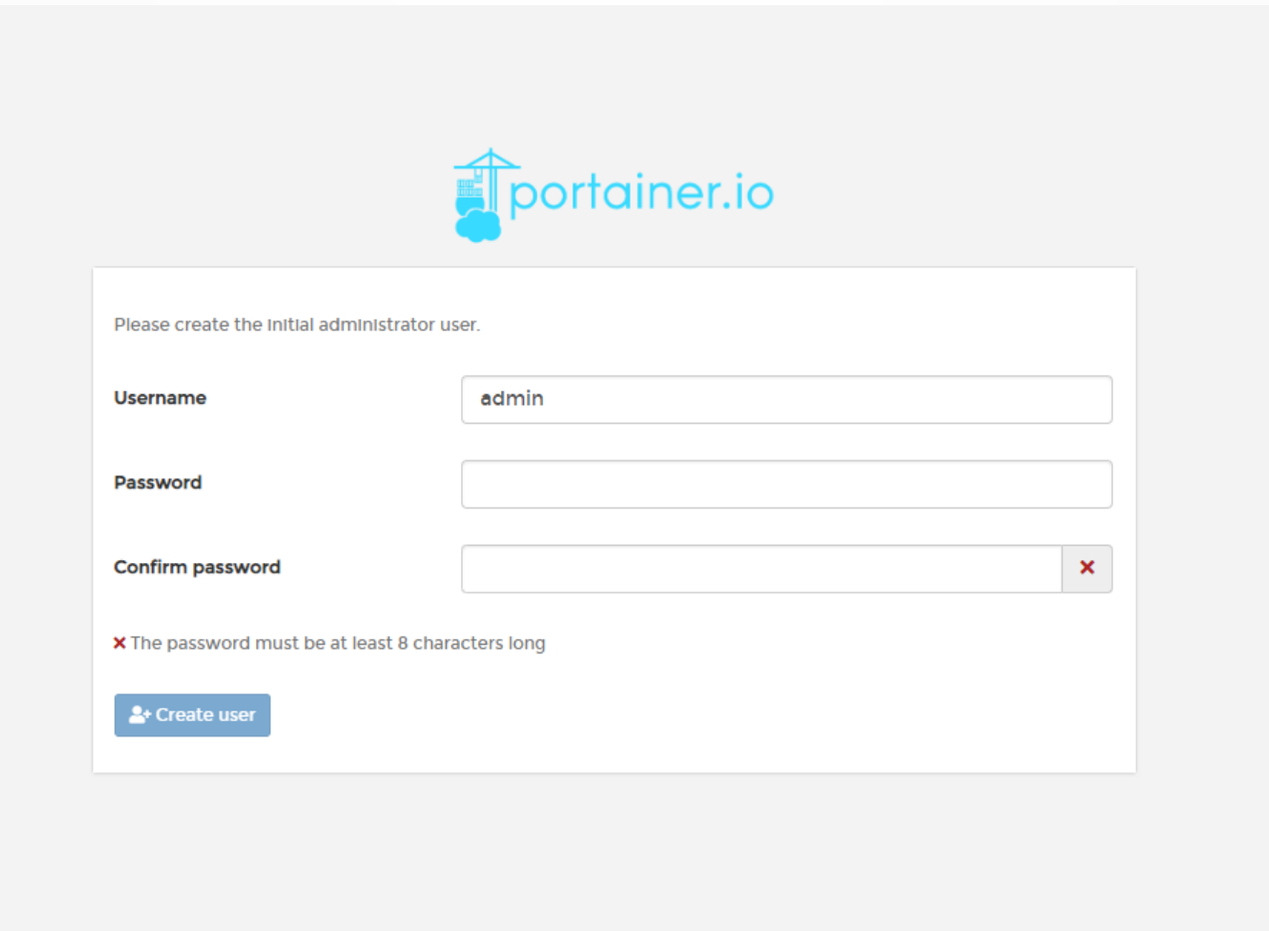
- Pull and run Portainer contrainer.

```
$ docker run -d -p 9000:9000 -v
/var/run/docker.sock:/var/run/docker.sock -v
portainer_data:/data portainer/portainer
```

- Open 192.168.99.100:9000

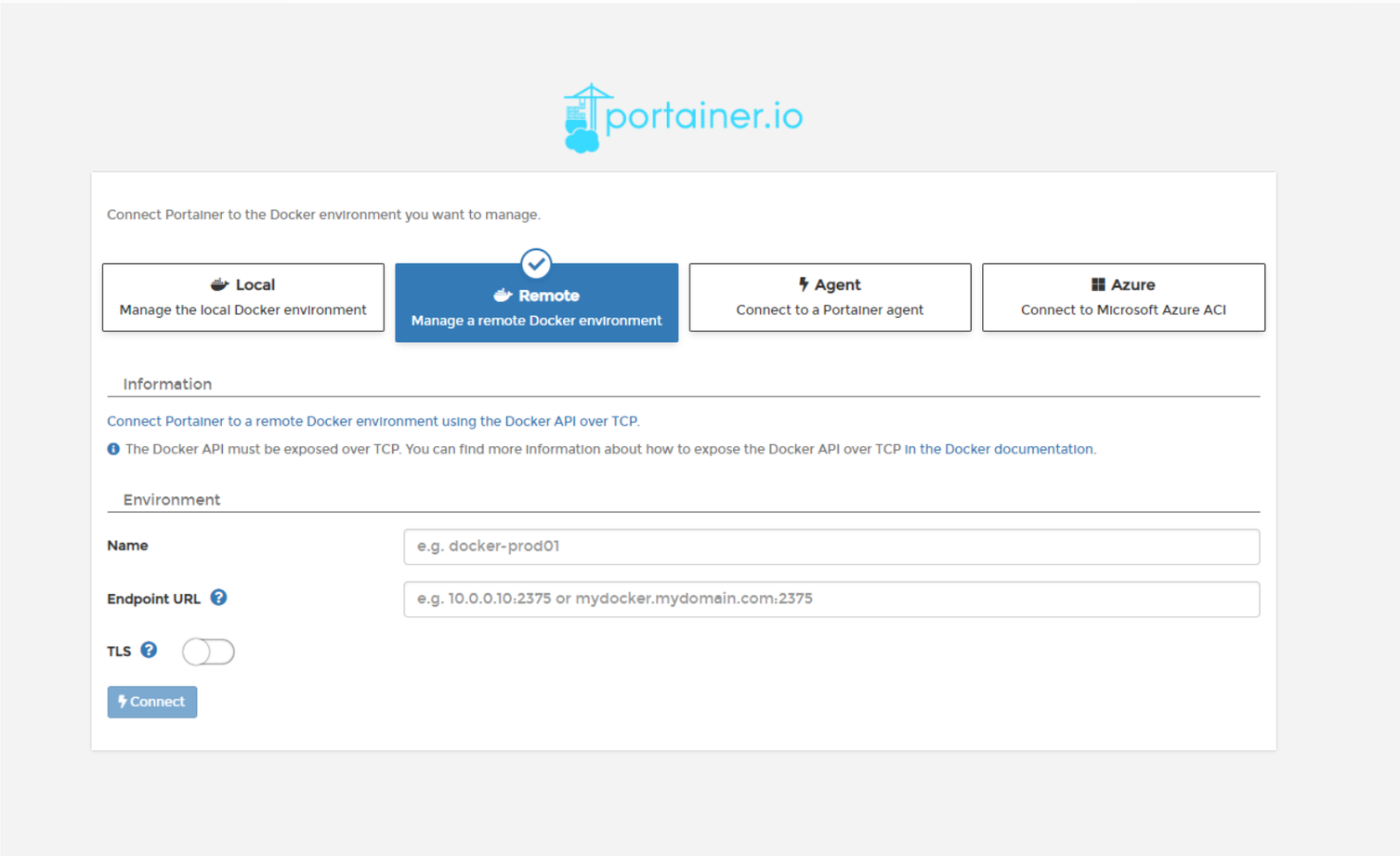
Portainer

Register page



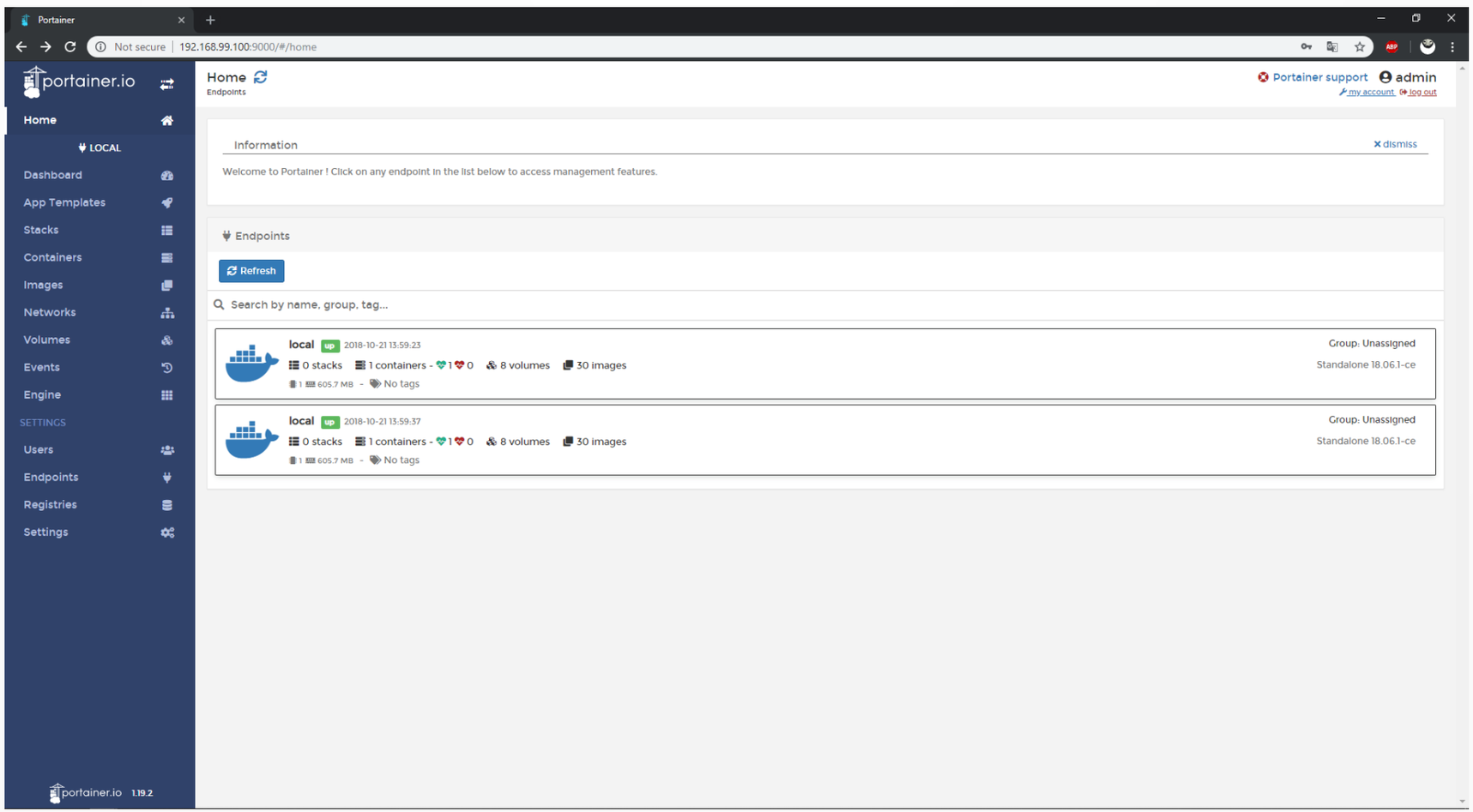
Portainer

Connect Docker environment



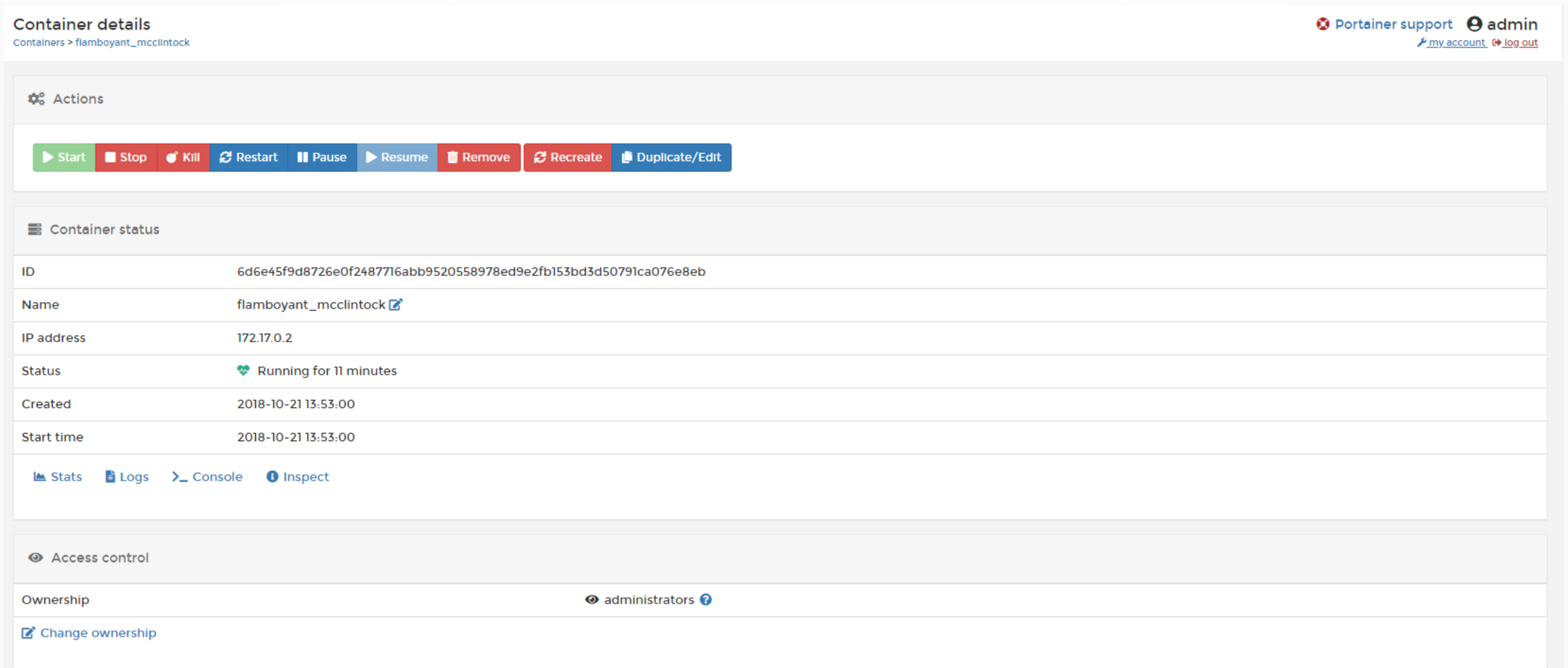
Portainer

Home page



Portainer

Container management



Thank you