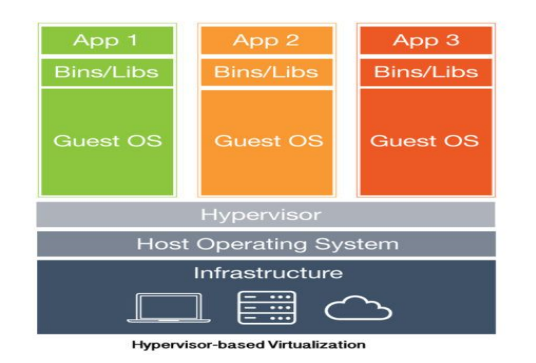
AXON ACTIVE
www.axon.vn

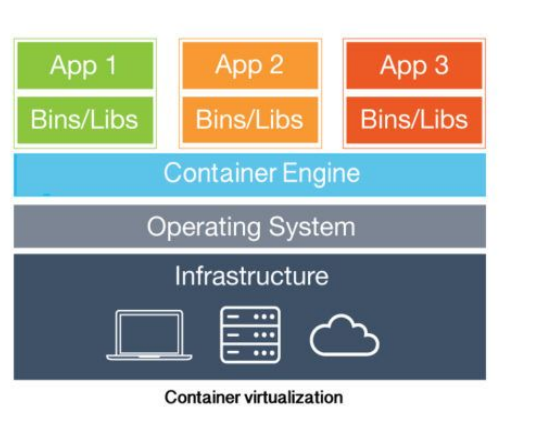
Mastering Docker

Agenda

- Docker
- Dockerfile
- CLI
- Volumes and networking
- Best practice
- Benefit
- Challenge

Virtualization vs Containerization

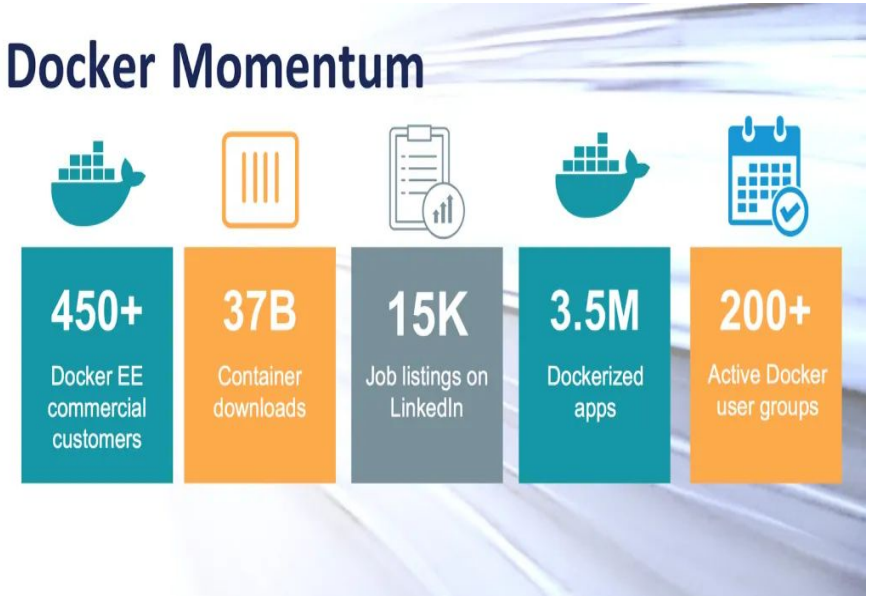




- **Virtualization (virtual machine):** are an abstraction of physical hardware turning one server into many servers. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries.

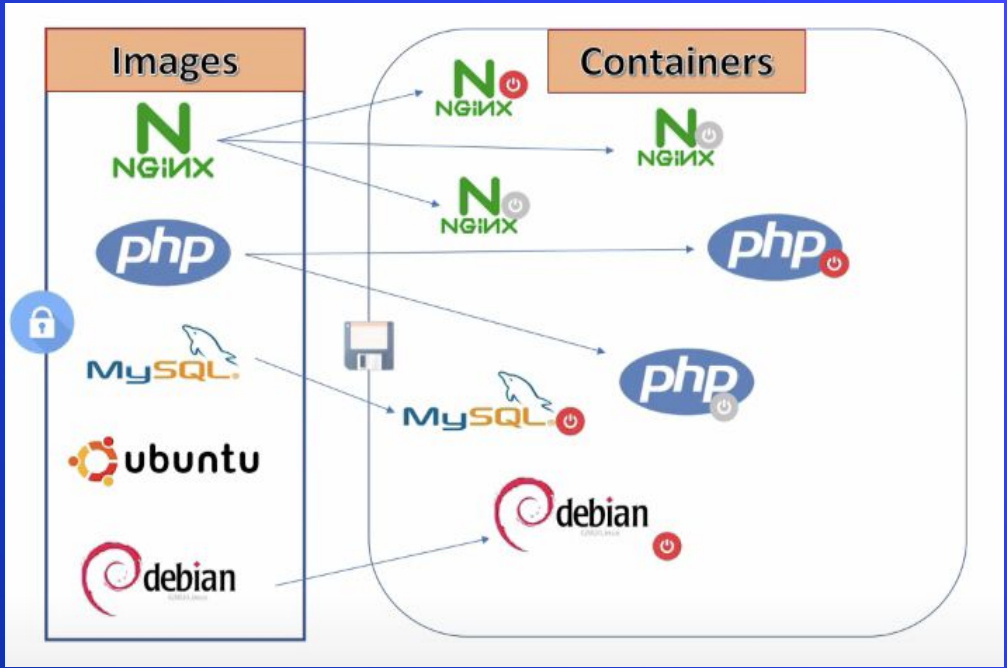
- **Containerization** : are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers

What is docker?



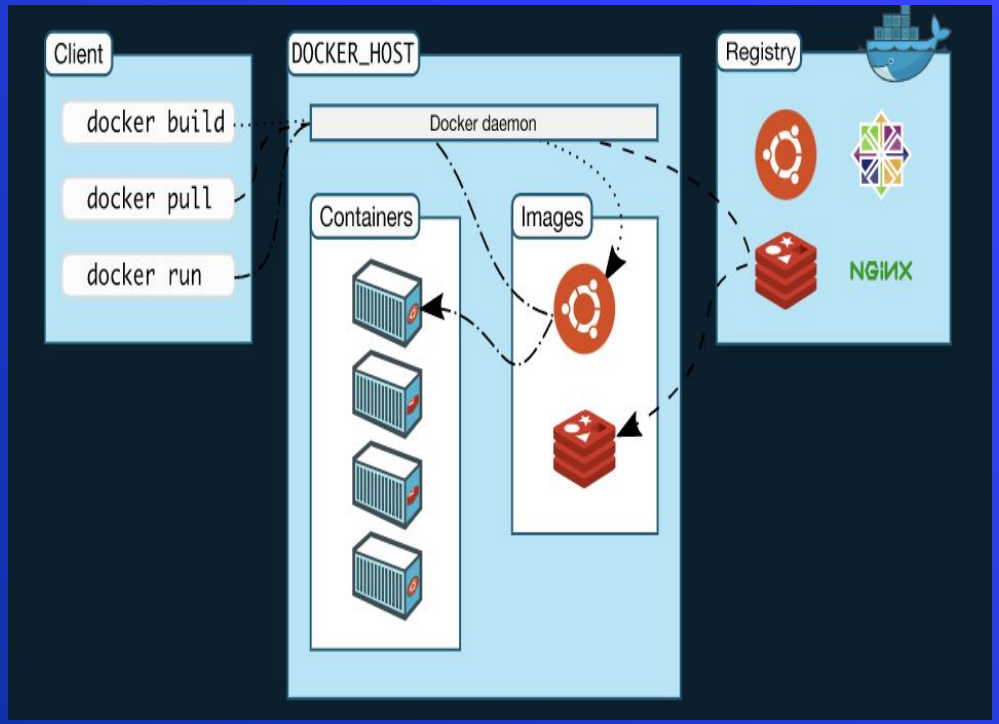
Docker is a famous container provider which is an open platform for developing, shipping and running applications.

Docker objects



- **Image**: a read-only template with instructions for creating a Docker container. To build an image, you need to create a Docker file.
- **Container**: a runnable instance of an image.

Docker architecture



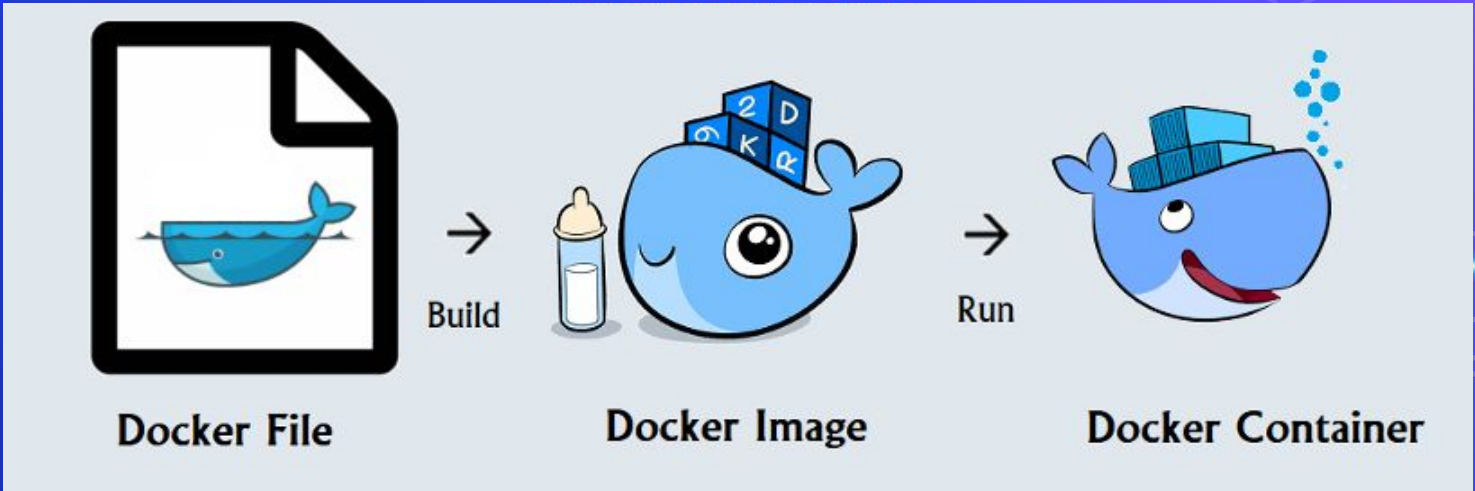
- **Use client - server architecture**.
- **Docker client**: interact with docker via CLI. It uses Docker API to send request to Docker Daemon.
- **Docker daemon**: listens for Docker API requests and manages Docker objects such as images, containers, networks and volume.
- **Docker registries**: a storage and content delivery system, holding named Docker images, available in different tagged versions

Dockerfile

The docker images creating process

What is Dockerfile

- Text document contains commands.



Dockerfile instructions

FROM *FROM ubuntu:18.04*	Define the base image
LABEL *LABEL maintainer=”dev”*	Adds metadata to an image
ENV *ENV myName John Doe*	Sets the environment variable <key> to the value <value>
ARG *ARG version=8.3.0*	Defines a variable pass at build-time to the builder.
WORKDIR *WORKDIR /opt/tomcat*	Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions
RUN *RUN mkdir myApp1*	Execute any commands (shell form/exec form).

Dockerfile instructions

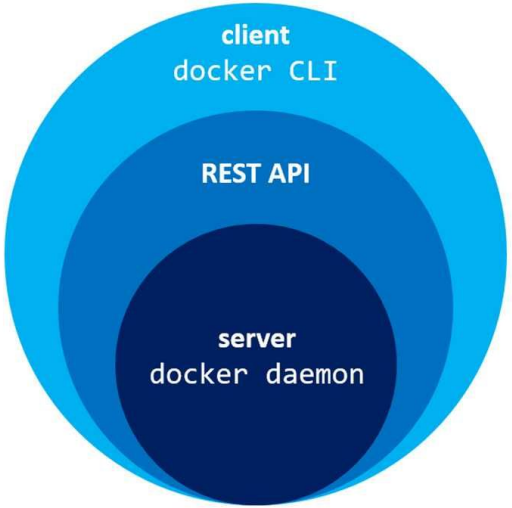
COPY *COPY test.txt myApp1*	Copies new files or directories from <src> host to the filesystem of the container at the path <dest>
ADD *ADD myApp1.tar.gz .*	Same with COPY with tar extraction and remote URL support.
CMD *CMD ["/opt/tomcat/bin/catalina.sh", "run"]*	Provide defaults for an executing container.
ENTRYPOINT *ENTRYPOINT ["docker-entrypoint.sh"]*	Same with CMD, allows to specify a command with parameters.
EXPOSE *EXPOSE 80*	Informs Docker that the container listens on the specified network ports at runtime
VOLUME *VOLUME /myvol*	Creates a mount point with the specified name and marks it as holding externally mounted volumes from native host.

Dockerfile example

```
FROM centos:7
ARG maintainer
LABEL maintainer=${maintainer:-"Axon Developer"}
RUN mkdir /opt/tomcat/
WORKDIR /opt/tomcat
ADD http://mirrors.viethosting.com/apache/tomcat/tomcat-8/v8.5.57/bin/apache-tomcat-8.5.57.tar.gz .
RUN tar xfz apache*.tar.gz
RUN mv apache-tomcat-8.5.57/* /opt/tomcat/
RUN yum -y -q install java
EXPOSE 8080
CMD ["/opt/tomcat/bin/catalina.sh", "run"]
```

Docker CLI

The main CLI for Docker, includes all docker commands



Docker image commands

Build an image from the Dockerfile in the current directory and tag the image

```
docker build -t myimage:1.0 .
```

List all images that are locally stored with the Docker Engine

```
docker image ls
```

Delete an image from the local image store

```
docker rmi myimage:1.0
```

Docker container commands

Run a container from your web app image version 3.9 image, name the running container “web” and expose port 5000 externally, mapped to port 80 inside the container.

```
docker container run --name web -p 5000:80 web-app-image:3.9
```

List the running containers (add --all to include stopped containers)

```
docker container ls
```

Print the last 100 lines of a container’s logs

```
docker container logs --tail 100 web
```

Docker container commands

Stop a running container through SIGTERM

```
docker container stop web
```

Stop a running container through SIGKILL

```
docker container kill web
```

Delete all running and stopped containers

```
docker container rm -f $(docker ps -aq)
```

Docker share commands

Pull an image from a registry

```
docker pull my_image:1.0
```

Retag a local image with a new image name and tag docker tag

```
docker tag my_image:1.0 myrepo/my_image:2.0
```

Push an image to a registry

```
docker push myrepo/my_image:2.0
```

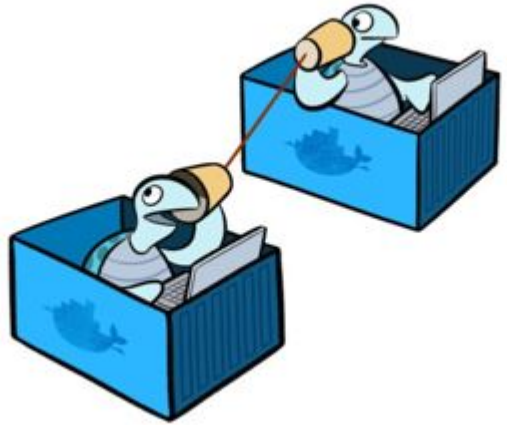
Docker more commands

To run tail command in container named app_web_1

```
docker exec app_web_1 tail logs/development.log
```

Docker Network

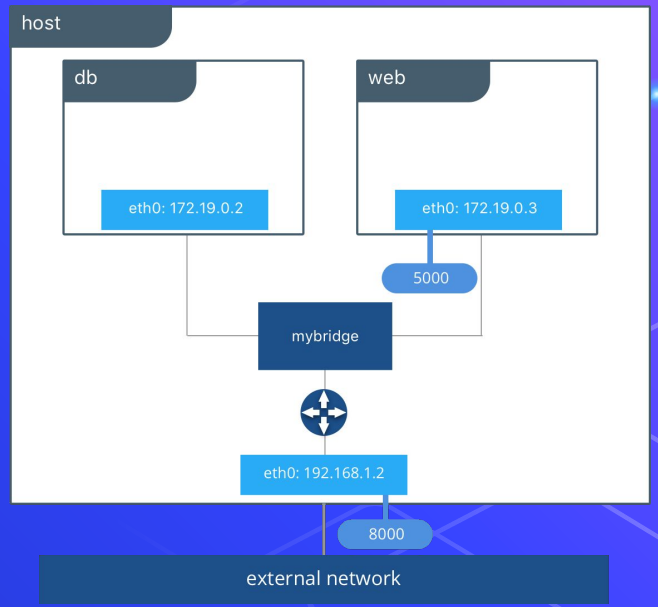
Networks are natural ways to isolate containers from other containers or other networks.



Network drivers

- **bridge:** used when your applications run in standalone containers that need to communicate.

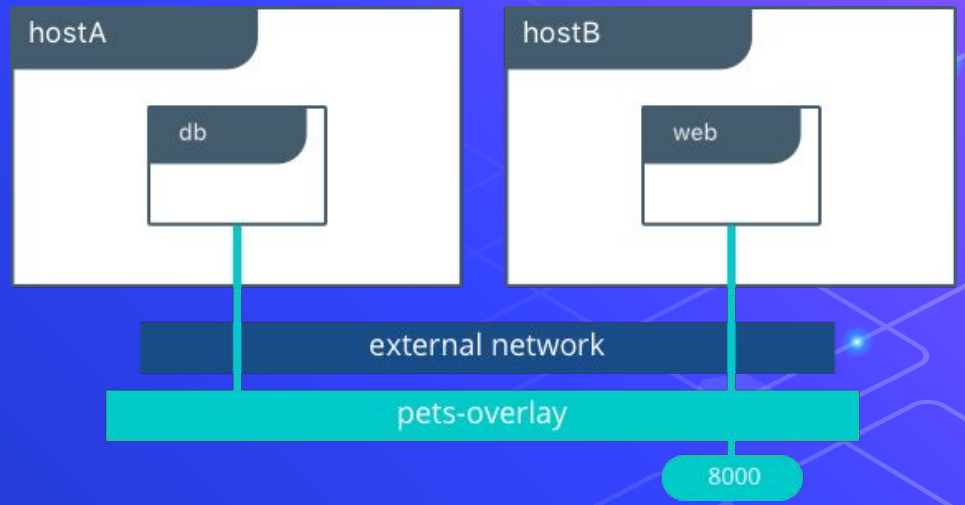
- **host:** For standalone containers, and use the host's networking directly.



Network drivers

- **overlay:** connect multiple Docker daemons together and enable swarm services to communicate with each other.

- **macvlan:** allow you to assign a MAC address to a container, making it appear as a physical device on your network.



Network drivers

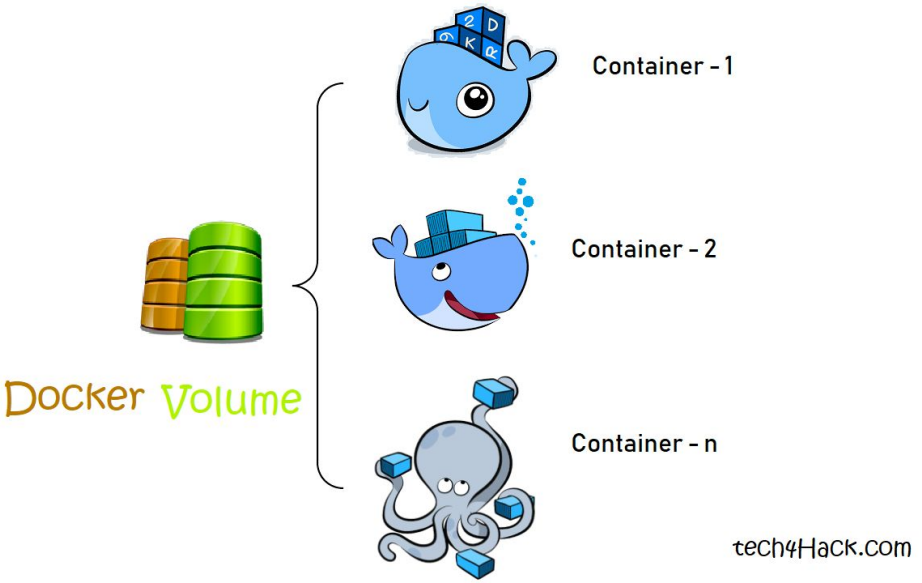
- **plugins:** You can install and use third-party network plugins with Docker.

- **none:** For this container, disable all networking.

Docker network command

`$ docker network ls`	List out the networks
`$ docker network create -d bridge my_bridge`	Create a network
`$ docker network connect my_bridge my_container`	Connect a container to a network
`$ docker network disconnect my_bridge my_container`	Disconnect a container from a network
`$ docker network inspect my_bridge`	Display detailed information on one or more networks

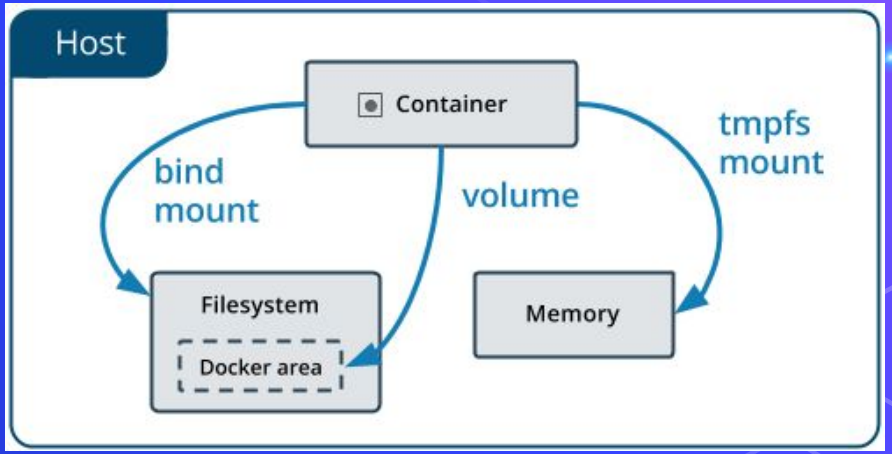
Data in Docker



manage

Types of mount

- **Volumes** *managed by Docker and should be Docker only* (/var/lib/docker/volumes/ on Linux).
- **Bind mounts** may be stored *anywhere* on the host system.
- **tmpfs mounts** are stored in the host system's memory only, and are never written to the host system's filesystem.



Docker volume command

`$ docker volume ls`	List out the volumes
`$ docker volume create my-vol`	Create a volume
`$ docker run -d --name nginx -v my-vol:/app nginx:latest`	Start a container with a volume
`$ docker volume rm my-vol`	Remove a volume
`$ docker volume inspect my-vol`	Display detailed information of a volume

Dockerfile best practice

Best practices for building better Docker images.

Reduce image size

- **Remove package manager cache:**
  Use of `apt-get update, apt-get install` should be paired with `rm -rf /var/lib/apt/lists/*` in the same layer.

- **Remove unnecessary dependencies:**
  Consider using a `--no-install-recommends` when `apt-get install` packages.

Reduce image size

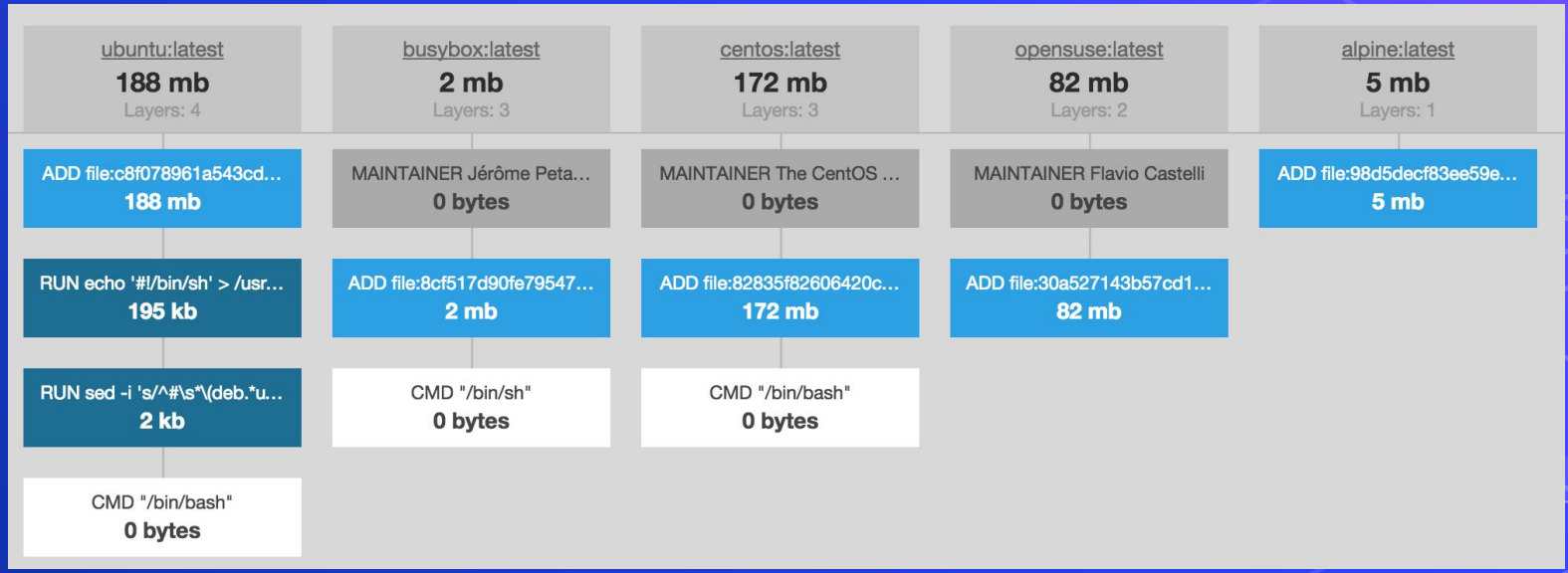
- **Leveraging the dockerignore File:**
  Should have **.dockerignore** file to ignore redundant thing when build the image.

```
$ ls -a
Dockerfile  temp.txt .dockerignore

$ cat .dockerignore
# ignore file *.tmp
*.tmp
# ignore folder temp/
temp/
# ignore file ReadMe
README.md
```

Reduce image size

- **Always look for minimal flavors:**
  Some of those tags have minimal flavors which means they are even smaller images.



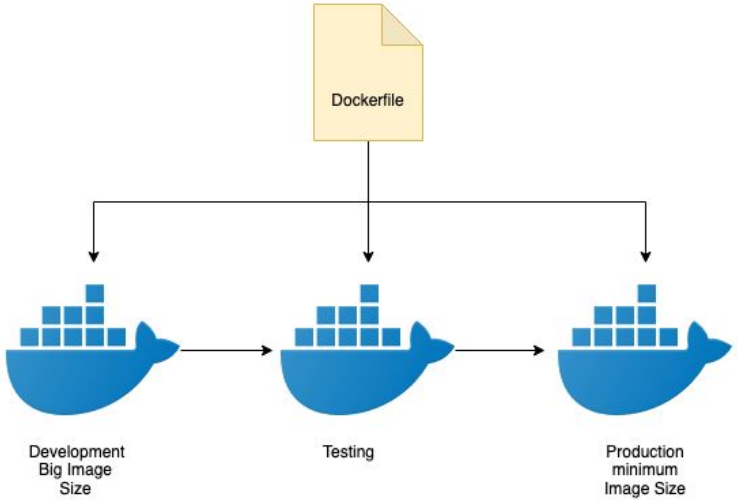
Reduce image size

- **Use multi-stage builds to remove build dependencies**
  Image is built with some redundant dependency, source code that we don't need to run the app, use multi-stage to remove it.

```
# Stage - Development
FROM Dev-Image as dev
. . .                    stage 0
. . .

# Stage - Testing
FROM Test-Image as test
COPY --from=dev /src/app .
. . .                    stage 1
. . .

# Stage - Production
FROM Minimum-Image as prod
COPY --from=dev /src/app .
. . .                    stage 2
. . .
```

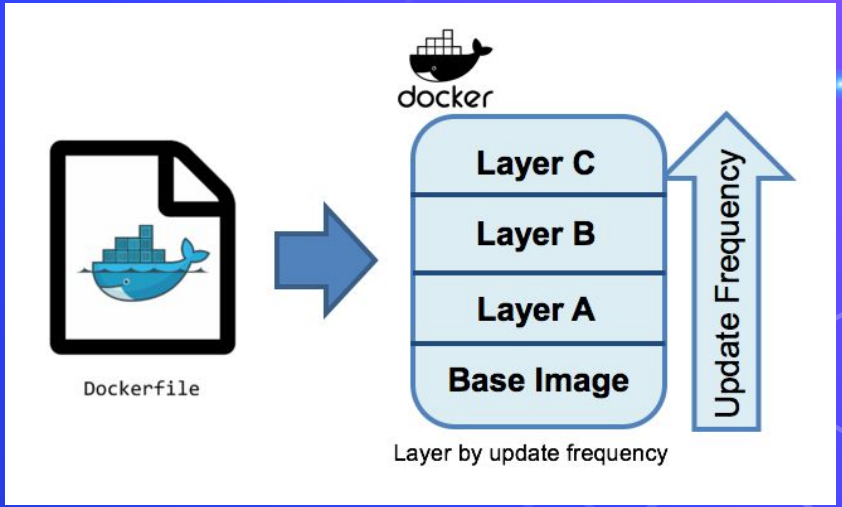


Incremental build time

- **Order matters for caching:**

  A step's cache is invalidated by *changing files or modifying lines*, subsequent steps will break.

  Order your steps from **least to most frequently changing** steps to optimize caching.

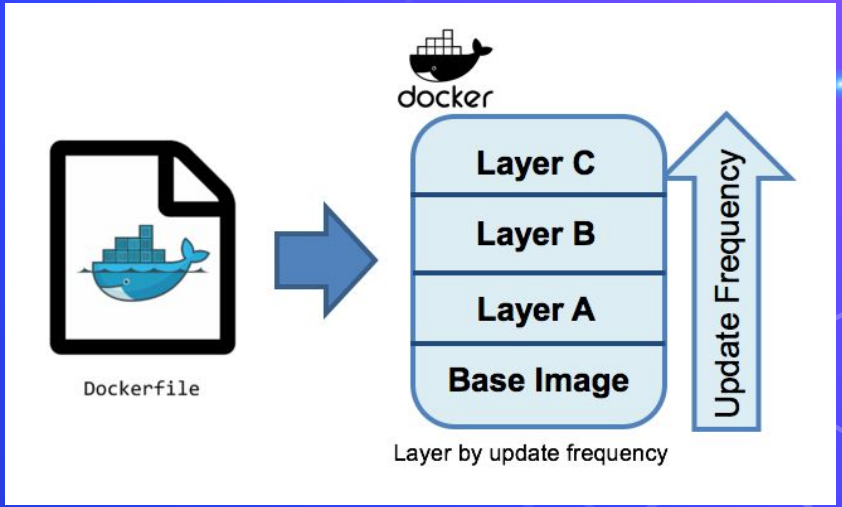


Incremental build time

- **More specific COPY to limit cache busts:**

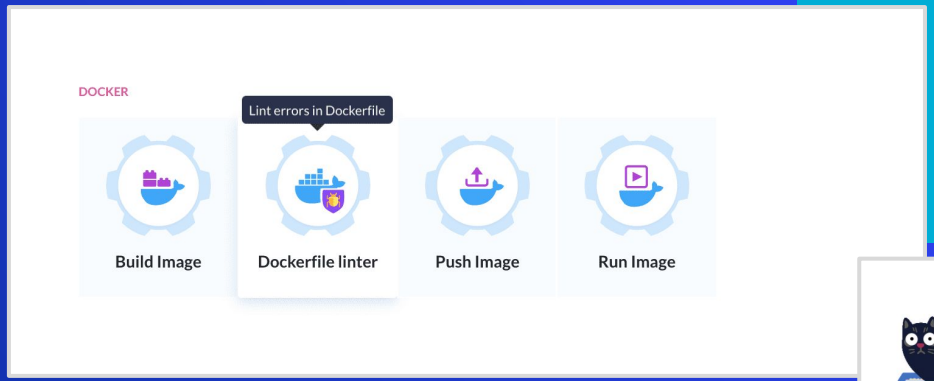
  When hen copying files into your image, make sure you are very specific about what you want to copy.

  Any changes to the files being copied will break the cache.



Useful tool for analyzing

- **fromlatest.io**
- **Dockerfile linter**
  Online version here: https://hadolint.github.io/hadolint/
- ...



Docker in production

Docker compose

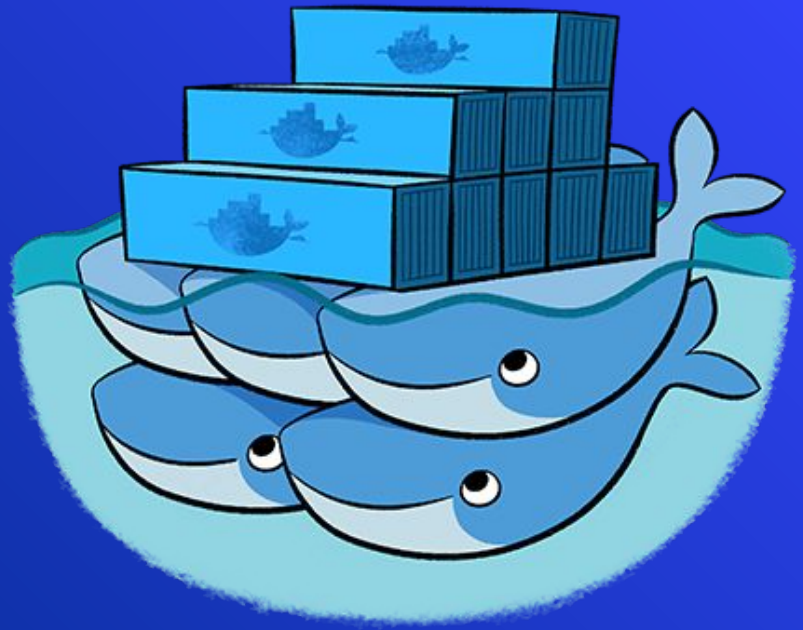
- **Define and run multi-container**
- **Use the YAML file config**
- **Play with CLI**

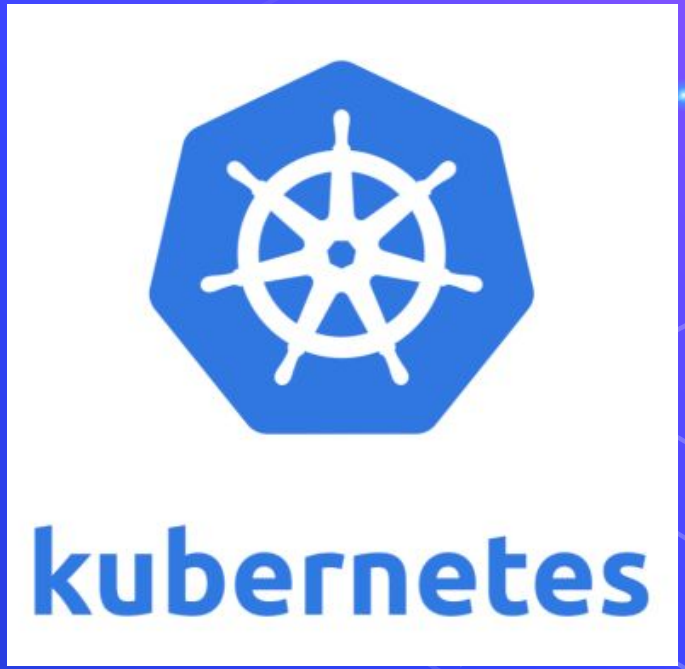
S1: Define Dockerfile for each service

S2: Define services in docker-compose.yml

S3: docker-compose up

Docker swarm & Kubernetes





Docker pitfalls

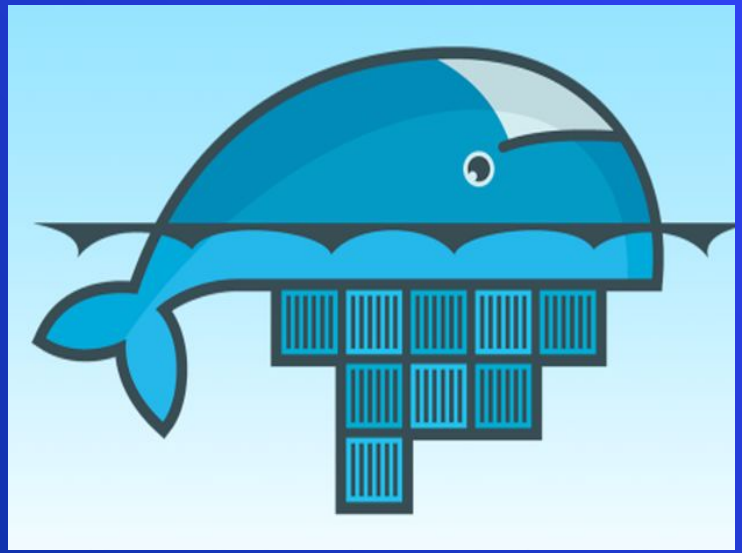
Some common mistakes should avoid when implementing Docker.

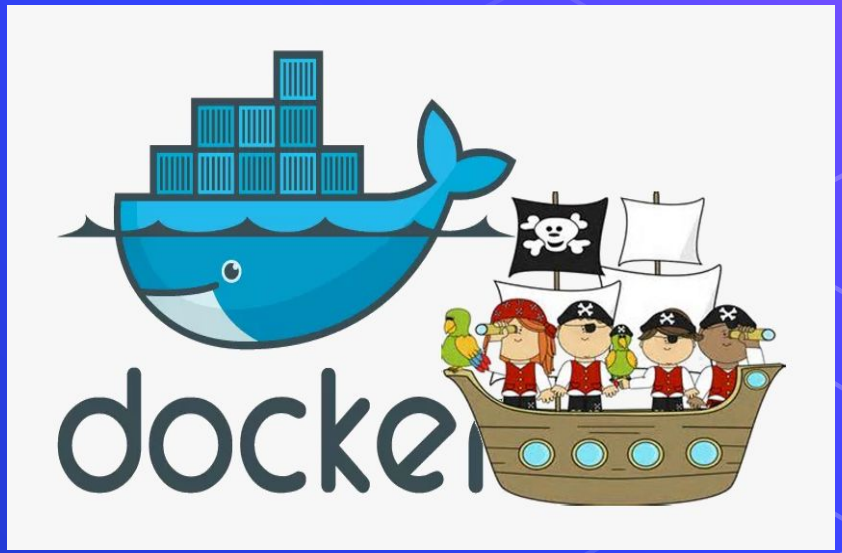
!Create large image

- Take more disk space, upload / download slow, hard to distribute.

!Storing important data in containers and registries

- Impact or loss data when container up / down.
- Sensitive data can be used via repository in registry.



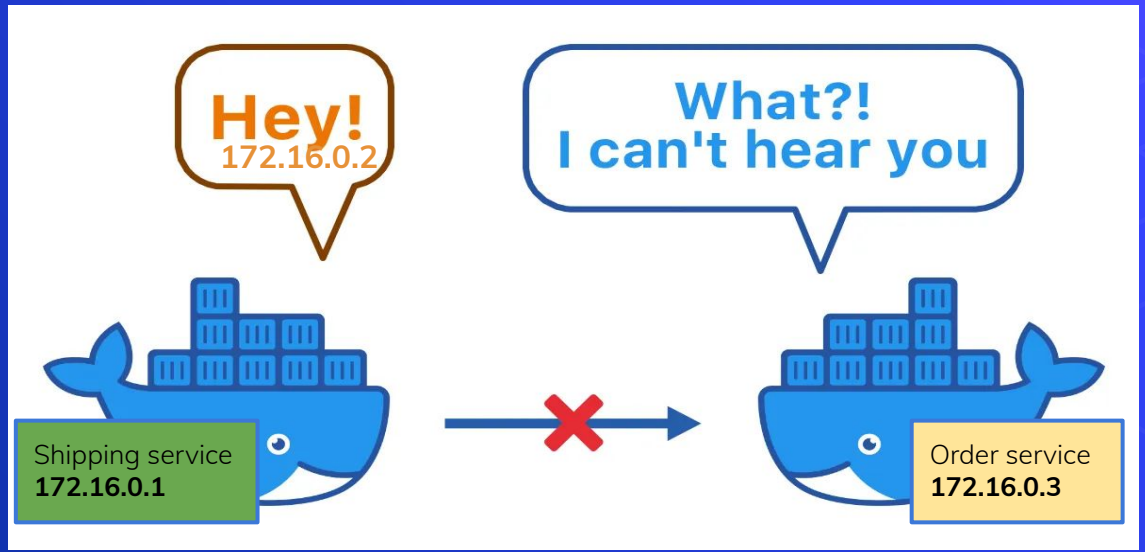


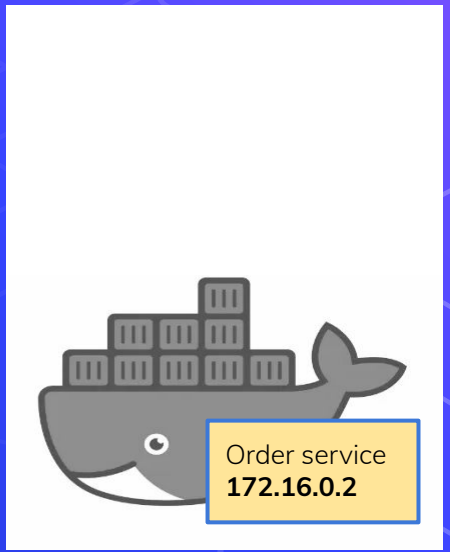
!Running multiple services in the same container

- May have more trouble managing, retrieving logs, and updating the processes individually.

!Rely on IP addresses

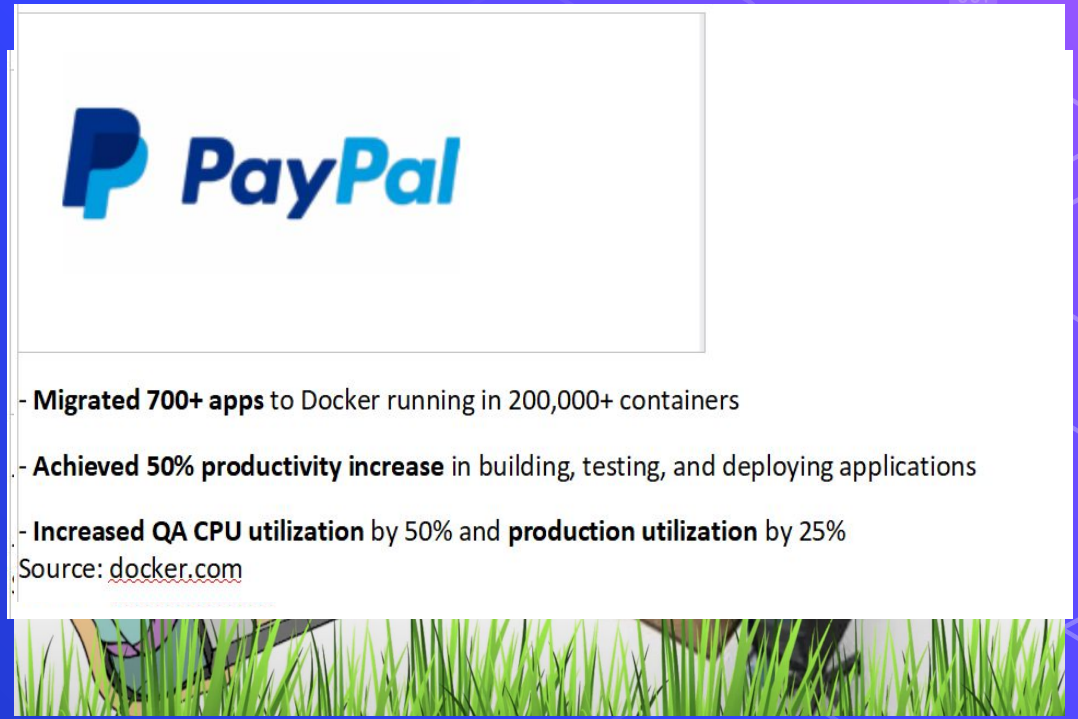
- Each container has internal IP address and it could change when start and stop container.





Benefit

- Save resources
- Lightweight, small and fast
- Productivity
- Large communities & support



Thanks!

Any questions?

Hand-on use case

bit.ly/mastering_docker