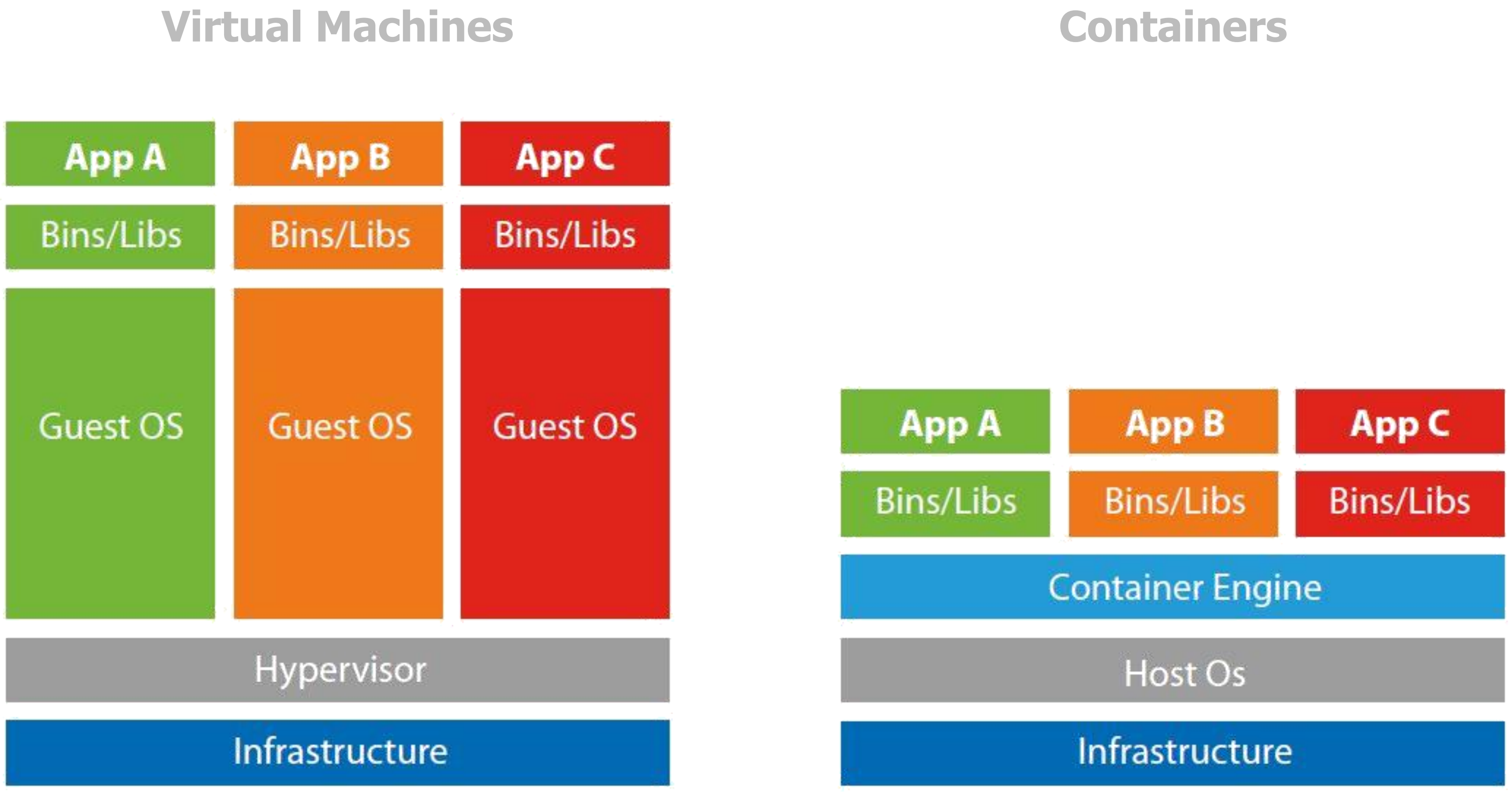
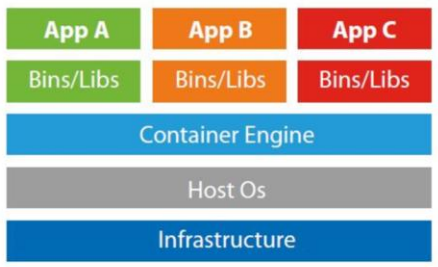
# Fundamentos de Docker

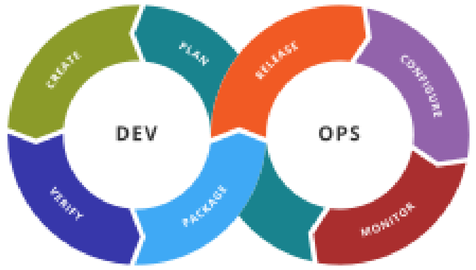
# Containers vs VMS

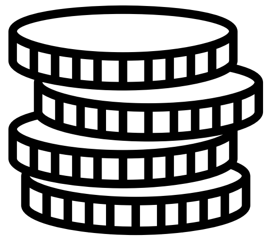


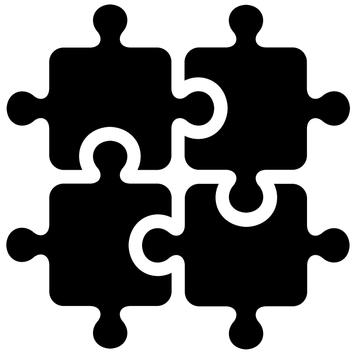
# Containers advantages

- Density and performance
- Native Cloud Applications(Scale-Out)
- Cost (licensing)
- DevOps
- CI/CD – faster deploy









## Docker

- Docker is an open platform for **developing, shipping, and running applications**.
- Docker enables you to **separate your applications from your infrastructure** so you can deliver software quickly.
- You can **manage your infrastructure in the same ways you manage your applications**.
- By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly **reduce the delay between writing code and running it** in production.

# Docker Ecosystem

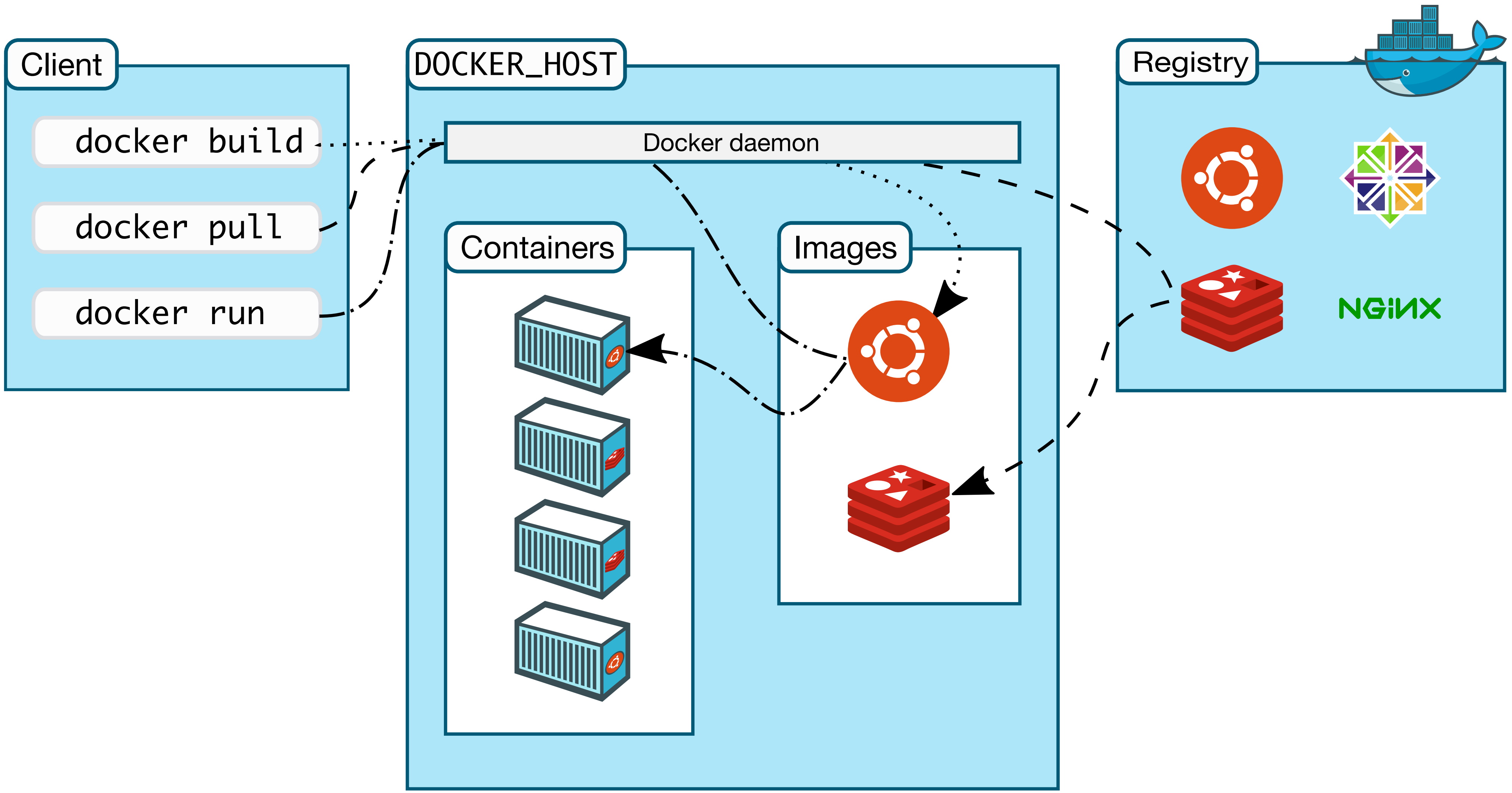


Image Source: docker.com

# Docker Ecosystem

- **Docker engine**
  - Container creation engine
  - Container Execution Engine
  - **Docker Daemon**
    - Build images → containers
    - Manage containers
    - RESTful APIs
  - **Docker-CLI**
    - Command line for Docker management
- **DockerHub**
  - https://hub.docker.com/
  - Offers Docker services
  - Public Image Library
  - Storage of our images
  - Automated builds

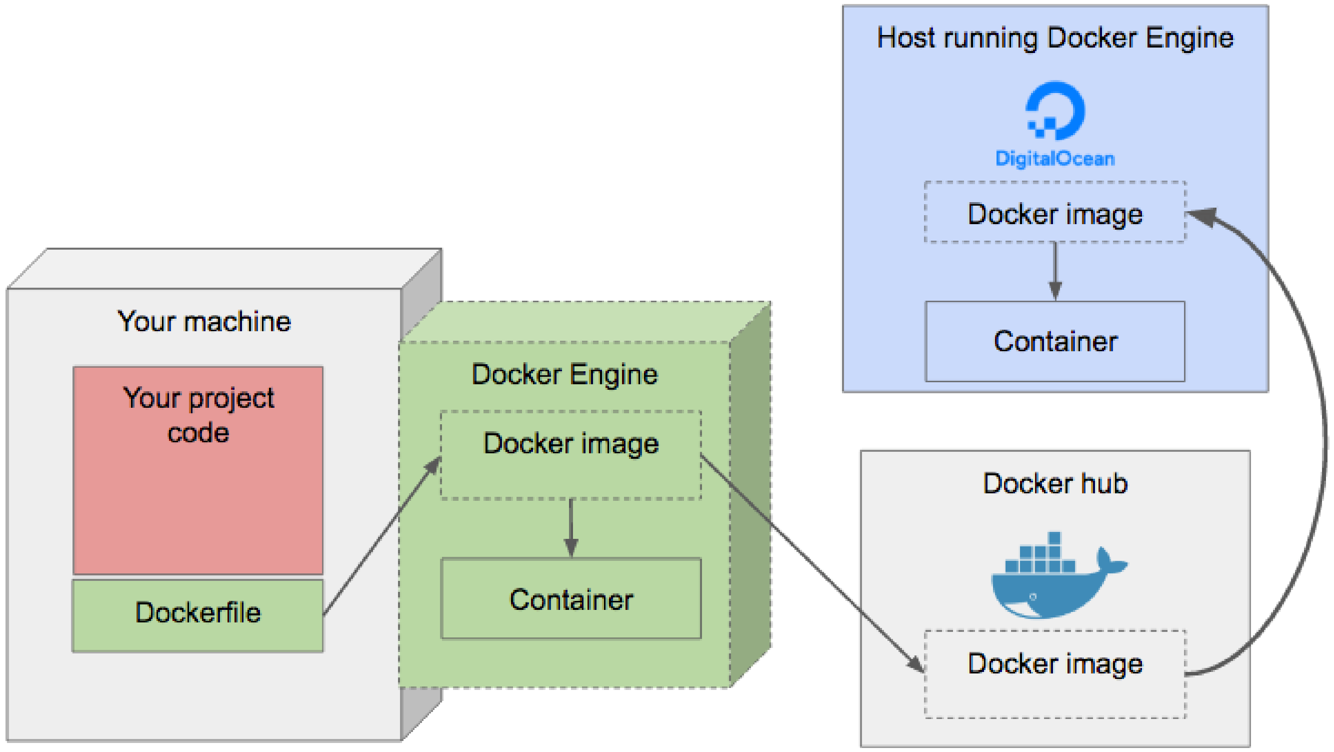


Image Source: lukewilson.net

# Docker commands

🔗 **Check that docker is correctly running and that you have permission to use the engine**

```
docker info
```

🔗 **Pull an image from the official registry, eg: debian:latest (you can browse https://store.docker.com if you want to find other images).**

```
docker pull debian:latest
```

🔗 **check images present in the docker engine:**

```
docker images
```

🔗 **Run a container from an image**

```
docker run debian:latest
```

🔗 **Show running and non running containers**

```
docker ps
docker ps -a
```

# Docker commands

🔗 **run a command**

```
docker run debian ls /bin
docker run debian cat /etc/motd
```

🔗 **interact with the shell**

```
docker run -i Debian
# -t for allocate a tty
docker run -t -i debian
# -d keeps running in the background
docker run -d -t -i debian
```

🔗 **start a stopped container**

```
docker ps –a
# -i to have stdin
docker start -i 85bcdca6c38f
docker start -i 85bcd
docker start -i 85
docker start -i 85bcdca6c38f07e3f8140cbf8b4ad37fd80d731b87c6945012479439a450a443
docker start -i pensive_hodgkin
```

# Docker commands

🔗 **commit**

```
# You can modify files inside a container. If you restart the same container you can note that these changes
are still present. However they will not be present in the other container (even if they are running the same
image) because docker uses a copy-on-write filesystem. Use the command docker diff to show the difference of
a container from its image.
# Remember that all changes inside a container are thrown away when the container is removed. If we want save
a container filesytem for later use, we have to commit the conainer (i.e take a snapshot).

docker commit CONTAINER

# This operation creates a new image. This image in turn can be used to start a new container.

docker run -it debian:jessie
         git
         apt-get update && apt-get install -y git
         git
         exit
docker commit <cont_id> <repo>/debian:<tag>
docker images
```

🔗 **remove**

```
docker rm
# removes all dead container.
docker prune
```

# Docker commands

## 🔗 docker run options

*--rm* to remove the container automatically when it terminates
*-d/--detach* to run a container in the background
*-u/--user* to run the container as a different user
*-w/--workdir* to start the container in a different directory
*-e/--env* to set an environment variable
*-h/--hostname* to set a different hostname (the host name inside the container)
*--name* to set a different name (the name of the container in the docker engine)

also you may type **docker run --help** to display all configuration keys

## 🔗 other docker commands

**docker cp** to transfer files from/into the container
**docker exec** to have launch a separate command (very useful for providing a debugging shell -> docker exec -t -i CONTAINER bash)
**docker top** to display the processes running inside the container
**docker stats** to display usage statistics
**docker logs** to display the container output
**docker attach** to reattach to the console of a detached container

## 🔗 ports

```
docker run -it -d -p 8888:8080 tomcat:8.0
docker logs <container_id>
```

## Docker commands

🔗 **push** publishes an image in dockerhub. You will need an account.

```
docker login <REGISTRY_HOST>:<REGISTRY_PORT>
docker tag <IMAGE_ID> <REGISTRY_HOST>:<REGISTRY_PORT>/<APPNAME>:<APPVERSION>
docker push <REGISTRY_HOST>:<REGISTRY_PORT>/<APPNAME>:<APPVERSION>

# push to dockerhub
docker login
docker tag my-image myhubusername/debian:v1
docker push myhubusername/debian:v1


# push to custom repo
docker login repo.company.com:3456
docker tag 19fcc4aa71ba repo.company.com:3456/myapp:0.1
docker push repo.company.com:3456/myapp:0.1
```

🔗 **delete images**

```
docker image rm <image_id>
docker rmi $(docker images | grep ‘<text>')
```

# Building containers

🔗 **Build** creates a new image from a previous image or a Dockerfile

```
# Generate an image from another existing (a copy with another page)
docker tag jboss/wildfly myimage:v1

#from a Dockerfile
docker build -t <repo>/<image_name>:<tag> <Dockerfile_path>
docker build -t <repo>/<image_name>:<tag> -f <NotDockerFileName_path>
#cleaning the cache
docker build -t <repo>/<image_name>:<tag> < Dockerfile_path> --no-cache=true

docker build -t dockerapp:v0.1 .
docker run -d -p 5000:5000 <new_image_id>
```

🔗 **Dockerfile** is a file that defines a new image to be created. It uses keywords like

```
         FROM defines the base image
         WORKDIR sets the working directory or context inside the image
         RUN a build step
         CMD the command the container executes by default
         COPY copies a file/folder to the container from a location to a destination in the Docker container.
         ADD copy files/directories into a Docker image
         EXPOSE  expose a port inside of the image to the outside world.
```

## Building containers

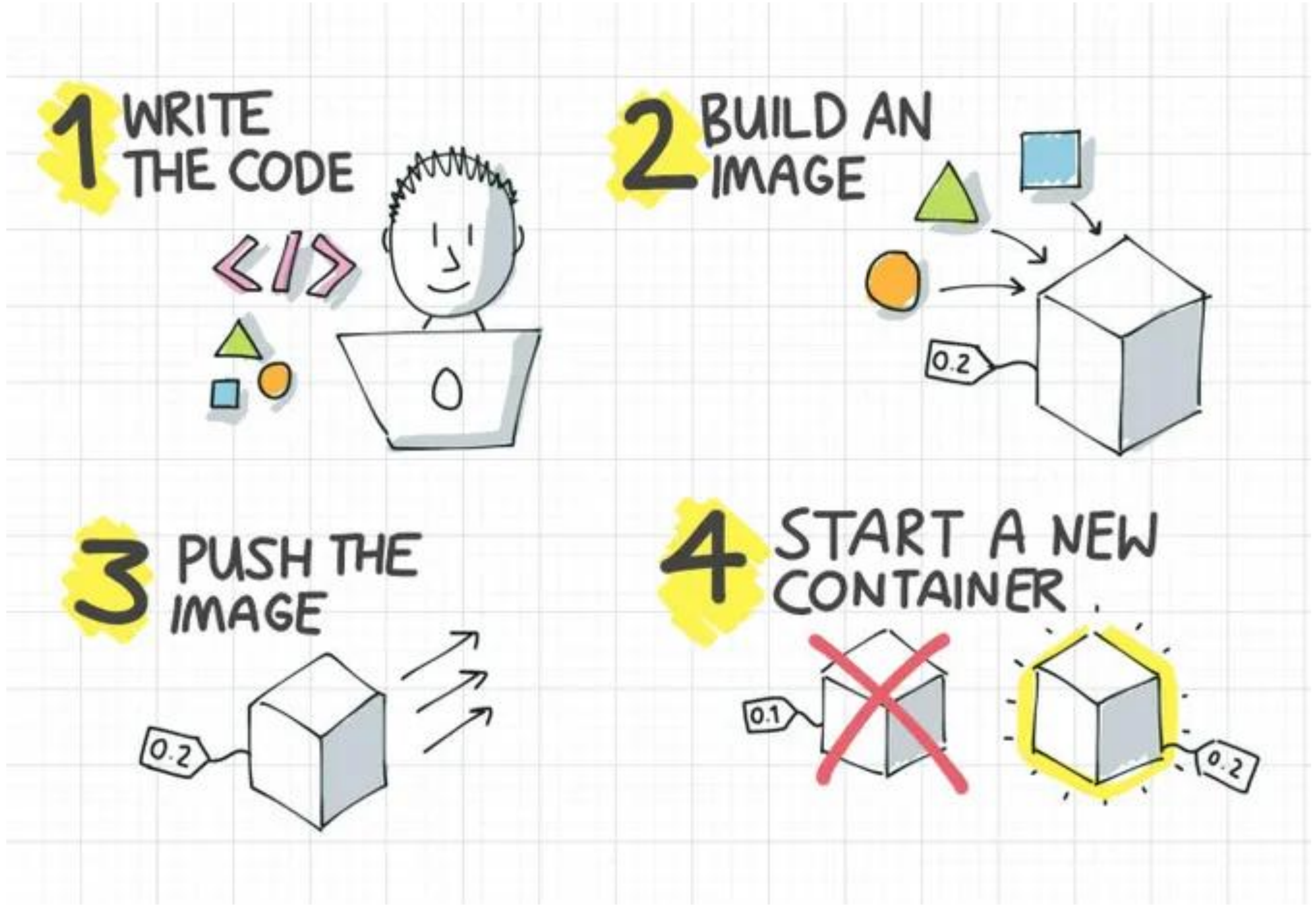
🔗 **Dockerfile**

```
#Dockerfile
FROM busybox
RUN touch testfile
RUN /bin/bash -c echo "Next build step..."
COPY testfile /

#Build the image
docker build -t local_busybox -f Dockerfile ./

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

## Developing with Containers



**Read:** https://www.tutorialworks.com/container-development-workflow/

## Next steps

**We would like to know your opinion!**

Please, let us know what you think about the content.
From Netmind we want to say thank you, we appreciate time and effort you have taking in answering all of that is important in order to improve our training plans so that you will always be satisfied with having chosen us
quality@netmind.es

# Thanks!

Follow us: