Workshop Docker

Chapter Devs + Infinitus

Walmart

Agenda

- Introduction
- Why the workshop?
- Requirements
- What is docker?
- Architecture
- Inside Docker
- Development with Docker
- Commands
- Best practices on Walmart

Intro

This Workshop has a series of labs and instructions to introduce you to containers and Docker.

We can help you run a container, inspect it and understand it as well as process isolation, create a Dockerfile, build an image from a Dockerfile, understand layers, tag and upload images to a docker registry, scale and update containers, and much more.

Why the workshop

This workshop has a series of additional tools to help you on your journey to containers and docker.

We believe that it is necessary to understand the basic concepts in order to create and develop better products to add value to our organization.

1. *Get Docker (Mac, Windows, Linux)*
2. *Docker Hub Registry*
3. *Play with Docker*
4. *Homebrew*

Requirements

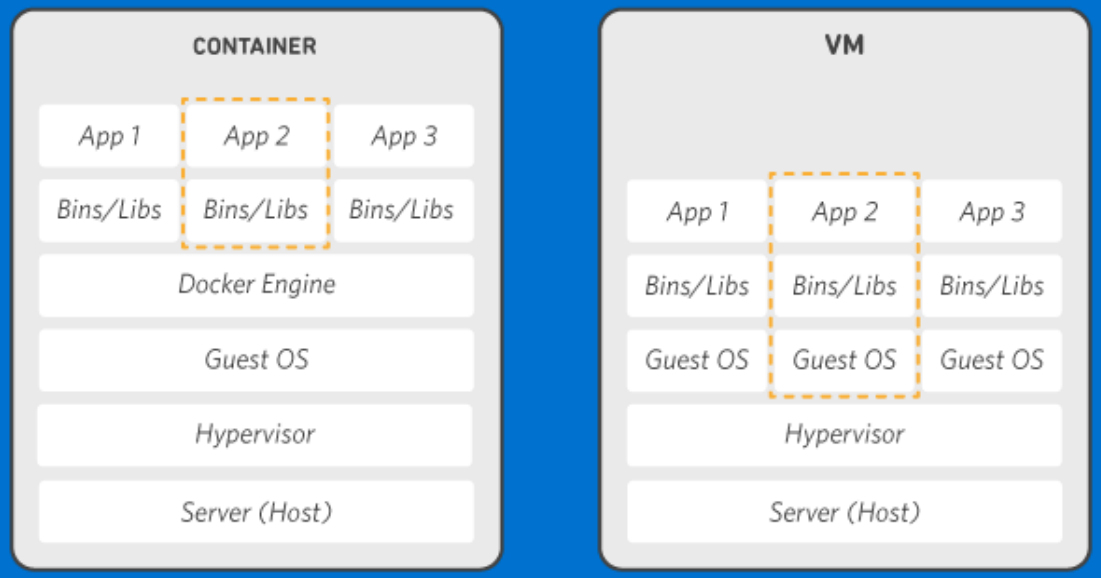
For this workshop you must have:

1. Docker CLI
2. Docker Engine
3. Docker Registry Account
4. Desire to learn :D

What is docker?

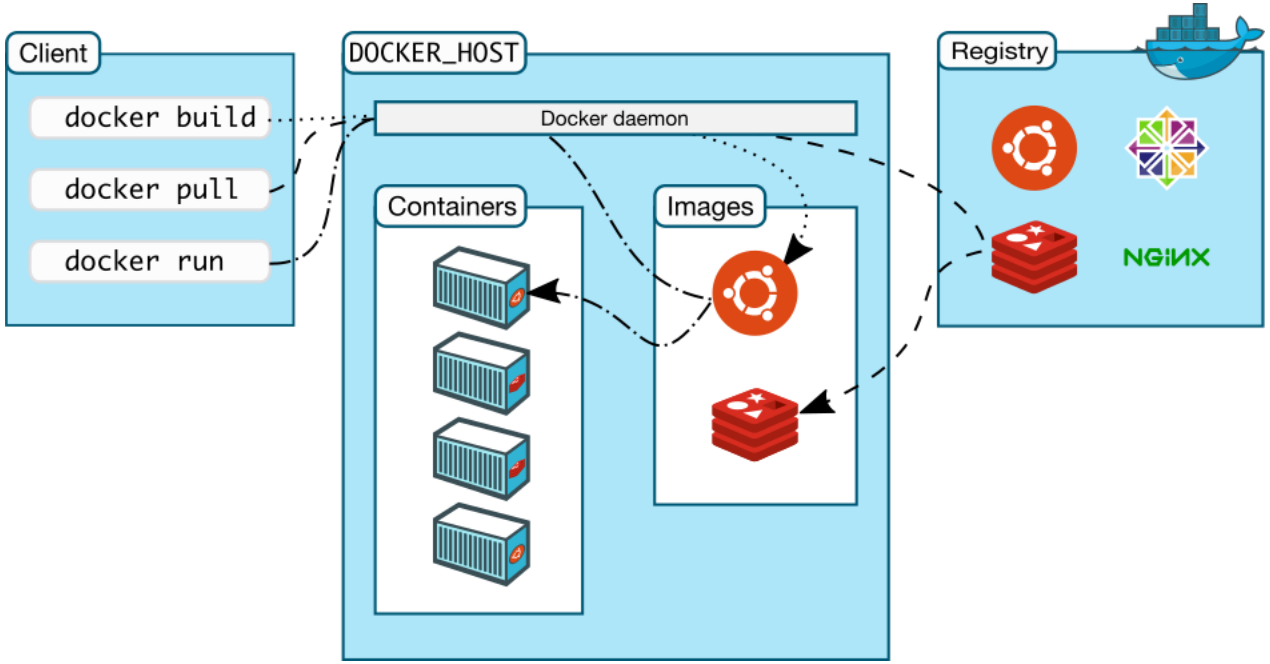
Docker is an open-source platform that allows you to quickly create, test, and deploy applications.

This platform packages our application with all its dependencies in containers.



Architecture

Docker runs on **client-server architecture**. The client communicates with the docker service(daemon), which does the work of compiling, running, and distributing the containers. The registry is in charge of storing the docker images.



Inside Docker

- **Images**

Read-only template with the instructions for creating a container, usually created from another image with additional customization.

- **Layers**

In an image, a layer is the modification of the image, represented by an instruction in the Dockerfile. Layers are applied in sequence to the base image to create the final image.

- **Containers**

It is an executable instance of an image, which is relatively isolated from other containers and from its host.

Development with Docker

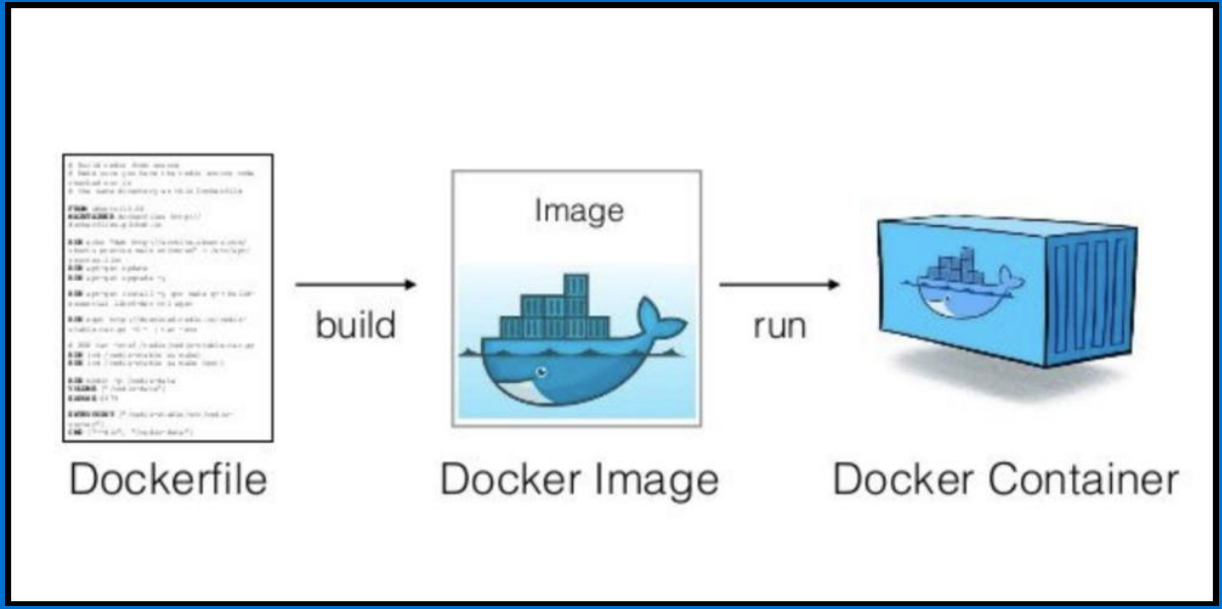
Docker Containers

Docker container is a runnable instance of an image, which is actually made by writing a readable/writable layer on top of some read-only layers.

The parent image used to create another image from a Dockerfile is read-only. When we execute instructions on this parent image, new layers keep adding up. These layers are created when we run docker build command.

Dockerfile

A Dockerfile is a text file which contains a series of commands or instructions. These instructions are executed in the order in which they are written. Execution of these instructions takes place on a base image.
On building the Dockerfile, the successive actions form a new image from the base parent image.



Dockerfile

- ***FROM***

It is used to create a new construction layer and sets which base image is used

Dockerfile

```
FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
```

- ***RUN***

Executes commands available on the base image which creates a new layer

Dockerfile

```
FROM ubuntu
RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'
```

Dockerfile

```
FROM ubuntu
RUN ["/bin/bash", "-c", "echo hello"]
```

Dockerfile

- ***CMD***

Provides default values for a running container. These values may be an executable and in which case an ENTRYPOINT must be specified

Dockerfile

```
FROM ubuntu
CMD echo "This is a test." | wc -
```

Dockerfile

```
FROM ubuntu
CMD ["/usr/bin/wc","--help"]
```

- ***ENTRYPOINT***

Allows you to configure a container to start as an executable

Dockerfile

```
FROM ubuntu
ENTRYPOINT ["top", "-b"]
CMD ["-c"]
```

Dockerfile

- CMD and ENTRYPOINT Rules
  - Every Dockefile must have a ***CMD*** or an ***ENTRYPOINT***
  - ***ENTRYPOINT*** *should be used to use containers as executables, examples: nginx, databases, etc.*
  - ***CMD*** is used to define defaults for using ***ENTRYPOINT*** or for executing ad-hoc commands in this container.

Dockerfile

- ***EXPOSE***

Informs Docker that the container listens on the specified network ports at runtime

```
Dockerfile
1 EXPOSE <port> [<port>/<protocol>...]
```

- ***LABEL***

It is an instruction that allows us to add ***METADATA*** to our containers

```
Dockerfile
1 LABEL multi.label1="value1" multi.label2="value2" other="value3"
```


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

Multistage

It is a process in which ***intermediate containers*** can participate in the construction of a final image. This has the following advantages:

Facilitates Dockerfile ***maintainability***

- Allows the final image to be ***free of unnecessary dependencies*** that were used in the compilation process

Title

```
FROM golang:1.7.3 AS builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /go/src/github.com/alexellis/href-counter/app .
CMD ["./app"]
```

Docker commands

Docker Commands

- docker build

Creates Docker images from a Dockerfile

Docker build

```
1 docker build -f Dockerfile -t images_name:tag .
```

Detail:

- -f : Specifies the name or location of the Dockerfile
- -t : Assigns the name of the image with its respective tag

Docker Commands

- docker run

Allows you to run a container from an image

Docker run

```
docker run --rm --name nginx_container -d -p 80:80 nginx
```

Detail:

- --rm : Allows the container to be removed once it stops.
- --name : Allows you to assign a name to a container
- -d : Allows to run a container in detached mode
- -p : Allows ports to be exposed to the host interface

Docker Commands

- docker container ls

Allows to list containers

Docker container

```
1 docker container ls --all --quiet
```

Detalle:

- --all, -a : List all containers up to the detainees
- --quiet, -q : Lists container IDs only

Docker Commands

- docker container stop

Allows to stop a container

Docker container

```
docker container stop container_name
```

- docker container start

Allows you to start a container

Docker container

```
docker container start container_name
```

Docker Commands

- docker container logs

Allows you to view the logs of a container

Docker container

```
1 docker container logs container_name --tail 100 --follow
```

Detalle:

- --tail, -n : Number of lines to display
- --follow, -f : Keeps track of the output being generated

Docker Commands

- docker container exec

Allows to execute a command in a running container

Docker container

```
docker exec  --interactive --tty container_name command
```

Detalle:

- --interactive, -i : Connect to STDIN
- --tty, -t : Assign pseudo-TTY terminal

Docker Commands

- docker stats

The docker stats command returns container consumption statistics such as memory, cpu and networking usage.

```
CONTAINER ID   NAME          CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O    PIDS
02017f310c3f   sad_vaughan   0.00%     3.793MiB / 1.944GiB   0.19%     1.66kB / 0B   205kB / 0B   5
```

Detalle:

- --all -a: Displays statistics of all running containers.

Docker Commands

- Concatenated commands

This CLI allows concatenated commands to simplify management tasks.

Docker container

```
1 docker container stop $(docker container ls -q) --time 10
```

Details:

- First, the command found in $() is executed, which outputs the container IDs
- Subsequently a docker container stop will be executed for each of the IDs with the --time flag to stop after 10 minutes.

Thank you!