EDEM

Máster en Data Analytics

Containers

Alberto Hernández Cantos

Agenda

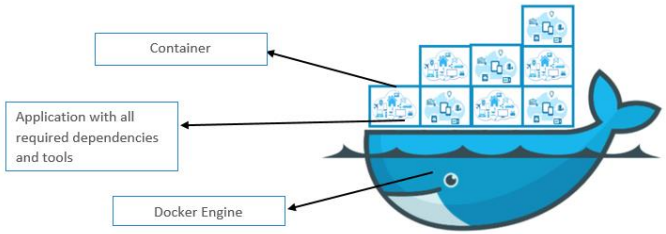
1. Containers

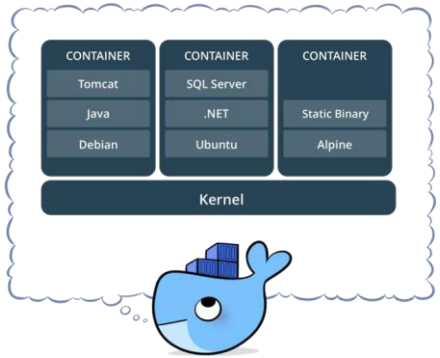
2. Docker

1 Containers

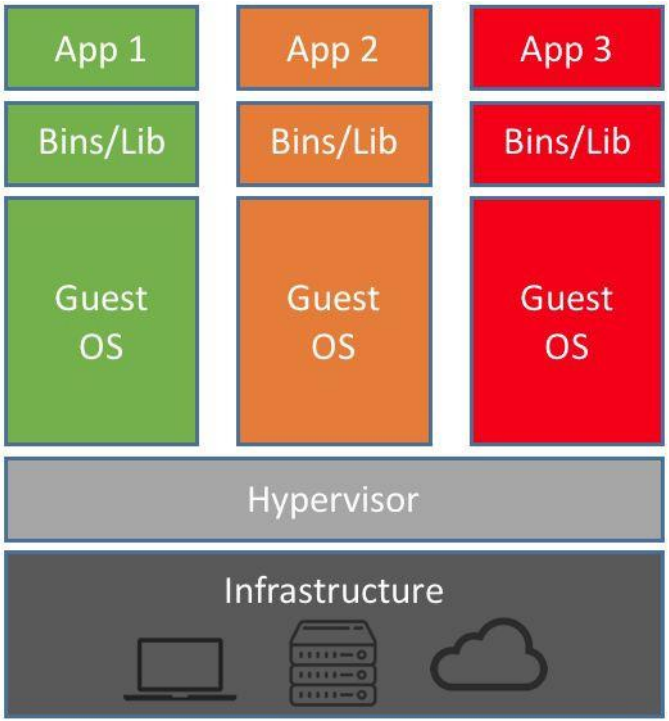
What is a Container?

- **Standardized packaging** for software and dependencies
- **Lightweight** Packaging of everything needed to run one o multiple processes: code, system tools, dependencies, etc.
- **Insolate** apps from each other
- **Share the same OS Kernel**
- Works with all major Linux and Windows Server

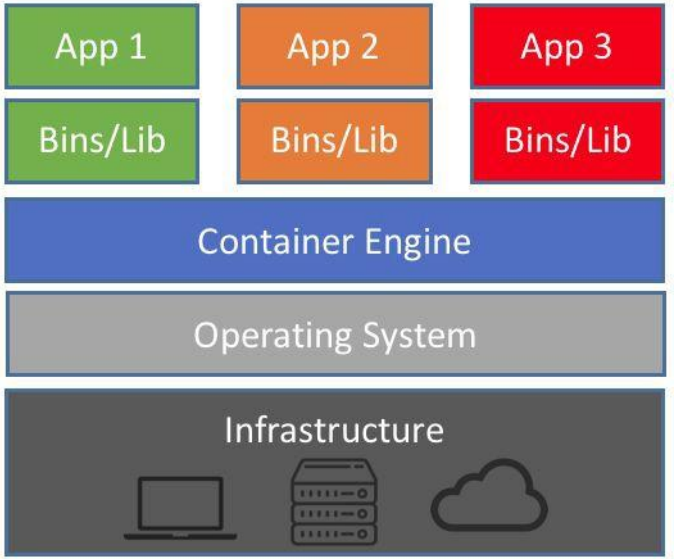




Containers vs. VMs

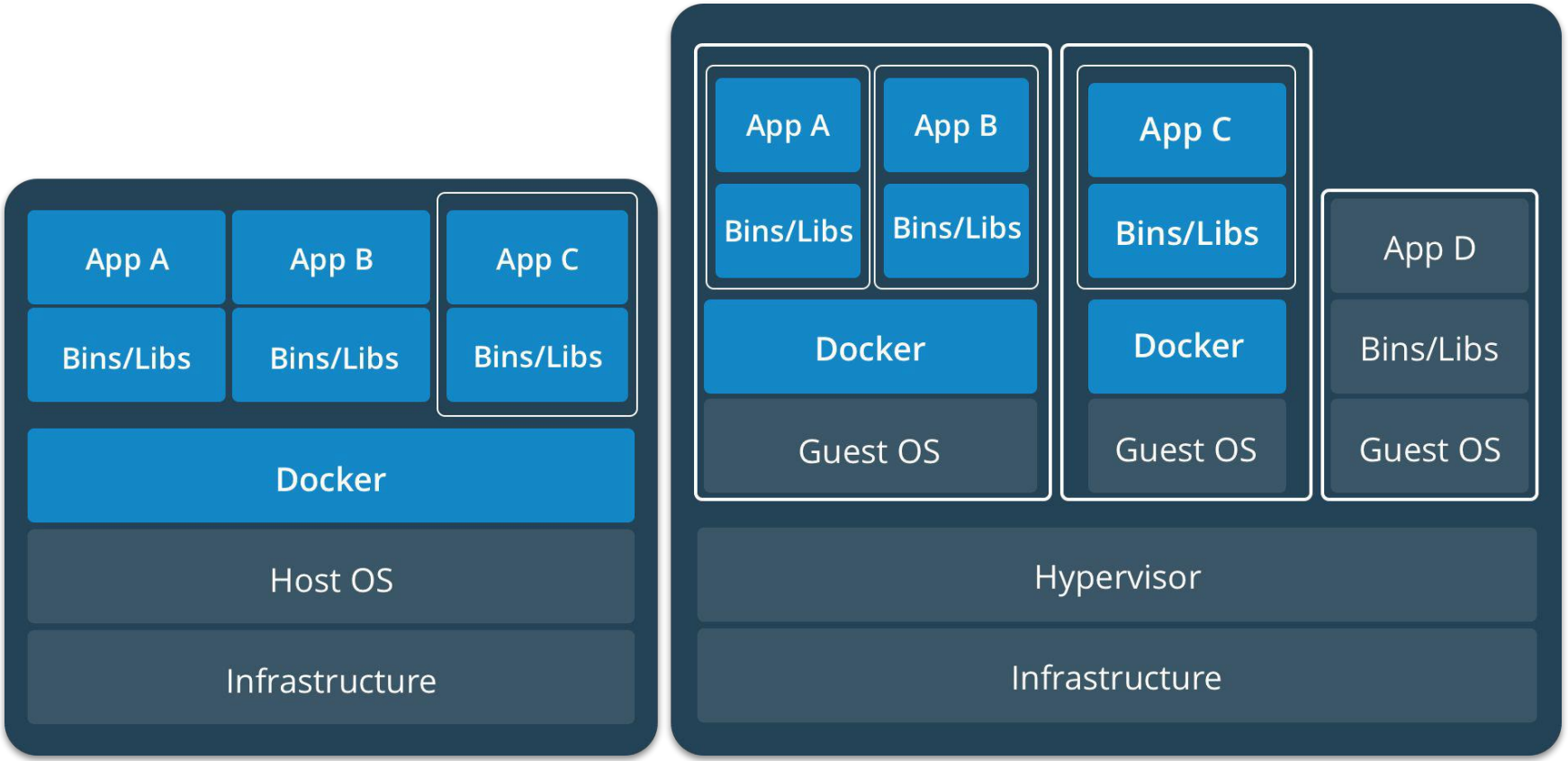


Machine Virtualization



Containers

Containers & VMs together



Container Runtimes

podman

Rocket

buildah

cri-o

skopeo

kata containers

2 Docker

Docker

- Docker is an open platform for developing, shipping, and running containerized applications
- With Docker, you can manage your infrastructure in the same way you manage your applications
- No OS to boot → Applications online in seconds

Hands-on

- Docker in your laptop
  - Windows Users (Windows 10 Enterprise & pro & home):

    https://docs.docker.com/desktop/windows/install/
  - Mac Users

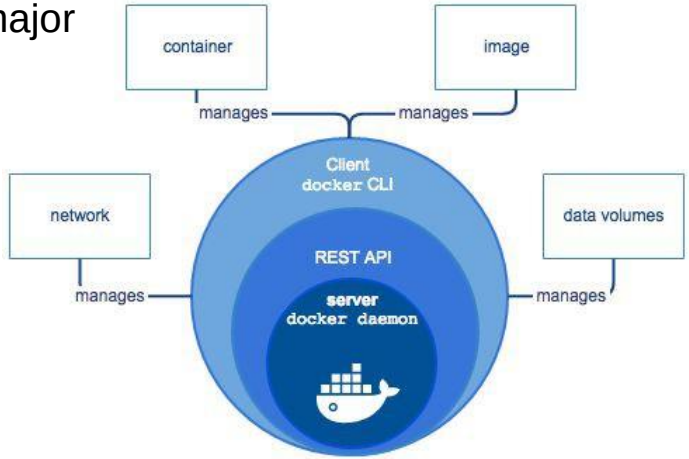
    https://docs.docker.com/desktop/mac/install/
  - Ubuntu Users

    https://docs.docker.com/engine/install/ubuntu/

Docker Versions

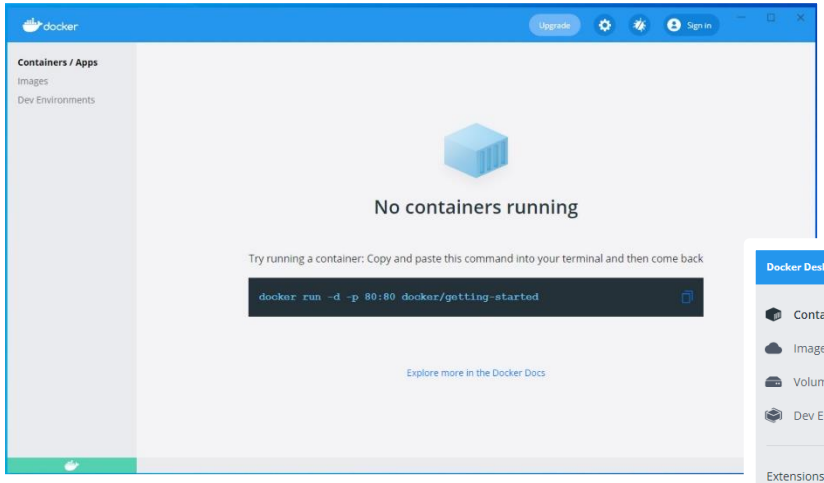
https://hub.docker.com/

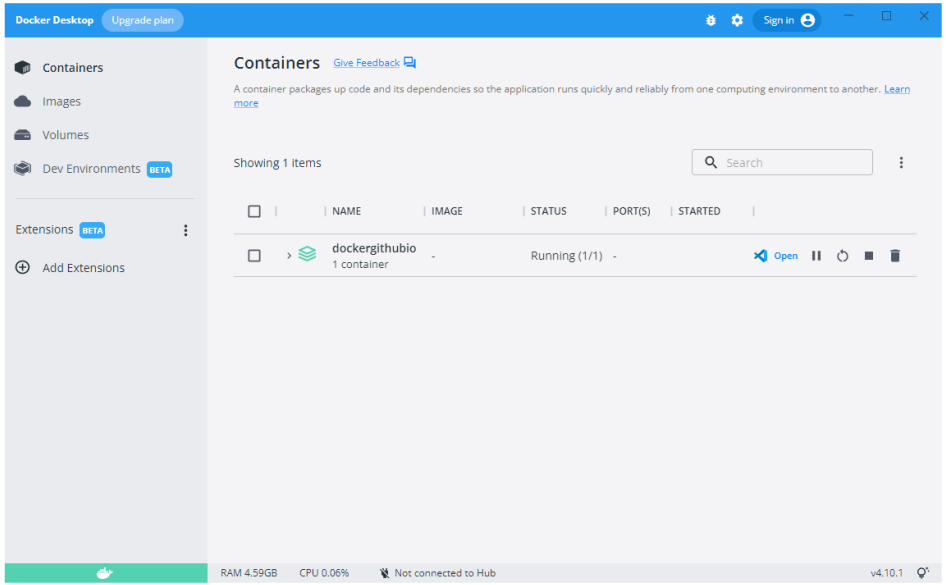
- **Docker Engine** is a client-server application with these major components:
  - Available for Linux
  - Docker Daemon (Server) (dockerd)
  - REST API
  - Docker Client (Command Line Interface (CLI))



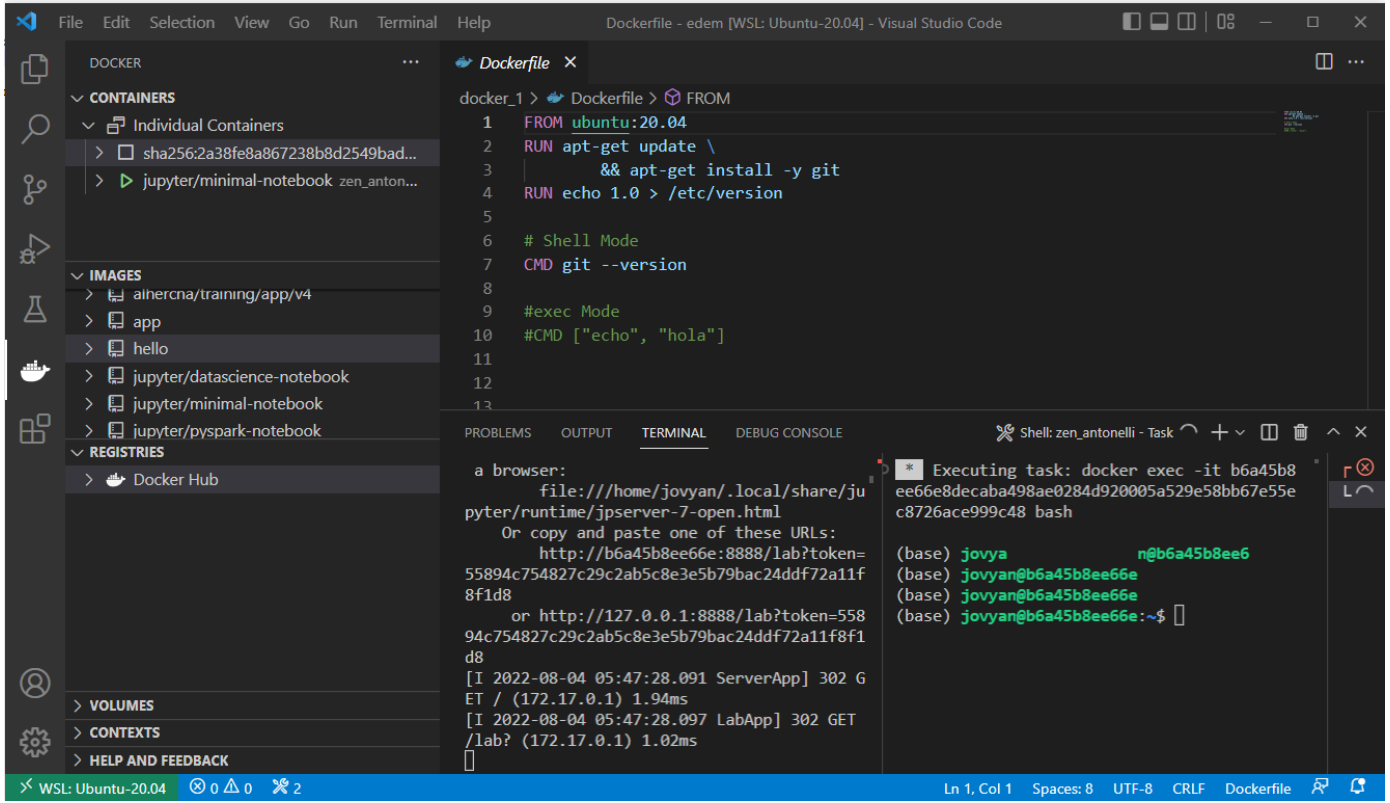
- **Docker Desktop**:
  - Contains Docker Engine
  - Available for Windows, Linux and Mac
  - Focus on Development Environments
  - Contains additional tools such as docker-compose, GUI, etc.
  - Offers different subscriptions & prices

Docker Desktop



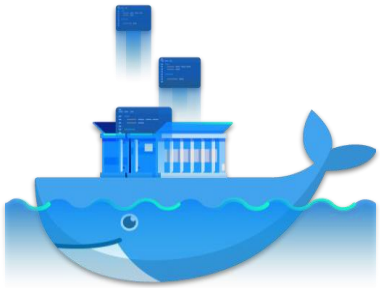


Visual Studio Code



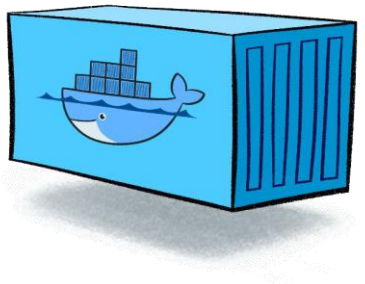
Docker Image

- A Docker Image is like an Instalación CD
- A read-only template
- Recognized by name or Image ID
- They are pushed to and can be pulled from Docker Hub



Docker Container

- Running instance of a Docker Image
- Provides similar isolation to VMs but lighter!
- Adds writable layer on top of image layers and works on it
- Can talk to other containers like processes in Linux
- Provide resources (CPU/RAM) to an image



Hands-on

- “‘docker run hello-world”’

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Hands-on – Basic Comands

- “docker version”
- “docker –help”
- “docker image”
- “docker image --help”
- “docker image ls”
- “docker ps”
- “docker ps -a”
- “docker run <image>”
- “docker run –it <image>”

```
root@b3672f89e27f:/# docker image ls
bash: docker: command not found
root@b3672f89e27f:/# exit
exit
droot@ESPC012268:/mnt/c/Users/aohz/git/edem# docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
ubuntu        latest    2dc39ba059dc   3 weeks ago      77.8MB
hello-world   latest    feb5d9fea6a5   12 months ago    13.3kB
root@ESPC012268:/mnt/c/Users/aohz/git/edem#
```

```
root@ESPC012268:/mnt/c/Users/aohz/git/edem# docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
3ced2feb80ed   hello-world   "/hello"   46 seconds ago   Exited (0) 46 seconds ago             happy_colden
28b96f06588a   ubuntu        "bash"     52 seconds ago   Exited (0) 51 seconds ago             blissful_saha
root@ESPC012268:/mnt/c/Users/aohz/git/edem#
```

Hands-on

- **Exercise 1A**

  1. List Docker images
  2. Create a new <**hello-world**> container
  3. List Docker images
  4. List running containers
  5. List all containers

- **Exercise 1B**

  1. List Docker images
  2. Create a new <**Ubuntu**> container
  3. List Docker images
  4. List running containers
  5. List all containers

TIP!

- docker run –it <image_id>

Hands-on – Background Containers

- “docker run nginx”
- “docker run –d nginx”
- “docker stop <container_id>”
- “docker start <container_id>”

- “docker rm <container_id>”
- “docker rmi <image_id>”

```
docker run nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform con
figuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/defau
lt.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/def
ault.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/07/19 14:29:04 [notice] 1#1: using the "epoll" event method
2022/07/19 14:29:04 [notice] 1#1: nginx/1.23.0
2022/07/19 14:29:04 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2022/07/19 14:29:04 [notice] 1#1: OS: Linux 5.10.102.1-microsoft-standard-WSL2
2022/07/19 14:29:04 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2022/07/19 14:29:04 [notice] 1#1: start worker processes
2022/07/19 14:29:04 [notice] 1#1: start worker process 31
2022/07/19 14:29:04 [notice] 1#1: start worker process 32
2022/07/19 14:29:04 [notice] 1#1: start worker process 33
2022/07/19 14:29:04 [notice] 1#1: start worker process 34
2022/07/19 14:29:04 [notice] 1#1: start worker process 35
2022/07/19 14:29:04 [notice] 1#1: start worker process 36
2022/07/19 14:29:04 [notice] 1#1: start worker process 37
2022/07/19 14:29:04 [notice] 1#1: start worker process 38
```

Hands-on

- **Exercise 2**

1. Run a nginx container
2. How many nginx images/containers do you have?
3. Run a new nginx container
4. How many nginx images/containers do you have?
5. Stop and start them
6. How many nginx images/containers do you have?

Hands-on – Running Commands in Containers

3 Main ways to interact with a Docker Container:

- “docker **run** ubuntu <command>”
- “docker **start** –i <container>
- “docker **exec** ubuntu <command>”
- “docker cp <source> <target>”

```
root@ESPC012268:/mnt/c/Users/aohz/git/edem/docker_1# docker run ubuntu ls -l
total 48
lrwxrwxrwx   1 root root    7 Aug  1 13:22 bin -> usr/bin
drwxr-xr-x   2 root root 4096 Apr 18 10:28 boot
drwxr-xr-x   5 root root  340 Sep  2 14:02 dev
drwxr-xr-x   1 root root 4096 Sep  2 14:02 etc
drwxr-xr-x   2 root root 4096 Apr 18 10:28 home
lrwxrwxrwx   1 root root    7 Aug  1 13:22 lib -> usr/lib
lrwxrwxrwx   1 root root    9 Aug  1 13:22 lib32 -> usr/lib32
lrwxrwxrwx   1 root root    9 Aug  1 13:22 lib64 -> usr/lib64
lrwxrwxrwx   1 root root   10 Aug  1 13:22 libx32 -> usr/libx32
drwxr-xr-x   2 root root 4096 Aug  1 13:22 media
drwxr-xr-x   2 root root 4096 Aug  1 13:22 mnt
drwxr-xr-x   2 root root 4096 Aug  1 13:22 opt
dr-xr-xr-x 182 root root    0 Sep  2 14:02 proc
drwx------   2 root root 4096 Aug  1 13:25 root
drwxr-xr-x   5 root root 4096 Aug  1 13:25 run
lrwxrwxrwx   1 root root    8 Aug  1 13:22 sbin -> usr/sbin
drwxr-xr-x   2 root root 4096 Aug  1 13:22 srv
dr-xr-xr-x  11 root root    0 Sep  2 14:02 sys
drwxrwxrwt   2 root root 4096 Aug  1 13:25 tmp
drwxr-xr-x  14 root root 4096 Aug  1 13:22 usr
drwxr-xr-x  11 root root 4096 Aug  1 13:25 var
```

Hands-on

Exercise 3a

1. List the content of an **Ubuntu** Container using run
2. Copy a file in the container
3. List the content of an Ubuntu Container using **start**

Exercise 3b

1. Create a **Nginx** Container
2. List the content of a **Nginx** Container using **exec**
3. Copy a file in the container
4. List the content of an Ubuntu Container using **exec**

EDEM

Hands-on – Iteractive Mode

- “docker run –it ubuntu”
- “docker start –i <container_id>”

- “docker exec –it <container_id> bash”

```
root@ESPC012268:/mnt/c/Users/aohz/git/edem/docker_1# docker run -it ubuntu bash
root@a59581325b42:/# ls -l
total 48
lrwxrwxrwx   1 root root    7 Aug  1 13:22 bin -> usr/bin
drwxr-xr-x   2 root root 4096 Apr 18 10:28 boot
drwxr-xr-x   5 root root  360 Sep  2 13:44 dev
drwxr-xr-x   1 root root 4096 Sep  2 13:44 etc
drwxr-xr-x   2 root root 4096 Apr 18 10:28 home
lrwxrwxrwx   1 root root    7 Aug  1 13:22 lib -> usr/lib
lrwxrwxrwx   1 root root    9 Aug  1 13:22 lib32 -> usr/lib32
lrwxrwxrwx   1 root root    9 Aug  1 13:22 lib64 -> usr/lib64
lrwxrwxrwx   1 root root   10 Aug  1 13:22 libx32 -> usr/libx32
drwxr-xr-x   2 root root 4096 Aug  1 13:22 media
drwxr-xr-x   2 root root 4096 Aug  1 13:22 mnt
drwxr-xr-x   2 root root 4096 Aug  1 13:22 opt
dr-xr-xr-x 179 root root    0 Sep  2 13:44 proc
drwx------   2 root root 4096 Aug  1 13:25 root
drwxr-xr-x   5 root root 4096 Aug  1 13:25 run
lrwxrwxrwx   1 root root    8 Aug  1 13:22 sbin -> usr/sbin
drwxr-xr-x   2 root root 4096 Aug  1 13:22 srv
dr-xr-xr-x  11 root root    0 Sep  2 13:44 sys
drwxrwxrwt   2 root root 4096 Aug  1 13:25 tmp
drwxr-xr-x  14 root root 4096 Aug  1 13:22 usr
drwxr-xr-x  11 root root 4096 Aug  1 13:25 var
```

Hands-on

- **Exercise 4 (Optional)**

  1. Get into an Ubuntu Container using run and list its content
  2. Get into an Ubuntu Container using start and list its content
  3. Get into an Ubuntu Container using exec and list its content

EDEM

Hands-on – Monitoring

- “docker logs <container_id>”
- “docker top <container_id>”
- “docker stats <container_id>”

```
UID                 PID                 PPID                C                   STIME               TTY                 T
IME                 CMD
root                829                 809                 0                   07:41               ?                   0
0:00:00             nginx: master process nginx -g daemon off;
systemd+            887                 829                 0                   07:41               ?                   0
0:00:00             nginx: worker process
systemd+            888                 829                 0                   07:41               ?                   0
0:00:00             nginx: worker process
systemd+            889                 829                 0                   07:41               ?                   0
0:00:00             nginx: worker process
systemd+            890                 829                 0                   07:41               ?                   0
0:00:00             nginx: worker process
systemd+            891                 829                 0                   07:41               ?                   0
```

```
CONTAINER ID   NAME                 CPU %     MEM USAGE / LIMIT   MEM %     NET I/O       BLOCK I/O   PIDS
551986a4bd6d   flamboyant_dubinsky   0.00%     7.277MiB / 12.3GiB   0.06%     1.16kB / 0B   0B / 0B     9
```

- “docker inspect <container_id> / <image_id>”
- “docker system df”

```
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          29        1         16.9GB    16.75GB (99%)
Containers      1         1         19.62MB   0B (0%)
Local Volumes   1         0         1.463GB   1.463GB (100%)
Build Cache     0         0         0B        0B
```

Hands-on

Exercise 5

1. Run a nginx container
2. Get information about the processes running on it
3. Get information about memory usage
4. Open another terminal:
5. Enter the container and run “apt-get update”
6. Any changes?

Hands-on - review

- **Exercise 6**

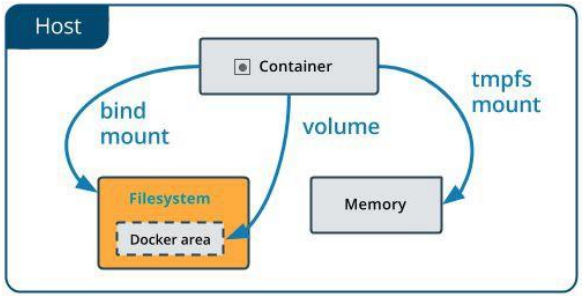
1. Run a Fedora Docker container
2. run wget Google.com
3. Is wget installed?
4. Install wget
5. Stop and start the container: is wget installed?
6. Install wget
7. Create a new container from the same image: is wget Installed?

TIP!

To install wget:
- yum upgrade
- `yum install wget`

Docker Volumes and Bind Mounts

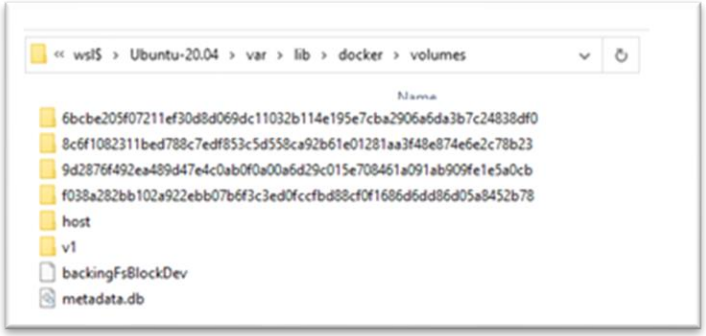
- What happens to the data if a container crash o removed?
  - Data could be lost!!!
- Docker has **two options** for containers to store files in the host machine:
  - Volumes (Anonymus and Nammed)
  - Bind mounts
- Volumes and Bind Mounts have the following **features**:
  - They allow to persist the information when the container is destroyed
  - They allow to share information with the host machine
  - They allow to share information with other docker containers



Docker Volume and Bind Mounts

Anonymous Volumes	Named Volumes	Bind Mounts
Created specifically for a single container	Created in general – not tied to any specific container	Location on host file system, not tied to any specific container
Survives container shutdown / restart unless --rm is used	Survives container shutdown / restart – removal via Docker CLI	Survives container shutdown / restart – removal on host fs
Can not be shared across containers	Can be shared across containers	Can be shared across containers
Since it's anonymous, it can't be re-used (even on same image)	Can be re-used for same container (across restarts)	Can be re-used for same container (across restarts)

Hands-on – Volumen & Bind Mounts



- Basic Volume Commands
  - docker volume create myvol
  - docker volume ls
  - docker volume inspect myvol
  - Docker volume prune
- Attaching **Volumes** to a container
  - **Named Volume**: docker run **-v <volume_name>:/<container_path>** <image_id>
  - **Anonymous Volume:** docker run **-v :/<container_path>** <image_id>
- Attaching **Bind Mounts** to a container
  - docker run –v **/<host_path>:<container_path>** <image_id>

Hands-on

Exercise 7a

1. Create an Ubuntu container with a volume mounted (ubuntu1)
2. Verify that the volumen has been created
3. Create another Ubuntu container with the same volume mounted (ubuntu2)
4. Create a file inside the mounted folder using ubuntu1
5. Verify it exists on Ubuntu2

Exercise 7b

1. Do the same thing with Bind mounts
2. Verify that no volumes are created

Hands-on

- **Exercise 8**

1. Create a docker container:

1. use <httpd> docker Image

2. Expose the port 80 (-p 80:80)

2. Mount docker path “/usr/local/apache2/htdocs/ “ to the host file system

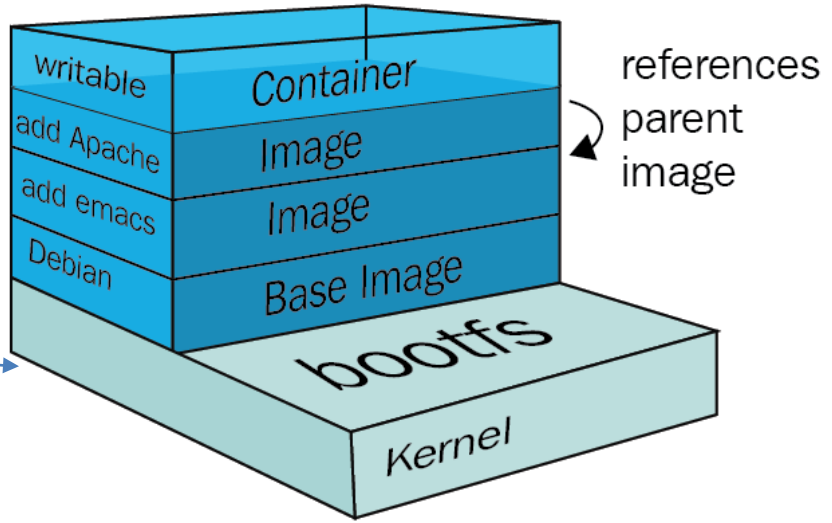
3. Create a index.html file inside the mounted folder

4. Type in your Browser localhost:80

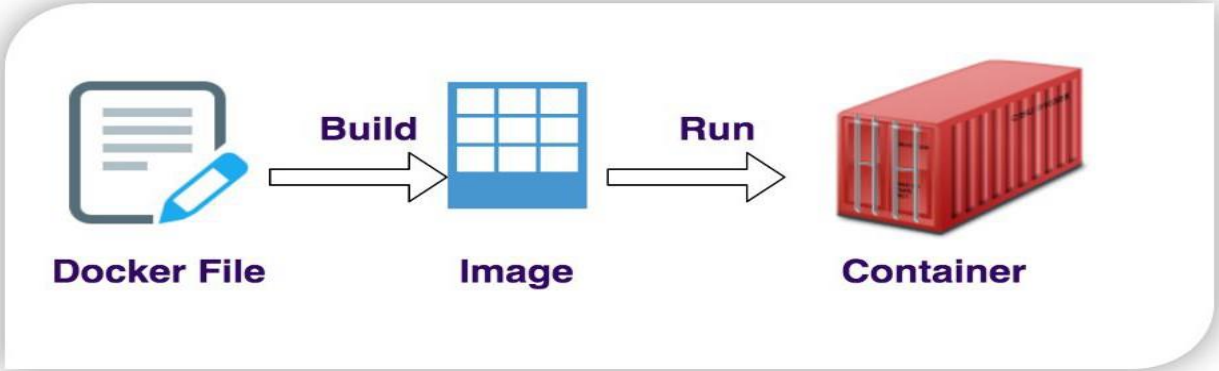
Docker Image Layers

- Docker images are made up of several read-only layers

- Assemble the last layer, the container (read/write)
- Assemble the N layers of the image (read)
- Assemble the root systems (read)
- Assemble the Boot System (read)



Docker File



Docker File

- A **DockerFile** is a text document that contains all the commands a user could call on the command line to **assemble an image**
  - You can consider DockerFile as **blueprint** of Docker Image

```
Dockerfile > ...
1   FROM ubuntu:18.04
2
3   RUN apt-get update
4   RUN apt-get install -y nginx
5
6   COPY entrypoint.sh /
7   COPY index.html /var/www/html/index.html
8
9   EXPOSE 80
10
11  ENTRYPOINT ["/entrypoint.sh"]
```

- DockerFile as a sequential set of instruction for Docker Engine
  - Order of sequence is important!!
  - Each instruction creates a layer
  - A stack of such sequenced layers managed by a filesystem becomes a docker image
  - Layers can be cached and reused by Docker

- “docker build .”: Command for building the docker image
- “docker run <image>”: Command for creating the docker container from the docker image

Hands-on – Build Image

- “docker build .”
- “docker build –t <tag_name> .”
  tag_name: {repository}/{image_name}:{version}

- “docker run <image>”

```
root@ESPC012268:/mnt/c/Users/aohz/git/edem# cd docker_0
root@ESPC012268:/mnt/c/Users/aohz/git/edem/docker_0# docker build .
Sending build context to Docker daemon  2.048kB
Step 1/5 : FROM ubuntu:18.04
18.04: Pulling from library/ubuntu
Digest: sha256:6fec50623d6d37b7f3c14c5b6fc36c73fd04aa8173d59d54dba00da0e7ac50ee
Status: Downloaded newer image for ubuntu:18.04
 ---> 35b3f4f76a24
Step 2/5 : RUN apt-get update
 ---> Using cache
 ---> aec74bb326db
Step 3/5 : RUN apt-get install -y nginx
 ---> Using cache
 ---> 50d3608b077e
Step 4/5 : EXPOSE 80
 ---> Using cache
 ---> 2f6952c0a3ca
Step 5/5 : ENTRYPOINT /usr/sbin/nginx -g "daemon off;"
 ---> Using cache
 ---> 8f612df28686
Successfully built 8f612df28686
root@ESPC012268:/mnt/c/Users/aohz/git/edem/docker_0#
```

```
Successfully built 8f612df28686
root@ESPC012268:/mnt/c/Users/aohz/git/edem/docker_0# docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
<none>        <none>    8f612df28686   11 minutes ago   165MB
ubuntu        18.04     35b3f4f76a24   20 hours ago     63.1MB
root@ESPC012268:/mnt/c/Users/aohz/git/edem/docker_0#
```

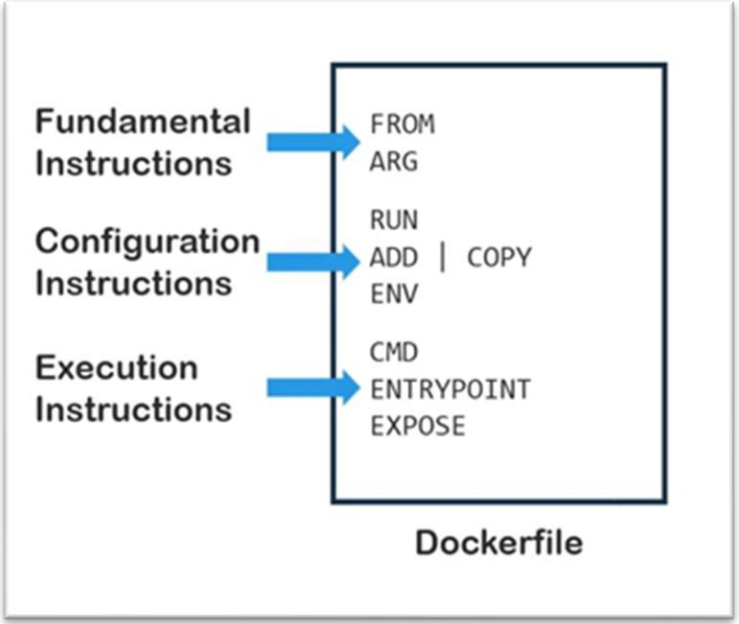
Docker File Structure

- It's a file with no extension called "**Dockerfile**"
- The instructions can be generally divided into three categories :
  - Fundamental
  - Configuration

  They are executed when the image is **build**
  - Execution:

  They are executed when the container is **run**

- Both CMD and ENTRYPOINT instructions define what command gets executed when running a container



Hands-on

Exercise 9

1. Create a new DockerFile to install git on Ubuntu 20.04
2. Build the docker image
3. List the docker images
4. Check the git version installed

TIP!

To install git:
- *apt-get update*
- *apt-get install -y git*

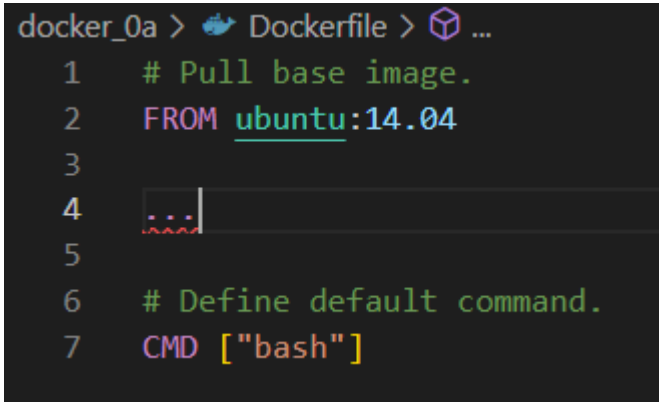
CMD and ENTRYPOINT

- Both CMD and ENTRYPOINT instructions define **what command gets executed when running a container**
- Dockerfile should specify at least one of CMD or ENTRYPOINT commands
- The **ENTRYPOINT** specifies a command that will always be executed when the container starts.
- The **CMD** specifies commands/arguments that will be fed to the ENTRYPOINT
- If the **ENTRYPOINT** is not defined, CMD is executed

CMD

- The **CMD** specifies commands/arguments that will be fed to the ENTRYPOINT
- A Dockerfile may contains multiple **CMD** instructions but only the last one is run
- **CMD** can be overridden via command line

- Use Cases:
  - **General Purpose Image** where the command can be customized when docker run is invoked



1. *$ docker run ubuntu* -> run “bash” as it is set in the dockerfile
2. *$ docker run ubuntu ls -l* -> run “ls –l ” because it overrides bash command
3. *$ docker run ubuntu cat /etc/hostname* -> run “cat /etc/hotmane”

ENTRYPOINT

- The **ENTRYPOINT** specifies a command that will always be executed when the container starts.
- The **CMD** specifies commands/arguments that will be fed to the ENTRYPOINT
- Use Cases:
  - **Single Purpose Image** where the command can't be overriden

```
docker_0a > Dockerfile > ...
1   FROM debian:wheezy
2   ENTRYPOINT ["/bin/ping"]
3   CMD ["localhost"]
```

$docker build –t myping .

1. *$ docker run myping* -> run "/bin/ping localhost"
2. *$ docker run myping ls -l* -> run ""/bin/ping ls –l "
3. *$ docker run myping google.com* -> run "/bin/ping google.com"

```
docker_0a > Dockerfile > ...
1   FROM debian:wheezy
2   CMD ["/bin/ping", "localhost"]
3
```

$docker build –t myping2 .

4. *$ docker run myping2* -> run "/bin/ping localhost"
5. *$ docker run myping2 ls -l* -> run ""ls –l "
6. *$ docker run myping2 google.com* -> try to run "google.com"

Hands-on

Exercise 10a

1. Create a new DockerFile to install git on Ubuntu 20.04
2. Use **<CMD>** to print the git version when the container runs
3. Build the image
4. Try to get into the container

Exercise 10b

1. Create a new DockerFile to install git on Ubuntu 20.04
2. Use **<ENTRYPOINT>** to print the git version when the container runs
3. Build the image
4. Try to get into the container

Exec and Shell Forms

- Two ways to define the ENTRYPOINT of CMD:
  - **Exec Form**: `["/bin/ping", "localhost"] -> Json Array`
  - **Shell Form**: `/bin/ping localhost`
- Exec form is the preferred way
- Exec form don't expand variables

- Multiple combinations of CMD, ENTRYPOINT with Shell or Exec Form may appear in the same Dockerfile

	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT ["exec_entry", "p1_entry"]
No CMD	*error, not allowed*	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD ["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

Arguments vs Env. Variables

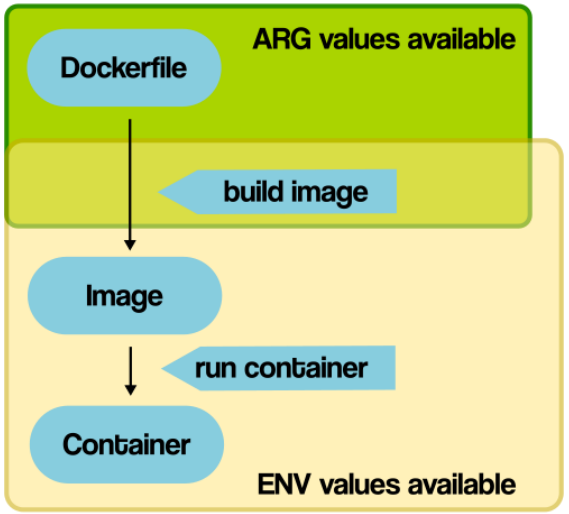
- Both are Dockerfile instructions

Arguments (ARG)

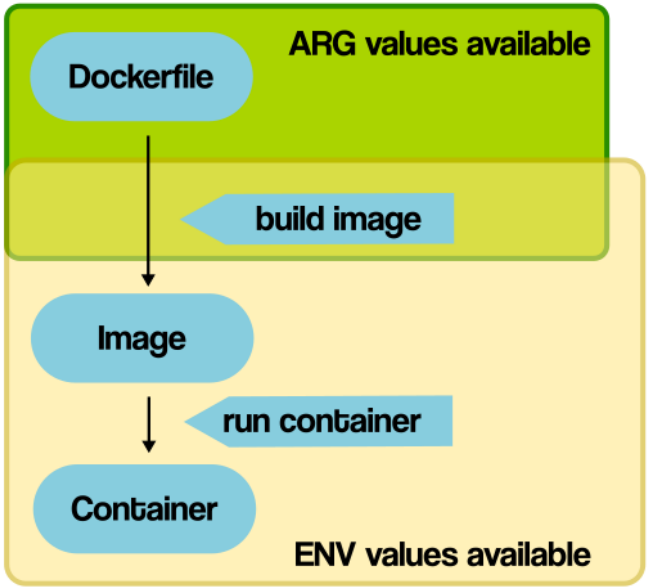
- Args are visible when the image is built
- Args are not visible inside the container (unless we assign its value to an environment variable (ENV env_1=$arg_1)
- Args can be overridden during build time using --build-arg VAR_NAME=6

Environment Variables (ENV)

- Env are visible during the image build and inside Docker containers
- Envs can't be overridden during build time



Arguments vs Env. Variables



Dockerfile content:

```
ARG required_var       # expects a value
ARG var_name=def_value # set default ARG value
ENV foo=other_value    # set default ENV value
ENV bar=${var_name}    # set default ENV value from ARG
```

Override Dockerfile ARG values on build:

```
$ docker build --build-arg var_name=value
```

Override Docker image ENV values on run:

```
$ docker run -e "foo=other_value" [...]
$ docker run -e foo [...] # pass from host
$ docker run --env-file=env_file_name [...] # pass from file
```

https://vsupalov.com/docker-env-vars

Arguments vs Env. Variables

docker_2a > Dockerfile > ...

```
1  FROM ubuntu:20.04
2  ARG VERSION=0.0
3  RUN echo "Version in RUN: $VERSION"
4  CMD echo "Version in CMD: $VERSION"
5
```

What is printed …?

1. In line 3, running `docker build .` ?
2. In line 3, running `docker build –build-arg VERSION=1.0` ?
3. In line 4, running `docker run <container>` ?
4. In line 4, running `docker run <container> –build-arg VERSION=1.0`?

docker_2b > Dockerfile > ...

```
1  FROM ubuntu:20.04
2  ENV VERSION=0.0
3  RUN echo "Version in RUN: $VERSION"
4  CMD echo "Version in CMD: $VERSION"
5
```

What is printed …?

5. In line 3, running `docker build .` ?
6. In line 3, running `docker build -e VERSION=1.0` ?
7. In line 4, running `docker run <container>` ?
8. In line 4, running `docker run <container> -e VERSION=1.0` ?

Hands-on

Exercise 11

1. Create an image from Ubuntu:18.04
2. Install nginx
3. Create a index.html file with the following content
4. Copy the index.html file to /var/www/html/index.html
5. Create and use entrypoint.sh script
6. Use entrypoint.sh script as ENTRYPOINT to start nginx

```
<html>
<head>
  <title> Ubuntu rocks!
</title>
</head>
<body> <p> This is my website!
</p> </body>
</html>
```

Index.html

```
#!/bin/bash
/usr/sbin/nginx -g "daemon
off;"
```

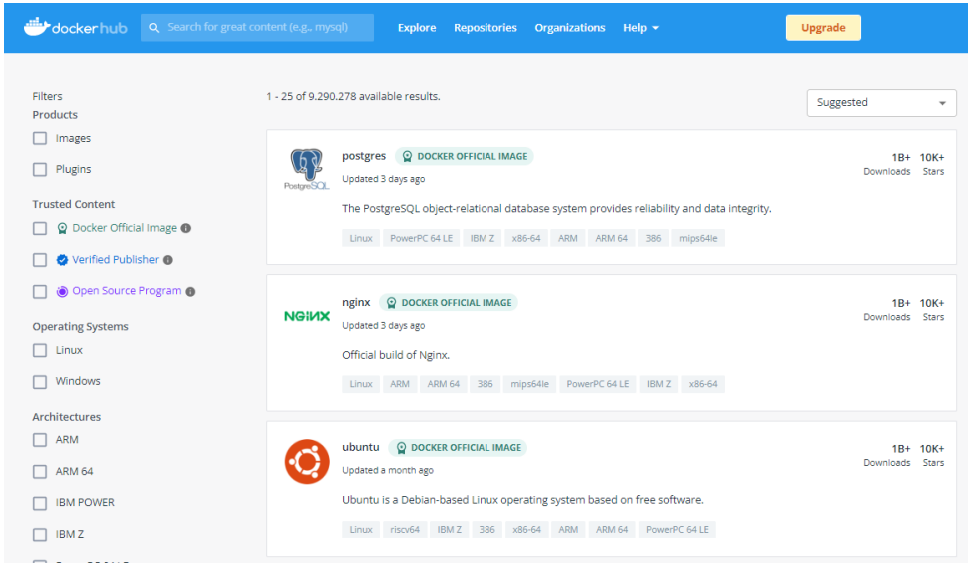
entrypoint.sh

Bonus!

- Change the Docker image to allow us to modify the webpage body on runtime

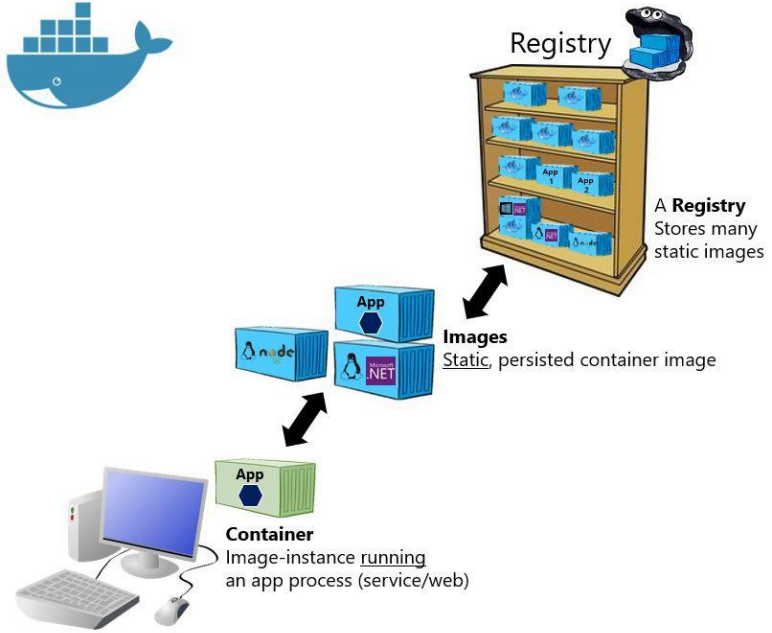
Docker Registry

- The Registry is a stateless, highly scalable server side application that stores and lets you distribute Docker images
  - Fully own your images distribution pipeline
  - Locally or using Docker Hub



- Docker login <Registry>
- Docker tag SRC_IMAGE[:TAG] TARGET_IMAGE[:TAG]
- Docker push <IMAGE>

Basic taxonomy in Docker



On-premises
('n' private organizations)

- **Hosted Docker Registry**
- **Docker Trusted Registry on-prem.**

Public Cloud
(specific vendors)

- **Docker Hub Registry**
- Docker Trusted Registry on-cloud
- **Azure Container Registry**
- AWS Container Registry
- Google Container Registry
- Quay Registry
- Other Cloud

EDEM

Docker Resource Consumption

- By default, a container has **no resource constraints**
- Docker provides ways to control how much memory, or CPU a container can use, setting **runtime configuration flags** of the docker run command
- If the kernel detects that there is **not enough memory**, it throws an Out Of Memory Exception and **any process is subject to killing**, including Docker and other important applications
- **Main Configuration Flags**:
  - **--memory**: The maximum amount of memory the container can use
  - **--memory-reservation**: The maximum amount of memory the container can use when low memory is available on the host machine
  - **--cpus**: Specify how much of the available CPU resources a container can use
- It is possible allow a container to **access GPU resources** installing the right drives
- **$**docker run -it --cpus=".5" **--memory-reservation='100m' –memory='1g'** ubuntu /bin/bash

Hands-on – House Keeping

- “docker system df”
- “docker system prune”

```
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all dangling images
  - all dangling build cache
```

- “docker container rm <container / containers>“
- “docker image rmi <image / images >“
- “docker volume rm <volume / volumes>”
- “docker volume prune <volume>”

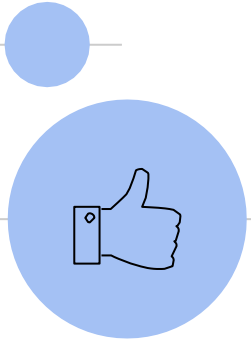
Hands-on

- **Exercise 11**

1. Stop all containers
2. Get information regarding the amount of disk space used by docker
3. Remove all unused resources
4. Remove last image, any Issue?
1. Remove last container and try again
5. Remove all containers
6. Remove all images
7. Remove all unused volumes

TIP!

- Docker commands can be concatenated with other commands
- Eg -> docker stop $(docker ps -q)



Thanks!

Any Questions ?

You can find me at

https://www.linkedin.com/in/alberto-hernandez-cantos-9502a8a3/

alhercan@gmail.com

1

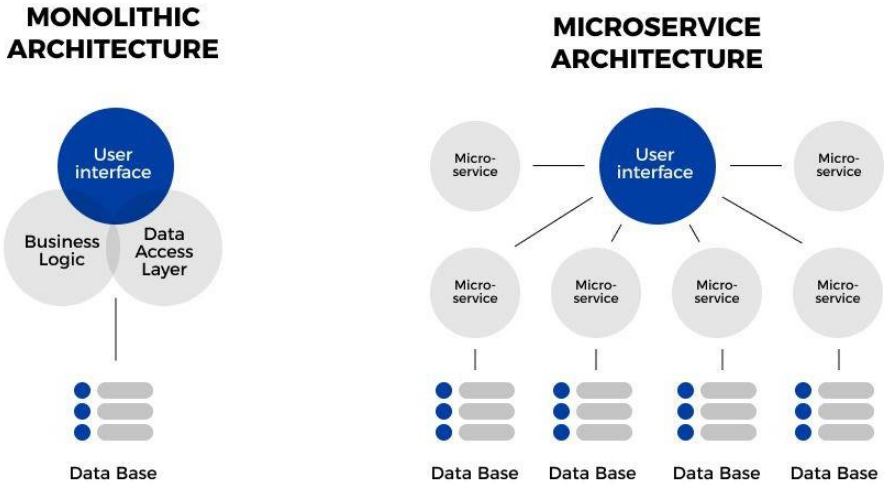
Microservice Architecture

Microservices

- Microservice Architecture is an architectural style that structures an application as a collection of services that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities
  - Owned by a small team

- The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications
  - It also enables an organization to evolve its technology stack

Microservice Architecture



- **Developer issues:**
  - Minor code changes require full re-compile and re-test
  - Application becomes single point of failure
  - Application is difficult to scale

- **Microservice:**
  - Break application into separate operations
  - Make the app independently, scalable, stateless, highly available by design