Introduccion

Containers

- Might be described as creating a small software package that can ru na particular app and its associated processes

Container becomes portable and can run on all types of computers in the same way without compataibility issues and dependencies. This is why Docker refers to a container as a **Standarized unit of Software**.

Benefits

- Not disruptive to a system
- Easy clean up afterwards
- No conflicting versions of software or libraries
- Lightweight: Optimized for a medium-powered laptop.
- Can run multiple containers on one server
- Orchestrators can help organize and run multiple containers, as Kubernetes or Openshift

VM vs Containers

VMs might have different apps present on a single VM. There might be a web server and a database along their backup scripts.

The container is using the underlying host computer OS, and not starting its own whenever it launches

Keys

- Isolate containers from one another
- control groups wich provide resource kernel quotas. Limit onecontainre from using up all the resources, such as the CPU or disk reads and wirtes on the host machine.

Dockerfile Basic Sample

In order to create a Docker image, a Dockerfile is required. This is a simple script that determines how an image is built frmo scratch.

One you use docker to build an image from a Dockerfile, you can run as many identical containers from it as you wish.

Base Images and Layering

Base images are important part of Docker and involve an image which you build your own image upon. Base images usually come with a tiny but functional OS.

Docker uses a layered filesystem that allows to add a base image without making your image much larger.

Whenever you want to create a container image, will be used. Docker will kown wisch base image to use when you specify a FROM line in the Dockerfile.

Microservices

Docker containers are better suited to a microservice architectura than a traditional monolithic architectura because a container can run autonomously without relying on the other centrlized processes.

```
FROM debian:stable
LABEL author=LinuxTrainingAcademy
LABEL Version=dockerbook

RUN apt update &&
    apt install -y nano &&
    apt clean

CMD ["/bin/bash"]
```

Docker Basics and Common Commands

Running a Container

When you run a container, the image that you are using will be downloaded, if it has not been already. If the image already exists, then there is nothing to download. if the image is not on oyur host system, Docker will download it for you.

```
docker run -dit debian
```

After Docker downloads the image, it starts a container using that image. The last line of the output is the container ID.

You can use the container ID to manage this particular container. You can use also a unique human-friendly name or a unique portion of that ID.

Options

- d: run container in the background
- t Allocates paseudo TTY or a pseudo terminal.
- i option stands for interactive. This allows you to type commands in the container.

Show Runing Containers

```
docker ps
```

```
docker stop 'id'
docker run nameimage
docker ps
```

```
# Display Images
docker images

# Inspecting Resources
docker inspect %%first thtree hash chars of CID%%
# Shows networking details, troubleshooting, settings, kernel settings,
etc.

# Assistance
docker --help
doker image --help
odcker image prune --help
```

Managing Docker Container Images

Download, view its history, tag, delete, view, clean up, for **IMAGES**

Layering Mode of a Container

- All docker images based on the same image, use disk space only to take up storage for added files to the base image.
- The layers in an image are read-only and each new layer is DELTA of any changes compared to the layer beneath it.
- Similar as Git control system. This speed ups the rebuilding images after small change has been made.

Download Images

```
docker pull nginx
# takes an image name as an argument
# it downloads that img to the local host system
```

```
# check images
docker images
```

```
# see how img was constructed
docker histroy nginx
# the output shows each layer line by line

# Tags
docker tag --help
# Show useful information
# If no tag, show 'latest'

docker tag nginx:latest nginx:myglog_stable
# create two images with the same ID but different alias
```

Dockerfiles

Dockerfiles can range from being very simple to very complex.

Example simple dockerfile

```
FROM debian:buster-slim
LABEL maintainer='NGINX Docker Maintainers <docker-maint@nginx.com>"
```

```
ENV NGINX_VERSION 1.21.1
ENV NJS_VERSION 0.6.1
ENV PKG_RELEASE 1~buster

RUN apt-get update \
    && apt-get install -y nginx \
    && apt-get clean

RUN ln-sf /dev/stdout /var/log/nginx/access.log \
    && ln -sf /dev/stderr /var/log/nginx/error.log

COPY index.html /var/www/html

EXPOSE 80
cmd["nginx","-g","daemon off;"]
```

- FROM: base image
- dockerbuild means that a layer will be created on the top of the base image in your image
- LABEL: entry is a good way of offering contact details for the author
- ENV introduce environment variables, like bash to use
- RUN instruction tells docker to run a shell command to help build your image
- COPY command copies a file from the local system into the image at build time. This copy statement will put a copy of the index.html file in the /var/www/html directory. COPY command is most commonly used to copy small config files into an image
- EXPOSE entry is telling Docker to open up TCP port 80 when a container runs from the image.
- CMD instruction runs the nginx binary with two options from inside the container
- daemon off keep ngninx running in foreground of the container. Elsewhere the container would stop running as soon as it started

Generic Tips

- Having fewer layers is generally a good idea
- 'apt-get clean' is a good practice to add command on debian based distros

Build

- `docker build -t mynginx .`
- -t option allows specify a tag for the image
- docker rmi nginx

Remove Images

- docker rmi --help
- docker rmi -f nginx:latet -> Force
- docker rmi --no-prune -> To not dfelete untagged parents

Clearing disk Space

- Non tagged images listed as tag are called DANGLING IMAGES
- docker image prune -> Remove all dangling images
- docker system prune -a
    - stopped containers
    - networks not used by at least one container
    - All images with at least one container associated to them
    - all build cache
- docker system prune
    - all stopped containers
    - All networks not used by at least one container
    - all dangling images
    - all dangling build cache
- docker system df -> see disk space docker is using

Exercise: Managing Images

Download the IMages

```
- use ```docker pull memcached:latest```
- download the apache image called httpd ```docker pull httpd:2-alpine
    - Contains apache version 2 running on Alpine Linux distro
- Check the images ```docker images```
```

Runnnig and Managing Docker Containers

Concepts

- To start a container and keep it running in the background you need to explicitly tell Docker to do so
- To keep it running you have to 'detach' it
- using -d to daemonize
- `docker run -dit debian`
- t & i works together for interact with the container os
    - i for interactivity
    - t for TTY

Naming

Similar to IP addresses and DNS, you acn give a conatiner name and refer to it this way, instead of its container ID.

- `docker run -dit --name=web debian`
- `docker ps`

Removing

1. STOP: `docker stop web`
    - yoy can also start a stopped container using `docker start`
2. REMOVE: `docker rm web`

Always Restart Policy

- If a container crashes for some reason, you might want it to restart on its own instead of waiting for a human to manually restart it
- `docker run -dit --restart=always name=container_name debian`
-
- OUTPUT: Name: always, MaximunRetryCount: 0;

if a container stops for any reason and no restart policy was specified, the conatiner will remain stopped since the default restart policy is 'no'.

Without supply a restart policy, if the container stops due to an error, docker engine restarts, or host machine is rebooted, the container will remains stopped.

Policies

- default
- on-failure -> restart due to an error

- unless-stopped: it will remain stopped if before a system reboot or engine restart the user manually stopped
- always

Stop

- docker stop container_name
- docker kill container_name

Kill it skips graceful shutdown. Kill might put the container in such a state that it will not be able to restart.

Sopped Containers

- DELETE STOPPED CONTAINERS -> `docker system prune`
- this will remove all stopped containers

Auto Deting Stopped Container

- `docker run --rm hello-world`

Exercise: Running Containers

- START A CONTAINER: docker run -dit redis
- START WITH A NAME: docker run -dit --name=redis_container redis
- STOP A NAMED CONT: docker stop container_name
- AUTO DELETE STOPPED CONTAINER: docker run --rm container_name

Logs

docker logs provides information directly from the STDOUT and STDERR of a container

- docker logs CONTAINER_NAME

Making a Container Publicly Available

- How to make an app that is running inside a container accesible from outside the Docker host machine.
- How to share data with containers running on your machine

Start up a container and open a specific netowrk port so that you can view the container web server in a web browser on your local machine. Pick an arbitrary TCP port number on your host machine to expose our container's network port on.

```
docker run  - -name our_nginx -d -p 8080:80 nginx
```

- `-p 8080:80` means open up TCP port 8080 on the docker host machine and redirect traffict to and from it, to TCP port 80 inside the container.
- `outside:inside`
- check with docker ps
- output in PORTS column 0.0.0.0:8080 -> 80/tcp

- La direccion IP 0.0.0.0 es conocida como direccion no especifica. El nombre tecnico original es "source asddress for this host on this network" *direccion de origen para este host en esta red*

- La IP 0.0.0.0 significa que todas las direcciones IPv4 se encuentran en el equipo local.

8. curl http://localhost:8080

- If docker hostm achine is remote, and you want to connect it from another system over the network, you will need the ip address of your docker host machine

- If you are accessing the docker host via SSH from your laptop you will need to determinate the IP address of the docker host system. on linux systems run `ip a`

you can get the logs running `docker logs container_name`

making webpages

```
mkdir webpages
cd webpages
echo 'hi from inside the container! > index.html
cd ..
```

```
docker run -p 8080:80 –name another_nginx -v
${PWD}/webpages:/usr/share/nginx/html:ro -d nginx
```

- -v option creates a volume for sharing files.
- -v followd by the path on the host machine followed by a : and where you want that data to be accessed in the container. after second semicolon can add options as `ro` wich mounts the volume in read only mode inside the container. This means that files can be changed from the docker host machine, but the container cannot alter the files.

Exercise: Making a container publicly available

```
# Start a container using Apache HTTP Server Image
# Image for apache http server is httpd
docker run –name apache_welcome -d -p 9900:80 httpd:latest
docker ps
curl http://localhosts:9900
```

Connecting to Running Containers

To access a container that is already running, docker priovides `docker exec` command

```
docker run -it --name apache httpd /bin/bash```
```

After executing it, you will be presented with a prompt inside the container and hostname root@container-id:/path/to/working/directory

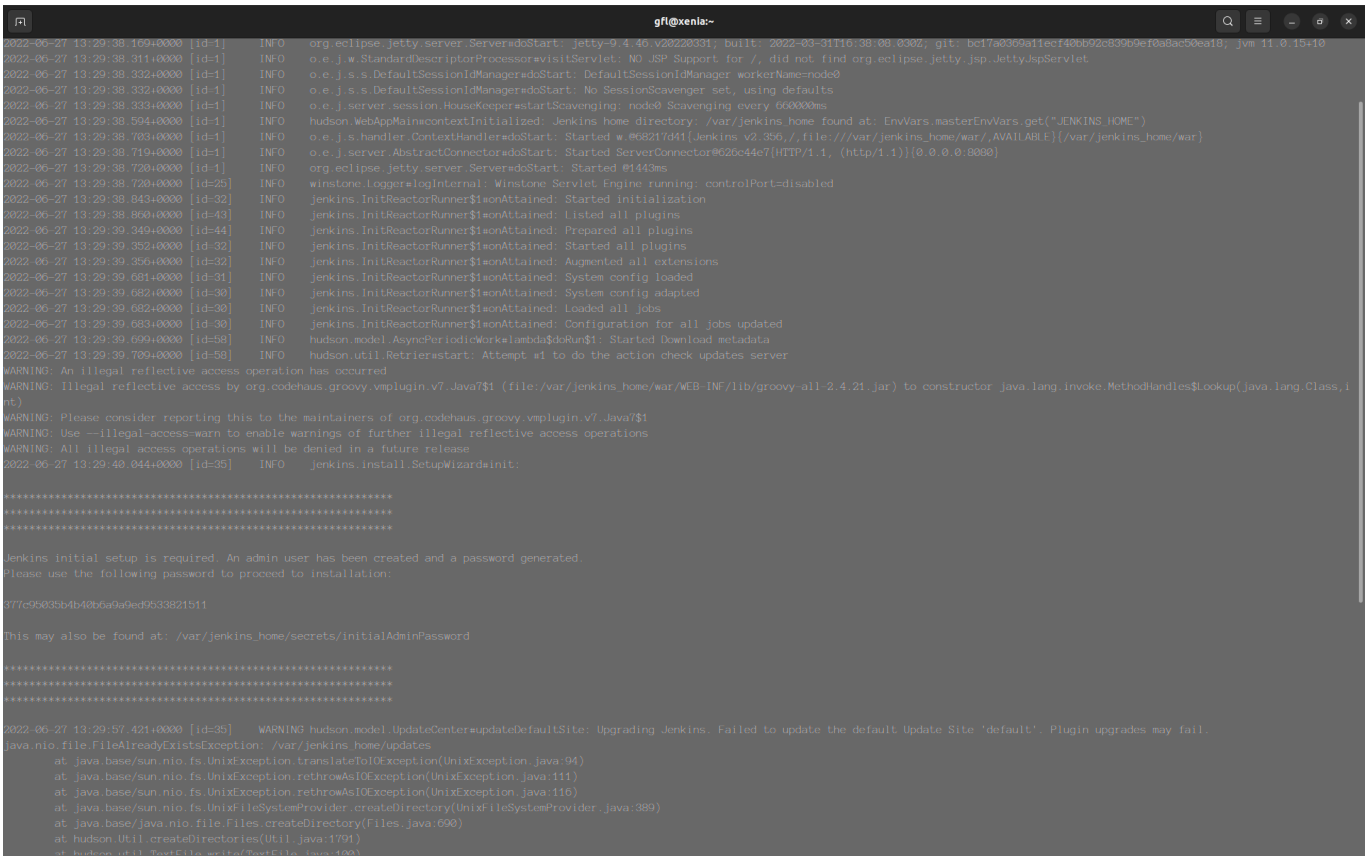
- `-it` options are required if you want to use an interactive shell
- `/bin/bash` es el comando para ejecutar dentro del conteiner. Analogo a la instruccion CMD del Dockerfile
- Se puede utilizar `dit` de igual manera

Connecting to Containers

```
docker run -it --name enter_redis redis /bin/bash
bash -version
exit
docker ps -a
# Container status EXITED
```

Docker Logging

```
docker run -dit --name busylogs -p 8080:8080 -p 50000:50000
jenins/jenkins:lts
docker logs busylogs
```



- Follow the changes: `-f`
- Timestamps: `-t`
  - `docker logs -t nombrecontainer`
  - `docker logs -t nombrecontainer --since 2022-01-01`
  - Logs recorded in `/var/log` directory

External Logging

Docker engine creates logs, sends its to the host machine syslog system - Linux: /var/log/messages - Debian/Ubuntu: /var/log/syslog

Journal Command

displays the last 20 entries in the logs from all the service on the system

```
journalctl -n 20
journalctl -u docker.service -n 20 # Filter for service
journalctl --no-pager -u docker.service -n 20
```

Docker Registries

The primary function of a registry are

- Store image files
- Pulling images down
- Push images back up

A repository within a registry is a collection of **one or more images**. A single repo can hold many docker images, each stored as a **TAG**. The tag refers to a version of the image, buy they also can refer different variations of an image.

```
docker pull docker.io/ubuntu:focal
docker pull registry.hub.docker.com/library/ubuntu:focal
# Same tag but different DNS name.

# If we check the images
docker images
>> ubuntu
>> registru.hub.docker.com/librar/ubuntu
# Equial Image ID. They are the same image

# RegistryAddress/Repository/Image:Tag
```

Credentials

How wolud log in, if a repository is set to private. There are a number of popular container image registries such Amazon Elastic Container Registry, Quary from Red Hat and Google's Container Registry.

For business apps it's common to have your own registry. Docker offers the option to make repositories private, requiring you in to gain access to the registry. Examining the trade=off between running your own registry for complete control or using a vendor, will usually help business decide wether or not host their images.

To access a private registry use the `docker login` command. In order to create and access your on repos, create an acc on docker Hub.

Docker Hub

DockerHub is the default registry, and for this reason you don;t have to specify a registry name with the docker login command. You can use this command to access images hosted on docker hub.

```
docker login OPTIONS SERVER -u username -p password
```

Dockerfiles: Building Images

- FROM: where to pull a base image from. It will be the base of your particular image.
- CMD: Instruction for running a shell command inside the container, or argument to another command
- RUN: Executes a command to help build image
- EXPOSE:Opening Network Ports
- VOLUME: allow you to specify a disk share
- COPY: used for copy files into the image from local disk
- LABEL: Add metadata as key-value par
- ENV: Environment variable to a container
- ENTRYPOINT: Shoudl be used almost at the end of the file to specify which executable will run when your container starts.
    - the following CMD entry after ENTRYPOINT should the pass options to the executable if required.

```
FROM debian:latest
LABEL maintainer='Your Name'
ENTRYPOINT['/bin/ping']
CMD['google.com']
```

```
# Building the dockerfile
docker  build -t username/dockerping:latest .
# Docker_ID/Repository:Tag
```

1. `docker build` expects a path to where it can find the Dockerfile that it will use to build the image. First docker downloads the base image, which came from FROM instruction in the Dockerfile. Next, docker ran all the commands and instructions in the dockerfile to create the resulting image.

```
# To upload it to Dcker Hub we need exec ```docker login``` command.
docker login
docker push Docker_ID/Repository:Tag
```

```
# 1. Docker Engine downloads the base image from the specified path
FROM debian:latest
LABEL maintainer='John Doe'
# 2. Docker engine runs ENTRYPOINT command /bin/ping
ENTRYPOINT ['/bin/ping']
# 3. Docker uses www.google.com as the defaul command argument.
CMD ['www.google.com']
```

```
# The CMD instruction can be overriden by supplying an option on the
command line
```

```
FROM debian:latest
LABEL maintainer=name
RUN apt update && \
    apt install -y traceroute && \
    apt install -y curl && \
    apt clean
ENTRYPOINT['/bin/bash']
```

```
docker build -t user/nettools nettools/
#               #ImageBase     #DirectoryPath
# -t option for tag
```

Override Entry Point

```
docker run --entrypoint /usr/bin/curl -it username/nettools docker.com
```

Dockerfile Default File Name

- `-f` option provide the name of the file when dockerfile has a diferent name tha 'Dockerfile'

```
docker build -t username/nettools:AlternativeTag -f
DockerfileAlternativeNam&& \e
```

Exercise: Build and push an image

1. Create an account on Docker Hub to host your own images: https://hub.docker.com/signup

Creating a Dockerfile

```
mkdir tempbuild
cd tempbuild
nano Dockerfile
```

```
# Creating the Dockerfile
FROM debian:stable
LABEL maintainer="ENTER_YOUR_NAME_HERE"
```

```
RUN apt update && apt install -y nano && apt clean
CMD ["/bin/bash"]
```

```
docker build -t YOUR_DOCKER_ID/nanotest:latest .          # build image
from dockerfile
docker images                                             # see if nanotest
image is present
docker run --name nanotest -it YOUR_DOCKER_ID/nanotest    # run image
which nano                                                # testing if nano
is available
exit
```

```
# Pushing the image to Docker Hub
docker login
docker push YOUR_DOCKER_ID/nanotest:latest
```

Docker Volumes

- How to persist and save data using Docker volumes.
- How to make data available to a container in read-only mode.
- Share asme data with multiple containers
- How to use ephemereal volumes and how to quickly remove unused volumes

Managing Volumes

- Images are bult to keep containers based on the small, portable and disposable
- Images will typically contain only the packages required to provide the intented service

It's best to discard a container without worrying about losing important data. If possible, you don't want important data only exist inside the container.

- TIP: Don;t want important data to only exist inside a container.
    - If you want to persist or save data generated or used by a container -> use a volume
    - A volume is ideal choice to share the same data between multiple containers

Command Options

- The `-v` or `--volume` option in a `docker run` command declare that we wanted to use a volume with a container
- The preferred way to mount volumes is `--mount` option, insthead of `-v` because is easier to use

Creating Volumes

```
docker volume create testdata
>> testdata
docker volume ls
>> ...
>> local testdata
docker volume rm testadata
>> testdata

docker volume create mydata1
>> mydata1
docker volume ls
>> local mydata1
docker volume # Inspect information about a volume
```

Inspect Command Output

- Created At
- Driver
- Labels
- Mountpoint

- Name
- Options
- Scope

Attaching Volumes

```
# Startup a container
# Attaching a volume to it
# => Container has access to the dta in that volume

docker run -d --name withvolume --mount
source=mydata1,destination=/root/volume nginx
docker ps

docker inspect withvolume | grep Mounts -A10
```

Mounts section shows that /root/volume is the volume's destination inside the container, and it is set to read and wirte mode, as indicated by "RW: true" line.

- SOURCE/SRC path: (host) /var/lib/docker/volumes/mydata1/_data/index.html
- DESTINATION/TARGET/DST: (inside the container) /root/volume

```
echo "Hello world from mydata1 volume" >
/var/lib/docker/volumes/mydata1/_data/index.html
cat /var/lib/docker/volumes/mydata1/_data/index.html

docker exec -it withvolume /bin/bash
cd /root/volume
cat intex.html
exit
```

- Docker allows to mount the same volume to multiple containers

Read Write Options

```
docker run -d --name readonlyvolume --mount
src=volume_name,dst=/usr/share/nginx/html,readonly nginx
# readonly and ro are equivalent
```

Ephemereal Volumes

- Ephemereal or Short-Lived volumes are auto-destroyed when the container that started them is stopped.

```
docker run -dit --name ephemereal_volume --mount
type=tmpfs,dst=/root/volume nginx
docker inspect ephemereal_volume | grep Mounts -A10
```

```
#output
Type:: tmpfs,
```

Space limit

```
docker run -dit --name ephe_vol --mount type=tmpfs,tmpfs-
size=256M,dst=/root/volume nginx
# No more than 256M of data can be stored in /root/volume
```

Ephemereal volumes are a powerful way of cleaning up after containers that need extra storage space, but only temporaly.

Limitations: You cannot share tmpfs volumes between containers. These types of volumes disappear after the container stops, they can make it more difficult to diagnose issues.

Volume Types

- `-v` option syntax

```
docker run -p 8080:80 --name nginx-with-vol -v
${PWD}/webpages:/usr/share/nginx/html:ro -d nginx
```

Volumes are less flexible because the mount target is a directory or just a file

Delete Volumes

```
docker rm vol1 vol2 vol3 volN
docker ps -a
docker volume ls
docker volume prune
```

Recap

```
docker volume create localvolume
docker volume inspect localvolume
echo "this file exist" > /var/lib/docker/volumes/localvolume/_data/file.txt
```

```
docker -d --name mountvolume --mount src=localvolume,dst=/data httpd
docker exec -it mountvolume /bin/bash

# inside the container shell
df -h
cat /data/file.txt

echo "Created from inside the container" > /data/from-container.txt
cat /data/from-container.txt
exit

# Outside the container
cat /var/lib/docker/volumes/localvolume/_data/from-container.txt
```

Temporal Volume

```
docker run -d --name tempvolume --mount type=tmpfs,dst=tempdata httpd
docker inspect tempvolume | grep Mounts -A10
```

Docker Networking

Linux Firewall

- Docker makes use of the Linux Kernel's built-in firewall NETFILTER.

Exploring a Container

```
docker dun -dit -p 8080:80 php:apache
```

```
iptables -nL "DOCKER"
#output
tartget prot opt source destination
ACCEPT tcp -- 0.0.0.0/0 172.17.0.2 tcp dpt:80
```

dpt means destination port. So docker is applyin a local, non-routable on the Internet, IP Address to our running container.

```
curl http://172.17.0.2
```

Docker Networking Model

```
curl http://172.17.0.1
curl http://127.0.0.1:8080
```

The output will be the HTML response from the web server. Docker's internal routing is seamlessly forwarding traffic.

Docker Swarm

Docker Swarm is a cluster management and orchestration tool povided by Docker.

Orchestrators

Orchestrators provide a number of diffrerent types of functionality which assist in keeping a containerized appplication live and available to its users.

- Autoscaling: Automatically adds or remove containers to deal with heavier demand or low demand
- Garbage Collection: Guarantee high availabilty by ensuring enough containers are running to provide the service.

Orchestrators can enhance how you might use containers for serving your software or providing your application as a service.