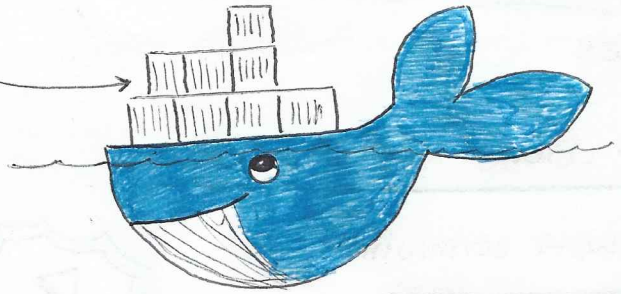
CRASH COURSE INTO DOCKER

Docker CC (1)

Docker : TOOL FOR RUNNING APPS IN ISOLATED ENVIORMENTS

ADVANTAGES on VM's

containers

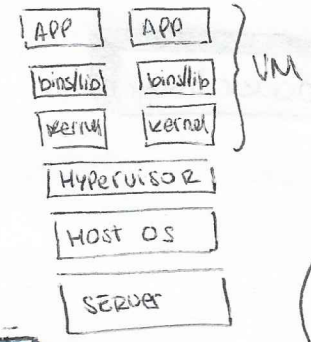


1. Same enviorment
   gives consistency, always acts the same.
2. Sandbox Projects
   keeps them secure, prevent any kind of conflicts
3. It works.

`docker run --help`

≠ with VM's

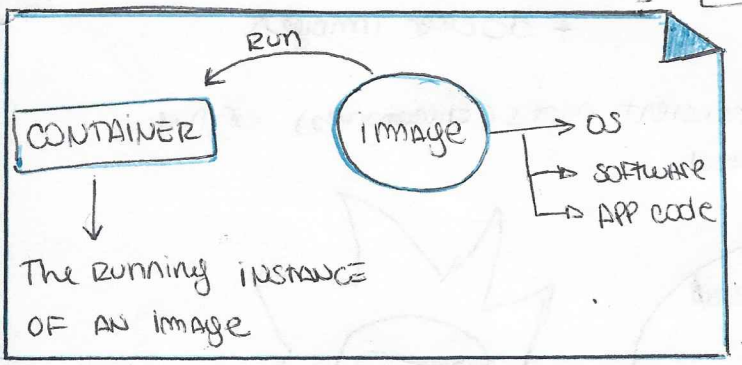
VM →



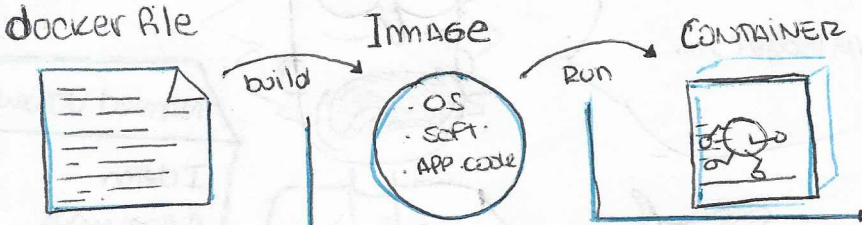
- Fast deploy
- can start up in minuts
- can be shutdown in minutes
- Fewer Resources
- Less memory and less disk space

"shares the kernel with the HOST"

Ubuntu/debian/... all build on the same kernel that docker uses to build their containers



containers can run specifics tasks.



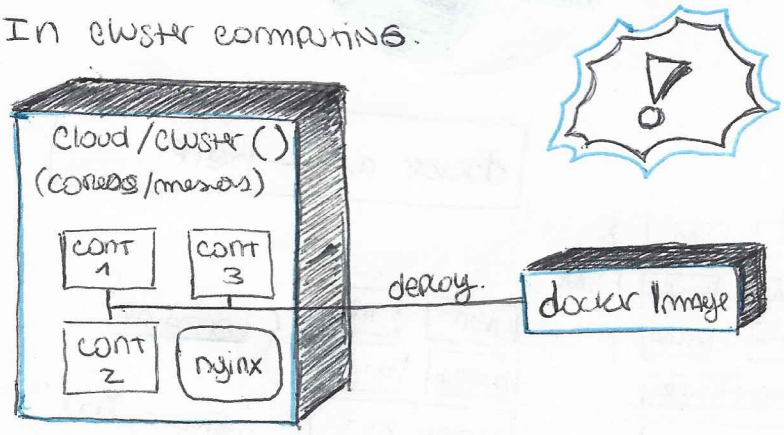
Docker Compose → for running multiple-container APP.

Daemon (computing). is a computer program that runs in background process, rather than being under direct control of an interactive user.

CoreOS/mesos are OS that runs on various computers and shared resources.

How does Docker gets awesom

- In cluster computing.



Docker Commands

(BASICS)

+ docker run <image>
  Creates a container for that image.
+ docker start/stop <name|id>
  restarts or stops a container
+ docker ps -a
  Lists all runing and stopped (-a) containers
+ docker rm <name|id>
+ docker images

Volumens

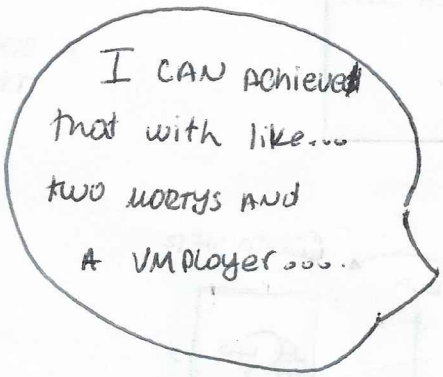
Volumes allows docker to ensure persistent parts (directories) of the container even if his last is ~~sto~~ stopped

Differenttypes

Hosts Volumes

lives on the Docker host's ~~volume~~ filesystem and it can be accessed from within the container

```
docker run -v /path/on/host:/path/in/container
```



Named Volumes

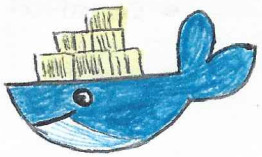
Idem Anonimous but with name

Docker volume create name

Anonymous Volumes

use ful when you would rather have Docker handle where the files are stored (it can be difficult to refer to the same volume over the time)

Docker Mastery Course (Udemy)



A BIT OF HISTORY OF MIGRATING TECHNOLOGY

- MAINFRAME → PC (90's)
- BARE METAL → VIRTUAL (00's)
- DATACENTER → CLOUD (10's)
- HOST → CONTAINER (NOW!)

- "DOCKER MIGRATION is A When not An if"

"Docker works with sysAdmin, devOps and siteOps"

#

i CONTAINERS ARE CONSISTENT ACROSS THE PLATFORM!

#

iKeep less time MAINTAINING AND MORE TIME INNOVATING!

landscape.cncf.io → Cloud Native Computing Foundation

DOCKER INSTALATION

DOCKER CE → FREE OPEN SOURCE (Community Edition)
DOCKER EE → ENTERPRISE EDITION

VERSION MATTERS AND ALSO HOW YOU INSTALL IT

3 MAJOR WAYS TO INSTALL DOCKER

- DIRECT: LINUX (≠ per distro), runs on the kernel
- MAC/WIN: SET OF TOOL THAT LETS YOU RUN DOCKER:

  MAC OS and WINDOWS 10 → NATIVELY DOES NOT SUPPORT DOCKER

  SO A "VM" MUST be running TO ALLOW DOCKER TO RUN ON THESE OS's
- THE CLOUD: DOCKER FOR google cloud, AWS, Azure, ...

CE vs EE

- Free
- Paid → Certified, Support

DOCKER CE

- STABLE — BETTER TESTED
- EDGE — NEW RELEASES

INSTALLATION

ON LINUX: # DO NOT USE THE built-in default PACKAGES

DOCKER AUTOMATE SCRIPT TO ADD DOCKER REPOS.

```
curl -sSL https://get.docker.com/ | sh
```

MAGIC!

Docker on Linux.

store.docker.com → Details on HOW-TO in every DISTRO.

YOU MUST USE `sudo` FOR MOST OF DOCKER COMMANDS ON MOST DISTROS

CHECK DOCKER VERSION: sudo docker version

TWO TOOLS THAT DO NOT COME BUNDLE WITH LINUX

- Docker Machine
- Docker Compose

BOTH CAN BE INSTALLED, AND MUST BE INSTALLED ON A LINUX DISTRO.

THE HOW-TO ESTÁ EN docs.docker.com.

PERO PARA TENER LA VERSIÓN ÚLTIMA CONVIENE IR A LA PARTE DE 'Releases' DE GITHUB.

SECTION 3

DOCKER VERSION ⟶ CHECK VERSION AND ENSURE IT'S WORKING

DOCKER INFO ⟶ MORE INFO AND CONFIG INFO.

DOCKER ⟶ --help + COMMANDS

DOCKER COMMANDS FORMAT

New Management Command Format : DOCKER <command> <sub-command> + <opt>

DOCKER <CMD> <SUB-CMD> <OPT>

" CONTAINER RUN ← NEW

" RUN ← old

docker container ls

List CONTAINERS RUNNING.

→ **ls -a** → LIST All CONT.

Image and Containers

Images : is THE APP THAT WE WANT TO RUN

Container : is AN INSTANCE OF AN IMAGE RUNNING AS A PROCESS

YOU CAN HAVE MULTIPLE CONT. OF THE SAME IMAGE.

docker container run --publish 80:80 nginx

- STARTS THE CONT. WITH A SPECIAL CMD IN THE DOCKER IMAGE
- STARTS A NEW CONTAINER FROM THAT IMAGE
- THE DOCKER ENGINE LOOKS FOR AN IMAGE CALLED "nginx" IN DOCKER HUB, AND PULLS THE LATEST VERSION
- The "--Publish 80:80" PART OF THE COMMAND EXPOSE THE PORT 80 FROM THE LOCAL MACHINE AND SENDS ALL TRAFIC TO PORT 80 ON THE RUNNING CONTAINER

--detach ⟶ RUNS CONTAINER IN BACKGROUND AND RETURNS THE UNIQUE CONTAINER ID

CONTAINER NAMES

IF WE DO NOT SPECIFY A CONTAINER NAME IT will generate ONE FROM A LIST OF IT RELATED LIST OF NAMES

`--name name.` → SPECIFY CONTAINER NAMES

`docker container logs cont name` → shows container logs

`docker container top cont. name` → PROCESESS RUNNING INSIDE THE CONTAINER

Remove containers

CAN ALSO BE THE 3 FIRST DIGITS OF THE ID

`docker container rm c.name1 cnam2 ...` YOU CAN'T REMOVE RUNNING CONTAINERS

`-f` DO it's ANYWAY

~~What happens in 'Docker Container Run'~~

#DO NOT COMPARE CONTAINERS WITH VIRTUAL MACHINES, ARE MORE LIKE PROCESSES RUNNING ON YOUR HOST

CONTAINERS ARE LIKE RESTRICTED PROCESS RUNNING INSDE OUR HOST OS

`docker top cont.nr` → List PROCESSES RUNNING INSIDE A SPECIFIC. CONTAINER

#YOU CAN CHECK CONTAINER's PROCESSES FROM THE HOST. (PS AUX)

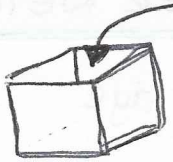
`--env` OR `-e` TO PASS IN VARIABLES, ENVIORMENT-VARIABLES

`docker container stop` → You know The drill

What's going on withe the containers?

`docker container inspect` → Details about the config

Get a shell inside the container.



`docker container run -it`

`-i` + `-t` = `-it`
- `-i` → -interactive
- `-t` → --tty

`docker container run [OPTIONS] IMAGE [commands] [ARGS]`

This way of running containers lets ↑ after IMAGE is specified.

`docker container run -it --name proxy IMAGE bash` ← EMPTY = DEFAULT command

↳ Prompts a bash shell inside the container

CONTAINERS RUNS AS LONG AS THEIR FIRST COMMAN RUNS

TO START A CONTAINER AND LAUNCH A SHELL YOU WOULD RATHER USE `-ai` than `-it`, AND START INSTEAD OF RUN BECAUSE YOU CONTAINER ALREADY EXISTS.

`docker ~~start~~ container START -ai CONT.NAME` → RUNS A CMD on A RUNNING container.

HOW TO JUMP INTO A RUNNING CONTAINER?

`docker CONTAINER exec -it CONT.NAME`

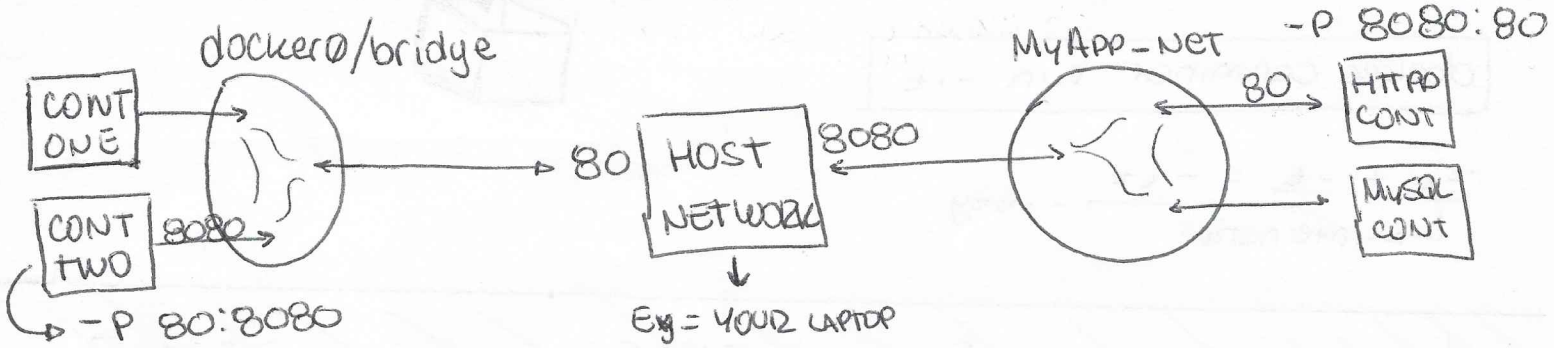
Batteries Included, but removable

Docker Network : concepts

`-p` | `--publish` HOST:CONTAINER (80:8888)

`docker container port CONT.NAME` → 80/tcp -> 0.0.0.0:80

EJEMPLO DOCKER NETWORKS



CONT ONE AND CONT TWO CON TALK FREELY OVER THE DEFAULT VPN - DOCKER (IT$ CREATE BY DEFAULT) BUT CAN'T TALK TO HTTPD CONT OR MySQL CONT UNLESS IT IS VIA THE HOST.

CLI Commands for Networking

`docker network ls` → LIST NETWORKS.

Default Networks

- bridge / docker0
- host
- none

docker0 or bridge } connects cont. to local network

`docker network inspect` + `network name` → shows details about the network

Also shows info and which containers are attached to that network

--network host

Connects container directly to host interface. Skipping the virtual net. interface of docker
(less secure but sometimes needed)

--network none

Removes eth0 and only leaves you with localhost interface in container

CREATE A DOCKER NETWORK

```
docker network create my_networ_name
```

Network DRIVERS (default: bridge)

- Built in or 3rd party extensions that gives you virtual networks features.
  → CREATES A SIMPLE VIRTUAL NETWORK locally with it's own subnet somewhere (172.17.0.0/...)

TO ATTACH A CONT. TO A specific networ:

```
docker container run ... --network net-name
```

Connect 3 containers

```
docker network connect net, contain.   → ATTACH
docker networ disconnect net, contain. → dis-ATTACH
```

NAming (!) (DNS NAMING)

Docker default sets the hostname to the containers name, but you can also set Aliases

containers should not rely on IP Addresses but on NAMES

It's OK to create custom networks for containers.

```
--rm
```

```
docker container run --rm ...
```
→ Removes/clean up/destroy container when it stops running.

"It is useful for when you need to test something inside a container

Docker Images

- It is an ordered collection of root file system changes and the corresponding execution parameters for use within a container runtime.
- It is NOT A COMPLETE OS. There is NO kernel, kernel modules (eg. drivers)

Images can be get from Docker Hub

- There is an oficial one that is ~~an~~ A public

OFICIAL ⟶ "IMAGENAME" ⟶ GREAT WAY TO START

NON-OFICIAL ⟶ "IMAGENAME/NAMEOFTHEREPO"

OFICIAL IMAGES HAS VERSIONS

TAGS to get images for our containers

IMAGE

USER/REPO : TAG

`docker pull IMAGENAME:TAG/VERSION` ⟶ PULL

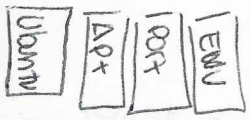
⟶ must be login sometimes

`docker image ls` ⟶ SHOWS IMAGES AND + INFO

Images History

`docker image history image name` ⟶ shows layer of changes made in an Image

Layers (of changes)

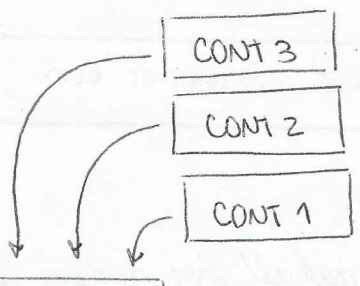


⟶ IMAGE

0 Change1 Change2

NEVER STORE THE SAME IMAGE DATA MORE THAN ONCE IN OUR FILE SYSTEM

⟶ IMAGE



CONTAINER IS ONLY THE RUNNING PROCESS

FOR INFO ABOUT THE IMAGE

```
docker image inspect imagename.
```

IMAGE REPOSITORIES THAT ARE OFFICIA GET TO OWN THE NAME OF THE IMAGE WITHOUT THE REPO NAME.

IMAGE TAGGING.

`<USER OR ORGANIZATION>/REPOSITORY : TAG` → IDENTIFY AN IMAGE

IT'S A POINTER TO A SPECIFIC IMAGE COMMIT

HOW TO RE-TAG AN IMAGE

SOURCE_IMAGE:TAG

```
docker image tag imgname newtag
```

→ TARGET_IMAGE:TAG

HOW TO PUSH

```
docker image push
```

(must be login to push)

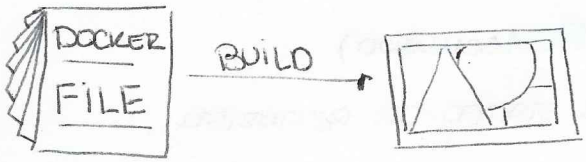
TO LOGIN

```
docke login
```

TO AUTHENTICATE DOCKE USE

```
CAT ~/.docker/config.json
```

Building Images



Dockerfile → IMAGE

Default name is Dockerfile with CAPS ."D". (LIKE MAKEFILE)

BUT TO BUILD A dockerfile with OTHER name

```
docker build -f other-dockerfile
```

THE DOCKERFILE → USES A LANGAGE THAT'S UNIQUE TO DOCKER

WORKS
TOP ↓ DOWN — ORDER MATTERS

THE FROM COMMAND → MUST BE IN EVERY DOCKER FILE

FROM debian:release → PACKAGE MANAGER IS THE MAIN REASON TO BUILD CONT. FROM UBUNTU, CENTOS, DEBIAN ...

THE RUN COMMAND

EXECUTES SHELL COMMANDS !

(UNZIPS, INSTALL PACKAGES, EDIT FILES)

BEST PRACTICE IS TO FEET EACH COMMAND INSIDE ONE LINE WITH "\" AND && BECAUSE EACH LINE IS ONE LAYER.

```
RUN COMMAND_1 \      → 1 LAYER
 && COMMAND_2 \      → 2nd LAYER
 && COMMAND_3 \      → 3rd LAYER
 && COMMAND_4        → 4th LAYER
```

LOGS

DOCKER HANDLE ALL OF THE LOGGING FOR US, SO, WE DO NOT HAVE TO LOG INTO ANOTHER LOG FILE

ALL WE HAVE TO DO IS REDIRECT TO STDOUT AND STDERR

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

→ HERE WE ARE REDIRECTING THE LOGS TO THE ORIGINAL NGINX LOGFILE

THE EXPOSE COMAND (OPTIONAL)

BY DEFAULT NO PORT BY DEFAULT IS EXPOSED, SO WE NEED TO EXPECIFY IT HERE

THE CMD COMMAND (IF NOT ... INHERIT FROM THE FROM COMMAND)

DEFAULT COMMAND THAT RUNS WHEN A CONTAINER IS STARTED OR RESTARTED.

Building Images

IMAGE NAME

CMD

IN THIS DIR ["exec", "Par1", ...]

```
docker images build -t custom_nginx .
```

→ EXECUTE IT BY STEPS, EACH STEP IS A LINE IN THE DOCKERFILE

DOCKER CASHED THE STEPS IN THE BUILD USING HASHKEYS SO WHEN YOU MAKE CHANGES AND RE-BUILD YOU WON'T HAVE TO RUN LINES OR STEPS THAT HAVEN'T BEEN CHANGED

THE ~~WORKDIR~~ WORKDIR COMMAND → RUNS A cd ~~dir~~

YOU CAN MAKE IT WITH A RUN cd DIR BUT THIS IS THE BES PRACTICE

THE COPY COMMAND

USED TO COPY FILES FROM LOCAL MACHINES/BUILD SERVER TO CONTAINER IMAGES.

Containers Lifetime

SECTION 4

CONTAINERS ARE IMMUTABLE AND EPHEMERAL

Immutable infraestructure": Just re-deploy, never change

PERSISTENT DATA → Volumes (1)
→ Bind Volumes (2)

(1) Volumes: MAKES SPECIAL LOCATION OUTSIDE OF CONTAINERS UFS

(2) Bind Volumes: Link CONTAINER PATH TO HOST PATH

Volumes and Persistent Data

THE VOLUME command (VOLUME PATH)

On the Dockerfile, TELL DOCKER TO CREATE A NEW VOLUME in THAT PATH. And ATTACHED it in THE CONTAINER

VOLUMES NEED MANUAL DELETION (JUST FOR ENSURANCE)

TO DO SO:

```
docker volume prune
```

RUNNING CONTAINERS STORES DATA ON A PATH

("/VAR/LIB/docker/volumes/ ~~~ /_data")

AND THEN ROUTED OR MAPPED TO A LOC. IN THE HOST

TO CHECK VOLUMES

```
docker volume ls
```

TO INSPECT

```
docker volume is
```

Named Volumes

TO SPECIFY A VOLUME ~~name~~. (CREATE)

```
docker container run ... -v name:/PATH
```

CREATE (NOT IN A RUN COMMAND) (ONLY WAY TO SPECIFY THE DRIVER)

```
docker volume create [OPT] [VOLUME]
```

MORE NAME-FRIENDLY AND MORE PATH-FRIENDLY

HOST ALWAYS WIN

Persistent Data: Bind Mounting

MAPS A HOST FILE OR DIRECTORY TO A CONTAINER ~~HOST~~ FILE OR DIRECTORY

ON BACKGROUND: IT IS JUST TWO LOCATIONS POINTING TO THE SAME FILE(S)

Bind mounts are hosts specific, so it need to be specified on the run cmd.

How-to

```
... RUN -V /PATH/in/HOST:/PATH/in/cont
```

FULL PATH — CONTAINER VOLUME

On windows

// DRIVE LETTER

SO, ARE THERE ALWAYS VOLUMES?

Docker Compose

2 PARTS

① YAML-Formatted file which describes our solution for

- Containers
- Network
- Volumes
- ENV VARS
- CONFIGS

YAML-code → Config-Language

② A CLI DOCKER-COMPOSE USED FOR LOCAL dev/test AUTOMATION with those YAML files

```
docker-compose --help
```

THE YAML FILE # EASY WAY TO DOCKER CONTAINER RUN

Version :

SERVICES # SAME AS DOCKER RUN.
- ~~IMAGE~~
- SERVICE NAME : Friendly name (ALSO DNS)
  - IMAGE :
  - COMMAND: REPLACE CMD
  - ENVIORMENT: -e → ENVIORMENT: ONEVAR : hola, TWOVAR : chau.
  - VOLUMES : -v
- SERVICE NAME 2 :

VOLUMES : ~~#~~ docker volume create

NETWORK : docker network create

DOCKER COMPOSE CLI (cool for local dev)

(download it separately)

`docker-compose up` → # Setups All volumes NETWORKS ETC.. And STARTS THEM (it has a -d)

`docker-compose down` → # STOPS all containers And remove cont/vol/net

`docker-compose log`

MORE COMMANDS: top ps ls ....

Using Compose to Build Images.

Build Images AT A RUN time.

`docker-compose build` → build Images ~~EVERY~~

build VARIABLES (buildARGuments)

VARS WE PASS DURING BUILDING IMAGES

SECTION 6.

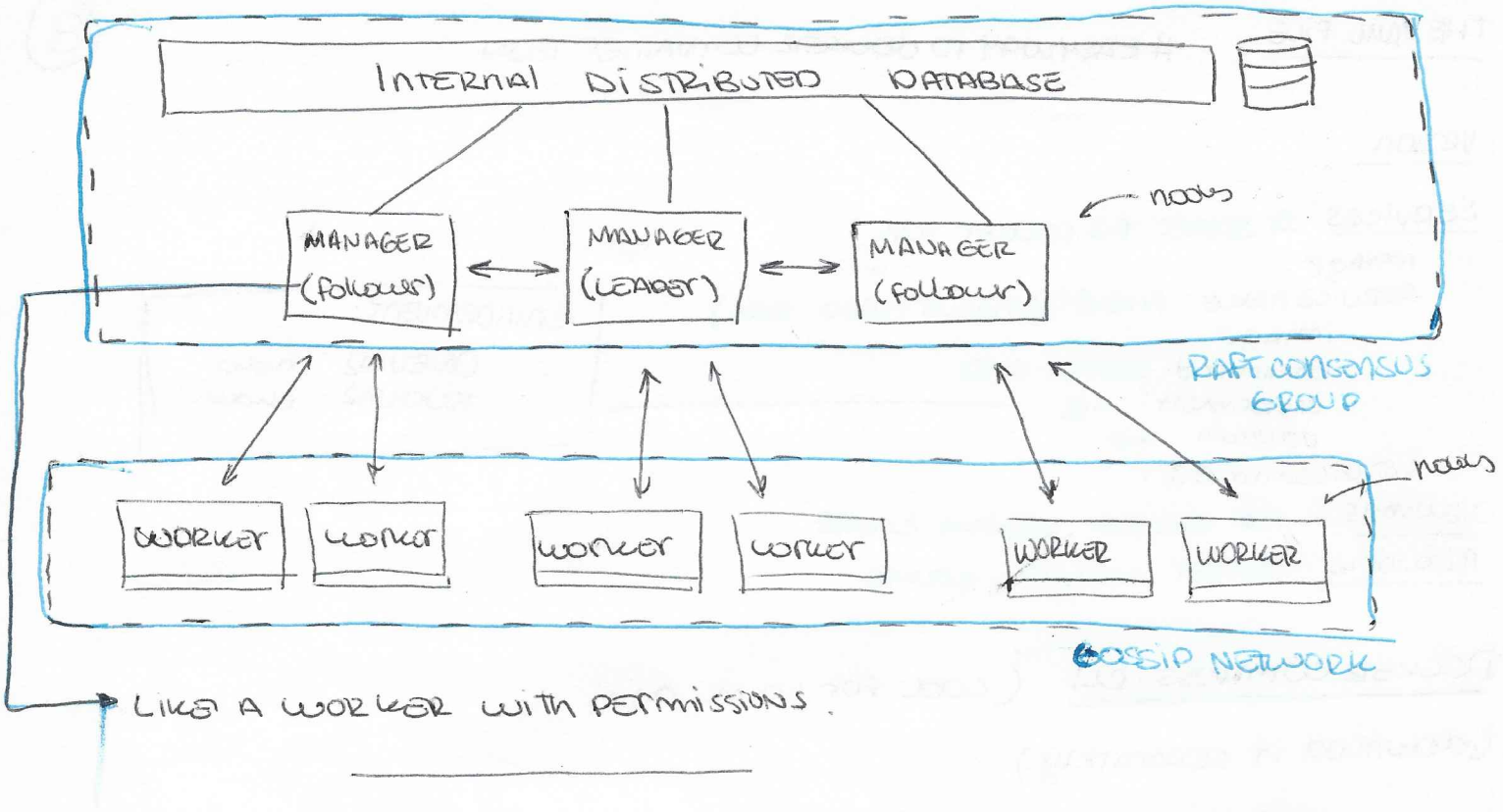
SWARM INTRO : Built-in orchestration

Swarm Mode : CLUSTERING SOLUTION BUILD INSIDE DOCKER

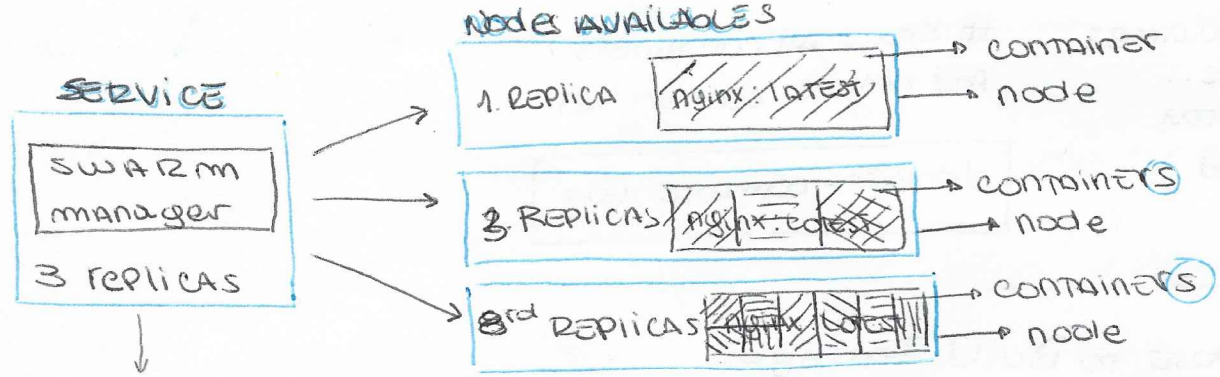
- WAY OF MANAGING MULTIPLE RUNNING CONTAINERS -

SWARM IS NOT ENABLED BY DEFAULT , ONCE ENABLED:

docker → SWARM



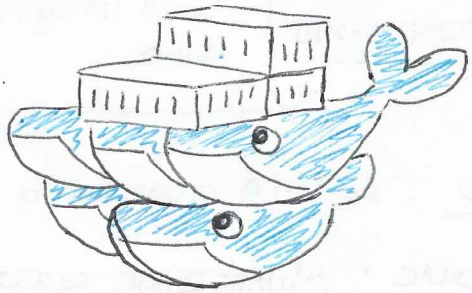
IN A SWARM, WE CALLED THE DOCKER SERVICE OUR NEW SCALABLE DOCKER RUN



DOCKER SERVICE CREATE

TO KNOW IF SWARM IS ACTIVE

DOCKER INFO



TO ENABLE SWARM

DOCKER SWARM INIT

DOCKER NODE INSPECT

DOCKER NODE
- LS
- PS

DOCKER NODE
- PROMOTE
- DEMOTE

The New Docker Run

`docker service --help`

CREATE
INSPECT
LOGS
LS → `docker service ls`

ID	MODE	NAME	REPLICAS	IMAGE
-	-	-	1/1	-

THE GOAL OF THE ORCHESTRATOR IS TO MAKE THIS NUMBERS MATCH

1/1 → HOW MANY ARE ACTUALLY RUNNING / HOW MANY YOU SPECIFIED TO RUN

PS → `docker service ps SERNAME`

SHOWS THE TASKS OR CONTAINER FOR A SERVICE

SIMILAR TO `docker container ls` but WITH NODES

`docker service update <ID> --replicas n`

SCALES A SERVICE SO IT ADDS MORE REP. (NOW THERE AR 3 COM RUNNING)

`docker update` → limiting and controlling resources without stoping a running container

`docker service update` → whole more options, START AND STOP CONTAINERS

↳ The whole point of swarm is to UPDATE AND MAKE CHANGES without having to take the entire thing down.

SWARM ENSURES CONSISTENT AVAILABILITY #

MANGING A 3 NODE CLUSTER

`docker swarm init --advertise-addr`

RUN ON NODE 1

HOST WHER YOU HAVE YOUR NODES.

SPITS THE

THE JOIN COMMAND HAS A JOIN-TOKEN → TO GET IT → `docker swarm join-token manager`

↓ SPITS COMMANDS TO ADD MANAGER NODES.

NODES → HAS CONTAINERS

`docker node ls` →

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STAT
*				

`docker node PS NODEN`

↳ do A CONTAINER IS on node

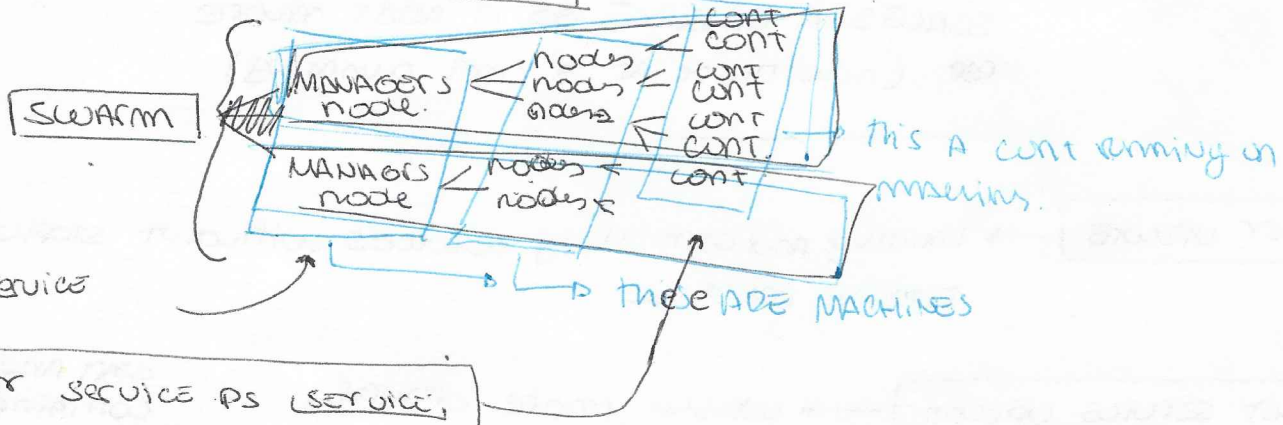
* → CURRENTLY TALKING TO THIS NODE

NOTE THAT A `docker node create` doesn't ☺

YOU NEED A MACHINE, EACH node is A MACHINE

`docker-machine create node_1` → CREATES A MACHINE.

`docker-machine SSH node_1` → CONNECTS TO THAT MACHINE

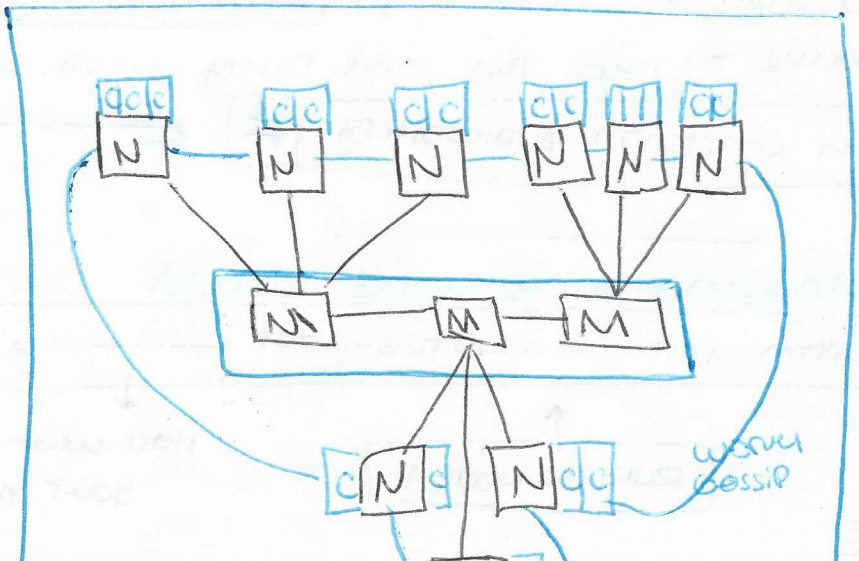


`docker service PS SERVICE`

↓

THE service is the nginx, httpd, drupal etc.

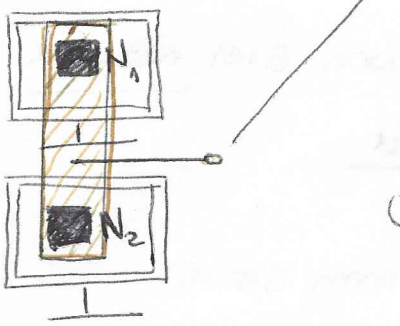
no matter how many



SWARM new NETWORK DRIVER OVERLAY

CREATE A NETWORK AS USUAL BUT WITH

- `--driver overlay` AND IT CREATES A SWARM-wide bridge network.
- CONTAINERS ACROSS A NETWORK CAN ACCESS ACROSS THEM through what it would be like A VLAN
- ONLY FOR INTRA SWARM COMUNICATION
- OPTIONAL IPSec.
- EACH SERVICE CAN BE CONNECTED TO ≠ NETWOR.



Routing Φ MESH

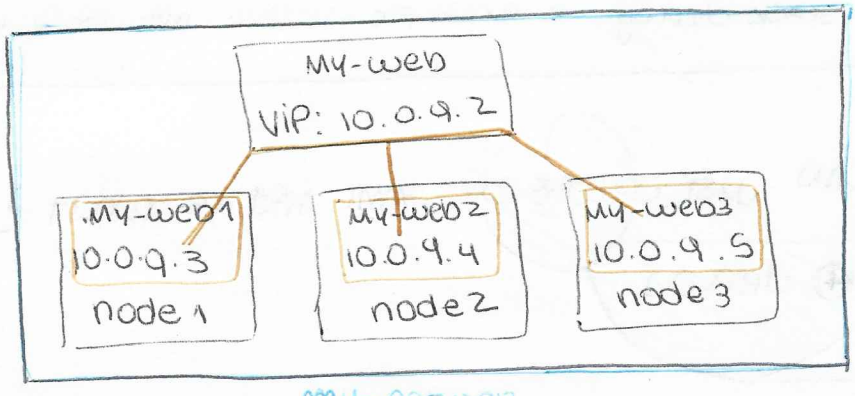
ROUTES INCOMING PACKETS FOR A SERVICE TO A PROPER TASK (USES IPVS)

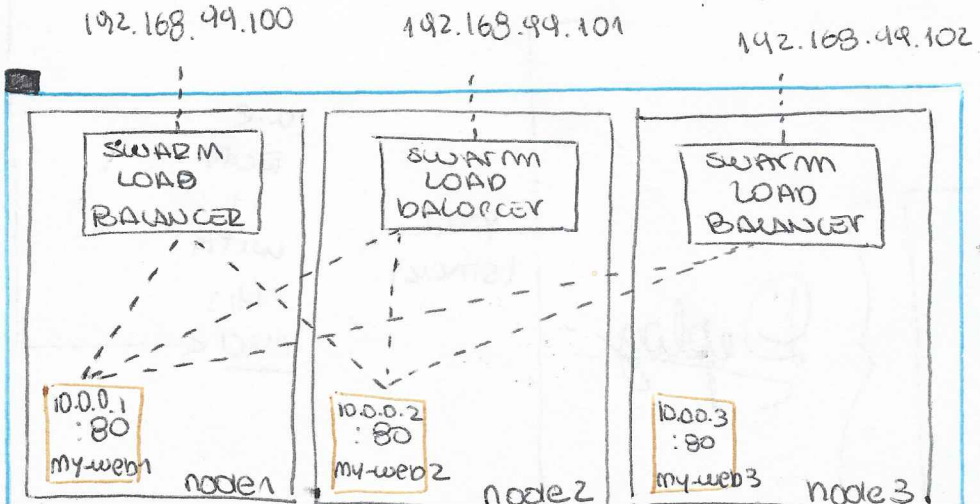
- SPAN ACROSS ALL NODES IN SWARM
- LOAD BALANCES SERVICES ACROSS ~~ALL NODES~~ THEIR TASKS.

TWO WAYS → CONT-TO-CONT
overlay network (VIP)

↓

EXTERNAL TRAFFIC INCOMING (ALL NODES LISTEN)





} LOAD BALANCER CAN DO THIS BECAUSE THEY ARE ON A SAME NETWORK

STACKS : Prod grade compose

Compose files for prod. swarm

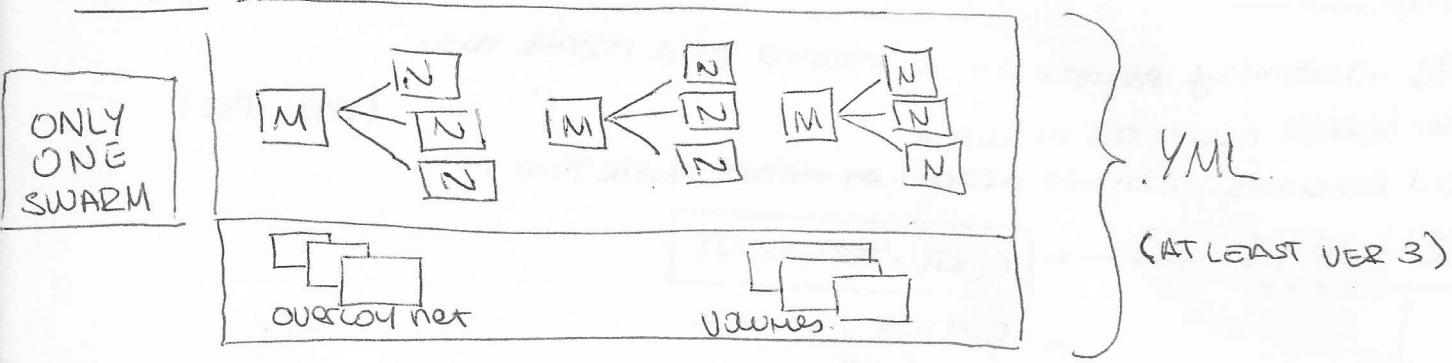
`docker STACK deploy`

STACKS accepts compose files. And do the hard work. Even network.

New deploy : key in Compose file. Can't do build

YOU DON'T NEED docker-compose CLI on Swarm server.

STACK



ON .YML FILE.

`docker STACK deploy -c example-voting-app-stack.yml APPname`

YOU CAN just update de .yml and re run it !

AT least -v 3.

.YML ④ deploy

```
version: '3'
services:
  web:
    image:
    deploy:
      replicas:
      restart_policy:
        condition:
      resources:
        limits:
          cpu:
          memory:
```

Deploy

.YML (STACK)

We RUN it with THIS cmd

SECRETS IN SWARM

EASIEST AND SECURE SOLUTION FOR STORING SECRETS.

SECRET: (SSH KEYS, TLS CERT., SENSIBLE DATA)

TO CREATE A SECRET
- FILE – `docker secret CREATE name hi.txt`
- CMD – `echo "~~" | docker...`

`docker secret` `ls` / `inspect` (WONT GIVE YOU THE SECRET)

DOCKER SWARM SECRETS IT'S AN SWARM ONLY THING BUT DOCKER KNOWS HOW TO MANAGE LOCALLY.

TO MAP A SECRET TO A SERVICE `--secret sec-name`
(SOMETIME WE USE `-e` TO DO SO TOO)
THERE ARE ALSO ENVIORMET_VAR_FILE SO WE CAN USE A FILE TO A ENV.

`docker service update --secret-rm` → REMOVES THE SECRET BUT RE-DEPLOYS THE SERVICE.

IN ORDER TO USE SECRETS. (VERSION: '3.1')

SECRETS IN SWARM STACKS

SECRETS ARE RM WHEN STACK IS RM

```
version: '3.1'
services:
  image: ——
  secrets:
    - PSQL-USER
    - PSQL-PASSWORD
  enviorment:
    POSTGRES_PASSWORD_FILE: /run/secrets/PSQL-PASSWORD
    OTHE_ENV-VAR: /run/sec..

secrets:
  PSQL_USER:
    file: ./PSQL_USER.TXT
  PSQL_PASSWORD:
```

(2) # ASIGN SECRETS. (FIRST (1))

(1) # DEFINE ALL SECRETS AT THE BOTTOM

Full App Lifecycle → docker-compose for more than one file

REV

REVELATION: †

THE DOCKER SERVICE IS TO DOCKER STACK
AS DOCKER RUN IS TO DOCKER COMPOSE

A DOCKER SERVICE is ONE OR MORE INSTANCES OF A SINGLE IMAGE deployed on one or more ~~of~~ machines.

A DOCKER STACK is A GROUP OF HETEROGENEOUS SERVICES described in the docker-compose.yml file.

docker swarm is a Clustering and orchestration tool.

UPDATESWARM

REV

docker stack deploy is THE WAY of updating a bunch of services.

WHOLE STACK

→ docker stack deploy -c file.yml stackname

docker ~~stack dep~~ service update --image myapp:newver servicename

JUST AN IMAGE OF A SERV.

docker service update --env-add NAME=name --publish-rm 8080

Update PORT (removes) And Add enviorment variables

docker service scale serv_1=8 serv_2=6

Updates the Amount of replicas of more than one service

Docker Healthcheck

STACK

Supported in docker file, swarm, compose and run.

- it exec the command in the container
- this can return only a binary result being exit 0 All OK! And exit 1 An Error

- Three container state: STARTING, HEALTHY, UNHEALTHY

```
docker run \
  --health-interval=5s \
  --health-cmd="____" \
  --health-retries=3 \
  ...
```

eg. on docker run.

Elastic search : 2

DOES NOT REPLACE MONITORING TOOL

Image Registries

Default Reg ⟶ Docker Hub

Best way of uploading images ⟶ Autobuild (with GitHub)

Build on every commit

Docker Cloud

- Uses Docker cloud hosters and
- bring-you-own-server
- Automate image build and deploy

Docker Registry ⟶ ~~and~~ PRIVATE GIT SERVER

SELF-HOSTED. ⟶ GitLab Container Registry

Docker Hub is for GitHub what Docker Store is for AppStore

3rd Party

Quay.io ⟶ most popular cloud based choice.

AWS, Google C., Azure

HAVE THEIR OWN Reg. OPTION.