HowTo ASIX Docker
Cloud - Machine - Swarm

Curs 2019 - 2020

Documentation:

- HowTo-ASIX-Clud-Machine-Swarm
- HowTo-ASIX-Docker
- HowTo-ASIX-Amazon-AWS-EC2-AMI-Cloud
- __docker_xuleta_ordres

Web:

- asix-m06
- asix-m11

Git

- edtasixm06
- edtasixm11

Docker Overview **6**
Docker Platform 6
Docker Engine 6
What can I use Docker for? 7
Docker Architecture 7
The Docker daemon 7
The Docker client 7
Docker registries 8
Docker objects 8
Images 8
Containers 8
Services 8
The underlying technology 9
Namespaces 9
Control groups 9
Union file systems 9
Container format 9

Conceptes Clau i Documentació **10**
Conceptes Clau 10
Docker Documentation 15
Docker Hub 17

Getting Started (original) **19**
Get Docker CE for Fedora 19
Get Started 20
Part 1: Setup 21
Part 2: Containers 21
Part 3: Services 23
Part 4: Swarm / machine 26
Install docker-machine 26
Creació de maquines virtuals VM 26
Afegir workers / Managers 27
Realitzar comandes a les VM 28
Deploy de la app 29
Part 5: Stacks 30
Portainer 31
Redis 32
Imatges / Captures 34

Getting Started (actual) **37**
QuickStart 37
Part 1: Orientation and Setup 37

Part 2: Containerizing an application 37
Part 3: Deploying to Kubernetes 38
Part 4: Deploying to Swarm 38

Getting Started with Docker Compose 41
Part 1-4: Setup - Create - Define - Build 41
Part 5-6: Edit - Rebuild - Update 43

Getting Started with Swarm 45
Part 1: Setup 45
Part 2: Create a Swarm 45
Part 3: Add nodes to a Swarm 46
Part 4: Deploy a service 46
Part 5: Inspect the service 46
Part 6: Change the scale 47
Part 7: Delete the service 47
Part 8: Rolling updates 47
Part 9 Drain / pause a node 48
Part 10: Use swarm mode routing mesh 50
Publish a port for tcp only or udp only 51
Bypass the routing mesh 52

Docker: Descripció General 53
Swarm 53
Swarm 53
Node 55
Routing Mesh 57
Docker-Machine 58
AWS EC2 58
Desplegament 58
Stack 58
Service 59
Task 61
Docker Compose 62
Service 63

Docker Machine 64
Overview 64
Install Docker Machine 65

Docker Network 66

AWS Amazon Web Services 68
Documentació a mirar: 68
Creació / Configuració del compte 68

Crear el compte 68
IAM Roles 68
IC2 Management 69
Key Pairs 70
AWS Command Line Interface 71
Deploy Docker Containers (ECS) 71
Fedora 71
Imatges AMI per a Amazon EC2 72
Imatges d'Atomic Host per a Amazon Public Cloud EC2 72
Format GP2 72
Format estàndard 72
Altres imatges 72
Crear una nova instància AMI de Fedora Cloud 73
Generar l'exemple de docker swarm 2 AMI en EC2 74
Una AMI amb 3web 1portainer 1visualizer 74
En un segon host 75

Docker Documentation: Getting Started 77
Swarm 77
Docker Machine 79
Install Docker Machine 79
Creació de maquines virtuals VM 80
Afegir workers / Managers 81
Realitzar comandes a les VM 82
Exemples docker machines on the cloud 83
Service 83
Cloud 83

Poti-poti 86
Portainer 86
Docker Daemon API 87
Docker offline documentation 87
Recursos 87

Pràctiques 87
Pràctiques en documents per als alumnes: 88
Pràctica: Desplegament Swarm local + AWS del comptador de visites de Getting Started Original 88
PAS 1: Disposar de Docker Engine en tots els hosts. 89
PAS 2: en els AMI AWS EC2 Obrir els ports 89
PAS 3: Crear el swarm i afegir-hi els nodes locals 89
PAS 4: Afegir els nodes remots de AWS EC2 89
Truc: Obrir un tunnel reverse 90
Pas 5: Engegar el desplegament 91

Docker Overview

Podeu consultar la documentació de Docker Overview a Docker Documentation.

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.

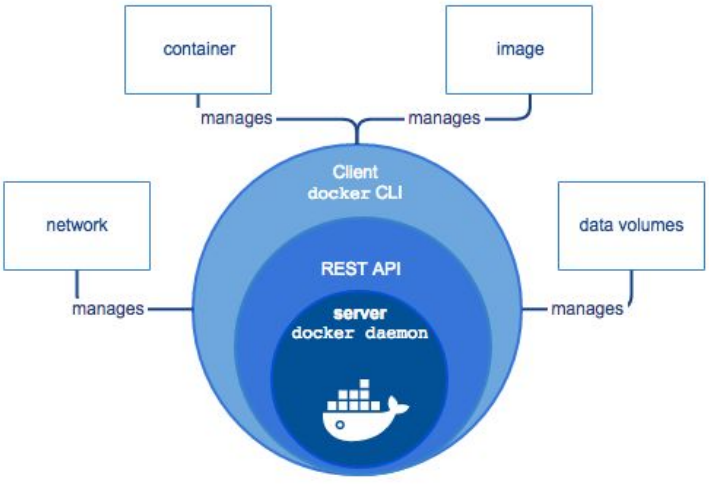
Docker Platform

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel.

Docker Engine

Docker Engine is a client-server application with these major components:

- A server which is a type of long-running program called a daemon process (the dockerd command).
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the docker command).



The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI.

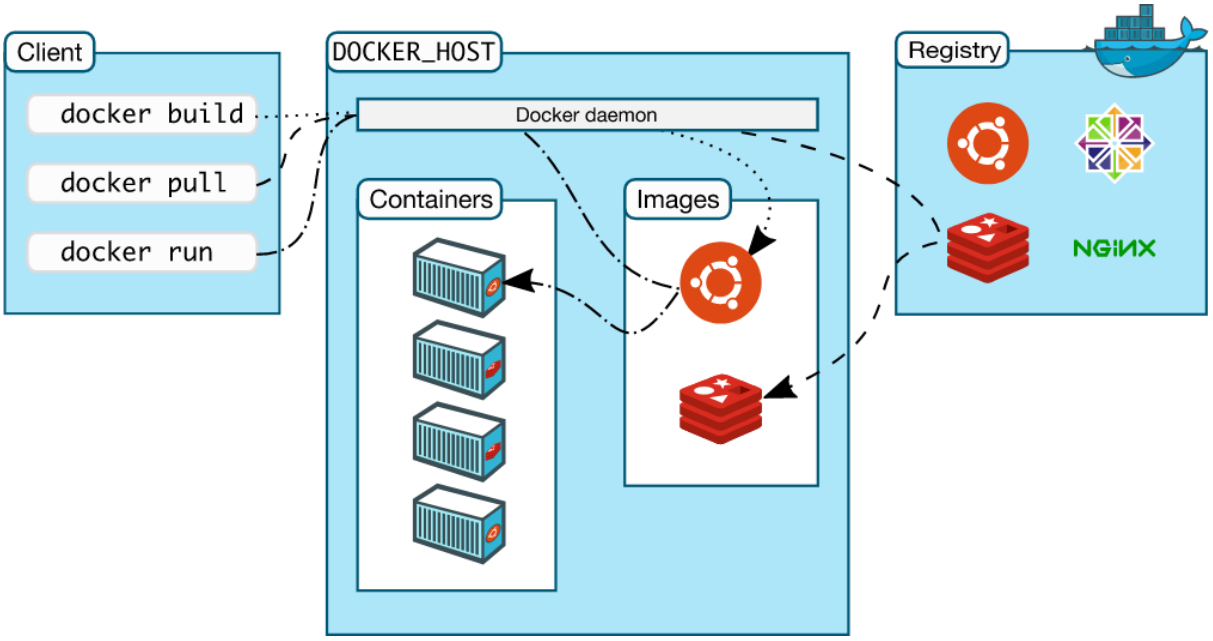
The daemon creates and manages Docker *objects*, such as images, containers, networks, and volumes.

What can I use Docker for?

Fast, consistent delivery of your applications
Responsive deployment and scaling
Running more workloads on the same hardware

Docker Architecture

Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.



The Docker daemon

The Docker daemon (***dockerd***) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client

The Docker client (***docker***) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd,

which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker registries

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects.

Images

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Containers

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container’s network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

Services

Services allow you to scale containers across multiple Docker daemons, which all work together as a swarm with multiple managers and workers. Each member of a swarm is a Docker daemon, and the daemons all communicate using the Docker API. A service allows you to define the desired state, such as the number of replicas of the service that must be available at any given time. By default, the service is load-balanced across all worker nodes.

The underlying technology

Docker is written in Go and takes advantage of several features of the Linux kernel to deliver its functionality.

Namespaces

Docker uses a technology called namespaces to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container. These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.
Docker Engine uses namespaces such as the following on Linux:

- The pid namespace: Process isolation (PID: Process ID).
- The net namespace: Managing network interfaces (NET: Networking).
- The ipc namespace: Managing access to IPC resources (IPC: InterProcess Communication).
- The mnt namespace: Managing filesystem mount points (MNT: Mount).
- The uts namespace: Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Control groups

Docker Engine on Linux also relies on another technology called control groups (cgroups). A cgroup limits an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints. For example, you can limit the memory available to a specific container.

Union file systems

Union file systems, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including AUFS, btrfs, vfs, and DeviceMapper.

Container format

Docker Engine combines the namespaces, control groups, and UnionFS into a wrapper called a container format. The default container format is libcontainer. In the future, Docker may support other container formats by integrating with technologies such as BSD Jails or Solaris Zones.

Conceptes Clau i Documentació

Conceptes Clau

Conceptes clau

- ❏ Container
- ❏ Task
- ❏ Servei
- ❏ Stack
- ❏ Node
- ❏ Swarm
- ❏ Machine

- ❏ Docker-compose.yml
- ❏ Orchestration

Descripció:

Un *container* és una instància d'una imatge en execució. Podem engegar containers al nostre criteri amb les ordres bàsiques de docker. A part de fer-ho manualment a la línia d'ordres, al container li podem definir recursos com volums/mounts, xarxes, recursos a usar de cpu i mem, etc. Quan ebomcallem un container amb totes aquestes característiques estem definint un *servei*.

Així doncs, un servei és un o n rèpliques d'un container amb les característiques que l'acompanyen. Podem definir serveis en la línia de comandes amb l'ordre *docker service* o podem generar un fitxer descriptiu del servei, un docker-compose.yml. Amb *docker-compose* podem defnir en un fitxer yml un conjunt de serveis que volem que s'executin junts, a mode d'aplicació (versió 2 dels fitxers docker-compose.yml).

Conceptualment un conjunt de serveis agrupats a mode de app formen el que docker anomena un *stack*. Un stack utilitza també els fitxers de definició docker-compose.yml (versió 3) però està pensat per ser desplegat en clusters i no en un únic host. Un conjunt de hosts que formen un cluster en terminologia docker s'anomena *swarm*.

Un swarm el formen *nodes* que poden realitzar el rol de worker o manager (leader). Els nodes poden ser hosts físics, virtuals amb màquines virtuals tipus Virtualbox o amb màquines virtuals docker-machine i també poden ser hosts del *Cloud* com per exemple màquines de AWS EC2.

Container

A container is a runtime instance of a docker image.

A Docker container consists of:

- A Docker image
- An execution environment
- A standard set of instructions

```
docker run --rm --name ldap -h ldap -d edtasixm06/ldapserver:18group
docker ps
docker stop ldap
```

Servei

A **service** is the definition of the tasks to execute on the manager or worker nodes. It is the central structure of the swarm system and the primary root of user interaction with the swarm.

A service is the definition of how you want to run your application containers in a swarm. At the most basic level a service defines which container image to run in the swarm and which commands to run in the container. For orchestration purposes, the service defines the “desired state”, meaning how many containers to run as tasks and constraints for deploying the containers.

Un servei és la descripció d’un container associant-li volumns,. Xarxes, número de rèpliques, caraterístiques de rendiment (cpu i memòria).

```
docker service create --replicas 3 --name ldap --publish 389:389 --detach=true \
        edtasixm06/ldapserver:18group
docker service ls
  ID                  NAME    MODE         REPLICAS    IMAGE            PORTS
  nkztjkp9jzv5        ldap    replicated   3/3      edtasixm06/ldapserver:18group   *:389->389/tcp
docker service rm ldap
```

Task

A **task** carries a Docker container and the commands to run inside the container. It is the atomic scheduling unit of swarm.

A task is the atomic unit of scheduling within a swarm. A task carries a Docker container and the commands to run inside the container. Manager nodes assign tasks to worker nodes according to the number of replicas set in the service scale.

Un container que s’executa com a servei, amb múltiples rèpliques, cada una d’aquestes rèpliques és un task, una unitat elemental de treball.

```
docker ps
CONTAINER ID    IMAGE                          COMMAND                  CREATED          STATUS
          PORTS               NAMES
e82642e24af2    edtasixm06/ldapserver:18group  "/opt/docker/start..."   5 minutes ago    Up 5 minutes
389/tcp             ldap.2.p8gpoxwyef4topedfcth3bdce
336f2a3382c6    edtasixm06/ldapserver:18group  "/opt/docker/start..."   5 minutes ago    Up 5 minutes
389/tcp             ldap.1.tmm5r0zsuozigvfdpoe1x7gz8
66dfe64904aa    edtasixm06/ldapserver:18group  "/opt/docker/start..."   5 minutes ago    Up 5 minutes
389/tcp             ldap.3.u4eadrs3yu4won296bucrl8w7
```

Docker-compose

Compose is a tool for defining and running complex applications with Docker. With Compose, you define a multi-container application in a single file, then spin your application up in a single command which does everything that needs to be done to get it running.

Els serveis es poden agrupar en un fitxer en format yml anomenat docker-compose. Usant l'ordre docker-compose es poden engegar conjunts de serveis, tots de cop, formant una app.

En aquest fitxer de definicions es poden definir els serveis, recursos, volums/mounts, xarxes, etc que cal engegar en conjunt.

```
version: "2"
services:
  ldap:
    image: edtasixm06/ldapserver:18group
    ports:
      - "389:389"
    volumes:
      - "ldapdata:/var/lib/ldap"
    networks:
      - ldapnet
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    networks:
      - ldapnet
networks:
  ldapnet:
volumes:
  ldapdata:
```

```
docker-compose up -d
    Creating network "tmp_ldapnet" with the default driver
    Creating volume "tmp_ldapdata" with default driver
    Creating tmp_visualizer_1 ... done
    Creating tmp_ldap_1  ... done
docker-compose down
```

Stack

When running Docker Engine in swarm mode, you can use docker stack deploy to deploy a complete application stack to the swarm. The deploy command accepts a stack description in the form of a Compose file.

Un conjunt de serveis per desplegar en un swarm formen un stack. Conté la definició dels serveis, rèpliques i recursos que cal engegar conjuntament formant una app.

Els stack es defineixen usant fitxers de definició docker-compose.yml versió 3.

```
version: "3"
services:
  ldap:
        image: edtasixm06/ldapserver:18group
        deploy:
        replicas: 3
        restart_policy:
        condition: on-failure
        resources:
        limits:
        cpus: "0.1"
        memory: 50M
        ports:
        - "389:389"
        volumes:
        - "ldapdata:/var/lib/ldap"
        networks:
        - ldapnet
  visualizer:
        image: dockersamples/visualizer:stable
        ports:
        - "8080:8080"
        volumes:
        - "/var/run/docker.sock:/var/run/docker.sock"
        deploy:
        placement:
        constraints: [node.role == manager]
        networks:
        - ldapnet
networks:
  ldapnet:
volumes:
  ldapdata:
```

Podem engegar i gestionar els stacks via l'ordre *docker stack*.

```
docker stack deploy -c docker-compose.yml myapp
Creating network myapp_ldapnet
Creating service myapp_ldap
Creating service myapp_visualizer

docker stack ls
NAME                SERVICES
myapp               2

docker stack ps myapp
ID                  NAME                IMAGE                          NODE                DESIRED STATE
CURRENT STATE           ERROR               PORTS
bjv26o4q2bve        myapp_visualizer.1  dockersamples/visualizer:stable  hostedt             Running
Running 30 seconds ago
i3m1lifpmzrn        myapp_ldap.1        edtasixm06/ldapserver:18group        hostedt             Running
        Running 35 seconds ago
0bqln4ld86b2        myapp_ldap.2        edtasixm06/ldapserver:18group        hostedt             Running
        Running 34 seconds ago
4m1jejgzp3pw        myapp_ldap.3        edtasixm06/ldapserver:18group        hostedt             Running
        Running 31 seconds ago

docker stack rm myapp
Removing service myapp_ldap
Removing service myapp_visualizer
Removing network myapp_ldapnet
```

Node

A **node** is an instance of the Docker engine participating in the swarm. You can also think of this as a Docker node. You can run one or more nodes on a single physical

computer or cloud server, but production swarm deployments typically include Docker nodes distributed across multiple physical and cloud machines.
A node is a physical or virtual machine running an instance of the Docker Engine in swarm mode.

Un node és un host (realment un docker-engine) membre d'un swarm (un cluster).

```
docker node ls
ID                            HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
1bcef6utixb0l0ca7gxuivsj0     worker2   Ready   Active
38ciaotwjuritcdtn9npbnkuz     worker1   Ready   Active
e216jshn25ckzbvmwlnh5jr3g *   manager1  Ready   Active        Leader
```

```
docker node ls
ID      HOSTNAME                                   STATUS   AVAILABILITY   MANAGER STATUS
        ENGINE VERSION
te3... *  ip-172-31-20-75.eu-west-2.compute.internal  Ready   Active        Leader          18.05.0-ce
0la5...   ip-172-31-30-30.eu-west-2.compute.internal  Ready   Active                        18.09.0
qml...    ip-172-31-31-57.eu-west-2.compute.internal  Ready   Active                        18.09.0
```

Swarm

A swarm consists of multiple Docker hosts which run in **swarm mode** and act as managers (to manage membership and delegation) and workers (which run swarm services). The cluster management and orchestration features. A swarm is a cluster of one or more Docker Engines running in swarm mode.

Un conjunt de nodes (hosts físics o virtuals) treballant conjuntament formant un cluster.

```
docker swarm init
docker swarm join --token SWMTKN-1-2am1ue865rkf3q440nf0f5cpbda4w934jhr1008\
        prstok39ime-cmvxb3xu88x9idscayk8e26un 172.31.31.57:2377

docker node ls
ID                          HOSTNAME                                   STATUS          AVAILABILITY     MANAGER
STATUS  ENGINE VERSION
zalibis343cx4q7r8f9uy00j3   ip-172-31-20-75.eu-west-2.compute.internal  Ready           Active
        18.05.0-ce
9g4x9e0wl5yu5lgrccqh8yden *  ip-172-31-31-57.eu-west-2.compute.internal  Ready                Active
        Leader              18.09.0

docker stack deploy -c docker-compose.yml myapp

docker swarm leave
docker swarm leave --force
```

Machine

Machine is a Docker tool which makes it really easy to create Docker hosts on your computer, on cloud providers and inside your own data center. It creates servers, installs Docker on them, then configures the Docker client to talk to them.

Amb la utilitat docker-machine podem generar màquines virtuals amb el docker-engine incorporat que poden treballar com a nodes.

És un mecanisme fàcil que porta docker per generar màquines virtuals molt simples que poden executar docker i ser nodes d'un swarm.

<generar la creació de dues machine i llistar-les><pendent>

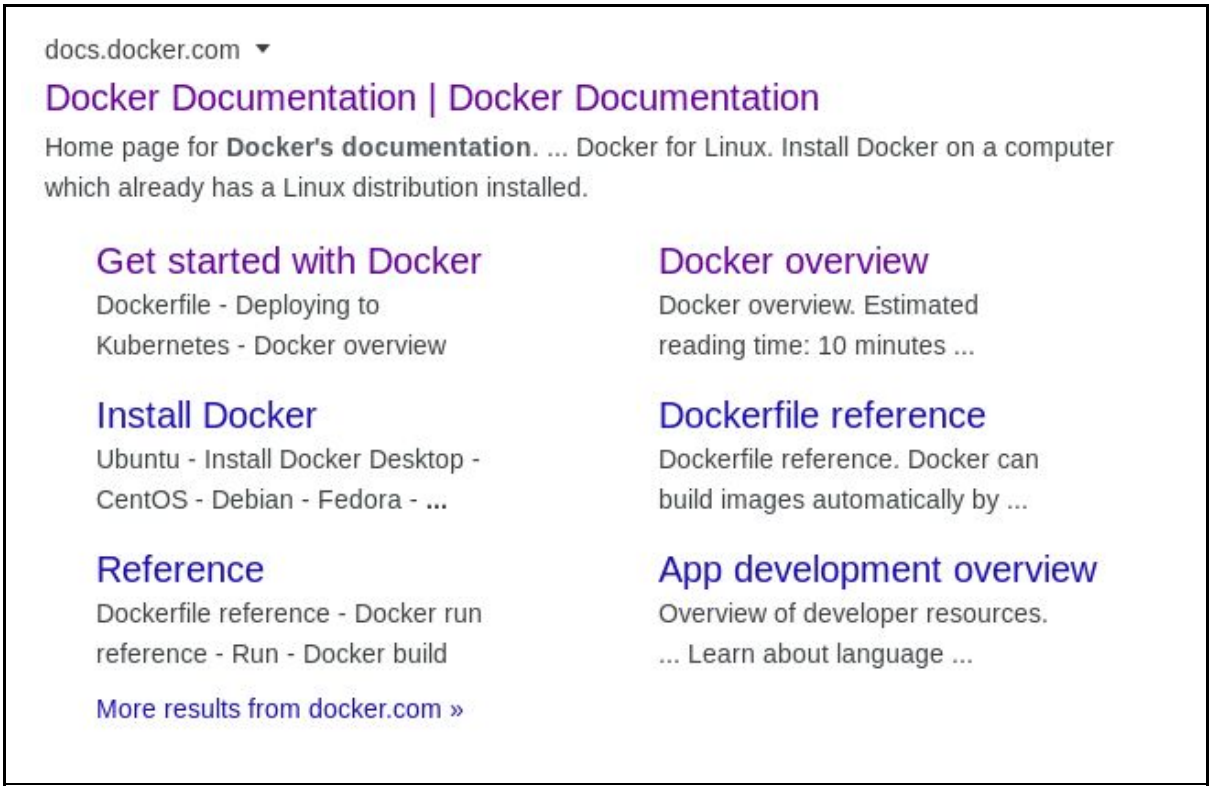
Orchestration

Orchestration és la capacitat de governar i posar en solfa tot aquest invent!

- Docker-compose
- Docker stack
- Docker swarm
- Kubernetes

Docker Documentation

Exemple de captura de google amb docker documentation:



Els principals elements a considerar de Docker Documentation són:

Get Docker

Com obtenir i instal·lar la última versió de Docker, el client i el servidor de Docker. Hi ha la versió Docker-CE (Comunity Edition) que és la versió lliure. L'altre producte de pagament és Docker-EE (Enterprise Edition).

Get Started

Docker proporciona varis tutorials de inicialització al seu 'entorn' anomenats "Get Started". En la vserió actual (Gener 2020) podem trobar-hi una descripció general *Docker Overview*, i un *Quickstart* format de 5 capítols o tasques.
La versió anterior de Get Started ara ha quedat fragmentada entre els diversos productes de docker. És aconsellable consultar:

- Get started with Docker Compose
- Get Started with Swarm mode

Search the docs

En la capsa de recerca podem posar qualsevol paraula o frase i es buscarà en tota la documentació de docker els apartats relacionats i els llistarà. Molt útil per buscar quina documentació hi ha referent a un tema (per exemple escrivint "volume") o per trobar algun apartat concret dins de la immensitat de la documentació.

Guides

Al menú lateral esquerra apareix tota la documentació de docker organitzada conceptualment, com a manuals d'explicació i aprenentatge. No ho sembla però és molt extensa ja que consta de molts temes en desplegables. A primer cop d'ull per exemple no hi veiem swarm o docker-compose, però hi són explicats dins dels temes en que s'enmarquen.

Els menús en temps de confeccionar aquesta documentació són:

- Get Docker
- Get Started
- Develop with Docker
- Configure Networking
- Manage Aplication Data
- Run your app in production ***la mare de tots els ous***
- Open Source at Docker
- Documentation Archive

Product Manuals

Manuals dels diferents productes de Docker. Són productes que es poden instal·lar per separat. Destaquen especialment:

- Docker Engine
- Docker compose
- Docker Hub

Reference

Conjunt de manuals de referència de les diverses eines de docker i dels formats de tipus de fitxers. Són els manuals on s'aborda ordre a ordre la seva descripció i les seves opcions (per a les ordres) i el format i directives dels fitxers de configuració.

A destacar en el cas de descripció del format dels fitxers:

- **Dockerfile**: descripció complerta dels fitxers Dockerfile de creció d'imatges automatitzades.
- **Compose**: descripció dels fitxers .yaml de configuració de desplegaments amb docker-compose. Existeixen diferents versions per a proposits (és a dir version 3 no substitueix version 2). A grans trets version 2 és per desplegaments en un host i version 3 per a desplagaments en un cluster via Docker Swarm.

Els manuals de referència del CLI (entorn de comandes) descriuen completament els entorns de:

- **Docker CLI**: el client docker (les ordres que fem en consola)
- **Compose CLI**: les ordres de consola de docker-compodse.
- **Daemon CLI**: el govern del servidor docker, el daemon dockerd.

També es descrien les **API** d'accés al Docker Engine i al Registry.

Samples

Proporciona l'accés a tutorials de funcionament de docker i de les seves eines i també accessos a la utilització d'altres aplicacions dockeritzades. Actualment la majoria de productes de software es lliuren també en format docker, aquí hi ha alguns exemples però s'aconsella accedir-hi via la propia documentació del producte al Docker Hub.

Jugeu per exemple amb Docker for Beginners o amb Docker Swarm mode.

Docker Hub

Docker Hub conjuntament amb GIT és la eina universal per a poder compartir imatges docker. Docker Hub és un repositori públic d'imatges on es poden carregar i descarregar imatges pròpies i d'altres.

Cal disposar d'un compte de Docker Hub (necessari per a fer aquests exercicis). L'usuari podrà pujar-hi imatges públiques (visibles i descarregables per tothom) i privades (només per al propi usuari).

La majoria de productes de software tenen els seus productes i explicacions de com utilitzar-los al Docker Hub, per exemple:

• Fedora • Ubuntu • Debian • Alpine	• httpd • Postgres • Mysql • Redis	• edtasim06 • edtasim11

• Busybox	• Python • Django • Java (JRE)	

Getting Started (original)

nota L'antic Get Started de docker (exemple amb comptador de visites) s'ha modificat per un de nou (un bulletin board). Per seguir l'antic exemple ara cal consultar: **Get Started with Docker Compose**.

La Docker documentation és molt àmplia i aconsellable de consultar i seguir. Hi podem trobar Get Started on a través d'un senzill tutorial de cinc passos podrem tenir una visió global dels elements que componen l'univers Docker. També podrem trobar una descripció de com instal·lar docker en diversos sistemes operatius a l'apartat.

Docker Documentation: Docker documentation
Get Docker: Get Docker
Get Started: Get Started
Get Started with Docker Compose: Get Stareted With Docker Compose
Get Sarted with Swarm mode: Get Started With Swarm mode
Play with Docker: Play with Docker (account needed)

Get Docker CE for Fedora

https://docs.docker.com/install/linux/docker-ce/fedora/

Es recomana consultar la documentació de docker Get Docker i seguir els passos indicats per instal·lar la versió més recent del Docker Engine per al sistema operatiu que s'estigui utilitzant. Es recomanable de fer-ho encara que a hores d'ara ja tinguem docker disponible al host, ja que sovint no és la versió més actualitzada del Docker Engine.

Aprendrem com:

- Desinstal·lar docker.
- Definir el repositori de docker.
- Instal·lar Docker Engine Comunity Edition.
- Verificar la instal·lació.
- Desinstal·lar docker-ce.

Desinstal·lar el docker actual:

```
$ sudo dnf remove docker \
                  docker-client \
                  docker-client-latest \
                  docker-common \
                  docker-latest \
                  docker-latest-logrotate \
                  docker-logrotate \
                  docker-selinux \
```

```
            docker-engine-selinux \
            docker-engine
```

Crear a DNF un nou repositori docker:

```
$ sudo dnf -y install dnf-plugins-core

$ sudo dnf config-manager \
    --add-repo \
    https://download.docker.com/linux/fedora/docker-ce.repo
```

So volguéssim activar/desactivar a part del repositori estable els repositoris edge i test fariem:

```
# Activar:
$ sudo dnf config-manager --set-enabled docker-ce-edge
$ sudo dnf config-manager --set-enabled docker-ce-test

# Desactivar:
$ sudo dnf config-manager --set-disabled docker-ce-edge
$ sudo dnf config-manager --set-disabled docker-ce-test
```

Instal·lar Docker-CE

```
$ sudo dnf -y install docker-ce
```

Verificar la instal·lació

```
$ sudo systemctl start docker
$ sudo docker run hello-world
```

Desinstal·lar docker:

```
$ sudo dnf -y remove docker-ce
$ sudo rm -rf /var/lib/docker
```

Get Started

Documentació de Docker documentation: [Get started]

1. Set up
2. Build an image
3. Scale your app
4. Distribute your app
5. Stack services
6. Deploy your app

Repositori GIT:

https://github.com/edtasixm11/getstarted.git

git clone https://github.com/edtasixm11/getstarted.git

Repositori DockerHub

docker pull edtasixm11/getstarted

Part 1: Setup

```
# docker --version
Docker version 1.10.3, build e03ddb8/1.10.3

# docker run hello-world
# docker image ls
# docker container ls --all
```

```
## List Docker CLI commands
docker
docker container --help

## Display Docker version and info
docker --version
docker version
docker info

## Execute Docker image
docker run hello-world

## List Docker images
docker image ls

## List Docker containers (running, all, all in quiet mode)
docker container ls
docker container ls --all
docker container ls -aq
```

Part 2: Containers

Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app
```

```
# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

requirements.txt

```
Flask
Redis
```

app.py

```
from flask import Flask
from redis import Redis, RedisError
import os
import socket

# Connect to Redis
redis = Redis(host="redis", db=0, socket_connect_timeout=2, socket_timeout=2)

app = Flask(__name__)

@app.route("/")
def hello():
    try:
        visits = redis.incr("counter")
    except RedisError:
        visits = "<i>cannot connect to Redis, counter disabled</i>"

    html = "<h3>Hello {name}!</h3>" \
           "<b>Hostname:</b> {hostname}<br/>" \
           "<b>Visits:</b> {visits}"
    return html.format(name=os.getenv("NAME", "world"),
hostname=socket.gethostname(), visits=visits)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80)
```

Build the image and run:

```
docker build -t friendlyhello .

docker run -p 4000:80 friendlyhello

Firefox → host:4000
```

Run in detach mode:

```
docker run -d -p 4000:80 friendlyhello
```

Aturar el container:

```
# docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
          PORTS                    NAMES
98ce2f0bc4dc        friendlyhello    "python app.py"        About a minute ago   Up About
a minute   0.0.0.0:4000->80/tcp   eager_lumiere

# docker container stop 98ce2f0bc4dc
```

Desar la imatge al repositori de Docker Hub:

```
$ docker login
$ docker tag friendlyhello edtasixm11/getstarted:part2
$  docker push edtasixm11/getstarted:part2
```

Executar l'aplicació directament del repositori Docker Hub:

```
$ docker run -d -p 4000:80 edtasixm11/getstarted:part2
```

Part 3: Services

Un servei és la execució de una sola imatge, indicant el número de rèpliques, ports, etc. **Nota** En aquest capítol GetStarted treballa principalment amb stack per fer el deploy del servei. Però també es poden fer les ordres service manualment a la línia de comandes, passant totes les opcions que es detallen en el .yml..

docker-compose.part3.yml

```
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    #image: username/repo:tag
    image: edtasixm11/getstarted:part2
    deploy:
      replicas: 5
      resources:
```

```
      limits:
        cpus: "0.1"
        memory: 50M
    restart_policy:
      condition: on-failure
  ports:
    - "80:80"
  networks:
    - webnet
networks:
 webnet:
```

Defineix la creació de dos elements:

- ❏ Un servei anomenat **web** (seran 5 rèpliques de la imatge edtasixm11/getatarted:part2
- ❏ Una web pròpia anomenada **webnet**

Inicialitzar el swarm per treballar amb Swarm + Stack

```
$ docker swarm init
Swarm initialized: current node (mdjotq5m8u6m77wnyl38gntoc) is now a manager.

To add a worker to this swarm, run the following command:
        docker swarm join --token
SWMTKN-1-3m1pb6khbjxfubyxdoykfzsoveq1zyclv5alvdyy0yvixzqzan-0cf54ommnk4n7g6j
koof2xobx 192.168.1.34:2377
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
instructions.
```

Iniciar el servei amb docker stack, nom del stack “getstartedlab”

```
$ docker stack deploy -c docker-compose.part3.yml getstartedlab
Creating network getstartedlab_webnet
Creating service getstartedlab_web
```

Observar la xarxa:

```
# docker network ls | grep webnet
c1syw8p1gfk2        getstartedlab_webnet   overlay         swarm
```

Verificar el funcionament:

```
# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                          PORTS
dg35efndfd14        getstartedlab_web   replicated          5/5                 edtasixm11/getstarted:part2    *:80->80/tcp

# docker service ps getstartedlab_web
ID                  NAME                IMAGE                          NODE                DESIRED STATE       CURRENT
STATE               ERROR               PORTS
mho8xsgmxqgm        getstartedlab_web.1 edtasixm11/getstarted:part2    hostedt             Running             Running
about a minute ago
```

```
pq2tjdei4t7o        getstartedlab_web.2   edtasixm11/getstarted:part2   hostedt   Running   Running
about a minute ago
nn5rjuueuy5w        getstartedlab_web.3   edtasixm11/getstarted:part2   hostedt   Running   Running
about a minute ago
l7kb8d0zqd3f        getstartedlab_web.4   edtasixm11/getstarted:part2   hostedt   Running   Running
about a minute ago
ql1uempptes7        getstartedlab_web.5   edtasixm11/getstarted:part2   hostedt   Running   Running
about a minute ago

# docker container ls
CONTAINER ID        IMAGE                         COMMAND             CREATED             STATUS              PORTS
            NAMES
9ae4a01b4251        edtasixm11/getstarted:part2   "python app.py"     2 minutes ago       Up 2 minutes        80/tcp
            getstartedlab_web.3.nn5rjuueuy5wev12xnzpbt2u0
7e7c17f62d55        edtasixm11/getstarted:part2   "python app.py"     2 minutes ago       Up 2 minutes        80/tcp
            getstartedlab_web.5.ql1uempptes7w3anpe1o233zy
59b398d7a9f6        edtasixm11/getstarted:part2   "python app.py"     2 minutes ago       Up 2 minutes        80/tcp
            getstartedlab_web.2.pq2tjdei4t7onr7mkib2vbckm
fd63c5200a50        edtasixm11/getstarted:part2   "python app.py"     2 minutes ago       Up 2 minutes        80/tcp
            getstartedlab_web.4.l7kb8d0zqd3fhxy7jdx9f894z
f9290fa8b7af        edtasixm11/getstarted:part2   "python app.py"     2 minutes ago       Up 2 minutes        80/tcp
            getstartedlab_web.1.mho8xsgmxqgmmwpgb0xo5ua5h

# docker stack ls
NAME                SERVICES
getstartedlab       1

# docker stack services getstartedlab
ID                  NAME                MODE                REPLICAS            IMAGE                         PORTS
redgl8pf5ffq        getstartedlab_web   replicated          5/5                 edtasixm11/getstarted:part2   *:80->80/tcp
```

Scale the app: directament modificant el fitxer de .yml i sense haver de fer stop de res. Cal fer un deploy en lloc del create:

```
# docker stack deploy -c docker-compose.part3.yml getstartedlab
Updating service getstartedlab_web (id: redgl8pf5ffqkv5ur1fk5c1ft)

# docker stack ls
NAME                SERVICES
getstartedlab       1

# docker stack services getstartedlab
ID                  NAME                MODE                REPLICAS            IMAGE                         PORTS
redgl8pf5ffq        getstartedlab_web   replicated          2/2                 edtasixm11/getstarted:part2   *:80->80/tcp

# docker service ps getstartedlab_web
ID                  NAME                  IMAGE                         NODE      DESIRED STATE   CURRENT
STATE               ERROR               PORTS
iawbo28idf6g        getstartedlab_web.1   edtasixm11/getstarted:part2   hostedt   Running         Running 16
minutes ago
yk14ktjc3rgn        getstartedlab_web.5   edtasixm11/getstarted:part2   hostedt   Running         Running 16
minutes ago
```

Tancar serveis i swarm

```
# docker stack rm getstartedlab
Removing service getstartedlab_web
```

```
Removing network getstartedlab_webnet

# docker swarm leave --force
Node left the swarm.
```

Part 4: Swarm / machine

A swarm is a group of machines that are running Docker and joined into a cluster. The machines in a swarm can be physical or virtual. After joining a swarm, they are referred to as **nodes**.

Install docker-machine

Cal instal·lar a part docker-machine:

```
$ base=https://github.com/docker/machine/releases/download/v0.14.0 &&
  curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &&
  sudo install /tmp/docker-machine /usr/local/bin/docker-machine
```

```
 docker-machine create --driver virtualbox myvm1
Running pre-create checks...
Error with pre-create check: "We support Virtualbox starting with version 5. Your
VirtualBox install is \"WARNING: The vboxdrv kernel module is not loaded. Either there is
no module\\n              available for the current kernel (4.7.9-200.fc24.x86_64) or it failed
to\\n              load. Please try load the kernel module by executing as root\\n\\n
dnf install akmod-VirtualBox kernel-devel-4.7.9-200.fc24.x86_64\\n          akmods
--kernels 4.7.9-200.fc24.x86_64 && systemctl restart systemd-modules-load.service\\n\\n
          You will not be able to start VMs until this problem is fixed.\\n5.1.26r117224\".
Please upgrade at https://www.virtualbox.org"
```

Creació de maquines virtuals VM

Per treballar amb el swarm n'hi ha prou amb una màquina però té més gràcia usar-ne unes quantes. En crearem de virtuals amb docker-machine.

Crear dues màquines basades en Virtualbox driver:

```
docker-machine create --driver virtualbox myvm1
docker-machine create --driver virtualbox myvm2
docker-machine create --driver virtualbox myvm3
```

```
# docker-machine create --driver virtualbox myvm1
Running pre-create checks...
```

```
(myvm1) Default Boot2Docker ISO is out-of-date, downloading the latest release...
(myvm1) Latest release for github.com/boot2docker/boot2docker is v18.03.1-ce
(myvm1) Downloading /root/.docker/machine/cache/boot2docker.iso from
https://github.com/boot2docker/boot2docker/releases/download/v18.03.1-ce/boot2docker.i
so...
(myvm1) 0%....10%....20%....30%....40%....50%....60%....70%....80%....90%....100%
Creating machine...
(myvm1) Copying /root/.docker/machine/cache/boot2docker.iso to
/root/.docker/machine/machines/myvm1/boot2docker.iso...
(myvm1) Creating VirtualBox VM...
(myvm1) Creating SSH key...
(myvm1) Starting the VM...
(myvm1) Check network to re-create if needed...
(myvm1) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
...
```

```
# docker-machine ls
NAME          ACTIVE   DRIVER      STATE          URL                       SWARM
DOCKER        ERRORS
custombox              none        Timeout
myvm1         -        virtualbox  Running   tcp://192.168.99.100:2376
v18.03.1-ce
myvm2         -        virtualbox  Running   tcp://192.168.99.101:2376
v18.03.1-ce
```

Fer del node myvm1 el manager del swarm, tot creant el swarm i declarant-se ell com a manager en fer el advertisement usant la seva adreça ip:

```
 docker-machine ssh myvm1 "docker swarm init --advertise-addr 192.168.99.100"
Swarm initialized: current node (tekibedmzdq337ukejqugq02r) is now a manager.

To add a worker to this swarm, run the following command:
        docker swarm join --token
SWMTKN-1-4umppecjmw2ojz7y5ciu5iq1kghrqvafn5t7q6mk71xy0qvvcj-d85ztgrvkjdhuefix2
ar6axby 192.168.99.100:2377
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
instructions.
```

Afegir workers / Managers

En la resposta explica com afegir nodes com a worker i com a managers:

- ❏ Worker:
  docker swarm join --token
  SWMTKN-1-4umppecjmw2ojz7y5ciu5iq1kghrqvafn5t7q6mk71xy0qvvcj-d85ztgrvkjdh
  uefix2ar6axby 192.168.99.100:2377

- ❏ Manager:
  docker swarm join-token manager

De fet es pot demanar en qualsevol moment al node manager que ens recordi què cal fer per poder afegir nodes com a workers o com a managers. Cal adreçar la ordre al node que fa de manager:

```
$ docker-machine ssh myvm1 \
      “docker swarm join-token manager / worker”
```

Activar com a worker la VM2:

```
# docker-machine ssh myvm2 "docker swarm join --token
SWMTKN-1-4umppecjmw2ojz7y5ciu5iq1kghrqvafn5t7q6mk71xy0qvvcj-d85ztgrvkjdhuefix2
ar6axby 192.168.99.100:2377"
This node joined a swarm as a worker.
```

Llistar els nodes del swarm:

```
# docker-machine ssh myvm1 "docker node ls"
ID                            HOSTNAME        STATUS    AVAILABILITY     MANAGER STATUS   ENGINE
VERSION
tekibedmzdq337ukejqugq02r *   myvm1           Ready     Active           Leader           18.03.1-ce
tj4mtswigcpm1f8lij0epftk6     myvm2           Ready     Active                            18.03.1-ce

[root@hostedt ~]# docker-machine ssh myvm2 "docker node ls"
Error response from daemon: This node is not a swarm manager. Worker nodes can't be
used to view or modify cluster state. Please run this command on a manager node or
promote the current node to a manager.
exit status 1
```

Realitzar comandes a les VM

Hi ha dos mecanismes:

- ❏ Usant docker-machine ssh <nomvm> “comanda-a-fer”. En aquest mode el filesystem és el del node, s’hi poden fer les ordres que fariem com si estem connectats per ssh.
- ❏ Modificar l’entorn perquè les comandes docker que realitzem a la consola es realitzin en realitat a la VM. En aquest mode el filesystem és el del host, podem usar els fitxers compose .yml del host, i les ordres docker que fem s’executen en la VM.

Comandes via docker-machine ssh:

```
$ docker-machine ssh myvm1 “docker node ls”
ID                            HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS  ENGINE VERSION
tekibedmzdq337ukejqugq02r *   myvm1     Ready   Active        Leader          18.03.1-ce

# docker-machine ssh myvm1 "uname -a"
Linux myvm1 4.9.93-boot2docker #1 SMP Thu Apr 12 17:05:48 UTC 2018 x86_64 GNU/Linux
```

Configurar la consola per a fer comandes directes a la VM:

```
$ docker-machine env myvm1
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/root/.docker/machine/machines/myvm1"
export DOCKER_MACHINE_NAME="myvm1"
# Run this command to configure your shell:
# eval $(docker-machine env myvm1)

$ eval (docker-machine env myvm1)

# aquesta comanda la fa directament en el node myvm1
$ docker node ls

# desconnectar l’entorn per retornar al host l’entorn de comandes:
$ eval (docker-machine eval -u)
```

Deploy de la app

Fer el deploy de la app amb docker stack

```
# docker stack deploy -c docker-compose.part3.yml getstartedlab
Creating network getstartedlab_webnet
Creating service getstartedlab_web
```

Observar que les 5 rèpliques es reparteixen entre els dos nodes:

```
# docker stack ps getstartedlab
ID            NAME                  IMAGE                          NODE   DESIRED STATE  CURRENT STATE           ERROR  PORTS
h1asbsgy4jv1  getstartedlab_web.1   edtasixm11/getstarted:part2    myvm2  Running        Running 6 minutes ago
07agar380cty  getstartedlab_web.2   edtasixm11/getstarted:part2    myvm1  Running        Running 6 minutes ago
o6jcscp169sb  getstartedlab_web.3   edtasixm11/getstarted:part2    myvm2  Running        Running 6 minutes ago
k9gurp0f5w6f  getstartedlab_web.4   edtasixm11/getstarted:part2    myvm1  Running        Running 6 minutes ago
ynwfi1ccwpsk  getstartedlab_web.5   edtasixm11/getstarted:part2    myvm2  Running        Running 6 minutes ago
```

```
# curl http://192.168.99.100:80
<h3>Hello World!</h3><b>Hostname:</b> 1fc12507ab77<br/><b>Visits:</b> <i>cannot
connect to Redis, counter disabled</i>

# curl http://192.168.99.101:80
<h3>Hello World!</h3><b>Hostname:</b> 0914532d7ab3<br/><b>Visits:</b> <i>cannot
connect to Redis, counter disabled</i>
```

Part 5: Stacks

Afegirem un nou servei al docker-compose.yml amb un “visualizer” que permet observar quins nodes i tasks hi ha en el swarm. Es una peça ja prefabricada que s’executa únicament en el manager.

Fitxer docker-compose.part5.yml

```
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: username/repo:tag
    deploy:
      replicas: 5
      restart_policy:
        condition: on-failure
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
    ports:
      - "80:80"
    networks:
      - webnet
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
      - webnet
networks:
  webnet:
```

```
]# eval $(docker-machine env myvm1)
[root@hostedt getstarted]# docker stack deploy -c docker-compose.part5-visualizer.yml
getstartedlab
Updating service getstartedlab_web (id: cqxp21ami4zyj7ssu2wu7wk2p)
Creating service getstartedlab_visualizer
```

Portainer

```
version: "3"
services:
  web:
        image: edtasixm11/getstarted:part2
        deploy:
        replicas: 3
        restart_policy:
        condition: on-failure
        resources:
        limits:
        cpus: "0.1"
        memory: 50M
        ports:
        - "80:80"
        networks:
        - webnet
  visualizer:
        image: dockersamples/visualizer:stable
        ports:
        - "8080:8080"
        volumes:
        - "/var/run/docker.sock:/var/run/docker.sock"
        deploy:
        placement:
        constraints: [node.role == manager]
        networks:
        - webnet
  portainer:
        image: portainer/portainer
        ports:
        - "9000:9000"
        volumes:
        - "/var/run/docker.sock:/var/run/docker.sock"
        deploy:
        placement:
        constraints: [node.role == manager]
        networks:
        - webnet
networks:
  webnet:
```

```
# docker stack deploy -c docker-compose.part5-portainer.yml getstartedlab
Creating network getstartedlab_webnet
Creating service getstartedlab_visualizer
Creating service getstartedlab_portainer
Creating service getstartedlab_web
```

Redis

```
version: "3"
services:
  web:
        image: edtasixm11/getstarted:part2
        deploy:
        replicas: 3
        restart_policy:
        condition: on-failure
        resources:
        limits:
        cpus: "0.1"
        memory: 50M
        ports:
        - "80:80"
        networks:
        - webnet
  visualizer:
        image: dockersamples/visualizer:stable
        ports:
        - "8080:8080"
        volumes:
        - "/var/run/docker.sock:/var/run/docker.sock"
        deploy:
        placement:
        constraints: [node.role == manager]
        networks:
        - webnet
  portainer:
        image: portainer/portainer
        ports:
        - "9000:9000"
        volumes:
        - "/var/run/docker.sock:/var/run/docker.sock"
        deploy:
        placement:
        constraints: [node.role == manager]
        networks:
        - webnet
```

```
  redis:
        image: redis
        ports:
        - "6379:6379"
        volumes:
        - "/home/docker/data:/data"
        deploy:
        placement:
        constraints: [node.role == manager]
        command: redis-server --appendonly yes
        networks:
        - webnet
networks:
  webnet:
```

Redis utilitza un comptador de visites que es desa en el container a /data, però aquest numero seria local al container i desapareixeria amb ell. Es desa el numero realment al host muntat el /data del container a "./data" del host, que en realitat és (/home/docker/data)

```
# docker-machine ssh myvm1 "mkdir ./data"
```

****nota**** si esteu fent el desplegament sense usar Docker Machines modifiqueu la ruta física del host manager on genereu el directori data, per exemple ***/var/tmp/data***. i assigneu permisos per tothom (chmod 777 /var/tmp/data).

Deploy de la app

```
# docker stack deploy -c docker-compose.part5-redis.yml getstartedlab
Updating service getstartedlab_web (id: zvw16tvdlb21giza020v3dt0m)
Updating service getstartedlab_visualizer (id: lc4k0dv1s4lfc5qnpd1rpnzpc)
Updating service getstartedlab_portainer (id: cud4my2clwd0ne74skq05292c)
Creating service getstartedlab_redis

# docker stack ls
NAME            SERVICES
getstartedlab           4

# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                             PORTS
cud4my2clwd0        getstartedlab_portainer   replicated          1/1                 portainer/portainer:latest
*:9000->9000/tcp
lc4k0dv1s4lf        getstartedlab_visualizer  replicated          1/1                 dockersamples/visualizer:stable
*:8080->8080/tcp
oglcnmjdu4e1        getstartedlab_redis       replicated          1/1                 redis:latest
*:6379->6379/tcp
zvw16tvdlb21        getstartedlab_web         replicated          3/3                 edtasixm11/getstarted:part2
*:80->80/tcp
```

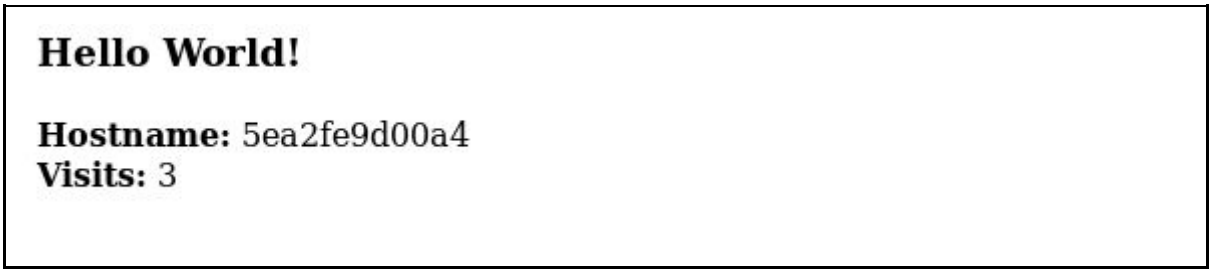
Resum:

- Accedint al port 80 de qualsevol node es mostra la pàgina de benvinguda amb un comptador de visites. Aquest és un servei amb múltiples rèpliques repartides entre els nodes.
- Accedint al port 8080 hi ha el visualizer. És un servei amb una única task en el node manager.
- Accedint al port 9000 hi ha el portainer que permet visualitzar/administrar tot!!. És una única instància corrent en el node manager.

**** nota**** en principi l'accés als ports ha de ser independent de en quin node s'executa la tasca, és a dir, podem accedir al portainer i al visualizer (que només s'executen en el node manager) usant tant l'adreça ip del manager com l'adreça ip dels workers (routing mesh).

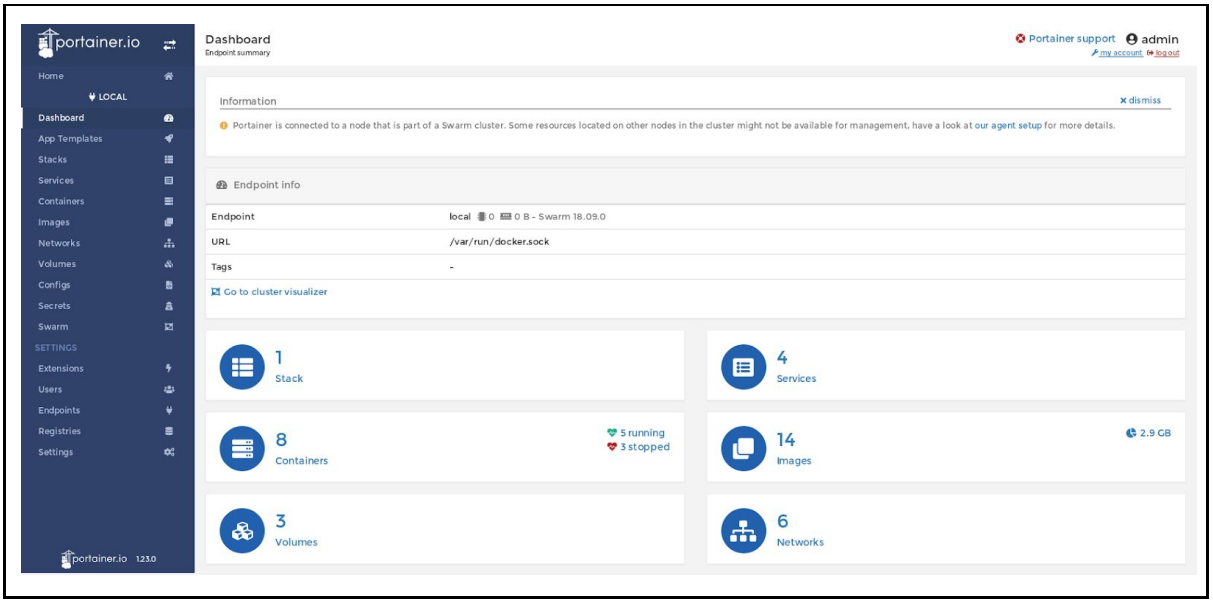
Imatges / Captures

Pàgina
(en qualsevol dels nodes: http://192.168.1.40:80)



Visualizer
(en qualsevol dels nodes: http://192.168.1.40:8080)

Portainer
(en qualsevol dels nodes http://192.168.1.40:9000)



Modificacions al desplegament:

- modifiqueu la quantitat de rèpliques que hi ha desplegades i observeu com es distribueixen en els nodes.
- Afegiu i elimineu nodes. Poseu-los en pause i drain.
- Observeu que a més del comptador de visites es mostra el nom (en realitat com que els containers no tenen nom directament assignat utilitzen el seu ID), cada visita que feu us pot atendre un container diferent.

Getting Started (actual)

Get Started de Docker Documentation:

1. Set up your Docker environment (Orientation and Setup).
2. Build an image and run it as one container (Containerizing an application).
3. Set up and use a Kubernetes environment on your development machine (Deploying Kubernetes).
4. Set up and use a Swarm environment on your development machine (Deploying to Swarm).
5. Share your containerized applications on Docker Hub (Sharing images on DockerHub).

QuickStart

Part 1: Orientation and Setup

- Doker Desktop
- Kubernetes
- Swarm

Part 2: Containerizing an application

Descarregar l'aplicació

```
git clone -b v1 https://github.com/docker-training/node-bulletin-board

cd node-bulletin-board/bulletin-board-app
```

Dockerfile

```
FROM node:6.11.5

WORKDIR /usr/src/app
COPY package.json .
RUN npm install
COPY . .

CMD [ "npm", "start" ]
```

Crear la imatge, executar i eliminar el container

```
docker image build -t bulletinboard:1.0 .

docker container run --publish 8000:8080 --detach --name bb bulletinboard:1.0
```

```
dd6515c20d88a7d898a359b03fc4d538bb02f0a20602bd1ce65cbfc83fa27da5

docker container rm --force bb
```

Part 3: Deploying to Kubernetes

bb.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bb-demo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      bb: web
  template:
    metadata:
      labels:
        bb: web
    spec:
      containers:
      - name: bb-site
        image: bulletinboard:1.0
---
apiVersion: v1
kind: Service
metadata:
  name: bb-entrypoint
  namespace: default
spec:
  type: NodePort
  selector:
    bb: web
  ports:
  - port: 8080
    targetPort: 8080
    nodePort: 30001
```

Deploying and checking

```
kubectl apply -f bb.yaml

kubectl get deployments
kubectl get services

kubectl delete -f bb.yaml
```

Part 4: Deploying to Swarm

bb-stack.yaml

```
version: '3.7'
```

```
services:
  bb-app:
        image: bulletinboard:1.0
        ports:
        - "8000:8080"
```

Deploying and checking

```
docker swarm init
docker stack deploy -c bb-stack.yaml demo

docker service ls

docker stack rm demo
Docker swarm leave --force
```

Getting Started with Docker Compose

Get Started with Docker Compose:
https://docs.docker.com/compose/gettingstarted/

1. Setup.
2. Create a Dockerfile.
3. Define services in a compose file.
4. Build and run your app.
5. Edit the compose file to add a bind mount.
6. Rebuild and run the app with compose.
7. Update the application.
8. Experiment with some other commands.

Components de l'aplicació:

- **app.py**: aplicació en python que utilitza flask i redis per mostrar una pàgina web amb un comptador de visites.
- **requirements.txt**: descripció dels requeriments de python per a funcionar. Necessari en fer el pip install.
- **Dockerfile**: per a la creació de la imatge de la app de python.
- **docker-compose.yml**: per al desplegament de dos serveis, serveu web (la app) i el servei redis.

Part 1-4: Setup - Create - Define - Build

Es desplegarà amb docker compose dos serveis: web i redis. El servei web és l'aplicació python que utilitzant flask mostra una pàgina web amb un comptador de visites. El fiyxer Dockerfile és el que descriu com crear la imatge de l'aplicació (no té nom). Utilitza app.py i requirements.txt, pip i es basa en una imatge de Alpine Linux. L'altre servei es diu redis i es basa directament en l'imatge estàndard de Redis.

app.py

```
import time
import redis
from flask import Flask

app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)

def get_hit_count():
    retries = 5
```

```
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)

@app.route('/')
def hello():
    count = get_hit_count()
    return 'Hello World! I have been seen {} times.\n'.format(count)
```

requirements.txt

```
flask
redis
```

Dockerfile

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP app.py
ENV FLASK_RUN_HOST 0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["flask", "run"]
```

docker-compose.yml

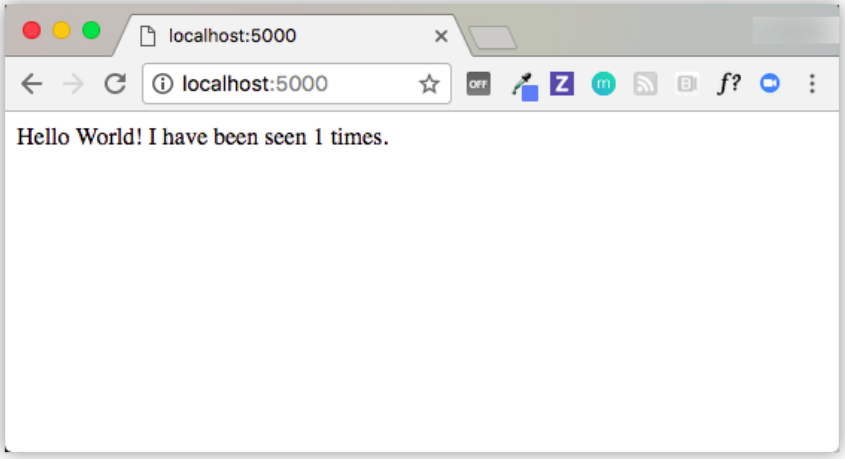
```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

Desplegament i execució

```
docker-compose up

<browser> http://localhost:5000
wget  http://localhost:5000

docker image ls
```

docker ps docker-compose ps
docker -compose down docker-compose -d up

Es mostrarà la pàgina web amb el missatge “Hello World” i un comptador de visites que es va incrementant a cada nova visita o refresc de la pàgina.



Part 5-6: Edit - Rebuild - Update

Aquesta part té per objectiu fer la aplicació ‘interectiva’. De manera que es poden fer canvis ‘en calent’ a l’aplicació i aquests es veuen refectits immediatament. Per fer això es munta el directori de desenvolupament (de fora) al directori /code (de dins).

Es modifica el fitxer de docker-compose per fer un bind-mount de manera que es munta el directori actiu (el de desenvolupament de l’aplicació) al directori /code de dins de l’aplicació. També es defineix la variable de flask per indicar-li que ha d’actuar en mode desenvolupament (en aquest mode es recarrega cada vegada que es modifica la aplicació).

Es torna a fer el desplegament de l’aplicació amb “docker-compose up” i observar que ara editant el fitxer app.py i per exemple canviant el missatge, aquest canvi es veu reflectit en la web.

docker-compose.yaml

```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
```

```
    environment:
      FLASK_ENV: development
  redis:
    image: "redis:alpine"
```

rebuild

```
docker-compose up
```

update

```
Edit app.py “Hello World from Docker”
<browser> refresh http://localhost:5000
```

```
docker-compose up -d
docker -compose ps

docker-compose run web env
docker-compose stop
docker -compose down --volumes
```

Getting Started with Swarm

Get Started with Swarm mode:
https://docs.docker.com/compose/gettingstarted/

1. Setup.
2. Create a swarm
3. Add nodes to the swarm
4. Deploy a service
5. Inspect the service
6. Scale the service
7. Delete the service
8. Apply rolling updates
9. Drain a node
10. Use swam mode routing mesh

Part 1: Setup

Caldrà disposar de tres hosts (poden ser reals o virtuals, es poden usar hosts de l'aula, màquins AMI de AWS EC2, màquines virtuals o màquines virtuals de Docker Machine).

Hosts: un host serà el manager, els altres dos seran worker1 i worker2.
Network: Cal disposar de les adreces IP dels tres hosts (en especial el manager). També cal assegurar-se que la comunicació per xarxa entre els hosts és permesa i en especial la utilització dels ports:

- TCP port **2377** for cluster management communications
- TCP and UDP port **7946** for communication among nodes
- UDP port **4789** for overlay network traffic

Part 2: Create a Swarm

En el manager

```
docker swarm init --advertise-addr <MANAGER-IP>

docker info
docker node ls

docker swarm join-token worker
docker swarm join-token manager
```

Crear el swarm, es generen els tokens necessaris per afegir nodes com a workers i com a managers.

Part 3: Add nodes to a Swarm

A cada worker

```
 docker swarm join --token
SWMTKN-1-0rmourpqssmmmru7zr2jlcpkcrmlq9an0vhlxw3xi3ntrniovn6-2fl8psdifv59bprsef
303h7z4 192.168.1.50:2377
```

El valor del token s'ha generat en crear el manager. Si no tenim a mà aquest valor es pot tornar a consultar en el manager:

```
docker swarm join-token worker
docker swarm join-token manager

docker node ls
```

Part 4: Deploy a service

En el manager:

```
docker service create --replicas 1 --name helloworld alpine ping docker.com

docker service ls
docker service ps helloworld
```

Es desplega un servei anomenat *helloworld* que utilitzant una imatge alpine fa un ping. Es desplega una única rèplica que s'executarà en un dels dnodes.

Part 5: Inspect the service

En el manager:

```
docker service inspect --pretty helloworld
docker service inspect helloworld

docker service ps helloworld
```

En el node on s'executi el servei:

```
docker ps
```

Part 6: Change the scale

En el manager:

```
# docker service scale <SERVICE-ID>=<NUMBER-OF-TASKS>

docker service scale helloworld=5

docker service ps helloworld
ID            NAME          IMAGE          NODE      DESIRED STATE  CURRENT STATE           ERROR  PORTS
k9oipcidtto9  helloworld.1  alpine:latest  lenovo    Running        Running 8 minutes ago
pmocrvncgmjz  helloworld.2  alpine:latest  lenovo    Running        Running 11 seconds ago
mxfasfjjcpmq  helloworld.3  alpine:latest  asus      Running        Running 58 seconds ago
ivf67txcp59w  helloworld.4  alpine:latest  asus      Running        Running 58 seconds ago
txh4zd3db44t  helloworld.5  alpine:latest  asus      Running        Running 58 seconds ago
```

En cada node:

```
docker ps
```

Es modifica el número de rèpliques del servei helloworld a cins, es desplegen en els nodes. Es pot observar en el manager amb l'odre docker service ps helloworld, en cada node es poden observar els containers (task) executant-se amb docker ps.

Part 7: Delete the service

En el manager:

```
docker service rm helloworld

docker service ls
docker ps
```

Part 8: Rolling updates

En el manager:

```
docker service create \
  --replicas 3 \
  --name redis \
  --update-delay 10s \
  redis:3.0.6

docker service ps redis
docker service inspect --pretty redis
docker ps
```

Llistat del docker service inspect --pretty

```
ID:		93ccxtwf9zrowfxm1ylv8ofyp
Name:		redis
Service Mode:	Replicated
 Replicas:	3
Placement:
UpdateConfig:
 Parallelism:	1
 Delay:		10s
 On failure:	pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Update order:		stop-first
RollbackConfig:
 Parallelism:	1
 On failure:	pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Rollback order:	stop-first
ContainerSpec:
 Image:
redis:3.0.6@sha256:6a692a76c2081888b589e26e6ec835743119fe453d67ecf03df7de5b7
3d69842
 Init:	false
Resources:
Endpoint Mode:	vip
```

Ara es desplega un nou servei anomenat redis basat en la imatge redis 3.0.6. Cada 10 segons docker comprova si hi ha actualitzacions a fer en el desplegament. Observeu que si en el worker no hi havia la imatge redis descarregada els primers intents han fallat.

En el manager:

```
docker service update --image redis:3.0.7 redis

docker service inspect --pretty redis
docker service ps redis
docker ps

docker service rm redis
```

Si en fer una actualització queda en pause degut a algun problema en la actualització, es pot reanudar amb l'ordre:

```
docker service update redis
```

Part 9 Drain / pause a node

En el manager:

```
docker node ls
docker node

docker service create --replicas 3 --name redis --update-delay 10s redis:3.0.6
docker service ps redis

docker node update --availability drain worker1

docker node inspect --pretty worker1
docker service ps redis
docker node ls
```

docker node inspect --pretty asus

```
ID:                     jeqk9p0ywuq7v7cknmdp099gi
Hostname:               asus
Joined at:              2020-02-04 17:34:40.170825898 +0000 utc
Status:
State:                  Ready
Availability:           Drain
 Address:       192.168.1.40
Platform:
 Operating System:      linux
 Architecture:          x86_64
Resources:
 CPUs:                  2
 Memory:        3.76GiB
Plugins:
 Log:           awslogs, fluentd, gcplogs, gelf, journald, json-file, local, logentries, splunk, syslog
 Network:       bridge, host, macvlan, null, overlay
 Volume:        local
Engine Version:         18.09.0
TLS Info:
 TrustRoot:
-----BEGIN CERTIFICATE-----
MIIBazCCARCgAwIBAgIUBSYXmiz6wSBEnmWWS1Sbko5YvjswCgYIKoZIzj0EAwIw
EzERMA8GA1UEAxMIc3dhcm0tY2EwHhcNMjAwMjA0MTcxMzAwWhcNNDAwMTMwMTcx
MzAwWjATMREwDwYDVQQDEwhzd2FybS1jYTBZMBMGByqGSM49AgEGCCqGSM49AwEH
A0IABB2Ubclc209XH5eS2Qi4twCtdN0/rElCu3DQ5Vy/m4PrvL+soJ2/4MJzGkO+
smnVyVldZtHBRXSfavrUNT8JUEqjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMB
Af8EBTADAQH/MB0GA1UdDgQWBBT/OBcCm/WKwOF7xYQsGJnKOiiKxjAKBggqhkjO
PQQDAgNJADBGAiEAqLsqOx1dmYeZMTm0yTddwcNxDxQTzVxcAvy3rj5GF7ACIQDC
xWw1wvq/zR1nZtN/cXr5nrdFovgswmFDOoC6EPfe+Q==
-----END CERTIFICATE-----

 Issuer Subject:    MBMxETAPBgNVBAMTCHN3YXJtLWNh
 Issuer Public Key:
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEHZRtyVzbT1cfl5LZCLi3AK103T+sSUK7cNDlXL+bg+u8v6ygnb/gwnMaQ76yadXJUh1m0cFFdJ9q+tQ1PwlQSg==
```

Podem posar un node que estava *active* a estat *drain*, això fa que no accepti tasques i que se li eliminin les que executava passant-les a un altre node. Atenció, es tracta de tasques del swarm, el host pot executar tranquil·lament containers amb docker run i amb docker-compose.

Podem tornar a activar el node fent en el manager:

```
docker node update --availability active worker1
```

```
docker node inspect --pretty worker1
docker service ps redis
docker node ls

docker service rm redis
```

Pausar un node fa que no acceti moves task però continua executant les que tenia assignades.

Part 10: Use swarm mode routing mesh

To use the ingress network in the swarm, you need to have the following ports open between the swarm nodes before you enable swarm mode:

- Port 7946 TCP/UDP for container network discovery.
- Port 4789 UDP for the container ingress network.

You must also open the published port between the swarm nodes and any external resources, such as an external load balancer, that require access to the port.

Crear un servei que publica el port 8080:

```
docker service create \
  --name my-web \
  --publish published=8080,target=80 \
  --replicas 2 \
  nginx
```

Per publicar un port per un servei ja definit:

```
docker service update \
  --publish-add published=<PUBLISHED-PORT>,target=<CONTAINER-PORT> \
  <SERVICE>
```

En una xarxa ingress routing mesh es pot accedir a qualsevol dels nodes i accedir al servei amb independència de si el node extà o no executant la tasca en concret. podem accedir via:

```
http://ip-manager:8080
http://ip-worker1:8080
http://ip-worker2:8080
```

Es mostrarà el servidor nginx:

Podem fer drain d'un dels node i verificar que usant la seva adreça ip encara respon al servei nginx:

```
docker node update --availability drain worker1
```

```
docker service rm my-web
```

Publish a port for tcp only or udp only

tcp only

```
docker service create --name dns-cache \
  --publish published=53,target=53 \
  dns-cache
```

```
docker service create --name dns-cache \
  -p 53:53 \
  dns-cache
```

tcp and udp

```
docker service create --name dns-cache \
  --publish published=53,target=53 \
  --publish published=53,target=53,protocol=udp \
  dns-cache
```

```
docker service create --name dns-cache \
  -p 53:53 \
  -p 53:53/udp \
  dns-cache
```

udp only

```
docker service create --name dns-cache \
  --publish published=53,target=53,protocol=udp \
  dns-cache
```

```
docker service create --name dns-cache \
  -p 53:53/udp \
  dns-cache
```

Bypass the routing mesh

To bypass the routing mesh, you must use the long --publish service and set mode to host. If you omit the mode key or set it to ingress, the routing mesh is used.

```
docker service create --name dns-cache \
  --publish published=53,target=53,protocol=udp,mode=host \
  --mode global \
  dns-cache
```

Usant mode=host no s'aplica per el port indicat el routing mesh sinó que s'sccedeix directament al port del host/node al que s'accedeix (pot ser que el servei el port tingui un task que hi escolta o no).

Docker: Descripció General

Swarm

Continguts:

- ❏ Docker swarm
- ❏ Docker node
- ❏ Routing Mesh
- ❏ Docker machine
- ❏ AWS EC2

Swarm

Docker Documentation:
Swarm mode overwiew
Getting Started with swarm mode
Docker swarm CLI

Description:

A swarm consists of multiple Docker hosts which run in **swarm mode** and act as managers (to manage membership and delegation) and workers (which run swarm services).
The cluster management and orchestration features.
Docker works to maintain that desired state. For instance, if a worker node becomes unavailable, Docker schedules that node's tasks on other nodes. A *task* is a running container which is part of a swarm service and managed by a swarm manager, as opposed to a standalone container.
One of the key advantages of swarm services over standalone containers is that you can modify a service's configuration, including the networks and volumes it is connected to, without the need to manually restart the service. Docker will update the configuration, stop the service tasks with the out of date configuration, and create new ones matching the desired configuration.
When Docker is running in swarm mode, you can still run standalone containers on any of the Docker hosts participating in the swarm, as well as swarm services. A key difference between standalone containers and swarm services is that only swarm managers can manage a swarm, while standalone containers can be started on any daemon. Docker daemons can participate in a swarm as managers, workers, or both.
In the same way that you can use Docker Compose to define and run containers, you can define and run swarm service stacks.

The swarm manager uses **ingress load balancing** to expose the services you want to make available externally to the swarm.

Característiques:

- Cluster management integrated with Docker Engine
- Decentralized design
- Declarative service model
- Scaling
- Desired state reconciliation
- Multi-host networking
- Service discovery
- Load balancing
- Secure by default
- Rolling updates

Commands:

- ★ docker swarm ca Display and rotate the root CA
- ★ docker swarm init Initialize a swarm
- ★ docker swarm join Join a swarm as a node and/or manager
- ★ docker swarm join-token Manage join tokens
- ★ docker swarm leave Leave the swarm
- ★ docker swarm unlock Unlock swarm
- ★ docker swarm unlock-key Manage the unlock key
- ★ docker swarm update Update the swarm

Tutorial:

Un swarm és un conjunt de nodes que formen un cluster. Un node pot ser un host físic, una màquina virtual, un host de un cloud o un host virtual docker-machine. Un node és un host que està executant el daemon de docker, el docker engine.

Els nodes poden ser:

- ❏ Manager: gestiona el swarm. N'hi pot haver varis. També executen tasks.
- ❏ Worker: només executa tasks (les que se li assignen).

Allò que es pot fer amb un swarm principalment és:

- Crear el swarm.
- Assignar nodes al swarm, que poden ser workers o managers.
- Demanar quin és el token de manager o de worker (per si no el tenim a mà).
- Eliminar nodes del swarm.

Les ordres per gestionar swarm són:

- docker swarm
- docker node

Es crea un swarm amb *docker swarm init*. Això a part de fer que el host es converteixi en un manager genera per stdout un missatge amb la instrucció necessària per afegir nodes al

swarm. Copiar aquesta ordre (amb el token) per afegir nodes al swarm. Es poden afegir nodes worker i manager. Per tornar a veure quin és l'ordre a fer per afegir nodes podem fer docker swarm *join-token worker|manager*.

El ***token*** és un hash i s'utilitza una clau CA-key per afegir/treure nodes al swarm. Hi pot haver varis managers i workers, almenys un manager. El ***leader*** s'escull per quorum.

Exemples:

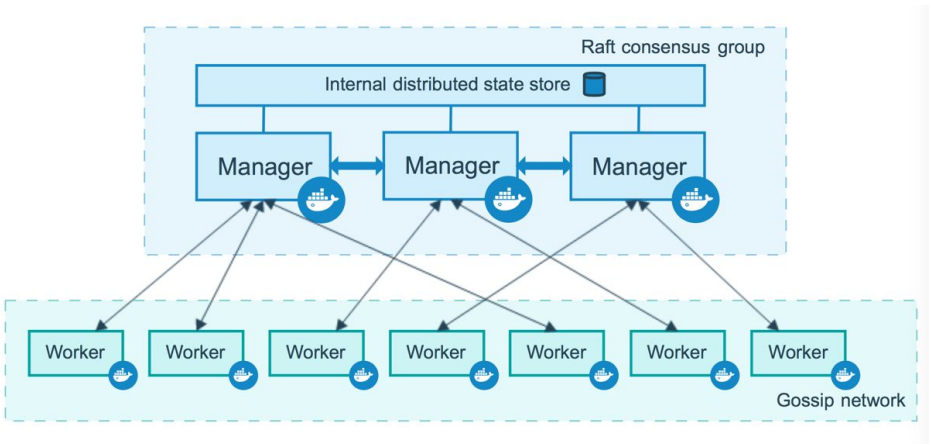
```
docker swarm init
docker swarm init --advertise-addr <MANAGER-IP>

docker swarm join   --token
SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c 192.168.99.100:2377

docker swarm join-token worker
docker swarm join-token manager

docker swarm leave
docker swarm leave --force
```

Node

Docker Documentation:
- How Nodes Work
- Docker Node CLI
- Manage nodes in a Swarm

Description:
A **node** is an instance of the Docker engine participating in the swarm. You can also think of this as a Docker node. You can run one or more nodes on a single physical computer or cloud server, but production swarm deployments typically include Docker nodes distributed across multiple physical and cloud machines.
To deploy your application to a swarm, you submit a service definition to a **manager node**. The manager node dispatches units of work called tasks to worker nodes.
Manager nodes also perform the orchestration and cluster management functions required to maintain the desired state of the swarm. Manager nodes elect a single leader to conduct orchestration tasks.
Worker nodes receive and execute tasks dispatched from manager nodes. By default manager nodes also run services as worker nodes, but you can configure them to run manager tasks exclusively and be manager-only nodes. An agent runs on each worker node and reports on the tasks assigned to it. The worker node notifies the manager node of the current state of its assigned tasks so that the manager can maintain the desired state of each worker.

There are two types of nodes: **managers** and **workers**.



Commands:

- ★ docker node demote Demote one or more nodes from manager in the swarm
- ★ docker node inspect Display detailed information on one or more nodes
- ★ docker node ls List nodes in the swarm
- ★ docker node promote Promote one or more nodes to manager in the swarm
- ★ docker node ps List tasks running on one or more nodes, defaults to current node
- ★ docker node rm Remove one or more nodes from the swarm
- ★ docker node update Update a node

Tutorial:

Podríem dir que un node és un host, físic o virtual, però en realitat és un hots que executa docker engine. La gestió que fem amb els nodes és llistar-los per observar en quin estat estan, per llistar les tasks que executen, eliminar-los i consultar-ne les característiques.

També podem degradar i promoure un node, és a dir, passar-lo de manager a worker i a l'inrevés.

The AVAILABILITY column shows whether or not the scheduler can assign tasks to the node:

- **Active** means that the scheduler can assign tasks to the node.
- **Pause** means the scheduler doesn't assign new tasks to the node, but existing tasks remain running.
- **Drain** means the scheduler doesn't assign new tasks to the node. The scheduler shuts down any existing tasks and schedules them on an available node.

The MANAGER STATUS column shows node participation in the Raft consensus:

- **No value** indicates a worker node that does not participate in swarm management.
- **Leader** means the node is the primary manager node that makes all swarm management and orchestration decisions for the swarm.

- **Reachable** means the node is a manager node participating in the Raft consensus quorum. If the leader node becomes unavailable, the node is eligible for election as the new leader.
- **Unavailable** means the node is a manager that can't communicate with other managers. If a manager node becomes unavailable, you should either join a new manager node to the swarm or promote a worker node to be a manager.

Label Metadata

Node labels provide a flexible method of node organization. You can also use node labels in service constraints. Apply constraints when you create a service to limit the nodes where the scheduler assigns tasks for the service.

Run docker node update --label-add on a manager node to add label metadata to a node. The --label-add flag supports either a <key> or a <key>=<value> pair. Pass the --label-add flag once for each node label you want to add

Exemples:

```
docker node ls
docker node update --availability drain <NODE-ID>
docker node update --availability active <NODE-ID>
docker node inspect --pretty worker1
docker node promote node-3 node-2
docker node demote node-3 node-2
docker node update --role manager
docker node update --role worker
docker node rm node-2
docker node update --label-add foo --label-add bar=baz node-1  node-1
```

Routing Mesh

Docker Documentation:
Use swarm mode routing mesh
Use overlay networks

Description:

Docker Engine swarm mode makes it easy to publish ports for services to make them available to resources outside the swarm. All nodes participate in an ingress **routing mesh**. The routing mesh enables each node in the swarm to accept connections on published ports for any service running in the swarm, even if there's no task running on the node. The routing mesh routes all incoming requests to published ports on available nodes to an active container.
To use the ingress network in the swarm, you need to have the following ports open between the swarm nodes before you enable swarm mode:

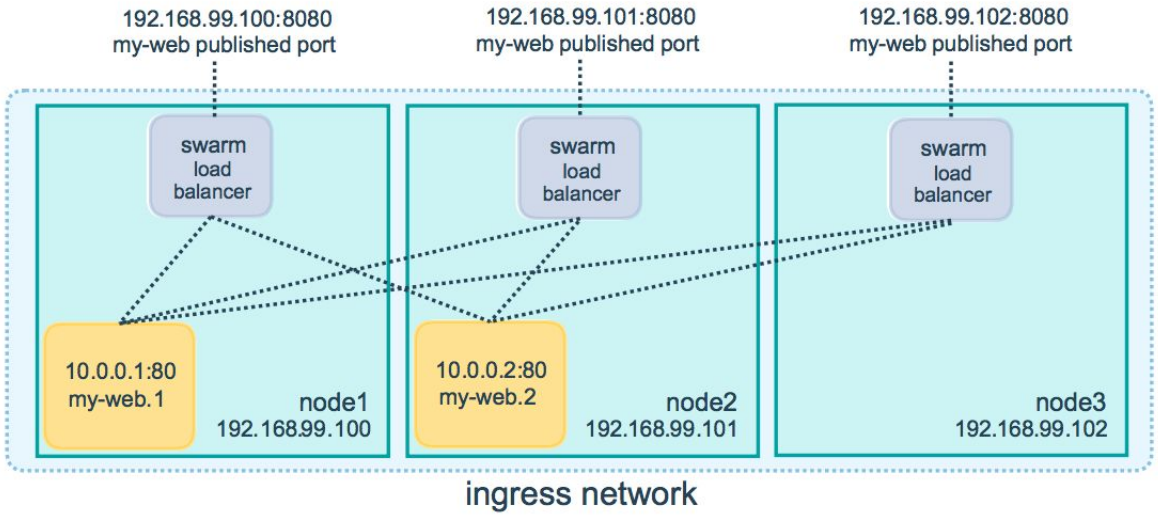
- Port 7946 TCP/UDP for container network discovery.

- Port 4789 UDP for the container ingress network.

You must also open the published port between the swarm nodes and any external resources, such as an external load balancer, that require access to the port.

```
$ docker service create  --name my-web  --publish published=8080,target=80  --replicas 2  nginx
```

The routing mesh listens on the published port for any IP address assigned to the node. For externally routable IP addresses, the port is available from outside the host. For all other IP addresses the access is only available from within the host.



Docker-Machine

<pendent>

AWS EC2

<pendent>

Desplegament

Continguts:

- ❏ Stack
- ❏ Service
- ❏ Task

- ❏ Docker compose 2.0
- ❏ Docker compose 3.x

Stack

Docker documentation:
Deploy a stack to a Swarm

Description:
When running Docker Engine in swarm mode, you can use docker stack deploy to deploy a complete application stack to the swarm. The deploy command accepts a stack description in the form of a Compose file.

Commands:

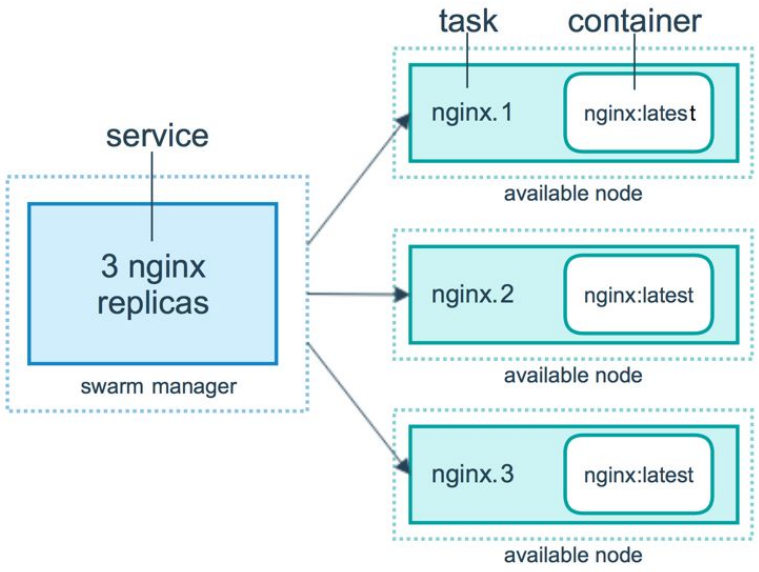
- ★ docker stack deploy Deploy a new stack or update an existing stack
- ★ docker stack ls List stacks
- ★ docker stack ps List the tasks in the stack
- ★ docker stack rm Remove one or more stacks
- ★ docker stack services List the services in the stack

Service

Docker Documentation:
How Services Work

Description:
A **service** is the definition of the tasks to execute on the manager or worker nodes. It is the central structure of the swarm system and the primary root of user interaction with the swarm.
When you create a service, you specify which container image to use and which commands to execute inside running containers.
In the **replicated services** model, the swarm manager distributes a specific number of replica tasks among the nodes based upon the scale you set in the desired state.
For **global services**, the swarm runs one task for the service on every available node in the cluster.

When you deploy the service to the swarm, the swarm manager accepts your service definition as the desired state for the service. Then it schedules the service on nodes in the swarm as one or more replica tasks. The tasks run independently of each other on nodes in the swarm.
For example, imagine you want to load balance between three instances of an HTTP listener. The diagram below shows an HTTP listener service with three replicas. Each of the three instances of the listener is a task in the swarm.



A container is an isolated process. In the swarm mode model, each task invokes exactly one container. A task is analogous to a “slot” where the scheduler places a container. Once the container is live, the scheduler recognizes that the task is in a running state. If the container fails health checks or terminates, the task terminates.

Tutorial:

Un servei passa per diversos estats, com són:

- Pending: A service may be configured in such a way that no node currently in the swarm can run its tasks. In this case, the service remains in state

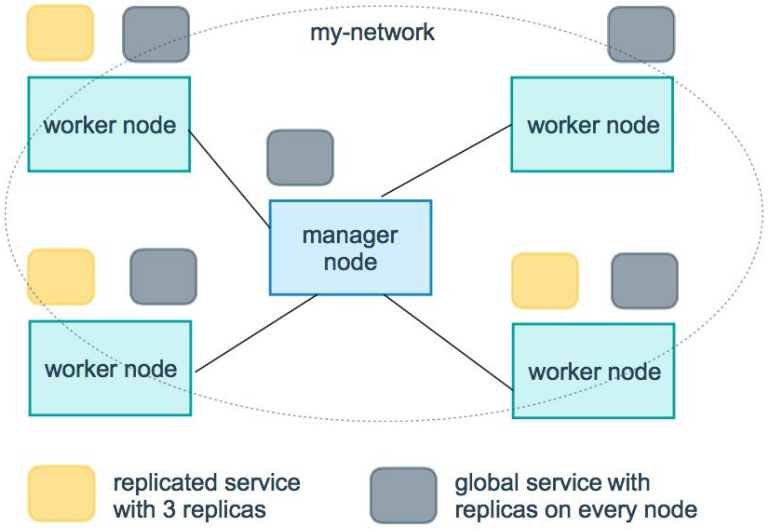
Tipus de serveis:

- Replicated
- Global

For a replicated service, you specify the number of identical tasks you want to run. For example, you decide to deploy an HTTP service with three replicas, each serving the same content.

A global service is a service that runs one task on every node. There is no pre-specified number of tasks. Each time you add a node to the swarm, the orchestrator creates a task and the scheduler assigns the task to the new node. Good candidates for global services are monitoring agents, an anti-virus scanners or other types of containers that you want to run on every node in the swarm.

The diagram below shows a three-service replica in yellow and a global service in gray.



Task

Description:
A **task** carries a Docker container and the commands to run inside the container. It is the atomic scheduling unit of swarm. Manager nodes assign tasks to worker nodes according to the number of replicas set in the service scale. Once a task is assigned to a node, it cannot move to another node. It can only run on the assigned node or fail.

A task is the atomic unit of scheduling within a swarm. When you declare a desired service state by creating or updating a service, the orchestrator realizes the desired state by scheduling tasks. For instance, you define a service that instructs the orchestrator to keep three instances of an HTTP listener running at all times. The orchestrator responds by creating three tasks. Each task is a slot that the scheduler fills by spawning a container. The container is the instantiation of the task. If an HTTP listener task subsequently fails its health check or crashes, the orchestrator creates a new replica task that spawns a new container.

Tasks are execution units that run once to completion. When a task stops, it isn't executed again, but a new task may take its place.

States of Swarm tasks:

- NEW The task was initialized.
- PENDING Resources for the task were allocated.
- ASSIGNED Docker assigned the task to nodes.
- ACCEPTED The task was accepted by a worker node. If a worker node rejects the task, the state changes to
- REJECTED. PREPARING Docker is preparing the task.
- STARTING Docker is starting the task.
- RUNNING The task is executing.
- COMPLETE The task exited without an error code.
- FAILED The task exited with an error code.

- SHUTDOWN Docker requested the task to shut down.
- REJECTED The worker node rejected the task.
- ORPHANED The node was down for too long.
- REMOVE The task is not terminal but the associated service was removed or scaled down.

Examples:

```
$ docker service ps webserver
ID            NAME            IMAGE          NODE          DESIRED STATE     CURRENT STATE           ERROR
PORTS
owsz0yp6z375      webserver.1       nginx          UbuntuVM          Running           Running 44 seconds ago
j91iahr8s74p      \_ webserver.1    nginx          UbuntuVM          Shutdown          Failed 50 seconds ago   "No such
container: webserver.…"
7dyaszg13mw2      \_ webserver.1    nginx          UbuntuVM          Shutdown          Failed 5 hours ago      "No such
container: webserver.…"
```

Docker Compose

Docker Documentation:
- Docker compose tutorial
- Install docker-compose
- Docker-compose file version 3 reference
- Compose command line reference

Description:

Compose is a tool for defining and running complex applications with Docker. With Compose, you define a multi-container application in a single file, then spin your application up in a single command which does everything that needs to be done to get it running.

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

Using Compose is basically a three-step process:

1. Define your app's environment with a Dockerfile so it can be reproduced anywhere.
2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
3. Run docker-compose up and Compose starts and runs your entire app.

Compose has commands for managing the whole lifecycle of your application:

- Start, stop, and rebuild services
- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service

Service

Continguts:

- ❏ Routing mesh
- ❏ Global
- ❏ Constraints
- ❏ Labels
- ❏ Replicas
- ❏ Recursos

Docker Documentation:
Deploy services in a swarm
Deploy a Stack to a swarm

Description:
To deploy an application image when Docker Engine is in swarm mode, you create a service. Frequently a service is the image for a microservice within the context of some larger application. Examples of services might include an HTTP server, a database, or any other type of executable program that you wish to run in a distributed environment.
When you create a service, you specify which container image to use and which commands to execute inside running containers. You also define options for the service including:

- the port where the swarm makes the service available outside the swarm
- an overlay network for the service to connect to other services in the swarm
- CPU and memory limits and reservations
- a rolling update policy
- the number of replicas of the image to run in the swarm

<afegir la descripció i exemples de definició de serveis >

Docker Machine

<veure l'explicació de docker machine al Getting Started original: Part 4: Swarm Machine>

Documentació:

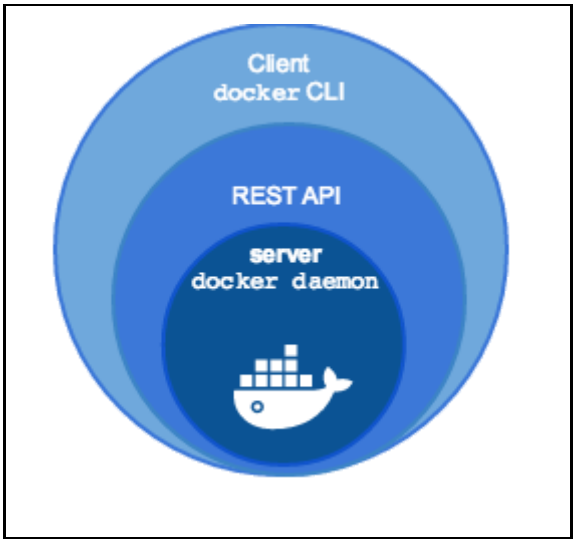
- Docker machine overview

Overview

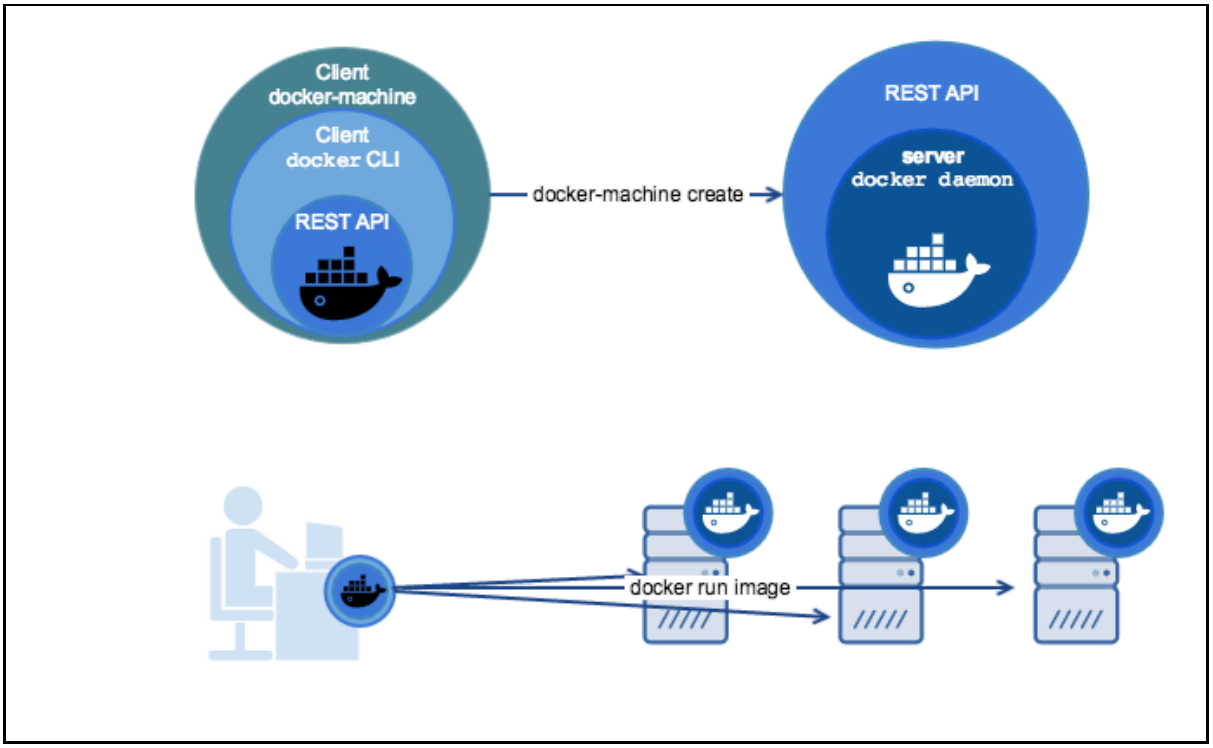
Docker Machine is a tool that lets you install Docker Engine on virtual hosts, and manage the hosts with docker-machine commands. You can use Machine to create Docker hosts on your local Mac or Windows box, on your company network, in your data center, or on cloud providers like Azure, AWS, or Digital Ocean.

Using docker-machine commands, you can start, inspect, stop, and restart a managed host, upgrade the Docker client and daemon, and configure a Docker client to talk to your host.

When people say "Docker" they typically mean **Docker Engine**, the client-server application made up of the Docker daemon, a REST API that specifies interfaces for interacting with the daemon, and a command line interface (CLI) client that talks to the daemon (through the REST API wrapper). Docker Engine accepts docker commands from the CLI, such as docker run <image>, docker ps to list running containers, docker image ls to list images, and so on.



Docker Machine is a tool for provisioning and managing your Dockerized hosts (hosts with Docker Engine on them). Typically, you install Docker Machine on your local system. Docker Machine has its own command line client docker-machine and the Docker Engine client, docker. You can use Machine to install Docker Engine on one or more virtual systems. These virtual systems can be local (as when you use Machine to install and run Docker Engine in VirtualBox on Mac or Windows) or remote (as when you use Machine to provision Dockerized hosts on cloud providers).



Install Docker Machine

```
curl -L
https://github.com/docker/machine/releases/download/v0.14.0/docker-machine-`uname
-s`-`uname -m` >/tmp/docker-machine && \
sudo install /tmp/docker-machine /usr/local/bin/docker-machine
```

Instal·lar VirtualBox

```
wget
https://download.virtualbox.org/virtualbox/6.1.2/VirtualBox-6.1-6.1.2_135662_fedora29-1.x
86_64.rpm

dnf -y install VirtualBox-6.1-6.1.2_135662_fedora29-1.x86_64.rpm
dnf -y install gcc make perl
```

```
docker-machine create --driver virtualbox default
```

Docker Network

Documentació:

- ❏ Docker Network Documentation
- ❏ Docker & iptables
- ❏ Docker Swarm reference Architecture
- ❏ Tutorial: Networking with standalone containers

Comandes

Docker network

docker network connect	Connect a container to a network
docker network create	Create a network
docker network disconnect	Disconnect a container from a network
docker network inspect	Display detailed information on one or more networks
docker network ls	List networks
docker network prune	Remove all unused networks
docker network rm	Remove one or more networks

Docker network create

--attachable	Enable manual container attachment
--aux-address	Auxiliary IPv4 or IPv6 addresses used by Network driver
--config-from	The network from which copying the configuration
--config-only	Create a configuration only network --driver , -d bridge Driver to manage the Network
--gateway	IPv4 or IPv6 Gateway for the master subnet
--ingress	Create swarm routing-mesh network
--internal	Restrict external access to the network
--ip-range	Allocate container ip from a sub-range
--ipam-driver	IP Address Management Driver
--ipam-opt	Set IPAM driver specific options
--ipv6	Enable IPv6 networking
--label	Set metadata on a network
--opt , -o	Set driver specific options
--scope	Control the network's scope
--subnet	Subnet in CIDR format that represents a network segment

Network Drives

Docker's networking subsystem is pluggable, using drivers. Several drivers exist by default, and provide core networking functionality:

- **bridge**: The default network driver. If you don't specify a driver, this is the type of network you are creating. **Bridge networks are usually used when your applications run in standalone containers that need to communicate.** See bridge networks.

- **host**: For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly. host is only available for swarm services on Docker 17.06 and higher. See use the host network.

- **overlay**: Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers. See overlay networks.

- **macvlan**: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses. Using the macvlan driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host's network stack. See Macvlan networks.

- **none**: For this container, disable all networking. Usually used in conjunction with a custom network driver. none is not available for swarm services. See disable container networking.

- Network plugins: You can install and use third-party network plugins with Docker. These plugins are available from Docker Hub or from third-party vendors. See the vendor's documentation for installing and using a given network plugin.

Bridge Networks

- ❏ Use bridge networks
- ❏ Tutorial: Networking with standalone containers

Per defecte són els tipus de xarxa que crea.
In terms of Docker, a bridge network uses a software bridge which allows containers connected to the same bridge network to communicate, while providing isolation from containers which are not connected to that bridge network. The Docker bridge driver automatically installs rules in the host machine so that containers on different bridge networks cannot communicate directly with each other. Bridge networks apply to containers running on the same Docker daemon host.

When you start Docker, a default bridge network (also called bridge) is created automatically, and newly-started containers connect to it unless otherwise specified. You can also create user-defined custom bridge networks.

Dos tipus:

- default bridge network (la típica 172.17.0.0/16)
- user-definied bridge network: la xarxa que definim amb l'ordre docker network create.

Containers connected to the same user-defined bridge network automatically expose all ports to each other, and no ports to the outside world.

Containers on the default bridge network can only access each other by IP addresses, unless you use the --link option, which is considered legacy. On a user-defined bridge network, containers can resolve each other by name or alias.

User-defined bridge networks are created and configured using docker network create. If different groups of applications have different network requirements, you can configure each user-defined bridge separately, as you create it.

Containers connected to the same user-defined bridge network effectively expose all ports to each other. For a port to be accessible to containers or non-Docker hosts on different networks, that port must be *published* using the -p or --publish flag.

When the container starts, it can only be connected to a single network, using --network. However, you can connect a running container to multiple networks using docker network connect.

A container's hostname defaults to be the container's ID in Docker. You can override the hostname using --hostname. When connecting to an existing network using docker network connect, you can use the --alias flag to specify an additional network alias for the container on that network.

En resum:

- Existeix una xarxa default-bridge (172.17.0.0/16), els containers només s'identifiquen entre ells per adreça ip, no per nom, dns. Més bàsica.
- Usar user-definied bridge network permet crear una xarxa per a cada grup de containers que la necessitin, tenen accés a tots els ports per ip i per nom de host d'entre tots els container que en formen part.
- En engegar un container només pot pertànyer a una xarxa però es pot assignar a més xarxes dinàmicament amb docker network connect.
- Un container que pertany a més d'una xarxa pot tenir àlies (més d'un nom de host).

Published Ports

By default, when you create a container, it does not publish any of its ports to the outside world. To make a port available to services outside of Docker, or to Docker containers which

are not connected to the container's network, use the --publish or -p flag. This creates a firewall rule which maps a container port to a port on the Docker host. Here are some examples.

-p 8080:80	Map TCP port 80 in the container to port 8080 on the Docker host.
-p 192.168.1.100:8080:80	Map TCP port 80 in the container to port 8080 on the Docker host for connections to host IP 192.168.1.100.
-p 8080:80/udp	Map UDP port 80 in the container to port 8080 on the Docker host.
-p 8080:80/tcp -p 8080:80/udp	Map TCP port 80 in the container to TCP port 8080 on the Docker host, and map UDP port 80 in the container to UDP port 8080 on the Docker host.

DNS Services

By default, a container inherits the DNS settings of the Docker daemon, including the /etc/hosts and /etc/resolv.conf.You can override these settings on a per-container basis.

--dns	The IP address of a DNS server. To specify multiple DNS servers, use multiple --dns flags. If the container cannot reach any of the IP addresses you specify, Google's public DNS server 8.8.8.8 is added, so that your container can resolve internet domains.
--dns-search	A DNS search domain to search non-fully-qualified hostnames. To specify multiple DNS search prefixes, use multiple -dns-search flags.
--dns-opt	A key-value pair representing a DNS option and its value. See your operating system's documentation for resolv.conf for valid options.
--hostname	The hostname a container uses for itself. Defaults to the container's ID if not specified.

Overlay Networks

- ❏ Use Overlay networks
- ❏ Tutorial: Networking with overlay networks

When you initialize a swarm or join a Docker host to an existing swarm, two new networks are created on that Docker host:

- an overlay network called **ingress**, which handles control and data traffic related to swarm services. When you create a swarm service and do not connect it to a user-defined overlay network, it connects to the ingress network by default.
- a bridge network called **docker_gwbridge**, which connects the individual Docker daemon to the other daemons participating in the swarm.

You can create user-defined overlay networks using docker network create, in the same way that you can create user-defined bridge networks. Services or containers can be connected to more than one network at a time. Services or containers can only communicate across networks they are each connected to.

You need the following ports open to traffic to and from each Docker host participating on an overlay network:

- **TCP port 2377** for cluster management communications
- **TCP and UDP port 7946** for communication among nodes
- **UDP port 4789** for overlay network traffic

Comandes:

```
$ docker network create -d overlay my-overlay
$ docker network create -d overlay --attachable my-attachable-overlay
```

By default, swarm services which publish ports do so using the routing mesh. When you connect to a published port on any swarm node (whether it is running a given service or not), you are redirected to a worker which is running that service, transparently. Effectively, Docker acts as a load balancer for your swarm services. Services using the routing mesh are running in *virtual IP (VIP) mode*. Even a service running on each node (by means of the --mode global flag) uses the routing mesh. When using the routing mesh, there is no guarantee about which Docker node services client requests.

Host Networking

- ❏ Use host networking
- ❏ Tutorial: Networking using the host network

f you use the host network mode for a container, that container's network stack is not isolated from the Docker host (the container shares the host's networking namespace), and the container does not get its own IP-address allocated. For instance, if you run a container which binds to port 80 and you use host networking, the container's application is available on port 80 on the host's IP address.
From a networking point of view, this is the same level of isolation as if the container service process were running directly on the Docker host and not in a container.

Network None

- ❏ Disable networking for a container

If you want to completely disable the networking stack on a container, you can use the --network none flag when starting the container. Within the container, only the loopback device is created.

The following example illustrates this:

```
$ docker run --rm -dit \
  --network none \
  --name no-net-alpine \
  alpine:latest \
  ash

$ docker exec no-net-alpine ip link show
$ docker exec no-net-alpine ip route show
```

Docker Compose

Documentació:

- ❏ Overview of docker compose
- ❏ Docker compose reference

Here is a sample Compose file from the voting app sample used in the Docker for Beginners lab topic on Deploying an app to a Swarm: Cats & Dogs

Docker Service

- ❏ Docker service create

AWS Amazon Web Services

Documentació a mirar:

- ❏ AWS Getting Started Resource Center
- ❏ Launch a virtual machine with ec2
- ❏ Welcom to the docker cloud docs
- ❏ Docker Cloud
- ❏ Link Amazon Web Services to Docker Cloud
- ❏ Docker for AWS IAM permissions

Creació / Configuració del compte

1. Crear el compte

Including use of Amazon EC2, Amazon S3, and Amazon DynamoDB
Visit aws.amazon.com/free for full offer terms

- Crear el compte
- Indicar una targeta de crèdit
- Es rep una trucada telefònica on cal introduïr el número que mostra la web. Va al revés que generalment es rep el número al telèfon, aqui el número és a la web d'enregistrament i la trucada és de veu, enregistrada, i cal escriure al codi al teclat del telèfon.
- Seleccionar el bàsic-plan-free.
- Ok

Documentació per 'engegar':
Getting started resource center

2. IAM Roles

3. IC2 Management

Documentació:

- ❏ **Amazon EC2 - User Guide for Linux Instances**

- ❏ Amazon EC2 Vídeos
- ❏ Getting Started Amazon center
- ❏ Launch a virtual machime with Amazon EC3 (10-minutes tutorial)
- ❏ Deploy Docker Containers on Amazon EC2 Container Service (10-minutes tutorial)

Descripció de caracterís tiques de les instàncies:

- ❏ Instance-types
- ❏ Tip: usar ligthsail: **Did you know?** AWS made it even easier to launch a Linux virtual private server. Jumpstart your virtual machine with Amazon Lightsail >>

Creació:

1. Clicar launch instance
2. Configure your instance:
    a. triem la primera de les AMI (free tier)
    Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type - ami-cae150b7
    b. Instance type: escollir el tipus de mem, cpu, etc.
    c. Launch
    d. Create a KeyPair, clau pública i privada. Escollir “create a new pair” i assignar-li un nom, per exemple “MyKeyPair”. Clicar a desar per desar la clau privada dins de .ssh. Això permet accedir via SSH a la instància.
    Desar la clau privada “MyKeyPair.pem” dins de .ssh amb permisos 600.
    Click Launch. Genera la instància.
    Podem crear alertes per si pasem del cost de free tier

SYSTEMS MANAGER SHARED RESOURCES
Managed InstancesActivationsDocumentsMaintenance WindowsParameter StorePatches
Launch Status
Your instances are now launching
The following instance launches have been initiated: i-00b87dbadf063389f View launch log
Get notified of estimated charges
Create billing alerts to get an email notification when estimated charges on your AWS bill exceed an amount you define (for example, if you exceed the free usage tier).
How to connect to your instances
Your instances are launching, and it may take a few minutes until they are in the **running** state, when they will be ready for you to use. Usage hours on your new instances will start immediately and continue to accrue until you stop or terminate your instances.
Click **View Instances** to monitor your instances' status. Once your instances are in the **running** state, you can **connect** to them from the Instances screen. Find out how to

connect to your instances.
Here are some helpful resources to get you started

- **How to connect to your Linux instance**
- Learn about AWS Free Usage Tier
- Amazon EC2: User Guide
- Amazon EC2: Discussion Forum

While your instances are launching you can also

- Create status check alarms to be notified when these instances fail status checks. (Additional charges may apply)
- Create and attach additional EBS volumes (Additional charges may apply)
- Manage security groups

e. View instances
f. Observar les dades de la instància, el tipus, la regió i especialment: **anotar l'adreça IP** per poder-hi connectar!.

3. Connect to your instance.
   Per connectar amb la instància farem SSH usant l'usuari prototipus "ec2-user" i el fitxer de claus que hem desat abans. Si la màquina és fedora l'user és "fedora".
   Exemple d'ordre:
   $ ssh -i ~/.ssh/MyKeyPair.pem ec2-user@<ip-instància>
   $ ssh -i ~/.ssh/MyKeyPair.pem fedora@<ip-instància>

   Ens podem trobar que en iniciar ens demana fer un update per actualitzar el sistema. Ya tà!

4. **Finalitzar la instància**:
   Al panell de control DashBoard de EC2
   Seleccionar la instància, desplegar **Actions** i clicar a **terminate**.

5. Netx Step (pagant!!!): Assignar un nom de domini a la instància:
   Getting a Domain for your instance

Pendent:!!!

- **User Guide for Linux Instances**
  https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html
- **AWS CLI command reference**
- **x**

Key Pairs

Si no recordem o em perdut al host físic les keypairs usades per accedir a les instàncies, podem descarregar-les de nou.

Al dashboard esquerra: key Pairs i llista les parelles de claus disponibles per nom.

To recover access to your Linux instance, you must create a new key pair to replace the lost key pair. For more information, see Connecting to Your Linux Instance If You Lose Your Private Key.

Amazon EC2 Key Pairs

4. AWS Command Line Interface

Documentació:

- AWS Command Line interface

5. Deploy Docker Containers (ECS)

Documentació:

- Deploy Docker Containers on Amazon Elastic Container Service (ECS)

6. Lightsail

Documentation: Lightsail (try free for one month)

7.
8. Title
9. Title
10. titlex

Fedora

- GetFedora

Imatges AMI per a Amazon EC2

Imatges d'Atomic Host per a Amazon Public Cloud EC2

Els enllaços de sota us proporcionaran els llistats dels Atomic Host en HVM (Hardware Virtual Machine) dels AMI disponibles per regió amb botons per llançar-los al vostre compte d'Amazon Web Services. També es proporcionen els id. dels AMI per a l'ús amb la consola o la línia d'ordres de les eines d'AWS.

Format GP2

Els AMI en format GP2 utilitzen un emmagatzematge SSD més ràpid; utilitzeu aquests AMI per a la velocitat, encara que tingueu en compte que els costos d'emmagatzematge seran més elevats que l'estàndard.

Format estàndard

Els AMI en format estàndard són més adequats si teniu dades amb poca freqüència d'accés i voleu mantenir baix el cost de l'emmagatzematge.

Imatges:

- ❏ Els AMI en HVM GP2
  EU West (London) ami-ca14f7ad
  EU West (Paris) ami-a5cc7dd8

- ❏ Els AMI en HVM estàndard
  EU West (London) ami-691af90e
  EU West (Paris) ami-fdcc7d80

Altres imatges

- Fedora cloud qcow2 642M
- Fedora cloud Raw 465M
- Imatge de contenidor de 40M
- Web fedora cloud Fedora-Container-Base-28-1.1.x86_64.tar.xz

Al get fedora es poden descarregar imatges de:

- Amazon EC2 que són directament dins el compte de Amazon. Hi ha dos tipus: GP2 oi Stàndard. Aquestes imatges també les podem posar en marxa des de Amazon seleccionant launch image i triant “Community AMIs”.
- Images per a vagrant: Vbox (642MB) i libvirt (610MB).
- Cloud: qcow2 (642MB) i Ray (463MB .xz)
- També Fedora container image de 40MB

Crear una nova instància AMI de Fedora Cloud

Des de dins de Amazon, launch instance, seleccionada la zona paris, escollir “community AMIs” de Fedora i escollir una de fedora Cloud, standard.

Llavors la t2.micro és elegible per a free tier. Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

Selecting a key-pair: Podem escollir un dels parells de claus que ja tenim, així el mateix conjunt de claus serveix per connectar a les diverses màquines.
The following instance launches have been initiated: i-0ced187dcda300e5c

Uii: en les màquines de fedora cloud el compte de usuari es diu **fedora**!!!.

Instal·lar Docker:

Seguir les instruccions per instal·lar en la màquina AMI docker. Calen els repositoris de docker-stable i de docker-edge (proporciona el docker-ce). No hi ha compleció del proompt. Cal posar-lo a part.

Fer que l’usuari fedora gestioni docker:

- Crear el grup docker
- Afegir l’usuari al grup

Instal·lar Docker Machine: **no anirà una machine de virtualbox en l’entorn Xen de EC2**

```
base=https://github.com/docker/machine/releases/download/v0.14.0 &&
  curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &&
  sudo install /tmp/docker-machine /usr/local/bin/docker-machine
```

Verificar que va:

```
# docker swarm init
# docker stack deploy -c docker-compose.part3.yml getstartedlab
```

Obrir els ports de Amazon: 80, 8080 i 900. I ja està. Connectar a la adreça pública amb un navegador als ports 80, 8080 i 9000.

Instal·lar Virtualbox:

https://www.if-not-true-then-false.com/2010/install-virtualbox-with-yum-on-fedora-centos-red-hat-rhel/

(fedora) wget http://download.virtualbox.org/virtualbox/rpm/fedora/virtualbox.repo

(centos) wget http://download.virtualbox.org/virtualbox/rpm/rhel/virtualbox.repo

Generar l'exemple de docker swarm 2 AMI en EC2

Una AMI amb 3web 1portainer 1visualizer

```
$ sudo su  -
# dnf -y upodate
# systemctl  start docker
# vim docker-compose.portainer.yml
version: "3"
services:
  web:
            image: edtasixm11/getstarted:part2
            deploy:
            replicas: 3
            restart_policy:
            condition: on-failure
            resources:
            limits:
            cpus: "0.1"
            memory: 50M
            ports:
            - "80:80"
            networks:
            - webnet
  visualizer:
            image: dockersamples/visualizer:stable
            ports:
            - "8080:8080"
            volumes:
            - "/var/run/docker.sock:/var/run/docker.sock"
            deploy:
            placement:
            constraints: [node.role == manager]
            networks:
            - webnet
  portainer:
            image: portainer/portainer
            ports:
            - "9000:9000"
            volumes:
            - "/var/run/docker.sock:/var/run/docker.sock"
            deploy:
            placement:
            constraints: [node.role == manager]
            networks:
            - webnet
networks:
  webnet:

# docker swarm init

# # docker stack deploy -c docker-compose.portainer.yml mylab
Creating network mylab_webnet
Creating service mylab_visualizer
```

```
Creating service mylab_portainer
Creating service mylab_web

# docker stack ps mylab
ID                  NAME                IMAGE                          NODE                                          DESIRED STATE
CURRENT STATE           ERROR               PORTS
hq06c9zqpg80        mylab_web.1         edtasixm11/getstarted:part2    ip-172-31-20-75.eu-west-2.compute.internal
Running             Running 21 seconds ago
zgqz72xi8hxx        mylab_portainer.1   portainer/portainer:latest     ip-172-31-20-75.eu-west-2.compute.internal
Running             Running 16 seconds ago
ykuwfteijju9        mylab_visualizer.1  dockersamples/visualizer:stable  ip-172-31-20-75.eu-west-2.compute.internal
Running             Running 9 seconds ago
kiabzf4zsr85        mylab_web.2         edtasixm11/getstarted:part2    ip-172-31-20-75.eu-west-2.compute.internal
Running             Running 21 seconds ago
9gvsjcvwbdjf        mylab_web.3         edtasixm11/getstarted:part2    ip-172-31-20-75.eu-west-2.compute.internal
Running             Running 21 seconds ago

# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                          PORTS
st5frrout878        mylab_portainer     replicated          1/1                 portainer/portainer:latest
*:9000->9000/tcp
2pccrnt0eu9g        mylab_visualizer    replicated          1/1                 dockersamples/visualizer:stable
*:8080->8080/tcp
go058nck1h2u        mylab_web           replicated          3/3                 edtasixm11/getstarted:part2    *:80->80/tcp
```

Assegurar-se d'obrir a la maquina AMI els ports d'acces: 80, 8080, 9000 , 2377
Seleccionar maquina, Security groups: launch-wizard-2, inbound: obrir els ports dels containers 80, 8080, 2377.
Atenció, obrir el port 2377 per poder comunicar els nodes entre si.

En un navegador connectar a :

```
Ipami:80    → pagina web amb comptador
Ipami:8080  → visualizer de contenidors docker
Ipami:9000  → portainer (administracio grafica docker)
```

Ports de gestió a tenir en compte:

- ❑ Nodes manager/worker Port 2377
- ❑ X
- ❑ X

En un segon host

Configurar i instal·lar docker

```
$ sudo su  -
$ dnf -y update

$ dnf -y install dnf-plugins-core
$ sudo dnf config-manager \
   --add-repo \
```

```
    https://download.docker.com/linux/fedora/docker-ce.repo
$ dnf config-manager --set-enabled docker-ce-edge

$ dnf install docker-ce
$ systemctl start docker

$ docker run hello-world
```

Node manager

```
$ docker swarm join-token worker
To add a worker to this swarm, run the following command:

        docker swarm join --token
SWMTKN-1-3dos33qzakllol8h0mvykjx1dbwk4sq3kcgh8vxuswot56uidb-c1ymwg5ssk36yb
bmvu1hzp38a 172.31.20.75:2377
```

Node worker:

```
# docker swarm join --token
SWMTKN-1-3dos33qzakllol8h0mvykjx1dbwk4sq3kcgh8vxuswot56uidb-c1ymwg5ssk36yb
bmvu1hzp38a 172.31.20.75:2377
This node joined a swarm as a worker.
```

Obrir ports en el worker:

- 80
- 8080
- 9000
- 2377 per gestio de nodes
- Port 7946 UDP for container network discovery.
- Port 7946 TCP for container network discovery.
- Port 4789 UDP for the container ingress network.

Obrir ports en manager:

- Port 7946 UDP for container network discovery.
- Port 7946 TCP for container network discovery.
- Port 4789 UDP for the container ingress network.

Ara es pot accedir a qualsevol dels elements per qualsevol de les dues IPS, sigui la màquina AMI que sigui i amb independència de si disposa del container o no.

Docker Documentation: Getting Started

Swarm

Documentació:
https://docs.docker.com/engine/swarm/

- Swarm mode overview
- Swarm mode key concepts
- *** **Get Started with Swarm mode**
- How Swarm nodes work
- Swarm task states
- Run docker in swarm mode
- Join nodes to a swarm
- Manage nodes in a swarm
- *** **Deplay services to a swarm**
- Store service configuration data
- Manage sensitive data with docker secrets
- Lock your swarm
- ** Manage swarm service networks → Use overlay networks
- Swarm administration guide
- Raft consensus in swarm mode

- Getting started with swarm mode
    - Getting started
    - Create a swarm
    - Add nodes to the swarm
    - Deploy a service
    - Inspect the service
    - Scale the service
    - Delete the service
    - Apply rolling updates
    - Drain a node
    - ** Use swarm mode routing mesh

Reference: Docker swarm
- docker swarm
- Docker swarm ca
- Docker swarm init
- Docker swarm join

- Docker swarm join-token
- Docker swarm leave
- Docker swarm unlock
- Docker swarm unlock-key
- Docker swarm leave

Command	Description
docker swarm ca	Display and rotate the root CA
docker swarm init	Initialize a swarm
docker swarm join	Join a swarm as a node and/or manager
docker swarm join-token	Manage join tokens
docker swarm leave	Leave the swarm
docker swarm unlock	Unlock swarm
docker swarm unlock-key	Manage the unlock key
docker swarm update	Update the swarm

Explore swarm mode CLI commands

- swarm init
- swarm join
- service create
- service inspect
- service ls
- service rm
- service scale
- service ps
- service update

docker, container, cluster, swarm

Docker Machine

Documentació:
https://docs.docker.com/machine/overview/

- Docker machine
- Docker machine overview
- Install machine
- Getting started with VM
- Provision hosts in the cloud
- Learn by example
  - Example: Provision Dockerized Digital Ocean Droplets
  - *** **Example: Provision Dockerized AWS EC2 Instances**
- Docker machine subcommand reference

Where to go next

- Create and run a Docker host on your local system using virtualization
- Provision multiple Docker hosts on your cloud provider
- Docker Machine driver reference
- Docker Machine subcommand reference

Where to go next

- Understand Machine concepts
- Docker Machine list of reference pages for all supported drivers
- Docker Machine driver for Oracle VirtualBox
- Docker Machine driver for Microsoft Hyper-V
- `docker-machine` command line reference

Where to go next

- Understand Machine concepts
- Provision a Docker Swarm cluster with Docker Machine

Install Docker Machine

```
base=https://github.com/docker/machine/releases/download/v0.14.0 &&
```

```
curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &&
sudo install /tmp/docker-machine /usr/local/bin/docker-machine
```

```
 docker-machine create --driver virtualbox myvm3
Running pre-create checks...
Error with pre-create check: "We support Virtualbox starting with version 5. Your
VirtualBox install is \"WARNING: The vboxdrv kernel module is not loaded. Either there is
no module\\n              available for the current kernel (4.7.9-200.fc24.x86_64) or it failed
to\\n              load. Please try load the kernel module by executing as root\\n\\n
dnf install akmod-VirtualBox kernel-devel-4.7.9-200.fc24.x86_64\\n          akmods
--kernels 4.7.9-200.fc24.x86_64 && systemctl restart systemd-modules-load.service\\n\\n
        You will not be able to start VMs until this problem is fixed.\\n5.1.26r117224\".
Please upgrade at https://www.virtualbox.org"
```

Creació de maquines virtuals VM

Per treballar amb el swarm n'hi ha prou amb una màquina però té més gràcis usar-ne unes quantes. En crearem de virtuals amb docker-machine.

Crear dues màquines basades en Virtualbox driver:

```
docker-machine create --driver virtualbox myvm1
docker-machine create --driver virtualbox myvm2
docker-machine create --driver virtualbox myvm3
```

```
# docker-machine create --driver virtualbox myvm1
Running pre-create checks...
(myvm1) Default Boot2Docker ISO is out-of-date, downloading the latest release...
(myvm1) Latest release for github.com/boot2docker/boot2docker is v18.03.1-ce
(myvm1) Downloading /root/.docker/machine/cache/boot2docker.iso from
https://github.com/boot2docker/boot2docker/releases/download/v18.03.1-ce/boot2docker.i
so...
(myvm1) 0%....10%....20%....30%....40%....50%....60%....70%....80%....90%....100%
Creating machine...
(myvm1) Copying /root/.docker/machine/cache/boot2docker.iso to
/root/.docker/machine/machines/myvm1/boot2docker.iso...
(myvm1) Creating VirtualBox VM...
(myvm1) Creating SSH key...
(myvm1) Starting the VM...
(myvm1) Check network to re-create if needed...
(myvm1) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
```

```
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
...
```

```
# docker-machine ls
NAME        ACTIVE   DRIVER      STATE          URL                       SWARM
DOCKER      ERRORS
custombox            none        Timeout
myvm1       -        virtualbox  Running        tcp://192.168.99.100:2376
v18.03.1-ce
myvm2       -        virtualbox  Running        tcp://192.168.99.101:2376
v18.03.1-ce
```

Fer del node myvm1 el manager del swarm, tot creant el swarm i declarant-se ell com a manager en fer el advertisement usant la seva adreça ip:

```
 docker-machine ssh myvm1 "docker swarm init --advertise-addr 192.168.99.100"
Swarm initialized: current node (tekibedmzdq337ukejqugq02r) is now a manager.

To add a worker to this swarm, run the following command:
        docker swarm join --token
SWMTKN-1-4umppecjmw2ojz7y5ciu5iq1kghrqvafn5t7q6mk71xy0qvvcj-d85ztgrvkjdhuefix2
ar6axby 192.168.99.100:2377
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
instructions.
```

Afegir workers / Managers

En la resposta explica com afegir nodes com a worker i com a managers:

- Worker:
  docker swarm join --token
  SWMTKN-1-4umppecjmw2ojz7y5ciu5iq1kghrqvafn5t7q6mk71xy0qvvcj-d85ztgrvkjdh
  uefix2ar6axby 192.168.99.100:2377

- Manager:
  docker swarm join-token manager

De fet es pot demanar en qualsevol moment al node manager que ens recordi què cal fer per poder afegir nodes com a workers o com a managers. Cal adreçar la ordre al node que fa de manager:

```
$ docker-machine ssh myvm1 \
        “docker swarm join-token manager / worker”
```

Activar com a worker la VM2:

```
# docker-machine ssh myvm2 "docker swarm join --token
SWMTKN-1-4umppecjmw2ojz7y5ciu5iq1kghrqvafn5t7q6mk71xy0qvvcj-d85ztgrvkjdhuefix2
ar6axby 192.168.99.100:2377"
This node joined a swarm as a worker.
```

Llistar els nodes del swarm:

```
# docker-machine ssh myvm1 "docker node ls"
ID                            HOSTNAME        STATUS          AVAILABILITY     MANAGER STATUS          ENGINE
VERSION
tekibedmzdq337ukejqugq02r *   myvm1           Ready           Active           Leader                  18.03.1-ce
tj4mtswigcpm1f8lij0epftk6     myvm2           Ready           Active                                   18.03.1-ce

[root@hostedt ~]# docker-machine ssh myvm2 "docker node ls"
Error response from daemon: This node is not a swarm manager. Worker nodes can't be
used to view or modify cluster state. Please run this command on a manager node or
promote the current node to a manager.
exit status 1
```

Realitzar comandes a les VM

Hi ha dos mecanismes:

- Usant docker-machine ssh <nomvm> “comanda-a-fer”. En aquest mode el filesystem és el del node, s’hi poden fer les ordres que fariem com si estem connectats per ssh.
- Modificar l’entorn perquè les comandes docker que realitzem a la consola es realitzin en realitat a la VM. En aquest mode el filesystem és el del host, podem usar els fitxers compose .yml del host, i les ordres docker que fem s’executen en la VM.

Comandes via docker-machine ssh:

```
$ docker-machine ssh myvm1 “docker node ls”
ID                            HOSTNAME        STATUS          AVAILABILITY     MANAGER STATUS          ENGINE
VERSION
tekibedmzdq337ukejqugq02r *   myvm1           Ready           Active           Leader                  18.03.1-ce

# docker-machine ssh myvm1 "uname -a"
Linux myvm1 4.9.93-boot2docker #1 SMP Thu Apr 12 17:05:48 UTC 2018 x86_64
GNU/Linux
```

Configurar la consola per a fer comandes directes a la VM:

```
$ docker-machine env myvm1
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/root/.docker/machine/machines/myvm1"
export DOCKER_MACHINE_NAME="myvm1"
# Run this command to configure your shell:
# eval $(docker-machine env myvm1)
```

```
$ eval (docker-machine env myvm1)

# aquesta comanda la fa directament en el node myvm1
$ docker node ls

# desconnectar l'entorn per retornar al host l'entorn de comandes:
$ eval (docker-machine eval -u)
```

Exemples docker machines on the cloud

DigitalOcean

```
$ docker-machine create --driver digitalocean
--digitalocean-access-token xxxxx docker-sandbox
```

For a step-by-step guide on using Machine to create Docker hosts on Digital Ocean, see the Digital Ocean Example.

AWS

```
$ docker-machine create --driver amazonec2 --amazonec2-access-key
AKI******* --amazonec2-secret-key 8T93C*******  aws-sandbox
```

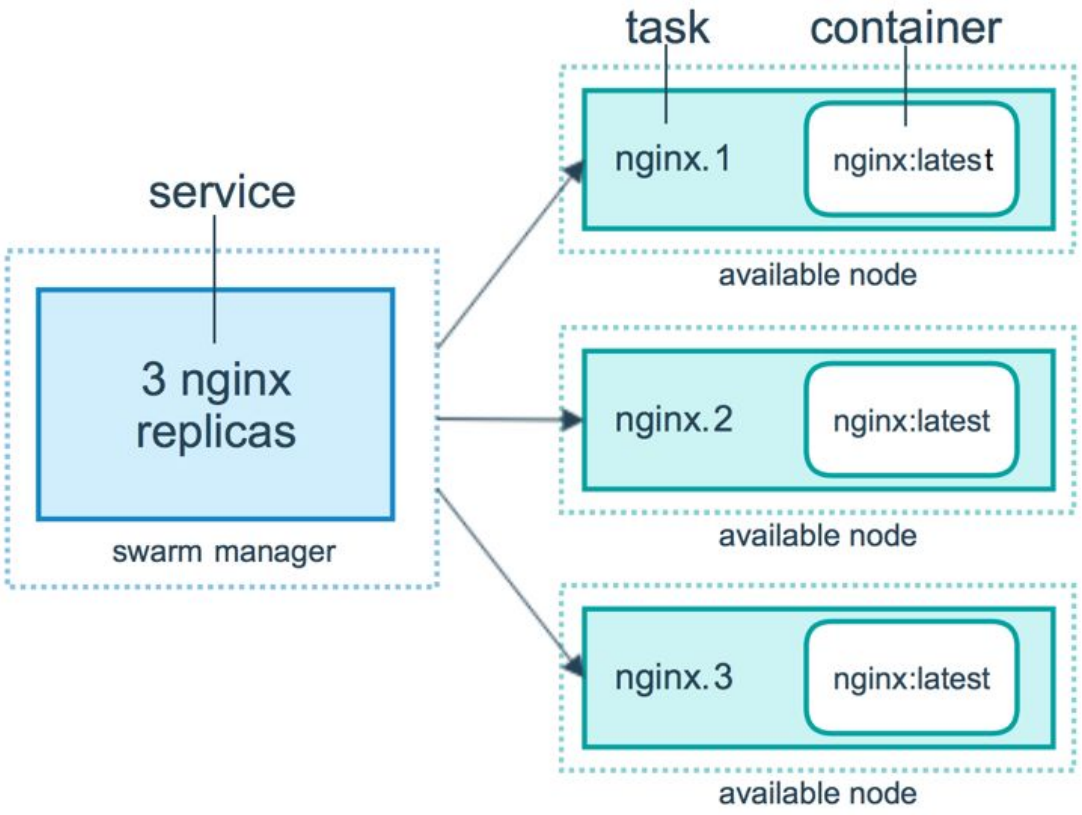
For a step-by-step guide on using Machine to create Dockerized AWS instances, see the Amazon Web Services (AWS) example.

Host

```
$ docker-machine create --driver none --url=tcp://50.134.234.20:2376
custombox
```

Service

For example, imagine you want to load balance between three instances of an HTTP listener. The diagram below shows an HTTP listener service with three replicas. Each of the three instances of the listener is a task in the swarm.



Cloud

Try out Docker Cloud!

We suggest using Docker Cloud as the most up-to-date way to run Docker on your cloud providers. To get started, see Docker Cloud docs home page, Docker Cloud Settings and Docker ID, and Swarms in Docker Cloud (Beta). If you are running Edge channel Docker for Mac or Windows, you can access your Docker Cloud account from those Docker desktop applications. See Docker Cloud (Edge feature) on Mac or Windows.

Docker Machine still works as described here, but Docker Cloud supercedes Machine for this purpose

Poti-poti

Portainer

Portainer és un container preparat per monitoritzar el daemon de docker del host. És un visor gràfic que permet veure tot el que fa el host, quines imatges te, quines networks, volumes, containers, etc.

Documentació:
https://medium.com/lucjuggery/about-var-run-docker-sock-3bfd276e12fd

```
# docker container run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock
portainer/portainer

# firefox → localhost:9000

# el primer cop cal crear usuari (admin) u un passwd de 8 caràcters (adminadmin)
```

A /var/run/docker.sock és un socket unix on escolta el daemon del docker.

```
# docker container ls
CONTAINER ID        IMAGE                  COMMAND             CREATED
STATUS              PORTS                  NAMES
e624b3770db0        portainer/portainer   "/portainer"        10 seconds ago        Up 7
seconds         0.0.0.0:9000->9000/tcp   suspicious_leavitt

# ls -la /var/run/docker*
-rw-r--r--. 1 root root    5  5 mai 13:16 /var/run/docker.pid
srw-rw----. 1 root docker   0  5 mai 13:16 /var/run/docker.sock
/var/run/docker:
total 0
drwx------.  7 root root  160  5 mai 13:17 .
drwxr-xr-x. 54 root root 1500  5 mai 13:16 ..
drwx------.  5 root root  160  5 mai 13:17 libcontainerd
drw-------.  2 root root   60  5 mai 13:16 libnetwork
srwxr-xr-x.  1 root root0  5 mai 13:16 metrics.sock
drwxr-xr-x.  2 root root   60  5 mai 13:17 netns
drwx------.  3 root root   60  5 mai 13:16 plugins
drwx------.  2 root root   40  5 mai 13:16 swarm

# file /var/run/docker.sock
/var/run/docker.sock: socket
```

Docker Daemon API

Es pot contactar i manar ordres HTTP per gestionar docker (per exemple amb Curl) via la seva API.:
“””All the HTTP endpoints defined in the Docker engine API v1.27 (last version to date), can thus be consumed through this unix socket.”””

Aií es poden crear + fer start de containers com aquest:

```
$ docker pull nginx
$ curl -XPOST --unix-socket /var/run/docker.sock -d '{"Image":"nginx"}' -H 'Content-Type: application/json' http://localhost/containers/create
```

```
$ curl -XPOST --unix-socket /var/run/docker.sock
http://localhost/containers/fcb6...7d65/start
```

Docker offline documentation

https://docs.docker.com/docsarchive/

Docs for v18.03 ***(current)*** are accessible at **https://docs.docker.com/**, or to view the docs offline on your local machine, run:
$ docker run -ti -p 4000:4000 docs/docker.github.io:latest

Docs for v1.12 are accessible at **https://docs.docker.com/v1.12/**, or to view the docs offline on your local machine, run:
docker run -ti -p 4000:4000 docs/docker.github.io:v1.12

Recursos

<descripció dels recursos d’imatges usats en el document i les pràctiques><pendent>

Pràctiques

0. Implementar l'exemple de Getting Started

1. Implementar un container bàsic com a servei d'un swarm de 5 nodes (1 manager i 4 workers).
    a. Crear tres rèpliques del servei.
    b. Provar amb altres serveis com: helloword, nginx, ldapserver, postgres, hostservices, etc.
    c. Ampliar-ho posant-hi un visualizer.

2. Ídem que exercici 1 però en mode global on a cada node hi corri una instància del container.

3. Pensar un model variat amb vàries rèpliques i serveis d'un sol node, per exemple:
    a. Ldapserver producer + varis ldapservers consumers.
    b. Ldap + kerberos + varis hosts sshserver

4. Observar que en un swarm en posar active un node no accepta noves tasques. En posar-lo drain se li eliminen, però continua contestat a les peticions via routing mesh.

Pràctiques en documents per als alumnes:

Practica01_containers_detach
Practica02_automated_build
Practica03_port_network
Practica04_version
Practica05_service
Practica06_volumes
Practica07_portainer
Practica08_documentation

Pràctica: Desplegament Swarm local + AWS del comptador de visites de Getting Started Original

Descripció:

Desplegar l'exemple de *Get Started Original* usant una/dues màquines locals i una/dues màquines a AWS. Es tracta del desplegament de containers web i redis per al funcionament d'una pàgina web amb comptador de visites. També hi afagirem un Portainer i un Visualizer.

Caldrà configurar les màquines AWS per que disposin de Docker i obrir els ports necessaris per permetre la connectivitat de docker Swarm i d'accés a la aplicació.

PAS 1: Disposar de Docker Engine en tots els hosts.

Instal·lar docker a tots els hosts seguint les indicacions de Docker Documentation Get Docker.

PAS 2: en els AMI AWS EC2 Obrir els ports

També cal assegurar-se que la comunicació per xarxa entre els hosts és permesa i en especial la utilització dels ports:

- TCP port **2377** for cluster management communications
- TCP and UDP port **7946** for communication among nodes
- UDP port **4789** for overlay network traffic

PAS 3: Crear el swarm i afegir-hi els nodes locals

En el manager

```
# systemctl  start docker

docker swarm init
Swarm initialized: current node (432axtays99osjals9wehemvg) is now a manager.
To add a worker to this swarm, run the following command:
        docker swarm join --token
SWMTKN-1-66s2di3axj5ahuiwgkf8qya20wzsl3v1508t3x6lm2bkg0xr4r-5ruquxvdr5t4pds0iwfoppafs 192.168.1.50:2377
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

En els workers locals

```
docker swarm join --token
SWMTKN-1-66s2di3axj5ahuiwgkf8qya20wzsl3v1508t3x6lm2bkg0xr4r-5ruquxvdr5t4pds0iwfoppafs 192.168.1.50:2377
```

PAS 4: Afegir els nodes remots de AWS EC2

Identificar l'adreça IP pública del manager

```
Per exemple des del host AMI AWS EC2
[root@ip-172-31-31-118 fedora]# netstat  -tn
Active Internet connections (w/o servers)
```

```
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0    308 172.31.31.118:22        79.155.7.25:37140       ESTABLISHED
```

En el worker AMI AWS EC2

```
docker swarm join --token
SWMTKN-1-66s2di3axj5ahuiwgkf8qya20wzsl3v1508t3x6lm2bkg0xr4r-5ruquxvdr5t4pds0iwfoppafs 79.155.7.25:2377
```

El node s'afegirà sempre i quant pugui accedir al manager a través de la seva adreça pública a través del port 2377. El problema és que en el manager el port està obert però en el router (de casa o de l'escola) no. Segurament NO es podrà unir al searm apropiadament.

Atenció: genera un missatge d'error indicant que no s'ha pogut connectar i que ho provarà en background. Però considera que forma part del swarm, tot i que no hi té connexió. Es millor eliminar-lo del swarm de moment amb:

```
docker swarm leave
```

Solucions:

- ❏ Configurar el router del costat de manager per obrir el port 2377
- ❏ Fer 'trampetes' i obrir un tunnel reverse que comuniqui la AMI de AWS EC2 amb el manager.

Truc: Obrir un tunnel reverse

En el manager obrirem un túnel ssh que deixarà obert el port 5001 en la AMI que connecta directament al port 2377 del manager. A la AMI la enganyarem dient-li que l'advertisement (el node manager) el trobarà connectant al localhost al port 5001.

En el manager

```
ssh -i .ssh/MyKeyPairWin.pem  -R  5001:localhost:2377  fedora@35.177.147.72

sleep 123456789& (o opció ssh per mantenir viva la conexió)
```

En el worker AMI

```
[root@ip-172-31-31-118 fedora]# docker swarm join --token
SWMTKN-1-66s2di3axj5ahuiwgkf8qya20wzsl3v1508t3x6lm2bkg0xr4r-5ruquxvdr5t4pds0i
wfoppafs localhost:5001
This node joined a swarm as a worker.
```

En el manager

```
 docker node ls
ID                            HOSTNAME              STATUS              AVAILABILITY         MANAGER STATUS      ENGINE VERSION
rieymhj83joe                  asus                  Ready               Active                                   18.09.0
h0e98c7uejy                   ip-172-31-31-11
8.eu-west-2.compute.internal  Ready                 Active                                   19.03.4
432axtays9 *                  lenovo                Ready               Active               Leader              18.09.0
```

Pas 5: Engegar el desplegament

En el manager

```
docker stack deploy -c docker-compose.yml myapp
```

Verificar conectant al qualsevol dels nodes als ports 80 (pàgina web amb comptador) 8080 (portainer) i 9000 (visualizer).

Fer drain d'un dels nodes i observar que es reestructuren els serveis.

En el manager:

```
docker node update --availability drain asus
```

```
docker node update --availability active asus
```

Pas 6: Aturar-ho tot

Aturar el desplegament, en el manager:

```
docker stack deploy -c docker-compose.yml myapp
```

Eliminar el Swarm workers / manager

```
docker swarm leave
```

```
docker swarm leave --force
```