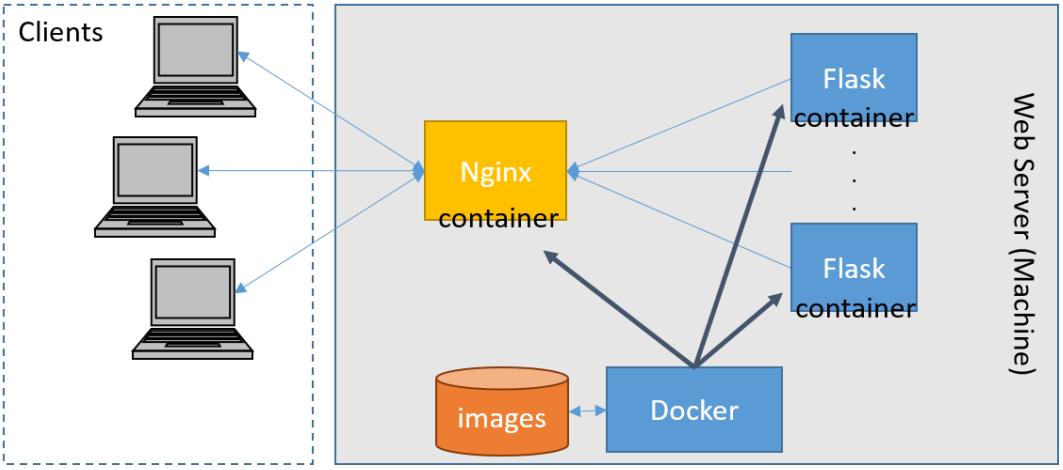
# Contenidors

Data prevista: 12 de Maig de 2021 – 21 de Maig de 2021

El fet de poder encapsular serveis en “contenidors” ens permet crear entorns d’execució i serveis llestos per ser desplegats en gairebé qualsevol entorn, reduint al màxim les dependències i software a instal·lar, ja que els components i serveis queden “auto-continguts” dins el contenidor. Al tenir cada component de l’entorn d’execució en un contenidor ens permet també escalar serveis amb facilitat: si cada contenidor pot sostenir certa càrrega, podem desplegar més o menys contenidors al moment per tal d’atendre clients sense gastar recursos de més o de menys. Aquesta sessió estarà centrada en el desplegament i experimentació amb contenidors, usant l’hypervisor de contenidors com a “hands-on” de laboratori.

## Treball de laboratori

Docker és un hypervisor encarregat de construir imatges de contenidor i d’executar instàncies d’aquestes imatges. En aquest laboratori prepararem un entorn amb una sèrie de contenidors fent de servei web (ex. Flask), i un contenidor que faci de Load Balancer (e.g. Nginx).




En aquest laboratori treballem amb **Ubuntu 18.04 LTS (64bits)** on instal·larem Docker, veurem serveis web i d’aplicacions com Nginx i Flask, i veurem els processos de sistema que tenen relació amb la containerització gràcies a les eines de Unix/Linux.

---

*Nota important: Compte amb fer “copy & paste”, ja que hi ha símbols com comes o guions que es poden copiar malament d’aquest document a consola. Si no us va una comanda, escriviu-la directament!*

---

## Instruccions de la pràctica

Aquesta sessió de laboratori la farem usant Màquines Virtuals, per tal de tenir tots el mateix entorn independentment de on fem la pràctica. Tot i que ja heu fet la sessió de Màquines Virtuals, al primer punt de la sessió trobareu les instruccions per muntar VMs amb **Ubuntu** i fer la pràctica dins la VM, a més de que tots puguem treballar amb el mateix entorn i instal·lar correctament Docker, i poder fer "ps", "ifconfig" i altres comandes Unix.

## Treball Previ

Mireu-vos el **vídeo d'introducció** que acompanya la pràctica. Aquí trobareu el context de la pràctica, i refrescareu conceptes vistos a classe.

## Grups de Treball

Per a aquesta pràctica ens distribuïm en parelles (**grups de 2 persones**). Una recomanació és que us connecteu mitjançant alguna plataforma que us permeti fer vídeo-conferència alhora que compartir la pantalla, de tal forma que una persona del grup faci el rol d'executor i comparteixi la pantalla. L'altre membre del grup pot fer de secretari, i anar anotant les respostes del qüestionari que cal anar responent.

## Qüestionari de Laboratori

Durant la pràctica trobarem preguntes que cal anar contestant a mesura que avanceu per la pràctica. Aquestes formen el **qüestionari de laboratori** que cal omplir i entregar al final de la sessió. El qüestionari **el pujarem pel Racó**, on veureu una pràctica oberta per a "Contenidors".

**ENTREGA:** Les respostes estaran en un document per grup indicant les preguntes amb les corresponents respostes (com si féssiu una memòria). Assegureu-vos de justificar bé les respostes, i suposeu al voltant d'una línia o dos per resposta, tampoc més. Donat que el Racó només permet entregues individuals, el "secretari" del grup pujarà la pràctica → Recordeu de posar els noms de tots els integrants del grup al document.

## Temps de Dedicació

Aquesta pràctica ha estat dissenyada per a que la pugueu realitzar en **menys de dues hores**, més uns quinze minuts per veure el vídeo d'introducció, i per poder passar a net el qüestionari i pujar-ho al Racó. Hem tingut en compte el que es triga en descarregar el software i les imatges de Docker en una connexió a casa, així que no heu de patir per no tenir una connexió d'alta velocitat.

## Períodes de Realització

Per a aquesta pràctica tindrem sessió síncrona el **12 de Maig**, en la que haureu de venir amb el vídeo d'introducció vist. Entre el 25 i el 2 de Desembre tindreu temps de fer la pràctica. El dia **19 de Maig** farem mitja sessió síncrona per resoldre dubtes i comentaris (l'altre mitja la dedicarem al següent laboratori), i tindreu dos dies més (fins el **21 de Maig**) per polir-la i entregar-la al Racó.

La pràctica estarà oberta **del 12 al 21 de Maig a les 23:59**. Eviteu entregar al darrer moment, per evitar possibles problemes tècnics, i per assegurar-vos de que heu pujat la pràctica correctament.

## Avaluació

Donat que no podem avaluar l'assistència ni participació a classe, l'avaluació tindrà en compte el **qüestionari de laboratori**, usant la següent fórmula:

Nota = (Qüestionari de Laboratori) → 10 * (Un punt per resposta correcta/total de preguntes)

## 0. Preparació de l'entorn de la Màquina Virtual

Ja hauríeu d'haver instal·lat i usat VirtualBox i Vagrant a la sessió anterior. Usarem aquestes tecnologies per a descarregar i preparar l'entorn per a aquesta sessió.

### Preparant VM Ubuntu

Primer, descarreguem una VM amb Ubuntu pre-instal·lat. Les següents comandes funcionen tant per Linux, MacOs com Windows, des de la consola de comandes:

```
mkdir vm_cpd3                    # Create a working directory
cd vm_cpd3                       # Enter the directory
vagrant init ubuntu/bionic64     # Download the "VM descriptor"
```

Ara, hem de modificar el descriptor de la VM, per donar-li més RAM, obrir ports per tenir-los disponibles i accedir, i donar-li un nom a la VM. Obrim el fitxer "Vagrantfile", i afegim les següents línies entre els tags de "config":

```
 config.vm.network "forwarded_port", guest: 8080, host: 8080
 config.vm.network "forwarded_port", guest: 5000, host: 5000

 config.vm.provider "virtualbox" do |vb|
       vb.memory = "4096"
       vb.name = "docker"
 end
```

Ara, la nostra VM està llesta per ser descarregada (el primer cop) i per iniciar-se:

```
vagrant up              # Descarrega (1er cop) i Inicia la VM
```

### Entrant a la VM

Ara podem entrar a la VM usant SSH directament:

```
vagrant ssh             # Obrir una sessió de Secure-Shell
```

Ara entrarem a la màquina virtual, llesta per a fer la sessió.

Recordeu: Per sortir només hem de fer "exit", i per aturar la VM podem fer "vagrant halt" des de fora. Per iniciar-la de nou, només ens cal fer "vagrant up". Si volem esborra la VM, podem fer "vagrant destroy", i fent "vagrant up" després tornarem a descarregar la imatge nova.

A partir d'aquí, ho farem tot **DINS LA VM**.

# 1. Docker i Contenidors

## Instal·lar Docker

Per si un cas la VM porta una versió de Docker preinstal·lada, primer cal eliminar-les per assegurar-nos de que instal·lem la que ens interessa per aquesta sessió, i després instal·lem les dependències i Docker-CE (Community Edition):

```
sudo apt-get -y remove docker docker-engine
sudo apt-get -y install apt-transport-https ca-certificates \
  curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo \
  apt-key add -
sudo add-apt-repository "deb [arch=amd64] \
  https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get -y  install docker-ce
```

Ara comprovem que tot s'ha instal·lat correctament:

```
sudo docker info
```

## Construir Imatges Docker

Per defecte Docker té un seguit d'imatges pre-dissenyades al repositori on-line de Docker. Un exemple és que podem descarregar un contenidor amb la darrera versió d'Ubuntu, fent:

```
sudo docker pull ubuntu
```

Ara, quan llistem les imatges de Docker que tenim disponibles al sistema, podem veure que "ubuntu" està preparada:

```
sudo docker images
```

I a partir d'ara podem instanciar un contenidor  a partir d'aquesta imatge:

```
sudo docker run –it ubuntu
```

Mentre que acabem d'obrir una sessió de terminal interactiu ("-it"), des d'un altre terminal, podem fer llistar els contenidors que tenim en marxa:

```
sudo docker container ls
```

Si fem "exit", sortim del contenidor, i veurem que ja no apareix a la llista de contenidors en marxa. Addicionalment, podem veure els contenidors creats i el seu estat (en marxa, finalitzats, ...) usant:

```
sudo docker ps –a
```

I d'aquesta manera, si hi ha cap problema, podem fer un "sudo docker stop <ID>", indicant la ID del contenidor que volem aturar.

**PREGUNTES**: Tenint en marxa un contenidor d'Ubuntu, obrir un segon terminal (si esteu dins una VM, haureu d'obrir un segon terminal DINS la VM: podeu fer diferents sessions de "vagrant ssh", p.e.)

a) Executeu dins el contenidor i fora del contenidor les comandes "ps -fA", "w", "who", "ls /home". Quines diferències ens trobem amb cada comanda i per què?

## Creant una Imatge Pròpia

Ara crearem una imatge pròpia de Docker. Anem a $HOME ("cd $HOME") i creem un directori on construirem la imatge:

```
cd $HOME
mkdir docker_test
cd docker_test
```

I amb un editor (vim, nano, ...) creem el següent fitxer anomenat "Dockerfile":

### $HOME/docker_test/Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python3-pip iputils-ping && pip3 install Flask
RUN echo "hello CPD" > /cpd.txt
```

**PREGUNTES**: Examinant les comandes de cada línia del Dockerfile

b) Què fan cadascuna (de les comandes pròpies de Docker)?
c) Quan s'executaran aquestes comandes?

Podeu trobar el manual de Docker build a https://docs.docker.com/engine/reference/builder

Ara, dins el directori on es troba el Dockerfile, ordenem a Docker construït la imatge (alerta "." al final):

```
sudo docker build --rm -t cpd_test:v1 .
```

Veurem si s'ha construït correctament si llistem les imatges de Docker altre cop. Aquí podem veure també el que ocupa cada imatge construïda.

**PREGUNTES**: Després de construir la imatge:

d) Quan ha trigat a construir-se la imatge i per què?
e) Quan ocupa respecte la imatge de Ubuntu? Haurien d'ocupar el mateix i per què?

Finalment, podem iniciar instàncies de la imatge usant la comanda de Docker "run" (i la farem interactiva). Si docker es queixa de que no troba "cpd_test:latest" podem indicar-li la versió "cpd_test:v1".

**PREGUNTES**: Un cop iniciem una instància

f) Quins "flags" hem hagut d'usar per iniciar una sessió interactiva?
g) Què trobem al directori arrel del contenidor?

## 2. Creant una Imatge de Flask

### Creació de la imatge

Dins del directori de "docker_test", crearem un directori "web", on posarem una aplicació web de Flask en python anomenada "webserver.py":

$HOME/docker_test/web/webserver.py

```
from flask import Flask
import socket

app = Flask(__name__)

@app.route("/")
def hello():
     return "Hello World from %s!\n" % (socket.gethostname())

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

Aquesta aplicació es posarà a escoltar un port (per defecte a Flask el 5000), i imprimirà un "Hola Món" al ser preguntat al directori arrel "/".

Ara, obrim una instància del contenidor "cpd_test:v1", indicant el següent:

```
sudo docker run -it --volume=$HOME/docker_test/web:/web cpd_test:v1
```

Estem dins de la instància del contenidor. Podem veure que se'ns ha muntat el directori "web" dins el contenidor, i ara podem executar dins:

```
python3 /web/webserver.py
```

Per poder saber quina IP tenim dins el contenidor, podem consultar el fitxer "/etc/hosts" on veurem l'adreça assignada al nostre contenidor. Des de fora, en un altre terminal, podem accedir fent un "curl" o si usem un navegador que tingui accés (p.e. "lynx"):

```
curl http://<ADREÇA-IP>:5000/
```

Si tot ha anat bé, hauríem de veure el Hello World, amb la ID del contenidor que ens respon.

Per aturar el container, podem sortir ("exit") si tenim una sessió interactiva, o fer un "docker stop <ID>", o podem eliminar tots els contenidors actius usant "docker rm $(docker ps -a -q)" si tenim prou permisos de *root*. Ex.: "sudo docker rm $(sudo docker ps -a -q)"

### Creació d'una Imatge Flask pròpia

Ara podem decidir afegir automàticament la aplicació a una imatge, creant una imatge derivada i indicant al Dockerfile que volem incloure el fitxer de la aplicació Flask. Per això creem un nou directori "$HOME/docker_test2", i copiem el directori "web" dins.

```
mkdir -p $HOME/docker_test2
```

```
cp -R $HOME/docker_test/web $HOME/docker_test2/
cd $HOME/docker_test2/
```

I ara toca crear el Dockerfile corresponent:

$HOME/docker_test2/Dockerfile

```
FROM cpd_test:v1
ADD web /web
CMD python3 /web/webserver.py
```

**PREGUNTES**: Examinant les noves comandes de Dockerfile

h) Què fan cadascuna (de les comandes pròpies de Docker)?
i) Què passarà automàticament quan s'instanciï un contenidor d'aquesta nova imatge?

Construïm la imatge, i ens assegurem que apareix al respositori local d'imatges Docker:

```
sudo docker build --rm -t cpd_test:v2 .
sudo docker images
```

Hauríem de poder veure la nova versió de "cpd_test" a la llista d'imatges disponibles. Ara, podem crear una instància, i com que aquesta nova versió és un "servei" (s'executarà el webserver al iniciar-se el contenidor), el podem executar de forma "no interactiva" i "de fons":

```
sudo docker run -td --name flask_cpd cpd_test:v2
```

Si observem els contenidors en marxa amb "sudo docker ps", veurem que "flask_cpd" s'està executant. A més, podem accedir als "logs" del contenidor usant "sudo docker logs flask_cpd", així com obrir una sessió "bash" amb el contenidor en marxa amb "sudo docker exec -it flask_cpd bash", com inspeccionar el contenidor amb "sudo docker inspect flask_cpd" i trobar la seva IP. Un cop tenim la IP, podem fer altre cop:

```
curl http://<ADREÇA-IP>:5000/
```

I si tot ha anat bé, tornarem a veure el "Hello World" amb la ID del contenidor.

Finalment, podem matar el contenidor amb "sudo docker kill flask_cpd" i "sudo docker rm flask_cpd".

## Enllaçant Contenidors

Per tal de no estar buscant les IPs dels contenidors manualment, en cas de que els vulguem connectar, podem indicar a Docker que els enllaci automàticament. Per això llançarem dos instàncies de la mateixa imatge:

```
sudo docker run -td --name flask_cpd1 cpd_test:v2
sudo docker run -td --name flask_cpd2 cpd_test:v2
sudo docker ps
```

Veurem els dos contenidors en marxa. Ara crearem una instància de “cpd_test:v1” anomenada “shell” que tingui enllaçats “flask_cpd1” (amb alies “flask1”) i “flask_cpd2” (amb alies “flask2”), i engegarem una consola interactiva:

```
sudo docker run -it --name shell --link flask_cpd1:flask1 \
 --link flask_cpd2:flask2 cpd_test:v1
```

**PREGUNTES**: Un cop dins de la instància “shell":

j) Què ens apareix de rellevant si fem un “cat /etc/hosts”?

Si fem “ping” a “flask1” i “flask2”, veurem que els dos contenidors ens responen. Sortim de “shell" i l’eliminem fent “sudo docker rm shell”.

**PREGUNTES**: Un cop dins de la instància “shell":

k) Segueixen vius els contenidors flask? Per què?

En cas de que tinguem contenidors corrents, els matarem a tots amb “docker kill <ID/NOM>” i “docker rm <ID/NOM>”.

## 3. Creant un Balancejador de Càrrega amb Nginx

El darrer apartat de la pràctica serà crear una imatge de contenidor que contingui el servei web/aplicacions Nginx, que ens farà de porta d’accés i balancejador de càrrega dels contenidors amb Flask. Primer de tot, crearem un directori a “$HOME/docker_test3”, prepararem els fitxers que requereix Nginx com a configuració (“nginx/default”), i prepararem el “Dockerfile” corresponent per a que 1) creï la imatge, 2) afegeixi la configuració d’Nginx, i 3) executi Nginx al iniciar-se:

```
mkdir -p $HOME/docker_test3/nginx
cd $HOME/docker_test3/
```

### $HOME/docker_test3/nginx/default

```
upstream myapp1 {
  server flask1:5000;
  server flask2:5000;
}
server {
  listen 8080;
  location / {
    proxy_pass http://myapp1;
  }
}
```

Aquesta configuració de Nginx farà un “Round-Robin” de les peticions que arribin. Nginx anirà repartint les peticions que rebi al port 8080, cap als dos Flask al seu corresponent port 5000.

### $HOME/docker_test3/Dockerfile

```
FROM cpd_test:v1
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y nginx
```

```
ADD nginx/default /etc/nginx/sites-available/default
CMD nginx -g 'daemon off;'
```

(Nota: La variable "DEBIAN_FRONTEND" l'hem de posar per evitar que certs paquets ens demanin coses per consola a l'hora de fer "apt-get" i el "build" es quedi esperant)

**PREGUNTES**: Al crear el Dockerfile:

l) Per què creem la imatge d'Nginx a partir de "cpd_test:v1" i no necessàriament "cpd_test:v2"?

Finalment, construïm la imatge de Nginx, i comprovem que està disponible al repositori:

```
sudo docker build --rm -t cpd_test:v3 .
sudo docker images
```

I ara, cal iniciar les instàncies de Flask (cpd_test:v2), connectar-les usant Nginx (cpd_test:v3) i provar-les:

```
sudo docker run -td --name flask_cpd1 cpd_test:v2
sudo docker run -td --name flask_cpd2 cpd_test:v2
sudo docker run -itd --name nginx --link flask_cpd1:flask1 \
 --link flask_cpd2:flask2 cpd_test:v3
sudo docker ps
```

**NOTA**: En cas de que surti alguna cosa malament, si el contenidor està en marxa, cal aturar-ho amb un "kill" i després "rm". Si el contenidor no s'ha arribat a engegar, amb un "rm" hi ha prou. Podem veure TOTS els contenidors (fins i tot els fallits) usant "sudo docker ps -a".

Ara veiem que tenim els 3 contenidors funcionant, i podem fer peticions a Nginx, que les reenviarà als diferents Flask: (podem inspeccionar el contenidor Nginx per descobrir la seva IP)

```
curl http://<ADREÇA-IP-NGINX>:8080/
```

Si fem repetides peticions a Nginx, veurem que l'ID del Flask que respon és diferent! Per acabar, podem tancar els contenidors (fent "kill" i "rm").

## Final de la Pràctica

**Abans d'acabar**: Recordeu entregar el qüestionari al final de la sessió! Responeu les PREGUNTES de forma breu (al voltant d'una línia o dues per resposta), i **entregueu el qüestionari** conforme heu anat fent la sessió.