



Prácticas Cloud. Contenedores Docker

Máster Universitario en Computación en la Nube y de Altas Prestaciones



- Utilizar las opciones básicas para **Gestionar Contenedores** mediante la tecnología **Docker**.
- Probar y testear dichas opciones a través de un host local.



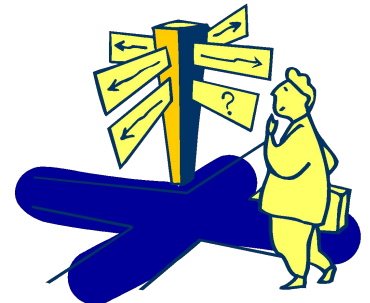
P1. Instalar Docker-CE (Community Edition)

P2. Comandos Docker

P3. Tu primer Contenedor

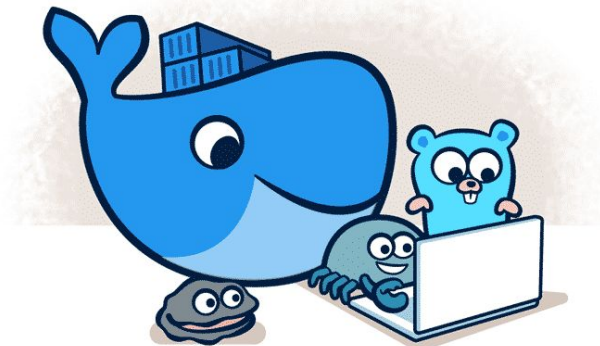
P4. Contenedor Servidor Web

P5. Crear un trabajo Batch (Octave)



P1. Instalar Docker-CE

- En esta práctica desplegarás una máquina virtual en Microsoft Azure sobre la que trabajar.
- Sobre la Máquina Virtual instalarás **Docker-CE (Community Edition)** para la gestión de contenedores.
- **Docker CE** (Community Edition) se trata de un entorno de desarrollo y ejecución de contenedores que es gratuito.
- Docker-CE te permitirá crear, testear, depurar, empaquetar y desplegar contenedores con la tecnología **Docker**.



P1. Instalar Docker-CE

Desplegando una MV

- Despliega una Máquina Virtual en Windows Azure con las siguientes características:

Grupo de recursos: Crea un grupo con nombre "gr<usuario_upvnet>" (Ej. **gramcaar**) o reutiliza el que ya tengas

Nombre Máquina Virtual: MVdocker-<usuario_upvnet> (Ej. **MVdocker-amcaar**)

Región: Norte de Europa (selecciona otra en caso de error por límite de cuotas, p.ej. *West Europe*)

Opciones de disponibilidad: No se requiere redundancia de la infraestructura

Tipo de seguridad: Estándar

Imagen: Ubuntu Server 22.04 LTS - x64 gen. 2

Instancia de Azure spot: No

Tamaño: Standard_B2s

Tipo de seguridad: Estándar

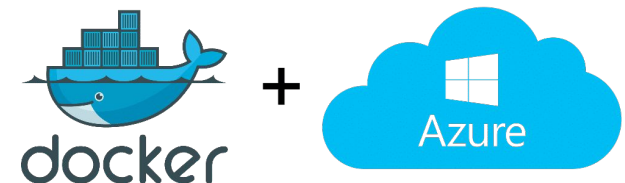
Usuario y contraseña o claves SSH (**más seguro**)

Permitir puerto de entrada ssh

Tipo de Disco: HDD

Habilita "Eliminar IP pública y NIC cuando se elimine la VM" en el apartado de "Redes".

Deshabilita cualquier opción de supervisión.



P1. Instalar Docker-CE

Pasos para instalar Docker Sobre Ubuntu

- Primero actualiza los paquetes existentes de tu MV mediante la herramienta ***apt-get***:

```
$ sudo apt-get update
```

- Para instalar Docker-CE es necesario instalar algunos paquetes como requisito previo, que permitan a APT usar paquetes a través de HTTPS:

```
$ sudo apt-get install ca-certificates curl
```

- Para instalar el repositorio oficial de Docker se debe de instalar la clave de GPG en la MV:

```
$ sudo install -m 0755 -d /etc/apt/keyrings
```

```
$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o  
/etc/apt/keyrings/docker.asc
```

```
$ sudo chmod a+r /etc/apt/keyrings/docker.asc
```

Nota: también puedes seguir la guía oficial de instalación: <https://docs.docker.com/engine/install/>

P1. Instalar Docker-CE

Pasos para instalar Docker Sobre ubuntu

- Agrega el repositorio de Docker a las fuentes de **APT**:

```
$ echo \  
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]  
  https://download.docker.com/linux/ubuntu \  
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \  
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

- Actualiza la base de datos de los paquetes de Docker en el repositorio recién agregado:

```
$ sudo apt-get update
```

P1. Instalar Docker-CE

Pasos para instalar Docker Sobre ubuntu



- Instala el Docker-CE en la MV:

```
$ sudo apt-get install docker-ce docker-ce-cli
```

- Testea que se ha instalado correctamente y que el Servicio está *active* y en ejecución, listo para utilizarse.

```
$ sudo systemctl status docker
```

```
amcaar@MVdocker-amcaar:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2024-10-20 20:00:33 UTC; 28min ago
 TriggeredBy: ● docker.socket
     Docs: https://docs.docker.com
    Main PID: 3307 (dockerd)
       Tasks: 9
      Memory: 26.4M
         CPU: 440ms
    CGroup: /system.slice/docker.service
            └─3307 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```


P1. Instalar Docker-CE

Pasos para instalar Docker Sobre ubuntu

- OPCIONALMENTE puedes probar a ejecutar un contenedor de prueba, donde se descargará una nueva imagen de contenedor, se ejecutará un contenedor docker y se imprimirá un mensaje de bienvenida:

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

```
amcaar@MVdocker-amcaar:~$ sudo docker run hello-world

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

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

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

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

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

P1. Instalar Docker-CE

- Aunque este último paso no sea necesario, es conveniente por practicidad para evitar utilizar ***sudo*** en todos los comandos docker.
- Se debe de agregar el usuario de la MV al grupo docker (creado en el proceso de instalación). Posteriormente, tendremos que reestablecer el Servicio.

```
$ sudo usermod -aG docker <tu_usuario>
```

- Comprueba que tu usuario de la MV se ha agregado al grupo docker

```
$ id -nG
```

- Reestablece el Servicio docker y la sesión de usuario.

```
$ sudo service docker restart
```

- Realiza un *Logout*, y vuelve a entrar en la sesión.



P1. Instalar Docker-CE

Evidencias

- Al final de esta actividad práctica deberás de tener instalado **Docker-CE** en la MV que has desplegado en la nube.
- Para el portafolio, hay que recoger las siguientes evidencias:
 - Captura de pantalla del dashboard de Azure donde se vea la máquina virtual creada.
 - Describir los comandos utilizados en el proceso de instalación.
 - Captura de pantalla del terminal con la salida de los comandos:

```
$ sudo docker --version
```

```
$ sudo docker info
```



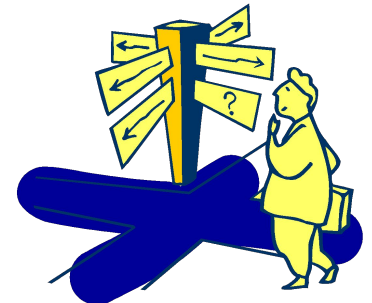
P1. Instalar Docker-CE (Community Edition)

P2. Comandos Docker

P3. Tu primer Contenedor

P4. Contenedor Servidor Web

P5. Crear un trabajo Batch (Octave)



P2. Comandos Docker

Docker images

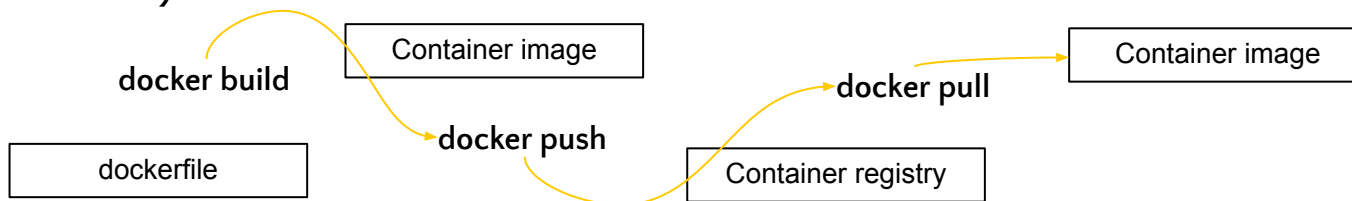
- El objetivo de esta práctica es que entiendas cómo funcionan los **comandos** más comunes de **Docker**.
- También que utilices y conozcas los comandos de Docker y manejes las opciones y parámetros básicos.

Uso

- Listado imagenes (docker images)
- Descarga (docker pull <imagen>)
- Ejecución (docker run)
- Reactivación (docker start)
- Ejecución (docker exec)
- Parada (docker stop)
- Borrado (docker rm)

Creación Imágenes

- Creación del contenedor
 - Vía creación del Dockerfile
 - Creación de la imagen (docker build).
 - Vía actualización
 - Descarga (docker pull) y actualización.
 - docker commit.
- Publicación (docker push)



P2. Comandos Docker

Docker images

- El comando ***docker images*** muestra las imágenes docker que hay en el registro del host (en vuestro caso la MV donde habéis instalado el **Docker-CE**).

```
$ docker images
```

Ejemplo: Muestra todas las imágenes del registro. Deberías de tener vacío el registro de imágenes del host, dado que todavía no hemos descargado ninguna.

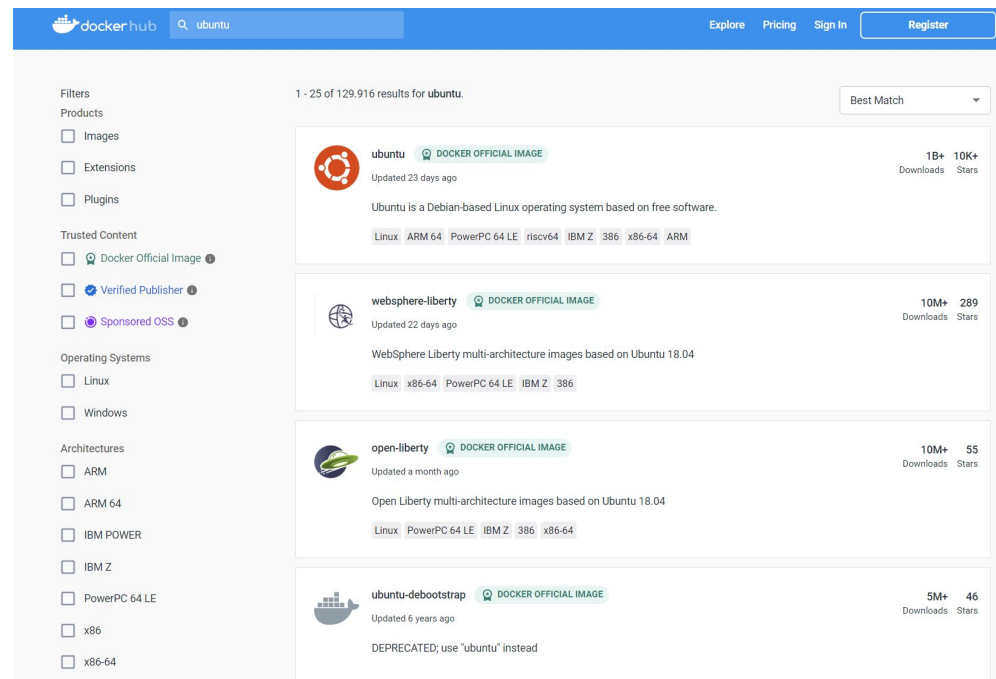
\$docker images

```
ccgc@ubuntu:~$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
```

P2. Comandos Docker

Registro Imágenes Docker Hub

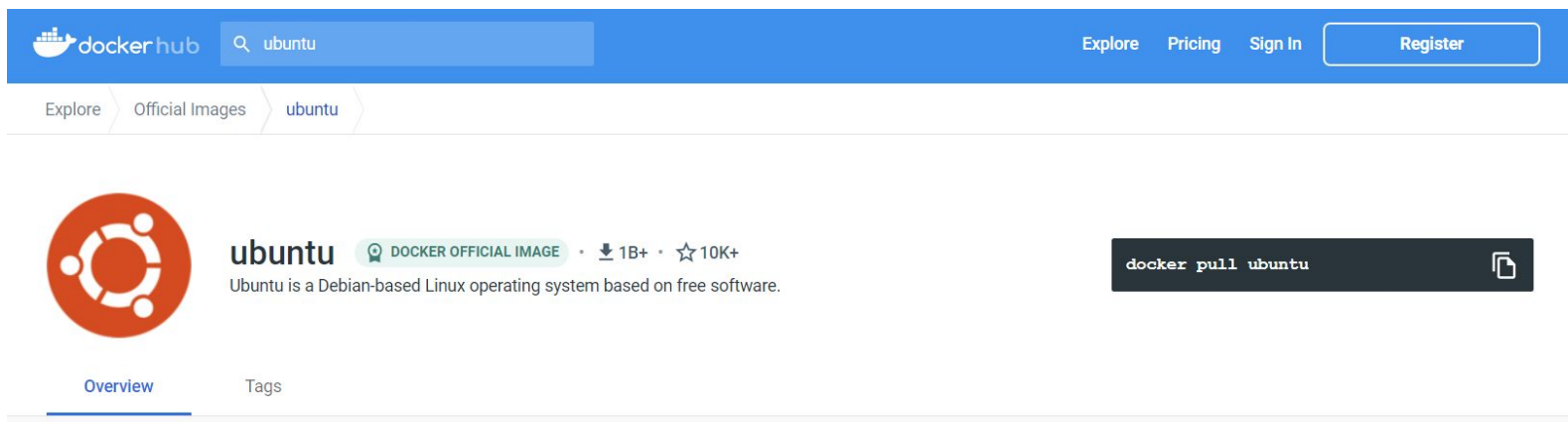
- DockerHub (<https://hub.docker.com>) es un repositorio (registro de imágenes) en la nube de imágenes docker.
- Proporciona a los Docker-CE de imágenes de contenedores que pueden ser descargadas al registro local del host para su uso.



P2. Comandos Docker

Registro Imagenes Docker Hub

- En Docker hub existen imágenes **públicas** que pueden ser descargadas por cualquier usuario o bien imágenes de uso **privado** que solo podrán ser descargadas por usuarios registrados.

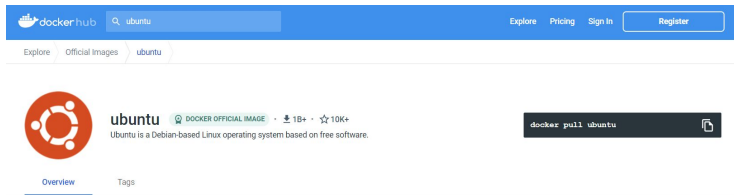


P2. Comandos Docker

Registro Imágenes Docker Hub

Docker pull

- El comando ***docker pull*** permite descargar imágenes desde el repositorio de la nube ***Docker hub*** al repositorio local para su uso. Se puede especificar que versión concreta de una imagen queremos descargar a través de los TAGS.



docker pull <etiqueta_imagen>

- Parámetros:
 <etiqueta_imagen> ☐ Etiqueta de la imagen a descargar desde Docker Hub.
- Ejemplo: Para descargar la última versión de **ubuntu** puedes lanzar el comando sin especificar ningún **tag**.

```
$ docker pull ubuntu
```

```
ubuntu@ubuntu:~$ sudo docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
cf92e523b49e: Pull complete
Digest: sha256:35fb073f9e56eb84041b0745cb714eff0f7b225ea9e024f703cab56aaa5c7720
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

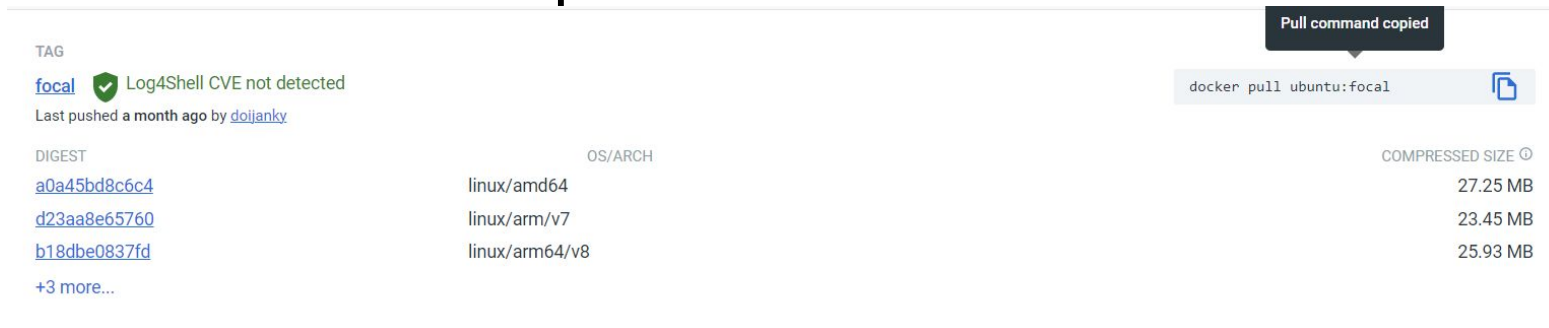
```
$ docker images
```

P2. Comandos Docker

Registro Imágenes Docker Hub

Docker pull

- Para descargar una versión concreta lo puedes especificar con los **tags**. Para ello puedes navegar por **Docker Hub** y buscar la versión que más te interese.



The screenshot shows the Docker Hub page for the 'ubuntu:focal' image. It includes a 'TAG' section with a security check, a table of image details, and a 'Pull command copied' notification.

DIGEST	OS/ARCH	COMPRESSED SIZE
a0a45bd8c6c4	linux/amd64	27.25 MB
d23aa8e65760	linux/arm/v7	23.45 MB
b18dbe0837fd	linux/arm64/v8	25.93 MB

Additional information: +3 more...
Last pushed a month ago by [doi4janky](#)

Pull command copied: `docker pull ubuntu:focal`

- Ejemplo: Descarga de una imagen de Ubuntu versión focal (v20) desde **Docker Hub**.

```
$ docker pull ubuntu:focal
```

```
$ docker images
```

P2. Comandos Docker

Docker ps

- El comando ***docker ps*** muestra un listado de los contenedores existentes (no de imágenes), así como información básica acerca de estos, tales como su ID, nombre de contenedor, el nombre de la imagen sobre la que se basa el contenedor, la hora en la que se creó el contenedor o el estado en el que se encuentra (creado, en ejecución, parado etc....), entre otros.

docker ps <options>

- Opciones básicas:
 - a ☐ muestra todos los contenedores información asociada, independientemente de su estado. Si no se pone esta opción, solo muestra los que están en marcha (Running).
- Ejemplo:** Muestra los contenedores existentes junto con la información asociada. En principio debería de salir una lista vacía, dado que todavía no se ha creado un contenedor.

```
$ docker ps -a
```

P2. Comandos Docker

Docker create

- El comando ***docker create*** crea el contenedor a partir de una imagen que tengamos en el registro local (p.e. Una de las imágenes ubuntu descargadas previamente de **Docker Hub**).
- Si la imagen no está en el registro local, la busca en **Docker Hub** y la descarga (**Docker pull**) automáticamente si existe en este registro.
- Al crear el contenedor, **NO** lo pone en funcionamiento y se queda parado (estado *created*) a la espera que lo pongan en funcionamiento.

docker create --name <etiqueta_contenedor> <imagen_contenedor> <comando>

- Parámetros básicos:
<imagen_contenedor> ☐ Imagen del registro local a utilizar para crear el contenedor.
<comando> ☐ Comando a ejecutar cuando el contenedor arranque.
- Opción básica
--name <etiqueta_contenedor> ☐ Etiqueta que se le quiere dar al contenedor para su referencia.
- **Ejemplo:** Crea un contenedor basado en la versión focal de ubuntu, que cuando se ponga en marcha, ejecutará el comando ***hostname***. El estado del contenedor creado es "Created" al realizar esta acción.

```
$ docker create --name mi_cont_ccgc_1 ubuntu:focal /bin/hostname
$ docker ps -a
```

P2. Comandos Docker

Docker start

- El comando ***docker start*** pone en funcionamiento un contenedor que esté previamente creado (p.e. creado mediante el comando ***docker create*** , aunque existen también otros comandos que pueden crear contenedores como ***docker run***) y parado.
- Este comando cambia el estado a “Exited”, el cual indica que el contenedor se puso en marcha y se terminó. También indica el tiempo desde que se paró el contenedor.

docker start <etiqueta_contenedor>

- Parámetros básicos:
 <etiqueta_contenedor> □ Etiqueta del contenedor a poner en marcha.
- **Ejemplo:** Arranca un contenedor, ejecuta el comando asociado a este y se vuelve a parar

```
$ docker start mi_cont_ccgc_1  
$ docker ps -a
```

P2. Comandos Docker

Docker run

- El comando ***docker run*** crea un contenedor a partir de una imagen que tengamos en el registro local.
- Si la imagen no está descargada en el registro local, la descarga de **Docker Hub** automáticamente (***docker pull***).
- Una vez creado el contenedor lo pone en funcionamiento.
- Permite opciones del contenedor que **docker start** no permite.

docker run <opciones> --name <etiqueta_contenedor> <imagen_contenedor> <comando>

- Argumentos básicos;
 - <imagen_contenedor> ☐ Imagen del registro local a utilizar para crear el contenedor.
 - <comando> ☐ Comando a ejecutar cuando el contenedor arranque.
 - Opciones básicas:
 - name <etiqueta_contenedor> ☐ Etiqueta a asignar al contenedor a poner en marcha. Debe ser una etiqueta que no esté en uso.
- i ☐ Abre un canal para interactuar con el proceso <comando> vía **Docker exec** o vía **pseudoterminal**.
 - El proceso ejecutado a través de <comando> debe de ser interactivo.
 - t ☐ Abre un pseudoterminal en consola para interactuar con el proceso <comando>. El terminal interactuará si la opción -i está activa.
 - d ☐ Contenedor se ejecuta en background.

P2. Comandos Docker

Docker run Ejemplos

- **Ejemplo 1:** Hace lo mismo que hacer **docker create + docker start**. La principal diferencia es que si la imagen no estuviera en el registro local la descargará automáticamente de **Docker Hub**. Crea un contenedor basado en la imagen ubuntu focal, lo pone en marcha, ejecuta el comando y para.

```
$ docker run --name mi_cont_ccgc_2 ubuntu /bin/hostname
```

```
$ docker ps -a
```

```
$ docker run --name mi_cont_ccgc_2 ubuntu /bin/hostname
```

¿Qué ocurre?



P2. Comandos Docker

Docker run Ejemplos

- **Ejemplo 2:**

```
$ docker run -t --name mi_cont_ccgc_3 ubuntu:focal /bin/hostname
```

¿Abre una terminal para interactuar? ¿porque?

```
$ docker ps -a
```

- Observarás que el contenedor está parado.

```
$ docker rm mi_cont_ccgc_3
```

- Este comando borra el contenedor.

P2. Comandos Docker

Docker run Ejemplos

- **Ejemplo 3:**

```
$ docker run -t --name mi_cont_ccgc_4 ubuntu:focal /bin/bash
```

¿Abre una terminal para interactuar? ¿por que?

¿Puedes interactuar con el proceso bash a través del pseudoterminal? ¿Por qué?

Abre un nuevo terminal y ejecuta:

```
$ docker ps -a
```

Observarás que el contenedor está en marcha, sin embargo no podemos interactuar con él.

P2. Comandos Docker

Docker run Ejemplos

- **Ejemplo 4:**

```
$ docker run -i -t --name mi_cont_ccgc_5 ubuntu:focal /bin/bash
```

¿Abre una terminal para interactuar? ¿porque?

¿puedes interactuar con el proceso bash?

```
$ docker ps -a
```

Observarás que el contenedor está en parado si has ejecutado *exit* en el pseudoterminal.

P2. Comandos Docker

Docker run Ejemplos

- **Ejemplo 5:**

```
$ docker run -d --name mi_cont_ccgc_6 ubuntu /bin/hostname
```

```
$ docker ps -a
```

- Observarás que el contenedor está parado.

P2. Comandos Docker

Docker run Ejemplos

- **Ejemplo 6:**

```
$ docker run -d --name mi_cont_ccgc_7 ubuntu /bin/bash
```

Crea el contenedor (**docker create**), el contenedor ejecuta el proceso interactivo *bash* en background.

Dado que bash es interactivo. ¿El contenedor sigue ejecutándose o se para?



P2. Comandos Docker

Docker run Ejemplos

- **Ejemplo 7:**

```
$ docker run -i -d --name mi_cont_ccgc_8 ubuntu /bin/bash
```

Crea el contenedor (**docker create**), el contenedor ejecuta el proceso interactivo *bash* en background.

Dado que bash es interactivo. ¿El contenedor sigue ejecutándose o se para?

¿Podemos interactuar de alguna forma?

```
$ docker ps -a
```

- Observarás que el contenedor está en marcha a la espera de interacción.

P2. Comandos Docker

Docker run Ejemplos

- **Ejemplo 8:**

```
$ docker run -i -t -d --name mi_cont_ccgc_9 ubuntu /bin/bash
```

Crea el contenedor (**docker create**), el contenedor ejecuta el proceso interactivo *bash* en background y abre un pseudoterminal para interaccionar.

¿Podemos interactuar a través del pseudoterminal con el contenedor?

\$ docker ps -a

- Observarás que el contenedor está en marcha a la espera de interacción.

P2. Comandos Docker

Docker Exec

- **"docker exec"** permite lanzar comandos sobre un contenedor que **ya esté en marcha** de forma interactiva.

\$ docker exec <opciones> <contenedor> <comando>

- Argumentos básicos:

<contenedor> ☐ Contenedor interactivo que esté en marcha.

<comando> ☐ Comando a ejecutar en el contenedor.

- Opciones básicos:

-i ☐ Abre un canal para interactuar con el proceso <comando>. El proceso ejecutado a través de <comando> debe de ser interactivo.

-t ☐ Abre un pseudoterminal en consola para interactuar con el proceso <comando> del contenedor.

P2. Comandos Docker

Docker exec Ejemplos

- **Ejemplo 1:**

```
$ docker exec mi_cont_ccgc_9 /bin/hostname
```

Lanza el comando *hostname* sobre el proceso del contenedor que tiene el canal interactivo activo. En este caso el proceso interactivo es */bin/bash*.

```
$ docker ps -a
```

Observaras que después de lanzar el comando, el contenedor sigue funcionando y NO está parado dado que el proceso interactivo sigue en marcha a la espera de nuevos comandos.

P2. Comandos Docker

Docker exec Ejemplos

- **Ejemplo 2:**

```
$ docker exec -i -t mi_cont_ccgc_9 /bin/bash
```

Lanza el comando */bin/bash* sobre el proceso del contenedor que tiene el canal interactivo activo, en este caso es otro */bin/bash*. El nuevo proceso lanzado tendrá también un canal interactivo con el que podremos interactuar vía pseudoterminal..

```
$ docker ps -a
```

Observaras que después de lanzar el comando e interactuar con el terminal, después de cerrar el pseudoterminal, el contenedor sigue funcionando y NO está parado dado que el proceso interactivo sigue en marcha a la espera de nuevos comandos.

P2. Comandos Docker

Docker stop

- El comando **docker stop** para un contenedor que esté en funcionamiento.

docker stop <etiqueta_contenedor>

Ejemplo:

Parar un contenedor que tengamos en marcha (*running*).

```
$ docker ps -a
```

Observarás que `mi_cont_ccgc_9` está activo

```
$ docker stop mi_cont_ccgc_9
```

```
$ docker ps -a
```

Observarás que `mi_cont_ccgc_9` está parado

P2. Comandos Docker

Docker commit

- El comando **docker commit** permite generar una nueva imagen de un contenedor a partir de un contenedor existente.
- Es útil cuando lanzamos un contenedor interactivo y a través del pesudoterminal podemos instalar paquetes, aplicaciones o servicios específicos dentro del contenedor. Cuando el contenedor está ya actualizado, podemos generar una imagen nueva a partir de dicho contenedor, la cual contendrá ya todos los cambios.

docker commit <etiqueta_contenedor> <etiqueta_nueva_imagen>

P2. Comandos Docker

Docker commit

Ejemplos

```
$ docker commit mi_cont_ccgc_9 ubuntu:mi_cont_ccgc_9
```

Crea una nueva imagen a partir del contenedor existente `mi_cont_ccgc_9` y se guarda con la etiqueta **ubuntu:mi_cont_ccgc_9** en el registro local.

```
$ docker images
```

- Observarás que has creado una nueva imagen en el registro de imágenes.

P2. Comandos Docker

Docker build

- El comando **docker build** crea una imagen docker a partir de un fichero Dockerfile y guarda la imagen en el registro local.

docker build <opciones> <Path>

- Parámetros básicos
 - <Path> □ Path donde se encuentra el fichero Dockerfile
- Opciones básicas:
 - t <TAG_nueva imagen> □ Etiqueta que se registrara en el registro de imágenes para la nueva imagen.

P2. Comandos Docker

Docker build

Ejemplo

- Copia este contenido en un fichero de texto y guárdalo con el nombre "Dockerfile":

```
FROM ubuntu
```

```
RUN apt-get update
```

```
RUN apt-get install -y python3
```

```
ENTRYPOINT /bin/bash
```

```
$docker build -t mi_cont_ccgc_9:build .
```

```
$ docker images
```

Observarás que se ha creado una nueva imagen

P2. Comandos Docker

Docker rm

- El comando ***docker rm*** elimina uno o más contenedores.

```
$ docker rm <opciones> <etiqueta_contenedor>
```

opciones:

-f ☐ fuerza a la eliminación de un contenedor en marcha.

Ejemplo:

```
$ docker rm mi_cont_ccgc_2
```

P2. Comandos Docker

Docker rmi

- El comando ***docker rmi*** elimina una imagen de contenedor de nuestro docker registry.

```
$ docker rmi <opciones> <etiqueta_imagen>
```

opciones:

-f ☐ fuerza a la eliminación de una imagen .

Ejemplo:

```
$ docker rmi ubuntu:mi_cont_ccgc_9
```


P2. Comandos Docker

Evidencias

- Al final de esta práctica deberás de tener en el registro de imágenes de tu máquina las imágenes generadas por los comandos de ejemplo que hemos visto.
- También deberás poder obtener la información (`docker ps -a`) de todos los contenedores creados.
- Haz una captura de ambas cosas y adjúntala al portafolio



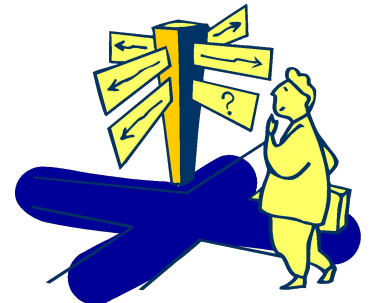
P1. Instalar Docker-CE (Community Edition)

P2. Comandos Docker

P3. Tu primer Contenedor

P4. Contenedor Servidor Web

P5. Crear un trabajo Batch (Octave)



P3. Tu primer Contenedor

Ejercicio: Crea un contenedor

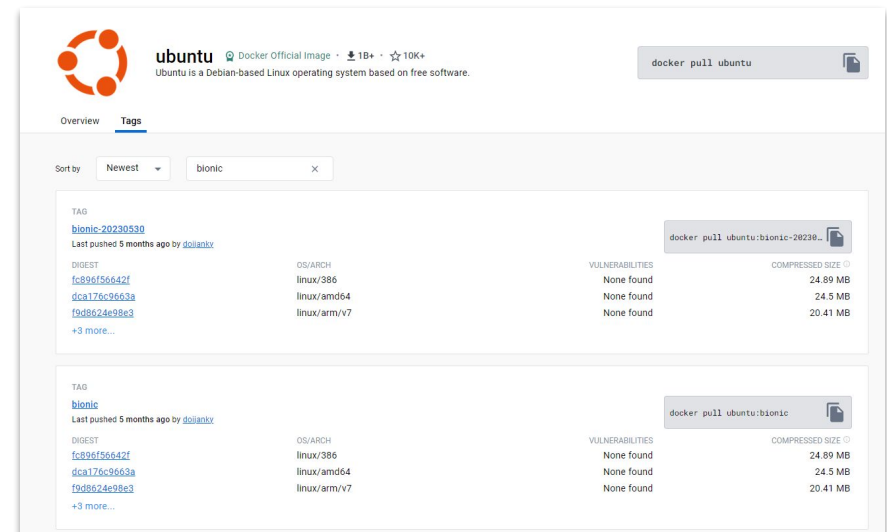
- El objetivo de esta práctica es que crees un contenedor básico de forma autónoma, utilizando los comandos vistos en el ejercicio anterior.
- El contenedor a crear debe seguir los pasos que se definen en la siguiente transparencia.



P3. Tu primer Contenedor

Ejercicio: Crea un contenedor

- Descarga una imagen del contenedor "ubuntu" versión "bionic". Búscala en Docker hub y descárgala en tu registro local.
- Lista las imágenes existentes en tu registro local para testar que se ha descargado correctamente la imagen ubuntu versión bionic.
- Pon el contenedor en marcha de forma interactiva de forma que se lance un proceso **bash**. Además debes de abrir un pseudoterminal para interactuar con el contenedor. Debes de llamar al contenedor "*mi_primer_cont*"
- A través del psedoterminal lanza dentro del contenedor los siguientes comandos a través de la sesión interactiva.
 - hostname
 - cat /etc/hosts
 - ls -l /
 - whoami
- Sal del pseudoterminal del contenedor.
 - exit
- Comprueba el estado del contenedor.
- Borra el contenedor activo.



P3. Tu Primer contenedor

Evidencias

- Para el portafolio, hay que recoger las siguientes evidencias:
 - Listar y describir los comandos Docker empleados para la realización del ejercicio.
 - Puedes añadir capturas de pantalla del terminal con la salida de los comandos.



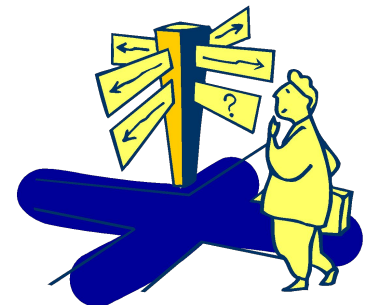
P1. Instalar Docker-CE (Community Edition)

P2. Comandos Docker

P3. Tu primer Contenedor

P4. Contenedor Servidor Web

P5. Crear un trabajo Batch (Octave)



P4. Contenedor Servidor Web

- El **objetivo** de esta práctica es que despliegues un servidor Web mediante un contenedor.
- Para ello, debes conocer cómo se gestionan los puertos de acceso a un contenedor a través del comando ***docker run***.



P4. Contenedor Servidor Web

Docker run Gestión de puertos

- La IP de un contenedor funcionando coincide con la de su host.
- **Docker run** puede definir unas reglas de redirección de puertos que permiten canalizar tráfico desde el exterior del host al interior del contenedor.
- Esta opción cobra sentido cuando se lanzan procesos interactivos (-t) en background (-d) (p.e. servicio web).
- Se establecen mediante la opción -p
puerto_host:puerto_contenedor

Ejemplo: Arranca un proceso bash interactivo y abre el puerto 8080 del host para acceder al puerto 80 del contenedor.

```
$ docker run -i -t -p 8080:80 --name <mi_contenedor> <imagen> /bin/bash
```


P4. Contenedor Servidor Web

Ejercicio 1: Crear imagen

- Utiliza la imagen de ubuntu:focal ya descargada en tu registro.
- Crea un contenedor en background y en modo interactivo con un pseudoterminal. El comando a lanzar será un *bash* y debes llámalo *mi_cont_serv_web*.
- Entra en el pseudoterminal y actualiza el contenedor con la instalación de apache2
 - `apt-get update`
 - `apt-get install -y apache2`
 - `echo "<h1> Pon tu nombre </h1>" > /var/www/html/index.html`
- Sal del pseudoterminal del contenedor y registra los cambios en una nueva imagen con la etiqueta *mi_img_serv_web:v1*
- Elimina el contenedor *mi_cont_serv_web* que tienes en marcha.
- Vuelve a crear con la nueva imagen un contenedor en background e interactivo que lance un *bash* y redirige el puerto 8080 del host al puerto 80 del contenedor.
- Lanza un nuevo proceso "*service apache2 start*" sobre el contenedor que acabas de crear y que debería estar en marcha a la espera de nuevos comandos.
- Testea que el puerto 8080 es accesible en la máquina y que a través de este accederá al puerto 80 donde está el servicio apache dentro del contenedor.
 - `curl localhost:8080`
`<h1>Matias Whorkshop</h1>`

P4. Contenedor Servidor Web

Ejercicio 2: Construir desde dockerfile

- Alternativamente al comando ***docker commit*** para crear una imagen, se puede crear a partir de un fichero dockerfile.
- Crea un fichero "Dockerfile" en la carpeta `"/home/usuario/"` con el contenido del siguiente y crea una nueva imagen `"mi_img_serv_web:v2"`:

```
FROM ubuntu:focal
RUN apt-get update
ENV TZ=Europe/Madrid
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ >
/etc/timezone
RUN apt-get install -y apache2
RUN echo "<h1>pon tu nombre</h1>" > /var/www/html/index.html
EXPOSE 80
ENTRYPOINT apache2ctl -D FOREGROUND
```

P4. Contenedor Servidor Web

Ejercicio 2: Construir desde dockerfile

- Pon en marcha un nuevo contenedor (en background) a partir de esta imagen nueva imagen *"mi_img_serv_web:v2"* y llámalo *"mi_cont_serv_web_build"*.
- Testea que el puerto 8080 es accesible en la máquina y que a través de este accederá al puerto 80 donde está el servicio apache dentro del contenedor.
 - `$ curl localhost:8080`

`<h1>Matias Whorkshop</h1>`
- Si quisiéramos acceder desde un navegador, deberíamos abrir el puerto 8080 en la MV.

Nota: si tienes el contenedor anterior en marcha, deberás utilizar otro puerto diferente al 8080, ya que lo está utilizando el otro contenedor. Prueba, por ejemplo, con el 8081

P4. Contenedor Servidor Web

Evidencias

- Para el portafolio, hay que recoger las siguientes evidencias:
 - Captura de pantalla del terminal donde se vean las dos imágenes de contenedor en tu registro local.
 - Capturas de pantalla del terminal donde aparezca cada uno de los contenedores en marcha con el puerto correspondiente abierto.
 - Capturas de pantalla accediendo a la web creada (salida del comando curl), en ambas pruebas.
 - Contenido del fichero Dockerfile creado.



P1. Instalar Docker-CE (Community Edition)

P2. Comandos Docker

P3. Tu primer Contenedor

P4. Contenedor Servidor Web

P5. Crear un trabajo Batch (Octave)



P5. Crear Un trabajo Batch

- El **objetivo** de este ejercicio es que despliegues un contenedor como un trabajo batch que procese unos ficheros como entrada.
- Para ello deberás de conocer cómo gestionar espacios compartidos de disco entre el host y contenedores (Volúmenes).
- También deberás conocer mecanismos para transferir ficheros del host a un contenedor y viceversa.



P5. Contenedor como Trabajo Batch

Docker run Gestión Volúmenes

- Montaje de volúmenes

- Es posible montar un volumen del sistema “host” sobre el contenedor cuando se pone en marcha
- **Opción -v** directorio_host:directorio_contenedor
- Ejemplo:
 - `$ docker run -i -t --name micont -v $HOME:/mnt ubuntu /bin/bash`
 - Dentro del contenedor comprobamos el montaje
 - `ls -la /mnt`
 - `echo “dentro_del_contenedor” > /mnt/fichero`
 - Salir y comprobar la existencia del fichero ¿quién es el propietario?

```
-rw-r--r-- 1 root root    6 Oct 23 18:36 fichero
```

P5. Contenedor como Trabajo Batch

Docker cp

Transferencia de Ficheros

- Este comando permite el copiado de datos desde host a un contenedor activo y viceversa.

```
$ docker cp ruta_origen ruta_destino
```

- Ejemplo: Copia el directorio temporal del contenedor al directorio temporal del host

```
$ docker cp mi_cont:/tmp /tmp
```

- Ejemplo: Copia el directorio temporal del host al directorio temporal del contenedor

```
$ docker cp /tmp mi_cont:/tmp
```


P5. Crear un trabajo Batch (Octave)

Ejercicio: Resolución Ecuaciones lineales

- Queremos resolver un sistema de ecuaciones lineales que tenemos en un fichero (en PoliformaT).

- Son 3 ficheros:

- octave/procesa.m
- octave/A
- octave/b



- El contenido de procesa.m es:

```
cd /octave
```

```
load A;
```

```
load b;
```

```
x=A\b;
```

```
save x;
```

```
err=norm(x-ones(size(x)))
```

- Vamos a resolver el problema mediante un contenedor.

- Cambiamos al directorio /octave
- Cargamos la matriz de coeficientes A
- Cargamos el vector de términos independientes b
- Resolvemos el sistema
- Guardamos la solución del sistema
- Calculamos el error cometido comparando con la solución esperada.

P5. Crear un trabajo Batch (Octave)

Ejercicio: Resolución Ecuaciones lineales

- Descarga los ficheros A, B y procesa.m de poliformaT y guárdalos en la carpeta */home/usuario/* de tu MV.

- Prepara en el registro local una imagen de contenedor con octave aplicando **al menos dos** de las opciones:

¡¡Puede dar [error de cuotas](#)!!

- **Opción 1:** Descargar una imagen válida de octave directamente desde DockerHub. Puedes utilizar la imagen [gnuoctave/octave](#) o [simexp/octave](#).
- **Opción 2:** Instalar manualmente octave sobre un contenedor ubuntu (más costoso) mediante el comando `apt-get install -y octave`, y después hacer el commit para guardar y crear la nueva imagen octave. Llama a esta imagen "mi_img_octave:op2".
- **Opción 3:** Hacer un dockerfile que construya la imagen con octave. "mi_img_octave:op3" a partir de un ubuntu versión focal.

P5. Crear un trabajo Batch (Octave)

Ejercicio: Resolución Ecuaciones lineales

- Lanza una de las tres imágenes creadas con un directorio compartido con el host y llámala "mi_cont_octave". El directorio del host será */home/usuario/* y la carpeta dentro del contenedor será */myvol*.
- Entra en el contenedor mediante un pseudoterminal y accede a la carpeta compartida dentro del contenedor */myvol*.
- Edita el fichero *procesa.m* para que el directorio dentro del contenedor sea tu carpeta compartida (*/myvol*)
- Sal del pseudoterminal.
- Ejecuta un proceso Matlab con el siguiente comando "*octave /myvol/procesa.m*" a través de los comandos docker sobre el contenedor creado.

P5. Trabajo Batch (Octave)

Evidencias

- Para el portafolio, hay que recoger las siguientes evidencias:
 - Captura de pantalla del terminal donde se vean las tres imágenes de contenedor en tu registro local.
 - `gnuoctave/octave` (o `simexp/octave`)
 - `mi_img_octave:op2`
 - `mi_img_octave:op3`
 - Captura de pantalla del terminal donde aparezcan los contenedores creados.
 - Captura de pantalla del terminal donde aparezca el volumen compartido con el host en el contenedor.



P5. Trabajo Batch (Octave)

Evidencias

- Una vez guardada las Evidencias **DESTRUYE** todos los recursos creados eliminando el grupo de recursos correspondiente.

