Prácticas.
Contenedores Docker (III)

Máster Universitario en Computación en la Nube y de Altas Prestaciones

Objetivos

- Entender cómo usar de forma apropiada las funcionalidades básicas proporcionadas por la **tecnología docker** para gestionar:
  - **procesos (batch e interactivos)** empaquetados en contenedores**.**
  - **creación de imágenes** de contenedores con ficheros Dockerfile.
  - gestión de imágenes de contenedores con **repositorios externos.**
  - **comunicaciones de red** entre contenedores.
- Probar y testear dichas opciones a través de un host local desplegado en la nube mediante Docker-CE.

Contenidos

Grid y Computación de Altas Prestaciones
GRyCAP

P8. Creación Automatizada de Imágenes.

P9. Gestión Repositorios Externos.

P10. Gestión Básica de Redes.

P8. Creación Automatizada de Imágenes
Comandos Básicos Docker

Grid y Computación de Altas Prestaciones
GRyCAP

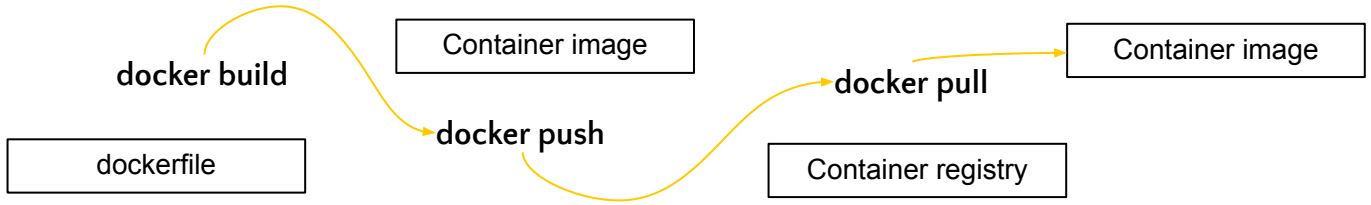
- El objetivo de esta práctica:
  - entender los **comandos** fundamentales de **Docker** para la creación de imágenes de contenedores de forma automatizada.
  - utilizar los comandos de Docker para la creación de imágenes docker.

Uso

- Listado imagenes (docker images)
- Descarga desde Docker Hub (docker pull <imagen>)
- Eliminar imagen (docker rmi)
- Consulta listado (docker ps)
- Consulta Contenedor (docker Inspect)
- Crear Contenedor (docker create)
- Activación (docker start)
- Parada (docker stop)
- Ejecución (docker Run)
- Borrado (docker rm)
- Consulta puertos (docker port)
- ejecución (docker attach)
- Ejecución (docker exec)
- Volúmenes (docker cp)

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)



P8. Creación Automatizada de Imágenes
Docker build

Grid y Computación de Altas Prestaciones GRyCAP

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

docker build <opciones><Path_Dockerfiile>

- **Parámetros:**

  <Path_Dockerfile> → Path donde se encuentra el fichero Dockerfile

- **Opciones**

  -t <TAG_nueva imagen> → Etiqueta que se registrara en el registro de imágenes local para la nueva imagen.

P8. Creación Automatizada de Imágenes
Docker build - Fichero DockerFile

Grid y Computación de Altas Prestaciones GRyCAP

- Un Dockerfile es un fichero de texto que contiene bloques de instrucciones (Ej. FROM, RUN, COPY, CMD etc…) que indican cómo se construye una imagen de Docker.
- Las imágenes se construyen por capas, en la que cada instrucción (de arriba a abajo del fichero Dockerfile) crea una capa nueva en la imagen basándose en la capa anterior.
- Ventajas:
  - **Reutilización** → Si cambias una instrucción de una capa, Docker reutiliza las capas anteriores y solo rehace las posteriores.
  - **Eficiencia** → Las capas se comparten entre imágenes. (Ej: dos imágenes basadas en ubuntu:focal comparten esa capa sin duplicarla).
  - **Ligereza** → Sólo las diferencias (capas nuevas) ocupan espacio extra.

P8. Creación Automatizada de Imágenes
Docker build - Fichero DockerFile - Instrucciones

Grid y Computación de Altas Prestaciones
GRyCAP

- Ejemplo de dockerfile.

```
FROM python:3.11-slim                              # Capa 1. Imagen base
LABEL maintainer="J. Damian Segrelles"             # Capa 2. Información opcional del autor
WORKDIR /app                                       # Capa 3. Directorio de trabajo dentro del contenedor
COPY requirements.txt /app/                        # Capa 4. Copiar archivos del host al contenedor
RUN pip install --no-cache-dir -r requirements.txt # Capa 5. Instalar dependencias
COPY . /app                                        # Capa 6. Copiar el resto de archivos del proyecto
EXPOSE 5000                                        # Capa 7. Exponer el puerto
VOLUME ["/app/data"]                               # Capa 8. Declarar un volumen persistente (para logs,
                                                   #          datos, etc.)
CMD ["python", "app.py"]                           # Capa 9. Comando por defecto al arrancar el contenedor
```

P8. Creación Automatizada de Imágenes
Docker build - Fichero DockerFile - Instrucciones

Grid y Computación de Altas Prestaciones
GRyCAP

- Las instrucciones más comunes en los Dockerfile son los siguientes:

1. FROM: Define la imagen base sobre la que se construirá la nueva imagen.

Ej. *FROM ubuntu:focal*

2. LABEL: Agrega metadatos (autor, versión, descripción) relativas a la imagen.
Ej. *LABEL maintainer="tucorreo@example.com" version="1.0"*

3. WORKDIR: Establece el directorio de trabajo dentro del contenedor. Cuando arranca el contenedor, en runtime se moverá a este directorio. Si el directorio no existe, lo crea automáticamente. Esta opción se puede también aplicar con *Docker create/run -w <directorio>*.

Ej. *WORKDIR /app*

4. COPY / ADD: Copia archivos de forma recursiva (carpetas y subcarpetas) desde el host al contenedor.

Ejs. *COPY . /app*
ADD https://www.web.com/fichero.sh /app/fichero.sh
ADD app.tar.gz /app #cópia y descomprime

(**ADD** hace lo mismo que **COPY** pero además permite descargar URLs o descomprimir `.tar`).

P8. Creación Automatizada de Imágenes
Docker build - Fichero DockerFile - Instrucciones

Grid y Computación de Altas Prestaciones GRyCAP

5. RUN: Ejecuta comandos durante la construcción (instalar paquetes, crear directorios, etc.). Se pueden encadenar varios comandos en una misma capa RUN utilizando &&.

Ej. *RUN apt-get update && apt-get install -y curl*

6. ENV: Declara variables de entorno dentro de la imagen. Esta opción se puede también aplicar a con *Docker create/run -e <variable>=<valor>*.

Ej. *ENV APP_ENV=production*

7. EXPOSE: Documenta qué puertos va a usar la aplicación pero no los mapea, solo es a nivel informativo.

Ej. *EXPOSE 8080*

8. VOLUME: Declara directorios que se deben montar como volúmenes persistentes anónimos. Los datos almacenados en ese volumen no se pierden aunque el contenedor se elimine. **Esta opción NO permite montar** directorios del host para persistencia como se hacía con **docker create -v**.

Ej. *VOLUME ["/data"]*

P8. Creación Automatizada de Imágenes
Docker build - Fichero DockerFile - Instrucciones

Grid y Computación de Altas Prestaciones GRyCAP

9. CMD: Indica el cmd de la imagen. Se recomienda escribirlo en formato JSON.

CMD ["executable","param1","param2"]

10. ENTRYPOINT: Define el entrypoint de la imagen. Este comando siempre se ejecuta cuando arranca el contenedor. Por lo general, si se define el ENTRYPOINT, el CMD se utiliza para pasarle argumentos al ENTRYPOINT. Se recomienda escribirlo en formato JSON.

Ej.

ENTRYPOINT ["python", "app.py"]

11. ARG: Permite pasar argumentos en tiempo de construcción del build al dockerfile .

Ej.

ARG VERSION=1.0

RUN echo "Building version $VERSION"

P8. Creación Automatizada de Imágenes
Docker build

- **Tarea 8.1:** Crea un contenedor que tenga python instalado. Para ello debes de Crea un Dockerfile que genere una imagen con las siguientes características:
- Llama a la nueva imagen mi_contenedor:t8.1
  - Debe de estar basada en ubuntu:focal. (FROM)
  - Debe tener instalado python (RUN).
    apt-get update && apt-get install -y python
  - La imagen tiene que definirse de forma que al crear y lanzar un contenedor basado en la imagen sea interactivo con el comando (/bin/bash).

```
FROM XXX
RUN XXX
ENTRYPOINT XXX
```

- Lanza y prueba un contenedor (mi_con1_build) con la nueva imagen. Una vez lanzado, entra en el contenedor y muestra la versión de python instalada. “python --version”

P8. Creación Automatizada de Imágenes
Docker build

Grid y Computación de Altas Prestaciones GRyCAP

- **Tarea 8.2:** Crea un contenedor que calcule la distanci entre dos puntos con la librería numpy. Para ello debes de crear un Dockerfile que genere una imagen con las siguientes características:
  - Llama a la nueva imagen mi_contenedor:t8.2
  - La imagen debe estar basada en la imagen construida en la tarea 8.1. (FROM).
  - Tiene que instalar la librería python 3.10 de numpy (RUN).
    - apt-get install -y python-numpy
  - Define una carpeta de trabajo /mis_aplicaciones_python (WORKDIR).
  - Descargate de poliformaT un programa python que calcula la distancia entre dos puntos utilizando la libreria numpy (fichero de poliformaT distancia.py) .
  - Copia desde el HOST a la carpeta de trabajo de la imagen del contenedor el programa python (COPY).
  - La imagen tiene que definirse de forma que al lanzar el contenedor, este sea interactivo con el proceso /bin/bash.
- Lanza y prueba un contenedor (mi_con2_build) con la nueva imagen.

P8. Creación Automatizada de Imágenes

Docker build

Grid y Computación de Altas Prestaciones

GR y CAP

P8. Creación Automatizada de Imágenes
Docker build

Grid y Computación de Altas Prestaciones GRyCAP

- **Tarea 8.3:** Crea un servidor nginx en un contenedor que persista los logs del servicio web en un volumen. Para ello debes de crear un Dockerfile que genere una imagen con las siguientes características:
  - Llama a la nueva imagen mi_contenedor:t8.3
  - Debe de estar basada en ubuntu:focal. (FROM)
  - Debe tener instalado python (RUN).
    - apt-get update
    - apt-get install -y nginx.
  - Crea un volumen que persista los logs del directorio /var/log/nginx (VOLUME)
  - Expone el puerto 80 como metadato del contenedor (EXPOSE)
  - La imagen tiene que definirse de forma que al crear y lanzar un contenedor basado en la imagen lance el servicio web con el comando (ENTRYPOINT ["nginx", "-g", "daemon off;"]).
- Lanza y prueba un contenedor (mi_con3_build) con la nueva imagen.

P8. Creación Automatizada de Imágenes
Docker build

Grid y Computación de Altas Prestaciones
GRyCAP

P8. Creación Automatizada de Imágenes
Evidencias

Grid y Computación de Altas Prestaciones
GRyCAP

- Al final de esta actividad práctica deberás de tener el entorno **Docker-CE** con una serie de contenedores creados.
- Verifica que están todos los contenedores con docker ps -a:
- Para el portafolio, hay que recoger las siguientes evidencias:
  - Captura de pantalla del terminal con la salida del comando anterior donde aparezca tu usuario.
  - Texto donde indique la tarea y el comando o comandos utilizados.

Contenidos

Grid y Computación de Altas Prestaciones
GRyCAP

P8. Creación Automatizada de Imágenes.

P9. Gestión Repositorios Externos.

P10. Gestión Básica de Redes.

P9. Gestión Repositorios Externos
Comandos Básicos Docker

Grid y Computación de Altas Prestaciones GRyCAP

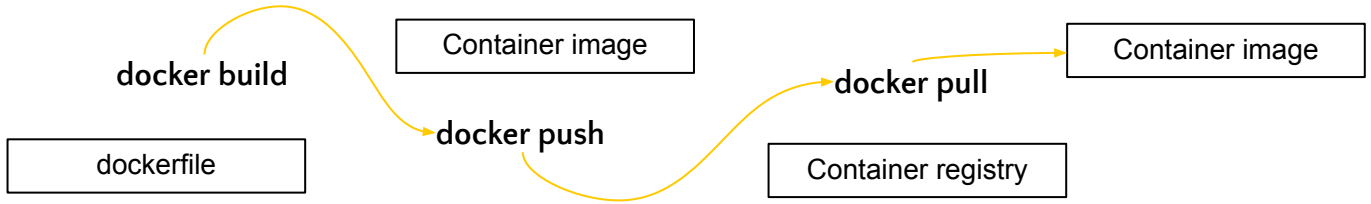
- El objetivo de esta práctica:
  - entender los **comandos** fundamentales de **Docker** para interactuar con repositorios externos y subir imágenes al repositorio.
  - utilizar los comandos de Docker para interactuar con repositorios externos.

Uso

- Listado imagenes (docker images)
- Descarga desde Docker Hub (docker pull <imagen>)
- Eliminar imagen (docker rmi)
- Consulta listado (docker ps)
- Consulta Contenedor (docker Inspect)
- Crear Contenedor (docker create)
- Activación (docker start)
- Parada (docker stop)
- Ejecución (docker Run)
- Borrado (docker rm)
- Consulta puertos (docker port)
- ejecución (docker attach)
- Ejecución (docker exec)
- Volúmenes (docker cp)

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)



P9. Gestión Repositorios Externos

- Cuando se crean imágenes, puede resultar interesante publicarlas en docker hub.
- El ciclo de vida es el siguiente:
  - Iniciar Sesión Docker Hub
  - Etiquetar Imagen del registro local
  - Subir imagen a Docker Hub

P9. Gestión Repositorios Externos
Docker login

Grid y Computación de Altas Prestaciones GRyCAP

- docker login conecta con un registro externo de imágenes para descargar y/o subir imágenes docker.
- Por defecto, el registro configurado es Docker Hub.

docker login <opciones> [registro]

- Parámetros
  [registro] → Registro de imágenes donde conectarse. si no se indica nada, por defecto es Docker Hub.
- Opciones
  -u <usuario> → Usuario registrado en Docker Hub.
  -p <password> → Password de acceso del usuario.

P9. Gestión Repositorios Externos
Docker login

- **Tarea 9.1:** Create una cuenta en Docker Hub (si no la tienes en https://hub.docker.com) y posteriormente haz un login con tu usuario/contraseña con docker.

P9. Gestión Repositorios Externos
Docker tag

Grid y Computación de Altas Prestaciones GRyCAP

- **docker tag** permite cambiar los tags de una imagen del registro local (docker login).
- Genera una nueva imagen en el repositorio local, con el fin de subirla a un repositorio externo.

docker tag [NOMBRE_IMAGEN_LOCAL][:TAG] <usuario>/[NOMBRE_IMAGEN_REGISTRO[:TAG]

- **Parámetros**

-[NOMBRE_IMAGEN_LOCAL][:TAG] → Nombre y tag de la imagen del registro local que se quiere subir al registro de docker hub.

-<usuario> → Usuario del registro externo.

-[NOMBRE_IMAGEN_registro][:TAG]→ Nombre y tag de como quieres subir a Docker Hub la imagen.

P9. Gestión Repositorios Externos
Docker tag

- **Tarea 9.2:** Cambia el tag de la imagen creada en la tarea 8.3 al tag <*usuario_docker_hub*>*/mi_contenedor:t8.3.*

P9. Gestión Repositorios Externos
Docker push

- **docker push** permite subir una imagen del registro local a un registro externo previamente "logeado" (docker login).....

docker push <usuario>/[NOMBRE_IMAGEN_registro][:TAG]

- **Parámetros**
  -<usuario> → Usuario del registro externo.
  -[NOMBRE_IMAGEN_registro][:TAG]→ Nombre y tag de la imagen a subir desde el registro local al repositorio externo.

P9. Gestión Repositorios Externos
Docker push

Grid y Computación de Altas Prestaciones
GRyCAP

- **Tarea 9.3:** Subir la imagen a la que has cambiado el tag en la tarea 9.2.

P9. Gestión Repositorios Externos
Evidencias

- Al final de esta actividad práctica deberás de tener el entorno **Docker-CE** con una serie de contenedores creados.
- Verifica que están todos los contenedores con docker ps -a:
- Para el portafolio, hay que recoger las siguientes evidencias:
  - Captura de pantalla del terminal con la salida del comando anterior donde aparezca tu usuario.
  - Texto donde indique la tarea y el comando o comandos utilizados.

Contenidos

P8. Creación Automatizada de Imágenes.

P9. Gestión Repositorios Externos.

P10. Gestión Básica de Redes.

P10. Gestión Básica de Redes

- Docker gestiona la conectividad entre contenedores a través de diferentes tipos de redes lógicas virtuales.
- Las comunicaciones entre contenedores y/o entre contenedor←→host se pueden realizar mediante:
  - **Redes bridge**
    - red bridge creada por defecto por docker.
    - redes bridge definidas por el usuario.
  - **La red propia del host.**
- Si queremos aislar a un contenedor de otros contenedores y del propio host existe la red **"none".**

P10. Gestión Básica de Redes

- **Docker network ls** permite mostrar todas las redes gestionados por docker.

docker network ls

El output del comando es una lista con todas las redes creadas con la siguiente información:

NETWORK ID: Identificador de la red (versión abreviada) de 12 caracteres.
NAME: Nombre de la red.
DRIVER: Driver utilizado para la gestión de la red.
SCOPE: El alcance o ámbito en el que esa red existe y puede ser usada.
local: red que solo existe en el host.
global: pueden conectar contenedores que corren en distintos hosts en un cluster.
…

P10. Gestión Básica de Redes

Grid y Computación de Altas Prestaciones GRyCAP

- **Tarea 10.1:** Lista todas las redes disponibles en tu Docker CE.

¿qué redes aparecen?

P10. Gestión Básica de Redes

- **Docker inspect** permite mostrar toda la información relativa a una red gestionado por docker.
- La información la muestra en formato JSON y muestra información relativa como las IPs asignadas a los distintos contenedores, rango de la subred, IP del gateway etc…

docker inspect <nombre red>

- **Parámetros**
  - **<nombre_red>** → nombre de la red bridge a inspeccionar.

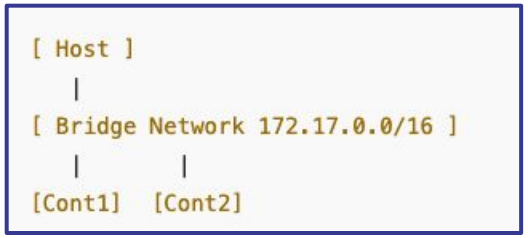
P10. Gestión Básica de Redes

- **Tarea 10.2:** Consulta las IPs privadas de todos los contenedores que has creado a lo largo de las prácticas. Todos ellos, deberían de estar en la red "bridge".

P10. Gestión Básica de Redes
Red bridge por Defecto

Grid y Computación de Altas Prestaciones GRyCAP

- La red bridge por defecto la crea Docker cuando se instala Docker CE. No se requiere de ningún comando en especial.
- Actúa como un **switch virtual interno** en la que se crea un interfaz virtual de red (**docker0**), asignando una subred y una IP privada a cada contenedor (subred 172.17.0.0/16), y un gateway (que es el host con IP 172.17.0.1) para las comunicaciones con el exterior (Internet).
- Todo el tráfico entre contenedor ↔ host pasa por esta interfaz virtual docker0, usando NAT si se comunica hacia fuera (internet) a través del gateway, aunque no se expone automáticamente los puertos al exterior. Se requiere de un mapeo desde el host a contenedores (docker create/run -p o -P).
- **Los contenedores pueden comunicarse libremente entre sí utilizando cualquier puerto (no hay reglas fireware)** usando la IP privada que tienen asignada. **El nombre del contenedor NO se puede utilizar en la bridge por defecto**.
- Las IPs se asignan de forma dinámica cada vez que se arranca un contenedor (docker start/run). Si se para el contenedor y se vuelve a arrancar puede que se le asigne una IP diferente.



P10. Gestión Básica de Redes
Red bridge por Defecto

Grid y Computación de Altas Prestaciones
GRyCAP

- **Tarea 10.3:** Crea dos contenedores:
  - un contenedor interactivo (/bin/bash) y llámalo mi_con1_red.
  - un contenedor que exponga el servicio nginx y llámalo mi_con1_red_web.

  Consulta las IPs privadas de los dos contenedores creados (docker inspect bridge). Lanza ambos contenedores y entra mediante pseudoterminal al contenedor interactivo. Una vez dentro:
  - instala curl (apt-get-update && apt-get install curl)
  - ejecuta "curl http://mi_con1_red_web"
  - Luego ejecuta "curl http://<IP_Privada_mi_con1_redweb>."

¿qué ocurre?

P10. Gestión Básica de Redes
Red Bridge definida por el usuario

Grid y Computación de Altas Prestaciones
GRyCAP

- La red bridge definida por el usuario son redes creadas por los propios usuarios de docker. Se requiere de un comando especial para su creación.
- Al igual que la red bridge por defecto, actúa como un **switch virtual interno**, asignando una subred privada y una IP privada a cada contenedor.
- La subred puede ser definida en la creación de la red bridge (Ej. 192.168.50.0/24) al igual que la IP del gateway (Ej. 192.168.50.1).
- El SCOPE de la red es local, sólo es accesible desde el docker del host.
- Permite que los contenedores de un mismo host (misma red bridge) se comuniquen entre sí, pero no expone automáticamente los puertos al exterior. Se requiere de un mapeo desde el host a contenedores (docker create/run -p o -P).
- **Los contenedores pueden comunicarse libremente entre sí utilizando cualquier puerto (no hay reglas fireware)** usando:
  - La IP privada asignada por Docker.
  - **El nombre del contenedor** (Diferencia con laa bridge por defecto).
- Las IPs se asignan de forma dinámica cada vez que se arranca un contenedor (docker start/run). Si se para el contenedor y se vuelve a arrancar puede que se le asigne una IP diferente.

P10. Gestión Básica de Redes
Docker network create / rm

Grid y Computación de Altas Prestaciones
GRyCAP

- **docker network create --driver bridge** crea una red virtual bridge gestionada por docker donde se puede personalizar parámetros como las IPs de la red privada a asignar y la IP privada el gateway, entre otros.

docker network create --driver bridge <opciones>[nombre_red]

- **Opciones**

  --subnet XX.XX.XX.XX/XX →subred privada. (Ej. 192.198.0.0/16). Se puede omitir, por lo que por defecto utilizará 172.17.0.0/16.

  --gateway XX.XX.XX.XX → IP del gateway (se asignará al host). Se puede omitir, por lo que por defecte le asignará el valor 1. (Ej. 172.17.0.1)

- **Parámetros**

  [Nombre_red] → Nombre de la red bridge creada por el usuario.

- El comando **docker network** elimina una red virtual gestionada por docker.

docker network rm [nombre_red]

- **Parámetros**

  [Nombre_red] → Nombre de la red a eliminar.

P10. Gestión Básica de Redes
Docker network create / rm

- **Tarea 10.4:** Crea tu propia red bridge y llámala "mi_red_bridge" con una subred definida como 192.168.0.0/24 y el gateway en la dirección 192.168.0.10.

P10. Gestión Básica de Redes
Docker create / run

- ***docker create*** crea un contenedor a partir de una imagen que tengamos en el registro local.
- El comando ***docker run*** tiene implícito **docker create,** por lo que las opciones de gestión de redes también se pueden utilizar en este comando de forma análoga.
- Para conectar un contenedor con una red distinta a la red bridge por defecto debemos de crear el contenedor de forma adecuada mediante "**docker create/run**" con las opciones específicas.

docker create <opciones> <nombre_imagen:etiqueta/s> <comando>

- Opciones nuevas para la gestión de redes

  --network → Nombre de la red donde se va a conectar el contenedor.

P10. Gestión Básica de Redes
Docker network create / rm

Grid y Computación de Altas Prestaciones
GRyCAP

- **Tarea 10.5:** Crea dos contenedores y conectalos a la red creada en la tarea 10.4:
  - un contenedor interactivo (/bin/bash) y llámalo mi_con2_red.
  - un contenedor que exponga el servicio nginx y llámalo mi_con2_red_web.

  Consulta las IPs privadas de los dos contenedores creados (docker inspect bridge). Lanza ambos contenedores y entra mediante pseudoterminal al contenedor interactivo. Una vez dentro:
    - instala curl (apt-get-update && apt-get install curl)
    - ejecuta "curl http://mi_con2_red_web"
    - Luego ejecuta "curl http://<IP_Privada_mi_con1_redweb>."

P10. Gestión Básica de Redes
Docker network create / rm

Grid y Computación de Altas Prestaciones
GRyCAP

¿qué ocurre?

P10. Gestión Básica de Redes
Docker network connect / disconnect

Grid y Computación de Altas Prestaciones
GRyCAP

- **docker connect / disconnect** permite conectar o desconectar un contenedor a una nueva red virtual existente en docker.

docker network connect/disconnect <opciones> [nombre_red] [nombre_contenedor]

- **Opciones**

  --ip XX.XX.XX.XX → IP a asignar al contenedor. Si no se especifica esta opción, se asigna una automáticamente dentro del rango especificado en la red virtual.

- **Parámetros**

  [Nombre_red] → Nombre de la red a conectar con el contenedor.
  [nombre_contenedor] → Nombre del contenedor donde conectar la red.

P10. Gestión Básica de Redes
Docker network connect

- **Tarea 10.6:** Conecta el contenedor mi_con1_red creada en la tarea 10.3 con la la red mi_red_bridge creada en la tarea 10.4:

P10. Gestión Básica de Redes
Red network host

- En la red bridge por defecto o creadas por el usuario, los contenedores toman una IP privada de una red aislada y el host no puede acceder directamente a los puertos internos del contenedor a menos que se haga un port mapping (por ejemplo docker run -p 8080:80 …).

- Podemos en el proceso de creación (**docker create/run --network**) o con el comando específico **docker network connect** conectar el contenedor a la misma red (**host**) que el host, por lo que el contenedor toma la IP del host y todos sus puertos

P10. Gestión Básica de Redes
Docker network create / rm

- **Tarea 10.7:** Crea y arranca sin mapear puertos un contenedor que exponga el servicio nginx y llámalo mi_con3_red_web y conectalo a la red host. Luego ejecuta "curl http://<IP_Host>"

P9. Gestión Repositorios Externos
Evidencias

- Al final de esta actividad práctica deberás de tener el entorno **Docker-CE** con una serie de contenedores creados.
- Verifica que están todos los contenedores con docker ps -a:
- Para el portafolio, hay que recoger las siguientes evidencias:
  - Captura de pantalla del terminal con la salida del comando anterior donde aparezca tu usuario.
  - Texto donde indique la tarea y el comando o comandos utilizados.